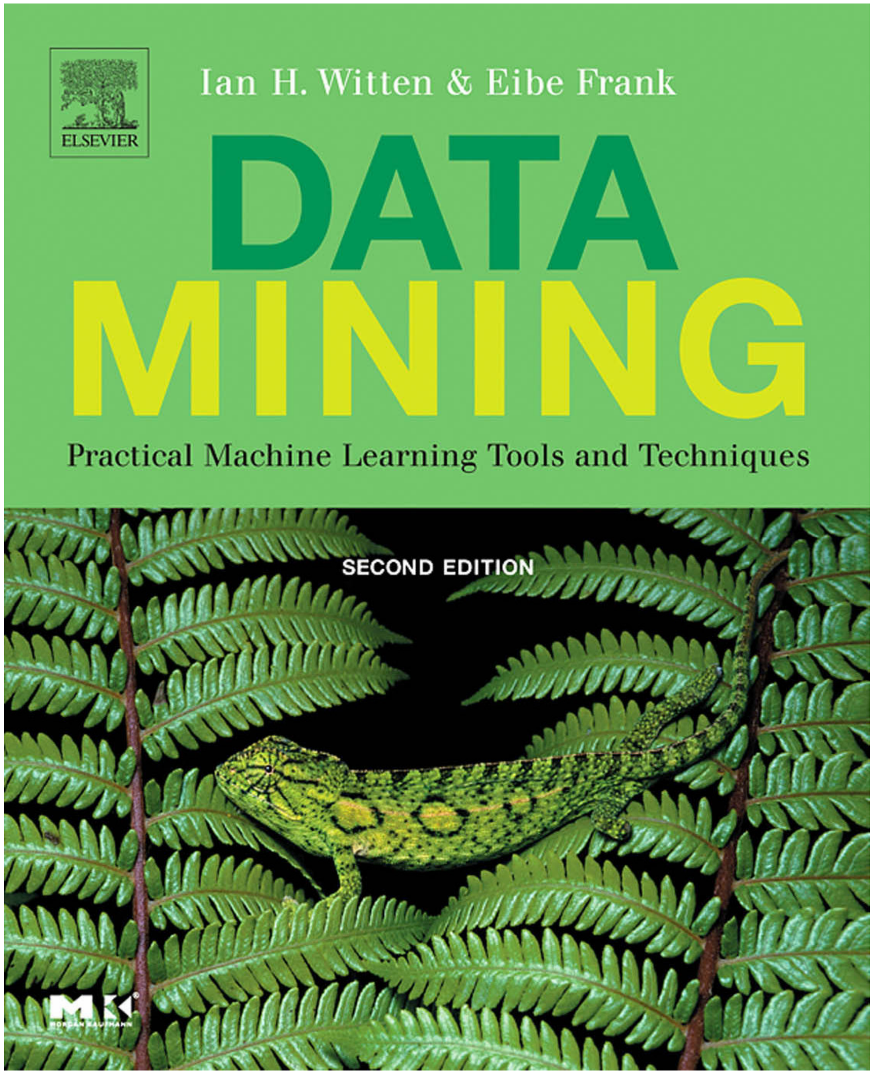


Chapter 3: Output: Knowledge Representation

Output: representing structural patterns

- Many different ways of representing patterns
  - Decision trees, classification rules, …
- Also called "knowledge" representation
- Representation determines inference method
- Understanding the output is the key to understanding the underlying learning methods
- Different types of output for different learning problems (e.g. classification, regression, …)

Output: Knowledge representation

- Decision tables
- Decision trees
- Classification rules
- Association rules
- Rules with exceptions
- Rules involving relations
- Trees for numeric prediction
- Instance-based representation
- Clusters

3.1 Decision tables

Decision tables

- Simplest way of representing output
- Use the same format as input!
- Main problem: selecting the right attributes

Decision table for the weather problem

Outlook	Temperature	Humidity	Windy	Play
sunny	hot	high	false	no
sunny	hot	high	true	no
overcast	hot	high	false	yes
rainy	mild	high	false	yes
rainy	cool	normal	false	yes
rainy	cool	normal	true	no
overcast	cool	normal	true	yes
sunny	mild	high	false	no
sunny	cool	normal	false	yes
rainy	mild	normal	false	yes
sunny	mild	normal	true	yes
overcast	mild	high	true	yes
overcast	hot	normal	false	yes
rainy	mild	high	true	no

3.2 Decision trees

Decision trees

- Nodes involve testing a particular attribute
- Usually, attribute value is compared to constant
- Other possibilities:
  - Comparing values of two attributes
  - Using a function of one or more attributes
- Unknown instance is routed down the tree
- Leaves assign classification, set of classifications, or probability distribution to instances

