Machine Learning Techniques for Data Mining

Eibe Frank
University of Waikato
New Zealand

PART II

Input: Concepts, instances, attributes

Preparing for learning

- Components of the input:
  - Concepts: kinds of things that can be learned
    - Aim: intelligible and operational concept description
  - Instances: the individual, independent examples of a concept
    - Note: more complicated forms of input are possible
  - Attributes: measuring aspects of an instance
    - We will focus on nominal and numeric ones
- Practical issue: a file format for the input

What's a concept?

- Styles of learning:
  - Classification learning: predicting a discrete class
  - Association learning: detecting associations between features
  - Clustering: grouping similar instances into clusters
  - Numeric prediction: predicting a numeric quantity
- Concept: thing to be learned
- Concept description: output of learning scheme

Classification learning

- Example problems: weather data, contact lenses, irises, labor negotiations
- Classification learning is *supervised*
  - Scheme is being provided with actual outcome
- Outcome is called the *class* of the example
- Success can be measured on fresh data for which class labels are known (*test data*)
- In practice success is often measured subjectively

Association learning

- Can be applied if no class is specified and any kind of structure is considered “interesting”
- Difference to classification learning:
  - Can predict any attribute’s value, not just the class, and more than one attribute’s value at a time
  - Hence: far more association rules than classification rules
  - Thus: constraints are necessary
    - Minimum coverage and minimum accuracy

Clustering

- Finding groups of items that are similar
- Clustering is *unsupervised*
  - The class of an example is not known
- Success of clustering often measured subjectively
- Example problem: iris data without class

	Sepal length	Sepal width	Petal length	Petal width
1	5.1	3.5	1.4	0.2
2	4.9	3.0	1.4	0.2
…				

Numeric prediction

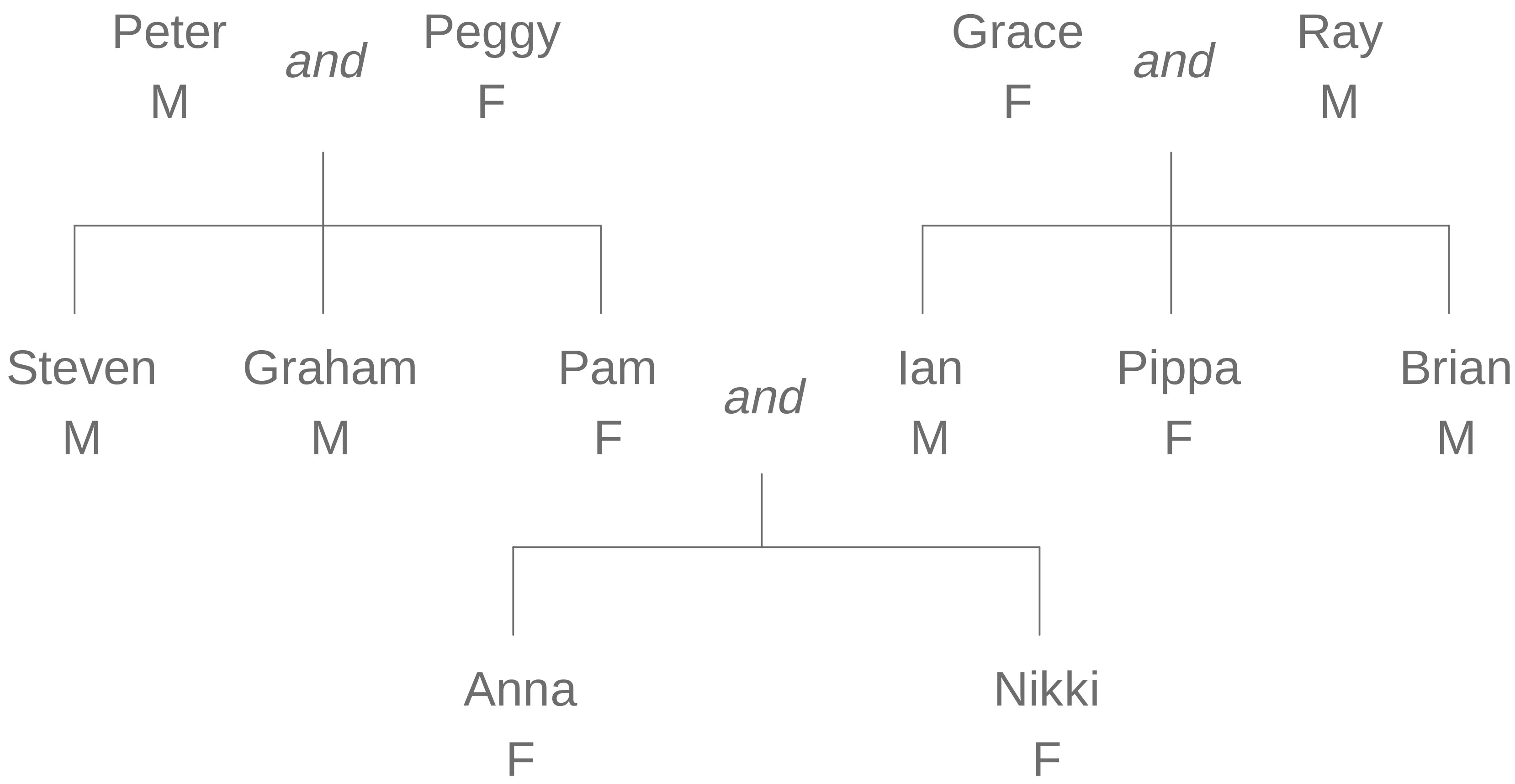
- Like classification learning but with numeric “class”
- Learning is supervised
  - Scheme is being provided with target value
- Success is measured on test data (or subjectively if concept description is intelligible)
- Example: modified version of weather data

Outlook	Temperature	Humidity	Windy	Play-time
Sunny	85	85	False	5
Sunny	80	90	True	0
…	…	…	…	…

What’s in an example?

