Contents

About this book

If you want to make it right,
make it wrong first.

What it is about

This book is about *knowledge discovery.* There are many excellent books on machine learning or data mining. And there are many excellent books on the different special aspects in these areas. Even though all the knowledge we are concernd with in computer science is *relational*, relational or logic machine learning or knowledge discovery is rather uncommon. Accordingly, there are fewer textbooks on this issue.
This book puts a strong emphasis on "knowledge"; what it is, how it can represented, and, finally, how new knowledge can be discovered from existing knowledge and some new observations. Since the our interpretation of "knowledge" is based on the notion of "discernability", all the methods discussed in this book are presented along the same paradigm: Learning means to acquire the ability of discriminating different things from each other. Since things are different if they are not equal, equality or rather equivalence plays a major role. And whenever there is equality or equivalence, there are certain degrees of equality: Things can be exactly the same, the can be the same in most cases or aspects, the can be roughly the same, not really the same, and they can be entirely different. Sometimes, they are even incomparable.
There are several well-known approaches to describe "similarity" between sets of objects. If we arrange all our objects by their properties, than their mutual distance to each other reflects their similarity. And if there are two different objects which have a zero distance we have to find another property so as to tell one from another. If all the objects are described by a set of features, then similarity means something like the number of features in which they agree. The utility of a feature for finding new knowledge is its information content. Since features induce equivalence relations, many features create many such relations. And with intersecting them, we gain a very fine-grained partitioning of the universe in many, many small classes of, well, equivalent or equal or similar things. Finally, we can describe objects and concepts by logic formulas or theories. Then, knowledge discovery means to refine our set of formulas such that we are able to deduce something that we were not be able to infer before.

Every paradigm is dedicated a chapter on its own.

How it is organised

This book tries to illustrate the common ideas behind all those different approaches in machine learning or knowledge discovery. If you take a look at a set of books each of which specialises in any of these areas, you will find a different idiosyncratic notation in each of them. This does not really help in understanding the common processes and the parts *in which they differ.* And it is important to understand the differences between them in order to gain knowledge about them. It is also the differences that make one or another paradigm more suitable in a certain domain. Therefore it is important to be able to see them clearly. As a consequence this textbook has a leitmotif: it is the example of objects like $\triangle$, ●, or ■—and their differences, their common properties and how to construct different concepts like "grey boxes" or "things with at most n corners". If you would consider this a plus for reading this book, then you might consider the next one as a minus: In order to stress the common characteristics of the theories, we need a common language. As a result, I tried to find a more or less consistent same notation or notational principle (like "a $\cup$ is to a $\subseteq$ what $\sqcup$ is to $\sqsubseteq$; and $\longrightarrow$ is to $\Longrightarrow$ what $\vdash$ is to $\models$."). I think it is a nice idea to use the same notation throughout a book covering several topics that usually use different notations. But the downside is that the result is another nomenclature. I beg the reader's foregiveness for the usage of greek letters, upright and slanted function names, fraktura characters and symbols you will never see again somwhere else (unless you attend my classes).

The language used in this book is English with a German accent. Apart from that I tried to find a pretty delicate balance between informal written text and a rather formal and exact notation. The text is to understand what all the formulas are about—and the formulas are there to have an undisputable and solid foundation for describing things. Additionally, there are many examples. As mentioned above, there is the running example of geometric shapes. But there are many other examples: some from every-day life, some famous examples, but also some rather surprising examples that require very special knowledge in areas not all readers will be famliar with but which I hope to be even more illustrative if you are (did you ever notice that it takes three variables in Lambda-calculus to define exclusive disjunction?).

Then, there are exercises. I labelled them with different numbers of marks of different colour. Just try to solve them, and you'll discover the knowledge that is required to explain the system behind the labelling...

Finally, there are "knowledge boxes". They are small grey boxes like this:

Box of Knowledge
A box of knowledge summarises the relevant results of a section in a punchline; preferrably in prose. By reading them alone, you ought to be able to tell someone else what this book is about and even to explain most important concepts in your own words.

Thanks to:

Helmar, who taught me to ask the right questions; Ivo, who was the first to introduce me to the beauty of formal thinking; and Bernhard with whom I discovered the combination of both.

Alexander, Jong-Hwa, and Peter for friendship, help, and support.

All researchers I met during the past fifteen years for their inspiration, discussion, clarification, and criticism.

All students who by their bravery and willingness to pass the exams contributed to all the previous versions.

David from CUP for patience.

Chapter 1

Introduction

Knowledge discovery, machine learning, data mining, pattern recognition, and rule invention are all about algorithms which are designed to extract knowledge from data and to describe patterns by rules.

One of the cornerstones of *(traditional) artificial intelligence* is the assumption that

> intelligent behaviour requires rational, knowledge–based decisive and active processes.

These processes of include the acquisition of new knowledge which we call *machine learning* or *knowledge discovery*. But when talking about *knowledge based systems* we first need to explain *knowledge*. If we try to define learning by intelligence, we need to explain intelligence, and if we want to explain intelligence, we need to explain knowledge. Bertrand Russell has given a very precise and in our case very helpful (and actually entirely sufficient) definition of *knowledge*:

> Knowledge is the ability of discriminating things from each other.

As a consequence, learning means to acquire the ability to recognise and differntiate between different things. Thus, the process of knowledge acquisition is a process that is initiated and (autonomously) run by a system that is about to learn by itself. L. G. Valiant said that

> Learning means to acquire a program without a programmer.

To us, it means:

Learning as Discovery of Knowledge
Learning means to acquire the ability of discriminating different things from each other without being told about every single instance.

1.1 Motivation

Like any other computer science or artificial intelligence discipline, machine learning research varies along many dimensions. This includes the interpretation of the learning process as a data processing technique, a rule discovery tool or a model of cognitive processes. Machine learning approaches can also be described in terms of their successful application in the real world.

1.1.1 Different Kinds of Learning

Engineering and Theory: As an engineer or computer scientist who seeks for patterns in data sets, one might ask how to extract knowledge from huge databases and how to make knowledge elicitation as efficient as possible. If, on the other hand, you are interested in the theory of computation then it would be much more interesting to see if there are fundamental limitations of learning in terms of complexity or in terms of the problem classes.
Data Driven Learning and Conceptual Learning: Data driven learning means to take all data (or, rather, observations) *without* much further information about the data and try to extract as much knowledge as possible. This means that data driven learning focuses on what one can learn from the supplied data. But quite often we already have a rather precise image of what we think the world is like. We then use a set of known concepts that a learning algorithm uses to describe unknown target concepts. The difference is that in data driven learning one tries to identify clusters of similar objects from a set of observations. Conceptual learning supplies our algorithm with background knowledge about the world. As an example, data driven learning may help to discover classes like *mammals* and *birds*. Using knowledge about *habitat* and *domestication*, conceptual learning is able to describe penguins and polar bears by their habitat and it can tell a dog from a wolf by their domestication.
Clustering or Classification and Scientific Discovery: Engineers are often concerned with huge sets of data, and the larger the sets are, the less is known about the (hidden) structures in data. In the course of developing growing data warehouses, some system operators are in the need of handling petabytes of data. With too much data around and too little knowledge about the data, one needs to devise algorithms that efficiently group similar cases into the same classes. Classification means to assign a new unseen case to one of the given class labels, but scientific discovery is rather concerned with finding new subclasses or relational dependencies between such classes. If such class hierarchies and dependencies are expressed in terms of rules, then the invention of a new concept and its description is what we call *scientific discovery.*
Algorithms and Cognitive Processes: Similar to the Engineering/Theory dichotomy, one can also focus on the algorithmic issues in data mining or understand machine learning as a metaphor for human learning. For example, many data sets can be explained using decision trees—but using modules of artificial neural networks one can evaluate psychological models of human problem solving, too.

Data Mining is a multi-stage (business) process leading to the automated detection of regularities in data which are useful in new situations. Especially in the context of very large data sets or data warehousing scenarios, knowledge can be compared to a gem that is very well hidden under a huge mountain of rock-hard data. Knowledge discovery requires the extraction of

- implicit (but hidden), previously unknown
- and potentially useful information from
- data

or even the search for relationships and global patterns that exist in databases.

1.1.2 Applications

Apart from all the (semantic-) web applications, biological applications ("computational biology") are gaining popularity; especially in genetics or large–array marker scans. For example, sequencing of the genetic code allows us to understand the encoded protein—given that we can understand how tertiary structures of proteins develop during folding. Similarly, pattern recognition on marker arrays help to identify genomic defects or other diseases, and the spatial properties of molecules can be expressed using a language of relations. It would be very interesting to explain—in terms of molecule structures—why some chemicals are carcinogenic while others are not, [Muggleton et al., 1992, Muggleton et al., 1998].

Patterns and rules are very popular research topics: Nearly all observations consist of complex patterns from which we try to abstract by generalisation. Furthermore, many similar observations are mapped onto class representatives (both a penguin and a sparrow are birds). Spam classifiers take emails as patterns of word occurrences and use rules defined by filters to keep your mailbox clean.

Recently, Data Mining has become one of the most important application areas in knowledge discovery. Just recall how it was a few years ago when you tried to find some information in data collections: The first problem was that we did not have enough data available—i.e. we knew what kind of information we were looking for and the amount of available data could be easily surveyed. But the result was that we could not find the information needed simply because it wasn't there. The second generation information retrieval problem was slightly different. Now there was enough data available, but the problem was to find it. First approaches required data items to be tagged with meta-data until more powerful indexing and search methods were developed.

Example 1.1 Information retrieval in the World Wide Web is a very lucid example: In the early 90's the web was so small that personal link lists (and those of others) were sufficient for exhaustive search. The next step introduced search engines like Yahoo (with manually tagged indices) and AltaVista and many other services competing for the largest search indices. Today, we simply "*google*" the web for information. ●

With an ever increasing amount of available data, the next question became how to integrate all the data in order to find the desired information. The answer was data warehouses. In 2005, Yahoo was reported to maintain a 100 PetaByte data warehouse and AT&T uses two data warehouses with together 1.2 ExaByte of data. The question that arises now is: what kind of information is hidden within all this data? And this is what *knowledge discovery* is all about. Commercially, it is referred to as *data mining*—because this is what we do in order to find some "gems", i.e. important pieces of information, in the huge pile of data. There is a small difference, though: we use the term "knowledge discovery" to describe the process of extracting new knowledge from a set of data about a set of data. This means that the acquisition of new knowledge requires us to build a new model of the data. "Data Mining" mostly refers to the extraction of parts of information with respect to a *given* model. One crucial problem is the interpretation of correlation: If two things correlate, it does not neccessarily mean there exists some kind of causal dependency between them. This is where *relational* knowledge discovery enters the game: Here, the primary interest is in the relations between *relations* and not the relations between *objects*.

Last but not least, knowledge discovery, data mining and machine learning are tools that can be used in any situation where the problem we are faced with is ill–posed. So if we are not able to devise an efficient deterministic algorithm that solves our problem within a predefined instance space, we simply give it a try and let the machine learn itself.
In quite many cases, the results are surprisingly good. In other cases they are not. But then, we could at least blame the machine for being a bad learner rather than blame us for being a bad programmer.

1.2 Related Disciplines

To us, machine learning means to learn how to discriminate different objects. There are many other disciplines which have a slightly different view on the same problem.

1.2.1 Codes and Compression

The idea behind coding is to represent a *meaningful message* by a suitable sequence of symbols. The representation process is called *encoding* and maps *plain text (symbols)* or *(source) messages* onto *codes* which are symbol strings, too. A one-to-one *substitution* of source symbols onto code symbols is called *encipherment*; the result a *cipher*. The reverse processes of reconstructing the original message from a code (cipher) is called *decoding* (*deciphering*).
We assume the reader to be familiar with the sequence of *Fibonacci numbers*. Yet, if being asked, no one will ever reply by saying "Fibonacci numbers? Sure: 1, 1, 2, 3, 5, 8, ...". So even if you were able to memorize the entire sequence of

Fibonacci numbers, you have not *learned* anything about them because learning would require the ability of (intensionally) explaining a *concept*.[1] This also relates to *data compression*: The less compressible our data, the more information it contains and the harder it is to learn and compress it further. Any good learner needs to be able to *compress* data by giving a rule that is able to describe the data.

Example 1.2 **RLE-Compression.** One of the simplest methods to encode and compress a stream of symbols is *run length encoding*. Consider the alphabet $\Sigma = \{\Box, \bigcirc\}$. Then, strings will contain repetitive occurrences of each symbol. For example:

$$\Box\Box\bigcirc\Box\bigcirc\bigcirc\bigcirc\bigcirc\bigcirc\Box\Box\Box\Box\bigcirc\bigcirc\Box\bigcirc\bigcirc\bigcirc\Box\Box\Box\bigcirc\cdots$$

A reasonable way to compress such a sequence would be to precede each symbol by the number it occurs until the other symbol appears. And if we assume that each sequence starts with a $\Box$ we can even drop the symbol itself since on every sequence of $\Box$s there can only be a sequence of $\bigcirc$s and vice versa. So, the above string can be compressed to

$$2.1.1.5.4.2.1.3.3.1.\cdots$$

which certainly is much shorter but requires a larger alphabet of symbols and a special delimiter symbol ".". ●

Finding (optimal, shortest) codes (that is, *encoding functions*) without the need for delimiters or any additional symbols is the topic of coding theory.

The notion of compression enters the game at two different points: First, a message that cannot be compressed further is free of any redundancies. Then, the message string itself must have maximum entropy and all the symbols occurring in the message are more or less of the same probability. However, they do *not* encode the same "amount of information": changing one symbol can result in just a small change after decoding, but it can also scramble the entire encoded message into a meaningless sequence of symbols. Second, a strong concept requires a complex representation language which basically is the same as finding a good *code*.

One example that we shall encounter is the encoding of messages (or hypotheses) into strings of symbols ("genomic codes"). The field of coding theory and compression has become a huge discipline of its own, which is why we refer to [MacKay, 2003]. If you are more interested in coding and cryptography, consult [Welsh, 19xx]; and for the advanced reader we recommend [Chaitin, 1987, Kolmogorov, 1965] and [Li and Vitanyi, 1993].

[1] Mind the difference between *intentional* and *intensional*. Explaining something intentionally means to explain something with a certain intention in mind. But explaining something intensionally (as opposed to extensionally) means to explain something by abstracts concepts instead of concrete examples.

Example 1.3 It is a crucial difference to encode the numbers $0, 1, 2, \ldots 255$ using the sum of four integers $w + x + y + z$ (with $0 \leq w < 64$ and $0 \leq x, y, z \leq 128$) or by using a binary representation with eight bits. Changing one symbol in each representation creates an error of at least 1; but using the decimal representation the maximum error is 64 while the maximum error for the binary representation is 128. ●

It is clear that a sequence of symbols which can be compressed pretty well (e.g. by RLE), obviously contains some kind of redundancy. If we know that the next five symbols we receive are □s, then the sender's effort in transmitting □□□□□ is simply a waste of time (and, as we shall see, a waste of bandwidth).

Machine Learning and Coding
Learning means to find an optimal code to describe observations.

Coding and its role in cryptography are far beyond the scope of this book, but they are indispensable for *information theory* as well. The basic idea behind information theory is that the average randomness of a sequence (and thus, its reverse redundancy) is a measure of the complexity of the system that emits the messages.

1.2.2 Information Theory

There is much confusion about the term of "information". Together with entropy, complexity or probability, terminology often gets in the way of a proper understanding. To avoid misconceptions from the very beginning, we first and foremost need to make clear one crucial point:

Information
Information is not so much about *what* is being said, but rather what *could* be said.

As we will discover in detail later on, there exists a measure of information content of a message expressed in terms of entropies:

$$H_{\max}(S) = \log_2 \omega \quad \text{and} \quad H(S) = -\sum_{i=1}^{\omega} p_i \log_2 p_i$$

The first formula describes the "information content" of a system in terms of it's complexity: it is determined by the number ω of possible states an element of the system can take (the log will be explained later). Since in communication theory not all possible messages are equally probable, Shannon introduced the second formula: It measures the information that is hidden in the probabilities that an element of the system takes a *certain* value. This can easily be explained by a coin flipping game: Note, that a coin can take only two states—heads (**1**) or tail (**0**). This means that $\omega = 2$. Therefore, $H_{\max}(coin) = \log_2 2 = \frac{\ln 2}{\ln 2} = 1$. In other words, any situation in which all possible outcomes are equally probable are situations of maximum entropy. The magnitude 1 of this measure of information is usually interpreted as the number of bits required to encode the information contained in one single element of the domain (here: one single coin). After flipping two different coins 50 times, we find

$$\begin{aligned} & H(\mathtt{01001111101000010010111110101010010011110101001000}) \\ = \quad & -0.5 \log_2 0.5 - 0.5 \log_2 0.5 = 1 \end{aligned}$$

and

$$\begin{aligned} & H(\mathtt{00000000100000000000000000000000000000000000000000}) \\ = \quad & -0.02 \log_2 0.02 - 0.98 \log_2 0.98 = 0.14 \end{aligned}$$

Obviously, the first coin is rather fair, while the second is not. As one can see, the (expected) probability of the outcome of an experiment (or, an *observation*, or the probability of a message) crucially defines the information content and therefore describes the complexity of such a system (or the information content or the bandwidth required). Cheating means to reduce the information content of a game: it reduces the uncertainty of the outcome. As we all know we can cheat the better the more we know (and vice versa). As we shall see later on, knowledge about the domain can be expressed in distributions on the domain. From the viewpoint of information theory we conclude:

Machine Learning and Information Theory
Machine learning means to organise our concepts in a way that minimises the entropy of the system.

[Shannon and Weaver, 1949] is the original publication; for a more recent and gentle introduction we recommend [Ash, 1965].

1.2.3 Minimum Description Length

The idea behind the Mdl-principle is quite similar to the old law of parsimony—usually known as *Ockham's razor*:

> *Frustra fit per plura quod potest fieri per pauciora*, [of Ockham, 1323]

i.e. there is no point living with many explanations if just a few suffice. There are hundreds of (modern) interpretations of this principle; but only few know that Aristotle already stated that

> *Nature does nothing in vain.*[2]

A modern explanation of the principle can be found in [Barron et al., 1998]. In machine learning, we can assume a hypothesis h to represent (a part of) a theory K which is to explain a set s of observations. The description length is a measure of the complexity of h with respect to s in terms of the costs for transmitting knowledge about s. As we shall see later, s is a set of entities together with a label that describes whether it belongs to the (unknown) target concept or

[2] [Anima iii 12 434^a31-2]. Aristotle also used the attributes *random* and *superfluous* in similar contexts.

not. The MDL-principle seeks to find the shortest possible description of a set K of observations. For two competing descriptions h and h', one assumes the *shorter* one to be the better description. The MDL-principle does *not* try to explain as many (unknown) objects of our domain as possible. It rather tries to *minimize* of the amount of knowledge that is required to classify an object correctly. The problem here is to define what it means for a theory or hypothesis to be better (i.e. shorter) than another. Any theory is to explain a set of observations, therefore the quality of our hypothesis depends at least on two properties concerning our observations:

- which subset s do we have to learn from, and
- what is the nature of the set of objects that we want to describe?

As we shall see at the end of this book, one usually needs more examples to learn concepts from more complex domains. But even if we have sufficiently many and good examples to find a hypothesis that fits our data, the result crucially depends on the *randomness* of the sender, too.

Machine Learning and MDL
Machine Learning means to find the shortest description or the simplest explanation of our observations.

Yet, it remains to explain what it means for a theory to be *simpler* than another one.

1.2.4 Kolmogoroff Complexity

Kolmogoroff complexity is an *algorithmic* measure of expressiveness. It is one of the most important concepts in the theory of computation after Turing's and Gödel's work. The idea behind algorithmic complexity has been independently developed by three scientists: Ray Solomonoff, [Solomonoff, 1964], Andreji Kolmogoroff, [Kolmogorov, 1965], and Gregory Chaitin, [Chaitin, 1966].

The general idea behind the measure of Kolmogoroff complexity is to measure the expressiveness of a system by the length of a program that generates such a system: The simpler the system, the shorter the description. But if there are some parts of huge complexity in the system it could be that the plan for this part only would require much more space than the object itself! In such a case it seems reasonable to transmit the object rather than a description of it.

A special case is when we divide a description which we *know* to produce wrong results into several parts: Then, the incorrect program together with information about *where* it produces wrong output and *what* the correct output would be, could still be much shorter than the smallest correct program. In terms of the MDL-principle we try to find the ideal trade-off between program complexity (length) and the length of a file of exceptions.[3]

[3] The problem is to determine the length of the smallest program with a certain output (or rather, to prove that it is the shortest program).

Example 1.4 Consider the set $s = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$ and the target concept $c = \{x \in U : x \mod 3 = 0\}$. Let h describe the set of all odd numbers in s: $h = \{1, 3, 5, 7, 9\}$. Imagine, the program for h would be represented as $2n + 1$. Then, h is wrong on the set $\{1, 5, 6, 7\}$. The description length of h would be the length of h plus the encoding of this set. Imagine it was `[2, n, +, 1, !, 1, 5, 6, 7]`. Then, h is a non-compressing description of *char*(c) because the length of the description string equals the cardinality of c. ●

The *Kolmogoroff-complexity* $C(\vec{x})$ of a sequence $\vec{x}$ of symbols is the length of the *shortest* binary program π that generates $\vec{x}$ and then halts. Here, "shortest binary program" means the shortest binary string that represents a program that can be interpreted by a universal Turing machine. This enables us to review the notion of compressibility: $\vec{x}$ is close to being incompressible, if $C(\vec{x}) \approx |\vec{x}|$. It definitely cannot be compressed any further, if $C(\vec{x}) > |\vec{x}|$ and the degree of compressibility (or information loss) increases with the term $f(\vec{x})$ in $C(\vec{x}) \approx |\vec{x}| - f(\vec{x})$.

Machine Learning and Complexity
Machine Learning is the task to find a program π with least $C(\pi|\vec{x})$ for any $\vec{x}$ in our problem class.

An excellent introduction into Kolmogoroff complexity for the advanced reader is [Li and Vitanyi, 1993]. For those interested in the theory of machine learning, we also recommend [Valiant, 1984] and [Anthony and Biggs, 1997, Kearns, 1990, Kearns and Vazirani, 1994].

1.2.5 Probability Theory

Probability theory mostly deals with urns and marbles.

Machine Learning and Probabilities
Machine Learning means to:

1. approximate an unknown probability distribution on our set of objects
2. by learning from statistical observations such that
3. the error probability is minimised

Let there be the following objects in our universe:

$$U = \{\blacktriangle, \square, \bullet, \blacksquare, \circ, \triangle\}$$

Each of these objects obviously has two properties: They have certain *shape* and *colour*. In probability theory, such features are represented by (not so) random variables S and C. Shape and colour of objects are independent from each other; and so are the random variables S and C. If we now assume the probability of picking any of these objects from our urn to be uniformly distributed (also called an *independent identical distribution*, or *iid* for short), then:

$$\begin{array}{rcl} \mu(\{x \in U : S(x) = \textit{triangle}\}) & = & \\ \mu(\{x \in U : S(x) = \textit{square}\}) & = & \\ \mu(\{x \in U : S(x) = \textit{circle}\}) & = & \frac{2}{6} = \frac{1}{3} \\ & \text{and} & \\ \mu(\{x \in U : C(x) = \textit{black}\}) & = & \\ \mu(\{x \in U : C(x) = \textit{white}\}) & = & \frac{3}{6} = \frac{1}{2} \end{array}$$

People quite often use a different notation and abbreviate $\Pr[S = c] := \mu(\{x \in U : S(x) = c\})$. Then,the probability of picking a black triangle from the urn is

$$\Pr[S = \textit{triangle}] \cdot \Pr[C = \textit{black}] = \frac{1}{3} \cdot \frac{1}{2} = \frac{1}{6} = \frac{|\{\blacktriangle\}|}{|\{\blacktriangle, \square, \bullet, \blacksquare, \bigcirc, \triangle\}|}$$

Things become more complicated the more they depend on each other. Accordingly we usually assume most things to be independent even if they are not. This is a wise decision from a computational point of view—but in true life, most observations depend on something else. Consider the concept *form* (F) which describes whether an object is *angular* or *round.* Of course, the colour has no influence on the general geometric form, but the shape certainly has. This leads us to the notion of *conditional probabilities* and Bayes' Rule. As an example consider the question: What is the probability of picking a triangle given the object is angular? We count:

$$\frac{|\{x \in D' | x \text{ is a triangle}\}|}{|\{x \in D' | x \text{ is angular}\}|} = \frac{|\{\blacktriangle, \triangle\}|}{|\{\blacktriangle, \square, \bigstar, \blacksquare, \triangle, \}|} = \frac{2}{5}$$

Bayes' law offers much more to the Bayesian learner: Supposing that we have a sequence of observations described by a set of n attributes together with some information about the classes $c \in \mathfrak{c}$ they belong to. Then, we can easily estimate the probability of some element's properties given that it belongs to some class: we simply scan our case library for each of the classes c and store all information concerning

$$\Pr[X_1 = v_1 \wedge \cdots \wedge X_n = v_n | X_{\mathfrak{c}} = c]$$

Similarly, we can also estimate the prior probabilities

$$\Pr[X_1 = v_1 \wedge \cdots \wedge X_n = v_n] \qquad \text{and} \qquad \Pr[X_{\mathfrak{c}} = c]$$

Together, we can for any new case $X_1 = v_1' \wedge \cdots \wedge X_n = v_n'$ determine the value of

$$\Pr[X_{\mathfrak{c}} = c | X_1 = v_1' \wedge \cdots \wedge X_n = v_n']$$

for each $c \in \mathfrak{c}$—and by reporting the c for which the above term delivers the largest value we have built a so–called *maximum a posterior* classifier.

1.2.6 Approximation and Search

Learning from examples can also be interpreted as an *approximation* problem. Whether or not the world we live in is continuous, a classification is just the value a *classifier function* delivers when given an object of the domain. In many classification scenarios, the function is binary—it decides whether some x belongs to a concept c or not: $t : U \to \mathbf{2}$ with $t(x) := char(c)(x)$. Inability to discern two objects x and y where one belongs to c and the other does not requires to *learn*: Our knowledge k is not precise enough to tell the objects from each other because $K(x) = K(y)$.

Function approximation means to take a set of supporting points and fit a function through them. Linear regression is one (simple) example; and we shall discover several more.

Imagine that $t : \mathbb{R} \to \mathbb{R}$ and we are given a set of support points $\mathbf{s} = \{\langle x_i, t_c(x_i)\rangle : i \in \mathbf{m}\}$. We now need to find a function h which equals t on $\mathbf{s}$, or, more realistic, which comes as close as possible to each point. There are simple approximations like linear functions and there are more complex ones like polynomials of grade k. We can even combine sets of such functions. But to *find* such a function, we need to *search* for it.

Machine Learning as Search
Machine Learning means to search for a function which approximates the target as close as possible.

Conclusion

Now that we have briefly described the influences of other (neighboring) disciplines we can try a first working definition of *machine learning* and *knowledge discovery*.

In most cases the data from which we want to extract knowledge is represented in *information systems*. Roughly speaking, an information system is just a table with all the rows representing entities and columns describing the objects' properties. The knowledge we want to acquire consists of rules describing a model which we can use to explain the data set and its structure.

Knowledge Discovery by Machine Learning
In general, we will use the term *machine learning* to denote a method by which ...

- a set of *data* stored in an information system
- is analysed and transformed
- to *extract (new) knowledge* from it and to refine our *model* of the world in order to
- increasingly *precisely discriminate more* concepts.

While inference and analysis tries to find out more about (less) things, learning is concerned with the task to find out (less precise) about more things.

Chapter 2

Relational Knowledge

Talking about the discovery of knowledge requires us to understand "knowledge" first. In the last chapter we already defined knowledge to be what it takes to discriminate different things from each other.
In this chapter, we will develop a more formal framework of knowledge structures which enables us to describe the process of discovery of new knowledge.

Information is something that may change knowledge, and knowledge is the ability to relate things by putting a structure on them. Knowledge is not made from the things we put into order and information does not change the things themselves. It is rather that knowledge is a set of relations to describe things and that information helps us to describe a relation's utility in classifying things. But then, how come people assume information to be describable by a set of entitites each of which can take a certain number of different states—if we do not know whether there are entities we have never seen before nor how many states they can possibly take? How come people explain causality by means of probabilistic dependence?

There are *many* definitions of what knowledge could be, and there are *many* approaches to knowledge representation formalisms. There appears to be knowledge of different qualities: factual knowledge, weak knowledge, procedural knowledge, hard knowledge, motor knowledge, world knowledge, behavioural knowledge are just a few. Our idea of knowledge is, sloppily and circulary defined, what we want to acquire by learning. Even when talking about motory skills it is all about discriminating things from each other: *bad* moves and *good* moves. This is a very blunt and vague generalisation—but for our purpose the following definitions of knowledge shall be sufficient.

2.1 Objects And Their Attributes

We now examine a few concepts from which we can build a *simple* theory of knowledge — without making too much assumptions about our world.

2.1.1 Collections Of Things: Sets

A set is usually considered to be a collection of discernible objects. Since a and a seem to be indistinguishable, we all agree that $\{a, a\} = \{a\}$. There is a very special collection of things, that which is empty: $\emptyset = \{\}$. Sets can be defined by naming all their members or by giving a description which all members of this set have in common:

$$\{\mathrm{m}, \mathrm{i}, \mathrm{s}, \mathrm{p}\} = \{x : x \text{ occurs in mississippi}\}$$

The left side of the equation shows an *extensional* set definition while the right side is an equivalent *intensional* description. Just as we can put a number of bags with different collections of objects into a larger bag, sets can contain other sets.

Example 2.1 Imagine the set r of those sets x , which do not contain themselves:

$$r \quad := \quad \{x | x \notin x\} \tag{2.1}$$

The set r seems to be quite an interesting collection of objects (some of which may contain other objects or collections thereof). This gives rise to the question whether r belongs to it itself: Is $r \in r$ or not? ●

This is known as Russell's paradox. It showed that Cantor's naïve set theory had a serious flaw — which cost many researchers sleepless nights. The smiles returned to the faces of those who were satisfied with a proper axiomatization of set theory after Skolem had finalized Zermelo and Fränkel's set theory. But those who remained sceptic could not but resent accepting Gödel's proof that axiomatic systems rich enough to describe integer arithmetic can describe problems that cannot be proven to be true or false within this system.

People tend to collect things in an ordered way, which is why many sets have some structure, too. So, for example, the set $\{\mathrm{a}, \mathrm{b}, \mathrm{c}, \ldots, \mathrm{z}\}$ together with a reflexive, transitive and antisymmetric relation forms a *partially ordered set.* Such a relation is usually known as an *ordering* relation $\leq$ (here, lexicographic ordering). However, some collections appear to have no structure, others seem to have a stronger than just an ordered structure.

For the remainder of the chapter, we assume the reader to be aquainted with basic set operations such as $\cup, \cap$ and $-$. Consider the following recursive set definition starting with the empty set:

$$\begin{aligned}
\mathbf{0} &:= \emptyset = \{\} \\
\mathbf{1} &:= \mathbf{0} \cup \{\mathbf{0}\} = \emptyset \cup \{\emptyset\} = \{\emptyset\} \\
\mathbf{2} &:= \mathbf{1} \cup \{\mathbf{1}\} = \{\emptyset, \{\emptyset\}\} \\
&\vdots
\end{aligned}$$

Even though $\emptyset$ trivially has no structure at all, the subsequent sets (and the set of all these sets) do have a structure. Hence, the following properties hold:

$$0 \leq x \Longleftrightarrow \mathbf{0} \subseteq \mathbf{x} \quad \text{and} \quad x < y \Longleftrightarrow \mathbf{x} \in \mathbf{y} \qquad x \leq y \Longleftrightarrow \mathbf{x} \subseteq \mathbf{y}$$

Exercise 2.1 ◊ Prove!

So if the cardinality of some enumerable set s is $|s|$, s can be mapped one-to-one on the set $\mathbf{s}$. This way, we have constructed the basis for integer arithmetics, and, by the way, gained a set (usually referred to as $\mathbb{N}$) of index sets.

To continue our work on sets we need to be able to describe a set of sets made up from a fixed (finite) repertoire of objects. We call $\wp(x)$ the *powerset* of x, the set of all subsets of x:

$$\wp(x) \quad := \quad \{y : y \subseteq x\} \tag{2.2}$$

It is clear that $|\wp(x)| = 2^x$ which equals the number of functions from x to $\mathbf{2}$. So, in general, there are y^x functions $f : \mathbf{x} \to \mathbf{y}$. Therefore, we sometimes write $\mathbf{2}^s := \wp(s)$.

But what is so special about integers and why do we insist on this issue in a textbook on knowledge discovery? The reason is quite simple: Knowledge has been described as the ability to discriminate different things from each other. We now know how to speak about collections of entities: we can represent any (countable) set of such entites by a set with a certain structure. The structure comes for free, and the relation defined by this structure allows us to relate any pair of entities to each other. Furthermore, it even allows us to relate any pair of collections of entities to each other:

Collections of different things

Any (countable) collection of x things can be *represented* by a set $\mathbf{x}$ using a one–to–one mapping $f : x \to \mathbf{x}$. On $\mathbf{x}$, we have relations $\subseteq$ $(\leq)$ and $\in$ which allow to order (and, thus, discriminate) different objects. Furthermore we can even relate sets of entities ("*classes*") to each other.

Exercise 2.2 ◊♦ Prove that Peano's axiomatisation of integer arithmetic is satisfied by our definition of $\mathbb{N}$ in the last section:

$$\exists 0 \in \mathbb{N} \tag{2.3}$$

$$\forall n \in \mathbb{N} : \exists succ(n) \in \mathbb{N} \tag{2.4}$$

$$\forall n \in \mathbb{N} : succ(n) \neq 0 \tag{2.5}$$

$$\forall m, n \in \mathbb{N} : succ(m) = succ(n) \to m = n \tag{2.6}$$

$$\forall x \subseteq \mathbb{N} : (0 \in x \land \forall n : n \in x \to succ(n) \in x) \to x = \mathbb{N} \tag{2.7}$$

where *succ* denotes the successor function.

2.1.2 Properties Of Things: Relations

In the last section, we already spoke of "order relations"—but what is a *relation* anyway? To keep things simple, we will stick to *binary relations*. For two sets s_0 and s_1, the *cartesian* or *cross product* is defined as:

$$s_0 \times s_1 \quad := \quad \{\langle x, y\rangle : x \in s_0 \wedge y \in s_1\} \tag{2.8}$$

$\langle x, y\rangle$ is called an *ordered pair* or *tuple*. It is clear, that in general $s_0 \times s_1 \neq s_1 \times s_0$.

Binary Relation

Definition 2.1 — Binary Relation.
A *binary relation* R is an arbitrary subset of a cross product of one or two base sets $R \subseteq s_0 \times s_1$ and write

$$xRy \quad :\Longleftrightarrow \quad \langle x, y\rangle \in R. \tag{2.9}$$

To indicate a left–to–right reading of the relation R we also write $R : s_0 \rightharpoondown s_1$. We call $s_0 = \mathrm{dom}(R)$ the *domain* and $s_1 = \mathrm{cod}(R)$ the *codomain*. ●

R needs not to be total; tuples may be defined only for a subset of the domain and codomain. The according subsets are called the *preimage* ${}^{\ulcorner}R$ and *image* (or *range*) $R^{\urcorner}$:

$$\begin{aligned} {}^{\ulcorner}Rs &= \{x : xRy, \text{ for all } y \in s \subseteq s_1\} \\ sR^{\urcorner} &= \{y : xRy, \text{ for all } x \in s \subseteq s_0\} \end{aligned}$$

If $s = \{x\}$, we also write ${}^{\ulcorner}Rx$ and $xR^{\urcorner}$ (called a *fibre*) and if $s_i = s$ we simply write ${}^{\ulcorner}R$ and $R^{\urcorner}$. Finally, we define two further very important operations on relations:

Converse, Complement

Definition 2.2 — Converse, Complement.
For any R we call $R^{\smile}$ the *converse* (or *inverse*) relation iff $xR^{\smile}y :\Longleftrightarrow yRx$.[1]
The *complement* $\overline{R}$ of R is defined as $x\overline{R}y :\Longleftrightarrow \neg(xRy)$. ●

There are two important binary relations: The *empty* relation $\bot := \emptyset = \{\} = \{\langle x, x\rangle : x \in s \wedge x \neq x\} \subseteq s_0 \times s_1$ and the *universal* relation $\top := \{\langle x, y\rangle : x \in s_0 \wedge y \in s_1\} = s_0 \times s_1$.
A natural way of a visual representation of two-dimensional data is a *map*. Binary relations are subsets of (binary) cross products and can be represented by matrices. For any binary relation $R \subseteq s_0 \times s_1$, we denote by $\mathbb{M}(R)$ a relation

[1] "Iff" means "if and only if".

matrix (called a *coincidence matrix*) describing R:

$$\mathbb{M}(R) := \begin{bmatrix} c_{\langle 0,0\rangle} & c_{\langle 0,1\rangle} & \cdots & c_{\langle 0,\mathbf{y}\rangle} \\ c_{\langle 1,0\rangle} & c_{\langle 1,1\rangle} & \cdots & c_{\langle 1,\mathbf{y}\rangle} \\ \vdots & & & \vdots \\ c_{\langle \mathbf{x},0\rangle} & c_{\langle \mathbf{x},1\rangle} & \cdots & c_{\langle \mathbf{x},\mathbf{y}\rangle} \end{bmatrix} \quad \text{with} \quad c_{\langle x,y\rangle} = \begin{cases} \mathbf{1} & \text{iff } xRy \\ \mathbf{0} & \text{else} \end{cases} \tag{2.10}$$

2.1.3 Special Properties of Relations

A very important class of binary relations are called *endorelations*. Such relations share the same set s as domain and codomain; i.e. $R \subseteq s \times s$.

Homogenuous Relations

Definition 2.3 — Homogenuous Relations.
A binary relation $R \subseteq s_0 \times s_1$ is called an *endorelation* or a *homogenuous binary relation* iff $s_0 = s_1$. ●

In addition to $\bot$ and $\top$ there is a third very important relation which is defined for endorelations only: $\mathit{1}_s := \{\langle x,x\rangle : x \in s\}$. The *identity relation* relates things to themeselves (and only to themselves). Subsets of $\mathit{1}_s$ are called *subidentities*.

Relations can have several special properties:

Properties of Relations

Definition 2.4 — Properties of Relations.
We call a binary relation $R \subseteq s \times s$:

reflexive	iff	xRx
symmetric	iff	$xRy \to yRx$
antisymmetric	iff	$xRy \wedge yRx \to x = y$
transitive	iff	$xRy \wedge yRz \to xRz$
difunctional	iff	$xRy \wedge zRy \wedge zRa \longrightarrow xRa.$

for all $w, x, y, z \in s$. ●

Reflexivity means to be able to "reflect" an object onto itself hence a relation R is reflexive, iff $\mathit{1} \subseteq R$. Symmetry means that something is the same no matter from where we look at it. Accordingly, there is no difference between pre-image and image and the relation between objects in them. This means that $R \subseteq R^{\smile}$. Antisymmetry is non-symmetry except reflexivity: If there are symmetric pairs in our relation, then the pairs must be reflexive, i.e. $\langle x,x\rangle$. Transitivity means that we are able to append tuples from the relation whenever one image can be taken as a pre-image. The element "inbetween" is called a *witness*. Finally, difunctionality is a very important property in the way that it expresses correspondence between pre-images and images of subsets of the domain. Whenever two objects share one object in their image, then the *entire* images of both objects are the same: $xRy \wedge zRy \Longrightarrow x\overline{R}^{\rceil} = z\overline{R}^{\rceil}$.
A binary relation is called a *partial ordering* relation, iff it is reflexive, transitive and antisymmetric.

⊕

Exercise 2.3 ◊⧫ Prove that $\subseteq$ on $\wp(s)$ forms a poset!

Equivalently, $\subseteq$ is a partial order on index sets and $\leq$ is a partial order on $\mathbb{N}$. Another very important property of some endorelations is their power of being able to group objects into disjoint subsets of the base set.

Equivalence Relation

Definition 2.5 — Equivalence Relation.
An *equivalence relation* is a binary relation $R \subseteq s \times s$ that is symmetric, transitive and reflexive: $xRy \to yRx$, $xRyRz \to xRz$, and xRx. ●

So why is an equivalence relation called an *equivalence relation*? Reflexivity states that every object is related to itself. Symmetry means, that if x *is-in-R-relation* to y, then y *is-in-R-relation* to x, too. For example, $\leq$ is *not* symmetric, since $2 \leq 3$ but not vice versa. In other words, to be equivalent two objects need to be somehow "*the same*". Finally, transitivity requires that if x is *equivalent* to y and y is *equivalent* to z, then x is *equivalent* to z, too.

Example 2.2 Let

$$A = \{a, b, \ldots, z\} \quad \text{and} \quad x = \{x_0x_1x_2 : x_i \in A, i \in \mathbf{3}\}. \tag{2.11}$$

Imagine three relations R_i with $x_0x_1x_2R_iy_0y_1y_2$ iff $x_i = y_i$. Then, R_i are equivalence relations which group all three letter strings by their i–th component: $abcR_0abb$ and $abcR_1abb$ but not $abcR_2abb$. ●

Note, that for symmetric (and thus, for every equivalence relation) R, ${}^{\ulcorner}Rx = xR^{\urcorner}$. For equivalence relations it also holds that either $xR^{\urcorner} = yR^{\urcorner}$ or that $xR^{\urcorner} \cap yR^{\urcorner} = \emptyset$. In other words, $R^{\urcorner}$ is a union of pairwise *disjoint* sets called (R)–*equivalence classes.*
For a class $xR^{\urcorner}$ we call x a *representative* and write $xR^{\urcorner} = [x]_R := \{y : xRy\}$. This class can be identified by each of it's elements, such that xRy implies $y \in [x]_R = [y]_R \ni x$.

Example 2.3 Let us consider a set of black and white geometric figures:

$$s = \{\circ, \lozenge, \square, \bullet, \blacklozenge, \blacksquare\}.$$

We define two relations $C, S \subseteq s \times s$:

$$\begin{array}{lll} xCy & \text{iff} & x \text{ and } y \text{ are of same colour} \\ xSy & \text{iff} & x \text{ and } y \text{ are of same shape} \end{array}$$

Then, $\bullet C \blacksquare$ and $\lozenge C \circ$ but not $\square C \blacksquare$. On the other hand, $\blacksquare S \square$ and $\lozenge S \blacklozenge$ but not $\circ S \square$. ●

The set $\{[x]_R : x \in s\}$ is called a partition of s; it is the set of all R-equivalence classes of R–indiscernible objects. A partition induced by some relation R is also called a *quotient set*:

$$s/R \quad = \quad \{[x]_R : x \in s\} \tag{2.12}$$

Example 2.4 The equivalence class $[\bullet]_C$ is the set of all objects $x \in s$ which are C–related to $\bullet$: It is the set of all black objects:

$$[\bullet]_C = \{\bullet, \blacklozenge, \blacksquare\}.$$

The relation C creates the following quotient or partition on s:

$$\begin{aligned} s/C &= \{\circ, \lozenge, \square, \bullet, \blacklozenge, \blacksquare\}/S \\ &= \{\{\circ, \lozenge, \square\}, \{\bullet, \blacklozenge, \blacksquare\}\}/S \end{aligned}$$

The elements of s/S often are referred to as *classes*; here the classes of *black* and *white* objects S generates three such classes, usually called *roundish*, *squarish* and *rhomboid*: $s/S = \{[x]_S : x \in s\} = \{\{\square, \blacksquare\}, \{\lozenge, \blacklozenge\}, \{\circ, \bullet\}\}$. ●

Exercise 2.4 (♢♢)

♢ Is there a relation that is both an order relation and an equivalence relation?

♢ What does it take to make a difunctional relation an equivalence relation?

2.1.4 Information systems

Information systems basically are what we all know as *relational databases*, or, even simpler, just tables or *feature-value maps*:

Information system

Definition 2.6 — Information system.
An information system $\mathfrak{I} = \langle s, \mathbf{F}, V_{\mathbf{F}} \rangle$ consists of a *base set* of objects s, a set $\mathbf{F} = \{f_i : i \in \mathbf{n}\}$ of n *features* which to each object in s assign a value of the feature's codomain V_i. An information function I delivers for some object $x \in s$ and feature $f \in \mathbf{F}$ the value $f(x)$:

$$I : s \times \mathbf{n} \to V_{\mathbf{F}} \quad \text{with} \quad I(x, i) = f_i(x) \tag{2.13}$$

All $f \in \mathbf{F}$ are (partial) *functions*. ●

Features create partitions in a rather natural way: The set $\{x \in s : f(x) = y\}$ is the set of all objects in s which share the same value $y \in V_f$. If f is a total function (i.e. $\ulcorner f = \mathrm{dom}(f) = s$), then all classes together form the quotient

$$s/f \quad := \quad \{[x]_f : x \in s\} \tag{2.14}$$

Therefore, a total function f induces an equivalence relation R_f as follows:

$$x_0 R_f x_1 \quad :\Longleftrightarrow \quad f(x_0) = f(x_1) = y \tag{2.15}$$

Equivalence relations R_f which are induced by functions f are also known as *kernel relations*.

Example 2.5 The information system

	s	*color*	*shape*
0	●	*black*	*circular*
1	□	*white*	*square*
2	◊	*white*	*rhombus*
3	♦	*black*	*rhombus*
4	■	*black*	*square*
5	○	*white*	*circular*

represents the knowledge shown in example 2.3.

The kernel relations can be represented in *kernel matrices* where all **1**–entries in $\mathbb{M}(R_f)$ are replaced by $f(x)$:

$$\mathbb{K}(R_f) = (o_{\langle x,y\rangle}) \quad := \quad \begin{cases} f(x) = f(y) & \text{iff } xR_fy \\ \perp & \text{otherwise} \end{cases} \tag{2.16}$$

Example 2.6 For S, $\mathbb{K}(S)$ is

$$\begin{array}{ccccccc} S & 0 & 1 & 2 & 3 & 4 & 5 \\ 0 & \textit{circular} & & & & & \textit{circular} \\ 1 & & \textit{square} & & & \textit{square} & \\ 2 & & & \textit{rhombus} & \textit{rhombus} & & \\ 3 & & & \textit{rhombus} & \textit{rhombus} & & \\ 4 & & \textit{square} & & & \textit{square} & \\ 5 & \textit{circular} & & & & & \textit{circular} \end{array} \tag{2.17}$$

We have seen that the features in an information system induce equivalence relations on the base set. We also know that all objects of some class $[x]_{R_f}$ are *indiscernible* using information of f. Therefore, the class $[x]_{R_f}$ is often called the concept of all $f(x)$–ish things. Given a set $\mathbf{F}$ of features we build a set $\mathbf{R} = \{R_f : f \in \mathbf{F}\}$ of equivalence relations on s. We then define a *discernability* matrix $\mathbb{D}(\mathbf{F})$ as follows:

$$\mathbb{D}(\mathbf{R}) = (d_{\langle x,y\rangle}) \quad = \quad \{R \in \mathbf{R} : y \notin [x]_R\} \tag{2.18}$$

The entries $d_{\langle x,y\rangle}$ of the matrix $\mathbb{D}(\mathbf{F})$ consist of the names of all relations by which x can be discriminated from y. Clearly, $d_{\langle x,x\rangle} = \emptyset$, because

$$d_{\langle x,x\rangle} \quad = \quad \{R \in \mathbf{R} : x \notin [x]_R\} = \emptyset.$$

In terms of common sense reasoning, any object x is indiscernible from itself; hence there are no equivalence relations or features which deliver varying information about x.

Example 2.7 The discernability matrix for examples 2.3 and 2.5 looks as follows:

$$\mathbb{D}(\mathbf{F}) \quad = \quad \begin{array}{ccccccc} & \bullet & \square & \lozenge & \blacklozenge & \blacksquare & \circ \\ \bullet & \emptyset & & & & & \\ \square & S,C & \emptyset & & & & \\ \lozenge & S,C & S & \emptyset & & & \\ \blacklozenge & S & S,C & C & \emptyset & & \\ \blacksquare & S & C & S,C & S & \emptyset & \\ \circ & C & S & S & S,C & S & \emptyset \end{array} \tag{2.19}$$

The concept of discernability will become of great importance when discussing the issue of a varying granularity of knowledge: If there are different objects that cannot be distinguished from each other (i.e. their according entry in a discernability matrix is $\emptyset$) then we need to *learn* by acquiring knowledge in form of new relations.

2.1.5 Structured Sets

Posets. A set s with a partial order relation $\sqsubseteq$ is called a *partially ordered* set $\langle s, \sqsubseteq \rangle$ or *poset*. The dual $\langle s, \sqsupseteq \rangle$ with $\sqsupseteq = \sqsubseteq^{\smile}$ is a poset, too.

Exercise 2.5 ($\lozenge$) Show that For two posets $\langle s_i, \sqsubseteq_i \rangle$, $i \in \mathbf{2}$, the product $\langle s_0 \times s_1, \sqsubseteq_\times \rangle$ is a poset with a *product partial order relation* $\sqsubseteq_\times$ on $s_0 \times s_1$ defined as follows:

$$\langle x_0, x_1 \rangle \sqsubseteq_\times \langle y_0, y_1 \rangle \text{ if } x_i \sqsubseteq_i y_i$$

For any two elements x and y of a set s, x and y are called *incomparable*, if neither $x \sqsubseteq y$ nor $y \sqsubseteq x$. If all x and y are comparable, then $\langle s, \sqsubset \rangle$ is a *total order*. Let there be two posets $\langle s_i, \sqsubseteq_i \rangle$, $i \in \mathbf{2}$. Suppose there is an isomorphism $f : s_0 \to s_1$ which preserves the ordering relation such that:

$$x \sqsubseteq_0 y \Longleftrightarrow f(x) \sqsubseteq_1 f(y).$$

Then, if x and y are incomparable in s_0, $f(x)$ and $f(y)$ must be incomparable in s_1, too.

An element $\top$ of a poset s is called a *maximal element* of s if there is no $y \in s$ such that $\top \sqsubseteq y$ and $\bot$ is called a *minimal element* of s if there is no $y \in s$ such that $y \sqsubseteq \bot$. Minimal or maximal elements need not neccessarily exist or even be unique. If there is exactly one maximal (minimal) element x of a set s, x is called the *greatest* (*least*) element. If there is a greatest element, it is often called a *unit* element denoted by $\top$; a least element is also called a *zero* element denoted by $\bot$.

For a subset $s' \subseteq s$, an element $x \in s$ is called an *upper* (*lower*) *bound* of s', if for all $x' \in s'$: $x' \sqsubseteq x$ ($x' \sqsupseteq x$). We call $x \in s$ a *least upper* (*greatest lower*) bound of $s' \subseteq s$, if x is an upper (lower) bound of s' and if for any other upper (lower) bound y of s' it holds that $x \sqsubseteq y$ ($x \sqsupseteq y$). The dual concepts of upper and lower bounds carry over to dual posets.

Lattices. A poset $\langle s, \sqsubseteq \rangle$ is called a *lattice* if every subset $\{x_0, x_1\} \subseteq s$ has a greatest lower and a least upper bound in s. The least upper bound of x_0 and x_1 is denoted $x_0 \sqcup x_1$ and is called the *join* of x_0 and x_1. The greatest lower bound of x_0 and x_1 is denoted $x_0 \sqcap x_1$ and is called the *meet* of x_0 and x_1.

Example 2.8 For any finite set s, $\mathbf{2}^s$ is a lattice with $\subseteq$ as order relation and $\cup$ and $\cap$ as join and meet. ●

The following equivalences show how order relations are connected to meet and join operators:

$$x \sqcup y = y \iff x \sqsubseteq y \quad \text{and} \quad x \sqcap y = x \iff x \sqsubseteq y \tag{2.20}$$

Meet and join are idempotent, commutative, associative and absorbing. Lattices which have a greatest and a least element are called *bounded*. and every finite lattice is boundedAs we have seen in the axiomatisation if integer arithmetic through set theory, the correspondence to sets and integers is trivial. We conclude this section on set theoretic structures with a small but very important example:

Example 2.9 We consider the set $\mathbf{z}$ of the first z natural numbers (including 0). There exists a natural bounded lattice, which looks as follows:

$$\mathbf{0} \subseteq \mathbf{1} \subseteq \mathbf{2} \subseteq \cdots \subseteq \mathbf{z}$$

where $x \sqsubseteq y :\iff \mathbf{x} \subseteq \mathbf{y}$. We define $x \sqcup y := \mathbf{x} \cup \mathbf{y}$. Then,

$$x \sqcup y = y \iff x \sqsubseteq y$$

Similarly, we define $x \sqcap y := \mathbf{x} \cap \mathbf{y}$. If we read the set names x, y as natural numbers (including 0), $\sqcup$ corresponds to the max operator and $\sqcap$ to the min operator. From this we can conclude that $\langle \mathbf{z}, \sqcup, \sqcap \rangle$ is a distributive lattice with least element 0 and largest element z. Next, we define a *complement* operation $\mathbf{x} \cup \overline{\mathbf{x}} = \mathbf{z} \iff x + \overline{x} = z$. So $\overline{\mathbf{x}}$ is the set of elements missing to make $\mathbf{x}$ equal to $\mathbf{z}$. In other words, y complements x with respect to z. One can easily verify that $\overline{x} \sqcap x = 0$.[2] ●

This example motivates the definition of a *Boolean algebra*:

Boolean Algebra

Definition 2.7 — Boolean Algebra.
The structure $\langle s, \sqcap, \sqcup, ^{-} \rangle$ is called a *Boolean Algebra*, if $\langle s, \sqcap, \sqcup \rangle$ is a lattice which satisfies

$$x \sqcap (y \sqcup z) = (x \sqcap y) \sqcup (x \sqcap z) \quad \text{and} \quad x \sqcup (y \sqcap z) = (x \sqcup y) \sqcap (x \sqcup z),$$

there exist a least element $0 \in s$ and a greatest element $1 \in s$ and for every $x \in s$ there exists $\overline{x}$ with

$$x \sqcap \overline{x} = 0 \quad \text{and} \quad x \sqcup \overline{x} = 1.$$

A Boolean algebra is a distributive lattice with complement. ●

[2] Note that this is true in Boolean algebras only. For any other algebra our definition of complementation satisfies the definition of the reltive pseudo-complement.

Exercise 2.6 (◊) Give a fourth interpretation of this algebra along the definitions of $\sqsubseteq, \subseteq$, and $\leq$ from the previous example!—Hint: Think logic.

Exercise 2.7 (♦) Look up "Heyting algebra" in literature and relate the definition of relative pseudocomplements to example 2.9 ! Focus on the sentence that "$\overline{\mathbf{x}}$ is the set of elments in **z** that are missing to make **x** equal to **z**"!

2.1.6 Relation Algebra

Without notifying the reader we implicitely introduced some ideas from relation algebra. An abstract algebra of relations is a structure of relations on a base set and operators on these relation that satisfy certain properties. So, for the sake of completeness, we now define what we already presumed earlier:

Relation Algebra

Definition 2.8 — Relation Algebra.
An algebra of relations is a structure

$$\mathfrak{R} = \left\langle \mathbf{R}, \sqcup, \sqcap, \circ, {}^{-}, {}^{\smile}, \bot, \top, \mathit{1} \right\rangle \tag{2.21}$$

with a *base set* **R**, binary operators $\sqcup, \sqcap$ and $\circ$, unary operators ${}^{-}$ and ${}^{\smile}$ and special elements $\bot, \top$ and *1* for which hold:

1. **R** together with $\sqcap$ and ${}^{-}$ form a boolean algebra,
2. **R** together with $\circ$ and *1* forms a monoid,
3. it holds that $(R^{\smile})^{\smile} = R$, $(R \circ S)^{\smile} = S^{\smile} \circ R^{\smile}$
4. ${}^{\smile}$ and $\circ$ distribute over $\sqcap$
5. and deMorgan's laws are satisfied.

The base set in relation algebra is the set of all binary endorelations over a base set: $\wp(s^2)$. The zero- and one-element in an algebra of relations are, of course, the *empty* relation $\bot$ and the *universal* relation $\top$. ●

It is very important to be able to speak about objects and sets of objects. For example, $5 \in \mathbb{N}$ and $\mathbb{N} \subset \mathbb{R}$. But as we are interested in *relations* we want to talk about the properties of different relations. So if $R \subseteq \mathbb{N} \times \mathbb{N}$ and $S \subseteq \mathbb{N} \times \mathbb{N}$, what is the difference between R and S? Or, simply speaking, it is nice to know that Clarabelle is a cow and that Pluto is a dog. It is even nicer to know that dogs are descendants of wolves and that cows are tame bovinae. But the nicest thing is to know that dogs are to wolves as cows are to bisons: they are the domesticated versions.

Example 2.10 Let $R, S \in \mathbb{N}^2$ and $xRy :\Longleftrightarrow x < y$ and $xSy :\Longleftrightarrow x \leq y$. In other words, $R =<$ and $S =\leq$ (it may look a bit odd the first time you read this). But whenever $x < y$, we also know that $x \leq y$. Therefore—and this might look even odder—we can state that $<\subset\leq$. ●

Let us take a look at a two–valued propositional logic again:

Example 2.11 Imagine a set $\mathrm{Var}_{\mathrm{PL}} = \{\mathrm{a}, \mathrm{b}, \mathrm{c}, \ldots\}$ which we call the set of propositional variables. Suppose there is total function $\alpha : \mathrm{Var}_{\mathrm{PL}} \to \mathbf{2}$. Since $\mathbf{0} \subseteq \mathbf{1}$ we know that for any $x, y \in \mathrm{Var}_{\mathrm{PL}}$

$$\alpha(x) = \mathbf{0} \Longrightarrow \alpha(x) \subseteq \alpha(y) \tag{2.22}$$

Similary, we know that if $\alpha(x) \neq \alpha(y)$, it holds that

$$\alpha(x) \cup \alpha(y) = \mathbf{1} \quad \text{and} \quad \alpha(x) \cap \alpha(y) = \mathbf{0} \tag{2.23}$$

We then define operators $\vee, \wedge, \longrightarrow$ and $\neg$ on Σ with the following properties:

$$\begin{array}{rcl} \alpha(x \wedge y) & := & \alpha(x) \cap \alpha(y) \\ \alpha(x \vee y) & := & \alpha(x) \cup \alpha(y) \\ \alpha(\neg x) & := & \{y : y \in \mathbf{2} - \{\alpha(x)\}\} \\ \alpha(x \longrightarrow y) & := & \alpha(\neg x) \cup \alpha(y) \end{array}$$

This way, we have defined syntax and semantics of propositional logic in only four lines. $\mathbf{2}$ is called the set of Boolean truth–values, written as $\mathbf{2} := \{\mathbf{0}, \mathbf{1}\}$. ●

Exercise 2.8 (♦) Define a three-valued propositional logic!

What is the benefit of relation algebra? Take a look at the definition (2.4) of properties of relations. All the properties were defined *element-* or *pointwise*. For example, to show that R is reflexive we have to show that xRx for all $x \in \mathrm{dom}(R)$. This might become a rather cumbersome task with large or even unknown domains. Instead of examining a relation elementwise, we can simply express its properties by equations in relational calculus. So if R is reflexive, then it holds that $\mathit{1} \subseteq R \Longleftrightarrow \mathit{1} \cup R = R$. Symmetry means that $R = R^{\smile}$ and transitivity that $R \circ R \subseteq R$.

So whenever we can prove that for some relation R the three equations are true we know it is a equivalence relation—without taking a closer look at a single object of our domain. As an example, consider the following proposition:

> A reflexive, difunctional relation is an equivalence relation.

Exercise 2.9 (♦) Prove the proposition using the standard set-theoretic properties!

Using the laws of relation algebra, the proof is very simple: Let R be difunctional and reflexive. Then, it holds that $\mathit{1} \subseteq R$ and $R \circ R^{\smile} \circ R = R$. In order to show it is an equivalence relation, we have to show that it is also symmetric $R \subseteq R^{\smile}$ and transitive $R \circ R \subseteq R$. We assume transitivity (Exercise!) and show that symmetry follows:

$$\begin{array}{rcll} R & = & R \circ R^{\smile} \circ R & \\ & \supseteq & \mathit{1} \circ R^{\smile} \circ \mathit{1} & \{\!\mid \text{by reflexivity} \mid\!\} \\ & = & R^{\smile} & \{\!\mid \text{by identity law} \mid\!\} \end{array}$$

Therefore, $R \supseteq R^{\smile}$. Similarly,

$$\begin{array}{rcll} R^{\smile} & = & (R \circ R^{\smile} \circ R)^{\smile} & \\ & \supseteq & (\mathit{1} \circ R^{\smile} \circ \mathit{1})^{\smile} & \{\!|\text{ by reflexivity }|\!\} \\ & = & R^{\smile\smile} & \{\!|\text{ by identity law }|\!\} \\ & = & R & \{\!|\text{ by converse }|\!\} \end{array}$$

which means $R \subseteq R^{\smile}$ and completes the proof.[3] Once we understood the general laws of relation algebra, we are not only able to talk *about* things, but also *about the relations* between them. Then, the set of all relations again is a set with relations of objects—so we can examine relations between relations, and so on. This is achievement is not just art for art's sake but it is a *fundamental* requirement for knowledge discovery:

> **Relations and Knowledge**
> If knowledge is defined by relations, then learning is about reasoning with relations on relations.

A relational view on knowledge is a view that delivers a vocabulary to speak about it for free.[4]

2.2 Knowledge Structures

We now examine families $\mathbf{P}, \mathbf{Q}, \mathbf{R}$ of equivalence relations over a base set s.

2.2.1 Concepts, Equivalence Relations, and Knowledge

Consider a set s and relations C, S as in example (2.3). The knowledge represented by these relations can be used to answer several questions:

1. What colour is a □?
   Hence, we examine $\mathbb{K}(C)$ and find $f_c(\Box)$'s value: □ is *white*!

2. What shape is a □?
   We examine the kernel matrix for S and find that $f_s(\Box) = \mathit{square}$.

3. But the interesting thing is that a □ is a *white square*:

$$\Box \in [\Box]_C \cap [\Box]_S$$

[3] As you can see, this proof is very, very simple. It is, in terms of notational and logic effort, much simpler than the proof on the previous page. But in order to find such a simple and short proof, it takes a human a few years of experience. The biggest advantage of all is that this proof can be carried out mechanically using an automated theorem proving system. Since such a system lacks all the intuition and experience, the proof takes about 75 steps there - but it is carried out in less than a quarter of a second.

[4] With all of its implications: While Boolean algebras are still decidable, relation algebras are *un*decidable.

We answered the question for white *and* squarish objects by intersecting the respective equivalence classes. How can we use our knowledge of the according equivalence relations? One approach is to create an overlay of the respective matrices.

$$\begin{array}{ccccccc}
C & \Diamond & \bullet & \blacksquare & \square & \circ & \blacklozenge \\
\Diamond & w & & & w & w & \\
\bullet & & b & b & & & b \\
\blacksquare & & b & b & & & b \\
\square & w & & & w & w & \\
\circ & w & & & w & w & \\
\blacklozenge & & b & b & & & b
\end{array}
\quad \text{and} \quad
\begin{array}{ccccccc}
S & \Diamond & \bullet & \blacksquare & \square & \circ & \blacklozenge \\
\Diamond & r & & & & & r \\
\bullet & & c & & & c & \\
\blacksquare & & & s & s & & \\
\square & & & s & s & & \\
\circ & & c & & & c & \\
\blacklozenge & r & & & & & r
\end{array}$$

The overlay of the kernel matrices $\mathbb{K}(C)$ and $\mathbb{K}(S)$ then is:

$$\begin{array}{ccccccc}
C \oplus S & \Diamond & \bullet & \blacksquare & \square & \circ & \blacklozenge \\
\Diamond & wr & & & w & w & r \\
\bullet & & bc & b & & c & b \\
\blacksquare & & b & bs & s & & b \\
\square & w & & s & ws & w & \\
\circ & w & c & & w & wc & \\
\blacklozenge & r & b & b & & & br
\end{array}$$

Finally, we consider the relation $R := C \cap S$ which discriminates objects by colour *and* shape and draw the according equivalence classes as $\mathbb{M}(C \cap S)$ (left matrix). We then identify colour and shape information by selecting only those entries in $\mathbb{K}(C) \oplus \mathbb{K}(S)$ for which the left matrix carries **1** and obtain the result in the right matrix:

$$\begin{array}{ccccccc}
R & \Diamond & \bullet & \blacksquare & \square & \circ & \blacklozenge \\
\Diamond & \mathbf{0} & \mathbf{0} & \mathbf{0} & & \mathbf{0} & \mathbf{0} \\
\bullet & \mathbf{0} & \mathbf{0} & \mathbf{0} & & \mathbf{0} & \mathbf{0} \\
\blacksquare & \mathbf{0} & \mathbf{0} & \mathbf{0} & & \mathbf{0} & \mathbf{0} \\
\square & & & & \mathbf{1} & & \\
\circ & \mathbf{0} & \mathbf{0} & \mathbf{0} & & \mathbf{0} & \mathbf{0} \\
\blacklozenge & \mathbf{0} & \mathbf{0} & \mathbf{0} & & \mathbf{0} & \mathbf{0}
\end{array}
\qquad
\begin{array}{ccccccc}
\bigcap & \Diamond & \bullet & \blacksquare & \square & \circ & \blacklozenge \\
\Diamond & & & & & & \\
\bullet & & & & & & \\
\blacksquare & & & & & & \\
\square & & & & ws & & \\
\circ & & & & & & \\
\blacklozenge & & & & & &
\end{array}$$

This means that $\square$ is a *white square*.

2.2.2 Operations on Equivalence Relations

A corollary from the observation that every equivalence relation induces a partition (the quotient) and vice versa is that two relations are equal, iff their quotients are equal. Let $R, S \in \mathbf{R}$ be two equivalence relations. Then,

$$\begin{aligned} R = S &\iff xRy \longleftrightarrow xSy \\ &\iff \{\langle x, y\rangle : xRy\} = \{\langle x, y\rangle : xSy\} \\ &\iff \{[x]_R : x \in s\} = \{[x]_S : x \in s\} \\ &\iff s/R = s/S \end{aligned}$$

Therefore, the intersection of equivalence relations can be defined in terms of intersections of equivalence classes:

Intersection of equivalence relations

Definition 2.9 — Intersection of equivalence relations.
For n equivalence relations $R_i \in \mathbf{R}$, $\bigcap_{i \in \mathbf{n}} R_i$ is an equivalence relation, too:

$$\begin{aligned} \bigcap_{i \in \mathbf{n}} R_i &= R_0 \cap R_1 \cap \cdots R_{n-1} \\ &= \{\langle x, y\rangle : xR_0y\} \cap \{\langle x, y\rangle : xR_1y\} \cap \cdots \cap \{\langle x, y\rangle : xR_{n-1}y\} \\ &= \{\langle x, y\rangle : xR_0y \wedge xR_1y \wedge \cdots \wedge xR_{n-1}y\} \end{aligned}$$

●

It also holds that

$$\begin{aligned} [x]_{R_0 \cap \cdots \cap R_n} &= \{y : x(R_0 \cap \cdots \cap R_{n-1})y\} \\ &= \{y : xR_0y\} \cap \cdots \cap \{y : xR_ny\} \\ &= [x]_{R_0} \cap \cdots \cap [x]_{R_n} \end{aligned}$$

By intersecting equivalence relations we obtain further, *smaller* equivalence relations with *finer* classes. This is worth a second thought: what does it mean for a certain equivalence relation to be a subset of another? Let us take a closer look:

$$\begin{aligned} R_0 \subseteq R_1 &\iff \{\langle x, y\rangle : xR_0y\} \subseteq \{\langle x, y\rangle : xR_1y\} \\ &\iff xR_0y \longrightarrow xR_1y \end{aligned}$$

Example 2.12 Consider the following equivalence relation R_3 on our example set:

R_3	◊	•	■	□	○	♦
◊	□			□		
•		△				△
■			◇			
□	□			□		
○					◯	
♦		△				△

$$= \left\{ \begin{array}{llll} \langle 0,0\rangle_{\square}, & \langle 0,3\rangle_{\square}, & \langle 3,0\rangle_{\square}, & \langle 3,3\rangle_{\square}, \\ \langle 1,1\rangle_{\triangle}, & \langle 1,5\rangle_{\triangle}, & \langle 5,1\rangle_{\triangle}, & \langle 5,5\rangle_{\triangle}, \\ \langle 2,2\rangle_{\diamond}, & \langle 4,4\rangle_{\bigcirc} & & \end{array} \right\}$$

If we recall the definition of C we can easily verify that $R_3 \subset C$:

C	◊	●	■	□	○	⧫
◊	w			w	w	
●		b	b			b
■		b	b			b
□	w			w	w	
○	w			w	w	
⧫		b	b			b

$$C = R_3 \cup \left\{ \begin{array}{c} \langle 1,2\rangle, \langle 2,1\rangle, \langle 1,4\rangle, \langle 4,1\rangle, \\ \langle 2,5\rangle, \langle 5,2\rangle, \langle 3,4\rangle, \langle 4,3\rangle \end{array} \right\}$$

In other words, R_3 is *finer* than C, and C is *coarser* than R_3; or, more formally, $R_3 \subseteq C \iff xR_3y \to xCy$.

This important result deserves a theorem:

Theorem 2.1 (Granularity of equivalence relations) Let there be two equivalence relations R_0 and R_1. Then:

1. $R_0 \subseteq R_1 \iff xR_0y \longrightarrow xR_1y$
2. $R_0 \subseteq R_1 \Longrightarrow [x]_{R_0} \subseteq [x]_{R_1}$

Exercise 2.10 ◊⧫ Prove!

There are, of course, more operations on (equivalence) relations than just intersections (i.e. conjunction): For equivalence relations $R, R_0, R_1 \subseteq s \times s$,

1. $\overline{R}$ is not an equivalence relation, but $\overline{R} \cup 1_s$ is.
2. $R^{\smile}$ is an equivalence relation.
3. $R_0 \cup R_1$ is not an equivalence relation.
4. $R_0 \circ R_1 = \{\langle x, z\rangle : \exists y : xR_0yR_z\}$ is not an equivalence relation.

So far, we talked about *objects*, sets or classes of objects and their relations to each other. But knowledge is made of *relations*, not of collections of things. Therefore, we call $\langle s, \mathbf{R}\rangle$ a *knowledge base.* To conclude this chapter on knowledge structures, we finally discuss the relations *between* different knowledge bases.

2.2.3 Indiscernability and Knowledge

Two objects x and y are *indiscernible* with respect to an equivalence relation R, if xRy. As a special case, if $x = y$ are equal, because $=$ is an equivalence relation: Two things that are the same are indiscernible from each other. With a set $\mathbf{R}$ of equivalence relations, $\bigcap \mathbf{R}$, is an equivalence relation, too. We find that if $\bigcap \mathbf{R} \in \mathbf{R}$, it is a minimal element: $\forall R \in \mathbf{R} : \bigcap \mathbf{R} \subseteq R$. It creates the finest partition on the base set s. If xRy for all $R \in \mathbf{R}$, then x and y are indiscernible with respect to $\mathbf{R}$ and it also holds that $x(\bigcap \mathbf{R})y$. The higher the resolution of our knowledge $\mathbf{R}$ on the objects of our domain, the less indiscernible objects there are—and the coarser our knowledge, the more indiscernible objects we have.

Indiscernability relation

Definition 2.10 — Indiscernability relation.
Let there be a set of equivalence relations $\mathbf{R}$. Then, we call

$$\bar{\bar{\mathbf{R}}} = \bigcap_{R \in \mathbf{R}} R = \bigcap \mathbf{R}$$

the *indiscernability relation* over $\mathbf{R}$. ●

Elements of s/R for any $R \in \mathbf{R}$ are called *elementary categories*. They correspond to R–equivalence classes and can be identified by appropriate kernels. Elements of $s/\bar{\bar{\mathbf{R}}}$ are called *basic categories*. Elementary categories (R–equivalence classes) are unions of basic categories ($\bar{\bar{\mathbf{R}}}$–equivalence classes), and a basic category is always a subset of exactly one R–equivalence class.

Exercise 2.11 ◊ Compute $\bar{\bar{\mathbf{R}}}$ for $\mathbf{R} = \{R_i : i \in \mathbf{3}\}$ as defined in example 2.2!

In other words, elementary categories are equivalence classes generated by some equivalence relation $R \subseteq \mathbf{R}$. Given an information system $\langle s, \mathbf{F}, V_{\mathbf{F}} \rangle$ we have a knowledge base $\mathbf{R}$ with $\mathbf{R} = \{R_f : f \in \mathbf{F}\}$. Every elementary category $[x]_{R_f}$ then has a "name", which is x's value under f or, simply speaking, the entry in R_f's kernel matrix, $f(x)$. Basic classes are sets of objects that are not discernible by any of the $R_f \in \mathbf{R}$. So if x and y are elements of the same basic class then $f(x) = f(y)$ for all $f \in \mathbf{F}$.

Example 2.13 Let $\mathbf{R} = \{C, S\}$ as in example 2.5. Then, $\bar{\bar{\mathbf{R}}} = \bigcap\{C, S\} = C \cap S$ is:

$$\begin{aligned} C \cap S &= \left\{ \begin{array}{lll} \langle 0,0\rangle_{\mathrm{b}}, & \langle 3,3\rangle_{\mathrm{b}}, & \langle 0,3\rangle_{\mathrm{b}}, \\ \langle 3,0\rangle_{\mathrm{b}}, & \langle 3,4\rangle_{\mathrm{b}}, & \langle 4,3\rangle_{\mathrm{b}}, \\ \langle 0,4\rangle_{\mathrm{b}}, & \langle 4,0\rangle_{\mathrm{b}}, & \langle 4,4\rangle_{\mathrm{b}}, \\ \langle 1,1\rangle_{\mathrm{w}}, & \langle 2,2\rangle_{\mathrm{w}}, & \langle 5,5\rangle_{\mathrm{w}}, \\ \langle 1,2\rangle_{\mathrm{w}}, & \langle 2,1\rangle_{\mathrm{w}}, & \langle 2,5\rangle_{\mathrm{w}}, \\ \langle 5,2\rangle_{\mathrm{w}}, & \langle 1,5\rangle_{\mathrm{w}}, & \langle 5,1\rangle_{\mathrm{w}} \end{array} \right\}_{R_0} \cap \left\{ \begin{array}{ll} \langle 0,0\rangle_{\mathrm{c}}, & \langle 5,5\rangle_{\mathrm{c}}, \\ \langle 0,5\rangle_{\mathrm{c}}, & \langle 5,0\rangle_{\mathrm{c}}, \\ \langle 1,1\rangle_{\mathrm{s}}, & \langle 4,4\rangle_{\mathrm{s}}, \\ \langle 1,4\rangle_{\mathrm{s}}, & \langle 4,1\rangle_{\mathrm{s}}, \\ \langle 2,2\rangle_{\mathrm{l}}, & \langle 3,3\rangle_{\mathrm{l}}, \\ \langle 2,3\rangle_{\mathrm{l}}, & \langle 3,2\rangle_{\mathrm{l}} \end{array} \right\}_{R_1} \\ &= \left\{ \begin{array}{lll} \langle 0,0\rangle_{\mathrm{bc}}, & \langle 5,5\rangle_{\mathrm{wc}}, & \langle 1,1\rangle_{\mathrm{ws}}, \\ \langle 4,4\rangle_{\mathrm{bs}}, & \langle 2,2\rangle_{\mathrm{wl}}, & \langle 3,3\rangle_{\mathrm{bl}} \end{array} \right\}_{C \cap S} = \mathit{1}_s \end{aligned}$$

Equivalently, we can take a look at the partitions instead:

$$\begin{aligned} s/R_0 &= \{\{0,3,4\},\{1,2,5\}\} \\ s/R_1 &= \{\{0,5\},\{1,4\},\{2,3\}\} \\ s/(R_0 \cap R_1) &= \{\{0\},\{1\},\{2\},\{3\},\{4\},\{5\}\} \end{aligned}$$

Neither R_0 nor R_1 are capable of discriminating all object; whereas their intersection is. ●

Whenever the indiscernability relation equals the identity, $s/\bar{\bar{\mathbf{R}}} = \{\{x\} : x \in s\}$, then knowledge $\mathbf{R}$ is maximal in a sense that we cannot have any more knowledge because we can already discriminate any two differents objects from each other. But what happens if we chose *different* sets **P** and **R** of equivalence relations? It could well be that $s/\bar{\bar{\mathbf{P}}} = s/\bar{\bar{\mathbf{R}}}$. Then, the information content of **P** and **R** seems to be the same as they create the same quotients. It could also be, that one knowledge base is finer than the other: If there is some $x \in s$ for which

$$[x]_{\bar{\bar{\mathbf{P}}}} \subset [x]_{\bar{\bar{\mathbf{R}}}}$$

then, **P** creates a smaller basic category including x than **R** does. Therefore, there are less objects that are **P**–indiscernible from x than there are **R**–indiscernible objects of x.

Equivalent Knowledge

Definition 2.11 — Equivalent Knowledge.
Let there be two knowledge bases with a base set s and families of equivalence relations **P** and **R**. **P** and **R** are *equivalent*, if:

$$\mathbf{P} \cong \mathbf{R} \quad :\Longleftrightarrow \quad \bar{\bar{\mathbf{P}}} = \bar{\bar{\mathbf{R}}} \Longleftrightarrow s/\mathbf{P} = s/\mathbf{R} \tag{2.24}$$

R is called *coarser* (*more general*) than **P**, if

$$\mathbf{P} \preceq \mathbf{R} \quad :\Longleftrightarrow \quad \bar{\bar{\mathbf{P}}} \subseteq \bar{\bar{\mathbf{R}}} \Longleftrightarrow [x]_{\bar{\bar{\mathbf{P}}}} \subseteq [x]_{\bar{\bar{\mathbf{R}}}} \tag{2.25}$$

If **R** is more general than **P**, then **P** is *finer* than **R**. ●

Knowledge representation is a research discipline on its own, and so is relational concept analysis.
Yet, there is still much more to knowledge. Human knowledge and human skills are no entities but rather processes that are massively parallel and unsynchronised. Accordingly, we have artificial neural networks as a simulation and we have, e.g. backpropagation for learning such networks. Our knowledge of the world is not very crisp, either. Concepts are vague, and rules are fuzzy, too. And many observations and conclusions drawn from them are probabilistic. This has lead to a multitude of different knowledge representation and inferencing methods. And, accordingly, to many different learning methods as well.
We now have a proper understanding of the required concepts and we have acquired the neccessary skills to study *knowledge* and *representations* thereof more detailed.

Chapter 3

From Data to Hypotheses

No software without a program, no program without an algorithm. No algorithm without a theory and no theory without a clear syntax and semantics. In this chapter we define the fundamental concepts that we need to speak about machine learning in a clear language without too much confusion.

If we try to put all important information about machine learning in just a small box, it would look like this:

Machine Learning
Machine learning is concerned with the problem to induce a *concept* from a *sample* of instances of our *domain.* Given a classification, the task is to define a mapping that approximates an unknown *target function* which assigns to each object a target class label.
The outcome is a hypothesis h of a certain *quality*; and the process of inducing such a hypothesis crucially depends on the *representation* of our domain.

This rather rough picture is described in detail in the following sections.
First of all we need to specify what we will be talking about and which terms we shall use so as to avoid too much confusion.

3.1 Representation

Machine Learning and Knowledge Discovery is concerned with:

- grouping *objects* (like entities, processes, atoms, complex structures, etc)
- from a *domain* (i.e. a set of such objects)
- into *target concepts*
- with respect to their *properties* and/or the *classes* they belong to

First of all, we need to ask ourselves how to *represent* our knowledge about the world. The part of the world that we live in and which we shall reason about is called the *domain*. In order to be able to talk about objects of the domain we need to have representations thereof (and their properties).[1]

Domain, Universe, Representation

Definition 3.1 — Domain, Universe, Representation.
Let $\mathfrak{D}$ denote our *domain* which is the part of the world that we are interested in. A *representation* is a morphism from the domain into a structure of objects called *representation space* or *universe* $\mathfrak{U}$:

$$\rho : \mathfrak{D} \to \mathfrak{U} \tag{3.1}$$

The base set of $\mathfrak{U}$ is denoted as U. ●

Note that for $x, y \in \mathfrak{D}$, $x = y$ does not imply that $\rho(x) =_{\mathfrak{U}} \rho(y)$: First, it can be that $\rho(x) \neq \rho(y)$. And second, even if $\rho(x) = \rho(y)$, it is not necessarily the case that $\rho(=) = \rho(=_{\mathfrak{U}})$.

Example 3.1 Let $\mathfrak{D} = \mathbb{Q}$ and $\mathfrak{U} = \mathbb{N}$. One possible representation of rational numbers as natural numbers is defined by $\rho(x) = \lceil |x| \rceil$. Then, e.g. $\rho(-6.81) = \rho(\frac{20}{3}) = \rho(\sqrt{48})$. ●

Exercise 3.1 (◊) (a) Define two more (reasonable) representation functions! (b) Define a reasonable $\rho : \mathbb{R} \to \mathbb{N}$. (c) Find a reasonable representation for the complex numbers $\mathbb{C}$ by more simple structures!

