UNIT- III

Tree models: Decision trees, Ranking and probability estimation trees, Tree learning as variance reduction. Rule models: Learning ordered rule lists, Learning unordered rule sets, Descriptive rule learning, First-order rule learning

1)Tree models:

- A tree model is a hierarchical structure of conditions, where leafs contain tree outcome.
- They represent recursive divide-and-conquer strategies.
- Tree models are among the most popular models in machine learning, because they are easy to understand and interpret:
- E.g., Kinect uses them to detect character pose.

a) Decision trees

b) Ranking and probability estimation trees

c) Tree learning as variance reduction

a) Decision trees

A **decision tree** is a flowchart-like structure in which each internal node represents a "test" on an attribute (e.g. whether a coin flip comes up heads or tails), each branch represents the outcome of the test, and each leaf node represents a class label (**Decision** taken after computing all attributes). The paths from root to leaf represent classification rules.

Decision Analysis, a decision tree and the closely related influence diagram are used as a visual and analytical decision support tool, where the expected values (or expected utility) of competing alternatives are calculated.

A decision tree consists of three types of nodes:

1. Decision nodes – typically represented by squares
2. Chance nodes – typically represented by circles
3. End nodes – typically represented by triangles

Decision trees are commonly used in operations research and operations management. If, in practice, decisions have to be taken online with no recall under incomplete knowledge, a decision tree should be paralleled by a probability model as a best choice model or online selection model algorithm. Another use of decision trees is as a descriptive means for calculating conditional probabilities.

Decision trees, influence diagrams, utility functions, and other decision analysis tools and methods are taught to undergraduate students in schools of business, health economics, and public health, and are examples of operations research or management science methods.

Decision tree rules

The decision tree can be linearized into decision rules, where the outcome is the contents of the leaf node, and the conditions along the path form a conjunction in the if clause. In general, the rules have the form: if condition1 and condition2 and condition3 then outcome.

Decision rules can be generated by constructing association rules with the target variable on the right. They can also denote temporal or causal relations

Decision tree

How to define BestSplit(D, F) for classification?

Assume that we have binary features and two classes only. Let $D^{\oplus}$ denote set of instances from positive class and $D^{\ominus}$ from negative class, $D = D^{\oplus} \cup D^{\ominus}$.

Let split D into D_1 and D_2 using an attribute. The best situation is where $D_1^{\oplus} = D^{\oplus}$ and $D_1^{\ominus} = \emptyset$ or $D_1^{\oplus} = \emptyset$ and $D_1^{\ominus} = D^{\ominus}$. In this cases the child node is said to be *pure*.

This, however, is unlikely in practice, thus we have to measure *impurity* of children nodes somehow.

Impurity measures

- Minority class $min(\dot{p}, 1-\dot{p})$ – proportion of misclassified examples if labeling leaf using majority class
- Gini index $2\dot{p}(1-\dot{p})$ – expected error if labeling examples in leaf randomly with prorability $\dot{p}$for positive class and $1-\dot{p}$ for negative class
- Entropy $-\dot{p}\log_2 \dot{p} - (1-\dot{p})\log_2(1-\dot{p})$ – expected amount of information in bits required to classify an example in leaf

Decision tree example

Suppose you come across a number of sea animals that you suspect belong to the same species. You observe their length in metres, whether they have gills, whether they have a prominent beak, and whether they have few or many teeth. Let the following be dolphins (positive class):

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

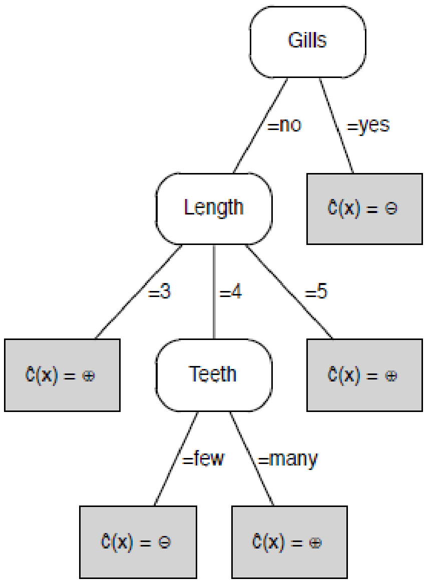
and the following be not dolphins (negative class):

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

Figure 5.1, p.130

Decision tree example

A decision tree learned on this data separates the positives and negatives perfectly.



Algorithm 5.2, p.137

Finding the best split for a decision tree

```
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 $\mathrm{Imp}(\{D_1, \ldots, D_l\}) < I_{\min}$ then
5          $I_{\min} \leftarrow \mathrm{Imp}(\{D_1, \ldots, D_l\})$;
6          $f_{\text{best}} \leftarrow f$;
7      end
8  end
9  return $f_{\text{best}}$
```

b) Ranking and probability estimation trees

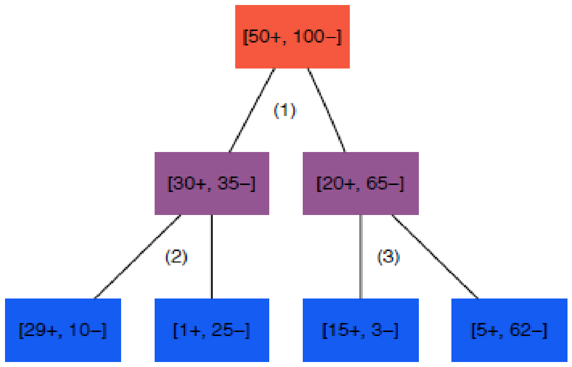
Decision trees divide the instance space into segments, by learning ordering on those segments the decision trees can be turned into rankers.

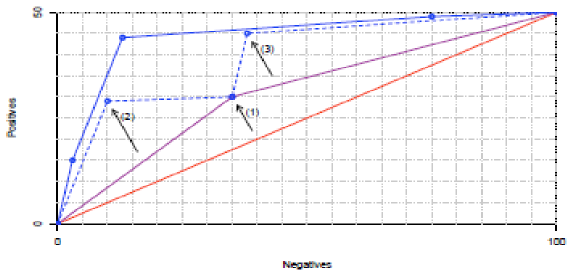
Thanks to access to class distribution in each leaf the optimal orderdering for the training data can be obtained from empirical probabilities $\dot{p}$ (of positive class).

