CHAPTER 4

# Concept learning

HAVING DISCUSSED A VARIETY of tasks in the preceding two chapters, we are now in an excellent position to start discussing machine learning models and algorithms for learning them. This chapter and the next two are devoted to logical models, the hallmark of which is that they use logical expressions to divide the instance space into segments and hence construct grouping models. The goal is to find a segmentation such that the data in each segment is more homogeneous, with respect to the task to be solved. For instance, in classification we aim to find a segmentation such that the instances in each segment are predominantly of one class, while in regression a good segmentation is such that the target variable is a simple function of a small number of predictor variables. There are essentially two kinds of logical models: tree models and rule models. Rule models consist of a collection of implications or if–then rules, where the if-part defines a segment, and the then-part defines the behaviour of the model in this segment. Tree models are a restricted kind of rule model where the if-parts of the rules are organised in a tree structure.

In this chapter we consider methods for learning logical expressions or *concepts* from examples, which lies at the basis of both tree models and rule models. In concept learning we only learn a description for the positive class, and label everything that doesn't satisfy that description as negative. We will pay particular attention to the generality ordering that plays an important role in logical models. In the next two chapters

https://doi.org/10.1017/CBO9780511973000.006 Published online by Cambridge University Press

The simplest logical expressions are equalities of the form Feature = Value and, for numerical features, inequalities of the form Feature < Value; these are called *literals*. Complex Boolean expressions can be built using logical connectives: *conjunction* $\wedge$, *disjunction* $\vee$, *negation* $\neg$ and *implication* $\rightarrow$. The following equivalences hold (the left two are called the *De Morgan laws*):

$$\neg(A \wedge B) \equiv \neg A \vee \neg B \qquad\qquad \neg\neg A \equiv A$$

$$\neg(A \vee B) \equiv \neg A \wedge \neg B \qquad\qquad A \rightarrow B \equiv \neg A \vee B$$

If Boolean expression $A$ is true of instance $x$, we say that $A$ *covers* $x$. The set of instances covered by expression $A$ is called its *extension* and denoted $\mathscr{X}_A = \{x \in \mathscr{X} | A \text{ covers } x\}$, where $\mathscr{X}$ denotes the instance space which acts as the universe of discourse (see Background 2.1 on p.51). There is a direct correspondence between logical connectives and operations on sets: e.g., $\mathscr{X}_{A \wedge B} = \mathscr{X}_A \cap \mathscr{X}_B$, $\mathscr{X}_{A \vee B} = \mathscr{X}_A \cup \mathscr{X}_B$ and $\mathscr{X}_{\neg A} = \mathscr{X} \setminus \mathscr{X}_A$. If $\mathscr{X}_A \supseteq \mathscr{X}_{A'}$, we say that $A$ is *at least as general as* $A'$, and if in addition $\mathscr{X}_A \not\subseteq \mathscr{X}_{A'}$ we say that $A$ is *more general than* $A'$. This *generality ordering* is a partial order on logical expressions as defined in Background 2.1. (More precisely: it is a partial order on the equivalence classes of the relation of logical equivalence $\equiv$.)

A *clause* is an implication $P \rightarrow Q$ such that $P$ is a conjunction of literals and $Q$ is a disjunction of literals. Using the equivalences above we can rewrite such an implication as

$$(A \wedge B) \rightarrow (C \vee D) \equiv \neg(A \wedge B) \vee (C \vee D) \equiv \neg A \vee \neg B \vee C \vee D$$

and hence a clause can equivalently be seen as a disjunction of literals or their negations. Any logical expression can be rewritten as a conjunction of clauses; this is referred to as *conjunctive normal form* (*CNF*). Alternatively, any logical expression can be written as a disjunction of conjunctions of literals or their negation; this is called *disjunctive normal form* (*DNF*). A *rule* is a clause $A \rightarrow B$ where $B$ is a single literal; this is also often referred to as a *Horn clause*, after the American logician Alfred Horn.

**Background 4.1.** Some logical concepts and notation.

we consider tree and rule models, which go considerably beyond concept learning as they can handle multiple classes, probability estimation, regression, as well as clustering tasks.

## 4.1 The hypothesis space

The simplest concept learning setting is where we restrict the logical expressions describing concepts to conjunctions of literals (see Background 4.1 for a review of important definitions and notation from logic). The following example illustrates this.[1]

**Example 4.1 (Learning conjunctive concepts).** Suppose you come across a number of sea animals that you suspect belong to the same species. You observe their length in metres, whether they have gills, whether they have a prominent beak, and whether they have few or many teeth. Using these features, the first animal can described by the following conjunction:

$$\mathsf{Length} = 3 \land \mathsf{Gills} = \mathsf{no} \land \mathsf{Beak} = \mathsf{yes} \land \mathsf{Teeth} = \mathsf{many}$$

The next one has the same characteristics but is a metre longer, so you drop the length condition and generalise the conjunction to

$$\mathsf{Gills} = \mathsf{no} \land \mathsf{Beak} = \mathsf{yes} \land \mathsf{Teeth} = \mathsf{many}$$

The third animal is again 3 metres long, has a beak, no gills and few teeth, so your description becomes

$$\mathsf{Gills} = \mathsf{no} \land \mathsf{Beak} = \mathsf{yes}$$

All remaining animals satisfy this conjunction, and you finally decide they are some kind of dolphin.

Despite the simplicity of this example, the space of possible concepts – usually called the *hypothesis space* – is already fairly large. Let's assume we have three different lengths: 3, 4 and 5 metres, while the other three features have two values each. We then have $3 \cdot 2 \cdot 2 \cdot 2 = 24$ possible instances. How many conjunctive concepts are there using these same features? We can answer this question if we treat the absence of a feature as an additional 'value'. This gives a total of $4 \cdot 3 \cdot 3 \cdot 3 = 108$ different concepts. While this seems quite a lot, you should realise that the number of possible extensions – sets of instances – is much larger: $2^{24}$, which is more than 16 million! That is, if you pick a random set of instances, the odds that you can't find a conjunctive concept that exactly describes those instances are well over 100 000 to 1. This is actually a good thing, as it forces the learner to generalise beyond the training data and cover instances that it hasn't seen before. Figure 4.1 depicts this hypothesis space, making use of the general-

[1] Inspired by `www.cwtstrandings.org`.

