Learning unordered rule sets

We next consider the alternative approach to rule learning, where rules are learned for one class at a time. This means we can further simplify our search heuristic: rather than minimising $\min(\dot{p}, 1-\dot{p})$, we can maximise $\dot{p}$, the empirical probability of the class we are learning.

We now have a rule set for the positive class. With two classes this might be considered sufficient, as we can classify everything that isn't covered by the positive rules as negative.

So let's learn some rules for the negative class. By the same procedure as in Example 6.3

Example 6.3 (Learning a rule set for one class). We continue the dolphin example. Figure 6.7 shows that the first rule learned for the positive class is

$$\cdot\textbf{if } \mathsf{Length} = 3 \textbf{ then } \mathsf{Class} = \oplus\cdot$$

The two examples covered by this rule are removed, and a new rule is learned. We now encounter a new situation, as none of the candidates is pure (Figure 6.8). We thus start a second-level search, from which the following pure rule emerges:

$$\cdot\textbf{if } \mathsf{Gills} = \mathsf{no} \wedge \mathsf{Length} = 5 \textbf{ then } \mathsf{Class} = \oplus\cdot$$

To cover the remaining positive, we again need a rule with two conditions (Figure 6.9):

$$\cdot\textbf{if } \mathsf{Gills} = \mathsf{no} \wedge \mathsf{Teeth} = \mathsf{many} \textbf{ then } \mathsf{Class} = \oplus\cdot$$

Notice that, even though these rules are overlapping, their overlap only covers positive examples (since each of them is pure) and so there is no need to organise them in an if-then-else list.

