What's next?

6 Rule models

- Learning ordered rule lists
  - Rule lists for ranking and probability estimation
- Learning unordered rule sets
  - Rule sets for ranking and probability estimation
  - A closer look at rule overlap ★
- Descriptive rule learning
  - Rule learning for subgroup discovery
  - Association rule mining

What's next?

Example 6.1, p.159

Learning a rule list I

Consider again our small dolphins data set with positive examples

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: $\text{Length} = 3 \wedge \text{Gills} = \text{no} \wedge \text{Beak} = \text{yes} \wedge \text{Teeth} = \text{few}$

p4: $\text{Length} = 5 \wedge \text{Gills} = \text{no} \wedge \text{Beak} = \text{yes} \wedge \text{Teeth} = \text{many}$

p5: $\text{Length} = 5 \wedge \text{Gills} = \text{no} \wedge \text{Beak} = \text{yes} \wedge \text{Teeth} = \text{few}$

and negatives

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

n2: $\text{Length} = 4 \wedge \text{Gills} = \text{yes} \wedge \text{Beak} = \text{yes} \wedge \text{Teeth} = \text{many}$

n3: $\text{Length} = 5 \wedge \text{Gills} = \text{yes} \wedge \text{Beak} = \text{no} \wedge \text{Teeth} = \text{many}$

n4: $\text{Length} = 4 \wedge \text{Gills} = \text{yes} \wedge \text{Beak} = \text{no} \wedge \text{Teeth} = \text{many}$

n5: $\text{Length} = 4 \wedge \text{Gills} = \text{no} \wedge \text{Beak} = \text{yes} \wedge \text{Teeth} = \text{few}$

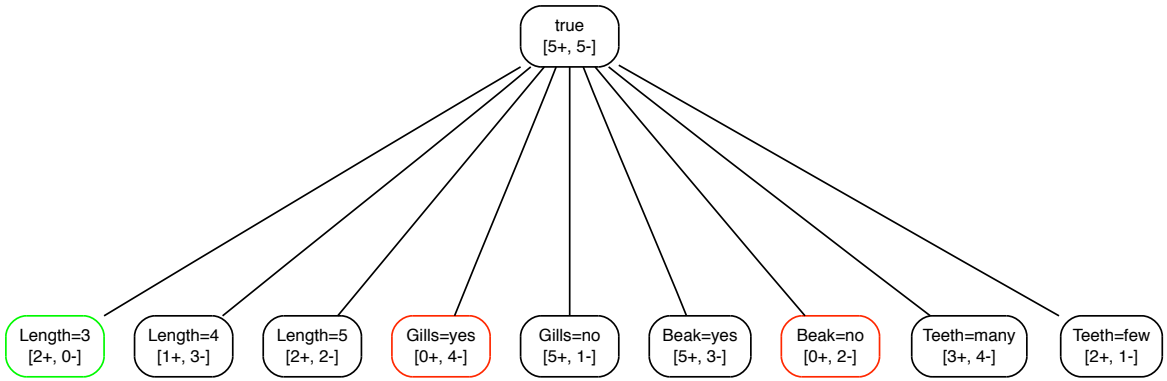
Example 6.1, p.159

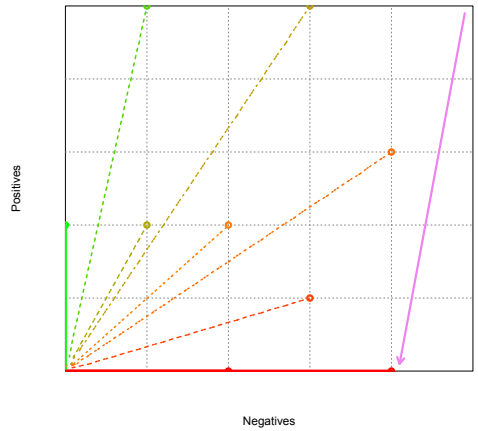
Learning a rule list II

- The nine possible literals are shown with their coverage counts in Figure 6.2 (left).
- Three of these are pure; in the impurity isometrics plot in Figure 6.2 (right) 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.

