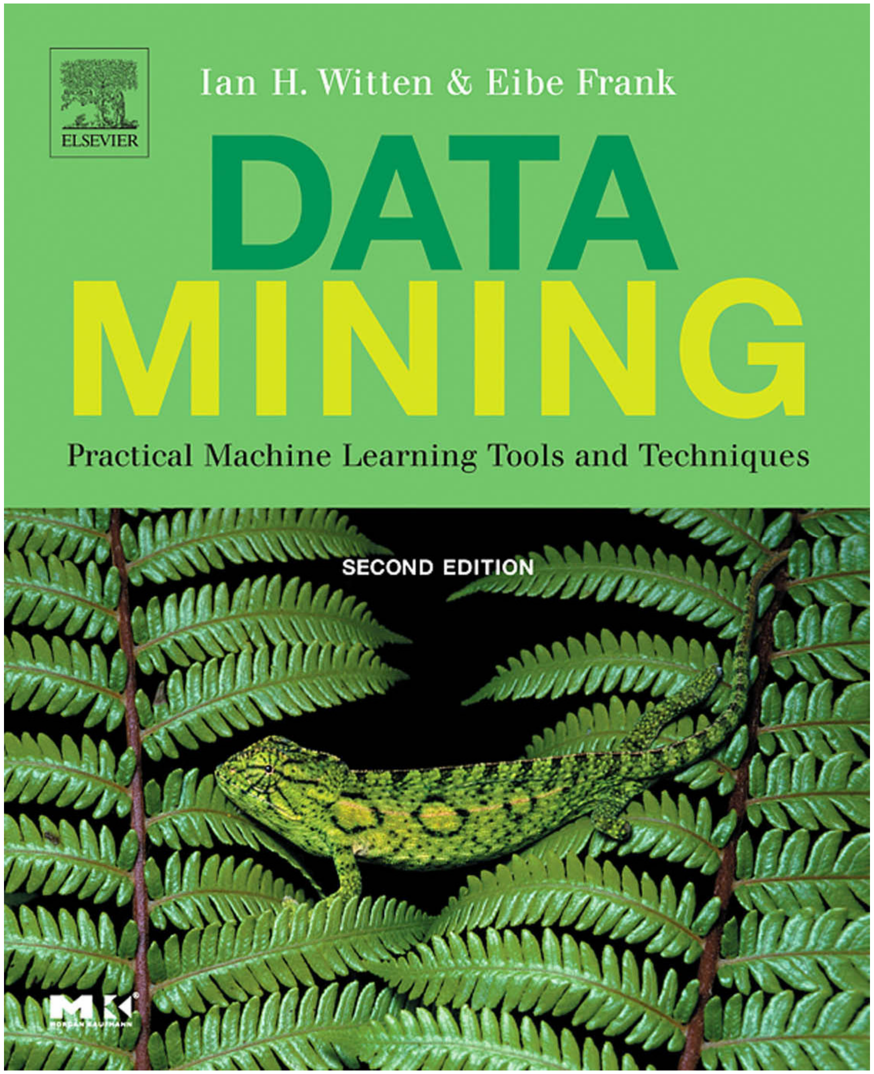


Chapter 4: Algorithms: The Basic Methods

Simplicity first

- Simple algorithms often work very well!
- There are many kinds of simple structure, eg:
  - One attribute does all the work
  - All attributes contribute equally & independently
  - A few attributes can be captured by a decision tree
  - Use simple logical rules
  - A weighted linear combination might do
  - Instance-based: use a few prototypes

Algorithms: The basic methods

- 1R Algorithm
- Naïve Bayes Classifier
- Constructing decision trees
- PRISM method
- Mining association rules
- Linear models
- k-nearest neighbor algorithm
- Clustering: k-means method

4.1 1R algorithm

1R algorithm

- An easy way to find very simple classification rule
- 1R: rules that all test one particular attribute
- Basic version
  - One branch for each value
  - Each branch assigns most frequent class
  - Error rate: proportion of instances that don't belong to the majority class of their corresponding branch
  - Choose attribute with lowest error rate (*assumes nominal attributes*)
- "Missing" is treated as a separate attribute value