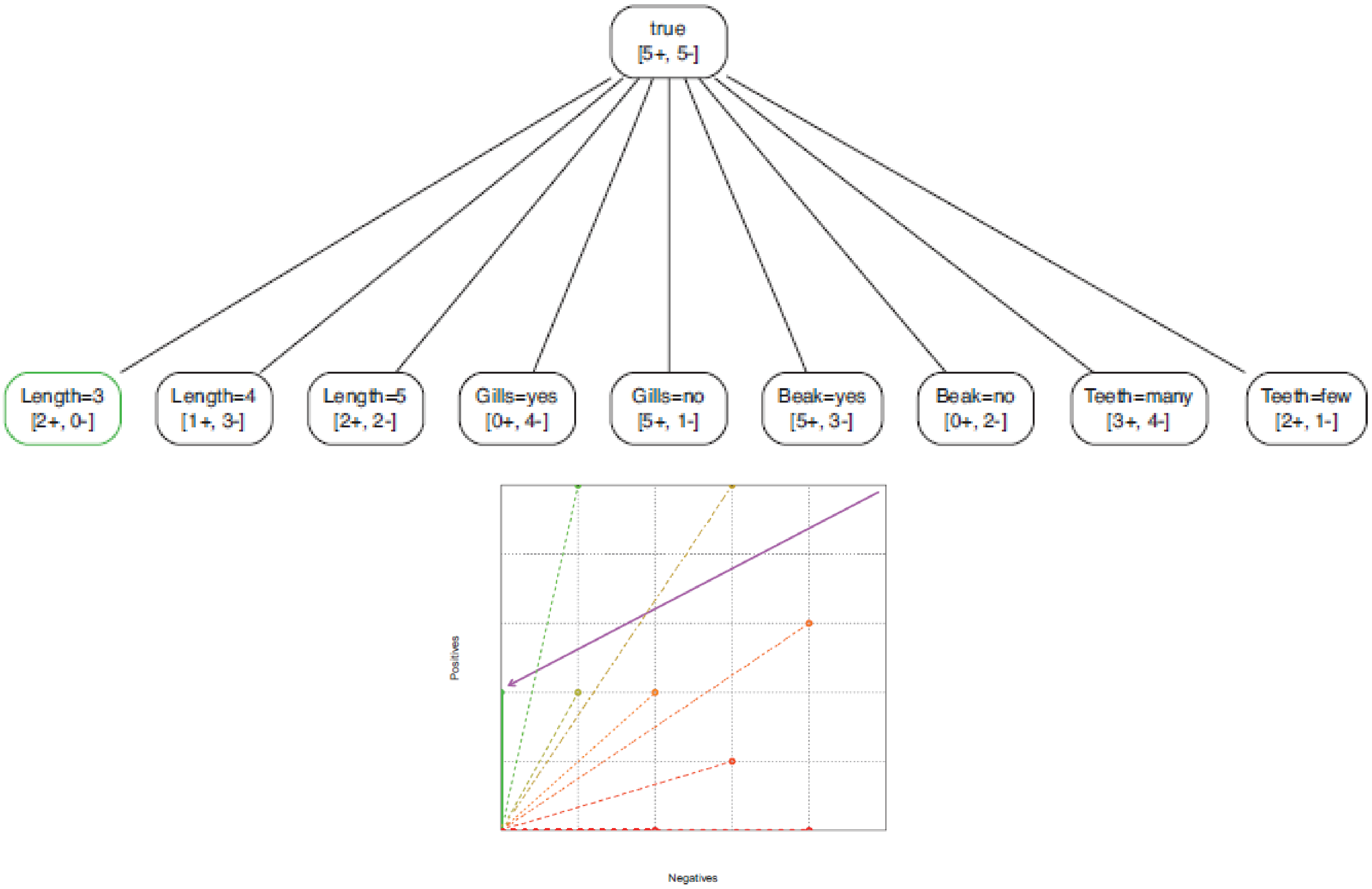
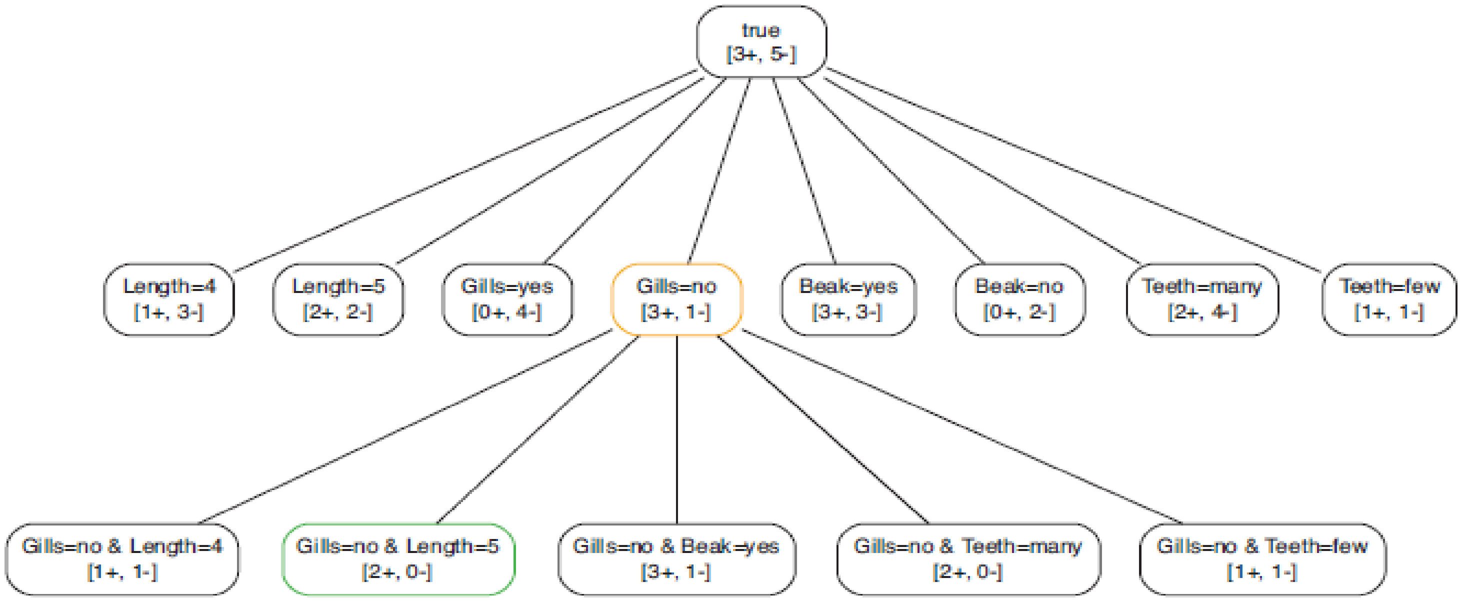


Figure 6.7. **(top)** The first rule is learned for the positive class. **(bottom)** Precision isometrics look identical to impurity isometrics (Figure 6.2); however, the difference is that precision is lowest on the x-axis and highest on the y-axis, while purity is lowest on the ascending diagonal and highest on both the x-axis and the y-axis.



