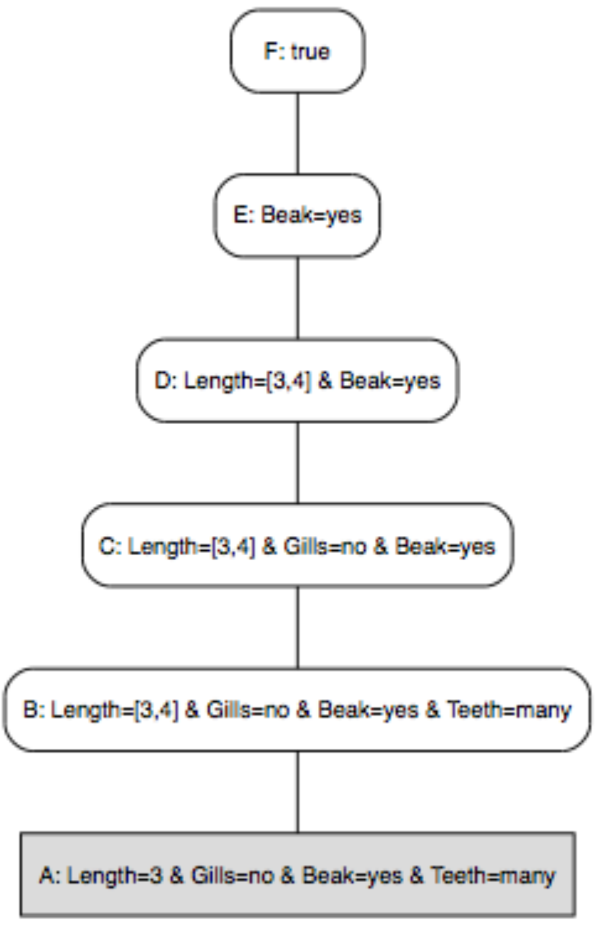
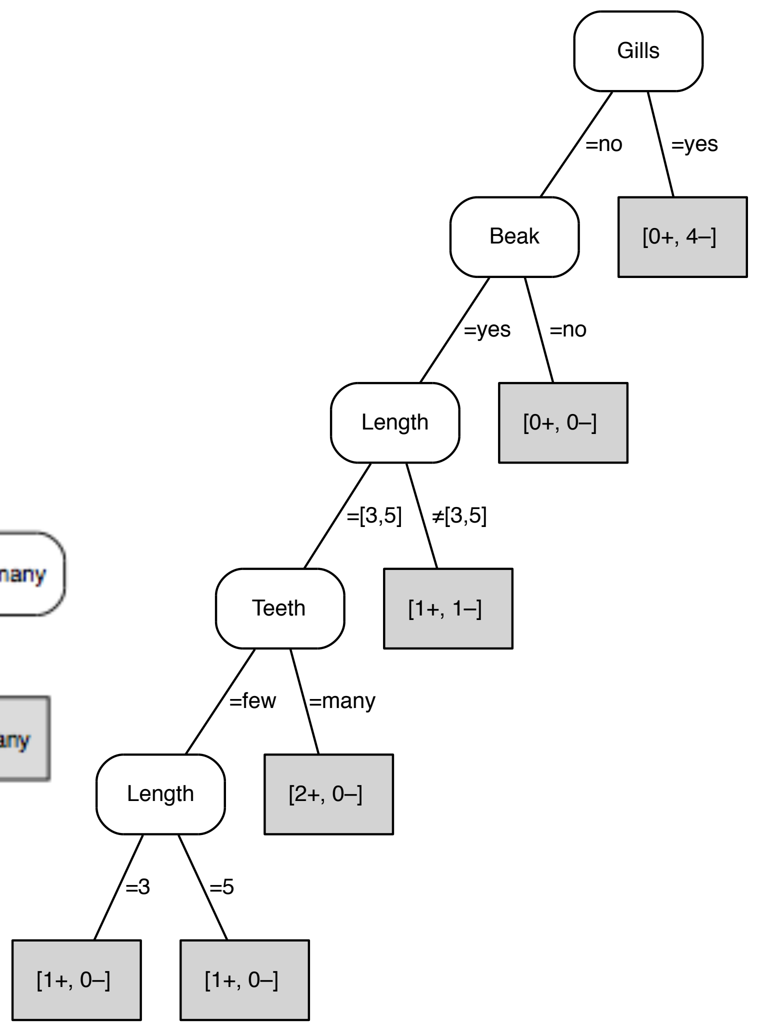
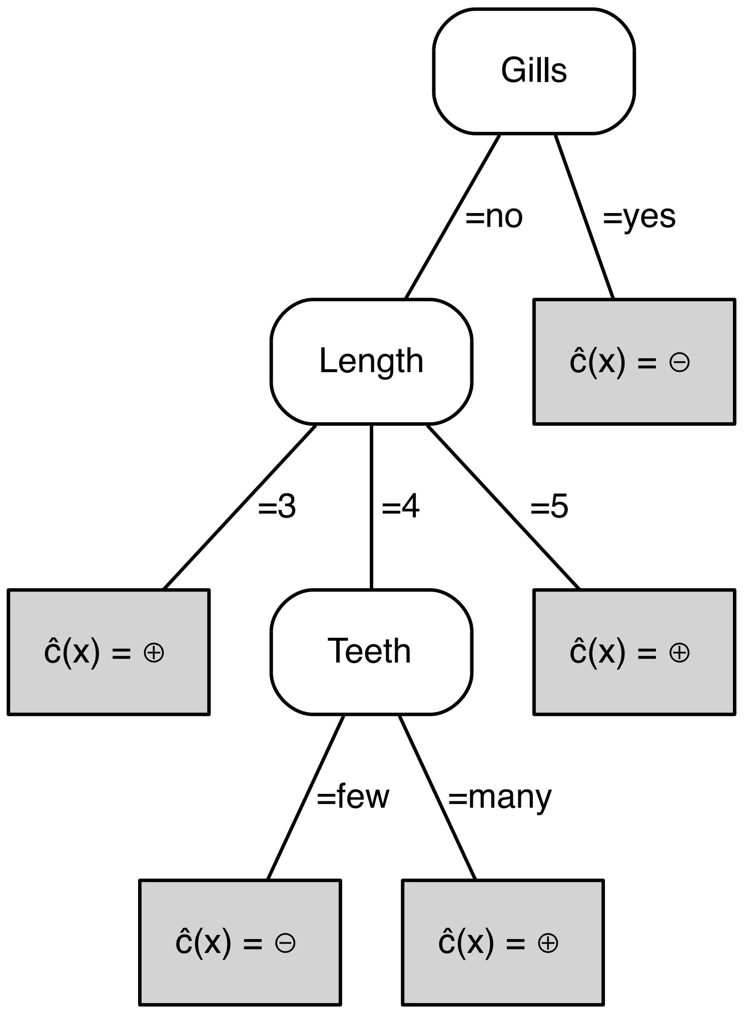
What's next?

5 Tree models
- Decision trees
- Ranking and probability estimation trees
  - Sensitivity to skewed class distributions
- Tree learning as variance reduction
  - Regression trees
  - Clustering trees

Paths as trees


F: true
E: Beak=yes
D: Length=[3,4] & Beak=yes
C: Length=[3,4] & Gills=no & Beak=yes
B: Length=[3,4] & Gills=no & Beak=yes & Teeth=many
A: Length=3 & Gills=no & Beak=yes & Teeth=many





(left) The path from Figure 4.6, redrawn in the form of a tree. The coverage numbers in the leaves are obtained from the data in Example 4.4. **(right)** A decision tree learned on the same data. This tree separates the positives and negatives perfectly.

Example 4.4, p.115 The dataset

Suppose we have the following five positive examples (the first three are the same as in Example 4.1):

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

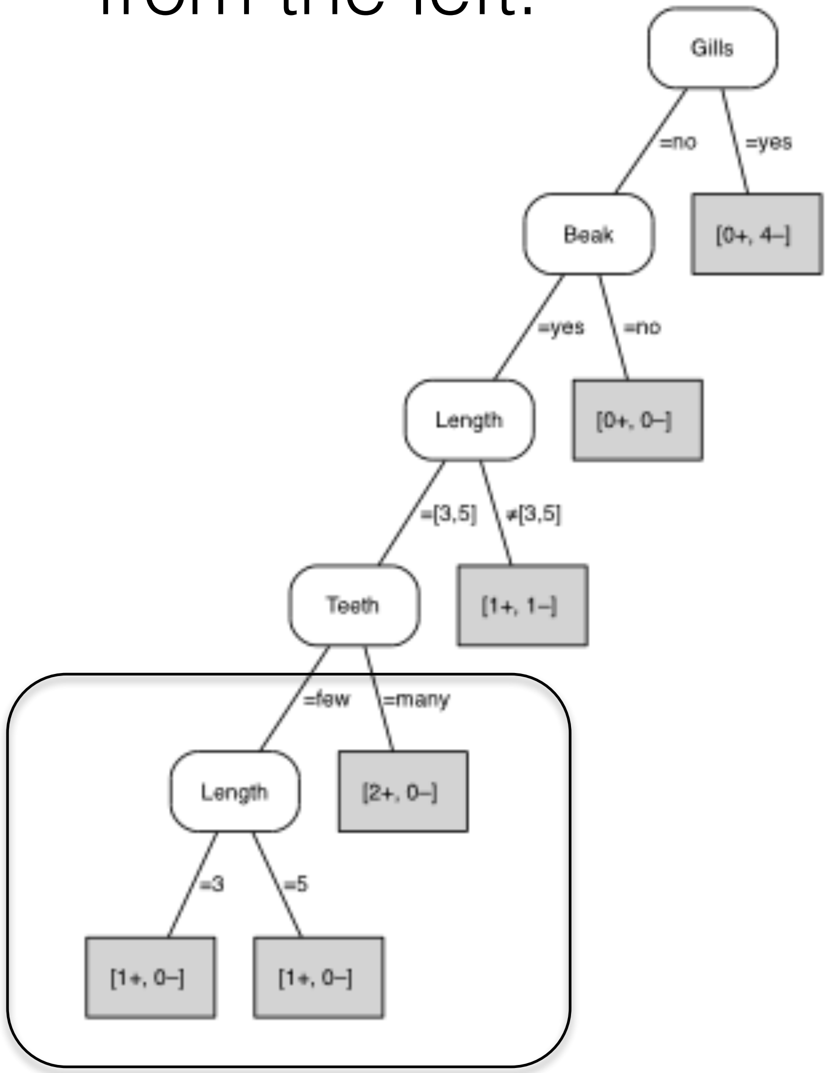
and the following negatives (the first one is the same as in Example 4.2):

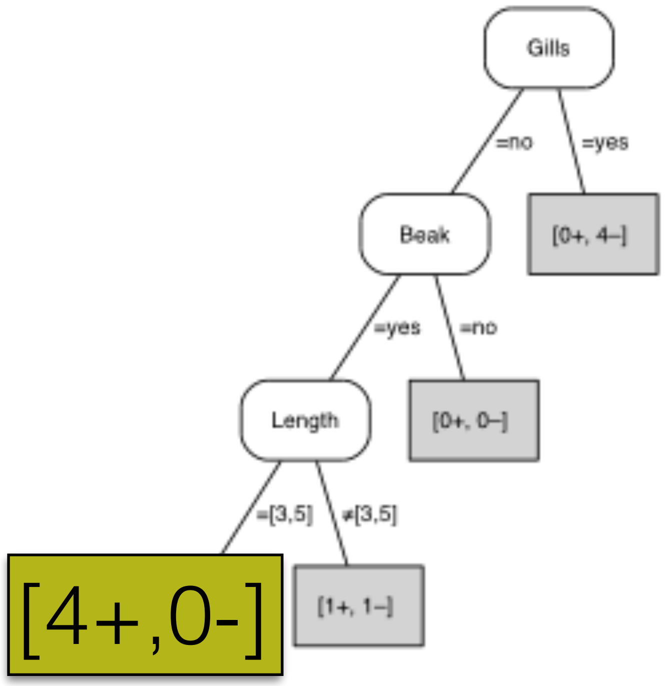
n1: Length = 5 ∧ Gills = yes ∧ Beak = yes ∧ Teeth = many
n2: Length = 4 ∧ Gills = yes ∧ Beak = yes ∧ Teeth = many
n3: Length = 5 ∧ Gills = yes ∧ Beak = no ∧ Teeth = many
n4: Length = 4 ∧ Gills = yes ∧ Beak = no ∧ Teeth = many
n5: Length = 4 ∧ Gills = no ∧ Beak = yes ∧ Teeth = few

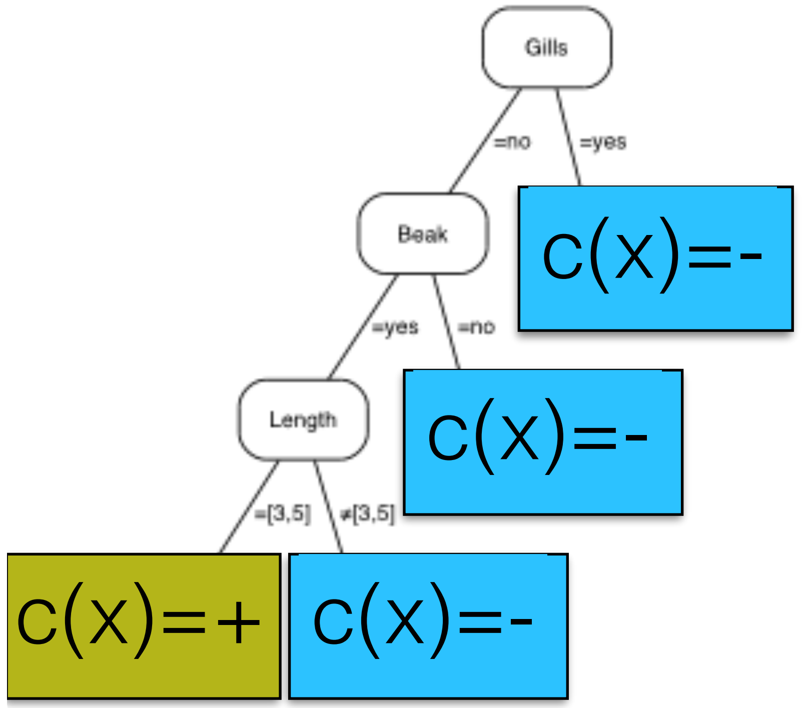
Modification of feature tree and transformation into a decision tree

- In order to represent the i-th concept from the bottom of the feature tree, we could prune the i leaves from the left and incorporate all of them in a single leaf

- For instance, we would like to represent the 3rd concept from the left:







- Turning the modified feature tree into a decision tree by labelling the leaves

Equivalent logical expressions

- From the feature tree we might obtain many equivalent logical expressions:

$$Gills = no \wedge Length = 3 \vee Gills = no \wedge Length = 5 \vee Gills = no \wedge Length = 4 \wedge Teeth = many$$

The above expression represents the positive concept;
other possibilities include for instance applying the distributive property $(A \wedge B) \vee (A \wedge C) = A \wedge (B \vee C)$ or applying the De Morgan laws.

Finally, we could represent the negative concept:
$(Gills=no \wedge Length=4 \wedge Teeth=few) \vee (Gills=yes)$
and then negate it.

Important point to remember

Decision trees are strictly more expressive than conjunctive concepts.

Decision trees correspond to logical expressions in Disjunctive Normal Form (DNF).
Suppose we build a tree, with each path corresponding to a conjunctive hypothesis covering one single example of the training set.
Then we have obtained a model, perfectly performing on the training set but which completely overfits it!

Important point to remember

One way to avoid overfitting and encourage learning is to deliberately choose a restrictive hypothesis language.

This is the inductive bias often reached by learning algorithms by:

- the way the hypothesis space is searched
- incorporating a penalty for the complexity of each hypothesis in the objective function

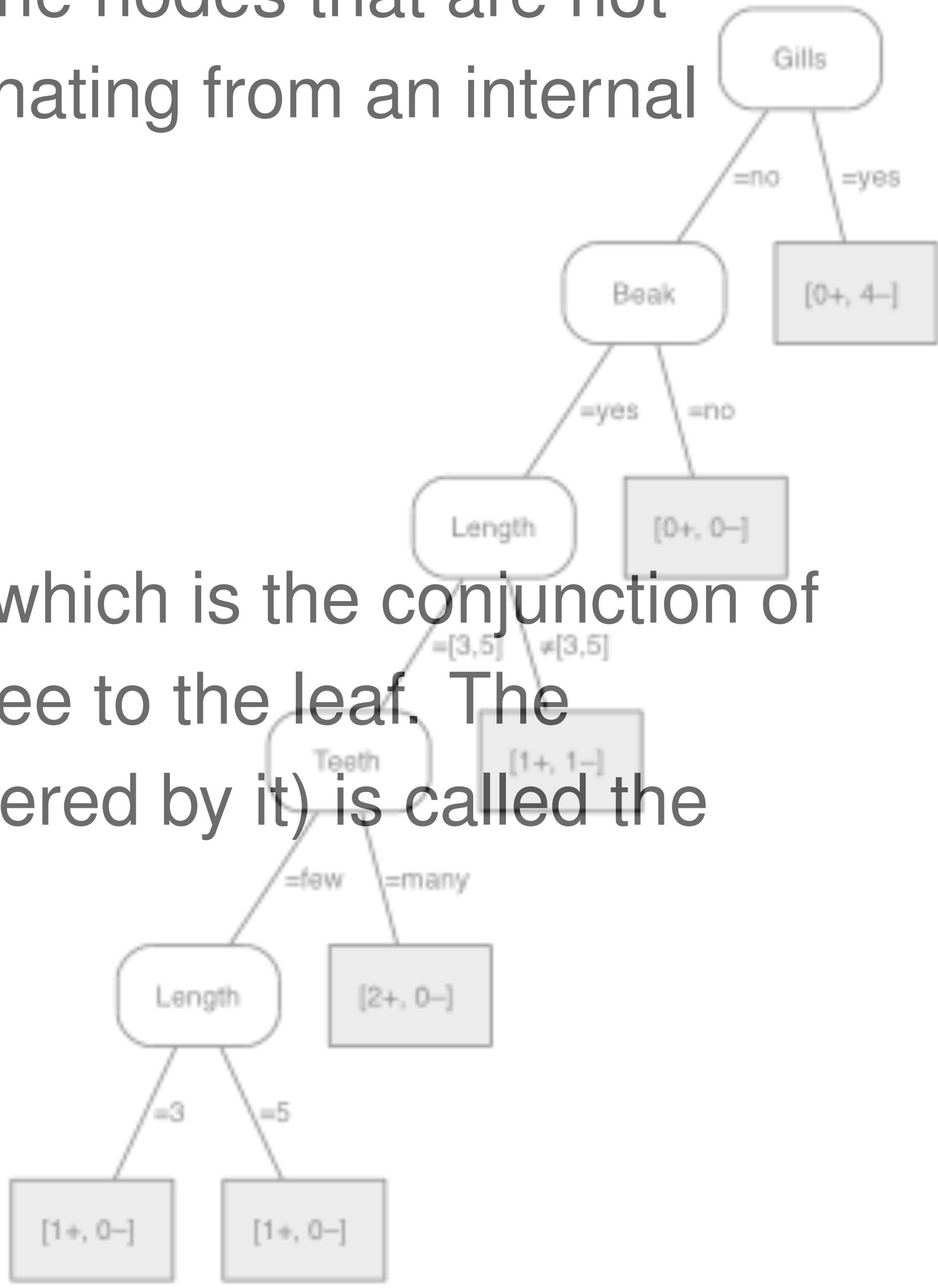
Definition 5.1, p.132

Feature tree

A *feature tree* is a tree such that each internal node (the nodes that are not leaves) is labelled with a feature, and each edge emanating from an internal node is labelled with a literal.

