Automated reasoning

Gerard A.W. Vreeswijk
Utrecht University

Version January 22, 2010

Chapter 11

Learning new rules

The existence of rules of inference is a necessary condition to reason with data. The previous chapters departed from the assumption that such rules exist, and can be used. But what if they do not exist? In that case we will have to produce, some would say *induce*, rules of inference from raw data. In this chapter, a number of algorithms will be formulated to learn rules of inference from data.

11.1 Cases

The idea behind rule learning is that rules can be learnt from cases. The notion of case is based on the more elementary notion of situation description. A *situation description* is a description of a situation at a particular moment in time consisting of a enumeration of properties that hold at that point in time, and/or several events that happened during that point in time. Think of it like a snapshot. An example of (a fragment of) a list of situation descriptions is

Timepoints	*Cases*
⋮	⋮
t2006:	Tour de France, World Championship Football, DeNiro Movie
t2007:	¬New president, DeNiro Movie, Eclipse
t2008:	Tour de France, Olympic Games, European Championship Football, New president, DeNiro Movie
t2009:	Tour de France, DeNiro Movie
t2010:	Tour de France, World Championship Football, DeNiro Movie
t2011:	Tour de France, DeNiro Movie
t2012:	Olympic Games, Tour de France, European Championship Football, DeNiro Movie
⋮	⋮

Here, time points are years, and each case is a (more or less arbitrary) enumeration of events that happened during that year. Thus, in 2006 there was the Tour de France, there was the World Championship Football, and in that year, a Robert DeNiro Movie came out. Then, in 2007, there was the Eclipse, we had the same president throughout that year and, again, a Robert DeNiro Movie came out. And so forth.

A more abstract example of a (chronological) list of situation descriptions is

Timepoint	*Situation description*
⋮	⋮
t2006:	$\neg a, b, \neg d, g, \neg j, \neg k, m$
t2007:	$b, \neg c, d, \neg e, f, h, \neg i, j$
t2008:	$a, c, \neg d, \neg e, f, \neg k, l, \neg m$
t2009:	$a, b, d, \neg g, h, j, k, \neg m$
t2010:	$\neg b, c, e, \neg f, g, \neg i, \neg j, n$
t2011:	$a, \neg c, e, \neg f, \neg k, l, \neg m$
⋮	⋮

A typical list of situation descriptions from which sensible rules can be distilled from, consists of about $1,000$ or more situations, where each situation consists of, say, 20 or more features. The more situation descriptions and the more features, the better.

Rules can be extracted from data in different ways. In this chapter, we will confine ourselves to discovering patterns in situations per sé, rather than discovering patterns in situations through time. For example, if c occurs every time a or b occurs, and if c is absent every time a and b are absent, then there is an algorithm in this chapter that will produce the rules $a \rightarrow c$ and $b \rightarrow c$.[1] However, this chapter does not provide methodes to discover causal patterns that may occur trough time. For example, if c happens at t_n every time a happens at t_{n-1}, then we might say that a causes c and write the rule $a \text{–}(causes)\!\!\rightarrowtail c$. Learning causal rules, however, is beyond the scope of this chapter. Be reassured: most if not all treatises on rule learning are aimed at discovering patterns in situations per sé and not at discovering causal patterns through time [77].

Now for cases. A *case* is a situation description in which one event is singled out, usually because we want to know under which particular circumstances this single event has happened. An example of a case list for f is

Timepoint	*Case*
⋮	⋮
t2006:	$\neg a, b, \neg d, g, \neg j, \neg k, m$
t2007:	$b, \neg c, d, \neg e, h, \neg i, j \succ f$
t2008:	$a, c, \neg d, \neg e, \neg k, l, \neg m \succ f$
t2009:	$a, b, d, \neg g, h, j, k, \neg m$
t2010:	$\neg b, c, e, g, \neg i, \neg j, n \succ \neg f$
t2011:	$a, \neg c, e, \neg k, l, \neg m \succ \neg f$
⋮	⋮

This list is made by by taking f out of the situation description and putting it at the RHS behind a $\succ$ -sign. Cases ending with an f are sometimes called *positive examples for* f, or *positive instances for* f, because they describe situations in which f occurs. Cases ending with $\neg f$ are sometimes called *negative examples for* f, or *negative instances for* f, because they describe situations in which $\neg f$ occurs. There are also cases that are neither positive nor negative. Case lists for other literals such as a, $\neg a$, b and $\neg b$, can be made in a similar fashion.

Representing cases by using the $\succ$ -notation demonstrates nicely that examples play the role of cases in machine learning. On the other hand, a disadvantage of the use of $\succ$ is that it is redundant and not really essential to the rule learning algorithms that we present in this chapter. If an algorithm is dedicated to learning rules on f, it simply singles out f without having to rely on a $\succ$ -symbol of some sort. Therefore, this chapter further works with the earlier mentioned and more general notion of situation description.

[1] Algorithm 10 on page 227.

11.2 Some concepts from machine learning

Machine learning terminology is the best vehicle to understand how rules can be learnt from cases. We therefore introduce some basic notions and concepts.

In machine learning a rule, or set of rules, would be called a *hypothesis*, and a case would be called an example, observation, or *instance*. A typical machine learning task, then, is to formulate a hypothesis, H, that explains the classification of observed instances $I_1, \ldots, I_n$ and predicts the classification of unobserved (unseen) instances $I_{n+1}, \ldots$. A typical machine learning task, for example, is to explain why an insurance company accepts certain insurance applications and rejects others. Another example is to discover which customers tend to buy a certain commercial product, based on customer profiles.

As a running example in this section we try to find an hypothesis H that explains the following series:

$$\begin{array}{lllll} (22,25) \to + & (76,54) \to - & (37,23) \to + & (37,37) \to + & (25,80) \to - \\ (34,75) \to - & (85,78) \to - & (22,38) \to + & (90,10) \to - & (50,50) \to - \end{array} \tag{11.1}$$

Thus, the first instance is the pair $(22, 25)$ that is classified positive, the second instance is the pair $(76, 54)$ that is classified negative, and so forth. The objective is to find, or rather produce, a simple hypothesis that explains all positive instances, and rules out (does not explain) all negative instances. An example of a hypothesis is "the set of positive points is formed by the disk that consists of all points with a distance less than 5 from $(25, 25)$". Another example of a hypothesis is "the set of positive points is formed by the set of all points (x, y) such that $x + y$ is even".

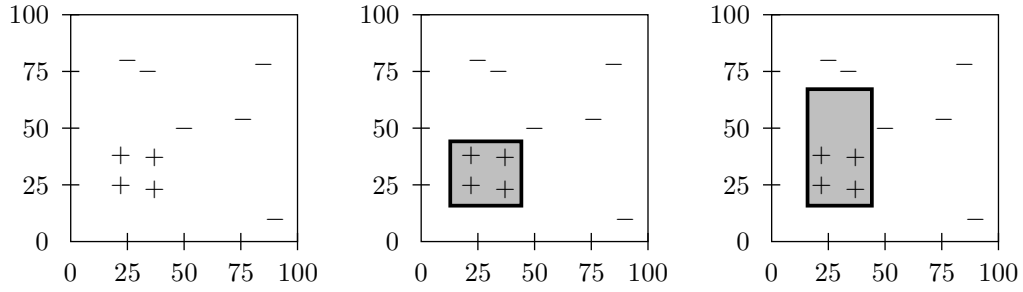


Figure 11.1: Left: positive and negative instances. Middle: a hypothesis that is consistent with the data. Right: another consistent hypothesis.

In machine learning, the hypotheses have a certain format. This format often follows from the data structure that is selected to represent the hypothesis. Let us say here that something is a hypothesis, if it is a closed rectangle within the hundred-field $[0, 100] \times [0, 100]$. More in particular, let us say that H is a hypothesis if and only if $H = [a, b] \times [c, d]$, where $a < b$, $c < d$ are integers between (or equal to) 0 and 100. Thus, a rectangle qualifies as a hypothesis only if its boundary lies on the grid of integers. The set of all possible hypotheses is called the *hypothesis space*, and is written $\mathcal{H}$. Thus,

$$\mathcal{H} = \{[a, b] \times [c, d] \mid a, b, c, d \in \{0, 1, \ldots, 100\}, \text{ and } a < b, c < d\} \tag{11.2}$$

Some hypotheses perform better than others. Performance is expressed in terms of coverage, matching and accuracy.

Classification Every hypothesis classifies instances either positive or negative. The hypothesis (rectangle) $H = [20, 30] \times [20, 30]$, for instance, classifies eight examples as negative and two examples as positive.

Match A hypothesis matches the data for better or for worse. The match indicates how well a hypothesis matches, or classifies, the data.

$$match(H) =_{Def} \frac{\text{instances correctly classified}}{\text{total number of instances (in the data)}} \tag{11.3}$$

If $match(H) = 1$, then H classifies all examples correctly; if $match(H) = 0$, then H classifies all examples incorrectly. (Also useful.) Two out of ten examples are classified incorrectly by $H = [20, 30]^2$, because $(37, 23)$ and $(37, 37)$ are classified incorrectly as negative. Hence, $match(H) = 8/10$.

Consistency A hypothesis is *consistent* with the data if it classifies all instances correctly, i.e, if $match(H) = 1$.