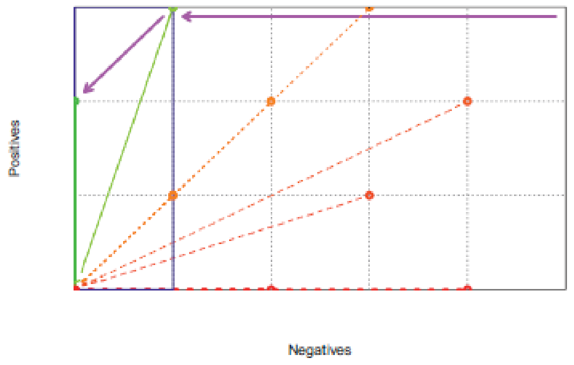
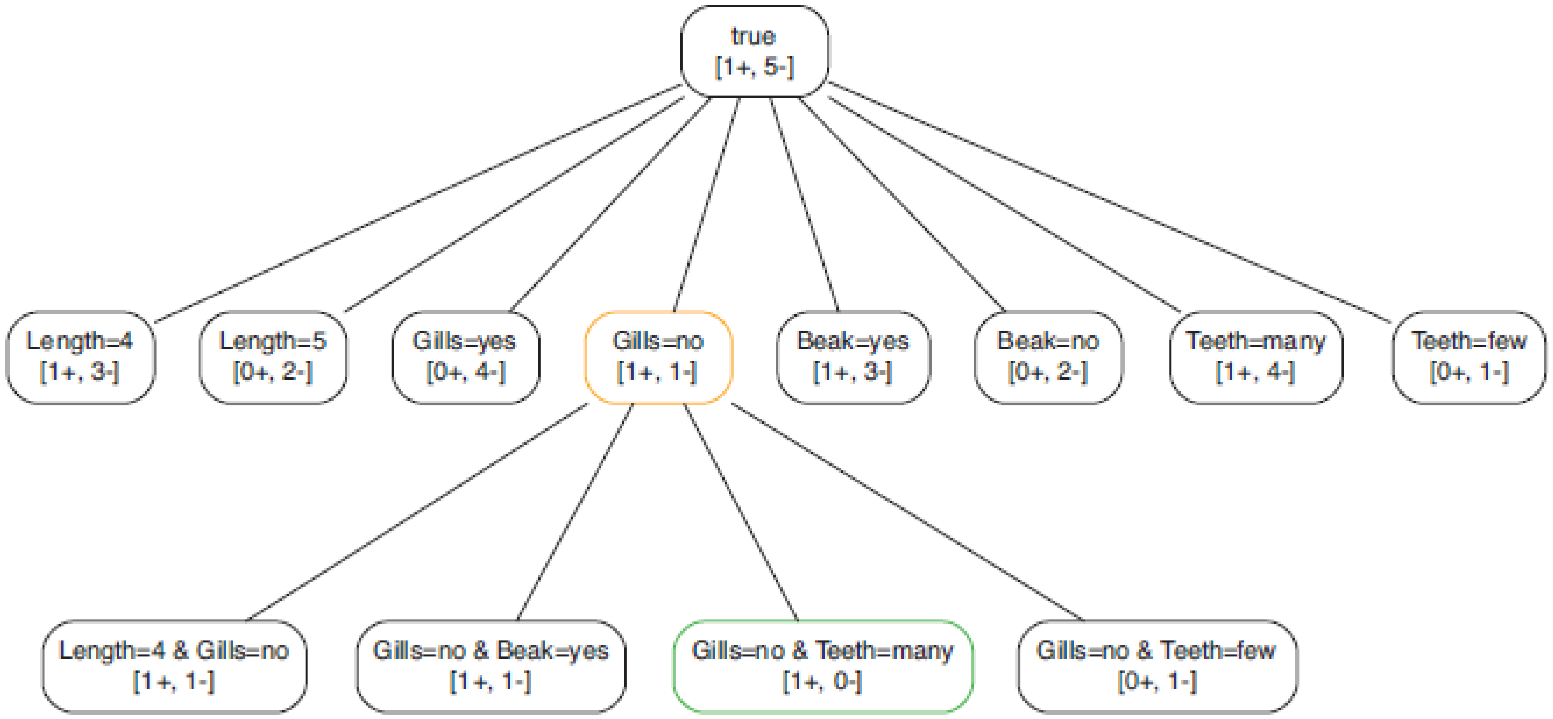


Figure 6.8. **(top)** The second rule needs two literals: we use maximum precision to select both. **(bottom)** The coverage space is smaller because the two positives covered by the first rule are removed. The blue box on the left indicates an even smaller coverage space in which the search for the second literal is carried out, after the condition Gills = no filters out four negatives. Inside the blue box precision isometrics overlap with those in the outer box (this is not necessarily the case with search heuristics other than precision).