Exercise 3.2 (♦) Discuss the properties of the representations you defined in the previous exercise. Are order relations preserved? What about operations?

Of course, our world $\mathfrak{D}$ has much more structure than just a set and, of course, so has $\mathfrak{U}$.

Concept, Class

Definition 3.2 — Concept, Class.
A *concept* is a meaningful subset $C \subseteq \mathfrak{D}$ of objects in our domain. A *(representation) class* is a subset $c \subseteq U$. ●

It is very important to understand that we defined a representation class independently from concepts. If we had defined a representation class c to be C's image under ρ, we would have to have knowledge about $\mathfrak{D}$ and ρ. It might occur a bit odd to the reader, but we do not know what $\mathfrak{D}$ looks like:

> **Representation Gap**
> Any representation or any model is an image of the domain, and the way the image is drawn, is always a lossy process as we shall see later. More importantly, knowledge discovery means to discover rules by *looking at the data* we are provided with. It means that all we have is $\mathfrak{U}$, we do not even know anything about ρ. Knowledge discovery takes place when we interpret the result (a description of a class c) back in $\mathfrak{D}$.

Accordingly, $C \neq D$ does not imply that $\rho(C) \neq_{\mathfrak{U}} \rho(D)$.

[1] Actually, we never ever talk about domains but always about our model of parts of the world which we assume to be "true".

Example 3.2 Imagine $\mathfrak{D} = \mathbb{R}$ and $\mathfrak{U} = \mathbb{N}$. Let there be two concepts in $\mathbb{R}$: the set of non-natural rational numbers $Q = \mathbb{Q} - \mathbb{N} \subset \mathbb{R}$ and the set of non-natural real numbers $R = \mathbb{R} - \mathbb{N} \subset \mathbb{R}$. We now define a partial representation $\rho : \mathbb{R} \to \mathbb{N}$ by

$$\rho(x) = \begin{cases} x, & \text{if } x \in \mathbb{N} \\ \mathit{undef} & \text{else} \end{cases}$$

Then, $Q \subset R$ but $\rho(Q) = \rho(R) = \emptyset$. ●

We already stated that concepts usually have a meaning. This is why we are able not only to collect all blue things and call the set of all blue things *blue*. Instead, we can also *explain* or *describe* the abstract property of something *being blue*. This is similar to having a general idea of "blueishness" and the ability to communicate this idea to someone else. This idea does not refer to all instances of the set of things that share the property of being blue, but to the intensional meaning of the concept "blueishness" itself. This is why we need to discriminate concepts from representation classes and intensional meaning from extensional enumeration.

Example 3.3 In his *Tractatus Logico-philosophicus*, the philosopher Ludwig Wittgenstein tried to establish a logically sound theory of meaning. In this case, human language was the representation space he examined. First, he defined the "world" to be everything "that is the case". His "world" corresponds to what we call domain. He also stated that models ("logical pictures") are based on elementary propositions about things (which correspond to characteristic functions of classes). Of course, he also came to the conclusion that there are things in the world that cannot be expressed in terms of our language. This left him speechless for a long time. Then, after several years more he finally concluded that *meaning of a word is its use*.
Linguists and especially those people dealing with lexical semantics (i.e. the meaning of the words or the units of language) face a similar problem. They eventually agreed that the meaning of the word "scooter" is *scooter'* (pronounced "scooter-prime"). ●

But still, there is more to concepts than just sets: Concepts obviously contain *more* information than sets of objects which are represented by representation classes. Therefore, we define *concept representations*:

Representation of a concept

Definition 3.3 — Representation of a concept.
A concept $C \in \mathfrak{D}$ is represented by classes of objects:

$$\rho : C \mapsto c \quad \text{where} \quad c = \bigcup_i \bigcap_j c_{i,j} \tag{3.2}$$

with $c_{i,j} \subseteq U$. ●

The idea behind joins of class intersections was discussed in the last chapter on knowledge representation already; here it demonstrates the difference between the *structure* of representation space $\mathfrak{U}$ and the mere set of representable objects U. While all the classes $c_{i,j} \subseteq U$ simply are representation classes, the complex expression $\bigcup \bigcap c_{i,j} \in \mathfrak{U}$ requires a structure that allows us to operate on classes to yield a description of a concept representation.[2] Of course, when evaluating the complex expression c, its value becomes a simple subset of U, too. Nevertheless, the difference is important even though all the important processes of knowledge discovery take place in representation space only where concepts from $\mathfrak{D}$ don't matter and expressions are inherently evaluated. Hence, we also use the words "concept representation", "representation class", and "class" interchangeably when clear from context.

Example 3.4 The concept $C =$"white square" subsumes many different objects, some of which can be represented as $\square, \Box, \Diamond$. The representation class $\{\square, \Box, \Diamond\}$ can be described by the concept $\{\square, \circ, \Diamond, \Box, \bigcirc, \lozenge\} \cap (\{\blacksquare, \blacksquare, \square, \Box\} \cup \{\blacklozenge, \Diamond, \blacklozenge, \lozenge\})$, i.e. all squares or rhomboids that are white. With $U = s$ as in example 2.3, we have $\rho($"white square"$) = \{\square\}$. ●

Now that we have different classes, we can define *sets of classes*. For example, the set $\{r, g, b\}$ with r, g, and b being classes of red, blue and green objects, respectively would be a classification of objects with respect to their colour.

Classification, $\mathfrak{c}$

Definition 3.4 — Classification, $\mathfrak{c}$.
A *classification* $\mathfrak{c}$ of objects from a set $s \subseteq U$ is a family of k classes $c_i \subseteq s$:

$$\mathfrak{c} = \{c_0, c_1, \ldots, c_{k-1}\} \tag{3.3}$$

with $c_i \subseteq s$ and $\bigcup c_i = s$, $i \in \mathbf{k}$. ●

For each c_i, we can define $\chi(c_i) : s \to \mathbf{2}$ which for each x delivers $\mathbf{1}$ if and only if $x \in c_i$ (note that $\chi(c_i)$ is not total on U). Usually, it is implicitly assumed, that:

$$s = U = \bigcup_{1 \leq i \leq k} c_i \quad \text{and} \quad c_i \cap c_j \neq \emptyset \longrightarrow c_i = c_j. \tag{3.4}$$

which makes all classes of a classification mutually exclusive and their respective characteristic functions total on U. Of course, concepts are not disjoint; quite often they overlap. Even if they did not overlap, disjointness would be hard to determine as concepts usually have rather vague boundaries (recall that concepts have a meaning).
The disjointness assumption is very popular since a *classifier* then becomes a function:

Classifier

Definition 3.5 — Classifier.

[2] Of course, c is a class too—but readers familiar with logic will have noticed that the definition of a concept representation simply corresponds to the conjunctive normal form of a propositional logic formula with the propositional variables defined by the *elementary categories*.

A *classifier* is a function $f_{\mathfrak{c}} : s \to \mathbf{k}$ such that

$$f_{\mathfrak{c}}(x) = i \in \mathbf{k} :\Longleftrightarrow x \in c_i \Longleftrightarrow \chi(c_i)(x) = 1 \tag{3.5}$$

In many cases we will consider binary classification problems with $\mathbf{k} = \mathbf{2}$. Then,

$$\mathfrak{c} = \{c, \overline{c}\}$$

and we abbreviate

$$f_c(x) \quad := \quad \chi(c)(x) \tag{3.6}$$

Simply speaking, a classifier is a function that implements a characteristic function of a representation class. ●

The problem is that we want to find such a function which shall be "correct" on object representations that we do not know while learning this function. Now that we have a rather clear language to talk about our domain and its elements and relations between them, we can reformulate our first working definition of machine learning into:

Concepts & Learning Classifiers
Concept learning means to answer the questions:

> What is the common property of a set of objects? What makes them different from all other objects? Which yet unknown concept do the objects belong to?

Learning a classifier means to answer the question:

> Given several classification examples, what are the underlying rules that we need to correctly classify new and unseen cases? How can we find a function that approximates a perfect classifier?

To answer this question and solve the problem, we

1. represent the problem domain $\mathfrak{D}$ in representation space $\mathfrak{U}$ with least possible loss through a representation function ρ

2. try to describe all object representations $\rho(x_i)$ in terms of representation classes $c \subseteq U$

3. and hope we can find some representation concepts that have some proper meaning when interpreted with respect to $\mathfrak{D}$.

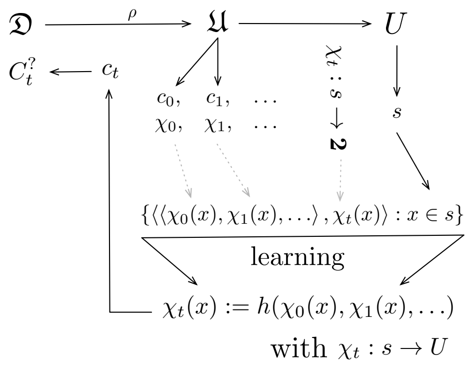
To satisfy the requirement of a "proper" meaning, our learning process should deliver a hypothesis for the classifier such that the following is true:

1. all *known* and most *unknown* objects $x_1, x_2, \ldots, x_m \in U$, to which we would attribute the property C, are represented by members of the concept representation.

2. For all non-C-ish objects y_i their respective representations $\rho(y_i)$ do not fall into c

Then, with some luck, the description of concept representations c in $\mathfrak{U}$ has a reasonable concept C' as preimage in $\mathfrak{D}$. Let us briefly summarise what we have learned so far:

The Data Mining Process
Take a look at figure 3.1. We build a model $\mathfrak{U}$ with a base set U to describe entities and relations of our domain $\mathfrak{D}$. Representation classes are used to define concepts and with characteristic functions and a subset of objects ($\mathbf{s}$) we learn a hypothesis h that shall approximate some concept c. The big question is, whether h actually is reasonable at all when interpreted in $\mathfrak{D}$.



$\mathfrak{D}$
ρ
$\mathfrak{U}$
U
$C_t^?$
c_t
$c_0,\ c_1,\ \ldots$
$\chi_0,\ \chi_1,\ \ldots$
$\chi_t : s \to \mathbf{2}$
s
$\{\langle\langle\chi_0(x), \chi_1(x), \ldots\rangle, \chi_t(x)\rangle : x \in s\}$
learning
$\chi_t(x) := h(\chi_0(x), \chi_1(x), \ldots)$
with $\chi_t : s \to U$

Figure 3.1: Domain, universe, representation

Example 3.5 Let the domain be the world of our solar system and the concepts and names that we have for all the objects and phenomena that occur there. There are objects like mars and jupiter and io and pluto, venus, halley and so on. There are concepts, too: *Planet*, *Morningstar*, *Moon*, or *Asteroid*. We now represent every single object of our solar system by the name of the object it is orbiting, the mean orbit radius (in AEs), the time it takes for one rotation around the center of the orbit (in days) and the mass of the object (in earth masses). For the sun, ρ delivers a representation as follows:

$$\langle \text{sun}, 0, 25, 333 \cdot 10^3 \rangle$$

which means that it only rotates around itself in 25 days and it is 333,000 times heavier than the earth. The earth itself would be represented as: $\langle \text{sun}, 1, 365, 1\rangle$. There are two concepts in $\mathfrak{D}$ that usually map onto the same representation:

$$\begin{aligned} \rho(\mathit{Morningstar}) = \rho(\mathit{Eveningstar}) &= \rho(\text{venus}) \\ &= \langle \text{sun}, 0.7, 225, 0.8 \rangle \end{aligned}$$

So once we talk about venus, we cannot determine whether we speak about the concept *MorningStar* or the *EveningStar* (except we had extra knowledge about the current time of the day). To make things even more complicated, there is another planet,

$$\rho(\text{mercury}) \quad = \quad \langle \text{sun}, 0.05, 88, 0.8 \rangle$$

which can be seen both at dusk and dawn, too. So if we invent a new representation concept

$$c := \{\langle \text{sun}, 0.7, 225, 0.8 \rangle, \langle \text{sun}, 0.05, 88, 0.8 \rangle\} = \{\rho(\text{mercury})\} \cup \{\rho(\text{venus})\},$$

c can be interpreted as the concept of *PlanetsOneCanSeeAtDuskOrDawn*. ●

Exercise 3.3 (◊) The unknown concept *Planet* could be learned, if we compare all the object representations of mars, jupiter, pluto and earth and try to find a feature that discriminates this set from the set of moon, io, or triton. How?

Exercise 3.4 (◊♦) Modify the ρ such that each object's representation also includes some knowledge about its mean surface temperature. Then, develop two different classifiers f_h and f'_h for the binary representation concept *MostlyHarmless* where the earth shall be the only instance.

Usually, several representation processes are concatenated or intertwined without being noticed. But as we are used to explain new observations in terms of our already existing model of the world, we sometimes fail to recognise something new as being new (and sometimes we don't even recognise it at all). Our tendency to put things into relation already makes us to represent things in a way they are comparable (even if they are not). Very popular formal analogies to such processes is the application of discretisation and quantisation in measuring or interpreting data. The problem with forgetting about such *biases* is that they may lead to a situation in which a theoretical result is carelessly transferred back into the domain (this is called confirmation bias). Doing so is very tempting as we are seemingly able to say even more about more things.

But the need for learning often arises from the need of *abstraction* and *compression*, which is why a certain *loss* of detail is desirable. We will discover that deliberate sub-optimality is *necessary* in order to guarantee a certain degree of accuracy, too. Therefore, *learning* could be interpreted as *pruning* away as much irrelevant knowledge as possible without losing the ability to discriminate important observations.[3]

[3] In fact, the human being also first needs to learn how to ignore before he can learn more.

3.2 Changing the Representation

It is very important to understand that the world we live in, our domain $\mathfrak{D}$, can be represented in many different ways:

$$\begin{tikzcd}
 & & \mathfrak{D} \arrow[dll, "\rho_1"'] \arrow[dl, "\rho_2"] \arrow[drr, "\rho_n"] & & \\
\mathfrak{U}_1 & \mathfrak{U}_2 & \cdots\cdots & & \mathfrak{U}_n
\end{tikzcd}$$

When working with information systems, the representation space $\mathfrak{U}$ is structured by a set $\mathbf{F}$ of features. Then, ρ maps $\mathfrak{D}$ on a feature space on the base set U:

$$\rho : \mathfrak{D} \to \mathfrak{I} \quad \text{with} \quad \rho(x) = \langle f_0(x), f_1(x), \ldots, f_{n-1}(x) \rangle \tag{3.7}$$

where $\mathfrak{I} = \langle U, \mathbf{F}, V_{\mathbf{F}} \rangle$. This is quite easy—but it is not easy to find a "good" representation. Whether a representation is good or not depends on its expressive power (are we able to formulate sufficiently precise hypotheses at all?), its computational complexity, and, last but not least, on its readability. Due to one or another problem with these requirements it sometimes is necessary to change a representation system into a more suitable one:

Example 3.6 Some things appear to be indistinguishable. For example, sequences of numbers: Are the following two numbers equal?

```
646942834178564068362
175483462854068362649
```

Of course, they are not. But are they equal in a sense that the number of occurrences of all the digits are the same? Without a lot of counting, the question cannot be easily answered. But if we change our point of view, the representation may change into

```
122334444566666788890
122334444556666788890
```

and it is easy to see that in the second sequence one 6 has changed into a 5. ●

Exercise 3.5 (◊) Compare the two representations from the previous example (random digit distribution versus sorted appearance) with respect to the computational complexity of tests for equality!

So what we need to do is to find an *appropriate* representation ρ' or a transform (shift) $\tau : \rho \mapsto \rho'$:[4]

$$\begin{tikzcd}
 & \mathfrak{D} \arrow[dl, "\rho"', ""{name=R}] \arrow[dr, "\rho'", ""'{name=RP}] & \\
\mathfrak{U} & & \mathfrak{U}'
\arrow[from=R, to=RP, bend right, "\tau"']
\end{tikzcd}$$

In Example 3.6, τ simply was the sorting function.

esentation
sform

Definition 3.6 — Representation Transform.
We call a mapping τ (in literature: φ) a *representation* transform (shift, change):

$$\vec{x} = \langle x_0, x_1, \ldots x_{n-1} \rangle \longmapsto \vec{\tau}(\vec{x}) = \langle \tau_0(\vec{x}), \tau_1(\vec{x}), \ldots, \tau_{N-1}(\vec{x}) \rangle \tag{3.8}$$

If $N < n$, τ is an operation that reduces dimensionality. A simple example could be *feature selection*, where $\tau(\mathbf{F}) = F \subset \mathbf{F}$. Sometimes, it is useful to choose $N > n$. ●

Exercise 3.6 (♦) Find example where $n < N$. Find examples where $N > n$ is appropriate! Where do $f_n, f_{n+1}, \ldots, f_{N-1}$ come from?

Example 3.7 ♦ Compare the random digit distribution (ρ) versus sorting and comparison ($\rho \circ \tau$) with respect to the computational complexity of tests for equality in Example 3.6. ●

Representation Change
A change of representation is a mapping from one representation system into another to make the representation or the learning problem less complex.

Example 3.8 Suppose, we want to learn the concept c of multiples of π; i.e. $c = \{n\pi : n \in \mathbb{N}\} \subseteq \mathbb{R}$. It is not easy to describe the concept of π on the ray of real numbers if you try to mark every $n\pi$ without any other tool (such as dividers). But if we transform the representation on the ray $\mathbb{R}$ into $\tau : \mathbb{R} \to (\mathbb{R} \times \mathbb{R})$, things become easier supposed that we find a suitable definition of τ. We choose

$$\tau : x \mapsto \langle x, \sin x \rangle$$

Then, $x \in c \iff \sin x = 0 \iff \tau(\rho(x)) = \langle x, 0 \rangle$, i.e. we found a rather simple characteristic function for the target concept: It is the set of all points where the curve intersects with the x-axis or the set of all tuples whose second argument is 0. ●

Another concrete example is the following description of Newton's law of gravity:

Example 3.9 Given two bodies of mass m_0 and m_1, Newton's law describes the gravitational force $f : \mathbb{R}^3 \to \mathbb{R}$ between them:

$$f(m_0, m_1, r) = c\frac{m_0 m_1}{r^2}$$

where r is the distance between two objects. If we were to represent objects in space we usually do so by defining their x, y, z–coordinates. Together with its mass m, we need eight real numbers to represent two objects:

$$\rho : \mathfrak{D} \to \mathbb{R}^8 \text{ with } \vec{x} = \langle x_0, x_1, y_0, y_1, z_0, z_1, m_0, m_1 \rangle$$

[4] Caution! Each arrow in this diagram may produce information loss.

This is a nice representation, but it does not really help in applying the function f as we need to know the distance r between m_0 and m_1 rather than the objects' positions. Therefore, we change the representation by mapping the position data of m_0 and m_1 onto their Euclidean distance; i.e. $\tau : \mathbb{R}^8 \to \mathbb{R}^3$ with

$$\tau(\vec{x}) = \left\langle \sqrt{\sum_{p \in \{x,y,z\}} (p_0 - p_1)^2}\,, m_0, m_1 \right\rangle \in \mathbb{R}^3$$

If f would be interpreted in the first representation (using the 8-tuples), we would have to evaluate the following expression:

$$f(x_0, x_1, y_0, y_1, z_0, z_1, m_0, m_1) = c \frac{m_0 m_1}{\sum_{p \in \{x,y,z\}} (p_0 - p_1)^2}$$

By using τ (and the inverse) we can now shift the definition of f into the representation or outwards (leaving the work to the one who has to interpret the data).[5] ●

Exercise 3.7 (♦) Give a second transform $\tau' : \tau(\rho) \to \rho''$ where all arithmetic operations we need are $+$ and $-$.

We now go a bit into detail using an example that has become famous for two reasons: It was used as an argument against the omnipotence of perceptrons and introduced the notion of *linear separability* into machine learning.

3.2.1 Linear separability

Consider the following linear equation $f_\wedge : \mathbf{2} \times \mathbf{2} \to \{\mathbf{1}, -\mathbf{1}\}$ which shall describe the logical "and":

$$f_\wedge(x, y) \quad = \quad \begin{cases} \mathbf{1}, & vx + wy + b > 0 \\ -\mathbf{1}, & vx + wy + b < 0. \end{cases} \tag{3.9}$$

Then, we need to find v, w such that $f_\wedge(x, y) = \operatorname{sgn}(vx + wy + b)$ satisfies equation (3.9). The hypothesis space can be illustrated by truth tables:

y			
1	**0**	**1**	
0	**0**	**0**	
$\wedge$	**0**	**1**	x

This problem can be described by three different representations: First, as the problem of defining a separating plane, secondly by defining exclusive conjunction in terms of other operators, and, finally, by a quick look at Church's encoding of Boolean operators.

[5] Another representation of Newton's law is given in [**?**]: "*Yakka foob mog. Grug pubbawup zink wattoom gazork. Chumble S̈puzz.*".

The logic of exclusive disjunction

In order to define $f_\wedge$ we choose as parameters $\langle v_\wedge, w_\wedge\rangle := \left\langle \frac{1}{2}, \frac{1}{2}\right\rangle$ and $b_\wedge = -\frac{3}{4}$ which gives

$$f_\wedge(x,y) \quad = \quad \operatorname{sgn}\left(\frac{1}{2}x + \frac{1}{2}y - \frac{3}{4}\right). \tag{3.10}$$

Using the same method, we define disjunction and Nand:

$$f_\vee \quad : \quad \langle v_\vee, w_\vee\rangle := \left\langle \frac{1}{2}, \frac{1}{2}\right\rangle \text{ and } b_\vee = -\frac{1}{4} \tag{3.11}$$

$$f_{\bar\wedge} \quad : \quad \langle v_{\bar\wedge}, w_{\bar\wedge}\rangle := \left\langle -\frac{1}{2}, -\frac{1}{2}\right\rangle \text{ and } b_{\bar\wedge} = \frac{3}{4}. \tag{3.12}$$

But the exclusive disjunction $f_{\dot\vee}$ cannot be expressed in terms of a linear function, since we need at least two cuts through the plane in order to separate all **0**s from all **1**s:

x_1			
1	**1**	**0**	
0	**0**	**1**	
$\dot\vee$	**0**	**1**	x_0

To solve this problem logically, one transforms $\dot\vee$ as follows:

$$x_0 \dot\vee x_1 \quad \Longleftrightarrow \quad (x_0 \bar\wedge x_1) \wedge (x_0 \vee x_1)$$

Logically, this is a simple trick—but if we understand logical operators as a special kind of functions it becomes clear why $\dot\vee$ poses a tricky problem. Our definition

$$f_{\dot\vee}(x,y) \quad := \quad f_\wedge(f_{\bar\wedge}(x,y), f_\vee(x,y))$$

shows that we run into a problem with operator scopes: we first evaluate the $\bar\wedge$-expression, then the $\vee$-expression and then pass their results to the out-most $\wedge$-expression.

A brief digression to λ-calculus

The Church-encoding of Boolean operators in λ-calculus explains the problem in a beautifully concise way: Lambda calculus requires to reformulate such a definition with "inline" definitions of anonymous unary functions. For example, $f_\wedge(x,y)$ becomes a function that upon receiving x delivers a *new* function with a parameter y. Any variable is a λ-expression and a variable itself evaluates to its own value. We represent a Boolean variable by a pair $x = (t, f)$ where the first argument represents **1** and the second one **0**. The function $\mathbf{1} :\to \{t, f\}$ then can be defined as

$$\mathbf{1} \quad := \quad \lambda t.\, \lambda f.\, t = t \tag{3.13}$$

i.e. **1** is a function that always returns t (the first argument of the Boolean value tuple). Similarly, $\mathbf{0} := \lambda t.\, \lambda f.\, f = f$. We now define $\wedge$ and $\vee$ for two Boolean variables x and y by

$$\wedge \quad := \quad \lambda x.\, \lambda y.\, xyx \tag{3.14}$$

$$\vee \quad := \quad \lambda x.\, \lambda y.\, xxy \tag{3.15}$$

Example 3.10 Let $x = \mathbf{1}$ and $y = \mathbf{0}$. Then

$$\begin{aligned}
x \wedge y \quad &= \quad \lambda t.\, \lambda f.\, tft = \lambda t.\, \lambda f.\, (\lambda t.\, \lambda f.\, t)(\lambda t.\, \lambda f.\, f)(\lambda t.\, \lambda f.\, t) \\
&\qquad \{\!|\text{ Evaluate first } (\lambda t.\, \lambda f.\, t) \text{ with two arguments and resolve by Def. of } \mathbf{0} \,|\!\} \\
&= \quad \lambda t.\, \lambda f.\, (\lambda t.\, \lambda f.\, f) = \lambda t.\, \lambda f.\, f \\
&\qquad \{\!|\text{ Again, by Def. of } \mathbf{0} \,|\!\} \\
&= \quad f = \mathbf{0}
\end{aligned}$$

●

Exercise 3.8 ($\blacklozenge$) For $x = \mathbf{1}$ and $y = \mathbf{0}$, compute $x \vee y$.

Now begins the fun: Let us consider negation. Negation means that if x, $\neg x$ is false, but if x is false, $\neg x$ is the case. The simplest method to define negation is

$$\neg \quad := \quad \lambda x.\, \lambda t.\, \lambda f.\, xft \tag{3.16}$$

Example 3.11 Let $x = \mathbf{0}$. Then

$$\begin{aligned}
\neg \quad &= \quad \lambda x.\, \lambda t.\, \lambda f.\, xft = \lambda t.\, \lambda f.\, \mathbf{0}ft \\
&\qquad \{\!|\text{ Defn. } \mathbf{0} \text{ and Evaluation } |\!\} \\
&= \quad \lambda t.\, \lambda f.\, (\lambda t.\, \lambda f.\, f)ft = \lambda t.\, \lambda f.\, t \\
&\qquad \{\!|\text{ Defn. } \mathbf{1} \,|\!\} \\
&= \quad t = \mathbf{1}
\end{aligned}$$

●

Exercise 3.9 ($\lozenge$) Compute $\neg\mathbf{1}$!

Exercise 3.10 ($\blacklozenge$) Define $\bar{\wedge}$!

Now, the trouble begins: The attentive reader will have noticed that we needed *three* variables to model the unary negation operator, while the binary operators

$\wedge$ and $\vee$ needed only two. So what does it look like with $\dot\vee$?

$$\begin{aligned}
x\dot\vee y &\iff ((\neg(x\wedge y))\wedge(x\vee y)),\ \text{hence:}\\
\dot\vee &:= \wedge((\neg(\wedge(xy)))(\vee(xy)))\\
&= \wedge((\neg\lambda x.\lambda y.xyx)(\lambda x.\lambda y.xxy))\\
&= \wedge((\lambda n.\lambda x.\lambda y.(xyx)ft)(\lambda x.\lambda y.xxy))\\
&= \lambda e.(\lambda n.\lambda x.\lambda y.(xyx)ft)(xxy)(\lambda n.\lambda x.\lambda y.(xyx)ft)\\
&= \lambda e.\lambda n.\lambda x.\lambda y.((xyx)ft)(xxy)((xyx)ft)
\end{aligned}$$

Obviously, $\dot\vee$ requires *more* variables to be evaluated than $\wedge$ or $\vee$.

Exercise 3.11 (♦♦) Compute the truth table for $\dot\vee$ using the definition above.

Computing a separating hyperplane

With this result in mind, we return to our functional view in representation space $\mathbf{2}^2$. Using the definitions from equations (3.10) and (3.11) we derive $f_{\dot\vee}(x,y) :=$

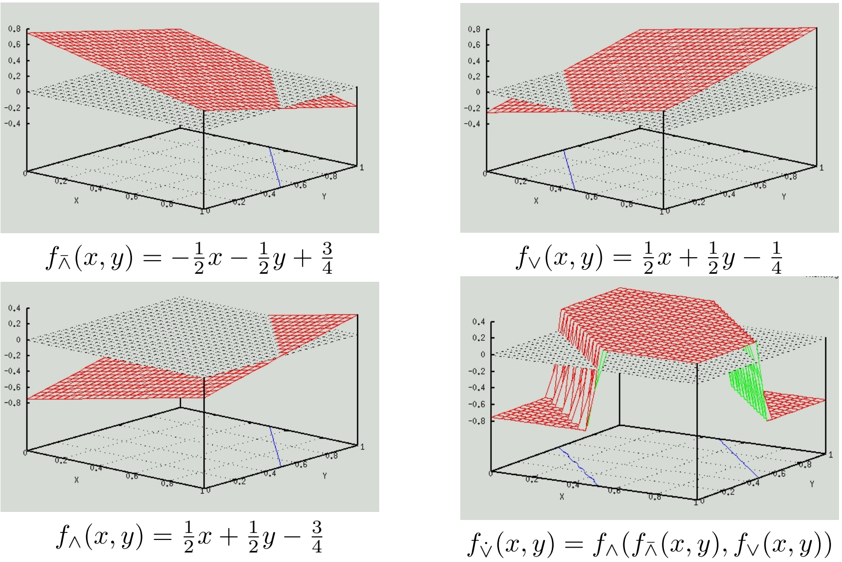
$$\operatorname{sgn}\left(\frac{1}{2}\operatorname{sgn}\left(-\frac{1}{2}x-\frac{1}{2}y+\frac{3}{4}\right)+\frac{1}{2}\operatorname{sgn}\left(\frac{1}{2}x+\frac{1}{2}y-\frac{1}{4}\right)-\frac{3}{4}\right) \qquad (3.17)$$

We now shift our representation space $\mathfrak{U}$ by a transform $\tau : \mathbf{2}^2 \to \mathbf{2}^3$ with $\tau(f(x,y)) = f'(x,y,x\bar\wedge y))$ for all binary logical operators f. Then, we can define:

$$\begin{aligned}
f'_\wedge(x,y,z) &:= (x\bar\wedge y)\bar\wedge(x\bar\wedge y)\\
f'_\vee(x,y,z) &:= (x\bar\wedge x)\bar\wedge(y\bar\wedge y)\\
f'_{\dot\vee}(x,y,z) &:= (x\bar\wedge z)\bar\wedge(y\bar\wedge z).
\end{aligned}$$

In order to to define conjunction and disjunction, we need two (x and y) variables, but for exclusive disjunction we need all three variables — i.e. once we represent our logic operators in $\mathbf{2}^3$, we do not need any additional variables to compute the result of $\dot\vee$, but we could not do it with only two variables in $\mathbf{2}^2$. Thus, we found a simple solution by shifting the problem into a more complex space. The idea behind a transform into higher dimensional spaces is nearly trivial: Figure 3.2 shows that the XOR-problem cannot by solved with a single cut through a plane. If you want to realise two cuts by one slice, then you have to fold the paper sheet. But folding requires a third dimension.

Example 3.13 Another representation change is that of rule extraction from decision trees which we will discuss in section 5.5 in detail. Imagine that a tree represents a conjunction of literals along each path leading to a conclusion in the leaf. Representing this tree as a set of implications allows us to weaken each single rule by dropping individual literals. The same operation on trees could result in syntactically incorrect hypotheses (i.e. trees with missing edges). ●



$f_{\bar{\wedge}}(x,y) = -\frac{1}{2}x - \frac{1}{2}y + \frac{3}{4}$

$f_{\vee}(x,y) = \frac{1}{2}x + \frac{1}{2}y - \frac{1}{4}$

$f_{\wedge}(x,y) = \frac{1}{2}x + \frac{1}{2}y - \frac{3}{4}$

$f_{\dot{\vee}}(x,y) = f_{\wedge}(f_{\bar{\wedge}}(x,y), f_{\vee}(x,y))$

Figure 3.2: Linear Separability

Simplifying the Framework (Figure 3.1)
In knowledge discovery we cannot try derive new knowledge about objects from observations in the real world. All we can do is to ground our enterprise on *representations of objects*. Since this is the case in *every* setting, we simply *identify* objects by their representations:

$$\langle \chi(c_0)(x), \chi(c_1)(x), \ldots \rangle \equiv_\rho x$$

The set of objects (or rather their representation) we work on is a subset s of all the objects U we can talk about; that is, U is the base set of our representation space $\mathfrak{U}$. The examples we have are representations of observations—and the information whether an object x belongs to an unknown concept is provided by a teacher signal

$$\chi(c_t) =_\rho t$$

The hypothesis h is a function extrapolating t on U.

From now on, we will speak of objects $x \in s$, target functions $t : s \to \mathbf{2}$, and hypotheses $h : U \to \mathbf{2}$. Nevertheless, the reader shall always keep in mind, that one of our fundamental assumptions is that ρ is *appropriate.*

3.3 Samples

In general, we learn from a *sequence of observations of examples.*

Samples
A sample $\mathbf{s}$ is a set or sequence of observations or instances from which we shall learn. Usually, each example in a sample is *labelled.* Understanding this label as a target function t's value, a sample is a set of support points. Machine learning means to approximate t through these points.

A sequence can be transformed into a simple set when we assume an implicit assignment of the position of an object to the object itself (i.e. $[a, a] = \{\langle a, 1\rangle, \langle a, 2\rangle\}$). Additionally, all objects in $\mathbf{s}$ are examples *for* something. This means that every example is labelled with a so-called *teacher signal* that explains what this object is a proper example for:

(Labelled) sample, $\mathbf{s}$, t

Definition 3.7 — (Labelled) sample, s, t.
A *sample* $\mathbf{s}$ is a set of m objects of the universe together with one out of k different *labels* defined by a *teacher* t:

$$\begin{aligned} \mathbf{s} &\subseteq U \times \mathbf{k} && (3.18)\\ \mathbf{s} &:= \{\langle x_0, t(x_0)\rangle, \langle x_1, t(x_1)\rangle, \ldots, \langle x_{m-1}, t(x_{m-1})\rangle\} && (3.19)\end{aligned}$$

with $x_i \in U$, $i \in \mathbf{m}$ and $t : U \to \mathbf{k}$. t is also referred to as *target function.* ●

If $\operatorname{cod}(t) = \mathbf{2}$, we sometimes speak of the *positive* and *negative* sample, written $\mathbf{s}^y := \{\langle x, y\rangle : \langle x, y\rangle \in \mathbf{s}\}$ for $y \in \mathbf{2}$.
But where do $\mathbf{s}$ and t come from? We consider t first. The aim in machine learning is to induce new concepts c. Therefore, we need a set of examples that belong to this yet unknown example. The target function t can be seen as the characteristic function of the unknown concept—but since it is unknown, we cannot define it as such. Actually, t is as unknown as c but we will try to learn a hypothesis h that approximates t.
The sample itself is, as suggested by the name, *sampled* from the universe. It contains a subset of all objects and the choice is subject to a sampling function. But there is even more to it: It seems reasonable to assume an (unknown) distribution μ over $\mathfrak{D}$ which is preserved by ρ:[6]

Example 3.14 It is much more likely to observe birds that can fly than birds that cannot; especially if the sample is drawn from the rain forest. In Antarctica, things are quite different. ●

Therefore, we define a sample to be the result of a sampling function that picks elements from our domain with respect to their "probabilities".

$S_\mu(m, t)$, sampling function, $\mathbf{s}$, sample

Definition 3.8 — $S_\mu(m, t)$, sampling function, s, sample.
A *sampling function* S generates a sample as a set $\mathbf{s}$ of m elements $x \in U$ with a target label $t(x)$:

$$\mathbf{s} = S_\mu(m, t) = \{\langle x_0, t(x_1)\rangle, \langle x_1, t(x_1)\rangle, \ldots, \langle x_{m-1}, t(x_{m-1})\rangle\} \qquad (3.20)$$

Due to the nature of μ, $S_\mu(m, t)$ is not a function but rather a *procedure*: It draws m times an x from U w.r.t.[7] μ with replacement. ●

[6] Note that "which is preserved by ρ" is a fundamental assumption.
[7] with respect to

Accordingly, it holds for two samples delivered by the same sampling method that

$$\mathbf{s}_1 \neq \mathbf{s}_2$$

Now we can reformulate the assumption that we only mentioned in footnote 6 until now:

Theorem 3.1 (Sampling Assumption) Let there be n samples $\mathbf{s}_0, \ldots, \mathbf{s}_{n-1}$ generated by repeated and independent execution of $S_\mu(m, t)$. Then for $i \in \mathbf{n}$,

$$\lim_{n \to \infty} \frac{|\{\mathbf{s}_i : \langle x, t(x) \rangle \in \mathbf{s}_i\}|}{n} = \phi(x).$$

where $\phi(x)$ denotes a normalised version of $\mu(\{x\})$ such that $\sum_{x \in U} \phi(x) = 1$.
The sampling assumption can be verbalised as follows:

> **Sampling Assumption**
> When drawing samples of objects from a base set with respect to an underlying distribution, then the probability that an object occurs in a sample is proportional to the "mass" of the object.

Accordingly, if we have enough (say, at least m) observations, then the probability of making a certain observation about x corresponds to the frequency of occurrences of x in the samples.[8]
But if we don't have a teacher around who could give us a teaching signal t we need to learn all by ourselves.

(Un-)supervised Learning

Definition 3.9 — (Un-)supervised Learning.
(Un-)supervised Learning means to learn from a (un-)labelled sample. An unlabelled sample is a sample where $t(x) = ?$ (which is a constant symbol denoting that t is not defined for x). ●

Supervised learning means that we have some evidence for and against an unknown concept c in $\mathfrak{U}$ given by $t : \mathfrak{U} \to \mathbf{2}$ but we have no *intensional* description of t. Unsupervised learning means that we even have no information about possible targets. Since there is no knowledge about t, the target function remains undefined. It is even unclear how to choose a proper codomain for t since we do not have any information about the target concept. In such a case we need to learn how to group objects together such that they form "meaningful" groups which might correspond to some concepts.
But even if we *have* a teacher, the teacher might give us incorrect information.

Example 3.15 Imagine a binary classification problem on $\mathfrak{D} = \{a, e, b, c\}$ and $\mathfrak{U} = \mathbf{2}$. Let the target concept be *vowel* $= \{a, e\}$. Now we apply ρ with $\rho(a) = \rho(b) = \mathbf{0}$ and $\rho(e) = \rho(c) = \mathbf{1}$. Let $t = id_{\mathbf{2}}$. Then we have the representation of e and c labelled as vowels and the representation of a and b as consonants. ●

[8] So if m is big enough, then the approximate value for the probabilities does not change significantly for increasing k.

The problem behind (lossy) representation functions is that they generate two kinds of *noise* on our example sets: 5

Noisy sample

Definition 3.10 — Noisy sample.
A sample is called *noisy* if the teacher's label does not agree with the actual characteristic function of the sought for concept.

- Let $x, y \in \mathfrak{D}$ and $x \neq y$. Let ρ be lossy such that $\rho(x) = \rho(y)$. Then, because t is function, it holds that $t(\rho(x)) = t(\rho(y))$. But if $\chi(c)(\rho(x)) \neq \chi(c)(\rho(y))$, t disagrees with the meaning of c.

- Let there be two samples $\mathbf{s}$ and $\mathbf{s}'$. If $\langle x, y \rangle \in \mathbf{s}$ and $\langle x, z \rangle \in \mathbf{s}'$ and $y \neq z$, then t is obviously inconsistent.[9]

●

Now that we have seen there is a huge difference between the domain and our representation, we need to make a large simplification: From now on, we forget about the domain and target concepts—*all we have is the data* we are given. The data we have is just a *representation* of the domain and so there might be a lot of information loss involved. We need to rely on the fact that $\mathfrak{U}$ is a sufficiently precise representation of $\mathfrak{D}$ and that $t : \mathfrak{U} \to \mathbf{2}$ is a sufficiently precise representation of t. Any results we will achieve, are results modulo the underlying ρ. And any such result may be pretty "accurate" on $\mathfrak{U}$, but whether it is a reasonable hypothesis only (and entirely) depends on our interpretation of the result back into $\mathfrak{D}$.

Essential Stipulations for Learning

1. The concept we want to learn is representable.
2. The data we have appropriately represents the domain.
3. If a concept representation is meaningful on a sufficiently large subset of data, it is true on all objects.

The last stipulation is commonly known as the *inductive assumption.* Finally, we assume that we can evaluate the quality of a hypothesis within $\mathfrak{U}$.

So far, we have derived the notion of t from the characteristic function of a set of objects. Therefore, t always was a binary function $t : \mathfrak{U} \to \mathbf{2}$. There are many more paradigms of describing concepts; there are, for example, fuzzy sets or complex concepts with more than just two possible values. In such cases, the codomain of t is a more complex set or even structure. For binary learning tasks, $\mathrm{dom}(t) = \mathbf{2}$; if the target concept shall discriminate k different values then $\mathrm{dom}(t) = \mathbf{k}$. A target function with fuzzy descriptions over $\mathbf{k}$ linguistic variables requires $\mathbb{R}^{\mathbf{k}}$ as codomain. Since most learning problems are binary or

[9] The reasons for this are manifold. One possible source of noise is that t is not a function as assumed but rather a relation. An example are classifications whose classes are not pairwise disjoint.

can be reduced to such problems we usually assume $\mathrm{dom}(t) = \mathbf{2}$ unless noted otherwise.

Now that we have defined what we are talking about and what the examples we are given look like, we can again reformulate our working definition of machine learning into a first description that we can safely call a "definition":

Machine Learning Algorithm, `Alg`

Definition 3.11 — Machine Learning Algorithm, `Alg`.
A machine learning algorithm `Alg` receives a *sample* $\mathbf{s}$ of representations of objects of our domain and computes a *hypothesis* h. The goal is to approximate the unknown target function $t : U \rightarrow$ by $h : U \rightarrow \mathrm{cod}(t)$ using the teaching signal $\chi(c) : s \rightarrow \mathrm{cod}(t)$:

$$\texttt{Alg}(\mathbf{s}) = h \approx t \tag{3.21}$$

where $\mathbf{s} = S_\mu(m, t)$ and c is the unknown concept we want to learn. Note that $\chi(c)$ is a partial, extensionally defined function whereas h is a total function that approximates t on the entire base set of the representation space. ●

Machine learning is a search procedure for an approximation h of the representation of a target concept in a (possibly very large) hypothesis space. This space cannot be searched exhaustively which is why the search requires heuristic guidance. Hence the solutions of the search process can be more or less accurate w.r.t. our representation of the target concept.
With `Alg` generating some h on $\mathfrak{U}$, we have a gained a hypothesis on our representation space. What needs to be done is to *interpret* h on $\mathfrak{D}$. Since ρ is not necessarily injective, there might not be a clear definition of an interpretation function $\rho^{\smile}$, and so it is not always clear how to interpret a hypothesis in the real world $\mathfrak{D}$.

Evaluation of h

Definition 3.12 — Evaluation of h.
Usually, the interpretation step $\rho^{\smile}$ is omitted. The hypothesis h is learned from data from U and it is evaluated within $\mathfrak{U}$. ●

This is a quite strong assumption, because it presupposes that our representation space suffices to describe everything we want to be able to describe. We shall come back to this when discussing *biases* in section 3.6.

3.4 Evaluation of Hypotheses

Whatever we learn may be "correct" or "wrong". Similarly, a hypothesis $h = \texttt{Alg}(\mathbf{s})$ may or may not agree with t on the sample or any other subset of U. However, we can define some kind of relative correctness of h as follows.

Correctness of h

Definition 3.13 — Correctness of h.
We call h $\langle s, f\rangle$-*correct*, if h agrees with f on s:

$$\forall x \in s : h(x) = f(x) \tag{3.22}$$

where s is an arbitrary subset of U. ●

So if `Alg` has learned h from $\mathbf{s}$, h is $\langle \mathbf{s}, t\rangle$–correct if and only if h agrees with t on the sample. Being able to discriminate agreement from disagreement, we can determine *sets* of instances on which h agrees with t or not. The set of examples on which we find an agreement is the set of objects that are learned correctly, the set of objects on which $\mathbf{t}$ and h disagree is the *error set* of h.

> **Evaluation of hypotheses (I)**
> The *error set* of a hypothesis h is the subset of instances on which a test of h disagrees with t. The relative size of such an error set (together with a probability distribution) can be used to describe the magnitude of the error of h on a set.
> Since we usually do not know the exact probability distribution and since we have only a subset of the entire domain for testing, we approximate the error by *precision* and *recall* and their respective generalisations *accuracy* and *coverage*.

3.4.1 Error Sets and Error Measures

Error set of h

Definition 3.14 — Error set of h.
The *error set* of h on s is the set of all objects in s for which h delivers a wrong prediction:

$$\mathrm{errset}_s(h,t) \quad := \quad \{x \in s : h(x) \neq t(x)\} \tag{3.23}$$

●

The size of the error set in relation to the size of the set is a numerical measure of the error of h on s:

Error Rate $\mathrm{err}(h,s)$ of h

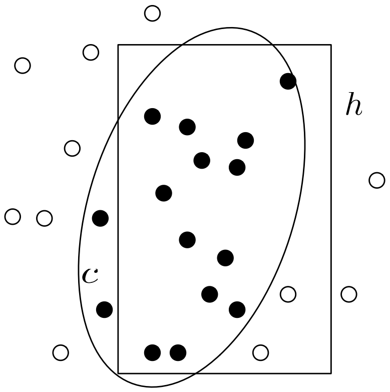
Definition 3.15 — Error Rate $\mathrm{err}(h,s)$ **of** h.
The *error rate* of h on s is defined as

$$\mathrm{err}_s(h,t) \quad := \quad \frac{|\mathrm{errset}(h,s)|}{|s|} \tag{3.24}$$

●

So far we have considered "binary" decisions only: an object x can be classified correctly or incorrectly. But for more complex domains, misclassification may have a quantitative measure, too: For example, ■ is not so different from ■ than it is from ∘. Orange is not as much different from red than blue. In order to define quantitative measures of errors, we require a distance measure (which in the example of color classification could be satisfied by different spatial representations of colours like the RGB or YUV models). A very simple idea is to use the numerical difference between possible values of t as such an error:

Example 3.16 Learning intervals on rays: Imagine $t : \mathbb{R} \to \mathbf{3}$, where the target classes represent the intervals $[\ldots, -5), [-5, 5]$ and $(5, \ldots]$. It is clear that the observation 6 is a positive example for class $\mathbf{2}$; i.e. $t(6) = \mathbf{2}$. We compare two hypotheses: $h(6) = \mathbf{1}$ and $h'(6) = \mathbf{0}$. Since $t(6) = \mathbf{2}$ is closer to class $\mathbf{1}$, the error of h should not be as big as the error of h'. ●



The ellipsoid area is the target concept c for which $\chi(c)x = \mathbf{1}$; the rectangle represents the hypothesis h.

The set of positive examples is the set of all black dots; the white dots denote an example in $\mathbf{s}$ with $t(x) = \mathbf{0}$.

Figure 3.3: A Hypothesis

More formally, we define:

(dist)-Error, $\text{error}_s(h,t)$ w.r.t. dist

Definition 3.16 — **(**dist**)-Error,** $\text{error}_s(h,t)$ **w.r.t.** dist.
Let $s \subseteq U$ and $\text{dist}(x,y)$ be a distance function on $\text{cod}(t)$. Then, the dist-*error* of h on s with respect to t is

$$\text{error}_s(h,t) = \sum_{x \in s} \text{dist}(h(x), t(x)) \tag{3.25}$$

The definition of the metric dist depends on the structure of $\text{cod}(t)$. Usually, one defines

- for continuous, numeric $\text{cod}(t)$:

$$\text{dist}(h(x), t(x)) := (h(x) - t(x))^2 \tag{3.26}$$

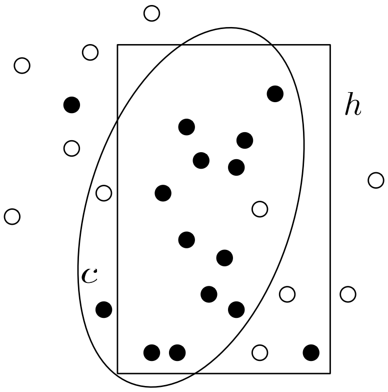
- for $\text{cod}(t) = \mathbf{2}$,

$$\text{dist}(h(x), t(x)) := \begin{cases} \mathbf{0}, & h(x) = t(x) \\ \mathbf{1}, & \text{otherwise} \end{cases} \tag{3.27}$$

- for nominal $\text{cod}(t) = \mathbf{k}$ one usually also chooses a simple binary distance measure.

$\text{error}_s(h,t)$ can be normalised by a factor $\frac{1}{|s|}$ assuming an independent identical distribution. ●

Exercise 3.12 (◊◊) Determine $\text{errset}_s(h, \chi(c))$, $\text{err}_s(h, \chi(c))$, and $\text{error}_t(h, \chi(c))$ for the example in figure 3.3. — Determine $\text{errset}_s(h,t)$, $\text{err}_s(h,t)$, and $\text{error}_s(h,t)$ for the example in figure 3.3. What is the difference?



The ellipsoid area is the target concept c for which $\chi(c)x = \mathbf{1}$; the rectangle represents the hypothesis h.

The set of positive examples is the set of all black dots; the white dots denote an example in $\mathbf{s}$ with $t(x) = \mathbf{0}$.

Figure 3.4: A hypothesis for a noisy sample

Exercise 3.13 (◊) Take a look at figure at figure 3.4 and compare it to figure 3.3. Explain errors and noise!

Now, the error measure has been parametrized by some kind of the magnitude of the error—some misclassifications are worse than others. But there is a second issue in error measures: A single "large" error that happens only once in a million times is not as disturbing as a nearly constant small misclassification error. Accordingly, one should take into account the probability of the event that causes an error: Since the occurrence for $x \in s$ it usually holds that $\phi(x) \neq \frac{1}{|s|}$, we can define the *true error* as the dist-error with respect to μ:

True error, $\mathrm{error}_s^{\mu}(h,t)$

Definition 3.17 — True error, $\mathrm{error}_s^{\mu}(h,t)$.
The *true error* of h on s w.r.t. t is

$$\begin{aligned}\mathrm{error}_s^{\mu}(h,t) &:= E_\mu\left(\mathrm{error}_s(h,t)\right) && (3.28)\\ &= \sum_{x\in s}\left(\mu(\{x\})\cdot \mathrm{dist}(h(x),t(x))\right) && (3.29)\end{aligned}$$

where E_μ denotes the expected value. With our inductive assumption we can approximate μ by ϕ. This results in

$$\mathrm{error}_s^{\mu}(h,t) \quad :\approx \quad \sum_{x\in s}\left(\phi(x)\cdot \mathrm{dist}(h(x),t(x))\right) \qquad (3.30)$$

Since μ is unknown by definition, we usually use the error measure as defined in equation (3.30). ●

Let us examine the difference between all those error measures by example: Consider the set $s = \{0, 1, \ldots, 10\}$ and functions denoting prime numbers, even and odd numbers, multiples of 3 and 4 and the Fibonacci-numbers:

$x \in s$	1	2	3	4	5	6	7	8	9	10
$prm(x)$	0	1	1	0	1	0	1	0	0	0
$evn(x)$	0	1	0	1	0	1	0	1	0	1
$odd(x)$	1	0	1	0	1	0	1	0	1	0
$f_3(x)$	0	0	1	0	0	1	0	0	1	0
$f_4(x)$	0	0	0	1	0	0	0	1	0	0
$fib(x)$	1	1	1	0	1	0	0	1	0	0
$g(x)$	3	2	1	2	2	1	2	2	1	2
$\mu_1(\{x\})$	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
$\mu_2(\{x\})$	0.1	0.2	0.25	0.1	0.01	0.1	0.05	0.08	0.1	0.01
$\mu_3(\{x\})$	0.015	0.05	0.1	0.07	0.12	0.028	0.08	10^{-9}	0.031	0.01

Then, for example,

$$\begin{aligned}
\mathrm{errset}_s(odd, prm) &= \{1, 2, 9\} \\
\mathrm{errset}_s(odd, fib) &= \{2, 7, 8, 9\} \\
\mathrm{errset}_s(prm, g) &= \{1, 2, 4, 5, 6, 7, 8, 9, 10\}
\end{aligned}$$

The according error rates are:

$$\mathrm{err}_s(odd, prm) = 0.3, \qquad \mathrm{err}_s(odd, fib) = 0.4, \qquad \mathrm{err}_s(prm, g) = 0.9$$

which is quite easy to see because $m = 10$. The value of $\mathrm{error}_s(h, t)$ depends on δ, of course. We define δ_1 as in (3.27) and δ_2 as in (3.26). Then, the corresponding results are:

δ_1			δ_2		
$\mathrm{error}_s(odd, prm))$	$=$	0.3	$\mathrm{error}_s(odd, prm))$	$=$	3.0
$\mathrm{error}_s(odd, fib))$	$=$	0.4	$\mathrm{error}_s(odd, fib))$	$=$	4.0
$\mathrm{error}_s(prm, g))$	$=$	0.9	$\mathrm{error}_s(prm, g))$	$=$	26

It is clear, that with δ_1, the dist-error equals the error rate. It is also clear that using dist_2 we obtain proportional values to δ_1 on binary learning problems: the term $(h(x) - t(x))^2$ delivers 0 if and only if $h(x) = t(x)$ and 1 if and only if $h(x) \neq t(x)$. Therefore $\mathrm{error}_s(h, t)$ with δ_1 simply delivers $|\mathrm{errset}_s(h, t)|$. But what if we take into account different distributions on U? Then,

			μ_2	μ_3
dist	h	t	$\mathrm{error}_s^\mu(h, t)$	$\mathrm{error}_s^\mu(h, t)$
1	odd	prm	0.40	0.096
2	odd	prm	0.40	0.096
1	odd	fib	0.43	0.161
2	odd	fib	0.43	0.161
1	prm	g	0.75	0.404
2	prm	g	2.12	0.764

Exercise 3.14 (◊) When computing $\mathrm{err}_{\mu_1,\delta_2}(prm, s)$ with $t = g$, the result is 2.6, but all other values remain unchanged. Explain!

Exercise 3.15 (◊♦♦) Write a program that computes all the error measures with arbitrary definitions of δ and μ and run it on the example data above.

> **Error Measures**
> Given $\mathbf{s} = S_\mu(m, t)$ and `Alg`, the true error of $h =$ `Alg`$(\mathbf{s})$ can only be estimated. The reason for this is that $s \subseteq U$, that μ is unknown, that t may be noisy, that dist can be chosen arbitrarily and that ϕ is just an approximation of μ.

In knowledge discovery, *representation* and error the definition of *error measures* are no less important than the induction process itself.

3.4.2 Precision, Accuracy, and Others

Since μ is unknown we need to *estimate* the error that h will produce on the data we shall fed into the system later on. But once we derived a hypothesis the following questions can help to draw a clearer picture of its error-behaviour:

1. How many $x \in s$ that h predicts to be y actually are y according to t? Simply speaking, it is the same as the number of predictions that are correct.
2. And how many $x \in s$ that are y according to t are predicted to be y by h? This question addresses the coverage of our hypothesis, i.e. it describes the number of objects that the hypothesis is able to classify.

These values are usually defined in terms of so–called *confusion matrices*. If the target function is a binary classifier, the according confusion matrix is a 2×2–matrix:

	$t(x)$	
$h(x)$	0	1
0	h and t agree that $x \in c_0$	h and t disagree
1	h and t disagree	h and t agree that $x \in c_1$

If $\mathrm{cod}(t) = k > 2$ we need to count the cases for each pair of predicted and actual target values. Then, we obtain a $k \times k$–matrix:

Definition 3.18 — Confusion Matrix.

Confusion Matrix

A *confusion matrix* for a learning problem with k classes is defined as follows:

	$t(x) =$				
$h(x) =$	0	1	$\cdots$	$k-1$	$\sum$
0	v_{00}	v_{10}	$\cdots$	$v_{(k-1)0}$	s_1
1	v_{01}	v_{11}	$\cdots$	$v_{(k-1)1}$	s_2
$\vdots$			$\ddots$		$\vdots$
$k-1$	$v_{0(k-1)}$	$v_{1(k-1)}$	$\cdots$	$v_{(k-1)(k-1)}$	s_{k-1}
$\sum$	t_0	t_1	$\cdots$	t_{k-1}	m

where:

1. v_{ij} is the number of objects for which t predicts class i and h predicts class j:
   $v_{ij} = |\{x \in \mathbf{s} : t(x) = c_i \wedge h(x) = c_j\}|$
2. s_j is the number of objects that h classifies as c_j:
   $s_j = |\{x \in \mathbf{s} : h(x) = c_j\}| = \sum_{l \in \mathbf{k}} v_{lj}$
3. t_i is the number of objects that t classifies as c_i:
   $t_i = |\{x \in \mathbf{s} : t(x) = c_i\}| = \sum_{l \in \mathbf{k}} v_{il}$

We use the index i to denote the columns of the matrix (fixed target) and j to denote rows (fixed hypotheses). ●

Exercise 3.16 (◊♦) Write down the confusion matrices for the graphical example in figure 3.3.

We introduced confusion matrices as a means to count the cases in which our hypothesis agrees or disagrees with the teacher signal. The first measure one usually is interested in, is "preciseness". Preciseness means not to make an wrong prediction in the first place:[10]

- How many $x \in s$ that h predicts to be y actually are y according to t ?

This leads us to the definition of *precision* and *accuracy*:

Precision of h

Definition 3.19 — Precision of h.
The *precision* of h is the fraction of the correct y-predictions in relation to all y-predictions on s. It is a "local" measure as it is defined in terms of a feature's value $j \in \text{cod}(f)$ and not for the entire feature distribution:

$$\begin{aligned} \text{prc}_s(h = i) &:= \frac{|\{x \in s : h(x) = t(x) = j\}|}{|\ \{x \in s : h(x) = j\}\ |} \\ &= \frac{v_{jj}}{\sum_{i \in \mathbf{k}} v_{ij}} = \frac{v_{jj}}{s_j} \end{aligned} \tag{3.31}$$

●

The first definition is quite lengthy and argues with error sets, whereas the last one uses the entries from a confusion matrix. The definition of precision can be visualised by confusion matrices: It is, for a given row j, the number in the j-th column (i.e. the entry on the diagonal) divided by the sum of all entries in the j-th row.

Accuracy of h

Definition 3.20 — Accuracy of h.
The *accuracy* of h is a generalisation of the precision measure. There exist

[10] Being overly precise often results in making too few predictions so as not to risk a wrong prediction.

several slightly different versions. However, they all share the property that *accuracy* shall describe the whole feature's precision. We define:

$$\text{accuracy}_s(h,t) \quad := \quad \frac{|\{x \in s : h(x) = t(x)\}|}{|s|} \frac{\sum_{i \in \mathbf{k}} v_{ii}}{m} \tag{3.32}$$

This value is simply the sum of the diagonal entries in the confusion matrix divided by the sum af all entries, m.

●

Since most classifier learning problems are binary (or are reduced to binary problems), we give a final example with much less notational effort: Any 2×2–confusion matrix of a binary classification problem ($\mathfrak{c} = \{c, -c\}$) has the form

	$t(x)$	
$h(x)$	c	$-c$
c	A	B
$-c$	C	D

Then,

1. $\text{prc}(h(x) = c) = \frac{A}{A+B}$ and $\text{prc}(h(x) = -c) = \frac{D}{C+D}$
2. $\text{accuracy}_s(h,t) = \text{prc}_s(h(x) = -c) = \text{acc}_s(h) = \frac{A+D}{m}$

Example 3.17 Imagine the following confusion matrix:

	$t(x)$		
$h(x)$	c_0	c_1	c_2
c_0	5	1	2
c_1	2	4	2
c_2	0	1	7

The accuracy of h is $\text{accuracy}_s(h,t) = (5 + 4 + 7)/24 = \frac{2}{3}$. h delivers the most precise predictions on c_2, while c_1 is worst ($\frac{2}{3}$ precision). ●

Our second question was:

2. How many $x \in s$ that are y according to t are predicted to be y by h?

The answer is given by measures called *recall* and *coverage*:

Recall of h

Definition 3.21 — Recall of h.
The *recall* of h is the fraction of the correct y-predictions in relation to all desired y-predictions on s:

$$\begin{aligned} & \text{rcl}_s(h = j) \\ := \quad & \frac{|\{x \in s : h(x) = t(x) = j\}|}{|\{x \in s : t(x) = j\}|} && (3.33) \\ = \quad & \frac{v_{jj}}{\sum_{j \in \mathbf{k}} v_{ji}} = \frac{v_{jj}}{t_j} && (3.34) \end{aligned}$$

●

It is the dual concept to precision. Therefore, it is simply the number in the j-th row of the j-th column divided by the sum of entries in the $\jmath$-th column. One might say that coverage is to recall what accuracy is to precision. But since accuracy is the "mass" of the diagonal line to the entire matrix weight, there is no rotation or translation that is sensible and/or delivers a meaningful result. Therefore, we define *coverage* as the average recall.

Coverage of h

Definition 3.22 — Coverage of h.
The *coverage* of h is defined as the average recall:

$$\begin{aligned}
\text{coverage}_s(h,t) &:= \frac{1}{k}\sum_{j\in\mathbf{k}} \text{rcl}_s(h=j) && (3.35)\\
&= \frac{1}{k}\sum_{j\in\mathbf{k}} \frac{v_{jj}}{t_j} && (3.36)\\
&= \frac{1}{k}\sum_{j\in\mathbf{k}} \frac{|\,\{x\in s : t(x)=h(x)=j\}\,|}{|\,\{x\in s : t(x)=j\}\,|} && (3.37)
\end{aligned}$$

Exercise 3.17 (◊) Why don't we define

$$\text{cov}_s(h) = \frac{1}{m}\sum_{i\in\mathbf{k}} t_i \cdot \text{rcl}_s(h(x)=c_i)\ ?$$

Again, we consider a 2 × 2–confusion matrices for an illustration with simplified notation:

	$t(x)$	
$h(x)$	c_0	c_1
c_0	A	B
c_1	C	D

where $\text{rcl}_s(h=c_0) = \frac{A}{A+C}$ and $\text{rcl}_s(h=c_1) = \frac{D}{B+D}$. $\text{cov}_s(h)$ is $\frac{1}{2}(\frac{A}{A+C} + \frac{B}{B+D})$.

Example 3.18 Imagine the following confusion matrix:

	$t(x)$		
$h(x)$	c_0	c_1	c_2
c_0	5	1	2
c_1	2	4	2
c_2	0	1	7

The recall values are $\frac{5}{7} \approx 0.71$, $\frac{4}{6} \approx 0.67$ and $\frac{7}{11} \approx 0.64$. Recall is best for class c_0 and worst for c_2. Therefore, $\text{cov}_s(h) = \frac{1}{3}(\frac{5}{7} + \frac{4}{6} + \frac{7}{11})$ which is approximately 0.67.

As one can see from the last example, there is a loose relation between precision and recall or accuracy and coverage: The more detailed our information, the greater the chance to be wrong on more general cases. As a rule of thumb: The higher precision, the lower recall—and vice versa. The same holds for accuracy and coverage. Accordingly, a more detailed evaluation takes into account *both* or a combination of them (known as f–measure in information retrieval).
One could also define

$$acc_s(h) = \min_{i \in \mathbf{k}} \mathrm{prc}(h(x) = c_i) \tag{3.38}$$

such that the overall hypothesis quality is determined by the weakest local predictive preciseness (coverage can be defined analogously). Furthermore, it is quite common to test the quality of h on a set s that is *disjoint* from the sample $\mathbf{s}$ that we used to learn h. Only then we can evaluate h's quality in terms of generalization: is h's predictive competence on new evidence as good as it is on the cases `Alg` has already used to build h? Therefore, one usually splits a provided sample $\mathbf{s}$ into two disjoint subsets for learning and evaluating h:

Training set, Validation set, $\mathbf{s}_{\text{train}}, \mathbf{s}_{\text{val}}$

Definition 3.23 — Training set, Validation set, $\mathbf{s}_{\text{train}}, \mathbf{s}_{\text{val}}$.
We split $\mathbf{s}$ into two disjoint samples $\mathbf{s}_{\text{train}}$ and $\mathbf{s}_{\text{val}}$. Only the *training set* $\mathbf{s}_{\text{train}}$ is used for learning. We then evaluate $\texttt{Alg}(\mathbf{s}_{\text{train}})$ against its predictions on objects from the *evaluation set* $\mathbf{s}_{\text{val}}$.[11] ●

A this point, a student usually claims he has understood accuracy and coverage measures. However, there are hundreds of different quality measures around—some have small "corrective" parameters, others are computed on specially chosen subsets of the training or validation set and so on. Therefore, one *always* has to be very, very careful when comparing the results of different evaluations—accuracy may not always be the same.
A detailed analysis of the predictive power of h always requires a decent amount of statistical evaluation including κ–analysis or ROC/AUC analysis. However, this is beyond the scope of this book. For the interested reader we recommend [?].

Evaluation of hypotheses (II)
Learning takes place in representation space. Therefore, evaluation of h must be carried out in $\mathfrak{U}$ as well. It is impossible to determine the true error of some h which is why we have to make do with error estimates and functions thereof (like accuracy, coverage, etc). To increase the accuracy of these estimates one divides the given sample into two disjoint subsets; one for training and one for evaluation.

3.5 Learning

In the last section we have introduced many methods and measures to evaluate hypotheses delivered by a machine learning algorithm given a set of examples.

[11] In many books on Machine Learning $\mathbf{s}_{\text{train}}$ is called a *test set* (with subscript t). This may lead to confusion, especially since some algorithms use a part of $\mathbf{s}_{\text{train}}$ for internally testing ($\mathbf{s}_{\text{test}} \subseteq \mathbf{s}_{\text{train}}$) by some validation bias.

But aren't the properties of hypotheses delivered by `Alg` given **s** properties of `Alg` as well (if we assume `Alg` to be deterministic)? The general idea leads to computational learning theory which is concerned with the question of learnability.
For now, we simply need to understand that from the input/output behavior of an algorithm one can infer several properties. Therefore, we ask:

- Given which data (sample), what kind of hypotheses are generated?

The definitions of correctness can be expanded to `Alg` as follows:

- `Alg` is called *s–correct*, if it generates a $\langle t, s\rangle$–correct h:

$$h(x) = t(x), \forall \langle x, t(x)\rangle \in s \text{ and } \texttt{Alg}(\mathbf{s}) = h$$

- `Alg` is called *correct*, if it is $\langle t, U\rangle$–correct h.

But the big question is: Are there correct `Alg` at all? Are there "nearly" correct `Alg`? Practically, this depends on the learning problem. But generally, answers to these questions or answers that even include an estimate of the degree of correctness are subject of learning theory again. Nevertheless, without going too much into detail here, we *always assume that given some example we can gather at least a rough picture*. This is the fundamental *inductive hypothesis* which is the only bias that is inseparably connected to any approach to machine learning:

Inductive hypothesis

Definition 3.24 — Inductive hypothesis.
If h approximates t sufficiently well on a sufficiently large sample **s**, it will approximate t on U, too. ●

Based on the inductive hypothesis, machine learning algorithms try to deliver "good" hypotheses. Practically, this creates the problem to solve *how* one can find a good approximation h of target concept t. Theoretically, it is the easy part: All one has to do, is to:

1. Collect all possible (representable) hypotheses
2. Arrange (order) them in a nice way
3. and then ... *search for the best*!

To get it done efficiently, some systematics might be quite helpful. But systematic search requires a guided search, which in turn needs a guide. Then, machine learning becomes a classical search problem: We need to find a good

1. representation space that can be navigated,
2. distance measure which helps us to compare alternatives and choose the most promising ones using a suitable
3. heuristics.

This not overly impressive insight makes a nice working definition of machine learning but does not suffice to replace definition 3.11. Therefore, we simply conclude:

> **Machine Learning as Search**
> Machine Learning means to *search* for a suitable hypothesis in a hypothesis space.

3.6 Bias

When concerned with search, one has to deal with two fundamental problems. First, if there are solutions, one should be able to find at least one. Second, if there are solutions, one should be able to find the "best" one as quickly as possible. To do so, it is a good plan not to search in places where we know we can not find a solution. It is also a good idea to look for solutions in places where we expect them to be. In other words:

- One would like to *restrict the search space.*
- One would like to define a *threshold.*
- One would like to determine *bounds* in a lattice.
- One needs a suitable *bias* in the search process.

By ordering the set of hypotheses, by restricting the search space or heuristically guiding the search thorough the space, we also predetermine whether we find a certain hypothesis at all and if so, when we will find it. By ruling out impossible cases, we *deliberately* introduce bias as *additional knowledge* for a better control. But at the same time we restrict ourselves in representation, abstraction and the degrees of freedom in the search for a solution. The more we reduce the expressiveness of the representation language (the complexity of representation space), the smaller the language becomes and, accordingly, the smaller the hypothesis space.

Example 3.19 Imagine we chose full predicate logic as the hypothesis space. By restricting ourselves to Horn logic we consider only a subset. A further language bias could be to restrict the number of different variables that may occur within a formula. ●

This kind of *language bias* is very effective, but it is very likely that by simplifying $\mathfrak{U}$ we cannot represent a hypothesis that explains our learning target in $\mathfrak{D}$ any more.

Exercise 3.18 (◊) Explain the danger of losing too much expressiveness by language bias! Build your arguments on the definitions of $\mathfrak{D}$, $\mathfrak{U}$, ρ, and τ!

Exercise 3.19 (⧫) If you are familiar with Prolog, consider the following definition of the membership predicate:

$$
\begin{array}{lcl}
\texttt{member}(X,[Y|Z]) & \texttt{:-} & X = Y. \\
\texttt{member}(X,[Y|Z]) & \texttt{:-} & \texttt{not}(X = Y), \texttt{member}(X,Z).
\end{array}
$$

Redefine the predicate with a minimal number of variable names. Why is such a definition more desirable?

Now that we have chosen an (already biased) representation space we need to think about how to quickly navigate through space in order to find a good solution as quickly as possible. By guiding the search we apply *search bias*: First of all, the general principle of search determines the behaviour of `Alg` finding a solution. Imagine how breadth-first search or depth first search would deliver different solutions on the same graph in hypothesis space. But since hypothesis space usually is very large one needs to utilise some kind of intelligent, heuristic search rather than exhaustive search. But then, the accuracy of the heuristic measure also determines the search result.

Example 3.20 Consider we want to learn intervals of real numbers on the ray. Then, choosing the step size by which we increase the interval boundaries in relation to the numbers given in $\mathbf{s}$ can be used as search bias; the longer we try to adjust our interval boundaries it occurs reasonable to reduce the step width in order to avoid some oscillation behavior—which would be another good idea for search bias. ●

Example 3.21 In the domain of logic expressions, consider the following two observations: $p =$ *I have seen lots of black crows* and $q =$ *I have never seen a white crow*. Now, which of the following inductive generalisations appears most suitable to you: $h =$ *There are no white crows* or $h' =$ *All crows are black*. The problem is that $p \mathrel{|}\!\!\approx q$ but $q \mathrel{|}\!\!\not\approx p$. And since entailment is not always easy to show, one would opt for an easier measure (heuristic) to determine which hypothesis to chose. ●

Finally, a very important kind of bias is *validation bias*. Simply speaking, validation bias defines our tolerance threshold that tells us when to stop searching and return the current best hypothesis as solution. There are many such biases possible. A simple example is that h has to reach a certain degree of accuracy or coverage on a validation set. One could also define a validation bias in conjunction with search bias as follows:

Example 3.22 Consider a search routine that delivers as solution the first local maximum of accuracy using a gradient descent search. Then, it would be a reasonable idea to decrease step size with decreasing gradient and, once the gradient has fallen below a predefined threshold, we simply abort and return the current hypothesis. ●

Exercise 3.20 (♦) Explain search and validation bias using $\mathbf{s} = \{\langle x, t(x)\rangle : x = 1, 2, \ldots n\}$ where t is the characteristic function of the subset of all prime numbers in $U = \{1, 2, \ldots n\}$. Assume that $\mu(\{x\}) = \frac{1}{n}$ for all $x \in U$.
(Hint: Choose several and some very large n!)

Figure 3.5 shows an abstract learning algorithm that utilises all three different kinds of biases: In lines 3 and 6 validation bias is applied. Non-deterministic

```
01  $H = \text{init}();$
02  $C = \{\};$
03  WHILE (!(stop_crit($C, t, \mathbf{s}_{\text{test}}$))) DO;
04  {
05      $h := \text{choose}(H)$; $H := H - \{h\}$;
06      IF (good($h$)) THEN
07          $C := \text{filter}(C \cup \{h\})$;
08      ELSE
09          $H := H \cup \text{refine}(h)$;
10      ENDIF
11  }
12  return($C$)
```

Figure 3.5: A biased Learning Algorithm

choice in line 5 usually is biased by search and filtering and refinement as in lines 7 and 9 require search and language bias.

Bias
Bias is a crucial concept in machine learning. On the one hand, *bias* makes learning feasible. But on the other hand, any kind of bias (may it be deliberate bias or unwanted bias such as noise) may cause accidental pruning of better results from hypothesis space.

Talking about *deliberate* bias, we also need to discuss *unwanted* biases. The most important ones are *sampling bias* and *selection bias.*
Any sub set of entities that is drawn from a larger set is a *biased sample*, if the probability of the entities being drawn is not independently identically distributed. In other words, any sample is biased by μ. But then, it would be rather *good* to have a *similar*, not independent identical distribution (i.i.d.) in the sample. The only problem is that, since μ is unknown, we just can't prove, whether our sample is biased in a way that corresponds to μ.

Biased samples

Definition 3.25 — Biased samples.
A sample is *biased* if some members of the domain are more likely to be chosen in the sample than others. The larger the sample, the more the distribution ϕ on $\mathbf{s}$ approximates the distribution μ on $\mathfrak{D}$. It is clear that any $\mathbf{s} = S_\mu(m, t)$ is *biased* to a certain degree since we can only draw finitely many finite samples (see theorem 3.1). ●

Selection bias is used to control sampling bias. If we know of a subset that preserves μ, then we choose just this subset as a sample. One famous example for "representative samples" is a small town in the North of Germany which

was a nearly *un*biased estimator for nationwide elections in Germany. Another important application of deliberate selection bias is learning by *boosting.*
Confirmation Bias is more a cognitive bias than a phenomenon that is observed in learning machines. Therefore, it plays a central role during the representation process: knowledge engineers usually represent data in a way that is already structured with respect to an intended meaning or a supposed model. Since a knowledge engineer does not realise that his understanding of the data already determines what kind of unknown information can be extracted, confirmation bias is a very important, hidden unwanted bias. Also, learning algorithms are susceptible to confirmation bias in a very interesting way. If we consider a learning algorithm as in figure 3.5, then it becomes obvious that the choice of some h in line 5 determines the behaviour of the algorithm in the next `WHILE`-loop. As a consequence, the *sequence* of examples and the sequence of hypotheses as generated play an important role in the selection of the next example and the choice of a next hypothesis.[12]
So far, we have seen that there is always hidden noise and hidden bias—and still hypothesis space is too big to be searched efficiently without any further deliberate bias. In fact, we shall discover that results can always be optimised or fine–tuned to a certain degree. But "cutting-edge" optimisation always tends to become optimisation with respect to a fixed set of observations—which will lead to *overfitting.* The reason for this is beautifully explained by the *no-free-lunch-theorem.*

The No Free Lunch Theorem

> There Ain't No Such Thing As A Free Lunch
>
> R. A. Heinlein, *The Moon Is A Harsh Mistress*, 1966

This means, that in any system (society) no-one can get anything by the price of nothing. Even if there is a happy hour—you (or somebody else) always pays for the loss (usually through higher regular prices). More formally, we define:

Theorem 3.2 (No-Free-Lunch Theorem (NFLT)) All algorithms from a set of search algorithms looking for an optimal solution using a (local) cost function perform exactly equally well when averaged over all possible cost functions (i.e. problems). [Wolpert and Macready, 1997]

If we assume the average to be constant, a higher peak performance means a lower average performance as shown in figure 3.6: The solid graph is a hypothesis with very high accuracy in just a small region and a less than average accuracy everywhere else. The dotted line represents a simple hypotheses that in average performs equally well—but without areas of excellent expertise or complete failure. This puts us in a quite embarrassing position somewhere between pride and prejudice: When being *biased*, one seeks to *optimize* a concept that is based

[12] There are many interesting results for so-called *online-learning* algorithms which for every single example they receive deliver a hypothesis which then becomes refined with every additional example. For example, some artificial neural network architectures and learning algorithms are very error prone in such cases, which is why sample sequences are shuffled.

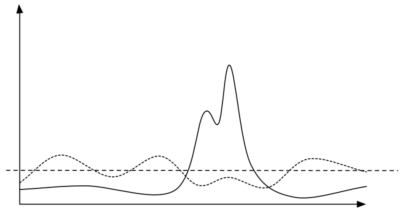


Figure 3.6: The No Free Lunch Theorem

on *prejudice*. Once we have found such a "optimal" concept, it is announced with quite inappropriate *pride*. But one consequence of the NFLT is that *the better your peak performance on known cases, the worse your performance on unknown cases.*

Biases
There are numerous biases involved in learning. Many of them are wanted, others are unwanted. The most important bias in learning is *inductive bias.* If we want to learn by examples, we *must* assume that from a small set of observations we can inductively conclude to all observable cases.

3.7 Overfitting

In terms of the no-free-lunch-theorem, *overfitting* refers to the phenomenon of peak performance on training data $\mathbf{s}_{\text{train}}$ but a dramatic loss of accuracy on the remaining data in $\mathbf{s}_{\text{test}}$ or $\mathbf{s}_{\text{val}}$.
How come we find ourselves in such a situation? Usually, we try to learn by adapting as precisely as possible. But learning to solve a specific task as accurate as possible does not necessarily help to improve our abilities in solving a similar task. So again with the no-free-lunch-theorem in mind, it appears more suitable to solve a few tasks somewhat better than solving one perfectly and being an utter failure on the remaining cases. Therefore:

Overfitting
In order to be able to learn, one should risk to perform *locally suboptimally.*

It is very nice if h agrees with t on s, but if h disagrees with t on all remaining $x \in U - s$, h is *useless*: First, U usually is much larger than s. And second, the reason to learn is to achieve knowledge from s which explains an entire concept we can use to describe arbitrary sets of new objects! The phenomenon of generating some h that precisely explains $\mathbf{s}$ and is useless otherwise is known as *overfitting*. Since t is defined on $\mathbf{s}$ only, it is hard to determine whether h is overfit. Therefore, $\mathbf{s}$ is split into disjoint $\mathbf{s}_{\text{train}}$ and $\mathbf{s}_{\text{val}}$. Then, Overfitting, overfit h

Definition 3.26 — Overfitting, overfit h.
A hypothesis h *overfits* on t on s if there is another hypothesis h' such that:

$$\begin{aligned} \mathrm{error}^{\mu}_{\mathbf{s}_{\mathrm{train}}}(h,t) &\leq \mathrm{error}^{\mu}_{\mathbf{s}_{\mathrm{train}}}(h',t) \\ \mathrm{error}^{\mu}_{\mathbf{s}_{\mathrm{val}}}(h,t) &> \mathrm{error}^{\mu}_{\mathbf{s}_{\mathrm{val}}}(h',t) \end{aligned}$$

h is overfit, if there is another hypothesis h' no better than h on the training sample but better on unseen cases.[13] ●

In other words, if h is overfit there is no real generalisation progress and hence no learning.

3.8 Summary

If you got this far, the rest of the book is a piece of cake.
This section gave an introductory overview into the scenario where knowledge discovery takes place. We are simply concerned with a set of data that is represented in an information system. Our task is to understand and formulate a general concept that is supported by a set of examples that may be labelled with some additional information from a teacher.
We have seen that machine learning can be related to compression, to function approximation, or information theory. We have learned, that there is a fundamental source of noise and information loss in the process of representation—but we also have seen that the change of representation may result in much easier learning problems and that some further deliberate bias may help to find better hypotheses even quicker.
In the following chapters we will discover the exciting research discipline of machine learning with many different algorithms from different approaches. The difference between all those approaches can be described by the primary motivation behind machine learning:

- Scientific Discovery vs. Data Mining
  Am I interested in the discovery of new knowledge that has not been discovered yet? Then, we will need a lot of background knowledge to describe our data in an appropriate representation space and we need to interpret the resulting hypothesis in our domain.
  Data Mining is more focused on the facts: What kinds of patterns are there in the data and what can be inferred from these patterns?

- Symbolic vs. Sub-symbolic
  Most people believe we think rationally, and most people assume that rational thinking means logical thinking. This is not the case. However, we *like* to think we think logically. And, undoubtedly, terminological representations have been proven means for communication knowledge.

[13] The choice of the actual error measure is subject to our own deliberate validation bias.

Accordingly, one might consider implementing learning as process that takes place in a space of logics with symbols of determined semantics. On the other hand, the human brain, which we can safely assume to be the place where learning takes place, is neither a discrete, nor a logical apparatus. Instead, if there is information at all, it is distributed.

- Engineering vs. Cognition
  Finally one can be interested in machine learning simply out of the need for handling or describing or explaining huge sets of data. Then one takes the data, and applies appropriate algorithms until the results appear reasonable. That would be an engineer's approach. A psychologist might be interested in a human's learning behavior and in order to test his theses he might want to simulate his model of human learning using a machine. If a machine then shows similar learning progress than a human does this might support the thesis that human learning works similar to the method implemented.

But whatever the objectives are when concerned with machine learning, one always has to keep in mind that:

Knowledge Discovery & Machine Learning
Knowledge discovery has nothing to do with knowledge—and Machine learning has nothing to do with learning. All we can do is to identify patterns and give them names or analyse their correlations. The process of knowledge discovery or machine learning has nothing to do with *intelligence*, either: Knowledge is *represented* by *symbols*, and reasoning about knowledge is *simulated* by simple rules for symbol manipulation. Machine Learning means to find such rules for symbol manipulation, and, therefore

Knowledge Discovery and Machine Learning are simple calculus.