Nominal and numeric attributes

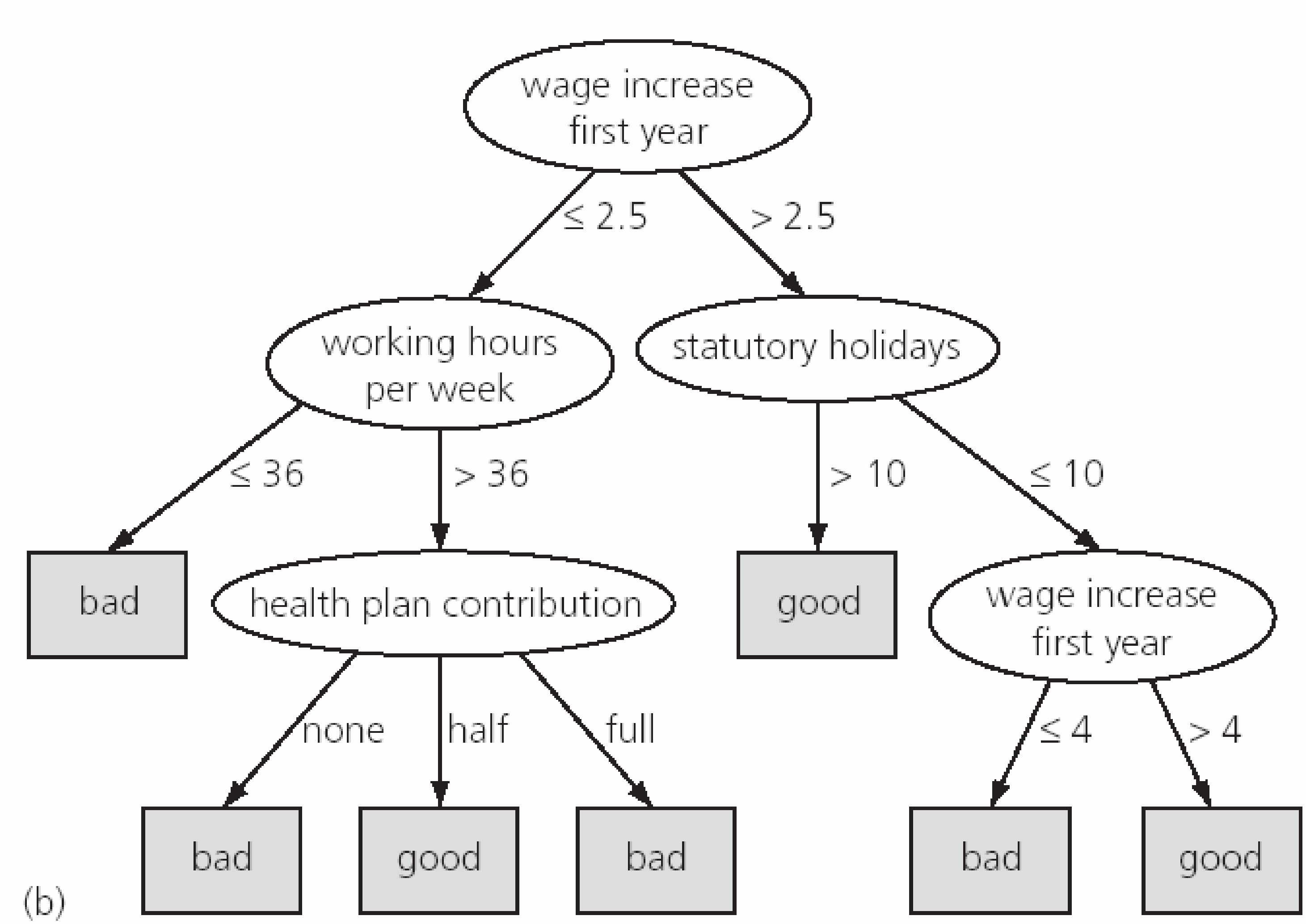
- Nominal:
  - Number of children usually equal to number values
  - Attribute won't get tested more than once
  - Other possibility: division into two subsets

- Numeric:
  - Test whether value is greater or less than constant
  - Attribute may get tested several times
  - Other possibility: three-way split (or multi-way split)
    - Integer: *less than, equal to, greater than*
    - Real: *below, within, above*

Missing values

- Does absence of value have some significance?
- Yes => “missing” is a separate value
- No => “missing” must be treated in a special way
  - E.g. assign instance to most popular branch

Decision tree for the labor data



3.3 Classification rules

Classification rules

- Popular alternative to decision trees
- *Antecedent* (precondition): a series of tests (just like the tests at the nodes of a decision tree)
- Tests are usually logically ANDed together (but may also be general logical expressions)
- *Consequent* (conclusion): set of classes or probability distribution assigned by rule
- Individual rules are often logically ORed together
  - Conflicts arise if different conclusions apply

From trees to rules

- Easy: converting a tree into a set of rules
  - One rule for each leaf:
    - Antecedent contains a condition for every node on the path from the root to the leaf
    - Consequent is class assigned by the leaf
- Produces rules that are very clear
  - Doesn't matter in which order they are executed
- But: resulting rules are unnecessarily complex
  - It needs to remove redundant tests/rules

