RULE MODELS

RULE MODELS ARE the second major type of logical machine learning models. Generally speaking, they offer more flexibility than tree models:

There are essentially two approaches to supervised rule learning. One is inspired by decision tree learning: find a combination of literals – the *body* of the rule, which is what we previously called a concept – that covers a sufficiently homogeneous set of examples, and find a label to put in the *head* of the rule. (ORDERED) *rule list*

The second approach goes in the opposite direction: first select a class you want to learn, and then find rule bodies that cover (large subsets of) the examples of that class. (UNORDERED) *rule sets*

Learning ordered rule lists

The key idea of this kind of rule learning algorithm is to keep growing a conjunctive rule body by adding the literal that most improves its homogeneity

It is natural to measure homogeneity in terms of purity, as we did with decision trees. You might think that adding a literal to a rule body is much the same as adding a binary split to a decision tree, as the added literal splits the instances covered by the original rule body in two groups: those instances for which the new literal is true, and those for which the new literal is false.

in decision tree learning we are interested in the purity of *both* children, In rule learning, on the other hand, we are only interested in the purity of one of the children: the one in which the added literal is true.

In fact, it doesn't even matter which of those impurity measures we use to guide the search, since they will all give the same result. To see this, notice that the impurity of a concept decreases with the empirical probability $p^{\cdot}$ (the relative frequency of covered positives) if $p^{\cdot} > 1/2$ and increases with $p^{\cdot}$ if $p^{\cdot} < 1/2$; see Figure 6.1.

We introduce the main algorithm for learning rule lists by means of an example.

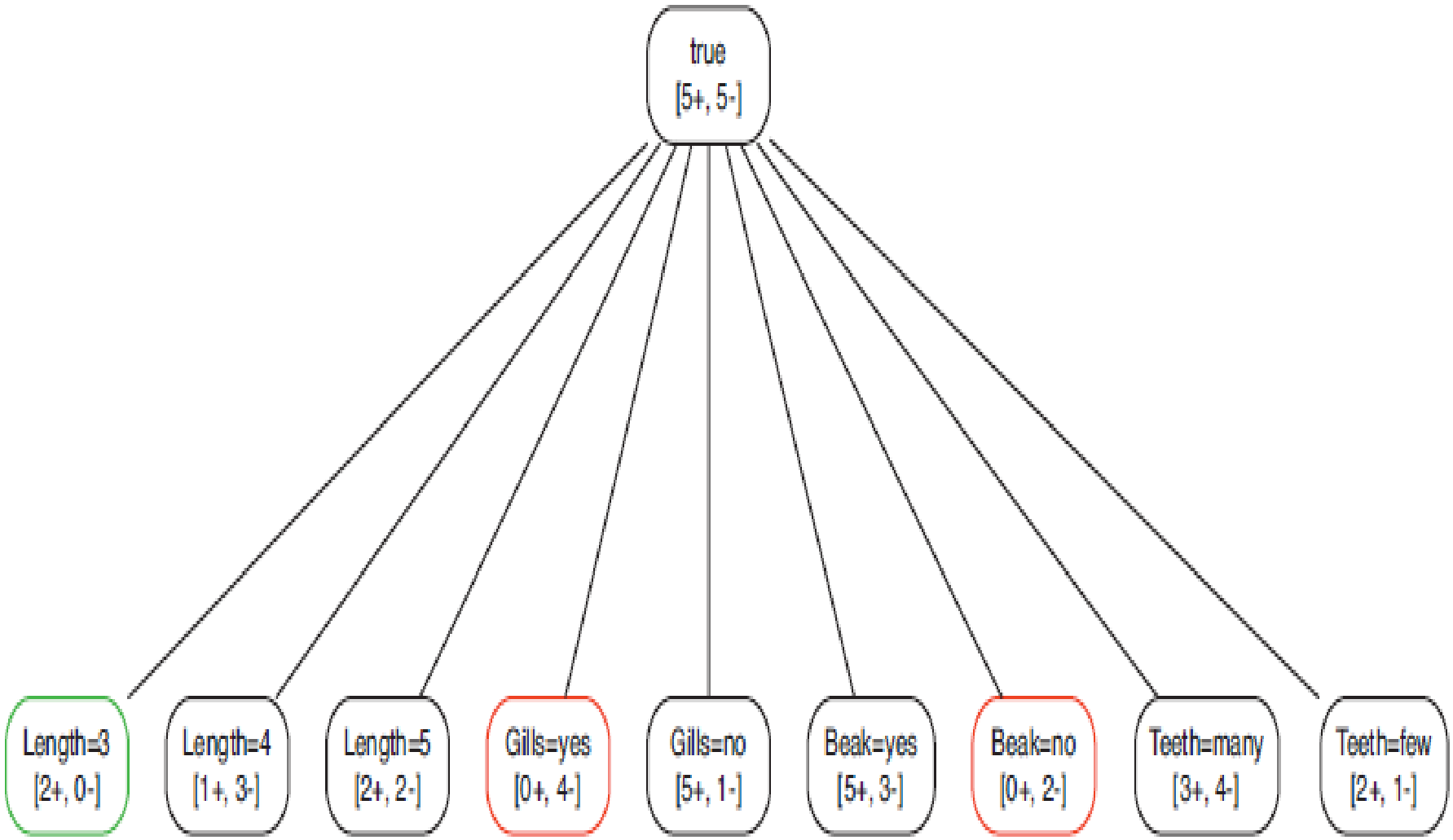
Example 6.1 (Learning a rule list). Consider again our small dolphins data set with positive examples

