3

An Introduction to Learning and Search

In this chapter, we introduce machine learning and data mining problems, and argue that they can be viewed as search problems. Within this view, the goal is to find those hypotheses in the search space that satisfy a given quality criterion or minimize a loss function. Several quality criteria and loss functions, such as consistency (as in concept learning) and frequency (in association rule mining) are presented, and we investigate desirable properties of these criteria, such as monotonicity and anti-monotonicity. These properties are defined w.r.t. the is more general than *relation and allow one to prune the search for solutions. We also outline several algorithms that exploit these properties.*

3.1 Representing Hypotheses and Instances

In Chapter 1, we presented several showcase applications of logical and relational learning. We also used these cases to introduce the tasks addressed by machine learning and data mining in an informal though general way. Recall that data mining was viewed as the task of finding all patterns expressible within a language of hypotheses satisfying a particular quality criterion. On the other hand, machine learning was viewed as the problem of finding that function within a language of hypotheses that minimizes a loss function. Within this view, machine learning becomes the problem of *function approximation*. Inspecting these views reveals that they are fairly close to one another, and that there are many common issues when looking at symbolic machine learning and data mining.

One of these issues is concerned with knowledge representation. How should patterns, functions, hypotheses and data be represented? It will be useful to distinguish different representation languages for data (instances or examples) and hypotheses (functions, concepts or patterns). Therefore, we assume there is

- a *language of examples* $\mathcal{L}_e$, whose elements are descriptions of instances, observations or data, and

- a *language of hypotheses* $\mathcal{L}_h$, whose elements describe hypotheses (functions or patterns) about the instances, observations or data.

In many situations, it is helpful to employ background knowledge in the mining and learning process. However, for ease of exposition, we postpone the discussion of background knowledge to Section 4.9.

The goal of data mining and machine learning is then to discover hypotheses that provide information about the instances. This implies that the relationship between the language of examples $\mathcal{L}_e$ and of hypotheses $\mathcal{L}_h$ must be known. This relationship can be modeled elegantly by viewing hypotheses $h \in \mathcal{L}_h$ as functions $h : \mathcal{L}_e \rightarrow \mathcal{Y}$ to some domain $\mathcal{Y}$. The learning task is then to approximate an unknown target function f well. This view is illustrated in Fig. 3.1. Different domains are natural for different learning and mining tasks. For instance, in *regression*, the task is to learn a function from $\mathcal{L}_e$ to $\mathcal{Y} = \mathbb{R}$, that is, to learn a real-valued function. As an illustration, consider that we want to learn to assign (real-valued) activities to a set of molecules. On the other hand, when learning definitions of concepts or mining for local patterns, $\mathcal{Y} = \{0, 1\}$ or, equivalently, $\mathcal{Y} = \{true, false\}$. In concept learning, the task could be to learn a description that matches all and only the active molecules. The resulting description is then the concept description.

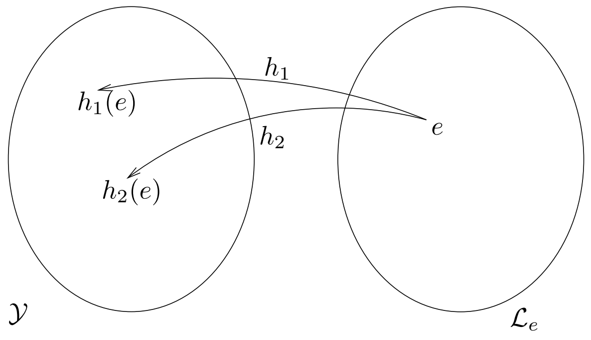

Fig. 3.1. Hypotheses viewed as functions

When the domain of the hypotheses is binary, that is, when $\mathcal{Y} = \{0, 1\}$, it is useful to distinguish the instances that are *covered* by a hypothesis, that is, mapped to 1, from those that are not. This motivates the following definition:

Definition 3.1. *The covers relation* $\mathbf{c}$ *is a relation over* $\mathcal{L}_h \times \mathcal{L}_e$*, and* $\mathbf{c}(h, e) =$ *true if and only if* $h(e) = 1$.

Thus the covers relation corresponds to a kind of matching relation. We will sometimes write $\mathbf{c}(h)$ to denote the set of examples in $\mathcal{L}_e$ covered by the hypothesis $h \in \mathcal{L}_h$. Furthermore, the set of examples from $D \subseteq \mathcal{L}_h$ covered by a hypothesis h will sometimes be denoted as $\mathbf{c}(h, D)$. So, $\mathbf{c}(h) = co(h, \mathcal{L}_e)$. This relation is graphically illustrated in Figure 3.2.

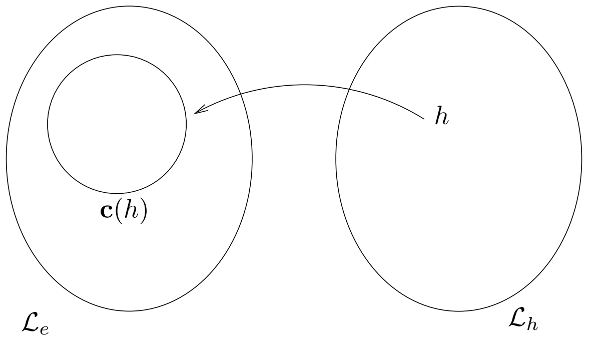

Fig. 3.2. The covers relation

Different notions of coverage as well as choices for $\mathcal{L}_e$ and $\mathcal{L}_h$ can be made. For logical and relational learning, this will be extensively discussed in the next chapter. For the present chapter, however, we will focus on using simple boolean or item-set representations that are so popular in machine learning and data mining. Because these representations are so simple they are ideal for introducing machine learning and data mining problems and algorithms.

3.2 Boolean Data

Due to their simplicity, boolean representations are quite popular within computational learning theory and data mining, where they are better known under the name *item-sets*. In boolean learning, an example is an interpretation over propositional predicates. Recall that this is an assignment of the truth-values $\{true, false\}$ to a set of propositional variables. In the terminology of boolean logic, Herbrand interpretations are often called *variable assignments*.

One of the most popular data mining tasks involving boolean data is that of basket analysis.

Example 3.2. In basket analysis, the aim is to analyze the purchases of clients in, for instance, a supermarket. There is one propositional variable for each of the products available in the supermarket. Assume we have the following set of products $\mathcal{I} = \{\mathsf{sausage}, \mathsf{beer}, \mathsf{wine}, \mathsf{mustard}\}$.

Consider then that the client buys sausage, beer and mustard. This corresponds to the interpretation or item-set $\{\mathsf{sausage}, \mathsf{beer}, \mathsf{mustard}\}$. In this case, the language of examples is

$$\mathcal{L}_e = \{I | I \subseteq \{\mathsf{sausage}, \mathsf{beer}, \mathsf{mustard}, \mathsf{wine}\}\}$$

For boolean data, various types of hypotheses languages have been employed. Perhaps, the most popular one is that of conjunctive expressions of the form $p_1 \wedge \ldots \wedge p_n$ where the p_i are propositional atoms. In the data mining literature, these expressions are also called *item-sets* and usually represented as $\{p_1, \cdots, p_n\}$; in the literature on computational learning theory [Kearns and Vazirani, 1994] they are known as *monomials.* So, in this case: $\mathcal{L}_h = \mathcal{L}_e$, which is sometimes called the *single-representation trick.* Using clausal logic, item-sets can be represented by the set of facts $\{p_1 \leftarrow, \ldots, p_n \leftarrow\}$, though this notation is less convenient because it is too lengthy. It will be convenient to use the notation $\mathcal{L}_\mathcal{I}$ to denote all item-sets or conjunctive expressions over $\mathcal{I}$, the set of all items. More formally,

$$\mathcal{L}_\mathcal{I} = \{I | I \subseteq \mathcal{I}\} \tag{3.1}$$

Continuing the basket analysis example above, the hypothesis that someone buys $\mathsf{mustard}$ and beer could be represented using $\mathsf{mustard} \leftarrow$ and $\mathsf{beer} \leftarrow$, or more compactly as $\{\mathsf{mustard}, \mathsf{beer}\}$. It is easily verified that this hypothesis covers the example $\{\mathsf{sausage}, \mathsf{beer}, \mathsf{mustard}\}$. The clause $\mathsf{mustard} \leftarrow \mathsf{sausage}, \mathsf{beer}$ describes an association rule, that is, a particular kind of pattern. It states than if a client buys beer and $\mathsf{sausage}$ she also buys $\mathsf{mustard}$. When the coverage relation is chosen to coincide with the notion of satisfiability, the example is covered by the clause.

When using purely logical descriptions, the function represented by a hypothesis is typically boolean. However, for the domain of item-sets it is also possible to specify real-valued functions. Consider, for instance, the function

$$h(e) = \mathsf{sausage} + 2 \times \mathsf{beer} + 4 \times \mathsf{wine} + \mathsf{mustard}$$

that computes the price of the basket e.

3.3 Machine Learning

The fundamental problem studied in machine learning is that of function approximation. In this setting, it is assumed that there is an unknown target function $f : \mathcal{L}_e \rightarrow \mathcal{Y}$, which maps instances in $\mathcal{L}_e$ to values in $\mathcal{Y}$. In addition, a set of examples E of the input-output behavior of f is given. The task is then to find a hypothesis $h \in \mathcal{L}_h$ that approximates f well as measured by a so-called *loss* function.