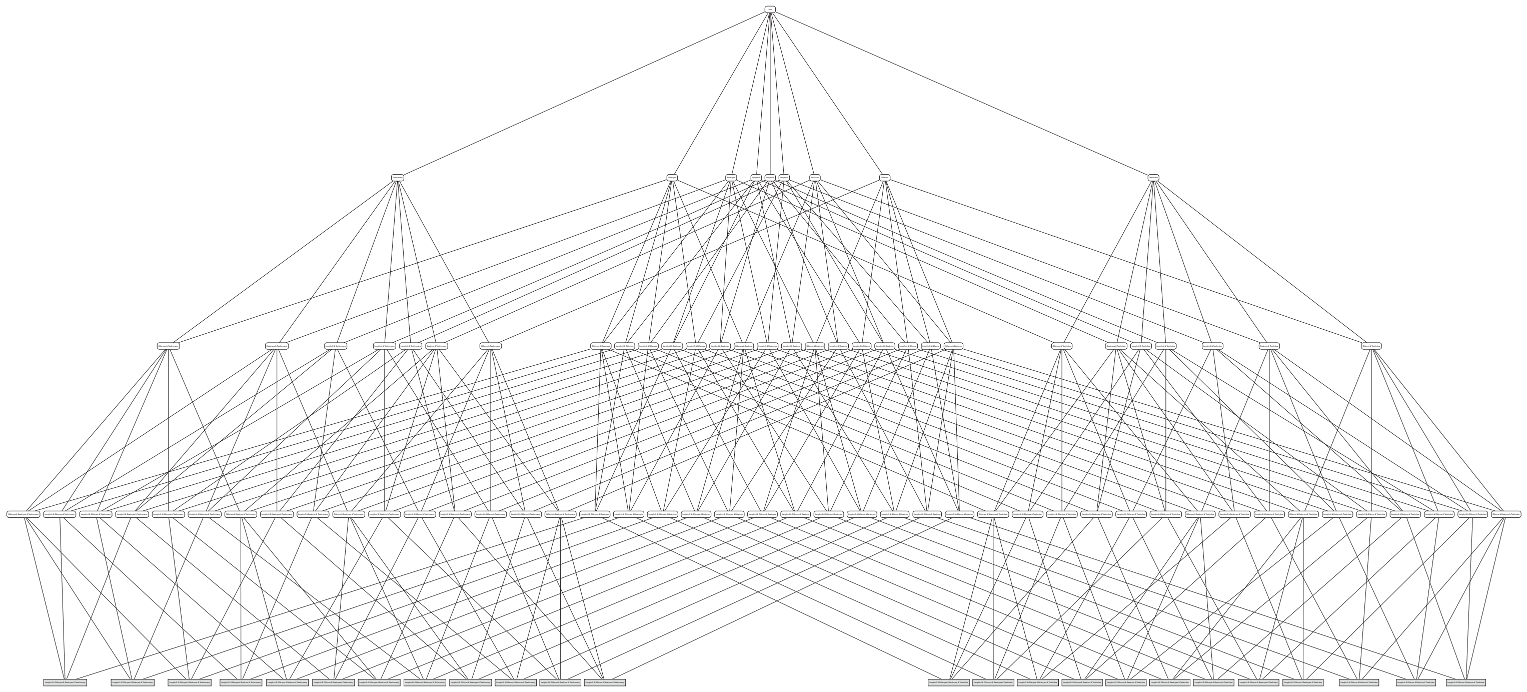

**Figure 4.1.** The hypothesis space corresponding to Example 4.1. The bottom row corresponds to the 24 possible instances, which are complete conjunctions with four literals each. The next row up are all 44 concepts with three literals each; then 30 concepts with two literals each; 9 concepts consisting of a single literal; and the top concept is the empty conjunction which is always true and hence covers all possible instances. A connecting line is drawn between concepts on consecutive layers if the higher one is more general than the lower one (i.e., the higher concept's extension is a superset of the lower's).

ity ordering (i.e., the subset relationship between concept extensions; see Background 4.1).

## Least general generalisation

If we rule out all concepts that don't cover at least one of the instances in Example 4.1, the hypothesis space is reduced to 32 conjunctive concepts (Figure 4.2). Insisting that any hypothesis cover all three instances reduces this further to only four concepts, the least general one of which is the one found in the example – it is called their *least general generalisation* (*LGG*). Algorithm 4.1 formalises the procedure, which is simply to repeatedly apply a pairwise LGG operation (Algorithm 4.2) to an instance and the current hypothesis, as they both have the same logical form. The structure of the hypothesis space ensures that the result is independent of the order in which the instances are processed.

Intuitively, the LGG of two instances is the nearest concept in the hypothesis space where paths upward from both instances intersect. The fact that this point is unique is a special property of many logical hypothesis spaces, and can be put to good use in learning. More precisely, such a hypothesis space forms a *lattice*: a partial order in which each two elements have a *least upper bound* (*lub*) and a *greatest lower bound* (*glb*). So, the LGG of a set of instances is exactly the least upper bound of the instances in that lattice. Furthermore, it is the greatest lower bound of the set of all generalisations of the instances: all possible generalisations are at least as general as the LGG. In this very precise sense, *the LGG is the most conservative generalisation that we can learn from the data*.

If we want to be a bit more adventurous, we could choose one of the more general hypotheses, such as Gills = no or Beak = yes. However, we probably don't want to choose the most general hypothesis, which is simply that every animal is a dolphin,

**Algorithm 4.1:** LGG-Set($D$) – find least general generalisation of a set of instances.

```
Input  : data $D$.
Output : logical expression $H$.
1 $x$ ←first instance from $D$;
2 $H$ ←$x$;
3 while instances left do
4     $x$ ←next instance from $D$;
5     $H$ ←LGG($H, x$) ;          // e.g., LGG-Conj (Alg. 4.2) or LGG-Conj-ID (Alg. 4.3)
6 end
7 return $H$
```

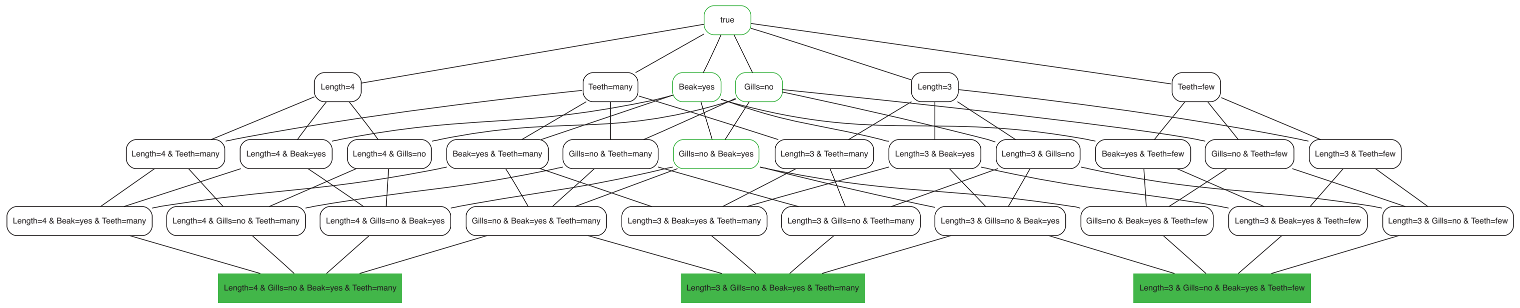
true
Length=4
Teeth=many
Beak=yes
Gills=no
Length=3
Teeth=few
Length=4 & Teeth=many
Length=4 & Beak=yes
Length=4 & Gills=no
Beak=yes & Teeth=many
Gills=no & Teeth=many
Gills=no & Beak=yes
Length=3 & Teeth=many
Length=3 & Beak=yes
Length=3 & Gills=no
Beak=yes & Teeth=few
Gills=no & Teeth=few
Length=3 & Teeth=few
Length=4 & Beak=yes & Teeth=many
Length=4 & Gills=no & Teeth=many
Length=4 & Gills=no & Beak=yes
Gills=no & Beak=yes & Teeth=many
Length=3 & Beak=yes & Teeth=many
Length=3 & Gills=no & Teeth=many
Length=3 & Gills=no & Beak=yes
Gills=no & Beak=yes & Teeth=few
Length=3 & Beak=yes & Teeth=few
Length=3 & Gills=no & Teeth=few
Length=4 & Gills=no & Beak=yes & Teeth=many
Length=3 & Gills=no & Beak=yes & Teeth=many
Length=3 & Gills=no & Beak=yes & Teeth=few

**Figure 4.2.** Part of the hypothesis space in Figure 4.1 containing only concepts that are more general than at least one of the three given instances on the bottom row. Only four conjunctions, indicated in green at the top, are more general than all three instances; the least general of these is Gills = no ∧ Beak = yes. It can be observed that the two left-most and right-most instances would be sufficient to learn that concept.

as this would clearly be an over-generalisation. Negative examples are very useful to prevent over-generalistion.

---

**Example 4.2 (Negative examples).** In Example 4.1 we observed the following dolphins:

p1: $\text{Length} = 3 \ \wedge \text{Gills} = \text{no} \ \wedge \text{Beak} = \text{yes} \ \wedge \text{Teeth} = \text{many}$
p2: $\text{Length} = 4 \ \wedge \text{Gills} = \text{no} \ \wedge \text{Beak} = \text{yes} \ \wedge \text{Teeth} = \text{many}$
p3: $\text{Length} = 3 \ \wedge \text{Gills} = \text{no} \ \wedge \text{Beak} = \text{yes} \ \wedge \text{Teeth} = \text{few}$

Suppose you next observe an animal that clearly doesn't belong to the species – a negative example. It is described by the following conjunction:

n1: $\text{Length} = 5 \ \wedge \text{Gills} = \text{yes} \ \wedge \text{Beak} = \text{yes} \ \wedge \text{Teeth} = \text{many}$

This negative example rules out some of the generalisations that were hitherto still possible: in particular, it rules out the concept $\text{Beak} = \text{yes}$, as well as the empty concept which postulates that everything is a dolphin.

---

The process is illustrated in Figure 4.3. We now have two hypotheses left, one which is least general and the other most general.

### *Internal disjunction*

You might be tempted to conclude from this and the previous example that we always have a unique most general hypothesis, but that is not the case in general. To demonstrate that, we are going to make our logical language slightly richer, by allowing a restricted form of disjunction called *internal disjunction*. The idea is very simple: if you observe one dolphin that is 3 metres long and another one of 4 metres, you may want to add the condition 'length is 3 or 4 metres' to your concept. We will write this as $\text{Length} = [3,4]$, which logically means $\text{Length} = 3 \vee \text{Length} = 4$. This of course only makes sense for features that have more than two values: for instance, the internal disjunction $\text{Teeth} = [\text{many}, \text{few}]$ is always true and can be dropped.