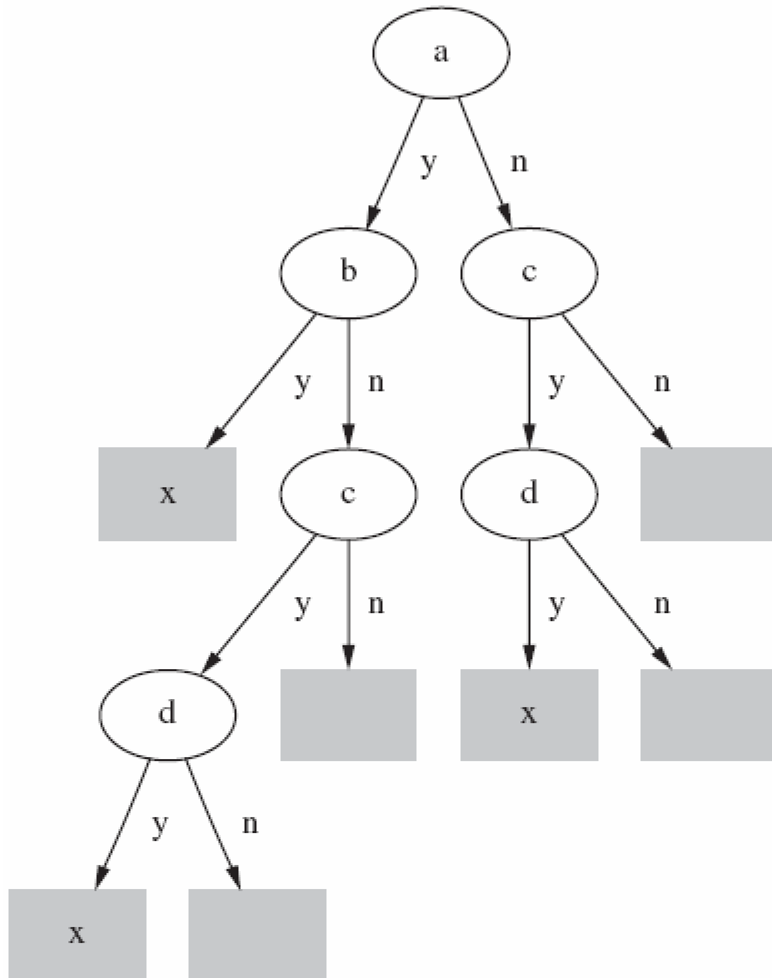
From rules to trees

- More difficult: transforming a rule set into a tree
  - Tree cannot easily express disjunction between rules
- Example: rules which test different attributes

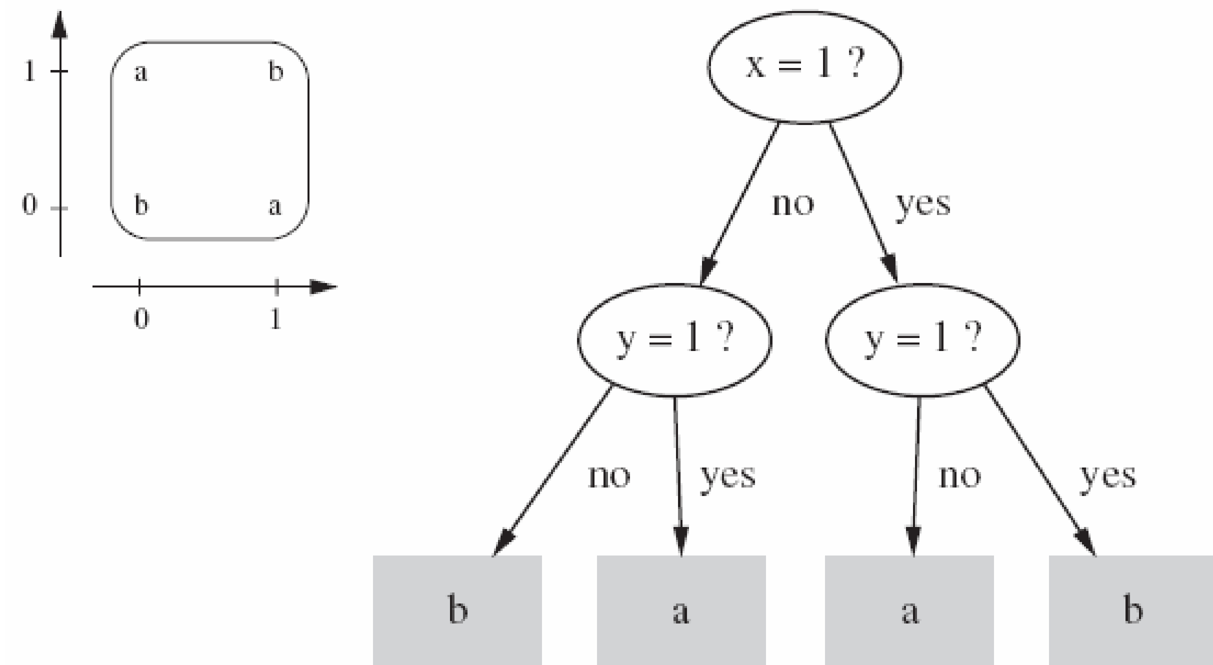
```
If a and b then x
If c and d then x
```

- Corresponding tree contains identical subtrees => “replicated subtree problem”