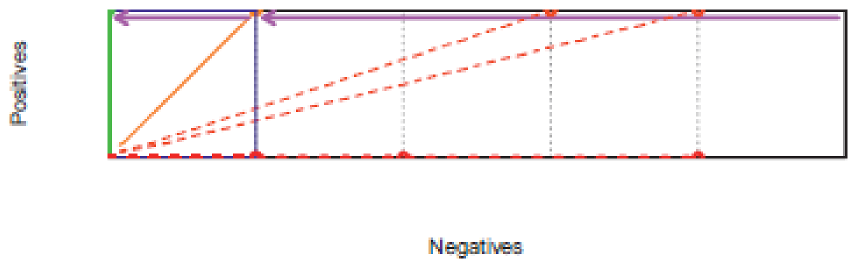


Figure 6.9. (top) The third and final rule again needs two literals. **(bottom)** The first literal excludes four negatives, the second excludes the one remaining negative.

However, this might introduce a bias towards the negative class as all difficult cases we're unsure about get automatically classified as negative. So let's learn some rules for the negative class. By the same procedure as in Example 6.3 we find the following rules (you may want to check this): ·**if** Gills = yes **then** Class = $\ominus$· first, followed by ·**if** Length = 4 $\wedge$ Teeth = few **then** Class = $\ominus$·.

The final rule set with rules for both classes is therefore

(R1) ·**if** Length = 3 **then** Class = $\oplus$·

(R2) ·**if** Gills = no $\wedge$ Length = 5 **then** Class = $\oplus$·

(R3) ·**if** Gills = no $\wedge$ Teeth = many **then** Class = $\oplus$·

(R4) ·**if** Gills = yes **then** Class = $\ominus$·

(R5) ·**if** Length = 4 $\wedge$ Teeth = few **then** Class = $\ominus$·

The algorithm for learning a rule set is given in Algorithm 6.3

Algorithm 6.3: LearnRuleSet(D) – learn an unordered set of rules.

```
Input  : labelled training data $D$.
Output : rule set $R$.
$R \leftarrow \emptyset$;
for every class $C_i$ do
    $D_i \leftarrow D$;
    while $D_i$ contains examples of class $C_i$ do
        $r \leftarrow$ LearnRuleForClass($D_i, C_i$) ;    // LearnRuleForClass: see Algorithm 6.4
        $R \leftarrow R \cup \{r\}$;
        $D_i \leftarrow D_i \setminus \{x \in C_i | x$ is covered by $r\}$ ;    // remove only positives
    end
end
return $R$
```

we now iterate over each class in turn, and furthermore that only covered examples for the class that we are currently learning are removed after a rule is found. The reason for this second change is that rule sets are not executed in any particular order, and so covered negatives are not filtered out by other rules. Algorithm 6.4 gives the algorithm for learning a single rule for a particular class

Algorithm 6.4: LearnRuleForClass(D, C_i) – learn a single rule for a given class.

```
Input  : labelled training data $D$; class $C_i$.
Output : rule $r$.
b ← true;
L ← set of available literals ;          // can be initialised by seed example
while not Homogeneous(D) do
    l ← BestLiteral(D, L, C_i) ;         // e.g. maximising precision on class C_i
    b ← b ∧ l;
    D ← {x ∈ D | x is covered by b};
    L ← L \ {l' ∈ L | l' uses same feature as l};
end
r ← ·if b then Class = C_i·;
return r
```

Example 6.4 (Rule sets as rankers). Consider the following rule set (the first two rules were also used in Example 6.2):

(A)	·**if** Length = 4 **then** Class = ⊖·	[1+,3−]
(B)	·**if** Beak = yes **then** Class = ⊕·	[5+,3−]
(C)	·**if** Length = 5 **then** Class = ⊖·	[2+,2−]

The figures on the right indicate coverage of each rule over the whole training set. For instances covered by single rules we can use these coverage counts to calculate probability estimates: e.g., an instance covered only by rule A would receive probability $\hat{p}(A) = 1/4 = 0.25$, and similarly $\hat{p}(B) = 5/8 = 0.63$ and $\hat{p}(C) = 2/4 = 0.50$.

Clearly A and C are mutually exclusive, so the only overlaps we need to take into account are AB and BC. A simple trick that is often applied is to average the coverage of the rules involved: for example, the coverage of AB is estimated as [3+,3−] yielding $\hat{p}(AB) = 3/6 = 0.50$. Similarly, $\hat{p}(BC) = 3.5/6 = 0.58$. The corresponding ranking is thus B – BC – [AB, C] – A, resulting in the orange training set coverage curve in Figure 6.11.