If there is any intelligence or knowledge involved in these processes at all, then it is only in *our* representation of the problem and *our* interpretation of the outcome.

Chapter 4

Clustering

> If we are given five pebbles, three marbles, four dice and two keys, then we have 14 little objects of four different kinds. We also have 8 objects made of stone, four made of wood and two made of metal. And we have seven toy objects, two office tools and five things we have collected during our last walk at the beach.

In the previous chapters, we have seen that relations can be used to represent knowledge about sets of things. We also discovered that learning means to *find* a suitable set of relations with which we can describe or define concepts (see Definition 2.11). Now, we describe a first approach to efficiently discover relational concept descriptions. Our starting point is an information system with a feature-based representation of the objects in our domain.

4.1 Concepts as Sets of Objects

Our working hypothesis is that knowledge is the ability to discriminate things and learning is knowledge acquisition. Therefore,

> learning means to acquire the ability to discriminate different objects from each other.

There are, in general, two different methods to group similar objects together and distinguish them from other groups of entities:

- building sets or classes of objects which we assume to share certain properties by grouping them into the same *cluster*
- or by inducing a *concept* that serves as a description of a representation class in terms of properties of objects.

The problem is that the latter requires more knowledge about the world and the entities, while the former just requires some kind of "distance" measure that reflects similarity of objects.

Exercise 4.1 $\Diamond$Find several classifications of the set

$$\{\oslash, \Diamond, \oslash, \bullet, \blacklozenge, \Diamond, \circ, \Box, \bullet, \blacksquare, \blacklozenge, \blacklozenge\}$$

by clustering the elements! — Do the same using conceptual descriptions of the objects!

To form groups of similar objects or in order to classify a certain (new or unknown) object as a member of one of the classes, we must be able to:

1. to tell which group of objects a new object belongs to

2. form cluster sets based on some information concerning their similarities

3. form clusters based on a teacher's information and conceptual descriptions

The first action means to *classify* objects with respect to their similarities to other objects. It is assumed that there already exists a classification and that it can be expressed in terms of the similarity measure. The latter two actions are processes by which we learn to group objects to form elements of a classification, i.e. classes. The first one is done by unsupervised learning and the second one by supervised learning. To illustrate the idea behind similarities and feature–based indiscernibility, consider the following example:

Example 4.1 Imagine we have four objects $\{\Diamond, \blacklozenge, \circ, \bullet\}$. They are described by the features *shape* and *fillstyle* which yields an information system as follows:

	shape	*fillstyle*
$\circ$	round	hollow
$\bullet$	round	solid
$\Diamond$	polygon	hollow
$\blacklozenge$	polygon	solid

So, $\bullet$ is similar to $\circ$, because they are both round, and $\Diamond$ is similar to $\circ$, because they are both hollow. But $\circ$ and $\blacklozenge$ have a maximum dissimilarity because they do not share any feature values. Geometrically, this yields the following diagram:

$$\begin{tikzcd}
\Diamond \arrow[d, "f"'] & \blacklozenge \arrow[dl, "\sqrt{f^2+s^2}"] \\
\circ & \bullet \arrow[l, "s"]
\end{tikzcd}$$

where the diagonal entries represent maximum dissimilarity if we assume the two features to create an euclidean two–dimensional space (with *fillstyle* creating the horizontal and *shape* defining the vertical dimension). ●

Exercise 4.2 $\blacklozenge$You might be surprised why this simple questions earns a black $\Diamond$: In the last example a fundamental assumption was not mentioned. Which one?

4.2 k–Nearest Neighbours

Nearest neighbour classification is a very simple and human thing to do: Imagine 90 percent of the people from a city block anywhere in the world speak, say, French. Someone else living in this neighbourhood will speak French with a very high probability, too. Reversely, if most of your neighbours live, for example, in Paris then the probability that you live in Paris too, is very high.

> **k–NN**
> k–NN is simply a majority voting method for classification: Assign to an unknown entity the same label as most of the most similar entities have.

Assume we arrange all objects of our domain in a space that is defined by the features we use to describe our entities. There can be two, three or many more such dimensions and they can be discrete or continuous. Then, we add knowledge concerning the target class of each object by assigning it an according label, we will receive a more or less coherent distribution of target labels in this space. If we now encounter a new instance, we simply put it on its proper place in this space and assign it the most common target class label among the k nearest instances.
To classify an object using information from its neighbourhood, we simply assign to it the class label most of the k–nearest neighbours share: Let $U = \mathbb{R}^n$, i.e. $|\mathbf{F}| = n$. Then, every $\vec{x} \in U$ has the form

$$\vec{x} = \langle f_0(x), f_1(x), \ldots, f_{n-1}(x) \rangle$$

The euclidean distance between two object $\vec{x}$ and $\vec{y}$ is

$$\mathrm{dist}(\vec{x}, \vec{y}) = \sqrt{\sum_{i \in \mathbf{n}} \left(f_i(x) - f_i(y)\right)^2} \tag{4.1}$$

Given an information system $\mathfrak{I} = \langle \mathfrak{U}, \mathbf{F} \cup \{t\}, V_{\mathbf{F}} \cup \mathbf{c}\} \rangle$, the classification for an unknown object $x \in \mathfrak{U}$ works as follows: First, using $\mathbf{F}$, x is represented as $\vec{x}$. Let $t : U \to \mathbf{c}$ be a classifier for $\mathfrak{c}$. Then $t(x) = ?$. Next we collect a set of $\vec{y} \in U$ as follows:

$$kNN(\vec{x}) := \{\vec{y} \in U : \mathrm{dist}(\vec{x}, \vec{y}) \leq r\} \tag{4.2}$$

where r is the smallest value such that N contains k elements. The k–nearest neighbour classifier then predicts that x belongs the same class that most of the k neighbours belong to: In other words, x has the "most common value". This concept is important enough to deserve a definition on its on:

Most common values, mcv

Definition 4.1 — Most common values, mcv.
Let s be a set and f a total function. We define the *most common value* of f on a set s as the value $c \in \overline{f}^{\urcorner}$ with the largest preimage ${}^{\ulcorner}\!fc$:

$$\mathrm{mcv}_f(s) \quad := \quad \arg_c \max \left\{ |{}^{\ulcorner}\!fc| : c \in \overline{f}^{\urcorner} \right\} \tag{4.3}$$

●

Defining a function that determines the most common value is rather cumbersome (and it is difficult to formalise it in a way that is easy to understand). Maybe this is why in most textbooks $\mathrm{mcv}_f(s)$ is defined in prose only. But now that we have a proper definition, we can happily carry on with a satisfying definition of a k–nearest neighbour hypothesis.

k–nearest neigbhour classification hypothesis

Definition 4.2 — k–nearest neigbhour classification hypothesis.
Using the most common value, we define

$$h_{\mathfrak{c}}^{kNN}(x) \quad := \quad \mathrm{mcv}_t(kNN(x))) \tag{4.4}$$

●

As already mentionend this is a simple voting approach: x is assigned the value that most eligible voters have.

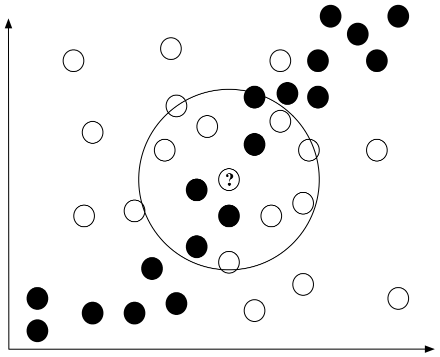


Figure 4.1: k-nearest neighbours

Figure 4.1 shows a $k = 12$ case where the radius around the unknown object in the center is just the right size for the sphere to cover 12 cases in space. In this sphere, most objects are white (7), which is why we would assign *white* as class label to $\vec{x}$ as well. Accordingly, the k–nearest neighbour algorithm can be defined as follows:

k–nearest neighbour Algorithm

Definition 4.3 — k–nearest neighbour Algorithm.
The following procedure predicts $\mathbf{t}(\vec{x}) \in \mathbf{k}$ for some $\vec{x} \in \mathfrak{U}$:

```
$kNN := \emptyset$; $r = 0$; $t_0 = t_1 = \cdots = t_{k-1} = 0$;
WHILE ($|kNN| < k$) DO
{
    $r := r + \varepsilon$;
    $kNN := \{\vec{y} : \mathrm{dist}(\vec{x}, \vec{y}) \leq r\}$
};
FOREACH ($i \in \mathbf{k}$) DO
{
    $t_i = |\{y \in kNN : \mathbf{t}(y) = i\}|$;
};
return $\arg\max_{i \in \mathbf{k}} t_i$
```

where ε is the step width with which we increase the radius of the sphere. ●

It is important to understand, that this classification here is simply due to the number of black or white dots in the sphere, but not to the location of $\vec{x}$ in relation to all other dots—it is a pure incident that $\vec{x}$ appears to belong a diagonal "milky way" of black dots entities.

Taking into account the spatial distribution of objects of each classes, we will find a centre of gravity for each class which also represent the "average" or prototypical representative of this cluster. This is depicted in figure 4.2. The diamonds represent the center of the classes of dots of according colour. Note that these points do not have to exist as actual data points but only represent the clusters that are formed by the objects. For an unclassified object $\vec{x}$ (labelled with a question mark) we then predict its class by the class of the nearest centroid—in this case the class of light grey objects. Note further that using a standard *kNN*-methods the result would be quite different!

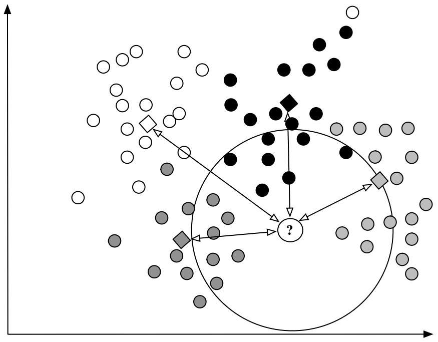


Figure 4.2: Clusters and Cluster Centers

So once we have a spatial representation of objects described by an information system, there are many different ways to model different metrics or to define different methods for classifiers all of which focus on different aspects in classification: While k–nearest neighbours is rather a majority voting approach, a distance measure rather refers to prototypes of clusters. Classifying an unknown object by computing its distance from or degree of membership to certain clusters or cluster centroids requires us to

1. have an extensional cluster description and/or
2. an intensional cluster description with boundaries and/or centre and radius information.

Classification itself is not the main issue in machine learning; machine learning is rather concerned with the problem of *finding* the clusters.

4.3 k–Means Clustering

Trying to learn clusters in an extensional distance–measure setting again offers several general methods. The two most important ones are to generate clusters on a set of data where we are given the number of clusters we want to obtain. The second clustering method divides the data set into clusters until the objects grouped together have a minimum mean distance but a maximum mean distance to all other objects.
We first discuss clustering with a fixed number of target clusters.

> **k–means Clustering**
> The idea behind k–means clustering is to randomly define k cluster centroids. Then, every object in our domain is assigned the cluster id of the closest cluster centroid. The actual dynamic means clustering now repeatedly recomputes the center of each cluster. Since the center "moves", it is quite likely the cluster itself moves, too: The bigger the step, the more likely some entities will be assigned different cluster id's in the next step—most likely those from boundary regions. If the centroids do not move any further, we are done.

If we want to discriminate k clusters, we randomly chose k initial cluster centroids $\vec{c}_i \in \mathbb{R}^n$, $i \in \mathbf{k}$. Then, every point $\vec{x}$ is assigned a class label i where

$$h(\vec{x}) = i := \arg\min_{i \in \mathbf{k}} \operatorname{dist}(\vec{x}, \vec{c}_i)$$

such that $c_i := \{x \in U : h(\vec{x}) = i\}$. The first step means to randomly distribute some cluster centroids (which correspond to the diamonds in figure 4.2) over the entire representation space. In the second step, each object is assigned the target value of the nearest such centroid as defined above. In most cases, the initial cluster centroids are distributed without any correspondence to actual clouds in the data distribution of the representation space. In order to make the cluster centroids move to where the actual data clouds are, we repeatedly redefine class centroids by

$$\vec{c}_i := \frac{1}{|c_i|} \sum_{\vec{x} \in c_i} \vec{x}$$

until the classification is "good enough". Every redefinition step causes all the centroids to move towards the center of the sets of points that were classified as objects of the corresponding class in the previous step. But since the set of objects that belong to this center is redefined in each iteration too, the center can move across the whole set. This yields an algorithm called *k–means clustering*:

Definition 4.4 — k–means Clustering.

k–means Clustering

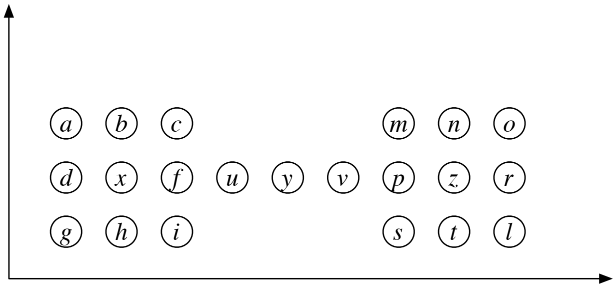
The following procedure generates k clusters on a set of multidimensional data $U \subseteq \mathbb{R}^n$:

```
FOREACH ($i \in \mathbf{k}$) DO { $\vec{c}_i := \texttt{randomelement}(\mathbb{R}^n)$; $c_i := \emptyset$; } DONE
WHILE ($\mathbf{1}$) DO
{   FOREACH ($\vec{x} \in U$) DO $h(\vec{x}) := \arg\min_{i \in \mathbf{k}} \mathrm{dist}(\vec{x}, \vec{c}_i)$ DONE
    FOREACH $i \in \mathbf{k}$ DO
    {   $c_i := \{\vec{x} \in U : h(\vec{x}) = i\}$;
        $\vec{c}_i := \frac{1}{n} \cdot \sum_{\vec{x} \in c_i} \vec{x}$;
        $\vec{\delta} := |\vec{o}_i - \vec{c}_i|$;
    } DONE
} DONE
```

with scalar multiplication $\cdot$ and component-wise addition. ●

A closer look reveals that this method depends on the initial distribution of centroids.

Exercise 4.3 ♦♦Discuss the k–means clustering algorithm on the following example:



Consider $k = 2$ with u and v or x and u being initial centroids.—Consider $k = 3$ with x, y and z or u, y and v being initial centroids.—Consider $k = 2$ and $k = 3$ with initial centroids u, v and u, v, z where point y is absent.

Exercise 4.4 (♦♦) Write a program that performs k–means clustering on n–dimensional data.

The problem with membership is that regions are not always defined by crisp boundaries. Just as we consider distances in space here and a centroid as a prototype of concept, some things more or less belong to a class or a concept. Similarly, an object that is close to the center of a cluster appears to be more like the prototypical element of the cluster than one object in the boundary region. As soon as boundaries are blurred, there is some kind of *fuzzification* involved. The idea is more than simple: Let $\mathrm{cod}(f) = V_f = \{v_0, v_1, \ldots, v_{m-1}\}$. Then, $\tilde{f}(x) = \langle \mathrm{dist}(x, c_0), \mathrm{dist}(x, c_1), \ldots \mathrm{dist}(x, c_{m-1})\rangle$. So instead of assigning x a single value $f(x)$, it is assigned a vector of distances to each of the value's representative centroid. If the all vectors are normalised such that the sum of their arguments becomes 1, then each argument expresses the probability that x takes value v_i.

Probabilistic (Fuzzy) Classification

Definition 4.5 — Probabilistic (Fuzzy) Classification.
Let $\mathfrak{c} = \{c_0, c_1, \ldots c_{k-1}\}$ be a classification. We define a *fuzzy classification* by assigning to each object a vector of k probability values each of which describes x's degree of membership to the according class:

$$\begin{aligned} \tilde{\chi}(c_i) &: U \to [0,1] \\ \tilde{\chi}(c_i)(x) &:= \phi(\{x\} \cap c_i) := n \cdot \mathrm{dist}(\vec{x}, \vec{c}_i) \end{aligned} \tag{4.5}$$

where n is a normalisation factor such that $\sum_{i \in \mathbf{k}} \mathrm{dist}(\vec{x}, \vec{c}_i) = 1$. ●

Exercise 4.5 (♦) Let there be a fuzzy classification $\mathfrak{c}$ with a fuzzy membership function $\tilde{\chi}(\mathfrak{c})$. Define a method for *defuzzification* which takes $\tilde{\chi}(\mathfrak{c})$ and returns k characteristic functions for each class in $\mathfrak{c}$ (see definition 3.5).

Exercise 4.6 (♦♦) Let there be two binary fuzzy classifications $\mathfrak{c} = \{c, -c\}$ and $\mathfrak{c}' = \{c', -c'\}$. Give a definition for $\chi(c \cap c')(x)$ in terms of $\tilde{\chi}(\mathfrak{c})$ and $\tilde{\chi}(\mathfrak{c}')$! Define the truth value of the expression $x \in c \vee x \in -c'$! — Congratulations! You now know everything one needs to know about Fuzzy Logic.

Using this fuzzy membership and the distance measure, one can easily define a *fuzzy k–means clustering* algorithm:

Fuzzy k–means Clustering

Definition 4.6 — Fuzzy k–means Clustering.
The following procedure generates k fuzzy clusters on a set $U = \mathbb{R}^n$ of data:

```
FOREACH $(i \in \mathbf{k})$ DO { $\vec{c}_i :=$ randomelement$(\mathbb{R}^n); c_i := \emptyset$ } DONE
WHILE $(\mathbf{1})$ DO
{   FOREACH $(\vec{x} \in U)$ DO
    {   $\vec{c}(\vec{x}) := \langle \mathrm{dist}(\vec{x}, \vec{c}_0), \mathrm{dist}(\vec{x}, \vec{c}_1), \ldots, \mathrm{dist}(\vec{x}, \vec{c}_{k-1}) \rangle$;
        $h(\vec{x}) := \arg\min_{i \in \mathbf{k}} \mathrm{dist}(\vec{x}, \vec{c}_i)$;
    } DONE
    FOREACH $i \in \mathbf{k}$ DO
    {   $c_i := \{\vec{x} \in \mathbb{R}^n : c(\vec{x}) = i\}$;
        $\vec{c}_i := (\sum_{\vec{x} \in c_i} (\mathrm{dist}(\vec{x}, \vec{c}_i) \cdot \vec{x})) / (\sum_{\vec{x} \in c_i} \mathrm{dist}(\vec{x}, \vec{c}_i))$;
    } DONE
} DONE
```

The only difference to non-fuzzy clustering is that the centroids do not move to the center to the objects but to the distance-weighed center. ●

There are hundreds of different improvements on these base algorithms. First of all, the validation bias (i.e. the stopping criterion; in our case **1**) can be defined in relation to other (dynamic) parameters, i.e. maximum intra–cluster distances. One can also introduce different distance measures like quadratic functions or other non–euclidean measures. It is also possible to have different dimensions of the vectors weighted differently. [**?**] discusses several more advances in clustering and many other probabilistic approaches. But with increasing demands of machine learning towards the induction of conceptual or "semantic" hypotheses, one wants to go a step beyond, say, "descriptive" clustering. For example, it

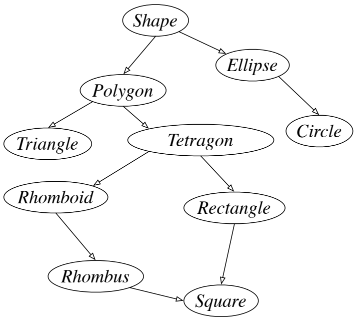


Figure 4.3: A Concept Hierarchy of Geometric Shapes

appears that if a human begins to cluster observations, he does so by grouping similar objects into few clusters and then recursively clusters each group with respect to an increased amount of detail in (dis-)similarity. At a certain (rather early) level, dogs and cats belong to the same cluster of pets. Only with more detailed knowledge, one can discriminate cats from dogs—and generalise cats and dogs to carnivore pets (as opposed to hamsters).

4.4 Incremental Concept Formation

In the last section we learned how to *classify* a new object into a given classification of clusters in space by using an euclidean distance measure as a measure of similarity. In the next step we discovered two algorithms to *discover* a classification based upon unsupervised clustering of multidimensional data. If we now understand multidimensional data points as entries in an information system, the task ahead is *unsupervised learning* of *concepts* from examples that are described by an information system. Just to recall the difference behind classes and concepts: Of course, concepts in U are *sets*. Therefore, they are classes as well. But the classes originate from classifications which in turn correspond to *elementary categories*. They are sets of things that share a certain property. A *concept* is a *description* of properties in terms of elementary categories (building *basic categories*): Being a "white square" means to be an element of the set of white things *and* of the set of squares.

Again, our goal is to group *similar* projects into one class but this time by generating hierarchies of concepts: Rhomboids are a special kind of tetragons, but squares are even more special. An they are all different from triangles, but even more different from circles. The relationships between geometric figures is shown in figure 4.3.

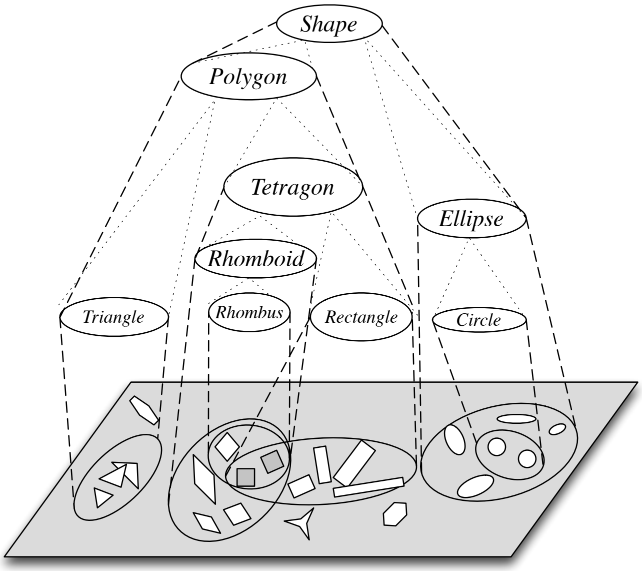


Figure 4.4: Hierarchical Incremental Clustering

Incremental Concept Formation
In contrast to classification with given classes or clustering given a fix number of target classes, *concept formation* seeks to build hierarchies of concepts that reflect partitions of objects at different levels of granularity. We can clearly discriminate parrots from robins and dogs from cats—but together they form the concepts of birds and mammals.
Incremental Concept Formation seeks to find a hierarchy of such concepts such that (1) the similarity of all objects in a class is maximised, (2) the dissimilarity of all concepts is maximised, and (3) the conceptual structure is as simple as can be.

The process of incremental (top–down refinement) clustering is depicted in figure 4.4: the basic classes correspond to the smallest clusters on the first level of abstraction. Here different objects of the same kind are grouped together: *Triangles*, *Rhomboids*, *Rectangles*, and *Circles*. On the second level, the cluster of *Tetragons* is made up from *Rhomboids* and *Rectangles*, while *Circles* are generalised to *Ellipses*. On third level, *Triangles* and *Tetragons* form *Polygons*, and all together finally form the concept of *Shapes*.
But the human cognitive apparatus does not really work in one direction only: When learning concepts, we simultaneously learn by abstraction (bottom–up, generalisation, unifying) but also by differentiation (top–down, specialising, discriminating). So, for example, the concept *Squares* can be defined as the intersection of *Rectangles* and *Rhombuses*. The point to start with in hierarchical

clustering actually is “somewhere in the middle”:

Example 4.2 The concept *Bird* (*Hammer*, *Car*) can be learned quicker than *Animal* (*Tool*, *Mobile*) or *Robin* (*Sledge hammer*, *van*). This is because the more general concepts (e.g. *Insect*) subsume subconcepts that may differ significantly (like *Flies* and *Beetles*) and because the rather detailed concepts are quite homogeneous (*Ants*) but not always clearly distinguishable from brother concepts (*Bees* and *Wasps*).[1] ●

The key to the right level of abstraction is *homogeneity*. Homogeneity within a class and separation between classes are expressed in terms of *intra–* and *inter–class similarities* respectively. Since we need probabilities to express the similarity measures, we observe:

Theorem 4.1 Let $g : U \to \mathbf{k}$ such that $g(x) = i :\Longleftrightarrow x \in c_i$ where $c_i \in \mathfrak{c} = U/g$. Let $\mathbf{F} = \{f_0, f_1, \ldots, f_{n-1}\}$ and $\vec{x} = \langle f_0(x), f_1(x), \ldots, f_{n-1}(x)\rangle$. Then, every $f \in \mathbf{F}$ is a random variable that assigns a value $v \in \mathrm{cod}(f)$ to x. We write

$$\begin{aligned}\Pr[F = v] &:= \phi(\{x \in U : f(x) = v\}) \\ &= \frac{|\{x \in U : f(x) = v\}|}{|U|}.\end{aligned} \tag{4.6}$$

Then, the probability that an object has a certain property given it belongs to a certain class is

$$\begin{aligned}\Pr[F = v \mid C] &= \Pr[F = v \wedge C]/\Pr[C] \qquad (4.7)\\ &= \frac{\phi(\{x \in U : f(x) = v\} \cap \{x \in U : \chi(c)(x) = \mathbf{1}\})}{|c|/|\mathfrak{U}|} \\ &= \frac{|\{x \in U : f(x) = v\} \cap c|}{|c|} \qquad (4.8)\end{aligned}$$

for $f \in \mathbf{F}$, $c \in \mathfrak{c}$ and $v \in \mathrm{cod}(f_i)$.

Exercise 4.7 (♦) Determine the probability that x belongs to a class c given that for some $f \in \mathbf{F}$, $f(x) = v$.

Intra/Inter–Class Similarity

Definition 4.7 — Intra/Inter–Class Similarity.
The *intra–class similarity* of a class $c \in \mathfrak{c}$ is the probability that object representations are similar given the information they belong to the same class:

$$\mathrm{sim}(c) := \frac{1}{|\mathbf{F}|}\sum_{f \in \mathbf{F}}\left(\frac{1}{|\mathrm{cod}(f)|}\sum_{v \in \mathrm{cod}(f)} \Pr[f(x) = v|C]\right) \tag{4.9}$$

The *inter–class dis-similarity* is the reverse: It is the probability that an object

[1] They *are* clearly distinguishable for experts but quite many people cannot tell the difference between a wasp, a bee, a bumble bee or harmless hover flies.—In fact, wasps are closer related to ants than to bees.

belongs to a certain class given it has a certain probability:

$$disim(c) \quad := \quad \frac{1}{|\mathbf{F}|}\sum_{f\in\mathbf{F}}\left(\frac{1}{|\mathrm{cod}(f)|}\sum_{v\in\mathrm{cod}(f)}\Pr[C|f(x)=v]\right) \tag{4.10}$$

●

Exercise 4.8 (♦) Explain, why $\Pr[C|F = v]$ expresses "dissimilarity" rather than $\Pr[\neg C|F \neq v]$!

So, the more homogeneous a class, the higher its intra–class similarities: Given they belong to the same class, the have many similar properties. The more a class can be discriminated from another one, the higher the probability a property determines the class membership. There are many different ways to define cluster homogeneity or heterogeneity and we presented just one based on our metaphor of distances in representation space. The general idea behind these two measures is illustrated in figure 4.5. The double-headed arrows (cluster radiuses) shall illustrate an intra-cluster similarity, while the lines connecting all the centroids represent the inter-class dissimilarities. A clustering appears to be more adequate, the higher the intra-class similarity and inter-class dissimilarities which means we have to find an ideal trade-off between both class similarities: The clusters shall be as homogeneous as possible and yet discriminating enough. Therefore, our task is to find $f \in \mathbf{F}$ which induces a partition that has maximal intra-class similarities and inter-class dissimilarities. In order to find such a good partition, we need a measure to describe the utility of such a partition; i.e. the feature f.

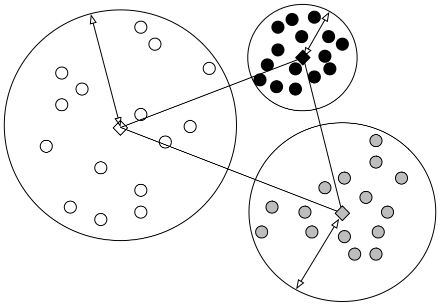


Figure 4.5: Intra– and Interclass Similarities

A straightforward way to define such a measure is to multiply the product of similarity and dissimilarities as induced by f with the prior probability of the according feature-value combinations:

Definition 4.8 — Partition Utility.

Partition Utility

The *utility* of a partition is described by

$$utility(\mathfrak{c}) \quad := \quad \sum_{c\in\mathfrak{c}} \frac{|c|}{|U|} \sum_{f\in\mathbf{F}} \sum_{v\in\mathrm{cod}(f)} \Pr[F = v|C] \cdot \Pr[C|F = v] \tag{4.11}$$

This is equivalent to evaluating a feature (hypothesis) h's utility in partitioning the universe ($U/h = \mathfrak{c}$). ●

To conclude the chapter on clustering methods, we examine an approach to incremental hierarchical clustering:

- *incrementally* means that we successively develop and refine a partition over a set of data with.
- *hierarchically* means that we do not create flat clusters but that we want to merge or split clusters if the data encountered suggests such operations of generalisation or specialisation.

In order to define an appropriate procedure, four different operators are required. These are in detail *cluster refinement*, *cluster introduction*, *cluster join* and *cluster split*. A refinement is required each time we encounter an object that shall belong to an already existing cluster. This can change the cluster form (just recall the k–means principle). If an object does not belong to any of the already existing clusters is taken to form a new cluster. It is easy to imagine that such an action would be chosen if the resulting intra–class similarities would decrease once the object is forced into one of the old clusters. Sometimes, objects bridge a gap between between two disjoint clusters. This means that the point belongs to both clusters to the same degree—and the result is, that the point is pretty well located in the centre of the joined clusters. Therefore it seams reasonable to induce a new cluster that subsumes this object and all the clusters that are similar to this object for more or less the same degree. Finally, whenever a cluster grows too large (in terms of the number of its members or a poor intra-class similarity in relation to other clusters), it seems a good idea to divide the cluster into more special subclasses. Another criterion is that whenever an object "disturbs" the balance of a cluster with already pretty low intra-class similarity, there mus exist several subgroups. Putting these operations together, one yields an abstract description of a hierarchical clustering algorithm that is both *agglomerative* and *divisive*; which means it builds a cluster hierarchy bottom-up and top-down for generalisation and specialisation, respectively.

Example 4.3 Consider our domain of geometric shapes as shown in figure 4.3. Recall at this point that in the picture "similarity" corresponds to "distance"—even though the vector representation of the objects can be much more complex: For the set of all these objects x which have at most 4 corners, we define:

$$\vec{x} \quad := \quad \langle x_0, y_0, \ldots, x_3, y_3, x_{c_1}, y_{c_1}, x_{c_2}, y_{c_2}, r_1, r_2 \rangle$$

Two arguments x_i, y_i define the x- and y-coordinates of the i-th corner, the two last pairs define a the center and r_1, r_2 the radiuses. Then a triangle is defined by a vector with only the first six arguments instantiated and the rest filled with zeros.
Suppose we begin incremental bi-directional hierarchical with known classes *Triangles*, *Tetragons* and *Ellipses*. Note that each of these classes appear on a different level in our target concept hierarchy. A closer look at each of the three sets shows that the cluster of *Tetragons* divides into two sub-clusters. At the same time, all the tetragons are closer to the triangles than to ellipses. Therefore, we would join the clusters to form a new one. The resulting clusters are *Rhomboids*, *Rectangles*, and *Polygons*.
A further analysis of *Ellipses* shows that there are two kinds: one for which the two center coordinates and radii are the same and one for which they are different. We therefore identify a sub-cluster *Circles*. Similarly, we find a subclass of *Rhomboids*; those whose four sides have the same length—*Rhombuses*.
Finally, there is huge difference between the three different clusters we have so far: *Ellipses* have 6 zero entries at the beginning, and *Triangles* and *Tetragons* have four zeroes at the end. Accordingly, we join the latter two and obtain *Polygons*. ●

We do not need to define an algorithm that implements this behaviour here: Top-Down construction (i.e. divisive clustering) will be discussed in the next chapter—and agglomerative methods fall into the category of generalisation operators which will be discussed in another chapter, too.

4.5 Relational Clustering

This chapter was concerned with a lot of distance measures, similarities, and vectors in high dimensional spaces. One might ask what this kind of clustering actually has to do with *relational knowledge discovery.* The answer is very simple.
Every object in space is represented by a vector. This vector comprises of arguments each of which corresponds to a function. Recall that *sim* and *disim* were defined by way of random variables. And random variables are functions—and each object representation can be formulated as the same vector where each component holds the corresponding value of a random variable.
Either way, representation space is a feature space. And this again means that all the object in this approach can be described by an information system. Now recall that every single feature of an information system induces an equivalence relation—and any clustering of a set of objects is a *classification.* This means that hierarchical clustering is simply repeated classifier learning. In other words: Hierarchical clustering means to find a family of equivalence relations

$$R_0 \subseteq R_1 \subseteq \cdots \subseteq R_{k-1} \tag{4.12}$$

such that $\bigcap_{i \in \mathbf{k}} R_i$ induces a partition with classes whose elements are most

similar. The less relations we choose and the coarser they are, the less clusters we can describe—and the more general our classification.
Clustering is nothing else than unsupervised classification—it's just that we assume (or define) some distance measure in order to describe similarity and to help ourselves get over the missing teacher signal.

Chapter 5

Information Gain

> Describing objects by features is a very common thing to do. Similarly, many decision support systems use a tree–like representation of cases, where every branch in the tree corresponds to a feature and its observed value. But which features can be used to model a certain concept? What would be the shortest and most meaningful rule with which we can describe a distinct set of objects using our features?

In the previous section we have seen how similarity measures can be used to group objects into (hopefully) meaningful clusters. Given an information system $\mathfrak{I}$, we now want to describe a feature's utility with respect to an object's classification $t(x)$. Relationally speaking, we need to recursively apply those features $f_i \in \mathbf{F}$ which generate a partition on U that is similar to U/R_t to learn a compressing classifier this way. It appears a good idea to start with a feature that appears to be most "similar" to t. A feature being quite similar to the target function can be assumed to carry relevant information with respect to t. And this leads us to the information theoretic notion of entropy.

> **Information Gain Driven Classifier Learning**
> While Clustering tries to find hierarchies of groups of objects, so-called *decision trees* represent a hierarchy of feature induced partitions. Unlike (unsupervised) similarity measures in clustering, one uses a target-specific information measure called *entropy.*

People often try to explain Shannon and Weaver's information theoretic measure of entropy by the laws of entropy in thermodynamics. In fact, this approach is much more demonstrative than the original works of Shannon and Weaver. However, both measures were develepoded independently from each other and with completely different motivation and background.

5.1 Entropy

In 1865, Rudolf Clausius introduced the notion of entropy into physics by describing a closed system of constant temperature T and the result of applying energy (i.e. heat) onto it (ΔQ). A common sense picture of this situation is that all particles in the system now move more vigorously; i.e. it becomes harder to tell “where” they are. The amount of applied energy becomes visible in the particle’s movement: Moving faster creates heat - and more frequent and more violent bumping into the walls creates pressure. The entropy increase (ΔS) is proportional to the amount of energy invested:

$$\Delta S = \frac{\Delta Q}{T}$$

Then, in 1877, Boltzmann stated that the entropy S of a system can be described by the number Ω of possible states consistent with its thermodynamic properties:

$$S = k \cdot \ln \Omega$$

where k is known as the famous Boltzmann constant.
In 1948, Shannon and Weaver described the entropy of a system as the probability-weighed sum of *bits* of information needed to describe the system state:

$$H(S) = -\sum_{i=1}^{\omega} p_i \log_2 p_i \tag{5.1}$$

Obviously, entropy and information are related somehow, and they are used to describe a ratio of measures that shows an additive behavior for exponential growth (hence the log).

Entropy and Information
The entropy $H(s)$ of a set s is a measure of the complexity of a system: Given all possible states of the system and their respective probabilities, the entropy describes the average length of the shortest description specifying an arbitrary system state.

In the 1940s Shannon and Weaver were working on the question of how much channel capacity one needs to securely transmit a message with a certain amount of noise involved. This gave rise to two questions:

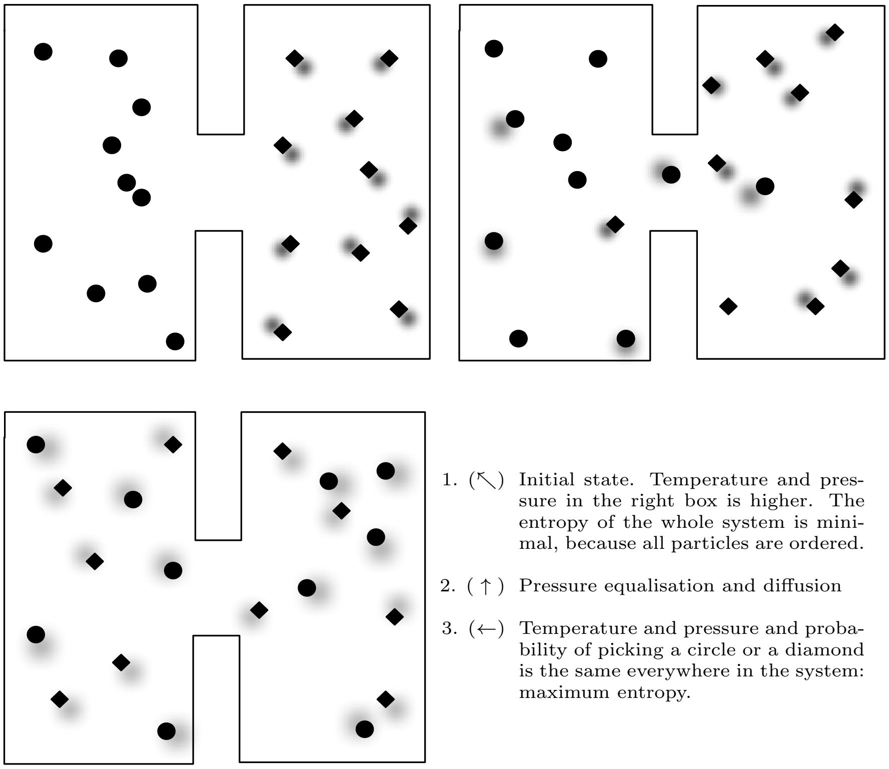
1. How does one measure the *amount of information*?
2. How does one measure the *capacity* of a communication channel?

First of all, it is very important not to confuse the two different terms information and meaning: Meaning usually denotes the semantic content of a message, while information is rather a measure of the complexity of the message source:

> [...] *information* must not be confused with meaning. [...] Information [...] relates not so much to what you *do* say, as to what you *could* say. That is, information is a measure of one’s freedom of choice when one selects a message.

[Shannon and Weaver, 1949]

Even a meaningful message loses information when repeated over and over again. Now that information is compared to degrees of freedom, it is clear why information theoretic entropy can be related to the definition of entropy as it is known in thermodynamics: The number of possible system states increases with the degree of freedom of each particle in it. To give you a very brief but demonstrative example, see figure 5.1.



1. (↖) Initial state. Temperature and pressure in the right box is higher. The entropy of the whole system is minimal, because all particles are ordered.
2. (↑) Pressure equalisation and diffusion
3. (←) Temperature and pressure and probability of picking a circle or a diamond is the same everywhere in the system: maximum entropy.

Figure 5.1: Entropy Change of a Closed System

But before we start defining a measure of information, it is a good idea to understand which properties we will require from this measure. These are:

- First, the more "usual" an event or message, the less is its information content. Or, the other way round: the less the probability we observe some event, the higher its information.

- Second, the information of a joint observation of two independent messages should be the sum of the information of the individual messages.

Let us reconsider our example from the introduction:

Example 5.1 When flipping a fair coin the probability for heads is the same as for tail: it is fifty percent in both cases. We have a maximum degree of freedom, a maximum degree of uncertainty and a maximum of information. A biased coin predestines the outcome of a throw: it decreases the degree of freedom, it introduces certainty and loses information.
Cheating makes the game more predictable: the probability of throwing heads is much lower than tails. Accordingly, we already *expect* tails and the number of throws where our expectation is not met and we make an erroneous prediction is rather small.
Playing a fair game increases the amount of information in a throw: Since all throws are conditionally independent it is clear that for each throw we have a fifty-fifty chance of either outcome. So the information in a message string generated by a source that is playing a fair game is much higher than that of a biased sender (a simple but true observation we make in our everyday lives as well). ●

In the last example we have seen how the probability of a signal in a message determines the information content; we also saw that information of several events are summed up. We now examine how likeliness of a sequence of events in relation to the number of possible events changes. This time, it is easier to examine the issue in the light of thermodynamics: Let there be a system with ω entities each of which can take n states. Then, the system can take one out of $\Omega = n^\omega$ different states.

Example 5.2 Consider

	Number of	
Entities n	states ω	System states n^ω
7	2	$2^7 = 128$
2	10	$10^2 = 100$

By adding *one* entity we can increase the number of possible system states by the factor $n = n^1$:

Entities n	states ω	System states n^ω
7+1 = 8	2	$2^{7+1} = 2^8 = 256$
2+1 = 3	10	$10^{2+1} = 10^3 = 1,000$

and adding m entitites results in

Entities n	states ω	System states n^ω
7+m	2	$2^{7+m} = 2^7 \cdot 2^m = 128 \cdot 2^m$
2+m	10	$10^{2+m} = 10^2 \cdot 10^m = 100 \cdot 10^m$

So the number of system states increases exponentially in the number of entities.

Exercise 5.1 ◊ For a fixed n, what happens if we add different numbers σ_1, σ_2 of states to ω? — Relate the growth rates of adding entitites and states!

So if we increase the number ω of entities (atoms, symbols) by the *factor* of a, then the possible number of system states (or messages) increases exponentially in a. Since $\Omega = n^{\omega}$, we have

$$\Omega' = n^{a \cdot \omega} = n^{(\omega)^a} = \Omega^a \tag{5.2}$$

with $\omega' = a \cdot \omega$. If we take the number of entities as the length of a message and the number of states as the number of different symbols, Ω is the number of possible messages. If we *add* to the length of the message, we have a factor in information content — and this is exactly the origin of the logarithm in Shannon and Weaver's measure of information.

Now that we have gained a pretty detailed idea of how entropy as a property of physical entities works, it is about time to consider information systems again. Here, we do not deal with particles, but with events that are described by variables or with objects that are described in information systems. First, we consider events that are described by several discrete[1] random variables: Let there be a set $\mathcal{F} = \{F_0, \ldots, F_{n-1}\}$ of random variables where F_i corresponds to elementary events represented by $f_i \in \mathbf{F}$. Then, all F_i can take values from $V_i = \mathrm{cod}(f)_i$. For every elementary event there is a measure μ_i describing the probabilities that for some $x \in U$, $f_i(x) = v$.[2] Events are described by sets of elementary events; in our case a vector of all values F_i. There is also a measure μ^n describing the probability of events:

$$\begin{aligned} & \mu^n(\{x \in U : f_i(x) = v_i) \\ = \; & \Pr[F_0 = v_0 \wedge F_1 = v_1 \wedge \cdots \wedge F_{n-1} = v_{n-1}] \\ = \; & p_x \end{aligned}$$

Usually, the probability of the co-occurence of two mutually independet events with two different probabilities results in the product of the probabilities. It is *very* important to understand that the assumption of mutual independence is a fundamental bias. Even worse, the sequential ordering of symbols as they appear in the sequence of a message is *not* independent! Information theoretic entropy makes an assumption that is true only in the context of thermodynamics, but *not* in the context of meaningful sequences.[3] Nevertheless we need to live with certain biases if we want our algorithms to perform sufficiently efficent.

[1] The discussion of continuous signals is beyond the scope of this lecture. The interested reader should consult the original article [?]; a modern book on information theory, coding and cryptography, [?], or the more recent textbooks on information, probability and statistics in knowledge and knowledge discovery, [?] and [?].

[2] We omit indices here to avoid overly extensive subscripting (v_{j_i}). It is assumed that for $f_i(x) = v_j$ it always holds that $v_j \in \mathrm{cod}(f)_i$.

[3] This has been shown by an impressive counterproof given by the work of Bletchely park in breaking the enigma of Shark.

In order to make the probabilities to behave additively when occuring together we now apply a simple trick: Instead of multiplying probabilities, we *add* the logarithms, and so get exactly what we were looking for:

entropy(x), Shannon's Entropy Measure

Definition 5.1 — entropy(x)**, Shannon's Entropy Measure**.
Shannon's measure of information content describes the information content of some observation that $f_i(x) = v$:

$$\mathrm{entropy}(f_i(x) = v) := \log_2 \frac{1}{\Pr[F_i = v]} \tag{5.3}$$

It is the negative logarithm of the probability that this observation is made. We also write $\mathrm{entropy}(v) = \log_2 \frac{1}{p_v}$ when clear from context. ●

But what about *sets*? Whenever we talk about several messages we always should weight each one by its own probability of occurence. This leads to a preliminary definition of information theoretic entropy as a probability weighed sum of information content: The entropy of a source ("sender") is the expected information content of a message being sent by this system:

$$\mathrm{entropy}(s) \quad = \quad -\sum_{i=1}^{\omega} p_i \log_2 p_i \tag{5.4}$$

So the entropy of a set s of possible messages is the probability weighted sum (i.e. "expected" or average value) over the information content of each symbol. The base 2 of the logarithm originates from the assumption that we deal with "particles" that can take only *two* different states. It can be easily computed using the following transformation:

$$\mathrm{entropy}(s) \quad = \quad -\sum_{i=1}^{\omega} p_i \log_2 p_i = -\sum_{i=1}^{\omega} p_i \frac{\ln p_i}{\ln 2}$$

In other words, *information* is a dimensionless measure just as entropy, but we agree to write "x bit" instead of "$x\frac{1}{\ln 2}$". Sadly, many arguments involving entropies are often written down in a rather sloppy notational way. But after a brief digression in the next section, we will give formally satisfying definitions and examples.

Exercise 5.2 (◊♦) Explain why the three actions described in figure 5.2 increase the entropy of a system using (a) Clausius's notion of entropy and (b) Boltzman's formula.

Exercise 5.3 (♦♦) Describe three actions on information systems that are equivalent to those shown in figure 5.2 and explain the increase of entropy using Shannon's measure of information.

5.2 Information and Information Gain

Relational knowledge discovery is the same all the time: We want to create a method with which we can construct aets of relations that we can use to

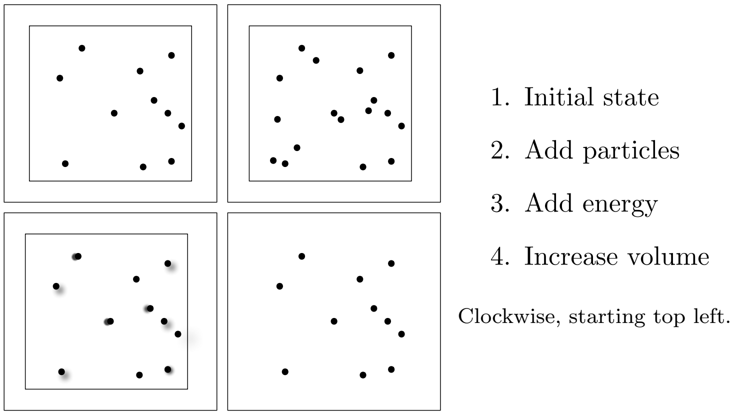


Figure 5.2: Three Ways to Increase a System's Entropy

describe a concept. The problem is just to guide the search in order to speed up the learning process:

Entropy and Information Gain
When learning classifiers, we will always refer to entropies in relation to a target classification. Given a set of observations with different target labels assigned, the entropy with respect to the target label describes the complexity of the learning problem. If we manage to partition the set into disjoint subset that have a lower entopy, then we have *gained* some information: The cuts performed during partitioning are correlated to the boundaries of the target classes. The idea behind information gain methods is to recursively partition the set using features which produce maximal information gain.

5.2.1 Entropy

We start right off with a definition:

Definition 5.2 — Entropy of a set s.

Entropy of a set s

We define the *(f–relative–) Entropy* of a set to be

$$\text{entropy}_f(s) = -\sum_{v \in \text{cod}(f)} \frac{|\{x \in s : f(x) = v\}|}{|s|} \log_2 \frac{|\{x \in s : f(x) = v\}|}{|s|} \quad (5.5)$$

When $f = t$, we also write $\text{entropy}_t(s) = \text{entropy}(s)$. ●

Since the true distribution is unknown, we use the relative frequency of observations to approximate the probabilities of symbols.[4] Furthermore, information

[4] Knowing μ means to have knowledge about what the sender is going to say. Such knowledge is used in source-coding compression methods.

content is always determined in relation to a certain property f of objects (which are, again, equivalence classes).

Example 5.3 Imagine the following set of six different symbols each in a white or black version:

$$\{\circ, \Box, \Diamond, \rhd, \lhd, \triangle, \bullet, \blacksquare, \blacklozenge, \blacktriangleright, \blacktriangleleft, \blacktriangle\}$$

Then, the information content of message that reads $\Diamond \bullet \Diamond \Box \Diamond \bullet$ is determined as follows:[5]

1. the set of symbols used is $\{\Diamond, \bullet, \Box\}$
2. the message length is 6, so the approximate probability of the three symbols are $p_\Diamond = \frac{3}{6} = \frac{1}{2}$, $p_\bullet = \frac{2}{6} = \frac{1}{3}$, and $p_\Box = \frac{1}{6}$
3. We assume this approximation of probabilities to be sufficiently precise and compute the information content as follows:

$$\begin{aligned} \text{entropy}_{\in}(\{\Diamond, \bullet, \Diamond, \Box, \Diamond, \bullet\}) &= -\sum_{x \in \{\Diamond, \bullet, \Box\}} p_x \log_2 p_x \\ &= -\frac{1}{2}\log_2\frac{1}{2} - \frac{1}{3}\log_2\frac{1}{3} - \frac{1}{6}\log_2\frac{1}{6} \end{aligned}$$

But actually, we could also determine the information of such a message with respect to colour or shape of each symbol:

Example 5.4 We now want to determine the entropy of the message set with respect to each object's colour which we assume to be described by a feature $c \in \mathbf{F}$:

$$\begin{aligned} & \text{entropy}_c(\{\Diamond, \bullet, \Diamond, \Box, \Diamond, \bullet\}) \\ = & -\sum_{c(x) \in \{\text{white},\text{black}\}} p_{c(x)} \log_2 p_{c(x)} \\ = & -\frac{|\{\Diamond, \Diamond, \Box, \Diamond\}|}{6}\log_2\frac{2}{3} - \frac{|\{\bullet, \bullet\}|}{6}\log_2\frac{1}{3} \\ \approx & \ 0.92 \end{aligned}$$

Just to be absolutely sure that information theoretic entropy of a set s is always a measure *with respect* to some property of objects, we give a last example:

[5] We assume all the symbols to be subscripted by a running index; i.e. $\{\Diamond_0, \bullet_1, \Diamond_2, \Box_3, \Diamond_4, \bullet_5\}$. Multiple occurences of a symbol in a set actually means multiple observations of the symbol.

Example 5.5 Let us consider the entire set s and determine its entropy with respect to the number of vertices an object has.

$$\begin{aligned}
& \text{entropy}_v(\{\circ, \square, \lozenge, \triangleright, \triangleleft, \triangle, \bullet, \blacksquare, \blacklozenge, \blacktriangleright, \blacktriangleleft, \blacktriangle\}) \\
= \; & \text{entropy}_v(\{0, 4, 4, 3, 3, 3, 0, 4, 4, 3, 3, 3\}) \\
= \; & - \sum_{v(x) \in \{0,3,4\}} p_{v(x)} \log_2 p_{v(x)} \\
= \; & -\frac{1}{6} \log_2 \frac{1}{6} - \frac{1}{3} \log_2 \frac{1}{3} - \frac{1}{2} \log_2 \frac{1}{2} \\
\approx \; & 0.597
\end{aligned}$$

Exercise 5.4 Here are a few exercises to get a feel for entropies:

♢ Give an example of an arbitrary set of the symbols with maximum entropy with respect to colour.

♢♦ Determine $\text{entropy}_f(\{\blacktriangleleft, \heartsuit, \triangleleft, \triangleright, \blacklozenge, \triangleleft, \blacktriangleright, \blacktriangleleft, \triangleleft, \lozenge, \triangle, \blacktriangle\})$ for three different f!

♢♦ Determine $\text{entropy}_f(s)$ for all features $f \in \mathbf{F}$ in figure 5.3(left).

♢♦♦ Determine $\text{entropy}_f(s)$ for all features $f \in \mathbf{F}$ in figure 5.3(right).

s	t	f	g	t'	id	c
1	1	1	1	1	1	1
2	1	1	1	1	2	1
3	1	2	1	1	3	1
4	0	2	1	2	4	1
5	0	3	2	2	5	1
6	0	3	2	2	6	1

s	f_0	f_1	f_2	f_3	t
0	1	●	♡	c	0
1	0	●	♠	b	/
2	2	●	♣	b	1
3	1	●	♣	c	/
4	1	●	♡	a	1
5	2	●	♣	b	1
6	2	●	♠	b	/
7	0	●	♠	a	1

Figure 5.3: Two Information Systems

5.2.2 Information

Recall that our idea was to select some $f \in \mathbf{F}$ which is *most informative* with respect to t. The set of our objects or observations has a certain entropy (measured with respect to an arbitrary property). Usually, we define the entropy with respect to the target classification t. This is why we agreed to drop the index in this case: $\text{entropy}_t(s) = \text{entropy}(s)$.

But the big question is: what is the *information in a feature*? It is, so to say, a measure of its entropy in relation to the entropy of s with respect to some property. In other words, there are two features involved here.

And this leads us to the definition of *feature entropy* or *feature information*:

$\text{entropy}_f(g, s)$, Feature Entropy (Information)

Definition 5.3 — $\text{entropy}_f(g, s)$**, Feature Entropy (Information).**
The information of a feature $g \in \mathbf{F}$ on a set s with respect to a feature $f \in \mathbf{F}$ is

$$\text{entropy}_f(g, s) = \sum_{v \in \text{cod}(g)} \frac{|\{x \in s : g(x) = v\}|}{|s|} \text{entropy}_f(\{x \in s : g(x) = v\}) \tag{5.6}$$

which is the (relative class size weighed)–entropy of g on the quotient s/R_f. As usual, we drop the index if $f = t$ and write $\text{entropy}(g, s) := \text{entropy}_t(g, s)$. ●

Exercise 5.5 ♦ Determine the value of $\text{entropy}_f(f, s)$!

Exercise 5.6 ♦ Determine the information of all features relative to t in figure 5.3.

Now that we want to learn how to approximate t it appears to be a good idea to partition s into classes induced by a feature with most information. This would reduce each class entropy and, therefore, create a partition that is closer to t than the partition induced by any other feature.

Reducing the entropy of a set means a loss of predictive uncertainty—that is, a loss of indeterminacy, a reduced number of degrees of freedom or, simply, *information gain*. Accordingly, we define:

Information Gain

Definition 5.4 — **Information Gain.**
We define the *information gain* obtained by a feature g on s with respect to f as:

$$\text{gain}_f(g, s) := \text{entropy}_f(s) - \text{entropy}_f(g, s) \tag{5.7}$$

The gain is the difference of the current entropy on s minus the information we gain by application of knowledge g. Again, we drop the index for $f = t$; i.e. we abbreviate $\text{gain}_t(g, s) := \text{gain}(g, s)$. ●

The larger $\text{gain}_f(g, s)$, the more entropy is lost, the greater is the information content of g and the larger is the information gain by using g to partition s.

Exercise 5.7 Let's practise some gain computations:

- ♦ Compute $\text{gain}_t(f, s)$, $\text{gain}_t(g, s)$, $\text{gain}_t(id, s)$ from the left part of figure 5.3.
- ◊♦ Compute $\text{gain}_t(f_i, s)$ with $i \in \mathbf{3}$, from the right information system in figure 5.3.
- ♦♦ Compute $\text{gain}_{f_i}(f_{i+1}, s)$ for $i \in \mathbf{2}$ from the right information system in figure 5.3.

Exercise 5.8 (♦) Prove or disprove: $\text{gain}_g(f, s) = \text{gain}_f(g, s)$.

Exercise 5.9 (♦♦) Write a small program that for an input string s determines $H(s)$.

5.3 Induction of Decision Trees

Decision Trees
A decision tree is a classifier representation which allows to classify an object with increasing accuracy by asking a sequence of questions about the values thos object takes under a certain feature.
Learning such a classifier means to build such a tree in a way that its leaf nodes represent sets of objects that are more or less contained in (subsets) of equivalence classes induced by the target feature. We can stop building a tree if at a current node all objects covered fall into the same target class—which means that its entropy is zero. If we can't reduce the entropy any further or if we run out of features, then we have to stop growing the tree, too.

As we have already mentioned and as we shall see in detail later, any feature f induces an equivalence relation R_f on s. Also, the binary target function $t(x) = \chi(c)x$ induces an equivalence relation R_t such that $s/R_t = \{s^{\mathbf{1}}, s^{\mathbf{0}}\}$. The entropies in $s^{\mathbf{1}}$ and $s^{\mathbf{0}}$ are 0. In order to approximate t we can also try to approximate s/R_t. We do so by hierarchically partitioning s using $R_{f_0}, R_{f_1}, \ldots$ until t-entropies in the resulting classes are 0. This is equivalent to building a tree with s as root node and all elements of a quotient induced by f as the successor nodes of the node f until the leaves are subsets of either $s^{\mathbf{1}}$ or $s^{\mathbf{0}}$. Such a tree is called a *decision tree*. But how can one build such a tree efficiently and how can one keep a tree as small as possible so as to guarantee a maximum compression? Not surprisingly, we will use the information theoretic entropy measure in order to guide our search.

5.3.1 Hunt's Classifier Trees and Quinlan's ID3

Decision trees are a widely accepted method for classifying objects. Most decision support systems make use of "flow-charts" in order to quickly identify a certain class (for example in medicine where a structured sequence of tests for symptoms quickly leads to a diagnosis).
Accordingly, the induction of such trees is still one of the most popular techniques in knowledge discovery. Because of their relative high efficiency and wide acceptance they are a standard method provided by nearly every Data Mining tool.
The rise of decision tree induction started with a system called Id3, [?]. Successors like C.45 and See5 (or C5.0) provided additional functionality such as dealing with continuous features values, or pruning and boosting [?, ?, ?, ?, ?]. The idea of hierarchical clustering was not new—it was the entropy measure of information that turned out to be the real knack. Cobweb, [?], and Unimem, [?], were about the first systems for clustering objects without a teaching signal. ClassIt, [?], added the idea of incremental concept formation while Cluster/2, [?], and its successor Cluster/S, [?], are non–incremental variants. We already learned about them in the previous chapter on clustering.
Then, Aq, [?, ?], and Cn2, [?], finally made use of a teaching signal to allow for supervised concept formation. But the idea of decision tree induction actually dates back to 1966, when [?] introduced *classifier trees* and developed he system

CLS. The pseudo-code of CLS is shown in figure 5.4. The idea behind this algorithm is to *divide and conquer* until all examples can be classified (see the recursive call on all induced classes in line 10). But the problem with it is that it involves a non-deterministic choice in line 9. Different features usually differ in their feature entropy relative to t so it seems a good idea to choose the feature with maximum information gain. Therefore, the idea behind TDIDT is simply to take CLS and add an information gain heuristic:

```
01  proc class (s)
02  {
03      IF (∀x, y ∈ s : t(x) = t(y)) THEN
04      {   return (s) }
05      ELSE
06      {
07          IF (F = ∅) THEN return (⊥)
08          ELSE
09          {   f := choose(F); F := F − {f};
10              return (⟨ class({x ∈ s : f(x) = v_0}),
                          . . . ,
                          class({x ∈ s : f(x) = v_|cod(f)|})⟩);
11          } FI
12      } FI
13  }
```

Figure 5.4: Hunt's Algorithm for Finding Classifier Trees

1. The root of the decision tree subsumes all entites $x \in s$.
   We choose the $f \in \mathbf{F}$ from which we expect a maximum information gain and create successor nodes for each $f(x) \in \overline{f}^{\urcorner}$.

2. Now check the nodes left to right:

   (a) If all entities subsumed by the current node either belong to s^+ or to s^-, label the node **1** or **0** respectively.

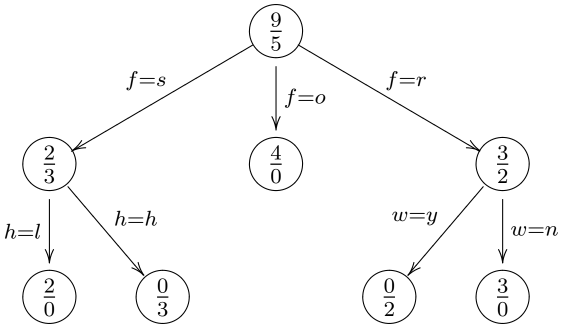
   (b) Otherwise, recursively choose the next $f \in \mathbf{F}$ which does not occur on the path from the current node back to the root and create successor nodes for each $f(x) \in \overline{f}^{\urcorner}$. If there is no attribute left, stop and report "Unsuccessful attempt".

Exercise 5.10 (♦) In the algorithm above, only features that have not been used yet are taken into account in step 2.(b). This sure makes the algorithm more efficient, but will the result change if we consider all features? Why?

Figure 5.5 shows the most famous decision tree: A decision tree is built to describe a large set of observations whether one should go for playing tennis depending on the current weather conditions. The data used for inducing this tree is are weather reports for the past two weeks:

Day	Forecast	Temperature	Humidity	Wind	t
1	sunny	high	high	no	**0**
2	sunny	high	high	yes	**0**
3	overcast	high	high	no	**1**
4	rainy	med	high	no	**1**
5	rainy	low	high	no	**1**
6	rainy	low	low	yes	**0**
7	overcast	low	low	yes	**1**
8	sunny	med	high	no	**0**
9	sunny	low	low	no	**1**
10	rainy	med	high	no	**1**
11	sunny	med	low	yes	**1**
12	overcast	med	high	yes	**1**
13	overcast	high	high	no	**1**
14	rainy	med	high	yes	**0**

As one can easily see, $s^+ = \{3, 4, 5, 7, 9, 10, 12, 12, 13\}$ and $s^- = \{1, 2, 6, 8, 14\}$ which makes 9 instances for $t(x) = \mathbf{1}$ and 5 instances for $t(x) = \mathbf{0}$. In figure 5.5, every node is labelled with a tuple $\left\langle \frac{p}{n} \right\rangle$ with p being the number of positive and n being the negative instances in this node.



f is the *forecast* (sunny, rainy or overcast), h is the *humidity* (low or high) and w is the *wind* (yes or no).

Figure 5.5: The Famous Golf Player's Example

All the sum of all positive numbers and all negative numbers in successor nodes equals the numbers in their parent node. In leaf nodes, either $p = 0$ or $n = 0$,

and the sum of all numbers in all leaf nodes is 14. So every node N in a decision tree subsumes a certain subset of elements.
Let us now take a more formal look at decision trees. We define:

Decision Tree

Definition 5.5 — Decision Tree.
A *decision tree* consists of *decision nodes* and *leaf nodes* (*class node*). A *decision node* N_f carries the name of a feature and edges to a set of successor nodes; one for each possible value of f:

$$N_f \quad := \quad \langle f, \{N_i : i \in |\mathrm{cod}(f)|\}\rangle \tag{5.8}$$

We call a node N a *leaf node*, if it has no successors. It represents the set of all objects which take a value y as defined by the edge from its parent node under the feature defined by its parent node:

$$N_y \quad := \quad \langle y, s\rangle \tag{5.9}$$

where $s \subseteq U$ (see definition of *coverage*). Note that this definition is entirely *descriptive* and that it does not give a recipe to build a decision tree. ●

For a better reading, we denote leaf nodes by simple capital letters like N and decision nodes N_f with indices denoting the feature f that determines its sucessor nodes. For a decision node N_f we refer to its successor nodes by N_i where $i \in |\mathrm{cod}(f)|$ or N_y with $y \in \mathrm{cod}(f)$.
So whenever a decision tree has more than just one leaf node, it must have at least one decision node. Then, the topmost decision node is the decision tree's root node. The tree for the golf players' example in figure 5.5 is formally represented as:

$$\left\langle f, \left\{ \begin{array}{l} \left\langle h, \left\{ \begin{array}{l} \langle c_{st}, \{x \in U : f(c) = s \wedge h(x) = t\}\rangle \\ \langle c_{sh}, \{x \in U : f(c) = s \wedge h(x) = h\}\rangle \end{array} \right\} \right\rangle, \\ \langle c_o, \{x \in U : f(x) = o\}\rangle, \\ \left\langle w, \left\{ \begin{array}{l} \langle c_{ry}, \{x \in U : f(c) = r \wedge w(x) = y\}\rangle \\ \langle c_{rn}, \{x \in U : f(c) = r \wedge w(x) = n\}\rangle \end{array} \right\} \right\rangle \end{array} \right\} \right\rangle \tag{5.10}$$

Obviously, a node in a decision tree "contains" a set of objects $x \in U$. The case is clear for leaf nodes—they do not possess successor nodes but only a subset $s \subseteq U$. But none of its elements occurs in any other set of any other leaf. As far as decision nodes are concerned, we simply define the set of objects in them as the set of all objects in all the leaf nodes under this decision node. Let us turn this simple idea into a satisfying definition:

Node Coverage

Definition 5.6 — Node Coverage.
A decision node N_f *covers* or *subsumes* all elements covered by *all* of its successor nodes: Let N_f be the root node of a tree. Then,

$$\begin{aligned} \mathrm{cvr}(N_f) \quad &= \quad \mathrm{cvr}(\langle f, \{N_i : i \in |\mathrm{cod}(f)|\}\rangle) && (5.11) \\ &:= \quad \bigcup_{y \in \mathrm{cod}(f)} \mathrm{cvr}(N_y) = U && (5.12) \end{aligned}$$

where for leaf trees,

$$\operatorname{cvr}(N_y) = \operatorname{cvr}(\langle y, s\rangle) = s \subseteq U. \tag{5.13}$$

We use cvr to determine the (sub-) set of elements that satisfy the conditions formulated along the edges of the tree. ●

Note that the coverage sets of all leaf nodes (and all nodes on the same layer of the tree) are pairwise disjoint and that their union always equals U. As we will discover later, a decision tree is simply a layered representation of partitions of increasing granularity.

Example 5.6 The decision tree in figure 5.5 has five leaf nodes and three decision nodes. The root node is N_f, with $\operatorname{cvr}(N_f) = s$ and three successor nodes N_s, N_o, and N_r. N_o happens to be one of the leaf nodes with $\operatorname{cvr}(N_o) = \{3, 7, 12, 13\}$. $N_s = N_h$ and $N_r = N_w$ are decision nodes with $\operatorname{cvr}(N_h) = \{1, 2, 8, 9, 11\}$ and $\operatorname{cvr}(N_w) = \{4, 5, 6, 10, 14\}$. They are all leaf nodes. ●

Exercise 5.11 ◊ Determine the coverage of the remaining four leaf nodes.

There is an extended graphical notation that adds information about the number of subsumed objects with respect to their classification: We agree on denoting all values

$$|\operatorname{cvr}(N_f) \cap c|$$

for all $c \in \mathfrak{c}$. In figure 5.5, for example, each node is labelled $\frac{p}{n}$ where p is the number of elements of U subsumed by N_f and for which $t(x) = \mathbf{1}$ and n is $|\operatorname{cvr}(N_f) \cap \{x \in U : t(x) = \mathbf{0}\}$. This comes in quite handy in binary classification tasks; for larger $\operatorname{cod}(t)$ one has to specify the labelling carefully (see the examples on page 5.9).

Finally, we need to define the semantics of a decision tree: what is the hypothesis defined by a node?

Decisive tree hypothesis

Definition 5.7 — Decisive tree hypothesis.
Every node N_f in a decision tree represents a hypothesis h_{N_f}:

$$h_{N_f}(x) \quad = \quad \operatorname{mcv}_t(\operatorname{cvr}(N_f)) \tag{5.14}$$

This means that every node is labelled with the majority of target classifications (see figure 5.6) and every decision node inherits the majority vote from the sum of its successors. ●

Example 5.7 The leaf node hypotheses of the tree in figure 5.5 are, reading the front from left to right: $h_{N_l} = \mathbf{1}$, $h_{N_h} = \mathbf{0}$, $h_{N_o} = \mathbf{1}$, $h_{N_y} = \mathbf{0}$, and $h_{N_n} = \mathbf{1}$. ●

Exercise 5.12 ◊ Determine the hypotheses represented by the decision trees!

Top-Down Decision Tree Induction
Inducing a decision tree means to recursively partition the set of all objects by equivalence relations represented by the features of the underlying information system. In each step, the feature chosen for partitioning is the one with maximum information gain.

From section 5.1 we know that the best feature to choose in each step is the one with maximum information gain. Accordingly, we can now formulate an algorithm for decision tree induction as shown in figure 5.6: Starting with the root node (i.e. a top decision node) F that covers all objects described by our sample, we recursively choose the feature with maximum information gain to split the current node into successor nodes where each one represents an equivalence class of objects with respect to this feature. We continue from left to right until we have classified all objects (i.e. all leaf nodes have zero t-entropy or until we run out of features). Finally, we can define the hypothesis represented by a tree N_f. For known elements x of our universe, the case is simple: we take all the leaf trees and determine the one which contains x. There is exactly one such leaf node N and we then apply a majority voting to assign a target label to x. If all the leaves N_c only contain subsets of target classes (formally spoken: if for all N_c it holds that $N_c \subseteq c \in \mathfrak{c}$) then the majority is always 100 per cent. Things are a bit more complicated if we want to determine $h(x)$ for some $x \notin \mathrm{cvr}(N_f)$. Then, we simply determine x's value under f and stuff it into the cover set of the according successor node. We repeat this until x arrives in a leaf node—and then we return the majority vote again of this leaf.

```
proc $tdidt(s, \mathbf{F})$
{
    IF ($\mathrm{entropy}_t(s) = 0$) THEN    % c.f. Defn. (5.5)
    {   return ($s$)   };
    ELSE
    {
        $f = \arg\max\{\mathrm{gain}_t(f, s) : f \in F\}$    % c.f. Defn. (5.15)

        $s'' := \{\}$
        FORALL ($v \in \mathrm{cod}(f)$) DO
        {
            $s' := tdidt(\{x \in s : f(x) = v\}, \mathbf{F} - \{f\})$;
            $s'' := s'' \cup \{\langle f, v, s' \rangle\}$;
        } DONE
    } ENDIF
    return ($s''$)
}
```

Figure 5.6: Top Down Induction of Decision Trees

This algorithm already motivates an idea towards decision tree pruning: If the error of a majority vote does not dramatically increase when pruning away all successors of a decision node, then why keep them at all?

Exercise 5.13 (◊) In definition 5.7, the hypothesis is defined by the sum of the majorities of the successors. The set of subsumed nodes is defined via set union. Why can we safely define the hypothesis equationally whereas $|s \cup s'| \leq |s| + |s'|$?

Exercise 5.14 (♦) Reproduce the decision tree in figure 5.5 by computing all necessary entropies and gains for all the given features.

Exercise 5.15 (◊♦) Build a decision tree from the following information system:

s	f	g	h	t	s	f	g	h	t
○	h	0	0	1	♦	s	4	0	1
□	h	4	45	0	△	h	3	0	0
▶	s	3	90	1	▷	h	3	90	1
◁	h	3	270	1	◀	s	3	270	0
●	s	0	0	1	■	s	4	315	0
◊	h	4	180	0	▲	s	3	0	0

Yet, there remain a few open questions: We will not always be able to create leaves that are subsets of either s^+ or s^-. What shall we do then? And, even if we do have enough different features, is it always a good idea to fully grow the tree? After all, a tree with one hundred per cent accuracy is likely to be overfit and a tree with leaves that cover only one element each surely is nonsense. But before considering to grow smaller trees only or cut large trees down we consider a few improvements of the gain function.

5.4 Gain Again

The problem with *keys* is that they are unique. It certainly is not a problem for your door key, and it is not a problem for keys as they are used in database systems. Keys help to quickly get access to a unique item.

A feature $f \in \mathbf{F}$ is called a key feature, if it is injective: $f(x) = f(y) \Longrightarrow x = y$. In such a case, s/R_f is a set of singletons, and a singleton set is trivially a set with no entropy in it. As a consequence, applying a key feature f always results in maximum information gain.

But key features are rather identifiers than properties that carry *information*: Passport numbers do not correlate to names or the places the according person lives; and course numbers do not always correlate to the course contents. Objects can be *identified* by keys, but they are not *described* by them. In other words, they have no *meaning*.

So if the gain function delivers a maximal value for key features, it simply over-estimates the amount of information in them. In fact, the more values a feature has, the more the gain function as defined in equation (5.7) tends to over-estimate its information. It seems a good idea to penalise features with "too many" values. This leads to the following definition:

Normalized Gain, $\mathrm{gain}_t^{\mathrm{norm}}(f,s)$

Definition 5.8 — Normalized Gain, $\mathrm{gain}_t^{\mathrm{norm}}(f,s)$.
Normalized gain is defined as the gain function gain weighed with the expected amount of information of f as estimated by the number of its values:

$$\mathrm{gain}_t^{\mathrm{norm}}(f,s) = \frac{\mathrm{gain}_t(f,s)}{\log_2 |\mathrm{cod}(f)|} \tag{5.15}$$

So $\mathrm{gain}_t^{\mathrm{norm}}(f,s)$ can be understood as normalised version of $\mathrm{gain}_t(f,s)$ with respect to f's expressiveness. ●

It is only very rarely the case that f actually takes all the values with equal probabilities: Even if there are many values, f will most likely have a non-uniform distribution, otherwise it would be a not very informative feature. As a consequence, $\mathrm{gain}_t^{\mathrm{norm}}(f,s)$ now tends to *underestimate* a feature's utility.

Example 5.8 Let $s = \{x \in \mathbb{N} : 1 \leq x \leq 100\}$ be the set of the first 100 natural numbers and $t(x) = \mathbf{1}$ if and only if x is odd. Considering $\mathit{1}(x) = x$, s/R_{1} is $\{\{x\} : x \in s\}$ and $\mathrm{cod}(\mathit{1}) = s$. Let $\mathit{prime}(x)$ be $\mathbf{1}$ if and only if x is a prime. Then, $\mathrm{cod}(\mathit{prime}) = \mathit{prime}^{\urcorner} = \mathbf{2} = \{\mathbf{1}, \mathbf{0}\}$ and

$$s/R_{\mathit{prime}} = \left\{ \begin{array}{l} \{x \in s : x = a \cdot b \wedge a \neq b \wedge a, b \in s\}, \\ \{x \in s : (x = 1) \vee (x = a \cdot b \wedge \mathit{prime}(a) = \mathbf{1} \wedge a, b \in s - \{1\})\} \end{array} \right\}$$

Since there are 25 primes in the first 100 natural numbers, We calculate $\mathrm{entropy}_{id}(s) = 0 < \mathrm{entropy}_{\mathit{prime}}(s) < 1 = \mathrm{entropy}_t(s)$. Next, $\mathrm{entropy}_t(\mathit{1}, s) = 0 < \mathrm{entropy}_t(\mathit{prime}, s) \approx 0.473 < 1 = \mathrm{entropy}_t(s)$. The gain functions then deliver

$$\begin{array}{rcl} \mathrm{gain}_t(\mathit{1}, s) & = & \mathrm{entropy}_t(s) - \mathrm{entropy}_t(\mathit{1}, s) = 1 - 0 = 1 \\ \mathrm{gain}_t(\mathit{prime}, s) & = & \mathrm{entropy}_t(s) - \mathrm{entropy}_t(\mathit{prime}, s) \approx 0.527. \end{array}$$

Clearly, the gain by *1* is 1, because *1* induces a partition of singletons. The gain by *prime* however is much less: The set of primes contains one even number and 24 odd numbers; the other class 49 even and 26 odd numbers. We now compute the normalised gain:

$$\begin{array}{rcl} \mathrm{gain}_t^{\mathrm{norm}}(\mathit{1}, s) & = & \dfrac{\mathrm{gain}_t(\mathit{1}, s)}{\log_2 |s/R_{id}|} = \dfrac{1}{\log_2 100} \approx 0.151 \\ \mathrm{gain}_t^{\mathrm{norm}}(\mathit{prime}, s) & = & \dfrac{\mathrm{gain}_t(\mathit{prime}, s)}{\log_2 |s/R_{\mathit{prime}}|} = \dfrac{1 - \mathrm{entropy}_t(\mathit{prime}, s)}{\log_2 2} \approx 0.527 \end{array}$$

Now, the primes appear to be a much better predictor for odd numbers. ●

There is a huge problem with codoamin size weighted information gain: If a feature f has a very large codomain but takes only very few different values, then $\mathrm{gain}_t^{\mathrm{norm}}(f,s)$ underestimates. Supposing that $|\mathrm{cod}(f)| \gg |f^{\urcorner}|$ it holds that

$$\frac{\mathrm{gain}_t(f,s)}{|\mathrm{cod}(f)|} < \frac{\mathrm{gain}_t(f,s)}{\log_2 |f^{\urcorner}|}.$$

Therefore it seems much more reasonable to take the cardinality of the range of f as a normalising factor. Thinking a bit further we can find an even better normalisation. Consider $\overline{f}^{\urcorner} = \{x, y\}$ and suppose $|\ulcorner f x| = 1$ and $|\ulcorner f y| = |s| - 1$. In such a case f helps to discriminate only one single object from all the others: its information content is poor. So instead of statically penalizing a feature f by the size of its codomain or range, it appears much more reasonable to take into account the *distribution* of the feature values—and this is again is measured in terms of entropy:

Gain Ratio

Definition 5.9 — Gain Ratio.
We define the *gain ratio* of a feature $f \in \mathbf{F}$ as its gain in relation to its splitting information. We compute the gain ration as a fraction of the actual gain $\text{gain}_t(f, s)$ (with respect to t) and the information of f:

$$\text{gain}_t^{\text{info}}(f, s) \quad = \quad \frac{\text{gain}_t(f, s)}{\text{entropy}_f(s)} \tag{5.16}$$

●

One can, of course, define many different gain functions; one for every problem domain. But one must always be aware of the biased introduced by the definition. For example, $\text{gain}_t^{\text{info}}(f, s)$ is just the same as $\text{gain}_t(f, s)$, if we assume the entropies of all features to be 1. Also, if all features have the same number of possible values, then $\text{gain}_t^{\text{norm}}(f, s)$ is the same as $\text{gain}_t(f, s)$. If we *know* that a certain assumption is true on our data then we can speed up the learning process considerably. If our knowledge of the domain is rather limited, every bias also limits the possible knowledge we want to discover.

Exercise 5.16 (◊-♦) Solve exercises 5.7, 5.8, 5.14, and 5.15 using all the different gain functions we have defined! — You might want to voluntarily solve exercise 5.9 first.

But even the most sophisiticated gain measure will not help to overcome the biggest problem: the "better" a tree for classification, the bigger the chance for overfitting. We pick up again the idea from the end of the previous section: With a highly accurate or simply huge tree we want to know whether we shall prune the tree—and, if so, how we can prune it.

5.5 Pruning

[**?**] describes an observation he made during a test run:

> [... Given] an artifical [random] data set with 10 [binary] attributes [...] with equal probability. The [target] class was also binary, **1** with probability 1/4, **0** with probability 3/4. One thousand randomly generated test cases were split into a training set of 500 and a test set of 500. From this data, C4.5's initial tree building routine produces a nonsensical tree of 119 nodes that has an error rate of more than 35% on the test cases.

It is clear that learning from random noise is nearly impossible.[6] Yet, the most interesting fact is that the resulting tree consists of 119 nodes and only reaches an accuracy of 65%. Since learning also requires some compression, it could be that a much smaller tree does not produce significantly worse results. As we already know it is reasonable to return a suboptimal hypothesis rather than an overfit one.

Pruning
Exhaustive decision trees tend to be (1) too big in relation to their accuracy, or (2) they even overfit (i.e. perform worse on test data). Accordingly, one would like to restrict tree growth or prune a fully grown tree afterwards. This way, the predictive accuracy/tree complexity can be increased and overfitting decisions can be cut off.
Prepruning uses error/complexity measures to stop the recursive deepening of a tree while postpruning allows to transform a tree into rules and prune individual rules or even only parts of rules.

An overly specific tree can be pruned by two different methods: First, during tree growth (that is, we stop the inductive tree building process prematurely), and second after exhaustive tree construction. These two methods are called *pre–* and *postpruning*, respectively:

- *Pre–Pruning* means to abort the tree induction process as soon as some criterion is fulfilled (it is a bit like growing Bonsai) whereas

- *Post–Pruning* requires exhaustive growth and post-mortem pruning (which is a bit like clearing the rain-forest).