A tree for a simple disjunction



The exclusive-or problem



If x=1 and y=0 then class = a

If x=0 and y=1 then class = a

If x=0 and y=0 then class = b

If x=1 and y=1 then class = b

A tree with a replicated subtree

- there are four attributes, *x, y, z,* and *w,*
- each can be 1, 2, or 3

If x=1 and y=1 then class = a

If z=1 and w=1 then class = a

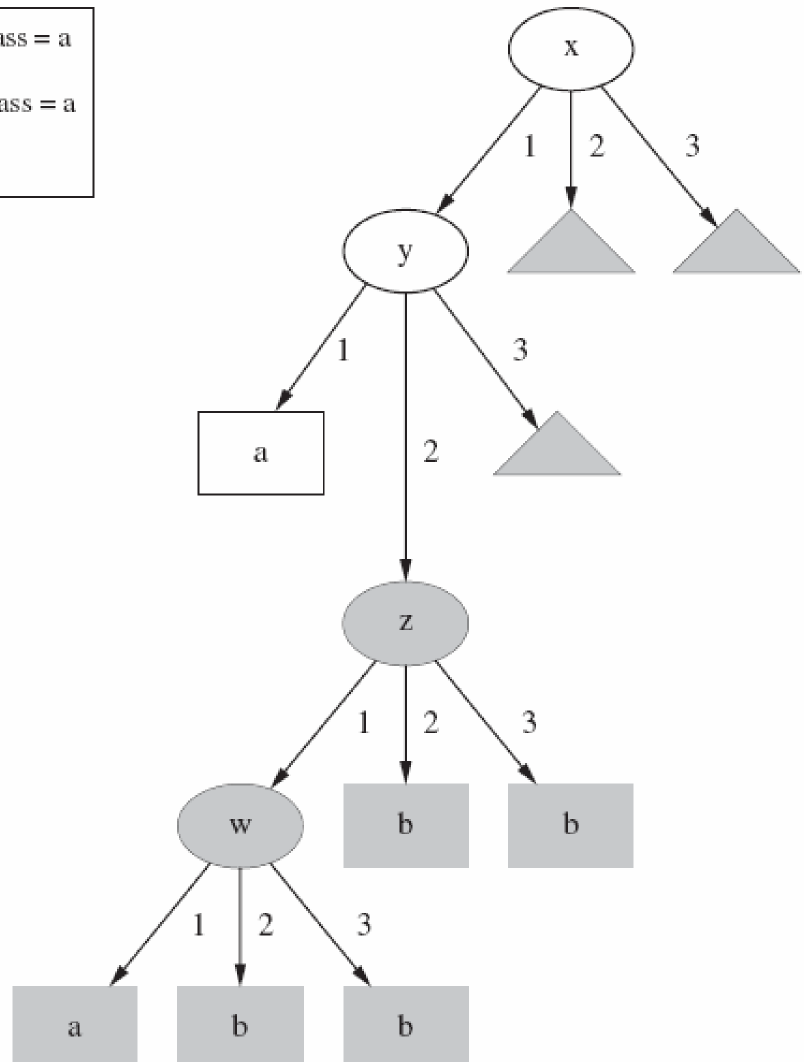
Otherwise class = b

A tree with a replicated subtree (II)

If x=1 and y=1 then class = a

If z=1 and w=1 then class = a

Otherwise class = b



Nuggets of knowledge

- Are rules independent pieces of knowledge? (It seems easy to add a rule to an existing rule base.)
- Problem: ignores how rules are executed
- Two ways of executing a rule set:
  - Ordered set of rules (“decision list”)
    - Order is important for interpretation
  - Unordered set of rules
    - Rules may overlap and lead to different conclusions for the same instance

Interpreting rules

- What if two or more rules conflict?
  - Give no conclusion at all?
  - Go with rule that is most popular on training data?
  - …
- What if no rule applies to a test instance?
  - Give no conclusion at all?
  - Go with class that is most frequent in training data?
  - …

Special case: boolean class

- Assumption: if instance does not belong to class “yes”, it belongs to class “no”
- Trick: only learn rules for class “yes” and use default rule for “no”

```
If x=1 and y=1 then class = a

If z=1 and w=1 then class = a

Otherwise class = b
```

- Order of rules is not important. No conflicts!
- Rule can be written in *disjunctive normal form*

3.4 Association rules

Association rules

- Association rules…
  - … can predict any attribute and combinations of attributes
  - … are not intended to be used together as a set
- Large number of possible associations
  - Output needs to be restricted to show only the most predictive associations
  - only those with high *support* and high *confidence*

Support and confidence of a rule

- Support: number of instances predicted correctly
- Confidence: number of correct predictions, as proportion of all instances that rule applies to
- Example: 4 cool days with normal humidity

```
If temperature = cool then humidity = normal
```

- Support = 4, confidence = 100%
- Normally: minimum support and confidence prespecified (e.g. support >= 2 and confidence >= 95% for weather data)

Interpreting association rules

- Interpretation is not obvious:

```
If windy = false and play = no then outlook = sunny
                                and humidity = high
```

- is *not* the same as

```
If windy = false and play = no then outlook = sunny
If windy = false and play = no then humidity = high
```

- It means that the following also holds:

```
If humidity = high and windy = false and play = no
   then outlook = sunny
```

3.5 Rules with exceptions

Rules with exceptions

- Idea: allow rules to have *exceptions*
- Example: rule for iris data

```
If petal length ≥ 2.45 and petal length < 4.45 then Iris versicolor
```