Figure 6.2, p.160

Searching for literals

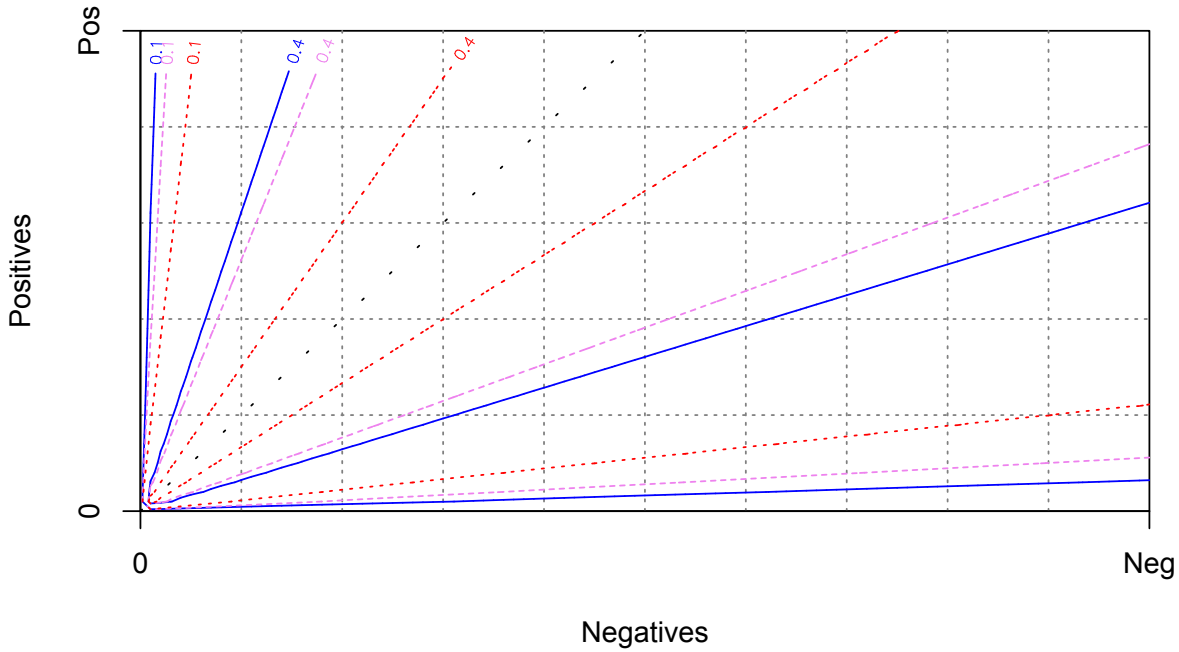




(left) All literals with their coverage counts on the data in Example 6.1. The ones in green (red) are pure for the positive (negative) class. **(right)** The nine literals plotted as points in coverage space, with their impurity values indicated by impurity isometrics (away from the ascending diagonal 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.