**Algorithm 4.2:** LGG-Conj$(x, y)$ – find least general conjunctive generalisation of two conjunctions.

```
   Input   : conjunctions $x, y$.
   Output  : conjunction $z$.
1  $z \leftarrow$ conjunction of all literals common to $x$ and $y$;
2  return $z$
```

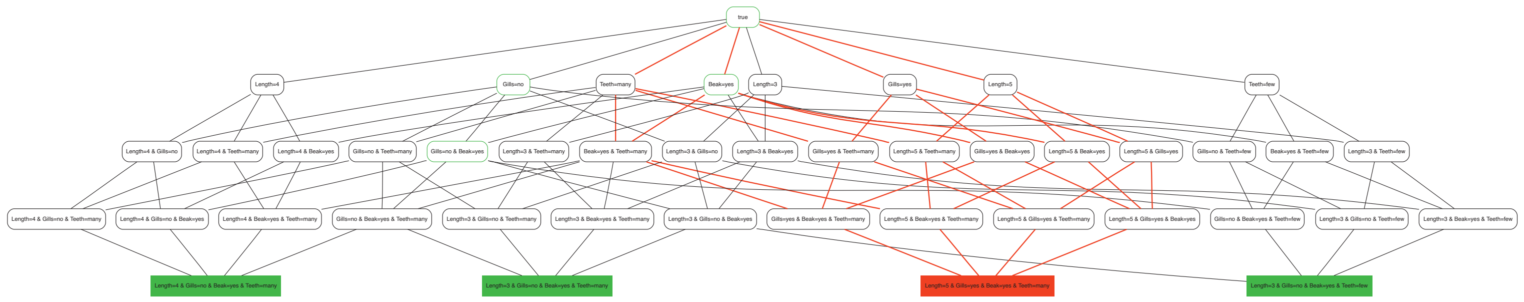
true
Length=4
Gills=no
Teeth=many
Beak=yes
Length=3
Gills=yes
Length=5
Teeth=few
Length=4 & Gills=no
Length=4 & Teeth=many
Length=4 & Beak=yes
Gills=no & Teeth=many
Gills=no & Beak=yes
Length=3 & Teeth=many
Beak=yes & Teeth=many
Length=3 & Gills=no
Length=3 & Beak=yes
Gills=yes & Teeth=many
Length=5 & Teeth=many
Gills=yes & Beak=yes
Length=5 & Beak=yes
Length=5 & Gills=yes
Gills=no & Teeth=few
Beak=yes & Teeth=few
Length=3 & Teeth=few
Length=4 & Gills=no & Teeth=many
Length=4 & Gills=no & Beak=yes
Length=4 & Beak=yes & Teeth=many
Gills=no & Beak=yes & Teeth=many
Length=3 & Gills=no & Teeth=many
Length=3 & Beak=yes & Teeth=many
Length=3 & Gills=no & Beak=yes
Gills=yes & Beak=yes & Teeth=many
Length=5 & Beak=yes & Teeth=many
Length=5 & Gills=yes & Teeth=many
Length=5 & Gills=yes & Beak=yes
Gills=no & Beak=yes & Teeth=few
Length=3 & Gills=no & Teeth=few
Length=3 & Beak=yes & Teeth=few
Length=4 & Gills=no & Beak=yes & Teeth=many
Length=3 & Gills=no & Beak=yes & Teeth=many
Length=5 & Gills=yes & Beak=yes & Teeth=many
Length=3 & Gills=no & Beak=yes & Teeth=few

**Figure 4.3.** A negative example can rule out some of the generalisations of the LGG of the positive examples. Every concept which is connected by a red path to a negative example covers that negative and is therefore ruled out as a hypothesis. Only two conjunctions cover all positives and no negatives: Gills = no ∧ Beak = yes and Gills = no.

**Example 4.3 (Internal disjunction).** Using the same three positive examples as in Example 4.1, the second and third hypothesis are now

$$\mathsf{Length} = [3,4] \wedge \mathsf{Gills} = \mathsf{no} \wedge \mathsf{Beak} = \mathsf{yes} \wedge \mathsf{Teeth} = \mathsf{many}$$

and

$$\mathsf{Length} = [3,4] \wedge \mathsf{Gills} = \mathsf{no} \wedge \mathsf{Beak} = \mathsf{yes}$$

We can drop any of the three conditions in the latter LGG without covering the negative example from Example 4.2. Generalising further to single conditions, we see that $\mathsf{Length} = [3,4]$ and $\mathsf{Gills} = \mathsf{no}$ are still OK but $\mathsf{Beak} = \mathsf{yes}$ is not, as it covers the negative example.

Algorithm 4.3 details how we can calculate the LGG of two conjunctions employing internal disjunction. The function $\mathsf{Combine\text{-}ID}(v_x, v_y)$ returns $[v_x, v_y]$ if $v_x$ and $v_y$ are constants, and their union if $v_x$ or $v_y$ are already sets of values: e.g., $\mathsf{Combine\text{-}ID}([3,4],[4,5])=[3,4,5]$.

## 4.2 Paths through the hypothesis space

As we can clearly see in Figure 4.4, in this example we have not one but two most general hypotheses. What we can also notice is that *every concept between the least general one and one of the most general ones is also a possible hypothesis*, i.e., covers all the positives and none of the negatives. Mathematically speaking we say that the set of

**Algorithm 4.3:** $\mathsf{LGG\text{-}Conj\text{-}ID}(x, y)$ – find least general conjunctive generalisation of two conjunctions, employing internal disjunction.

```
Input  : conjunctions $x, y$.
Output : conjunction $z$.
z ←true;
for each feature $f$ do
    if $f = v_x$ is a conjunct in $x$ and $f = v_y$ is a conjunct in $y$ then
        add $f = \mathsf{Combine\text{-}ID}(v_x, v_y)$ to $z$ ;          // Combine-ID: see text
    end
end
return $z$
```

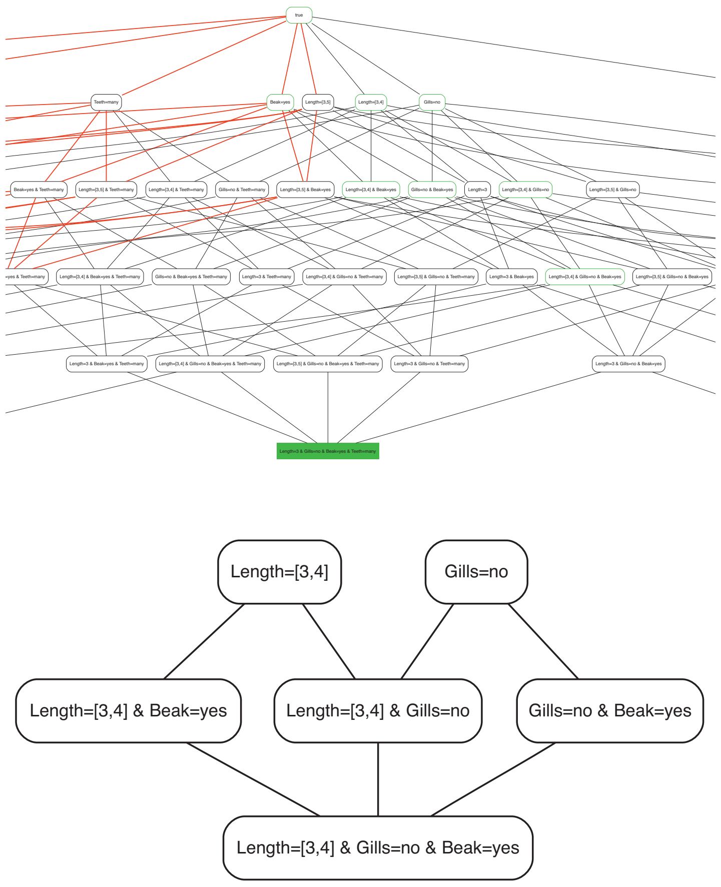

**Figure 4.4. (top)** A snapshot of the expanded hypothesis space that arises when internal disjunction is used for the 'Length' feature. We now need one more generalisation step to travel upwards from a completely specified example to the empty conjunction. **(bottom)** The version space consists of one least general hypothesis, two most general hypotheses, and three in between.

hypotheses that agree with the data is a *convex set*, which basically means that we can interpolate between any two members of the set, and if we find a concept that is less general than one and more general than the other then that concept is also a member of the set. This in turn means that we can describe the set of all possible hypotheses by its least and most general members. This is summed up in the following definition.