Given

- a language of examples $\mathcal{L}_e$;
- a language of hypotheses $\mathcal{L}_h$;
- an unknown target function $f : \mathcal{L}_e \rightarrow \mathcal{Y}$;
- a set of examples $E = \{(e_1, f(e_1)), \cdots, (e_n, f(e_n))\}$ where each $e_i \in \mathcal{L}_e$;

- a loss function $loss(h, E)$ that measures the quality of hypotheses $h \in \mathcal{L}_h$ w.r.t. the data E;

Find the hypothesis $h \in \mathcal{L}_h$ that minimizes the loss function, that is, for which

$$h = \arg\min loss(h, E) \tag{3.2}$$

As already indicated, various machine learning tasks can be obtained by varying $\mathcal{Y}$. In the simplest case of *binary classification* or *concept learning*, $\mathcal{Y} = \{1, 0\}$, and the task is to learn how to discriminate positive from negative examples. When working with item-sets, this could be baskets that are profitable or not. A natural loss function for this task minimizes the *empirical risk*:

$$loss_{er}(E, h) = \frac{1}{|E|} \sum_i |f(e_i) - h(e_i)| \tag{3.3}$$

So, minimizing the empirical risk corresponds to minimizing the number of errors made on the training data E. Note, however, that minimizing the empirical risk does not guarantee that the hypothesis will also have a high accuracy on unseen data. This view on classification can easily be generalized to take into account more than two classes.

A *regression* setting is obtained by choosing $\mathcal{Y} = \mathbb{R}$. The task is then to learn to predict real values for the examples. As an example of such a function, consider learning a function that predicts the profit the shop makes on a basket. The most popular loss function for regression minimizes the sum of the squared errors, the so-called *least mean squares* loss function:

$$loss_{lms}(E, h) = \sum_i (f(e_i) - h(e_i))^2 \tag{3.4}$$

Finally, in a probabilistic setting, the function to be approximated can be replaced by a probability distribution or density. A popular criterion in this case is to maximize the (log) likelihood of the data; cf. Chapter 8.

This view of machine learning as function approximation will be useful especially in later chapters, such as Chapter 8 on probabilistic logic learning and Chapter 9 on distance and kernel-based learning.

3.4 Data Mining

The purpose of most common data mining tasks is to find hypotheses (expressible within $\mathcal{L}_h$) that satisfy a given quality criterion $\mathcal{Q}$. The quality criterion $\mathcal{Q}$ is then typically expressed in terms of the coverage relation **c** and the data set D. This can be formalized in the following definition:

Given

- a language of examples $\mathcal{L}_e$;
- a language of hypotheses (or patterns) $\mathcal{L}_h$;
- a data set $D \subseteq \mathcal{L}_e$; and
- a quality criterion $\mathcal{Q}(h, D)$ that specifies whether the hypothesis $h \in \mathcal{L}_h$ is acceptable w.r.t. the data set D;

Find the set of hypotheses

$$Th(\mathcal{Q}, D, \mathcal{L}_h) = \{h \in \mathcal{L}_h \mid \mathcal{Q}(h, D) \textit{ is true}\} \tag{3.5}$$

When the context is clear, we will often abbreviate $Th(\mathcal{Q}, D, \mathcal{L}_h)$ as Th. This definition has various special cases and variants. First, the data mining task can be to find *all* elements, k elements or just one element that satisfies the quality criterion $\mathcal{Q}$. Second, a large variety of different quality criteria are in use. These can be distinguished on the basis of their *global*, *local* or *heuristic* nature. *Local* quality criteria are predicates whose truth-value is a function of the hypothesis h, the covers relation **c** and the data set D only. On the other hand, a *global* quality criterion is not only a function of the hypothesis h, the covers relation **c** and the data set D, but also of the other hypotheses in $\mathcal{L}_h$.

One function that is commonly used in data mining is that of frequency. The frequency $freq(h, D)$ of a hypothesis h w.r.t. a data set D is the cardinality of the set **c**(h,D):

$$freq(h, D) = \mid \mathbf{c}(h, D) \mid \tag{3.6}$$

In this definition, the *absolute* frequency is expressed in absolute terms, that is, the frequency is a natural number. Sometimes, frequency is also expressed *relatively* to the size of the data set D. Thus the *relative* frequency is

$$rfreq(h, D) = \frac{freq(h, D)}{\mid D \mid} \tag{3.7}$$

An example of a local quality criterion is now a minimum frequency constraint. Such a constraint states that the frequency of a hypothesis h on the data set D should exceed a threshold, that is, $\mathcal{Q}(h, D)$ is of the form $freq(h, D) > x$ where x is a natural number or $rfreq(h, D) > y$ where y is a real number between 0 and 1. These criteria are local because one can verify whether they hold by accessing the hypothesis h and D only. There is no need to know the frequency of the other hypotheses in $\mathcal{L}_h$.

An example of a global quality criterion is to require that the accuracy of a hypothesis h w.r.t. a set of positive P and negative example N is maximal. The accuracy $acc(h, P, N)$ is then defined as

$$acc(h, P, N) = \frac{freq(h, P)}{freq(h, P) + freq(h, N)}. \tag{3.8}$$

The maximal accuracy constraint now states

$$\mathcal{Q}(h, P, N) = \big(h = \arg\max_{h \in \mathcal{L}_h} acc(h, P, N)\big) \tag{3.9}$$

This constraint closely corresponds to minimizing the empirical loss in a function approximation setting.

Because the machine learning and data mining views are quite close to one another, at least when working with symbolic representations, we shall in the present chapter largely employ the data mining perspective. When shifting our attention to take into account more numerical issues, in Chapter 8 on probabilistic logic learning and Chapter 9 on distance and kernel-based learning, the machine learning perspective will be more natural. The reader must keep in mind though, that in most cases the same principles apply and are, to some extent, a matter of background or perspective.

3.5 A Generate-and-Test Algorithm

Depending on the type and nature of the quality criterion considered, different algorithms can be employed to compute $Th(\mathcal{Q}, D, \mathcal{L}_h)$. For a given quality criterion and hypotheses space, one can view mining or learning as a search process. By exploiting this view, a (trivial) algorithm based on the well-known generate-and-test technique in artificial intelligence can be derived. This so-called *enumeration algorithm* is shown in Algo. 3.1.

Algorithm 3.1 The enumeration algorithm

```
for all $h \in \mathcal{L}_h$ do
  if $\mathcal{Q}(h, D) = true$ then
    output $h$
  end if
end for
```

Although the algorithm is naive, it has some interesting properties: whenever a solution exists, the enumeration algorithm will find it. The algorithm can only be applied if the hypotheses language $\mathcal{L}_h$ is *enumerable*, which means that it must be possible to generate all its elements. As the algorithm searches the whole space, it is inefficient. This is a well-known property of generate-and-test approaches. Therefore, it is advantageous to structure the search space in machine learning, which will allow for its pruning. Before discussing how the search space can be structured, let us illustrate the enumeration algorithm. This illustration, as well as most other illustrations and examples in this chapter, employs the representations of boolean logic.

Example 3.3. Reconsider the problem of basket analysis sketched in Ex. 3.2. In basket analysis, there is a set of propositional variables (usually called items)

$\mathcal{I} = \{\mathsf{s} = \mathsf{sausage}, \mathsf{m} = \mathsf{mustard}, \mathsf{b} = \mathsf{beer}, \mathsf{c} = \mathsf{cheese}\}$. Furthermore, every example is an interpretation (or item-set) and the hypotheses are, as argued in Ex. 3.2, members of $\mathcal{L}_{\mathcal{I}}$. Consider also the data set

$$D = \{\{\mathsf{s}, \mathsf{m}, \mathsf{b}, \mathsf{c}\}, \{\mathsf{s}, \mathsf{m}, \mathsf{b}\}, \{\mathsf{s}, \mathsf{m}, \mathsf{c}\}, \{\mathsf{s}, \mathsf{m}\}\}$$

and the quality criterion $\mathcal{Q}(h, D) = (freq(h, D) \geqslant 3)$. One way of enumerating all item-sets in $\mathcal{L}_{\mathcal{I}}$ for our example is given in Fig. 3.3. Furthermore, the item-sets satisfying the constraint are underlined.

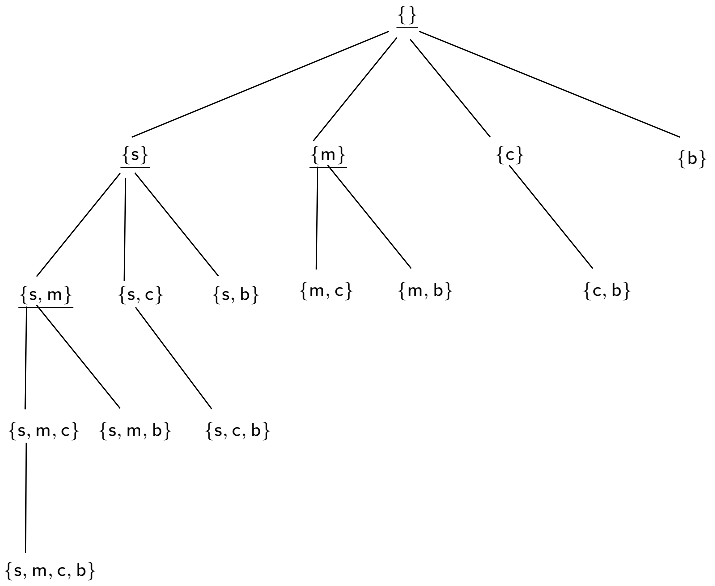

Fig. 3.3. Enumerating and testing monomials or item-sets

3.6 Structuring the Search Space

One natural way to structure the search space is to employ the *generality* relation.

Definition 3.4. *Let $h_1, h_2 \in \mathcal{L}_h$. Hypothesis h_1* is more general than *hypothesis h_2, notation $h_1 \preceq h_2$, if and only if all examples covered by h_2 are also covered by h_1, that is, $\mathbf{c}(h_2) \subseteq \mathbf{c}(h_1)$.*

We also say that h_2 is a specialization of h_1, h_1 is a generalization of h_1 or h_2 is more general than h_1.[1] This notion is illustrated in Fig. 3.4. Furthermore,

[1] It would be more precise to state that h_2 is at least as general than h_1. Nevertheless, we shall use the standard terminology, and say that h_1 is more general than h_2.

when $h_1 \preceq h_2$ but h_1 covers examples not covered by h_2 we say that h_1 is a proper generalization of h_2, and we write $h_1 \prec h_2$.