Coverage An instance I is said to be *covered* by a hypothesis if that hypothesis classifies I as positive.

$$coverage(H) =_{Def} \text{ total number of data-instances covered} \tag{11.4}$$

The coverage of a hypothesis may change, depending on whether new instances are classified positive.

Range The range of a hypothesis is equal to the total number of instances that are covered by it, regardless whether the instances occur in the data or not.

$$range(H) =_{Def} \text{ total number of instances (seen and unseen) covered} \tag{11.5}$$

The range of a hypothesis never changes, even in the presence of additional instances.

Accuracy A hypothesis may be more or less accurate for the instances it covers.

$$accuracy(H) =_{Def} \frac{\text{number of positive data-instances covered}}{\text{total number of data-instances covered}} \tag{11.6}$$

The accuracy of a hypothesis indicates its performance on the data it covers. Both examples covered by $H = [20, 30]^2$ are classified correctly. Hence, $accuracy(H) = 2/2 = 1$.

There is always a tradeoff between accuracy and coverage. Accuracy can be improved by reducing the range (= coverage) of an hypothesis. An extremely accurate hypothesis can always be obtained by identifying it with a particular instance. However, such an hypothesis can only be applied to that particular instance and will be of little use to classify future instances. Conversely, if we enlarge the range of an hypothesis, it can be applied to a larger number of unseen instances, but the downside of such hypotheses is that their accuracy often decreases to a degree below of what is acceptable.

Specificity Hypothesis H_1 is said to be *as specific as* hypothesis H_2 if all instances that covered by H_1 are covered by H_2 as well. In that case, H_2 is said to be *as general as* hypothesis H_1. Often, the notion of specificity is used to bring some order in the hypothesis space so that the search for a hypothesis proceeds from general to specific or vice versa.

The above terminology suffices to proceed with the main objective of this chapter, namely, to formulate algorithms that learn rules from cases.

PROBLEMS (Sec. 11.2)

1. How many hypothesis does $\mathcal{H}$ specified by Eq. (11.2) contain? (Solution.)
2. What is the most general hypothesis in $\mathcal{H}$? (Solution.)

(a) Is it unique? (Solution.)
(b) How many instances does it cover? (Solution.)
(c) How many instances are classified correctly? (Solution.)
(d) How well are the instances matched by the most general hypothesis? (Solution.)
(e) What is the accuracy of the most general hypothesis? (Solution.)

3. What is the most specific hypothesis in $\mathcal{H}$? Same questions as with (2). (Solution.)
4. Let H be the hypothesis $[20, 60]^2$. Same questions as with (2). (Solution.)
5. Formulate a hypothesis H that is consistent with the data as provided by (11.1). (Solution.)
6. Formulate a most general hypothesis H that is consistent with the data. Is it unique? (Solution.)
7. Formulate a most specific hypothesis H that is consistent with the data. Is it unique? (Solution.)
8. Explain why a hypothesis H_1 with $match(H_1) = 0$ can be more useful than a hypothesis H_2 with $match(H_2) = 1/2$.
9. Usually, the accuracy of a hypothesis increases if it is made more specific. Show with an example that, in some cases, the accuracy of a hypothesis may decrease even if it is made more specific.

11.3 Learning one rule

In this section we describe two ways of learning a rule. The first algorithm is simple but inefficient. The second algorithm is more sophisticated and solves all mentioned shortcomings of the first algorithm.

An exhaustive algorithm

Suppose the following five cases:

1.	a, c, b, d	4.	$\neg a, \neg b$
2.	b, a, d	5.	$a, \neg d$
3.	$b, c, \neg d$		

and suppose that we would like to learn a rule for d on the basis of these five cases. The idea is to begin with a rule with an empty antecedent, and gradually make the antecedent more specific until the rule covers all positive instances (Case 1,2) and no negative instances (Case 3,4,5). (Case 4 is considered negative here as well, since d is absent.) Other approaches abstain in such cases, i.e., other approaches do not classify cases in which d and $\neg d$ do not occur.) Thus, the algorithm moves from general to specific.

The rule for d with the empty antecedent is

$$\rightarrow d.$$

This rule covers all positive examples, but also three negative examples, viz. 3,4 and 5. One way to avoid the negative examples is to make the antecedent more specific. This can be done in five different ways, since five other relevant literals occur in the data, viz. a, b, c, $\neg a$ and $\neg b$. (Literal $\neg c$ does not occur in the data and, hence, does not need to be taken into account.) Thus, the five possible one-step specializations of the rule "$\rightarrow d$" are

Specialization	*Performance*	*Action*
$a \rightarrow d$	covers negative example nr. 5	make antecedent more specific
$b \rightarrow d$	covers negative example nr. 3	make antecedent more specific
$c \rightarrow d$	violates positive example nr. 2	remove rule
$\neg a \rightarrow d$	violates positive example nr. 1	remove rule
$\neg b \rightarrow d$	violates positive example nr. 1	remove rule

Of the specializations thus obtained, some must be improved (viz. the first two), while others become useless (viz. the last four). The antecedents of the rules that perform better are further specialized. For the moment, let us neglect all specializations of "$b \rightarrow d$," and focus on all specializations of "$a \rightarrow d$":

Specialization	*Performance*	*Action*
$a, b \rightarrow d$	covers all positive examples and avoids all negative examples	keep rule
$a, c \rightarrow d$	violates positive example nr. 2	remove rule
$a, \neg b \rightarrow d$	violates positive example nr. 1	remove rule

The (antecedent of the) rule "$a, b \rightarrow d$" is general enough to cover the positive instances 1 and 2, yet specific enough to miss the negative instances 3,4 and 5. So this is a good hypothesis. The remaining two rules are dropped because they miss a positive instance. The algorithm goes on (eventually producing another consistent hypothesis $b, a \rightarrow d$) but we stop here. The entire algorithm is listed as Algorithm 8. When this algorithm is implemented and applied to the above

Algorithm 8 Exhaustive algorithm to learn all most-general consistent rule antecedents

Input: P, a list of positive instances
Input: N, a list of negative instances
- set O to [null-hyp]; # list of open hypotheses, i.e., hypotheses that can be improved
- set C to [ ] # will contain closed hypotheses, i.e., hypotheses that cannot be improved
while O has elements **do** # there are hypothesis that can be improved
  - replace each h in O by all one-step specializations of h
  for each h in O **do**
    if h misses (i.e., fails to cover) some member of P **then** # h is too specific
      - remove h from O, and move on to the next element in O
    else if h misses every member of N **then** # h is consistent
      - remove h from O; # since it does not need to be further specialized
      - add h to C, provided h is not more specific than some element in C # otherwise, h would not be a real contribution
  # if, at this point, O contains hypothesis, these hypothesis are too general and must be specialized
O is now empty
- **return** C;

five instances, it produces an output as displayed in Fig. 11.2 on the facing page.

A pleasant property of Algorithm 8 is that it is complete: it finds all most-general hypotheses that are consistent with the data. But the algorithm also possesses a number of less pleasant properties:

1. It explores all possible specializations of all rule antecedents, which leads to a combinatorial explosion of the search space. (Exercise 11.3.)
2. The algorithm produces no hypotheses if the data is inconsistent, e.g., if the data contains errors. See Fig. 11.3 on page 224, 2nd diagram.
3. The algorithm produces bad hypotheses if the data is noisy. See Fig. 11.3 on page 224, 3rd diagram.

```
== Positive for d: ====================================================
Sit1: a, b, c, d
Sit2: a, b, d
== Negative for d: ====================================================
Sit3: ~d, b, c
Sit4: ~b, ~d, ~a
Sit5: a, ~d
========================================================================
Specializing Hyp0: (the null hyp) => 4 new hypotheses (total 5).
Hyp1: a (from Hyp0) covers -Sit5: a, ~d => make it more specific.
Hyp2: b (from Hyp0) covers -Sit3: ~d, b, c => make it more specific.
Hyp3: c (from Hyp0) violates +Sit2: a, b, d => remove it.
Hyp4: ~a (from Hyp0) violates +Sit1: a, b, c, d => remove it.
Hyp5: ~b (from Hyp0) violates +Sit1: a, b, c, d => remove it.
Specializing Hyp1: a (from Hyp0) => 3 new hypotheses (total 9).
Specializing Hyp2: b (from Hyp0) => 3 new hypotheses (total 13).
Hyp10: a, b (from Hyp2) is consistent with all examples
Remember Hyp10 as a good hypothesis.
Hyp11: b, c (from Hyp2) violates +Sit2: a, b, d => remove it.
Hyp12: b, ~a (from Hyp2) violates +Sit1: a, b, c, d => remove it.
Hyp13: ~b, b (from Hyp2) violates +Sit1: a, b, c, d => remove it.
Hyp6: a, b (from Hyp1) is consistent with all examples
However, Hyp10 was discovered earlier and is as general as Hyp6,
so it makes no sense to further specialize Hyp6.
Hyp7: a, c (from Hyp1) violates +Sit2: a, b, d => remove it.
Hyp8: a, ~a (from Hyp1) violates +Sit1: a, b, c, d => remove it.
Hyp9: ~b, a (from Hyp1) violates +Sit1: a, b, c, d => remove it.
All hypothesis are now made as specific as possible.
=== Final hypothesis list: ============================================
Hyp10: a, b (from Hyp2)
========================================================================
```

Figure 11.2: Output of exhaustive general-to-specific (EGS)

4. Algorithm 8 produces all hypothesis that are consistent with the data where in practice we often want the *best* hypothesis, whether or not it is consistent with the data. (If it is inconsistent, this is due to the data and not to the hypothesis itself, since it is impossible to produce a consistent hypothesis on the basis of inconsistent data.)