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

Figure 5.4, p.140

Growing a tree

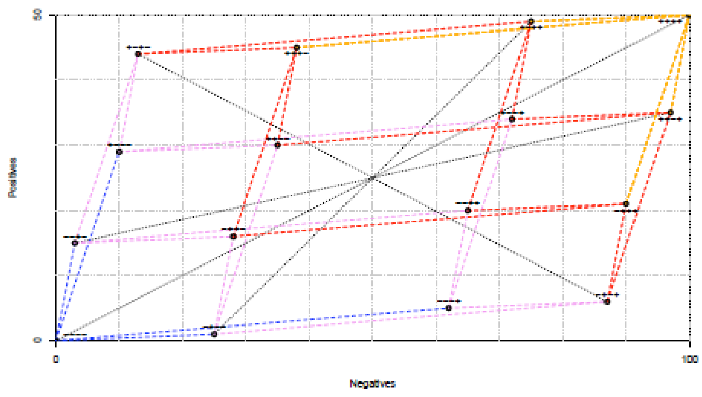




(left) Abstract representation of a tree with numbers of positive and negative examples covered in each node. Binary splits are added to the tree in the order indicated. **(right)** Adding a split to the tree will add new segments to the coverage curve as indicated by the arrows. After a split is added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

Figure 5.5, p.141

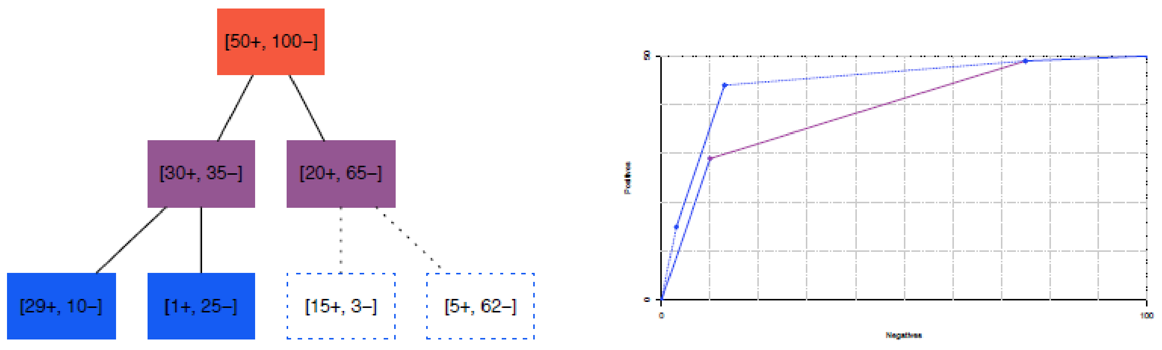
Labelling a tree



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.

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.

- Prunning must not improve classification accuracy on training set
- However may improve generalization accuracy on test set
- A popular algorithm for pruning decision trees is *reduced-error pruning* that employs a separate *prunning set* of labelled data not seen during training.

Algorithm 5.3, p.144

Reduced-error pruning

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

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

Example 5.3, p.144

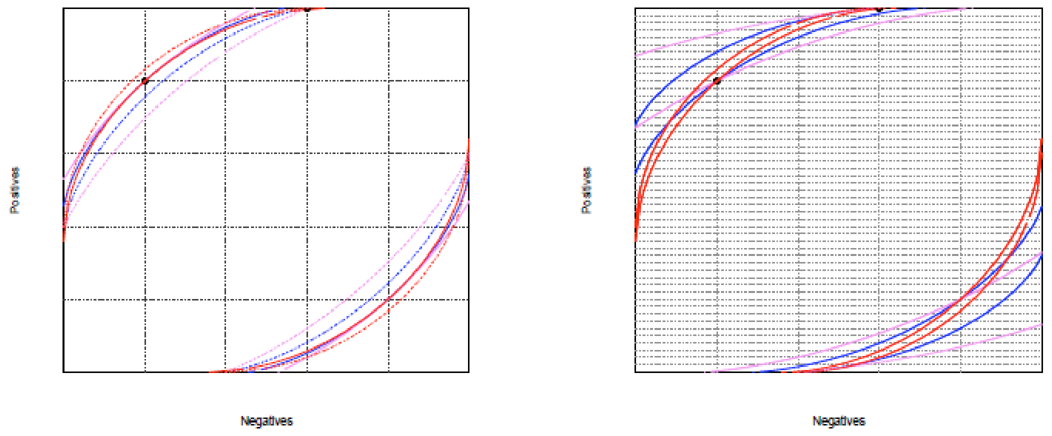
Skew sensitivity of splitting criteria II

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

- ☞ 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.