In both cases the pruning methods can fail to produce a proper tree–cut.
When pre–pruning, one can stop growing the tree if an added branch would not result in some information gain that is beyond a certain threshold (which again can be chosen statically or dynamically). Other simple heuristics are based on maximumn number of nodes or leaves, branches per (sub-) tree or simply leaf size. [**?**] proposes a χ^2–test for significance of further branching a node. A completley different method of prepruning is some kind of a validation bias which tells us when a growing tree is considered to be good enough. Either way, prepruning is always *myopic*: Since information gain decreases with tree depth, we can be sure not to lose more information as gained in the last step but it could be we stop just a few steps too early.
Postpruning means to first grow a tree and then prune it. It can be regarded to as a generate-and-test method with all its advantages and disadvanteges:

> [...] but this cost [of postpruning] is offset against benefits due to a more thorough exploration of possible partitions. *Growing and pruning trees is slower but more reliable.* ([**?**])

In the following sections we will present two kinds of post pruning: *Reduced error pruning* defines an error estimate on decision trees (or rather their nodes). Then it prunes away leaves until an error threshold is reached. A completely

[6] Still, some people try, [Müller, 2008].

different approach first disassembles the tree into an unordered set of rules. Then, *rule post pruning* tries to drop irrelrevant rules or rule antecedents in order to generalise the rule set.

5.5.1 Reduced Error Pruning

Reduced error pruning cuts off subtrees (leaves or entire subtrees for internal nodes) to reduce the error of the hypothesis represented by that tree. In order to evaluate a node's error, one uses the following error estimate:

Leaf/Node error rate

Definition 5.10 — Leaf/Node error rate.
Let N be a node in a decision tree.

1. If N is a leaf node, then

$$\mathrm{err}(N,t) := \frac{|\mathrm{errset}(N,t)|}{|\mathrm{cvr}(N)|} \tag{5.17}$$

   where $\mathrm{errset}(N,t) := \{x \in \mathrm{cvr}(N) : h_N(x) \neq t(x)\}$.

2. If N_f is a decision node, then it has a set of successor nodes $N := \{N_i : 0 \leq i < |\mathrm{cod}(f)|\}$. Similarly, we define:

$$\mathrm{err}_f(N,)\, t = \gamma(N_f)\frac{\sum_{N_i \in N} |\mathrm{errset}(N_i,t)|}{\mathrm{cvr}(N_f)} \tag{5.18}$$

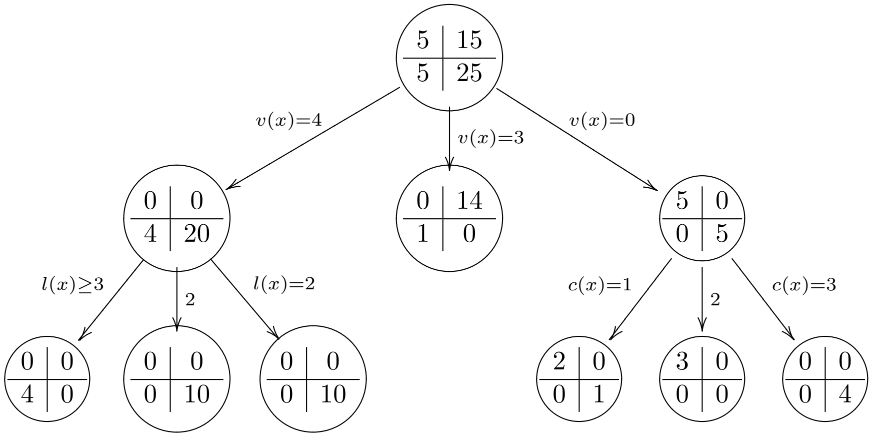
$\gamma(N_f)$ is a weight function with which the error estimate can be adjusted to the branching factor (i.e. the cardinality of $\mathrm{cod}(f)$), the depth of N_f, or any other additional costs or benefits. ●

For now it suffices to take $\mathrm{err}(N,t)$ as an error measure that increases from root to leaves (we will come back to this later). This is somehow counterintuitive, but the reason for it is in the pruning algorithm we shall describe now. From the monotonicity it follows that if we prune away a subtree, the error measure overestimates the error in the remaining leaf by at least the depth of the pruned tree. A top-down pruning algorithm is shown in figure 5.7.

Example 5.9 Recall our example domain as shown in figure 4.3. Now let there be a a set s with $|s| = 50$ different geometric objects. We want to classify them into four different classes: Diamonds, triangles, rhomboids, and ellipses; i.e. $\mathrm{cod}(t) = \mathbf{4}$. Let $\mathbf{F} = \{v,l,c\}$ where the features describe the number of vertices of an object, the number of edges with equal length and curvatures[7],

[7] With $\mathrm{cod}(c) = \{1,2,3\}$ we describe shapes with constant curvature, non-negative curvature and arbitrary curvature.

respectively. Now imagine $tdidt(s, \{v, l, c\})$ delivers a tree N as follows:



The four quadrants in each node represent the number of objects for which:

$$t(x) = \begin{array}{c|c} \square & \oslash \\ \hline \Diamond & \bigcirc \end{array}$$

We assume $\gamma(N) = 1$. Then, all leaves except two nodes have zero error:

$$\mathrm{err}_s(\begin{array}{c|c} 2 & 0 \\ \hline 0 & 1 \end{array}, t) = \frac{1}{3} \qquad \mathrm{err}_s(\begin{array}{c|c} 0 & 14 \\ \hline 1 & 0 \end{array}, t) = \frac{1}{15}$$

The error estimates for the second layer of the tree are,

$$\mathrm{err}_s(\begin{array}{c|c} 0 & 0 \\ \hline 4 & 2 \end{array}, t) = \frac{4}{24} = \frac{1}{6} \qquad \mathrm{err}_s(\begin{array}{c|c} 5 & 0 \\ \hline 0 & 5 \end{array}, t) = \frac{5}{10} = \frac{1}{2}$$

For the root node we obtain $\frac{5+15+5}{50} = \frac{25}{50} = \frac{1}{2}$. ●

Exercise 5.17 ♦ Determine the node errors for the tree in figure 5.5!

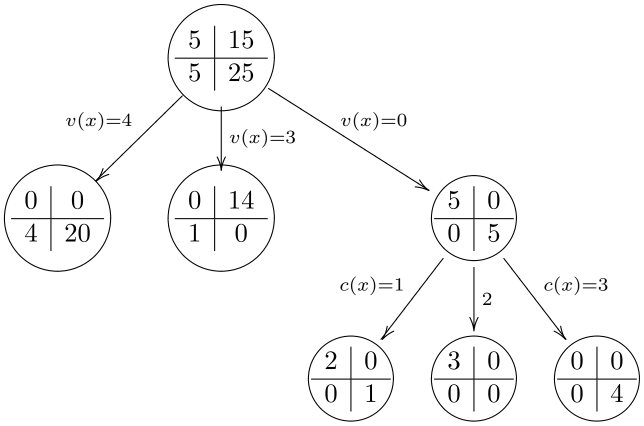
We apply the pruning algorithm from figure 5.7 on the tree from example 5.9 and call $prune(N)$ with N being the root node. Since $\frac{1}{2} > \vartheta$, we call *prune* recursivly on all of the root node's successor nodes. For $f(x) = a$, the error is $\frac{1}{6} \leq \frac{1}{5}$. Therefore, its successor nodes are pruned away. For $f(x) = b$ it is $\frac{1}{15}$, too. But since it is a leaf node, there is nothing left to be pruned away. For $f(x) = c$ it is still $\frac{1}{2}$ which is why we need to call *prune* recursively on all of its

```
01 proc $prune(N,\vartheta)$
02 {   IF  $(\mathrm{err}_s(N,t)) \leq \vartheta)$  THEN
03     {  return($N$)  };
04     ELSE
05     {  FOREACH  $N_i \in sucessors(N)$  DO
06        {   $N_i := prune(N_i,\vartheta)$   };
07        DONE
08        return  ($N$);
09     } ENDIF
10 }
```

Note that when returning N in line 8, its value has changed since the procedure call because of the reassignment of N_i in line 6.
Simple improvements include a dynamic change of ϑ in line 6 or an adaptation of γ used in $\mathrm{err}_s(N,t)$).

Figure 5.7: Top-Down Reduced Error Pruning

successor nodes. The resulting tree after reduced error pruning is



Another method for reduced error pruning is to start at the leaf nodes, prune a node and all of its brothers, if the error in the parent node does not increase by more than a certain threshold. The advantage of this algorithm is that we do not recompute error measures as in algorithm shown in figure 5.7.
There are many related error measures and according algorithms; the presented error complexity measure is used in Cart, [?]. [?] and [?] describe a minimal error pruning method and [?] gives a empirical comparison of different such pruning methods.

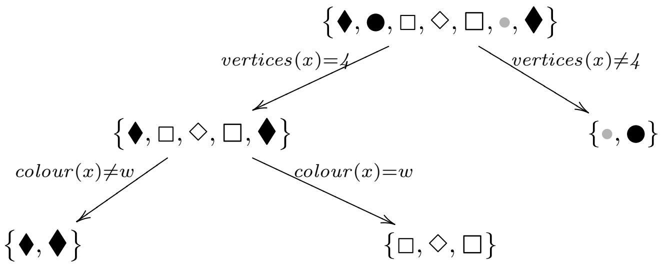
5.5.2 Rule-Based Post-Pruning

A completely different idea is to translate a tree into a set of rules and then prune the rule set. This method has a huge advantage: one can delete entire rules from a set of rules and rules can be weakened by pruning their antecedents.

Rule Representation of Decision Trees
A decision tree can be expressed by a set of rules where each rule represents a path in the tree. The premises of each rule is a conjunction of all the feature–value restrictions along the edges and the conclusions consist of the hypotheses represented by the leaf nodes.

Decision trees can be translated into a set of rules very easily. As a tree is a hierarchical partitioning with respect to increasingly fine grained equivalence relations, tree traversal from root to leaf is simply a conjunction of predicates derived from the intersections of the equivalence relations induced by the respective features.

Example 5.10 Consider the following binary decision tree:



This tree can be interpreted as a set of four rules:

$$\begin{array}{llll}
(\mathit{vertices}(x) = 4) & & \longrightarrow & \mathit{tetragon}(x) \\
(\mathit{vertices}(x) = 4) & \wedge \quad (\mathit{colour}(x) = \mathit{black}) & \longrightarrow & \mathit{rhombus}(x) \\
(\mathit{vertices}(x) = 4) & \wedge \quad (\mathit{colour}(x) = \mathit{white}) & \longrightarrow & \mathit{square}(x) \\
(\mathit{vertices}(x) \neq 4) & & \longrightarrow & \mathit{cirlce}(x)
\end{array}$$

The root node represents the fact that everything in our domain is a geometric object: *object*(x). ●

Formally, every leaf in a decision is connected to the root along a path through several decision nodes along edges that define an object's value for the decision node's feature:

Rule Representation of Decision Trees

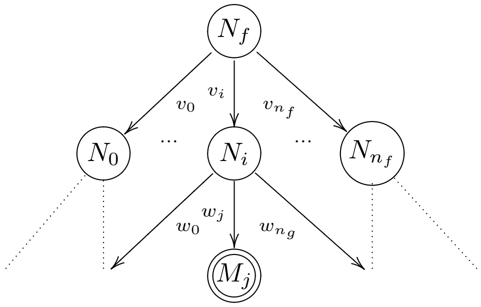
Definition 5.11 — Rule Representation of Decision Trees.
Let N_f be the root node of a decision tree. As hypothesis it always returns the most common class value. We define

$$\varphi(N_f) := \big\{(h(x) = h_{N_f}(x))\big\} \tag{5.19}$$

This set, when read as a set of literals, is a Horn clause with an empty premise:

$$\mathbf{1} \longrightarrow h(x) = h_{N_f}(x)$$

Every successor node has the feature–value restrictions along its path as premises. Let M_j be the j-th successor of N_i so that M_j subsumes all those objects covered



$$\mathbf{1} \longrightarrow (h(x) = h_{N_f}(x))$$

$$(f(x) = v_i) \longrightarrow (h(x) = h_{N_i}(x))$$

$$(f(x) = v_i) \wedge (g(x) = w_j) \longrightarrow$$
$$(h(x) = h_{M_j}(x))$$

Figure 5.8: Converting a decision tree into a set of decision rules

by N_i for which $g(x) = w_j$. Then,

$$\begin{aligned} \varphi(M_j) \quad := \quad & \varphi(N_i) \\ & - \{(h(x) = h_{N_i}(x))\} \\ & \cup \left\{\neg(g(x) = w_j), (h(x) = h_{M_j}(x))\right\} \end{aligned} \tag{5.20}$$

This expression adds the literal $g(x) = w_j$ to the set of premises and changes the tree hypothesis to hypothesis represented by M_j. ●

An example is shown in figure 5.8. Let us consider the tree from figure 5.5 again.

Example 5.11 We build the tree from beginning at the root node all the way down to the leaves:

$$\varphi(N_t) \quad = \quad \{(h(x) = h_{N_t}(x))\}$$

The root node is divided by *forecast* into three successor nodes. The first one is N_s with

$$\begin{aligned} \varphi(N_s) \quad &= \quad \varphi(N_t) - \{(h(x) = h_{N_t}(x))\} \cup \{\neg(\mathit{forecast}(x) = s), (h(x) = h_{N_s}(x))\} \\ &= \quad (\mathit{forecast}(x) = s) \longrightarrow (h(x) = \mathrm{mcv}_t(\mathrm{cvr}(N_s))) \\ &= \quad (\mathit{forecast}(x) = s) \longrightarrow (h(x) = \mathbf{0}) \end{aligned}$$

The next step is to partition N_s by *humidity*:

$$\begin{aligned} \varphi(N_l) \quad &= \quad \varphi(N_s) - \{(h(x) = h_{N_s}(x))\} \cup \{\neg(\mathit{humidity}(x) = l), (h(x) = h_{N_l}(x))\} \\ &= \quad (\mathit{forecast}(x) = s) \wedge (\mathit{humidity}(x) = l) \longrightarrow (h(x) = \mathrm{mcv}_t(\mathrm{cvr}(N_l))) \\ &= \quad (\mathit{forecast}(x) = s) \wedge (\mathit{humidity}(x) = l) \longrightarrow (h(x) = \mathbf{1}) \end{aligned}$$

Now the rule set $\{\varphi(N_s), \varphi(N_l)\}$ obviously is inconsistent. But as long as we can't prove that the weather is sunny with low humidity, we could still try and prove sunny forecast only. The first, more general rule represented by N_s recommends *not* to play golf if it's sunny. The second, more specific rule, specifies an *exception*: Even if we shan't play golf because it is too sunny (i.e. in any case covered by the N_s) we may go though and play golf—but only if the air is not too humid.

Exercise 5.18 ♦ Determine the rule set representing all paths in the tree in figure 5.5!

In oder to apply rule post pruning to a decision tree, we:

1. Induce a full tree (unbiased)
2. Translate the tree into rules
3. Prune the rules instead of the tree
4. Sort and/or translate rules back into a tree

With rule-based postpruning we are able to rune away subtrees just as we did in prepruning. In addition, we can also delete *parts* of paths which, graphically, does not result in a tree any more but rather in a forest. To give you an idea, we first examine geometric shape domain again.

Example 5.12 Recall the decision tree from example 5.10. Obviously, the feature *colour* is sufficient to discriminate squares from non-squares but it is not sufficient to tell circles from tetrangles in general (and specifically, rhombuses): All the squares are white ($\diamond, \square, \Box$) and none of the other shapes are white. On the other hand, ● and ⧫ or ⧫ are black but only the former one is a circle whilst the latter two shapes are rhombuses. So whenever discriminating squares from non-squares we can safely drop the restriction to tetragons ($vertices(x) = 4$). Let us take at the leaf nodes. From

$$\begin{array}{lclcl}
(vertices(x) = 4) & \wedge & (colour(x) = black) & \longrightarrow & rhombus(x) \\
(vertices(x) = 4) & \wedge & (colour(x) = white) & \longrightarrow & square(x) \\
(vertices(x) \neq 4) & & & \longrightarrow & cirlce(x)
\end{array}$$

we delete the first premise of the second rule and obtain

$$\begin{array}{lclcl}
(vertices(x) = 4) & \wedge & (colour(x) = black) & \longrightarrow & rhombus(x) \\
 & & (colour(x) = white) & \longrightarrow & square(x) \\
(vertices(x) \neq 4) & & & \longrightarrow & cirlce(x)
\end{array}$$

The result is a tree that is not a proper decision tree anymore! And if $x \in \{\square, \Diamond, \Box\} \Longrightarrow t(x) = \mathbf{1}$, then we can drop the entire root decision tree:

$$\begin{array}{rcl}
White & \longrightarrow & Square \\
\neg White & \longrightarrow & \neg Square
\end{array}$$

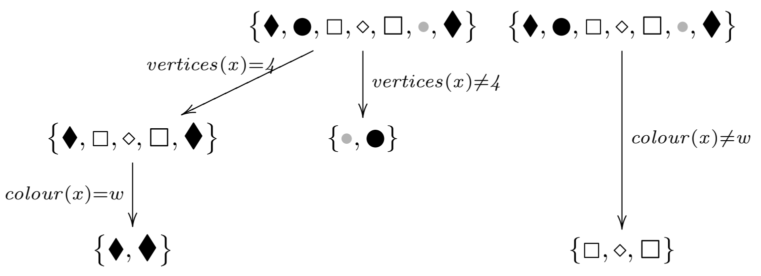


Figure 5.9: Rule Pruning results in a forest of trees

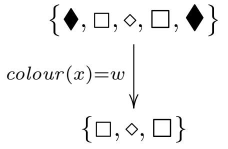
This example shows that rule-based postpruning—despite its increased computational effort due to full tree expansion—has at least three important advantages over ordinary (pruned) decision trees:

- Even though trees are a very simple and scrutable representations of hypotheses, small rule sets are better suited for conceptual descriptions.

- Based on rule sets and appropriate operators (rule and literal dropping) we can formulate both more specific and generic hypotheses with lesser error.

- With rule sets we can express that are not representable by a single decision tree at all

The "forest" of trees corresponding to the rule sets from the previous example is shown in figure 5.5.2. After the second pruning step, the forest in figure 5.5.2 finally implodes to



5.6 Conclusion

This chapter presented a purely relational view on a popular machine learning method, namely decision tree induction. The various algorithms are nothing else than a heuristically driven search for a partitioning of the base set with respect to a decision attribute (i.e. the target function). The relational view makes it much easier to understand several interesting properties of decision trees.

There remain a few open questions.
First, we have not explained how to deal with (quasi–) continuous feature values. If, for example, the humidity in figure 5.5 is not described by three qualitivative linguisitic variables but by an integer from the interval $[0, 99]$, then we would have to split a node in up to 100 successor nodes which is not really informative. Instead, on applies a quantisation algorithm like *binning* first. This yields a "discretised" set of few linguisitic variables each of which represents an interval over the domain. This actually is a lossy representation shift—and, in fact, one can observe quite often that results improve from deliberate information loss through quantisation. But this will be discussed later.
To conclude this section on decision trees, we summarise:

1. decision trees are a very simple and, thus, widely accepted method for representing classifiers.

2. Objects of the domain are described by attribute–value pairs

3. All features have discrete codomains. This can be forced by quantisation.

4. TDIDT can cope with noisy training data due to its entropy based measure of information

5. An exhaustive search for smallest, **s**–consistent tree is NP–complete as shown by Hyafil and Rivest in 1976, but an information gain guided search is a very quick greedy approach.

Finally, we again stress the fact that entropy usually assumes an uniform distribution over all possible system states. This is taken into account by a probability weighted sum in Shannon's measure. In thermodynamics we operate with mutually independent states of closed systems. In a domain of information processing, it is hardly the case that messages in a sequence are mutually independent—and it is not clear what it means for an information system to be closed (an issue that might point to the frame problem in AI again). Therefore, decision tree induction is an efficient but biased tool for relational knowledge discovery.

Chapter 6

Rough Set Theory

> Any algorithm we have come across so far makes several (severe) assumptions on the domain. Together with the knowledge we feed into our learning systems, the representation itself and the implementation of algorithms may result in heavy biases. But what if we just look at the objects we are given and their relational properties? Why should we try to discriminate indistinguishable objects instead of interpreting indiscernability as "being–of–the–same–breed"—whatever our current knowledge of different existing breeds is?

At the beginning of the last chapter we discovered that features induce equivalence relations and that equivalence relations create blocks of indiscernible objects; i.e. "small blocks of similar, equal, or equivalent things". Any two objects in an equivalence class cannot be distinguished from each other, but two objects from different classes can be well discriminated. For our informations systems usually provide a large number of features, we also have many equivalence relations. Furthermore, any intersection of any subset of such equivalence relations forms a new equivalence relation, too. And since equivalence relations are relations, and since relations are sets, it appears an interesting idea to consider the *intersection* of equivalence relations as a much finer and more detailed partitioning of our base set.

6.1 Knowledge and Discernability

For an arbitrary set of equivalence relations $\mathbf{R} = \{R_i : i \in \mathbf{n}\}$, the intersection

$$\bigcap_{i \in \mathbf{n}} R_i \tag{6.1}$$

is an equivalence relation, too.

Exercise 6.1 That early already? Yes, that early! Prove that the intersection of two equivalence relations is an equivalence relation! — Since you have read the first chapter, this exercise is not even worth a single ◊.

Now that the intersection of two equivalence relations somehow corresponds to a logical conjunction, it is clear that the blocks get smaller since the premises become more special (we already discovered this fact at the end chapter 5). Accordingly, the most detailed knowledge we can get from $\mathbf{R}$ is $\bigcap \mathbf{R}$:

Indiscernability Relation

Definition 6.1 — Indiscernability Relation.
For a set $\mathbf{R}$ of equivalence relations over s we call

$$\bar{\bar{\mathbf{R}}} = \bigcap_{R \in \mathbf{R}} R = \bigcap \mathbf{R} \tag{6.2}$$

the *indiscernability relation over* $\mathbf{R}$. Of course, $\bar{\bar{\mathbf{R}}}$ is an equivalence relation. Sometimes, we shall write $x \overset{\mathbf{R}}{=} y :\Longleftrightarrow x\bar{\bar{\mathbf{R}}}y \Longleftrightarrow \forall f \in \mathbf{F} : f(x) = f(y)$. ●

With given knowledge $\mathbf{R}$ we can examine s in terms of different partitions, blocks and blocks of different partitions:

- Elements of s/R for any $R \in \mathbf{R}$ are called *elementary categories*.
  They correspond to R–equivalence classes and determine the sets of objects that share the same value under the feature that induced R.
- Elements of $s/\bar{\bar{\mathbf{R}}}$ are called ($\mathbf{R}$–) *basic categories* (wrt. $\mathbf{R}$).
  Any element of the quotient $s/\bar{\bar{\mathbf{R}}}$ is a set of objects which are indiscernible using *all* the relations in $\mathbf{R}$.

Since basic categories include the knowledge of several relations, they usually are finer than elementary categories while elementary categories are coarser.

Exercise 6.2 ♦ Prove that elementary categories are unions of basic categories, and that a basic category is always a subset of exactly one basic category.

The finest partition of s—which is induced by $\bar{\bar{\mathbf{R}}}$—is what we call the *(indiscernability) knowledge* about s. More technically speaking, it forms a *knowledge base*.

Knowledge Base

Definition 6.2 — Knowledge Base.
For an information system $\mathfrak{I} = \langle s, \mathbf{F}, V_{\mathbf{F}} \rangle$, we use $\mathbf{R}$ to denote the set of equivalence relations induced by $\mathbf{F}$:

$$R_i \in \mathbf{R} \Longleftrightarrow f_i \in \mathbf{F} \text{ and } xR_iy \Longleftrightarrow \forall x \in s : f_i(x) = f(y)$$

Then, we call

$$\mathfrak{K} = \left\langle s, \mathbf{R} \cup \left\{ \bar{\bar{\mathbf{P}}} : \mathbf{P} \subseteq \mathbf{R} \right\} \right\rangle \tag{6.3}$$

the *knowledge base* (of $\mathbf{R}$ on s). ●

Exercise 6.3 (♦) Show that $\bar{\bar{\mathbf{R}}} = \bigcap \left\{ \bar{\bar{\mathbf{P}}} : \mathbf{P} \subseteq \mathbf{R} \right\}$!

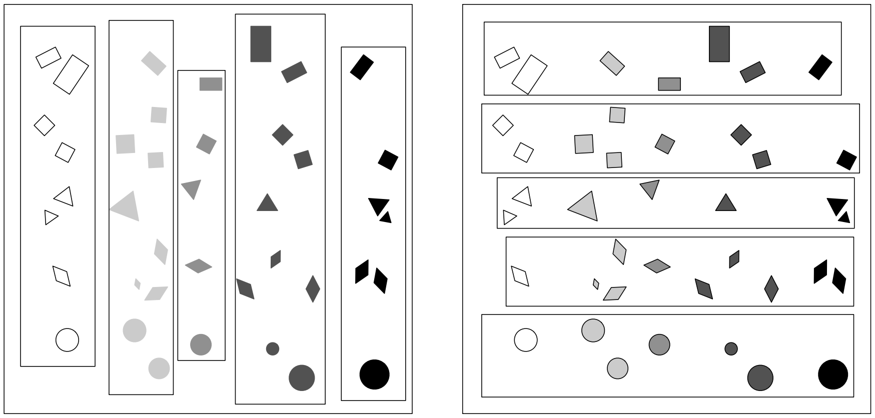


Figure 6.1: A universe and two equivalence relations on it

A knowledge base does not contain any more information than the information system from which it is derived—the only thing that makes knowledge from information is that we are able to express the information in several ways. We now shall see why.

Example 6.1 Figure 6.1 shows a set of objects and two equivalence relations on it. Each equivalence class of the color- or shape relation forms an elementary category. The left illustrutiona shows the equivalence classes with respect to colour, the other one shows the shape-elemtary classes. ●

Indiscernability
An indiscernability relation describes all objects of the universe by means of *all* the relational knowledge we have. Different objects that belong to the same equivalence class of such an indiscernability relation are not distinguishable by any piece or the entirety our knowledge.
If we are able to describe all the objects well enough, then we are interested in minimal sets of knowledge that still suffice to distinguish different things from each other.

By intersecting the shape and colour-relation from figure 6.1 we obtain a much finer partition of the set: we can, for example, discriminate dark squares from white squares (which we could not without the knowledge of colours) and gray circles from gray diamonds squares (which we could not without the knowldege of shapes). This is shown in figure 6.2. Let us take a look at a further equivalence relation: Imagine a feature *vertices* $: U \to \mathbb{N}$ that describes the number of corners of an object. The induced equivalence relation would then partition our base set into three blocks: One containing all the circles ($vertices(x) = 0$), one containing all the diamonds, squares, rhombusesm and quadrangles

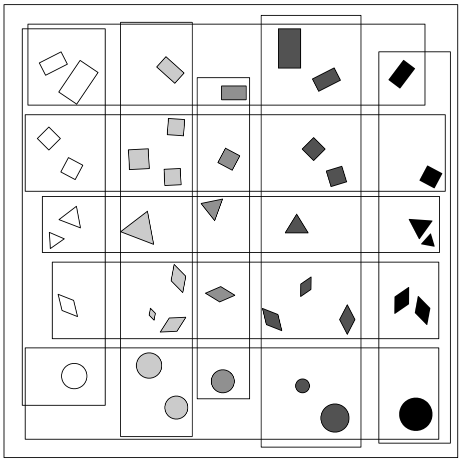


Figure 6.2: An Indiscernability Relation

($vertices(x) = 4$), and one containing all the triangles ($vertices(x) = 3$); see figure 6.3. As one can see, every block of the *shape*–partition is a subset of a block of the *vertices*–partition. In other words, all elementary *vertice*–classes are subsets of elementary *shape*–classes and speaking in terms of equivalence relations, R_{shape} is a *subset* of $R_{vertices}$.

Exercise 6.4

- Prove that $R_{shape} \subseteq R_{vertices}$!
- Define the according knowledge base for the information system underlying the examples in figures 6.1– 6.3.

Whenever we have more detailed knowledge (here: *shape*) that allows us to formulate more accurate concepts, we usually say that the additional information contributes to our knowledge by *refining* our previously rather rough conceptualisations.[1] This leads us to the following definition:

Refinement

Definition 6.3 — Refinement.
For two sets $\mathbf{P}, \mathbf{R}$ of equivalence relations we say that $\mathbf{P}$ is *coarser* (*more general*) than $\mathbf{R}$,

$$\mathbf{R} \preceq \mathbf{P} \quad \text{if and only if} \quad \bar{\bar{\mathbf{R}}} \subseteq \bar{\bar{\mathbf{P}}} \tag{6.4}$$

If $\mathbf{P}$ is coarser than $\mathbf{R}$, then $\bar{\bar{\mathbf{R}}}$ *refines* $\bar{\bar{\mathbf{P}}}$. ●

[1] An extreme case is where $s/\bar{\bar{\mathbf{P}}} = \{\{x\} : x \in s\}$. Then, $\bar{\bar{\mathbf{P}}}$ creates a set of singleton equivalence classes and *all* objects $x \in s$ are pairwise discernible using knowledge $\mathbf{P}$.

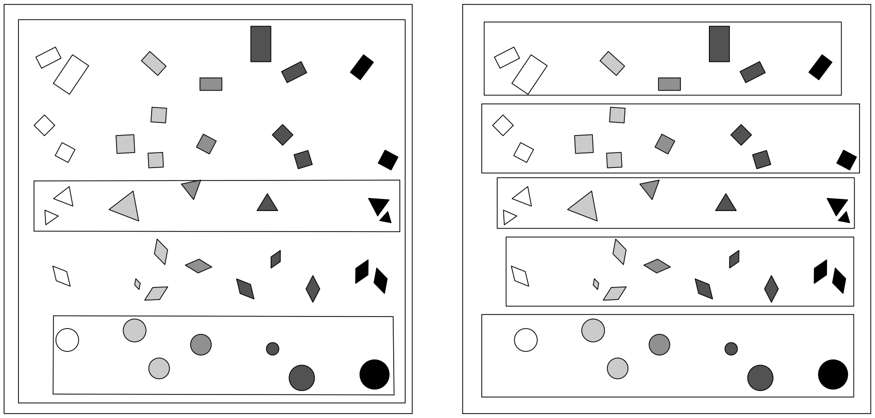


Figure 6.3: Another equivalence relation (left)

The choice of $\preceq$ as a symbol for refinement is, as one can see in the definition, motivated by the fact that the refined induced equivalence relation, when viewed as sets of tuples, in fact is a *subset* of the coraser relation. This becomes clear when representing the relations graphically in coincidence $\mathbb{M}(\bar{\bar{\mathbf{R}}})$ or kernel matrices $\mathbb{K}(\bar{\bar{\mathbf{R}}})$ (see section 2.2.1). In order to have the reader become more familier with the formal aspects (rather than just painting Venn-diagrams), we shall now apply our definitions to a subproblem of our graphical example:

Example 6.2 Let $\mathfrak{K} = \langle s, \mathbf{R} \rangle$ be an information system as follows:

$$\begin{aligned} s &= \{\bullet, \blacksquare, \blacktriangle, \bullet, \blacksquare, \blacktriangle, \bullet, \blacksquare, \blacktriangle, \} \\ \mathbf{R} &= \{R_{color}, R_{shape}, R_{polygon}\} \end{aligned}$$

We choose $\mathbf{P} = \{R_{shape}, R_{polygon}\} \subset \mathbf{R}$. Then,

$$\begin{aligned} & s/\bar{\bar{\mathbf{P}}} \\ = \; & s/(\bigcap \mathbf{P}) \\ = \; & \pi_{shape} \cdot \pi_{polygon} \\ = \; & \{\{\bullet, \bullet, \bullet\}, \{\blacksquare, \blacksquare, \blacksquare\}, \{\blacktriangle, \blacktriangle, \blacktriangle\}\} \odot \{\{\bullet, \bullet, \bullet\}, \{\blacksquare, \blacksquare, \blacksquare, \blacktriangle, \blacktriangle, \blacktriangle\}\} \\ = \; & \{\{\bullet, \bullet, \bullet\}, \{\blacksquare, \blacksquare, \blacksquare\}, \{\blacktriangle, \blacktriangle, \blacktriangle\}\} \\ = \; & \pi_{shape} = s/R_{shape} \end{aligned}$$

As one can see, the *shape* partition is finer than the *polygon*–partition.[2] ●

[2] The circled dot '$\odot$' stands for pairwise intersection: $s_0 \cdot s_1 := \{s'_0 \cap s'_1 : s'_0 \in s_0, s'_1 \in s_1\}$.

So what if two sets of equivalence relations of a knowledge base have exactly the same degree of coarsesness in the sense that one refines the other and vice versa? From $x \leq y$ and $y \leq x$ we usually infer that $x = y$. Similarly, mutual refinement defines *equivalence* of knowledge:

Equivalence of Knowledge

Definition 6.4 — Equivalence of Knowledge.
$\mathbf{R}$ and $\mathbf{P}$ are called *equivalent*, if:

$$\mathbf{R} \equiv \mathbf{P} \quad :\Longleftrightarrow \quad \bar{\bar{\mathbf{R}}} \preceq \bar{\bar{\mathbf{P}}} \wedge \bar{\bar{\mathbf{P}}} \preceq \bar{\bar{\mathbf{R}}} \Longleftrightarrow \bar{\bar{\mathbf{R}}} = \bar{\bar{\mathbf{P}}} \tag{6.5}$$

Then, $s/\mathbf{R} = s/\mathbf{P}$. ●

Note that $\mathbf{P} = \mathbf{R}$ implies $\mathbf{P} \equiv \mathbf{R}$ but *not* vice versa. This is a very simple fact, but an important observation though: It expresses the fact, that the same information contains the same knowledge, but that the same knowledge can be expressed in different terms!

Keep in mind that $\bar{\bar{\mathbf{R}}} = R$!
Note that whenever we speak of a set $\mathbf{R}$ of equivalence relations, $\bar{\bar{\mathbf{R}}}$ is an equivalence relation, too. Therefore, if we consent to denote by R an arbitrary equivalence relation, there is no difference between $\mathbf{P} \equiv \mathbf{R}$ or $P = R$, since $P = \bar{\bar{\mathbf{P}}} = \bar{\bar{\mathbf{R}}} = R$. Nevertheless we will carefully distinguish between sets $\mathbf{R}$ of relations and relations R. Therefore $\bar{\bar{\mathbf{R}}}$ is a single relation (e.g. $R := \bar{\bar{\mathbf{R}}}$), but for a single relation R, $\bar{\bar{R}}$ is not defined (whereas $\overline{\overline{\{R\}}} = R$ is well-defined).

Example 6.3 Recall example 6.2: As one can see, $R_{shape} \preceq R_{polygon}$ such that $\pi_{shape} \cdot \pi_{polygon} = \pi_{shape}$. We chose $\mathbf{P} = \{R_{shape}, R_{color}\} \subset \mathbf{R}$. Then,

$$\begin{aligned} & s/\bar{\bar{\mathbf{P}}} \\ = \ & \pi_{shape} \cdot \pi_{color} \\ = \ & \{\{\bullet, \bullet, \bullet\}, \{\blacksquare, \blacksquare, \blacksquare\}, \{\blacktriangle, \blacktriangle, \blacktriangle\}\} \cdot \{\{\bullet, \blacksquare, \blacktriangle\}, \{\bullet, \blacksquare, \blacktriangle\}, \{\bullet, \blacksquare, \blacktriangle\}\} \\ = \ & \{\{\bullet\}, \{\blacksquare\}, \{\blacktriangle\}, \{\bullet\}, \{\blacksquare\}, \{\blacktriangle\}, \{\bullet\}, \{\blacksquare\}, \{\blacktriangle\}\} \end{aligned}$$

Then,

$$\begin{aligned} & \{\blacksquare, \blacksquare, \blacksquare\} \cup \{\bullet, \blacksquare, \blacktriangle\} \\ = \ & \{\blacksquare\} \cup \{\blacksquare\} \cup \{\blacksquare\} \cup \{\bullet\} \cup \{\blacktriangle\} \\ = \ & \{\blacksquare, \blacksquare, \blacksquare, \bullet, \blacktriangle\} \end{aligned}$$

is the $\mathbf{P} = \{R_{shape}, R_{color}\}$–category of squarish or dark objects. ●

We shall end this introductory section with a small remark: In most books on rough set theory there are Venn-diagram-like examples for the concepts presented in this chapter. They all differ slightly from the figures 6.1- 6.4 in this book. Take figure 6.1: The boundaries of the different equivalence relations do *not* coincide, not even in cases where they create the same line of distinction between discernible objects. The reason for this is that relations carry *intensional knowledge* while discernability on object level carries information about memberships; i.e. *extensional knowledge*. So if two collections of things (called a set) are equal, then an object belongs to one of these collections if and only if it belongs to the other collection as well. This called (*elementwise* or *pointwise*) equality of sets. But if we understand sets as collections being *defined* in terms of relations, the sets are defined by their meaning which is not the same as their content. Accordingly, a set can be a subset of another when looking at it elementwise—but it may also have an "empty" region outside the superset (see figure 6.4 in the next section).

6.2 Rough Knowledge

Let us briefly recall the definition of a knowledge base: A knowledge base contains all intersections of all subsets of our set of relations $\mathbf{R}$. Accordingly, $R \in \mathbf{R} \rightarrow \bar{\bar{\mathbf{R}}} \subseteq R$ and all pairwise intersections of relations are elements of $\mathfrak{K} = \langle s, \mathbf{R} \rangle$, and so are the elements of the closure under intersections. The set of equivalent relations is not closed under set union, but we can still a whole lot of equivalence relations to our knowledge base: Let there be two elementary classes from possibly different relations:

$$[x]_P \text{ and } [y]_R$$

Then, $\chi([x]_P \cup [y]_R)$ induces an equivalence relation of index 2. If we iterate this process, we can construct equivalence relations for *any union* of basic categories of $\bar{\bar{R}}$. By using intersections to derive basic classes from elementary ones and unions to form groups of basic categories we can define and describe a huge set of subsets of s. This set is the set of concepts we can express—and it corresponds to what we have called *hypothesis space*.

Data, Information and Knowledge
Information helps us to discriminate objects from each other. Knowledge, on the other hand, allows us to operate on elementary categories to describe concepts by conjunctions (intersection) and disjunctions (union) of basic properties. So

- *Data* is a set of representations of observations,
- *information* is what we need to define a structure on the data, and
- *knowledge* is the toolbox we have in order to define such a structure.

So data are instances of concepts, information is what tells us whether an object belongs to a concept or not and knowledge is what allows us to define a concept. Usually, there are different ways to describe the same concept.

6.2.1 Rough Approximations

We use these relations to define a set or concept $c \subseteq s$:

Definability

Definition 6.5 — Definability.
A concept (i.e. a set) c is called $\mathbf{R}$–*definable*, if there exists a set $\dot{c} \subseteq s$ such that c equals the union set of all R-equivalence classes of all objects in $\dot{c}$:

$$c \text{ is } \mathbf{R}\text{–definable} :\Longleftrightarrow \exists\, \dot{c} \subseteq s : c = \bigcup_{x \in \dot{c}} [x]_{\bar{\bar{\mathbf{R}}}}.$$

Otherwise, c is called **R**–*undefinable*.
The dot notation $\dot{c}$ is an allusion to the *point*-wise definition of sets. ●

The set $\dot{c}$ is called a set of *representatives* and, of course, every element of an equivalence class is a representative of its class: Let there be a classification $\mathfrak{c} = s/R$. Then, a set $\dot{c}$ is a set of representatives for $\mathfrak{c}$ (or, equivalently, for R), if:

$$\bigcup_{x \in \dot{c}} [x]_R = s \text{ and } \forall x, y \in \dot{c} : x \neq y \longrightarrow [x]_R \neq [y]_R$$

This allows us to *reduce* the amount of data required to store the information from our underlying information system in two different ways:

1. Reduction by knowledge:
   We already discovered that $\forall R \in \mathbf{R} : \bar{\bar{\mathbf{R}}} \preceq R$. Accordingly, the information in $\mathfrak{I}$ can be reconstructed by $\bar{\bar{\mathbf{R}}}$. We can extend $\mathfrak{I}$ to $\mathfrak{K}$ by adding all possible concept descriptions using all possible subsets of relations in $\mathbf{R}$.

2. Reduction by representatives:
   Using the *knowledge* of $\mathfrak{K}$, a concept can be *represented* by "examples" which stand for equivalence classes. By unison of the equivalence classes represented by them the concept can be reconstructed

Example 6.4 (On the complexity of knowledge) Let us assume there is a base set s with m elements in it. There are $2^{|s|}$ subsets of s, so there can be 2^m different classes. We want to describe one out of 2^m concepts by using a set of equivalence relations. Usually, we are given such a set of, say n, different relations $\mathbf{R} = \{R_i : i \in \mathbf{n}\}$. But how many possible equivalence relations are there on a base set of size m? The number of equivalence relations is the n-th Bell number; in our case

$$B(n) = \sum_{k \in \mathbf{n}} \binom{n}{k} B(k)$$

The Bell numbers increase at a truly exorbitant rate; for considerable small sets s of size $n = 20$ one can already define more than 5 trillion (!) different equivalence relations.
Let us examine the set of basic categories we can use to define a concept $c \subsetneq s$ using a set $\mathbf{R}$ of relations: The number of basic categories equals the index of $\bar{\bar{\mathbf{R}}}$. If we assume c to be definable at all there must exist a union of basic classes (or, equivalently, a union of intersections of elementary classes) that equals c. This expression needs not to be unique; even for fixed c and $\mathbf{R}$ there can be many different ways to define c. The number of possible hypotheses that can be built by union sets of basic categories is $2^{|s/\bar{\bar{\mathbf{R}}}|}$, i.e. 2 to the power of the index of $\bar{\bar{\mathbf{R}}}$. Of course some of the unions sets will be the same but it does not affect the fact that there are so many different ways to *compute* union sets. Let us call the set of all these union sets Φ. To describe c in terms of some $\varphi = c_1 \cup c_2 \cup \cdots \cup c_k$

(and recall, that c may be definable by a huge number of such unions) we can pick a representative for every basic category c_i in any of these descriptions. This means in turn that there are $\prod_{i \in \mathbf{k}} |c_i|$ different sets of representatives on can use to describe φ.
These few number should demonstrate the infeasability of an uninformed brute force method to discover knowledge. But they also demonstrate the vast amount complexity hidden in knowledge—or, vice versa, it demonstrates that with relatively little additional knowledge one can cover information of immanely additional complexity. ●

But what if our knowledge is not sufficient to describe x *exactly*? If x is only *nearly* R–definable, we speak of *approximate* definitions:

Lower/Upper Approximations, Rough set

Definition 6.6 — Lower/Upper Approximations, Rough set.
Let there be a concept $c \subseteq s$ and an equivalence relation R. We define the *lower* and *upper R–approximation* of c as follows:

$$\begin{aligned} [\![R]\!]c &:= \{x : x \in s \wedge [x]_R \subseteq c\} && (6.6) \\ \langle\!| R |\!\rangle c &:= \{x : x \in s \wedge [x]_R \cap c \neq \emptyset\} && (6.7) \end{aligned}$$

The *R–boundary* of a concept c is the set defined by the difference of the upper and the lower approximation:

$$(\!| R |\!)c \quad := \quad \langle\!| R |\!\rangle c - [\![R]\!]c \qquad (6.8)$$

The narrower this region, the closer our approximation to c. For a given set c, the *(R-)-rough (c-)set* is defined as the tuple $\langle [\![R]\!]c, \langle\!| R |\!\rangle c \rangle$. ●

We can define a *rough characteristic function* $\chi_R : \wp(s) \to \left\{\mathbf{0}, {}^{\mathbf{1}}\!/_{\mathbf{2}}, \mathbf{1}\right\}$ as follows:

$$\chi_R(c)(x) \quad := \quad \begin{cases} \mathbf{1}, & \text{if } x \in [\![R]\!]c \\ {}^{\mathbf{1}}\!/_{\mathbf{2}}, & \text{if } x \in (\!| R |\!)c \\ \mathbf{0}, & \text{if } x \notin \langle\!| R |\!\rangle c \end{cases} \qquad (6.9)$$

Lower and Upper Approximations
A set or a concept c usually has crisp boundaries: An object belongs to it or not. But if there are two indiscernible objects $x \overset{\mathbf{R}}{\equiv} y$, and one of them (say, x) is element of c—is then $y \in c$, too?

- Yes it is,
  . . . if *all* elements of $[x]_{\mathbf{R}}$ are in c; i.e. if $[x]_R \subseteq c$.
- Perhaps it is,
  . . . if *some* elements of $[x]_{\mathbf{R}}$ are in c; i.e. if $[x]_{\mathbf{R}} \cap c \neq \emptyset$.

An equivalent definition of lower and upper approximations is:

$$\begin{aligned} [\![R]\!]c &:= \bigcup \{c' \in s/R : c' \subseteq c\} && (6.10) \\ \langle\!| R |\!\rangle c &:= \bigcup \{c' \in s/R : c' \cap c \neq \emptyset\} && (6.11) \end{aligned}$$

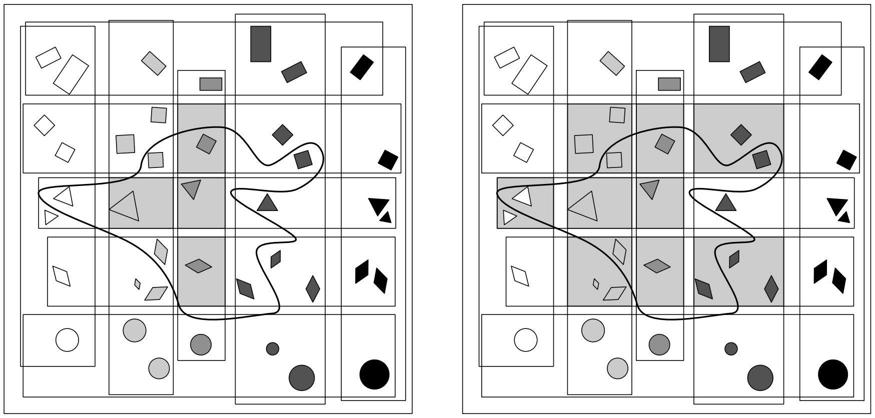


Figure 6.4: Lower and upper approximations of a concept

Quite obviously, it holds that:

$$[\![R]\!]c \subseteq c \subseteq \langle\!|R|\!\rangle c. \tag{6.12}$$

The lower approximation of a set is the union of all basic categories that are fully included in c. This means, that if a basic category $[x]_R$ is a proper subset of c, then x and all objects that are indiscernible from x are instances of c. So whenever an object falls into the category $[x]_{\bar{\bar{\bar{\mathbf{R}}}}}$, then *it must be* an element of c.[3] The upper approximation of a concept c is the set of all objects that belong to classes that intersect with c. So if some $x \in c$, then all objects y which cannot be distinguished from x, belong to the upper approximation as well. The reason is that we know that x is an instance of c. Then one might induce that all other object that are indiscernible from x also *may* belong to c. Since we are talking equivalence relations here, objects can be equivalent or not—but there is nothing inbetween. This might make the idea behind upper approximations a bit harder to grasp. It might be easier to understand when weakening the notion of equivalence to similarity:[4] If x belongs to c and y is *similar* to x, then y could be an element of c, too. Figure 6.4 shows the lower and upper approximations of a concept c in our running example.

Example 6.5 Again, consider $\mathfrak{K}$ as in example 6.2. Let the (unknown)

[3] This why we use the notation $[\![R]\!]$ instead of $\underline{R}$: x being an element of the lower approximation of a concept c neccessarily implies that x belongs to c. In modal logic, such operators are denoted by boxes ($\Box$) or box-like symbols. The dual construction, a sufficiency criterion, is represented by diamonds ($\diamond$) which is why we shall use $\langle\!|R|\!\rangle$ for upper approximations.

[4] Which, formally, can be achieved by dropping the requirement of transitivity.

concept be defined in terms of the following objects:

$$c \;:=\; \{\blacktriangle, \textcolor{gray}{\blacksquare}, \textcolor{gray}{\blacktriangle}, \textcolor{gray}{\bullet}\}\,.$$

Then,

$$\begin{aligned}
[\![R_{colour}]\!]c &= \bigcup\{s' \in s/R_{colour} : s' \subseteq c\} = \{\textcolor{gray}{\blacksquare}, \textcolor{gray}{\blacktriangle}, \textcolor{gray}{\bullet}\}\\
[\![R_{shape}]\!]c = [\![R_{polygon}]\!]c &= \bigcup\{s' \in s/R_{shape} : s' \subseteq c\}\\
&= \bigcup\{s' \in s/R_{polygon} : s' \subseteq c\} = \{\}
\end{aligned}$$

It means that whenever an object is equivalent to $\textcolor{gray}{\blacksquare}$, $\textcolor{gray}{\blacktriangle}$ or $\textcolor{gray}{\bullet}$ with respect to its colour, then it definitely belongs to c—while information about shape does not suffice to give a clear definition. On the other hand, the upper approximations result in supersets of c:

$$\begin{aligned}
\langle\!|R_{colour}|\!\rangle c &= \bigcup\{s' \in s/R_{colour} : s' \cap c \neq \emptyset \subseteq c\}\\
&= \{\textcolor{gray}{\blacksquare}, \textcolor{gray}{\blacktriangle}, \textcolor{gray}{\bullet}\} \cup \{\blacklozenge, \blacktriangle, \bullet\}\\
\langle\!|R_{shape}|\!\rangle c = \langle\!|R_{polygon}|\!\rangle c &= \bigcup\{s' \in s/R_{shape} : s' \cap c \neq \emptyset \subseteq c\}\\
&= \bigcup\{s' \in s/R_{polygon} : s' \cap c \neq \emptyset \subseteq c\} = s
\end{aligned}$$

So the upper approximation of the colour relation will classify any gray or black object as a member of c (which is wrong for $\blacksquare$ and $\bullet$), but the upper approximations of the remaining relations do not deliver any helpful information: whichever shape an object has, it may or may not belong to c (because c contains representatives of all possible shapes).

Now recall that $\bar{\bar{\bar{\mathbf{P}}}}$ for any set of relations $\mathbf{P} \subseteq \underline{\mathbf{R}}$ is a relation itself: Since $R_{shape} \subset R_{polygon}$, we know that $\bar{\bar{\bar{\mathbf{P}}}} \cap \{R_{shape}\} \subseteq \bar{\bar{\bar{\mathbf{P}}}} \cap \{R_{polygon}\}$. So let $\mathbf{P}_1 := \{R_{colour}, R_{shape}\}$ and $\mathbf{P}_2 := \{R_{colour}, R_{polygon}\}$. Then,

$$\begin{aligned}
[\![\bar{\bar{\bar{\mathbf{P}}}}_1]\!]c &= \bigcup\left\{s' \in s/\bar{\bar{\bar{\mathbf{P}}}}_1 : s' \subseteq c\right\}\\
&= \{\blacktriangle\} \cup \{\textcolor{darkgray}{\bullet}, \textcolor{darkgray}{\blacksquare}, \textcolor{darkgray}{\blacktriangle}\} = c
\end{aligned}$$

whereas

$$\begin{aligned}
[\![\bar{\bar{\bar{\mathbf{P}}}}_2]\!]c &= \bigcup\left\{s' \in s/\bar{\bar{\bar{\mathbf{P}}}}_s : s' \subseteq c\right\}\\
&= \{\textcolor{darkgray}{\bullet}, \textcolor{darkgray}{\blacksquare}, \textcolor{darkgray}{\blacktriangle}\} \subset c
\end{aligned}$$

and

$$\begin{aligned}
\langle\!|\bar{\bar{\bar{\mathbf{P}}}}_2|\!\rangle c &= \bigcup\left\{s' \in s/\bar{\bar{\bar{\mathbf{P}}}}_s : s' \cap c \neq \emptyset\right\}\\
&= \{\blacktriangle, \blacksquare\} \cup \{\textcolor{darkgray}{\bullet}, \textcolor{darkgray}{\blacksquare}, \textcolor{darkgray}{\blacktriangle}\} \supset c
\end{aligned}$$

Not surprisingly, finer relations contribute to more accurate approximations. $\textcolor{gray}{\bullet}$

In the preceding example, we already made use of a fact that, even if it seems obvious, deserves a lemma:

Theorem 6.1 (Inclusion of Relations carries over to Refinement) For any $\mathbf{P} \subseteq \mathbf{R}$ and $R, R' \in \mathbf{R}$ it holds:

$$R \subseteq R' \iff \bar{\bar{\mathbf{P}}} \cap \{R\} \preceq \bar{\bar{\mathbf{P}}} \cap \{R'\} \iff \overline{\overline{\mathbf{P} \cup \{R\}}} \subseteq \overline{\overline{\mathbf{P} \cup \{R'\}}} \tag{6.13}$$

Exercise 6.5 (⧫) Prove the "$\Longleftarrow$"-direction of the first equivalence in equation (6.13)!

We have already understood what it means for a concept to be definable using an equivalence relation R. Now, upper and lower approximations provide us with a tool for *rough definability.*

Rough Definability

Definition 6.7 — Rough Definability.
We say that

1. c is R-definable, iff $[\![R]\!]c = \langle\!|R|\!\rangle c$.
2. c rough wrt R, iff $[\![R]\!]c \neq \langle\!|R|\!\rangle c$.

●

Exercise 6.6 (⧫) Is it true that if $[\![R]\!]c = c$, then $[\![R]\!]x = \langle\!|R|\!\rangle c = c$?

To complete this introductory section on the basic concepts of rough set data analysis we need to explain what it means to deliver an approximation of a *classification* rather than a single *class.* The definition is straightforward.

Rough approximations of classifications

Definition 6.8 — Rough approximations of classifications.
Let there be a classification $\mathfrak{c} = \{c_i : i \in \mathbf{k}\}$. The *(R-) rough ($\mathfrak{c}$-) approximations* are defined as the sets of all R-approximatinos of all c_i:

$$[\![R]\!]\mathfrak{c} \quad := \quad \{[\![R]\!]c_i : i \in \mathbf{k}\} \tag{6.14}$$

$$\langle\!|R|\!\rangle\mathfrak{c} \quad := \quad \{\langle\!|R|\!\rangle c_i : i \in \mathbf{k}\} \tag{6.15}$$

The *(R)-rough ($\mathfrak{c}$-) classification* is the set of all R-rough classes:

$$\{\langle [\![R]\!]c_i, \langle\!|R|\!\rangle c_i \rangle : i \in \mathbf{k}\} \tag{6.16}$$

●

As we have seen in example 6.5 already, upper and lower approximations can be defined for indiscernability relations and, therefore, for sets of relations. We write

$$[\![\mathbf{R}]\!]s := [\![\bar{\bar{\mathbf{R}}}]\!]s \quad \text{and} \quad \langle\!|\mathbf{R}|\!\rangle s := \langle\!|\bar{\bar{\mathbf{R}}}|\!\rangle s \tag{6.17}$$

in order to avoid notational overhead.

Rough Approximations
Concepts c can be approximated by lower and upper bounds using knowledge $\mathbf{R}$. An object is element of the lower approximation, if all its $\mathbf{R}$–equal objects are in c, too. An object is element of the upper approximation, if there it has an $\mathbf{R}$–equal object that belongs to c.
A *rough set* is a tuple consisting of the lower and upper approximation; and a *rough classification* is the set of all rough classes.

6.2.2 Degrees of Roughness

We already discovered that $\llbracket R \rrbracket c \subseteq \llparenthesis R \rrparenthesis c$. It follows immediately that $|\llbracket R \rrbracket c| \leq |\llparenthesis R \rrparenthesis c|$. So there exists a natural way of comparing the *roughness* of two equivalence relations P and R with respect to a reference set c by comparing their relative sizes of the boundary regions: If

$$\frac{|\llbracket P \rrbracket c|}{|\llparenthesis P \rrparenthesis c|} \quad \leq \quad \frac{|\llbracket R \rrbracket c|}{|\llparenthesis R \rrparenthesis c|} \tag{6.18}$$

then it appears that R is more accurate than P. This leads us to a numerical measure of roughness:

Sharpness, Roughness

Definition 6.9 — Sharpness, Roughness.
The *sharpness* or *accuracy* of the R-approximation of c is defined as

$$\mathit{sharp}_R(c) \quad = \quad \frac{|\llbracket R \rrbracket c|}{|\llparenthesis R \rrparenthesis c|} \tag{6.19}$$

We define $\mathit{roughness}_R(c) = 1 - \mathit{sharp}_R(c)$. ●

As already motivated by definition 6.8, the interesting part is to make a meaningful statement concerning the quality of classifications instead of single classes. Accordingly, we define:

Sharpness (Classification)

Definition 6.10 — Sharpness (Classification).
The *sharpness* or *accuracy* of the R-approximation of a classification $\mathfrak{c}$ is defined as

$$\mathit{sharp}_R(\mathfrak{c}) \quad := \quad \frac{\sum_{i \in \mathbf{k}} |\llbracket R \rrbracket c_i|}{\sum_{i \in \mathbf{k}} |\llparenthesis R \rrparenthesis c_i|} \tag{6.20}$$

$$\mathit{utility}_R(\mathfrak{c}) \quad := \quad \frac{\sum_{i \in \mathbf{k}} |\llbracket R \rrbracket c_i|}{|s|} \tag{6.21}$$

While $\mathit{sharp}_R(\mathfrak{c})$ describes the ratio of sure knowledge to vague knowledge, $\mathit{utility}_R(\mathfrak{c})$ describes the percentage of correctly R–classifiable objects in s wrt $\mathfrak{c}$. ●

Where there are rough boundaries, there is vague membership, too: Let there be a concept $c \subseteq s$ and an object $x \in s$. Then, "$\in$" is a family of equivalence relations. This becomes clear when defining membership through characteristic function (instead of the other way round): If the characteristic function $\chi(c)$ is a feature in $\mathbf{F}$ then $\in_c$ is an equivalence relation induced by it.[5] It is clear that rough membership is always membership with respect to R–approximations. We write:

$$\begin{array}{rcccl} x \in_{\llbracket R \rrbracket} c & :\Longleftrightarrow & \llbracket R \rrbracket c.x & \Longleftrightarrow & x \in \llbracket R \rrbracket c \\ x \in_{\llparenthesis R \rrparenthesis} c & :\Longleftrightarrow & \llparenthesis R \rrparenthesis c.x & \Longleftrightarrow & x \in \llparenthesis R \rrparenthesis c \end{array} \tag{6.22}$$

[5] The entire family of $\in$-relations then is the set of relations induced by all $\{\chi(c) : c \in \wp(s)\}$.

This way, we can reformulate the relation between approximations pointwise

$$\llbracket R\rrbracket c \subseteq c \subseteq \llparenthesis R\rrparenthesis c \quad \text{if and only if} \quad \llbracket R\rrbracket c.x \Longrightarrow x \in c \Longrightarrow \llparenthesis R\rrparenthesis c.x$$

and it follows immediately that

$$x \in_{\llbracket R\rrbracket} c \Longrightarrow x \in c \Longrightarrow x \in_{\llparenthesis R\rrparenthesis} c \tag{6.23}$$

Example 6.6 Let us take another look at our example world in $\mathfrak{K}$ as defined in example 6.2:

$$\begin{array}{rcl} s &=& \{\bullet, \blacksquare, \blacktriangle, \textcolor{darkgray}{\bullet}, \textcolor{darkgray}{\blacksquare}, \textcolor{darkgray}{\blacktriangle}, \textcolor{lightgray}{\bullet}, \textcolor{lightgray}{\blacksquare}, \textcolor{lightgray}{\blacktriangle}\} \\ c &=& \{\blacktriangle, \textcolor{lightgray}{\blacksquare}, \textcolor{lightgray}{\blacktriangle}, \textcolor{lightgray}{\bullet}\} \end{array}$$

with relations $\mathbf{R} = \{R_{color}, R_{shape}, R_{polygon}\}$. Using our results from example 6.5, we find that

$$\begin{array}{rrlll} & \llparenthesis R_{colour}\rrparenthesis c.\blacksquare & \text{because} \quad \blacksquare \in & \llparenthesis R_{colour}\rrparenthesis & \{\blacktriangle, \textcolor{lightgray}{\blacksquare}, \textcolor{lightgray}{\blacktriangle}, \textcolor{lightgray}{\bullet}\} \\ \text{but} & \neg\llbracket R_{colour}\rrbracket c.\blacksquare & \text{because} \quad \blacksquare \notin & \llbracket R_{colour}\rrbracket & \{\blacktriangle, \textcolor{lightgray}{\blacksquare}, \textcolor{lightgray}{\blacktriangle}, \textcolor{lightgray}{\bullet}\} \quad = \{\textcolor{lightgray}{\blacksquare}, \textcolor{lightgray}{\blacktriangle}, \textcolor{lightgray}{\bullet}\} \end{array}$$

Similarly,

$$\begin{array}{rrlll} & \llparenthesis R_{shape}\rrparenthesis c.\textcolor{lightgray}{\bullet} & \text{because} \quad \textcolor{lightgray}{\bullet} \in & \llparenthesis R_{shape}\rrparenthesis & \{\blacktriangle, \textcolor{lightgray}{\blacksquare}, \textcolor{lightgray}{\blacktriangle}, \textcolor{lightgray}{\bullet}\} \quad = s \\ \text{but} & \neg\llbracket R_{shape}\rrbracket c.\textcolor{lightgray}{\bullet} & \text{because} \quad \textcolor{lightgray}{\bullet} \notin & \llbracket R_{shape}\rrbracket & \{\blacktriangle, \textcolor{lightgray}{\blacksquare}, \textcolor{lightgray}{\blacktriangle}, \textcolor{lightgray}{\bullet}\} \quad = \{\} \end{array}$$

●

Similar considerations apply to rough set inclusion: The usual set notation $c' \subseteq c$ means that if some $x \in s$ is c'-ish, then it is c-ish, too: $x \in c' \Longrightarrow x \in c$. so if some x is R–roughly c'–ish, it means that x is an element of an R–approximation of c'–ish objects.

$$c' \subseteq_{\llbracket R\rrbracket} c \quad :\Longleftrightarrow \quad \llbracket R\rrbracket c' \subseteq \llbracket R\rrbracket c \tag{6.24}$$

$$c' \subseteq_{\llparenthesis R\rrparenthesis} c \quad :\Longleftrightarrow \quad \llparenthesis R\rrparenthesis c' \subseteq \llparenthesis R\rrparenthesis c \tag{6.25}$$

Exercise 6.7 ($\Diamond$) Does this mean that every roughly c'-ish x is R–roughly c–ish, too?

Example 6.7 In our example world, it holds that

$$\begin{array}{rcl} \llbracket R_{colour}\rrbracket c' &=& \llbracket R_{colour}\rrbracket \{\textcolor{lightgray}{\blacksquare}, \textcolor{lightgray}{\blacktriangle}, \textcolor{lightgray}{\bullet}\} = \{\textcolor{lightgray}{\blacksquare}, \textcolor{lightgray}{\blacktriangle}, \textcolor{lightgray}{\bullet}\} \\ &\subseteq& \llbracket R_{colour}\rrbracket \{\blacktriangle, \textcolor{lightgray}{\blacksquare}, \textcolor{lightgray}{\blacktriangle}, \textcolor{lightgray}{\bullet}\} = \{\textcolor{lightgray}{\blacksquare}, \textcolor{lightgray}{\blacktriangle}, \textcolor{lightgray}{\bullet}\} \\ &=& \llbracket R_{colour}\rrbracket c \\ && \text{and} \\ \llparenthesis R_{colour}\rrparenthesis c' &=& \llparenthesis R_{colour}\rrparenthesis \{\textcolor{lightgray}{\blacksquare}, \textcolor{lightgray}{\blacktriangle}, \textcolor{lightgray}{\bullet}\} = \{\textcolor{lightgray}{\blacksquare}, \textcolor{lightgray}{\blacktriangle}, \textcolor{lightgray}{\bullet}\} \\ &\subseteq& \llbracket R_{colour}\rrbracket \{\blacktriangle, \textcolor{lightgray}{\blacksquare}, \textcolor{lightgray}{\blacktriangle}, \textcolor{lightgray}{\bullet}\} = \{\blacktriangle, \textcolor{lightgray}{\blacksquare}, \textcolor{lightgray}{\blacktriangle}, \textcolor{lightgray}{\bullet}\} \\ &=& \llbracket R_{colour}\rrbracket c \end{array}$$

●

By mutual inclusion we can define rough equality: Upper and lower R-approximations of sets allow for a definition of at least two rough equality relations:

$$c =_{\llbracket R\rrbracket} c' :\Longleftrightarrow \llbracket R\rrbracket c = \llbracket R\rrbracket c' \quad \text{and} \quad c =_{\langle\!| R |\!\rangle} c' :\Longleftrightarrow \langle\!| R |\!\rangle c = \langle\!| R |\!\rangle c' \tag{6.26}$$

Again, R-rough equality is rather a property of R than a property of two rougly equal objects $x, y \in s$. The interesting thing about inclusion is not *set* inclusion. Of course it is nice to know, that some x is c-ish because it is c'-ish. But the really interesting thing is what it means for *different relations* to define such dependencies: Imagine that $\llbracket P\rrbracket c \subseteq \llbracket R\rrbracket c$. Then, obviously, R is able to identify *more* objects to be c-ish than P. In other words R appears to have more information about c.

Example 6.8 Recall example 6.7. Now we shall compare different *relations* instead of different sets. For more interesting results we chose another target concept c with

$$c := \{\blacksquare, \blacksquare, \blacksquare, \bullet\}$$

Then,

$$\begin{aligned}
\llbracket R_{shape}\rrbracket c &= \llbracket R_{shape}\rrbracket \{\blacksquare, \blacksquare, \blacksquare, \bullet\} = \{\blacksquare, \blacksquare, \blacksquare\} \\
&\supseteq \llbracket R_{polygon}\rrbracket \{\blacksquare, \blacksquare, \blacksquare, \bullet\} = \{\} \\
&= \llbracket R_{polygon}\rrbracket c
\end{aligned}$$

As one can see here, the finer and *smaller* relation based on knowledge about *shape* has a *larger* lower approximation of c than the lower approximation by way of *polygon.* Intuitively the reason is that a *larger* relation (such as $R_{polygon}$) creates *less* equivalence classes. Since $R_{shape} \subseteq R_{polygon}$, we know that every *shape*-basic category is a subset of a *polygon*-basic category. Therefore, the *polygon*-knowledge is coarser than the *shape*-knowledge. And the finer the knowledge, the smaller the steps we can make to approximate a concept. The *polygon*-knowledge is not fine enough to describe basic categories that are fully included by c, but *shape* is. For upper approximations, we obtain:

$$\begin{aligned}
\langle\!| R_{shape} |\!\rangle c &= \langle\!| R_{shape} |\!\rangle \{\blacksquare, \blacksquare, \blacksquare, \bullet\} = \{\blacksquare, \blacksquare, \blacksquare, \bullet, \bullet, \bullet\} \\
&\subseteq \langle\!| R_{colour} |\!\rangle \{\blacksquare, \blacksquare, \blacksquare, \bullet\} = \{\bullet, \bullet, \bullet, \blacksquare, \blacksquare, \blacksquare, \blacktriangle, \blacktriangle, \blacktriangle\} = s \\
&= \langle\!| R_{colour} |\!\rangle c
\end{aligned}$$

For the upper approximation, the converse is true: The finer the knowledge, the smaller the upper approximation. ●

We can conclude:

Knowledge and Concept Approximations
The finer a relation, the more basic categories. The more basic categories, the smaller the equivalence classes. The smaller the equivalence classes, the higher their descriptive power. The higher the descriptive power, the more knowledge. The more knowledge, the smaller the boundary region.

Exercise 6.8 (◇– ♦) Confirm all the examples! Determine $\llbracket \mathbf{P}\rrbracket c$ and $\langle\!| \mathbf{P} |\!\rangle c$ for several $\mathbf{P} \subseteq \mathbf{R}$ and various $c \subseteq s$.

6.3 Rough Knowledge Structures

The most important issue when understanding machine learning as compression is to delete redundant information. In most data collections there is redundancy—i.e. there is data which does not contribute to the information content of the entire set of data. However, redundancy has a huge advantage in every day life: Redundant data can be used to reconstruct information that would be lost given that a certain datum is missing.[6]

Example 6.9 Let us take a look at our standard example using the underlying information system $\mathfrak{I}$ with the target concept c as defined in example 6.6:

$x \in s$	id	f_{colour}	f_{shape}	$f_{polygon}$	$\chi(c)$
▲	1	▬	△	**1**	**1**
●	2	▬	○	**0**	**1**
■	3	▬	□	**1**	**0**
▲	4	▬	△	**1**	**1**
■	5	▬	□	**1**	**0**
●	6	▬	○	**0**	**0**
●	7	▬	○	**0**	**0**
■	8	▬	□	**1**	**1**
▲	9	▬	△	**1**	**0**

Now, through the eyes of a database engineer, why do we store so–called *oid*s in the *id*–column, if all the objects are unique anyway? Why do we store the $f_{polygon}$–information if it contains less information than f_{shape}? And why don't we just drop all the columns and use *id*instead? ●

Exercise 6.9 Compare the utility of all the features:

- ◊ Determine $H_c(f, s)$ for all $f \in \{id, f_{colour}, f_{shape}, f_{polygon}\}$!
- ♦ Determine $H_c(f, s)$ for $f := f_{colour} \times f_{shape}$!
- ♦♦ Determine $utility_{\mathbf{P}_i}(\mathfrak{c})$ for: $\mathbf{P}_1 = \{R_{colour}\}, \mathbf{P}_2 = \{R_{shape}\}, \mathbf{P}_3 = \{R_{polygon}\}, \mathbf{P}_4 = \mathbf{P}_1 \cup \mathbf{P}_2$ and $\mathbf{P}_5 = \mathbf{P}_1 \cup \mathbf{P}_3$

Let us now reconsider the knowledge base $\mathfrak{K}$. We concluded that knowledge (and the ability to discriminate different things from each other) can be considered to be an equivalence relation $\bar{\bar{\mathbf{R}}}$. Some concepts are definable, others are only roughly definable - and in some cases out knowledge is maximal in the sense that we can discriminate every single object from every other single object:

$$s/\bar{\bar{\mathbf{P}}} = \{\{x\} : x \in s\}$$

In any case, there may be many different ways to define some $\mathbf{P} \subseteq \mathbf{R}$ all of which create the same quotient set. But as soon as $\mathbf{P}$ is a proper subset of $\mathbf{R}$ and the quotient sets are the same, then it seems it did not hurt to drop some "information" from $\mathbf{R}$.

[6] Can you reconstruct a text from an ASCII-file where one byte is missing? Can you do the same for the same text in zipped form?

Relations that do not add to the knowledge are *dispensable*, and, in fact, they are even *redundant*. So if $\bar{\bar{\mathbf{R}}}$ does not change when removing R from $\mathbf{R}$, then R does *not contribute* any elementary class that is not already describable in terms of intersections of basic categories from $\mathbf{R} - \{R\}$. It is, in other words, *redundant* knowledge.

Dispensability, Reducibility

Definition 6.11 — Dispensability, Reducibility.
We call $R \in \mathbf{R}$ *dispensable* in $\mathbf{R}$, if

$$\bar{\bar{\mathbf{R}}} = \overline{\overline{\mathbf{R} - \{R\}}} \tag{6.27}$$

Otherwise, R is *indispensable*. $\mathbf{R}$ is called *irreducible*, if R is indispensable in $\mathbf{R}$ for all $R \in \mathbf{R}$, otherwise R is called *reducible* or *redundant*. ●

Irreducibility of $\mathbf{R}$ means that there is no $R \in \mathbf{R}$, which we can remove from $\mathbf{R}$ without changing $\bar{\bar{\mathbf{R}}}$ (i.e. losing "crispness"). Redundant sets of knowledge are reducible by deleting dispensable relations.

Example 6.10 In our example world,
$\mathbf{P} = \{R_{shape}, R_{colour}\} \subset \mathbf{R}$ is irreducible, since both R_{shape} and R_{colour} are indispensable:

$$\begin{aligned} s/\bar{\bar{\mathbf{P}}} &= \{\{\bullet\},\{\blacktriangle\},\{\blacksquare\},\{\bullet\},\{\blacktriangle\},\{\blacksquare\},\{\bullet\},\{\blacktriangle\},\{\blacksquare\}\} \\ s/R_{shape} &= \{\{\bullet,\bullet,\bullet\},\{\blacksquare,\blacksquare,\blacksquare\},\{\blacktriangle,\blacktriangle,\blacktriangle\}\} \\ s/R_{colour} &= \{\{\bullet,\blacktriangle,\blacksquare\},\{\bullet,\blacktriangle,\blacksquare\},\{\bullet,\blacktriangle,\blacksquare\}\} \end{aligned}$$

Since $s/\bar{\bar{\mathbf{P}}} \neq s/R_{shape}$ and $s/\bar{\bar{\mathbf{P}}} \neq s/R_{colour}$, both relations are indispensable in $\mathbf{P}$. Hence, $\mathbf{P}$ is irreducible.
In contrast to this, $\mathbf{R}$ is redundant:

$$\begin{aligned} s/\bar{\bar{\mathbf{R}}} = &= \{\{\bullet\},\{\blacktriangle\},\{\blacksquare\},\{\bullet\},\{\blacktriangle\},\{\blacksquare\},\{\bullet\},\{\blacktriangle\},\{\blacksquare\}\} \\ &= s/\bar{\bar{\mathbf{P}}} \end{aligned}$$

●

Exercise 6.10 ♦ For the example above, let $x = \{\blacktriangle, \blacktriangle, \blacksquare\}$.—Determine $\langle\!|R|\!\rangle_{color}x$, $\langle\!|R|\!\rangle_{shape}x$, $[\![R]\!]_{color}x$, and $[\![R]\!]_{shape}x$.

There are relations that seem to be dispensable since they do not contribute to our ability to discriminate things (i.e. our knowledge). Thus it seems a good idea to *reduce* a knowledge base by discarding all dispensable relations. Accordingly, what remains after deleting redundant knowledge is a *reduct*:

Reducts

Definition 6.12 — Reducts.
We call $\mathbf{P}$ ($\mathbf{P} \subseteq \mathbf{R}$) a *reduct* of a set of relations $\mathbf{R}$, if $\mathbf{P}$ is irreducible and carries the same indiscernability knowledge as $\mathbf{R}$. We write:

$$\mathbf{P} \in \mathrm{Red}(\mathbf{R}) :\Longleftrightarrow \begin{cases} \mathbf{P} \text{ is irreducible} \\ \bar{\bar{\mathbf{P}}} = \bar{\bar{\mathbf{R}}}. \end{cases} \tag{6.28}$$

In general, reducts are *not unique*. ●

Since the uniqueness property does not hold in general, there may exist several reducts for a given family $\mathbf{R}$ of equivalence relations.

Example 6.11 Let us first take a look at our standard example $\mathfrak{K}$: For $\mathbf{R} = \{R_{shape}, R_{colour}, R_{polygon}\}$, $\mathbf{P} = \{R_{shape}, R_{colour}\} \in \mathrm{Red}(\mathbf{R})$. However, it is the only reduct: $\mathrm{Red}(\mathbf{R}) = \{\mathbf{P}\}$. For the information system $\mathfrak{I}$ and the induced knowledge base $\mathfrak{K}'$, we have $\mathbf{R}' = \mathbf{R} \cup \{\mathit{1}\}$. Then $\mathbf{R}'$ has two reducts:

$$\mathrm{Red}(\mathbf{R}') = \{\{R_{colour}, R_{shape}\}, \{\mathit{1}\}\} \tag{6.29}$$

Exercise 6.11

- ◊ Prove equation (6.29) in example 6.11!
- ♦ Show that $\mathbf{R}$ and $\{R_{colour}, R_{shape}, \mathit{1}\}$ are not reducts of $\mathbf{R}$!
- ♦ Prove that if $|\mathrm{Red}(\mathbf{R})| > 1$, then $\bigcup \mathrm{Red}(\mathbf{R}) \notin \mathrm{Red}(\mathbf{R})$.
- ♦ Show that for $\mathrm{Red}(\mathbf{R}) = \{\mathbf{P}_i : i \in \mathbf{n}\}$ and $\forall i, j \in \mathbf{n} : \mathbf{P}_i = \mathbf{P}_j \longrightarrow i = j$, their union $\mathbf{Q} := \{R : \exists i \in \mathbf{n} : R \in \mathbf{P}_i\} = \bigcup \mathrm{Red}(\mathbf{R})$ is not a reduct: $\mathbf{Q} \notin \mathrm{Red}(\mathbf{R})$!

So if there are several reducts for $\mathbf{R}$, say, $\mathbf{P}$ and $\mathbf{Q}$, the knowledge in $\mathbf{R}$ can be expressed by different subsets of knowledge (recall example 6.4). Usually, we would expect the knowledge in our knowledge base to consist of three different kinds of knowledge regarding their "importance": There is ...

- some kind of "basic" knowledge without which we were hopelessly lost,
- some kind of "detail" knowledge which, when present, suffices to speak about the world up to a desired level of detail,
- and additional, abundant knowledge that increases our eloquence to speak about the world but which actually does not contribute to the level of detail (and, hence, is redundant).

To decide whether a relation belongs to one or another set of knowledge depends on what we want to be able to express and which other parts of knowledge we already have. This is why there are reducts and why they are not unique.
Arguing from the point of view of conceptual knowledge modeling, we would assume there is some fundamental knowledge that all semantically equivalent formulations of the knowledge should share. If there is such *core* knowledge, it should be contained in all reducts: There seem to be relations that are simply absolutly indispensable and they occur in every reduct—even though they may not provide sufficient knowledge on their own.

Core

Definition 6.13 — Core.
The set of relations in $\mathbf{R}$ which occur in *every* reduct $\mathbf{P} \in \mathrm{Red}(\mathbf{R})$ is called the *core* of $\mathbf{R}$:

$$\begin{aligned} \mathrm{Cor}(\mathbf{R}) &:= \{R \in \mathbf{R} : \mathbf{P} \in \mathrm{Red}(\mathbf{R}) \longrightarrow R \in \mathbf{P}\} && (6.30) \\ &= \bigcap_{\mathbf{P} \in \mathrm{Red}(\mathbf{R})} \mathbf{P} = \bigcap \mathrm{Red}(\mathbf{R}) && (6.31) \end{aligned}$$

If the core is not empty, then there exist relations that are indispensible in any attempt to describe a concept. If the core is empty, it is so because there do not exist any reducts, or because there are several disjoint sets of relations one of which must appear in any reduct (see example 6.11). ●

We conclude: For any $\mathbf{Q} \in \text{Red}(\mathbf{R})$ it holds that $\text{Cor}(\mathbf{R}) \subseteq \mathbf{Q} \subseteq \mathbf{R}$ and if $\mathbf{Q}$ is irreducible, then any $\mathbf{P} \subseteq \mathbf{Q}$ is irreducible, too.

Rough Knowledge Structres
Rough knowledge is what one has if one is able to roughly describe new concepts in terms of old ones. The roughness of the description depends on the knowledge. Knowledge discovery means to *reduce* data by discarding redundancy. Some parts of the knowledge are dispensable in general, others are not because they are required in any reduct. A small part of knowledge may be essential and absolutely irreducible.

Example 6.12 Consider the set s of differently coloured geometric shapes and the set of relations as in example 6.11. We examine subsets $\mathbf{P}_i \subseteq \mathbf{R}$:

i	$\mathbf{P}_i$	$s/\bar{\bar{\mathbf{P}}}_i$
1	$\mathbf{R}$	{{●}, {▲}, {■}, {●}, {▲}, {■}, {●}, {▲}, {■}}
2	$\{\mathit{1}\}$	{{●}, {▲}, {■}, {●}, {▲}, {■}, {●}, {▲}, {■}}
3	$\{R_{colour}, R_{shape}, R_{polygon}\}$	{{●}, {▲}, {■}, {●}, {▲}, {■}, {●}, {▲}, {■}}
3	$\{R_{colour}, R_{shape}\}$	{{●}, {▲}, {■}, {●}, {▲}, {■}, {●}, {▲}, {■}}
4	$\{R_{colour}, R_{polygon}\}$	{{●}, {▲, ■}, {●}, {▲, ■}, {●}, {▲, ■}}
5	$\{R_{colour}\}$	{{●, ▲, ■}, {●, ▲, ■}, {●, ▲, ■}}

$\mathbf{P}_1, \mathbf{P}_2, \mathbf{P}_3$ and $\mathbf{P}_4$ are reducts of $\mathbf{R}$; $\mathbf{P}_4$ and $\mathbf{P}_5$ are not. The core is the intersection of reducts: $\text{Cor}(\mathbf{R}) = \bigcap \text{Red}(\mathbf{R}) = \{R_{color}\}$. ●

Reducts and Cores are powerful concepts, but what we need is a method to compare expressiveness of relations with respect to (i.e. *relative* to) a given classification. Since we agreed that classes of a classification are disjoint, a classifiction is just another partition—which in turn can be considered to be the result of building the quotient of another equivalence relation.[7] Therefore, relative expressiveness is something that we can measure for any pair of (sets of) equivalence relations.

6.4 Relative Knowledge

A special kind of structured knowledge
It depends on the concept which parts of knowledge are redundant and which parts are required or essential. Usually, we are given such a concept c or a classification $\mathfrak{c}$. Much more interesting are the following two questions:

- If there are several approximations of a concept by several different reducts: which one shall we choose?
- Given two sets of relations, how can we describe their potential to describe the knowledge in their respective counterpart?

Do different parts of knowledge somehow *depend* on each other?

[7] And, of course by any other *set* $\mathbf{S}$ of equivalence relations and the quotient $s/\bar{\bar{\mathbf{S}}}$.