These disadvantages make that the exhaustive algorithm is merely of theoretical interest. The next algorithm (presented below) is heuristic, and meets all shortcomings mentioned above.

PROBLEMS (Sec. 11.3)

1. Suppose 50 different literals occur in a data file, an suppose we would like to learn a rule for the 50th literal z, say.

   (a) How many different specializations of the rule "$\rightarrow z$" are possible? (Solution.)

   (b) What is the size of the search tree if every antecedent is specialized to 48 literals eventually? (Solution.)

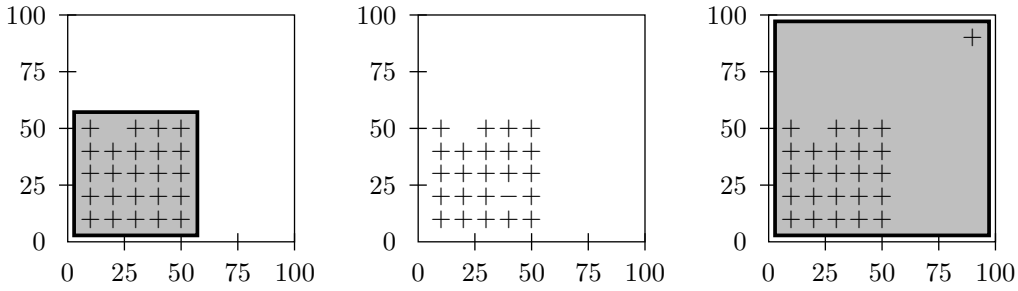


Figure 11.3: Left: data that can be covered by a consistent hypothesis; Middle: data "polluted" with a noisy instance (possibly an error); hence, no consistent hypothesis; Right: noisy positive instance (error?) → forces a general hypothesis. (Hypothesis space 11.2 on page 219 is assumed.)

A heuristic algorithm

The second algorithm does roughly the the same as the first algorithm, with the following differences:

1. Rather than checking for consistency, a *score* is computed for every open hypothesis. There are different score definitions for different purposes, but for the moment the definition

   $$score(H) =_{Def} match(H)$$

   (i.e. the ratio of observed instances that are classified correctly) is fine.
2. Not all open hypothesis are further specialized but only the b best scoring open hypothesis are further specialized. The others are dropped. The number b is fixed number, and is called the *beam-size*. This strategy ensures that the collection of open hypotheses does not grow exponentially, but is always of size b.

The heuristic algorithm begins with the most general rule antecedent as well. This time we do not check whether individual data-instance is classified correctly, but simply compute the score of that particular rule antecedent. We then specialize all antecedents that are open for improvement, and compute their score. An example of this process is displayed in Fig. 11.4 on the facing page. Like Alg. 8, the search starts with the rule with the empty antecedent "$\rightarrow z$". A score is computed and from the figure we can see that the most general hypothesis "$\rightarrow z$" matches 40% of the instances. Then "$\rightarrow z$" is specialized in the same manner as with Alg. 8 on page 222, namely, by adding one literal (does not matter which literal). We see that all (visible) specializations of "$\rightarrow z$" score better than their parent, so search is continued on these hypotheses, provided that the beam-size ≥ 3. (If the beam-size is 2, for instance, then the branch starting with "$c \rightarrow z$" would not be searched.) If every specialization scores less than the parent hypothesis, then the parent apparently cannot be improved, and is put in the list of closed hypothesis. This is the case with "$\neg a, c \rightarrow z$". Search continues until all open hypothesis are closed. Then the best-scoring hypothesis is returned. The entire algorithm is described on page 226.

Algorithm 9 uses a *heuristic* to guide the search. This means that the algorithm assumes that the best hypothesis is a specialization of one of the b best-scoring hypothesis. This assumption is sometimes false. It may happen, for instance, that a dropped hypothesis could have been specialized into a more specific one that is as general but better scoring than the ultimate hypothesis returned by Algorithm 9. This is the price to be paid for carrying out a memory-friendly heuristic search rather than a memory-exhaustive complete search. (Compare this to inviting b applicants for a job interview on the basis of l letters of application ($b < l$). There is a chance you miss out on the best candidate because he/she is not on the top-b list

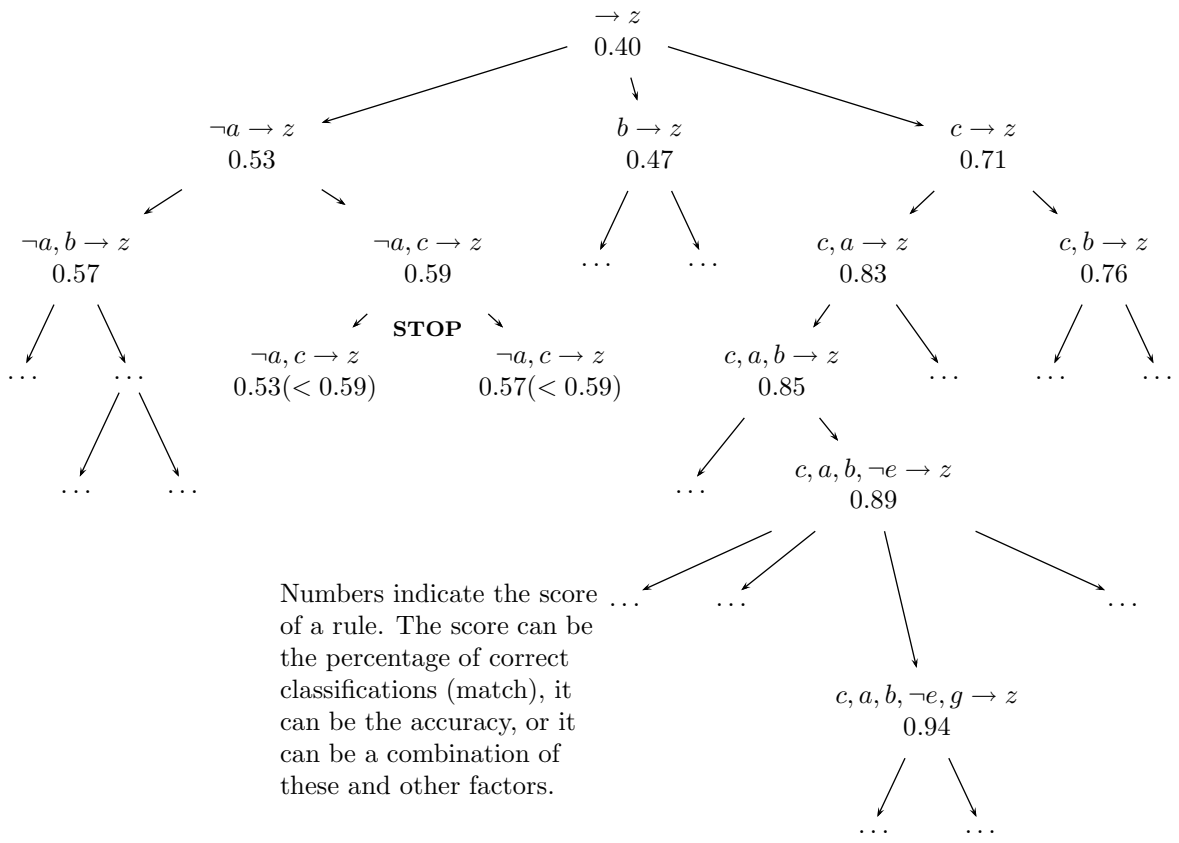


Figure 11.4: Learning one rule by making the most promising antecedent more specific.

because he/she wrote a bad letter.) Experimental studies fortunately suggest that heuristic search almost always leads to the right hypothesis.

Another property of Algorithm 9 is that the final hypothesis does not need to be consistent with the data. This is not necessarily a disadvantage, since it is impossible to produce a consistent hypothesis on the basis of inconsistent data (Fig. 11.3 on the preceding page, 2nd diagram). A best-scoring hypothesis always exists, and the heuristic algorithm will almost always find such a hypothesis. Algorithm 9 also performs well if the data contains noise (Fig. 11.3 on the facing page, 3rd diagram), especially if covering a noisy positive instance would enforce the inclusion of many negative instances. (Why?)

11.4 Learning sets of rules

A disadvantage of the hypotheses space used by Algorithm 8 (page 222) and Algorithm 9 (page 226), is that a hypothesis is identical to precisely one rule, or rule antecedent. In many situations, however, merely one rule antecedent does not suffice to classify all instances accurately. To show why, let us get back to the situation outlined in Section 11.2 on page 219, where instances are points in the plane, and hypotheses are closed rectangles. If the instances are distributed as displayed in Fig. 11.6 on page 229, then the exhaustive algorithm (page 222) would produce no hypothesis, while the heuristic algorithm (page 226) would produce an hypothesis that matches at best 69% of the instances. The second diagram in Fig. 11.6 on page 229 suggests what to do: use a more refined hypothesis representation, in which a hypothesis can be a disjunction (or union) of rule antecedents.