**Definition 4.1 (Version space).** *A concept is complete if it covers all positive examples. A concept is consistent if it covers none of the negative examples. The version space is the set of all complete and consistent concepts. This set is convex and is fully defined by its least and most general elements.*

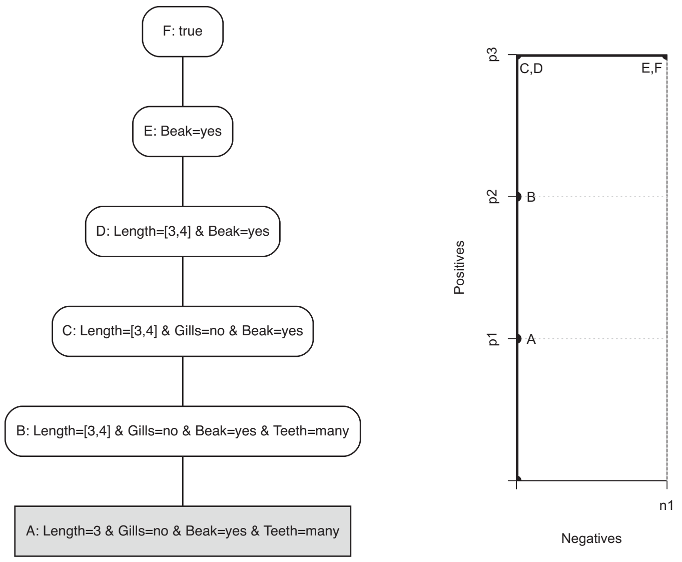
F: true
E: Beak=yes
D: Length=[3,4] & Beak=yes
C: Length=[3,4] & Gills=no & Beak=yes
B: Length=[3,4] & Gills=no & Beak=yes & Teeth=many
A: Length=3 & Gills=no & Beak=yes & Teeth=many
Positives
p1
p2
p3
A
B
C,D
E,F
n1
Negatives

**Figure 4.5.** **(left)** A path in the hypothesis space of Figure 4.3 from one of the positive examples (p1, see Example 4.2 on p.110) all the way up to the empty concept. Concept A covers a single example; B covers one additional example; C and D are in the version space, and so cover all three positives; E and F also cover the negative. **(right)** The corresponding coverage curve, with ranking p1 – p2 – p3 – n1.

We can draw a useful connection between logical hypothesis spaces and the coverage plots introduced in Chapter 2. Suppose you were to follow a path in the hypothesis space from a positive example, through a selection of its generalisations, all the way up to the empty concept. The latter, by construction, covers all positives and all negatives, and hence occupies the top-right point $(Neg, Pos)$ in the coverage plot. The starting point, being a single positive example, occupies the point $(0, 1)$ in the coverage plot. In fact, it is customary to extend the hypothesis space with a bottom element which doesn't cover any examples and hence is less general than any other concept. Taking that point as the starting point of the path means that we start in the bottom-left point $(0, 0)$ in the coverage plot.

Moving upwards in the hypothesis space by generalisation means that the numbers of covered positives and negatives can stay the same or increase, but never decrease. In other words, *an upward path through the hypothesis space corresponds to a coverage curve* and hence to a ranking. Figure 4.5 illustrates this for the running example. The chosen path is but one among many possible paths; however, notice that if a path, like this one, includes elements of the version space, the corresponding coverage curve passes through 'ROC heaven' $(0, Pos)$ and AUC $= 1$. In other words, such paths are optimal. Concept learning can be seen as the search for an optimal path through the hypothesis space.

What happens, you may ask, if the LGG of the positive examples covers one or more negatives? In that case, any generalisation of the LGG will be inconsistent as well. Conversely, any consistent hypothesis will be incomplete. It follows that the version space is empty in this case; we will say that the data is not *conjunctively separable*. The following example illustrates this.

---

**Example 4.4 (Data that is not conjunctively separable).** Suppose we have the following five positive examples (the first three are the same as in Example 4.1):

p1: Length = 3 ∧ Gills = no ∧ Beak = yes ∧ Teeth = many
p2: Length = 4 ∧ Gills = no ∧ Beak = yes ∧ Teeth = many
p3: Length = 3 ∧ Gills = no ∧ Beak = yes ∧ Teeth = few
p4: Length = 5 ∧ Gills = no ∧ Beak = yes ∧ Teeth = many
p5: Length = 5 ∧ Gills = no ∧ Beak = yes ∧ Teeth = few

and the following negatives (the first one is the same as in Example 4.2):

n1: Length = 5 ∧ Gills = yes ∧ Beak = yes ∧ Teeth = many
n2: Length = 4 ∧ Gills = yes ∧ Beak = yes ∧ Teeth = many
n3: Length = 5 ∧ Gills = yes ∧ Beak = no ∧ Teeth = many
n4: Length = 4 ∧ Gills = yes ∧ Beak = no ∧ Teeth = many
n5: Length = 4 ∧ Gills = no ∧ Beak = yes ∧ Teeth = few

The least general complete hypothesis is Gills = no ∧ Beak = yes as before, but this covers n5 and hence is inconsistent. There are seven most general consistent hypotheses, none of which are complete:

| | |
|---|---|
| Length = 3 | (covers p1 and p3) |
| Length = [3, 5] ∧ Gills = no | (covers all positives except p2) |
| Length = [3, 5] ∧ Teeth = few | (covers p3 and p5) |
| Gills = no ∧ Teeth = many | (covers p1, p2 and p4) |
| Gills = no ∧ Beak = no | |
| Gills = yes ∧ Teeth = few | |
| Beak = no ∧ Teeth = few | |

The last three of these do not cover any positive examples.

---

### *Most general consistent hypotheses*

As this example suggests, finding most general consistent hypotheses is considerably more involved than finding least general complete ones. Essentially, the process is one of enumeration. Algorithm 4.4 gives an algorithm which returns all most general consistent specialisations of a given concept, where a minimal specialisation of a concept is one that can be reached in one downward step in the hypothesis lattice (e.g., by adding a conjunct, or removing a value from an internal disjunction). Calling the algorithm with $C =$ true returns the most general consistent hypotheses.

Figure 4.6 shows a path through the hypothesis space of Example 4.4, and the corresponding coverage curve. We see that the path goes through three consistent hypotheses, which are consequently plotted on the $y$-axis of the coverage plot. The other three hypotheses are complete, and therefore end up on the top of the graph; one of these is, in fact, the LGG of the positives (D). The ranking corresponding to this coverage curve is p3 – p5 – [p1,p4] – [p2,n5] – [n1–4]. This ranking commits half a ranking error out of 25, and so AUC = 0.98. We can choose one concept from the ranking by applying the techniques discussed in Section 2.2. For instance, suppose that classification accuracy is the criterion we want to optimise. In coverage space, accuracy isometrics have slope 1, and so we see immediately that concepts C and D (or E) both achieve the best accuracy in Figure 4.6. If performance on the positives is more important we prefer the complete but inconsistent concept D; if performance on the negatives is valued more we choose the incomplete but consistent concept C.

### *Closed concepts*

It is worthwhile to reflect on the fact that concepts D and E occupy the same point in coverage space. What this means is that generalising D into E by dropping Beak = yes does not change the coverage in terms of positive and negative examples. One could

**Algorithm 4.4:** MGConsistent($C, N$) – find most general consistent specialisations of a concept.

```
   Input   : concept $C$; negative examples $N$.
   Output  : set of concepts $S$.
1  if $C$ doesn't cover any element from $N$ then return $\{C\}$;
2  $S \leftarrow \emptyset$;
3  for each minimal specialisation $C'$ of $C$ do
4      $S \leftarrow S \cup$ MGConsistent($C', N$);
5  end
6  return $S$
```

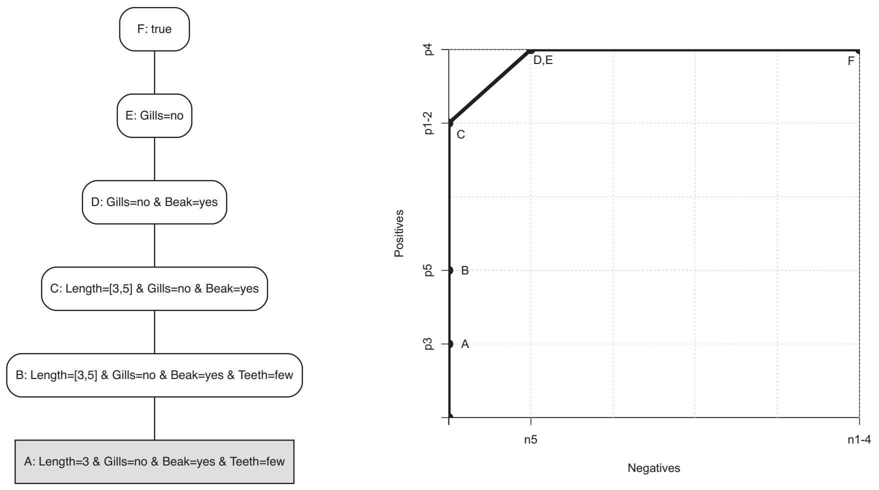
F: true
E: Gills=no
D: Gills=no & Beak=yes
C: Length=[3,5] & Gills=no & Beak=yes
B: Length=[3,5] & Gills=no & Beak=yes & Teeth=few
A: Length=3 & Gills=no & Beak=yes & Teeth=few
p4
p1-2
p5
p3
Positives
D,E
F
C
B
A
n5
n1-4
Negatives