For a given information system or knowledge base all we did so far was to fish for subsets of knowledge satisfying conditions of necessity or sufficiency. This process was the first step away from a pointwise characterisation of data towards a relational one: Knowledge is indispensable if it is required to tell different objects from each other—or, conceptually, if it cannot be reduced without increasing coarseness.
We shall now examine the *relationship* between two different sets of knowledge with respect to their expressive power: Is one part of knowledge sufficient to describe another part of knowledge?

Example 6.13 Let there be a concept $c \subseteq s$. Then, we can assume there is an equivalence relation R_c (of index 2) induced by the characteristic function $\chi(c)$. It creates a classification $\mathfrak{c} = \{c, \overline{c}\} := s/R_c$. Now imagine a set $\mathbf{P} \subseteq \mathbf{R}$. In machine learning $\chi(c)$ is called the target function t_c which we want to approximate. Of course, lower and upper approximations come in quite handy at this point.
In order to pick the best fitting hypothesis from hypothesis space, we need to find a set $\mathbf{P}$ such that we can construct c from $s/\overline{\overline{\mathbf{P}}}$ as accurate as possible. ●

The first thing we need then is to determine the region in c (and thus, the subdomain of R_c), for which $\mathbf{P}$ delivers correct predictions.

Relative Positive Knowledge

Definition 6.14 — Relative Positive Knowledge.
Let there be two equivalence relations P and Q. The *P-positive region of Q (on s)* is the union set of all P-elementary classes that belong to Q-elementary classes.

$$\begin{aligned} [\![P \trianglelefteq Q]\!]s &:= \bigcup_{q \in s/Q} [\![P]\!]q = \bigcup_{\mathfrak{q}} [\![P]\!]q_i && (6.32) \\ &= [\![P]\!]q_0 \cup [\![P]\!]q_1 \cup \cdots \cup [\![P]\!]q_{k-1} && (6.33) \end{aligned}$$

where s/Q is a partition or a classification of objects in s with respect to knowledge Q: $\mathfrak{q} = \{q_0, q_1, \ldots, q_{k-1}\} = s/Q$. The P-positive region is the set of objects for which by P we can positively confirm their membership to Q-elementary classes. ●

If rough set theory is new for you (or if you are already familiar with rough set theory and the additional symbol $\trianglelefteq$ occurs a bit odd to you), then try to read it out loud as the word "*for*":

Relative Positive Knowledge
Technically speaking, the P-positive region of Q, $[\![P \trianglelefteq Q]\!]s$, is the (disjoint) union set of all P-lower approximations of all $c \in s/Q$.
Its meaning is that $[\![P \trianglelefteq Q]\!]s$ is the set of all objects where *P–knowledge suffices for modelling Q–knowledge.*

Let us take a closer look at this: We always considered approximations with respect to a given or otherwise somehow predefined concept c or a set $\mathfrak{c}$ of such classes. By comapring P with Q we examine P–approximations with respect to "dynamically defined" classifications $\mathfrak{q} = \{q_0, q_1, q_2, \ldots, q_{k-1}\} = s/Q$ as defined by Q.

Exercise 6.12 (◊) In most textbooks on rough sets, $[\![P \trianglelefteq Q]\!]s$ is denoted $Pos_P(Q)$. Let there be an arbitrary classification $\mathfrak{c}$ of s. Show that

$$Pos_P(Q) = \bigcup_{c \in \mathfrak{c}} [\![P \trianglelefteq Q]\!]c$$

to prove that our definition allows a more detailed usage and *relative* usage of the term "P-positive region of Q".

Exercise 6.13 (◊-♦♦) Prove the following equations:

$$\begin{aligned} [\![P \trianglelefteq Q]\!]s &= \bigcup_{i \in \mathbf{k}} \{x : [x]_P \subseteq q_i\} \\ &= \bigcup_{i \in \mathbf{k}} \{x : [\![P]\!]q_i.x \longrightarrow [\![P]\!]q_i \subseteq q_i\} \end{aligned}$$

The last equation of the preceding exercise appears to be trivial: if something is element of a lower set approximation, then it is an element of this set. But it is *not* that easy. q_i is defined in terms of Q while $[\![P]\!]q_i$ is defined in terms of P (or, to be precise, "in terms of P in terms of Q"). Herein, P and Q are independent in the sense that they do not "know" about each other. Take a look at figure 6.5: Let the regions defined by the paraxial rectangles be the equivalence classes induced by P and the rotated rectangles be the Q-classes. Now, a single object $x \in s$ can be an element of $[\![P \trianglelefteq Q]\!]s$, $[\![Q \trianglelefteq P]\!]s$, both or none of them. We take a closer look at these cases:

Example 6.14 What does it mean for some x (not) to be member of a positive region?

(a) $x \in [\![P \trianglelefteq Q]\!]s$ means that x can positively be classified into a Q-class by P-knowledge. This means that $[x]_P$ must be a subset of $[x]_Q$. In figure 6.5, this is true for all the black dots •.

(b) The dual case is $[\![Q \trianglelefteq P]\!]s.x$; i.e. it is the set of all Q-equivalent objects that are R–equivalent, too. These are the grey objects •.

(c) There are objects which belong to both sets: an example is ••. the paraxial class is a subset of a rotated class whih in turn is a subset of a paraxial class again.

(d) Finally, there is the set of all objects (∘) for which neither P– nor Q–knowledge is sufficient to describe the corresponding counterpart.

●

At this point, it becomes clear why we introduced the dotted membership notation. The equation

$$[\![P]\!]c.x \longrightarrow [\![Q]\!]c.x$$

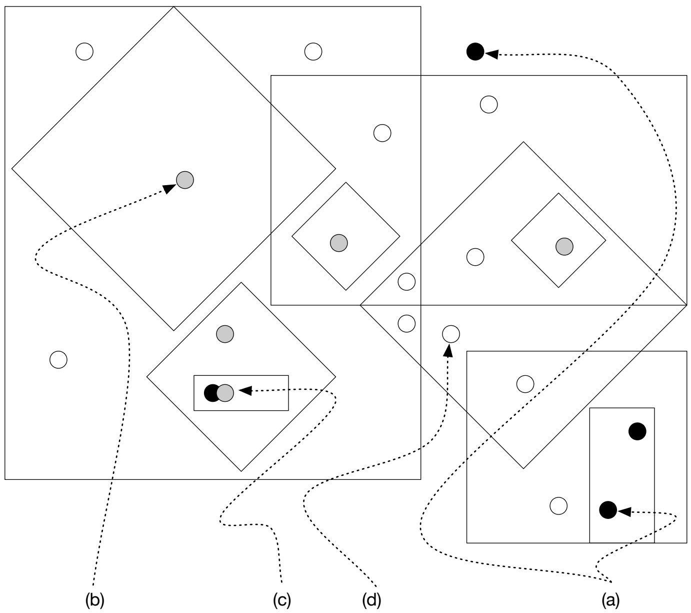


Figure 6.5: The P-positive region of Q, $[\![P \trianglelefteq Q]\!]s$

reads: "if x is c-ish according to our knowledge P, then it is c-ish acording to Q, too". If this implication is true for some x, x is element of the P-positive region of Q and so are all objects $y \in s$ that are P-equivalent:

$$[\![P \trianglelefteq Q]\!]c.x \iff [\![P]\!]c.x \longrightarrow [\![Q]\!]c.x \tag{6.34}$$

Of course, we can lift the above definitions from simple sets s to classifications $\mathfrak{c}$:

$$[\![P \trianglelefteq Q]\!]\mathfrak{c} \;\; := \;\; \bigcup_{c \in \mathfrak{c}} [\![P \trianglelefteq Q]\!]c \tag{6.35}$$

But what if P is an indisernability relation built from a set $\mathbf{P}$ of relations? And, similarly, what if $Q = \bar{\bar{\mathbf{Q}}}$? Can we lift our defintion to sets of relations as well?—Yes! Again, we generalise our idea to sets of equivalence relations by using their respective indiscernability relations:

$$[\![\mathbf{P} \trianglelefteq \mathbf{Q}]\!]s := [\![\bar{\bar{\mathbf{P}}} \trianglelefteq \bar{\bar{\mathbf{Q}}}]\!]s \tag{6.36}$$

Then again, $[\![\mathbf{P} \trianglelefteq \mathbf{Q}]\!]s$ is the set of objects where $\mathbf{P}$ can be used *for* describing $\mathbf{Q}$-knowledge. The direction of "$\trianglelefteq$" is motivated by the following equivalent definition:

$$[\![\mathbf{P} \trianglelefteq \mathbf{Q}]\!]s \;\; := \;\; \left\{ x \in s : \exists q \in s/\bar{\bar{\mathbf{Q}}} : [x]_{\bar{\bar{\mathbf{P}}}} \subseteq q \right\}; \tag{6.37}$$

i.e. $\trianglelefteq$ is defined in terms of $\subseteq$.

Exercise 6.14 (♦) Prove the equivalence of equation (6.37) and equation (6.36)!

Now we can try to lift the definition of dispensability to relative dispensability: A relation R in $\mathbf{P}$ is dispensable with respect to $\mathbf{Q}$, if the $\bar{\bar{\mathbf{P}}}$ positive region of $\bar{\bar{\mathbf{Q}}}$ remains the same for $\overline{\overline{\mathbf{P}-\{R\}}}$:

Relative Indispensability

Definition 6.15 — Relative Indispensability.
Let there be two families of equivalence relations $\mathbf{P}$ and $\mathbf{Q}$. A relation $R \in \mathbf{P}$ is called $\mathbf{Q}$–*dispensable*, if:

$$\llbracket \mathbf{P} \trianglelefteq \mathbf{Q} \rrbracket s = \llbracket \mathbf{P} - \{R\} \trianglelefteq \mathbf{Q} \rrbracket s \tag{6.38}$$

Otherwise, it is called *indispensable*. ●

Considering the knowledge defined by the indiscernability relation on equivalence relations $\mathbf{Q}$, we examine a set of equivalence relations $\mathbf{P}$. If the set of objects that can be correctly $\bar{\bar{\mathbf{Q}}}$–classified by $\bar{\bar{\mathbf{P}}}$ remains the same if we discard R from $\mathbf{P}$, then R is redundant in our attempt to $\mathbf{Q}$–classify objects using knowledge $\mathbf{P}$. The notion of dispensability motivated the concept of depedence (c.f. definition 6.11). Therefore, relative dispensability gives rise to relative irreducability. If a relation is indispensable in a set $\mathbf{P}$ with respect to a set $\mathbf{Q}$, $\mathbf{P}$ cannot reduced any further without losing information. Speaking in terms of objects in our universe, reducing $\mathbf{P}$ implies loss of knowledge with respect to $\mathbf{Q}$, if $\llbracket \mathbf{S} \trianglelefteq \mathbf{Q} \rrbracket s \subset \llbracket \mathbf{P} \trianglelefteq \mathbf{Q} \rrbracket s$ for any $\mathbf{S} \subset \mathbf{P}$.

Relative Implication

Definition 6.16 — Relative Implication.
Let there be three sets of relations, $\mathbf{P}, \mathbf{Q}$ and $\mathbf{R}$. $\mathbf{P}$ is called $\mathbf{Q}$–*irreducible*, iff every $R \in \mathbf{P}$ is $\mathbf{Q}$–indispensable:

$$\forall R \in \mathbf{P} : \llbracket \mathbf{P} \trianglelefteq \mathbf{Q} \rrbracket s \subset \llbracket (\mathbf{P} - \{R\}) \trianglelefteq \mathbf{Q} \rrbracket s \tag{6.39}$$

$\mathbf{P}$ $\mathbf{Q}$–*implies* $\mathbf{R}$, if the $\mathbf{R}$-positive region for $\mathbf{Q}$ is included in the $\mathbf{P}$-positive region for $\mathbf{Q}$.

$$\llbracket \mathbf{R} \trianglelefteq \mathbf{Q} \rrbracket s \subseteq \llbracket \mathbf{P} \trianglelefteq \mathbf{Q} \rrbracket s \tag{6.40}$$

We then write $\mathbf{P} \overset{\mathbf{Q}}{\supset} \mathbf{R}$. Note the inverted notation: The positive region of $\mathbf{R}$ is smaller than (i.e. included in) the positive region of $\mathbf{P}$. This means, that $\mathbf{P}$ can explain *more* than $\mathbf{R}$ or that $\mathbf{R}$ is *weaker* than $\mathbf{P}$. Therefore, $\mathbf{P}$ *implies* $\mathbf{Q}$. ●

Note that relative implication is not defined in terms of a subset relation between $\mathbf{P}$ and $\mathbf{R}$ but rather on their respective positive regions with respect to $\mathbf{Q}$. Therefore, and this is why it is called *implication*, $\supset$ is a property of the knowledge encoded in sets of relations where the amount of knowledge is measured extensionally by the sets of objects that can be classified correctly. In literature on Rough Sets there is no such thing as *implication* but *dependency*. It is written $\Rightarrow$ and defined as follows: $\mathbf{P} \Rightarrow \mathbf{Q} :\Longleftrightarrow \bar{\bar{\mathbf{P}}} \subseteq \bar{\bar{\mathbf{Q}}}$ and spoken as "$\mathbf{Q}$ depends on $\mathbf{P}$".

Exercise 6.15 (♦) Prove that $\mathbf{P} \Rightarrow \mathbf{Q} \iff \mathbf{P} \overset{\top}{\supset} \mathbf{Q}$!

Since $\mathbf{P} \preceq \mathbf{P} - \{R\}$, it holds that $[\![\mathbf{P} - \{R\} \trianglelefteq \mathbf{Q}]\!]s$ is a proper subset of $[\![\mathbf{P} \trianglelefteq \mathbf{Q}]\!]s$ if and only if $\mathbf{P}$ is $\mathbf{Q}$–independent.

Exercise 6.16 (◊) Prove.

Example 6.15 Let us, again, consider the knowledge base from example 6.9. Then, for $s' = \{\bullet, \blacktriangle, \blacksquare\} \subseteq s$ we find:

$$\begin{aligned}
[\![R_{colour} \trianglelefteq R_{shape}]\!] \quad s &= \textstyle\bigcup_{q \in s/R_{shape}} [\![R_{colour}]\!] q \\
&= [\![R_{colour}]\!]\,\{\bullet, \bullet, \bullet\} \cup [\![R_{colour}]\!]\,\{\blacktriangle, \blacktriangle, \blacktriangle\} \cup \\
&\quad\ [\![R_{colour}]\!]\,\{\blacksquare, \blacksquare, \blacksquare\} = \\
&= \{\} \cup \{\} \cup \{\} = \emptyset
\end{aligned}$$

$$\begin{aligned}
[\![R_{colour} \trianglelefteq R_{shape}]\!] \quad s' &= [\![R_{colour}]\!]\,\{\bullet\} \cup [\![R_{colour}]\!]\,\{\blacktriangle\} \cup [\![R_{colour}]\!]\,\{\blacksquare\} \\
&= \{\bullet\} \cup \{\blacktriangle\} \cup \{\blacksquare\} = \{\bullet, \blacktriangle, \blacksquare\} \\
&= s'
\end{aligned}$$

$$\begin{aligned}
[\![R_{polygon} \trianglelefteq R_{shape}]\!] \quad s &= [\![R_{polygon}]\!]\,\{\bullet, \bullet, \bullet\} \cup [\![R_{polygon}]\!]\,\{\blacktriangle, \blacktriangle, \blacktriangle\} \cup \\
&\quad\ [\![R_{polygon}]\!]\,\{\blacksquare, \blacksquare, \blacksquare\} \\
&= \{\bullet, \bullet, \bullet\}
\end{aligned}$$

$$\begin{aligned}
[\![R_{polygon} \trianglelefteq R_{shape}]\!] \quad s' &= [\![R_{polygon}]\!]\,\{\bullet\} \cup [\![R_{polygon}]\!]\,\{\blacktriangle\} \cup [\![R_{polygon}]\!]\,\{\blacksquare\} \\
&= \{\bullet, \blacktriangle, \blacksquare\} \\
&= s'
\end{aligned}$$

$$\begin{aligned}
[\![R_{shape} \trianglelefteq R_{polygon}]\!] \quad s &= [\![R_{shape}]\!]\,\{\bullet, \bullet, \bullet\} \cup [\![R_{shape}]\!]\,\{\blacktriangle, \blacktriangle, \blacktriangle, \blacksquare, \blacksquare, \blacksquare\} \\
&= \{\bullet, \bullet, \bullet, \blacktriangle, \blacktriangle, \blacktriangle, \blacksquare, \blacksquare, \blacksquare\} = s
\end{aligned}$$

$$\begin{aligned}
[\![R_{shape} \trianglelefteq R_{polygon}]\!] \quad s' &= [\![R_{shape}]\!]\,\{\bullet\} \cup [\![R_{shape}]\!]\,\{\blacktriangle, \blacksquare\} \\
&= \{\bullet, \blacktriangle, \blacksquare\} \\
&= s'
\end{aligned}$$

Example 6.16 Let us take a closer look at dependent sets of knowledge:

$$\begin{aligned}
[\![\{R_{shape}, R_{colour}\} \trianglelefteq \mathit{1}]\!]s &= s \\
&\supset \\
\{\bullet, \bullet, \bullet\} &= [\![\{R_{polygon}, R_{colour}\} \trianglelefteq \mathit{1}]\!]s
\end{aligned}$$

This shows that $\{R_{shape}, R_{colour}\}$ *1*–implies $\{R_{polygon}, R_{colour}\}$:

$$\{R_{shape}, R_{colour}\} \overset{\mathit{1}}{\supset} \{R_{polygon}, R_{colour}\}\,.$$

And, again, dependency and indispensability gives rise to the definition of reducts:

Example 6.17 In our running example, it holds that

$$[\![\mathbf{R} \trianglelefteq \mathit{1}]\!]s = [\![\mathbf{R} - \{R_{polygon}\} \trianglelefteq \mathit{1}]\!]s \tag{6.41}$$

This means that $R_{polygon}$ is $\mathit{1}$-dispensable and since $\mathit{1} \preceq R$ for all $R \in \mathbf{R}$, $R_{polygon}$ is dispensable. Equivalently, there exists a reduct of $\mathbf{R}$ without $R_{polygon}$ in it. ●

Accordingly, we define:

Relative Reducts and Relative Cores

Definition 6.17 — Relative Reducts and Relative Cores.
Let there be three sets of equivalence relations $\mathbf{P}, \mathbf{Q}, \mathbf{R}$ and $\mathbf{R} \subseteq \mathbf{P}$. $\mathbf{R}$ is called a $\mathbf{Q}$–*relative reduct* of $\mathbf{P}$, if and only if:

1. $[\![\mathbf{R} \trianglelefteq \mathbf{Q}]\!]s = [\![\mathbf{P} \trianglelefteq \mathbf{Q}]\!]s$
2. $\mathbf{R}$ is $\mathbf{Q}$–independent

The set of all $\mathbf{Q}$–reducts of $\mathbf{P}$ is denoted $\mathrm{Red}(\mathbf{P} \trianglelefteq \mathbf{Q})$. Then,

$$\mathrm{Cor}(\mathbf{P} \trianglelefteq \mathbf{Q}) \quad := \quad \bigcap \mathrm{Red}(\mathbf{P} \trianglelefteq \mathbf{Q}) \tag{6.42}$$

is called the $\mathbf{Q}$-*relative core of* $\mathbf{P}$. ●

The above definition can be broken down as follows: A $\mathbf{Q}$-relative reduct is a subset $\mathbf{R}$ of $\mathbf{P}$ such that the expressivenness of $\mathbf{P}$ with respect to $\mathbf{Q}$ does not suffer from the loss of relations missing in $\mathbf{R}$.
First, it is said that $[\![\mathbf{R} \trianglelefteq \mathbf{Q}]\!]s = [\![\mathbf{P} \trianglelefteq \mathbf{Q}]\!]s$. That's quite simple—$\mathbf{R}$ has the same positive region of $\mathbf{Q}$ as $\mathbf{P}$ has. They are, pointwise, equivalent with respect to $\mathbf{Q}$: an object of s belongs to the $\mathbf{P}$–positive region of $\mathbf{Q}$ if and only if it belongs to the $\mathbf{R}$–positive region of $\mathbf{Q}$. Speaking in terms of extensional set definitions it does not matter at all whether we choose $\mathbf{P}$ or $\mathbf{R}$: both have the same expresive power and even if they do not suffice to describe all the $\mathbf{Q}$-knowledge, they can correctly describe exactly the same subset of objects in s.
Secondly, $\mathbf{Q}$–independency of $\mathbf{R}$ requires that we cannot drop any relation $R \in \mathbf{R} \subseteq \mathbf{P} \subseteq \mathbf{R}$ without losing information relative to $\mathbf{Q}$-knowledge: $\mathbf{R}$ is $\mathbf{Q}$–independent, if every $R \in \mathbf{R}$ is $\mathbf{Q}$-indispensable. It means that for all $R \in \mathbf{R}$,

$$[\![\mathbf{R} \trianglelefteq \mathbf{Q}]\!]s \neq [\![\mathbf{R} - \{R\} \trianglelefteq \mathbf{Q}]\!]s$$

In other words, $\mathbf{R}$ cannot be reduced any further and it is a smallest subset of $\mathbf{P}$ still capable of describing $\mathbf{Q}$ to the same extent as $\mathbf{P}$.

Exercise 6.17 (♦) Show that $\mathrm{Red}(\mathbf{P} \trianglelefteq \mathbf{P}) = \mathrm{Red}(\mathbf{P})$.

It should have become clear that rough set theory is concerned with *relations* and their expressive power on *actual* data rather than sets of objects with terminologic information. We can define a relation's dispensability or utility with respect to a given classification, we can define reducts and cores with respect to given classification information. In other words, our toolbox of rough set operators is now complete and we can describe what we need to *learn* new concepts in a supervised learning setting:

Given a target classification $\mathfrak{c}_t$ defined by an unknown target funtion $t : s \to \mathbf{k}$ and its induced equivalence relation R_t, we want to extract knowledge in terms of cores and reducts relative to this classification from a dependent knowledge base. But there is still an open issue here: What does it mean for $\mathfrak{K}$ that t is unknown? It means that the intensional *definition* of t is not known. All we have is that for some $s \subseteq U$, we know a family of characteristic functions

$$\chi : \wp(s) \to (s \to \mathbf{2})$$

which for every $x \in s$ determines whether $x \in c \in \wp(s)$, i.e. $\chi(c)(x) = \mathbf{1} \iff x \in c$. These functions provide us with the *labels* in our sample we can use for a learning task. Accordingly, we would pick a sample $\mathbf{s}$ from $\mathfrak{K}$ of the form

$$\mathbf{s} \quad = \quad \{\langle x, \langle \chi(c_0)(x), \ldots \chi(c_{k-1})(x) \rangle\rangle : x \in s' \subseteq s\} \tag{6.43}$$

If $k = 2$, then $\mathfrak{c} = \{c_0, c_1\} = \{c, \overline{c}\}$ and $\chi(c)(x) = 1 \iff \chi(\overline{c})(x) = 0$. Therefore, binary learning problems result in much simpler sample structures:

$$\mathbf{s} = \{\langle x, \chi(c)(x) \rangle : x \in s' \subseteq s\}$$

Similary, samples for k-ary classifications can be represented by samples of tuples $\langle x, i \rangle$ if $\chi(c_i)(x) = 1$. For nominal learning problems, there is no difference between a multi-valued domain of the target function or a an according sparse vector of 0's with only one 1. Things are a bit different for ordinal target values, because the ordering on $\mathrm{cod}(t)$ has to be preserved in a binary vector representation.
s' is a *subset* of s—and it is so for a good reason:

Learning by Rough Sets
Knowledge Discovery by rough sets means to find smallest subsets of relations that are capable of describing presented data and which can be used to (roughly) describe whether *any* object belongs to a certain class or not.

We are now able to compare sets of relations to other sets of relations with respect to their expressive power concerning a new/unknown concept based upon observed data:

- The more knowledge we have, the finer (”crispier”) is our view on the domain.
- We can build specific (basic) concept descriptions from elementary concepts by intersections and complex concepts by unions of basic concepts.
- We can identify dispensable relations, reducts and cores.
- We have a set of arithmetic measures we can use to quantitatively estimate

With a few more arithmetic measures of sharpness and utility, we can:

1. Identify redundant relations.

2. Discover unknown concepts.
3. Find minimal, sufficently accurate approximations of known concepts.

And this is what we need to learn by finding knowledge that has been lost in too much information.

6.5 Knowledge Discovery

Given a knowledge base $\mathfrak{K}$ with $\mathbf{R} = \{R_{f_i} : i \in \mathbf{n}\} \cup \{R_t\}$, we call t a *decision attribute* which describes a new (unknown) concept $c = \{x \in s : t(x) = \mathbf{1}\}$. Whenever t is binary, it is the charcteristic function of the target oncept c; for nominal t, s/R_t is a target classification $\mathfrak{c} = \{\{x \in s : t(x) = i\} : i \in \mathbf{k}\}$. The information system $\mathfrak{I}\langle s, \mathbf{F}, V_{\mathbf{F}}\rangle$ then has a special feature t called the *decision attribute*; and, accordingly, the table representing $\mathfrak{I}$ together with t is called a *decision table*:

s	0	1	$\cdots$	$n-1$	f_{t}
x_0	$f_0(x_0)$	$f_1(x_0)$	$\cdots$	$f_{n-1}(x_0)$	$\mathsf{1}$
x_1	$f_0(x_1)$	$f_1(x_1)$	$\cdots$	$f_{n-1}(x_1)$	$\mathsf{0}$
$\vdots$	$\vdots$	$\vdots$		$\vdots$	$\vdots$

But using our newly acquired relational language of (rough) concepts, we can reformulate what it means to be concerned with *new* knowledge about *unknown* concepts c:
We call c a *new* or *unknown* concept, if there is no relation $R_f \in \mathbf{R}$ for which f is a characteristic function of c (or, at least, approximately equal). Still, c is (roughly) *learnable* by $\mathfrak{K}$, if c can be (roughly) described by $\bar{\bar{\mathbf{R}}}$. Then, the characteristic function of this (rough) set in terms of unions of basic classes is, in fact, new *terminologic knowledge*. The mere (rough) pointwise approximation of a concept c does not make any knowledge. To *learn* c means to find a *description* of t in terms of knowledge $\mathbf{R}$. And this brings us back to approximations of c.
At the same time, learning means to be able to *compress* data. We already do so by giving intensional definitions of sets of objects. But compression and especially *generalisation* could not take place without *dropping* knowledge. Lossless compression means to drop redundant knowledge only: It means to find *reducts*. Generalising also asks for dropping knowledge that at first glimpse weakens the *sharpness* of a concept description. In order to avoid over-generalisation we sopuld discard only those parts of knowledge whose *utility* appears rather weak.

Knowledge Discovery by Rough Sets
Knowledge discovery means to define a *new* relation R_h for which $s/R_h \approx s/R_t$. R_h is defined in terms of $\bar{\bar{\mathbf{P}}}$–elementary classes where $\mathbf{P} \subseteq \mathbf{R}$ is as small as possible and $[\![R_h \trianglelefteq R_t]\!]$s s big as possible.

Actually, knowledge discovery means to discover *dependencies* of knowledge in relation to (newly) observed data. Therefore, we need to ask the following questions:

- Reasoning about knowledge:
  1. Are there *redundant* relations?
  2. Are there *similar* relations?
  3. Are there redundant or similar *complex* concepts?
  4. Is there (in)*dispensable* knowledge around?
- Reasoning about new observations:
  1. Can the new concept be described by existing relation?
  2. Can it be approximated?
  3. If so, to which extent and accuracy?

So what we do is to analyse our knowledge and decide which elements we shall keep and which we shall discard so as to yield a minimal set of relations that induces an optimal approximation of the target partition. An exhaustive search in hypothesis space is infeasible. But we are lucky: Our framework allows us to define a few measures that help to heuristically guide our search for promising hypotheses. There are basically two issues in relational knowledge discovery that correspond to unsupervised and supervised learning:

Identifying potentially interesting sets of objects. If there are sets of objects which cannot be distinguished by $\bar{\bar{\mathbf{R}}}$ (that is, all the blocks in $s/\bar{\bar{\mathbf{R}}}$) then each of these sets may represent an unknown concept—simply because of the fact that we are unable to describe them. This case, however, is rather rare; usually, we collect and store data model so as to be able to explicate the information therein. On the hand this implies a (possibly strong) bias because strict modelling prevents learning beyond what our preconception expects. It requires an expert to understand whether $s/\bar{\bar{\mathbf{R}}}$ represents a reasonable breakdown of the structure of our domain. Finding such sets is, theoretically, simple: All we have to do is to compute $s/\bar{\bar{\mathbf{R}}}$. For the expert who is expected to interpret our findings this may not be helpful at all. Most likely he will ask for a simpler representation of the domain structure. Then we are facing the problem of (efficiently) finding reducts. Since there may be many such reducts the next problem is to chose the one which is most explanatory to the expert.

Circumscribing a new property in terms of present knowledge. Let there be some knowledge, say, $\mathbf{Q}$. Imagine we can use $\mathbf{Q}$ to describe a certain classification $\mathfrak{c}$. This means that some $\mathfrak{c}$ is (roughly) definable by $\mathbf{Q}$. What does it mean to *learn* then? It means to acquire some *new* knowledge to describe $\mathbf{Q}$. Since $\mathbf{Q} \subseteq \mathbf{R}$, a new "chunk" of knowledge $\mathbf{P} \subseteq \mathbf{R}$ can be considered to be a hypothesis describing $\mathbf{Q}$ if it is able to describe $\mathfrak{c}$ suffciently well. In this case, $\mathbf{Q}$ provides a teacher signal t, and we are *searching* for a set $\mathbf{P} \in \wp(\mathbf{R})$ to approximate $\mathbf{Q}$. There are several methods to determine whether $\mathbf{P}$ is a good candidate to describe $\mathbf{Q}$. An ideal hypothesis $\mathbf{P}$ should satisfy the following requirements:

1. **P** should be *small*; ideally, it should be smaller than **Q** and much smaller than **R**.
2. **P** should be *novel.* If **P** is more or less the same than **Q**, then there seems no reason to consider **P** as *newly discovered* knowledge.
   A first idea towards modelling novelty is that **P** should not share significantly more than $\mathrm{Cor}(\mathbf{R} \trianglelefteq \mathbf{Q})$ relations with **Q**.
3. **P** should not be much less accurate than **Q**; i.e. **Q** should (at least to a certain degree) depend on **P**.
4. Finally, **P** should be more general than **Q**.

Generality of knowledge chunks can be determined with respect to different parts of our universe: Let **R** be the set of all relations we can use to describe our domain. Then, all expressions over elements of **R** form the *representation space.* A target concept (or classification) $\mathfrak{c}$ is given by an (unknown) function t. It is represented by a (partially defined) decision attribute. Now, let there be two hypotheses **P** and **Q** to describe $\mathfrak{c}$. Both **P** and **Q** can be assumed to be (proper) subsets of **R**. In order to be able to compare both hypotheses we need to estimate their respective accuracy. For this, we have a measure as defined in definition 6.10. In terms of *searching* a subsumption lattice of hypotheses we need an according order relation modelling generality. We can compare two sets **P** and **Q** of knowledge according to their generality by comparing their respective positive regions for a "reference" set: So **P** is more general than **Q**, if

$$\llbracket \mathbf{Q} \trianglelefteq \mathbf{R} \rrbracket s' \approx \llbracket \mathbf{P} \trianglelefteq \mathbf{R} \rrbracket s' \quad \text{and} \quad \llbracket \mathbf{Q} \trianglelefteq \mathbf{R} \rrbracket s \subseteq \llbracket \mathbf{P} \trianglelefteq \mathbf{R} \rrbracket s \tag{6.44}$$

for $s' \subseteq s$. It means that knowledge **P** is able to positively correct classify more objects of the domain than **Q**.

Exercise 6.18 (◊) The last definition of generality is not to be confused with coverage. — Explain!

Again, the question is where to start searching, how to proceed—and when to stop.

Searching for the best Hypothesis
Rough Set Data Analysis allows us to identify potentially interesting sets of objects. This is what we refer to as *unsupervised learning.*
Rough Set Data Analysis also allows us to acquire new knowledge by searching for new sets of knowledge that can describe a classification defined by a decision attribute. This is what is called *supervised learning*

6.5.1 Utility

Utility is a measure that says something about the *degree* uf usefulness. It describes *how much* a relation (or a set of relations) contributes to the ability of describing knowledge. Definition 6.10 gave us a means to estimate a relations usefulness with respect to a class or classification.

In order to quantitavely describe the usefulness of a set of relations, we could simply determine the relative size of the positive region:

(Relative) Utility

Definition 6.18 — (Relative) Utility.
We define the *(relative) utility of* **P** *(on s) for describing knowledge* **Q** to be

$$\mathit{utility}_s(\mathbf{P} \trianglelefteq \mathbf{Q}) \quad := \quad \frac{|[\![\mathbf{P} \trianglelefteq \mathbf{Q}]\!]s|}{|s|} \tag{6.45}$$

The quotient describes the the relative number of objects which by **P** can be correctly clasified with respect to the classification $\mathfrak{c} = s/\bar{\bar{\mathbf{Q}}}$. ●

In prose, the utility of a set of relations is the quotient of the number of correctly classified objects using this set of relations and the number of all objects under consideration, $|s|$. When designing an algorithm to determine reducts, the algorithm will proceed stepwise; i.e. in order to decide in eavery single step which relation to drop or to add, we would compute the utility of a singleton set $\mathbf{P} = \{P\}$ for **Q**:

$$\mathit{utility}_s(\mathbf{P} \trianglelefteq \mathbf{Q}) \quad = \quad \frac{|\bigcup_{c_i \in s/\bar{\bar{\mathbf{Q}}}} [\![P]\!]c_i|}{|s|} = \frac{1}{|s|} \sum_{i=0}^{\mathbf{n}} |[\![P]\!]c_i| \tag{6.46}$$

where $n = |s/\bar{\bar{\mathbf{Q}}}|$. The equality holds because of the pairwise disjointness of equivalence classes.
Usually, utility is a measure with respect to a target calssification. But since a target classification is just a set quotient as any other quotient, we can use utility to describe the dependencies between any *arbitrary* sets of equivalence relations. And, of course, we can extend the definition of utility to classifications rather than just sets:

$$\mathit{utility}_{s/\bar{\bar{\mathbf{R}}}}(\mathbf{P} \trianglelefteq \mathbf{Q}) \quad := \quad \frac{1}{|s|} \sum_{c_i \in s/\bar{\bar{\mathbf{R}}}} |[\![\mathbf{P} \trianglelefteq \mathbf{Q}]\!]c_i| \tag{6.47}$$

It is clear, that if $\mathit{utility}_s(\mathbf{P} \trianglelefteq \mathbf{Q}) = 1$, **P** is good enough for describing **Q** on the entire domain. In this case it must hold that $[\![\mathbf{P} \trianglelefteq \mathbf{Q}]\!]s = s$ and therefore, $\mathbf{P} \preceq \mathbf{Q}$. We can also conclude, that **P** implies **Q**: $\mathbf{P} \supset \mathbf{Q}$. If, on the other hand, $\mathit{utility}_s(\mathbf{P} \trianglelefteq \mathbf{Q}) = 0$ then the numerater must be 0. This can happen only if $[\![\mathbf{P} \trianglelefteq \mathbf{Q}]\!]s$ is empty and that means, that *all* lower **P**–approximations of **Q**-classes are empty: $\forall c \in s/\bar{\bar{\mathbf{Q}}} : [\![\mathbf{P}]\!]c = \emptyset$. In other words, **P** can not positively describe anything that **Q** is concerned with.
The interesting case is when $0 < \mathit{utility}_s(\mathbf{P} \trianglelefteq \mathbf{Q}) < 1$. Then, the bigger the value is, the bigger the positive region. The smaller it is, the less **P** seems to be appropriate for describing **Q**. Another equivalent formulation is that the average sharpness of **P**–approximations of **Q**–classes increases as the utility of **P** for **Q** increases (and vice versa).[8] Finally, it means that **P** implies **Q** not entirely, but at least up to a certain degree.

[8] If we define $R :=\in \mathbf{P}$ and $\mathfrak{c} := s/\mathbf{Q}$, then we we can see that $\mathit{utility}_s(\mathbf{P} \trianglelefteq \mathbf{Q})$ is the same as $\mathit{utility}_R(\mathfrak{c})$ (see definition 6.10).

This way, we are able to order sets of relations with respect to the degree of their ability of mutually describing each other. Then, we would want to find a smallest set $\mathbf{P}$ of equivalence relations with maximum utility to escribe each other set—thus we would reduce redundant knowledge. If such a set would also prove to have a high utility for describing $\mathfrak{c}$, we would have found a beautiful, non–redundant and concise description of our target classification.

Partial Implication

Definition 6.19 — Partial Implication.
Let there be knowledge base $\mathfrak{K} = \langle s, \mathbf{R} \rangle$ and $\mathbf{P}, \mathbf{Q} \subseteq \mathbf{R}$. We say that $\mathbf{P}$ *implies* $\mathbf{P}$ to a degree k, iff:

$$k = utility_s(\mathbf{P} \trianglelefteq \mathbf{Q}) = \frac{|[\![\mathbf{P} \trianglelefteq \mathbf{Q}]\!]s|}{|s|} \tag{6.48}$$

We then write $\mathbf{P} \overset{k}{\supset} \mathbf{Q}$. ●

So if: Every set $\mathbf{P}$ or $\mathbf{Q}$ consists of relations or attributes. Therefore, utility or partial dependance means "utility of a set of attributes". If we recosider the early definition of lower and upper approximations it becomes clear, that the utility of a set of attributes is not determined uniformly by all attributes, but that some attributes may contribute more than others. This way, we can deduce a measure of individual attribute significance from the definition of a set's utility or dependence.

6.5.2 Attribute Significance

Some attributes seem to have a greater descriptive power than others. All machine learning methods try to find the most significant ones first so as to to quickly converge to a solution. Decision trees utilise an information measure, support vector machines generate separating planes. In rough set theory, we want to find a smallest set of attributes which we accordingly may assume to be more significant than redundant attributes.
Justy as a reminder, let us reconsider the way information gain methods tackle this problem:

Example 6.18 Consider again the following information system:

s	f_0	f_1	f_2	f_3	$f_{\mathfrak{t}}$
0	1	•	♡	c	0
1	0	•	♠	b	/
2	2	•	♣	b	1
3	1	•	♣	c	/
4	1	•	♡	a	1
5	2	•	♣	b	1
6	2	•	♠	b	/
7	0	•	♠	a	1

A quick analysis shows that

$$H(s) = -\frac{1}{8}\log_2\frac{1}{8} - \frac{3}{8}\log_2\frac{3}{8} - \frac{4}{8}\log_2\frac{4}{8} \approx 1.406.$$

and that for all $f_i \in \mathfrak{F}$, $H(f_i, s)$ has the following values

1. $\frac{2}{8}H(\{1,7\}) + \frac{3}{8}H(\{0,3,4\}) + \frac{3}{8}H(\{2,5,6\}) \approx 1.293$ and $Gn(f_0, s) = 0.113$
2. $\frac{1}{8}H(\{5\}) + \frac{3}{8}H(\{0,2,4\}) + \frac{4}{8}H(\{1,3,6,7\}) \approx 0.749$ and $Gn(f_1, s) = 0.657$.
3. $\frac{2}{8}H(\{0,4\}) + \frac{3}{8}H(\{1,6,7\}) + \frac{3}{8}H(\{2,3,5\}) \approx 0.508$ and $Gn(f_2, s) = 0.898$.
4. $\frac{2}{8}H(\{4,7\}) + \frac{4}{8}H(\{1,2,5,6\}) + \frac{2}{8}H(\{0,3\}) = 0.75$ and $Gn(f_3, s) = 0.656$.

As one can easily see, feature f_2 provides maximum information gain. One would expect it is the feature that is most significant.

How do we extract the most significant (useful) attributes without any assumptions or heuristic guidelines? Let $\mathfrak{c} = s/\mathbf{S}$ and imagine $\mathbf{Q}$ is our current subset of attributes that we want to reduce in order to find a reduct. To determine a minimal set of attributes $\mathbf{P} \subseteq \mathbf{Q}$, $\mathbf{R} = \mathbf{Q} - \mathbf{P}$ with maximum utility, we need to:

1. drop attributes R such that $\mathbf{P} = \mathbf{Q} - \{R\}$ where the loss of R causes least *utility loss* on $\mathfrak{c} = s/\mathbf{S}$.

2. keep attributes $R \in \mathbf{P}$ that would cause most *utility loss* with respect to $\mathfrak{c} = s/\mathbf{S}$.

This is simply the idea behind top–down or bottom–up computation of reducts *relative to* $\mathbf{S}$. First, note that

$$\begin{aligned} \mathit{utility}_s(\mathbf{P} \trianglelefteq \mathbf{S}) &\leq \mathit{utility}_s(\mathbf{Q} \trianglelefteq \mathbf{S}) & (6.49)\\ \mathit{utility}_s((\mathbf{Q} - \mathbf{P}) \trianglelefteq \mathbf{S})\mathit{utility}_s(\mathbf{R} \trianglelefteq \mathbf{S}) &\leq \mathit{utility}_s(\mathbf{Q} \trianglelefteq \mathbf{S}) & (6.50) \end{aligned}$$

Then, based on relative utility we define an attribute's *significance* similar to *information gain* (see equation (5.7)):

Attribute signifiance

Definition 6.20 — Attribute signifiance.
The *significance* of a set of attributes is the *loss of utility* caused by its removal:

$$\mathit{significance}_s(\mathbf{P} \trianglelefteq \mathbf{S}) = \mathit{utility}_s(\mathbf{Q} \trianglelefteq \mathbf{S}) - \mathit{utility}_s(\mathbf{Q} - \mathbf{P} \trianglelefteq \mathbf{S}) \quad (6.51)$$

is called the *(*$\mathbf{Q}$*-relative) relative significance* of $\mathbf{P}$ for $\mathbf{S}$.

Exercise 6.19 (♦) Even though $\mathbf{Q} = \mathbf{P} \dot{\cup} \mathbf{R}$,

$$\mathit{significance}_s(\mathbf{P} \trianglelefteq \mathbf{S}) + \mathit{significance}_s(\mathbf{R} \trianglelefteq \mathbf{S}) = \mathit{significance}_s(\mathbf{Q} \trianglelefteq \mathbf{S})$$

is in general *not* true. Prove!

The idea behind knowledge discovery by rough set then means deftly chosing

R (top-down) such that

$$\mathit{significance}_s(\mathbf{P} \trianglelefteq \mathbf{S}) = \mathit{utility}_s(\mathbf{Q} \trianglelefteq \mathbf{S}) - \mathit{utility}_s((\mathbf{R}) \trianglelefteq \mathbf{S})$$

is *maximised*, or

P (bottom-up) such that

$$\mathit{significance}_s(\mathbf{R} \trianglelefteq \mathbf{S}) = \mathit{utility}_s(\mathbf{Q} \trianglelefteq \mathbf{S}) - \mathit{utility}_s((\mathbf{Q} - \mathbf{R}) \trianglelefteq \mathbf{S})$$

is *minimised*

This is actually a *search problem*, and thus—according to Michie—a machine learning problem.
Putting Rst into an application means to provide efficient tools to compute **P** as a hypothesis to describe a target **S** when we start off with a given set **Q** of attributes (top–down) or nothing; i.e. an empty set of relations.

6.6 Conclusion

What is rough set theory good for, you might ask? What does it offer that entropy-based heuristially guided decision tree induction does not have? Well, the answer is:

Less!

It definitively lacks one important property, and this is *the value of a feature for an object.* If, in a decision tree, N_f is the least node subsuming two objects x and y, then we know that N_f has at least two different successor nodes N_x and N_y subsuming x and y respectively. We also know that $\varphi(N_x)$ and $\varphi(N_y)$ differ in only one literal in the precondition. This difference is exactly in the value of feature f: $f(x) = v_x \neq v_y = f(y)$.
In rough set theory, all we know is that $f(x) \neq f(y)$ because $x \neq_{R_f} y$. We lost knowledge about the actual values v_x and v_y. But does it hurt? Not at all. First, we can reconstruct v_x by asking $\mathfrak{I}$ about $f(z)$ for an arbitrary $z \in [x]_{R_f}$. We need to save the value for one element of this class only and know that all equivalent objects share the same value. Second, we do not only not really lose information—we even compressed our knowledge further: The information that from some decision node on downwards $f(z) = v_x$ holds for all subsumed objects, is stored in *every* subsumed node. Sure we can create a more efficient data structure but at latest when we try some kind of rule based post pruning, we need all the literals in every node again.
So there is not really a big loss here. According to our understanding of knowledge as the ability to discriminate different things from each other we have not lost anything.
Another argument could be that, of course, we lost the *structure* of a decision tree. Is that true? You guessed it: No (as far as we are concerned with *definable*

classifications). Let us take a closer look at the front of a decision tree d we induced to describe t. Of course, it represents a classification, say $\mathfrak{d} = s/D$ with every leaf node being an equivalence class. Then (presupposing t is definable), it holds that

$$\forall c_i \subseteq s : c_i \in \mathfrak{d} \Longrightarrow c_i \in s/R_t.$$

Let $\mathbf{D} \subseteq \mathbf{R}$ be the set of all the equivalence relations induced by all the features used in any of the decision nodes. It surprisingly holds that

$$\mathbf{D} \preceq D$$

because features cannot appear more than just once in a tree. So unless the least subsuming node of two leaf nodes is the root node, their rule representations contain *different* literals. The difference becomes clear when comparing figures 4.4 and 6.2. The first one illustrates that on every level of the tree, only equivalence classes are further partitioned, while $\bar{\bar{\mathbf{R}}}$ partitions the entire set s. It is, so to say, a flat tree and $\bar{\bar{\mathbf{D}}}$ is the equivalence relation that is used as decision attribute in the root node. We can then simply enumerate all the classes and for each class we find a representative for which we find all the values required for all the features in $\mathbf{D}$ that define this class. Then, if

$$\mathbf{D} \preceq R_t,$$

$\mathbf{D}$ is a very good hypothesis for t. And it is not so bad, if

$$\mathbf{D} \overset{\gamma}{\supset} R_t$$

for large γ.

Still one might argue that the *loss* of information by not entirely partitioning the universe now is lost. Decision trees do not only express which features are important to make a certain decision. They also express a relative dependence of features: If a feature g appears in a decision node below N_f, then we know that g is somehow required by f in order to approximate t. If it does *not* appear in a subtree below N_f, then it is not significant or already included in the indiscernability relation defined by all the features in literals in $\varphi(N_f)$.

It follows that

utility	corresponds to	*information* and
significance	corresponds to	*information gain.*

The only difference is that decision trees presupposes the validity of the assumed entropy measure, while rough set data analysis simply takes an unbiased look at the data.

Chapter 7

Inductive Logic Learning

> Common sense expert knowledge usually consists of *rules*, or at least, rules are used to communicate expert knowledge. Such rules are represented using a suitable formal system, that is, some kind of a (terminological) logic. Herein, the terminology and factual knowledge can be seen as a knowledge base; i.e. it is what we can feed into a system which we want to induce more general and previously unknown laws from our observations. This is what inductive logic learning is concerned with.

Every object x of our universe actually is a *pattern*. Assuming that we have an information system $\mathfrak{I} = \langle U, \mathbf{F}, V_{\mathbf{F}} \rangle$ describing all elements of our domain, then x is nothing else than a vector:

$$\rho(x) = \vec{x} = \langle f_0(x), f_1(x), \ldots, f_{n-1}(x) \rangle$$

With a suitable representation function ρ, similar objects should be represented by similar vectors. Similarity is not so much a property of vectors but rather a property of an according distance measure of which there are many. We know that assuming any of these measures in representation space is a huge bias—for any method relying on this measure of similarity implies "natural" similarity of objects in the real world. This is one minor motivation that led us to rough set data analysis.

But this is not the only reason why one would like to consider another possible solution avoiding this kind of bias. Usually, there is more to a complex domain than just patterns. There are dependencies. If, for example the pattern ■ is "similar" to the pattern □ and if the pattern ● is "similar" to the pattern •, then we often find that ▲ is "similar" to ▲. As one can see, objects are similar to each other if they have the same shape. What happened here? First, there is a *rule.* If there is something like this and something like that, *then* there is something like so. Second, this rule actually implements "similarity": Two

things are of the same breed if they have the same shape. We did not presuppose too much and are rewarded with a concept and its procedural definition.
Many people prefer rules to patterns; especially when it comes down to describing concepts. If we want to discover new concepts the knack is to find new rules that describe our data. There are special kinds of rules which together form a *logic program*. Rule discovery means to *invent logic programs*.

7.1 From Information Systems to Logic Programs

Let there be an information system $\mathfrak{I} = \langle s, \mathbf{F}, V_{\mathbf{F}} \rangle$. Every feature f_i in $\mathbf{F}$ is defined as a *function*

$$f_i : s \to V_i.$$

For $\mathfrak{I}$ we also defined an information *function*, $I : s \times \mathbf{n} \to V_{\mathbf{F}}$, with $I(x, i) = f_i(x)$. But why do database managers speak of "tables" and "*relations*"?

7.1.1 Functions and Relations

Functions are special cases of relations such that

$$f : s \rightharpoondown V_f$$

Therefore, every feature f is a heterogenous binary relation. In the last chapter we took a closer look at a more generalised interpretation of f:

$$R : s \rightharpoondown s \text{ where } xRy \iff f(x) = f(y)$$

Therefore, an information system $\mathfrak{I}$ provides us with two sets of relations of different kinds:

- The set $\mathbf{F}$ contains heterogeneous binary relations that relate objects of our domain to some property and
- the set $\mathbf{R}$ contains equivalence relations on s relating pairs of objects of our domain that share the same properties.

R defines a classification $=s/R$, and we know there is a set $\dot{c} \subset s$ of representatives such that the union of all R-equivalence classes of elements of $\dot{c}$ forms the partition $\mathfrak{c}$. Therefore,

$$\forall x \in \dot{c}, y \in s : y \in [x]_R \Longrightarrow f(y) = f(x)$$

Reformulating $f : s \to V_f$ as a relation $F : s \rightharpoondown V_f$, this becomes

$$\forall x \in \dot{c}, y \in s : \forall z \in V_f : xRy \Longrightarrow yFf(x)$$

And, finally, speaking in terms of relation algebra, it holds that

$$R^{\smile}F = F \text{ and, by symmetry of equivalence relations, } RF = F \tag{7.1}$$

In databases, $f \in \mathbf{F}$ are called the "columns" of a *table*, and all $x \in s$ form the *columns*.

Exercise 7.1 (◊♦) The entire table is often referred to as a (single) relation. Why? — Read up the definition of *heterogeneous (binary) relations*! Be sure to know about the difference between *relation algebra* and *relational algebra*!

Relations are the algebraic counterpart of what in first order logic (FOL) is called a *predicate*.

7.1.2 Semantics of First Order Logic

We assume the reader to be familiar with first order logic, but in order to elucidate its connection to relational knowledge representation and discovery, let us briefly recall the most important pieces from its semantics.
The syntax of FOL is defined by the construction of terms (Ter_Σ) and formulas (Fml_Σ) over a *signature* of function symbols, predicate symbols, connectives, quantifiers and variable symbols (Var_Σ). Their *meaning* is a mapping from into a structure called a Σ–algebra:

Σ–Algebra $\mathfrak{A}$

Definition 7.1 — Σ–Algebra $\mathfrak{A}$.
Let Σ be a signature. We define a structure $\mathfrak{A}$, called a Σ*–algebra*, as follows:

1. s_A is the base set of $\mathfrak{A}$
2. For $n > 0$ and a function symbol $f : s^n \to s \in \Sigma$,
   there is a function $f_A : s_A^n \to s_A$
3. For $n > 0$ and a predicate symbol $p : s^n \in \Sigma$,
   there is a relation $p_A \subseteq s_A^n$
4. For $n = 0$ and a constant symbol $c :\to s \in \Sigma$,
   there is $c_A \in s_a$
5. for $n = 0$ and an atom $p :\in \Sigma$,
   there is a *truth value* in $\mathbf{2} := \{\mathbf{0}, \mathbf{1}\}$

●

A Σ-algebra provides a structure in which we can interpret a formula. Proper interpretation requires not only to map predicate of function symbols onto relations and functions respectively. Variables need to have a value, and terms must be evaluated:

Assignment α,
Evaluation $\downharpoonleft\cdot\downharpoonright^{\mathfrak{A}}_{\alpha}$

Definition 7.2 — Assignment α, Evaluation $\downharpoonleft\cdot\downharpoonright^{\mathfrak{A}}_{\alpha}$.
An *assignment* or *valuation* is defined by a function $\alpha : \mathrm{Var}_{\mathrm{FOL}} \to s_a$ mapping variable symbols onto elements of s_A. A modified assignment

$$\alpha' := \alpha\,[X_0 \mapsto x_0, \ldots, X_{n-1} \mapsto x_{n-1}]$$

is defined as

$$\alpha'(X) = \begin{cases} x_i & \text{for } X \in \{X_i : i \in \mathbf{n}\} \\ \alpha(X) & \text{otherwise} \end{cases} \tag{7.2}$$

An *evaluation* is a function delivering the result of resolving a term in its Σ-algebra $\mathfrak{A}$ based on the current variable assignment α:

$$\downharpoonleft : \downharpoonright^{\mathfrak{A}}_{\alpha} \mathrm{Ter}_{\Sigma} \to s_a$$

For variable symbols $X \in \mathrm{Var}_{\Sigma} \subseteq \mathrm{Ter}_{\Sigma}$, we define $\downharpoonleft X \downharpoonright^{\mathfrak{A}}_{\alpha} := \alpha(X)$. Complex terms are evaluated recursively:

$$\downharpoonleft f(t_0, \ldots, t_{n-1}) \downharpoonright^{\mathfrak{A}}_{\alpha} \quad := \quad f_A(\downharpoonleft t_0 \downharpoonright^{\mathfrak{A}}_{\alpha}, \ldots, \downharpoonleft t_{n-1} \downharpoonright^{\mathfrak{A}}_{\alpha})$$

●

So far, all we can do is to compute terms—but the interesting thing is to evaluate formulas $\varphi \in \mathrm{Fml}_{\Sigma}$. FOL-formulas, when "evaluated", should deliver a value that tells us whether a formula describes a situation which is somehow "true" or "false". "True" and "false" are values in $\mathfrak{A}$, but the meaning of a formula to be "true" is, that it is *valid*:

Validity, $\models$

Definition 7.3 — Validity, $\models$.
Let $\mathfrak{A}$ be a Σ-algebra and $\varphi, \psi \in \mathrm{Fml}_{\mathrm{FOL}}$. We call φ *valid in* $\mathfrak{A}$ *under* α ($\mathfrak{A} \models_{\alpha} \varphi$), if:

$$\begin{array}{lll}
\mathfrak{A} \models_{\alpha} p(t_0, \ldots, t_{n-1}) & :\Longleftrightarrow & \langle \downharpoonleft t_0 \downharpoonright^{\mathfrak{A}}_{\alpha}, \ldots, \downharpoonleft t_{n-1} \downharpoonright^{\mathfrak{A}}_{\alpha} \rangle \in p_A \\
\mathfrak{A} \models_{\alpha} p & :\Longleftrightarrow & \downharpoonleft p \downharpoonright^{\mathfrak{A}}_{\alpha} = \mathbf{1} \\
\mathfrak{A} \models_{\alpha} (\neg\varphi) & :\Longleftrightarrow & \mathfrak{A} \not\models_{\alpha} \varphi \\
\mathfrak{A} \models_{\alpha} (\varphi \wedge / \vee / \to \psi) & :\Longleftrightarrow & \mathfrak{A} \models_{\alpha} \varphi \text{ and/or/implies } \mathfrak{A} \models_{\alpha} \psi \\
\mathfrak{A} \models_{\alpha} \forall X : \varphi & :\Longleftrightarrow & \forall a \in s_A : \mathfrak{A} \models_{\alpha[X \mapsto a]} \varphi \\
\mathfrak{A} \models_{\alpha} \exists X : \varphi & :\Longleftrightarrow & \exists a \in s_A : \mathfrak{A} \models_{\alpha[X \mapsto a]} \varphi
\end{array}$$

Herein, $\alpha[X \mapsto a]$ means that the function α at X is redefined so as to deliver the value $a \in s_A$.
φ is *valid in* $\mathfrak{A}$, if $\mathfrak{A} \models_{\alpha} \varphi$ for all α, and φ is called *valid*, if φ is valid in every $\mathfrak{A}$. ●

7.1.3 Deduction

Deduction or inference means to draw conclusions. So if there is an argument which we agree to be admissible somehow, we want to be able to draw according conclusions in $\mathfrak{A}$. This suggests to call $\mathfrak{A}$ a *model*:

Model, $\mathfrak{A} \models \varphi$

Definition 7.4 — Model, $\mathfrak{A} \models \varphi$.
A Σ-algebra $\mathfrak{A}$ is called *a model* of a formula $\varphi \in \mathrm{Fml}_{\Sigma}$, iff:

$$\mathfrak{A} \models \varphi \quad :\Longleftrightarrow \quad \mathfrak{A} \models_{\alpha} \varphi \tag{7.3}$$

for all assignments α. For a set of formulas $\Phi \subset \mathrm{Fml}_{\Sigma}$,

$$\mathfrak{A} \models \Phi \quad :\Longleftrightarrow \quad \forall \varphi \in \Phi : \mathfrak{A} \models \varphi \tag{7.4}$$

●

The question whether an algebra is a model depends on the interpretation of the signature, and, of course, on the assignments.

Exercise 7.2 ⧫ Learn about a special kind of algebras and models; so-called *Herbrand algebras* and *Herbrand models* (for example, [Mazzola et al., 2006], [Sperschneider and Antoniou, 1991] or [Ehrig et al., 2001]).

A naturally or intuitively valid inference means that from one formula (or a set of formulas) follows another. It means that if the first formula has a model, the second one should be valid in this algebra as well! This makes inference a relation between models.

Definition 7.5 — Entailment, $\Phi \mathrel{|\!\!\approx} \varphi$.

Entailment, $\Phi \mathrel{|\!\!\approx} \varphi$

A set $\Phi \subseteq \mathrm{Fml}_\Sigma$ *entails* a formula $\varphi \in \mathrm{Fml}_\Sigma$, if and only if every model of Φ is a model of φ, too:

$$\Phi \mathrel{|\!\!\approx} \varphi \quad :\Longleftrightarrow \quad \forall \mathfrak{A} : \mathfrak{A} \models \Phi \Longrightarrow \mathfrak{A} \models \varphi. \tag{7.5}$$

Usually, the entailment relation is denoted $\models$; in order to avoid confusion we use $\mathrel{|\!\!\approx}$ instead. ●

The deduction rule that is most common in every day life is *modus ponens*. It states that if from a certain premise we may draw a certain conclusion, and if that premise is a valid statement, then the conclusion must be valid, too.

Modus Ponens

Thinking implies being; or, as Descartes put it: "Cogito ergo sum". The premise is *to be thinking*, and once *you think*, you *must be*. Therefore, if you *think*, you *are*. In other words:

$$\mathfrak{A} \models \varphi \;\wedge\; \mathfrak{A} \models (\varphi \longrightarrow \psi) \;\Longrightarrow\; \mathfrak{A} \models \varphi.$$

Entailment is simple to understand but difficult to show: One has to find *all* models of Φ and then needs to check whether each model is a model of φ, too. This is, for a machine, at least cumbersome if not impossible. But what is logic for good if there is no proper way to implement entailment?
There appears to be a loophole, though: Suppose that $\Phi \mathrel{|\!\!\approx} \varphi$. Then,

$$\begin{array}{rlcccc} & \forall \mathfrak{A} : \mathfrak{A} \models & \Phi & \Longrightarrow & \mathfrak{A} \models & \varphi \\ \text{iff} & \forall \mathfrak{A} : \mathfrak{A} \not\models & \varphi & \Longrightarrow & \mathfrak{A} \not\models & \Phi \\ \text{iff} & \forall \mathfrak{A} : \mathfrak{A} \models & (\neg\varphi) & \Longrightarrow & \mathfrak{A} \not\models & \Phi \\ \text{iff} & \forall \mathfrak{A} : \mathfrak{A} \models & \Phi & \Longrightarrow & \mathfrak{A} \not\models & (\neg\varphi) \end{array} \tag{7.6}$$

In FOL, there exist *two* truth values, **0** and **1**. So if $\Phi \mathrel{|\!\!\approx} \varphi$, it cannot be the case that $\Phi \mathrel{|\!\!\approx} (\neg\varphi)$, too. And this means that

$$\Phi \mathrel{|\!\!\approx} \varphi \quad \Longleftrightarrow \quad \Phi \cup \{(\neg\varphi)\} \;\text{ has no model.} \tag{7.7}$$

or

$$\forall \mathfrak{A} : ((\mathfrak{A} \models \Phi \Longrightarrow \mathfrak{A} \models \varphi) \quad \Longleftrightarrow \quad \mathfrak{A} \not\models (\neg\varphi)) \tag{7.8}$$

This is a bit cumbersome to write down, so we define a special symbol $\Box$ for FOL-formulas that do not possess *any* model:

$$\forall\mathfrak{A} : \mathfrak{A} \not\models \Box \tag{7.9}$$

Together with equation (7.7) this gives:

$$\Phi \models\!\!\approx \varphi \iff \Phi \cup \{(\neg\varphi)\} \models\!\!\approx \Box \tag{7.10}$$

But still there are universal quantifiers involved in this definition.
First order logic is a formal system whose syntax is given by a grammar. Syntactically correct symbol strings make terms and formulas. The validity of formulas requires to inspect interpretations and so does entailment. The semantics of FOL was defined in a way that allows to compute a term's value in an algebra, but, by a method called *substitution*, we can anticipate this: Let $\sigma = [X_0 \mapsto t_0, X_1 \mapsto t_1, \ldots, X_{n-1} \mapsto t_{n-1}]$ denote a substitution such that its application to $\varphi \in \mathrm{Fml}_\Sigma$, written by juxtaposition $\varphi\sigma$, yields a formula ψ in which every occurrence of variables X_i is replaced by $t_i \in \mathrm{Ter}_\Sigma$. There is a lemma which states that it does not matter whether we carry out a substitution and then evaluate the formula or whether we evaluate the formula with a modified assignment:

$$\mathfrak{A} \models_\alpha \varphi[X \mapsto t] \iff \mathfrak{A} \models_{\alpha[X \mapsto a]} \varphi \tag{7.11}$$

where a is the value of t in $\mathfrak{A}$. This way, we can choose where to carry out variable assignments: on syntactic level or on semantic level. The idea is to find a similar technique to make proving entailment easier—by shifting the problem back onto the syntactic level.
There are many such methods, and they are called *calculi.* They all consist of different sets of term rewriting rules. For illustration, we give one example:

Example 7.1 For all sets Φ of well-formed FOL-formulas we define the following rule: If a string $v \in \Phi$ and if there is a string $(v \to w) \in \Phi$, then w is a string in Φ, too. Such a rule usually is written as follows:

$$\frac{v \quad (v \to w)}{w}$$

Carrying out this simple term rewriting rule we do what semantically corresponds to the modus ponens. ●

The good news is there are some calculi which are sound and complete, like Hilbert calculi, sequent calculi, or resolution. Whenever one can derive φ from Φ, then Φ entails φ and whenever Φ entails φ, then φ can be derived from Φ. The bad news is these calculi cannot be implemented to perform efficient in all cases (the interested reader might want to learn more about Kurt Gödel's completeness theorem). But the worst news is that for every "interesting" set of formulas Φ, it is in general *undecidable* (c.f. the famous works of Alonzo Church and Alan Turing). Nearly all problems are "interesting"—especially, in the case

of knowledge discovery. You will not be satisfied if you had not learned anything interesting while reading this book, for example, will you?
The question is: Is there a subset of predicate logic which still allows to formulate most "interesting" problems and for which there exists a satisfying calculus? Well, there is. And, together with equation (7.11) it allows us to prove things this way:

$$\Phi \mathrel{|\!\!\approx} \varphi \iff \Phi \cup \{(\neg\varphi)\} \vdash \square \tag{7.12}$$

Sets of FOL-formulas can be syntactically transformed into sets of formulas all of which have the same form. Examples are *disjunctive* and *conjunctive normal form* where the latter one is the most important to us. The derivation calculus to which we alluded in the previous chapter is called *resolution* and it is based on the idea of proofs by *refutation* as defined in equation (7.12).

Definition 7.6 — FOL-**Resolution**.

FOL-Resolution

Let there be two formulas φ and ψ which both consist of a disjunction of either positive or negative atomic formulas (called *literals*). For a disjunction of literals $\bigvee_{i\in l}\lambda_i$ we simply write $\{\lambda_0,\ldots,\lambda_{n-1}\}$.
Let $\lambda_\varphi \in \varphi$ and $(\neg\lambda_\psi) \in \psi$ and a unifying substitution σ such that $\lambda_\varphi\sigma = \lambda_\psi\sigma$. We then define the *resolution rule* RES as follows:

$$\frac{\{\kappa_0,\ldots,\lambda_\varphi,\kappa_{n-1}\} \qquad \{\nu_0,\ldots,(\neg\lambda_\psi),\nu_{m-1}\}}{\{\kappa_0,\ldots,\kappa_{n-1},\nu_0,\ldots,\nu_{m-1}\}\,\sigma}\mathsf{RES} \tag{7.13}$$

For two *parent clauses* φ and *psi* we denote the conclusion after applying RES the *resolvent* and write $\varphi \sqcap^{\sigma}_{\mathsf{RES}} \psi$. A derivation in the resolution calculus is written $\{\varphi,\psi\} \vdash_{\mathsf{RES}} \chi = \varphi \sqcap^{\sigma}_{\mathsf{RES}} \psi$. ●

We conclude:

Resolution
The resolution principle is based on the modus ponens rule. The structure of a resolution proof is based on refutation.

$$\begin{array}{rcll}
\Phi \vdash \varphi & \Longrightarrow & \Phi \mathrel{|\!\!\approx} \varphi & \checkmark \\
\Phi \cup \{(\neg\varphi)\} \vdash \square & \Longrightarrow & \Phi \mathrel{|\!\!\approx} \varphi & \checkmark \\
 & & & \\
\Phi \mathrel{|\!\!\approx} \varphi & \not\Longrightarrow & \Phi \vdash \varphi & ☠ \\
\Phi \mathrel{|\!\!\approx} \varphi & \Longrightarrow & \Phi \cup \{(\neg\varphi)\} \vdash \square & \checkmark
\end{array}$$

We end this section with a nice observation: Let there be a function *depth* that delivers the depth of a term as follows:

$$depth(t) \quad = \quad \begin{cases} 0, & t \in \mathrm{Var} \vee \mathrm{Ter}^0 \\ \max\{depth(t_i) : i \in \mathbf{n}\} + 1, & t = f(t_0,\ldots,t_{n-1}) \end{cases}$$

The depth of of a literal is the maximum depth of all the terms in the literal; and the depth of a clause is the maximum depth of all the literals in it. Since resolving a literal requires the application of a substitution, the depth of the resolvent cannot be smaller than its parent clauses. The corollary is:

$$depth(\varphi) > \mathrm{depth}(()\psi) \Longrightarrow \varphi \not\mathrel{|\!\!\approx} \psi \tag{7.14}$$

unless ψ is a tautology.

7.2 Horn Logic

7.2.1 Logic Programs

Still, full FOL leaves us with many possible collections of positive and negative atomic formulas which creates a problem of choice in efficient resolution proofs. Therefore, we restrict FOL to the following subset.

Horn Logic

Definition 7.7 — Horn Logic.
Horn logic HOL is the subset of FOL containing a set $\Phi' \in \mathrm{Fml}_{\mathrm{HOL}}$ for every set $\Phi \subseteq \mathrm{Fml}_{\mathrm{FOL}}$ such that

1. $\forall \mathfrak{A} : \ \mathfrak{A} \models \Phi \Longleftrightarrow \mathfrak{A} \models \Phi'$
2. $\forall \varphi \in \Phi' : \ \varphi = ((\psi_0 \wedge \psi_1 \wedge \cdots \psi_{n-1}) \longrightarrow \psi_n)$

where all ψ_i, $i \in \mathbf{n} \cup \{\mathbf{n}\}$ are *atomic formulas*. By simple transformation, every formula φ has the form

$$(\neg\psi_0) \vee (\neg\psi_1) \vee \cdots \vee (\neg\psi_{n-1}) \vee \psi_n$$

Written as

$$\varphi = \{\neg\psi_0, \neg\psi_1, \ldots, \neg\psi_{n-1}, \psi\}$$

it is called a *clause* and each element a *literal*. By restricting the length of the conclusion of the rule to one atomic formula, every such clause has *at most one positive literal*, also called a *Horn clause*, [Horn, 1951]. ●

According to the definition above, there are three different types of Horn formulas:

1. *Facts* are unary clauses that contain exactly one positive literal. A fact $p(t_0, \ldots, t_{n-1})$ is also written[1]
$$\mathtt{p}(t_0, \ldots, t_{n-1}). \tag{7.15}$$
2. A *rule* consists of exactly one positive literal and at least one negative literal. Rules are written as
$$\begin{aligned} & (\neg\psi_0) \vee (\neg\psi_1) \vee \cdots \vee (\neg\psi_{m-1}) \vee \mathtt{p}(t_0, \ldots, t_{n-1}) \\ = \ & \{\neg\psi_0, \neg\psi_1, \ldots, \neg\psi_{m-1}, \mathtt{p}(t_0, \ldots, t_{n-1})\} \\ = \ & \mathtt{p}(t_0, \ldots, t_{n-1})\mathtt{:-}\ \psi_0, \ldots, \psi_{m-1}. \end{aligned} \tag{7.16}$$
3. A *goal* consists of negative literals only:
$$\mathtt{?-}\ \psi_0, \ldots, \psi_{n-1}. \tag{7.17}$$

[1] The dot "." belongs to the syntax definition of a fact. The reason why we write p for p will become clear later.

4. The *empty clause*, $\Box$, has no literal:

$$\{\} = \Box. \tag{7.18}$$

The huge benefit now is that SLD-Resolution is sound and complete with respect to Horn Logic and, especially, that there is a simple and (more or less) efficient algorithm to carry out proofs. This is a more than just fair compensation for losing some expressiveness.

Logic Programs
A *literal* is a positive or a negative atomic formula. A set of literals is called a *clause*. A *Horn clause* is a clause with at moste one positive literal. A set of Horn clauses is a *predicate definition*. A set of predicate definitions is called a *logic program*.

7.2.2 Induction of Logic Programs

The method by which a machine acquires a logic program to describe a set of facts (or HOL-formulas) is called *inductive logic programming*. By the nature of HOL, this learning paradigm focusses on *rule induction*. In terms of knowledge representation and discovery it means: The sample can be formulated as a set of *facts* and the background knowledge can be formulated using a terminological framework as it is provided by some expert. Hypotheses are formulated with respect to the same terminology and, therefore, are easy to interpret.

Example 7.2 Here is an example of what it means for a knowledge base to "naturally" or "intuitively" represent an expert's knowledge. Let there be two people, *rhineheart* and *anderson*. They both work for the same company, *metacortex*, a computer firm running an expert system called the *oracle*. While *rhineheart* is a `manager`, *anderson* is a `programmer`. Yet, both of them may `use` the *oracle*. *smith*, who is not a *metacortex* employee, may not `use` use the *oracle* program.
The question is: Who, in general, may use a *metacortex* program? We represent factual knowledge Π as follows:

$$\left\{\begin{array}{ll} \texttt{manager}(\mathit{rhinehart}). & \texttt{programmer}(\mathit{anderson}). \\ \texttt{employer}(\mathit{rhinehart}, \mathit{metacortex}). & \texttt{employer}(\mathit{anderson}, \mathit{metacortex}). \\ \texttt{runs}(\mathit{metacortex}, \mathit{oracle}). & \texttt{program}(\mathit{smith}). \end{array}\right\}$$

Our sample $\mathbf{s}$ looks as follows:

$$\left\{\begin{array}{l} \langle \texttt{may\_use}(\mathit{anderson}, \mathit{oracle}), \mathbf{1}\rangle\,, \\ \langle \texttt{may\_use}(\mathit{rhineheart}, \mathit{oracle}), \mathbf{1}\rangle\,, \\ \langle \texttt{may\_use}(\mathit{smith}, \mathit{oracle}), \mathbf{0}\rangle \end{array}\right\}$$

Obviously, someone may operate programs of a company in which he is employed only:

$$\begin{array}{l} \texttt{may\_use}(\mathit{Program}, \mathit{Person})\texttt{:-} \\ \quad \texttt{runs}(\mathit{Company}, \mathit{Program}), \\ \quad \texttt{employer}(\mathit{Company}, \mathit{Person}). \end{array}$$

This rule forms terminological knowledge—if present. If it is not present in Π, then it forms a hypothesis H which together with Π could explain $\mathbf{s}$. ●

Exercise 7.3 (♦) Prove that $\Pi \cup H \mathrel{|\!\!\approx} \varphi \iff \langle \varphi, \mathbf{1} \rangle \in \mathbf{s}$!

Note that the rule is more general than the set of facts: First, it makes use of *variables* rather than atoms and second, some knowledge appears to be *dispensable* as it seems that it is irrelevant whether a person is a manager or a programmer—as long as he works for the according company. Logic programs live from *variables*. What may sound trivial, has two crucial consequences: From the computational point of view, variables are something we would like to avoid whenever possible. But from a logic point of view, (common) variables *connect* literals. Literals that are not connected are independent, and it could well be that parts of a rule containing irrelevant/independent variables only can be pruned away. As we have seen in the previous section, logic programs are a simple method to formalise HOL theories. So if we use HOL as a representation language for samples and background knowledge it would be desirable to make the machine learn a logic program by it self. Russell supports the idea of induction as the origin of scientific axioms, [Russell, 1992, Russell, 1995], and Polya worked on inductive reasoning and scientific discovery by analogy,[György, 1968]. Popper, on the other hand, claims that scientific proof theory must be based on the deductive test, since induction itself is invalid, [Popper, 2002]. We shall see later, that Popper's claim is at least weakly supported by our notion of induction. For now, it shall suffice to give a common sense description of inductive generalisation:

Inductive Generalisation
Machine learning in the context of terminological knowledge means to inductively generalise over a set of observed facts.

We know try to formulate the idea a bit more formally:

ILP Learning Problem

Definition 7.8 — ILP Learning Problem.
Let Π be a satisfiable set of Σ-formulas and Σ the signature defined by it. We call Π the (terminological) *background knowledge*.
A *sample* is a set of atomic Σ-formulas φ_i together with a truth value that describes their desired satisfiability conditions:

$$\begin{aligned} \mathbf{s} &= S(m,t) \\ &= \left\{ \langle \varphi_i, t(\varphi_i) \rangle : \; t(\varphi_i) = \downharpoonleft\varphi_i\downharpoonright^{\mathfrak{A}}_{\alpha} \in \mathbf{2} \text{ for } i \in \mathbf{m} \right\} \end{aligned} \tag{7.20}$$

The *learning problem* is to find a $H \subseteq \mathrm{Fml}_{\mathrm{FOL}}$ for which

$$\Pi \cup H \mathrel{|\!\!\approx} \varphi_i \iff t(\varphi_i) = \mathbf{1} \tag{7.21}$$

holds. ●

Therein, $\mathfrak{A}$ is the Herbrand Algebra with signature Σ defined in Π and $\downharpoonleft\cdot\downharpoonright^{\mathfrak{A}}_{\alpha}$ is the Herbrand interpretation. A learning problem would not be a learning problem, if equation (7.21) was satisfied for $H = \emptyset$, i.e. if $\Pi \mathrel{|}\!\approx \varphi_i \iff t(\varphi_i) = \mathbf{1}$ already. So we are left with the task to induce new formulas H such that XXX???XXX er with Π we can prove what we want to be true and disprove what we want to be wrong. We restricted ourselves to HOL, so we can reformulate: We want to find H such that

$$\Pi \cup H \vdash_{\text{SLD}} \varphi_i \iff t(\varphi_i) = \mathbf{1}$$

Note that there are a few things different here: First, $S(m, t)$ is missing a μ. The choice function implemented by S here is *deterministic* and defined by SLD-resolution: The set of examples is a *sequence* of facts. Second, the union $\Pi \cup H$ suggests a *monotonic* refinement of knowledge. We know that if $\Pi \cup \{\varphi\}$ is unsatisfiable, so is $\Pi \cup H \cup \{\varphi\}$.

Exercise 7.4 ◊ Read up about the "Compactness Theorem".

Knowledge discovery tries to *generalise* in order to find new knowledge; this means, that the deductive closure of Π should be a subset of $\Pi \cup H$. Together it means that knowledge discovery by logic induction is at most monotonic in the sense that it preserves falsity. This observation might help mollify Sir Popper's objections.

7.2.3 Entailment, Generality and Subsumption

Generality comes in many flavours. An equivalence relation is more general than another, if it is a superset. Most sets are considered to be more general than their subsets. In Heyting algebras, any expression $x \sqcup y$ is more general than both x and y: The maximum of two natural numbers is greater or equal than both, the union of two sets is greater or equal than both, and the disjunction of two propositional variables is true under three assignments whereas each single variable is true under only two of them.

Generality
One thing is *more general* than another if it is bigger in some way.

It appears just natural to define a relation of generality on sets of formulas along the same line:

$$\Phi \text{ is more general than } \Psi \;:\iff\; \Phi \sqcap \Psi = \Phi$$

What is it that remains the same for a set of formulas if we add a more special one? It is the set of formulas entailed by it.

Theory, $Th(\Phi)$

Definition 7.9 — Theory, $Th(\Phi)$.
The *theory* of a set Φ of formulas is the closure of Φ under $\mathrel{|}\!\approx$:

$$Th(\Phi) := \{\varphi : \Phi \mathrel{|}\!\approx \varphi\}$$

●

Exercise 7.5 (◊◊) Give examples for finite and infinite theories!—Think of Ter^0_Σ, s_A, and function symbols in Σ.

Exercise 7.6 (♦) Just for thinking: Given a theory Θ, are there different Φ_i with $Th(\Phi_i) = \Theta$? Is there a smallest one? Is there exactly one smallest?

Let us briefly consider what it means for one theory to be more general than another:

Example 7.3 Let us define a relation of generality as follows: Φ is more general than Ψ, if $Th(\Psi) \subseteq Th(\Phi)$. By equation (7.22),

$$Th(\Psi) \subseteq Th(\Phi) \iff \forall \varphi : \Psi \mathrel{|\!\approx} \varphi \Longrightarrow \Phi \mathrel{|\!\approx} \varphi$$

But if $\Phi \mathrel{|\!\approx} \varphi$ and $\Psi \mathrel{|\!\not\approx} \varphi$, then $\Psi \mathrel{|\!\approx} (\neg\varphi)$. And since $Th(\Psi) \subseteq Th(\Phi)$, $\Phi \mathrel{|\!\approx} (\neg\varphi)$, too. Hence,

$$\Phi \mathrel{|\!\approx} \varphi \quad \text{and} \quad \Phi \mathrel{|\!\approx} (\neg\varphi) \tag{7.22}$$

which is a rather uncomfortable situation. It seems we have been caught in a logic trap. ●

The property of equivalence is unaffected, though.

Equivalence, $\Phi \cong \Psi$

Definition 7.10 — Equivalence, $\Phi \cong \Psi$.
Let there be two formula sets $\Phi \neq \Psi$. Φ and Ψ are called *equivalent*, written $\Phi \cong \Psi$, if their corresponding theories are the same: $Th(\Phi) = Th(\Psi)$. ●

It is nice to compare sets of formulas—but what we have to do is to induce sets of formulas by stepwise induction of single formulas. This requires a relation of generality between single formulas.

Example 7.4 Let Π consist of the following facts:

$$\Pi = \{\mathit{colour}(■, \mathit{black}), \mathit{colour}(■, \mathit{black})\}$$

Let the sample be as follows:

$$S = \{\mathit{shape}(■, \mathit{square}), \mathit{shape}(■, \mathit{square})\}$$

What do we need in order to infer the examples from Π? Well, all black objects are squares:

$$h_0 = \mathit{colour}(X, \mathit{black}) \longrightarrow \mathit{shape}(X, \mathit{square})$$

This is true in the sense that $\Pi \cup \{h_0\} \models S$. And it remains true even if we add another observation: $\Pi \cup \{h_0\} \models S \cup \{\mathit{colour}(\bigcirc, \mathit{white})\}$. On the other hand, the observation of a white circle allows us to reformulate h_0:

$$h_1 = \neg\mathit{colour}(X, \mathit{white}) \longrightarrow \mathit{shape}(X, \mathit{square})$$

Obviously, there are *two* different hypotheses both of which are compatible with our background knowledge and both of which are able to describe all our observations. But what happens if there appears

$$\mathit{colour}(■, \mathit{light}) \; ?$$

We observe a strange phenomenon:

$$\Pi \cup \{h_0\} \not\models \mathit{shape}(■, \mathit{square}) \quad \text{whereas} \quad \Pi \cup \{h_1\} \models \mathit{shape}(■, \mathit{square})$$

By common sense it is clear that every model of h_0 is a model of h_1, too—but *not* vice versa! And this means, that h_0 *entails* h_1:

$$\{h_0\} \mathrel{|\!\approx} h_1$$

By abuse of language we abbreviate and write $h_0 \mathrel{|\!\approx} h_1$. ●

This leads to the following definition of generality:

Generality of FOL-Formulas, $\varphi \mathrel{|\!\precsim} \psi$

Definition 7.11 — Generality of FOL-Formulas, $\varphi \mathrel{|\!\precsim} \psi$.
Let φ, ψ be formulas. φ is called *more general than ψ (with respect to Π)*, if

$$\Pi \cup \{\varphi\} \mathrel{|\!\approx} \psi$$

In other words: whenever φ is valid, so is ψ. If φ is more general than ψ we say that φ *subsumes* ψ, written $\varphi \mathrel{|\!\precsim} \psi$. ●

Don't be confused about the symbol $\mathrel{|\!\precsim}$: It does not means that φ is somehow *less* than ψ but rather that the set of models of φ is a subset of the set of models of ψ.