Algorithm 9 Batch-processing of instances to learn the best matching rule antecedent. Search is heuristically guided by the score of a hypothesis.

Input: P, a list of positive instances
Input: N, a list of negative instances
Input: *SCORE*, a subroutine that computes the quality or performance of an hypothesis
Input: b, an integer ≥ 1, representing the beam-size
1: - set O to [null-hyp];
2: - set C to [ ]; # no hypothesis closed, yet
3: **while** O has elements **do**
4: - set S to [ ]; # will contain specializations of O that score better than their parent
5: **for** each h in O **do**
6: S_h := all better-scoring one-step specializations of h
7: **if** S_h is empty **then** # h cannot be improved
8: - put h in C if $h \notin C$
9: **else**
10: - augment S with S_h (remove duplicates, if necessary)
each h in O has now been either closed or specialized
11: - set O to the b best-scoring elements of S # so that O is reduced to a manageable size
O is now empty
12: - **return** the best-scoring element of C;

Learning disjunctions does not require much additional programming effort. Any algorithm can be used to generate the elementary hypotheses (i.e., rule antecedents). Since we have Algorithm 9 (the heuristic algorithm), we use that one. The only thing we will have to beware of is that elementary hypotheses (i.e. rectangles) must be assessed differently if they are used as parts of disjunctive hypothesis. The value of an elementary hypothesis is no longer determined by its performance on all data (Eq. 11.3 on page 220), but rather by its accuracy on the data it covers (Eq. 11.6 on page 220). Other hypotheses, then, are responsible for covering the remaining positive instances. From this observation it would follow that an appropriate score function for hypotheses that are going to be used in a disjunction is their accuracy (Eq. 11.6 on page 220). However, this is too simple an assumption. The problem of this approach is that a hypothesis can be always made more accurate by specializing it, downright to the case in which every rule covers precisely one positive instance. This is not the way to go, however, since such a set of rules is of no use classifying future instances. The problem is that every future instance will be classified as negative, unless it equals a positive instance. This phenomenon is called *overfitting*.

Overfitting is a term from statistics, adopted by the machine learning community. Examples of overfitting:

1. *Given:* A teacher draws a set of points on a piece of paper, approximately lying on a straight line
2. *General concept that lies at the basis of what is manifest:* A straight line.
3. *Overfitting:* Student draws a polygon (line with bends) that goes through all points. This relatively complex solution neglects the more simple notion of straight line.

a. *Given:* Three red objects, shown by parent to 2-year old child.
b. *General concept that lies at the basis of what is given:* The concept "red".
c. *Overfitting:* Child rejects all other red objects as being red, because it believes that "redness" only holds for the three objects that were indicated by the parent.

What happens with overfitting, is that the pupil (or, more generally, the learning algorithm) focuses too much on the examples given, and neglects that there might be a general principle that lies underneath given the examples. The examples are learnt "too well," at the expense of the general structure that is suggested by the hypothesis-representation. Thus, in machine

learning there is always a tradeoff between specializing on the examples on the one hand, and enforcing a more general hypothesis representation on the other hand.

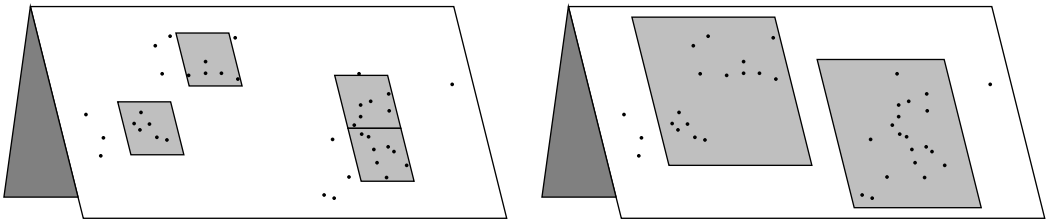


Figure 11.5: Tradeoff in choosing a disjunctive hypothesis. Left: precise but complex covering; Right: simple but imprecise covering.

Here is a practical analogy: Suppose an electric sun-collector has been damaged by hail. In fact, hail stones have damaged some of the collector cells. The sun-collector has about 100,000 of such cells. To avoid a short-circuit in times of rain, most (but not necessarily all) cells must be covered by plastic tissue which is cut in the form of rectangles. A technician is willing to repair the sun-collector. His rate is 50,= Euro for the first patch plus 10,= Euro for every additional patch. This suggests that the application of a small number of large tissues is the best solution. However, spots where tissue is applied do no longer produce electricity. Every cell that is covered but otherwise undamaged, costs 4.68 Euro on a yearly basis. Thus, from the viewpoint of generating electricity, applying several small tissues would be better. This indicates that there is a tradeoff between simplicity and precision. A simple solution with one or two tissues is cheap but diminishes the output of electricity. A complex solution with many small applications maintains electricity output but is expensive. The same holds for formulating a disjunctive hypothesis. There is a tradeoff between an exact covering with many specific (and complex) hypothesis and global covering with only a few simple general hypotheses. The first solution has the advantage of exactness, but the price that has to be paid is overfitting: the hypotheses are tailored at the data, do not represent a global concept, and have little predictive power. The second solution has the advantage of simplicity, but misclassifies a considerable number of observed instances. As so often, the best solution lies somewhere in the middle of two extremes.

Algorithm 10 Batch-processing of instances to learn a set of rules — sequential covering

Input: P, a list of positive instances
Input: N, a list of negative instances
Input: $SCORE'$, a subroutine that computes the performance of an elementary hypothesis
Input: l, a lower bound for what is acceptable as a score
1: - set R to [ ];
2: **while** P has elements **do**
3: - let r be the rule that is produced by Algorithm 9 with parameters $SCORE'$, P and N
4: - leave the while-loop if $accuracy(r) \geq l$;
5: - put r in R;
6: - remove all members of P that are covered by r;
7: - **return** (R, P); # P contains all instances not covered by R

To construct a solution, let

$$accuracy'(H) =_{Def} \begin{cases} accuracy(H) & \text{if } accuracy(H) \text{ is defined,} \\ 0 & \text{otherwise.} \end{cases} \tag{11.7}$$

This ensures that the accuracy of a hypothesis is always defined, so that the algorithm never

stops because of a zero-division. Further, let

$$score(H) = a * accuracy'(H) + b * range(H) + c * coverage(H) \tag{11.8}$$

where $0 \leq a, b, c$ are parameters to regulate the importance, or weight, of aspects of hypotheses such as **a**ccuracy, range (**b**earing), and **c**overage. The definition of range and coverage are given at page 220.

The range of a rule (Definition 11.5 on page 220) may be estimated as follows. With n different proposition letters, for instance $n = 4$ and $L = \{a, \neg a, b, \neg b, c, \neg c, d, \neg d\}$, it is possible to form 3^n different cases, since every proposition letter offers three choices: include the proposition letter in the situation description, include its negation, or include none of them. (For example, L gives rise to $3^4 = 81$ possible cases.) All rules with an empty antecedent cover all cases. All rules with one literal in the antecedent cover one third of all cases, which is 3^{n-1}. In general, all rules with k literals in the antecedent cover 3^{n-k} cases, which means that the range of a rule is divided by three every time the antecedent is specialized by means of one literal. Thus, the range of a rule is proportionally equal to

$$1/3^{|\text{length of the antecedent}|}. \tag{11.9}$$

Now the parameter b can be used to regulate the impact of this quantity on the overall score of a rule antecedent.

Tuning a, b, and c has different effects, which is summarized in the following table:

a	b	c	*rule set produced*
large	small	small	overfitting: many rules, most of them with large (i.e., specific) antecedents; hence, many future instances will be classified negative
small	large	small	few rules, most of them with small (i.e., general) antecedents; many positive instances from the data remain uncovered
small	small	large	few rules, some of which are specific; most of the positive instances from the data are covered

Algorithm 10 on the preceding page works as follows: apply Algorithm 9 (the heuristic general-to-specific algorithm) with the new score function (given by Eq. 11.8) to the set of positive instances P. This yields a rule r_1. Remove all elements that are covered by r_1 from P, and apply Algorithm 9 again. This yields a rule r_2. Repeat this process, until the accuracy of rule r_i falls below some lower bound l. If $l = 0$, then Algorithm 10 goes all the way and may end with an inaccurate rule of inference. (which is not necessarily bad, since all positive examples are covered, be it that some positive instances are covered with inaccurate and/or extremely specific rules). If $l > 0$, then the algorithm ends with $R = \{r_1, \ldots, r_{i-1}\}$ and P containing all positive instances not covered by R. The indexes run to $i - 1$ instead of i, because rule r_i is the last rule that is made by the algorithm. It is not included in R, though, because it is the first rule for which the accuracy is less than l. If $l \rightsquigarrow 1$, R will contain fewer rules.

11.5 Learning a complete set of rules

If you do not know which rules you want, for example, if you try to discover general patterns in the data, then generally you do not know which features exhibit regularities and which not. In such cases it might be a good idea compute rules for *all* literals.

Algorithm 11 on the facing page describes how to produce rules for all literals that possess a certain accuracy.