Pseudo-code or 1R Algorithm

```
For each attribute,
  For each value of that attribute, make a rule as follows:
    count how often each class appears
    find the most frequent class
    make the rule assign that class to this attribute-value.
  Calculate the error rate of the rules.
Choose the rules with the smallest error rate.
```

Example: 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

Evaluating the weather attributes

	Attribute	Rules	Errors	Total errors
1	outlook	sunny → no	2/5	4/14
		overcast → yes	0/4	
		rainy → yes	2/5	
2	temperature	hot → no*	2/4	5/14
		mild → yes	2/6	
		cool → yes	1/4	
3	humidity	high → no	3/7	4/14
		normal → yes	1/7	
4	windy	false → yes	2/8	5/14
		true → no*	3/6	

The attribute with the smallest number of errors

	Attribute	Rules	Errors	Total errors
1	outlook	sunny → no	2/5	4/14
		overcast → yes	0/4	
		rainy → yes	2/5	
2	temperature	hot → no*	2/4	5/14
		mild → yes	2/6	
		cool → yes	1/4	
3	humidity	high → no	3/7	4/14
		normal → yes	1/7	
4	windy	false → yes	2/8	5/14
		true → no*	3/6	

Dealing with numeric attributes

- Discretize numeric attributes
- Divide each attribute's range into intervals
  - Sort instances according to attribute's values
  - Place breakpoints where class changes (majority class)
  - This minimizes the total error

Weather data with some numeric attributes

Outlook	Temperature	Humidity	Windy	Play
sunny	85	85	false	no
sunny	80	90	true	no
overcast	83	86	false	yes
rainy	70	96	false	yes
rainy	68	80	false	yes
rainy	65	70	true	no
overcast	64	65	true	yes
sunny	72	95	false	no
sunny	69	70	false	yes
rainy	75	80	false	yes
sunny	75	70	true	yes
overcast	72	90	true	yes
overcast	81	75	false	yes
rainy	71	91	true	no

Example: *temperature* from weather data

64	65	68	69	70	71	72	72	75	75	80	81	83	85
yes	no	yes	yes	yes	no	no	yes	yes	yes	no	yes	yes	no

- Discretization involves partitioning this sequence by placing breakpoints wherever the class changes,

yes | no | yes yes yes | no no | yes yes yes | no | yes yes | no

The problem of overfitting

- Overfitting is likely to occur whenever an attribute has a large number of possible values
- This procedure is very sensitive to noise
  - One instance with an incorrect class label will probably produce a separate interval
- Attribute will have zero errors
- Simple solution: *enforce minimum number of instances in majority class per interval*

Minimum is set at 3 for temperature attribute

- The partitioning process begins

```
yes  no  yes  yes | yes . . .
```

- the next example is also *yes,* we lose nothing by including that in the first partition

```
yes  no  yes  yes  yes | no  no  yes  yes  yes | no  yes  yes  no
```

- Thus the final discretization is

```
yes  no  yes  yes  yes  no  no  yes  yes  yes | no  yes  yes  no
```

- the rule set

$$\text{temperature: } \begin{array}{l} \leq 77.5 \rightarrow \text{yes} \\ > 77.5 \rightarrow \text{no} \end{array}$$

Resulting rule set with overfitting avoidance

Attribute	Rules	Errors	Total errors
Outlook	Sunny → No	2/5	4/14
	Overcast → Yes	0/4	
	Rainy → Yes	2/5	
Temperature	≤ 77.5 → Yes	3/10	5/14
	> 77.5 → No*	2/4	
Humidity	≤ 82.5 → Yes	1/7	3/14
	> 82.5 and ≤ 95.5 → No	2/6	
	> 95.5 → Yes	0/1	
Windy	False → Yes	2/8	5/14
	True → No*	3/6	