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 negatives

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 nine possible literals are shown with their coverage counts in Figure 6.2 (top). Three of these are pure; in the impurity isometrics plot in Figure 6.2 (bottom) they end up on the x-axis and y-axis. One of the literals covers two positives and two negatives, and therefore has the same impurity as the overall data set; this literal ends up on the ascending diagonal in the coverage plot.


true
[5+, 5-]
Length=3
[2+, 0-]
Length=4
[1+, 3-]
Length=5
[2+, 2-]
Gills=yes
[0+, 4-]
Gills=no
[5+, 1-]
Beak=yes
[5+, 3-]
Beak=no
[0+, 2-]
Teeth=many
[3+, 4-]
Teeth=few
[2+, 1-]

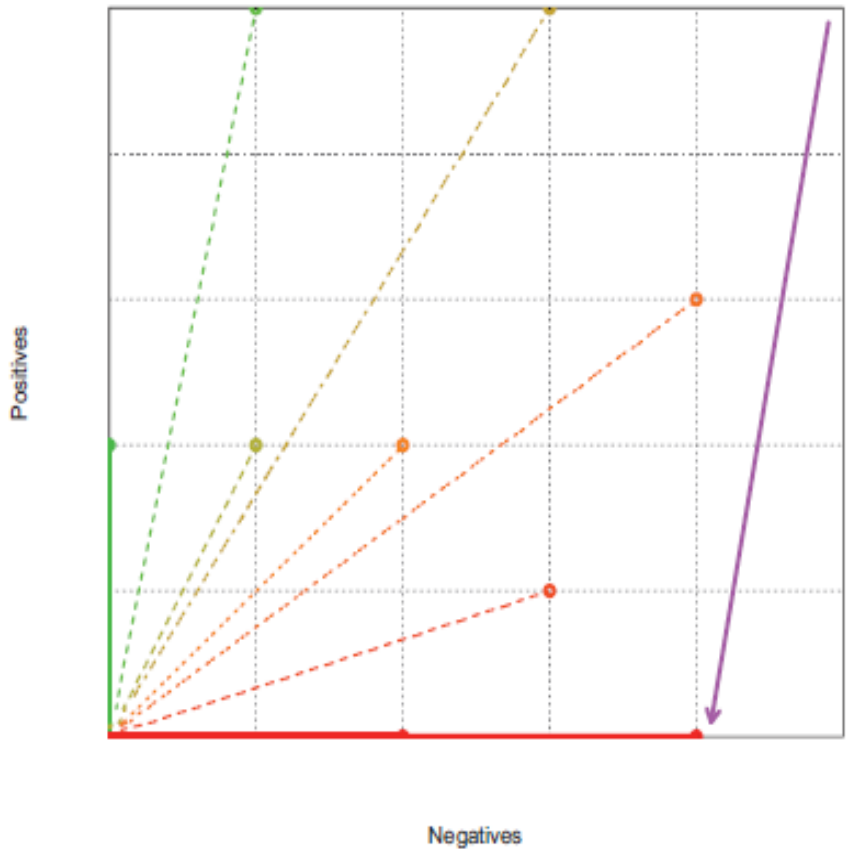


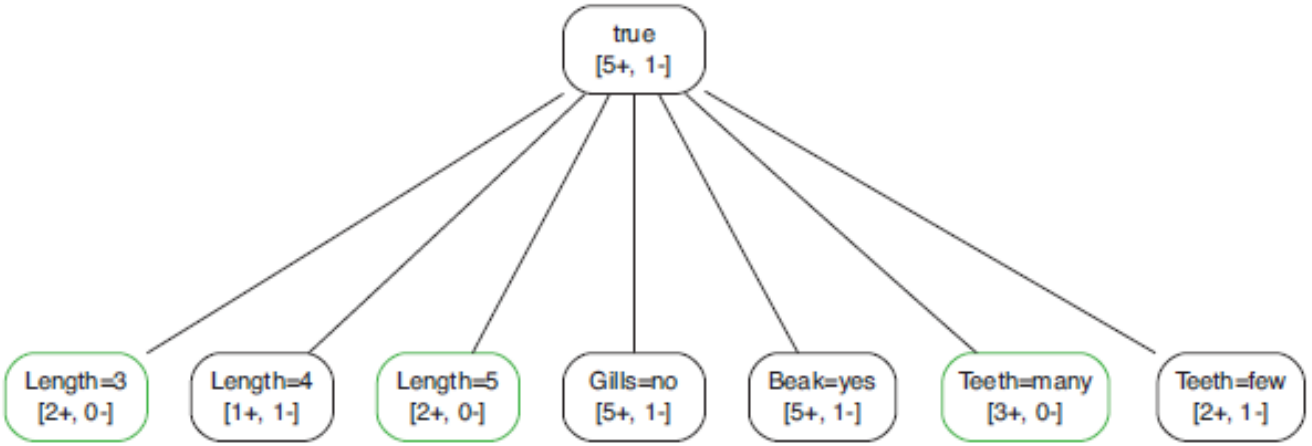
Figure 6.2. **(top)** All literals with their coverage counts on the data in Example 6.1. The ones in green (red) are pure for the positive (negative) class. **(bottom)** The nine literals plotted as points in coverage space, with their impurity values indicated by impurity isometrics (away from the ascending diagonal is better). Impurity values are colour-coded: towards green if $\dot{p} > 1/2$, towards red if $\dot{p} < 1/2$, and orange if $\dot{p} = 1/2$ (on a 45 degree isometric). The violet arrow indicates the selected literal, which excludes all five positives and one negative.