Let us now compare this rule set with the following rule list ABC:

·**if** Length = 4 **then** Class = ⊖·	[1+,3−]
·**else if** Beak = yes **then** Class = ⊕·	[4+,1−]
·**else if** Length = 5 **then** Class = ⊖·	[0+,1−]

A closer look at rule overlap

We have seen that rule lists always give convex training set coverage curves, but that there is no globally optimal ordering of a given set of rules. The main reason is that rule lists don't give us access to the overlap of two rules $A \wedge B$: we either have access to $A = (A \wedge B) \vee (A \wedge \neg B)$ if the rule order is AB, or $B = (A \wedge B) \vee (\neg A \wedge B)$ if it is BA. More generally, a rule list of r rules results in only r instance space segments (or $r+1$ in case we add a default rule). This means that we cannot take advantage of most of the 2^r ways in which rules can overlap. Rule sets, on the other hand, can potentially give access to such overlaps, but the need for the coverage counts of overlapping segments

In order to understand this further, we introduce in this section the concept of a *rule tree*: a complete feature tree using the rules as features.

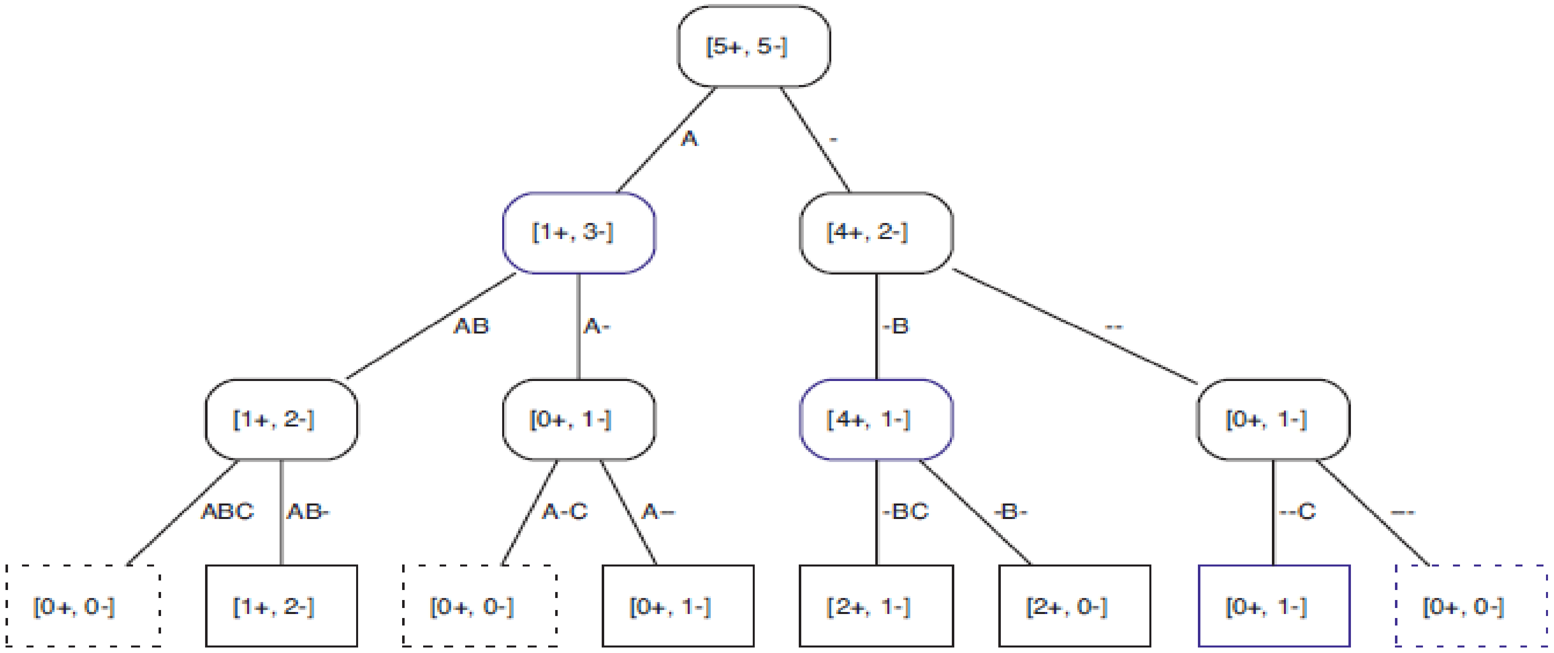


Figure 6.12. A rule tree constructed from the rules in Example 6.5. Nodes are labelled with their coverage (dotted leaves have empty coverage), and branch labels indicate particular areas in the instance space (e.g., A-C denotes $A \wedge \neg B \wedge C$). The blue nodes are the instance space segments corresponding to the rule list ABC: the rule tree has better performance because it is able to split them further.

Example 6.5 (Rule tree). From the rules in Example 6.4 we can construct the rule tree in Figure 6.12.