4.2 Naïve Bayes Classifier

Naïve Bayes Classifier

- “Opposite” of 1R: use all the attributes
- Two assumptions: Attributes are
  - *equally important*
  - *statistically independent*
    - I.e., knowing the value of one attribute says nothing about the value of another
- Equally important & independence assumptions are never correct in real-life datasets

Bayes Theorem

- **Probability of event *H* given evidence *E*:**

$$\Pr[H|E] = \frac{\Pr[E|H]\Pr[H]}{\Pr[E]}$$

- *Pr[H]*: *A priori* probability of *H*
  - Probability of event *before* evidence is seen
- *Pr[H|E]*: *posteriori* probability of *H*
  - The probability of *H* conditional on *E*
- *Pr[E|H]*: Posterior probability of X
- *Pr[E]*: *A priori* probability of *E*

Naïve Bayes for classification

- **Classification learning: what's the probability of the class given an instance?**
  - Evidence E = instance
  - Event H = class value for instance
- Naïve assumption: evidence splits into parts (i.e. attributes) that are *independent*

$$Pr[H|E] = \frac{Pr[E1|H]Pr[E2|H]\ldots Pr[En|H]Pr[H]}{Pr[E]}$$

Naïve Bayes classifier

- Hypothesis *H* is the class.
- *Pr [E]*: can be ignored as it is constant for all classes.

$$\Pr(H \mid E) = \Pr(H)\prod_{k=1}^{n}\Pr(E_k \mid H)$$

- *Pr(H)* is the ratio of total samples in class *H* to all samples.

Naïve Bayes classifier

- For Categorical attribute:
  - *$Pr(E_k|H)$* is the frequency of samples having value E_k in class *H*.

- For Continuous (numeric) attribute:
  - *$Pr(E_k|H)$* is calculated via a Normal or Gaussian density function.

Naïve Bayes classifier

- Having pre-calculated all $Pr(E_k|H)$ to classify an unknown sample E:
  - Step 1: For all classes calculate $P(H|E)$.
  - Step 2: Assign sample E to the class with the highest $Pr(H|E)$.

Naïve Bayes classifier

Outlook			Temperature			Humidity			Windy			Play	
	yes	*no*		*yes*	*no*		*yes*	*no*		*yes*	*no*	*yes*	*no*
sunny	2	3	hot	2	2	high	3	4	false	6	2	9	5
overcast	4	0	mild	4	2	normal	6	1	true	3	3		
rainy	3	2	cool	3	1								
sunny	2/9	3/5	hot	2/9	2/5	high	3/9	4/5	false	6/9	2/5	9/14	5/14
overcast	4/9	0/5	mild	4/9	2/5	normal	6/9	1/5	true	3/9	3/5		
rainy	3/9	2/5	cool	3/9	1/5								

- E.g. *Pr(outlook=sunny | play=yes) = 2/9*

  Pr(windy=true | play=No) = 3/9

Probabilities for weather data

- A new day:

Outlook	Temperature	Humidity	Windy	Play
sunny	cool	high	true	?

$$\text{likelihood of } yes = 2/9 \times 3/9 \times 3/9 \times 3/9 \times 9/14 = 0.0053.$$

$$\text{likelihood of } no = 3/5 \times 1/5 \times 4/5 \times 3/5 \times 5/14 = 0.0206.$$

- Conversion into a probability by normalization:

$$\text{Probability of } yes = \frac{0.0053}{0.0053 + 0.0206} = 20.5\%,$$

$$\text{Probability of } no = \frac{0.0206}{0.0053 + 0.0206} = 79.5\%.$$

Bayes's rule

- The hypothesis H (class) is that ***play*** will be '***yes***' Pr[*H*|*E*] is 20.5%
- The evidence *E* is the particular combination of attribute values for the new day:

  outlook = *sunny*
  temperature = *cool*
  humidity = *high*
  windy = *true*

Weather data example

$$
\begin{aligned}
Pr\,[yes|E] = \; & Pr\,[Outlook=Sunny|yes] \\
& \times Pr\,[Temperature=Cool|yes] \\
& \times Pr\,[Humidity=High|yes] \\
& \times Pr\,[Windy=True|yes] \\
& \times Pr\,[yes]
\end{aligned}
$$

$$\Pr[yes|E] = 2/9 \times 3/9 \times 3/9 \times 3/9 \times 9/14$$

The "zero-frequency problem"