**Figure 4.6. (left)** A path in the hypothesis space of Example 4.4. Concept A covers a single positive (p3); B covers one additional positive (p5); C covers all positives except p4; D is the LGG of all five positive examples, but also covers a negative (n5), as does E. **(right)** The corresponding coverage curve.

say that the data suggests that, in the context of concept E, the condition Beak = yes is implicitly understood. A concept that includes all implicitly understood conditions is called a *closed concept*. Essentially, a closed concept is the LGG of all examples that it covers. For instance, D and E both cover all positives and n5; the LGG of those six examples is Gills = no ∧ Beak = yes, which is D. Mathematically speaking we say that the closure of E is D, which is also its own closure – hence the term 'closed concept'. This doesn't mean that D and E are logically equivalent: on the contrary, since $\mathscr{X}_{\rm D} \subset \mathscr{X}_{\rm E}$ – the extension of D is a proper subset of the extension of E – there exist instances in $\mathscr{X}$ that are covered by E but not by D. However, none of these 'witnesses' are present in the data, and thus, as far as the data is concerned, D and E are indistinguishable. As can be seen in Figure 4.7, limiting attention to closed concepts can considerably reduce the hypothesis space.

In this section we have looked at the problem of learning a single logical expression that covers most or all positive examples and few or no negative examples. We have seen that such concepts live in a hypothesis space ordered by generality, and learning a concept can be understood as finding a good path through that hypothesis space. Such a path has a natural interpretation as a ranker, which allows a connection with coverage curves and ROC curves. On the other hand, insisting on a single conjunction of feature-value literals is a strong limitation; in the next section we look at ways to relax it.

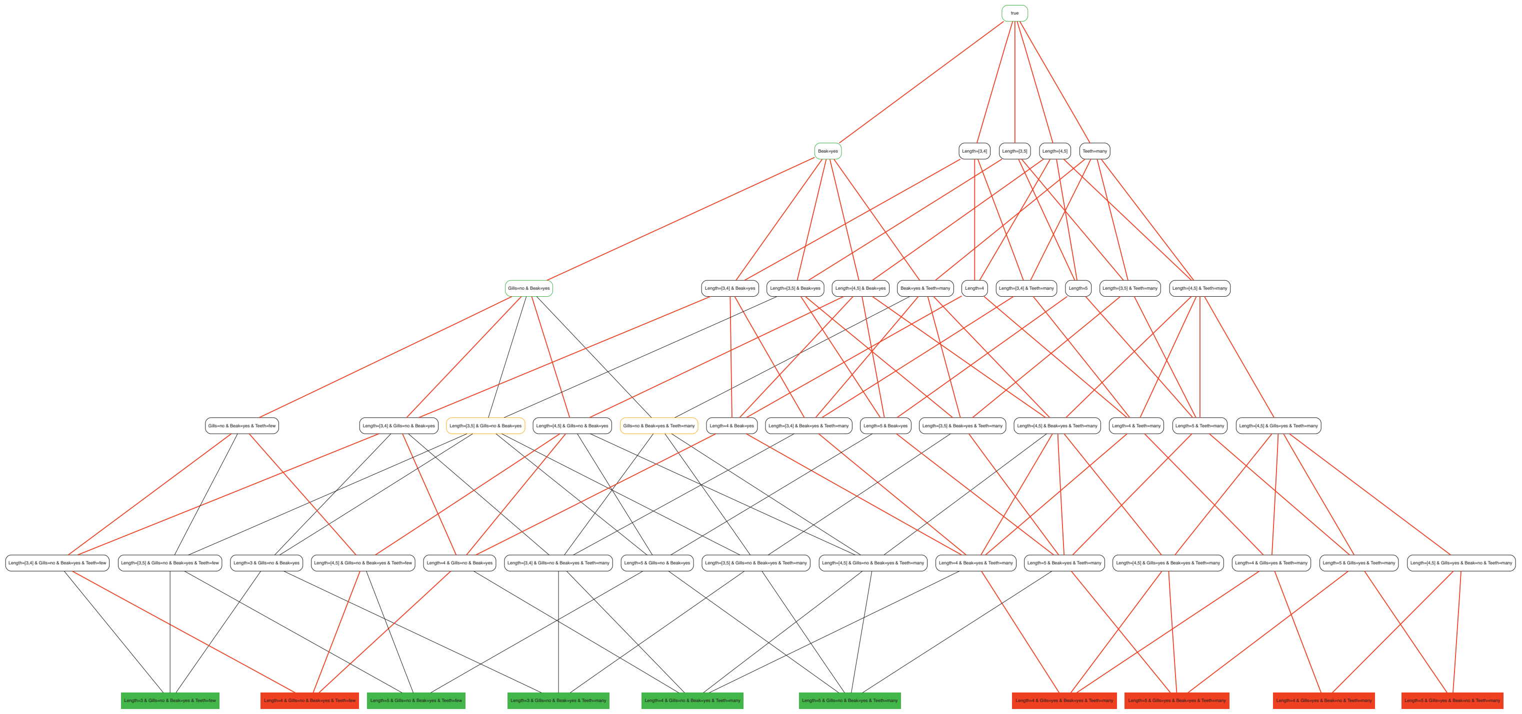
true
Beak=yes
Length=[3,4]
Length=[3,5]
Length=[4,5]
Teeth=many
Gills=no & Beak=yes
Length=[3,4] & Beak=yes
Length=[3,5] & Beak=yes
Length=[4,5] & Beak=yes
Beak=yes & Teeth=many
Length=4
Length=[3,4] & Teeth=many
Length=5
Length=[3,5] & Teeth=many
Length=[4,5] & Teeth=many
Gills=no & Beak=yes & Teeth=few
Length=[3,4] & Gills=no & Beak=yes
Length=[3,5] & Gills=no & Beak=yes
Length=[4,5] & Gills=no & Beak=yes
Gills=no & Beak=yes & Teeth=many
Length=4 & Beak=yes
Length=[3,4] & Beak=yes & Teeth=many
Length=5 & Beak=yes
Length=[3,5] & Beak=yes & Teeth=many
Length=[4,5] & Beak=yes & Teeth=many
Length=4 & Teeth=many
Length=5 & Teeth=many
Length=[4,5] & Gills=yes & Teeth=many
Length=[3,4] & Gills=no & Beak=yes & Teeth=few
Length=[3,5] & Gills=no & Beak=yes & Teeth=few
Length=3 & Gills=no & Beak=yes
Length=[4,5] & Gills=no & Beak=yes & Teeth=few
Length=4 & Gills=no & Beak=yes
Length=[3,4] & Gills=no & Beak=yes & Teeth=many
Length=5 & Gills=no & Beak=yes
Length=[3,5] & Gills=no & Beak=yes & Teeth=many
Length=[4,5] & Gills=no & Beak=yes & Teeth=many
Length=4 & Beak=yes & Teeth=many
Length=5 & Beak=yes & Teeth=many
Length=[4,5] & Gills=yes & Beak=yes & Teeth=many
Length=4 & Gills=yes & Teeth=many
Length=5 & Gills=yes & Teeth=many
Length=[4,5] & Gills=yes & Beak=no & Teeth=many
Length=3 & Gills=no & Beak=yes & Teeth=few
Length=4 & Gills=no & Beak=yes & Teeth=few
Length=5 & Gills=no & Beak=yes & Teeth=few
Length=3 & Gills=no & Beak=yes & Teeth=many
Length=4 & Gills=no & Beak=yes & Teeth=many
Length=5 & Gills=no & Beak=yes & Teeth=many
Length=4 & Gills=yes & Beak=yes & Teeth=many
Length=5 & Gills=yes & Beak=yes & Teeth=many
Length=4 & Gills=yes & Beak=no & Teeth=many
Length=5 & Gills=yes & Beak=no & Teeth=many

**Figure 4.7.** The hypothesis space is reduced considerably if we restrict attention to closed concepts. There are three, rather than four, complete concepts (in green), and two, rather than seven, most general consistent closed concepts (in orange). Notice that the latter are both specialisations of the LGG of the positives, and hence it is possible to select a path that includes both the LGG and a most general consistent hypothesis.