- Instance: specific type of example
  - Thing to be classified, associated, or clustered
  - Individual, independent example of target concept
  - Characterized by a predetermined set of attributes
- Input to learning scheme: set of instances/dataset
  - Represented as a single relation/flat file
- Rather restricted form of input
  - No relationships between objects
- Most common form in practical data mining

A family tree



Family tree represented as a table

Name	Gender	Parent1	parent2
Peter	Male	?	?
Peggy	Female	?	?
Steven	Male	Peter	Peggy
Graham	Male	Peter	Peggy
Pam	Female	Peter	Peggy
Ian	Male	Grace	Ray
Pippa	Female	Grace	Ray
Brian	Male	Grace	Ray
Anna	Female	Pam	Ian
Nikki	Female	Pam	Ian

The “sister-of” relation

First person	Second person	Sister of?
Peter	Peggy	No
Peter	Steven	No
…	…	…
Steven	Peter	No
Steven	Graham	No
Steven	Pam	Yes
…	…	…
Ian	Pippa	Yes
…	…	…
Anna	Nikki	Yes
…	…	…
Nikki	Anna	yes

First person	Second person	Sister of?
Steven	Pam	Yes
Graham	Pam	Yes
Ian	Pippa	Yes
Brian	Pippa	Yes
Anna	Nikki	Yes
Nikki	Anna	Yes
All the rest		No

Closed-world assumption

A full representation in one table

First person				Second person				Sister of?
Name	Gender	Parent1	Parent2	Name	Gender	Parent1	Parent2	
Steven	Male	Peter	Peggy	Pam	Female	Peter	Peggy	Yes
Graham	Male	Peter	Peggy	Pam	Female	Peter	Peggy	Yes
Ian	Male	Grace	Ray	Pippa	Female	Grace	Ray	Yes
Brian	Male	Grace	Ray	Pippa	Female	Grace	Ray	Yes
Anna	Female	Pam	Ian	Nikki	Female	Pam	Ian	Yes
Nikki	Female	Pam	Ian	Anna	Female	Pam	Ian	Yes
All the rest								No

```
If second person's gender = female and
   first person's parent = second person's parent
   then sister-of = yes
```

Generating a flat file

- Process of flattening called “denormalization”
  - Several relations are joined together to make one
- Possible with any finite set of finite relations
- Problematic: relationships without pre-specified number of objects
  - Example: concept of *nuclear-family*
- Denormalization may produce spurious regularities that reflect structure of database
  - Example: “supplier” predicts “supplier address”

The “ancestor-of” relation

First person				Second person				Sister of?
Name	Gender	Parent1	Parent2	Name	Gender	Parent1	Parent2	
Peter	Male	?	?	Steven	Male	Peter	Peggy	Yes
Peter	Male	?	?	Pam	Female	Peter	Peggy	Yes
Peter	Male	?	?	Anna	Female	Pam	Ian	Yes
Peter	Male	?	?	Nikki	Female	Pam	Ian	Yes
Pam	Female	Peter	Peggy	Nikki	Female	Pam	Ian	Yes
Grace	Female	?	?	Ian	Male	Grace	Ray	Yes
Grace	Female	?	?	Nikki	Female	Pam	Ian	Yes
Other positive examples here								Yes
All the rest								No

Recursion

- Infinite relations require recursion

```
If person1 is a parent of person2
  then person1 is an ancestor of person2
If person1 is a parent of person2 and
  and person2 is an ancestor of person3
  then person1 is an ancestor of person3
```

- Appropriate techniques are known as “inductive logic programming” (e.g. Quinlan’s FOIL)
  - Problems: (a) noise and (b) computational complexity

Multi-instance problems

- Each example consists of several instances
- E.g. predicting drug activity
  - Examples are molecules that are active/not active
  - Instances are confirmations of a molecule
  - Molecule active (example positive) ⇨ at least one of its confirmations (instances) is active (positive)
  - Molecule not active (example negative) ⇨ all of its confirmations (instances) are not active (negative)
- Problem: identifying the “truly” positive instances

What's in an attribute?

- Each instance is described by a fixed predefined set of features, its "attributes"
- But: number of attributes may vary in practice
  - Possible solution: "irrelevant value" flag
- Related problem: existence of an attribute may depend of value of another one
- Possible attribute types ("levels of measurement"):
  - *Nominal*, *ordinal*, *interval* and *ratio*

Nominal quantities

- Values are distinct symbols
  - Values themselves serve only as labels or names
  - *Nominal* comes from the Latin word for name
- Example: attribute “outlook” from weather data
  - Values: “sunny”,”overcast”, and “rainy”
- No relation is implied among nominal values (no ordering or distance measure)
- Only equality tests can be performed

Ordinal quantities

- Impose order on values
- But: no distance between values defined
- Example: attribute “temperature” in weather data
  - Values: “hot” > “mild” > “cool”
- Note: addition and subtraction don’t make sense
- Example rule: temperature < hot ⇨ play = yes
- Distinction between nominal and ordinal not always clear (e.g. attribute “outlook”)

Interval quantities

- Interval quantities are not only ordered but measured in fixed and equal units
- Example 1: attribute “temperature” expressed in degrees Fahrenheit
- Example 2: attribute “year”
- Difference of two values makes sense
- Sum or product doesn’t make sense
  - Zero point is not defined!