- What if an attribute value doesn't occur with every class value?
  - e.g. "Humidity = high" for class "yes" Probability will be zero!
    Pr [Humidity=High | yes]=0
  - *A posteriori* probability will also be zero!
    Pr [yes | E]=0
  - (No matter how likely the other values are!)
- Correction: add 1 to the count for every attribute value-class combination (*Laplace estimator)*
- Result: probabilities will never be zero!

Modified probability estimates

- In some cases adding a constant different from 1 might be more appropriate
- Example: attribute *outlook* for class '*yes*'

$$\frac{2+\mu/3}{9+\mu} \qquad \frac{4+\mu/3}{9+\mu} \qquad \frac{3+\mu/3}{9+\mu}$$

sunny *overcast* *rainy*

- Weights don't need to be equal but they must sum to 1 ($p1$, $p2$, and $p3$ sum to 1)

$$\frac{2+\mu p_1}{9+\mu} \qquad \frac{4+\mu p_2}{9+\mu} \qquad \frac{3+\mu p_3}{9+\mu}$$

Missing values

- Training: instance is not included in frequency count for attribute value-class combination
- Classification: attribute will be omitted from calculation
- Example: if the value of *outlook* were missing in the example

Outlook	Temperature	Humidity	Windy	Play
?	cool	high	true	?

  - Likelihood of "yes" = 3/9 x 3/9 x 3/9 x 9/14 = 0.0238
  - Likelihood of "no" = 1/5 x 4/5 x 3/5 x 5/14 = 0.0343
  - P("yes") = 0.0238 / (0.0238 + 0.0343) = 41%
  - P("no") = 0.0343 / (0.0238 + 0.0343) = 59%

Numeric attributes

- Usual assumption: attributes have a *normal* or *Gaussian* probability distribution
- The *probability density function* for the normal distribution is defined by two parameters:
- *Sample mean μ*

$$\mu = \frac{1}{n}\sum_{i=1}^{n} x_i$$

- *Standard deviation σ*

$$\sigma = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \mu)^2}$$

- Then the density function *f(x) is:*

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{\frac{(x-\mu)^2}{2\sigma^2}}$$

Statistics for weather data

Outlook			Temperature			Humidity			Windy			Play	
	yes	*no*		*yes*	*no*		*yes*	*no*		*yes*	*no*	*yes*	*no*
sunny	2	3		83	85		86	85	false	6	2	9	5
overcast	4	0		70	80		96	90	true	3	3		
rainy	3	2		68	65		80	70					
				64	72		65	95					
				69	71		70	91					
				75			80						
				75			70						
				72			90						
				81			75						
sunny	2/9	3/5	*mean*	73	74.6	*mean*	79.1	86.2	false	6/9	2/5	9/14	5/14
overcast	4/9	0/5	*std. dev.*	6.2	7.9	*std. dev.*	10.2	9.7	true	3/9	3/5		
rainy	3/9	2/5											

Example density value

- If we are considering a *yes* outcome when *temperature* has a value of 66
- We just need to plug x = 66, μ = 73, and σ = 6.2 into the formula
- The value of the probability density function is:

$$f(temperature = 66 \mid yes) = \frac{1}{\sqrt{2\pi} \cdot 6.2} e^{\frac{(66-73)^2}{2 \cdot 6.2^2}} = 0.0340$$

Classifying a new day

- A new day:

Outlook	Temperature	Humidity	Windy	Play
sunny	66	90	true	?