Figure 5.7, p.146

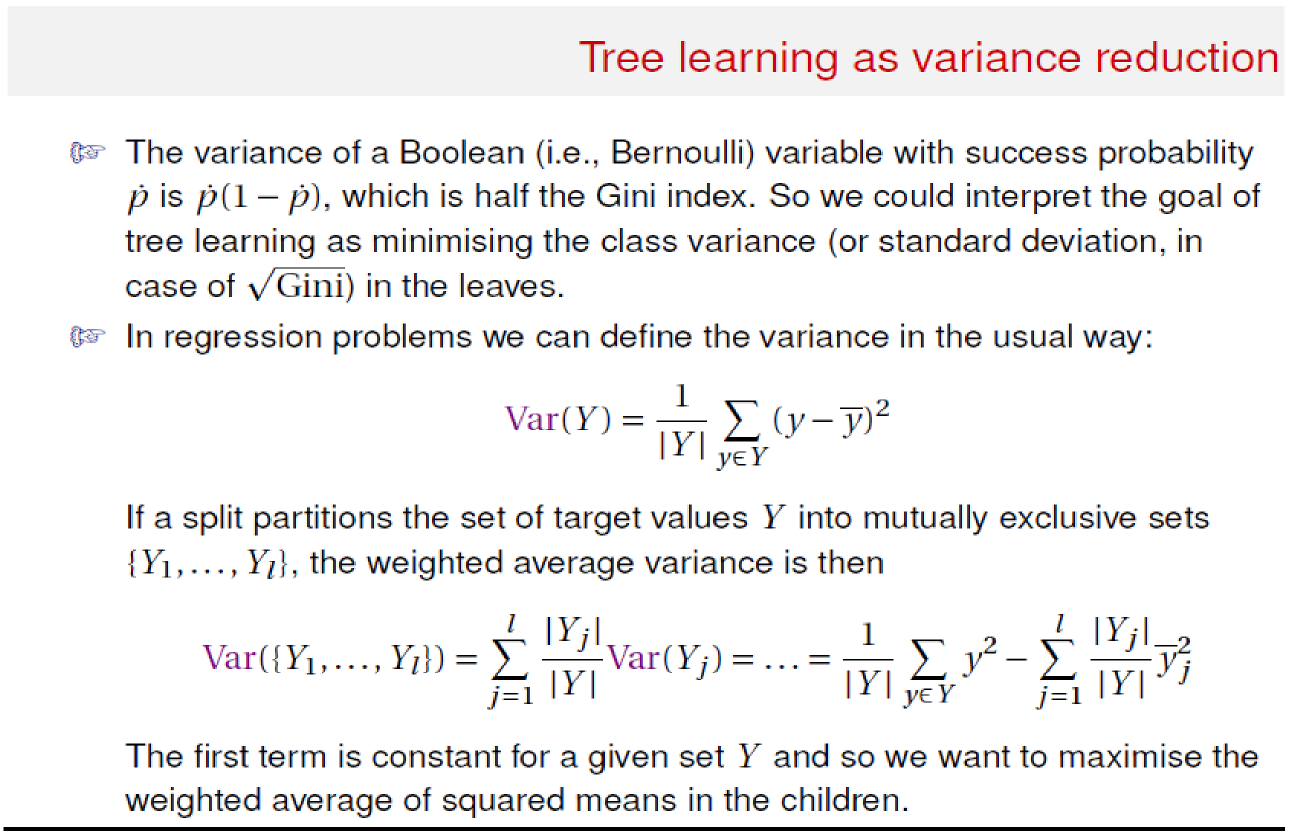
Skew sensitivity of splitting criteria



(left) ROC isometrics for entropy in blue, Gini index in violet and $\sqrt{\text{Gini}}$ in red through the splits $[8+,2-][2+,8-]$ (solid lines) and $[10+,6-][0+,4-]$ (dotted lines). Only $\sqrt{\text{Gini}}$ prefers the second split. **(right)** The same isometrics after inflating the positives with a factor 10. All splitting criteria now favour the second split; the $\sqrt{\text{Gini}}$ isometrics are the only ones that haven't moved.

c) Tree learning as variance reduction

(i)Regression Tree



Tree learning as variance reduction

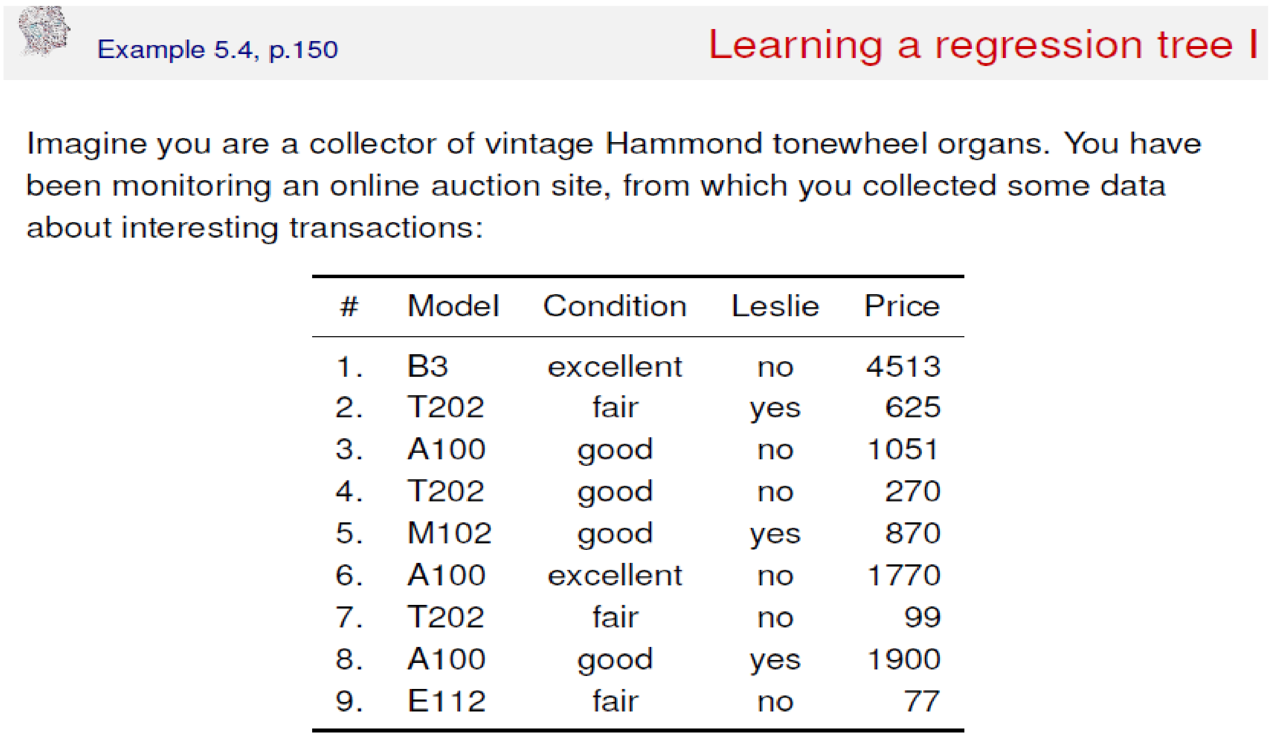
- The variance of a Boolean (i.e., Bernoulli) variable with success probability $\dot{p}$ is $\dot{p}(1-\dot{p})$, which is half the Gini index. So we could interpret the goal of tree learning as minimising the class variance (or standard deviation, in case of $\sqrt{\text{Gini}}$) in the leaves.
- In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

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

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.



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

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.
There are three features, hence three possible splits:

Model = [A100, B3, E112, M102, T202]
[1051, 1770, 1900][4513][77][870][99, 270, 625]
Condition = [excellent, good, fair]
[1770, 4513][270, 870, 1051, 1900][77, 99, 625]
Leslie = [yes, no] [625, 870, 1900][77, 99, 270, 1051, 1770, 4513]

The means of the first split are 1574, 4513, 77, 870 and 331, and the weighted average of squared means is $3.21 \cdot 10^6$.
The means of the second split are 3142, 1023 and 267, with weighted average of squared means $2.68 \cdot 10^6$;
for the third split the means are 1132 and 1297, with weighted average of squared means $1.55 \cdot 10^6$.
We therefore branch on Model at the top level. This gives us three single-instance leaves, as well as three A100s and three T202s.