Figure 6.1, p.158

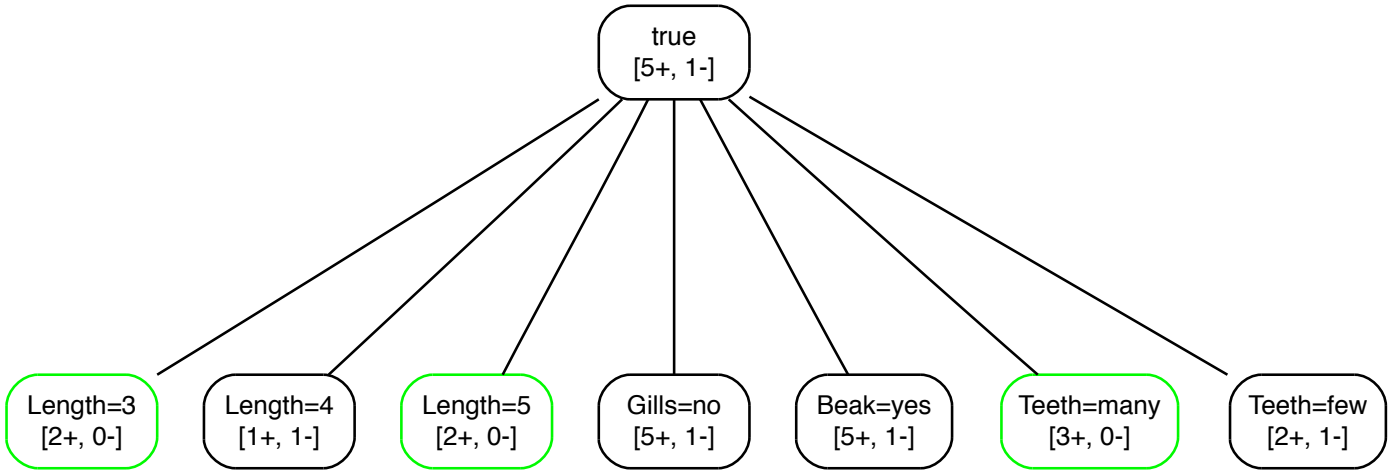
Equivalence of search heuristics

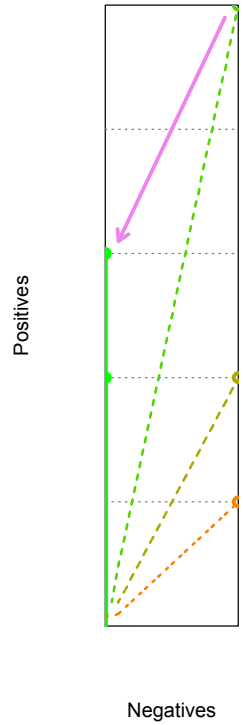


ROC isometrics for entropy (rescaled to have a maximum value of 1/2), Gini index and minority class. The grey dotted symmetry line is defined by $\dot{p} = 1/2$: each isometric has two parts, one above the symmetry line (where impurity decreases with increasing empirical probability $\dot{p}$) and its mirror image below the symmetry line (where impurity is proportional to $\dot{p}$).