## 4.3 Beyond conjunctive concepts

Recall from Background 4.1 that a conjunctive normal form expression (CNF) is a conjunction of disjunctions of literals, or equivalently, a conjunction of clauses. The conjunctions of literals we have looked at until now are trivially in CNF where each disjunction consists of a single literal. CNF expressions are much more expressive, particularly since literals can occur in several clauses. We will look at an algorithm for learning Horn theories, where each clause $A \rightarrow B$ is a Horn clause, i.e., $A$ is a conjunction of literals and $B$ is a single literal. For ease of notation we will restrict attention to Boolean features, and write $\mathsf{F}$ for $\mathsf{F} = \mathsf{true}$ and $\neg\mathsf{F}$ for $\mathsf{F} = \mathsf{false}$. In the example below we adapt the dolphins example to Boolean variables $\mathsf{ManyTeeth}$ (standing for $\mathsf{Teeth} = \mathsf{many}$), $\mathsf{Gills}$, $\mathsf{Short}$ (standing for $\mathsf{Length} = 3$) and $\mathsf{Beak}$.

When we looked at learning conjunctive concepts, the main intuition was that uncovered positive examples led us to generalise by dropping literals from the conjunction, while covered negative examples require specialisation by adding literals. This intuition still holds if we are learning Horn theories, but now we need to think ‘clauses’ rather than ‘literals’. Thus, if a Horn theory doesn’t cover a positive we need to drop all clauses that violate the positive, where a clause $A \rightarrow B$ violates a positive if all literals in the conjunction $A$ are true in the example, and $B$ is false.

Things get more interesting if we consider covered negatives, since then we need to find one or more clauses to add to the theory in order to exclude the negative. For example, suppose that our current hypothesis covers the negative

$$\mathsf{ManyTeeth} \wedge \mathsf{Gills} \wedge \mathsf{Short} \wedge \neg\mathsf{Beak}$$

To exclude it, we can add the following Horn clause to our theory:

$$\mathsf{ManyTeeth} \wedge \mathsf{Gills} \wedge \mathsf{Short} \rightarrow \mathsf{Beak}$$

While there are other clauses that can exclude the negative (e.g., $\mathsf{ManyTeeth} \rightarrow \mathsf{Beak}$) this is the most specific one, and hence least at risk of also excluding covered positives. However, the most specific clause excluding a negative is only unique if the negative has exactly one literal set to $\mathsf{false}$. For example, if our covered negative is

$$\mathsf{ManyTeeth} \wedge \mathsf{Gills} \wedge \neg\mathsf{Short} \wedge \neg\mathsf{Beak}$$

then we have a choice between the following two Horn clauses:

$$\mathsf{ManyTeeth} \wedge \mathsf{Gills} \rightarrow \mathsf{Short}$$
$$\mathsf{ManyTeeth} \wedge \mathsf{Gills} \rightarrow \mathsf{Beak}$$

Notice that, the fewer literals are set to $\mathsf{true}$ in the negative example, the more general the clauses excluding the negative are.

The approach of Algorithm 4.5 is to add *all* of these clauses to the hypothesis. However, the algorithm applies two clever tricks. The first is that it maintains a list $S$ of negative examples, from which it periodically rebuilds the hypothesis. The second is that, rather than simply adding new negative examples to the list, it tries to find negatives with fewer literals set to true, since this will result in more general clauses. This is possible if we assume we have access to a *membership oracle* $Mb$ which can tell us whether a particular example is a member of the concept we're learning or not. So in line 7 of the algorithm we form the *intersection* of a new negative $x$ and an existing one $s \in S$ – i.e., an example with only those literals set to true which are true in both $x$ and $s$ – and pass the result $z$ to the membership oracle to check whether it belongs to the target concept. The algorithm also assumes access to an *equivalence oracle* $Eq$ which either tells us that our current hypothesis $h$ is logically equivalent to the target formula $f$, or else produces a *counter-example* that can be either a false positive (it is covered by $h$ but not by $f$) or a false negative (it is covered by $f$ but not by $h$).

**Algorithm 4.5:** Horn($Mb$, $Eq$) – learn a conjunction of Horn clauses from membership and equivalence oracles.

```
    Input  : equivalence oracle $Eq$; membership oracle $Mb$.
    Output : Horn theory $h$ equivalent to target formula $f$.
1   $h \leftarrow$ true;                              // conjunction of Horn clauses, initially empty
2   $S \leftarrow \emptyset$ ;                        // a list of negative examples, initially empty
3   while $Eq(h)$ returns counter-example $x$ do
4       if $x$ violates at least one clause of $h$ then          // $x$ is a false negative
5           specialise $h$ by removing every clause that $x$ violates
6       else                                                   // $x$ is a false positive
7           find the first negative example $s \in S$ such that (i) $z = s \cap x$ has fewer true
            literals than $s$, and (ii) $Mb(z)$ labels it as a negative;
8           if such an example exists then replace $s$ in $S$ with $z$, else append $x$ to the
            end of $S$;
9           $h \leftarrow$ true;
10          for all $s \in S$ do                                 // rebuild $h$ from $S$
11              $p \leftarrow$ the conjunction of literals true in $s$;
12              $Q \leftarrow$ the set of literals false in $s$;
13              for all $q \in Q$ do $h \leftarrow h \wedge (p \rightarrow q)$;
14          end
15      end
16  end
17  return $h$
```

---

**Example 4.5 (Learning a Horn theory).** Suppose the target theory $f$ is

$$(\mathsf{ManyTeeth} \land \mathsf{Short} \rightarrow \mathsf{Beak}) \land (\mathsf{ManyTeeth} \land \mathsf{Gills} \rightarrow \mathsf{Short})$$

This theory has 12 positive examples: eight in which ManyTeeth is false; another two in which ManyTeeth is true but both Gills and Short are false; and two more in which ManyTeeth, Short and Beak are true. The negative examples, then, are

n1: $\mathsf{ManyTeeth} \land \mathsf{Gills} \land \mathsf{Short} \land \neg\mathsf{Beak}$
n2: $\mathsf{ManyTeeth} \land \mathsf{Gills} \land \neg\mathsf{Short} \land \mathsf{Beak}$
n3: $\mathsf{ManyTeeth} \land \mathsf{Gills} \land \neg\mathsf{Short} \land \neg\mathsf{Beak}$
n4: $\mathsf{ManyTeeth} \land \neg\mathsf{Gills} \land \mathsf{Short} \land \neg\mathsf{Beak}$

$S$ is initialised to the empty list and $h$ to the empty conjunction. We call the equivalence oracle which returns a counter-example which has to be a false positive (since every example satisfies our initial hypothesis), say n1 which violates the first clause in $f$. There are no negative examples in $S$ yet, so we add n1 to $S$ (step 8 of Algorithm 4.5). We then generate a new hypothesis from $S$ (steps 9–13): $p$ is $\mathsf{ManyTeeth} \land \mathsf{Gills} \land \mathsf{Short}$ and $Q$ is $\{\mathsf{Beak}\}$, so $h$ becomes $(\mathsf{ManyTeeth} \land \mathsf{Gills} \land \mathsf{Short} \rightarrow \mathsf{Beak})$. Notice that this clause is implied by our target theory: if ManyTeeth and Gills are true then so is Short by the second clause of $f$; but then so is Beak by $f$'s first clause. But we need more clauses to exclude all the negatives.

Now, suppose the next counter-example is the false positive n2. We form the intersection with n1 which was already in $S$ to see if we can get a negative example with fewer literals set to true (step 7). The result is equal to n3 so the membership oracle will confirm this as a negative, and we replace n1 in $S$ with n3. We then rebuild $h$ from $S$ which gives ($p$ is $\mathsf{ManyTeeth} \land \mathsf{Gills}$ and $Q$ is $\{\mathsf{Short}, \mathsf{Beak}\}$)

$$(\mathsf{ManyTeeth} \land \mathsf{Gills} \rightarrow \mathsf{Short}) \land (\mathsf{ManyTeeth} \land \mathsf{Gills} \rightarrow \mathsf{Beak})$$

Finally, assume that n4 is the next false positive returned by the equivalence oracle. The intersection with n3 on $S$ is actually a positive example, so instead of intersecting with n3 we append n4 to $S$ and rebuild $h$. This gives the previous two clauses from n3 plus the following two from n4:

$$(\mathsf{ManyTeeth} \land \mathsf{Short} \rightarrow \mathsf{Gills}) \land (\mathsf{ManyTeeth} \land \mathsf{Short} \rightarrow \mathsf{Beak})$$

The first of this second pair will subsequently be removed by a false negative from the equivalence oracle, leading to the final theory

$$\begin{aligned}(\mathsf{ManyTeeth} \wedge \mathsf{Gills} \rightarrow \mathsf{Short}) \wedge \\ (\mathsf{ManyTeeth} \wedge \mathsf{Gills} \rightarrow \mathsf{Beak}) \wedge \\ (\mathsf{ManyTeeth} \wedge \mathsf{Short} \rightarrow \mathsf{Beak})\end{aligned}$$

which is logically equivalent (though not identical) to $f$.