The set of literals at a node is called a *split*.

Each leaf of the tree represents a logical expression, which is the conjunction of literals encountered on the path from the root of the tree to the leaf. The extension of that conjunction (the set of instances covered by it) is called the *instance space segment* associated with the leaf.



Learning a feature tree

- A feature tree is a compact way of representing a number of conjunctive concepts in a hypothesis space
- The learning problem is to decide which of the possible concepts will be the best to solve the given task
- When we will see rule learners, we will see that they learn one concept at a time
- Instead, tree learners perform a top-down search for all these concepts at once

Algorithm 5.1, p.132

Growing a feature tree

A generic learning procedure for most tree learners

Algorithm GrowTree(D, F) – grow a feature tree from training data.

```
Input  : data $D$; set of features $F$.
Output : feature tree $T$ with labelled leaves.
1 if Homogeneous($D$) then return Label($D$);
2 $S \leftarrow$ BestSplit($D, F$) ;          // e.g., BestSplit-Class (Algorithm 5.2)
3 split $D$ into subsets $D_i$ according to the literals in $S$;
4 for each $i$ do
5     if $D_i \neq \emptyset$ then $T_i \leftarrow$ GrowTree($D_i, F$) ;
6     else $T_i$ is a leaf labelled with Label($D$);
7 end
8 return a tree whose root is labelled with $S$ and whose children are $T_i$
```

Growing a feature tree

Algorithm 5.1 gives the generic learning procedure common to most tree learners. It assumes that the following three functions are defined:

Homogeneous(D) returns true if the instances in D are homogeneous enough to be labelled with a single label, and false otherwise;

Label(D) returns the most appropriate label for a set of instances D;

BestSplit(D, F) returns the best set of literals to be put at the root of the tree.

These functions depend on the task at hand: for instance, for classification tasks a set of instances is homogeneous if they are (mostly) of a single class, and the most appropriate label would be the majority class.(*) For clustering tasks a set of instances is homogenous if they are close together, and the most appropriate label would be some exemplar such as the mean.

(*) If there is more than a majority class, then one of them could be chosen uniformly random

Comments on GrowTree

- GrowTree is a *divide and conquer* algorithm.
  It divides the data into subsets and builds a feature tree for each subset. Then, (at return) it combines the trees into a single one.

- It is recursive, since each subproblem (building a tree for each subset of the data) is of the same form as the original problem.

- Line 1 and line 6 stop the recursion assigning a label to a leaf node.

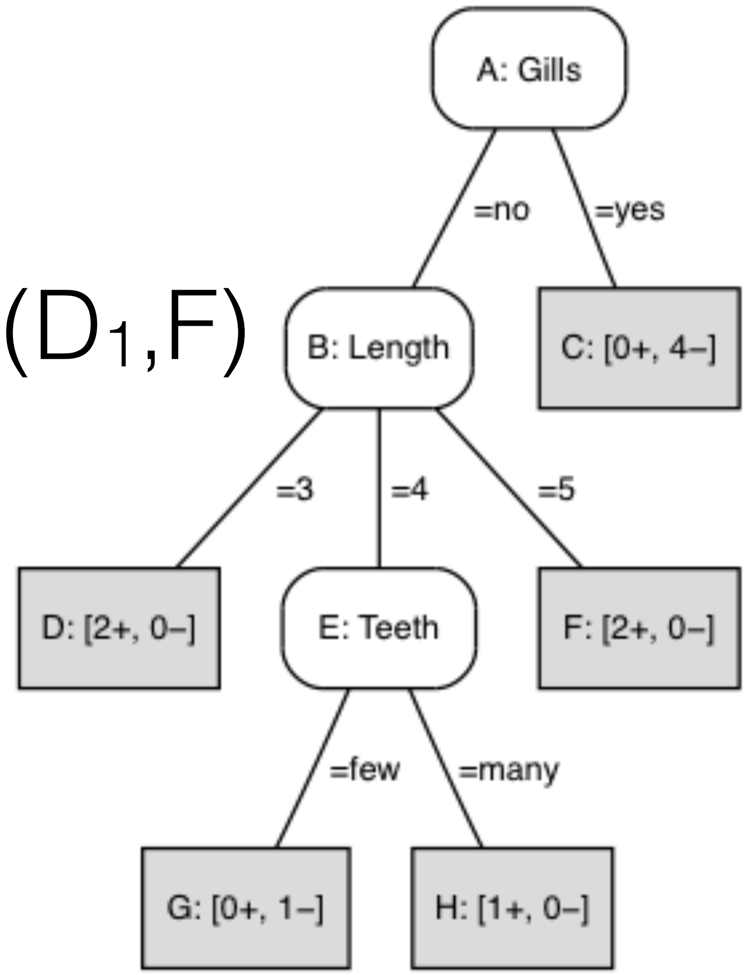
- It is greedy: it selects the best alternative and never reconsiders this choice (which might get to a sub-optimal solution).
  An alternative would be a backtracking search algorithm at the expense of an increased computation time and memory requirements.

Execution and calls of **GrowTree**

GrowTree(D,F) D=

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

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



GrowTree(D₁,F)

D_1=

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

n5: Length = 4 ∧ Gills = no ∧ Beak = yes ∧ Teeth = few

GrowTree(D_4,F)

p2: Length = 4 ∧ Gills = no ∧ Beak = yes ∧ Teeth = many

D_4=

n5: Length = 4 ∧ Gills = no ∧ Beak = yes ∧ Teeth = few

What’s next?

Notation

- Let us first suppose we are dealing with boolean features that perform a binary split of a set of instances D into D_1 and D_2.
- Let us also suppose we have 2 classes: + and -
- The goodness of a split is determined by the *purity* of D_1 and D_2 in terms of the class of their examples.
- The purity of a partition of n examples, containing n^+ examples of the positive class and n^- examples of the negative class, depends only on the relative magnitude of n^+ and n^- (and does not change if both of them are multiplied by the same amount).
  Purity can be defined in terms of the *empirical probability*:

$$\dot{p} = \frac{n^+}{n^+ + n^-}$$

- Furthermore, the result should not change if we swap the role of the positive with the negative.

Some ways to measure impurity

- **Minority class:** $min\{\dot{p}, 1-\dot{p}\}$ $1 - max\{\dot{p}_i\}$ *i>2 classes*
  or the error rate as it would be measured with the proportion of misclassified examples if all the examples in the leaf node were labelled with the majority class

- **Gini index:** $2\dot{p}(1-\dot{p})$ $\sum_i \dot{p}_i(1-\dot{p}_i)$
  it is the expected error if the examples in the leaf node were labelled randomly: positive with probability $\dot{p}$, negative with probability (1-$\dot{p}$). The probability of a false positive is $\dot{p}$(1-$\dot{p}$) and the probability of a false negative is (1-$\dot{p}$)$\dot{p}$.

- **Entropy:** $-\dot{p} \cdot log_2(\dot{p}) - (1-\dot{p}) \cdot log_2(1-\dot{p})$ $-\sum_i \dot{p}_i \cdot log_2(\dot{p}_i)$
  it is the expected information (in bits, see following slides on entropy) contained in a message informing us on the class of a randomly chosen example in the leaf node.

Figure 5.2, p.134

Measuring impurity I

Indicating the impurity of a single leaf D_j as $\text{Imp}(D_j)$, the impurity of a set of mutually exclusive leaves $\{D_1, \ldots, D_l\}$ is defined as a weighted average

$$\text{Imp}(\{D_1, \ldots, D_l\}) = \sum_{j=1}^{l} \underbrace{\boxed{\frac{|D_j|}{|D|}}}_{\text{probability of an example falls into the partition } D_j} \text{Imp}(D_j)$$

where $D = D_1 \cup \ldots \cup D_l$.

Decision Trees

- Separate the dataset into disjoint partitions guided by the objective function:
  - objective function: each partition is pure in the target attribute

- The objective function is to measure purity of partitions obtained after split.
  One such measure is entropy

- **Entropy** is a measure of the *amount of confusion*
  - try to reduce the entropy of the target attribute at each partition

Entropy

- Entropy is measured in **bits**
- It can be used both to:
  - quantify the number of units (bits) required to represent the information (and eventually to communicate it by a message)
  - measure the amount of information associated to the message
- If an experiment has n possible (equiprobable) outcomes, the number of bits required to represent any of these outcomes is b:

$$b = \lceil \log_2 n \rceil$$

Example 1

- Suppose we roll a die. It has 6 outcomes.
- If we want to represent and communicate one outcome we need b bits:

$$b = \lceil \log_2 6 \rceil = \lceil 2.58 \rceil = 3$$

- If we communicate first, only the parity of the outcome, we need b_1 bits:

$$b_1 = \lceil \log_2 2 \rceil = 1$$

- After the first message, some confusion about the outcome remains, but this confusion is reduced by the amount of information already sent about the parity of the outcome.

Example 2

- There are 3 possible outcomes for each odd or even output of a die
- We need b_2 bits to represent it:

$$b_2 = \lceil \log_2 3 \rceil = \lceil 1.58 \rceil = 2$$

- This measures the amount of confusion remaining about the outcome and it is the amount of information that we still need to communicate, given the already released information (parity)

$$b_2 = b - b_1 = \log_2 6 - \log_2 2 = 2.58 - 1 = 1.58$$

- The gained information by the first message is:
  Gained information=initial amount of confusion – remaining amount of confusion

$$b_1 = b - b_2 = \log_2 6 - \log_2 3 = 2.58 - 1.58 = 1$$

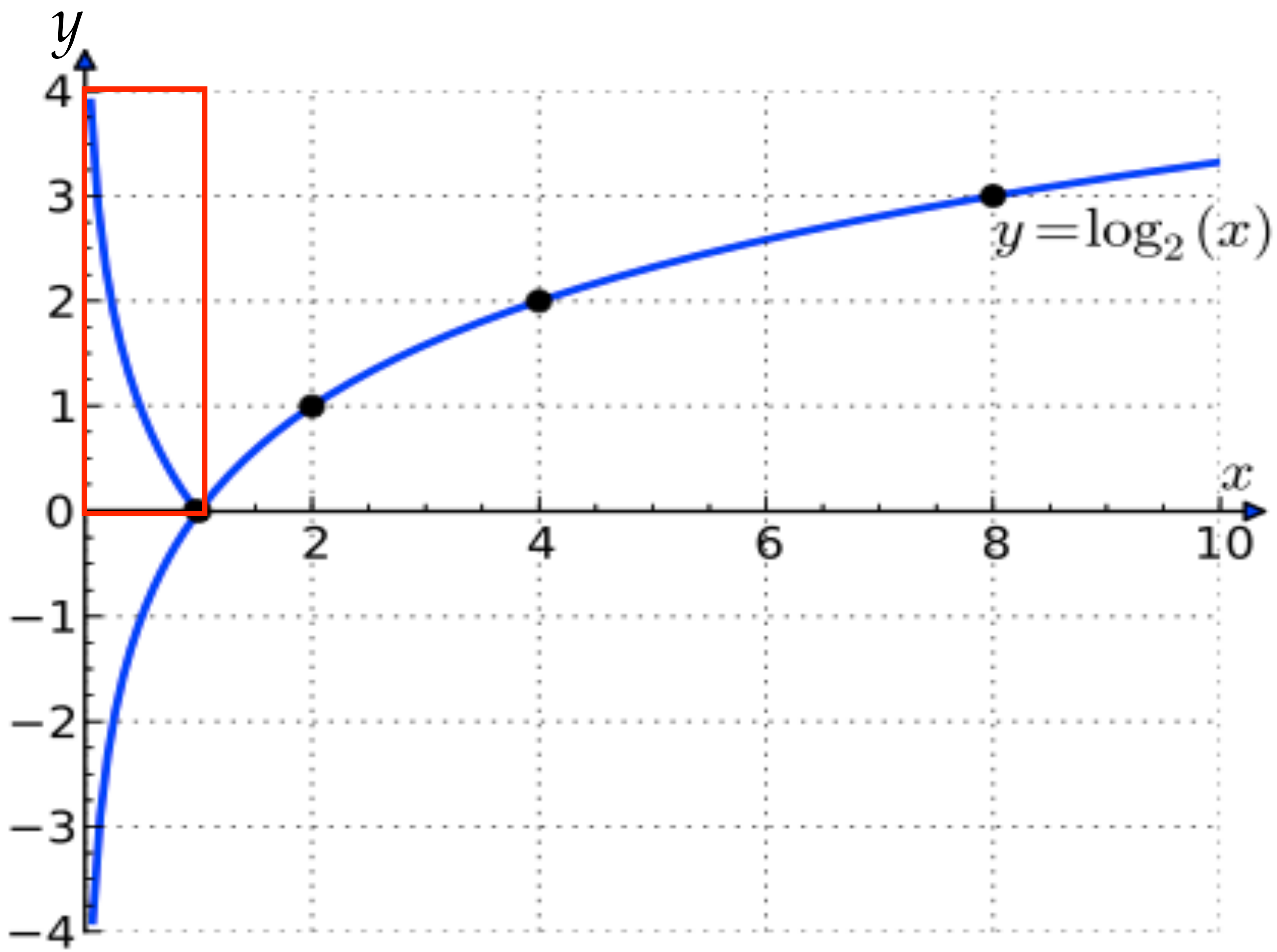
Information (1)

- If an event E has probability p to occur, the information gained when we observe it is:

$$I(E) = \log_2(\frac{1}{p}) = -\log_2 p$$

- In an event E is highly probable (p~1) the amount of information that we gain from the observation of E is low
- If an event E is not probable (p~0) the amount of information that we gain from E is high

Information plot ($0 \leq x \leq 1$)

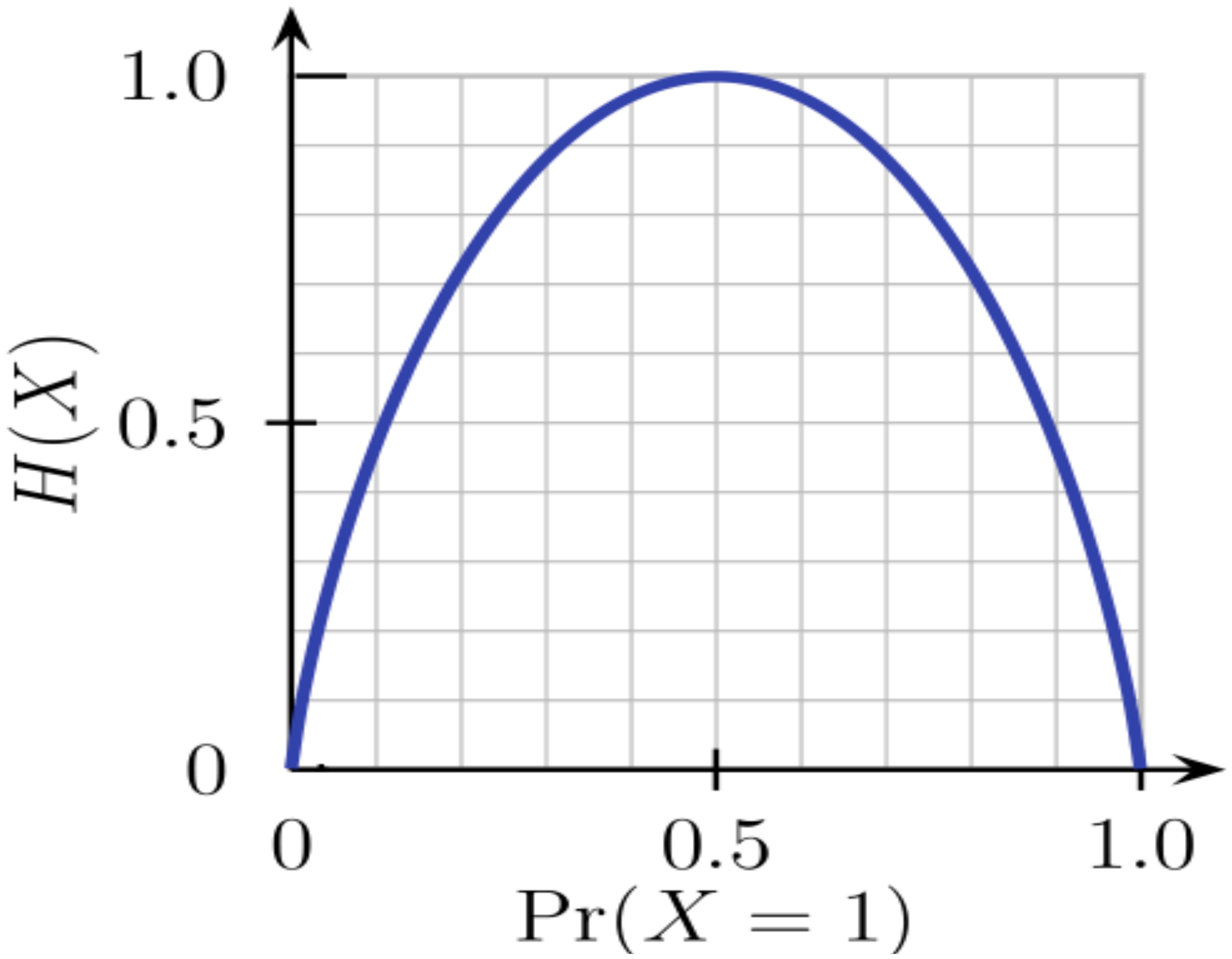


Information (2)