Example 5.4, p.150

Learning a regression tree III

For the A100s we obtain the following splits:

Condition = [excellent, good, fair]	[1770][1051, 1900][]
Leslie = [yes, no]	[1900][1051, 1770]

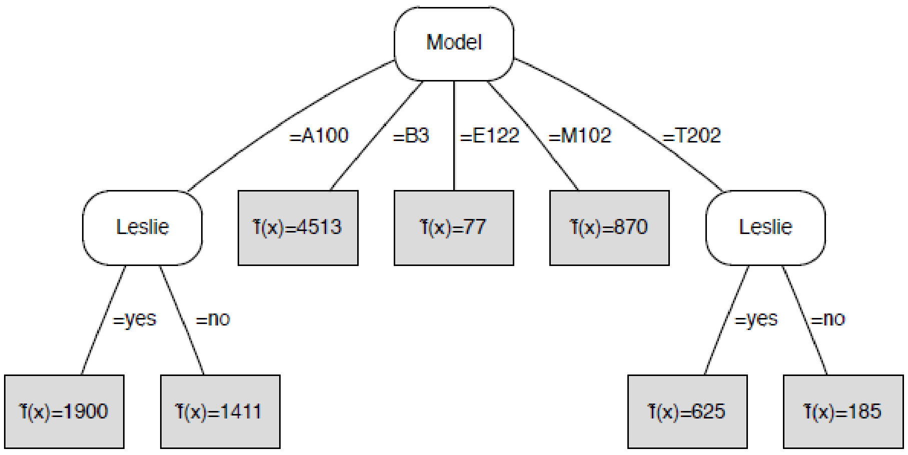
Without going through the calculations we can see that the second split results in less variance (to handle the empty child, it is customary to set its variance equal to that of the parent). For the T202s the splits are as follows:

Condition = [excellent, good, fair]	[][270][99, 625]
Leslie = [yes, no]	[625][99, 270]

Again we see that splitting on Leslie gives tighter clusters of values. The learned regression tree is depicted in Figure 5.8.

Figure 5.8, p.150

A regression tree



A regression tree learned from the data in Example 5.4.

(ii)Clustering Tree

Example 5.5, p.152

Learning a clustering tree I

Assessing the nine transactions on the online auction site from Example 5.4, using some additional features such as reserve price and number of bids, you come up with the following dissimilarity matrix:

$$
\begin{array}{ccccccccc}
0 & 11 & 6 & 13 & 10 & 3 & 13 & 3 & 12 \\
11 & \mathbf{0} & 1 & \mathbf{1} & 1 & 3 & \mathbf{0} & 4 & 0 \\
6 & 1 & \mathbf{0} & 2 & 1 & \mathbf{1} & 2 & \mathbf{2} & 1 \\
13 & \mathbf{1} & 2 & \mathbf{0} & 0 & 4 & \mathbf{0} & 4 & 0 \\
10 & 1 & 1 & 0 & 0 & 3 & 0 & 2 & 0 \\
3 & 3 & \mathbf{1} & 4 & 3 & \mathbf{0} & 4 & \mathbf{1} & 3 \\
13 & \mathbf{0} & 2 & \mathbf{0} & 0 & 4 & \mathbf{0} & 4 & 0 \\
3 & 4 & \mathbf{2} & 4 & 2 & \mathbf{1} & 4 & \mathbf{0} & 4 \\
12 & 0 & 1 & 0 & 0 & 3 & 0 & 4 & 0
\end{array}
$$

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.

Example 5.5, p.152

Learning a clustering tree II

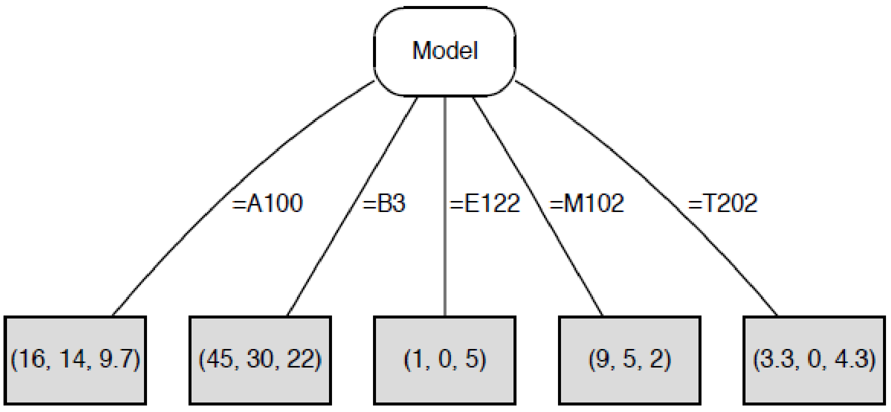
Using the same features from Example 5.4, the three possible splits are (now with transaction number rather than price):

Model = [A100, B3, E112, M102, T202]	[3,6,8][1][9][5][2,4,7]
Condition = [excellent, good, fair]	[1,6][3,4,5,8][2,7,9]
Leslie = [yes, no]	[2,5,8][1,3,4,6,7,9]

The cluster dissimilarity among transactions 3, 6 and 8 is $\frac{1}{3^2}(0+1+2+1+0+1+2+1+0) = 0.89$; and among transactions 2, 4 and 7 it is $\frac{1}{3^2}(0+1+0+1+0+0+0+0+0) = 0.22$. The other three children of the first split contain only a single element and so have zero cluster dissimilarity. The weighted average cluster dissimilarity of the split is then $3/9 \cdot 0.89 + 1/9 \cdot 0 + 1/9 \cdot 0 + 1/9 \cdot 0 + 3/9 \cdot 0.22 = 0.37$. For the second split, similar calculations result in a split dissimilarity of $2/9 \cdot 1.5 + 4/9 \cdot 1.19 + 3/9 \cdot 0 = 0.86$, and the third split yields $3/9 \cdot 1.56 + 6/9 \cdot 3.56 = 2.89$. The Model feature thus captures most of the given dissimilarities, while the Leslie feature is virtually unrelated.

Figure 5.9, p.154

A clustering tree



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

2) Rule models:

They are four types of rule models

a) Learning ordered rule lists, b) Learning unordered rule sets,

c) Descriptive rule learning   d) First-order rule learning

a) Learning ordered rule lists

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 $R$.
1 $R \leftarrow \emptyset$;
2 while $D \neq \emptyset$ do
3     $r \leftarrow$ LearnRule($D$) ;                // LearnRule: see Algorithm 6.2
4     append $r$ to the end of $R$;
5     $D \leftarrow D \setminus \{x \in D | x$ is covered by $r\}$;
6 end
7 return $R$
```

Example 6.1, p.159 **Learning a rule list I**

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

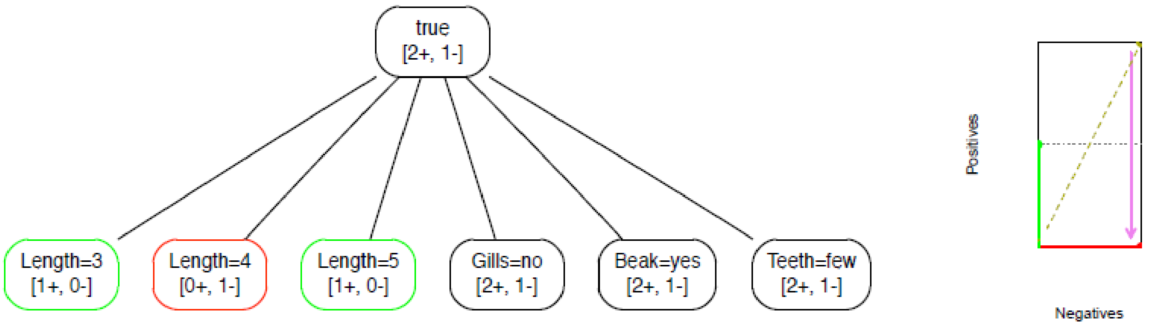
Example 6.1, p.159

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 6.4, p.162

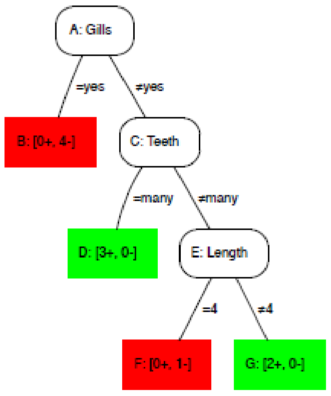
Constructing the third rule

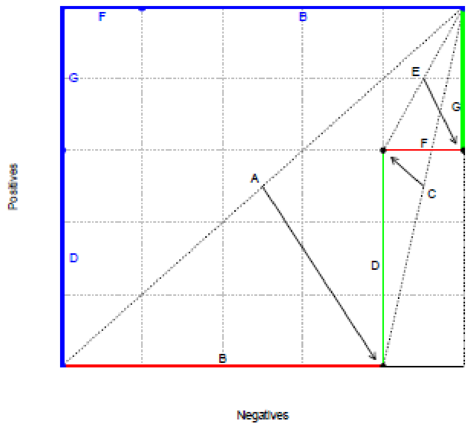


(left) The third rule covers the one remaining negative example, so that the remaining positives can be swept up by a default rule. **(right)** This will collapse the coverage space.

Figure 6.5, p.164

Rule list as a tree

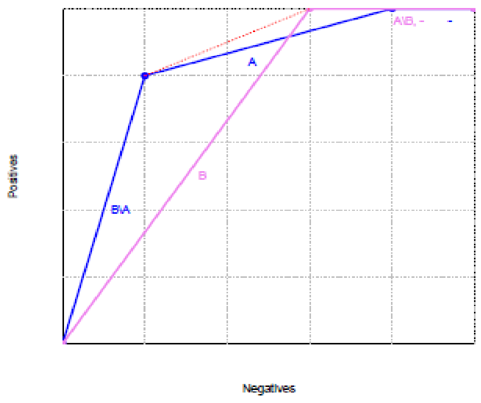




(left) A right-branching feature tree corresponding to a list of single-literal rules. **(right)** The construction of this feature tree depicted in coverage space. The leaves of the tree are either purely positive (in green) or purely negative (in red). Reordering these leaves on their empirical probability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

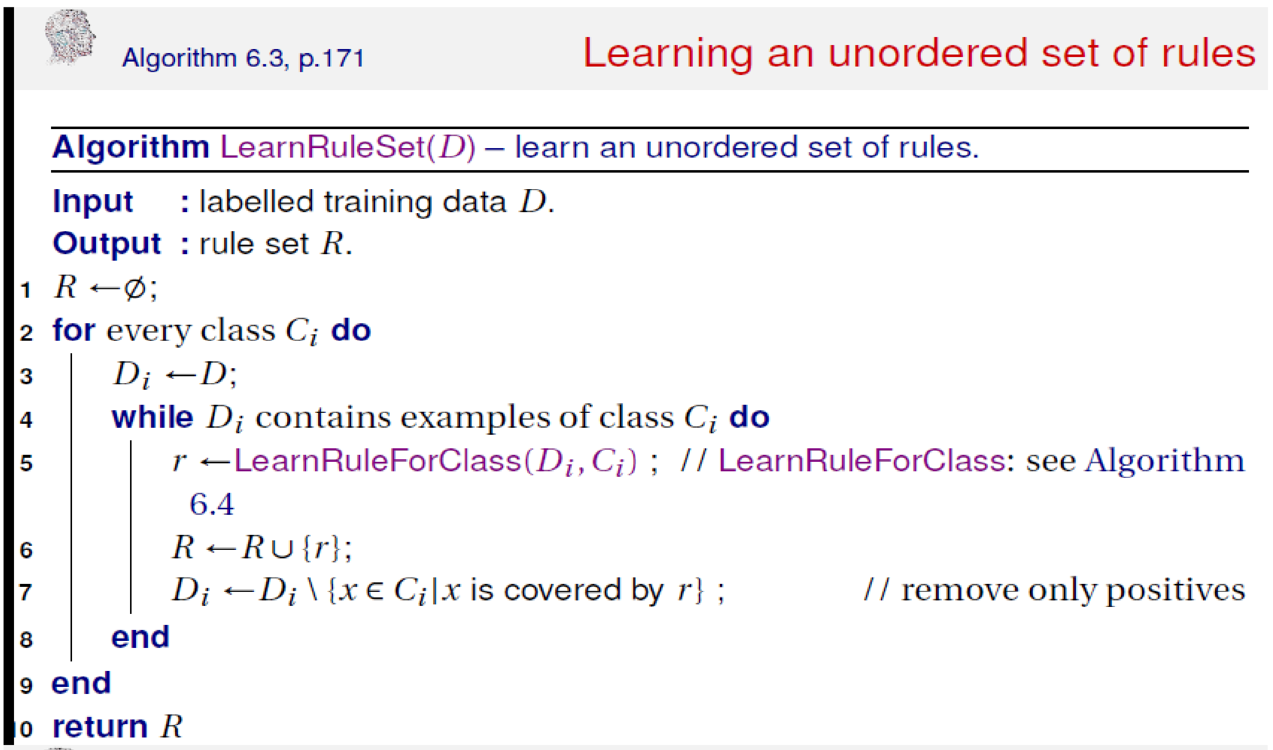
i)Rules List For Ranking and Probability Estimations

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



Coverage curves of two rule lists consisting of the rules from Example 6.2, in different order (AB in blue and BA in violet). B \ A corresponds to the coverage of rule B once the coverage of rule A is taken away, and '-' denotes the default rule. The dotted segment in red connecting the two curves corresponds to the overlap of the two rules A ∧ B, which is not accessible by either rule list.

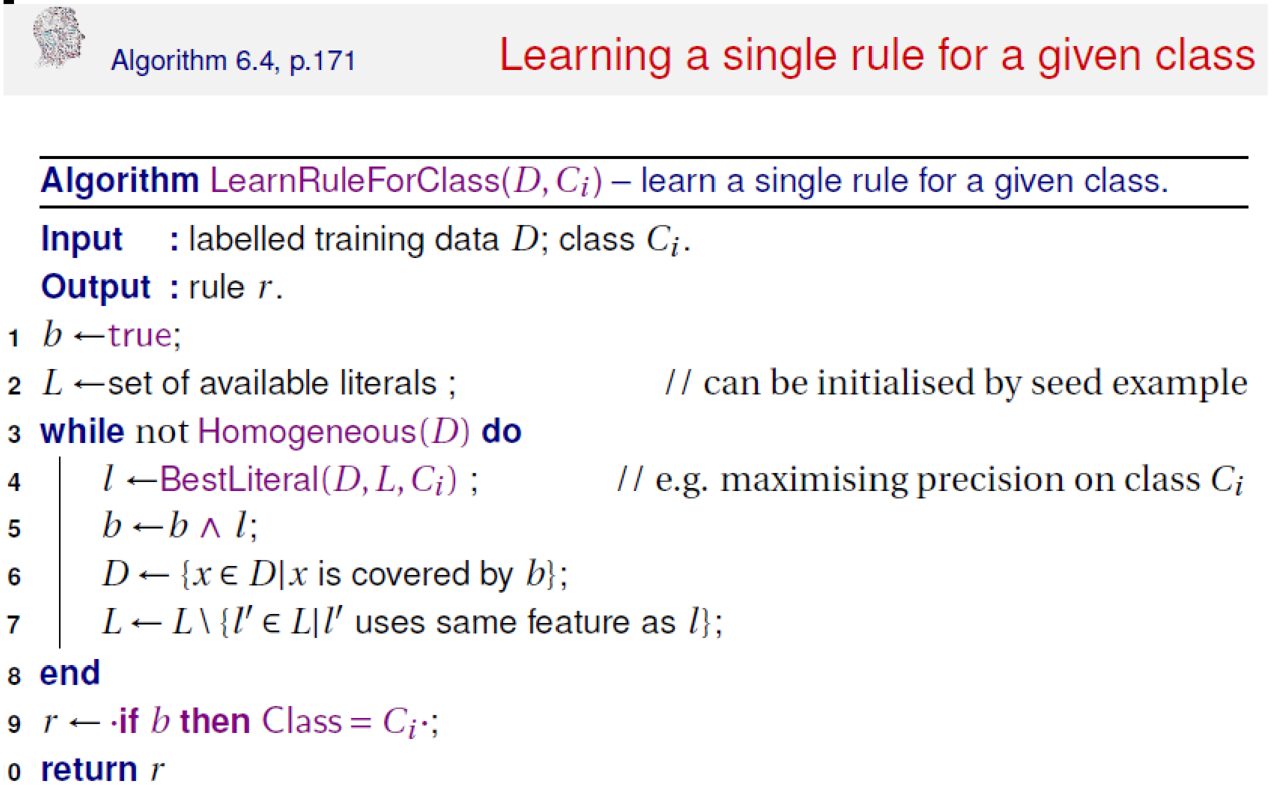
b)Learning ordered rule lists


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 $R$.
$R \leftarrow \emptyset$;
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 $R$
```