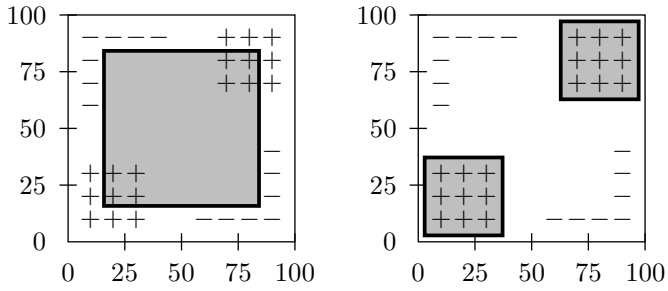


Figure 11.6: Left: best-scoring hypothesis (from hypothesis space defined at Eq. 11.2 on page 219), scores $(8 + 14)/(18 + 14) = 0.69$; Right: better hypothesis but outside hypothesis space, scores 1.00.

Distilling patterns, rules and and regularities from data without knowing what is important is a form of *unsupervised learning*: only with hindsight we can tell which (combinations of) literals exhibit regularities, and which (combinations of) literals do not. For example, for a large data set with literals $\{a, \ldots, z\neg a, \ldots \neg z\}$, the last algorithm may produce

Algorithm 11 Batch-processing of instances to learn all rules for all literals

Input: P, a list of positive instances
Input: N, a list of negative instances
Input: l, a lower bound for rule accuracy
1: - set R_{all} to [ ];
2: **for** each literal L that occurs in the data **do**
3: - augment R_{all} with all rules produced by Algorithm 10 with target literal L;
4: - **return** R_{all}; # all rules are as accurate as l

$$\begin{aligned}
t, \neg h, i, \neg s &\quad \text{–(0.98)}\rightarrow \quad b \\
\neg i, \neg s &\quad \text{–(0.87)}\rightarrow \quad y \\
t, \neg h, e &\quad \text{–(1.00)}\rightarrow \quad e \\
\neg e, n, d &\quad \text{–(0.94)}\rightarrow \quad b \\
\neg o, \neg f &\quad \text{–(0.91)}\rightarrow \quad y \\
\neg t, \neg h, e &\quad \text{–(1.00)}\rightarrow \quad e \\
\neg s, e, \neg c, t, \neg i, \neg o, n &\quad \text{–(0.96)}\rightarrow \quad b,
\end{aligned}$$

assuming that $l = 0.80$. It is only *after* the application of this algorithm, however, that we learn that b, y, and e are the only literals for which learning made sense.

11.6 Rule parametrization

Rule strength and degree of belief (DOB) are important rule properties when a rule is used (rather than created).[2] For example, if

$$\text{DOB}(r_1) = 0.45 \text{ where } r_1 = \text{“}x, y \text{ –(0.91)}\rightarrow z\text{”}$$

then r_1 has DOB 0.45 and strength 0.91. Roughly, the rule strength indicates the certainty with which a rule antecedent implies the consequent, while the support of a rule indicates the

[2] Chapter 8 on page 149.

credibility of a rule in its entirety. (See also Table 11.1.) In this section it is explained how, once that rule is learnt, its strength and support can be defined in terms of accuracy and coverage, respectively.

Strict rules with strict support. Strict rules a-priori, such as: "all circles are round," "the law is the law," "every bachelor is unmarried," as well as inductive rules a-posteriori with massive affirmation: "all restaurants are open between 7-9 PM".	*Strict rules with defeasible support.* Strict rules a-posteriori, for which no counterexamples have been found: "all clerks are nice" (in the sense: "every clerk I've met, was nice"), "all apples are round" (in the sense: "until now, I didn't see apples that were not round).
Defeasible rules with strict support. Defeasible rules a-priori: "rolling a dice usually doesn't produce a six," "most rectangles are not squares (if length and breadth are random variables that are uniformly distributed over $\{1, 2, 3, 4, 5\}$)," as well as inductive rules a-posteriori with massive affirmation: "most birds fly".	*Defeasible rules with defeasible support.* Typically rules a-posteriori: "I've met some boxers, and, apart from one exception, most of them were kind". With enough confirmation, such rules may become strictly supported.

Table 11.1: Rule modalities

How accuracy determines, or at least *indicates*, rule strength can be seen by observing that accuracy is defined as the correct classification ratio among observed instances that are covered. The latter does not say much about the correct classification ratio among observed *and* unobserved instances that are covered by the rule, but since observed instances are by definition all the instances we have, it is reasonable to expect that the rule accuracy will be about the same on unseen instances. (In addition, probability theory is able to indicate the standard deviation of such an estimation, but that would carry us too far here—all we need is one number to fill the slot "rule strength".) Thus, after rules have been learnt, it is reasonable to set rule strength equal to the accuracy of a rule.

How coverage relates to support is less obvious, since coverage is concerned with the range of a rule on seen instances, while support is concerned with the overall credibility of a rule. So what makes a rule more credible than other rules? A simple answer is: by the number of instances that were used to define, or create, the rule. If many instances were used to create a rule, then it reasonable to attach a firm belief to this rule. [Seeing $9,000,000$ people and observing that 49% is male, for example, justifies a high DOB for the rule $human(X) \dashrightarrow_{(0.49)} male(X)$.] Conversely, if few instances were used to create a rule, then it is reasonable to attach a small belief, or degree of support, to this rule in its entirety. However, this approach comes with at least two problems. The first problem is that "the number of instances that were used to create a rule" is an ill-defined concept. Is it the total number of positive and negative instances that were used in the process to form the rule, *or* is it the total number of positive and negative instances that are covered by the rule once it is created?

As long as R covers unseen instances, we have $coverage(R)/range(R) < 1$. In such cases, we cannot be sure about R's true accuracy, because unseen instances in the range of R may change the sampled accuracy, and the latter is the only indication we have for the true rule accuracy.

We may only know the true accuracy of R if all instances in R's range are seen, i.e., if $coverage(R)/range(R) = 1$. In that case, we may read the true accuracy from the sampled accuracy. In all intermediate cases, $coverage(R)/range(R)$ is a fair indicator of the credibility of R's accuracy. Probability theory is able to indicate the confidence interval of the true accuracy, given the coverage and given the sampled accuracy. However, that would carry us too far here—all we need is a number to indicate the credibility of the rule. For the moment, let us say that the credibility of the sampled accuracy—the DOB of R—is equal to the square root of $coverage(R)/range(R)$.

On page 228 it was indicated that $range(R) = 3^{n-k}$, where k is the length of the antecedent of R, n the number of different proposition letters. Thus

$$\begin{aligned}\text{DOB}(R) &= \text{credibility of}(R)\\ &= \text{credibility of sampled accuracy of}(R)\\ &= \sqrt{coverage(R)/range(R)}\\ &= \sqrt{coverage(R)/3^{n-k}}\\ &= 3^{(k-n)/2}coverage(R)\end{aligned}$$

where $coverage(R)$ is the number of data elements that are covered by R.

PROBLEMS (Sec. 11.6)

1. (Learning one rule.) Consider the following fifteen instances:

$$\begin{array}{lll} a,b,c_1,x & a,b,c_6,x & \neg a,b,c_{11},x\\ a,b,c_2,x & a,b,c_7,x & \neg a,b,c_{12},x\\ a,b,c_3,x & a,b,c_8,x & a,\neg b,c_{13},x\\ a,b,c_4,x & a,b,c_9,y & a,\neg b,c_{14},y\\ a,b,c_5,x & a,b,c_{10},\neg x & \neg a,\neg b,c_{15},\neg x \end{array} \tag{11.10}$$

   We use rules to classify cases for x. The score of a rule R is determined by the following formula:

$$score(R) = a * accuracy'(R) + b * range(R) + c * coverage(R) \tag{11.11}$$

   (a) Suppose $a = 1$, $b = 0$ and $c = 0$. Apply the heuristic general-to-specific (HGS) algorithm to learn the best scoring rule for x. (Solution.)

   (b) Same question with $a = 1$, $b = 0$ and $c = 0.023$. (Solution.)

   (c) Suppose $a = 1$ and $b = 0$. For which values of c does the search yield $a, b \rightarrow x$ and $b, a \rightarrow x$? (Solution.)

11.7 Learning first-order rules

In the previous sections we discussed algorithms for learning sets of propositional (i.e., variable-free) rules. In this section, we consider learning rules that contain variables. Our motivation for considering such rules is that they are more expressive than propositional rules. Inductive learning of first-order rules is often referred to as *inductive logic programming* (ILP), because this process can be seen as automatically inferring Prolog programs from instances.

Learning rules from instances that are formulated in the language of first-order logic is not much harder than learning rules from in the language of propositional logic. All methods and

techniques from propositional rule learning carry through to the first-order case. In particular, the exhaustive general-to-specific algorithm (EGS) and the heuristic general-to-specific algorithm (HGS) remain applicable.

A method from propositional rule learning that does *not* carry through to the first-order case, is specializing the antecedent. In the propositional case, a rule antecedent can be specialized by a literal that occurs in the data but does not occur in the antecedent. Since there are a restricted number of such literals, the number of alternative specializations remains within reasonable bounds. In the first-order case, the situation is different. Here, literals are predicates or equalities that may contain terms. Since terms may by arbitrarily large (think of $f(x), f(f(x)), f(f(f(x)))$, the number of alternative specializations may increase exponentially and worse. Therefore, almost all practical algorithms for learning sets of first-order rules works with a first-order language without function symbols. In this way, the number of alternative specializations again remains within reasonable bounds.