$$\text{likelihood of } yes = 2/9 \times 0.0340 \times 0.0221 \times 3/9 \times 9/14 = 0.000036$$

$$\text{likelihood of } no = 3/5 \times 0.0221 \times 0.0381 \times 3/5 \times 5/14 = 0.000108$$

$$\text{Probability of } yes = \frac{0.000036}{0.000036 + 0.000108} = 25.0\%$$

$$\text{Probability of } no = \frac{0.000108}{0.000036 + 0.000108} = 75.0\%$$

Missing values

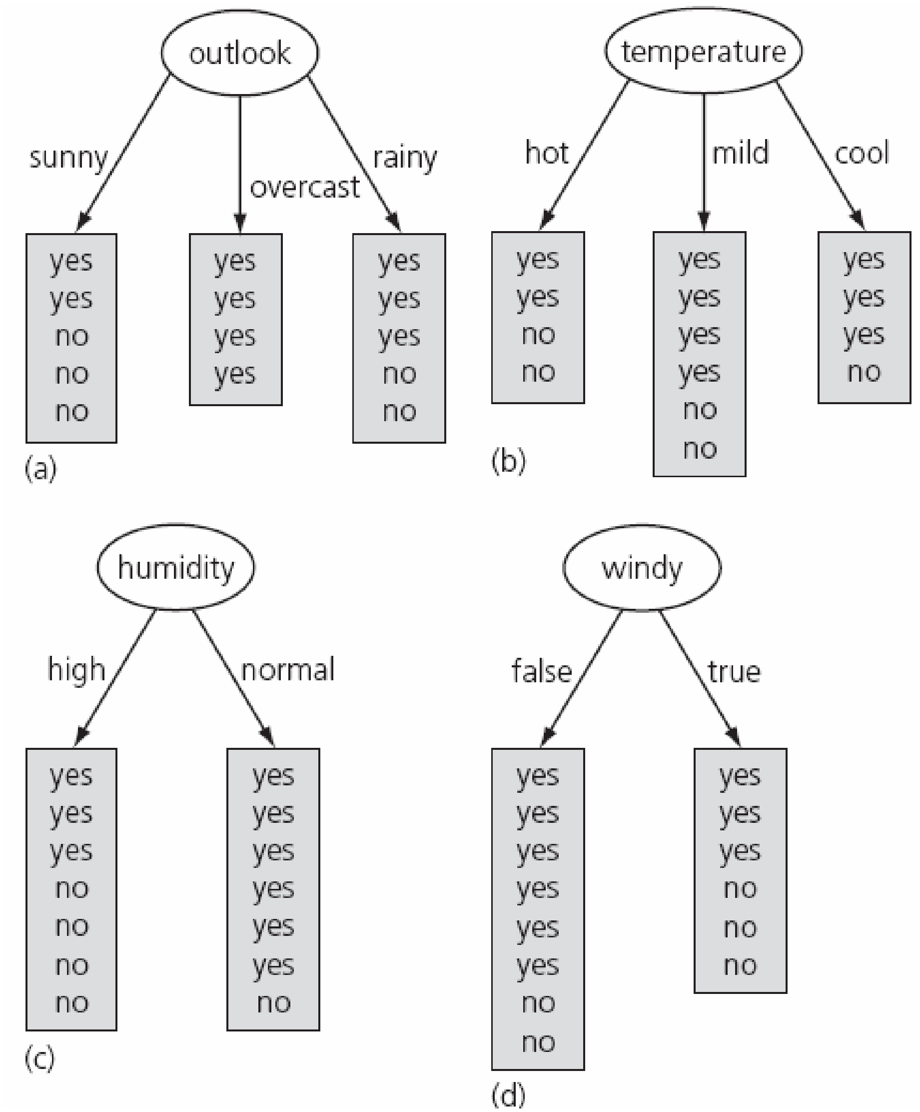
- Missing values during training are not included in calculation of mean and standard deviation