- New instance:

Sepal length (cm)	Sepal width (cm)	Petal length (cm)	Petal width (cm)	Type
5.1	3.5	2.6	0.2	*Iris setosa*

- Modified rule:

```
If petal length ≥ 2.45 and petal length < 4.45 then
   Iris versicolor EXCEPT if petal width < 1.0 then Iris setosa
```

A more complex example

- Exceptions to exceptions to exceptions …

```
Default: Iris-setosa
except if petal-length ≥ 2.45 and petal-length < 5.355
          and petal-width < 1.75
       then Iris-versicolor
            except if petal-length ≥ 4.95 and petal-width < 1.55
                   then Iris-virginica
                   else if sepal-length < 4.95 and sepal-width ≥ 2.45
                        then Iris-virginica
       else if petal-length ≥ 3.35
            then Iris-virginica
                 except if petal-length < 4.85 and sepal-length < 5.95
                        then Iris-versicolor
```

Advantages of using exceptions

- Rules can be updated incrementally
  - Easy to incorporate new data
  - Easy to incorporate domain knowledge
- People often think in terms of exceptions
- Each conclusion can be considered just in the context of rules and exceptions that lead to it
  - Locality property is important for understanding large rule sets
  - "Normal" rule sets don't offer this advantage

More on exceptions

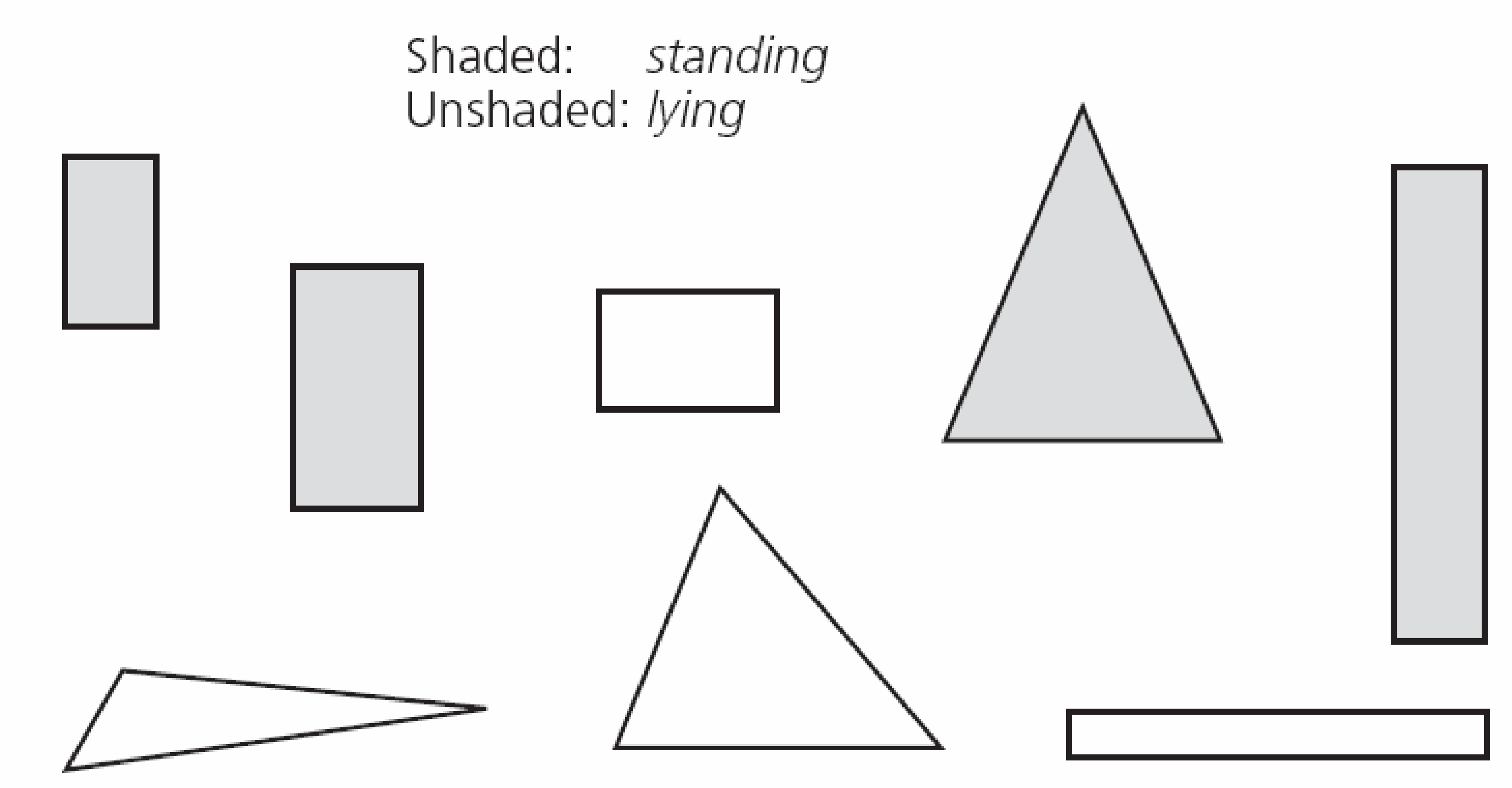
- Default...except if...then...
  is logically equivalent to
- if...then...else
  (where the else specifies what the default did)
- But: exceptions offer a psychological advantage
  - Assumption: defaults and tests early on apply more widely than exceptions further down
  - Exceptions reflect special cases