- If an experiment has n outcomes (E_1, .. E_n), each with probability $p_1, p_2, \ldots, p_n$ then the average amount of information we gain when we observe a generic outcome of the experiment is the *expected information* or entropy:

$$H(E) = \sum_{i=1}^{n} p_i \cdot \log_2(\frac{1}{p_i}) = -\sum_{i=1}^{n} p_i \cdot \log_2(p_i)$$

Entropy *H(X)* of a binary output *X=1/X=0*



Example 4.4, p.115

The dataset

Suppose we have the following five positive examples (the first three are the same as in Example 4.1):

p1: $\text{Length}=3 \land \text{Gills}=\text{no} \land \text{Beak}=\text{yes} \land \text{Teeth}=\text{many}$
p2: $\text{Length}=4 \land \text{Gills}=\text{no} \land \text{Beak}=\text{yes} \land \text{Teeth}=\text{many}$
p3: $\text{Length}=3 \land \text{Gills}=\text{no} \land \text{Beak}=\text{yes} \land \text{Teeth}=\text{few}$
p4: $\text{Length}=5 \land \text{Gills}=\text{no} \land \text{Beak}=\text{yes} \land \text{Teeth}=\text{many}$
p5: $\text{Length}=5 \land \text{Gills}=\text{no} \land \text{Beak}=\text{yes} \land \text{Teeth}=\text{few}$

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

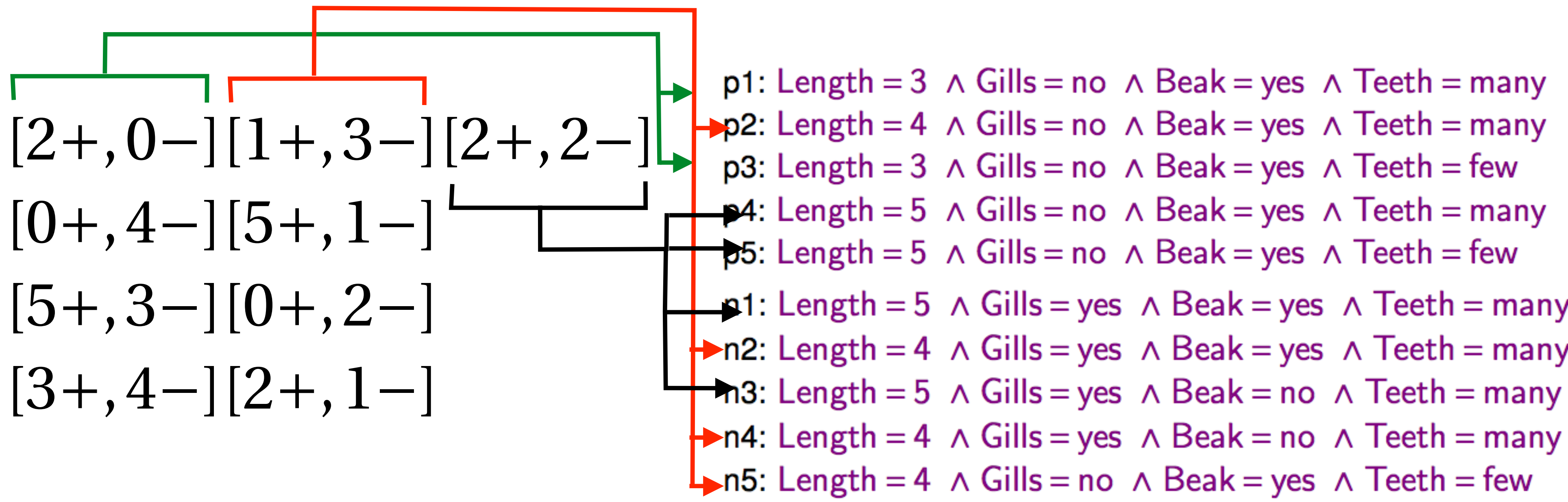
n1: $\text{Length}=5 \land \text{Gills}=\text{yes} \land \text{Beak}=\text{yes} \land \text{Teeth}=\text{many}$
n2: $\text{Length}=4 \land \text{Gills}=\text{yes} \land \text{Beak}=\text{yes} \land \text{Teeth}=\text{many}$
n3: $\text{Length}=5 \land \text{Gills}=\text{yes} \land \text{Beak}=\text{no} \land \text{Teeth}=\text{many}$
n4: $\text{Length}=4 \land \text{Gills}=\text{yes} \land \text{Beak}=\text{no} \land \text{Teeth}=\text{many}$
n5: $\text{Length}=4 \land \text{Gills}=\text{no} \land \text{Beak}=\text{yes} \land \text{Teeth}=\text{few}$

Example 5.1, p.135

Calculating impurity I

Consider again the data in Example 4.4. We want to find the best feature to put at the root of the decision tree. The four features available result in the following splits:

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



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

Example 5.1, p.135

Calculating impurity using entropy (1)

Let us calculate the expected impurity obtained after the split on Length.
We will consider first entropy and then Gini index and we will compare. them.

Length = [3,4,5] $[2+,0-][1+,3-][2+,2-]$

$= -0.25\frac{log_{10}(0.25)}{log_{10}(2)} = 0.5$

[2+,0-] is pure ➞ entropy([2+,0-])=0

[1+,3-] ➞ entropy([1+,3-])= $-(1/4)\log_2(1/4) - (3/4)\log_2(3/4) = 0.5 + 0.31 = 0.81$

[2+,2-] ➞ maximum impurity = entropy([2+,2-])=1

$= -0.75\frac{log_{10}(0.75)}{log_{10}(2)} = 0.31$

The expected entropy is: $2/10 \cdot 0 + 4/10 \cdot 0.81 + 4/10 \cdot 1 = \boxed{0.72}$

Example 5.1, p.135

Calculating impurity using entropy (2)

Similar calculations for the other three features give the following entropies:

Gills $4/10 \cdot 0 + 6/10 \cdot \big(-(5/6)\log_2(5/6) - (1/6)\log_2(1/6)\big) = \boxed{0.39;}$

Beak $8/10 \cdot \big(-(5/8)\log_2(5/8) - (3/8)\log_2(3/8)\big) + 2/10 \cdot 0 = 0.76;$

Teeth $7/10 \cdot \big(-(3/7)\log_2(3/7) - (4/7)\log_2(4/7)\big)$
$+3/10 \cdot \big(-(2/3)\log_2(2/3) - (1/3)\log_2(1/3)\big) = 0.97.$

We thus clearly see that ‘Gills’ is an excellent feature to split on; ‘Teeth’ is poor; and the other two are somewhere in between.

Calculation of the index index

- The Gini Index is on the scale from (0-0.5)

Length = [3,4,5] $[2+,0-][1+,3-][2+,2-]$

The expected gini index is:

$$\overbrace{\underset{\frac{|D_1|}{|D|}}{2/10}\cdot 2\cdot \underset{\dot{p}_1\cdot(1-\dot{p}_1)}{(2/2\cdot 0/2)}}^{[2+,0-]} + \overbrace{\underset{\frac{|D_2|}{|D|}}{4/10}\cdot 2\cdot \underset{\dot{p}_2\cdot(1-\dot{p}_2)}{(1/4\cdot 3/4)}}^{[1+,3-]} + \overbrace{\underset{\frac{|D_3|}{|D|}}{4/10}\cdot 2\cdot \underset{\dot{p}_3\cdot(1-\dot{p}_3)}{(2/4\cdot 2/4)}}^{[2+,2-]} = 0.35$$

- As regards the remaining splits:

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

The expected gini index values are:

Gills	$4/10\cdot 0+6/10\cdot 2\cdot(5/6\cdot 1/6)=\boxed{0.17;}$
Beak	$8/10\cdot 2\cdot(5/8\cdot 3/8)+2/10\cdot 0=0.38;$
Teeth	$7/10\cdot 2\cdot(3/7\cdot 4/7)+3/10\cdot 2\cdot(2/3\cdot 1/3)=0.48.$

Conclusion:
The best gini index (lowest) is for Gills.
The worst is Teeth (almost the maximum).
Results are in agreement with entropy.

Algorithm 5.2, p.137

Finding the best split for a decision tree

The best split will minimise the impurity of the subsets $D_1,..,D_l$

Algorithm BestSplit-Class(D, F) – find the best split for a decision tree.

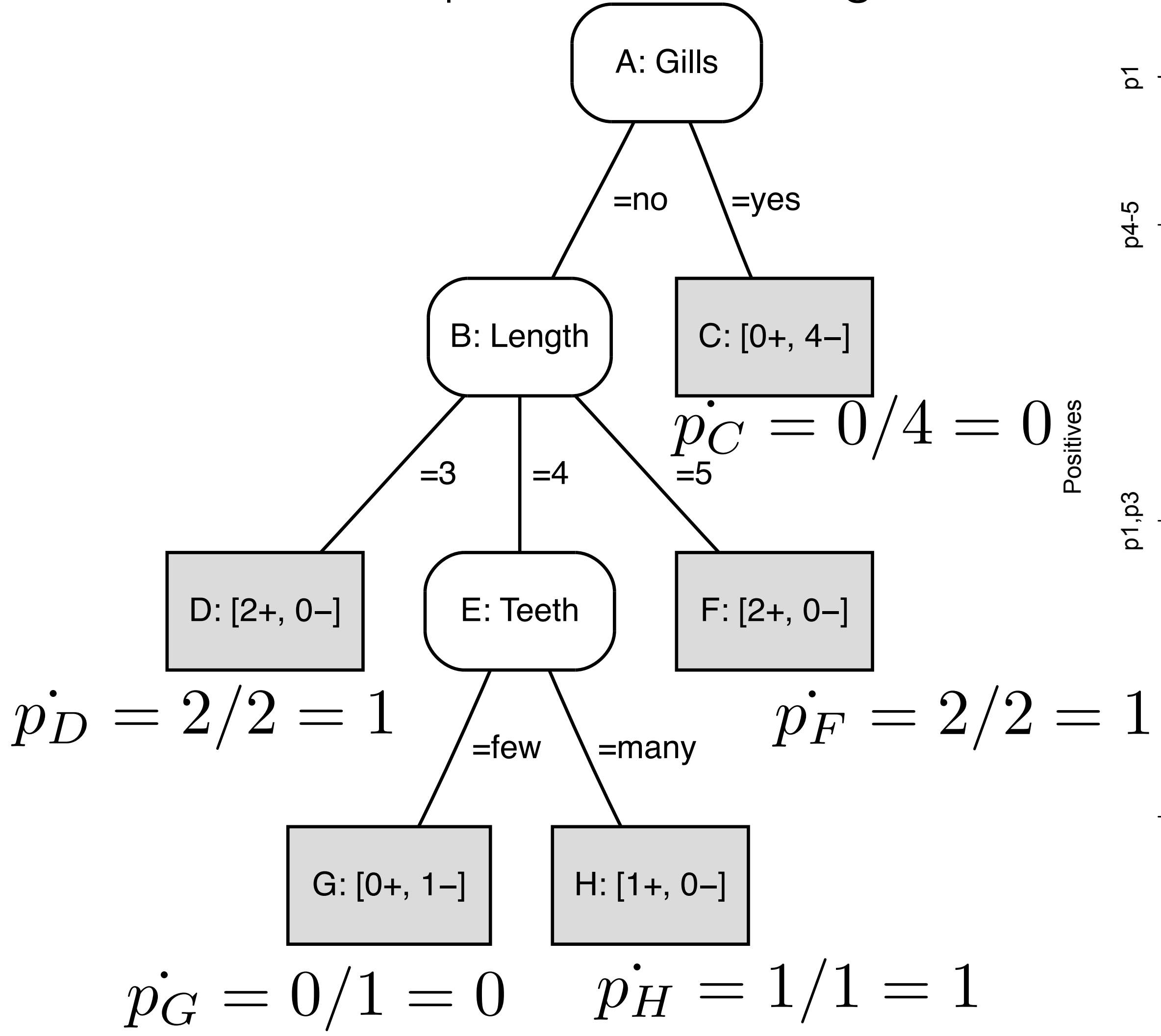
```
Input  : data $D$; set of features $F$.
Output : feature $f$ to split on.
1 $I_{\min} \leftarrow 1$;
2 for each $f \in F$ do
3     split $D$ into subsets $D_1,\ldots,D_l$ according to the values $v_j$ of $f$;
4     if Imp($\{D_1,\ldots,D_l\}$) $< I_{\min}$ then
5         $I_{\min} \leftarrow$ Imp($\{D_1,\ldots,D_l\}$);
6         $f_{\text{best}} \leftarrow f$;
7     end
8 end
9 return $f_{\text{best}}$
```

24=3 (Length values) *2 (Gills)*2 (Beak) *2 (Teeth) possible instances in instance space

Figure 5.3, p.137

Decision tree for dolphins

This tree assigns a class also to 14 instances of the search space that were not part of the training set (10). This is why it generalizes! The negative class has been assigned to most of them because of the leaf C, the closest to the root. It corresponds to a strong observed regularity (4/5 of negative examples have gills).



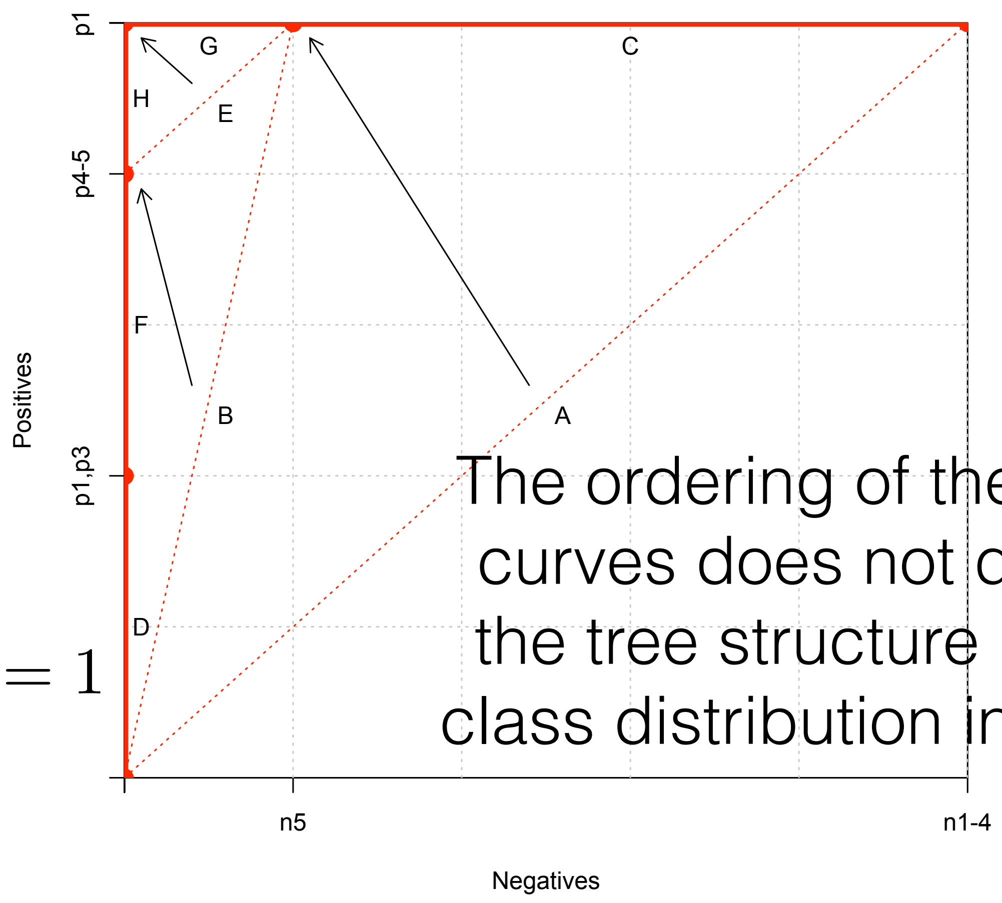
$\dot{p_C} = 0/4 = 0$

$\dot{p_D} = 2/2 = 1$

$\dot{p_F} = 2/2 = 1$

$\dot{p_G} = 0/1 = 0$

$\dot{p_H} = 1/1 = 1$



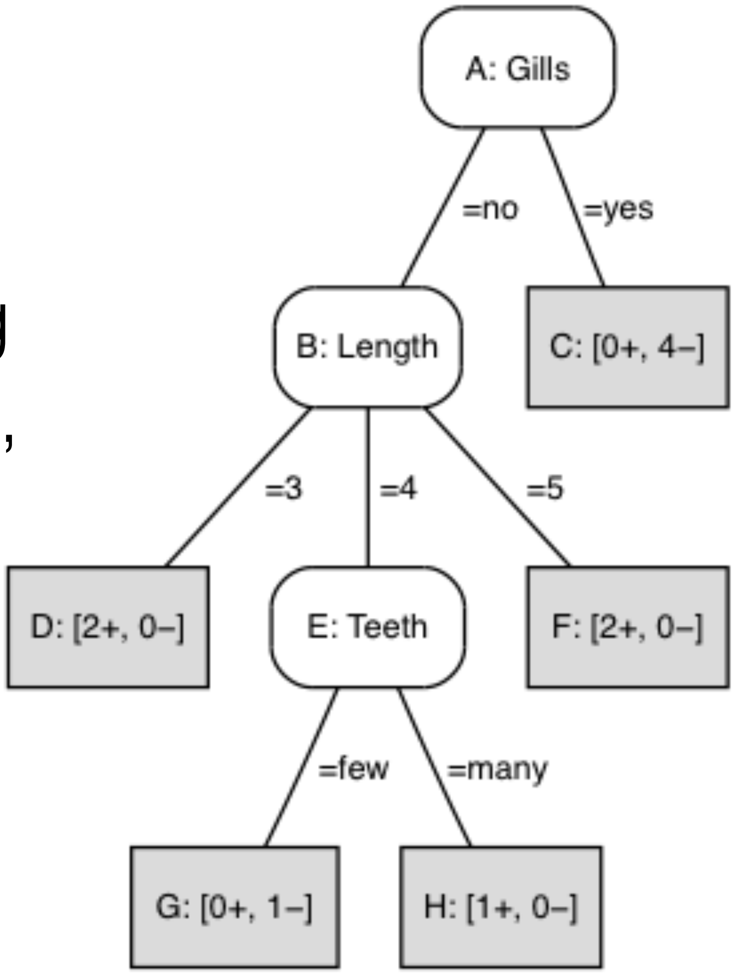
The ordering of the coverage curves does not depend on the tree structure but on the class distribution in the leaves

(left) Decision tree learned from the data in Example 4.4. **(right)** Each internal and leaf node of the tree corresponds to a line segment in coverage space: vertical segments for pure positive nodes, horizontal segments for pure negative nodes, and diagonal segments for impure nodes.