Figure 6.3, p.161

Constructing the second rule

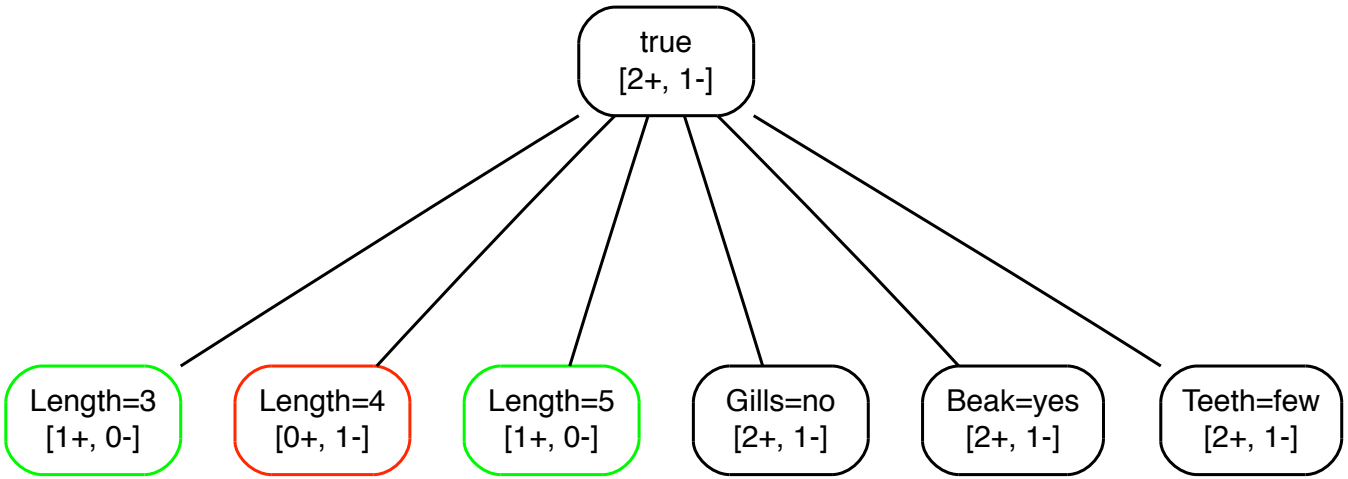


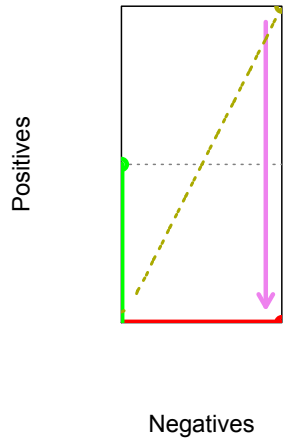


(left) Revised coverage counts after removing the four negative examples covered by the first rule found (literals not covering any examples are omitted). **(right)** We are now operating in the right-most 'slice' of Figure 6.2.

Figure 6.4, p.162

Constructing the third rule





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

Algorithm 6.1, p.163

Learning an ordered list of rules

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

Input : labelled training data D.
Output : rule list R.

$R \leftarrow \emptyset$;
while $D \neq \emptyset$ **do**
  $r \leftarrow$LearnRule(D) ; // LearnRule: see Algorithm 6.2
  append r to the end of R;
  $D \leftarrow D \setminus \{x \in D | x \text{ is covered by } r\}$;
end
return R

Algorithm 6.2, p.164

Learning a single rule

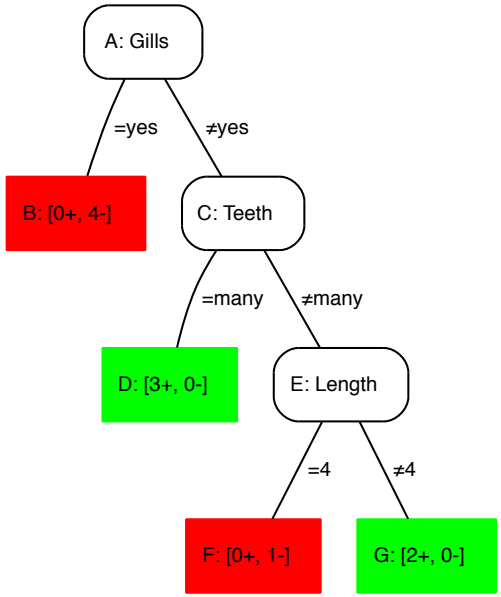
Algorithm LearnRule(D) – learn a single rule.

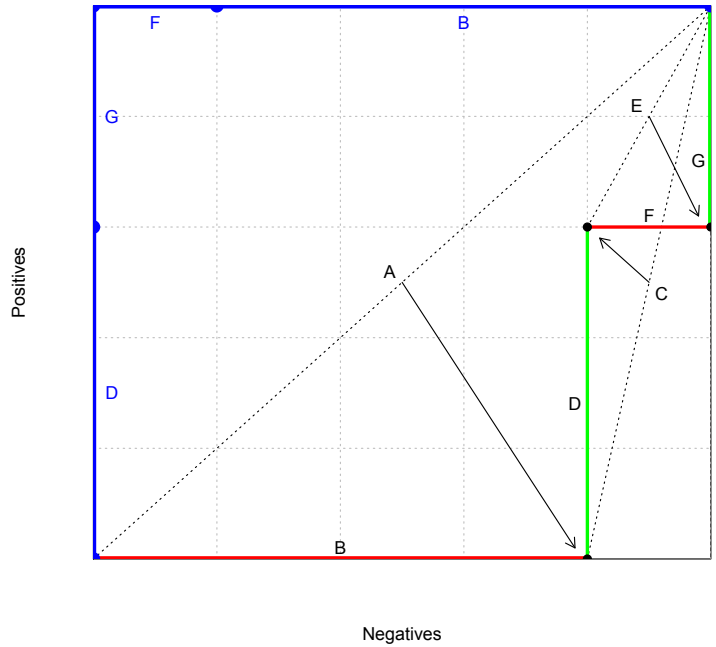
Input : labelled training data D.
Output : rule r.