3.6 Rules involving relations

Rules involving relations

- So far: all rules involved comparing an attribute value to a constant
(e.g. temperature < 45)
- These rules are called "propositional"
- What if problem involves relationships between examples
(e.g. family tree problem from above)?
  - Can't be expressed with propositional rules
  - More expressive representation required

The shapes problem



A propositional solution

Width	Height	Sides	Class
2	4	4	standing
3	6	4	standing
4	3	4	lying
7	8	3	standing
7	6	3	lying
2	9	4	standing
9	1	4	lying
10	2	3	lying

```
if width ≥ 3.5 and height < 7.0 then lying
if height ≥ 3.5 then standing
```

A relational solution

- Comparing attributes with each other

```
if width > height then lying
if height > width then standing
```

- Rules of this form is called relational
- Generalizes better to new data
- Standard relations: =, <, >
- Simple solution: add extra attributes
  (e.g. a binary attribute *is width < height?*)

3.7 Trees for numeric prediction

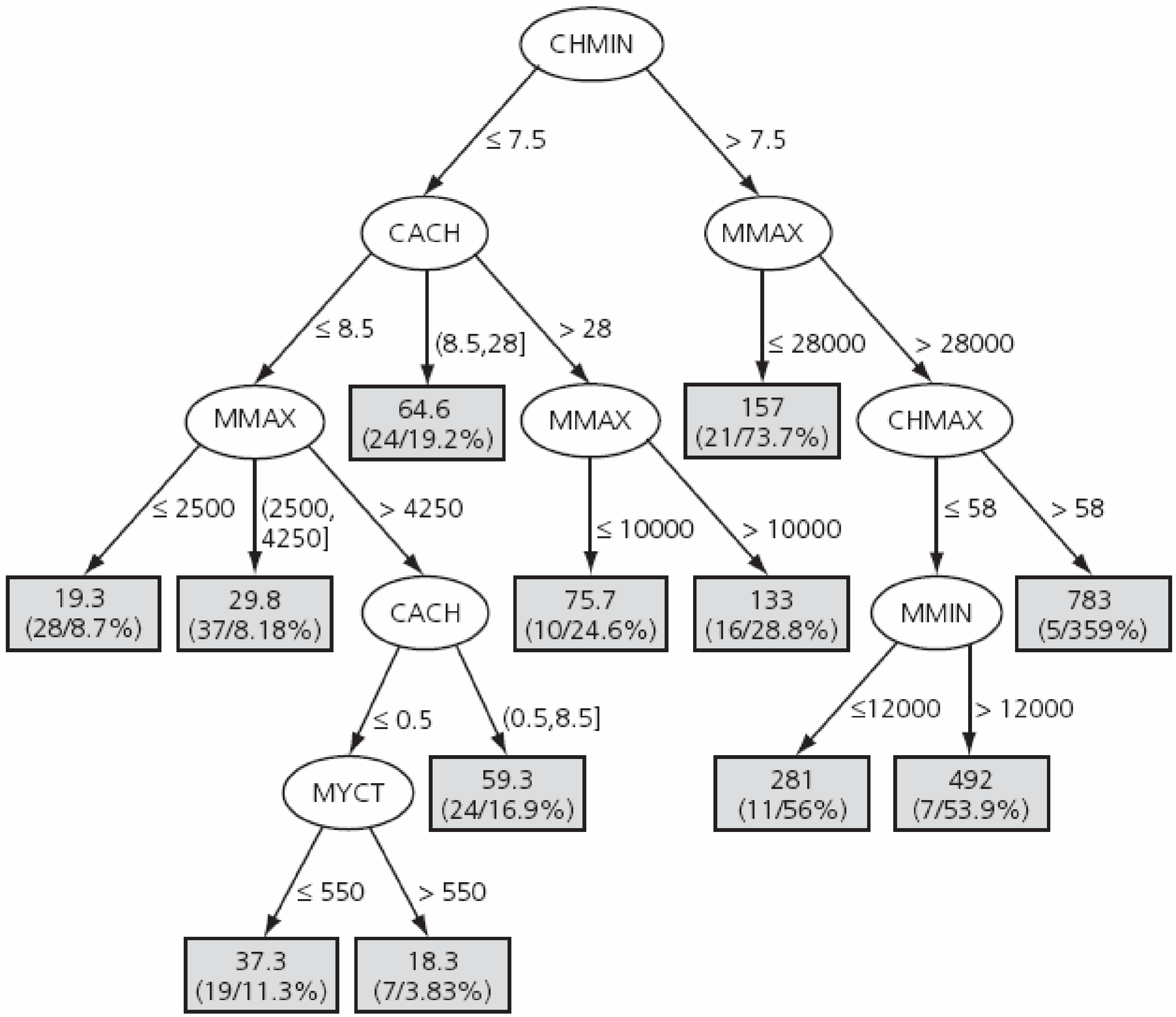
Trees for numeric prediction

- *Regression*: the process of computing an expression that predicts a numeric quantity
- *Regression tree*: “decision tree” where each leaf predicts a numeric quantity
  - Predicted value is average value of training instances that reach the leaf
- *Model tree:* “regression tree” with linear regression models at the leaf nodes