Exercise 7.7 (◊) Prove that

$$\varphi \mathrel{|\!\precsim} \psi \iff \left\{\langle\mathfrak{A}, \alpha\rangle : \downharpoonright\varphi|^{\mathfrak{A}}_{\alpha} = \mathbf{1}\right\} \subseteq \left\{\langle\mathfrak{A}, \alpha\rangle : \downharpoonright\psi|^{\mathfrak{A}}_{\alpha} = \mathbf{1}\right\} \tag{7.23}$$

Generality is a concept that can be defined in many different ways. Let us, even though there is no such thing as a distribution in FOL, just try a little Gedankenexperiment.

Example 7.5 Suppose there is a set Π of n Horn formulas and a distribution μ on Π. We define a μ–relative generality of a Horn formula set Φ as

$$\mathit{gen}^{\Pi}_{\mu}(\Phi) = \mu^n(\{\psi \in \Pi : \Phi \mathrel{|\!\approx} \psi\})$$

Then, we call a set Φ of Horn formulas more general than a set Ψ of Horn formulas, if its generality is greater than the generality of Ψ:

$$\Phi \mathrel{|\!\precsim}^{\Pi}_{\mu} \Psi :\iff \mathit{gen}^{\Pi}_{\mu}(\Phi) \geq \mathit{gen}^{\Pi}_{\mu}(\Psi)$$

This relation has an interesting property:

$$\forall\sigma \in \Pi : (\Phi \mathrel{|\!\approx} \sigma \longrightarrow \Psi \mathrel{|\!\approx} \sigma) \implies \Psi \mathrel{|\!\precsim}^{\Pi}_{\mu} \Phi \tag{7.29}$$

$$\text{but} \quad \Psi \mathrel{|\!\precsim}^{\Pi}_{\mu} \Phi \not\Longrightarrow \forall\sigma \in \Pi : \Phi \mathrel{|\!\approx} \sigma \longrightarrow \Psi \mathrel{|\!\approx} \sigma \tag{7.30}$$

It means that $\mathrel{|\!\precsim}^{\Pi}_{\mu}$ is complete with respect to $\mathrel{|\!\approx}$, but it is not correct. ●

Exercise 7.8 (♦) Prove.

What could be the motivation behind defining such a strange measure of generality? As you know, the statemant that showing $\Phi \mid\!\approx \varphi$ is not trivial in FOL is a bold understatement. But in the example above, Π is finite and all the formulas we are concerned with are Horn formulas. The measure μ can be considered as a value describing a formula's *importance*—be it whatever you like. It just means that if $\mu(\{\sigma_0\}) \geq \mu(\{\sigma_1\})$ it is more important to be able to show σ_0 than it is to show σ_1. To determine $gen_{\mu}^{\Pi}(\Phi)$, try to prove one σ after another; and for each successful proof of σ, we add $\mu(\{\sigma\})$ to the generality of Φ. We can do the same for Ψ. Even though a more general formula set may be not correct, it allows to derive σ in most "important" cases. This way gen_{μ}^{Π} is a heuristic measure that may well help to efficiently find a theory that is able to describe most the important cases—which quite often is good enough.[2]

We conclude that it is a very good idea to look out for a cheap version of $\mid\!\preccurlyeq$; some relation that we can confirm, in the best case, by syntactically means only and from which we can more or less reliably infer whether one formula entails another. Lucky us, there is such a subsumption relation:

θ–subsumption, $t \mid\!\lhd t'$

Definition 7.12 — θ–subsumption, $t \mid\!\lhd t'$.
Let $t, t' \in \mathrm{Ter}_{\mathrm{FOL}}$ and let λ, λ' be literals from $\mathrm{Fml}_{\mathrm{FOL}}$.
A term t θ-*subsumes* a term t', iff there is a substitution θ such that $t\theta$ becomes t (the same applies to literals):

$$t \mid\!\lhd t' :\Longleftrightarrow \exists\theta : t\theta = t' \quad \text{and} \quad \lambda \mid\!\lhd \lambda' :\Longleftrightarrow \exists\theta : \lambda\theta = \lambda' \tag{7.31}$$

A term or literal v is called a generalization over a set of terms or literals w_i, iff for every w_i, v subsumes w_i:

$$v \mid\!\lhd \{w_0, \ldots, w_{n-1}\} :\Longleftrightarrow \forall i \in \mathbf{n} : \; v \mid\!\lhd w_i.$$

●

Exercise 7.9 (♦) Prove that for literals λ, λ', $\lambda \mid\!\lhd \lambda'$ iff $\lambda \mid\!\approx \lambda'$!

θ-subsumption appears to be a very nice property that can be evaluated very efficiently by simple term unification. But this holds for simple terms or literals only. We are not able to decide whether for two *non-unary clauses* one subsumes the other yet. Let there be two Horn clauses of equal length:

$$\begin{aligned} \varphi = \kappa_0 \quad &\texttt{:-} \quad \kappa_1, \ldots, \kappa_{n-1} \\ \psi = \lambda_0 \quad &\texttt{:-} \quad \lambda_1, \ldots, \lambda_{n-1} \end{aligned}$$

Then it seems just reasonable to argue that $\varphi \mid\!\lhd \psi$, if there is a unifier such that $\varphi\theta\psi$, or, in detail, if

$$\forall i \in \mathbf{n} : \; kappa_i\theta = \lambda_i$$

[2] As always, there remains to explain where μ comes from. But this is an issue that we will come back to in the next chapter.

The nice thing is that Horn clauses have at most one positive literal each. This makes two clauses of different length still look a bit like the same:

$$\begin{aligned}\varphi = \kappa_0 \quad &\texttt{:-} \quad \kappa_1, \ldots, \kappa_{m-1}\\ \psi = \lambda_0 \quad &\texttt{:-} \quad \lambda_1, \ldots, \lambda_{n-1}\end{aligned}$$

Assume that $m < n$. Let us examine two different cases: First, we assume that

$$\forall i \in \mathbf{m} : \lambda_i \theta = \kappa_i \tag{7.32}$$

That's a good start, but: ψ cannot be more general than φ for another reason: In order to conclude $\kappa_0 = \lambda_0\theta$, ψ requires $\lambda_m\theta \wedge \cdots \wedge \lambda_{n-1}\theta$ to be true, too. Hence ψ has stronger restrictions on $\lambda_0\theta$ than φ has on κ_0 which makes the set of models of ψ a subset of the set of models of φ.
Second, we assume the reverse case:

$$\forall i \in \mathbf{m} : \kappa_i \theta = \lambda_i \tag{7.33}$$

Up to m, it is again the case that φ subsumes ψ literalwise. Additionally, once $\psi\theta$ is satisfiable, φ is satisfiable, too. This is a much better concept for describing the generality relation between two formulas.

θ–subsumption, $\varphi \mathrel{|\!\lhd} \psi$

Definition 7.13 — θ–subsumption, $\varphi \mathrel{|\!\lhd} \psi$.
Let $\varphi, \psi \in \mathrm{Fml}_{\mathrm{HOL}}$. φ *θ-subsumes* ψ, if for a subset of ψ, each of its literals is subsumed by a literal in φ:

$$\varphi \mathrel{|\!\lhd} \psi \quad :\Longleftrightarrow \quad \varphi\theta \subseteq \psi \tag{7.34}$$

Note that $\varphi \mathrel{|\!\lhd} \psi$ implies that $\varphi \mathrel{|\!\approx} \psi$ but not vice versa. ●

Exercise 7.10 (♦) Show that $\varphi \mathrel{|\!\approx} \psi \;\not\Longrightarrow\; \varphi \mathrel{|\!\lhd} \psi$.

Recall our hierarchical representation of the domain of geometric shapes in section 4.4.

Example 7.6 We model a part of this concept hierarchy as a small logic program Π:

$$\begin{aligned}\mathit{shape}(X, \mathit{tetragon}) \quad &\texttt{:-} \quad \mathit{shape}(X, \mathit{square}).\\ \mathit{shape}(X, \mathit{tetragon}) \quad &\texttt{:-} \quad \mathit{shape}(X, \mathit{rhombus}).\\ \mathit{shape}(X, \mathit{angled}) \quad &\texttt{:-} \quad \mathit{shape}(X, \mathit{tetragon}).\\ \mathit{shape}(X, \mathit{angled}) \quad &\texttt{:-} \quad \mathit{shape}(X, \mathit{triangle}).\end{aligned}$$

Suppose we also have knowledge about form and colour of $\bullet$, $\blacksquare$, $\Diamond$, $\blacklozenge$, $\triangle$ and $\blacktriangle$. Then, we observe the following sample:

$$\mathbf{s} \;=\; \left\{ \langle \bullet, \mathbf{0} \rangle, \langle \blacksquare, \mathbf{1} \rangle, \langle \Diamond, \mathbf{0} \rangle, \left\langle \blacklozenge, \mathbf{1} \right\rangle, \langle \triangle, \mathbf{0} \rangle, \langle \blacktriangle, \mathbf{1} \rangle \right\}$$

which provides us with information about instances of a concept *black_polygon*. Let us compare three possible hypotheses:

$$\begin{array}{lll} h_0 & := & \mathit{black\_polygon}(X)\texttt{:-}\ \mathit{colour}(X, \mathit{black}), \mathit{shape}(X, \mathit{tetragon}). \\ h_1 & := & \mathit{black\_polygon}(X)\texttt{:-}\ \mathit{colour}(X, \mathit{black}), \mathit{shape}(X, \mathit{angled}). \\ h_2 & := & \mathit{black\_polygon}(X)\texttt{:-}\ \mathit{shape}(X, \mathit{tetragon}). \end{array}$$

Then,

$$h_2 \mathrel{|\!\lhd} h_0 \text{ but } h_2 \not\mathrel{|\!\lhd} h_1 \text{ and } h_1 \not\mathrel{|\!\lhd} h_0$$

but $h_2 \mathrel{|\!\approx} h_0$ and $h_2 \mathrel{|\!\approx} h_1$ and $h_1 \mathrel{|\!\approx\!|\!\lhd} h_0$. ●

It is nice to know that θ-subsumption is sufficient for entailement. But we did not take into account the entirety of our background knowledge and the provided sample. Since

$$\Pi \cup \{h_0\} \mathrel{|\!\approx} \mathit{black\_polygon}(\triangle)$$

and $\triangle$ obviously is *not* a black polygon, h_0 is *too general*: It allows to infer a proposition that S tells us to be false. h_1 seems to model S pretty well:

$$\Pi \cup \{h_1\} \mathrel{|\!\approx} \mathit{black\_polygon}(X) \iff X \in \{\blacksquare, \blacklozenge, \blacktriangle\}$$

Finally, h_0 is too *specific*, because $\Pi \cup \{h_0\} \mathrel{|\!\approx} \mathit{black\_polygon}(X) \iff X \in \{\blacksquare, \blacklozenge\}$, i.e.

$$\Pi \cup \{h_0\} \not\mathrel{|\!\approx} \mathit{black\_polygon}(\blacktriangle).$$

In the last example we considered the satisfaction sets of hypotheses in order to find a description of their generality. We already know this idea: In example 7.5 we defined the generality of a formula based on the importance of all formulas that can be deduced. With a slight adjustment[3]

$$\mathit{gen}_\mu^{\Pi\trianglelefteq\mathbf{s}}(\{h_i\}) := \mu^n(\{\varphi : \Pi \cup \{h_i\} \mathrel{|\!\approx} \varphi \Longleftrightarrow \langle\varphi, \mathbf{1}\rangle \in \mathbf{s}\})$$

we obtain a measure based on the size of the satisfaction set of h_i. If we assume $\mu(\{\varphi\}) = \frac{1}{|\mathbf{s}|}$, then

$$\mathit{gen}_\mu^{\Pi}(\{h_0\}) = \frac{4}{6},\ \mathit{gen}_\mu^{\Pi}(\{h_1\}) = \frac{6}{6},\ \text{and } \mathit{gen}_\mu^{\Pi}(\{h_2\}) = \frac{1}{6}$$

Interestingly, *gen* appears rather to model an error measure than a measure of generality. This is due to the fact that in the calculation above we did not simply count the number of derivable formulas but rather the number of *correctly* derivable formulas where correctness is expressed *relative to* **s**. We define:

Relative Generality of FOL-Formulas, $\varphi \mathrel{|\!\precsim}_\Gamma \psi$

Definition 7.14 — Relative Generality of FOL**-Formulas,** $\varphi \mathrel{|\!\precsim}_\Gamma \psi$.
Let φ, ψ be FOL-formulas, Π a logic program, and Γ be a set of FOL-formulas. φ is called *more general than ψ relative to* Γ, written $\varphi \mathrel{|\!\precsim}_\Gamma \psi$, if

$$\forall\chi \in \Gamma :\ \Pi \cup \{\psi\} \mathrel{|\!\approx} \chi \implies \Pi \cup \{\varphi\} \mathrel{|\!\approx} \chi$$

[3] Note the $\trianglelefteq$-symbol!

Whenever a "test case" χ is provable by ψ, so it can be proved by φ. If φ is more general than ψ relative to Γ we also say that φ Γ-*subsumes* ψ, written $(\varphi \unlhd \Gamma) \mid\!\preccurlyeq (\psi \unlhd \Gamma)$. ●

Now we have a set of different working definitions of generality all of which help forming a partially ordered set of hypotheses.

Exercise 7.11 ⧫ Show that $\mid\!\preccurlyeq$ as in definition 7.11, as in definition 7.14 and $\mid\!\lhd$ are partial order relations on the set of Horn clauses.

A partial order relation can be defined equationally, too. Usually, one defines $x \sqsubseteq y :\Longleftrightarrow x \sqcup y = y$ in relation algebra. What would be the according meet and/or join operators in a lattice with any of our partial order relations? This question is not of academic interest only. Example 7.6 is an impressive demonstration of what it means to *overgeneralise* or *overspecialise.* The safest thing (and we shall see, that even this is not safe enough always) is to make steps as small as possible. When we are looking for a formula subsuming a set of formulas, then it would be wise to try the *most specific generalisation* first, and if we want to specialise from a set of formulas, it seems a good idea to start off with the *most general specialisation.* Since we are mostly concerned with generalisation and since θ-subsumption worked quite well (it's so easy to compute!), we define the *least general (*θ*-) generalisation* as follows:

Least General Generalisation (lgg), $\curlyvee$

Definition 7.15 — Least General Generalisation (lgg), $\curlyvee$.
Let φ, ψ, χ, ξ be terms, literals, or FOL-Formulas. We say that φ is a least general generalsiation of χ and ξ, if every other generalisation ψ of χ and ξ as a generalsiation of φ, too.

$$\begin{aligned} \varphi = \chi \curlyvee \xi \quad :\Longleftrightarrow \qquad & \varphi \mid\!\preccurlyeq \{\chi, \xi\} \\ & \wedge \forall \psi :\ \psi \mid\!\preccurlyeq \{\chi, \xi\} \Longrightarrow \psi \mid\!\preccurlyeq \varphi \end{aligned} \tag{7.35}$$

For $\xi_0 \curlyvee \cdots \curlyvee \xi_{n-1}$ we write $\bigcurlyvee \{\xi_0, \ldots, \xi_{n-1}\}$. If $\mid\!\preccurlyeq$ is defined by $\mid\!\lhd$, we write $\triangledown$ for $\curlyvee$, [**?**]. ●

Note that the least general generalisation is nothing else than a least upper bound. Hence the definition carries over to *arbitrary subsumption relations*! For our purpose it suffices to consider two alternative versions:

1. Using θ-subsumption $\mid\!\lhd$ as an order relation, the least general generalisation is denoted by $\triangledown$.

2. We use the notation $\mid\!\preccurlyeq$ as order relation with join operator $\curlyvee$ also to describe the subsumption with respect to *implication* ($\mid\!\approx$).

The $\triangledown$ can be considered the dual of the most general unifier μ on the set of terms or literals.

Exercise 7.12 (◊) Prove that for φ, ψ in $\mathrm{Ter}_{\mathrm{FOL}}$ or literals, $\varphi \mid\!\lhd \psi \Longleftrightarrow \{\varphi, \psi\}\, \mu = \psi$.

Least General Generalisation
Looking for new knowledge that can describe things we weren't able to describe before, we need to refine our theory by generalising our hypotheses of the target concept.
Generality is a relation that is easy to describe not to determine. The trick is to find a feasable definition of generaltiy that is sufficiently close to what our intended measure of generality is. Since knowledge disovery is a step-wise search process, it is very important to define operators that allow for small steps so as not to become prone to overgeneralsiation.

7.3 Heuristic Rule Induction

Suppose there is a set of facts that we want to describe. If all these facts share a common term structure it may be possible to find a factual representation of a hypotheses by computing the lgg or even more general facts than the lgg. But as we have seen in example 7.6 already we usually look for rules rather than for facts: Sometething is x-ish, *if* it is y-ish and z-ish, or if it's not an a but a b. Quite actually, we do not look for a *singe* rule h but rather for a *set* of rules H (see definition 7.8).
Therefore, in order to discover new (rule-based) knowledge, we start off with a set of facts $\mathbf{s}$ and some background knowledge Π and an (empty) set H. We then (repeatedly) refine H until $\Pi \cup H$ describes the target sufficiently well.

7.3.1 Refinement Operators on H

Let us consider clauses first. In simplified notation, a rule clause looks as follows:

$$\begin{array}{lcl} \texttt{pred}(\vec{t_0}) & \texttt{:-} & \texttt{pred}_1(\vec{t_1}), \\ & & \ldots \\ & & \texttt{pred}_{k-1}(\vec{t}_{k-1}), \texttt{pred}_k(\vec{t_k}), \texttt{pred}_{k+1}(\vec{t}_{k+1}), \\ & & \ldots \\ & & \texttt{pred}_{n-1}(\vec{t}_{n-1}). \end{array}$$

where $\vec{t_i}$ are *sequences* of terms; i.e. the length $\ell(\vec{t_i})$ is the arity of the i-th predicate symbol $\texttt{pred}_i$. Let us assume that $\vec{X_i}$ is the set of (free) variables in $\vec{t_i}$. As we know from the definition of θ-subsumption substituting variables from $\bigcup \vec{X_i}$ with new terms makes this rule more specific. Concerning the occurence of literals, we observe:

- dropping a literal $\texttt{pred}_k$ from the rule body makes the rule *more general* and

- adding a literal $\texttt{pred}_n$ from the rule body makes the rule *more specific.*

While literals form clauses, clauses build logic programs. Similary, we can

- *generalise* H by *adding* a rule, thus increasing the set of inferrable formulas

```
$H := \emptyset$;
WHILE $(P \cup N \neq \emptyset)$ DO
{
    $\varphi := choose(P)$
    $H := generalise(H, \varphi)$
    $P := P - \{\varphi\}$
    $\psi := choose(\psi, N)$
    $H := specialise(H, \psi)$
    $N := N - \{\psi\}$
};
```

Figure 7.1: Refining H

- *specialise* H by *deleting* rules which reduces $Th(\Pi \cup H)$.

This motivates a very simple algorithm for refinement of logic programs:
First, extract two sets $E^{\mathbf{0}}$ and $E^{\mathbf{1}}$ from $\mathbf{s}$:

$$E^x \quad := \quad \{\varphi \ : \ \langle \varphi, x \rangle \in \mathbf{s}^x\} \tag{7.36}$$

$E^{\mathbf{1}}$ is called the set of *positive examples*; $E^{\mathbf{0}}$ the set of *negative examples*.
In the second step, determine all those formulas $\varphi \in E^{\mathbf{1}}$ which cannot be deduced and all formulas $\psi \in E^{\mathbf{0}}$ that are provable:

$$P := \left\{\varphi \in E^{\mathbf{1}} : \Pi \cup H \mathrel{|\!\not\approx} \varphi\right\} \qquad N := \left\{\psi \in E^{\mathbf{0}} : \Pi \cup H \mathrel{|\!\approx} \psi\right\}$$

Then, P and N are the sets of formulas which are "*misclassified*" by our program. Therefore, in step three, *refine* H by generalising until all positive examples are covered and specialising it until no negative examples are covered. For an initially empty H, the procedure as shown in figure 7.1 can be applied: Not surprisingly, the crucial part is hidden in implementation of the choice function *choose* and the refinement operations *generalise* and *specialise*.
In the beginning we would start of with generating rules without body literals, i.e. facts which, by finding a "suitable" generalisation should cover as many elements of $E^{\mathbf{1}}$ as possible. Then, add literals to the rue bodies so as to minimise the set N of mistakenly subsumed negative examples. FOIL is an information gain guided heuristical version of this algorithm.

7.3.2 Heuristic Refinement

In [Quinlan, 1991, Quinlan and Cameron-Jones, 1993, Quinlan and Cameron, 1995], Quinlan presents an algorithm for induction of FOL-formulas, called First Order Induction of Logic Programs (FOIL). It adds the idea of information gain (see sections 5.2 and 5.3) to the algorithm in figure 7.1. The resulting algorithm is

```
01  $H := \emptyset$
02  WHILE $E^{\mathbf{1}} \neq \emptyset$ DO
03    $h :=$ p($\vec{X}$):- .
04    WHILE ($E^{\mathbf{0}} \neq \emptyset$) DO
05      $\lambda := \arg\max_\lambda \mathit{FoilGn}(\lambda, h)$
06      $h := h \cup \{\lambda\}$
07      $E^{\mathbf{0}} := \{\varphi \in E^{\mathbf{0}} : h \vdash \varphi\}$
08    DONE
09    $H := H \cup \{h\}$
10    $E^{\mathbf{1}} := E^{\mathbf{1}} - \{\varphi \in E^{\mathbf{1}} : H \vdash \varphi\}$
11  DONE
12  RETURN $H$
```

In line 3, $\texttt{p}(\vec{X})$ with $\texttt{p} : s^n \in \Sigma$ and $n = \ell(\vec{X})$ is the "best predictor" for t. In line 5, λ is a literal chosen from Σ that is added as a body literal to h in line 6.

Figure 7.2: Foil

shown in figure 7.2. Again, there remain a few open questions: First, what does it mean to be a "best predictor"? Second, how does *FoilGn* work, and finally, where do the variables $\vec{X}$ come from?

The best predictor is a literal with maximum support on $E^{\mathbf{1}}$, where support is the relative number of examples subsumed by this literal:

$$\mathit{support}(\varphi, \Phi) := \frac{|\{\psi \in \Phi : \varphi \mathrel{|\!\!\lhd} \psi\}|}{|\Phi|} \tag{7.37}$$

Supposing that all elements in $E^{\mathbf{1}}$ share a common predicate symbol, the case is pretty clear.

The next two questions address the process of literal adding. Let us first briefly discuss the origin of variables in newly introduced body literals. Consider a clause like

$$\begin{array}{lll} \texttt{pred}(\vec{t_0}) & \texttt{:-} & \texttt{pred}_1(\vec{t_0}), \\ & & \ldots \\ & & \texttt{pred}_{k-1}(\vec{t}_{k-1}), \underline{\qquad} \texttt{pred}_{k+1}(\vec{t}_{k+1}), \\ & & \ldots \\ & & \texttt{pred}_{n-1}(\vec{t}_{n-1}). \end{array}$$

Note that the k-th literal is missing—it is the one we are about to add. Which variables can we assume to be of interest in defining the k-th body literal? There are different kinds of variables: If X does not occur anywhere else in the clause, then it is a free, singleton variable. Singleton variables do not impose any restrictions on the satisfaction set of the clause.[4] As a consequence, such variables are rather useless.[5] Then, there are variables that appear in the head or in the body or in both. Variables that appear in the rule head obviously play an important role in the definition of the semantics of the predicate. Next, variables that occur in a literal together with a variable from the head, seem to be important, too; a bit less, maybe—but still "connected" to the head variables. We can iterate this process to define a meausre of variable linkage. FOIL does not take into account considerations like these, but we will rediscover them when talking about inverted entailment. Finally, the *sequence* of literals has a large impact on the procedural semantics of a logic program: It does not make sense to try proving a literal with a free variable if this variable is bound to a value in following literals. This gives rise to the following strategy for finding useful variable bindings: When we add

$$\texttt{pred}_k(X_0, X_1, \ldots, X_{n_k-1})$$

we know that: `pred` must be a known predicate name with arity n_k and at least one X_j must occur somewhere in $\vec{t_i}$, $i \in \mathbf{k}$.[6] We can also bind a Variable to another one using unification: Then, $\texttt{pred}(X_j, t)$ actually becomes $X_j = t$ where X_j and all variables in t must occur in $\vec{t_i}$, $i \in \mathbf{k}$.

Example 7.7 Consider again example 7.6. To make things easier, we change our representation ρ to following unary predicates:

`white`(◊). `white`(△).
`black`(●). `black`(■). `black`(⧫). `black`(▲).
`triangle`(△). `triangle`(▲).
`tetragon`(⧫). `tetragon`(◊). `tetragon`(■).
`circle`(●).

Again, we want to learn `black_polygon` using the sample **s** from example 7.6. The first hypothesis according to line 3 in the algorithm (fig. 7.2) is then

$$h = \texttt{black\_polygon}(X)\texttt{:-}\ .$$

It has a maximum support of 1. On the other hand, it is way to general as it can be satisfied by $\sigma = [X \mapsto \blacksquare]$, $[X \mapsto \triangle]$, or $[X \mapsto \Diamond]$. Therefore, we need to specialise h. The signature provides five different candidates with two different variables each and the old predicates:

$\texttt{black}(X)$, $\texttt{white}(X)$, $\texttt{triangle}(X)$, $\texttt{tetragon}(X)$, $\texttt{circle}(X)$,
$\texttt{black}(Y)$, $\texttt{white}(Y)$, $\texttt{triangle}(Y)$, $\texttt{tetragon}(Y)$, $\texttt{circle}(Y)$,
$\texttt{shape}(X,Y)$, $\texttt{shape}(Y,X)$, $\texttt{shape}(X,Z)$, $\texttt{shape}(Z,X)$, $\texttt{shape}(Z,Y)$,
$\texttt{colour}(X,Y)$, $\texttt{colour}(Y,X)$, $\texttt{colour}(X,Z)$, $\texttt{colour}(Z,X)$, $\texttt{colour}(Z,Y)$,

[4] Which is why, when consulting a clause with a singleton variable in it, a Prolog interpreter throws a warning message. In most cases this is due to a spelling mistake. If it isn't, then it is recommended to use so-called *anonymous* variables "_" instead.

[5] There is an important exception to which we shall come back later

[6] In FOIL, it is assumed that new literals are *appended* to the body; i.e. $k = n$.

As one can see, the second row is entirely useless—why should we introduce a free variable Y that is not connected to the clause head? For similar reasons, $\mathtt{shape}(Z, Y)$ and $\mathtt{colour}(Z, Y)$ are out of question. It seems a good idea to penalise introduction of new variables which makes the unary predicate symbols the prime candidates. ●

We now have to choose between five different predicates modeling two features. A well-known heuristical method is to chose the literal that brings about the largest information gain. Therefore, we need to adjust the definition of entropy (see definition 5.2) and information (see definition 5.3) to our special needs here. The sets we examine here are sets of formulas. They can be ground facts as in sets of examples or entire clauses as in hypotheses that we are about to refine. Our interest lies in the proportion of things that we want to deduce and those we don't (or rather which we want to prove false). The FOIL algorithm works its way through the set of positive and negative examples by iteratively adding to H and in each step delete the positive examples that are covered and negative examples that are excluded (see lines 7 and 10 in figure 7.2). Therefore, the entropy of $\mathbf{s}$ is simply the entropy of the set of examples with respect to their target classification:

$$\text{entropy}_t(\mathbf{s}) \quad := \quad \sum_{x \in \mathbf{2}} \frac{|E^x|}{m} \log_2 \frac{|E^x|}{m} \tag{7.38}$$

where $m = |s|$. Using this entropy measure we can determine the information of a single literal:

$$\begin{aligned} & \text{entropy}_t(\mathtt{p}(\vec{X}), \mathbf{s}) \qquad\qquad (7.39) \\ = \; & \sum_{x \in \mathbf{2}} \frac{|\{\varphi \in E^x : \mathtt{p}(\vec{X}) \mathrel{|\!\approx} \varphi\}|}{|E^x|} \, \text{entropy}_t(\{\varphi \in E^x : \mathtt{p}(\vec{X}) \mathrel{|\!\approx} \varphi\}) \end{aligned}$$

Example 7.8 Let us now determine the information gain for adding the different literals to our (initially empty) hypothesis H with *black_polygon* being our target predicate. First, $\text{entropy}_t(\mathbf{s}) = 1$. Then, we arrange s such that we can easily compute the entropies:

X	$\mathtt{white}(X)$	$\mathtt{triangle}(X)$	$\mathtt{tetragon}(X)$	$\mathtt{circle}(X)$	$t(X)$
◇	**1**	**0**	**1**	**0**	**0**
△	**1**	**1**	**0**	**0**	**0**
•	**0**	**0**	**0**	**1**	**0**
■	**0**	**0**	**1**	**0**	**1**
◆	**0**	**0**	**1**	**0**	**1**
▲	**0**	**1**	**0**	**0**	**1**

We compare:

$$
\begin{aligned}
& \mathrm{entropy}_t(\mathit{white}, \{\Diamond, \triangle, \bullet, \blacksquare, \blacklozenge, \blacktriangle\}) \\
= \; & -\frac{|\{\Diamond, \triangle\}|}{6}\mathrm{entropy}_t(\{\Diamond, \triangle\}) - \frac{|\{\bullet, \blacksquare, \blacklozenge, \blacktriangle\}|}{6}\mathrm{entropy}_t(\{\bullet, \blacksquare, \blacklozenge, \blacktriangle\}) \\
= \; & -\frac{1}{3} \cdot 0 - \frac{2}{3} \cdot \left(-\frac{1}{4}\log_2\frac{1}{4} - \frac{3}{4}\log_2\frac{3}{4}\right) \approx \frac{2}{3} \cdot 0.81 = 0.54
\end{aligned}
$$

Similarly,

$$
\begin{aligned}
& \mathrm{entropy}_t(\mathit{tetragon}, \{\Diamond, \triangle, \bullet, \blacksquare, \blacklozenge, \blacktriangle\}) \\
= \; & -\frac{1}{2}\mathrm{entropy}_t(\{\Diamond, \blacksquare, \blacklozenge\}) - \frac{1}{2}\mathrm{entropy}_t(\{\triangle, \bullet, \blacktriangle\}) \\
= \; & -\frac{1}{2}\left(-\frac{1}{3}\log_2\frac{1}{3} - \frac{2}{3}\log_2\frac{2}{3} - \frac{1}{3}\log_2\frac{1}{3} - \frac{2}{3}\log_2\frac{2}{3}\right) \\
= \; & -\frac{1}{3}\log_2\frac{1}{3} - \frac{2}{3}\log_2\frac{2}{3} \approx 0.92
\end{aligned}
$$

●

Exercise 7.13

◊ Determine the entropies of *black* and *triangle*!

♦ Determine $\mathit{utility}_s(\{P\} \trianglelefteq \{t\})$ for $P \in \{\mathit{white}, \mathit{triangle}, \mathit{circle}, \mathit{tetragon}\}$!

♦ Determine $\mathit{significance}_s(\{P\} \trianglelefteq \{t\})$ for $P \in \{\mathit{white}, \mathit{triangle}, \mathit{circle}, \mathit{tetragon}\}$!

♦♦ Determine

$$
\begin{aligned}
& \mathrm{Red}(\{\mathit{white}, \mathit{black}, \mathit{triangle}, \mathit{tetragon}, \mathit{circle}\} \trianglelefteq \{\mathit{black\_polygon}\}) \\
\text{and} \quad & \mathrm{Cor}(\{\mathit{white}, \mathit{black}, \mathit{triangle}, \mathit{tetragon}, \mathit{circle}\} \trianglelefteq \{\mathit{black\_polygon}\}).
\end{aligned}
$$

In the examples above we implicitely counted the number of possible assignements of a variable to a ground term. The quality of a literal $\mathtt{p}(\vec{X})$ depends on the *number* of possible instantiations of $\vec{X}$ such that it subsumes a positive or negative example. This is a question of *subsumption* rather than of *entailment*. Accordingly we want to choose $\mathtt{p}(\vec{X})$ to maximise the set $\{\varphi \in E^{\mathbf{1}} : \mathtt{p}(\vec{X}) \mathrel{|\!\lhd} \varphi\}$ and minimise the set $\{\varphi \in E^{\mathbf{0}} : \mathtt{p}(\vec{X}) \mathrel{|\!\lhd} \varphi\}$. The reason is clear: Suppose we want to learn a *binary* predicate, e.g. `has_more_edges`. Then, on our domain of six different objects, there are $6^2 = 36$ different bindings that we would have to check.

Example 7.9 The predicate `has_more_edges` has the following meaning: `has_more_edges` : $s^2 \in \Sigma$ and $\rfloor\mathtt{has\_more\_edges}(X, Y)\lfloor^{\mathfrak{A}}_{\alpha} = \mathbf{1} :\Longleftrightarrow \alpha(X) R \alpha(Y)$

where the relation R is defined as follows

$$\begin{array}{c|cccccc} R & \Diamond & \triangle & \bullet & \blacksquare & \blacklozenge & \blacktriangle \\ \hline \Diamond & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} \\ \triangle & \mathbf{0} & \mathbf{1} & \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{1} \\ \bullet & \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \blacksquare & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} \\ \blacklozenge & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} \\ \blacktriangle & \mathbf{0} & \mathbf{1} & \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{1} \end{array} \tag{7.40}$$

This simple table leads to a very interesting observation: If we fix α at the point Y to $\bullet$, i.e. $\alpha := \alpha\,[Y \mapsto \bullet]$, then $\texttt{has\_more\_edges}(X, Y)$ is true for any other instantiation of X—but if we choose $\alpha := \alpha\,[X \mapsto \bullet]$, then $\texttt{has\_more\_edges}(X, Y)$ is satisfiable only if $Y = \bullet$.

The consequence is that a literal can be assigned a measure defined by the number of possible instantiations that are compatible with $\mathbf{s}$. So while in equation (7.39) the information content of $\mathtt{p}(\vec{X})$ was measured in terms of *entailed examples*, we avoid $\models\!\!\approx$ by counting the number of *instantiations* of $\vec{X}$ that are compatible with $\mathbf{s}$. For this reason, we examine the *Herbrand universe*—that is, roughly speaking—the set of all ground terms that we can build over our signature Σ and which we can substitute for variables in formulas. It is the base set A of a Herbrand interpretation $\mathfrak{A}$.[7] Luckily, we do not have any function symbols here so we can't construct terms of bigger than atomic complexity—that makes the base set A of $\mathfrak{A}$ the set of instances for variables. In our example, it means that

$$\downharpoonleft x \downharpoonright^{\mathfrak{A}}_{\alpha} = x \text{ for all } x \in \{\Diamond, \triangle, \bullet, \blacksquare, \blacklozenge, \blacktriangle\} = A$$

Accordingly, we define a variant of FOIL-Gain to describe the utility of adding a literal $\lambda = \mathtt{p}(\vec{X})$ to a clause h as follows:

FOIL-Gain, $\text{gain}_t^{\text{FOIL}}(\lambda, h)$

Definition 7.16 — Foil-Gain, $\text{gain}_t^{\textbf{Foil}}(\lambda, h)$.
The estimated gain of adding literal λ to a clause h is measured by the entropy loss on the set of h–compatible assignments with respect to $\mathbf{s}$:

$$\text{gain}_t^{\text{FOIL}}(\lambda, h) \quad = \quad c_\lambda \cdot \left(\log_2 \frac{p_\lambda}{p_\lambda + n_\lambda} - \log_2 \frac{p_h}{a_h} \right) \tag{7.41}$$

Herein, $-\log_2(p_h/a_h)$ describes information of h:

$$p_h \quad := \quad |\left\{\alpha \in A^{\text{Var}} : \downharpoonleft h \downharpoonright^{\mathfrak{A}}_{\alpha} = \mathbf{1}\right\}|$$

A^{Var} is the set of all assignments, $a_h = |A^{\text{Var}}|$. The expected information after adding λ is described by:

$$\begin{aligned} p_\lambda &:= |\left\{\alpha \in A^{\text{Var}} : \downharpoonleft h \cup \{\lambda\} \downharpoonright^{\mathfrak{A}}_{\alpha} = \mathbf{1}\right\}| \\ n_\lambda &:= |\left\{\alpha \in A^{\text{Var}} : \downharpoonleft h \downharpoonright^{\mathfrak{A}}_{\alpha} = \mathbf{0} \wedge \downharpoonleft h \cup \lambda \downharpoonright^{\mathfrak{A}}_{\alpha} = \mathbf{0}\right\}| \end{aligned}$$

[7] A closer look at Herbrand models is beyond the scope of this chapter; the interested reader is encouraged to read [Huth and Ryan, 2004, Mazzola et al., 2006].

Finally, the term is weighted by a factor

$$c_\lambda \quad := \quad \left| \left\{ \alpha \in A^{\mathrm{Var}} : \downharpoonleft h \downharpoonright^{\mathfrak{A}}_{\alpha} = \mathbf{1} \wedge \downharpoonleft h \cup \lambda \downharpoonright^{\mathfrak{A}}_{\alpha} = \mathbf{0} \right\} \right|$$

describing the "selectivity" of λ. ●

Example 7.10 Let the initial hypothesis be the most general description of our target predicate:

$$H = \{h\} = \{\texttt{has\_more\_edges}(X, Y)\texttt{:-}\ .\}$$

The question is which literal λ we shall add to the rule bude of h. We can choose from all predicates in Σ and equality:

$$\Sigma = \{\texttt{white} : s, \texttt{triangle} : s, \texttt{tetragon} : s, \texttt{circle} : s\}$$

The Herbrand universe (the set of instances we can choose from) is s:

$$A = \left\{\lozenge, \triangle, \bullet, \blacksquare, \blacklozenge, \blacktriangle\right\} = s$$

Since the set of variabes we have to consider is $\{X, Y\}$, there are $6 \cdot 6 = 36$ different assignments α (which by the substitution lemma, equation (7.11)) correspond to 36 different substitutions σ. For each of this substitutions we check the validity of the literals with respect to $\mathtt{s}$. For $h = \texttt{has\_more\_edges}(X, Y)\texttt{:-}\ .$, the matrix in equation (7.40) shows the number of correct instantiations. There are 25 **1**–entries which means that there are 25 different possible substitutions that agree with $\mathtt{s}$. Accordingly, the entropy of h alone can be described by $-\log_2(25/36) \approx 0.53$. For all other predicate symbols, the instantiation of Y is irrelevant. Using formula (7.41), we obtain:

λ	p_λ	n_λ	c_λ	$\mathrm{gain}_t^{\textsc{Foil}}(\lambda, h)$
white(X)	9	3	15	1.11
triangle(X)	6	6	19	−2.84
tetragon(X)	18	0	7	9.47
circle(X)	1	5	24	−49.41

(7.42)

As a result, the biggest gain is delivered by the literal tetragon, which is quite understandable since every tetragon has four edges and there is no object with more than four edges. The resulting hypothesis is then

$$H = \texttt{has\_more\_edges}(X, Y)\texttt{:-}\ \texttt{tetragon}(X).$$

Still, this result is far from being perfect: The fact that $\triangle R \bullet$ is not covered, since $\texttt{has\_more\_edges}(\triangle, \bullet)$ fails. ●

Exercise 7.14 ($\blacklozenge$) Complete the example! First, examine whether $H = h \cup \{\lambda\}$ requires an additional body literal. Then continue with the outer loop of the algorithm in figure 7.2, add a new clause head $\texttt{has\_more\_edges}(\vec{X})$ and continue with adding body literals.

Comparing the outcome of a FOIL-learning process to a decision tree induction process as described in section 5.3 is not straightforward. Here, we deal with *relations* between pairs of variables—namely whether X has more edge than Y. Every unary predicate corresponds to a single feature, but explaining a relation between *two* variables requires information about both variable instantiations. As a consequence, we need to create a table with 36 rows—where for every feature we need two columns to represent all possible bindings of two variables. The target feature t then is defined by

$$t(\langle X, Y\rangle) = \alpha(X) R \alpha(Y)$$

which corresponds to all the entries in equation (7.42). So the information system that we would have to feed into a decision tree induction procedure would be

			wht		tri		trg		crc		hme
	X	Y	X	Y	X	Y	X	Y	X	Y	(X,Y)
1	◊	◊	**1**	**1**	**0**	**0**	**1**	**1**	**0**	**0**	**1**
2	◊	△	**1**	**1**	**0**	**1**	**1**	**0**	**0**	**0**	**1**
3	◊	•	**1**	**0**	**0**	**0**	**1**	**0**	**0**	**1**	**1**
⋮											⋮
18	•	▲	**0**	**0**	**0**	**1**	**0**	**0**	**1**	**0**	**0**
⋮											⋮
36	▲	▲	**0**	**0**	**1**	**1**	**0**	**0**	**0**	**0**	**1**

The resulting tree is shown in figure 7.3

Exercise 7.15

◊ Which leaf node has a non-zero error rate?–Which assignments are wrongly classified as a satisfying instantiation?

⧫ Verify the decision tree in figure 7.3 by and compare to your results from exercise 7.14.

7.4 Inducing Horn Theories From Data

The general idea behind the induction of logic programs is the discovery of relatinoal knowledge. Just as we discovered rough set data analysis to be an abstract version of decision tree induction without a heuristic information gain measure as in decision tree induction, we now examine an unbiased method for inducing Horn theories from sets of examples. In section 7.2.3 we already defined several order relations that mimick a "*more-general-than*"-relation. Based on these order relation we were able to define refinement operators, $\curlyvee$ and $\triangledown$ (see definition 7.15). With generalising sets of terms by $\triangledown$, we already gained a lot—it just requires another little representation shift.

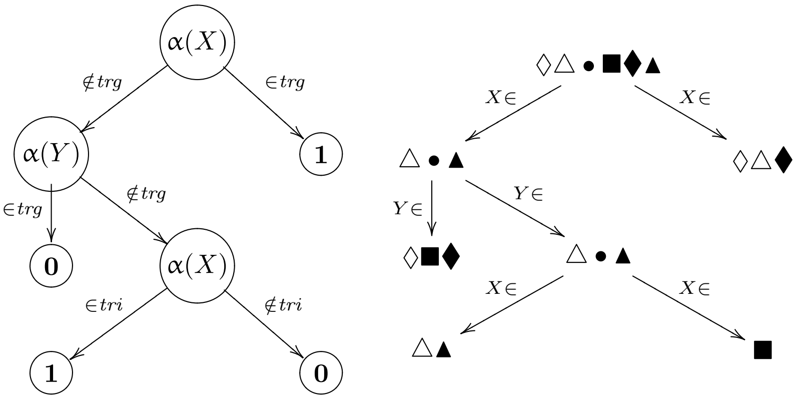


Figure 7.3: A decision tree for example 7.10

Example 7.11 Instead of representing an object's properties by a number predicates, we introduce a *data structure*. While in example 7.8, the signature Σ contained several unary predicate symbols we now add a function symbol $r : \mathbf{2}^4 \rightarrow s$ to Σ'. Therein, each argument represents a predicate from Σ. For example,

$$\rho(\blacksquare) = r(\mathbf{0}, \mathbf{0}, \mathbf{1}, \mathbf{0}, \mathbf{1})$$

The entire set of objects together with their target function then becomes:

$$\begin{array}{llllll}
\texttt{black\_polygon(} & r(\mathbf{1}, & \mathbf{0}, & \mathbf{1}, & \mathbf{0}), & \mathbf{0}) \\
\texttt{black\_polygon(} & r(\mathbf{1}, & \mathbf{1}, & \mathbf{0}, & \mathbf{0}), & \mathbf{0}) \\
\texttt{black\_polygon(} & r(\mathbf{0}, & \mathbf{0}, & \mathbf{0}, & \mathbf{1}), & \mathbf{0}) \\
\texttt{black\_polygon(} & r(\mathbf{0}, & \mathbf{0}, & \mathbf{1}, & \mathbf{0}), & \mathbf{1}) \\
\texttt{black\_polygon(} & r(\mathbf{0}, & \mathbf{0}, & \mathbf{1}, & \mathbf{0}), & \mathbf{1}) \\
\texttt{black\_polygon(} & r(\mathbf{0}, & \mathbf{1}, & \mathbf{0}, & \mathbf{0}), & \mathbf{1})
\end{array}$$

Then, $E^{\mathbf{1}}$ is the set of all objects whose last argument is $\mathbf{1}$ and thus indicates that the structure in the first argument represents an instance of the target concept.

By this trick we transformed a set of formulas with different predicate symbols into a set of of literals with identical predicate symbols. At the same time, the information about the properties of an object is shifted from the assignments of variables to instantiations of a term with function symbol r. This allows us to reason about variable instantiations by comparing ground term structures.

Representation Shifts Revisited
Representing properties of objects by predicates requires a relation for each predicate. The actual proposition about a concrete object then is stored a variable assignment. Reasoning about relations between objects is then a task to be carried out on possible variable assignments.
By shifting the meaning of predicates into term structures, the concrete objects become ground terms. Ground terms can be compared syntactially whithout bothering about variable instantiations.

At this point it becomes clear why we insisted on examining the idea behind representation to such detail. It is very important to understand the huge difference such a small representation change makes in inference—and, even more, in inductive reasoning.

7.4.1 Syntactic Generalisation Revisited

Now that we transformed our knowledge into a set of unifiable literals with identical term structures we can:

- Specialise our knowledge:
  Suppose there are two unifiable clauses φ and ψ containing non-ground terms. Then, there exists a μ such that $\chi = \varphi\mu = \psi\mu$. Furthermore, we know that $\varphi \mid\!\approx \chi$ and $\psi \mid\!\approx \chi$.
- Generalise our knowledge:
  Given two (ground) clauses χ and ξ, there exists a least general generalsiation $\varphi = \chi \triangledown \xi$ and we know that $\varphi \mid\!\approx \chi$ and $\varphi \mid\!\approx \xi$.

Exercise 7.16 ($\Diamond$) Prove that given

$$\Pi = \{\texttt{black\_poygon}(X)\texttt{:- black}(X), \texttt{tetragon}(X)., \texttt{black}(\blacksquare)., \texttt{tetragon}(\blacksquare).\}$$

it holds that $\rfloor\texttt{black\_polygon}(\blacksquare)\lfloor^{\mathfrak{A}}_{\alpha} = \mathbf{1}$.

Let us examine two positive instances of our target concept, black_polygon.

$$\begin{aligned}\chi &= \{\texttt{black\_polygon}(r(\mathbf{0},\mathbf{0},\mathbf{1},\mathbf{0}),\mathbf{1})\}\\ \xi &= \{\texttt{black\_polygon}(r(\mathbf{0},\mathbf{1},\mathbf{0},\mathbf{0}),\mathbf{1})\}\end{aligned}$$

Then,

$$\varphi = \chi \triangledown \xi = \{\texttt{black\_polygon}(r(\mathbf{0},X,Y,\mathbf{0}),\mathbf{1})\}$$

Next, let us examine the set of formulas ψ for which $\Pi \cup \{\varphi\} \mid\!\approx \psi$: There are $2^{\mathbf{2}}$ different assignments instantiating each of the 2 variables X and Y with one of the truth values from $\mathbf{2}$:

α				
X	Y	$r : \mathbf{2}^4$	$x \in A$	t
$\mathbf{0}$	$\mathbf{0}$	$r(\mathbf{0},\mathbf{0},\mathbf{0},\mathbf{0})$	?	$\mathbf{1}$
$\mathbf{0}$	$\mathbf{1}$	$r(\mathbf{0},\mathbf{0},\mathbf{1},\mathbf{0})$	{■, ⧫}	$\mathbf{1}$
$\mathbf{1}$	$\mathbf{0}$	$r(\mathbf{0},\mathbf{1},\mathbf{0},\mathbf{0})$	{▲}	$\mathbf{1}$
$\mathbf{1}$	$\mathbf{1}$	$r(\mathbf{0},\mathbf{1},\mathbf{1},\mathbf{0})$	?	$\mathbf{1}$

We note two interesting things: First, there are two assignments that do not map to a corresponding constant in A, because an object cannot have two shapes at once or none at all. Second, there is one evaluation of the term that can be mapped onto propositions about *two different objects*. It means that $\rfloor\texttt{black\_polygon}(X), \mathbf{1}\lfloor^{\mathfrak{A}}_{\alpha}$ is $\mathbf{1}$ for all $x \in U$ for which $\rho(x) = r(\mathbf{0}, Y, Z, \mathbf{0})$ where $\alpha(X) = \neg\alpha(Y)$. And this happens to be the set $\{■, ♦, ▲\}$, which—hooray!—is the set of all black polygons.

Exercise 7.17 (♦) This maybe is a bit too much work for an exercise, but you should try nevertheless:
Find a representation ρ that is suitable for modeling `has_more_edges`. Then, transform the knowledge according to ρ and examine the hypotheses generated by applying $\triangledown$ to various (wisely chosen) subsets of $E^{\mathbf{1}}$!

After all, resolution is just another ordinary calculus. If we simply try and "mirror" specialising operators to yield generalsiing operators, why shan't we try and invert the resolution rule itself?
Since the resolution rule crusially depends on unification and, thus, on substitutions we need to think about *inverse substitutions*. There is a small problem associated with simply reverse application of substitutions:

Example 7.12 Let $\psi = p(X, a)$ and $\sigma = [X \mapsto a]$. Then

$$\psi\sigma = p(a, a).$$

If we now "revert" σ to $\sigma^{\smile} = [a \mapsto X]$, we suddenly have

$$\varphi = p(X, X)$$

and $\varphi \mid\!\lhd \psi$ (note that $\psi \not\mid\!\approx p(b, b)$, but $\varphi \mid\!\approx p(b, b)$!). In other words, $\psi\sigma\sigma^{\smile} \neq \psi$.
●

Exercise 7.18 Prove that $\psi\sigma\sigma^{\smile} \mid\!\lhd \psi$ but $\psi \not\mid\!\lhd \psi\sigma\sigma^{\smile}$!

In order to avoid this problem, one needs a more technical definition of inverse substitutions involving positions of subterms.

Subterm positions, $t\langle i\rangle$

Definition 7.17 — Subterm positions, $t\langle i\rangle$.
Let $t \in \mathrm{Ter}$ be a term with function symbol $f : s^n \to s \in \Sigma$ and $t_i, i \in \mathbf{n}$ its arguments. Then, $t\langle i\rangle$ is t's i-th argument. By recursion, $t\langle i\rangle\langle j\rangle$ is the j-th argument of the i-th argument of t.
We abbreviate the composition of several position indices by simply concatenating them: $t\langle i_0, \ldots, i_{m-1}\rangle := t\langle i_0\rangle \cdots \langle i_{k-1}\rangle$. ●

Let $\sigma = [X_i \mapsto t_i : i \in \mathbf{n}]$. We then determine the positions of all X_i in t, where each X_i may occur several times and rewrite σ as

$$\sigma = [(X_i, \vec{p_i}) \mapsto t_i : X_i \text{ occurs at } p_i \text{ in } t] \tag{7.43}$$

This allows us to define a position sensititve inversion of substitutions:

Inverse substitution, σ^{-1}

Definition 7.18 — Inverse substitution, σ^{-1}.
Let $t \in \text{Ter}$ and $(X_i, \vec{p_i})$ with $X_i \in \text{Var}, i \in \mathbf{n}$ variables occuring in t at positions $\vec{p_i}$. Let σ be a substitution acting on t. Then, $\sigma^{\smile}$ is called an *inverse substitution*, if

$$t\sigma\sigma^{\smile} = t$$

where for every $t \mapsto (X, \vec{p})$ in σ^{-1} there exists $(X, \vec{p}) \mapsto t$ in σ. ●

Exercise 7.19 (◊) For example 7.12, define σ^{-1}.

Exercise 7.20 (◊◊) Show that any σ^{-1} is a subset of $\sigma^{\smile}$. First, argue with the number of occurences of variables; then give a second proof by comparing the relational properties of σ^{-1} and $\sigma^{\smile}$.

Exercise 7.21 Let $\Phi = \{\varphi_i : i \in \mathbf{3}\}$ with

$$\begin{aligned}
\varphi_0 &= \mathtt{p}(X, f(Y)) \\
\varphi_1 &= \mathtt{p}(g(X), f(g(X)) \\
\varphi_2 &= \mathtt{p}(a, Z)
\end{aligned}$$

♦ Determine all pairwise unifiers for all pairs of formulas in Φ

◊ Determine all unifiers for Φ.

♦ Determine $\chi_{ij} = \varphi_i \triangledown \varphi_j$, for all $i, j \in \mathbf{3}$ and $\bigtriangledown \Phi$.

♦♦ For each lgg χ_{ij} and every formula φ in Φ, determine σ^{-1} such that $\varphi\sigma^{-1} = \chi_{ij}$.

If you are familiar with the Prolog programming language, you might have learned, that clauses are nothing else than just special terms. Given a Horn clause

$$\lambda \leftarrow \lambda_0, \ldots, \lambda_{n-1} = \mathtt{pred}(\vec{X})\mathtt{:-}\ \mathtt{pred}_0(\vec{X}_0), \ldots, \mathtt{pred}_{n-1}(\vec{X}_{n-1}).$$

the internal representation is a binary term with function symbol `:-` , the first argument being the rule head and the second argument the *list* of body literals:

$$\mathtt{:-}\ (\mathtt{pred}(\vec{X}), [\mathtt{pred}_0(\vec{X}_0), \ldots, \mathtt{pred}_{n-1}(\vec{X}_{n-1})]).$$

There remains one problem: How can we unify two lists of different length? Again, the internal representation of Prolog helps a lot. Lists are nothing else than recursive terms where the function symbol represents concatenation:

$$\begin{aligned}
& \mathtt{:-}\ (\mathtt{pred}(\vec{X}), [\mathtt{pred}_0(\vec{X}_0), \ldots, \mathtt{pred}_{n-1}(\vec{X}_{n-1})]) \\
= \quad & \mathtt{:-}\ (\mathtt{pred}(\vec{X}), [\mathtt{pred}_0(\vec{X}_0)|[\mathtt{pred}_1(\vec{X}_1)|[\cdots|[\mathtt{pred}_{n-1}(\vec{X}_{n-1})|\mathtt{nil}]\cdots]]]) \\
= \quad & \leftarrow (\lambda, \bullet(\lambda_0 \bullet (\lambda_1 \bullet (\cdots(\lambda_{n-1} \bullet [\ ])\cdots))))
\end{aligned}$$

Now let there be two clauses $\varphi = \{\kappa, \kappa_0, \ldots, \kappa_{n-1}\}$ and $\psi = \{\lambda, \lambda_0, \ldots, \lambda_{m-1}\}$. Assuming that $m = n$, φ and ψ are unifiable, if κ_i and λ_i are unifiable. Since the order of literals in a clause has no impact on its declarative semantics we can reorder them to allow a match. If $m > n$, we simply unify the matching subterms and then add the remaining literals of ψ with applying the unifier to them.

One problem remains: If there are two clauses χ and ξ and their lgg $\varphi = \chi \triangledown \xi$, it is quite likely that the deductive closure becomes incompatible with Π.

Example 7.13 Again, we need another representation ρ for our example domain. This time, we add predicate symbols *colour* : s^2 and *shape* : s^2 to our signature; similar to the representation we chose in example 7.6. Then, Π becomes

$$\begin{array}{l}
\mathtt{colour}(\bullet, \mathtt{black}).\ \mathtt{colour}(\blacksquare, \mathtt{black}).\ \mathtt{colour}(\Diamond, \mathtt{white}).\\
\mathtt{colour}(\blacklozenge, \mathtt{black}).\ \mathtt{colour}(\triangle, \mathtt{white}).\ \mathtt{colour}(\blacktriangle, \mathtt{black}).\\
\mathtt{shape}(\bullet, \mathtt{circle}).\ \mathtt{shape}(\blacksquare, \mathtt{square}).\ \mathtt{shape}(\Diamond, \mathtt{diamond}).\\
\mathtt{shape}(\blacklozenge, \mathtt{diamond}).\ \mathtt{shape}(\triangle, \mathtt{triangle}).\ \mathtt{shape}(\blacktriangle, \mathtt{triangle}).
\end{array}$$

If we now encounter two ground clauses, namely

$$\begin{array}{rcl}
\chi & = & \mathtt{black\_polygon}(\blacksquare)\mathtt{:-}\ \mathtt{colour}(\blacksquare, \mathtt{black}), \mathtt{shape}(\blacksquare, \mathtt{square}).\\
\xi & = & \mathtt{black\_polygon}(\blacklozenge)\mathtt{:-}\ \mathtt{colour}(\blacklozenge, \mathtt{black}), \mathtt{shape}(\blacklozenge, \mathtt{diamond}).
\end{array}$$

the least general generalisation becomes

$$\varphi = \chi \triangledown \xi \quad = \quad \mathtt{black\_polygon}(X)\mathtt{:-}\ \mathtt{colour}(X, \mathtt{black}), \mathtt{shape}(X, Y).$$

and via $\sigma = [X \mapsto \bullet, Y \mapsto \mathtt{circle}]$ we can conclude $\Pi \cup \{\varphi\} \approx$ *black_polygon*$(\bullet)$.
●

This is a strong argument for *including factual background knowledge* into the process of inducing generalisations. It means that we look for some $\varphi = \chi \triangledown \xi$ for which $\Pi \cup \{\varphi\} \not\approx E^{\mathbf{0}}$. In other words, we are looking for a generalisation that is valid *relative* to E.

Definition 7.19 — Relative Least Generalisation, $\triangledown_\Phi$**.**
Let Φ be a set of ground facts and κ, λ be two literals. Then,

Relative Least Generalisation, $\triangledown_\Phi$

$$\kappa \triangledown_\Phi \lambda \quad := \quad (\{\kappa\} \cup \neg\Phi) \triangledown (\{\lambda\} \cup \neg\Phi) \tag{7.44}$$

The *least general generalisation relative to* Φ is defined in terms of ordinary lgg where each literal is expanded to a full clause with Φ as rule body and the original literal as rule head. ●

The problem with $\triangledown_\Phi$ is that it is applied to *clauses* rather than literals. The definition only covers unary clauses—and single literals are simple terms with their outmost term constructor symbol being e predicate symbol rather than a function symbol.
But the definition works even on clauses with non-empty bodies: Since clauses are sets of literals with no ordering in them, we can rearrange them by, e.g. lexicographc ordering without changing their semantics. Then, the fact that two clauses χ and ξ have a least general generalisation, means that χ and ξ have a common subset of literals. If the positive literals in $\kappa \in \chi$ and $\lambda \in \xi$ have a generalsiation, then the relative least general generalsiation has exactly on positive literal, too (otherwise it is a goal clause):

$$(\kappa \triangledown_\lambda)\mathtt{:-}\ (\chi_0 \triangledown \xi_0), (\chi_1 \triangledown \xi_1), \ldots, (\chi_k \triangledown \xi_k), \varphi_0, \ldots, \varphi_m.$$

where chi_i and ξ_i are pairs of literals from χ and ξ respectively and $\Phi = \{\varphi_i : i \in \mathbf{m}\}$. If χ and ψ are unary clauses (as in the definition above) or if they do not have any predicate symbols in their body literals in common, then the expression from above implodes to the case of unary clauses:

$$(\kappa \triangledown_{\lambda})\texttt{:-}\ \varphi_0, \ldots, \varphi_m.$$

Example 7.14 Let us try and compute $\chi \triangledown_{\Pi} \xi$ with $\chi = \texttt{black\_polygon}(\blacksquare)$, $\xi = \texttt{black\_polygon}(\blacklozenge)$ and Π as in the previous example. Then (with some abbreviations),

$$\begin{array}{rl}
 & \chi \triangledown_{\Pi} \xi = (\{\chi\} \cup \neg\Pi) \triangledown (\{\xi\} \cup \neg\Pi) \\
= & \texttt{bp}(\blacksquare)\texttt{:-} \begin{array}{llllll} \texttt{c}(\bullet,\texttt{b}), & \texttt{c}(\blacksquare,\texttt{b}), & \texttt{c}(\Diamond,\texttt{w}), & \texttt{c}(\blacklozenge,\texttt{b}), & \texttt{c}(\triangle,\texttt{w}), & \texttt{c}(\blacktriangle,\texttt{b}), \\ \texttt{s}(\bullet,\texttt{c}), & \texttt{s}(\blacksquare,\texttt{s}), & \texttt{s}(\Diamond,\texttt{d}), & \texttt{s}(\blacklozenge,\texttt{d}), & \texttt{s}(\triangle,\texttt{t}), & \texttt{s}(\blacktriangle,\texttt{t}). \end{array} \\
\triangledown & \\
 & \texttt{bp}(\blacklozenge)\texttt{:-} \begin{array}{llllll} \texttt{c}(\bullet,\texttt{b}), & \texttt{c}(\blacksquare,\texttt{b}), & \texttt{c}(\Diamond,\texttt{w}), & \texttt{c}(\blacklozenge,\texttt{b}), & \texttt{c}(\triangle,\texttt{w}), & \texttt{c}(\blacktriangle,\texttt{b}), \\ \texttt{s}(\bullet,\texttt{c}), & \texttt{s}(\blacksquare,\texttt{s}), & \texttt{s}(\Diamond,\texttt{d}), & \texttt{s}(\blacklozenge,\texttt{d}), & \texttt{s}(\triangle,\texttt{t}), & \texttt{s}(\blacktriangle,\texttt{t}). \end{array}
\end{array}$$

The inverse substitution required to find the lgg of the two clauses above is $\sigma^{-1} = [\blacksquare \mapsto X, \blacklozenge \mapsto X]$:

$$\chi \triangledown_{\Pi} \xi \quad = \quad \texttt{bp}(X)\texttt{:-} \begin{array}{llllll} \texttt{c}(\bullet,\texttt{b}), & \texttt{c}(X,\texttt{b}), & \texttt{c}(\Diamond,\texttt{w}), & \texttt{c}(X,\texttt{b}), & \texttt{c}(\triangle,\texttt{w}), & \texttt{c}(\blacktriangle,\texttt{b}), \\ \texttt{s}(\bullet,\texttt{c}), & \texttt{s}(X,\texttt{s}), & \texttt{s}(\Diamond,\texttt{d}), & \texttt{s}(X,\texttt{d}), & \texttt{s}(\triangle,\texttt{t}), & \texttt{s}(\blacktriangle,\texttt{t}). \end{array}$$

Since X is the only variable ocuring in the clause head and the rule body, and since all literals that do not contain X are valid ground literals anyway (because they come from Π), we can simplify the clause to

$$\chi \triangledown_{\Pi} \xi \quad = \quad \texttt{bp}(X)\texttt{:-} \begin{array}{llllll} \texttt{c}(\bullet,\texttt{b}), & \texttt{c}(X,\texttt{b}), & \texttt{c}(\Diamond,\texttt{w}), & \texttt{c}(X,\texttt{b}), & \texttt{c}(\triangle,\texttt{w}), & \texttt{c}(\blacktriangle,\texttt{b}), \\ \texttt{s}(\bullet,\texttt{c}), & \texttt{s}(X,\texttt{s}), & \texttt{s}(\Diamond,\texttt{d}), & \texttt{s}(X,\texttt{d}), & \texttt{s}(\triangle,\texttt{t}), & \texttt{s}(\blacktriangle,\texttt{t}). \end{array}$$

This can be reduced further to

$$\varphi := \chi \triangledown_{\Pi} \xi \quad = \quad \texttt{bp}(X)\texttt{:-}\ \texttt{c}(X,\texttt{b}), \texttt{s}(X,\texttt{s}), \texttt{s}(X,\texttt{d}). \tag{7.45}$$

Finally, let us recall definition 7.13. We already found a more general rule as far as variable instantiations are concerned by applying a the least general inverse substitution. But we can generalise *further* by dropping body literals. Since φ has three body literals there are $\mathbf{2}^3 = 8$ possible candidates for rule bodies. As one can imagine,

$$\begin{array}{rcl} h_1 & = & \texttt{black\_polygon}(X)\texttt{:-}\ \texttt{colour}(X,\texttt{black}), \texttt{shape}(X,\texttt{square}). \\ h_1 & = & \texttt{black\_polygon}(X)\texttt{:-}\ \texttt{colour}(X,\texttt{black}), \texttt{shape}(X,\texttt{diamond}). \end{array}$$

are the most promising ones—and it can be found efficiently by calculating the support of the different rules (see equation (7.41)) or the gain of involved literals (definition 7.16). ●

Exercise 7.22 (◊♦) Find all eight possible candidates and compute their support, accuracy, coverage!—It is clear that if a literal occurs in all candidates (here, $\texttt{colour}(X, \texttt{black})$), is indispensable. Compute the gain expected when adding either of the remaining literals!

Exercise 7.23 (♦♦- ♦) The goal is to induce a rule for `has_more_edges`. Π consists of the knowledge from the preceeding examples. Add to Π a set E of ground facts stating that triangles have more edges than circles, that black squares have more edges than triangles and that white tetragons have more edges then everything (Note: E is a set of ground facts—i.e. a set of literals with predicate symbol `has_more_edges` and two arguments which are *objects*!). Then, choose the representation of Π wisely and apply the relative least generalisation procedure in order to induce a rule describing the target!

Least general generalisation appears quite powerful, and relative least generalisations even more. But one big disadvantage of using ∇_{Π} is the possibly huge amount of body literals—which gives rise to the need for a bias. Furthermore, with unrestricted definite clauses as Π, ∇_{Π} is not neccessarily finite (think of variables, function symbols and term construction!). But even if finite, extensional background knowldege may become intractable: The worst case number of literals in a relative least generalisation is $(|\Pi| + 1)^{|E|}$.
For the true knowledge discoverer, all the methods discussed so far lack an important skill: The hypothesis always consists of one single clause or in a set of clauses all of which share a common clause head. Even more disappointing is that the algorithms described cannot really induce *new knowledge*: New knowledge means to be able to discriminate different things that we were not able to tell from each other before, and we might have gained a small progress here. But what it means to tell two different things from each other is more than just saying "■ has more edges than ○".[8] Knowledge means also to be able to describe *why* this is the case—and new qualities usually are described by *new terms*. A really knowing system should be able to say: "■ has more edges than ○, because ■ is a *polygon* and ○ is an *ellipsoid*—and *polygons have at least 3 edges* whereas *ellipsoids have no edges*".
Acquiring knowledgeable new knowledge requires the ability to *invent* new predicates (i.e. predicates). Here we go:

7.4.2 Inverting Resolution

Wouldn't it be nice, if whenever we encounter a sequence or set of similar, ground facts, we could (after a while) generalise? Well, we can: Given $\Phi := \{p(a)., p(b)., p(c)., p(d)\}$ a simple application of lgg results in $\curlyvee \Phi = p(X)$, and, by definition, $p(X) \mathrel{|\!\lhd} \Phi$ which implies $p(X) \mathrel{|\!\approx} \Phi$.
Sometimes, the facts we observe cannot be generalised that easily: Given $\Phi := \{p(0), p(s(s(0))), p(s(s(s(0)))), \ldots\}$ a simple application of lgg results in $\curlyvee \Phi = p(X)$. But actually, p only describes the set of *even* numbers,

$$\Phi = \left\{p(s^i(0)) : i \in \mathbb{N} \wedge i \mod 2 = 0\right\}.$$

[8] Note that ○ is an object we have not seen yet!

The simple lgg would deliver a hypothesis $p(X)$ that is wrong in infinite many cases. What we would like to have is richer term syntax so we could induce $p(s^{2n}(0))$.
Very close to this problem—but solvable if we were not restricted to single clauses with body literals from Π only—is the following: What *is* an even number, anyway? Wouldn't it be great to understand that $p(0)$ and $p(s(s(X))) \leftarrow p(X)$? Similarly, if we observe

$$\Phi = \{lt(0, X), lt(s(0), s(s(0))), lt(s(s(0)), s(s(s(0)))), lt(s(0), s(s(s(0))))\}$$

a mere generalisation would not work out well: It is *not* true, that every X is less than any Y.
Finally, and this is where we stopped at in the last section, consider the following case:

$$\begin{array}{ll} min(0, [0, s(s(s(0))), s(s(0))]). & min(0, [s(s(s(0))), 0, s(s(0))]). \\ min(s(0), [s(s(0))), s(0), s(0)]). & min(s(s(0)), [s(s(0))]). \end{array}$$

What does it take to understand that first,

$$min(X, [X])$$

and second

$$min(X, [Z|Y]) \leftarrow foo(X, Z), min(X, Y).$$

Where does *foo* come frome, and what is its meaning? From an analysis of the four examples above, we can deduce that foo can be defined as follows:

$$\begin{array}{lll} foo(0, s(s(0))), & foo(s(s(0)), s(s(s(0)))), & foo(0, s(s(s(0)))), \\ foo(s(0), s(0)), & foo(s(s(0)), s(s(0))) & \end{array}$$

Now comes what is *really* intelligent knowledge discovery: From the five ground terms above, we can induce that

$$foo(0, X) \quad \text{and} \quad foo(s(X), s(Y))\text{:-}\ foo(X, Y). \tag{7.46}$$

In the course of trying to learn *min* we came across a property describing a relation between two objects. This property, as we did not know how to name it, was called *foo*. In a next step we tried to understand what *foo* actually means and invented the beautiful littel recursive predicate definition above. The result is simply terrific: We invented a definition of the *less-than*-relation! This way we have discovered a new predicate and learned its definition in the same step.

We will now present three refinement operators one can apply in order to derive new, more general clauses from old ones which hopefully represent suitable hypotheses for the target concept.

Truncation

Recall that resolution itself is a proof calculus that is used for inference or deduction. Even though our proofs are kind of reverse arguments by showing a contradiction to the negation of the goal, resolution remains a *forward* calculus (hence we write "$\vdash$" rather than "$\dashv$"). It is clear that

$$\texttt{:-}\ \texttt{black\_polygon}(\blacksquare). \quad \text{and} \quad \texttt{black\_polygon}(X)\texttt{:-}\ \texttt{shape}(X, \texttt{square}).$$

resolves to

$$\texttt{:-}\ \texttt{shape}(\blacksquare, \texttt{square})$$

with $\sigma = [X \mapsto \blacksquare]$. But what if we knew there were two *positive ground literals* (we restrict ourselves to literals here) such that the facts they represent are true? Let there be two such ground literals λ and κ. Suppose that $\Pi \mathrel{|\!\!\approx} \{\lambda, \kappa\}$. Then we know that negating λ and κ allows for a derivation of the empty clause, or, semantically speaking, that

$$\Pi \cup \{\neg\kappa, \neg\lambda\} \mathrel{|\!\!\approx} \{\}$$

So if this is the case, there must have been clauses which resolved with $\neg\kappa$ and $\neg\lambda$—otherwise we were not able to deduce the empty clause.
Now let us try a little twist in the argument: Supposing that $E = \{\kappa, \lambda\}$ but $\Pi \mathrel{\not|\!\!\approx} E$, the task ahead is to find H such that $\Pi \cup H \mathrel{|\!\!\approx} E$. So wouldn't it be a great idea to give it a try and *guess* a rule that might allow for resolving $\neg\kappa$ (and, in a second step λ, too)? The head of the rule we guess is clear: it is a literal that unifies with κ. The definition of the rule body is not that clear because there may be many literal candidates around in Π. The simplest way is to take $\neg\lambda$ as a body literal—since we know that λ should follow from $\Pi \cup H$, too. As a resolution diagram, this looks as follows:

$$\begin{tikzcd}
\texttt{:-}\ \kappa \arrow[dr] & & (\kappa\texttt{:-}\ \lambda)\sigma^{\smile} \\
& \lambda \arrow[ur, "\sigma^{\smile}"'] &
\end{tikzcd} \tag{7.47}$$

Definition 7.20 — Truncation.

Truncation

Let there be two ground literals κ, λ. The *truncation operator* induces a rule $\kappa_0\texttt{:-}\ \lambda_0$ such that $\neg\kappa \sqcap^{\sigma}_{\mathsf{RES}} \kappa_0\texttt{:-}\ \lambda_0$ resolves to $\lambda = \lambda_0\sigma$. ●

Example 7.15 As a simple example, imagine two new predicate symbols `bigger` and `smallereq` with meaning as intended by their names. Their corresponding order relations are irreflexive (strict, $>$) and reflexive ($\leq$). Supposing that we encouter the following two ground facts

$$\begin{aligned}
\kappa &= \texttt{smallereq}(\Diamond, \blacklozenge) \\
\lambda &= \texttt{bigger}(\blacklozenge, \Diamond)
\end{aligned}$$

we construct the following resolution scheme:

$$\begin{tikzcd}
\texttt{:- smallereq}(\Diamond,\blacklozenge) \arrow[dr] & & (\texttt{smallereq}(\Diamond,\blacklozenge)\texttt{:- bigger}(\blacklozenge,\Diamond))^{\sigma^{\smile}} \\
& \texttt{:- bigger}(\blacklozenge,\Diamond) \arrow[ur, "\sigma^{\smile}"'] &
\end{tikzcd}$$

Next we need to find $\sigma^{\smile}$, such that in

$$\texttt{smallereq}(\Diamond,\blacklozenge)\texttt{:- bigger}(\blacklozenge,\Diamond)$$

terms are replaced with variables in a way such that after unifying the more general rule head with κ the resolution rule delivers λ. As on can see immediately, one possible inverse substitution is

$$\sigma^{\smile} = \left[\Diamond \mapsto X, \blacklozenge \rightarrow Y\right].$$

Then, the resulting rule becomes:

$$h = \texttt{smallereq}(X,Y)\texttt{:- bigger}(Y,X). \tag{7.48}$$

This example, even though very simple, already demonstrates one crucial problem: The semantics of the invented rule depends on the choice of κ and λ. If, in the example above, we had

$$\begin{aligned}
\kappa &= \texttt{bigger}(\blacklozenge,\Diamond)\\
\lambda &= \texttt{smallereq}(\Diamond,\blacklozenge)
\end{aligned}$$

the outcome would have been

$$h' = \texttt{bigger}(X,Y)\texttt{:- smallereq}(Y,X). \tag{7.49}$$

Rule h appears quite reasonable, but h' is not valid in $\mathfrak{A}$, because

$$\bullet \leq \bullet \not\approx \bullet > \bullet.$$

So if there are n candidate literals on to which we could apply the truncation operator, there are n^2 (or, to be more precise, $n \cdot (n-1)$) possible pairings. Furthermore, there is huge set of inverse substitutions we could apply:

Example 7.16 This time, let

$$\begin{aligned}
\kappa &= \texttt{smallereq}(\blacksquare,\triangle)\\
\lambda &= \texttt{bigger}(\blacklozenge,\Diamond)
\end{aligned}$$

Apart from the fact that there are two possible arrangements for κ and λ in the resolution scheme—how could we determine a *meaningful* hypothesis by way of $\sigma^{\smile}$? Even the least general generalisation via

$$\sigma^{\smile} = \left[\blacksquare \mapsto V, \triangle \mapsto X, \blacklozenge \mapsto Y, \Diamond \mapsto Z\right]$$

leads to

$$h'' = \texttt{bigger}(V, X)\texttt{:-}\ \texttt{smallereq}(Y, Z).$$

which is far too general: If there is any pair of objects Y and Z where Y is strictly bigger than Z, then everything (V) is smaller or equal than anything else (X). Again, this statement is *not* valid in $\mathfrak{A}$ for the same reason why h' is not valid (apply $\theta = [V \mapsto \bullet, X \mapsto \bullet, Y \mapsto \bullet, Z \mapsto \bullet]$). ●

Truncation comes with several problems: It requires two "suitable" ground literals, it requires a lucky hand at decising which one shall be taken as resolvent, it requires a "wise" choice of $\sigma^{\smile}$—and still it is very likely to bluntly overgeneralise. Still it is very useful as it relatively quickly can generate a huge set of hypotheses which can be tested quite efficiently as well: Given a set $H = \{h_i : i \in \mathbf{n}\}$ of hypotheses generated by truncation, we can determine the support $\mathit{support}(h_i, E^{\mathbf{1}})$ of each of its members. Most support values will be close to 1 because of the proneness of truncation towards overgeneralsiation. As a consequence, other error measures or measures of accuracy (see sections 3.4) should be considered. General hypotheses are not useless though: In section 7.3 we have learned that the FOIL-algorithm deliberately generates rules with empty bodies in order to specialise them in its inner refinement loop.

Truncation
Truncation is a very simple generalisation operator that takes two ground literals and delivers a guess for a rule that allows to derive one of these literals from the other. It seems not a very wise yet useful operator and it can be fine-tuned towards efficiacy and efficiency using heuristic measures.

Exercise 7.24 (♦ ♦♦ ♦♦♦) Let

$$\Phi = \left\{ \begin{array}{l} \texttt{has\_more\_edges}(\blacksquare, \bullet), \\ \texttt{shape}(\blacksquare, \texttt{square}) \\ \texttt{circle}(\bullet) \end{array} \right\}$$

♦ Try several combinations for truncation!—Try several inverse substitutions!

♦♦ For several hypotheses of differing generality, determine their support, error and accuracy with respect to $s = \left\{\lozenge, \triangle, \bullet, \blacksquare, \blacklozenge, \blacktriangle\right\}$!

♦♦♦ For several hypotheses, apply the inner loop of the FOIL-algorithm to specialsie them!

We close this section on truncation with a special case: Every successful resolution proof ends in an empty clause. If we agree to resolve exatly one literal per resolution step (as we do in SLD-resolution), we know that one parent clause must be a negative literal and the other one a positive literal. If there are *several* facts sharing the same predicate in $\Pi \cup E$, then there are *several* possibilities to derive the empty clause. Let there be two such unary clauses, not neccessarily unifiable, but sharing the same predicate symbol $\texttt{p}(\vec{t_a})$ and $\texttt{p}(\vec{t_b})$. Negating them and assuming both of them appear in the last step of a resolution proof, this requires the existence of a fact $\mathtt{p}(\vec{t})$ where $\vec{t_a}$ and $\vec{t}$ are unifiable and $\vec{t_b}$ and $\vec{t}$ are unifiable by most general unifiers μ and ν:[9]

$$\begin{tikzcd}
\{\mathtt{:-}\ \mathtt{p}(\vec{t_a})\} \arrow[dr, "\mu"] & & \{\mathtt{p}(\vec{t})\} \arrow[dl, "\mu"'] \arrow[dr, "\nu"] & & \{\mathtt{:-}\ \mathtt{p}(\vec{t_b})\} \arrow[dl, "\nu"'] \\
& \{\} & & \{\} &
\end{tikzcd}$$

Since μ is a most general unifier, we can construct a *generalisation* $\mathtt{p}(\vec{t_0})$ of $\mathtt{p}(\vec{t})$: $\mathtt{p}(\vec{t_0})\sigma = \mathtt{p}(\vec{t})$.[10] Since $t\mu = t_a\mu$ and $t_0\sigma = t$ it follows that $t_0\sigma\mu = t\mu$. Hence, $\mathtt{p}(\vec{t_0}) \mathrel{|\!\lhd} \mathtt{p}(\vec{t_a})$ (and the same for t_b with ν):

$$\begin{tikzcd}
& & \{\mathtt{p}(\vec{t_0})\} \arrow[d, "\sigma"'] \arrow[ddl, bend right, "\sigma\mu"'] \arrow[ddr, bend left, "\sigma\nu"] & & \\
\{\mathtt{:-}\ \mathtt{p}(\vec{t_a})\} \arrow[dr, "\mu"] & & \{\mathtt{p}(\vec{t})\} \arrow[dl, "\mu"'] \arrow[dr, "\nu"] & & \{\mathtt{:-}\ \mathtt{p}(\vec{t_b})\} \arrow[dl, "\nu"'] \\
& \{\} & & \{\} &
\end{tikzcd}$$

This special case simply states that given two (ground) facts, any generalisation of them can be considered a candidate for a hypothesis.