What's next?

Ranking trees

- Tree models divide the instance space into segments.
- They can be turned into rankers by learning an ordering on those segments.
- Tree models have access to the local distribution of classes in leaves (given by the empirical probabilities) which can be used to construct a leaf ordering which is optimal for the training set
- The optimal ordering for the model tree is: [D-F],H,G,C obtained ordering the leaves in a non-increasing order by the empirical probabilities in the leaves, and breaking ties by giving precedence to leaves covering a larger number of positives



Important point to remember

The ranking obtained from the empirical probabilities in the leaves of a decision tree yields a convex ROC curve on the training data.

- The slope of the coverage curve segment with empirical probability p is p/(1-p). It is a monotonic transformation: p↦p/(1-p): if p'>p then p'/(1-p')>p/(1-p).
  Thus the ordering on non-increasing p will produce a coverage curve made by segments with non-increasing slope which means that the curve is convex.

Example 5.2, p.139

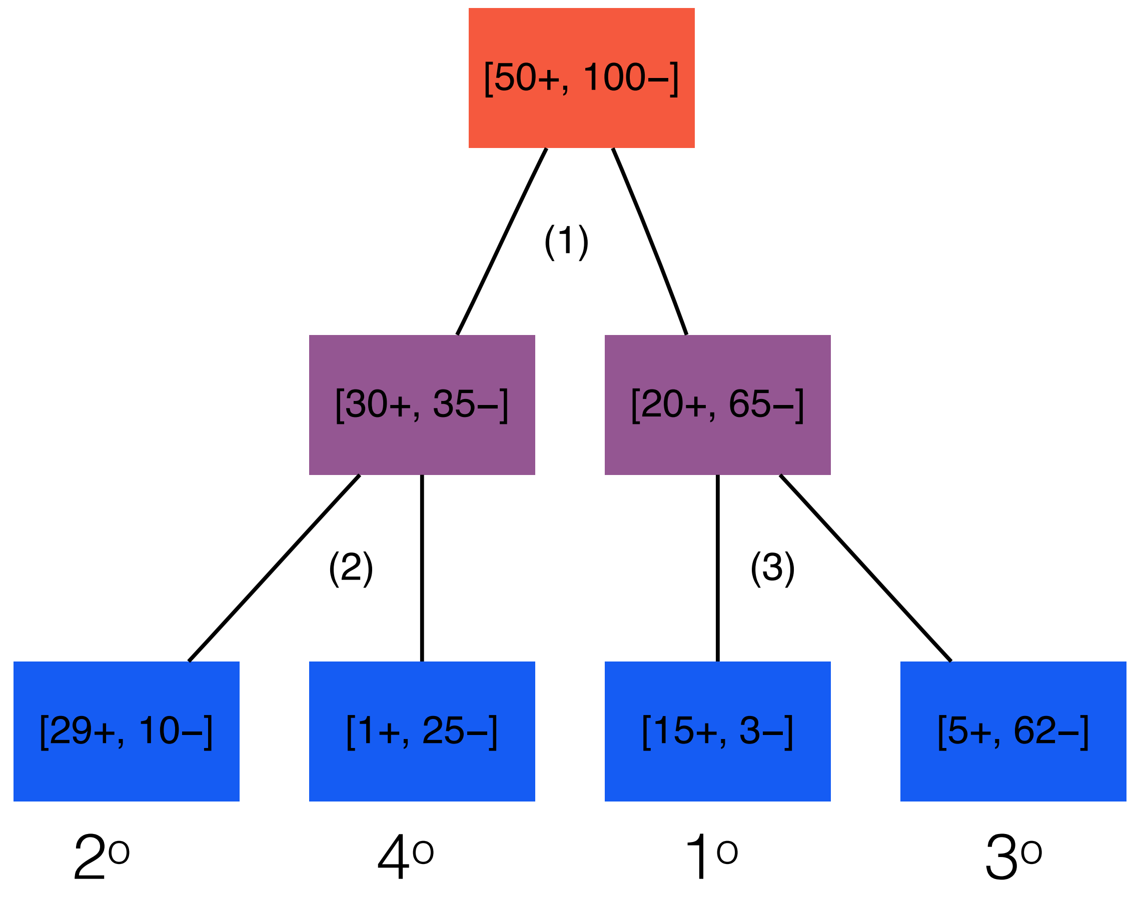
Growing a tree

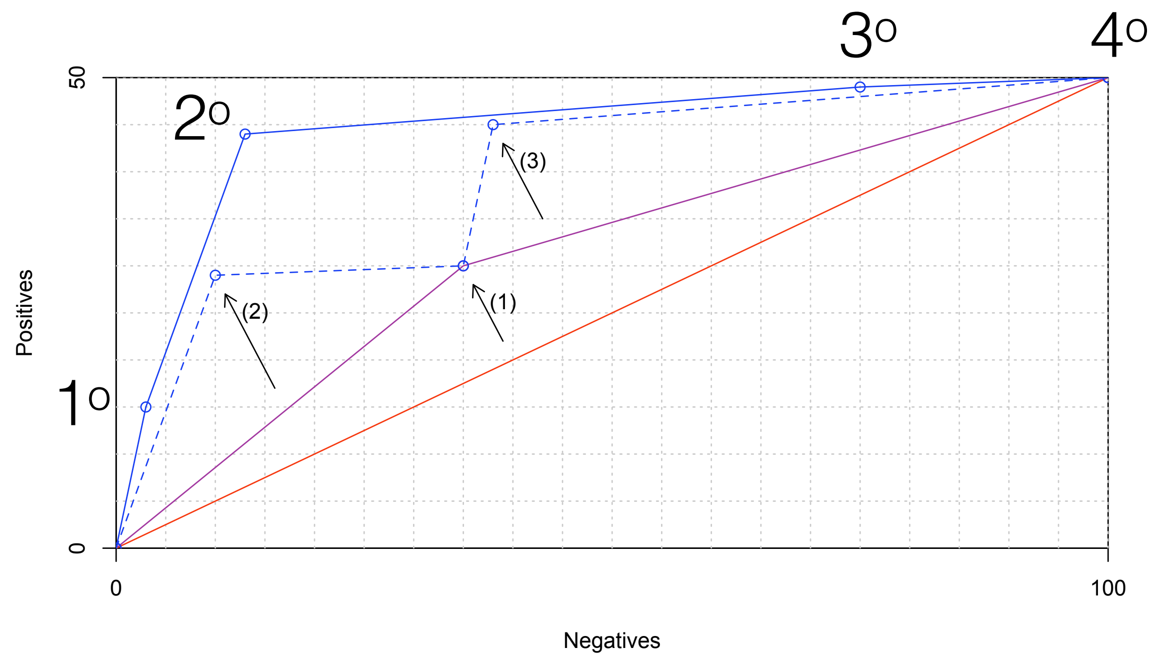
Consider the tree in Figure 5.4 (left). Each node is labelled with the numbers of positive and negative examples covered by it: so, for instance, the root of the tree is labelled with the overall class distribution (50 positives and 100 negatives), resulting in the trivial ranking $[50+,100-]$. The corresponding one-segment coverage curve is the ascending diagonal (Figure 5.4 (right)).

- Adding split (1) refines this ranking into $[30+,35-][20+,65-]$, resulting in a two-segment curve.
- Adding splits (2) and (3) again breaks up the segment corresponding to the parent into two segments corresponding to the children.
- However, the ranking produced by the full tree – $\dot{p}_2 = 29/39 = 0.74$ $\dot{p}_4 = 1/26 = 0.0038$ $[15+,3-][29+,10-][5+,62-][1+,25-]$ – is different from the left-to-right $\dot{p}_1 = 15/18 = 0.83$ $\dot{p}_3 = 5/67 = 0.075$ ordering of its leaves, hence we need to reorder the segments of the coverage curve, leading to the top-most, solid curve. This reordering always leads to a convex coverage curve

Figure 5.4, p.140

Growing a tree

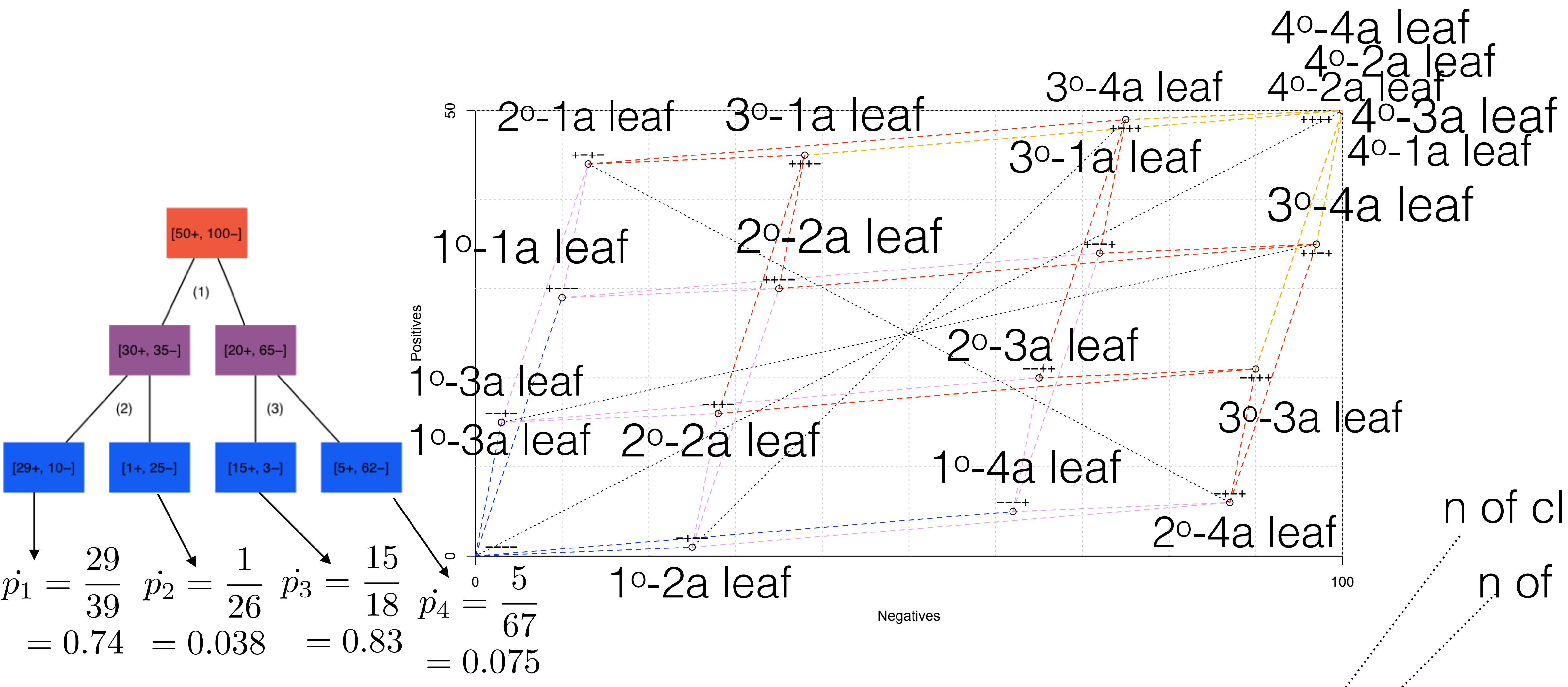




(left) Abstract representation of a tree with numbers of positive and negative examples covered in each node. Binary splits are added to the tree in the order indicated. **(right)** Adding a split to the tree will add new segments to the coverage curve as indicated by the arrows. After a split is added the segments may need reordering, and so only the solid lines represent actual coverage curves.

Figure 5.5, p.141

Labelling a tree



$\dot{p}_1 = \frac{29}{39} = 0.74$ $\dot{p}_2 = \frac{1}{26} = 0.038$ $\dot{p}_3 = \frac{15}{18} = 0.83$ $\dot{p}_4 = \frac{5}{67} = 0.075$

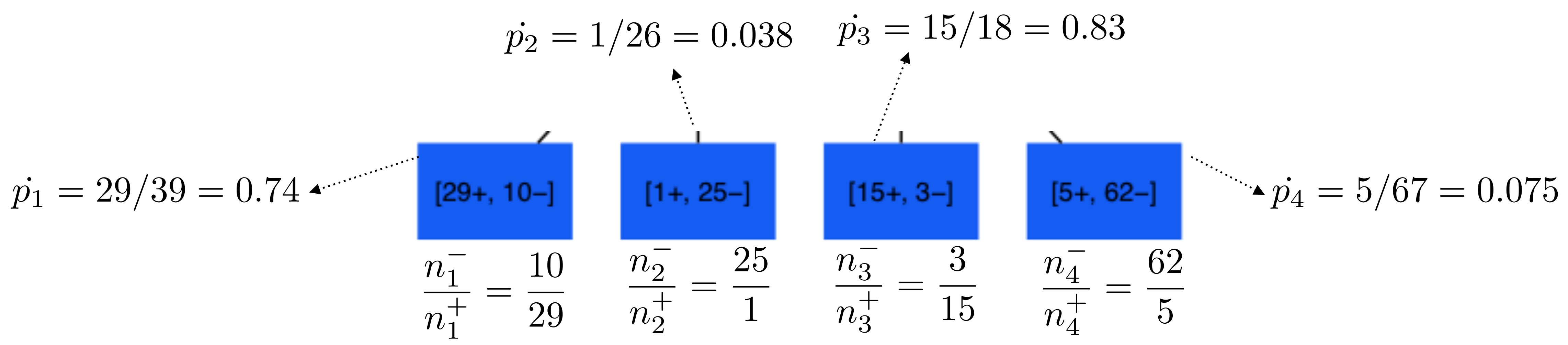
Graphical depiction of all possible labellings and all possible rankings that can be obtained with the four-leaf decision tree in Figure 5.4. There are $2^4 = 16$ possible leaf labellings; e.g., '$+-+-$' denotes labelling the first and third leaf from the left as + and the second and fourth leaf as $-$. There are $4! = 24$ possible blue-violet-red-orange paths through these points which start in $----$ and switch each leaf to + in some order; these represent all possible four-segment coverage curves or rankings.

Choosing a labelling based on costs

Assume the training set class ratio $clr = 50/100$ is representative. We have a choice of five labellings, depending on the expected cost ratio $c = c_{\mathrm{FN}}/c_{\mathrm{FP}}$ of misclassifying a positive in proportion to the cost of misclassifying a negative:

$+-+-$	would be the labelling of choice if $c = 1$, or more generally if $10/29 < c < 62/5$;
$+-++$	would be chosen if $62/5 < c < 25/1$;
$++++$	would be chosen if $25/1 < c$; i.e., we would always predict positive if false negatives are more than 25 times as costly as false positives, because then even predicting positive in the second leaf would reduce cost;
$--+-$	would be chosen if $3/15 < c < 10/29$;
$----$	would be chosen if $c < 3/15$; i.e., we would always predict negative if false positives are more than 5 times as costly as false negatives, because then even predicting negative in the third leaf would reduce cost.