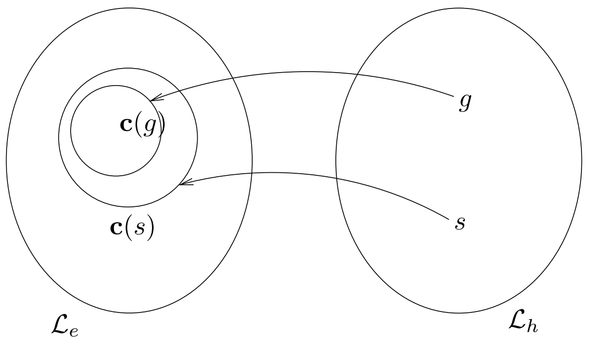

Fig. 3.4. Hypothesis g is more general than hypothesis s

Notice that the generality relation is transitive and reflexive. Hence, it is a *quasi-order*. Unfortunately, it is not always anti-symmetric since there may exist several hypotheses that cover exactly the same set of examples. Such hypotheses are called *syntactic variants*. Syntactic variants are undesirable because they introduce redundancies in the search space. In theory, one can obtain a *partial order* by introducing equivalence classes and working with a canonical form as a representative of the equivalence class. In practice, this is not always easy, as will be explained in the next chapter.

Example 3.5. Consider the task of basket analysis used in Ex. 3.3. The conjunction $\mathsf{sausage} \land \mathsf{beer}$ is more general than $\mathsf{sausage} \land \mathsf{beer} \land \mathsf{cheese}$, or when using set notation $\{\mathsf{sausage}, \mathsf{beer}\}$ is more general than $\{\mathsf{sausage}, \mathsf{beer}, \mathsf{cheese}\}$ because the former is a subset of the latter.

Furthermore, if we would possess background knowledge in the form of a taxonomy stating, for instance, that $\mathsf{alcohol} \leftarrow \mathsf{beer}$; $\mathsf{food} \leftarrow \mathsf{cheese}$; $\mathsf{food} \leftarrow \mathsf{sausage}$; and $\mathsf{food} \leftarrow \mathsf{mustard}$, then the conjunction $\mathsf{food} \land \mathsf{beer}$ together with the background theory would be more general than $\mathsf{sausage} \land \mathsf{beer}$. Using the taxonomy, specific baskets such as $\{\mathsf{sausage}, \mathsf{beer}\}$ can be completed under the background theory, by computing the least Herbrand model of the item-set and the background theory, yielding in our example $\{\mathsf{sausage}, \mathsf{beer}, \mathsf{food}, \mathsf{alcohol}\}$. This is the learning from interpretations setting, that we shall discuss extensively in the next chapter. Whenever an example contains sausage in this setting the resulting completed example will contain food as well.

Continuing the illustration, if we assume that the examples only contain the items from $\mathcal{I}$ and are then completed using the clauses listed above, then the conjunctions $\mathsf{alcohol} \land \mathsf{cheese}$ and $\mathsf{beer} \land \mathsf{cheese}$ are syntactic variants, because there is only one type of item belonging to the category alcohol.

When the language $\mathcal{L}_h$ does not possess syntactic variants, which will be assumed throughout the rest of this chapter, the generality relation imposes a *partial order* on the search space and can be graphically depicted using a so called Hasse diagram. This is illustrated in Fig. 3.5.

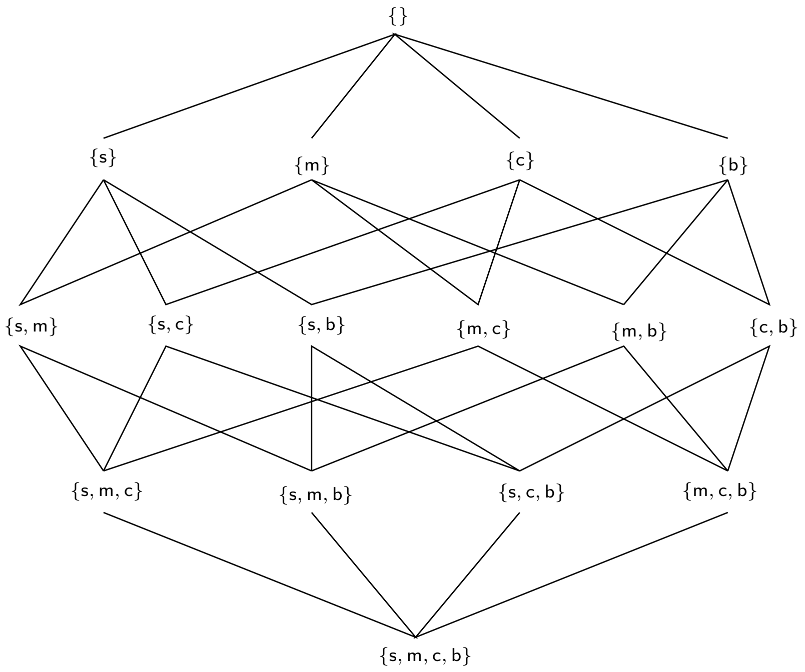

Fig. 3.5. The partial order over the item-sets

It is often convenient to work with a special notation for the maximally general top element $\top$ and the maximally specific bottom element $\bot$ such that $\mathbf{c}(\top) = \mathcal{L}_e$ and $\mathbf{c}(\bot) = \emptyset$. Furthermore, when the elements $\top$ and $\bot$ do not exist in $\mathcal{L}_h$ they are often added to the language. For item-sets, $\top = \emptyset$ and $\bot = \mathcal{I}$.

3.7 Monotonicity

The *generality* relation imposes a useful structure on the search space provided that the quality criterion involves monotonicity or anti-monotonicity.

A quality criterion $\mathcal{Q}$ is *monotonic* if and only if

$$\forall s, g \in \mathcal{L}_h, \forall\ D \subseteq \mathcal{L}_e : (g \preceq s) \wedge \mathcal{Q}(g, D) \rightarrow \mathcal{Q}(s, D) \tag{3.10}$$

It is *anti-monotonic*[2] if and only if

$$\forall s, g \in \mathcal{L}_h, \forall\ D \subseteq \mathcal{L}_e : (g \preceq s) \wedge \mathcal{Q}(s, D) \rightarrow \mathcal{Q}(g, D) \tag{3.11}$$

To illustrate this definition, observe that a minimum frequency constraint $freq(h, D) \geqslant x$ is anti-monotonic and a maximum frequency constraint $freq(h, D) \leqslant x$ is monotonic. Similarly, the criterion that requires that a given example be covered (that is, $e \in \mathbf{c}(h)$) is anti-monotonic and the one that requires that a given example is not covered (that is, $e \notin \mathbf{c}(h)$) is monotonic. On the other hand, the criterion $acc(h, P, N) \geqslant x$ is neither monotonic nor anti-monotonic.

Exercise 3.6. Let $A_1(h, D)$ and $A_2(h, D)$ be two anti-monotonic criteria, and $M_1(h, D)$ and $M_2(h, D)$ be two monotonic ones. Are the criteria $\neg A_1(h, D)$; $A_1(h, D) \vee A_2(h, D)$; $A_1(h, D) \wedge A_2(h, D)$; their duals $\neg M_1(h, D)$; $M_1(h, D) \vee M_2(h, D)$; $M_1(h, D) \wedge M_2(h, D)$; and the combinations $A_1(h, D) \wedge M_1(h, D)$ and $A_1(h, D) \vee M_1(h, D)$ monotonic and/or anti-monotonic? Argue why.

Exercise 3.7. Show that the criterion $acc(h, P, N) \geqslant x$ is neither monotonic nor anti-monotonic.

Exercise 3.8. * Consider the primitives $free(m)$ for item-sets, which is true if and only if none of the subsets of m have the same frequency as m, and $closed(m)$, which is true if and only if none of the super-sets of m have the same frequency as m. Do freeness and closedness satisfy the anti-monotonicity or monotonicity property? Argue why.

When the quality criterion is monotonic or anti-monotonic it is a good idea to employ the generality relation on the search space and to use specialization or generalization as the basic operations to move through the search space. The reason for this is given by the following two properties, which allow us to prune the search.

Property 3.9. (Prune generalizations) If a hypothesis h does not satisfy a monotonic quality criterion then none of its generalizations will.

Property 3.10. (Prune specializations) If a hypothesis h does not satisfy an anti-monotonic quality criterion then none of its specializations will.

These properties directly follow from the definitions of monotonicity and anti-monotonicity in Eqs. 3.10 and 3.11.

Example 3.11. Reconsider Ex. 3.3 and the anti-monotonic minimum frequency criterion. Because sausage $\wedge$ beer does not satisfy the minimum frequency constraint, none of its specializations do. They can therefore be pruned away, as illustrated in Fig. 3.7.

[2] In the literature, the definitions of the concepts of monotonicity and anti-monotonicity are sometimes reversed.