When predicates may contain variables, an antecedent can be specialized by predicates and equalities containing variables or constants (but no compound terms). More precisely, suppose the current rule being considered is

$$R: \ l_1, \ldots, l_n \rightarrow px_1, \ldots, x_k$$

where $l_1, \ldots, l_n$ are literals that form the rule antecedent (body) and where $px_1, \ldots, x_k$ is the literal that forms the consequent (head). There are different ways to specialize the antecedent:

1. Add a predicate $qy_1, \ldots, y_m$, where q occurs in the data, and $y_1, \ldots, y_m$ are new variables, or variables that are already present in the rule, such that at least one of the y-variables already occurs in R.
2. Add an equality of the form $x = y$, where both x and y already occur in R.
3. Add the negation of either (1) or (2).

To select the most promising such literal, we will have to compute the score of the extended, or specialized, rule over the test data. This is done by instantiating R in all possible ways with all possible variable bindings for all variables in R. This normally would give a huge amount of rule instantiations, but since we are working with a first-order language without function symbols, and since the data set contains a limited number of constants, this is doable. The next step, then, is to compute the score of all instantiations of R. The score of R itself is then equal to the average of the score of all instantiations.

11.8 Other ways to learn rules

In this chapter, we have presented a number of specific algorithms, that differ in some aspects from other rule learning algorithms. But there are other ways to learn rules.

Some of the choices are:

1. *Batch vs. real-time processing of instances.* Also known as off-line vs. on-line, or as non-incremental vs. incremental processing of instances. In this chapter, we have chosen for batch processing of instances.

2. *General-to-specific vs. specific-to-general development of hypotheses.* In this chapter, we have chosen for moving from general to specific hypotheses. A combination of both search methodes is also possible.

3. *Sequential covering vs. exception-to-exception.* In this chapter, we have chosen for sequential covering.

4e, p. 213: Ronald's explanation

$$E = \text{too-late} \leftarrow\!(0.80)\!- \text{walk} \leftarrow\!(1.00)\!- \text{flat-tire } [0.50]$$

is defeated by

$$F = \text{flat-tire} \leftarrow\!(1.00)\!- \text{missed-the-bus } [0.75].$$

(When you missed the bus, you cannot be late because of a flat tire.)

5a, p. 214: A possible solution in the style of propositional logic is displayed in Table B.1 on the next page. a possible solution in the style of first-order logic is displayed in Table B.2. (Table B.1 could be extended with rules that express that a payment of a particular amount excludes payments of other amounts. This does not have to be specified in the first-order framework, because it follows from the fact that the "pay"-function assumes a unique value.

5b, p. 215: An explanation for a nett profit of \$190,000 in the propositional setting is

$$\begin{array}{l} e = \text{nett190} \leftarrow\!(1.00)\!- \text{cost10} \leftarrow\!(1.00)\!- \text{test } [1.00] \\ \qquad\qquad\qquad\qquad\quad \text{pay200} \leftarrow\!(1.00)\!- \text{soaked } [0.20] \\ \qquad\qquad\qquad\qquad\qquad\qquad\quad \text{drill } [1.00] \end{array}$$

The first-order version looks like

$$\begin{array}{l} e = \text{nett-profit}(x) = 190 \leftarrow\!(1.00)\!- \text{cost}(x) = 10 \leftarrow\!(1.00)\!- \text{test}(x) = \text{yes } [1.00] \\ \qquad\qquad\qquad\qquad\qquad\qquad\quad \text{pay}(x) = 200 \leftarrow\!(1.00)\!- \text{oil}(x) = \text{soaked } [0.20] \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \text{drill}(x) = \text{yes } [1.00] \end{array}$$

In both cases, "nett-profit$(x) = 190$" receives a DOS of 0.20. The argument e grounds in the assumptions that we have done a seismic test, that we drill, and that the ground is soaked with oil.

Argument e can be defeated, because it grounds in the questionable assumption "oil$(x) =$ soaked". A defeater of e is

$$\begin{array}{l} d = \text{nett-profit}(x) = 40 \leftarrow\!(1.00)\!- \text{cost}(x) = 10 \leftarrow\!(1.00)\!- \text{test}(x) = \text{yes } [1.00] \\ \qquad\qquad\qquad\qquad\qquad\qquad\quad \text{pay}(x) = 50 \leftarrow\!(1.00)\!- \text{oil}(x) = \text{wet } [0.30] \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \text{drill}(x) = \text{yes } [1.00] \end{array}$$

In both cases, "nett-profit$(x) = 40$" receives a DOS of 0.30. (There are other defeaters of e.)

1, p. 220: There are 100 closed integer-bounded intervals $[a, b]$ such that $a < b$ that start with 0, there are 99 similar intervals that start with 1, ..., there is one closed interval that starts with 99. Thus, $[0, 100]$ possesses $1 + \ldots + 100 = 100 \times 101/2 = 5050$ closed intervals. [There are other ways to count the possible number of intervals, for example by first choosing one point of the interval (101 choices) and then choosing another point of the interval, other than the first point (100 choices). We then obtain 101×100 possible intervals. Since we did not distinguish start and end points, we have counted intervals double, so that we have to divide 101×100 by two.]

In all, $[0, 100]^2$ possesses $5050^2 = 25502500$ different integer-bounded closed rectangles with an interior.

2, p. 220: The most general hypothesis is $H = [0, 100]^2$. It covers all instances.

2a, p. 220: The most general hypothesis is unique, at least in the hypothesis space $\mathcal{H}$ defined in 11.2 on page 219.

A priori data:

Proposition	DOB
dry	0.50
wet	0.30
soaked	0.20

Rules to determine cost:

Proposition	DOB
test –(1.00)→ cost10	1.00
¬test –(1.00)→ cost0	1.00

Rules to predict seismic patterns:

Proposition	DOB
dry, test –(0.10)→ closed	1.00
dry, test –(0.30)→ open	1.00
dry, test –(0.60)→ diffuse	1.00
dry, ¬test –(0.33)→ closed	1.00
dry, ¬test –(0.33)→ open	1.00
dry, ¬test –(0.33)→ diffuse	1.00

Proposition	DOB
wet, test –(0.30)→ closed	1.00
wet, test –(0.40)→ open	1.00
wet, test –(0.30)→ diffuse	1.00
wet, ¬test –(0.33)→ closed	1.00
wet, ¬test –(0.33)→ open	1.00
wet, ¬test –(0.33)→ diffuse	1.00

Rules to predict seismic patterns:

Proposition	DOB
soaked, test –(0.50)→ closed	1.00
soaked, test –(0.40)→ open	1.00
soaked, test –(0.10)→ diffuse	1.00
soaked, ¬test –(0.33)→ closed	1.00
soaked, ¬test –(0.33)→ open	1.00
soaked, ¬test –(0.33)→ diffuse	1.00

Rules to determine revenue:

Proposition	DOB
dry, drill –(1.00)→ pay-70	1.00
dry, ¬drill –(1.00)→ pay0	1.00
wet, drill –(1.00)→ pay50	1.00
wet, ¬drill –(1.00)→ pay0	1.00
soaked, drill –(1.00)→ pay200	1.00
soaked, ¬drill –(1.00)→ pay0	1.00

Rules to exclude disjoint attributes:

Proposition	DOB
closed –(1.00)→ ¬open	1.00
closed –(1.00)→ ¬diffuse	1.00
open –(1.00)→ ¬closed	1.00
open –(1.00)→ ¬diffuse	1.00
diffuse –(1.00)→ ¬open	1.00
diffuse –(1.00)→ ¬closed	1.00

Proposition	DOB
dry –(1.00)→ ¬wet	1.00
dry –(1.00)→ ¬soaked	1.00
wet –(1.00)→ ¬dry	1.00
wet –(1.00)→ ¬soaked	1.00
soaked –(1.00)→ ¬wet	1.00
soaked –(1.00)→ ¬dry	1.00

Rules to determine nett profit:

Proposition	DOB
cost0, pay-70 –(1.00)→ nett-profit-70	1.00
cost0, pay0 –(1.00)→ nett-profit0	1.00
cost0, pay50 –(1.00)→ nett-profit50	1.00
cost0, pay200 –(1.00)→ nett-profit200	1.00

Proposition	DOB
cost10, pay-70 –(1.00)→ nett-profit-80	1.00
cost10, pay0 –(1.00)→ nett-profit-10	1.00
cost10, pay50 –(1.00)→ nett-profit40	1.00
cost10, pay200 –(1.00)→ nett-profit190	1.00

Table B.1: Defeasible rules for the oil wildcatters example (propositional version).