Explanation of the cost limits



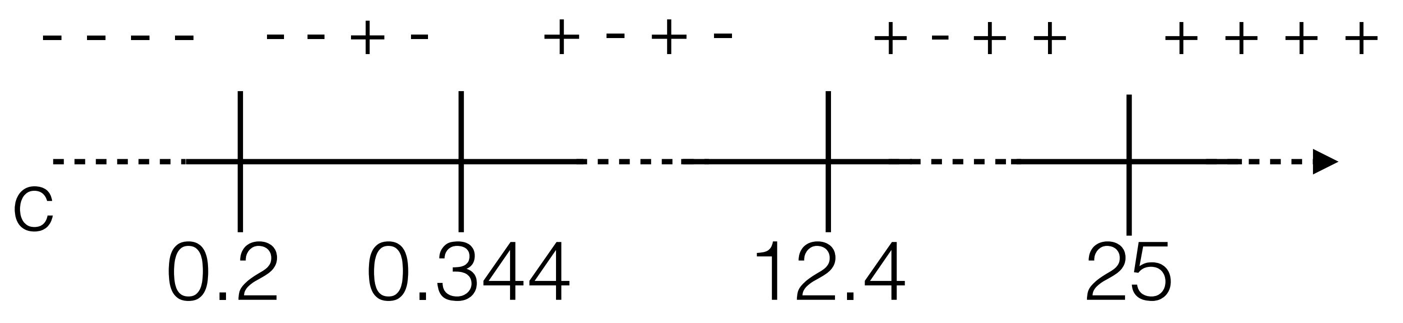
$\dot{p}_2 = 1/26 = 0.038$
$\dot{p}_3 = 15/18 = 0.83$
$\dot{p}_1 = 29/39 = 0.74$
[29+, 10–]
[1+, 25–]
[15+, 3–]
[5+, 62–]
$\dot{p}_4 = 5/67 = 0.075$
$\frac{n_1^-}{n_1^+} = \frac{10}{29}$
$\frac{n_2^-}{n_2^+} = \frac{25}{1}$
$\frac{n_3^-}{n_3^+} = \frac{3}{15}$
$\frac{n_4^-}{n_4^+} = \frac{62}{5}$

cost of misclassifying positives

expected cost ratio: $c=C_{FN}/C_{FP}$

cost of misclassifying negatives

- - - - - with c<3/15=0.2
- - - + - with 3/15=0.2 ≤ c < 10/29=0.344
- + - + - with 10/29= 0.344 ≤ c < 62/5=12.4
- + - + + with 62/5=12.4 ≤ c < 25/1=25
- + + + + with c≥25



Misclassifying costs

	estimated class: positive	estimated class: negative
real class: positive	TP	FN c_{FN}
real class: negative	FP c_{FP}	TN

Deploying a feature tree

- Exploiting the class distribution in the leaves we can turn a feature tree into:

1. a **ranking tree**, if we order the leaves on non-increasing empirical probabilities (which is optimal on the training set)

2. a **probability estimator**, in which we predict the empirical probabilities in each leaf, applying the Laplace or m-estimate smoothing (to make the estimate more robust in small leaves) $\dot{p} = \frac{n^{+}+1}{n_{tot}+n_c}$ $\dot{p_i} = \frac{n_i + m \cdot \pi_i}{n_{tot}+m}$

3. a **classifier**, in which we choose the operating conditions as a consequence of the proportion of the frequencies of the classes:

$$clr = \frac{Pos}{Neg}$$

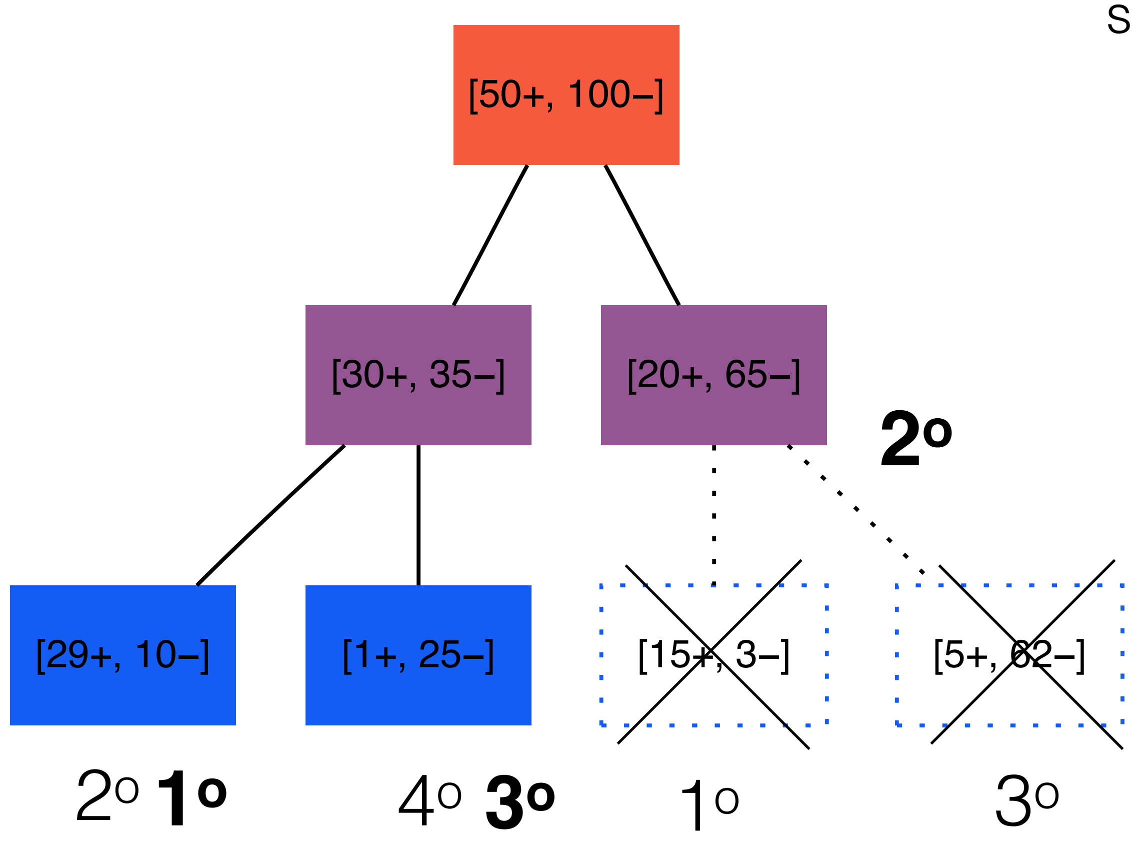
and of the cost ratio (ratio of the misclassification costs):

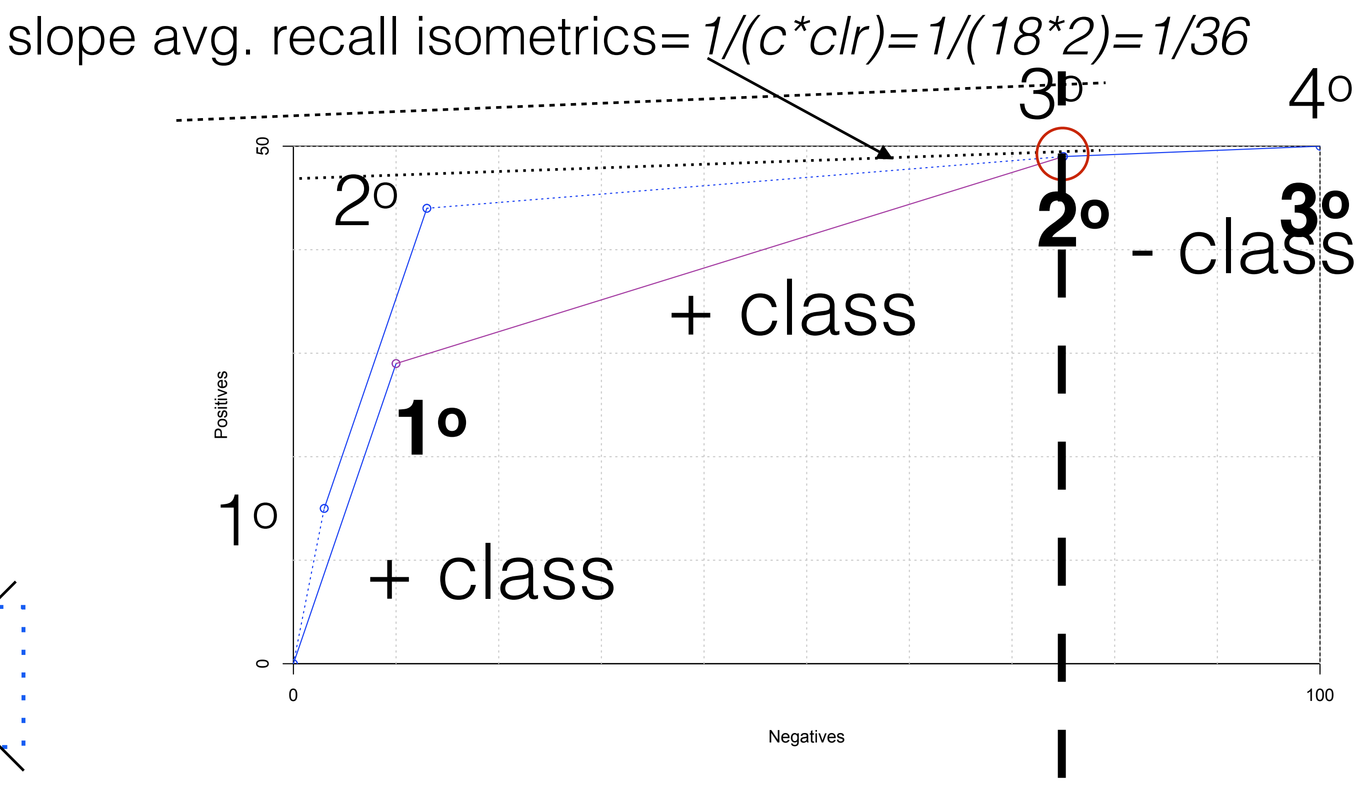
$$c = \frac{c_{FN}}{c_{FP}}$$

and find the optimal operating point (decision threshold) on the ROC curve at the intersection with the isometric curve (points of equal accuracy) with a slope of $\frac{1}{c \cdot clr}$
As a result, the leaves before the operating point will predict positive; negative the remaining ones.

Figure 5.6, p.143

Pruning a tree





(left) To achieve the labelling + – ++ we don't need the right-most split, which can therefore be pruned away. **(right)** Pruning doesn't affect the chosen operating point, but it does decrease the ranking performance of the tree.

Advantage of pruning: it simplifies the model (e.g. for communication to someone)
However, unless the tree is used for classification purposes only, the best labelling at leaf nodes should be chosen according to the operating conditions (classes prevalence and misclassification costs).

Deciding on the class according to *c*

[29+, 10–]	[1+, 25–]	[15+, 3–]	[5+, 62–]
$\frac{n_1^-}{n_1^+} = \frac{10}{29}$	$\frac{n_2^-}{n_2^+} = \frac{25}{1}$	$\frac{n_3^-}{n_3^+} = \frac{3}{15}$	$\frac{n_4^-}{n_4^+} = \frac{62}{5}$

- At leaf *i* we multiply the ratio $\frac{n_i^-}{n_i^+}$ by the factor $\frac{1}{c} = \frac{c_{FP}}{c_{FN}}$ obtaining:

$$\frac{n_i^-}{n_i^+} \cdot \frac{c_{FP}}{c_{FN}}$$

- In this way, we take into account the different misclassification costs as follows:
  in leaf node *i*, each positive example, if turned into an error, was considered equivalent to observing a number of positive examples equal to the cost of misclassifying a positive *c_{FN};*
  each error on the negatives as if it was equivalent to observing a number of negatives equal to *c_{FP}*

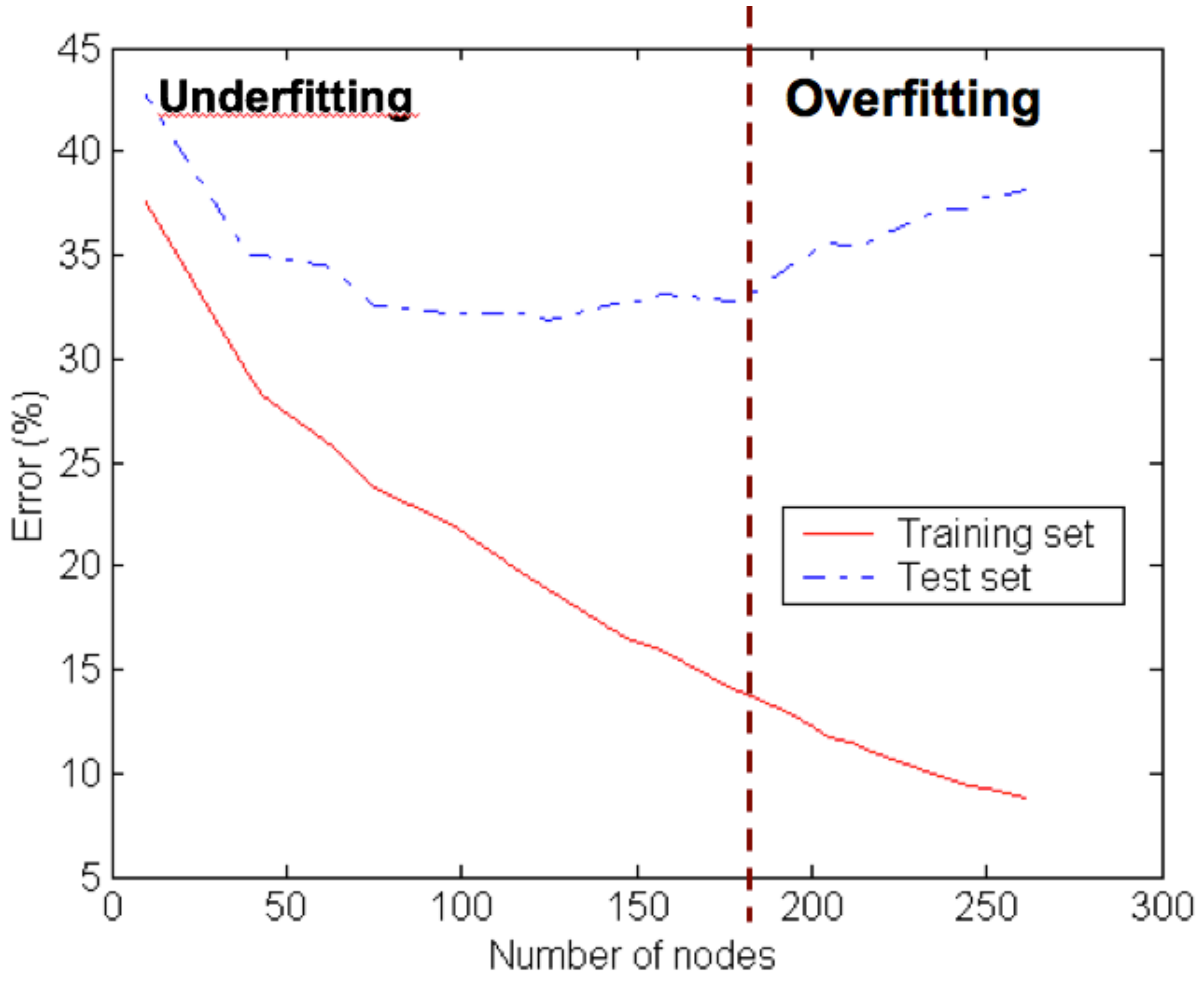
- On the result $\frac{n_i^-}{n_i^+} \cdot \frac{c_{FP}}{c_{FN}}$ we follow the majority class decision:
  - If $\frac{n_i^-}{n_i^+} \cdot \frac{c_{FP}}{c_{FN}} > 1$, we predict the class on the numerator (negative) because the misclassification cost would be higher than the opposite class;
    otherwise we predict the positive class.

Exercise

- We assume $c=1.20$ and $clr=15$
- Which will be the class predicted by each leaf node?

[29+, 10–]	[1+, 25–]	[15+, 3–]	[5+, 62–]
$\frac{n_1^-}{n_1^+} = \frac{10}{29}$	$\frac{n_2^-}{n_2^+} = \frac{25}{1}$	$\frac{n_3^-}{n_3^+} = \frac{3}{15}$	$\frac{n_4^-}{n_4^+} = \frac{62}{5}$

Underfitting And Overfitting



Algorithm 5.3, p.144

Reduced-error pruning

To reduce the chances of overfitting a pruned tree is suggested

Algorithm PruneTree(T, D) – reduced-error pruning of a decision tree.

```
Input  : decision tree $T$; labelled data $D$.
Output : pruned tree $T'$.
1 for every internal node $N$ of $T$, starting from the bottom do
2     $T_N$ ←subtree of $T$ rooted at $N$;
3     $D_N \leftarrow \{x \in D | x$ is covered by $N\}$;
4     if accuracy of $T_N$ over $D_N$ is worse than majority class in $D_N$ then
5         replace $T_N$ in $T$ by a leaf labelled with the majority class in $D_N$;
6     end
7 end
8 return pruned version of $T$
```

This algorithm employs a separate data-set (pruning-set) to estimate the accuracy of the pruned tree

Estimating Generalization Errors

- An alternative to the technique of reduced error pruning on the pruning set is the estimation directly on the training set of the generalization error (error on the test set) during the construction of the tree.
- We add a penalty *k* for any leaf node. The penalty does not allow the creation of the leaf if it does not decrease the error of the parent node of at least *k+1.*
- Re-substitution errors: error on training: $\sum_{i=1}^{N} e_i$
- Generalization errors: error on testing: $\sum_{i=1}^{N} e'_i$
- Without a test-set, for the estimation of the generalization error on the training set, we suppose:
  for each leaf node *i*: $e_i'(t) = (e_i(t)+0.5)$

  0.5 is a penalty given to each leaf: The creation of a leaf is allowed as soon as it improves the classification of the parent node of at least a number of instances equal to $\lceil 0.5 \rceil = 1$
- Total errors: $e'(T) = e(T) + N * 0.5$ (*N*: number of leaf nodes)
- For a tree with 30 leaf nodes and 10 errors on training (out of 1000 instances):
  - Training error = 10/1000 = 1%
  - Generalization error = (10 + 30*0.5)/1000 = 2.5%

Example 5.3, p.144

Skew sensitivity of splitting criteria I

Suppose you have 10 positives and 10 negatives, and you need to choose between the two splits $[8+,2-][2+,8-]$ and $[10+,6-][0+,4-]$.

- ☞ You duly calculate the weighted average entropy of both splits and conclude that the first split is the better one.
- ☞ Just to be sure, you also calculate the average Gini index, and again the first split wins.
- ☞ You then remember somebody telling you that the square root of the Gini index was a better impurity measure, so you decide to check that one out as well. Lo and behold, it favours the second split...! What to do?

This is because entropy and Gini index are sensitive to changes in the class frequency distribution.

Example 5.3, p.144

Skew sensitivity of splitting criteria II

You then remember that mistakes on the positives are about ten times as costly as mistakes on the negatives. *c=10*

- ☞ You're not quite sure how to work out the maths, and so you decide to simply have ten copies of every positive: the splits are now $[80+,2-][20+,8-]$ and $[100+,6-][0+,4-]$.
- ☞ You recalculate the three splitting criteria and now all three favour the second split.
- ☞ Even though you're slightly bemused by all this, you settle for the second split since all three splitting criteria are now unanimous in their recommendation.

Inflating artificially in the training set a certain number (equal to the cost ratio *c*) of examples of the most costly class is an alternative to using the cost ratio.

Important point to remember

Entropy and Gini index are sensitive to fluctuations in the class distribution, $\sqrt{\text{Gini}}$ isn't.

Inflation of Examples in the Training set

With the example 5.3 (p. 144) we have seen that:

- Inflating artificially in the training set a certain number (equal to the cost ratio $c=c_{FN}/c_{FP}$) of examples of the most costly class is an alternative to using the cost ratio to determine the operative condition.
- Similarly, if we want to mimic an unbalanced distribution with the class ratio equal to $clr=Pos/Neg$ in the training set, we oversample the positive class with a factor clr if $clr>1$ or oversample the negative class with a factor equal to $1/clr$ if $clr<1$.
- As a negative effect, this oversampling will increase the training times.