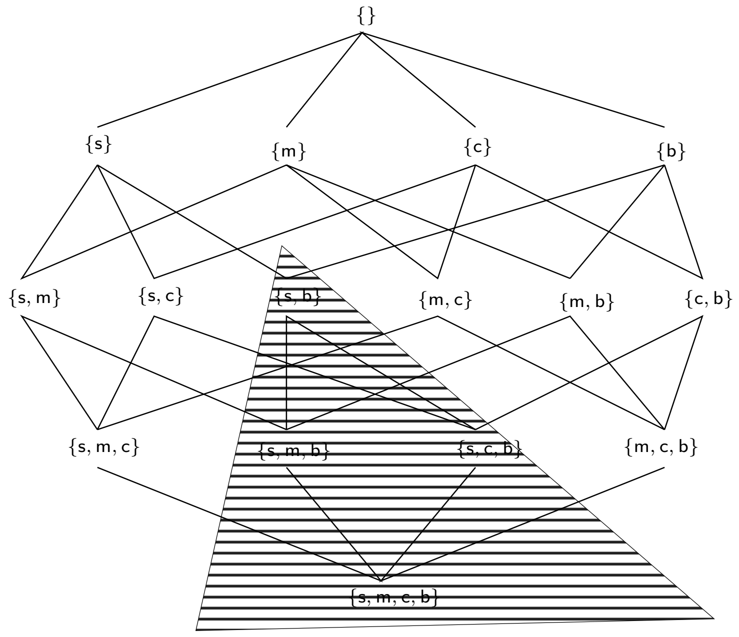

Fig. 3.6. Pruning specializations

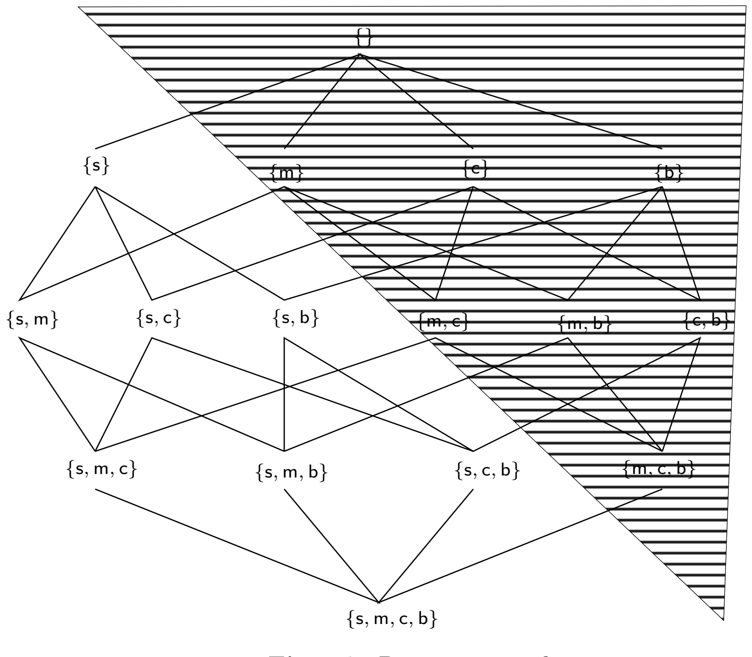

Fig. 3.7. Pruning generalizations

Example 3.12. Reconsider Ex. 3.3 and the monotonic constraint that requires that the example $\{\mathsf{m}, \mathsf{b}, \mathsf{c}\}$ not be covered. Because $\mathsf{mustard} \wedge \mathsf{beer} \wedge \mathsf{cheese}$ covers this example, all its generalizations can be pruned away as illustrated in Fig. 3.7.

3.8 Borders

When monotonic and/or anti-monotonic criteria are used, the solution space has so-called borders. Before introducing borders, let us introduce the $max(T)$ and $min(T)$ primitives:

$$max(T) = \{h \in T \mid \neg\exists t \in T : h \prec t\} \tag{3.12}$$

$$min(T) = \{h \in T \mid \neg\exists t \in T : t \prec h\} \tag{3.13}$$

Intuitively, the maximal elements are the most specific ones. These are also the largest ones when interpreting the symbol $\prec$ as smaller than or equal to. Furthermore, more specific hypotheses are typically also longer.

Example 3.13. Let $T = \{\mathsf{true},\ \mathsf{s},\ \mathsf{m},\ \mathsf{s} \wedge \mathsf{m}\}$. Then $max(T) = \{\mathsf{s} \wedge \mathsf{m}\}$ and $min(T) = \{\mathsf{true}\}$.

Observe that when the hypothesis space $\mathcal{L}_h$ is finite, $max(T)$ and $min(T)$ always exist. When $\mathcal{L}_h$ is infinite, this need not be the case. We illustrate this using string patterns.

Example 3.14. * Many data sets can be conveniently represented using strings over some alphabet Σ; cf. also Chapter 4. An *alphabet* Σ is a finite set of symbols. A *string* $s_1 s_2 ... s_n$ is then a sequence of symbols $s_i \in \Sigma$. For instance, the string over the alphabet $\Sigma = \{\mathsf{a}, \mathsf{c}, \mathsf{g}, \mathsf{t}\}$

$$\mathsf{atgcccaagctgaatagcgtagaggggttttcatcatttgaggacgatgtataa}$$

might represent a sequence of DNA. When working with strings to represent patterns and examples, a natural coverage relation is provided by the notion of substring. A string $S = s_1 s_2 ... s_n$ is a *substring* of a string $T = t_1 t_2 ... t_k$, if and only if $s_1 ... s_n$ occur at consecutive positions in $t_1 ... t_k$, that is, there exists a j for which $s_1 = t_j$, $s_2 = t_{j+1}$, ..., and $s_n = t_{j+n}$. For instance, the string *atgc* is a substring of *aatgccccc* with $j = 2$. For the language of all strings over the alphabet Σ, that is, Σ^*, $max(\Sigma^*)$ does not exist. To avoid such complications in this chapter, we assume that $\mathcal{L}_h$ is finite.

For finite languages and monotonic and anti-monotonic quality criteria $\mathcal{Q}$, the solution space $Th(\mathcal{Q}, D, \mathcal{L}_h)$ has boundary sets that are sometimes called borders. More formally, the S-border of maximally specific solutions w.r.t. a constraint $\mathcal{Q}$ is

$$S\big(Th(\mathcal{Q}, D, \mathcal{L}_h)\big) = max\big(Th(\mathcal{Q}, D, \mathcal{L}_h)\big) \tag{3.14}$$

Dually, the G-border of maximally general solutions is

$$G\big(Th(\mathcal{Q}, D, \mathcal{L}_h)\big) = min\big(Th(\mathcal{Q}, D, \mathcal{L}_h)\big) \tag{3.15}$$

Example 3.15. Reconsider Ex. 3.13. The set T is the set of solutions to the mining problem of Ex. 3.3, that is, $T = Th((freq(h, D) \geqslant 3), D, \mathcal{L}_m)$; $S(T) = max(T) = \{\mathsf{s} \wedge \mathsf{m}\}$ and $G(T) = min(T) = \{\mathsf{true}\}$.

The S and G sets are called borders because of the following properties.

Property 3.16. If $\mathcal{Q}$ is an anti-monotonic predicate, then

$$Th(\mathcal{Q}, D, \mathcal{L}_h) = \{h \in \mathcal{L}_h \mid \exists s \in S\big(Th(\mathcal{Q}, D, \mathcal{L}_h)\big) : h \preceq s\}$$

Property 3.17. If $\mathcal{Q}$ is a monotonic predicate, then

$$Th(\mathcal{Q}, D, \mathcal{L}_h) = \{h \in \mathcal{L}_h \mid \exists g \in G\big(Th(\mathcal{Q}, D, \mathcal{L}_h)\big) : g \preceq h\}$$

Thus the borders of a monotonic or an anti-monotonic predicate completely characterize the set of all solutions. At this point the reader may want to verify that the S set in the previous example fully characterizes the set T of solutions to an anti-monotonic query as it contains all monomials more general than the element $\mathsf{s} \wedge \mathsf{m}$ of $S(T)$.

Furthermore, when $\mathcal{Q}$ is the conjunction of a monotonic and an anti-monotonic predicate $\mathcal{M} \wedge \mathcal{A}$ (and the language $\mathcal{L}_h$ is finite), then the resulting solution set is a *version space*. A set T is a *version space* if and only if

$$T = \{h \in \mathcal{L}_h \mid \exists s \in S(T), g \in G(T) : g \preceq h \preceq s\} \tag{3.16}$$

For version spaces, the S and G set together form a *condensed* representation for the version space. Indeed, in many (but not all) cases the border sets will be smaller than the original solution set, while characterizing the same information (as it is possible to recompute the solution set from the border sets).

Example 3.18. Consider the constraint $\mathcal{Q} = (freq(h, D) \geqslant 2) \wedge (freq(h, D) \leqslant 3)$ with D defined as in Ex. 3.3:

$$D = \{\{\mathsf{s}, \mathsf{m}, \mathsf{b}, \mathsf{c}\}, \{\mathsf{s}, \mathsf{m}, \mathsf{b}\}, \{\mathsf{s}, \mathsf{m}, \mathsf{c}\}, \{\mathsf{s}, \mathsf{m}\}\}$$

Then $S(Th)$ =$\{\mathsf{s} \wedge \mathsf{m} \wedge \mathsf{c}, \mathsf{s} \wedge \mathsf{m} \wedge \mathsf{b}\}$, and $G(Th) = \{\mathsf{b}, \mathsf{c}\}$ as shown in Fig. 3.8.

Exercise 3.19. Give an example of a quality criterion $\mathcal{Q}$ of the form $\mathcal{M} \vee \mathcal{A}$, with $\mathcal{M}$ a monotonic predicate and $\mathcal{A}$ an anti-monotonic one, a data set D and a language $\mathcal{L}_h$, such that $Th(\mathcal{Q}, D, \mathcal{L}_h)$ is not a version space.