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 $r$.
$b \leftarrow$ true;
$L \leftarrow$ set of available literals ;        // can be initialised by seed example
while not Homogeneous($D$) do
    $l \leftarrow$ BestLiteral($D, L, C_i$) ;     // e.g. maximising precision on class $C_i$
    $b \leftarrow b \wedge l$;
    $D \leftarrow \{x \in D | x$ is covered by $b\}$;
    $L \leftarrow L \setminus \{l' \in L | l'$ uses same feature as $l\}$;
end
$r \leftarrow \cdot$if $b$ then Class $= C_i \cdot$;
return $r$
```

c)Descriptive rule learning

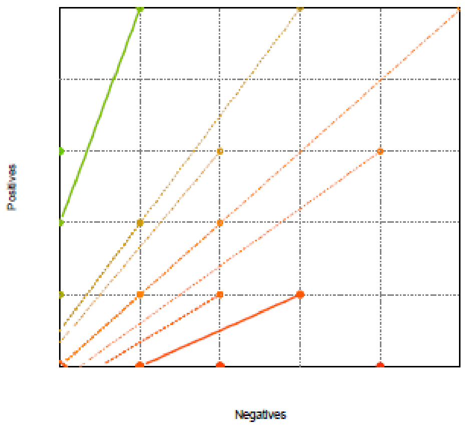
i)Rule Learning for subgroup discovery

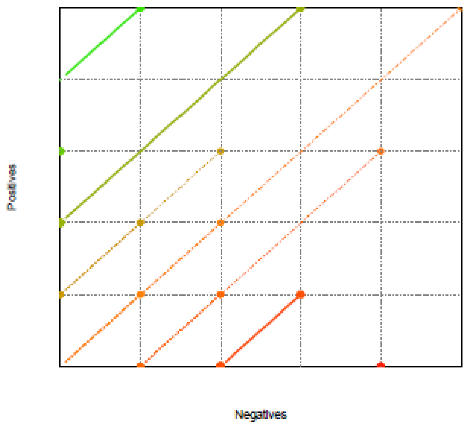
Subgroups are subsets of the instance space – or alternatively, mappings $\hat{g} : \mathscr{X} \to \{\text{true}, \text{false}\}$ – that are learned from a set of labelled examples $(x_i, l(x_i))$, where $l : \mathscr{X} \to \mathscr{C}$ is the true labelling function.

☞ A good subgroup is one whose class distribution is significantly different from the overall population. This is by definition true for pure subgroups, but these are not the only interesting ones.

☞ For instance, one could argue that the complement of a subgroup is as interesting as the subgroup itself: in our dolphin example, the concept Gills = yes, which covers four negatives and no positives, could be considered as interesting as its complement Gills = no, which covers one negative and all positives.

☞ This means that we need to move away from impurity-based evaluation measures.

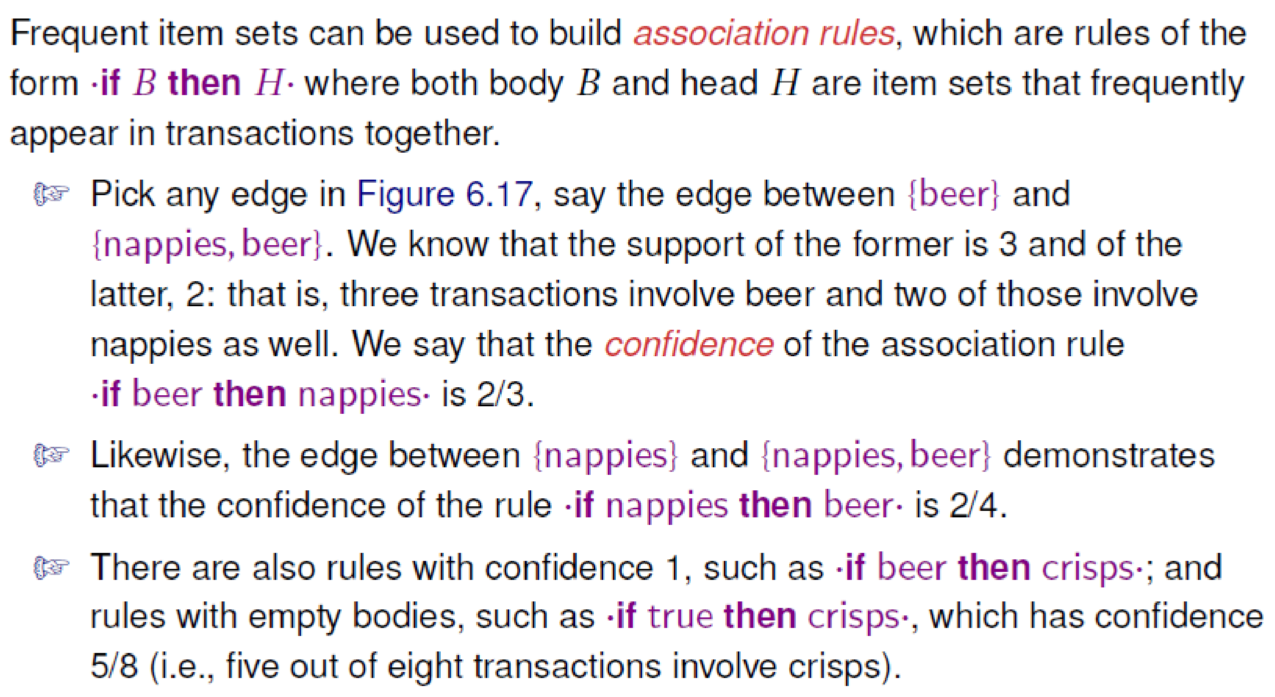




(left) Subgroups and their isometrics according to Laplace-corrected precision. The solid, outermost isometrics indicate the best subgroups. **(right)** The ranking changes if we order the subgroups on average recall. For example, $[5+,1-]$ is now better than $[3+,0-]$ and as good as $[0+,4-]$.