A priori data:

Sentence	DOB
$\text{oil}(x) = \text{dry}$	0.50
$\text{oil}(x) = \text{wet}$	0.30
$\text{oil}(x) = \text{soaked}$	0.20

Rules to predict seismic patterns:

Sentence	DOB
$\text{oil}(x) = \text{dry},\ \text{test}(x) = \text{yes} \text{–}(0.10)\!\!\rightarrow \text{seismic}(x) = \text{closed}$	1.00
$\text{oil}(x) = \text{dry},\ \text{test}(x) = \text{yes} \text{–}(0.30)\!\!\rightarrow \text{seismic}(x) = \text{open}$	1.00
$\text{oil}(x) = \text{dry},\ \text{test}(x) = \text{yes} \text{–}(0.60)\!\!\rightarrow \text{seismic}(x) = \text{diffuse}$	1.00
$\text{oil}(x) = \text{dry},\ \text{test}(x) = \text{no} \text{–}(0.33)\!\!\rightarrow \text{seismic}(x) = \text{closed}$	1.00
$\text{oil}(x) = \text{dry},\ \text{test}(x) = \text{no} \text{–}(0.33)\!\!\rightarrow \text{seismic}(x) = \text{open}$	1.00
$\text{oil}(x) = \text{dry},\ \text{test}(x) = \text{no} \text{–}(0.33)\!\!\rightarrow \text{seismic}(x) = \text{diffuse}$	1.00
$\text{oil}(x) = \text{wet},\ \text{test}(x) = \text{yes} \text{–}(0.30)\!\!\rightarrow \text{seismic}(x) = \text{closed}$	1.00
$\text{oil}(x) = \text{wet},\ \text{test}(x) = \text{yes} \text{–}(0.40)\!\!\rightarrow \text{seismic}(x) = \text{open}$	1.00
$\text{oil}(x) = \text{wet},\ \text{test}(x) = \text{yes} \text{–}(0.30)\!\!\rightarrow \text{seismic}(x) = \text{diffuse}$	1.00
$\text{oil}(x) = \text{wet},\ \text{test}(x) = \text{no} \text{–}(0.33)\!\!\rightarrow \text{seismic}(x) = \text{closed}$	1.00
$\text{oil}(x) = \text{wet},\ \text{test}(x) = \text{no} \text{–}(0.33)\!\!\rightarrow \text{seismic}(x) = \text{open}$	1.00
$\text{oil}(x) = \text{wet},\ \text{test}(x) = \text{no} \text{–}(0.33)\!\!\rightarrow \text{seismic}(x) = \text{diffuse}$	1.00
$\text{oil}(x) = \text{soaked},\ \text{test}(x) = \text{yes} \text{–}(0.50)\!\!\rightarrow \text{seismic}(x) = \text{closed}$	1.00
$\text{oil}(x) = \text{soaked},\ \text{test}(x) = \text{yes} \text{–}(0.40)\!\!\rightarrow \text{seismic}(x) = \text{open}$	1.00
$\text{oil}(x) = \text{soaked},\ \text{test}(x) = \text{yes} \text{–}(0.10)\!\!\rightarrow \text{seismic}(x) = \text{diffuse}$	1.00
$\text{oil}(x) = \text{soaked},\ \text{test}(x) = \text{no} \text{–}(0.33)\!\!\rightarrow \text{seismic}(x) = \text{closed}$	1.00
$\text{oil}(x) = \text{soaked},\ \text{test}(x) = \text{no} \text{–}(0.33)\!\!\rightarrow \text{seismic}(x) = \text{open}$	1.00
$\text{oil}(x) = \text{soaked},\ \text{test}(x) = \text{no} \text{–}(0.33)\!\!\rightarrow \text{seismic}(x) = \text{diffuse}$	1.00

Rules to determine cost:

Sentence	DOB
$\text{test}(x) = \text{yes} \text{–}(1.00)\!\!\rightarrow \text{cost}(x) = 10$	1.00
$\text{test}(x) = \text{no} \text{–}(1.00)\!\!\rightarrow \text{cost}(x) = 0$	1.00

Rules to determine revenue:

Sentence	DOB
$\text{oil}(x) = \text{dry},\ \text{drill}(x) = \text{yes} \text{–}(1.00)\!\!\rightarrow \text{pay}(x) = -70$	1.00
$\text{oil}(x) = \text{dry},\ \text{drill}(x) = \text{no} \text{–}(1.00)\!\!\rightarrow \text{pay}(x) = 0$	1.00
$\text{oil}(x) = \text{wet},\ \text{drill}(x) = \text{yes} \text{–}(1.00)\!\!\rightarrow \text{pay}(x) = 50$	1.00
$\text{oil}(x) = \text{wet},\ \text{drill}(x) = \text{no} \text{–}(1.00)\!\!\rightarrow \text{pay}(x) = 0$	1.00
$\text{oil}(x) = \text{soaked},\ \text{drill}(x) = \text{yes} \text{–}(1.00)\!\!\rightarrow \text{pay}(x) = 200$	1.00
$\text{oil}(x) = \text{soaked},\ \text{drill}(x) = \text{no} \text{–}(1.00)\!\!\rightarrow \text{pay}(x) = 0$	1.00

Rule to determine nett profit:

Sentence	DOB
$\text{cost}(x) = \text{u},\ \text{pay}(x) = \text{v} \text{–}(1.00)\!\!\rightarrow \text{nett-profit}(x) = \text{v}-\text{u}$	1.00

Table B.2: Defeasible rules for the oil wildcatters example (first-order version).

2b, p. 220: 10 instances.

2c, p. 220: Only positive instances should be covered. Thus, four instances are classified correctly.

2d, p. 220: Number of instances classified correctly divided by the total number of instances = 4/10.

2e, p. 221: Number of positive instances covered divided by the total number of instances covered = 4/10.

3, p. 221: An (arbitrary) example of a most specific hypothesis is $H = [85, 86] \times [11, 12]$. There are more most specific hypotheses; in fact there are as many such hypotheses as there are unit squares on the $[0, 100]^2$-grid, namely 100×100. Let us proceed with H. This hypothesis classifies all 10 instances as negative, hence six correctly so. Further, H covers 0 instances, so that $match(H) = 6/10$ and $accuracy(H) = 0/0 =$ `undef`.

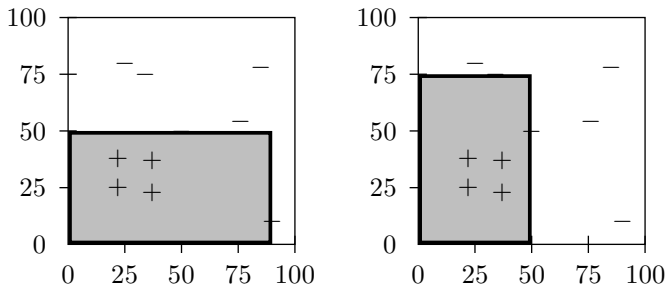
4, p. 221: $H = [20, 60]^2$.

Instance	Class	H's classification
$(22, 25)$	+	+
$(76, 54)$	-	-
$(37, 23)$	+	+
$(37, 37)$	+	+
$(25, 80)$	-	-
$(34, 75)$	-	-
$(85, 78)$	-	-
$(22, 38)$	+	+
$(90, 10)$	-	-
$(50, 50)$	-	+

H classifies 10 instances of which 9 are classified correctly. Thus, $match(H) = 9/10$. Further, H covers 5 instances, of which 4 are covered correctly. Thus, $accuracy(H) = 4/5$.

5, p. 221: $H = [20, 40]^2$ does the job. It covers all positive instances, and no negative instance. $match(H) = 10/10 = accuracy(H) = 4/4 = 1$. Of course, other solutions are possible.

6, p. 221: There are two most general hypothesis that are consistent with the data, viz. $H_1 = [0, 90] \times [0, 50]$ and $H_2 = [0, 50] \times [0, 75]$. Thus, a most general hypothesis that are consistent with the data is not unique.



7, p. 221: $H = [22, 37] \times [25, 38]$ is *the* most specific hypothesis. It is included in every other hypothesis that is consistent with (11.1).

1a, p. 223: An antecedent can be formed by any (possibly empty) combination of the remaining 49 literals. This number is equal to the number of subsets of a set of 49 literals, namely, $2^{49} \approx 10^{29}$. If antecedents are ordered, there are even more possibilities, namely $49 + 49 \cdot 48 + 49 \cdot 48 \cdot 47 + \ldots$.

1b, p. 223: In the first level there are 49 choices. At each node in the second level there are 48 choices, hence, $49 \cdot 48$ choices at the second level. Therefore, the entire search tree contains $49! \approx 10^6 4 > 10^2 9$ nodes. If beam search is applied, it will be clear that most nodes will not be visited.