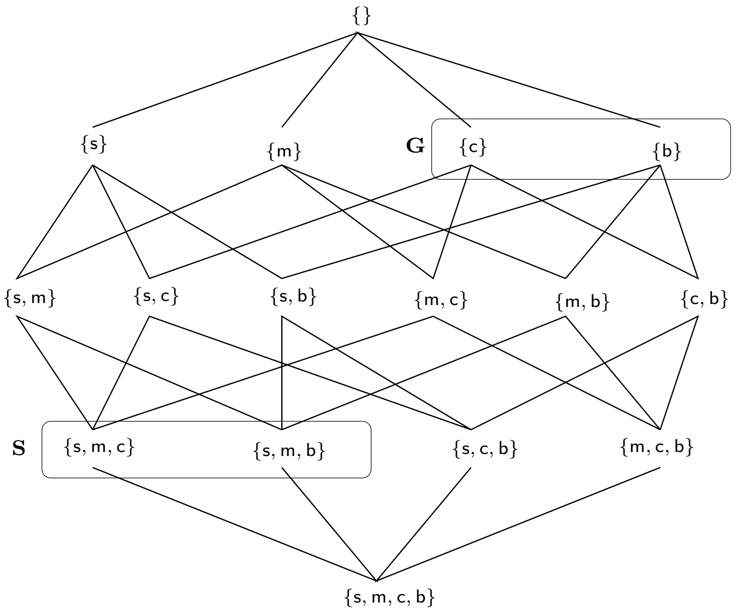

Fig. 3.8. A version space

In concept learning, one is given sets of positive and negative examples P and N. The goal of learning (in an idealized situation where no noise arises), is then to find those hypotheses h that cover all positive and none of the negative examples. Thus concept learning tasks employ the constraint

$$(rfreq(h, P) \geqslant 100\%) \wedge (rfreq(h, N) \leqslant 0\%) \tag{3.17}$$

This is the conjunction of an anti-monotonic and a monotonic predicate. Thus, the solution set to an idealized concept learning task is a version space (according to Eq. 3.16).

Exercise 3.20. Find the S and G sets corresponding to the criterion of Eq. 3.17 when $P = \{\{\mathsf{s}, \mathsf{m}, \mathsf{b}\}, \{\mathsf{s}, \mathsf{c}, \mathsf{b}\}\}$ and $N = \{\{\mathsf{b}\}, \{\mathsf{b}, \mathsf{c}\}\}$.

The S and G sets w.r.t. a version space Th are the so-called *positive* borders because they contain elements that belong to Th. In data mining, one sometimes also works with the negative borders. The negative borders contain the elements that lie just outside Th. The negative borders S^- and G^- can be defined as follows:

$$S^-(Th) = min\big(\mathcal{L}_h - \{h \in \mathcal{L}_h \mid \exists s \in S(Th) : h \preceq s\}\big) \tag{3.18}$$

$$G^-(Th) = max(\mathcal{L}_h - \{h \in \mathcal{L}_h \mid \exists g \in G(Th) : g \preceq h\}) \tag{3.19}$$

Example 3.21. Reconsider the version space Th of Ex. 3.18. For this version space, $S^-(Th) = \{\mathsf{c} \wedge \mathsf{b}\}$ and $G^-(Th) = \{\mathsf{s} \wedge \mathsf{m}\}$.

Finally, note that the size of the border sets can grow very large. Indeed, for certain hypothesis languages (such as item-sets), the size of the G set can grow exponentially large in the number of negative examples for a concept-learning task. Nevertheless, it should be clear that the size of any positive border set can never be larger than that of the overall solution set.

3.9 Refinement Operators

In the previous two sections, it was argued that the generality relation is useful when working with monotonic and/or anti-monotonic quality criteria. The present section introduces *refinement operators* for traversing the search space $\mathcal{L}_h$. The large majority of operators employed in data mining or machine learning algorithms are generalization or specialization operators. They generate a set of specializations (or generalizations) of a given hypothesis. More formally,

A *generalization operator* $\rho_g : \mathcal{L}_h \rightarrow 2^{\mathcal{L}_h}$ is a function such that

$$\forall h \in \mathcal{L}_h : \rho_g(h) \subseteq \{c \in \mathcal{L}_h \mid c \preceq h\} \tag{3.20}$$

Dually, a *specialization* operator $\rho_s : \mathcal{L}_h \rightarrow 2^{\mathcal{L}_h}$ is a function such that

$$\forall h \in \mathcal{L}_h : \rho_s(h) \subseteq \{c \in \mathcal{L}_h \mid h \preceq c\} \tag{3.21}$$

Sometimes, the operators will be applied repeatedly. This motivates the introduction of the level n refinement operator ρ^n:

$$\rho^n(h) = \begin{cases} \rho(h) & \text{if } n = 1, \\ \bigcup\limits_{h' \in \rho^{n-1}(h)} \rho(h') & \text{if } n > 1 \end{cases} \tag{3.22}$$

Furthermore, $\rho^*(h)$ denotes $\rho^\infty(h)$.

Many different types of generalization and specialization operators exist and they are useful in different types of algorithms. Two classes of operators are especially important. They are the so-called *ideal* and *optimal* operators, which are defined below for specialization operators (the corresponding definitions can easily be obtained for generalization operators). ρ is an *ideal* operator for $\mathcal{L}_h$ if and only if

$$\forall h \in \mathcal{L}_h : \rho(h) = min(\{h' \in \mathcal{L}_h \mid h \prec h'\}) \tag{3.23}$$

So, an ideal specialization operator returns all children for a node in the Hasse diagram. Furthermore, these children are *proper* refinements, that is, they are not a syntactic variant of the original hypothesis. Ideal operators are used in heuristic search algorithms.

ρ is an *optimal* operator for $\mathcal{L}_h$ if and only if for all $h \in \mathcal{L}_h$ there exists exactly one sequence of hypotheses $\top = h_0, h_1, ..., h_n = h \in \mathcal{L}_h$ such that

$h_i \in \rho(h_{i-1})$ for all i. Optimal refinement operators are used in complete search algorithms. They have the property that, when starting from $\top$, no hypothesis will be generated more than once.

An operator for which there exists *at least one* sequence from $\top$ to any $h \in \mathcal{L}_h$ is called *complete*, and one for which there exists *at most one* such sequence is *nonredundant*.

Example 3.22. We define two specialization operators ρ_o and ρ_i for the item-sets over $\mathcal{I} = \{\mathsf{s}, \mathsf{m}, \mathsf{b}, \mathsf{c}\}$ using the lexicographic $\ll$ order over $\mathcal{I}$ $\mathsf{s} \ll \mathsf{m} \ll \mathsf{c} \ll \mathsf{b}$:

$$\rho_i(M) = M \cup \{j\} \qquad \text{with } j \in (\mathcal{I} - M) \tag{3.24}$$

$$\rho_o(M) = M \cup \{j\} \qquad \text{with } \forall l \in M : l \ll j \tag{3.25}$$

By repeatedly applying ρ_i to $\top$, one obtains the Hasse diagram in Fig. 3.5, where an edge between two nodes means that the child node is one of the refinements according to ρ_i of the parent node. On the other hand, when applying ρ_o to $\top$, one obtains the tree structure depicted in Fig. 3.3. Both ρ_o^* and ρ_i^* generate all hypotheses in $\mathcal{L}_h$ from $\top$, but there is only a single path from $\top$ to any particular hypothesis using ρ_o. Remark also that using ρ_o amounts to working with a *canonical* form, where $m_1 \wedge ... \wedge m_k$ is in canonical form if and only if $m_1 << ... << m_k$. Repeatedly applying ρ_o on $\top$ only yields hypotheses in canonical form.

Two other operations that are useful in learning and mining algorithms are the *minimally general generalization mgg* and the *maximally general specialization mgs*:

$$mgg(h_1, h_2) = min\{h \in \mathcal{L}_h \mid h \preceq h_1 \wedge h \preceq h_2\} \tag{3.26}$$

$$mgs(h_1, h_2) = max\{h \in \mathcal{L}_h \mid h_1 \preceq h \wedge h_2 \preceq h\} \tag{3.27}$$

If the *mgg* (or *mgs*) operator always returns a unique generalization (or specialization), the operator is called the *least general generalization lgg* or *least upper bound lub* (or the *greatest lower bound glb*). If the *lgg* and *glb* exist for any two hypotheses $h_1, h_2 \in \mathcal{L}_h$, the partially ordered set $(\mathcal{L}_h, \preceq)$ is called a *lattice*. For instance, the language of item-sets $\mathcal{L}_\mathcal{I}$ is a lattice, as the following example shows.

Example 3.23. Continuing the previous example, the operators compute

$$mgg(M_1, M_2) = M_1 \cap M_2 \tag{3.28}$$

$$mgs(M_1, M_2) = M_1 \cup M_2 \tag{3.29}$$

For instance, $mgg(\mathsf{s} \wedge \mathsf{m} \wedge \mathsf{b}, \mathsf{s} \wedge \mathsf{b} \wedge \mathsf{c}) = lgg(\mathsf{s} \wedge \mathsf{m} \wedge \mathsf{b}, \mathsf{s} \wedge \mathsf{b} \wedge \mathsf{c}) = \{\mathsf{s} \wedge \mathsf{b}\}$.

The *mgg* and *lgg* operations are used by specific-to-general algorithms. They repeatedly generalize the current hypothesis with examples. The dual operations, the *mgs* and the *glb*, are sometimes used in algorithms that work from general-to-specific.

Exercise 3.24. Define an ideal and an optimal generalization operator for item-sets.

Exercise 3.25. * Define an ideal and an optimal specialization operator for the hypothesis language of strings Σ^*, where $g \preceq s$ if and only if g is a substring of s; cf. Ex. 3.14. Discuss also the operations mgg and mgs.

3.10 A Generic Algorithm for Mining and Learning

Now everything is in place to adapt the enumeration algorithm of Algo. 3.1 to employ the refinement operators just introduced. The resulting algorithm is shown in Algo. 3.10. It is a straightforward application of general search principles using the notions of generality.