The corresponding coverage point is indicated by the arrow in Figure 6.2 (bottom). You can think of this arrow as the right-most bit of the coverage curve that results if we keep on following a downward path through the hypothesis space by adding literals.

Most rule learning algorithms now proceed as follows: they remove the examples covered by the rule just learned from consideration, and proceed with the remaining examples. This strategy is called *separate-and-conquer*, in analogy with the divideand- conquer strategy of decision trees

As is shown in Figure 6.3, we can understand this as working in a smaller coverage space. After going through the numbers, we find the next rule learned is

·**if** Teeth = many **then** Class = ⊕·



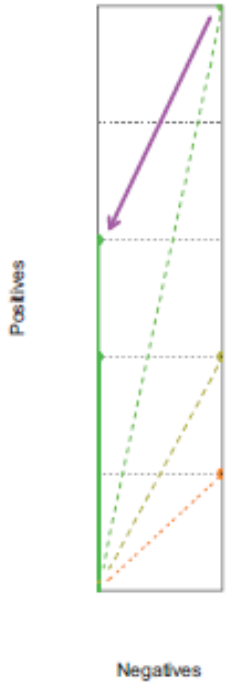


Figure 6.3. **(left)** Revised coverage counts after removing the four negative examples covered by the first rule found (literals not covering any examples are omitted). **(right)** We are now operating in the right-most 'slice' of Figure 6.2 on p.160.

As I mentioned earlier, we should be cautious when interpreting this rule on its own, as against the original data set it actually covers more negatives than positives! In other words, the rule implicitly assumes that the previous rule doesn't 'fire'; in the final rule model we will precede it with 'else'.