Ratio quantities

- Ratio quantities are ones for which the measurement scheme defines a zero point
- Example: attribute "distance"
  - Distance between an object and itself is zero
- Ratio quantities are treated as real numbers
  - All mathematical operations are allowed
- But: is there an "inherently" defined zero point?
  - Answer depends on scientific knowledge (e.g. Fahrenheit knew no lower limit to temperature)

Attribute types used in practice

- Most schemes accommodate just two levels of measurement: nominal and ordinal
- Nominal attributes are also called “categorical”, ”enumerated”, or “discrete”
  - But: “enumerated” and “discrete” imply order
- Special case: dichotomy (“boolean” attribute)
- Ordinal attributes are called “numeric”, or “continuous”
  - But: “continuous” implies mathematical continuity

Transforming ordinal to boolean

- Simple transformation allows to code ordinal attribute with *n* values using *n-1* boolean attributes
- Example: attribute "temperature"

Original data

Temperature
Cold
Medium
Hot

→

Transformed data

Temperature > cold	Temperature > medium
False	False
True	False
True	True

- Better than coding it as a nominal attribute

Metadata

- Information about the data that encodes background knowledge
- Can be used to restrict search space
- Examples:
  - Dimensional considerations (i.e. expressions must be dimensionally correct)
  - Circular orderings (e.g. degrees in compass)
  - Partial orderings (e.g. generalization/specialization relations)

Preparing the input

- Denormalization is not the only issue
- Problem: different data sources (e.g. sales department, customer billing department, …)
  - Differences: styles of record keeping, conventions, time periods, data aggregation, primary keys, errors
  - Data must be assembled, integrated, cleaned up
  - “Data warehouse”: consistent point of access
- External data may be required (“overlay data”)
- Critical: type and level of data aggregation

The ARFF format

```
%
% ARFF file for weather data with some numeric features
%
@relation weather

@attribute outlook {sunny, overcast, rainy}
@attribute temperature numeric
@attribute humidity numeric
@attribute windy {true, false}
@attribute play? {yes, no}

@data
sunny, 85, 85, false, no
sunny, 80, 90, true, no
overcast, 83, 86, false, yes
...
```

Attribute types

- ARFF supports numeric and nominal attributes
- Interpretation depends on learning scheme
  - Numeric attributes are interpreted as
    - ordinal scales if less-than and greater-than are used
    - ratio scales if distance calculations are performed (normalization/standardization may be required)
  - Instance-based schemes define distance between nominal values (0 if values are equal, 1 otherwise)
- Integers: nominal, ordinal, or ratio scale?

Nominal vs. ordinal

- Attribute “age” nominal

```
If age = young and astigmatic = no and
  tear production rate = normal then recommendation = soft
If age = pre-presbyopic and astigmatic = no and
  tear production rate = normal then recommendation = soft
```

- Attribute “age” ordinal
  (e.g. “young” < “pre-presbyopic” < “presbyopic”)

```
If age ≤ pre-presbyopic and astigmatic = no and
  tear production rate = normal then recommendation = soft
```

Missing values

- Frequently indicated by out-of-range entries
  - Types: unknown, unrecorded, irrelevant
  - Reasons: malfunctioning equipment, changes in experimental design, collation of different datasets, measurement not possible
- Missing value may have significance in itself (e.g. missing test in a medical examination)
  - Most schemes assume that is not the case $\Rightarrow$ "missing" may need to be coded as additional value

Inaccurate values

- Reason: data has not been collected for mining it
- Result: errors and omissions that don't affect original purpose of data (e.g. age of customer)
- Typographical errors in nominal attributes $\Rightarrow$ values need to be checked for consistency
- Typographical and measurement errors in numeric attributes $\Rightarrow$ outliers need to be identified
- Errors may be deliberate (e.g. wrong zip codes)
- Other problems: duplicates, stale data

Getting to know the data

- Simple visualization tools are very useful for identifying problems
  - Nominal attributes: histograms (Distribution consistent with background knowledge?)
  - Numeric attributes: graphs (Any obvious outliers?)
- 2-D and 3-D visualizations show dependencies
- Domain experts need to be consulted
- Too much data to inspect? Take a sample!

Machine Learning Techniques for Data Mining

Eibe Frank
University of Waikato
New Zealand

PART III

Output: Knowledge representation

Representing structural patterns

- Many different ways of representing patterns
  - Decision trees, rules, instance-based, …
- 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, …)

Decision tables

- Most rudimentary form of representing output:
  - Use the same format as input!
- Decision table for the weather problem:

Outlook	Humidity	Play
Sunny	High	No
Sunny	Normal	Yes
Overcast	High	Yes
Overcast	Normal	Yes
Rainy	High	No
Rainy	Normal	No

- Main problem: selecting the right attributes

Decision trees

- "Divide-and-conquer" approach produces tree
- 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
- Leaves assign classification, set of classifications, or probability distribution to instances
- Unknown instance is routed down the tree