$b \leftarrow$ true;
$L \leftarrow$ set of available literals;
while not Homogeneous(D) **do**
    $l \leftarrow$ BestLiteral(D, L) ; // e.g., highest purity; see text
    $b \leftarrow b \wedge l$;
    $D \leftarrow \{x \in D | x$ is covered by $b\}$;
    $L \leftarrow L \setminus \{l' \in L | l'$ uses same feature as $l\}$;
end
$C \leftarrow$ Label(D) ; // e.g., majority class
$r \leftarrow$ ·**if** b **then** Class $= C$·;
return r

Figure 6.5, p.164

Rule list as a tree





(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.

Important point to remember

Rule lists inherit the property of decision trees that their training set coverage curve is always convex.

What's next?

Example 6.3, p.167

Learning a rule set for class $\oplus$

Figure 6.7 shows that the first rule learned for the positive class is

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

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:

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

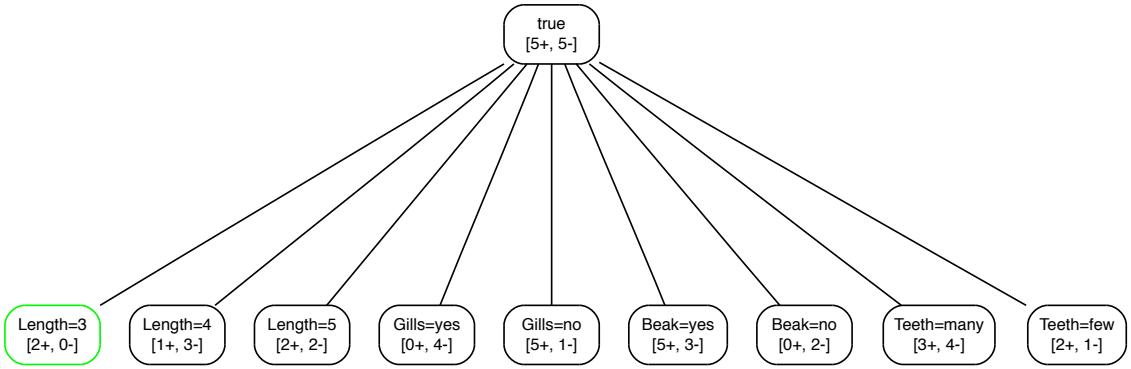
To cover the remaining positive, we again need a rule with two conditions (Figure 6.9):

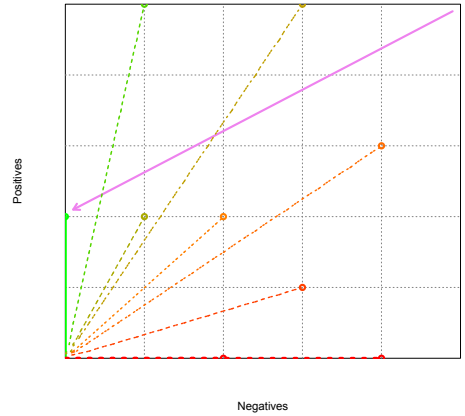
$$\cdot\textbf{if } \text{Gills} = \text{no} \wedge \text{Teeth} = \text{many} \textbf{ then } \text{Class} = \oplus\cdot$$