We are now left with two positives and one negative (Figure 6.4). This time it makes sense to choose the rule that covers the single remaining negative, which is

if Length = 4 **then** Class = ⊖

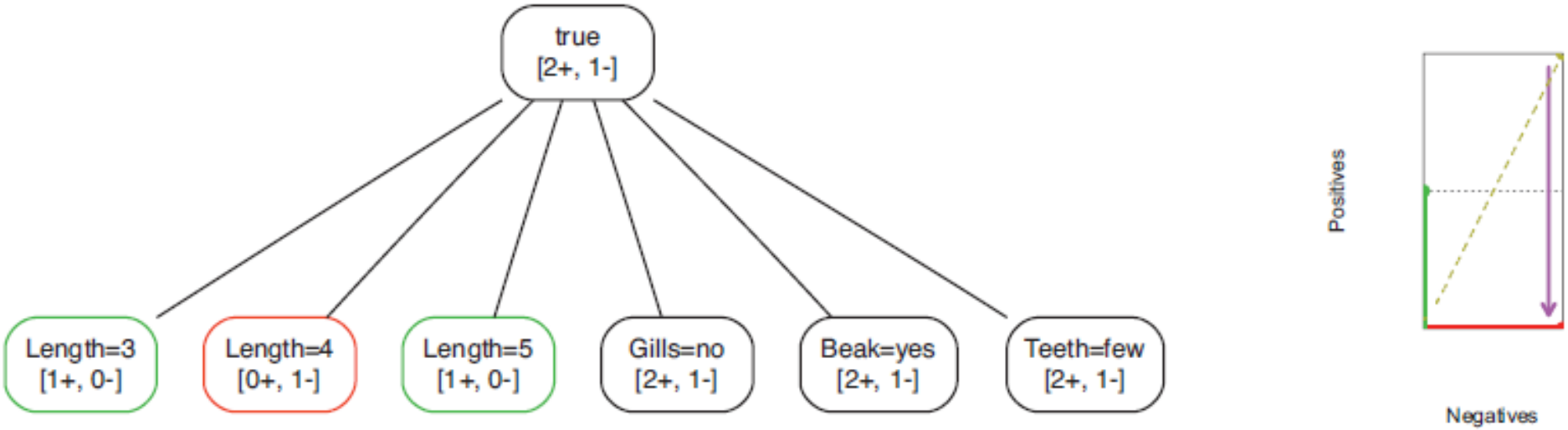


Figure 6.4. **(left)** The third rule covers the one remaining negative example, so that the remainin positives can be swept up by a default rule. **(right)** This will collapse the coverage space.

then as follows:

·**if** Gills = yes **then** Class = ⊖·

·**else if** Teeth = many **then** Class = ⊕·

·**else if** Length = 4 **then** Class = ⊖·

·**else** Class = ⊕·

Organising rules in a list is one way of dealing with overlaps among rules.

For example, consider the following rule list:

·**if** $P \wedge Q$ **then** Class = $\oplus$·

·**else if** R **then** Class = $\ominus$·

If we wanted to make these mutually exclusive the second rule would become

·**if** $\neg(P \wedge Q) \wedge R$ **then** Class = $\ominus$·

or equivalently,

·**if** $(\neg P \vee \neg Q) \wedge R$ **then** Class = $\ominus$·

Algorithm 6.1: LearnRuleList(D) – learn an ordered list of rules.

Input : labelled training data D.

Output : rule list R.

```
R ← ∅;
while D ≠ ∅ do
    r ← LearnRule(D) ;          // LearnRule: see Algorithm 6.2
    append r to the end of R;
    D ← D \ {x ∈ D | x is covered by r};
end
return R
```