Peter's recipe for decision tree learning

- ☞ First and foremost, I would concentrate on getting good ranking behaviour, because from a good ranker I can get good classification and probability estimation, but not necessarily the other way round.
- ☞ I would therefore try to use an impurity measure that is distribution-insensitive, such as $\sqrt{\text{Gini}}$; if that isn't available and I can't hack the code, I would resort to oversampling the minority class to achieve a balanced class distribution.
- ☞ I would disable pruning and smooth the probability estimates by means of the Laplace correction (or the m-estimate).
  in order to make the estimate more robust in little leaves
- ☞ Once I know the deployment operation conditions, I would use these to select the best operating point on the ROC curve (i.e., a threshold on the predicted probabilities, or a labelling of the tree).
- ☞ (optional) Finally, I would prune away any subtree whose leaves all have the same label.

$$\dot{p}_i^{\,+} = \frac{n_i^+ + 1}{n_i + 2} \qquad \dot{p}_i^{\,+} = \frac{n_i^+ + m \cdot \pi^+}{n_i + m}$$

i indicates the *i*-th leaf,
π^+ the a priori probability of class +,
m is the number of classes

What's next?

Tree learning as variance reduction

- The variance of a Boolean (i.e., Bernoulli) variable with success probability $\dot{p}$ is $\dot{p}(1-\dot{p})$, which is half the Gini index. So we could interpret the goal of tree learning as minimising the class variance (or standard deviation, in case of $\sqrt{\text{Gini}}$) in the leaves.
- In regression problems we can define the variance in the usual way:

$$\text{Var}(Y) = \frac{1}{|Y|}\sum_{y\in Y}(y-\overline{y})^2$$

If a split partitions the set of target values Y into mutually exclusive sets $\{Y_1,\ldots,Y_l\}$, the weighted average variance is then

$$\text{Var}(\{Y_1,\ldots,Y_l\}) = \sum_{j=1}^{l}\frac{|Y_j|}{|Y|}\text{Var}(Y_j) = \ldots = \boxed{\frac{1}{|Y|}\sum_{y\in Y}y^2} - \boxed{\sum_{j=1}^{l}\frac{|Y_j|}{|Y|}\overline{y}_j^2}$$

The first term is constant for a given set Y and so we want to maximise the weighted average of squared means in the children.

(So that the partitions are well separate and distinguishable).

Regression trees

- In regression problems the target values of the set Y are continuous

- In Algorithm *BestSplit* we replace the measure *Imp* with the function *Var(Y):* therefore we identify the partitions of examples that reduce the average squared distance around the mean

$$\mathrm{Var}(Y) = \frac{1}{|Y|} \sum_{y \in Y} (y - \overline{y})^2$$

- Furthermore, in *Var(Y)* since the first term is fixed for a given parent node, $\frac{1}{|Y|} \sum_{y \in Y} y^2$ BestSplit maximises the second term:

$$\sum_{j=1}^{l} \frac{|Y_j|}{|Y|} \overline{y}_j^2$$

- Function *Label(Y)* returns the mean value in the set of values *Y* that fall in the leaf.

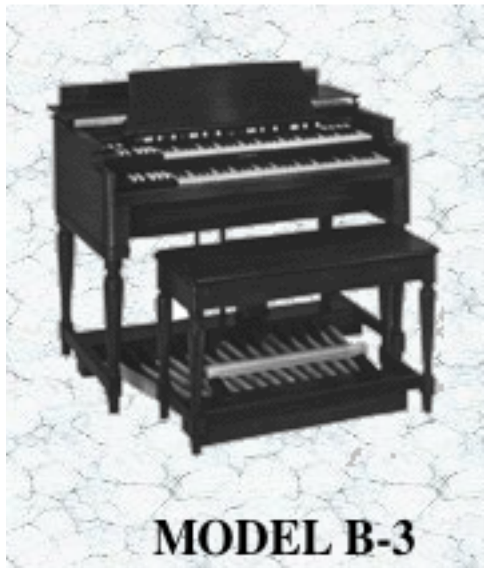
- Function *Homogeneous(Y)* returns true if the variance of the target values in Y is below a certain threshold

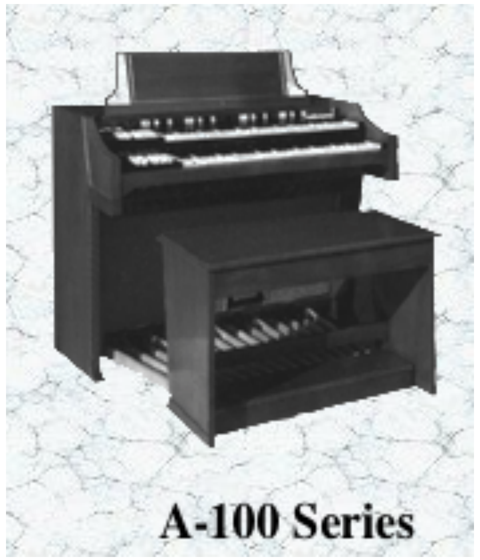
- Regression trees are susceptible to overfitting (due to the leaves with few examples)

Example 5.4, p.150

Learning a regression tree I

Imagine you are a collector of vintage Hammond tonewheel organs. You have been monitoring an online auction site, from which you collected some data about interesting transactions:





#	Model	Condition	Leslie	Price
1.	B3	excellent	no	4513
2.	T202	fair	yes	625
3.	A100	good	no	1051
4.	T202	good	no	270
5.	M102	good	yes	870
6.	A100	excellent	no	1770
7.	T202	fair	no	99
8.	A100	good	yes	1900
9.	E112	fair	no	77

Hammond organ with Leslie speaker

Example 5.4, p.150

Learning a regression tree II

From this data, you want to construct a regression tree that will help you determine a reasonable price for your next purchase. (The target is the attribute *price*). There are three features, hence three possible splits:

$\text{Model} = [\text{A100}, \text{B3}, \text{E112}, \text{M102}, \text{T202}]$

$[1051, 1770, 1900][4513][77][870][99, 270, 625]$

$\text{Condition} = [\text{excellent}, \text{good}, \text{fair}]$

$[1770, 4513][270, 870, 1051, 1900][77, 99, 625]$

$\text{Leslie} = [\text{yes}, \text{no}]$ $[625, 870, 1900][77, 99, 270, 1051, 1770, 4513]$

Let us determine the best feature to split on, based on variance of the target feature y (*price*) of examples in the partitions Y_j.
As said, we will maximise the objective function:

$$\sum_{j=1}^{l} \frac{|Y_j|}{|Y|} \overline{y}_j^2$$

Exercise solution on regression tree for auctions (1)

- **Partition on Model**

Model=A100	y=[1051,1770,1900]	$\overline{y} = \frac{1051+1770+1900}{3} = \frac{4721}{3} = 1574$
Model=B3	y=[4513]	$\overline{y} = 4513$
Model=E112	y=[77]	$\overline{y} = 77$
Model=M102	y=[870]	$\overline{y} = 870$
Model=T202	y=[99,270,625]	$\overline{y} = \frac{99+270+625}{3} = \frac{994}{3} = 331$

$$\sum_{j=1}^{l} \frac{|Y_j|}{|Y|} \overline{y}_j^2 = \frac{1}{3}(1574)^2 + \frac{1}{9}(4513)^2 + \frac{1}{9}(77)^2 + \frac{1}{9}(870)^2 + \frac{1}{3}(331)^2 = \boxed{3.21 \cdot 10^6}$$

Exercise solution on regression tree for auctions (2)

- **Partition on Condition**

Condition=excellent y=[1770,4513]

$$\overline{y} = \frac{1770 + 4513}{2} = \frac{6283}{2} = 3142$$

Condition=good y=[270,870,1051,1900]

$$\overline{y} = \frac{270 + 870 + 1051 + 1900}{4} = \frac{4091}{4} = 1023$$

Condition=fair y=[77,99,625]

$$\overline{y} = \frac{77 + 99 + 625}{3} = \frac{801}{3} = 267$$

$$\sum_{j=1}^{l} \frac{|Y_j|}{|Y|}\overline{y}_j^2 = \frac{2}{9}(3142)^2 + \frac{4}{9}(1023)^2 + \frac{3}{9}(267)^2 = \boxed{2.68 * 10^6}$$

Exercise solution on regression tree for auctions (3)

- **Partition on Leslie**

Leslie=yes   y=[625,870,1900]

$$\overline{y} = \frac{625+870+1900}{3} = \frac{3395}{3} = 1132$$

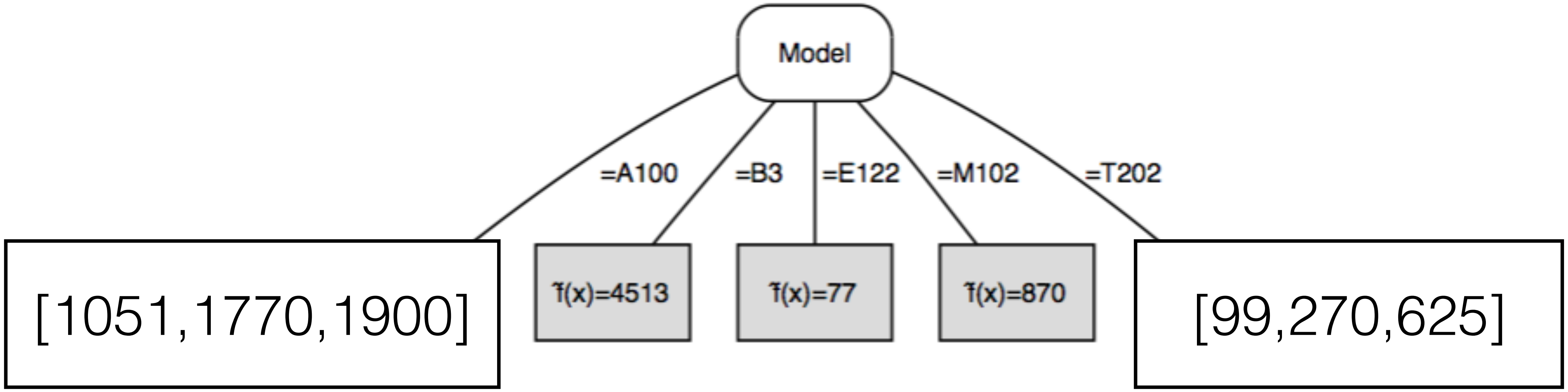
Leslie=no   y=[77,99,270,1051,1770,4513]

$$\overline{y} = \frac{77+99+270+1051+1770+4513}{6} = \frac{7780}{6} = 1297$$

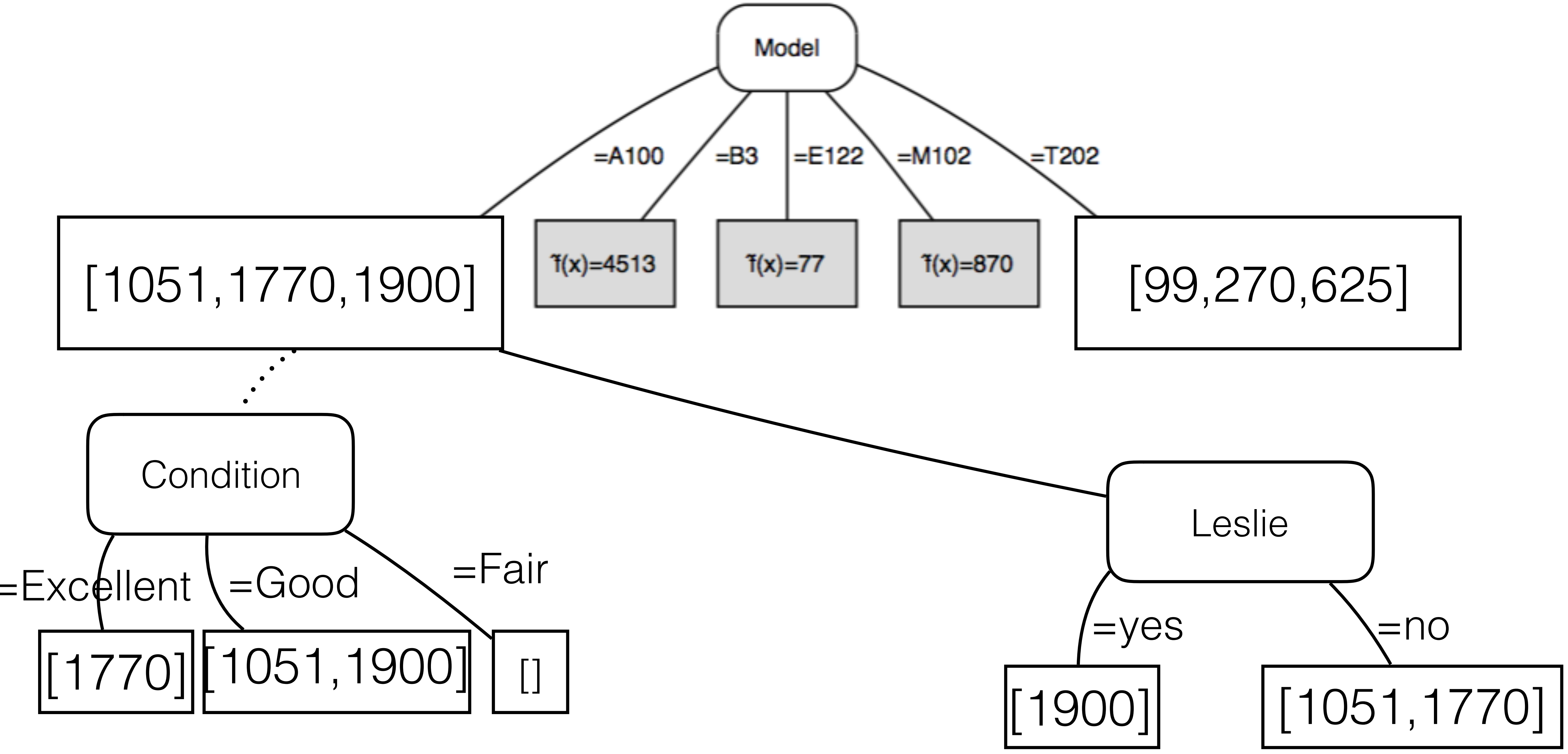
$$\sum_{j=1}^{l} \frac{|Y_j|}{|Y|} \overline{y}_j^2 = \frac{3}{9}(1132)^2 + \frac{6}{9}(1297)^2 = \boxed{1.55 * 10^6}$$

Conclusion: the highest is obtained with partition on Model ($3.21*10^6$)

Regression tree after first partition (on Model)



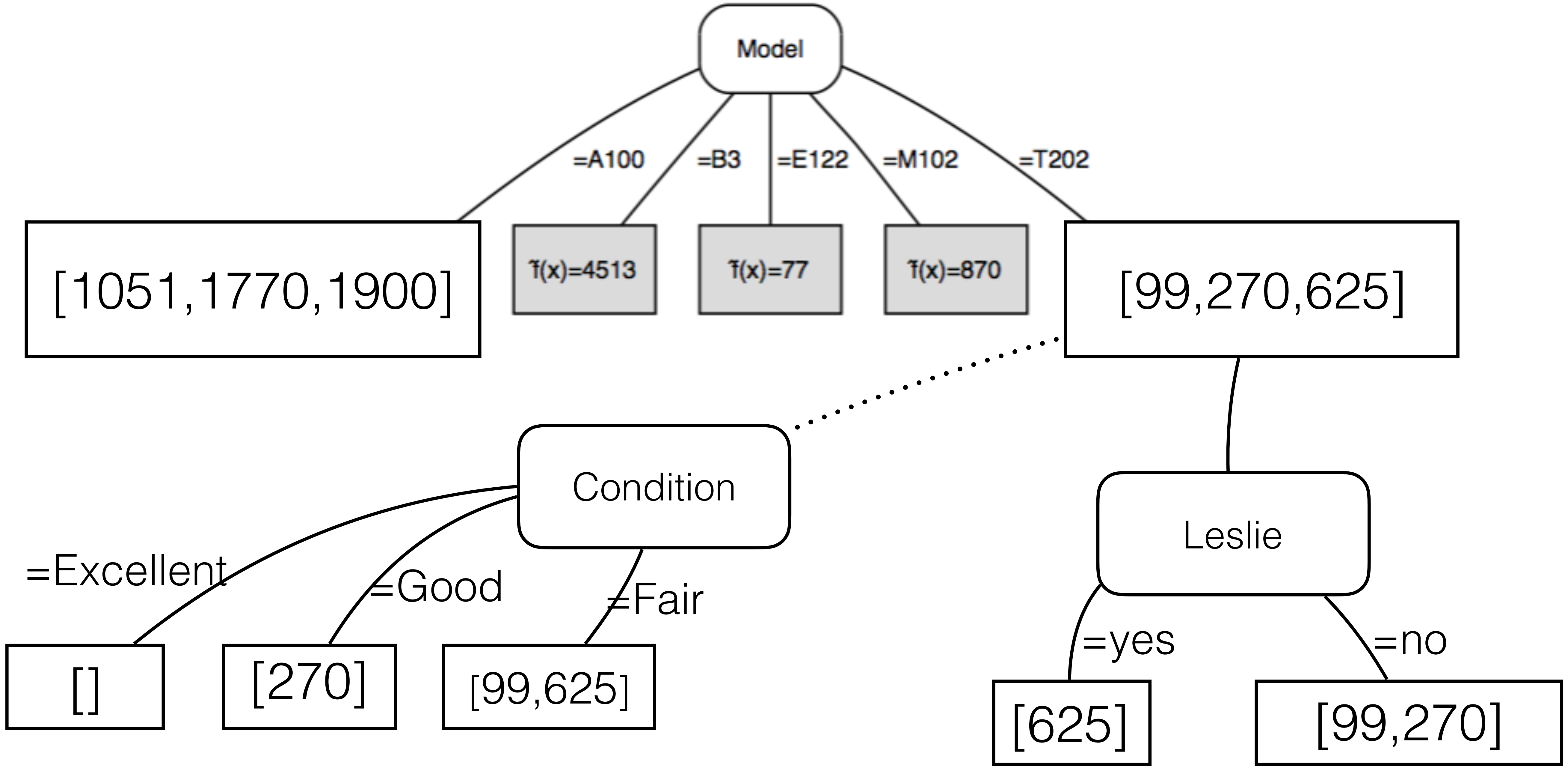
Partition on Condition



$$\sum_{j=1}^{l} \frac{|Y_j|}{|Y|}\overline{y}_j^2 = \frac{1}{4}(1770)^2 + \frac{2}{4}(\frac{1051+1900}{2})^2 + \frac{1}{4}(1574)^2 = 2.4 * 10^6$$

$$\sum_{j=1}^{l} \frac{|Y_j|}{|Y|}\overline{y}_j^2 = \frac{1}{3}(1900)^2 + \frac{2}{3}(\frac{1051+1770}{2})^2 = \boxed{2.5 * 10^6}$$