Algorithm 3.2 A generic algorithm

```
Queue := $Init$;
Th := $\emptyset$;
while not $Stop$ do
    $Delete$ $h$ from Queue
    if $\mathcal{Q}(h,D)$ = true then
        add $h$ to $Th$
        Queue := Queue $\cup \rho(h)$
    end if
    Queue := $Prune$(Queue)
end while
return Th
```

The algorithm employs a Queue of candidate hypotheses and a set Th of solutions. It proceeds by repeatedly deleting a hypothesis h from Queue and verifying whether it satisfies the quality criterion $\mathcal{Q}$. If it does, h is added to Th; otherwise, all refinements $\rho(h)$ of h are added to the Queue. This process continues until the *Stop* criterion is satisfied. Observe that there are many *generic* parameters (shown in *italics*) in this algorithm. Depending on the particular choice of parameter, the algorithm behaves differently. The *Init* function determines the starting point of the search algorithm. The initialization may yield one or more initial hypotheses. Most algorithms start either at $\top$ and only specialize (the so-called *general-to-specific* systems), or at $\bot$ and only generalize (the *specific-to-general systems*). The function *Delete* determines the actual search strategy: when *Delete* is first-in-first-out, one obtains a breadth-first algorithm, when it is last-in-first-out, one obtains a depth-first algorithm, and when it deletes the best hypothesis (according to some criterion or heuristic), one obtains a best-first algorithm. The operator ρ determines the size and nature of the refinement steps taken through the search space. The function *Stop* determines when the algorithm halts. As argued at the start of this chapter, some algorithms compute *all* elements, k elements or an approximation of an element satisfying $\mathcal{Q}$. If all elements are desired, *Stop* equals Queue=$\emptyset$; when k elements are sought, it is $\mid Th \mid= k$. Finally, some algorithms *Prune* candidate hypotheses from the Queue. Two basic types of pruning exist: *heuristic* pruning, which prunes away those parts of the search space that appear to be uninteresting, and *sound* pruning, which prunes away those parts of the search space that cannot contain solutions.

As with other search algorithms in artificial intelligence, one can distinguish *complete* algorithms from *heuristic* ones. *Complete* algorithms compute all elements of $Th(\mathcal{Q}, D, \mathcal{L}_h)$ in a systematic manner. On the other hand, *heuristic* algorithms aim at computing one or a few hypotheses that score best w.r.t. a given heuristic function. This type of algorithm does not guarantee that the best hypotheses are found.

In the next few subsections, we present a number of instantiations of our generic algorithm. This includes: a complete general-to-specific algorithm in Sect. 3.11, a heuristic general-to-specific algorithm in Sect. 3.12, a branch-and-bound algorithm for finding the top k hypotheses in Sect. 3.13, and a specific-to-general algorithm in Sect. 3.14. Afterward, a further (advanced) section on working with borders is included, before concluding this chapter.

3.11 A Complete General-to-Specific Algorithm

We now outline a basic one-directional complete algorithm that proceeds from general to specific. It discovers all hypotheses that satisfy an anti-monotonic quality criterion $\mathcal{Q}$. It can be considered an instantiation of the generic algorithm, where

- *Init*= $\{\top\}$;
- *Prune*(Queue)= $\{h \in \text{Queue} \mid \mathcal{Q}(h, D) = false\}$,
- *Stop*=(Queue= $\emptyset$), and
- ρ is an optimal refinement operator.

Furthermore, various instantiations of *Delete* are possible.

Example 3.26. Reconsider Ex. 3.3 and assume Algo. 3.3 employs a breadth-first search strategy (obtained by setting *Delete* to first-in-first-out). Then the algorithm traverses the search tree shown in Fig. 3.9 in the order indicated.

Observe that for anti-monotonic quality criteria, Algo. 3.3 only prunes hypotheses that cannot be a solution, and whose children cannot be a solution either. Observe also that the use of an optimal refinement operator is, strictly speaking, not necessary for the correctness of the algorithm but is essential for its efficiency. Indeed, if an ideal operator would be employed instead, the same hypotheses would be generated over and over again.

There exists also a dual algorithm, that searches from specific to general and applies generalization rather than specialization.

Algorithm 3.3 A complete general-to-specific algorithm

```
Queue := $\{\top\}$;
Th := $\emptyset$;
**while** not $Queue = \emptyset$ **do**
   *Delete* $h$ from Queue
   **if** $\mathcal{Q}(h,D)$ = true **then**
      add $h$ to $Th$
      Queue := Queue $\cup \rho_o(h)$
   **end if**
**end while**
**return** Th
```

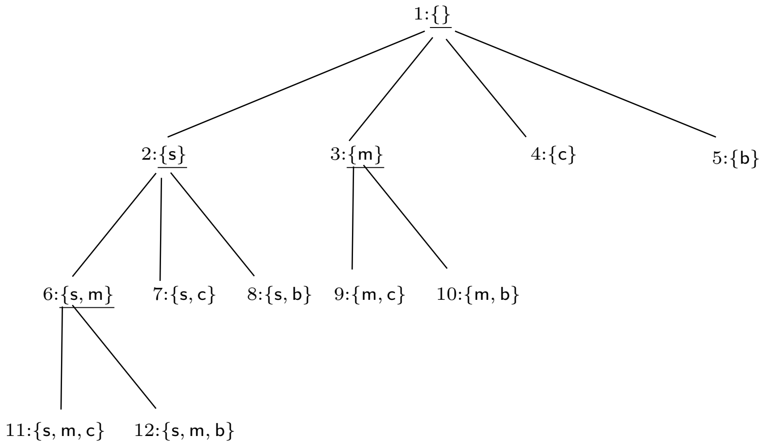

Fig. 3.9. Illustrating complete search w.r.t. an anti-monotonic predicate

Exercise 3.27. Describe the dual algorithm and illustrate it at work on the same data set and hypothesis language, but now use the constraint $(freq(h, D) \leqslant 2)$.

3.12 A Heuristic General-to-Specific Algorithm

The complete algorithm works well provided that the quality criterion is anti-monotonic or monotonic. However, there exist many interesting mining and learning tasks for which the quality criterion is neither monotonic nor anti-monotonic. Furthermore, one might not be interested in all solutions but perhaps in a single best solution or an approximation thereof. In such cases, it is too inefficient to perform a complete search because the pruning properties no longer hold. Therefore, the only resort is to employ a heuristic function f in a greedy algorithm. Such an algorithm is shown in Algo. 3.4.

The algorithm again works from general to specific and keeps track of a Queue of candidate solutions. It repeatedly selects the best hypothesis h from

Queue (according to its heuristic) and tests whether it satisfies $\mathcal{Q}$. If it does, the algorithm terminates and outputs its solution; otherwise, it continues by adding all refinements (using an ideal operator) to the Queue. The Queue is typically also *Pruned*.

Again, the algorithm can be viewed as an instantiation of Algo. 3.10. The following choices have been made:

- *Delete* selects the best hypothesis,
- *Stop*=$\mid Th \mid= 1$,
- an ideal refinement operator is employed,
- *Prune* depends on the particular instantiation. Very often *Prune* retains only the best k hypotheses, realizing a beam search.

Note that – because of the use of a heuristic and a greedy search strategy – it is essential that an ideal operator is being used. Greedy algorithms focus on the currently most interesting nodes and prune away the others. Should an optimal refinement operator be used instead of an ideal one, the direct neighborhoods of the nodes of interest would not be fully explored.

Algorithm 3.4 A heuristic general-to-specific algorithm

```
Queue := {$\top$};
Th := $\emptyset$;
while Th = $\emptyset$ do
  Delete the best $h$ from Queue
  if $\mathcal{Q}(h,D)$ = true then
    add $h$ to $Th$
  else
    Queue := Queue $\cup \rho_i(h)$
  end if
  Queue := Prune(Queue)
end while
return Th
```

Example 3.28. Let $P = \{\{\mathsf{s, m, b}\}, \{\mathsf{s, b, c}\}\}$ and $N = \{\{\mathsf{s, m}\}, \{\mathsf{b, c}\}\}$ be the data sets; assume that the heuristic function used is $m(h, P, N)$ and that the quality criterion is true if $m(h, P, N) > m(h', P, N)$ for all $h' \in \rho_i(h)$. The $m(h, P, N)$ function is a variant of the accuracy defined in Eq. 3.8:

$$m(h, P, N) = \frac{freq(h, P) + 0.5}{freq(h, P) + freq(h, N) + 1} \tag{3.30}$$

The m function is used instead of *acc* to ensure that when two patterns have equal accuracy, the one with the higher coverage is preferred. Assume also that a beam search with $k = 1$, that is, hill climbing, is used. This results in the search tree illustrated in Fig. 3.12. The nodes are expanded in the order indicated.

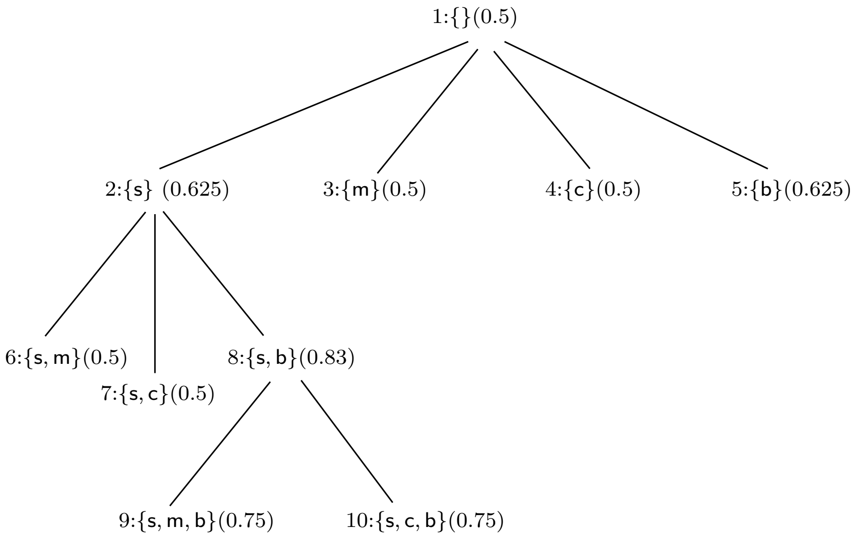

Fig. 3.10. Illustrating heuristic search

3.13 A Branch-and-Bound Algorithm

For some types of problem, a combination of the previous two algorithms – branch-and-bound – can be used. A *branch-and-bound* algorithm aims at finding the best hypothesis (or, best k hypotheses) w.r.t. a given function f, that is,

$$\mathcal{Q}(h, D, \mathcal{L}_h) = \big(h = \arg\max_{h' \in \mathcal{L}_h} f(h')\big) \tag{3.31}$$

Furthermore, branch-and-bound algorithms assume that, when working from general to specific, there is a bound $b(h)$ such that

$$\forall h' \in \mathcal{L}_h : h \preceq h' \rightarrow b(h) \geqslant f(h') \tag{3.32}$$

Given the current best value (or current k best values) v of the hypotheses investigated so far, one can safely prune all refinements of h provided that $v \geqslant b(h)$.