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, p.168

Learning a rule set

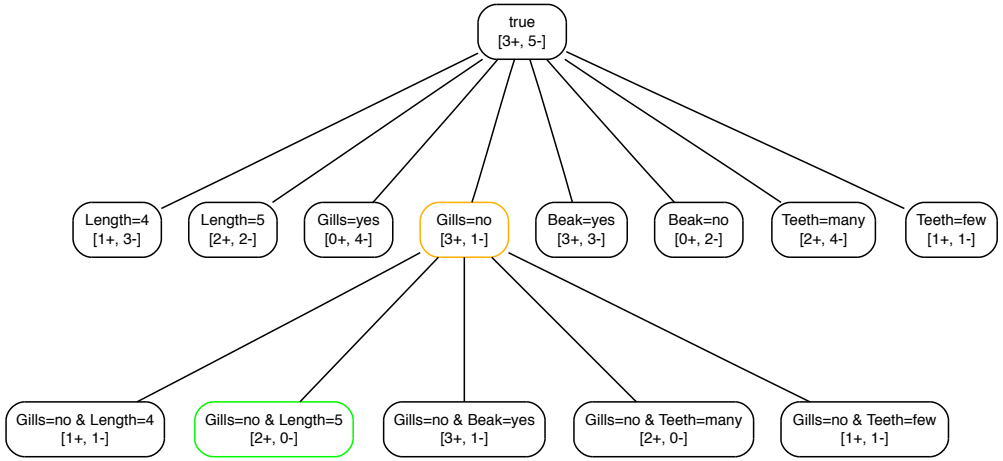


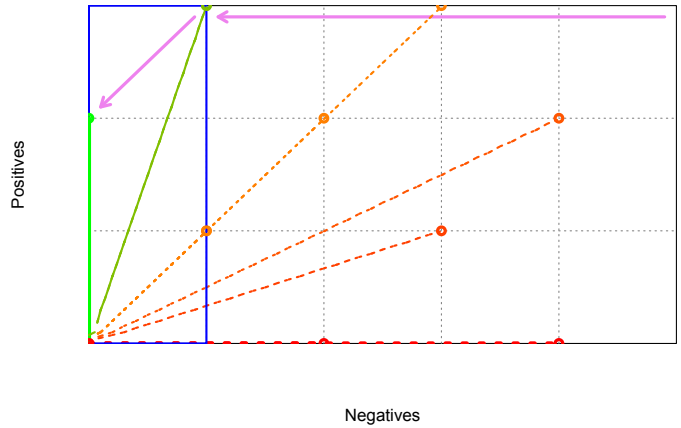


(left) The first rule is learned for the positive class. **(right)** Precision isometrics look identical to impurity isometrics (Figure 6.2); however, the difference is that precision is lowest on the x-axis and highest on the y-axis, while purity is lowest on the ascending diagonal and highest on both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

Figure 6.8, p.169

Learning the second rule

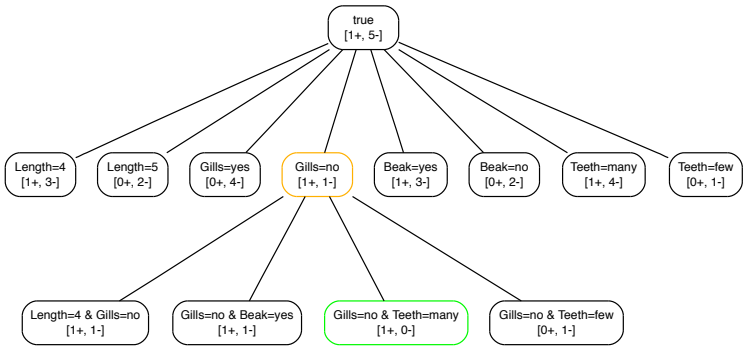


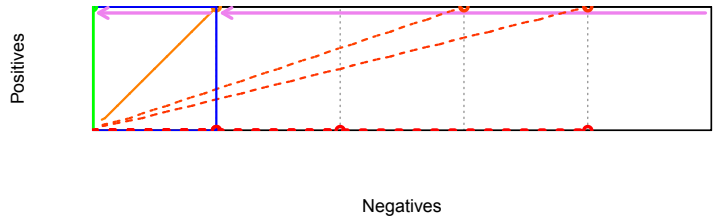


(left) The second rule needs two literals: we use maximum precision to select both. **(right)** The coverage space is smaller because the two positives covered by the first rule are removed. The blue box on the left indicates an even smaller coverage space in which the search for the second literal is 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, p.170

Learning the third rule





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

Algorithm 6.3, p.171

Learning an unordered set of rules

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

Input : labelled training data D.
Output : rule set R.

```
1  R ← ∅;
2  for every class C_i do
3  |   D_i ← D;
4  |   while D_i contains examples of class C_i do
5  |   |   r ← LearnRuleForClass(D_i, C_i) ;  // LearnRuleForClass: see Algorithm
   |   |   6.4
6  |   |   R ← R ∪ {r};
7  |   |   D_i ← D_i \ {x ∈ C_i | x is covered by r} ;        // remove only positives
8  |   end
9  end
10 return R
```