Intra-Construction

In order to understand the next refinement operator, *intra-construction*, we first need to discuss a special method for transforming logic programs which preserves their declarative semantics. Consider the following logic program Π that consists of two horn clauses:

$$\Pi \quad = \quad \left\{ \begin{array}{lllllllll} \mathtt{p}(\vec{X}) & \mathtt{:-} & \kappa_0, & \ldots, & \kappa_{k-1}, & \lambda_0, & \ldots, & \lambda_{l-1}. \\ \mathtt{p}(\vec{X}) & \mathtt{:-} & \kappa_0, & \ldots, & \kappa_{k-1}, & \nu_0, & \ldots, & \nu_{n-1}. \end{array} \right\}$$

We observe that both clauses contain the same k literals κ_i, $i \in \mathbf{k}$. Obviously, κ_i expresses a property that is significant to both rules defining $\mathtt{p}$. So why don't we move them outside and define a new "piece of meaning" that is shared by both clauses? We rewrite Π:

$$\Pi' \quad = \quad \left\{ \begin{array}{llllllll} \mathtt{p}(\vec{X}) & \mathtt{:-} & & \mathtt{q}(\vec{Y}), & & \lambda_0, & \ldots, & \lambda_{l-1}. \\ \mathtt{p}(\vec{X}) & \mathtt{:-} & & \mathtt{q}(\vec{Y}), & & \nu_0, & \ldots, & \nu_{n-1}. \\ \mathtt{q}(\vec{Y}) & \mathtt{:-} & \kappa_0, & \ldots, & \kappa_{k-1}. & & & \end{array} \right\}$$

[9] Note that this is just a proposition about the *existence* of $\mathtt{p}(\vec{t})$. It does not say anything about the actual value of $\vec{t}$; nor does it require $\vec{t}$ to to be different from $\vec{t_a}$ or $\vec{t_b}$.

[10] If $\vec{t}$ consists of variables only, then $\sigma = \emptyset$ and $\vec{t_0} = \vec{t}$.

Then, Π and Π' are equivalent. Similarly, we can move λ_i and ν_j outside so as to make the definition of p look more homogenous. Accordingly, Π can be also transformed into Π'':

$$\Pi'' = \left\{ \begin{array}{llllll} \mathtt{p}(\vec{X}) & \mathtt{:-} & \kappa_0, & \ldots, & \kappa_{k-1}, & \mathtt{r}(\vec{Y}) \quad . \\ \mathtt{p}(\vec{X}) & \mathtt{:-} & \kappa_0, & \ldots, & \kappa_{k-1}, & \mathtt{r}(\vec{Y}), \quad . \\ \mathtt{r}(\vec{Y}) & \mathtt{:-} & \lambda_0, & \ldots, & \lambda_{l-1}. & \\ \mathtt{r}(\vec{Y}) & \mathtt{:-} & \nu_0, & \ldots, & \nu_{n-1}. & \end{array} \right\}$$

Finally, we can apply both transforms to construct

$$\Pi''' = \left\{ \begin{array}{lllll} \mathtt{p}(\vec{X}) & \mathtt{:-} & \mathtt{q}(\vec{Y}), & \mathtt{r}(\vec{Z}). & \\ \mathtt{q}(\vec{Y}) & \mathtt{:-} & \kappa_0, & \ldots, & \kappa_{k-1}. \\ \mathtt{r}(\vec{Z}) & \mathtt{:-} & \lambda_0, & \ldots, & \lambda_{l-1}. \\ \mathtt{r}(\vec{Z}) & \mathtt{:-} & \nu_0, & \ldots, & \nu_{n-1}. \end{array} \right\}$$

Exercise 7.25 (◊) Explain: Is $\vec{Y} = \vec{X}$ in Π'? And what is in $\vec{Z}$ in Π'''?

Exercise 7.26 (⧫) Show that $\Pi \mid\!\approx \varphi \iff \Pi' \mid\!\approx \varphi \iff \Pi'' \mid\!\approx \varphi \iff \Pi''' \mid\!\approx \varphi$

By adding a generalisation step to this *folding procedure*[11] we define the intra-construction operator that is very suitable for inducing entirely new predicates along with their definition:

Definition 7.21 — Intra-Construction.

Intra-Construction

Given two clauses with a common set of unifiable literals, *intra-construction* derives a new clause with a new body literal defined by common set of literals in two separate clauses:

$$\frac{\mathtt{p}(\vec{X})\mathtt{:-}\ \kappa_k, \lambda_l. \qquad \mathtt{p}(\vec{Y})\mathtt{:-}\ \kappa_k, \nu_n.}{(\mathtt{q}(\vec{Y})\mathtt{:-}\ \lambda_l.)\sigma^{\smile}_{\lambda} \qquad (\mathtt{p}(\vec{X})\mathtt{:-}\ \kappa_k, \mathtt{q}(\vec{Y}).)\sigma^{\smile}_{\kappa} \qquad (\mathtt{q}(\vec{Y})\mathtt{:-}\ \nu_n.)\sigma^{\smile}_{\nu}}$$

with $k \in \mathbf{k}, l \in \mathbf{l}, n \in \mathbf{n}$ such that

$$(\mathtt{p}(\vec{X})\mathtt{:-}\ \kappa_k, \lambda_l.)\sigma^{\smile}_{\kappa} \cup \sigma^{\smile}_{\lambda} \quad \mid\!\lhd \quad \mathtt{p}(\vec{X})\mathtt{:-}\ \kappa_k, \lambda_l. \tag{7.50}$$

$$(\mathtt{p}(\vec{X})\mathtt{:-}\ \kappa_k, \nu_n.)\sigma^{\smile}_{\kappa} \cup \sigma^{\smile}_{\nu} \quad \mid\!\lhd \quad \mathtt{p}(\vec{X})\mathtt{:-}\ \kappa_k, \nu_n. \tag{7.51}$$

Note that the body literals of the newly defined rules also subsume their respective instantiations in the parent clauses. ●

Intra-construction has generalsiation steps built in two places: First, the remaining rule body κ_k is generalised to $\kappa_k\sigma^{\smile}_{\kappa}$ and second, the respective rule bodies are generalisations of the original body literals by application of (suitable) $\sigma^{\smile}_{\lambda}$ and $\sigma^{\smile}_{\nu}$. A third generalisation step that can be added with just a little more effort is motivated by the definition of θ-subsumption: As we already have discovered,

[11] The reverse procedure is called *fanning*. Both are quite common in logic programming to speed up and (re-) structure programs. In contrast to our considerations the *procedural* semantics of Prolog then plays a major role.

literal dropping is one method of generalising Horn clauses. Intra-construction now allows for a several options of where to drop literals. Dropping a literal $\kappa_i\sigma^{\smile}_{\kappa}$ from $\mathtt{p}(\vec{X})\mathtt{:-}\ \kappa_k, \mathtt{q}(\vec{Y})$ is more general than dropping κ_i from both original rule bodies—and it is, of course even more general than dropping some κ_i in only one of the clauses. Dropping a literal $\lambda_i\sigma^{\smile}_{\lambda}$ from $(\mathtt{q}(\vec{Y})\mathtt{:-}\ \lambda_l.)\sigma^{\smile}_{\lambda}$ makes no big difference to dropping it from $\mathtt{p}(\vec{X})\mathtt{:-}\ \kappa_k, \lambda_l$ (except for the fact that, of course, $\lambda_i\sigma^{\smile}_{\lambda} \,|\!\!\lhd\, \lambda_i$). By dropping literals from the definitions of q we can generalise pretty selectively. Finally, we can try and apply more inverse substitutions and simultaneously drop literals λ_i and ν_j until the rule bodies are equal:

$$\frac{\mathtt{p}(\vec{X})\mathtt{:-}\ \kappa_0, \ldots, \kappa_{k-1}, \lambda_0 \ldots, \lambda_{l-1}. \qquad \mathtt{p}(\vec{X})\mathtt{:-}\ \kappa_0, \ldots, \kappa_{k-1}, \nu_0 \ldots, \nu_{n-1}.}{(\mathtt{p}(\vec{X})\mathtt{:-}\ \mathtt{q}(\vec{Y}), \mathtt{r}(\vec{Z}).)\sigma^{\smile}_{\kappa} \qquad (\mathtt{q}(\vec{Y})\mathtt{:-}\ Q.)\sigma^{\smile}_{\lambda} \qquad (\mathtt{r}(\vec{Y})\mathtt{:-}\ R.)\sigma^{\smile}_{\nu}} \tag{7.52}$$

where $Q \subseteq \left\{\kappa_i\sigma^{\smile}_i : i \in \mathbf{k}\right\}$ and $R \subseteq \{\chi\ :\ \exists i, j, \sigma : \chi\sigma = \lambda_i = \nu_j\}$.

Example 7.17 Consider the program Π that consists of the following two clauses:

$$\mathtt{black\_polygon}(\blacksquare) \quad \mathtt{:-} \quad \mathtt{black}(\blacksquare), \mathtt{square}(\blacksquare). \tag{7.53}$$

$$\mathtt{black\_polygon}(\blacktriangle) \quad \mathtt{:-} \quad \mathtt{black}(\blacktriangle), \mathtt{has\_more\_edges}(\blacktriangle, \bullet). \tag{7.54}$$

First of all, the generalised clause head is

$$\mathtt{black\_polygon}(X) = \bigtriangledown \{\mathtt{black\_polygon}(\blacksquare), \mathtt{black\_polygon}(\blacktriangle)\} \tag{7.55}$$

with $\sigma^{\smile} = [\blacksquare \mapsto X, \blacktriangle \mapsto X]$. Both clauses share a common body literal black. We introduce a new predicate symbol p of the same arity as the number of common arguments and receive

$$\left\{\begin{array}{lll} \mathtt{p}(\blacksquare) & \mathtt{:-} & \mathtt{black}(\blacksquare)., \\ \mathtt{p}(\blacktriangle) & \mathtt{:-} & \mathtt{black}(\blacktriangle). \end{array}\right\}\sigma^{\smile} \quad = \quad \{\mathtt{p}(X)\mathtt{:-}\ \mathtt{black}(X).\} \tag{7.56}$$

The next step is to move the two non-unifying literals outside. We define

$$\begin{aligned} & \left\{\begin{array}{l} \mathtt{q}(\blacksquare)\mathtt{:-}\ \mathtt{square}(\blacksquare)., \\ \mathtt{q}(\blacktriangle)\mathtt{:-}\ \mathtt{has\_more\_edges}(\blacktriangle, \bullet). \end{array}\right\}\sigma^{\smile} \\ = & \left\{\begin{array}{l} \mathtt{q}(X)\mathtt{:-}\ \mathtt{square}(X)., \\ \mathtt{q}(X)\mathtt{:-}\ \mathtt{has\_more\_edges}(X, \bullet). \end{array}\right\} \end{aligned} \tag{7.57}$$

All in all, the resulting logic program H looks as follows:

$$\begin{array}{lll} \mathtt{black\_polygon}(X) & \mathtt{:-} & \mathtt{p}(X), \mathtt{q}(X). \\ \mathtt{p}(X) & \mathtt{:-} & \mathtt{black}(X). \\ \mathtt{q}(X) & \mathtt{:-} & \mathtt{square}(X). \\ \mathtt{q}(X) & \mathtt{:-} & \mathtt{has\_more\_edges}(X, \bullet). \end{array} \tag{7.58}$$

At the very end we now have a set of Horn clauses that actually look like a proper semantic definition of what it means for an object to be a black polygon.

Exercise 7.27 (◊⧫⧫⧫)

- ⧫ Determine support, accuracy and coverage of the body literals of the new rules defining $\mathtt{q}(X)$!
- ◊ Prove that $\mathtt{q}(X)\mathtt{:-}\ \mathtt{square}(X) \mathrel{|\!\approx} \mathtt{q}(X)\mathtt{:-}\ \mathtt{has\_more\_edges}(X, \bullet)$.
- ⧫ Can you find a program $H' \subset H$ for which $H' \mathrel{|\!\approx} \varphi \Longleftrightarrow H \mathrel{|\!\approx} \varphi$?
- ⧫ Define a procedure which mechanises the process to find such subsets!

You might have realised that you were betrayed at a very small but crucially important point in the argumentation of intra-construction. Recall equation 7.57: Given the new predicate symbol q, we silently agreed that it shall be unary—even though in the second definition of black_polygon, the second body literal is a *binary* predicate has_more_edges. A little bit of abstraction exemplifies the problem: Let there be two clauses again,

$$\begin{array}{lllllllllll} \varphi = & (& \mathtt{p}(\vec{X}) & \mathtt{:-} & \kappa_0, & \ldots, & \kappa_{k-1}, & \lambda_0, & \ldots, & \lambda_{l-1}. &)\sigma_\varphi \\ \psi = & (& \mathtt{p}(\vec{X}) & \mathtt{:-} & \kappa_0, & \ldots, & \kappa_{k-1}, & \nu_0, & \ldots, & \nu_{n-1}. &)\sigma_\psi \end{array} \tag{7.59}$$

Note that σ_φ acts on the *entire* clause φ just as σ_ψ acts on entire ψ. The consequence is that $\vec{X}\sigma_\varphi = \vec{t}_\varphi \neq \vec{t}_\psi = \vec{X}\sigma_\psi$. This does not hurt at all because we want to construct a new rule head that is a generalsiation of φ and ψ anyway. Similarly, we can find generalsiations of $\kappa_i\sigma_\lambda$ and $\kappa_i\sigma_\nu$. Now there may happen something very interesting:

Example 7.18 Imagine $\chi = \mathtt{p}(A, B, C)\mathtt{:-}\ \mathtt{q}(X, C), \mathtt{r}(A, C)$ which θ-subsumes both of the following two clauses:

$$\begin{array}{llllllll} \varphi = & (& \mathtt{p}(A,B,C) & \mathtt{:-} & \mathtt{q}(X,C), & \mathtt{r}(A,C), & \mathtt{u}(c), \quad \mathtt{v}(c,B). &)\sigma_\varphi \\ = & & \mathtt{p}(a,f(B),C) & \mathtt{:-} & \mathtt{q}(c,C), & \mathtt{r}(a,C), & \mathtt{u}(c), \quad \mathtt{v}(c,f(b)). & \\ \psi = & (& \mathtt{p}(A,B,C) & \mathtt{:-} & \mathtt{q}(B,C), & \mathtt{r}(A,C), & \mathtt{a}(V,C,B). &)\sigma_\psi \\ = & & \mathtt{p}(A,f(b),C) & \mathtt{:-} & \mathtt{q}(f(b),C), & \mathtt{r}(A,C), & \mathtt{a}(a,C,f(b)). & \end{array} \tag{7.60}$$

with $\sigma_\varphi = [A \mapsto a, B \mapsto f(b), X \mapsto c]$ and $\sigma_\psi = [B \mapsto f(b), V \mapsto a]$. For φ, we observe that the set of common variables in the clause head and in the q- and r-literals is a prober subset of the clause head of χ—whereas all variables occuring in χ also occur in ψ. ●

So if in the example above we were to generalsie from the rule heads and the matching κ-literals in φ there would be no evidence that the second argment of the rule head has any meaning at all. Fortunately, the seconde argument of the rule head is *linked to* to the first argument of the q–literal in ψ, which, by inverse construction of χ via $\varphi \mathbin{\nabla} \psi$ forces a binding between those two places. The second—much more important—problem occurs when we try to fold out the remaining body literals: We want to introduce a new predicate definition with a new predicate symbol s. Then, obviously,

$$\mathtt{s}(\vec{X}) \quad \mathtt{:-} \quad \mathtt{u}(c), \mathtt{v}(c, f(b)).\sigma_\varphi^{\smile} \tag{7.61}$$

$$\mathtt{s}(\vec{X}) \quad \mathtt{:-} \quad \mathtt{a}(a, C, f(b)).\sigma_\psi^{\smile} \tag{7.62}$$

The first one allows for several inverse substitutions:

- $\sigma^{\smile}_{\varphi} = [c \mapsto X]$ suggests a generalsiation to $\texttt{s}(B)\texttt{:-}\ \texttt{u}(B), \texttt{V}(B)$.

- a slightly more general approach already results in two alternative rule heads: Assuming that C is *free* in φ, we could simply drop it from the clause head and write $\texttt{s}(B)\texttt{:-}\ \texttt{a}(X), \texttt{b}(X, B)$.[12] On the other hand, the then common appearance of X in $\texttt{u}(X), \texttt{v}(X, f(b))$ and $\texttt{p}(A, B, C)\texttt{:-}\ \texttt{q}(X, C)$ suggests that the semantics of $\texttt{s}$ depends on $\texttt{q}$—and the semantics of $\texttt{p}$ depends on the value of X. Therefore, it would safer to induce $\texttt{s}(B, X)\texttt{:-}\ \texttt{u}(X), \texttt{v}(X, B)$.

- $\sigma^{\smile}_{\varphi} = [\,]$ leaves the rule body untouched—yet we have two options to construct the clause head: Both $\texttt{s}(c, B)\texttt{:-}\ \texttt{u}(c), \texttt{v}(c, B)$ and $\texttt{s}(B)\texttt{:-}\ \texttt{u}(c), \texttt{v}(c, B)$ seem reasonable generalsiations.

Similarly, ψ offers a whole lot of optinos for generalsiations. We do not go too much into detail, because the idea should be clear by now. Instead, we now point out *how insanely many options* there are to construct a new predicate and its definition:

- First, the empty inverse substitution leads to the same arity dilemaa as the empty inverse substitution on φ. We could infer three different unary clause heads (if we assume the ordering to be irrelevant), three different binary clause heads and one tertiary clause head which sums up to seven alternatives for the empty inverse substitution only.

- For all of these options mentioned before, we can now find a whole set of inverse substitutions acting on all the subsets of terms ocuring in any place in $\texttt{a}(a, C, f(b))$. The subterms are $\{a, C, b, f(b)\}$ such that we can conctruct sixteen (2^4) different substitution schemata; and for each we can chose from any combination of variables!

Exercise 7.28 (♦♦) From a purely semantic point of view, the safest thing would be to carry all the variables with us all the time. This means, that any rule head $\texttt{p}(\vec{X})$ can be expanded to $\texttt{p}(\vec{X}\vec{V})$ where $\vec{V}$ contains *all* variables occuring in all the literals of the clause.—Redefine the intra-construction operator such that all variables are kept at all times! Then, define a (heuristic) procedure to determine minimal sets of "relvant" variables in order to cut down the predicate arities after the intra-construction procedure.

It should have become clear by now that refinement via intra-construction is well defind just as truncation is. But it is also as computationally expensive as truncation (actually, a lot more). If we restrict ourselves to clauses with *exactly one* body literal, we can at least give a simple description of an intra-construction procedure. Figure 7.4 presents such a pseudo–code program. Even though the restriction to single body literals is so strict that the most important parts of the expressiveness of HOL is lost, it still illustrates our desperate need for good heurisitc measures to guide the search for promising hypotheses.

[12] Or, even more general, just $\texttt{s}(B)\texttt{:-}\ \texttt{u}(X), \texttt{v}(Y, B)$.

```
01  $\sqcap = \left\{ \mathtt{p}(\vec{X}_i)\,\mathtt{:-}\ \lambda_i.\ : i \in \mathbf{n} \right\}$
02  $\lambda := \lambda_0;$
03  FOR $i := 1$ TO $n$
04      $\lambda := \lambda \triangledown \lambda_i$
05  NEXT
06  $\kappa := \mathtt{q}(V_0, \ldots, V_{k-1})$ where: $\{V_j : j \in \mathbf{k}\} \subseteq \ulcorner\!(\bigcup_{i \in \mathbf{n}} \{\sigma_i : \lambda\sigma_i = \lambda_i\})$
07  RETURN $\left\{ \mathtt{p}(\vec{X}_i)\,\mathtt{:-}\ \kappa. \right\} \cup \{\kappa\sigma_i : i \in \mathbf{m}\}$ for some $\mathbf{m} \subseteq \mathbf{n}$
```

Note three nondeterministic operations here: In line 6, a "suitable" subset of variables has to be chosen. In the same line, the σ_i are not uniquely defined; all that is required is that $\lambda \mathrel{|\!\!\lhd} \lambda_i$. Finally, the new body literal κ is defined by m new unary clauses that are chosen from n different variable assignments.

Figure 7.4: Intra-construction for binary clauses

Intra–construction
Intra-construction is a refinement operator that allows to invent new predicates. The principle is based on two ideas: First, fold out common literals of several instances of a rule and then generalise the resulting rule; second, invent new predicates for all disjoint sets of body literals and generalise them.
The big problem with inventing new predicates is: Since we do not know what they mean, we don't know on which variables they really depend. We also don't know which generalsiation from a huge number of options we should apply. And sadly we don't even know how to call it and give it a proper name for the new predicate...

Exercise 7.29 (♦) Give a rough estimate of the number of different programs that can be induced from $\left\{ \mathtt{p}(\vec{X}_i)\,\mathtt{:-}\ \lambda_i.\ : i \in \mathbf{n} \right\}$ by the algorithm shown in figure 7.4

Absorption

There is one very important type of clauses that we already spoke of but which we are not able to induce yet. These are clauses with multiple occurences of literals—especialy those, where the rule head predicate symbol also appears in the rule body. It is the class of *recursive* predicates like in equation (7.46).
Recursion is a simple concept—but trying to induce recursive predicates is not that easy. Just recall the problems we have had with finding suitable inverse substitutions. On the one hand we need not care about the set of body literals to choose and, therefore, the solution to the problem which arguments are connected to each other comes for free. The real problem we are concerned with is recursion on *recursive types* of terms. But logic programs have no such thing as a *type*. It is impossible for us to infer the structure of a generalisation of a term from its "sort": If $t_0 = 3$ and $t_1 = 2.0$, what is $t_0 \triangledown t_1$—is it some $X \in \mathbb{N}$ or $X \in \mathbb{R}$? If $t_0 = [\,]$ and $t_1 = [a, b]$—is $t_0 \triangledown t_1$ it just a free variable X that can

be instnatiated with whatever we like? Or is it a *list*? Or a list with at most two elements? And if $t_1 = [X]$, do we want to allow some $t \not\lhd \{t_0, t_1\}$ where t also can take a recursive list type like $[[\,], [[\,]], [[\,], [[\,]]]]$?
To make things easier at the beginning, we shall consider a Horn subset of PL first. Consider the following two propositional Horn clauses:

$$\mathtt{A{:-}\,\vec{B}}. \quad \text{and} \quad \mathtt{C{:-}\,A,\vec{D}}. \tag{7.63}$$

There is not much we can do in SLD-resolution, but note that in the left clause `A` is the rule head (hence a positive literal) and in the right clause it is a negative literal. So in full resolution we could infer

$$\mathtt{C{:-}\,\vec{B},\vec{D}}. \tag{7.64}$$

Let us now reverse this resultion step: Imagine $\mathtt{A{:-}\,\vec{B}}$ and $\mathtt{C{:-}\,\vec{B},\vec{D}}$ were known. Then,

$$\begin{tikzcd}
\mathtt{A{:-}\,\vec{B}} \arrow[dr] & & \mathtt{C{:-}\,A,\vec{D}} \\
& \mathtt{C{:-}\,\vec{B},\vec{D}} \arrow[ur] &
\end{tikzcd}$$

where the direction of the arrows indicates our inductive process. There is not much of recursion in here—but before we come to it, let us lift our idea of *absorption* to FOL. Since variables are enough of a hassle and since we agreed that E consists of single literals only, simplify our inverse resolution scheme to a version with a unary parent clause only:

$$\begin{tikzcd}
\mathtt{p}(\vec{X}_0). \arrow[dr, "\sigma"] & & \mathtt{q}(\vec{Y}_1)\mathtt{{:-}}\ \lambda_0, \ldots, \lambda_{k-1}, \mathtt{p}(\vec{X}_1), \lambda_{k+1}, \ldots, \lambda_{n-1}. \\
& (\mathtt{q}(\vec{Y}_0)\mathtt{{:-}}\ (\lambda_0, \ldots, \lambda_{n-1}))\theta\sigma \arrow[ur, "\theta^{\smile}"] &
\end{tikzcd} \tag{7.65}$$

where σ unifies $\mathtt{p}(\vec{X}_0)$ and $\mathtt{p}(\vec{X}_1)$ and θ is a specialsiation of λ_i.

Absorption

Definition 7.22 — Absorption.
The absorption operator takes a unary clause and a definite clause and returns a generalisatin of the clause that is obtained by adding the literal of the unary clause to the rule body:

$$\frac{\{\kappa\}\,\sigma \qquad (\{\nu\} \cup \{\neg\lambda_i : i \in \mathbf{n}\})\theta}{\{\nu\} \cup \{\neg\lambda_i : i \in \mathbf{n}\} \cup \{\neg\kappa\}} \tag{7.66}$$

We rewrite and change the instantiating assignments σ and θ into inverse substitutions:

$$\frac{\kappa. \qquad \nu\mathtt{{:-}}\ \lambda_0, \ldots, \lambda_{n-1}.}{(\nu\mathtt{{:-}}\ \lambda_0, \ldots, \lambda_{k-1}, (\kappa\sigma^{\smile}), \lambda_{k+1}, \ldots, \lambda_{n-1}.)\theta^{\smile}} \tag{7.67}$$

●

There remain two open questions: Where is recursion and how do we define the inverse substitutions? The first question is easy to answer. Absorption can be used to learn recursive predicates by chosing κ and $\nu \mathtt{:-}\ \lambda_0, \ldots, \lambda_{n-1}$ such that both κ and ν are literals sharing a common predicate symbol! From the resolution scheme and the absorption rule, we then know that

$$\kappa \mid\!\lhd \nu \text{ because } \nu = \kappa\sigma\theta \tag{7.68}$$

The question about how to compute suitable inverse substitutions cannot be answered that easily. Therefore, we give an example illustrating the induction process of a recursive predicate with manual, “wise” choices of $\sigma^{\smile}$ and $\theta^{\smile}$.

Example 7.19 This time we cannot give a decent example in our domain of geometric objects as it is difficult to find a proper instance for a recursive predicate here. Therefore, please recall the example from the beginning of this section where we invented the *less-than*-relation in the course of learning the *minimum*-predicate. ‘Imagine now that

$$\begin{aligned} \kappa &= \mathtt{lt}(X, s(X)) \\ \nu &= \mathtt{lt}(Y, s(s(Y))) \end{aligned}$$

Furthermore, let us assume that $n = 0$, i.e. there are no further λ_i-body literals involved. Then, the recursive rule we are about to invent has the form

$$(\nu \mathtt{:-}\ (\kappa\sigma^{\smile}))\theta^{\smile}$$

This becomes

$$\begin{tikzcd} \mathtt{lt}(X, s(X)) \arrow[dr] & & (\mathtt{lt}(Y, s(s(Y))) \mathtt{:-}\ (\mathtt{lt}(X, s(X)))\sigma^{\smile})\theta^{\smile} \\ & \mathtt{lt}(Y, s(s(Y))). \arrow[ur] & \end{tikzcd}$$

By miraculuous inspiration we choose $\sigma^{\smile} = [s(X) \mapsto Z]$. Hence,

$$\begin{tikzcd} \mathtt{lt}(X, s(X)) \arrow[dr] & & (\mathtt{lt}(Y, s(s(Y))) \mathtt{:-}\ \mathtt{lt}(X, Z))\theta^{\smile} \\ & \mathtt{lt}(Y, s(s(Y))). \arrow[ur] & \end{tikzcd}$$

Then, a second miracle happens and we feel like chosing $\theta^{\smile} = [s(Y) \mapsto Z, Y \mapsto X]$:

$$\begin{tikzcd} \mathtt{lt}(X, s(X)) \arrow[dr] & & \mathtt{lt}(X, s(Z)) \mathtt{:-}\ \mathtt{lt}(X, Z) \\ & \mathtt{lt}(Y, s(s(Y))). \arrow[ur] & \end{tikzcd}$$

The result is a new, logically correct arithmetic rule: Whenever X is less than Z, then X is less than $Z + 1$, too.

```
LET $\varphi := \{\kappa\}$, $\psi := \nu \cup \{\neg\lambda_i : i \in \mathbf{n}\}$
$T_p := \{\langle t, p\rangle : t$ is a term at position $p$ in $\varphi \cup \psi\}$
Choose $T'_p \subseteq T_p$
Find an equivalence relation $\equiv$ on $T'_p \times T'_p$ where:
   $R \equiv S$ if and only if
   a) $s \mid\!\lhd r$ for all $\langle s, p\rangle \in S$ and $\langle r, p\rangle \in R$
   b) for all $\langle s, p\rangle \in S$, $s$ occurs in $\varphi$
   c) for all $\langle r, q\rangle \in R$, $r$ occurs in $\psi$
Compute $\sigma$ such that $S\sigma = R$ for all $R \equiv S$
$\theta^{\smile}_i = \left[\langle r, \langle p_1, \ldots, p_n, q_1, \ldots, q_m\rangle\rangle \mapsto V :\ \text{for all}\ \ r \in T'_p/\equiv\right]$, where:
   all $V$ are different variables which do not occur within $\varphi \cup \psi$
RETURN $(\nu \cup \{\neg\lambda_i : i \in \mathbf{n}\} \cup \kappa\sigma)\theta^{\smile}$
```

Figure 7.5: An informal description of the absorption procedure

Exercise 7.30 ♦♦♦ Let there be $\kappa = \mathtt{lt}(0, X)$ and $\nu = \mathtt{lt}(0, s(Y))$. Find $\sigma^{\smile}$ and $\theta^{\smile}$ such that you can induce $\mathtt{lt}(s(X), s(Y))\mathtt{:-}\ \mathtt{lt}(X, Y)$.—Why is this a wonderful result?

To gain a short impression of the algorithmic formulation of absorption and all the problems connected to it, take a look at figure 7.5. For our purpose it is not important to really understand the algorithm outlined in figure 7.5. The interested reader can find a more detailed elaboration in [Muggleton and Buntine, 1988]. But what *is* interesting, is the sheer monstrous complexity of this algorithm. It is clear that absorptaion can only be described by this algorithm—but not implemented.

Exercise 7.31 (◊♦ ⋯ ♦) Take a closer look at the algorithm in figure 7.5. Give an estimate for the size of T_p (♦♦). Determine the number of possible T'_p (◊). How many relations $\equiv$ are there (♦)? For a given equivalence relation $\equiv$, how long does it take to examine whether it satisfies conditions a)–c) (♦♦)?

Absorption
The absorption comes with bad news and good news: It is very powerful when it comes down to learning recursive predicates. The bad news is that it is virtually infeasible and very hard to implement in a way that allows an efficient application. The good news is: As far as relational knowledge representation is concerned there is no need for learning recursive predicates at all.

In the previous sections we discussed a method for inducing logic programs that was based on two concepts of generality. The first one is that of θ-subsumption. Given a formula φ, one can simply try and create some $\sigma^{\smile}$ and return $\varphi\sigma^{\smile}$ as a hypothesis. A slightly more intelligent method included literal dropping. Including another set of formulas Φ into this process, we discovered ∇_Φ as a generalsiation procedure relative to a given set of knowledge. The second concept that we modified to create a new method for inducing new Horn formulas was resolution. We discovered three different special cases for which we described one operator each.

But after all, unification and Deduction are derivation processes. Our hope is that reversal of derivation leads to something like induction. In both cases, there is no semantics involved. Yet the presented "induction calculi" are, procedurally, infeasible. So in order to implement a working system, we need strong biases. There are several possible solutions: First, we can restrict the language—for example by allowing only a certain maximum number of variables or literals in each clause. We can also try and introduce types or sorts into our language. But most importantly, a good search procedure is required—based on a suitable guiding heuristics.

In the next chapter we will discover a few more; and we will rediscover several ideas to greater detail.

7.4.3 Semantic Biases

Inverting resolution means to define a calculus with a rule set that implements a new derivation relation $\dashv$. We were looking for some H such that for a given Π we can explain E; which means that $\Pi \cup H \mathrel{|\!\!\approx} E$. Since semantics was too expensive, we reduced $\mathrel{|\!\!\approx}$ to $\vdash$ and developed refinement operators like ∇_{Φ}, intra-construction etc., such that $\mathit{refine}(\Pi \cup H) \vdash \varphi \Longrightarrow \mathit{refine}(\Pi \cup H) \mathrel{|\!\!\approx} \varphi$. In this section, we will discover and a few more techniques for defining biases and discuss known ones too more detail.

Restricting model size

The great things about *ground* models is, that there are no variables any more. Recall the notion of a *Herbrand interpretation* (footnote 7). With at least one non-constant function symbol in Σ, the Herbrand base is (due to recursion) infinite. This is, theoreticaly, a very bad starting basis for an efficcient search for H. However, by the *Compactness Theorem* and the work of Löwenheim and Skolem, we know that it suffices to find an unsatisfiable subset to show that $\Pi \cup H$ is unsatsfiable. The procedure of a refutation proof by resuloution makes use of exactly this property: It is based on the efficient construction of a unsatisfiable subset (by adding the negation of the goal) and, by the law of the excluded middle, conclude that if the negation leads to failure, the non-negated goal must have been true.

Exercise 7.32 ⧫ *Look up the Compactness Theorem and the Löwenheim-Skolem-Theorem.*

We now define a method for constructing *finite* Models which are admittedly not complete - since defined by finite application of the resolution rule:

h–easy ground Models

Definition 7.23 — h–easy ground Models.
Let Φ be a set of of HOL-formulas over a signature Σ. We define

$$\mathit{Th}^{h}_{\vdash}(\Phi) \quad := \quad \left\{\varphi \in \mathrm{Fml}_{\mathrm{HOL}} : \Phi \vdash^{i} \varphi \wedge i \leq h\right\} \qquad (7.69)$$

where $\vdash$ denotes a SLD-resolution step.[13] ●

So instead of searching a potentially infinite hypothesis space, we restrict ourselves to a finite one. Additionally, this restriction is defined in terms of the minimum length of derivations required to infer a formula.

Restricting the set of possible assignments

In the last section we already mentioned the concept of *connectedness*. We discovered that a variable occuing in a clause body has—due to the nature of SLD-reolution—a certain impact on the semantics of a predicate. More specifically, a variable occuring in a clause is predetermined in the number of possible bindings by the number of variables, their assignments, and their common appearance. This requires the notion of *variable depth* in a clause: Let there be a clause

$$\lambda_0 \texttt{:-}\ \lambda_1, \ldots, \lambda_{n-1}.$$

Then, all the variables occuring in the clause head λ_0 have depth 0. If some variable X occurs in L_i for the first time, then its depth is the maximum depth of all varibales occuring in the literals L_j plus one (where $i \in \mathbf{n}$ and $j \in \mathbf{i}$):

$$\text{depth}(X) \quad := \quad \max\{\text{depth}(X) : X \text{ occurs in } \lambda_j, j \in \mathbf{i}\} + 1 \qquad (7.70)$$

We now formulate a very important property of variables: It is their *degree (of freedom)*:

Example 7.20 On our base set $s = \{\Diamond, \triangle, \bullet, \blacksquare, \blacklozenge, \blacktriangle\}$, we examine the number of (type compatible) variable assignements for the predicates `bigger`, `colour`, `shape`, and `has_more_edges` (abbreviated by `hme`):

Predicate	# possible α	# satisfying σ
`bigger`	$6 \cdot 6 = 36$	$2 \cdot 4 = 8$
`colour`	$6 \cdot 2 = 12$	$4 + 2 = 6$
`shape`	$6 \cdot 3 = 18$	$1 + 1 + 2 + 2 = 6$
`hme`	$6 \cdot 6 = 36$	$5 + 4 + 3 = 12$

Imagine now we would like to model a predicate that describes whether printing one object *needs_more_ink* than another. Then,

`nmi`$(\blacklozenge, X)$ is true for all X, because $\blacklozenge$ requires most ink of all objects in s.
`nmi`$(\triangle, X)$ requires that $X = \Diamond$,
because any other object needs more ink than $\oslash$.
`nmi`$(X, \bullet)$ requires $X \in \{\blacksquare, \blacklozenge, \blacktriangle\}$

[13] This definition is a simplification of the more abstract idea of h–easiness: h is recursive, computable function, such that for each φ_i which is valid in our model we need at most $h(i)$ derivation steps to deduce φ_i from Φ. Here, define h to be a constant value.

Obviously, all black objects require more ink than white objects, and for most black objects, it holds that the more edges it has, the more ink it requires (the exception is that ♦ needs more ink even though it has not more edges than ■).
●

We define:

Free-Assignment-Degree of a variable

Definition 7.24 — Free-Assignment-Degree of a variable.
Let there be a clause with n literals,

$$\lambda_0 \quad \texttt{:-} \quad \lambda_1, \ldots, \lambda_{n-1}.$$

and the i–th literal

$$\lambda_i \quad = \quad \texttt{p}_i(X_0, \ldots, X_k)$$

The *free-assignment-degree* of X_j in λ_i (written $\mathrm{degfree}(X_j, \lambda_i)$) is the number of variables in $\{X_j, \ldots, X_k\}$ which also appear in $\{X_0, \ldots, X_{j-1}\}$ or in any other literal λ_k, $k \in \mathbf{i}$. ●

As a rule of thumb, $\mathrm{degfree}(X_j, \lambda_i)$ increase with growing i and j—which is due to the top-down and left-to-right resolving strategy of SLD-Resolution.

Example 7.21 Let

$$\texttt{nmi}_0(X_0, Y_1)\texttt{:-}\ \texttt{colour}_1(X_0, C_1), \texttt{colour}_2(Y_0, C_1), \texttt{hme}_3(X_0, Y_1).$$

where the idendixing is just to identify the occurence of a variable in a different literal. It states that for two objects of the same colour, the one with more edges requires more ink. The depths of all the variables are:

	X_0	Y_0	X_1	C_1	Y_2	C_2	X_3	Y_3
$\mathrm{depth}(X) =$	0	0	0	1	0	1	0	0

The values of $\mathrm{degfree}(X_j, \lambda_i)$ are as follows:

	λ_i			
$\mathrm{degfree}(\cdot)$	0	1	2	3
$j = 0$	0	1	1	2
$j = 1$	0	0	1	1

Supposing that $\alpha(X_0) = \blacktriangle$, there are six possible substitutions for Y_0 of which only three are valid ($\alpha(Y_0) \in \{\lozenge, \triangle, \bullet\}$). The value of C is not restricted yet; So for all three variables X, Y, C in the clause we have $1 \cdot 6 \cdot 2 = 12$ different assignments to chose from. Since $X_1 = X_0$, there is only one possible value for C_1; namely `black`—ruling out the factor 2 from the number of possible instantiations such that we are left with only six alternatives. In literal 2, the fact that $C_2 = C_1 =$ `black`, reduces the number of possible instantiations for Y_2 to only 4 because there are only four black objects in s. In literal 3, all our literals are instantiated. We conclude that there can be at most four different values for Y (rather than six). ●

A literal λ is called *determinate*, if each new variable in it has exactly **one possible** binding given the bindings of the other variables. We can lift this property to clauses and predicates and say that a clause is determinate, if all of its literals are determinate and a predicate is determinate if all of its clauses are determinate.

ij-determinacy

Definition 7.25 — ij-determinacy.
A literal λ is called *i–determinate*, if it is determinate and if

$$\max\{\mathrm{depth}(X) : X \text{ occurs in } X\} \leq i$$

i.e. the maximum depth of its variables is bounded by i.
A clause λ_0 :- $\lambda_1, \ldots, \lambda_{n-1}$ is called ij–determinate, iff:

1. i is the maximum depth at which a determinate variable occurs.

2. j is the maximum degree of any variable occurring in $\lambda_1, \ldots, \lambda_{n-1}$.

●

An example for the application of the biases discussed so far is the learning sustem Golem, [Muggleton and Feng, 1992, Muggleton and Feng, 1990, **?**]. It takes two sets of ground facts, $E^{\mathbf{0}}$ and $E^{\mathbf{1}}$, as sample and Π as background knowledge. Π is also assumed to be ground; if it is not, it is represented by an h-easy version of it. The result is that Π consists of ground facts only (or can be transferred into one). On this set of formulas, ∇_{Π} is applied on pairs of examples to infer hypotheses for the target predicate. Variable introduction is biased by setting a threshold for i and j: We may not generalise to new variables or simply introduce new variables if they would exceed the ij-determinacy. With these heuristics, Golem performs more less the same learning loop as Foil does:

1. To learn a single rule, randomly pick $\varphi, \psi \in E^{\mathbf{1}}$ and compute $\nabla_{\Pi}(\varphi, \psi)$ until coverage on $E^{\mathbf{1}}$ cannot be increased any more and no $e \in E^{\mathbf{0}}$ is covered

2. Remove redundant literals

3. To learn sets of rules, remove the examples that are covered by a rule and continue until all examples are covered (none are left).

The directedness of SLD-Resolution suggests a directedness of determinacy. In fact the definition of determinacy is motivated by SLD-resolution.

Restricting declaraitve semantics by procedural semantics

If you are familiar with the Prolog programming language, you might have seen *predicate specifications.* The expression

$$\mathtt{p}(+X, +Y, -Z)$$

meas that when trying to resolve a (negative) literal $\mathtt{p}(X', Y', Z')$, X' and Y' need to be *ground* terms, whereas Z' will be substituted after all body literals of the clause have been resolved.

Example 7.22 Consider our predicate b*lack* as in example 7.7. It takes one single argument and succeeds, if the provided argument represents a black object:

$$?\!-\!\mathtt{black}(\bullet) \sqcap_{\mathrm{RES}} \mathtt{black}(\bullet) = \{\}$$

In this case, we receive a definite answer, `Yes.`, for $\mu = \{\}$. But what if we ask $?\!-\!\mathtt{black}(X)$? Then, we have several options to resolve the goal using several substitutions that in Prolog are discovered by backtracking:

$$\mu_0 = [X \mapsto \bullet]\,,\ \mu_1 = [X \mapsto \blacksquare]\,,\ \mu_2 = \left[X \mapsto \blacklozenge\right]\,,\ \mu_3 = [X \mapsto \blacktriangle]$$

This is useful for collecting the set of all things we know to be black. In our case, we want to use the predicate `black` to *check* whether a *certain instance* is black. The convention then is to declare the predicate `black` as a predicate whose argument needs to be ground when trying to resolve it. It is specified by writing $\mathtt{black}(+X)$. ●

The previous example was very simple because the predicate took only one argument. Now consider example 7.13 and equation 7.45 of the subsequent example. We induced a predicate consisting of two clauses

$$\begin{aligned} h_0 &= \mathtt{black\_polygon}(X)\mathtt{:-}\ \mathtt{colour}(X, \mathtt{black}), \mathtt{shape}(X, \mathtt{square}). \\ h_1 &= \mathtt{black\_polygon}(X)\mathtt{:-}\ \mathtt{colour}(X, \mathtt{black}), \mathtt{shape}(X, \mathtt{diamond}). \end{aligned}$$

which were to defined to check whteher a certain object X is a black polygon. An experienced Prolog programmer never would write such a predicate since if he was asked to implement a predicate `white_polygon`, he would have to copy the clauses and replace all occurences of the ground term `black` with `white`. Instead, he would write:

$$\begin{aligned} \mathtt{coloured\_polygon}(X, C) \quad \mathtt{:-} \quad & \mathtt{colour}(X, C), \\ & \mathtt{member}(X, [\mathtt{square}, \mathtt{diamond}]). \end{aligned} \tag{7.71}$$

Exercise 7.33 (◊⧫)

- ◊ Specify the predicate `coloured_polygon` and all predicates appearing in the body of the clause. For `member`, consult a Prolog manual or, for example, [Bratko, 1986].
- ⧫ Compute the free assignment degree of all the variable occurences in the predicate definition of equation (7.71). Then, determine the smalles i and j such that the predicate is ij-determinate!

If we want to specify a predicate which delivers a definite answer subsitution, things are a bit different.

Example 7.23 In order to retrieve objects from our knowledge base which satisfy certain goals, we would implement a predicate $\texttt{is\_there\_a}(+C, +S, -X)$ as follows:

$$\begin{aligned} \texttt{is\_there\_a}(C, S, X) \quad \texttt{:-} \quad & \texttt{shape}(X, S), && (7.72) \\ & \texttt{colour}(X, C). && (7.73) \end{aligned}$$

By SLD-Resolution, and an initial assignment $\alpha(C) = \texttt{black}$ and $\alpha(S) = \texttt{circle}$, we resolve $?{-}\texttt{shape}(X, \texttt{circle})$ against the fact $\texttt{colour}(\bullet, \texttt{circle})$ by applying $\mu_0 = [X \mapsto \bullet]$. The substitution lemma then implies the assignment $\alpha(X)$ changes to $\alpha\,[X \mapsto \bullet]$ such that $\alpha(X) = \bullet$. Hence $?{-}\texttt{colour}(X, \texttt{black})$ becomes $?{-}\texttt{colour}(\bullet, \texttt{black})$ which, by $\mu_1\,[]$, resolves with the fact $\texttt{colour}(\bullet, \texttt{black})$ to the emtpy clause. Now we know that

$$\forall \alpha \in s^{\mathrm{Var}} : \alpha(X) = \bullet \Longrightarrow \Pi \approx\!\!\!\!|_{\alpha}\ \texttt{is\_there\_a}(\texttt{black}, \texttt{circle}, X). \qquad (7.74)$$

This means the goal is true for a substitution $\sigma = \{X \mapsto \bullet\}$ and we can answer: Yes, there is an X such that X is a black circle—namely $X = \bullet$. ●

Exercise 7.34 (◊♦♦♦♦)

- ◊ In the last example we have shown that $\Pi \vdash_{\mathrm{SLD}} \texttt{is\_there\_a}(\texttt{black}, \texttt{circle}, \bullet)$. Why is equation (7.74) true, too?
- ♦ What happens for $?{-}\texttt{is\_there\_a}(\texttt{white}, \texttt{triangle}, X)$?
- ♦ Let there be an alternative definition of the predicate is_there_a:

  $$\begin{aligned} \texttt{is\_there\_a}(C, S, X) \quad \texttt{:-} \quad & \texttt{colour}(X, C), \\ & \texttt{shape}(X, S). \end{aligned}$$

  Compute the free assignment degree of all the variable occurences. Then, determine the smallest i and j such that the predicate is ij-determinate!
- ♦♦ What happens for $?{-}\texttt{is\_there\_a}(\texttt{white}, \texttt{triangle}, X)$?
  What happens for $?{-}\texttt{is\_there\_a}(\texttt{black}, \texttt{tetragon}, X)$?

From definition of biases we discussed so far and the observations we made by examining SLD-proofs, we an conclude that it is a good idea to restrict the search space for hypotheses by predetermination of predicate specifications. If concerned with learning from examples rather than with rule invention, we also know the predicate name and arity for which we want to learn a definition. Specifications of predicates are also known as *modes*. Modes like

> The target has a head literal, the arguments of which shall satisfy the I/O behaviour determined by $p(\texttt{+}X, \texttt{+}Y, \texttt{-}Z)$

can be found in Progol or *m*Foil.
Finally, a very simple method for reducing search space is simply by limiting the language available to formulate a hypothesis. Such limitations, also known as *predicate schemes*, like

> The target is a 3–literal Horn clause which matches $\texttt{p}(\_, \_, X)\,\texttt{:-}\ Q(X, Y), \texttt{p}(\_, X, Y)$.

are used in Mobal, [?].[14]

7.4.4 Inverted Entailment

All our efforts so far concentrated on the fact that $\mid\!\approx$ is too expensive to be checked for every single guess what might be an appropriate hypothesis. As a consequence we focussed on finding syntactical refinement operators that at least come close to inverting entialment.

Example 7.24 Computing (relative) least generalisations with respect to θ-subsumption are not least general with respect to implication. Imagine that

$$\begin{array}{rcl}\chi & := & \mathtt{p}(f(f(\mathtt{0}))) \mathtt{:-}\ \mathtt{p}(\mathtt{0}). \\ \xi & := & \mathtt{p}(f(\mathtt{1})) \mathtt{:-}\ \mathtt{p}(\mathtt{1}).\end{array}$$

The least general generalisation with respect to θ-subsumption is

$$\varphi = \chi \triangledown \xi = \mathtt{p}(f(x)) \mathtt{:-}\ \mathtt{p}(y).$$

But the least general generalsiation with respect to logic implication (i.e. entailment) is:

$$\psi = \chi \curlyvee \xi = \mathtt{p}(f(x)) \mathtt{:-}\ \mathtt{p}(x).$$

where $\psi = \chi\sigma^{\smile} = \xi\sigma^{\smile}$ with

$$\sigma^{\smile} = [f(\mathtt{0}) \mapsto x, f(\mathtt{1}) \mapsto x, \mathtt{0} \mapsto x, \mathtt{1} \mapsto y]$$

It is clear that

$$\varphi = \mathtt{p}(f(x)) \mathtt{:-}\ \mathtt{p}(y) \mid\!\approx \mathtt{p}(f(x)) \mathtt{:-}\ \mathtt{p}(x) = \psi$$

and it is also clear that φ is way too general! ●

Our example contains is a *recursive* predicate definition. This clause contains the same literal twice: The positive version makes the rule head, the negative one is element of the rule body. So when trying to resolve p, we have to resolve p again—just with another argument. Therefore, clauses that define recursive predicates are also known as *self-resolving* clauses. From this point of view one can see that φ in the preceeding example is really useless: proving a certain instance of p by proving *any* any instance of it leads into a viciuos circle.

Exercise 7.35 (◊) Define a criterion by which one can easily determine whether an arbitrary self-resolving clause maybe useful or certainly is not useful at all! — Think of biases!

There is, however, one important observation that we have not taken into account so far. [Gottlob, 1987] states that:

Definition 7.26 — Gottlob's Lemma.

Gottlob's Lemma

Let there be two clauses φ and ψ as follows:

$$\begin{array}{rcl} \varphi & = & \{\kappa_0, \ldots, \kappa_{k-1}, \neg\kappa_k, \ldots, \neg\kappa_{m-1}\} \\ \psi & = & \{\lambda_0, \ldots, \lambda_{l-1}, \neg\lambda_l, \ldots, \neg\lambda_{n-1}\} \end{array}$$

where for

$$\begin{array}{rcl} \varphi^{\mathbf{1}} = \{\kappa_0, \ldots, \kappa_{k-1}\} & \text{and} & \psi^{\mathbf{1}} = \{\lambda_0, \ldots, \lambda_{l-1}\} \\ \varphi^{\mathbf{0}} = \{\neg\kappa_k, \ldots, \neg\kappa_m\} & \text{and} & \psi^{\mathbf{0}} = \{\neg\lambda_0, \ldots, \neg\lambda_n\} \end{array}$$

$\varphi^{\mathbf{1}}, \psi^{\mathbf{1}}$ contain only positive literals and $\varphi^{\mathbf{0}}, \psi^{\mathbf{0}}$ contain only negative literals. Then,

$$\models \varphi \longrightarrow \psi \quad \Longrightarrow \quad \varphi^{\mathbf{1}} \mathrel{|\!\!\lhd} \psi^{\mathbf{1}} \wedge \varphi^{\mathbf{0}} \mathrel{|\!\!\lhd} \psi^{\mathbf{0}} \tag{7.75}$$

So when one clause implies another, then its subsets of positive and negative literals subsume the corresponding subsets of the other. ●

Formula (7.75) can be transformed into

$$\varphi \mathrel{|\!\!\approx} \psi \quad \Longrightarrow \quad \varphi^{\mathbf{1}} \mathrel{|\!\!\lhd} \psi^{\mathbf{1}} \wedge \varphi^{\mathbf{0}} \mathrel{|\!\!\lhd} \psi^{\mathbf{0}} \tag{7.76}$$

The hypothesis H we are to infer shall at least entail all the positive examples of our target concept:

$$\Pi \cup H \mathrel{|\!\!\approx} E^{\mathbf{1}}$$

where, as you will remember, $E^{\mathbf{1}}$ is a set of *(ground) facts* $\{\{\lambda_0\}, \ldots, \{\lambda_{n-1}\}\}$. Then, we can rewrite the formula as

$$\Pi \cup H \mathrel{|\!\!\approx} \{\{\lambda_0\}, \ldots, \{\lambda_{n-1}\}\}$$

Now, we try a little trick and require H to *consist of facts only*, too. Then we can write

$$\Pi \cup \{\{\kappa_0\}, \ldots, \{\kappa_{m-1}\}\} \mathrel{|\!\!\approx} \{\{\lambda_0\}, \ldots, \{\lambda_{n-1}\}\}$$

This means that there is a substution such that $\Pi \cup \{\{\kappa_0\}, \ldots, \{\kappa_{m-1}\}\} \cup \{\{\neg\lambda_0\}, \ldots, \{\neg\lambda_{n-1}\}\}$ is not satisfiable. Furthermore, by equation (7.75), the set of positive literals κ_i subsumes the set of positive λ_j, and the same holds for the negative literal subsets. If we now *add the negation* of $E^{\mathbf{1}}$ to Π we can infer a set of formulas (or, more specifically, a conjunction of literals), which we *do not want to be implied* by the hypothesis H we are looking for. Negation on clauses with free variables in them requires skolemisation in order to avoid existential quantifiers. Let there be a clause

$$\begin{array}{rcl} \varphi & = & \kappa_0, \ldots, \kappa_{k-1} \texttt{:-}\ \lambda_0, \ldots, \lambda_{l-1}. \\ & = & \{\kappa_0, \ldots, \kappa_{k-1}, \neg\lambda_0, \ldots, \neg\lambda_{l-1}\} \end{array}$$

[14] Note that schemes like these explicitely include depth, degree, determinacy, and specification biases. Note also, that Q is a placeholder for a predicate name; it is, so to say, a second order variable.

such that $\varphi^{\mathbf{0}} = \{\lambda_j : j \in \mathbf{l}\}$ and $\varphi^{\mathbf{1}} = \{\kappa_j : i \in \mathbf{k}\}$. Then, its *complement* is the *set* of all *unary clauses* where each clause is the skolemised negation of a literal in φ:

$$\overline{\varphi} \quad := \quad \{\{\kappa_0\}, \ldots, \{\kappa_{k-1}\}, \{\lambda_0\}, \ldots, \{\lambda_{l-1}\}\}\,\sigma_{sk} \tag{7.77}$$

Note that this is not a single clause as was φ but an entire clausal theory consisting of a *conjunction* of unary, ground clauses.[15] Using complementation we define:

Bottom literal set, BotLit

Definition 7.27 — Bottom literal set, BotLit.
Let Π be a set of clauses and φ a clause. The set

$$\mathrm{BotLit}(\Pi, \varphi) := \{\lambda : \Pi \cup \overline{\varphi} \mathrel{|\!\approx} \neg\lambda \text{ and } \lambda \text{ is a ground literal}\} \tag{7.78}$$

is called the *bottom set of φ for* Π. ●

Roughly speaking, it is the set of all facts that are rejected when adding the negation of φ to our background knowledge. Now comes the trick: If we assume φ to be *true*—especially, if $\varphi \in E^{\mathbf{1}}$, then a *subset* of the according bottom literal set is a clause that can be considered a good hypothesis for φ!

Inverse Entailment

Definition 7.28 — Inverse Entailment.
Let Π be a set of clause. Let there be a set of ground facts $E^{\mathbf{1}}$, the positive examples. Let $\varphi = \{\kappa\}$ be clause (in this case, a fact) from $E^{\mathbf{1}}$. Let

$$\mathrm{BotLit}(\Pi, \varphi) = \{\lambda : \Pi \cup \{\{\neg\kappa\}\} \mathrel{|\!\approx} \neg\lambda \text{ and } \lambda \text{ is a ground literal}\}$$

Then, by inverse entailment, we consider H with

$$H \mathrel{|\!\lhd} B\sigma_{sk} \text{ for some } B \subseteq \mathrm{BotLit}(\Pi, \varphi)$$

a *hypothesis* derived by *inverse entailment* from φ wrt Π. ●

There are a few problems here: First, the bottom literal set is not neccessarily finite. Luckily, the definition of inverse entailment includes a nice workaround as we only need to consider subsets. Therefore, we can confine our search to finite sets. The second problem is that there are still *too many* of them. And finally there is a small gap between the generality of definitions and the examples we kept in mind: The definitions work for *arbitrary clauses* wheras we spoke of ground facts as formulas only. What we are looking for is located just in the gap between—definite clauses. But first, let us consider a small example.

Example 7.25 Let

$$\begin{aligned}\Pi &= \left\{\begin{array}{l}\mathtt{polygon}(X)\mathtt{:-}\ \mathtt{edges}(X,4).,\\ \mathtt{black\_tetragon}(X)\mathtt{:-}\ \mathtt{black}(X), \mathtt{polygon}(X), \mathtt{square}(X).\end{array}\right\}\\ \varphi &= \mathtt{black\_tetragon}(X)\mathtt{:-}\ \mathtt{edges}(X,4), \mathtt{black}(X).\end{aligned}$$

[15] More generally, the complement of any formula is its negation with all variables replaced by Skolem constants.

Then, complementation leads to

$$\overline{\varphi} \quad = \quad \{\texttt{:- black\_tetragon}(\blacksquare).,\texttt{edges}(\blacksquare,4).,\texttt{black}(\blacksquare).\}$$

As a result,

$$\text{BotLit}(\Pi,\varphi) \quad = \quad \{\texttt{square}(\blacksquare),\neg\texttt{black}(\blacksquare),\neg\texttt{edges}(\blacksquare,4),\neg\texttt{polygon}(\blacksquare)\}$$

By inverse entailment we can build *several* causes ψ for which $\psi \mid\!\triangleleft\, \varphi$:

$$\begin{aligned}
\psi_0 &= \texttt{square}(\blacksquare)\texttt{:- black}(\blacksquare),\texttt{edges}(\blacksquare,4),\texttt{polygon}(\blacksquare).\\
\psi_1 &= \texttt{square}(X)\texttt{:- black}(X),\texttt{edges}(X,4),\texttt{polygon}(X).\\
\psi_2 &= \texttt{square}(X)\texttt{:- edges}(X,4),\texttt{polygon}(X).\\
\psi_3 &= \texttt{square}(X)\texttt{:- black}(X),\texttt{polygon}(X).
\end{aligned}$$

Exercise 7.36 (♦♦)

- ♦ Is there a clause ψ' for which neither $\Pi \cup \{\psi'\} \not\mid\!\approx \texttt{square}(\square)$ nor $\Pi \cup \{\psi'\} \not\mid\!\approx \texttt{square}(\blacksquare)$? Why?
- ♦ Evaluate ψ_i, $i \in \mathbf{4}$ with respect to the base set $\left\{\Diamond,\bullet,\triangle,\bigcirc,\blacksquare,\blacklozenge,\square,\blacktriangle\right\}$.

The method of inverse entailment was first presented in [Muggleton, 1995]. Along with its theory, Muggleton presented an implementation in which the construction of saturants and the subsequent search for clauses subsuming them is guided by a heuristic measure of compression (rther than an entropy-based measure) and strong biases on the form of the clauses that are to be considered. These biases are defined in terms of so-called *mode-declarations* which are similar to all the biases we discovered in section 7.4.3.
A beautiful concise summary of the most important ideas behind "semantic" induction is [Yamamoto, 1997]. It refers to the most influential articles on which inverted entailment is based as well as the two most completive contributions; [Rouveirol, 1992] and [Nienhuys-Cheng and Wolf, 1996].

7.5 Summary

Inductive Logic Programming was at it's peak in the mid 1990's which is why still many standard references are fifteen years old by now. One of the first textbooks on ILP is [**?**]. A bit more detailed information can be found in [**?**]. A very readable and concise summary is an extended article by Muggleton and de Raedt published in the Machine Learning Journal, [**?**]. Recent textbooks are [Raedt, 2008]; [Kersting and Raedt, 2000, Raedt and Kersting, 2004] and [Kersting, 2008] focus on extending ILP by probabilisitc methods.

Inductive logic programming is about finding logic programs which describe unknown target concepts. A program clause is a definite clause; clauses with at least one negative literal are rule clauses. The rule head represents a relation between its arguments—an n-ary predicate symbol is interpreted as an n-ary relation. The entire predicate is defined by a *disjunction* of clauses, and every clause by a *conjunction* of literals. These literals again refer to other predicates; usually to predicates that are predefined in Π. Therefore, the predicate definition

$$\varphi = \texttt{black\_polygon}(X) \quad \texttt{:-} \quad \texttt{black}(X), \texttt{tetragon}(X). \tag{7.79}$$

$$\psi = \texttt{black\_polygon}(X) \quad \texttt{:-} \quad \texttt{black}(X), \texttt{triangle}(X). \tag{7.80}$$

is just about the same as

$$x \in \mathit{black\_polygon} \subseteq s \quad :\Leftarrow \quad x \in (\mathit{black} \cap \mathit{tetragon}) \cup (\mathit{black} \cap \mathit{triangle})$$

If we agree that anything we can't prove is false, then our hypothesis becomes

$$\mathit{black\_polygon} \quad := \quad (\mathit{black} \cap \mathit{tetragon}) \cup (\mathit{black} \cap \mathit{triangle})$$

So if objects, say ■, ▲ and ◆ are all black and either a tetragon or a triangle, they are indiscernible by our knowledge about black polygons. In other words, it does not matter at all whether the property of being black or being a tetragon is expressed in terms of features in an according information system, by predicates or by relations.

Whenever two objects are elements of the satisfaction set of a predicate, they are indiscernible with respect to the relation by which this predicate is interpreted: For $\mathbf{R} = \{\mathit{colour}, \mathit{tetragon}\}$ and the program above, it holds that

$$\begin{aligned} \{\blacksquare, \blacklozenge\} = [\![\mathbf{R}]\!]\,\{\blacksquare, \blacklozenge\} \quad &\Longleftrightarrow \quad \blacksquare \bar{\bar{\mathbf{R}}} \blacklozenge \\ &\Longleftrightarrow \quad \Pi \cup \{\varphi, \psi\} \mathrel{|}\!\!\approx \left\{ \begin{array}{l} \texttt{black\_polygon}(\blacksquare), \\ \texttt{black\_polygon}(\blacklozenge) \end{array} \right\} \end{aligned}$$

If there are two concepts with instances

$$\begin{aligned} \{\blacktriangle, \blacklozenge, \bullet\} &\subseteq c_0 \\ \{\triangle, \blacklozenge, \bullet\} &\subseteq c_1 \end{aligned}$$

then, with $\mathbf{R} = \{\mathit{Colour}, \mathit{Size}, \mathit{Shape}\}$,

$$[\![\mathbf{R}]\!] c_0 = c_0 = \langle\!|\mathbf{R}|\!\rangle c_0 \quad \text{and} \quad [\![\mathbf{R}]\!] c_1 = c_1 = \langle\!|\mathbf{R}|\!\rangle c_1$$

Similarly, and there is just a simple representation shift involved herein,

$$\begin{aligned} \texttt{c\_0}(X) \quad &\texttt{:-} \quad \texttt{colour}(X, \texttt{black}), \texttt{size}(X, Y), \texttt{shape}(X, Z). \\ \texttt{c\_1}(X) \quad &\texttt{:-} \quad \texttt{colour}(X, Y), \texttt{size}(X, \texttt{small}), \texttt{shape}(X, Z). \end{aligned}$$

As one can see, the above predicate definitions are highly redundant—and, equivalently, there exist reducts of $\mathbf{R}$: For $\mathbf{P} = \{\mathit{Colour}\}$ and $\mathbf{Q} = \{\mathit{Size}\}$

$$[\![\mathbf{P}]\!] c_0 = c_0 = \langle\!|\mathbf{P}|\!\rangle c_0 \quad \text{and} \quad [\![\mathbf{Q}]\!] c_1 = c_1 = \langle\!|\mathbf{Q}|\!\rangle c_1$$

The same holds for the logic program: The free variables do not at all contribute to the restriction of the satisfaction set for the predicates. Hence,

$$\begin{array}{lcl}
\texttt{c\_0}(X) & \texttt{:-} & \texttt{colour}(X, \texttt{black}). \\
\texttt{c\_1}(X) & \texttt{:-} & \texttt{size}(X, \texttt{small}).
\end{array}$$

Finally, let us consider rule induction. We observe

$$\begin{array}{rcl}
\blacklozenge, \blacksquare, \blacktriangle, \bullet \in \mathit{black} & = & s - \{\lozenge, \oslash\} = s - \mathit{white} \\
\blacksquare, \blacktriangle, \bullet, \lozenge \in \mathit{large} & = & s - \left\{\blacklozenge, \oslash\right\} = s - \mathit{small}
\end{array}$$

and

$$\blacksquare, \lozenge, \blacklozenge \in \mathit{tetragon}, \;\; \mathit{triangle} = \{\oslash\}\,, \;\; \mathit{circle} = \{\bullet\}$$

then, for $c = \left\{\blacklozenge, \blacksquare, \blacktriangle\right\}$ we find a reduct $\mathbf{R} = \{\mathit{Colour}, \mathit{Shape}\}$ such that $[\![\mathbf{R}]\!]c = c = \langle\!|\mathbf{R}|\!\rangle$. For the according facts

```
white(◊). white(△).
black(●). black(■). black(⧫). black(▲).
triangle(△). triangle(▲).
tetragon(⧫). tetragon(◊). tetragon(■).
circle(●).
```

the induced rules are

$$\begin{array}{lcl}
\texttt{c}(X) & \texttt{:-} & \texttt{black}(X), \texttt{tetragon}(X). \\
\texttt{c}(X) & \texttt{:-} & \texttt{black}(X), \texttt{triangle}(X).
\end{array}$$

In this chapter we have discovered three ways of finding relational descriptions of new concepts by inducing logic programs which define according satisfaction sets.

Chapter 8

Learning and Ensemble Learning

We discussed several paradigms of how to extract relational knowledge from sets of data. All of were based on an intuitive understanding of what it means to learn a new concept. In this chapter we will try to find a more abstract definition of *learnability.*
The second topic of this chapter is motivated by two questions: Why should we learn one complex classifier instead of several simple ones—and why should we learn from all observations instead from only the difficult ones?

8.1 Learnability

It is worth a second a thought whether some question can be answered from a set of information before one tries to extract the knowledge required to derive such an answer from all the data. There is no use in trying to answer a question to which we have no answer; it is hard to try solving an ill-posed problem, and it is not a wise idea to formulate an answer in a language that is not sufficiently expressive in order to grasp what we want to say[1].
Let us recall the first definition of a learning algorithm (definition 3.11): Presupposing that a concept c actually *is learnable* (whatever that means), a learning algorithm is a procedure that from a set of examples induces a function that computes an approximation of c:

$$\texttt{Alg}(S_\mu(m,t)) = h \approx t$$

[1] Things are a bit different in everyday life, though. But trying to extract some formal concept from insufficient information often results in rather funny results, [Müller, 2008]

Reverting this definition we can try to formulate a definition of *learnability*:

Learnability

Definition 8.1 — Learnability.
A problem $x \in c$ is *(correct) learnable*, if there is an algorithm $\mathtt{Alg}$ which, given a sample $S_\mu(m, t)$, delivers a function h such that $h = \chi(c)$. ●

This definition is quite rigid, and there exist weaker versions:

1. First, we consider only (finite) *subsets* $s \subseteq U$. It is much easier to examine only a subset of objects and it is simpler to find a hypothesis explaining fewer examples.

2. A problem is *approximately correct* learnable, if $h \approx \chi(c)$. In this case one needs to decide whether $h \approx c$. It requires an *error measure*, of which there are two basic types: In section 3.4.1 we defined *pointwise* error measures and measures based on sets of such errors; in section 3.4.2 we introduced error measures that evaluate hypotheses as a whole.

3. Finally, a problem is *probably approximately correct (PAC)* learnable if, with a certain confidence, we can gurantee an approximately correct hypothesis.

8.1.1 Probably approximately correct learning

The idea of PAC learning was coined by Leslie Valiant in 1984, [Valiant, 1984].

PAC Learnability

Definition 8.2 — PAC Learnability.
A problem in $\mathfrak{U}$ is *PAC-learnable*, if there exists a *probably approximately correct (PAC)* learning algorithm $\mathtt{Alg}$. $\mathtt{Alg}$ is PAC, if there exists a lower bound for the number of required examples such that for all samples of equal or greater length, with a probability of at most δ, t can *not* be approximately correct learned with an error of less than ε.
Let $0 < \varepsilon, \delta < 1$. Let $\mathtt{Alg}$ be a learning algorithm for $\mathfrak{U}$ with U being the base set of $\mathfrak{U}$.Let $t = \chi(c)$, $c \subseteq U$, and μ a (probability) distribution on U. $\mathtt{Alg}$ is PAC, if:

$$\exists m_0 : m \geq m_0 \Longrightarrow \mu^m \left\{ \mathtt{s} :\in S_\mu(m, t) : \mathrm{error}_U^\mu(\mathtt{Alg}(\mathtt{s}), t) < \varepsilon \right\} > 1 - \delta \qquad (8.1)$$

Therein, m_0 is a value that must be computable by a function of ε and δ. The ":∈" is there to denote, that $S_\mu(m, t)$ is not a function, but nondeterministically delivers $\mathtt{s}$. ●

PAC Learnability
A problem is *probably approximately correct learnable*, if there is an algorithm for which only by defining a minimum confidence and maximum error, we can guarantee that for any sample which contains at least a fixed minimum number of examples, this algorithm with a certain probability delivers a hypothesis that is sufficiently accurate—with an arbitrary and unknown distribution on our universe.

Example 8.1 Let $U = \mathbb{N}$ and $c_{\text{odd}} = \{1, 3, 5, \ldots\}$ be the set of odd numbers. Then, $h(x) = 1 :\Longleftrightarrow x \mod 2 = 0$ is a hypothesis for which $h = \chi(c_{\text{odd}})$. It is a correct hypothesis. ●

Example 8.2 Let U be the set of objects with shape $\square, \circ, \Diamond$, or $\triangle$ and a colour that is *white*, *light*, *dark*, or *black*. Let c_{wwbb} be the set of sequences of two white objects followed by two black objects (i.e. all sequences have length 4^n):

$$c_{\text{wwbb}} = \left\{ (w_0 w_1 b_0 b_1)^n : \begin{array}{l} i \in \mathbf{2} \wedge w_i, b_i \in U \wedge \\ \mathit{colour}(w_i) = \mathit{white} \wedge \mathit{colour}(b_i) = \mathit{black} \end{array} \right\}$$

We define a distance function on the set of colour values $\text{dist}_{\text{colour}} : \text{cod}(\mathit{colour}) \times \text{cod}(\mathit{colour}) \to \left\{0, \frac{1}{4}, \frac{1}{2}, \frac{3}{4}, 1\right\}$ as follows:

$\text{dist}_{\text{colour}}$	□	▨	▩	■
□	0	$\frac{1}{4}$	$\frac{3}{4}$	1
▨	$\frac{1}{4}$	0	$\frac{1}{4}$	$\frac{3}{4}$
▩	$\frac{3}{4}$	$\frac{1}{4}$	0	$\frac{1}{4}$
■	$\frac{1}{4}$	$\frac{3}{4}$	$\frac{1}{4}$	0

Suppose the hypothesis generated by some `Alg` is the characteristic function of the set

$$c_{wldb} = \{(x_{\text{white}} x_{\text{light}} x_{\text{dark}} x_{\text{black}})^n : x_i \in U \wedge \mathit{colour}(x_i) = i\}$$

Based on the definition of $\text{dist}_{\text{colour}}$, we define a distance measure on U^{4n} as follows:

$$\begin{aligned} \text{dist}(x_0 x_1 x_2 x_3, y_0 y_1 y_2 y_3) &:= \frac{1}{4} \sum_{i \in \mathbf{4}} \text{dist}_{\text{colour}}(x_i, y_i) \\ \text{dist}(x_0 x_1 x_2 x_3 v, y_0 y_1 y_2 y_3 w) &:= n \frac{\text{dist}(x_0 x_1 x_2 x_3, y_0 y_1 y_2 y_3) + \text{dist}(v, w)}{n+1} \end{aligned}$$

for $v, w \in U^{4n}$. We compare an element $x \in c_{\text{wwbb}}$ and $y \in c_{\text{wldb}}$:

$x =$	□	□	■	■	□	□	■	$\cdots$
$y =$	□	▨	▩	■	□	▨	▩	$\cdots$
$\text{dist}_{\text{colour}} =$	0	$\frac{1}{4}$	$\frac{1}{4}$	0	0	$\frac{1}{4}$	$\frac{1}{4}$	$\cdots$
$\text{dist} =$	$\vert$	$\frac{1}{8}$		$\vert$				$\cdots$

which demonstrates that the entire distance for any sequence of length n of symbols from c_{wwbb} to any sequence of length n of symbols from c_{wldb} is 0.125. Therefore, $h = \chi(c_{\text{wldb}})$ is ε–correct w.r.t. c_{wwbb} if $\varepsilon > 0.125$. ●

Now, let us consider the following problem: Let $U = [0, 1]^2 \subset \mathbb{R} \times \mathbb{R}$, where for an instance $x = \langle c_x, s_x \rangle$ the colour of x is described by c_x and the size of x

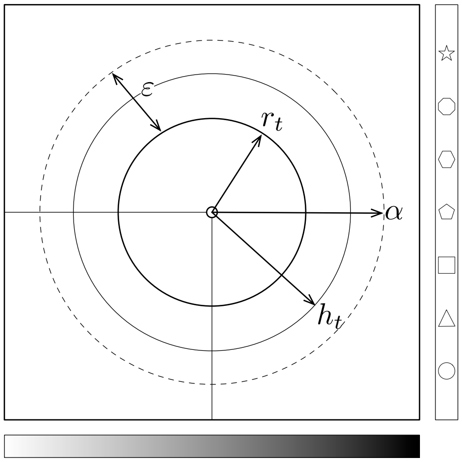


Figure 8.1: Learning Discs

by s_x. Note that we have left the universe with objects of only finitely many colours and sizes—the representation space is infinite! The target concept t is the set of objects that belong to the disc with center $\left\langle \frac{1}{2}, \frac{1}{2} \right\rangle$ and radius $r_t = \frac{1}{4}$. In other words,

$$t(x) = 1 :\Longleftrightarrow x \in \left\{ x \in U : \sqrt{(c_x - \frac{1}{2})^2 + (s_x - \frac{1}{2})^2} \leq r_t \right\} \tag{8.2}$$

For an illustration of this problem, see figure 8.1. In order to compare two objects, we define:

$$\operatorname{dist}(x, y) = \sqrt{(c_x - c_y)^2 + (s_x - s_y)^2} \text{ and } \|x\| = \operatorname{dist}(x, \left\langle \frac{1}{2}, \frac{1}{2} \right\rangle)$$

Then we allow a certain degree of uncertainty which may result in wrong classification results. We add an "ε-region of tolerance" to the target concept which means that we may accept slightly larger values than r_t as distance to $\left\langle \frac{1}{2}, \frac{1}{2} \right\rangle$:[2]

$$c_\varepsilon := \{x \in U : \|x\| \leq r_t + \varepsilon\}.$$

The ε-tolerance-area c_{err} is a "rim" around the target concept with a width of ε.

[2] "Larger" instead of "smaller or larger", because the algorithm we shall use can only overestimate the value of r_t. The same argument applies to adding ε to t_t instead of actually defining an error *region around* the target concept.

Learning by generalsiation:

```
$r_h := 0$
$\mathbf{s} := S_\mu(m,t)$
FOREACH ($\langle x, t(x)\rangle \in \mathbf{s}$) DO
{    IF $t(x) = 1$ THEN
         $r_h := \max(\{r_h, \|x\|\})$
}
RETURN ($r_h$)
```

Learning by specialisation:

```
$r_h := 1$
$\mathbf{s} := S_\mu(m,t)$
FOREACH ($\langle x, t(x)\rangle \in \mathbf{s}$) DO
{    IF $t(x) = 0$ THEN
         $r_h := \min(\{r_h, \|x\|\})$
}
RETURN ($r_h$)
```

where $t(x) = 1 :\Longleftrightarrow \|x\| \leq r_c$.

Figure 8.2: Learning a disc

Example 8.3 Let us try a purely geometric interpretation of our example: For $r_t = \frac{1}{4}$ and $\varepsilon = \frac{1}{8}$,

$$A(c) = \frac{1}{16}\pi \approx 0.196 \approx 20\% \tag{8.3}$$

$$A(c_\varepsilon) = \frac{9}{64}\pi \approx 0.442 \approx 44\% \tag{8.4}$$

$$A(c_{\text{err}}) = A(c_\varepsilon) - A(c) = \frac{5}{64}\pi \approx 0.245 \approx 25\% \tag{8.5}$$

It means that, presupposing a continuous uniform distribution, about 20% of all objects are elements of our target concept, while 80% are not. In 56% of all cases we can correctly reject an object; and in 25% of all cases we we may or may not make a false positive prediction. ●

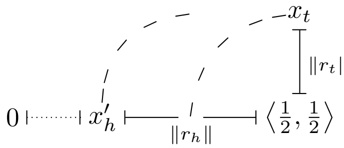
A disk (or rather its radius) is PAC-learnable, if there is a PAC algorithm for it. Since the disc is defined by a certain radius r_t around $\left\langle \frac{1}{2}, \frac{1}{2} \right\rangle$, the learning alorithm has to find with a certain probability a value r_h such that the difference between r_t and r_h is at most ε. An algorithm to approximate r_c by r_h is quite simple: Start with an empty hypothesis (that is, $r_h = 0$) and for every positive example x, redefine r_h to $\|x\|$, if $\|x\| > r_h$. Alternatively, we can start off with a most general hypothesis and successively restrict it: Let $r_h > \frac{1}{2}$ and for each *negative* example that is closer to $\left\langle \frac{1}{2}, \frac{1}{2} \right\rangle$ than r_h, decrease r_h to the newly observed radius until we have processed all examples. The algorithms are shown in figure 8.2; we will from now on refer to the specialiastion algorithm. After m iterations, with at least one positive example in $\mathbf{s}$, it holds that

$$\frac{1}{2} \geq r_h \geq r_c.$$

The error set $\mathrm{errset}_U(h,t)$ is the rim with width $r_h - r_c$ around the disc with radius r_c. More precisely,

$$\mathrm{errset}_U(h,t) = \left\{ x \in U : r_h \geq \sqrt{(\frac{1}{2} - c_x)^2 + (\frac{1}{2} - s_x)^2} > r_t \right\}$$

But what about the actual *error* of h with respect to t? And what about the reliability of these algorithms?
We examine the specialisation algorithm as shown in figure 8.2 on the right side for the disk learning in figure 8.1. A closer look at our learning problem reveals that there is some kind of symmetry involved, since all discs represented by hypotheses are concentric to $\langle \frac{1}{2}, \frac{1}{2} \rangle$:



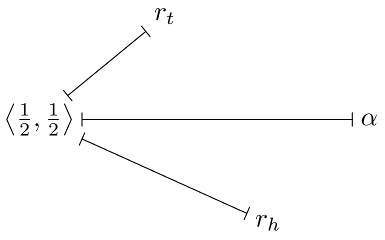
where $\|x'_h\| = \|x_h\| = r_h$ and $\|x'_t\| = \|x_t\| = r_t$. The error set of r_h is the rim containing all the objects that are farther away from the center than r_t but no more than r_h:

$$\mathrm{errset}_U(h,t) = \{x \in U : r_t < \|x\| \leq r_h\} \tag{8.6}$$

In order to guarantee that h's error is not larger than some ε, we look for a *worst* hypothesis that just satisfies the error restriction. This worst case is

$$\alpha = \max\{r : \mu(F_r) \leq \varepsilon\} \tag{8.7}$$

where F_r is the disk wit radius r.[3] In other words, α is the largest possible radius for a disk such that its (entire) area has an error of at most α. This adds an entry to our diagram above, which, after slight transforms, looks as follows:



Now it's time to think about errors: Due to our definition of α, it holds that

$$\mu(\{x \in U : r_t < \|x\| \leq \alpha\}) \leq \varepsilon. \tag{8.8}$$

[3] N.B. The attentative reader will have noticed that we do not make a proper distinction between a measure on the set of radiuses describing disks and a measure of the areas of these disks. However, this does not give rise to any problem in the running argument.

Since $r_h \leq \alpha$, it is also true that

$$\mu(\{x \in U : r_t < \|x\| \leq r_h\}) \leq \varepsilon. \tag{8.9}$$

It is also true, that the "outer space" completes to probability 1:

$$\mu(\{x \in U : 0 < \|x\| \leq \alpha\}) + \mu(\{x \in U : \alpha < \|x\| \leq 1\}) = 1. \tag{8.10}$$

For the true error of r_h we conclude:

$$\begin{aligned} \mathrm{error}^{\mu}_{U}(r_h, t) &= \mu(\{x \in U : r_t < \|x\| \leq r_h\}) \\ &\leq \mu(\{x \in U : r_t < \|x\| \leq \alpha\}) \\ &\leq \varepsilon \end{aligned} \tag{8.11}$$

Great! Obviously it is possible to learn a hypothesis that is at least ε–good. But what is the probability of being able to find it? In order to find a hypothesis we need to *see* an example that is better than α! The probability of *not seeing* such an example is the probability that none of the m examples is in the rim is at most $(1-\varepsilon)^m$. Therefore, with probability of at least $1-(1-\varepsilon)^m$ there *is* at least one such required example in the sample:

$$\mu^m(\{\mathbf{s} :\in S_{\mu}(m,t) : \mathrm{error}^{\mu}_{h}(r_{,)} r_t U \leq \varepsilon\}) \geq 1-(1-\varepsilon)^m \tag{8.12}$$

This means that the probability of picking a "good" example depends only on ε and, of course, on m. So what does it mean for the value δ of confidence? If we substitute $(1-\varepsilon)^m$ from equation (8.12) for δ in equation (8.1) then we find that

$$\delta > (1-\varepsilon)^m$$

So we need to find a formula for computing a value for $m_0 \leq m$, which is the minimum value for which the above inequality becomes true. The choice $m_0 = \lceil \frac{1}{\varepsilon} \ln \frac{1}{\delta} \rceil$ satisfies all our requirements—and that means, that the problem we wanted to learn is probably approximately correct learnabe: We have described an algorithm (see figure 8.2) and we have shown that by a mere definition of a maximally tolerated error of ε and a confidence δ we can compute the number of required examples—absolutley independently from any knowledge about μ! This is, in fact, a really impressive conclusion. If you still do not share a certain excitement or if you are asking yourself for the benefit of all these calculation, then maybe the following example will eradicate all doubts:

Example 8.4 In the example above and with $\delta = \varepsilon = 0.1$, we need 23 examples to learn an arbitrary disk with an arbitrary underlying distribution. ●

Exercise 8.1 (♦♦♦)

- ♦ Prove the statement from the preceding example!
- ♦ What is the value of δ, if we restrict m_0 to 18 and keep $\varepsilon = 0.1$?