1a, p. 231: If $a = 1$, $b = 0$ and $c = 0$,

$| \rightarrow x$, score: $9/15 = 0.60$
- $| a \rightarrow x$, score: $8/12 = 0.67$
  - $| a, b \rightarrow x$, score: $8/10 = 0.80$
    - $| a, b, c_1 \rightarrow x$, score: $1/1 = 1.00$ $\surd$
    - $| a, b, c_2 \rightarrow x$, score: $1/1 = 1.00$ $\surd$
    - $| \ldots$
    - $| a, b, c_8 \rightarrow x$, score: $1/1 = 1.00$ $\surd$
    - $| a, b, c_9 \rightarrow x$, score: $0/1 = 0.00$
    - $| a, b, c_{10} \rightarrow x$, score: $0/1 = 0.00$
    - $| a, b, c_{11} \rightarrow x$, score: $1/1 = 1.00$
    - $| a, b, c_{12} \rightarrow x$, score: $0/1 = 0.00$
    - $| a, b, c_{13} \rightarrow x$, score: $0/1 = 0.00$
    - $| a, b, c_{14} \rightarrow x$, score: $0/1 = 0.00$
    - $| a, b, c_{15} \rightarrow x$, score: $0/1 = 0.00$
- $| b \rightarrow x$, score: $9/12 = 0.75$
  - $| b, a \rightarrow x$, score: $8/10 = 0.80$
    - specializations as with $a, b \rightarrow x$
    - $\ldots$
- $| \neg a \rightarrow x$, score: $1/3 = 0.33$
  - $| \neg a, b \rightarrow x$, score: $1/2 = 0.50$
    - $| \neg a, b, c_{11} \rightarrow x$, score: $1/1 = 1.00$ $\surd$
  - $| \neg a, \neg b \rightarrow x$, score: $0/1 = 0.00$
- $| \neg b \rightarrow x$, score: $0/3 = 0.00$
- $| c_1 \rightarrow x$, score: $1/1 = 1.00$ $\surd$
- $| c_2 \rightarrow x$, score: $1/1 = 1.00$ $\surd$
- $| \ldots$
- $| c_8 \rightarrow x$, score: $1/1 = 1.00$ $\surd$
- $| c_9 \rightarrow x$, score: $0/1 = 0.00$
- $| c_{10} \rightarrow x$, score: $0/1 = 0.00$
- $| c_{11} \rightarrow x$, score: $1/1 = 1.00$ $\surd$
- $| c_{12} \rightarrow x$, score: $0/1 = 0.00$
- $| c_{13} \rightarrow x$, score: $0/1 = 0.00$
- $| c_{14} \rightarrow x$, score: $0/1 = 0.00$
- $| c_{15} \rightarrow x$, score: $0/1 = 0.00$

Thus, with $a = 1$, $b = 0$ and $c = 0$, rules are specialized into the extreme. Since the maximum score is 1.00, and several rules score 1.00, the best rule is among $a, b, c_1 \rightarrow x$, $a, b, c_2 \rightarrow x$, $a, b, c_8 \rightarrow x$, $\neg a, b, c_{11} \rightarrow x$, $c_1 \rightarrow x$, $c_2 \rightarrow x$, $\ldots$, $c_8 \rightarrow x$, and $c_{11} \rightarrow x$.

1b, p. 231: If $a = 1$, $b = 0$ and $c = 0.023$,

$| \rightarrow x$, score: $9/15 + 0.023 \times 15 = 0.945$

| $a \to x$, score: $8/12 + 0.023 \times 12 = 0.946$
  | $a, b \to x$, score: $8/10 + 0.023 \times 10 = 1.03$ $\surd$
    | $a, b, c_1 \to x$, score: $1/1 + 0.023 \times 1 = 1.02$ stop: $\downarrow$
    | $a, b, c_2 \to x$, score: $1/1 + 0.023 \times 1 = 1.02$ stop: $\downarrow$
    | ...
    | $a, b, c_8 \to x$, score: $1/1 + 0.023 \times 1 = 1.02$ stop: $\downarrow$
    | $a, b, c_9 \to x$, score: $0/1 + 0.023 \times 1 = 0.02$ stop: $\downarrow$
    | $a, b, c_{10} \to x$, score: $0/1 + 0.023 \times 1 = 0.02$ stop: $\downarrow$
    | $a, b, c_{11} \to x$, score: $1/1 + 0.023 \times 1 = 1.02$ stop: $\downarrow$
    | $a, b, c_{12} \to x$, score: $0/1 + 0.023 \times 1 = 0.02$ stop: $\downarrow$
    | $a, b, c_{13} \to x$, score: $0/1 + 0.023 \times 1 = 0.02$ stop: $\downarrow$
    | $a, b, c_{14} \to x$, score: $0/1 + 0.023 \times 1 = 0.02$ stop: $\downarrow$
    | $a, b, c_{15} \to x$, score: $0/1 + 0.023 \times 1 = 0.02$ stop: $\downarrow$
| $b \to x$, score: $9/12 + 0.023 \times 12 = 1.026$
  | $b, a \to x$, score: $8/10 + 0.023 \times 10 = 1.03$ $\surd$
    specializes further as with $a, b \to x$
    ...
| $\neg a \to x$, score: $1/3 + 0.023 \times 3 = 0.399$
  | $\neg a, b \to x$, score: $1/2 + 0.023 \times 2 = 0.546$
    | $\neg a, b, c_{11} \to x$, score: $1/1 + 0.023\times = 1.02$ $\surd$
  | $\neg a, \neg b \to x$, score: $0/1 + 0.023 \times 1 = 0.02$ stop: $\downarrow$
| $\neg b \to x$, score: $0/3 + 0.023 \times 3 = 0.069$ $\surd$ (all specializations decrease the score)
| $c_1 \to x$, score: $1/1 + 0.023 \times 1 = 1.02$ $\surd$ (*id.*)
| $c_2 \to x$, score: $1/1 + 0.023 \times 1 = 1.02$ $\surd$ (*id.*)
| ...
| $c_8 \to x$, score: $1/1 + 0.023 \times 1 = 1.02$ $\surd$ (*id.*)
| $c_9 \to x$, score: $0/1 + 0.023 \times 1 = 0.02$ $\surd$ (*id.*)
| $c_{10} \to x$, score: $0/1 + 0.023 \times 1 = 0.02$ $\surd$ (*id.*)
| $c_{11} \to x$, score: $1/1 + 0.023 \times 1 = 1.02$ $\surd$ (*id.*)
| $c_{12} \to x$, score: $0/1 + 0.023 \times 1 = 0.02$ $\surd$ (*id.*)
| $c_{13} \to x$, score: $0/1 + 0.023 \times 1 = 0.02$ $\surd$ (*id.*)
| $c_{14} \to x$, score: $0/1 + 0.023 \times 1 = 0.02$ $\surd$ (*id.*)
| $c_{15} \to x$, score: $0/1 + 0.023 \times 1 = 0.02$ $\surd$

"Stop: $\downarrow$" means that the algorithm stops with specializing the rule, because the score decreases. This time, the maximum score is 1.03, which is for $a, b \to x$ and $b, a \to x$. If order in the antecedent does not matter, there is precisely one best rule, viz. $a, b \to x$.

1c, p. 231: To reach an optimum score at $a, b \to x$ and $b, a \to x$, we reason as follows. To ensure that $\to x$, $a \to x$ and $b \to x$ are specialized into $a, b \to x$ and $b, a \to x$, the score of $\to x$, $a \to x$ and $b \to x$ must be less than the score of $a, b \to x$ and $b, a \to x$:

$$9/15 + 15c < 8/12 + 12c,$$
$$8/12 + 12c < 8/10 + 10c, \text{ and}$$
$$9/12 + 12c < 8/10 + 10c,$$

which is equivalent to

$$c < 0.0233, \; c < 0.0433, \text{ and } c < 0.0250. \tag{B.15}$$

To ensure that $a, b \to x$ and $b, a \to x$ need not be specialized further, the score of all specializations of $a, b \to x$ and $b, a \to x$ must be less than the score of $a, b \to x$ and $b, a \to x$. This amounts to

$$8/10 + 10c > 1 + c,$$

which is equivalent to

$$c > 0.0222. \tag{B.16}$$

Combining (B.15) and (B.16) yields $0.222 < c < 0.233$. If $c \geq 0.233$ the search will stop too early; if $c \leq 0.222$, the search will overshoot its goal.

2a, p. 236: The network tries to express that lions are more similar to tigers than, e.g, lions to fishes. Fishes are dissimilar to mammals (-1). Pigs are neither similar nor dissimilar to lions and tigers. I thought that makes sense...

2b, p. 236: The node values express a selection of a coherent subset of "lion," "tiger," "fish," "pig," with indifference as regarding to whether "pig" should be included in this set.

2c, p. 236: Local coherence between "lion" and "fish" is $1 \times -1 \times -1 = 1$.

2d, p. 236: Global coherence is $0 + 0 + 0 + 1 + 1 + 1 = 3$.

2e, p. 236: Change node value of fish from -1 into 1. Local coherency between tiger and lion decreases with 2 each, so that global coherency decreases with 4 to -1. (Other solutions are possible.)

2f, p. 236: If we are searching for an optimal solution, we are allowed to change only the node values. Trying out different values (!) yields that the global coherency can be increased to 4 if "pig" is set to 1.

2g, p. 236: Yes: a perfect solution is an assignment of activation values for which every (in)coherence relation between two propositions is satisfied. There are four non-zero links, so a perfect solution would have all 4 such links satisfied. The previous item shows that this is possible.

3, p. 236: If $P \sim Q \sim R \nsim P$, for instance, then accepting P implies accepting Q, and accepting Q implies accepting R. Accepting R, however, implies rejecting P. A similar impossible situation arises if we assume P rejected. Thus, the highest coherence that can be obtained for this network is $1 + 1 + (-1) = 1$. (The more simple combination $P \sim Q \nsim P$ does not qualify as an example of a non-perfect network because it isn't a coherence network.)

4, p. 236: Suppose nodes can take on values from $\{-1, 0, 1\}$, and links $\{-1, 0, 1\}$. (Links with value 0 can be considered as not to exist.) The first node can take on three values, the connecting link can take on three values, and the second node can take on three values. This makes for $3^3 = 27$ combinations. Due to symmetry, there are actually $3^2 \cdot 2 = 18$ properly different combinations, since the value of the second node must be chosen such that it is unequal to the value of the first node.

2a, p. 240: One solution is.

```
C -> D
B contradicts D
A -> B
A,B -> C
observation A
A -> B ~ C -> D
```