The branch-and-bound algorithm essentially combines the previous two algorithms: it performs a complete search but selects the hypotheses greedily (according to their f values) and prunes on the basis of the bounds. Furthermore, it computes a single best hypothesis (or k best hypotheses) as in the heuristic algorithm.

Example 3.29. Consider the function $f(h) = freq(h, P) - freq(h, N)$. The quality criterion aims at finding patterns for which the difference between the frequency in P and in N is maximal. For this function $f(h)$, the bound

$b(h) = freq(h, P)$ satisfies the requirement in Eq. 3.32 because specializing a hypothesis can only decrease the frequencies $freq(h, P)$ and $freq(h, N)$. Thus the maximum is obtained when $freq(h, P)$ remains unchanged and $freq(h, N)$ becomes 0.

The resulting search tree, applied to P and N of Ex. 3.20, is shown in Fig. 3.13. The values for the hypotheses $(b(h); f(h))$ are shown for each node. The order in which the nodes are visited is indicated. Nodes 3, 4 and 6 are pruned directly after generation and node 7 is pruned after node 8 has been generated. Finally, since node 5 cannot be further expanded, node 8 contains the optimal solution.

Let us also stress that branch-and-bound algorithms are used with statistical functions such as entropy and chi-square; cf. [Morishita and Sese, 2000].

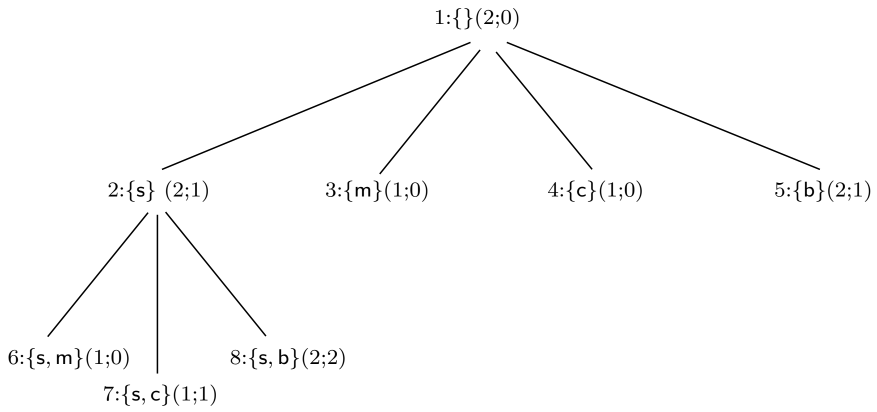

Fig. 3.11. Illustrating a branch-and-bound algorithm

Exercise 3.30. Specify the branch-and-bound algorithm formally.

3.14 A Specific-to-General Algorithm

To illustrate a specific-to-general algorithm, we consider an algorithm that implements a cautious approach to generalization in this section. Assume that the goal is to find the minimal generalizations of a set of hypotheses (or positive examples). The quality criterion can be specified as

$$\mathcal{Q}(h, D) = \big(h \in max(\{h \in \mathcal{L}_h \mid \forall d \in D : h \preceq d\}\big) \tag{3.33}$$

One possible algorithm to compute $Th(\mathcal{Q}, D, \mathcal{L}_h)$ is shown in Algo. 3.5. It starts from $\perp$ and repeatedly generalizes the present hypotheses until all examples or hypotheses in D are covered. Observe that the algorithm is an instance of the generic algorithm Algo. 3.10 (it is left as an exercise to the reader to verify this), and also that the efficiency of the algorithm can be improved by, for instance, incrementally processing the examples in D. This would eliminate our having to test the overall criterion $\mathcal{Q}$ over and over again.

Algorithm 3.5 A cautious specific-to-general algorithm

```
Queue := $\{\bot\}$;
Th := $\emptyset$;
while Queue $\neq \emptyset$ do
  Delete a hypothesis $h$ from Queue
  if $\mathcal{Q}(h,D)$ = true then
    add $h$ to $Th$
  else
    select a hypothesis $d \in D$ such that $\neg(h \preceq d)$
    Queue := Queue $\cup$ $mgg(h, d)$
  end if
end while
return Th
```

Example 3.31. Consider the data set $D = \{ \mathsf{s} \wedge \mathsf{m} \wedge \mathsf{c}, \mathsf{s} \wedge \mathsf{m} \wedge \mathsf{b}, \mathsf{s} \wedge \mathsf{m} \wedge \mathsf{c} \wedge \mathsf{b}\}$. When the examples are processed from right to left, the following hypotheses are generated: $\bot = false$, $\mathsf{s} \wedge \mathsf{m} \wedge \mathsf{c} \wedge \mathsf{b}$, $\mathsf{s} \wedge \mathsf{m} \wedge \mathsf{b}$, and $\mathsf{s} \wedge \mathsf{m}$.

Exercise 3.32. Design a data set in the domain of the strings where the S set (returned by Algo. 3.5) is not a singleton.

3.15 Working with Borders*

Algo. 3.5 can also be regarded as computing the S set w.r.t. the anti-monotonic constraint $rfreq(h, D) \geqslant 1$, which raises the question of how to compute, exploit and reason with border sets. This will be addressed in the present section.

3.15.1 Computing a Single Border

First, observe that a single border (either the S or G set w.r.t. an anti-monotonic or a monotonic constraint) can be computed in two dual ways: general-to-specific or specific-to-general using a simple adaptation of Algo. 3.3. To illustrate this point, assume that the goal is to find the G set w.r.t. a monotonic criterion $\mathcal{Q}$ and that we work from general to specific. Two modifications are needed to Algo. 3.3 for addressing this task (shown in Algo. 3.6): 1) refine only those hypotheses that do not satisfy the quality criterion $\mathcal{Q}$, and 2) move only those elements to Th that effectively belong to G. W.r.t.

1), note that if a hypothesis is a member of the G set, then all of its (proper) specializations satisfy $\mathcal{Q}$ even though they cannot be maximally general and therefore do not belong to G. W.r.t. 2), note that it is possible to test whether a hypothesis h that satisfies $\mathcal{Q}$ is maximally general by computing $\rho'_i(h)$ and testing whether elements of $\rho'_i(h)$ satisfy $\mathcal{Q}$. Only if none of them satisfies $\mathcal{Q}$can one conclude that $h \in G$. Here, ρ'_i denotes an ideal generalization operator, even though the general direction of the search is from general to specific.

Algorithm 3.6 Computing the G border general-to-specific.

```
Queue := {$\top$};
Th := $\emptyset$;
**while** not *Queue* = $\emptyset$ **do**
    *Delete* $h$ from Queue
    **if** $\mathcal{Q}(h,D)$ = true and $h \in G$ **then**
        add $h$ to $Th$
    **else if** $\mathcal{Q}(h,D)$ = false **then**
        Queue := Queue $\cup \rho_o(h)$
    **end if**
**end while**
**return** Th
```

The dual algorithm for computing G from specific to general can be obtained by starting from $\perp$ and by generalizing only those hypotheses h that satisfy Q (and do not belong to G). Even though the two algorithms compute the same result, the efficiency with which they do so may vary significantly. The direction that is to be preferred typically depends on the application.

By exploiting the dualities, one can devise algorithms for computing S as well as the negative borders.

Exercise 3.33. Compute the S set for the problem of Ex. 3.3 using the general-to-specific algorithm.

3.15.2 Computing Two Borders

Second, consider computing the borders of a version space as illustrated in Fig. 3.8. This could be the result of a quality criterion $\mathcal{Q}$ that is the conjunction of an anti-monotonic and a monotonic predicate. To compute these borders, we can proceed in several ways. One of these first computes one border (say the S set) using the techniques sketched above and then uses that set to constrain the computation of the other border (say the G set). When searching from general to specific for the G set and with the S set already given, hen all hypotheses h that are not more general than an element in the S set can safely be pruned in Algo. 3.6. By exploiting the various dualities, further algorithms can be obtained.

Example 3.34. Suppose $S = \{\mathsf{s} \wedge \mathsf{m} \wedge \mathsf{c}\}$. When computing G from general to specific, b and all its refinements can be pruned because $\neg(\mathsf{b} \preceq \mathsf{s} \wedge \mathsf{m} \wedge \mathsf{c})$.

3.15.3 Computing Two Borders Incrementally

Third, suppose that one is given already a version space (characterized by its S and G sets) and the task is to update it in the light of an extra constraint. This is the setting for which the original theory of version spaces was developed by Tom Mitchell. He considered concept-learning, which is concerned with finding all hypotheses satisfying $rfreq(h, P) = 1 \wedge rfreq(h, N) = 0$ w.r.t. sets of positive and negative examples P and N. This criterion can be rewritten as:

$$p_1 \in \mathbf{c}(h) \wedge \ldots \wedge p_k \in \mathbf{c}(h) \wedge n_1 \notin \mathbf{c}(h) \wedge \ldots \wedge n_l \notin \mathbf{c}(h) \qquad (3.34)$$

where the p_i and the n_j are the members of P and N, respectively. Mitchell's candidate elimination algorithm processed the examples incrementally, that is, one by one, by updating S and G to accommodate the new evidence. His algorithm is shown in Algo. 3.7.