ii) Association rule mining:

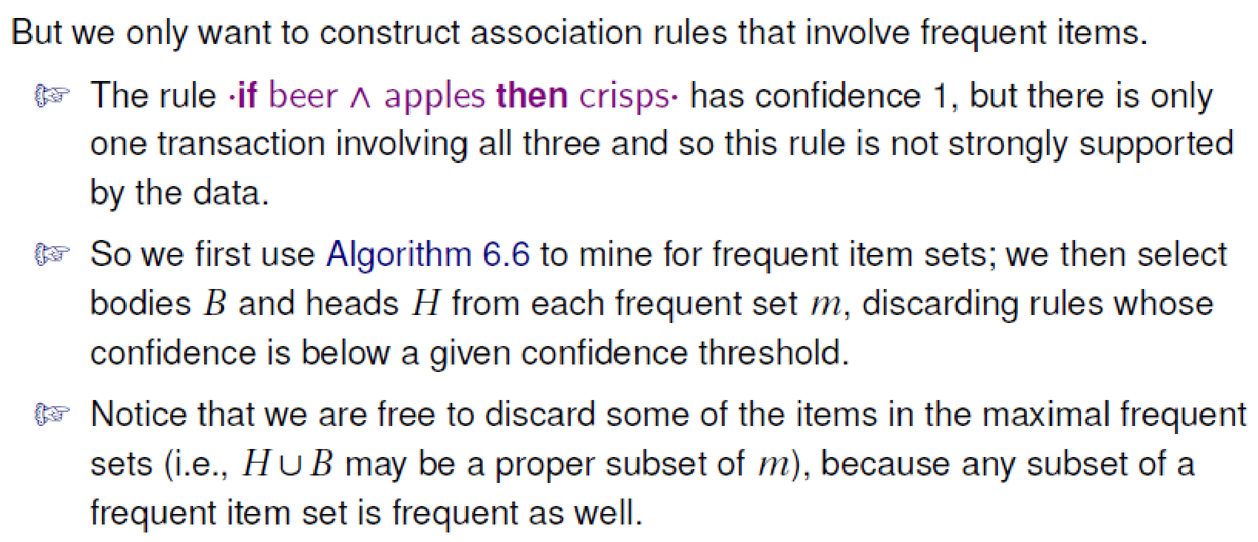
Rule-1



Frequent item sets can be used to build *association rules*, which are rules of the form ·**if** B **then** H· where both body B and head H are item sets that frequently appear in transactions together.

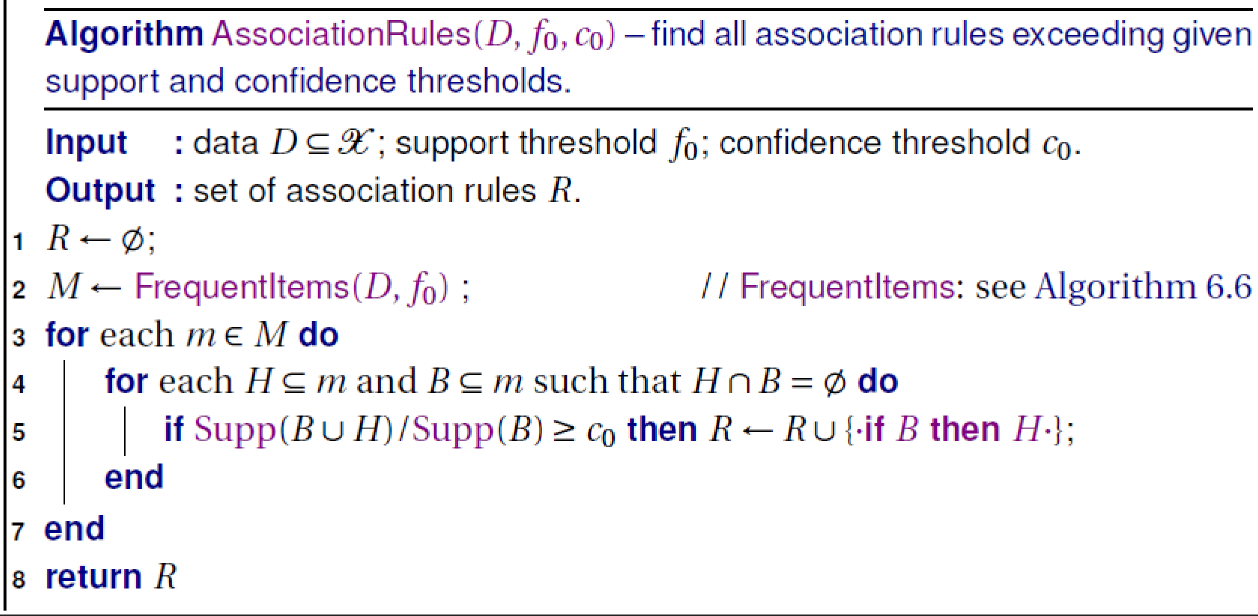
- ☞ Pick any edge in Figure 6.17, say the edge between {beer} and {nappies, beer}. We know that the support of the former is 3 and of the latter, 2: that is, three transactions involve beer and two of those involve nappies as well. We say that the *confidence* of the association rule ·**if** beer **then** nappies· is 2/3.
- ☞ Likewise, the edge between {nappies} and {nappies, beer} demonstrates that the confidence of the rule ·**if** nappies **then** beer· is 2/4.
- ☞ There are also rules with confidence 1, such as ·**if** beer **then** crisps·; and rules with empty bodies, such as ·**if** true **then** crisps·, which has confidence 5/8 (i.e., five out of eight transactions involve crisps).

Rule-2



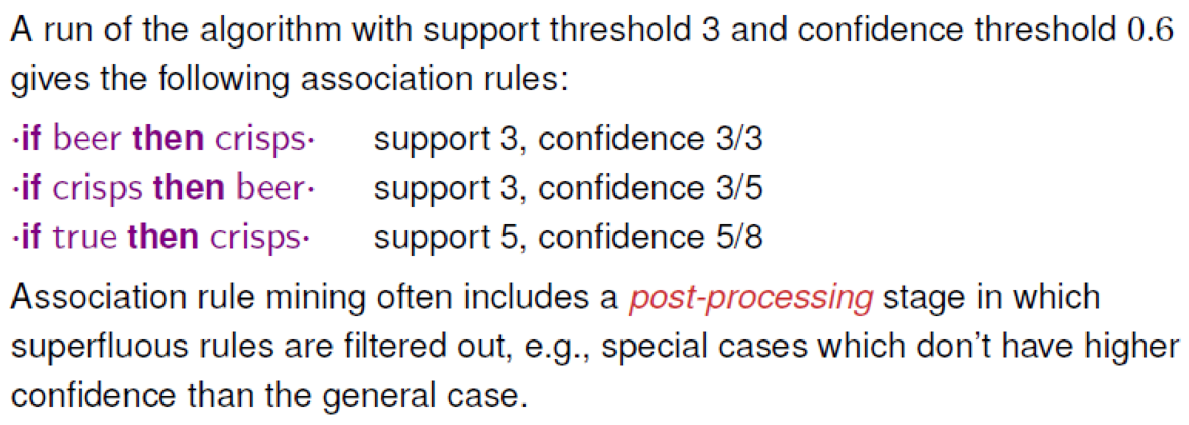
But we only want to construct association rules that involve frequent items.

- ☞ The rule ·**if** beer $\wedge$ apples **then** crisps· has confidence 1, but there is only one transaction involving all three and so this rule is not strongly supported by the data.
- ☞ So we first use Algorithm 6.6 to mine for frequent item sets; we then select bodies B and heads H from each frequent set m, discarding rules whose confidence is below a given confidence threshold.
- ☞ Notice that we are free to discard some of the items in the maximal frequent sets (i.e., $H \cup B$ may be a proper subset of m), because any subset of a frequent item set is frequent as well.