Nominal and numeric attributes

- Nominal attribute: number of children usually equal to number values $\Rightarrow$ attribute won't get tested more than once
  - Other possibility: division into two subsets
- Numeric attribute: test whether value is greater or less than constant $\Rightarrow$ 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 $\Rightarrow$ "missing" is a separate value
- No $\Rightarrow$ "missing" must be treated in a special way
  - Solution A: assign instance to most popular branch
  - Solution B: split instance into pieces
    - Pieces receive weight according to fraction of training instances that go down each branch
    - Classifications from leave nodes are combined using the weights that have percolated to them

Classification rules

- Popular alternative to decision trees
- *Antecedent* (pre-condition): 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): classes, 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 unambiguous
  - Doesn’t matter in which order they are executed
- But: resulting rules are unnecessarily complex
  - Pruning to remove redundant tests/rules

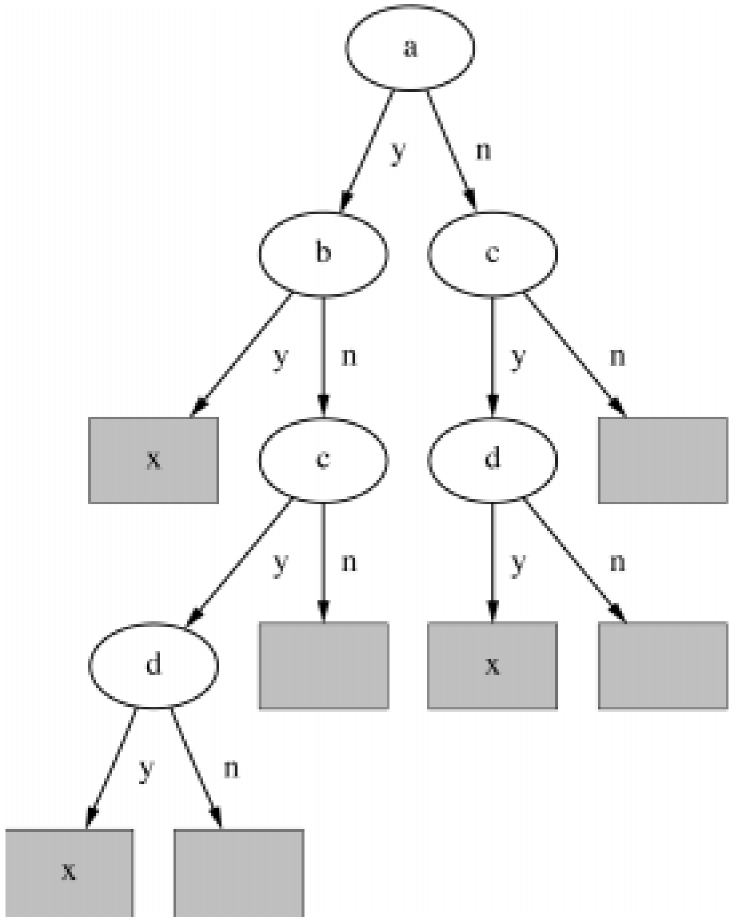
From rules to trees

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

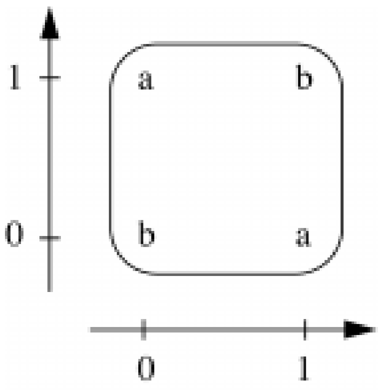
```
If a and b then x
If c and d then x
```

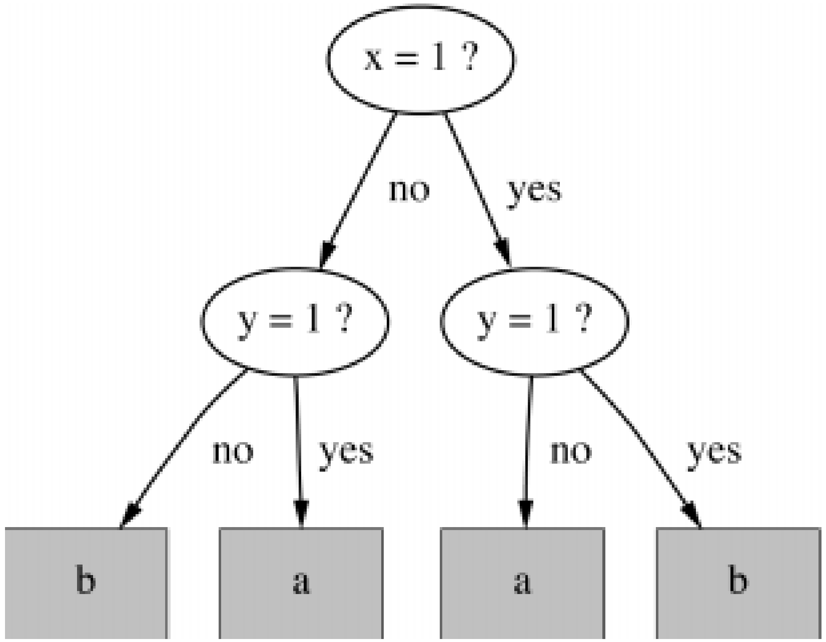
- Symmetry needs to be broken
- Corresponding tree contains identical subtrees ($\Rightarrow$ “replicated subtree problem”)