Algorithm 3.7 Mitchell's candidate elimination algorithm

```
$S := \{\bot\}$ ; $G := \{\top\}$
for all examples $e$ do
  if $e \in N$ then
    $S := \{s \in S \mid e \in \mathbf{c}(s)\}$
    for all $g \in G : e \in \mathbf{c}(g)$ do
      $\rho_g := \{g' \in ms(g, e) \mid \exists s \in S : g' \preceq s\}$
      $G := G \cup \rho_g$
    end for
    $G := min(G)$
  else if $e \in P$ then
    $G := \{g \in G \mid e \notin \mathbf{c}(g)\}$
    for all $s \in S : e \notin \mathbf{c}(s)$ do
      $\rho_s := \{s' \in mgg(s, e) \mid \exists g \in G : g \preceq s'\}$
      $S := S \cup \rho_s$
    end for
    $S := max(S)$
  end if
end for
```

The algorithm employs a new operation $ms(g, e)$, the minimal specialization w.r.t. e:

$$ms(g, e) = min(\{g' \in \mathcal{L}_h \mid g \preceq g' \wedge e \notin \mathbf{c}(g')\}) \qquad (3.35)$$

Example 3.35. Applied to item-sets over $\mathcal{I}$, this yields

$$ms(M_1, M_2) = \begin{cases} \{M_1\} & \text{if } \neg(M_1 \preceq M_2) \\ \{M_1 \cup \{i\} \mid i \in \mathcal{I} - (M_1 \cup M_2)\} & \text{otherwise} \end{cases} \tag{3.36}$$

For instance, $ms(\mathsf{s}, \mathsf{s} \wedge \mathsf{m}) = \{\mathsf{s} \wedge \mathsf{c}, \mathsf{s} \wedge \mathsf{b}\}$ and $ms(\mathsf{s}, \mathsf{s} \wedge \mathsf{m} \wedge \mathsf{b}) = \{\mathsf{s} \wedge \mathsf{s}\}$.

The candidate elimination algorithm works as follows. It starts by initializing the S and G sets to the $\bot$ and $\top$ elements, respectively. It then repeatedly updates these sets whenever the next example is not handled correctly by all the elements of S and G. Let us now sketch the different steps for a positive example (the ones for a negative one are dual). Whenever an element g of G does not cover the positive example e, the element g is too specific and is pruned away. This is because in order to cover e, the hypothesis g should be generalized, but this is not allowed since it would yield hypotheses that lie outside the current version space. Secondly, whenever an element s of S does not cover the positive example e, the mgg operator is applied on the elements g and e, yielding a set $mgg(e, g)$ of minimally general generalizations. From this set, only those elements are retained that lie within the version space, that is, those that are more specific than an element of G. Finally, only the maximal elements of S are retained in order to obtain a proper border and to remove redundancies. Without this step, the algorithm still works correctly in the sense that all elements of the version space will lie between an element of S and G. It may only be that some elements are redundant and do not lie at the proper border.

Example 3.36. Let us now employ the candidate elimination algorithm to the sets of examples $P = \{\{\mathsf{s}, \mathsf{m}, \mathsf{b}\}, \{\mathsf{s}, \mathsf{c}, \mathsf{b}\}\}$ and $N = \{\{\mathsf{b}\}, \{\mathsf{b}, \mathsf{c}\}\}$. So, the resulting S and G form the answer to Exer. 3.20. When first processing the positive examples and then the negative ones, the following sequence of S and G sets is obtained:

$$\begin{array}{ll} S{=}\{\bot\} & G{=}\{\top\} \\ S{=}\{\mathsf{s} \wedge \mathsf{m} \wedge \mathsf{b}\} & G{=}\{\top\} \\ S{=}\{\mathsf{s} \wedge \mathsf{b}\} & G{=}\{\top\} \\ S{=}\{\mathsf{s} \wedge \mathsf{b}\} & G{=}\{\mathsf{s}\} \end{array}$$

Exercise 3.37. What happens to S and G when the examples are processed in a different order? Process the negative examples before the positive ones.

The use of version space representations for concept learning has some interesting properties. When the S and G sets become identical and contain a single hypothesis, one has converged upon a single solution, and when S or G becomes empty, no solution exists. Finally, the intermediate borders obtained using an incremental algorithm (such as the candidate elimination algorithm) can be used to determine whether a hypothesis h can still belong to the solution space. Furthermore, when learning concepts, the intermediate borders can be used to determine which examples contain new information. Indeed, under the assumption that a solution to the concept learning task exists within $\mathcal{L}_h$, any example covered by all elements in S must be positive, and any example covered by no element in G must be negative.

Exercise 3.38. * When $\mathcal{L}_h = \mathcal{L}_e$, the constraint $e \in \mathbf{c}(h)$ can often be rewritten as $h \preceq e$, and the dual one, $e \notin \mathbf{c}(h)$, as $\neg(h \preceq e)$, where h is the target hypothesis and e a specific positive or negative example. Consider now the dual constraints $e \preceq h$ and $\neg(e \preceq h)$. Are these constraints monotonic or anti-monotonic? Also, can you illustrate the use of these constraints and compute the corresponding version space? Finally, can you extend the candidate elimination algorithm to work with these constraints?

Exercise 3.39. Try to learn the father/2 predicate (in a relational learning setting) in the following context. Let the background theory B be the set of facts

$$\begin{array}{ll}
\mathsf{male(luc)} \leftarrow & \mathsf{parent(luc, soetkin)} \leftarrow \\
\mathsf{female(lieve)} \leftarrow & \mathsf{parent(lieve, soetkin)} \leftarrow \\
\mathsf{female(soetkin)} \leftarrow &
\end{array}$$

the hypothesis language $\mathcal{L}_h$ consist of single clauses, $\mathcal{L}_h = \{\mathsf{father(X,Y)} \leftarrow body \mid body \subseteq \{\mathsf{parent(X,Y), parent(Y,X), male(X), female(X), female(Y)}\}\}$, and let $P = \{\mathsf{father(luc, soetkin)}\}$, and $N = \{\mathsf{father(lieve, soetkin), father(luc, luc)}\}$

The candidate elimination algorithm illustrates how the borders of a version space can be computed incrementally. The candidate elimination algorithm works with constraints in the form of positive and negative examples. One remaining question is whether one can also devise algorithms that incrementally process a sequence of other types of monotonic and anti-monotonic constraints. One way of realizing this adapts Algo. 3.6. We discuss how to process an extra monotonic constraint $\mathcal{Q}$ w.r.t. to an already existing version space characterized by G and S.

The adaptation works as follows. First, the elements of S that do not satisfy $\mathcal{Q}$ are discarded. Second, the approach of Algo. 3.6 is taken but 1) all hypotheses that are not more general than an element of S are pruned, and 2) only those hypotheses that are more specific than an element of the original G set are tested w.r.t. $\mathcal{Q}$.

3.15.4 Operations on Borders

Another approach to incrementally process a conjunction of monotonic and anti-monotonic constraints intersects version spaces. Version space intersection employs the following operations:

$$int_s(S_1, S_2) = max\big(\{s \mid s \in mgg(s_1, s_2) \text{ with } s_1 \in S_1 \wedge s_2 \in S_2\}\big) \quad (3.37)$$
$$int_g(G_1, G_2) = min\big(\{g \mid g \in mgs(g_1, g_2) \text{ with } g_1 \in G_1 \wedge g_2 \in G_2\}\big) \quad (3.38)$$

The following property, due to Hirsh [1990], holds

Property 3.40. Let VS_1 and VS_2 be two version spaces with border sets S_1, G_2 and S_2, G_2, respectively. Then $VS = VS_1 \cap VS_2$ has border sets $int_s(S_1, S_2)$ and $int_g(G_1, G_2)$ respectively.

Version space intersection can now be used for learning concepts. To realize this, compute the version spaces that correspond to each of the single examples and incrementally intersect them.

Exercise 3.41. Solve the concept learning task in Exer. 3.20 by applying version space intersection.

3.16 Conclusions

This chapter started by formalizing data mining and machine learning tasks in a general way. It then focused on mechanisms for computing the set of solutions $Th(\mathcal{Q}, D, \mathcal{L}_h)$. The search space $\mathcal{L}_h$ was structured using the important *generality* relation. Various quality criteria were proposed and their properties, most notably monotonicity and anti-monotonicity, were discussed. It was shown that these properties impose borders on the set of solutions $Th(\mathcal{Q}, D, \mathcal{L}_h)$, and the notion of a version space was introduced. To compute $Th(\mathcal{Q}, D, \mathcal{L}_h)$ various algorithms were presented. They employ refinement operators which are used to traverse the search space. Ideal operators are especially useful for performing heuristic search and optimal ones for complete search. Some of the algorithms work from general to specific, other ones from specific to general. Finally, some algorithms work with the borders and there are algorithms (such as candidate elimination) that are bidirectional.

3.17 Bibliographical Notes

The formulation of the data mining task using $Th(\mathcal{Q}, D, \mathcal{L}_h)$ is due to Mannila and Toivonen [1997]. The notions of generality and border sets and their use for machine learning are due to Mitchell [1982, 1997] and, in a data mining context, to Mannila and Toivonen [1997]. The material presented here follows essentially the same lines of thought as these earlier works but perhaps presents a more integrated view on data mining and concept learning. The algorithms contained in this section are derived from [Mitchell, 1982, 1997, Mannila and Toivonen, 1997, De Raedt and Kramer, 2001, De Raedt and Bruynooghe, 1992a, Morishita and Sese, 2000]. Many of them belong to the folklore of machine learning and data mining. Algorithm 3.3 has a famous instantiation in the context of item-set and association rule mining; cf. [Agrawal et al., 1993]. Refinement operators were introduced in inductive logic programming in Ehud Shapiro's seminal Ph.D. thesis [1983]. Their properties were later studied by Nienhuys-Cheng and de Wolf [1997]. The notion of an ideal refinement operator introduced has been slightly adapted (w.r.t.

[Nienhuys-Cheng and de Wolf, 1997]) for educational purposes. The notion of an optimal refinement operator in relational learning is due to De Raedt and Bruynooghe [1993].