The Horn algorithm combines a number of interesting new ideas. First, it is an *active learning* algorithm: rather than learning from a provided data set, it constructs its own training examples and asks the membership oracle to label them. Secondly, the core of the algorithm is the list of cleverly chosen negative examples, from which the hypothesis is periodically rebuilt. The intersection step is crucial here: if the algorithm just remembered negatives, the hypothesis would consist of many specific clauses. It can be shown that, in order to learn a theory consisting of $m$ clauses and $n$ Boolean variables, the algorithm requires $O(mn)$ equivalence queries and $O(m^2n)$ membership queries. In addition, the runtime of the algorithm is quadratic in both $m$ and $n$. While this is probably prohibitive in practice, the Horn algorithm can be shown to always successfully learn a Horn theory that is equivalent to the target theory. Furthermore, if we don't have access to an equivalence oracle the algorithm is still guaranteed to 'almost always' learn a Horn theory that is 'mostly correct'. This will be made more precise in Section 4.4.

## *Using first-order logic*

Another way to move beyond conjunctive concepts defined by simple features is to use a richer logical language. The languages we have been using so far are *propositional*: each literal is a proposition such as Gills = yes – standing for 'the dolphin has gills' – from which larger expressions are built using logical connectives. *First-order predicate logic*, or first-order logic for short, generalises this by building more complex literals from *predicates* and *terms*. For example, a first-order literal could be BodyPart(Dolphin42, PairOf(Gill)). Here, Dolphin42 and PairOf(Gill) are terms referring to objects: Dolphin42 is a constant, and PairOf(Gill) is a compound term consisting of the function symbol PairOf and the term Gills. BodyPart is a binary predicate forming a proposition (something that can be true or false) out of two terms. This richer language brings with it a number of advantages:

- ☞ we can use terms such as Dolphin42 to refer to individual objects we're interested in;

- ☞ the structure of objects can be explicitly described; and
- ☞ we can introduce variables to refer to unspecified objects and quantify over them.

To illustrate the latter point, the first-order literal $\mathsf{BodyPart}(x, \mathsf{PairOf}(\mathsf{Gill}))$ can be used to refer to the set of all objects having a pair of gills; and the following expression applies universal quantification to state that everything with a pair of gills is a fish:

$$\forall x : \mathsf{BodyPart}(x, \mathsf{PairOf}(\mathsf{Gill})) \rightarrow \mathsf{Fish}(x)$$

Since we modified the structure of literals, we need to revisit notions such as generalisation and LGG. Remember that for propositional literals with internal disjunction we used the function Combine-ID for merging two internal disjunctions: thus, for example, $\mathsf{LGG\text{-}Conj\text{-}ID}(\mathsf{Length} = [3,4], \mathsf{Length} = [4,5])$ returns $\mathsf{Length} = [3,4,5]$. In order to generalise first-order literals we use variables. Consider, for example, the two first-order literals $\mathsf{BodyPart}(\mathsf{Dolphin42}, \mathsf{PairOf}(\mathsf{Gill}))$ and $\mathsf{BodyPart}(\mathsf{Human123}, \mathsf{PairOf}(\mathsf{Leg}))$: these generalise to $\mathsf{BodyPart}(x, \mathsf{PairOf}(y))$,signifying the set of objects that have a pair of some unspecified body part. There is a well-defined algorithm for computing LGGs of first-order literals called *anti-unification*, as it is the mathematical dual of the deductive operation of *unification*.

**Example 4.6 (Unification and anti-unification).** Consider the following terms:

| | |
|---|---|
| $\mathsf{BodyPart}(x, \mathsf{PairOf}(\mathsf{Gill}))$ | describing the objects that have a pair of gills; |
| $\mathsf{BodyPart}(\mathsf{Dolphin42}, \mathsf{PairOf}(y))$ | describing the body parts that Dolphin42 has a pair of. |

The following two terms are their unification and anti-unification, respectively:

| | |
|---|---|
| $\mathsf{BodyPart}(\mathsf{Dolphin42}, \mathsf{PairOf}(\mathsf{Gill}))$ | describing Dolphin42 as having a pair of gills; |
| $\mathsf{BodyPart}(x, \mathsf{PairOf}(y))$ | describing the objects that have a pair of unspecified body parts. |

So we see that in first-order logic literals already have quite a rich structure, owing to the use of variables. We will revisit this in Section 6.4 when we discuss how to learn classification rules in first-order logic.

## 4.4 Learnability

In this chapter we have seen several hypothesis languages for concept learning, including conjunctions of literals (possibly with internal disjunction), conjunctions of Horn clauses, and clauses in first-order logic. It is intuitively clear that these languages differ in expressivity: for example, a conjunction of literals is also a conjunction of Horn clauses with empty if-part, so Horn theories are strictly more expressive than conjunctive concepts. The downside of a more expressive concept language is that it may be harder to learn. The field of computational learning theory studies exactly this question of *learnability*.

To kick things off we need a *learning model*: a clear statement of what we mean if we say that a concept language is learnable. One of the most common learning models is the model of *probably approximately correct* (*PAC*) learning. PAC-learnability means that there exists a learning algorithm that gets it mostly right, most of the time. The model makes an allowance for mistakes on non-typical examples: hence the 'mostly right' or 'approximately correct'. The model also makes an allowance for sometimes getting it completely wrong, for example when the training data contains lots of non-typical examples: hence the 'most of the time' or 'probably'. We assume that typicality of examples is determined by some unspecified probability distribution $D$, and we evaluate the error rate $err_D$ of a hypothesis with respect to this distribution $D$. More formally, for arbitrary allowable error rate $\epsilon < 1/2$ and failure rate $\delta < 1/2$ we require a PAC-learning algorithm to output with probability at least $1-\delta$ a hypothesis $h$ such that $err_D < \epsilon$.

Let's assume for the moment that our data is noise-free, and that the target hypothesis is chosen from our hypothesis language. Furthermore, we assume our learner always outputs a hypothesis that is complete and consistent with the training sample. There is a possibility that this zero training error is misleading, and that the hypothesis is actually a 'bad' one, having a true error over the instance space that is larger than $\epsilon$. We just want to make sure that this happens with probability less than $\delta$. I will now show that this can be guaranteed by choosing the training sample large enough. Suppose our hypothesis space $H$ contains a single bad hypothesis, then the probability it is complete and consistent on $m$ independently sampled training examples is at most $(1-\epsilon)^m$. Since $1-\epsilon \le e^{-\epsilon}$ for any $0 \le \epsilon \le 1$, we have that this probability is at most $e^{-m\epsilon}$. We want this to be at most $\delta$, which can be achieved by setting $m \ge \frac{1}{\epsilon} \ln \frac{1}{\delta}$. Now, $H$ may contain several bad hypotheses, say $k \le |H|$; then the probability that at least one of them is complete and consistent on $m$ independently sampled training examples is at most $k(1-\epsilon)^m \le |H|(1-\epsilon)^m \le |H|e^{-m\epsilon}$, which is at most $\delta$ if

$$m \ge \frac{1}{\epsilon}\left(\ln|H| + \ln\frac{1}{\delta}\right) \tag{4.1}$$

This is called the *sample complexity* of a complete and consistent learner. The good news is that it is linear in $1/\epsilon$ and logarithmic in $1/\delta$. Notice that this suggests that it is exponentially cheaper to reduce the failure rate than it is to reduce the error. Any learning algorithm that takes time polynomial in $1/\epsilon$ and $1/\delta$ to process a single training example will therefore also take polynomial training time, another requirement for PAC-learnability. However, finding a complete and consistent hypothesis is not tractable in many hypothesis languages.