A tree for a simple disjunction



The exclusive-or problem

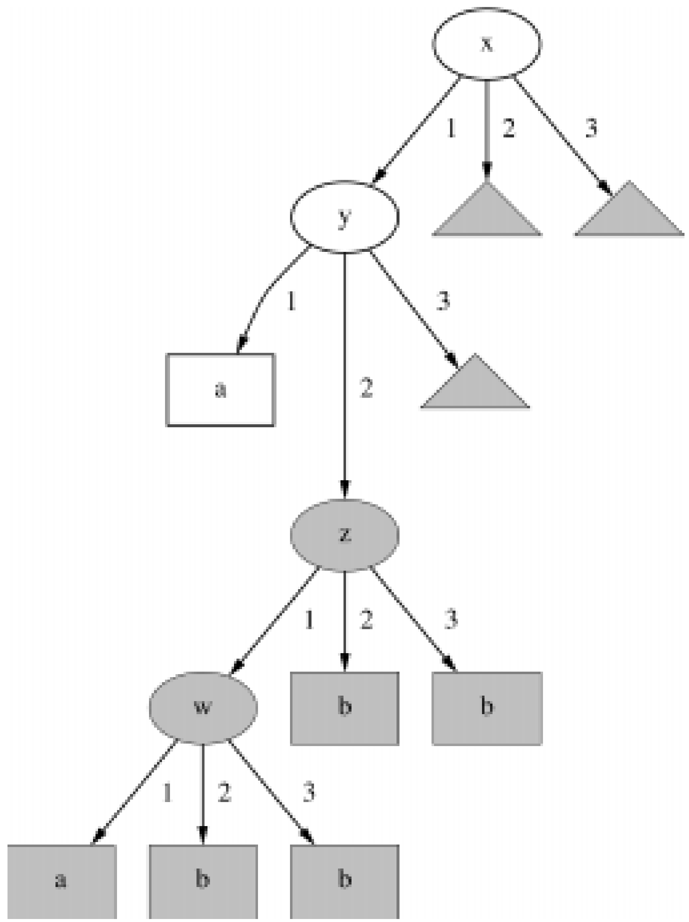





```
If x = 1 and y = 0
  then class = a
If x = 0 and y = 1
  then class = a
If x = 0 and y = 0
  then class = b
If x = 1 and y = 1
  then class = b
```

A tree with a replicated subtree

```
If x = 1 and y = 1
  then class = a
If z = 1 and w = 1
  then class = a
Otherwise class = b
```



“Nuggest” 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*

Association rules

- Association rules…
  - … can predict any attribute and combinations of attributes
  - … are not intended to be used together as a set
- Problem: immense number of possible associations
  - Output needs to be restricted to show only the most predictive associations $\Rightarrow$ 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
```

$\Rightarrow$ Support = 4, confidence = 100%

- Normally: minimum support and confidence pre-specified (e.g. 58 rules with support $\geq$ 2 and confidence $\geq$ 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
```

- However, it means that the following also holds:

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

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	Sepal width	Petal length	Petal width	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`

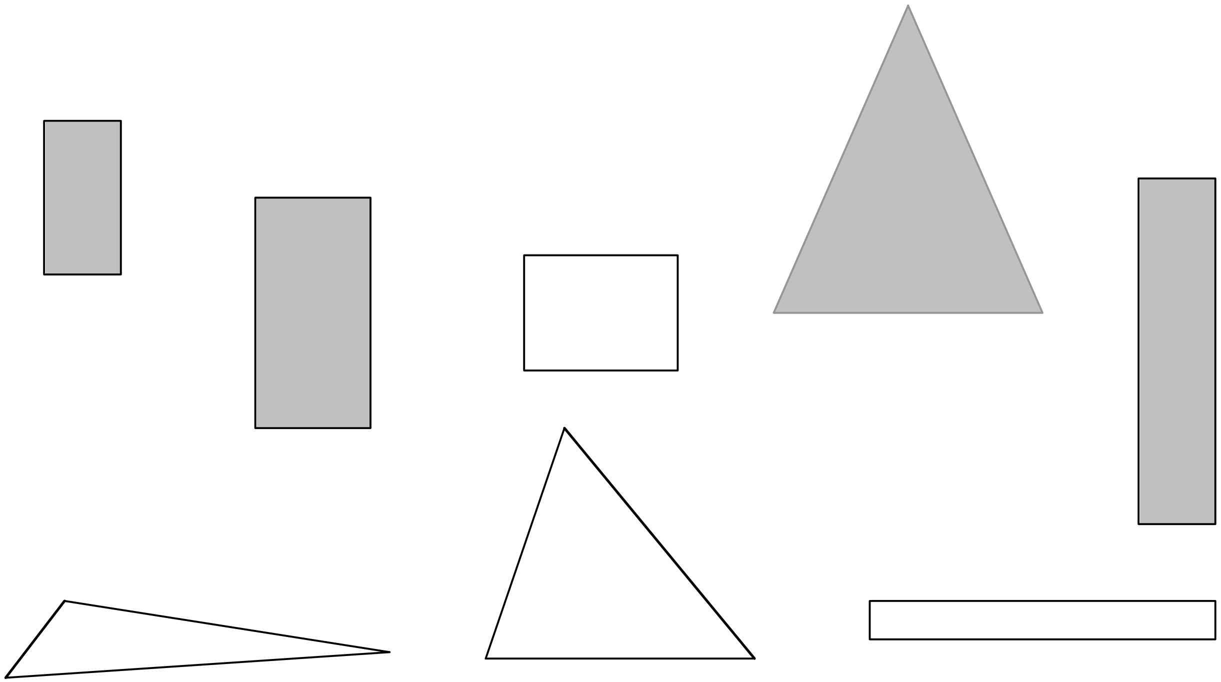
  (where the else specifies what the default did)
- But: exceptions offer a psychological advantage
  - Assumption: defaults and tests early on apply more widely than exceptions further down
  - Exceptions reflect special cases