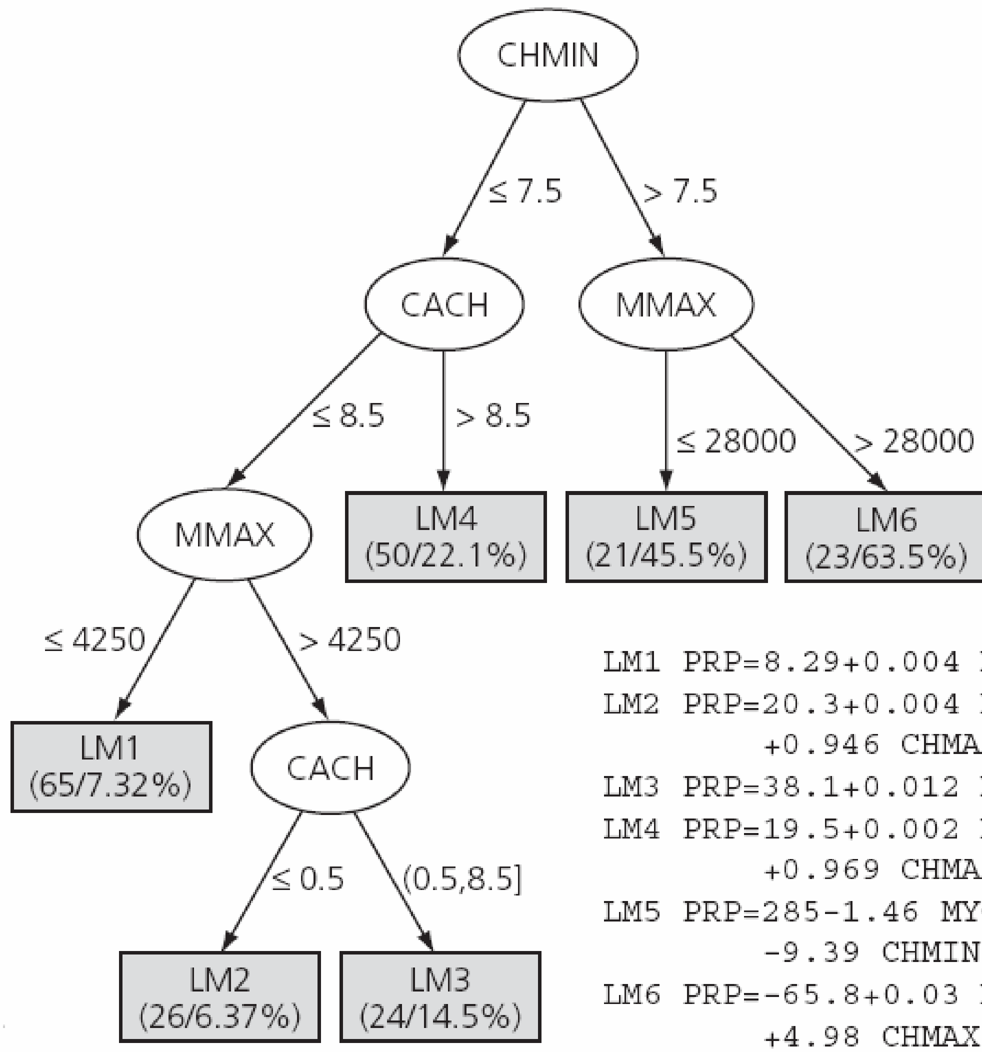
Linear regression for the CPU data

```
PRP =
  -56.1
  +0.049 MYCT
  +0.015 MMIN
  +0.006 MMAX
  +0.630 CACH
  -0.270 CHMIN
  +1.46 CHMAX
```

Regression tree for the CPU data



Model tree for the CPU data




```
LM1 PRP=8.29+0.004 MMAX+2.77 CHMIN
LM2 PRP=20.3+0.004 MMIN-3.99 CHMIN
        +0.946 CHMAX
LM3 PRP=38.1+0.012 MMIN
LM4 PRP=19.5+0.002 MMAX+0.698 CACH
        +0.969 CHMAX
LM5 PRP=285-1.46 MYCT+1.02 CACH
        -9.39 CHMIN
LM6 PRP=-65.8+0.03 MMIN-2.94 CHMIN
        +4.98 CHMAX
```

3.8 Instance-based representation

Instance-based representation

- Simplest form of learning: *rote learning*
  - Training instances are searched for instance that most closely similar to new instance
  - The instances themselves represent the knowledge
  - Also called *instance-based* learning
- Instance-based learning is *lazy* learning
- Similarity function defines what's "learned"