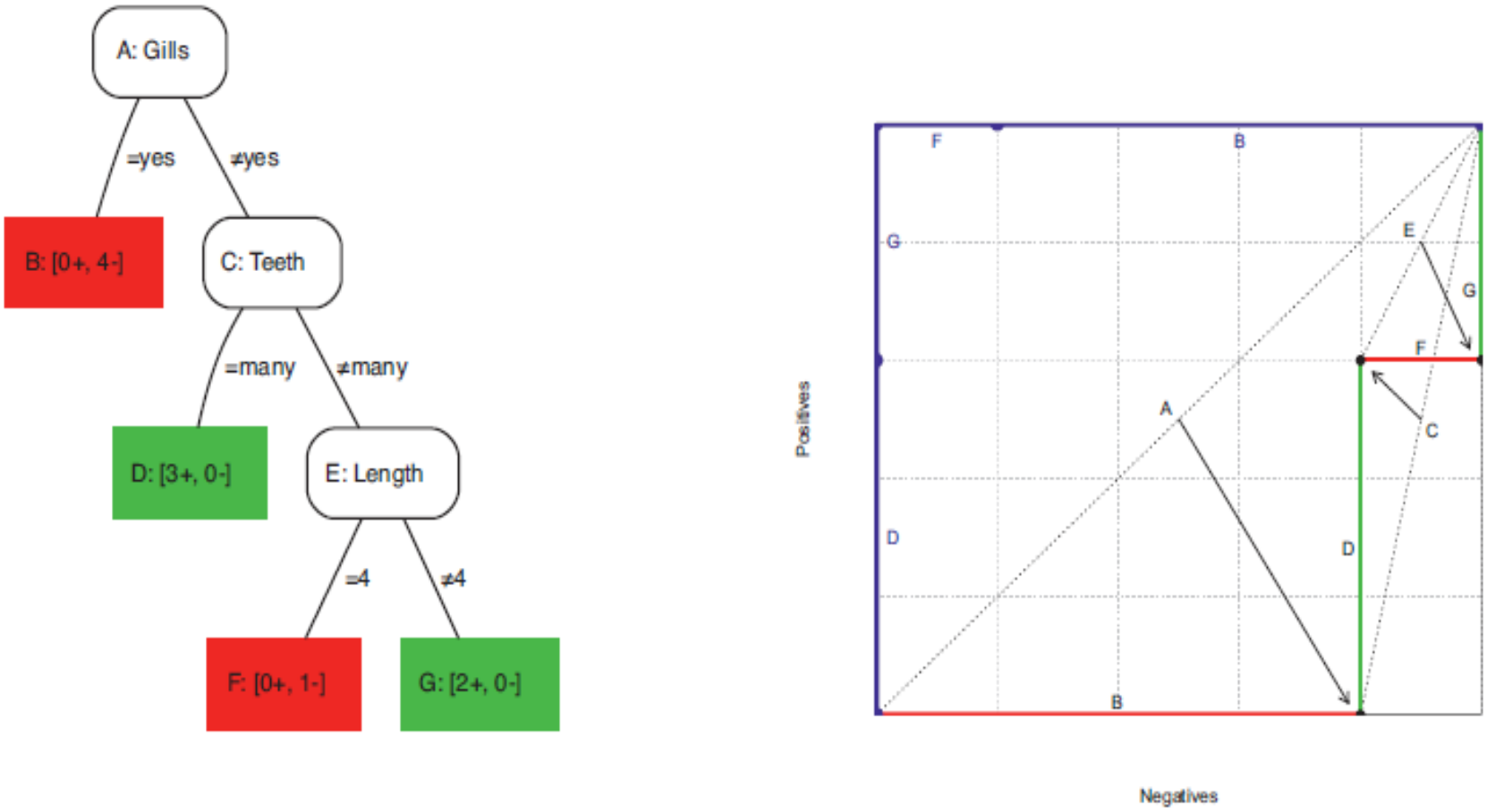


Figure 6.5. **(left)** A right-branching feature tree corresponding to a list of single-literal rules. **(right)** The construction of this feature tree depicted in coverage space. The leaves of the tree are either purely positive (in green) or purely negative (in red). Reordering these leaves on their empirical probability results in the blue coverage curve. As the rule list separates the classes this is a perfect coverage curve.

Rule lists for ranking and probability estimation

Turning a rule list into a ranker or probability estimator is as easy as it was for decision trees. Due to the covering algorithm we have access to the local class distributions associated with each rule.

Example 6.2 (Rule lists as rankers). Consider the following two concepts:

(A)	Length = 4	p2	n2, n4–5
(B)	Beak = yes	p1–5	n1–2, n5

Indicated on the right is each concept's coverage over the whole training set. Using these concepts as rule bodies, we can construct the rule list AB:

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