Algorithm 6.4, p.171

Learning a single rule for a given class

Algorithm 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
```

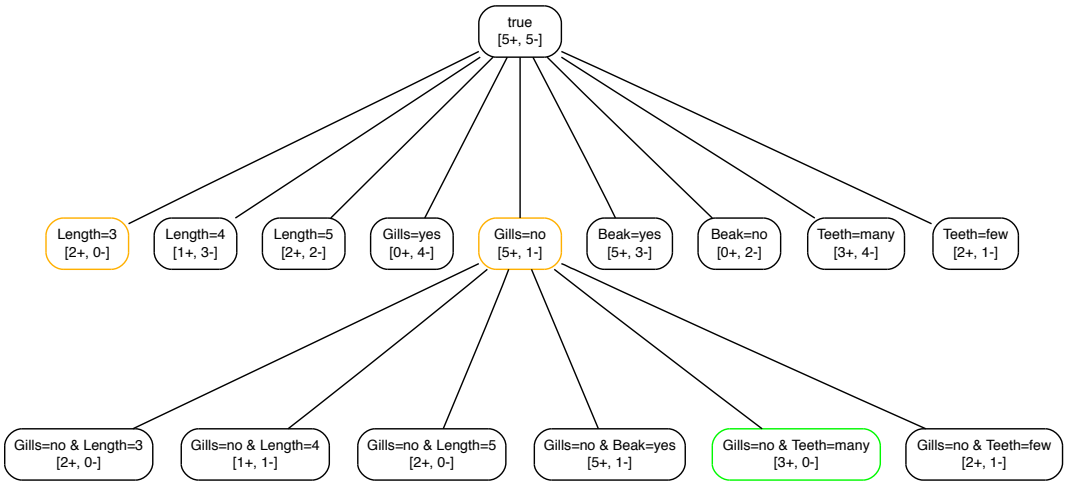
The need for probability smoothing

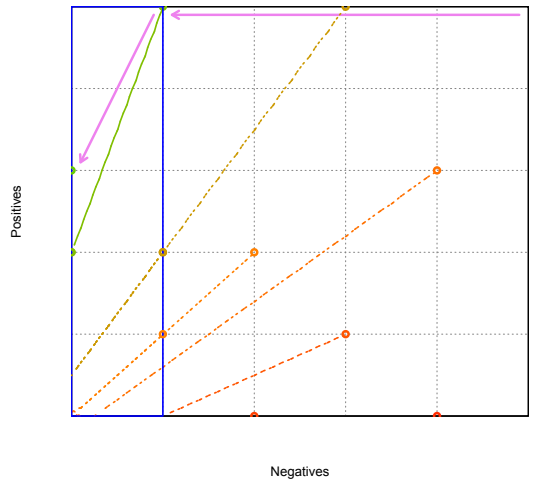
One issue with using precision as search heuristic is that it tends to focus a bit too much on finding pure rules, thereby occasionally missing near-pure rules that can be specialised into a more general pure rule.

- Consider Figure 6.10 (left): precision favours the rule ·**if** Length = 3 **then** Class = ⊕·, even though the near-pure literal Gills = no leads to the pure rule ·**if** Gills = no ∧ Teeth = many **then** Class = ⊕·.
- A convenient way to deal with this 'myopia' of precision is the Laplace correction, which ensures that $[5+,1-]$ is 'corrected' to $[6+,2-]$ and thus considered to be of the same quality as $[2+,0-]$ aka $[3+,1-]$ (Figure 6.10 (right)).

Figure 6.10, p.172

Using the Laplace correction





(left) Using Laplace-corrected precision allows learning a better rule in the first iteration. **(right)** Laplace correction adds one positive and one negative pseudo-count, which means that the isometrics now rotate around $(-1,-1)$ in coverage space, resulting in a preference for more general rules.