Instance-based learning methods

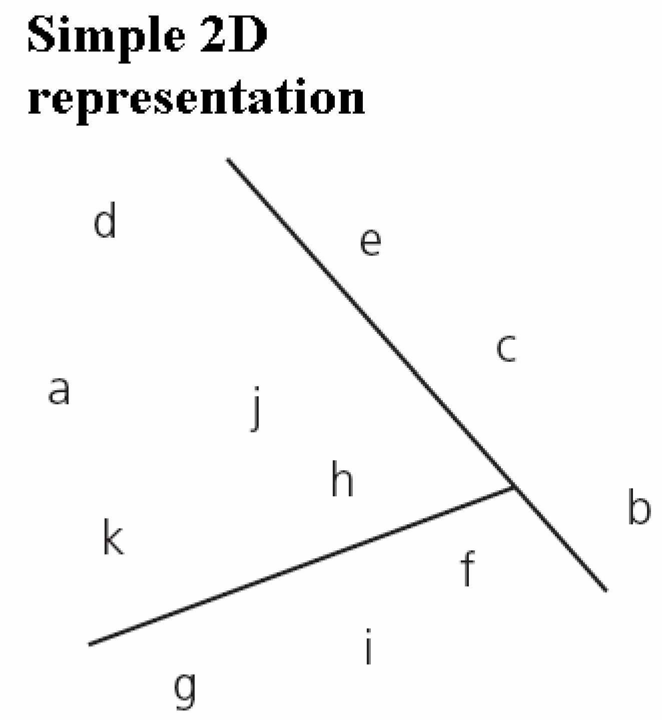
- *Nearest-neighbor* method:
  - each new instance is compared with existing ones using a distance metric, and the closest existing instance is used to assign the class to the new one
- *k-nearest-neighbor* method:
  - more than one nearest neighbor is used, and the majority class of the closest *k* neighbors

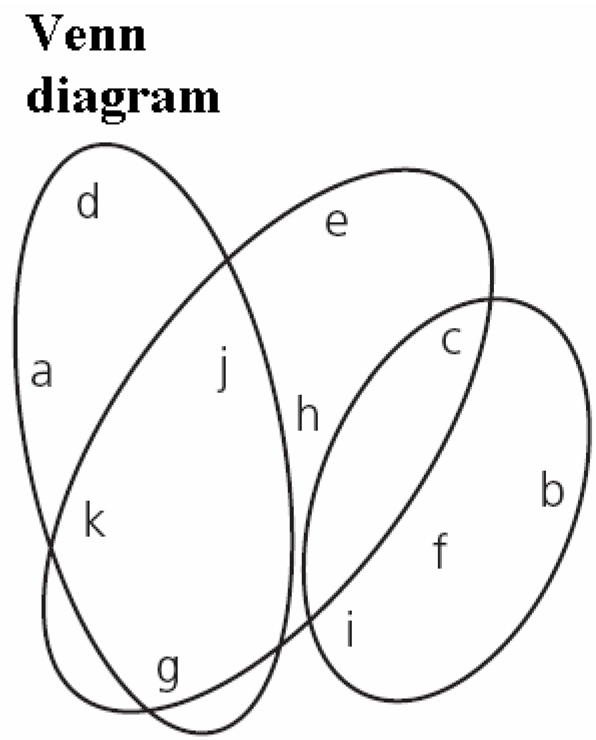
3.9 Clusters

Clusters

- The output takes the form of a diagram that shows how the instances fall into clusters.
- Different cases:
  - ***Simple 2D representation:*** involves associating a cluster number with each instance
  - ***Venn diagram:*** allow one instance to belong to more than one cluster
  - ***Probabilistic assignment:*** associate instances with clusters probabilistically
  - ***Dendrogram:*** produces a hierarchical structure of clusters (dendron is the Greek word for tree)

Representing clusters (I)



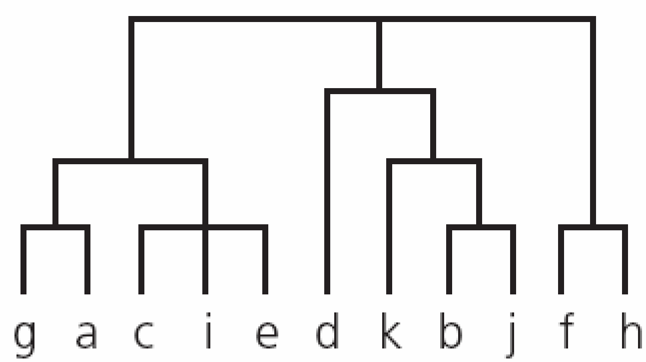


Representing clusters (II)

Probabilistic assignment

	1	2	3
a	0.4	0.1	0.5
b	0.1	0.8	0.1
c	0.3	0.3	0.4
d	0.1	0.1	0.8
e	0.4	0.2	0.4
f	0.1	0.4	0.5
g	0.7	0.2	0.1
h	0.5	0.4	0.1

Dendrogram



The end of

Chapter 3: Output: Knowledge Representation

The distance function

- Simplest case: one numeric attribute
  - Distance is the difference between the two attribute values involved
- Several numeric attributes: normally, Euclidean distance is used and attributes are normalized
- Nominal attributes: distance is set to 1 if values are different, 0 if they are equal
  - E.g. distances between the values *red, green,* and *blue?*
  - the distance between *red* and *red* is zero and between *red* and *green* is one.
- Are all attributes equally important?
  - Weighting the attributes might be necessary