Partition on Condition

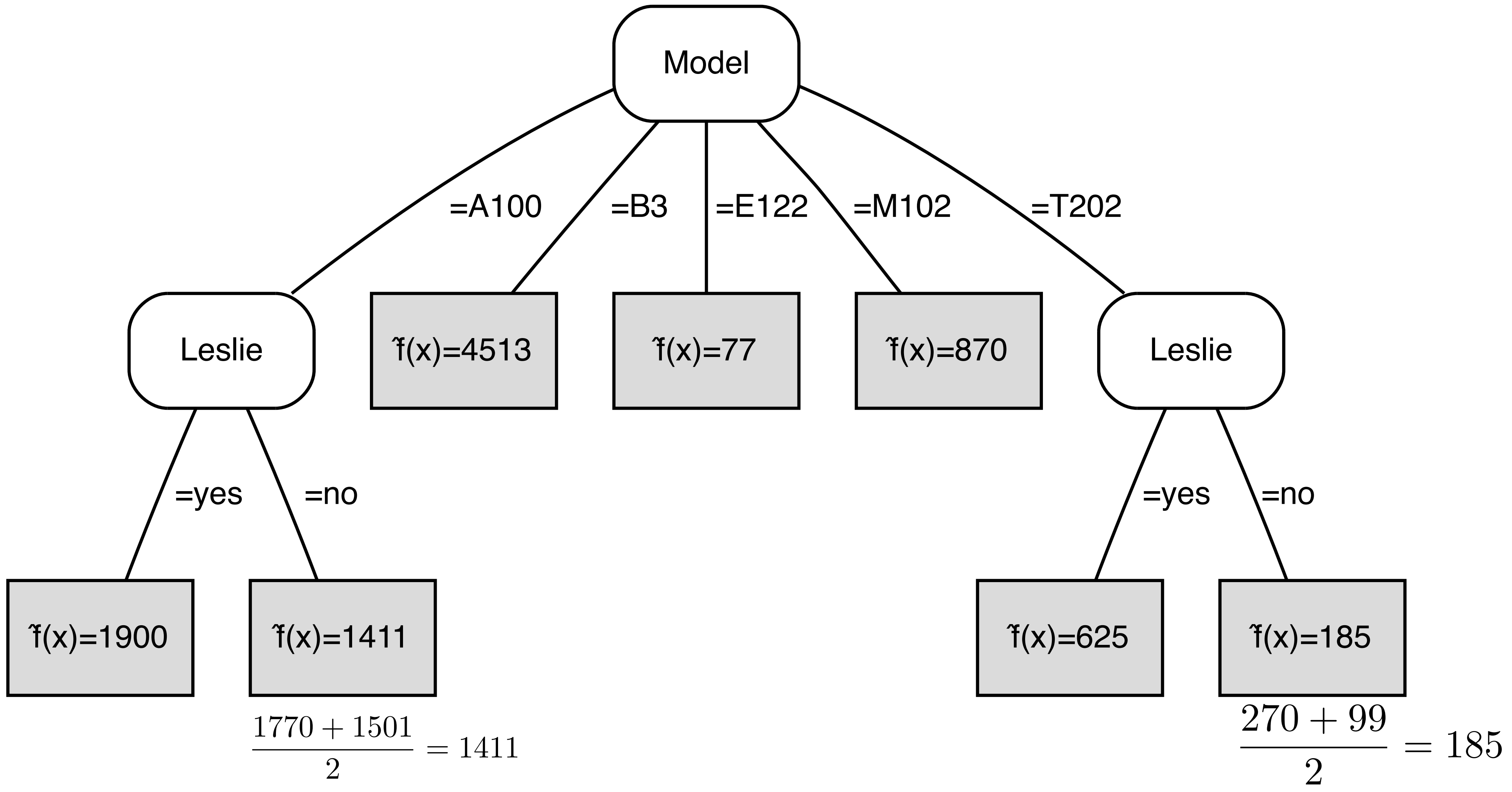


$$\sum_{j=1}^{l} \frac{|Y_j|}{|Y|}\overline{y}_j^2 = \frac{1}{4}(331)^2 + \frac{1}{4}(270)^2 + \frac{2}{4}(\frac{99+625}{2})^2 = 111 * 10^3$$

$$\sum_{j=1}^{l} \frac{|Y_j|}{|Y|}\overline{y}_j^2 = \frac{1}{3}(625)^2 + \frac{2}{3}(\frac{99+270}{2})^2 = \boxed{153 * 10^3}$$

Figure 5.8, p.150

A regression tree



A regression tree learned from the data in Example 5.4.

Clustering trees

- We introduce an abstract function $Dis : X \times X \mapsto R$ that measures the dissimilarity of two instances x_1 and x_2 in X.

- The higher $Dis(x_1, x_2)$ the less similar x_1 and x_2 are.

- The dissimilarity of a set (or cluster) of instances in D can be measured by:

$$Dis(D) = \frac{1}{|D|^2} \sum_{x_1 \in D} \sum_{x_2 \in D} Dis(x_1, x_2)$$

- The lower $Dis(D)$ the better will be the cluster of instances in D.

- *BestSplit(D,F)* is implemented with a weighted average of $Dis(D_j)$, where D_j is one of the partitions obtained after the split at the parent node:

$$Dis(\{D_1, ..., D_l\}) = \sum_{j=1}^{l} \frac{|D_j|}{|D|} \cdot Dis(D_j)$$

Dissimilarity with Euclidean Distance

- If an object x_j is described by *d* numerical features in a space $X \subseteq R^d$, each object can be described by a vector of *d* components, $\vec{x_j} = \langle x_{j1}, x_{j2}, \cdots, x_{jd} \rangle$ where x_{ji} represents the value of the feature i in object j.

- The dissimilarity $Dis(x_1,x_2)$ can be computed by a function of their distance:

$$Dis(x_1, x_2) = (\vec{x_1} - \vec{x_2})^2 = \sum_{i=1}^{d} (x_{1i} - x_{2i})^2$$

- The variance of the feature i in a set of N objects is:

$$Var_i(X) = \frac{1}{N} \sum_{j=1}^{N} [x_{ji} - \bar{x_{.i}}]^2$$

- The sum of the variances of all the features, can be interpreted as the average squared euclidean distance between vectors in a *d*-dimensional space.

$$\sum_{i=1}^{d} Var_i(X) = \frac{1}{N} \sum_{i=1}^{d} \sum_{j=1}^{N} [x_{ji} - \bar{x_{.i}}]^2 = \frac{1}{N} \sum_{j=1}^{N} (\vec{x_j} - \bar{\vec{x}})^2$$

Smart Computation of the Dissimilarity of a Dataset

- Since the average squared distance between any pairs of objects is twice the variance:

$$Dis(X) = \frac{1}{N^2} \sum_{\vec{x'} \in X} \sum_{\vec{x''} \in X} (\vec{x'} - \vec{x''})^2 = 2 \cdot Var(X)$$

- and variance *Var(X)* can be computed as a sum of all $Var_i(X)$:

$$Var(X) = \sum_{i=1}^{d} Var_i(X)$$

- In order to compute the dissimilarity of a dataset X all we need is the average vector instance $\vec{\bar{x}}$ and the computation of $Var_i(X)$, both of which can be computed in $O(|D|)$.

Example 5.5, p.152

Learning a clustering tree I

Assessing the nine transactions on the online auction site from Example 5.4, (see Ex. 5.6, p.154) using some additional features such as reserve price and number of bids, you come up with the following dissimilarity matrix: (between any transactions pairs)

	T1	T2	T3	T4	T5	T6	T7	T8	T9
T1	0	11	6	13	10	3	13	3	12
T2	11	**0**	1	**1**	1	3	**0**	4	0
T3	6	1	**0**	2	1	**1**	2	**2**	1
T4	13	**1**	2	**0**	0	4	**0**	4	0
T5	10	1	1	0	0	3	0	2	0
T6	3	3	**1**	4	3	**0**	4	**1**	3
T7	13	**0**	2	**0**	0	4	**0**	4	0
T8	3	4	**2**	4	2	**1**	4	**0**	4
T9	12	0	1	0	0	3	0	4	0

(we suppose this matrix is given by external advice)

This shows, for instance, that the first transaction is very different from the other eight. The average pairwise dissimilarity over all nine transactions is 2.94.

$$Dis(X) = \frac{1}{N^2}\sum_{\vec{x'}\in X}\sum_{\vec{x''}\in X}(\vec{x'}-\vec{x''})^2 = 2\cdot Var(X)$$

Example 5.6, p.154

Clustering with Euclidean distance I

We extend our Hammond organ data with two new numerical features, one indicating the reserve price and the other the number of bids made in the auction.

	Model	Condition	Leslie	Price	Reserve	Bids
T1	B3	excellent	no	45	30	22
T2	T202	fair	yes	6	0	9
T3	A100	good	no	11	8	13
T4	T202	good	no	3	0	1
T5	M102	good	yes	9	5	2
T6	A100	excellent	no	18	15	15
T7	T202	fair	no	1	0	3
T8	A100	good	yes	19	19	1
T9	E112	fair	no	1	0	5

Learning a clustering tree (1)

- We will learn a clustering tree by allowing feature with discrete values to be considered for the split conditions in the tree

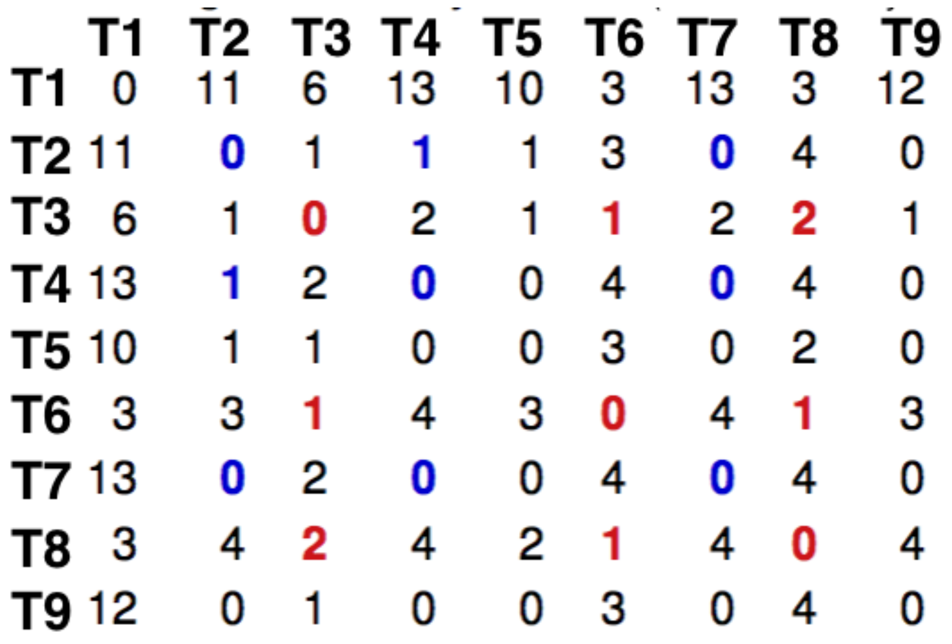
Model = [A100, B3, E112, M102, T202] [T3,T6,T8] [T1][T9] [T5][T2,T4,T7]
Condition = [excellent, good, fair] [T1,T6][T3,T4,T5,T8][T2,T7,T9]
Leslie = [yes, no] [T2,T5,T8] [T1,T3,T4,T6,T7,T9]

- Instead, we will use continuous features (like **bid, reserve and price**) to evaluate the variance of the examples in the partitions

- In this first example, we apply

$$Dis(D) = \frac{1}{|D|^2} \sum_{x_1 \in D} \sum_{x_2 \in D} Dis(x_1, x_2)$$

on the basis of dissimilarity matrix, to evaluate the dis-similarity of each partition



	T1	T2	T3	T4	T5	T6	T7	T8	T9
T1	0	11	6	13	10	3	13	3	12
T2	11	0	1	1	1	3	0	4	0
T3	6	1	0	2	1	1	2	2	1
T4	13	1	2	0	0	4	0	4	0
T5	10	1	1	0	0	3	0	2	0
T6	3	3	1	4	3	0	4	1	3
T7	13	0	2	0	0	4	0	4	0
T8	3	4	2	4	2	1	4	0	4
T9	12	0	1	0	0	3	0	4	0

Learning a clustering tree (2)

Model = [A100, B3, E112, M102, T202] [T3,T6,T8] [T1][T9] [T5] [T2,T4,T7]

Condition = [excellent, good, fair] [T1,T6] [T3,T4,T5,T8][T2,T7,T9]

Leslie = [yes, no] [T2,T5,T8] [T1,T3,T4,T6,T7,T9]

$$Dis([T3,T6,T8]) = \frac{1}{3^2}\{Dis(T3,T3) + Dis(T3,T6) + Dis(T3,T8) + \cdots + Dis(T3,T8) + Dis(T6,T8) + Dis(T8,T8)\}$$

$$Dis([T3,T6,T8]) = \frac{1}{9}\{0+1+2+1+0+1+2+1+0\} = 0.89$$

$$Dis([T1]) = Dis([T_i]) = 0$$

	T1	T2	T3	T4	T5	T6	T7	T8	T9
T1	0	11	6	13	10	3	13	3	12
T2	11	0	1	1	1	3	0	4	0
T3	6	1	0	2	1	1	2	2	1
T4	13	1	2	0	0	4	0	4	0
T5	10	1	1	0	0	3	0	2	0
T6	3	3	1	4	3	0	4	1	3
T7	13	0	2	0	0	4	0	4	0
T8	3	4	2	4	2	1	4	0	4
T9	12	0	1	0	0	3	0	4	0

Learning a clustering tree (3)

Model = [A100, B3, E112, M102, T202] [T3,T6,T8] [T1][T9] [T5] [T2,T4,T7]

Condition = [excellent, good, fair] [T1,T6] [T3,T4,T5,T8][T2,T7,T9]

Leslie = [yes, no] [T2,T5,T8] [T1,T3,T4,T6,T7,T9]

$$Dis([T2,T4,T7]) = \frac{1}{3^2}\{Dis(T2,T2) + Dis(T2,T4) + Dis(T2,T7) + \cdots + Dis(T3,T7) + Dis(T4,T7) + Dis(T7,T7)\}$$

$$Dis([T2,T4,T7]) = \frac{1}{9}\{0+1+0+1+0+0+0+0+0\} = 0.22$$

$$Dis([T1]) = Dis([T_i]) = 0$$

$$Dis(\{D_1, ..., D_l\}) = \sum_{j=1}^{l} \frac{|D_j|}{|D|} \cdot Dis(D_j)$$

$$Dis(Split\_on\_Model) = \frac{3}{9}0.89 + \frac{1}{9}0 + \frac{1}{9}0 + \frac{1}{9}0 + \frac{3}{9}0.22 = \boxed{0.37}$$

	T1	T2	T3	T4	T5	T6	T7	T8	T9
T1	0	11	6	13	10	3	13	3	12
T2	11	0	1	1	1	3	0	4	0
T3	6	1	0	2	1	1	2	2	1
T4	13	1	2	0	0	4	0	4	0
T5	10	1	1	0	0	3	0	2	0
T6	3	3	1	4	3	0	4	1	3
T7	13	0	2	0	0	4	0	4	0
T8	3	4	2	4	2	1	4	0	4
T9	12	0	1	0	0	3	0	4	0

Learning a clustering tree (4)

Condition = [excellent, good, fair] [T1,T6][T3,T4,T5,T8][T2,T7,T9]