Notice that the term $\ln|H|$ arose because in the worst case almost all hypotheses in $H$ are bad. However, in practice this means that the bound in Equation 4.1 is overly pessimistic. Still, it allows us to see that concept languages whose size is exponential in some parameter $n$ are PAC-learnable. For example, the number of conjunctions over $n$ Boolean variables is $3^n$, since each variable can occur unnegated, negated or not at all. Consequently, the sample complexity is $(1/\epsilon)\,(n\ln 3 + \ln(1/\delta))$. For example, if we set $\delta = 0.05$ and $\epsilon = 0.1$ then the sample complexity is approximately $10(n \cdot 1.1 + 3) = 11n + 30$. For our dolphin example with $n = 4$ this is clearly pessimistic, since there are only $2^4 = 16$ distinct examples! For larger $n$ this is more realistic. Notice also that the PAC model is distribution-free: the learner is not given any information about the instance distribution $D$. This is another source for pessimism in the bound on the sample complexity.

We may not always be able to output a complete and consistent hypothesis: for instance, this may be computationally intractable, the target hypothesis may not be representable in our hypothesis language, or the examples may be noisy. A reasonable strategy would be to choose the hypothesis with lowest training error. A 'bad' hypothesis is then one whose true error exceeds the training error by at least $\epsilon$. Using some results from probability theory, we find that this probability is at most $e^{-2m\epsilon^2}$. As a result, the $1/\epsilon$ factor in Equation 4.1 is replaced by $1/2\epsilon^2$: for $\epsilon = 0.1$ we thus need 5 times as many training examples compared to the previous case.

It has already been mentioned that the $|H|$ term is a weak point in the above analysis. What we need is a measure that doesn't just count the size of the hypothesis space, but rather gives its expressivity or capacity in terms of classification. Such a measure does in fact exist and is called the *VC-dimension* after its inventors Vladimir Vapnik and Alexey Chervonenkis. We will illustrate the main idea by means of an example.

---

**Example 4.7 (Shattering a set of instances).** Consider the following instances:

$m =$ ManyTeeth $\land$ $\neg$Gills $\land$ $\neg$Short $\land$ $\neg$Beak

$g =$ $\neg$ManyTeeth $\land$ Gills $\land$ $\neg$Short $\land$ $\neg$Beak

$s =$ $\neg$ManyTeeth $\land$ $\neg$Gills $\land$ Short $\land$ $\neg$Beak

$b =$ ¬ManyTeeth ∧ ¬Gills ∧ ¬Short ∧ Beak

There are 16 different subsets of the set $\{m, g, s, b\}$. Can each of them be represented by its own conjunctive concept? The answer is yes: for every instance we want to exclude, we add the corresponding negated literal to the conjunction. Thus, $\{m, s\}$ is represented by ¬Gills ∧ ¬Beak, $\{g, s, b\}$ is represented by ¬ManyTeeth, $\{s\}$ is represented by ¬ManyTeeth ∧ ¬Gills ∧ ¬Beak, and so on. We say that this set of four instances is *shattered* by the hypothesis language of conjunctive concepts.

The VC-dimension is the size of the largest set of instances that can be shattered by a particular hypothesis language or model class. The previous example shows that the VC-dimension of conjunctive concepts over $d$ Boolean literals is at least $d$. It is in fact equal to $d$, although this is harder to prove (since it involves showing that no set of $d+1$ instances can be shattered). This measures the capacity of the model class for representing concepts or binary classifiers. As another example, the VC-dimension of a linear classifier in $d$ dimensions is $d+1$: a threshold on the real line can shatter two points but not three (since the middle point cannot be separated from the other two by a single threshold); a straight line in a two-dimensional space can shatter three points but not four; and so on.

The VC-dimension can be used to bound the difference between sample error and true error of a hypothesis (which is the step where $|H|$ appeared in our previous arguments). Consequently, it can also be used to derive a bound on the sample complexity of a complete and consistent learner in terms of the VC-dimension $D$ rather than $|H|$:

$$m \geq \frac{1}{\epsilon}\max\left(8D\log_2\frac{13}{\epsilon}, 4\log_2\frac{2}{\delta}\right) \tag{4.2}$$

We see that the bound is linear in $D$, where previously it was logarithmic in $|H|$. This is natural, since to shatter $D$ points we need at least $2^D$ hypotheses, and so $\log_2|H| \geq D$. Furthermore, it is still logarithmic in $1/\delta$, but linear times logarithmic in $1/\epsilon$. Plugging in our previous values of $\delta = 0.05$ and $\epsilon = 0.1$, we obtain a sample complexity of $\max(562 \cdot D, 213)$.

We conclude that the VC-dimension allows us to derive the sample complexity of infinite concept classes, as long as they have finite VC-dimension. It is furthermore worth mentioning a classical result from computational learning theory which says that a concept class is PAC-learnable if and only if its VC-dimension is finite.

## 4.5 Concept learning: Summary and further reading

In this chapter we looked at methods for inductive concept learning: the process of constructing a logical expression defining a set of objects from examples. This problem was a focus of early work in artificial intelligence (Winston, 1970; Vere, 1975; Banerji, 1980), following the seminal work by psychologists Bruner, Goodnow and Austin (1956) and Hunt, Marin and Stone (1966).

- ☞ In Section 4.1 we considered the structure of the hypothesis space: the set of possible concepts. Every hypothesis has an extension (the set of instances it covers), and thus relationships between extensions such as subset relationships carry over to the hypothesis space. This gives the hypothesis space a lattice structure: a partial order with least upper bounds and greatest lower bounds. In particular, the LGG is the least upper bound of a set of instances, and is the most conservative generalisation that we can learn from the data. The concept was defined in the context of first-order logic by Plotkin (1971), who showed that it was the mathematical dual of the deductive operation of unification. We can extend the hypothesis language with internal disjunction among values of a feature, which creates a larger hypothesis space that still has a lattice structure. Internal disjunction is a common staple of attribute-value languages for learning following the work of Michalski (1973). For further pointers regarding hypothesis language and hypothesis space the reader is referred to (Blockeel, 2010*a*,*b*).

- ☞ Section 4.2 defined complete and consistent hypotheses as concepts that cover all positive examples and no negative examples. The set of complete and consistent concepts is called the version space, a notion introduced by Mitchell (1977). The version space can be summarised by its least general and most general members, since any concept between one least general hypothesis and another most general one is also complete and consistent. Alternatively, we can describe the version space by all paths from a least general to a most general hypothesis. Such upward paths give rise to a coverage curve which describes the extension of each concept on the path in terms of covered positives and negatives. Concept learning can then be seen as finding an upward path that goes through ROC heaven. Syntactically different concepts can have the same extension in a particular data set: a closed concept is the most specific one of these (technically, the LGG of the instances in its extension). The notion is studied in formal concept analysis (Ganter and Wille, 1999) and was introduced in a data mining context by Pasquier, Bastide, Taouil and Lakhal (1999); Garriga, Kralj and Lavrač (2008) investigate its usefulness for labelled data.

- ☞ In Section 4.3 we discussed the Horn algorithm for learning concepts described by conjunctions of Horn rules, first published in Angluin *et al.* (1992). The algorithm makes use of a membership oracle, which can be seen as an early form of active learning (Cohn, 2010; Dasgupta, 2010). Horn theories are superficially similar to classification rule models which will be studied in Chapter 6. However, there is an important difference, since those classification rules have the target variable in the then-part of the rule, while the Horn clauses we are looking at here can have any literal in the then-part. In fact, in this chapter the target variable is not part of the logical language at all. This setting is sometimes called *learning from interpretations*, since examples are truth-value assignments to our theory. The classification rule setting is called *learning from entailment*, since in order to find out whether a particular rule covers an example we need to apply logical inference. De Raedt (1997) explains and explores the differences between these two settings. Further introductions to first-order logic and its use in learning are given by Flach (2010*a*) and De Raedt (2010).

☞ Section 4.4 briefly reviewed some basic concepts and results in learnability theory. My account partly followed Mitchell (1997, Chapter 7); another excellent introduction is given by Zeugmann (2010). PAC-learnability, which allows an error rate of $\epsilon$ and a failure rate of $\delta$, was introduced in a seminal paper by Valiant (1984). Haussler (1988) derived the sample complexity for complete and consistent learners (Equation 4.1), which is linear in $1/\epsilon$ and logarithmic in $1/\delta$ and the size of the hypothesis space. The VC-dimension as a measure of the capacity of a hypothesis language was introduced by Vapnik and Chervonenkis (1971) in order to quantify the difference between training error and true error. This allows a statement of the sample complexity in terms of the VC-dimension (Equation 4.2) which is due to Blumer, Ehrenfeucht, Haussler and Warmuth (1989). This same paper proved that a model class is PAC-learnable if and only if its VC-dimension is finite.