Rules involving relations

- So far: all rules involved comparing an attribute-value to a constant (e.g. temperature < 45)
- These rules are called “propositional” because they have the same expressive power as propositional logic
- 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

- Target concept: *standing up*
- Shaded: *standing* Unshaded: *lying*



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
```

- Generalizes better to new data
- Standard relations: =, <, >
- But: learning relational rules is costly
- Simple solution: adding extra attributes (e.g. a binary attribute *is width < height?*)

Rules with variables

- Using variables and multiple relations:

```
If height_and_width_of(x,h,w) and h > w then standing(x)
```

- The top of a tower of blocks is standing:

```
If height_and_width_of(x,h,w) and h > w and is_top_of(x,y)
  then standing(x)
```

- The whole tower is standing:

```
If height_and_width_of(z,h,w) and h > w and is_top_of(x,z) and
  standing(y) and is_rest_of(x,y) then standing(x)
If empty(x) then standing(x)
```

- Recursive definition!

Inductive logic programming

- Recursive definition can be seen as logic program
- Techniques for learning logic programs stem from the area of "inductive logic programming (ILP)"
- But: recursive definitions are extremely hard to learn in practice
  - Also: very few practical problems require recursion
  - Thus: many ILP techniques are restricted to non-recursive definitions to make learning easier

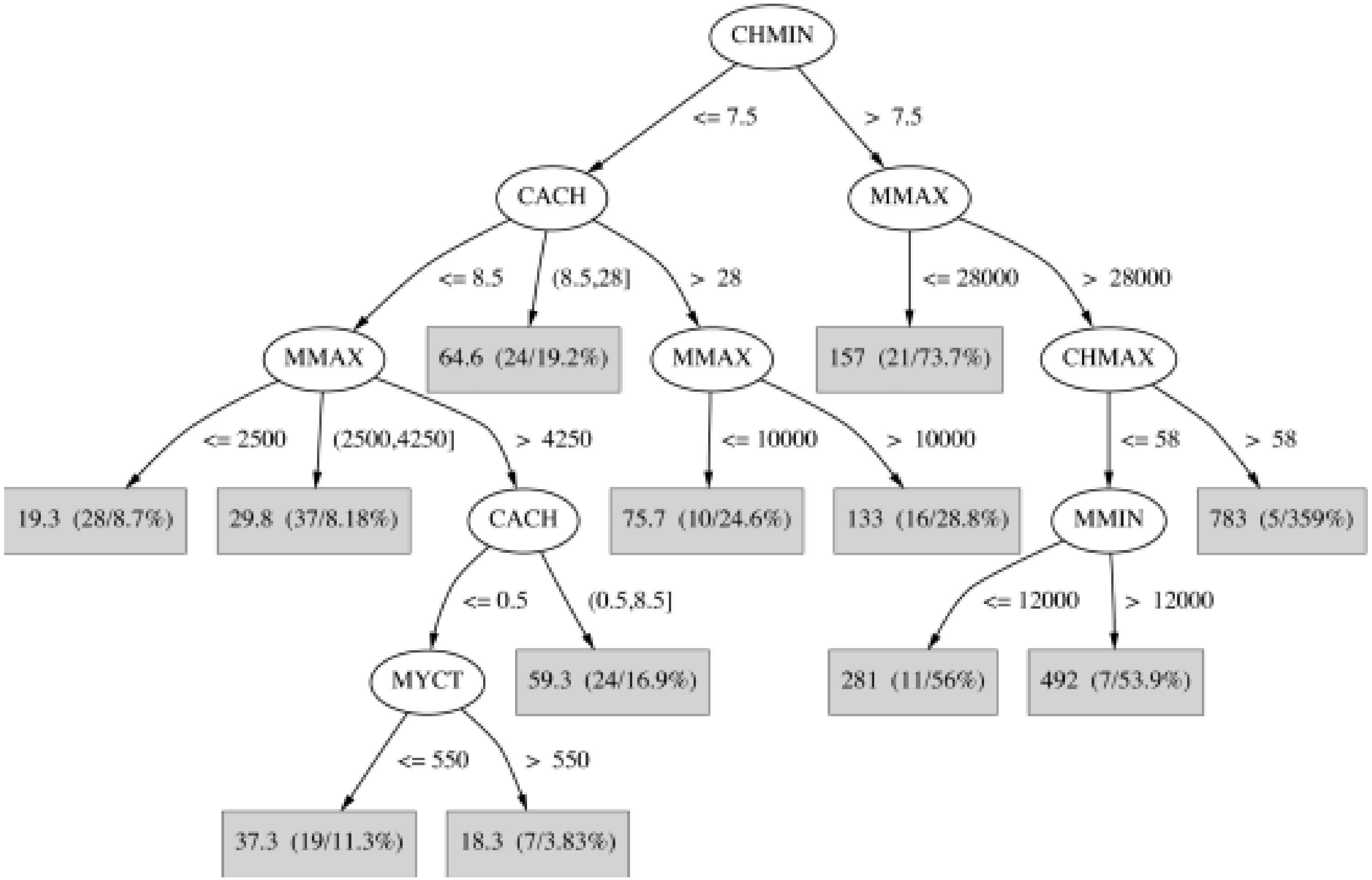
Trees for numeric prediction

- *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 patches approximate continuous function

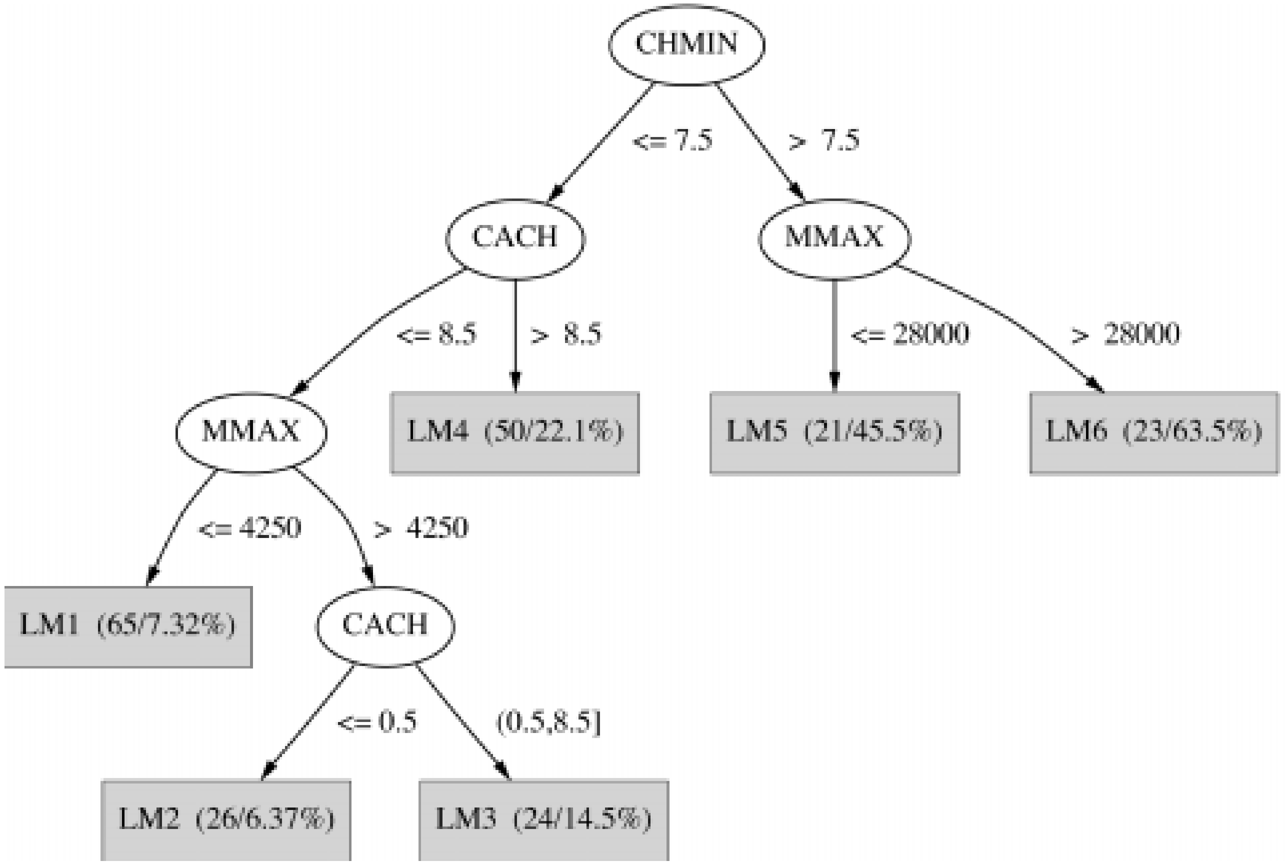
Linear regression for the CPU data