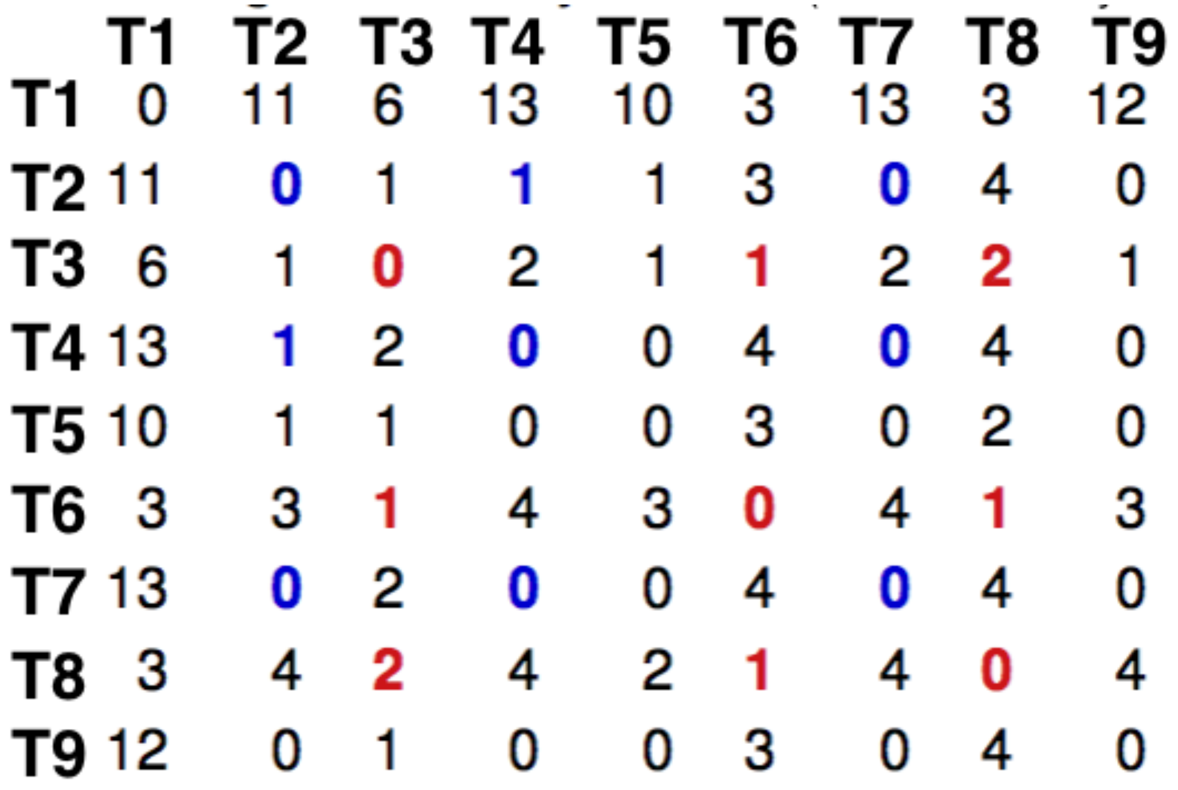
Leslie = [yes, no] [T2,T5,T8] [T1,T3,T4,T6,T7,T9]

$$Dis(Split\_on\_Condition) = \frac{2}{9}1.5 + \frac{4}{9}1.19 + \frac{3}{9}0 = \boxed{0.86}$$

$$Dis(Split\_on\_Leslie) = \frac{3}{9}1.56 + \frac{6}{9}3.56 = \boxed{2.89}$$

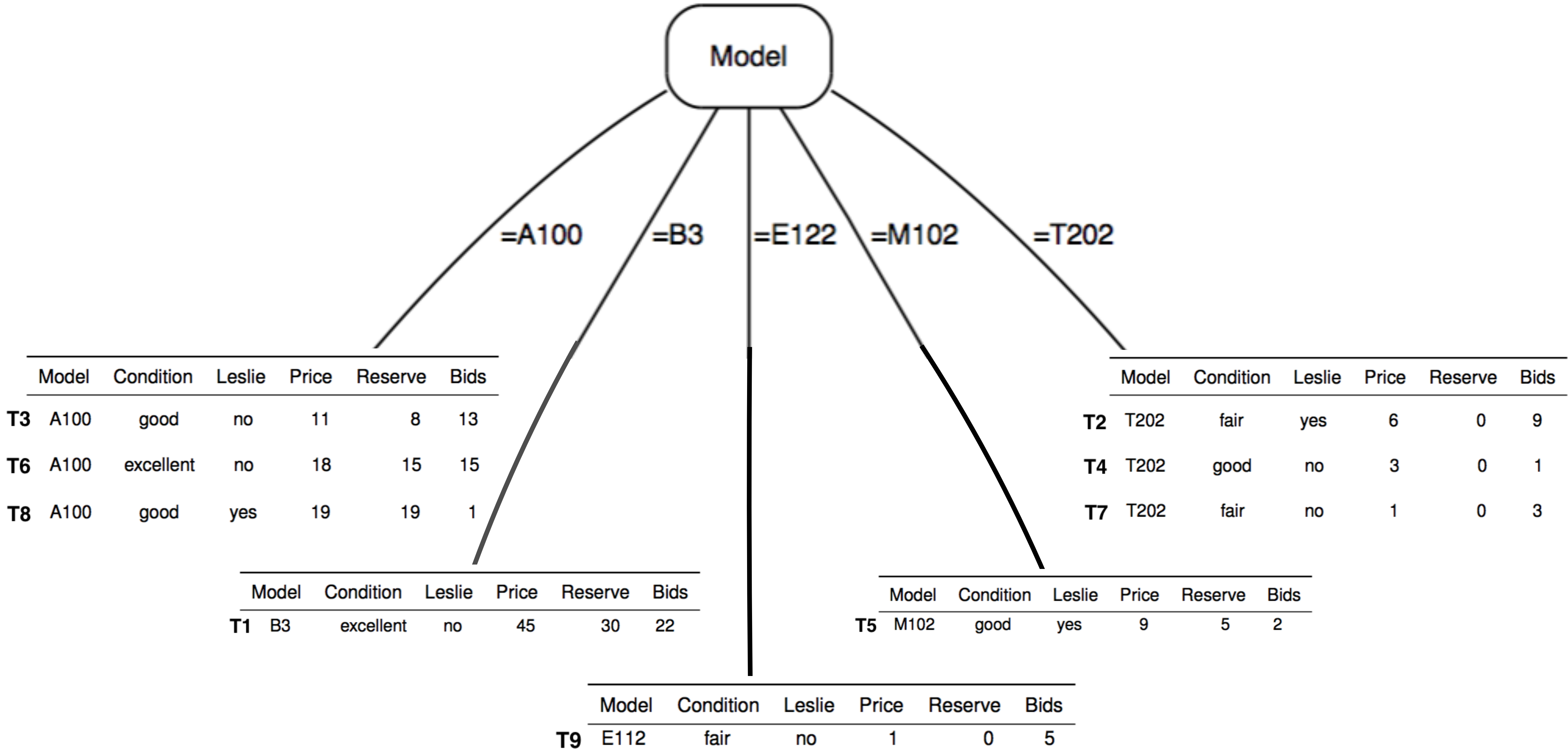
As a conclusion, the best split is on **Model** which reduces the most the objective function

Instead **Leslie** is virtually unrelated with the value of price because the obtained dissimilarity (2.89) is comparable to the dissimilarity in the original values (overall dataset) (2.94)



	T1	T2	T3	T4	T5	T6	T7	T8	T9
T1	0	11	6	13	10	3	13	3	12
T2	11	0	1	1	1	3	0	4	0
T3	6	1	0	2	1	1	2	2	1
T4	13	1	2	0	0	4	0	4	0
T5	10	1	1	0	0	3	0	2	0
T6	3	3	1	4	3	0	4	1	3
T7	13	0	2	0	0	4	0	4	0
T8	3	4	2	4	2	1	4	0	4
T9	12	0	1	0	0	3	0	4	0

The resulting tree



	Model	Condition	Leslie	Price	Reserve	Bids
T3	A100	good	no	11	8	13
T6	A100	excellent	no	18	15	15
T8	A100	good	yes	19	19	1

	Model	Condition	Leslie	Price	Reserve	Bids
T1	B3	excellent	no	45	30	22

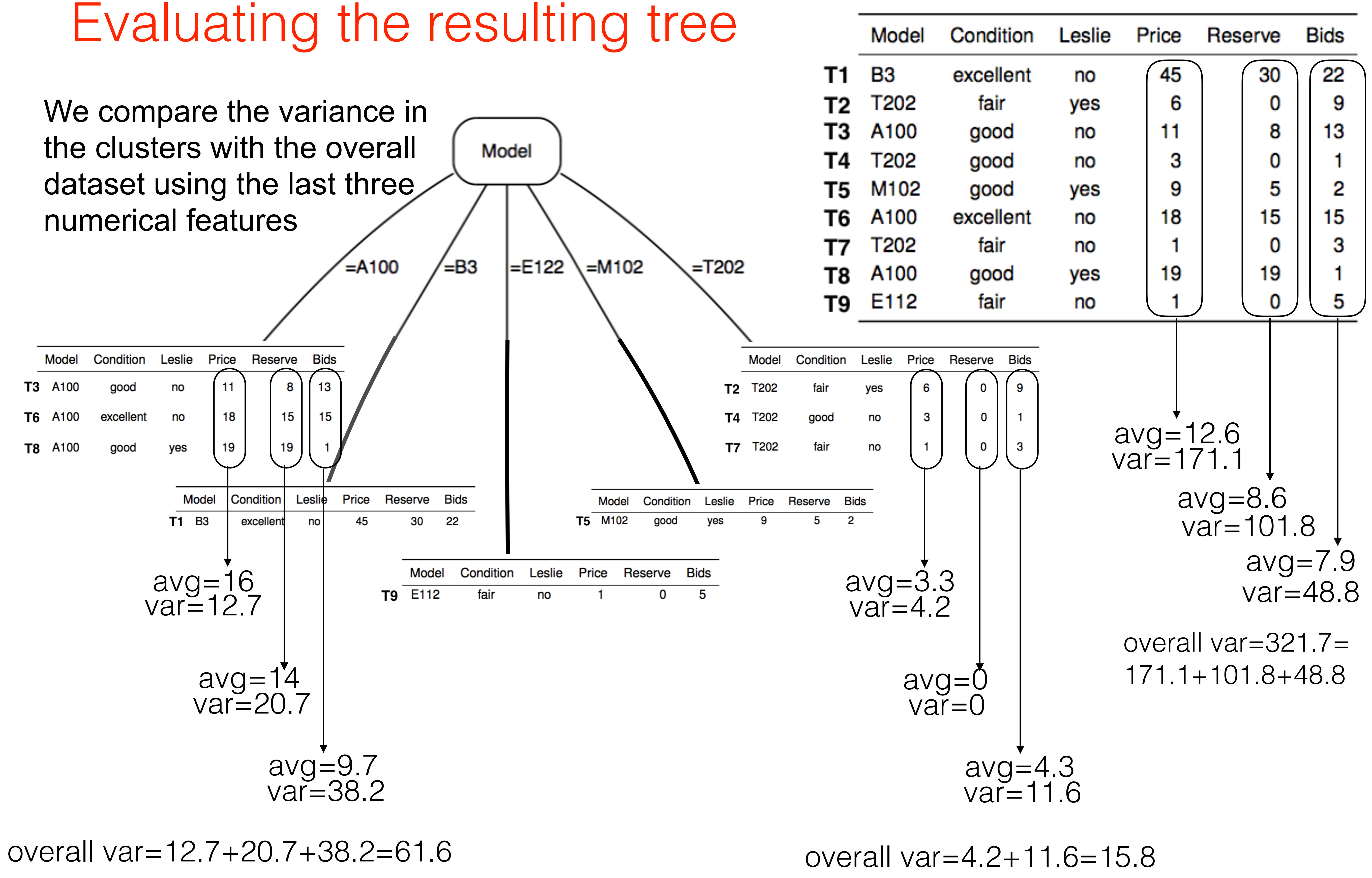
	Model	Condition	Leslie	Price	Reserve	Bids
T9	E112	fair	no	1	0	5

	Model	Condition	Leslie	Price	Reserve	Bids
T5	M102	good	yes	9	5	2

	Model	Condition	Leslie	Price	Reserve	Bids
T2	T202	fair	yes	6	0	9
T4	T202	good	no	3	0	1
T7	T202	fair	no	1	0	3

Evaluating the resulting tree

We compare the variance in the clusters with the overall dataset using the last three numerical features



	Model	Condition	Leslie	Price	Reserve	Bids
T1	B3	excellent	no	45	30	22
T2	T202	fair	yes	6	0	9
T3	A100	good	no	11	8	13
T4	T202	good	no	3	0	1
T5	M102	good	yes	9	5	2
T6	A100	excellent	no	18	15	15
T7	T202	fair	no	1	0	3
T8	A100	good	yes	19	19	1
T9	E112	fair	no	1	0	5

avg=12.6
var=171.1

avg=8.6
var=101.8

avg=7.9
var=48.8

overall var=321.7=
171.1+101.8+48.8

	Model	Condition	Leslie	Price	Reserve	Bids
T3	A100	good	no	11	8	13
T6	A100	excellent	no	18	15	15
T8	A100	good	yes	19	19	1

	Model	Condition	Leslie	Price	Reserve	Bids
T1	B3	excellent	no	45	30	22

	Model	Condition	Leslie	Price	Reserve	Bids
T9	E112	fair	no	1	0	5

	Model	Condition	Leslie	Price	Reserve	Bids
T5	M102	good	yes	9	5	2

	Model	Condition	Leslie	Price	Reserve	Bids
T2	T202	fair	yes	6	0	9
T4	T202	good	no	3	0	1
T7	T202	fair	no	1	0	3

avg=16
var=12.7

avg=14
var=20.7

avg=9.7
var=38.2

overall var=12.7+20.7+38.2=61.6

avg=3.3
var=4.2

avg=0
var=0

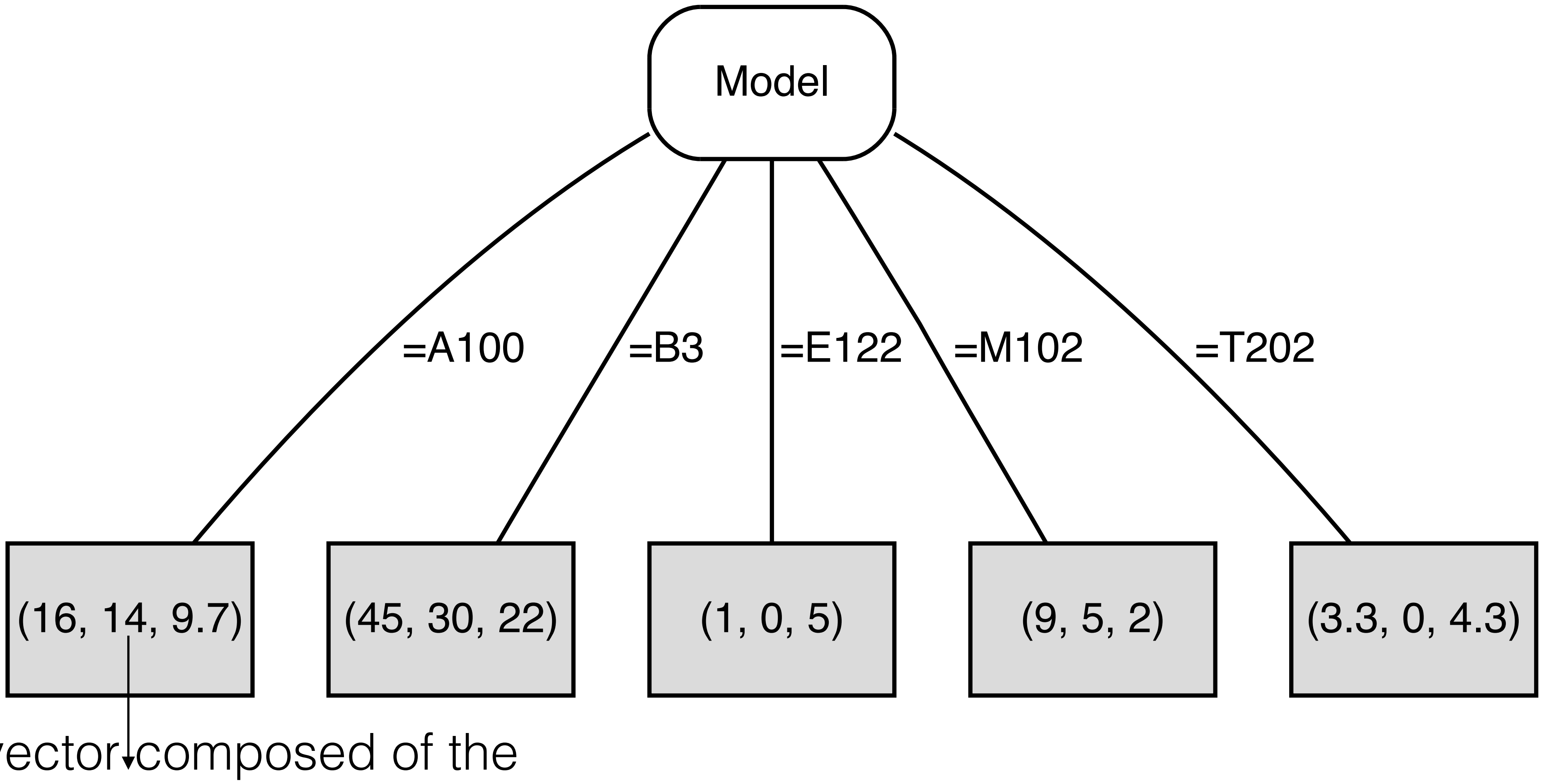
avg=4.3
var=11.6

overall var=4.2+11.6=15.8

Thus clustering reduced the average squared distance between the objects w.r.t. the beginning.

Figure 5.9, p.154

A clustering tree



cluster mean vector composed of the
mean of the 3 numerical features

A clustering tree learned from the data in Example 5.6 using Euclidean distance on the numerical features.

Observations on Clustering Trees

- **Smaller clusters** tend to have a lower dissimilarity and provoke overfitting.
    - It is recommended setting aside a pruning set to remove the splits (in lower levels of the tree) if they do not improve the cluster cohesion (dissimilarity does not decrease) on the pruning set.
    - Furthermore, single examples (outliers) can dominate in the computation of the dissimilarity.
      It might be beneficial to remove them and recompute the measure Dis.
- **How do we label each cluster?**
    - We choose the *best representative instance*
    - It might be that instance that has the lowest total dissimilarity with all the other instances of the same cluster (called the *medoid*).

Relationship between dissimilarity, cluster cohesion and cluster separation

- Given a dataset D, its variance (and *Dis(D)*) are constant
- Consider a set of k clusters $(D_k, \ldots, D_k)$ of D
- Let us evaluate the total sum of errors (between each data point and the overall centroid, c). $\vec{c} = \frac{1}{|D|} \sum_{\vec{x} \in D} \vec{x}$
  Let us call it TSSE.

$$TSSE = \sum_{\vec{x} \in D} (\vec{x} - \vec{c})^2 = |D| \cdot Var(D) = \frac{|D|}{2} Dis(D)$$

cluster cohesion (related to the weighted average variance)

- We can see that: $TSSE = WSSE + BSSE$

cluster separation

$$WSSE = \sum_{i=1..k} \sum_{\vec{x} \in D_i} (\vec{x} - \vec{c_i})^2 \qquad \vec{c_i} = \frac{1}{|D_i|} \sum_{\vec{x} \in D_i} \vec{x}$$

$$BSSE = \sum_{i=1..k} |D_i| (\vec{c_i} - \vec{c})^2$$

Splitting with Numerical Features

- In these examples, we used categorical features for splitting and numerical features (defined as features whose values can be ordered) for dissimilarity calculations

- In practice, also numerical features could be used also for splitting.

- A condition for the split would be the following:

$$f \geq t \quad or \quad f < t$$

with t a value of f.

- Function *BestSplit* would need to find the optimal value t. This task is strictly related to *discretization* (see cap. 10).