```
Algorithm AssociationRules($D, f_0, c_0$) – find all association rules exceeding given
support and confidence thresholds.
Input  : data $D \subseteq \mathscr{X}$; support threshold $f_0$; confidence threshold $c_0$.
Output : set of association rules $R$.
$R \leftarrow \emptyset$;
$M \leftarrow$ FrequentItems($D, f_0$) ;               // FrequentItems: see Algorithm 6.6
for each $m \in M$ do
    for each $H \subseteq m$ and $B \subseteq m$ such that $H \cap B = \emptyset$ do
        if Supp($B \cup H$)/Supp($B$) $\geq c_0$ then $R \leftarrow R \cup \{$·if $B$ then $H$·$\}$;
    end
end
return $R$
```

Association rule example



A run of the algorithm with support threshold 3 and confidence threshold 0.6 gives the following association rules:

·if beer then crisps·	support 3, confidence 3/3
·if crisps then beer·	support 3, confidence 3/5
·if true then crisps·	support 5, confidence 5/8

Association rule mining often includes a *post-processing* stage in which superfluous rules are filtered out, e.g., special cases which don't have higher confidence than the general case.

d) First-order rule learning

→One of the most *expressive* and *human readable* representations for learned hypotheses is sets of *production rules* (*if-then* rules).

→Rules can be derived from other representations (e.g., decision trees) or they can be learned *directly*. Here, we are concentrating on the direct method.

→An important aspect of direct rule-learning algorithms is that they can learn sets of *first-order rules* which have much more representational power than the *propositional* rules that can be derived from decision trees. Learning first-order rules can also be seen as automatically inferring *PROLOG programs* from example.

Propositional versus First-Order Logic:

→*Propositional Logic* does not include variables and thus cannot express general relations among the values of the attributes.

Example 1: in Propositional logic, you can write: *IF (Father$_1$=Bob) ^ (Name$_2$=Bob)^ (Female$_1$=True) THEN Daughter$_{1,2}$=True.*

This rule applies only to a specific family!

Example 2: In First-Order logic, you can write

IF Father(y,x) ^ Female(y), THEN Daughter(x,y)

This rule (which you cannot write in Propositional Logic) applies to any family!

Learning Propositional versus First-Order Rules:

→Both approaches to learning are useful as they address different types of learning problems.

→Like Decision Trees, Feed forward Neural Nets and IBL systems, Propositional Rule Learning systems are suited for problems in which no substantial relationship between the values of the different attributes needs to be represented.

→In First-Order Learning Problems, the hypotheses that must be represented involve relational assertions that can be conveniently expressed using first-order representations such as ***horn clauses*** (**$H \leftarrow L_1 \wedge \ldots \wedge L_n$**