```
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



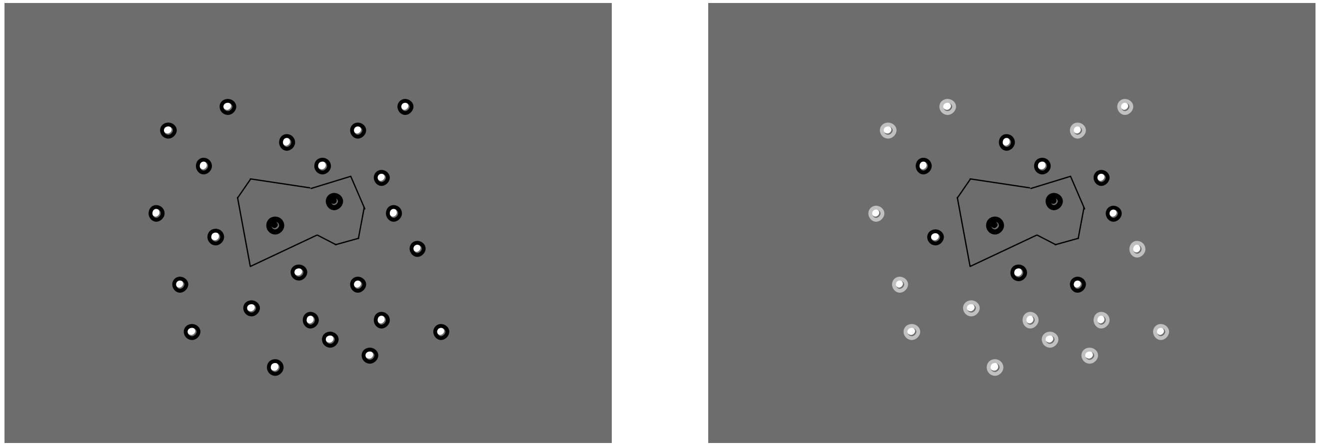
Instance-based representation

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

The distance function

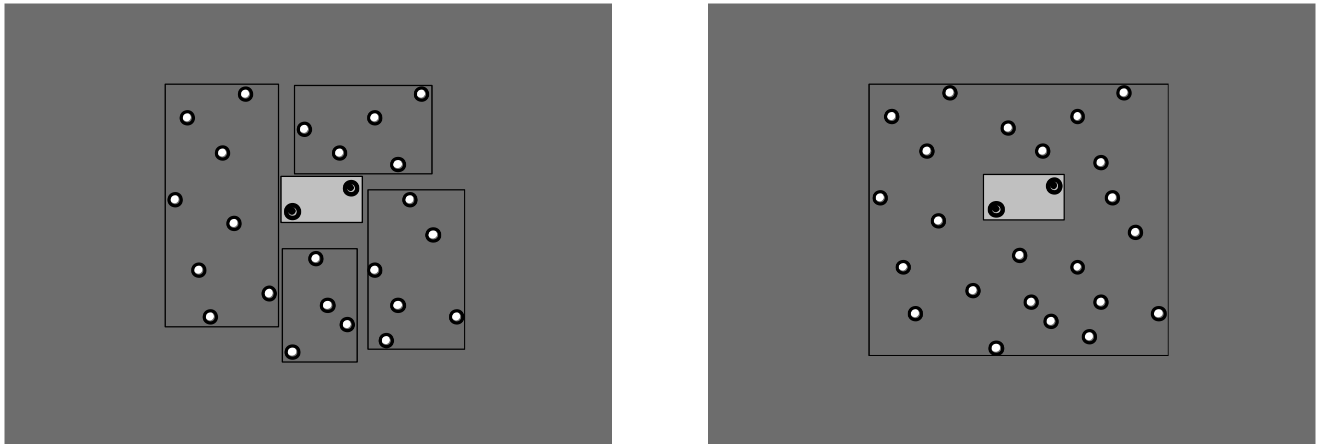
- Simplest case: one numeric attribute
  - Distance is the difference between the two attribute values involved (or a function thereof)
- Several numeric attributes: normally, Euclidean distance is used and attributes are normalized
- Nominal attributes: distance is set to 1 if values are different, 0 if they are equal
- Are all attributes equally important?
  - Weighting the attributes might be necessary

Learning prototypes



- Only those instances involved in a decision need to be stored
- Noisy instances should be filtered out
- Idea: only use *prototypical* examples

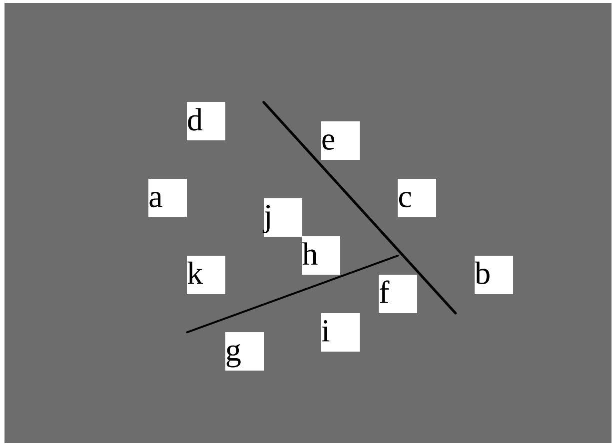
Rectangular generalizations



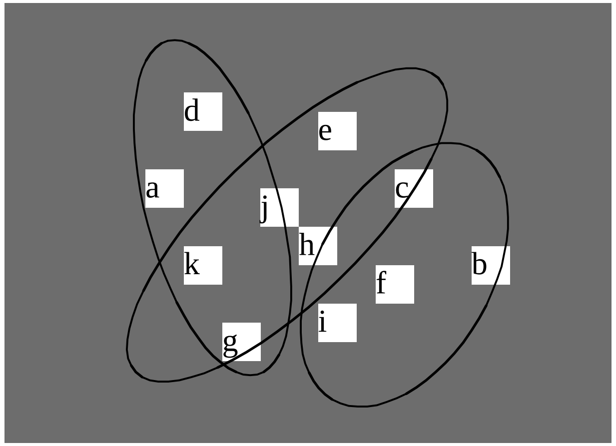
- Nearest-neighbor rule is used outside rectangles
- Rectangles are rules! (But they can be more conservative than “normal” rules.)
- Nested rectangles are rules with exceptions

Representing clusters I

Simple 2-D representation



Venn diagram



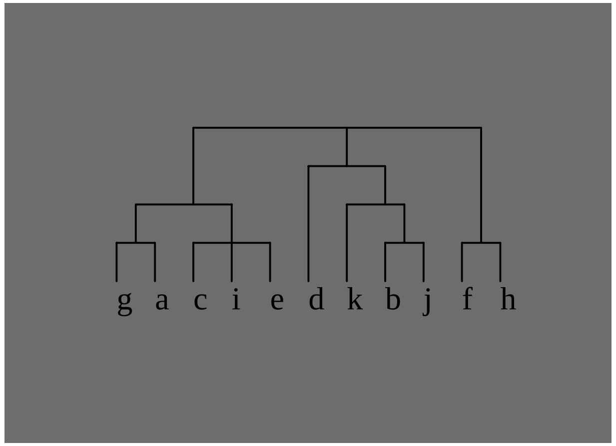
Overlapping clusters

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



NB: dendron is the Greek word for tree