- How many examples does it take if we are satisfied with a hypothesis that is better than random (that means that ε is just slightly larger than $\frac{1}{2}$) in 99% of all runs?

For the latter two questions you might want to write a little program that calculates a "table" of the dependencies between ε, δ, and m_0!

What we have shown so far is that we could find an estimate of the complexity for learning two–dimensional continous problems. The reason for this was our running example and the idea behind finding a classifier that works sufficiently well with a minimal number of features, relations, or predicates—short: with a minimum number of dimensions. The funny thing is that we reduced the two-dimensional problem of learning a disk to the one-dimensional problem of learning an *interval* on the real numbers; or, to be more precise, a ray. A more illustrative version of the proof presented here (including the details on how to find a term describing m_0) can be found in a great "manual of computational theory", [Anthony and Biggs, 1997].

8.1.2 Learnability and learning algorithms

For our running example domain, we showed that we can learn *any* concept that can be visualised by a disc in the plane where the plane itself is defined by a pair of features from $\mathbf{F}$. Actually, we worked on a much harder problem than in all examples before because until then, our world was *discrete*: Objects had a certain colour like black, dark, or white, and they had a distinct shape like squares, circles, or triangles. Even the size of an object was discrete: they were small or large. In the universe $\mathfrak{U}$ we created in this chapter, objects can have *any* shade of grey from white to black and *any* size from a minimum size 0 to a maximum sice 1. It means that $\mathfrak{U}$ is infinite; it is even uncountably infinite. Yet we can find an iterative algorithm which comes to a halt after a finite number of steps and output a hypothesis satisfying a preset quality bias. This is, even modestly speaking, a wonderful result—for now.

Since the representation spaces that we were concerned with so far are discrete, *most* learning problems we examined so far are PAC! This is simply due to the fact that for every *finite* hypothesis or representation space, there is a very simple algorithm which is PAC. It is so simple that it does not deserve a figure and a pseudo-code formulation, but it is so important that it deserves a knowledge box:

Finite Problems are PAC Learnable
Every learning problem with a finite representation space is PAC learnable:

If the set U is finite, then just enumerate all of its elements, and sort them into two sets; one set representing the target concept and one set representing its complement. Then, for $\delta = 1$ and $\varepsilon = 0$, one can choose $m_0 = |U|$. This is neither elegant nor efficient—but it works.

Every learning problem with a finite hypothesis space and a representable hypothesis is PAC learnable:

In this case, just enumerate all hypotheses, and iterate them until we find one satisfies our needs! If we want to learn a hypothesis with zero error, it requires our representation to be fine enough to describe the target concept. If it is not, we can at least find a best hypothesis after running through all possible hypotheses.

But usually, U is *infinite*. This is why we restricted the domain for learning to a finite subset $s \subseteq U$ at the very beginning of our journey (see figure 3.1). In real life, any information system which we take as a base for our learning problem is "finite": First, there is only a finite amount of data s. Second, there is only a finite set of features $\mathbf{F}$; and third, for each feature $f \in \mathbf{F}$, $\text{cod}(f) = V_f$ is finite, too. The domain it represents, however usually is *not* finite. Because of the inductive assumption (see section 3.3) we can live with a finite subset of the domain. The restriction to only a finite set of features corresponds to a restriction of our language of hypotheses. And finally, even continuous functions usually are quantised and are, for a defined interval, finite.

Clustering and Learnability

If you take a look at figure 8.1 again, the similarity of all approaches discussed so far becomes obvious: Learning a k-NN classifier means to identify clusters. Note that the description of the problem domain we gave in section 4.2 we already lifted our representation from a discrete information system to $\mathbb{R}^n$ with an Euclidean distance measure in equation (4.1). Note also that the k-NN hypothesis itself as defined equation (4.2 is just the same as in equation (8.2)!

Decision Trees. In chapter 5 we discovered an appoach to an information gain based heuristics search procedure for hierarchical clustering. The algorithms presented therein are based on discrete features, too. However, there exist extensions that are capable of working with real-valued features. They simply perform an internal quantisation by identifying relevant intervals. There are many algorithms for quantisation—ranging from a simple equal frequency binning to, again, entropy based methods. But basically, quantisation again is *clustering*, too.

Rough Sets and Learnability

In rough set theory the notion of upper and lower boundaries provide us with a beautiful vacabulary to circumscribe what in PAC-learning we have called the ε-region. The definition of equivalence relations and all concepts required in rough set theory are *not* limited to finite sets even though all examples were defined on finite domains. The equivalence class of natural numbers is a proper set of the rational numbers which in turn is a proper subset of real numbers—and they all are at least denumerable infinite.

Exercise 8.2 (♦♦) Define a relation $Q \subseteq \mathbb{R} \times \mathbb{R}$ such that $[\![Q]\!]\mathbb{Q} = \mathbb{N}$ and $\langle\!|Q|\!\rangle\mathbb{Q} = \mathbb{R}$!

We also noted that $(\!|R|\!)c$ is the region of uncertainty around c and it is a straightforward idea to define

$$\mathrm{errset}_U(\mathbf{R}, t) := \{x \in (\!|\mathbf{R}|\!)c : x \notin \{x \in U : t(x) = \mathbf{1}\}\}$$

But what about the distance of an object from the actual boundary of the concept? In the rough set approach there are only nominal class values, no ordinals. And if there is no (given) order, there is no (canonical) distance. But there is a *natural order*: Keeping in mind one of the baselines of this book, discernability is what makes knowledge. Accordingly, two objects are more similar the more knowledge it requires to distinguish between them. Therefore, *the minimal number of relations* $|\mathbf{P}|$ we need to add to $\mathbf{R}$ such that

$$x \in c \Longrightarrow x \in [\![\mathbf{P} \cup \mathbf{R}]\!]c$$

A measure based on this idea is very well suited for any "bottom-up" rough set classifier learning algorithm where (starting with an empty set or a known core) we add relations to specialise the hypothesis until it is "good" enough (which, in turn, can be estimated using any of the error measures described in section 3.4).

Learnability and ILP

One of the most interesting topics in learning logic programs is the induction of recursive predicates. From the point of view of a logician recursive predicates are self-resolving clauses. The unification involved in this process results in the construction of possibly infinite term structures. This problem could only be solved by examining Herbrand models or by strong syntactical biases. The introduction of these biases, may it be the restriction to ground facts, to programs Π with only unit clauses in them, ij-determinacy or any similar method helps drawing a beautiful line between learning problems: Only few ILP-problems *are* PAC-learnable; and they all require very strict assumptions in the form of the biases discussed in the chapter on ILP. In general, PAC-learnability is known to be a very pessimistic concept: If there is a simple learning problem, it is PAC in most cases; if it is interesting, it is not PAC. Whenever we can formulate a hypothesis for which we cannot determine whether $h(x) = \mathbf{1}$ or $h(x) = \mathbf{0}$ in polynomial time for some $x \in U$, then *any* problem in representation space is not PAC-learnable. As an example, consider the following little logic program that we assume some learning algorithm has generated to describe our hypothesis:

```
h(0,X,Y):- Y is X+1.
h(X,0,Y):- Z is X-1,h(Z,1,Y).
h(X,Y,Z):-
  U is X-1,
  V is Y-1,
  h(X,V,W),
  h(U,W,Z).
```

Exercise 8.3 (⧫) Check the validity of the hypothesis for the following triplets: $\langle 1,1,3\rangle$, $\langle 2,2,9\rangle$, $\langle 3,3,81\rangle$, and $\langle 4,4,6561\rangle$!

The argument of Non-PACness is based on an important article that we shall rediscover in section 8.3. Let us summarise:

> **A problem is learnable...**
> If there is an algorithm that can effectively (and efficiently) deliver a hypothesis solving the problem with a certain confidence and a certain maximum error.

But how do we learn in every day life? If one problem is too big, we try to *divide and conquer*. That is, we try to brake down the big problem into many small ones each of which are simple to learn. Another method is to repeatedly learn the problem where in each iteration we focus on what we don't know yet.

8.2 Decomposing the Learning Problem

Imagine you want to learn a large set of facts for an exam. One method is to write down every single fact on a file card and the flip through the pile over and over again until you know them all. Usually, your collection will not cover all the topics; but it is a representative sample of all questions that might occur in an exam.

This gives rise to several problems: Sometimes, you have too few examples cards to infer a model that is good enough (in terms of passing the exam). There are three different reasons. If you don't have enough cards, your hypothesis might be too general, it could be overfit or it is simply bad because you picked the wrong set of cards. Sometimes, you have too many cards. This results in the problems of computational and representational complexity. With too many aspects or too many examples you are not able to find a hypothesis until examination day which sufficiently well describes what you need to know in order to pass. And finally, it could be that the method you apply to learn is not appropriate for the problem which means that *any* hypothesis is rather weak. But wouldn't it be a good idea to use *many* weak learners resulting in *many* weak hypotheses which (for example, by voting) then solve a new probem as a team? Yes! It is a good idea.

It is easier to learn a less complex problem than a very complicated one and it is easier to learn from a smaller set of facts and within a smaller set of hypotheses. We examined the process of sampling in detail in section 3.3. But it was not before definition 3.23 that we began to think about a probably positive side–effect of the non-determinacy of $S_T(,.)$he first idea that comes to mind is called *sub-sampling*. It means to learn on subsets and then combine the classifiers for all the subsets to receive a classifier on the entire set. The problem is to find a method of combining all the classifers $h_i = \texttt{Alg}(\mathbf{s}_i)$ with $\mathbf{s}_i \subset \mathbf{s}$ to one single hypothesis h. There exist many methods for choosing $\mathbf{s}_i \subset \mathbf{s}$, and, as you can imagine, they all have different impacts on the final result. Either way, we use the term "subsampling" for any kind of systematic *downsampling* as, e.g. k–th selection sampling or repeated draws with replacement. The second idea is *re-sampling*. It means that we learn a set of classifiers from *different samples* $h_i = \texttt{Alg}(S_\mu(m, t))$. It means a repeated nondeterministic execution of $S_\mu(m, t)$, also resulting in a set of samples $\mathbf{s}_i$.[4]

Exercise 8.4 (◊) Give a short example for each of the three methods mentioned above and explain the differences.

8.2.1 Bagging

Bagging means to find a set of experts, form a single committee and then present th problem to the committee rather than an individual expert.

Bagging
Bagging is a meta-learning approach, where the same learning algorithm is presented a number of different samples each of which result in an individual predictor. The hypothesis is defined by an aggregation of all predictors.
The idea behind bagging is to make learning simpler on subsets of the problem and to yield a better result by the "knowledge of majority".

If we have a sample $\mathbf{s}$ from which it is hard to learn a good hypothesis, then we generate a set of subsamples $\mathbf{s}_i$, learn hypotheses h_i (representing experts) from then and then combine all hypotheses to one (being the committee) that shall describe the whole sample. "Bagging" is an acronym formed from *bootstrapping* which refers to an initial process of forming a set of samples that is used for the subsequent process. After learning comes *aggregating* which means that we have to combine all hypotheses.

Aggregation of hypotheses, h_{agg}

Definition 8.3 — Aggregation of hypotheses, h_{agg}.
Let there be set of k samples $\mathbf{s}_i$, $i \in \mathbf{k}$. If $\mathrm{cod}(t)$ is numerical, the *aggregated* hypothesis is the *average* over all $h_i = \mathbf{A}(\mathbf{s}_i)$.

$$h_{agg}(x) = \frac{1}{k}\sum_{i=1}^{k} h_i(x) = \frac{1}{k}\sum_{i=1}^{k} \mathbf{A}(\mathbf{s}_i)(x) \tag{8.13}$$

[4] We have used the terms "sub"–sampling and "re"–sampling very informally. Sampling itself is a broad research field with strong links to the theory of information. In this diction we would state that downsampling comes a along with a loss of bandwidth, and thus, information. One idea is to deliberately lose only those parts that cause confusion in learning and keep those from which we gain knowledge: In image recognition, edges are made explicit by increasing contrast—which means to throw away all nuances.

If t is nominal, i.e. $\mathrm{cod}(t) = \{1, 2, \ldots, j\}$, the *aggregated* hypothesis is

$$h_{agg}(x) \quad = \quad \mathrm{mcv}_{\pi(2)}(\{\langle i, h_i(x)\rangle : i \in \mathbf{n}) \tag{8.14}$$

i.e. delivers the answer j on which most h_i agree.[5] ●

With a given set of k (different) samples $\mathbf{s}_i$ the case is simple—basically, we are done with bagging. The problem is that usually we are supplied only *once* with a *fix* sample $\mathbf{s} = S_\mu(m, t)$. In order to apply bagging on one single sample, we have to find a method by which we can *generate* a set $\{sample_i : i \in \mathbf{k}\}$. A simple method is to independently choose *subsets* of $\mathbf{s}$.
As already noted, ϕ is unknown. By our inductive hypothesis, ϕ is preserved by $\mathbf{s}$. Therefore it should be preserved under the average of repeatedly *randomly* drawing subsets, too.

Bootstrap approximation

Definition 8.4 — Bootstrap approximation.
Given some sample $\mathbf{s}$ of length m, we define k data sets $\mathbf{s}_i, i \in \mathbf{k}$ each of length $m_i \leq m$. Every sample $\mathbf{s}_i$ contains m_i examples all of which are drawn *at random* and *with replacement.* Note that if an example is drawn several times for $\mathbf{s}_i$, it appears only once in $\mathbf{s}_i$. ●

As a result of this procedure we now have a set of k *different* subsamples $\mathbf{s}_i$ which we hope to form k different hypotheses h_i as well and from which we can define an aggregated hypothesis. This method makes use of two very important properties inu our learning scenario:

1. Since $\mathbf{s}_i$ are drawn at *random* (i.e. presupposing a uniform distribution on $\mathbf{s}$) and since $\mathbf{s}$ is drawn with respect to μ, the *average* probability of an object x to occur in a sample $\mathbf{s}_i$ is $\phi(setx)$ (see definition 3.8 and theorem 3.1).

2. All $\mathbf{s}_i$ are nearly always pairwise *different* (this is an immeadiate conjecture of the previous observation).
   Throughout the book we made another implicit assumption: For different samples, the same algorithm will produce different hypothesis as well (see section 3.6 and theorem 3.2).

This gives rise to a very important question: What does it take for a learning algorithm to deliver different hypotheses for different examples? And, even more important: Is the difference between different examples proportinal to the difference between the resulting hypotheses?

Example 8.5 Imagine we observe sequences of objects from our running example—and the target function is a predictor for the colour value of any

[5] The most common value function mcv was defined in equation (4.3). We need to collect all results together with the number i of the predictor such that multiple occurence in the set is preserved. The function π_2 is simply the projection on the second element of the tuples.

object in U. Again, a small representation shift on our underlying information system helps a lot:

colour	□	░	▒	▓	■
$\rho : U \to \mathit{Colour}$	*white*	*light*	*grey*	*dark*	*black*
$\rho' : U \to [0,1]$	0	$\frac{1}{4}$	$\frac{1}{2}$	$\frac{3}{4}$	1

Let $t : U^n \to [0,1]$ with

$$t(\langle x_0, x_1, \ldots, x_{n-1}\rangle) = \frac{1}{n} \sum_{i \in \mathbf{n}} \mathit{colour}(x)$$

where n is an odd number. So t is simple the arithmetic mean of grey-scale values. Note that this value is any real number from the interval $[0,1]$, whereas all the values x_i are from the set $\left\{0, \frac{1}{4}, \frac{1}{2}, \frac{3}{4}, 1\right\}$. Let us now imagine that our hypotheses space contains only four hypotheses:

$$\begin{aligned} h(x)_0 &= 0 \\ h(x)_1 &= \mathrm{mcv}_{\mathit{colour}}(\{\langle i, \mathit{colour}(x_i)\rangle : i \in \mathbf{n}\}) \\ h(x)_2 &= \mu_{1/2}(\{\langle i, \mathit{colour}(x_i)\rangle : i \in \mathbf{n}\}) \\ h(x)_3 &= 1 \end{aligned}$$

which means that our learning algorithm can choose from four predictors only: The first one always predicts white, the second one the most common value, the third one the median, and the forth one always predicts black. It is clear by intuition that for a *small* change in the supplied training sample the resulting hypotheses will not differ *too much*. Only with a very small sample size or large differences in the distribution of target values in between two samples $\mathbf{s}$ and $\mathbf{s}'$ it could happen that $\texttt{Alg}(\mathbf{s}) = h_0$ and $\texttt{Alg}(\mathbf{s}') = h_3$. ●

Exercise 8.5 (♢ — ♦) This exercise requires the reader to be familier with statistics (but then it should be easy): Can you give an estimate for the expected (maximum) error of hypotheses h_0, h_1, h_2, and h_3? — For the interested reader we suggest to read up about Chebychev's theorem in measure theory.

Example 8.6 Multi layer perceptrons are universal function approximators which for a small change in their input produce only a small change in the output. However, only a small change in the presented sample can result in drastically different hypotheses (sometimes even by only a rearrangement of example sequences); see [Breiman, 1996]. ●

The property of "local robustness" against changes in the presented samples is known as *stability*:

Stability of Learning Algorithms

Definition 8.5 — Stability of Learning Algorithms.
An algorithm $\texttt{Alg}$ is called *stable*, if for a "small" change from $\mathbf{s}$ to $\mathbf{s}'$,

$$\texttt{Alg}(\mathbf{s}) = h \approx h' = \texttt{Alg}(\mathbf{s}') \tag{8.15}$$

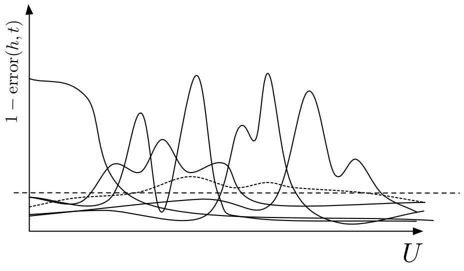


Figure 8.3: Bagging for free lunch

`Alg` is called *unstable*, if a small change in $\mathbf{s}$ can result in a large change of the predictive behaviour of h. ●

Since t remains constant, two hypotheses h and h' delivered by an *unstable* algorithm have different errors on one common test set. In allusion to the No-Free-Lunch–Theorem the situation can be illustrated as in figure 8.3. An unstable base learner will produce many hypotheses that specialise on different parts of the universe. And in sum, there are "specialists" for nearly every subset of the domain which in a process of aggregation together with a majority of all other hypotheses will more likely come to a correct prediction. We now circumstantiate the intuitive understanding of bagging with a (slightly simplified) version of the argument in [Breiman, 1994].
Assume there is a sample $\mathbf{s} = S_\mu(m,t)$ (drawn with respect to and representing μ). Let $\text{cod}(t)$ be numerical. By definition 3.16, we choose the quadratic error measure for numerical values as in equation (3.26). Then, the aggregated hypothesis h_{agg} is an *average* predictor for μ learned on a sample with distribution ν:

$$h_{agg}(x) = E_\nu(h(x)) \tag{8.16}$$

where E_ν denotes the expected (target) value for x based on the distribution on $\mathbf{s}$. Bagging increases the quality of learning, if it reduces the expected error of the hypothesis. Therefore, let us take a look at errors:

$$\text{error}_s(h,t) \quad = \quad E_\nu(E_\mu(\text{dist}(t(x), h(x))))$$

and the error of h_{agg} on is

$$\text{error}_{agg}(h,_{)}\, ts \quad = \quad E_\mu(\text{dist}(t(x), E_\nu(h(x))).$$

With some knowledge in statistics, this leads to the following inequality:

$$\text{error}_s(h_{agg}, t) \leq \text{error}_s(h,t) \tag{8.17}$$

The less equal both sides are, the lower is h_{agg}'s mean squared error in relation to that of h. And this explains why bagging works for unstable learners only: Let $\mathbf{s}_i$, $i \in \mathbf{k}$, be a sequence of similar, but different samples.

- If **A** is stable, all h_i will be similar to h_{agg} and the both sides of the inequality are nearly *equal*.
- If **A** is unstable, we have an increasing chance of *different* h_i. The more of them differ to increasing extent, the more increases the right side over the left.

The more μ differs from the distribution ν on the sample, the more likely $\mathbf{s}_i$ will differ from $\mathbf{s}$. Accordingly, the improvement by bagging depends on the stability/instability turnover point of `Alg`—and this point depends on μ, the size m of $\mathbf{s}$, the difference between μ and ν and, finally, `Alg` itself. If, however, some $h = \texttt{Alg}(\mathbf{s})$ is nearly optimal, then no bagging will improve the results.

Bagging
Bagging is a very simple method that allows to improve learning results by aggregating several partial hypotheses. It works only algorithms that are unstable.

8.3 Improving by Focusing on Errors

Consider again the scenario of learning from a set of file cards. It is a good idea to learn subsets as we have seen in the previous section. But nobody does. What we do instead is somekind of *boosting*:

Boosting
Boosting means to iterate the learning process where in each iteration we put more emphasis on the objects that we were not able to describe properly in the previous iteration.

The idea behind ensemble learning—may it be bagging or boosting—is that in general it is much more complicated to learn in one shot rather than by divide-and-conquer. When trying to learn several subproblems in parallel, there is the risk of putting effort in learning the same thing twice because one learner does not know what the other does. This is an argument for an iterated learning approach. The second argument is that one usually focuses on what still has to be done rather than what we've already accomplished. Such a general principle can be applied to *all* learning problems and *all* learning algorithms (called *base learner*) in a boosting approach. Let us reformulate the knowledge box from above with a small touch of formal terminology:

Boosting algorithm

Definition 8.6 — Boosting algorithm.
A *boosting algorithm* repeatedly calls a (weak) *base learner* that produces a locally accurate but globally bad hypothesis with each time a *different* sample. The result is a sequence of hypotheses where all the errors we made in a step i, have an increased probability to be picked as a learning example in step $i + 1$:

If $x \in \mathrm{errset}_s(h_i, t)$, then

$$\phi \left\{ \mathbf{s}_i : \begin{array}{c} \langle x, t(x) \rangle \in \mathbf{s}_i \wedge \\ \mathbf{s}_i :\in S(m,t)^i \end{array} \right\} \leq \phi \left\{ \mathbf{s}_{i+1} : \begin{array}{c} \langle x, t(x) \rangle \in \mathbf{s}_{i+1} \wedge \\ \mathbf{s}_{i+1} :\in S(m,t)^{i+1} \end{array} \right\} \tag{8.18}$$

where $S(m,t)^k$ denotes the set of all samples that are drawn by $S_\mu(m,t)$ in the k-th iteration. This means, that examples which we cannot classify correctly will be dealt with in the next step with increasing probability. ●

Note that ϕ is a distribution over a set of samples as elementary events whereas μ is a distribution over U. As we know, it is μ which determines the behaviour of $S_\mu(m,t)$; and the expression in equation 8.18 actually states, that μ has to change from step i to $i+1$ because it is the "importance" of some x that shall influence the behaviour of the learning algorithm or the choice of samples submitted to this algorithm.

But even if we would know *how* to change μ, we couldn't—for we don't know what to change. At this point the idea of subsampling comes in quite handy: We assume to have *one* single fixed sample $\mathbf{s} = S_\mu(m,t)$. Then, we work with a distribution ϕ on this sample rather than on U.

All algorithms we have discussed so far did not make use of any additional information such as a distribution of the sample. However, there are many such algorithms; just imagine an explicit weight value that, for example, locally distorts a distance measure in a k-means clustering algorithm. Then, centroids would not just wander to the plain center of gravity of in scenario as shown in figure 4.2, but they would be drawn more into the direction of "heavy weights". If the base learner does *not* make use of the probability distribution over $\mathbf{s}$, then we can simply simulate this by sub-sampling where $\mathbf{s}_i$ are drawn from $\mathbf{s}$ with respect to the changing distribution. Still we have to decide what the initial probability distribution ϕ_0 shall look like. The simplest way is to assume an independent identical distribution.[6] Based on this choice, we can now examine a first version of a boosting algorithm as introduced by [Schapire, 1990]. First, we need to redefine $\mathrm{cod}(t) := \{-1, 1\}$ for arithmetic reasons. Then, let us assume there is a base learning algorithm `Alg` which is better than random. Such a learning algorithm is called a *weak learner*. Finally, we define:

$$\forall e \in \mathbf{s} : \phi_0(\{e\}) := \frac{1}{m} \tag{8.19}$$

Then, we iterate `Alg` k times and after the i–th iteration define ϕ_{i+1} by increasing $\phi_i(\{\langle x, t(x)\rangle\})$ if $x \in \mathrm{errset}_s(h_i, t)$ and then normalise using a factor $\frac{1}{\nu_i}$ to ensure that ϕ_{i+1} is a proper probability distribution again. The new probability value of such an object is defined in terms of its old probability, an adaptation value, and the difference of h_i and t measured in terms of their product (remember that we defined $\mathrm{cod}(t) = \{-1, 1\}$). Then, `Alg` is called again—either on the same sample with the new distribution or on a new sample that is drawn with

[6] Even though this *appears* to be an unbiased choice it is not: There is no qualitative difference between the assumption of i.i.d. or a binomial distribution or any other.

```
$i := 0$; $\mathbf{s} := S_\mu(m,t)$; $\phi_0 :=$i.i.d.
WHILE $(i < k)$ DO
{    *choose* $\alpha_i \in \mathbb{R}$;
     $h_i := \mathtt{Alg}(\phi_i, \mathbf{s})$;
     $\forall \langle x, t(x)\rangle \in \mathbf{s} : \phi_{i+1}(\{x\}) := \frac{1}{\nu_i}\phi_i(\{x\})^{-\alpha_i h_i(x) t(x)}$;
     $i := i + 1$;
}
return$\big(h_{agg} := \operatorname{sgn} \sum_{i \in \mathbf{k}} \alpha_i h_i(x)\big)$
```

Figure 8.4: AdaBoost.B0, [?]

respect to the new distribution. The process ends after k rounds and the result is the average of all hypotheses: -1, if the weighted sum of all hypotheses's answers is negative and 1 otherwise. The pseudo-code of this algorithm (called *AdaBoost.B0*) is shown in figure **??**.

Example 8.7 Imagine the following, artificial, case which is just to illustrate the boosting mechanism (we neglect α_i and simply produce a new i.i.d. ϕ_{i+1} on the previously wrong classified examples): $s = \{\bullet, \triangle, \blacksquare, \square\}$ and $t(x) = \mathbf{1} :\longleftrightarrow$ *colour*$(x) =$ *black*. Let $\phi_0(\{x\}) = \frac{1}{m} = \frac{1}{4}$. In this case, let us assume that `Alg` delivers a hypothesis h_0 when presented $\mathbf{s}_0 = \{\langle x, t(x)\rangle : x \in s\}$:

s	$\bullet$	$\triangle$	$\blacksquare$	$\square$
t	1	-1	1	-1
ϕ_0	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$
h_0	1	1	-1	1

The errorset is $\text{errset}_s(h_0, t) = \{\triangle, \blacksquare, \square\}$. We have h_0 which is correct on $\bullet$ and now repeat learning on a new sample:

s	$\bullet$	$\triangle$	$\blacksquare$	$\square$
t	1	-1	1	-1
ϕ_1	0	$\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{3}$
h_1	1	-1	1	1

Such that the error set of h_1 is $\{\square\}$. Therefore, in the second step,

s	$\bullet$	$\triangle$	$\blacksquare$	$\square$
t	1	-1	1	-1
ϕ_2	0	0	0	1
h_2	1	-1	-1	1

But if `Alg` still is not able to learn □, then it does not change anything anymore. Let us aggregate now:

s	●	△	■	□
t	1	-1	1	-1
$\sum h_1$	$1+1+1$	$1-1-1$	$-1+1+1$	$1+1+1$
	3	-1	1	3
h_{agg}	1	-1	1	1

This is of course an oversimplified example, because it assumes that in the $i+1$-st run, `Alg` knows that it simply shall copy the answer of h_i for all objects that have a probability of 0. It does not only need knowledge about other but also about ϕ—which is quite much. ●

At the very beginning, we stated that there is no real difference between learning a binary problem or a nominal problem. The argument was that we can shift a representation from a nominal function $t_N : U \to \mathbf{k}$ to k binary functions $t_i : U \to \mathbf{2}$ with $t_i(x) = \mathbf{1} :\Longleftrightarrow t(x) = i$ thereby decomposing on k–ary learning problem to k binary ones. There are two boosting algorithms which slightly differ in how they cope with nominal target value sets. The first one suffices to show the most imortant property of boosting:

The Advantage of Boosting
Let there be a weak base learning algorithm which in k turns produces an error of $\text{error}_s^{\phi_i}(h_i, t) \leq \frac{1}{2}$ on the *training sample*. Then there exists an upper bound for the error of the aggregated hypothesis which drops exponentially in the number of iterations.

The major improvement in AdaBoost.M1 is that it has a bail-out-option if the error of the base learner *increases* above $\frac{1}{2}$ and that the entire sample error of the hypothesis is taken into account. The error of a hypothesis h_i is the probability weighed sum of all false predictions:

$$err_i := \text{error}_s^{\phi_i}(h_i, t) \tag{8.20}$$

as defined in equation (3.30) and a binary distance measure as in equation (3.27). If this error is larger than $\frac{1}{2}$, the procedure terminates immediately (otherwise, it would be an ever increasing error). Then, ϕ_i is changed by a factor $\frac{err_i}{1-err_i}$ on every false prediction of h_i. The rest of the algorithm is just about the same as AdaBoost.B0, except, of course, for the final hypothesis. It is simply the response for which the cumulated error is minimal. The pseudo-code is shown in figure 8.5. To see how ϕ_i changes in AdaBoost.M1 over time, see the following example:

Example 8.8 This time, we have a set of twelve objects and wrong predictions for every odd integer in the i-th iteration.

```
$i := 0$; $\mathbf{s} := S_\mu(m,t)$; $\phi_0$ :=i.i.d.
WHILE $(i < k)$ DO
{
    $h_i :=$ Alg$(\phi_i, \mathbf{s})$;
    $err_i := \phi_i(\text{errset}_s(h_t, t))$;
    IF $err_i > \frac{1}{2}$ THEN $k := i - 1$ ; break;
    $\alpha_i := err_i/(1 - err_i)$
    FORALL $x \in s$ DO
    {   IF $(h_i(x) \neq t(x))$ THEN
            $\phi_{i+1}(\{x\}) := \frac{\alpha_i}{\nu_i}\phi_i(\{x\})$;
        ELSE
            $\phi_{i+1}(\{x\}) := \frac{1}{\nu_i}\phi_i(\{x\})$;
        ENDIF
    } 15}
return $\left(h_{agg}(x) = \arg\max_c \left\{\left(\sum_{i\in\mathbf{k}} \ln \frac{1}{\alpha_i}\right) : h_i(x) = c\right\}\right)$
```

Note that just as in AdaBoost.B0, the call of Alg with information ϕ_i can be simulated by subsampling (h_i := Alg$(S_{\phi_i}(m', t))$ on the base set s of $\mathbf{s}$).

Figure 8.5: AdaBoost.M1, [?]

$x \in s$	$h_i(x) = t(x)$	ϕ_i	$\nu_i \cdot \phi_i$	$h_{i+1}(x) = t(x)$	ϕ_{i+1}
1	0	.05	.05	.088	
2	1	.1	.048	.085	
3	0	.025	.025	.044	
4	1	.2	.096	.169	
5	0	.025	.025	.044	
6	1	.15	.072	.127	
7	0	.075	.075	.132	
8	1	.025	.012	.021	
9	0	.025	.025	.044	
10	1	.15	.072	.127	
11	0	.125	.125	.221	
12	1	.025	.012	.021	

The error of h_i is $err_i = .05+.025+.025+.075+.025+.125 = .325$. Accordingly, $\alpha_i \approx .481$. The sum in the third column is .614 such that we have to normalise by $\nu_t = \frac{1}{.614} \approx 1.629$. Multiplication with ϕ_i then gives ϕ_{i+1} as in the last column. ●

We end this rather theoretical excursion to ensemble learning by the boosting theorem: Suppose, that for $err_i \leq \frac{1}{2}$ for all $i \in \mathbf{k}$ and let $\delta_i = \frac{1}{2} - err_i$. Then,

$$\begin{aligned}
\text{error}_s(h_{agg}, t) &= \frac{|\{i : h_{agg}(x_i) \neq t(x_i)\}|}{m} \\
&\leq \prod_{i \in \mathbf{k}} \sqrt{1 - 4\delta_i^2} \\
&\leq e^{\left(-2 \sum_{i \in \mathbf{k}} \delta_i^2\right)}
\end{aligned}$$

It means, that in this case the error of the aggregated hypothesis can be reduced with increasing iterations; actually, the error converges exponentionally in the number of iterations against zero. The downside of this really impressing result is that the error we are talking about is the error on the training sample; i.e. s is the base set of $\mathbf{s}_{\text{train}}$. Therefore, the entire boosting approach depends on the adequacy of the chosen sample and a maximum accuracy on this sample will nearly always lead to overfitting and much less impressive results on a test set. But, supposing the saple is chosen wisely, it holds that whenever the error on the training sample decreases, so it does on any other sample (hopefully), [**?**]. In general, it is a good idea to think of boosting as:

- repreated sub- or resampling on a set of observations with
- a given (unknown) distribution *measure* on the entities being observed
- and a *deliberate adaptive sampling bias* by an error-dependent change of the probability distribution ϕ that is taken for sampling or as additional knowledge for `Alg`.

In many cases, boosting helps a lot (for both binary and nominal problems) but it may require some changes of the base learner. A simple case is where we simulate the `Alg`$(\phi, \mathbf{s})$ by `Alg`$(S_\phi(m, t))$. An extension of AdaBoost.M1, called

AdaBoost.M2, explicitly requires `Alg` to work with ϕ_i and to deliver a value in $[0,1]$ as a measure of plausibility) back to the boosting algorithm as it is required for a more sophisticated change in ϕ_{i+1}.

8.4 A Relational View on Ensemble Learning

So why bagging, boosting and all those measures, probabilities and expectation values? Knowledge about a square means to be able to say that $\square$ is a square and $\triangle$ is not. And a $\lozenge$ is not a $\square$ as a $\circ$ is not a $\blacktriangle$. There is no probably, there is no approximately, and there is no lucky sampling in relational knowledge discovery. Right? — Wrong!

8.4.1 Dividing the sample set

Bagging means to learn classifiers that specialise on subsets of the sample. Why shan't we try and learn rough set classifiers for subsets of the universe? This time, let us take a closer look at rough set classifiers

Relational Bagging. In a relational setting, bagging means that for a set of k subsamples $\mathbf{s}_i$ of size $m' \leq m$, `Alg` computes k hypotheses $\mathbf{P}_i$ approximating $c_i := s_i \cap c$ with $s_i = \{x : \langle x, t(x)\rangle \in \mathbf{s}_i\}$. The most important thing to know before asking whether a certain learning algorithm can benefit from bagging is whether the algorithm is stable or not. Finding a rough set classifier is an *unstable* procedure.

Exercise 8.6 (♦) Give an example!

Once we found k hypotheses, we need to aggregate them. Assume there is a function *agg*, we want that

$$\mathbf{P}_{agg} := agg(\{\mathbf{P}_i : i \in \mathbf{k}\}) \approx t$$

Since we are dealing with rough classifiers, we have a weak and a strong interpretation of target class predictions:

$t(x)$	**0**	**1**
rough	$x \notin [\![\mathbf{P}_{agg}]\!]c$	$x \in \langle\!|\mathbf{P}_{agg}|\!\rangle c$
strict	$x \notin \langle\!|\mathbf{P}_{agg}|\!\rangle c$	$x \in [\![\mathbf{P}_{agg}]\!]c$

It means, that if $x \in (\!|\mathbf{P}_{agg}|\!)$, the rough classifier could answer "$x \in c$", because if x is in the boundary region, it is in the upper approximation, too. Also, the classifier could answer "$x \notin c$", because the boundary region has an empty intersection with the lower approximation. For the same reason, the strict predictor cannot at all deliver an answer because the boundary region is exactly the region whose elements are neither outside the upper approximation nor inside the lower. In equation (6.9) we defined the rough characterisitc function with three values. Using this definition, k sets of equivalence relations model k

characteristi function on k sets $c_i \subseteq c$. Without a further analysis of the c_i we could build an aggregated classifier function by voting again:

$$\chi_{agg}(c)(x) := \mathrm{mcv}_{\pi_2} \{\langle i, \chi_i(c_i)(x)\rangle : i \in \mathbf{k}\} \tag{8.21}$$

Another, very simple method is as follows:

$$h_{agg}(x) := \begin{cases} \mathbf{1}, & \text{if } x \in \bigcup_{i \in \mathbf{k}} \llbracket \mathbf{P}_i \rrbracket c_i \\ \mathbf{1/2}, & \text{if } x \in (\bigcup_{i \in \mathbf{k}} \llparenthesis \mathbf{P}_i \rrparenthesis c_i) - (\bigcup_{i \in \mathbf{k}} \llbracket \mathbf{P}_i \rrbracket c_i) \\ \mathbf{1}, & \text{else.} \end{cases} \tag{8.22}$$

Exercise 8.7 (♦) Change the definition of h_{agg} by replacing $\bigcup$ with $\bigcap$ or by using a few more set operations. Are they equivalent?—What are ther differences?

What does bagging mean for the relation sets? Let us assume, that everey c_i is definable. Then, there exist reducts $\mathbf{P}_i \in \mathrm{Red}_{c_i}(\mathbf{R})$ that define c_i but even if $\bigcup_{i \in \mathbf{k}} c_i = c$, it is *not* true that

$$\bigcup_{i=1}^{k} \mathbf{P}_i \in \mathrm{Red}_c(\mathbf{R}) \tag{8.23}$$

The problem here is that reducts are not unique: $\mathrm{Red}_{c_i}(\mathbf{R}) = \{\mathbf{P}_i^1, \mathbf{P}_i^2, \ldots, \mathbf{P}_i^{r_i}\}$. Furthermore, the union of two (different) reducts is never a reduct. Cores are unique, but—if there are at least two disjoint reducts—can be empty, which is not very helpful either: If we try to construct a reduct of $\mathbf{R}$ for c by some operations on the cores of $\mathbf{P}_i$, it might be the case that some cores themselves are empty, or that two of them are disjoint—which leaves the candidate for the core of $\mathbf{R}$ for c empty (even if it is not). All we can state for sure is that

$$\text{whenever } R \in \mathrm{Cor}_{c_i}(\mathbf{R}) \text{ then } R \in \mathrm{Cor}_c(\mathbf{R}) \tag{8.24}$$

Since the core $\mathrm{Cor}_{c_i}(\mathbf{R})$ can be empty, we define a table $T(i,j) = |\{R \in \mathrm{Red}_{c_j}(\mathbf{R}) : R \in \mathbf{P}_i\}|$ with $1 \leq j \leq k$ and $1 \leq i \leq |\mathrm{Red}_{c_j}(\mathbf{R})|$. If $T(i,j) = |\mathrm{Red}_{c_j}(\mathbf{R})|$, then R_i must be an element of $\mathrm{Cor}_c(\mathbf{R})$. Computing this table is computationally infeasible because we would have to check all reducts for all c_i. Accordingly, there is no canonical efficient bottom-up implementation to determine a core from reducts using a bagging-like method.

If, on the other hand, we try to construct reducts from cores (i.e. top-down), we can apply a very simple method (since cores are unique): For a set s of objects, the discernability matrix $\mathbb{D}(\mathbf{P})$ contains the names of all relations by which x_i and x_j can be discriminated. If for some i,j it holds that $D(i,j) = \{R\}$, then R must be an element of $\mathrm{Cor}_s(\mathbf{F})$, because then, R is the *only* relation by which x_i and x_j can be discriminated. The runtime complexity of computing the core this way is $\mathcal{O}(\frac{1}{2}|\mathbf{R}|m^2)$, where the worst case is $\mathrm{Cor}_c(\mathbf{R}) = \mathbf{R}$. Using the same method in bagging with k bags of size m', we obtain $\mathcal{O}(\frac{1}{2}|\mathbf{R}|n(m')^2)$. Furthermore, the algorithm in figure 8.6 benefits from parallel computations of smaller discernability matrices.

```
PROC $relBag$ $(\mathbf{F}, s, k, M)$
{   $\mathbf{H} := \mathbf{F};\ \mathbf{C} = \{\}$
    FOREACH($i \in \mathbf{k}$)
    {   $\mathbf{s}_i :=$ randomselect($M, \mathbf{s}$);
        $\mathbf{C}_i :=$ core($\mathbf{s}_i, \mathbf{H}$);
        $\mathbf{H} := \mathbf{H} - \mathbf{C}_i;\ \mathbf{C} := \mathbf{H} + \mathbf{C}_i;$                % + denotes concatenation
    };
    $\mathbf{H} := \underline{sortby}(\beta, \mathbf{H})$;
    WHILE $(\mathrm{error}_t(\mathbf{C}, s) \geq \varepsilon \ \vee \mathbf{H} = \{\})$
    {   $R :=$ first($\mathbf{H}$); $\mathbf{R} :=$ tail($\mathbf{H}$);
        IF $(\mathrm{error}_t(\mathbf{C}, s) > \mathrm{error}_t(\mathbf{C} \cup \{R\}, s))$ THEN $\{\mathbf{C} := \mathbf{C} + \{R\}\}$;
        $\mathbf{H} := \mathbf{R}$;
    }
    return ($\mathbf{C}$);
}
```

Figure 8.6: Relational Bagging

The complexity of finding reducts depends on the number of relations and the size of the data set. But it also depends on the number k of bags one chooses, the number m' of the size of the smaller samples and the "validity" of the hypotheses that are generated. The converse implication of equation (8.24),

$$R \notin \mathrm{Cor}_{c_i}(\mathbf{R}) \Longrightarrow R \notin \mathrm{Cor}_c(\mathbf{R}) \tag{8.25}$$

becomes true only for $m' \approx m$ and, until $m' = m$, large k.

Clusters and Trees. The popularity of clustering in statistical domains and the simplicity of the bagging procedure has led to a widespread combination of the two methods; especially in data mining scenarios. There even exists a library for bagged clustering based on [Leisch, 1999] with an application to market segmentation, [Dolnicar and Leisch, 2000]. [Dudoit and Fridlyand, 2003] describe an application in the area of bio-informatics; a discipline of increasing importance and with a strong backgorund in statistical methods. Accordingly, the authors stress the use of bagging in reducing the variation over several runs of clustering algorithms by the averaging behaviour of bagging.
Decision tree induction is, as we have seen, basically the same as recursive partitioning with heuristic guidance. Tree induction algorithms have been used extensively in empirical evaluation of both bagging and boosting, [Freund and Schapire, 1996] and [Quinlan, 1996, Quinlan, 2001].

Bagging for ILP. Averaging out a large variance on hypothess is very important in statistial approaches. Another prime candidate for bagging is ILP. The first reason is that there are no probablities in Horn logic which makes bagging easier than boosting in a straightforward implementation. One of the biggest problems in ILP is search. FOIL does an information gain guided search (see 7.3) whereas PROGOL uses an A^*-like algorithm with heuristics described in section 7.4.3. All these algorithms start off with a single sample—and have to carry out the search sequentially; considering, evaluating and refining (or refusing) each hypothesis after another. By choosing several *subsets* of the sample one can try and induce a hypothesis for each of these subsets. Exactly this idea is described and implemented by [and David Page et al., 2002] and, subsequently, in [Raymond J. Mooney, 2004].

8.4.2 Focusing on Errors

In a relational boosting approach we do not have a probability distribution which we could adjust in order to focus on learning error sets.
Instead of boosting the probability of those examples that are misclassified by a hypothesis h_i, we only remove the set of already correctly classified objects from the set of entities to be taken into consideration; i.e. we restrict the search for a hypothesis to what in rough set theory is the boundary region.

Boosting rough sets. Given a hypothesis $\mathbf{H}$, the problem is to find a relation R which is a good candidate to rule out as many elements from the boundary region as possible by adding or removing it from $\mathbf{H}$.
This can be done only heuristically and, as such, is a source of bias. The algorithm shown in Fig. **??** uses a function sortby to pick the "best" relation (determined by the heuristic function β). One possible heuristics could be information gain (see Sect. **??**) or many other, computationally even cheaper methods (for example, choosing R whose index has a certain property). Another, though more expensive method, is to validate R against a test sample $\mathbf{s}_{\text{test}}$ and choose $\beta(R) = \text{error}_t(R, \mathbf{s}_{\text{test}})^{-1}$. To learn c from $\mathbf{F}$ we chose $R \in \mathbf{F}$ with $R = \arg\max\{\beta(R) : R \in \mathbf{F}\}$, hoping that it generates a fine grained partition that has a minimal boundary region on the target concept. We then iterate this process on the boundary region only.
Note that—in contrast to standard boosting—we do not keep a sequence of hypotheses, but we iteratively build a reduct starting from the empty set. As such, it is a bottom-up learning algorithm.

Boosting Clusters an trees. Just as in bagging, the increasing popularity of boosting showed the highest impact on otherwise popular knowledge discovey methods (see the paragraph on clusters and trees in section 8.4.1). A pretty recent development and evalaution of an application of boosting to clustering is described in [Frossyniotis et al., 2004]. Recall also that boosting is a trick to add more information to a learning problem than what it is explicitely provided with. A distribution μ, which we assume to be unknown, basically is nothing but *knowledge*. For example, we can for each (target) concept in $\mathfrak{U}$ add a dimension

```
PROC $\textit{relBoost}$ $(\mathbf{H}, s)$
{   IF $(\underline{\textit{good\_enough}(\mathbf{H})})$ THEN return$(\mathbf{H})$;
    $\mathbf{C} := \underline{\textit{sortby}(\beta, \overline{\mathbf{F}}} - \mathbf{H})$;
    WHILE $(\mathrm{error}_t(\mathbf{H}, s) \geq \varepsilon)$
    {   $\mathbf{C} =: [C|\mathbf{R}]$;
        IF$(\mathrm{error}_t(\mathbf{H} \cup C, s) < \mathrm{error}_t(\mathbf{H}, s))$ THEN
        { $\textit{relBoost}(\mathbf{H} \cup C, \mathrm{errset}_t(\mathbf{H} \cup C, s))$; }
        ELSE
        { return$(\mathbf{H} \cup \textit{relBoost}(\mathbf{R}, \mathrm{errset}_t(\mathbf{H} \cup C, s))$; }
    }
    return$(\perp)$;
}
```

Figure 8.7: Relational “Boosting” by Re-learning Errors Only

(i.e. feature) to the underlying information system $\mathfrak{I}$; each of which encodes the characteristic function of the concept. Then, each such feature defines a distribution itself with $\phi_c(\{x\}) = \frac{1}{m}$, iff $x \in c$. As a result, the *product* space carries all the information as it inherits the product measure. The task of learning c then becomes, statistically speaking, the task of learning ϕ_c. Adding such knowledge to the learning algorithm by expanding $\mathtt{Alg}(\mathbf{s})$ to $\mathtt{Alg}(\phi_c, \mathbf{s})$ (as in line 4 of the algorithms in figures 8.4 and 8.5) or by subsampling with respect to ϕ_c may help to improve the qualitity of the resulting hypothesis. Another kind of knowledge is “local equivalence”: It means that we are not able to describe an entire equivalence class but only equivalence between selected pairs of objects. This is known as *linkage* in clustering. A relational point of view is that a linkage relation (or its dual concept, *non-linkage*) represent subsets of equivalence relations. The search for equivaence relations then is the search for so–called *minimal rectangles* which are difunctional relations, [Ali Jaoua, 2009]. Adding reflexivity to them (which is a safe thing to do as we can assume every singe object to be linked to itself) then induces an equivalence relation (see section 2.1.6). Using this additional knowledege (which, again, can be expressed in terms of a distribution on the set of all pairs of objects) is used for a boosted clustering method described in [Yi Liu, 2007].

8.5 Summary

We have seen that PAC learning is a very pessimistic view on knowledge discovery but it very well reflects what knowledge discovery is about:

The learning dilemma
If a problem is finite, it can be solved by enumeration—which cannot really be considered to be a procedure that builds on what we would call *knowledge*.
In general, the more *interesting* a problem is, the more *complex* it is, too. One aspect of *interestingness* of knowledge is the sheer *need* for it. And the stronger the need, the more likely it is that the knowledge is not trivial and well hidden—if it exists at all.
The morale is: What we know or what we can learn with small effort is not interesting—and anything that is interesting is hard to learn.

In the references you will quite often find the term "data mining"—and with the recent advent (or rediscovery) of relational methods "relational data mining" as well. People often explain this term by the metaphor of mining for rare diamonds in a huge pile of gravel. To us, knowledge discovery means to find answers for the following questions:

- Is there a cognoscible[7] structure in the pile of gravel?
- If so, can this structure be explained in terms of our knowledge?
- And if we then find something else, are we able to explain why we found it, and where it would be promising region to look for another piece?

All these problems are not easy. In this chapter we have described two ensemble-techniques and their applications to relational methods. Both of them are *divide–and–conquer* strategies: *Bagging* means to divide the pile into several heaps and have each of them analysed by its own. *Boosting* means to start off with a simple random search in the pile which in its behaviour is constantly being refined: If our hypothesis is good enough for a certain part of th pile, we know we can search it efficiently later on and focus on finding another, refined technique that will increase the current efficiency on the remaining part of the pile.
This metaphor, in contrast to the mining metaphor, also illustrates the difference between knowledge discovery and data mining:

Mining for Information and the discovery of knowledge
Instead of digging for valuable *pieces* of information in a mine of data, knowledge discovery is about finding *concepts* and *procedures* that describe whether mining will iscover something at all—and, if so, where it would be best to dig first.

[7] Note that one synonym for "cognoscible" is "discernible"!

Chapter 9

The Logic of Knowledge

If we assume knowledge to be what it takes to make rational decisions, knowledge is not logic. It is not even logic when we assume it is representable in an information system.
Many decisions (and every answer to a question *is* a decision) are far from being discrete, deterministic, or deductively comprehensable.
Yet, the simplest question we can ask is: "Is x equal to y?". And it takes knowledge in the form of the ability to discern different things from each other in order to decide whether one should answer "Yes" or "No".

During the last decades, Machine Learning evolved from theories of reasoning in Artificial Intelligence to an essential component of software systems. Statistical methods outperform logic based approaches in most application domains—and with increasing computational power it became possible to *generate-and-test* classifiers. As a more sophisticated approach *ensemble learning* implements divide-and-conquer strategies on the learning problem. With the further increase of data collections (e.g. data warehouses), the problem we are facing is not concerned with *how* we can induce a classifier that supports our model assumptions on the data but rather to understand *what* kind of information there actually *is*. In Machine Learning, this approach is known as *knowledge discovery.*

9.1 Knowledge Representation

Since we are used to describe knowledge in the language of terminologic logic, we quite often identify knowledge representation with logic models. In chapter 3, we have seen that knowledge can be represented in many, many different ways. Each representation formalism has its own advantages and downsides.

We may also find several alternative representations for one an the same set of knowledge. If there are two such alternatives, they can be equivalent—or not. In the case they are equivalent, we can argue that there is a lossless representation shift from one representation spae into the other (and back). This is not just a funny thig to do, but it also has a practical relevance: It can be that the kind of knowledge that we are looking for can be found much easier in one representation system than in another. In such a case we would first translate the source representation into the one that is easy to work with, induce a hypothesis and then—if we wish—translate it back into the original language.

If the two systems are *not* equivalent then a representation shift is always conjoined to some loss of information. Such a loss can be deliberate for two reasons: First, if the loss is tolerable, then we can accept a weaker hypothesis. The level of tolerance is often defined by a tradeoff between the loss of accuracy and the gain of efficiency. Second, loss can be usefull. Imagine again the metaphor of searching a diamond in a heap of gravel. The heap is the source representation. We now apply a lossy transform and use a huge fan to blow all the stones onto a large conveyor belt. This transform is lossy, because the strong air stream takes away all the dust and little pieces of gravel that are too light to be a valuable diamond.

Next, we asked ourselves where all the evidence comes from that we build our hypotheses upon. One of the most important terms we discovered in the section 3.3 on *samples*: Our "quest for knowledge" is *biased* in many, many ways. Some biases are due to the representation (one can't express the number $\frac{2}{7}$ in $\mathbb{N}$), others are due to the samples or the method by which the samples are taken. Again, just as with representations of different advantages and disadvantages, biases are not always negative. If we know that the number we are looking for is $\frac{2}{7}$, then there is no need to inspect all numbers in $\mathbb{R}$!

Finally, we asked ourselves what it means for a hypothesis to be "good". This actually, is just another bias—because it is always that we stop our search for a hypothesis once it is good enough. On the other hand, it is by no means guaranteed that any artifical measure of quality (of wich tere are so many) reflects "adequacy", "suitability" or any kind of measure to describe whether it is meaningful or not in real life.

All of these concepts—representation, representation shifts, information loss, biases, and all the different measures of quality could be very well described in terms of relations only.

9.2 Learning

The second part of the book consists of the description of several relational or logic learning paradigms:

- clustering, where one wants to find a meaningful partitioning of the data set;

- decision tree induction which is about an efficient search for hierarchical clustering;
- rough set data analysis as an unbiased and exhausitve approach to find minimal sets of relations that suffice to describe the target concept;
- and finally, inductive logic programming which can be considered an improvement of the relational approaches where the extension of concepts in a relational description corresponds to the satisfaction set of an according predicate definition.

As one can see from this very brief list and the description of all paradigms in less than a sentence each, the similarities are huge.

9.2.1 Clustering

Given a predefined set of clusters, the question to which class in a classification $\mathfrak{c} = \{c_i : i \in \mathbf{n}\}$ a certain object $x \in U$ belongs, k–NN is a simple *voting* algorithm that assigns to x the same class identifier $i \in \mathbf{n}$ as most of of the k nearest neighbours of x have. The notion of "near" presupposes a distance measure—and there are many such measures. The simplest one is a binary measure we use in the evaluation of hypotheses as well: If two objects are the same, their distance is 0; otherwise it is 1. Reducing a multiclass learning problem ($|\mathfrak{c}| > 2$) to n binary problems is not possible using this measure for we are not able to decide wich classifier should be trusted if there are several ones that return 1. But if we can use an euclidean measure, then the minimum distance of an object to all the cluster centroids can be taken as an aggregated hypothesis.[1]
Learning clusters means to learn equivalence classes by defining disjoint subsets of objects representing classes. It starts by randomly choosing class representatives: the random choice of centroids c_i. Then, for every object $x \in U$, the algorithm has to decide for which i it shall be true that $x \in [c_i]$. This decision is made by asking a distance measure again—but also, and this also forms a bridge between the spatial representation as in figure 4.2 and the *relational* notion of "rectangles" (see section 8.4.2).

9.2.2 Decision Tree

I hesitated whether the induction of decision trees should be included in this book. After all, there are many other machine learning approaches that were not mentioned in this book; Artificial neural networks, Bayesian reasoning, Support Vector Machines just to name a few. Even though decision tree induction make use of an entropy based heuristics, it is still a relational method. The resulting tree is just a "recipe" for classfiying a new and unknown object along a hierarchy of partitions. Whether such a hierarchy is expressed in terms of a tree, by

[1] This shows the connectedness of a relational representation, representation transforms, a clustering approach, and the notion of using a set of hypotheses compute an aggregated hypothesis as coined by bagging.

a concept hierarchy as in figure 4.3, by a recursive relational notation as in equation (5.10) or by a set of rules as in section 5.5.2—it's just matter of personal preference. The only thing that makes decision tree induction a bit different from all other relational approaches is just that it does not work without biases—but *with* biases, it is one of the most efficient ones.

9.3 Rough Sets

Rough set data analysis is, finally, relational learning without any bells and whistles. What it makes so special though, is, first the non-numerical method of representing vague membership and, more importantly, its framework for reasoning about sets of relations rather than about relations between objects. Decision tree induction does not "think" about dependencies between relations. It simply chooses the one that has the highest expected information gain in each step. When applying a rule-based post-pruning method, there is a nice effect showing that this greedy method is in fact myopic: If it wasn't, then rule based post-pruning could never result in a forest (see figure 5.5.2). Rough set data analysis has a broader view on the hypothesis space: If there are different alternatives for adding a relation P or Q to a set of relations $\mathbf{R}$ such that both $(\!|\mathbf{R} \cup \{P\}|\!) c_i subseteq (\!|\mathbf{R}|\!)$ and $(\!|\mathbf{R} \cup \{Q\}|\!) c_i subseteq (\!|\mathbf{R}|\!)$, then both$\mathbf{R} \cup \{P\}$ and $\mathbf{R} \cup \{Q\}$ are candidates for promising reducts. If then c_i then is just one of many classes in $\mathfrak{c}_i$, the power of rough set data analysis lies in the comparison of the utility of P and Q in terms of $\mathfrak{c}$: Whereas decision tree induction simply chooses between P and Q with respect to their information content on c, we can take into account $[\![\{P\} \trianglelefteq \mathbf{R}]\!]\mathfrak{c}$ and its relation to $[\![\{Q\} \trianglelefteq \mathbf{R}]\!]\mathfrak{c}$. Another very important thing about rough set theory is that has been studied in connection with multi-valued and multi-modal logics ([Orlowska, 1993, Yao, 2003, Düntsch, 1997]) and formal concept analysis, [Xu et al., 2008, Düntsch et al., 2007].

9.4 Inductive Logic programming

The last learning paradigm we discussed was inductive logic programming. It is by far the most powerful of all the approaches presented. Simple ILP learning problems are PAC-learnable [Džeroski et al., 1992], but it is also, in terms of computational effort, the most expensive of all methods presented here. Especially when not restricted to horn clauses—which is required when we want to work with a proper negation and a non-restricted logic representation language—then one has to abandon all hope. Similarly, the logic of negation as failure makes it not easy to properly define when a negative example is not implied by a theory. On the other hand, it is exactly this what brings about the idea of Heyting-Algebras as interpretations of logic programs and which connects the semantics of logic programs to rough set theories.
Understanding Horn theories as approximations is not a new idea, [Kautz et al., 1995], but there were only few contributions from the inductive logic programming

community which is mostly due to the generally rather negative results, e.g. [Nock and Jappy, 1998]. Every $R \in \mathbf{F}$ also defines a binary predicate $\mathtt{r}(x, f_R(x))$.[2] The satisfaction set of $\mathtt{r}$ is the set of all instantiations of x for which $\mathtt{r}$ holds and whose meaning equals the corresponding R-equivalence class:

$$[x]_R = \{y \in \mathfrak{U} : \mathtt{r}(y, f_R(x))\}.$$

In order to derive $\mathtt{r}(x, v)$ from a given set of clauses $\sqcap$, that is, $\sqcap \cup \sqcap \vdash \mathtt{r}(x, v)$, one needs to show that there is a correct answer substitution θ such that $(\sqcap \cup \sqcap \cup \{\neg\mathtt{r}(x, v)\})\theta \vdash \square$. An optimal hypothesis $\sqcap$ guarantees that

$$\forall\theta : \mathtt{r}(X, f_R(X))\theta \Longleftrightarrow (\sqcap \cup \sqcap\{\neg\mathtt{r}(X, V)\})\theta \vdash \square \tag{9.1}$$

Since $\vdash$ is correct but not complete, we are able to give a lower approximation of r where $[\![\sqcap]\!]r$ describes a subset of the satisfaction set of r:

$$[\![\sqcap]\!]r \quad :\longleftrightarrow \quad \{y : (\sqcap \cup \sqcap \cup \{\neg\mathtt{r}(X, V)\})\theta \vdash \square\} \tag{9.2}$$

Another method is to encode each feature f into a predicate symbol $\mathtt{f}$. Then a Horn theory $\sqcap \cup \sqcap$ models an information system, if:

$$\forall x \in \mathfrak{U}\, \forall f \in \mathbf{F} : \sqcap \cup \sqcap \vdash \mathtt{f}(x, y) \Longleftrightarrow f(x) = y$$

A *Horn reduct* $\sqcap'$ of $\sqcap$ is a set of clauses where for each clause $\varphi \in \sqcap'$ there is a clause $\psi \in \sqcap$ and a substitution θ such that $\varphi \subseteq \psi\theta$ and $\sqcap$ and $\sqcap'$ induce the same theory. Translating the original definition of a reduct, $\sqcap' \in \mathrm{Red}_s(\sqcap)$ holds if $\sqcap' \in \mathrm{Red}_s(\sqcap)$ is true and the satisfaction set of $\sqcap'$ equals the satisfaction set of $\sqcap$ on s. As an example, let us consider the case of *literal dropping* and its relationship to building reducts. Let there be two clauses,

$$\begin{array}{rcl} \mathtt{t}(x, \mathbf{1}) & \leftarrow & \mathtt{p}_1(x, f_1(x)) \wedge \cdots \wedge \mathtt{p}_k(x, f_k(x)) \wedge \mathtt{p}_{k+1}(x, f_{k+1}(x)) \text{ and} \\ \mathtt{t}(x, \mathbf{1}) & \leftarrow & \mathtt{p}_1(x, f_1(x)) \wedge \cdots \wedge \mathtt{p}_k(x, f_k(x)). \end{array}$$

Obviously, the former implies the latter. If the satisfaction sets of both are the same, then the second clause is a *reduct* of the first one by dropping one literal, or, equivalently, by dropping one feature or equivalence relation. If

$$\begin{array}{rcl} \mathtt{t}(x, \mathbf{1}) & \leftarrow & \mathtt{p}_1(x, a) \wedge \mathtt{p}_2(x, b) \\ \mathtt{t}(x, \mathbf{1}) & \leftarrow & \mathtt{p}_1(x, a) \wedge \mathtt{p}_2(x, c) \end{array}$$

one can induce $\mathtt{t}(x, \mathbf{1}) \leftarrow \mathtt{p}_1(x, a) \wedge \mathtt{p}_2(x, y)$ or even $\mathtt{t}(x, \mathbf{1}) \leftarrow \mathtt{p}_1(x, a)$.

[2] $R \in \mathbf{F}$ is an equivalence relation induced by $f_R \in \mathbf{F}$, whereas $r \subseteq \mathfrak{U}$ denotes a concept. $\mathtt{r}$ finally is a predicate, whose definition is unknown and needs to be learned such that its satisfaction set approximates r.

9.5 Summary

One of the main goals of this book is to present several paradigms of knowledge discovery in a unifying framework. This may result in a state of mild confusion—some students reported to me, that it all appears to be the same anyway and I should stop repeating myself. Well, at a certain level of abstraction things *are* equal. By now, you know why: Just drop all clusters except for one, prune you decision tree right after the root node, choose as a hypothesis the set of relations with only the universal relation in it or define a predicate with variables in the head only and an empty body.
Of course, the different algorithms are *not* the same. But they share common properties. This offers two very important opportunities:

- If one looks at a problem from different perspectives, it is more likely to identify general problems in solving the problem and to find mistakes or weaknesses in other formalisations.

- If one method can solve a certain problem with some difficulties and another method can solve another problem with some other difficulties, too—maybe the can profit from each other by changing problems; or, from the point of view of a programmer: change the paradigm rather than the problem.

The more knowledge one puts into something, the less new knowledge one can get out of it. Therefore, one should let the facts speak for themselves and build models on observation—rather than looking for examples where one hopes they would satisfy ones model.

Chapter 10

Indexes and Bibliography

Notation

The following fonts are used in this book. With a few exception, their meanings are:

a, b, c, . . .	Roman small characters	Entities/Atoms/Objects, elements of a set
$\ldots, x, y, z$	Italic small roman characters	Variables
$\ldots, f, g, h, \ldots$	Italic small roman characters	Functions
$\ldots, X, Y, Z$	Capital italic roman characters	Random Variables
$\ldots, R, S, T, \ldots$	Capital italic roman characters	Relations
$\ldots, \mathbf{F}, \ldots, \mathbf{P}, \mathbf{Q}, \mathbf{R}, \ldots$	Capital bold roman characters	Sets of relations
$\mathfrak{A}, \mathfrak{B}, \mathfrak{C}, \ldots$	Capital gothic characters	Structures of any kind
$\alpha, \gamma, \delta, \ldots$	Lowercase greek characters	Variables, constants, parameters
$\varphi, \psi, \ldots, \kappa, \lambda, \nu,$		Formulas, literals
$\mu, prob$	distributions	
μ		most general unifier

Special Sets

$\mathbf{F}$	The set of features
$\mathfrak{D}$	Domain
$\mathfrak{U}$	Universe, our representation space
U	the base set of $\mathfrak{U}$
c	a concept $c \subset s$, $c \subset U$
$\mathfrak{c}$	a classification is a set of concepts: $\mathfrak{c} = \{c_0, c_1, \ldots\}$
$\mathbf{0}, \mathbf{1}, \mathbf{2}, \ldots, \mathbf{k}, \ldots$	sets with $0, 1, 2, \ldots, k, \ldots$ elements with $\mathbf{0} = \emptyset$ and $\mathbf{k} = (k-1) \cup \{(k-1)\}$
$\mathbf{2}$	the set $\{\mathbf{1}, \mathbf{0}\}$
$\mathbb{N}$	natural numbers $\{1, 2, 3, \ldots\}$
$\mathbb{Q}$	Rational numbers
$\mathbb{R}$	real numbers
$\mathbf{s}$	sample

Special variables or functions

m	Usually, $\lvert\mathbf{s}\rvert$.
n	Usually, $\lvert\mathbf{F}\rvert$ of features $f \in \mathbf{F}$. m and n are also used for $f : s^m \to s^n$ or any upper bound.
k	the number $\lvert\mathfrak{c}\rvert$ of classes in a classification $\mathfrak{c} = \{c_0, c_1, \ldots, c_{k-1}\}$
i, j	running indices
k, l	running indices / boundaries
$\vec{x}$	vector $\langle x_0, x_1, \ldots, x_{n-1} \rangle$
σ	substitution
ε, δ	small values; usually errors and differences
μ	probability distributions

Relations, Functions, Operators

$R, S, \ldots$	Relation; usually endorelations
	declared as $R \subseteq s \times s$ or $R : s \rightharpoondown s$
$\mathbf{R}$	a set of relations
	usually the set of equivalence relations induced by $\mathbf{F}$
$\mathrm{dom}(R)$	domain of R
$\mathrm{cod}(R)$	codomain of R
${}^{\ulcorner}Rs$	preimage of s under R
	$\{x \in \mathrm{dom}(R) : xRy \wedge y \in s \subseteq \mathrm{cod}(R)\}$
$sR^{\urcorner}$	image of s under R
	$\{y \in \mathrm{cod}(R) : xRy \wedge x \in s \subseteq \mathrm{dom}(R)\}$
$R^{\smile}$	converse $R^{\smile} = \{\langle y, x\rangle : xRy\}$
$\overline{R}$	complement $\overline{R} = \{\langle y, x\rangle : \neg xRy\}$

$f, g, \ldots$	functions $f, g : s_0 \to s_1$ with s_i arbitrary sets.
$\mathbf{F}$	a set of functions
R_f	equivalence relation induced by function f: $xRy \equiv f(x) = f(y)$
s/R	quotient; partition induced by equivalence relation R
s/f	quotient; partition induced by function f and R_f
$[x]_R$	equivalence class of x induced by R

Special Functions

$f, g, h, \ldots$	any function
$\chi(s)$	characteristic function of a set s into $\mathbf{2}$
$t_{\mathfrak{c}}$	target function $t_{\mathfrak{c}} : U \to \mathbf{k}$
t_c	target function $\chi(c)$.
ρ	representation $\rho : \mathfrak{D} \to \mathfrak{U}$
τ	transform/representation shift: $\tau : \mathfrak{U} \to \mathfrak{U}'$
$\lvert y\rvert_x$	the number of occurences of x in y (with y a vector or a word).
$\lvert x\rvert$	if x is a scalar, $\lvert x\rvert$ is its absolute value
	if x is a set, then $\lvert x\rvert$ is x's cardinality
$\ell(\vec{x})$	dimension of $\vec{x}$
$\ell(w)$	length of word w
$\chi(s)$	characteristic function of set s

Arithmetics

$x \cdot y$	(or xy) scalar multiplication
$x \cdot \vec{y}$	(or $x\vec{y}$) scalar multiplication
$\vec{x} \cdot \vec{y}$	(or $\vec{x}\vec{y}$) dot product
$s_0 \odot s_1$	pairwise intersection of sets of sets: $\{s_0' \cap s_1' : s_0' \in s_0, s_1' \in s_1\}$
$\circ$	concatenation
$\bullet$	Hadamard product; component-wise product of two vectors/matrices
$\times$	set/cross product

Notational Oddities

$\vec{x}$ a vector $\langle x_0, x_1, \ldots, x_{n-1} \rangle \in s_0 \times s_1 \times \cdots \times s_{n-1}$ with length $|\vec{x}| = n$. Vectors are *not* written as x to avoid confusion with number sets $\mathbf{x}$.

$x[i]$ the i-the component $x[i] = x_i \in s_i$ of $\vec{x}$ (used to avoid double indexing)

sums $\sum_{i=0}^{n-1} x_i = \sum_{i \in \mathbf{n}} x_i$

Bibliography

[Ali Jaoua, 2009] Ali Jaoua, Rehab Duwairi, S. E. S. B. Y. (2009). *Relations and Kleene Algebra in Computer Science*, volume 5827/2009 of *Lecture Notes in Computer Science*, chapter Data Mining, Reasoning and Incremental Information Retrieval through Non Enlargeable Rectangular Relation Coverage, pages 199–210. Springer.

[and David Page et al., 2002] and David Page, Costa, V. S., and Shavlik, J. (2002). An empirical evaluation of bagging in inductive logic programming. In *Proceedings of the 12th international conference on Inductive logic programming.*

[Anthony and Biggs, 1997] Anthony, M. and Biggs, N. (1997). *Computational Learning Theory.* Cambridge University Press, 2nd edition.

[Ash, 1965] Ash, R. B. (1965). *Information Theory.* Dover Publications.

[Barron et al., 1998] Barron, A., Rissanen, J., and Yu, B. (1998). The minimum description length principle in coding and modeling. *IEEE Transactions on Information Theory*, 44(6):2743–2760.

[Bratko, 1986] Bratko, I. (1986). *Prolog for artificial intelligence.* Addison-Wesley, London.

[Breiman, 1994] Breiman, L. (1994). Bagging predictors. Technical Report 421, University of California, Berkeley.

[Breiman, 1996] Breiman, L. (1996). Heuristics of instability and stabilization in model selection. *The Annals of Statistics*, 24(6).

[Chaitin, 1966] Chaitin, G. J. (1966). On the length of programs for computing binary sequences. *JACM*, 13:547–569.

[Chaitin, 1987] Chaitin, G. J. (1987). *Information, Randomness and Incompleteness - Papers on Algorithmic Information Theory.* World Scientific Press, Singapore.

[Dolnicar and Leisch, 2000] Dolnicar, S. and Leisch, F. (2000). Getting more out of binary data: Segmenting markets by bagged clustering. Technical Report 71, Vienna University of Economics and Business Administration.

[Dudoit and Fridlyand, 2003] Dudoit, S. and Fridlyand, J. (2003). Bagging to improve the accuracy of a clustering procedure. *Bioinformatics*, 19(9).

[Düntsch, 1997] Düntsch, I. (1997). A logic for rough sets. *Theoretical Computer Science*, 179(1-2):427–436.

[Düntsch et al., 2007] Düntsch, I., Gediga, G., and Orłowska, E. (2007). Relational attribute systems ii: Reasoning with relations in information structures. In *Transactions on Rough Sets VII*, number 4400 in LNCS. Springer.

[Džeroski et al., 1992] Džeroski, S., Muggleton, S., and Russell, S. (1992). PAC-learnability of determinate logic programs. In *Proceedings of the 5th ACM Workshop on Computational Learning Theory*, pages 128–135, New York, NY. ACM Press.

[Ehrig et al., 2001] Ehrig, H., Mahr, B., Cornelius, F., Große-Rhode, M., and Zeitz, P. (2001). *Mathematisch-strukturelle Grundlagen der Informatik*. Springer, 2 edition.

[Freund and Schapire, 1996] Freund, Y. and Schapire, R. E. (1996). Experiments with a new boosting algorithm. In *Proc. 19th Intl. Conf. Machine Learning*.

[Frossyniotis et al., 2004] Frossyniotis, D. S., Likas, A. C., and Stafylopatis, A. (2004). A clustering method based on boosting. *Pattern Recognition Letters*, 25(6):641 – 654.

[Gottlob, 1987] Gottlob, G. (1987). Subsumption and implication. *Information Processing Letters*, 24(2):109–111.

[György, 1968] György, P. (1968). *Induction and Analogy in Mathematics*, volume 1 of *Mathematics and Plausible Reasoning*. Princeton University Press.

[Horn, 1951] Horn, A. (1951). On sentences which are true of direct unions of algebras. *ournal of Symbolic Logic*, 16(1):14–21.

[Huth and Ryan, 2004] Huth, M. and Ryan, M. (2004). *Logic in Computer Science*. Cambridge University Press, 2 edition.

[Kautz et al., 1995] Kautz, H., Kearns, M., and Selman, B. (1995). Horn approximations of empirical data. *Artificial Intelligence*, 74(1).

[Kearns, 1990] Kearns, M. J. (1990). *The Computational Complexity of Machine Learning*. MIT Press.

[Kearns and Vazirani, 1994] Kearns, M. J. and Vazirani, U. V. (1994). *An Introduction to Computational Learning Theory*. MIT Press.

[Kersting, 2008] Kersting, K. (2008). *Probabilistic Inductive Logic Programming*. Springer.

[Kersting and Raedt, 2000] Kersting, K. and Raedt, L. D. (2000). Bayesian logic programs. In Cussens, J. and Frisch, A., editors, *Proceedings of the Work-in-Progress Track at the 10th International Conference on Inductive Logic Programming*, pages 138–155.

[Kolmogorov, 1965] Kolmogorov, A. (1965). Three approaches to the quantitative definition of information. *Prob. Inf. Trans.*, 1:1–7.

[Leisch, 1999] Leisch, F. (1999). Bagged clustering. Technical Report 51, Vienna University of Economics and Business Administration.

[Li and Vitanyi, 1993] Li, M. and Vitanyi, P. (1993). *An introduction to Kolmogorov complexity and its applications*. Springer-Verlag, Berlin.

[MacKay, 2003] MacKay, D. J. C. (2003). *Information Theory, Inference, and Learning Algorithms*. Cambridge University Press.

[Mazzola et al., 2006] Mazzola, G. B., Milmeister, G., and Weissmann, J. (2006). *Comprehensive Mathematics for Computer Scientists*, volume 1-2. Springer, 2 edition.

[Muggleton, 1995] Muggleton, S. (1995). Inverse entailment and Progol. *New Generation Computing*, 13:245–286.

[Muggleton and Buntine, 1988] Muggleton, S. and Buntine, W. (1988). Machine invention of first-order predicates by inverting resolution. In *Proceedings of the 5th International Conference on Machine Learning*, pages 339–352. Kaufmann.

[Muggleton and Feng, 1990] Muggleton, S. and Feng, C. (1990). Efficient induction of logic programs. In *Proceedings of the First Conference on Algorithmic Learning Theory*, pages 368–381, Tokyo. Ohmsha.

[Muggleton and Feng, 1992] Muggleton, S. and Feng, C. (1992). Efficient induction of logic programs. In Muggleton, S., editor, *Inductive Logic Programming*, pages 281–298. Academic Press, London.

[Muggleton et al., 1992] Muggleton, S., King, R., and Sternberg, M. (1992). Protein secondary structure prediction using logic-based machine learning. *Protein Engineering*, 5(7):647–657.

[Muggleton et al., 1998] Muggleton, S., Srinivasan, A., King, R., and Sternberg, M. (1998). Biochemical knowledge discovery using Inductive Logic Programming. In Motoda, H., editor, *Proc. of the first Conference on Discovery Science*, Berlin. Springer-Verlag.

[Müller, 2008] Müller, M. E. (2008). Learning from noise. *AI Magazine*, 2(29).

[Nienhuys-Cheng and Wolf, 1996] Nienhuys-Cheng, S.-H. and Wolf, R. D. (1996). Least generalizations under implication. In Muggleton, S. H., editor, *Proc. ILP'96*. Springer.

[Nock and Jappy, 1998] Nock, R. and Jappy, P. (1998). Function-free Horn clauses are hard to approximate. In Page, D., editor, *Proceedings of the Eighth Inductive Logic Programming Workshop (ILP98)*, Berlin. Springer-Verlag. LNAI 1446.

[of Ockham, 1323] of Ockham, W. (1323). *Summa totius logicae.* n/a.

[Orlowska, 1993] Orlowska, E. (1993). Reasoning with incomplete information: Rough set based information logics. In *Proceedings of the SOFTEKS Workshop on Incompleteness and Uncertainty in Information Systems*, pages 16–33.

[Popper, 2002] Popper, K. R. (2002). *The Logic of Scientific Discovery.* Routledge.

[Quinlan, 1991] Quinlan, J. (1991). Determinate literals in inductive logic programming. In *IJCAI-91: Proceedings of the Twelfth International Joint Conference on Artificial Intelligence*, pages 746–750, San Mateo, CA:. Morgan-Kaufmann.

[Quinlan and Cameron, 1995] Quinlan, J. and Cameron, R. (1995). Induction of logic programs: FOIL and related systems. *New Generation Computing*, 13:287–312.

[Quinlan and Cameron-Jones, 1993] Quinlan, J. and Cameron-Jones, R. (1993). FOIL: a midterm report. In Brazdil, P., editor, *Proceedings of the 6th European Conference on Machine Learning*, volume 667 of *Lecture Notes in Artificial Intelligence*, pages 3–20. Springer-Verlag.

[Quinlan, 2001] Quinlan, J. R. (2001). Relational data mining. In Džeroski, S., editor, *Relational learning and boosting.* Springer.

[Quinlan, 1996] Quinlan, R. (1996). Bagging, boosting, and c4.5. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, pages 725–730. AAAI Press.

[Raedt, 2008] Raedt, L. D. (2008). *Logical and Relational Learning.* Cognitive Technologies. Springer.

[Raedt and Kersting, 2004] Raedt, L. D. and Kersting, K. (2004). Probabilistic inductive logic programming. In Ben-David, S., Case, J., and Maruoka, A., editors, *Proceedings of the 15th International Conference on Algorithmic Learning Theory*, volume 3244 of *Lecture Notes in Computer Science.* Springer-Verlag.

[Raymond J. Mooney, 2004] Raymond J. Mooney, Prem Melville, L. R. T. J. S. I. d. C. D. D. P. V. . S. C. (2004). *Data Mining: Next Generation Challenges and Future Directions*, chapter Relational Data Mining with Inductive Logic Programming for Link Discovery. AAAI Press.

[Rouveirol, 1992] Rouveirol, C. (1992). Extensions of inversion of resolution applied to theory completion. In Muggleton, S., editor, *Inductive Logic Programming*. Academic Press, London.

[Russell, 1992] Russell, B. A. W. (1992). *Theory of Knowledge: The 1913 Manuscript*. Routledge.

[Russell, 1995] Russell, B. A. W. (1995). *An Inquiry into Meaning and Truth*. Routledge.

[Schapire, 1990] Schapire, R. E. (1990). The strength of weak learnability. *Machine Learning*, 5:197–227.

[Shannon and Weaver, 1949] Shannon, C. E. and Weaver, W. (1949). *The Mathematical Theory of Communication*. University of Illinois Press, Urbana. Reprints 1963, 1998.

[Solomonoff, 1964] Solomonoff, R. (1964). A formal theory of inductive inference. *Information and Control*, 7:376–388.

[Sperschneider and Antoniou, 1991] Sperschneider, V. and Antoniou, G. (1991). *Logic - A Foundation for Computer Science*. Addison-Wesley.

[Valiant, 1984] Valiant, L. (1984). A theory of the learnable. *Communications of the ACM*, 27:1134–1142.

[Welsh, 19xx] Welsh, D. (19xx). *Codes and Cryptography*. Oxford University Press.

[Wolpert and Macready, 1997] Wolpert and Macready (1997). No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82.

[Xu et al., 2008] Xu, F., Yao, Y., and Miao, D. (2008). Rough set approximations in formal concept analysis and knowledge spaces. In *Foundations of Intelligent Systems*, LNCS. Springer.

[Yamamoto, 1997] Yamamoto, A. (1997). Which hypotheses can be found with inverse entailment? In Lavrač, N. and Džeroski, S., editors, *Proceedings of the Seventh International Workshop on Inductive Logic Programming*, pages 296–308. Springer-Verlag, Berlin. LNAI 1297.

[Yao, 2003] Yao, Y. Y. (2003). On generalizing rough set theory. In *Rough Sets, Fuzzy Sets, Data Mining, and Granular Computing (Proc. 9th Int. Conf.)*, number 2639 in LNAI. Springer.

[Yi Liu, 2007] Yi Liu, Rong Jin, A. K. J. (2007). Boostcluster: Boosting clustering by pairwise constraints. In *The Thirteenth International Conference on Knowledge Discovery and Data Mining*.

Index