4.3 Constructing decision trees

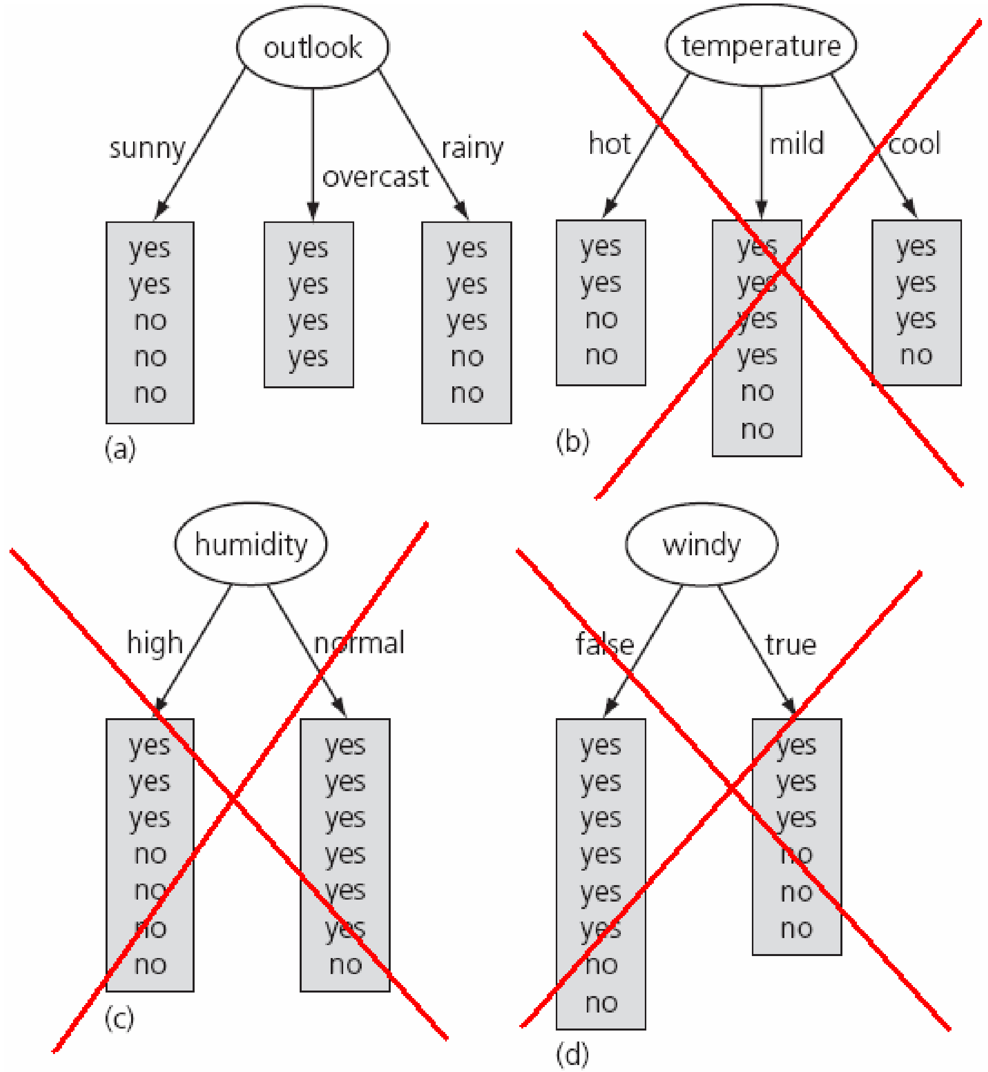
Constructing decision trees

- Strategy: top down
- Recursive *divide-and-conquer*
  - First: select attribute for root node
    Create branch for each possible attribute value
  - This splits instances into subsets
    One for each branch extending from the node
  - Then: repeat recursively for each branch, using only instances that reach the branch
- Stop if all instances at a node have the same class

Which attribute to select?



Which attribute to select?



Criterion for attribute selection

- Which is the best attribute?
  - Want to get the smallest tree
  - Heuristic: choose the attribute that produces the "purest" nodes
- Popular *impurity criterion*: ***information gain***
  - Information gain increases with the average purity of the subsets
  - It is measured in ***bits***
- Strategy: choose attribute that gives greatest information gain

Criterion for attribute selection

- Nodes with homogeneous class distribution are preferred

- Need a measure of node impurity:

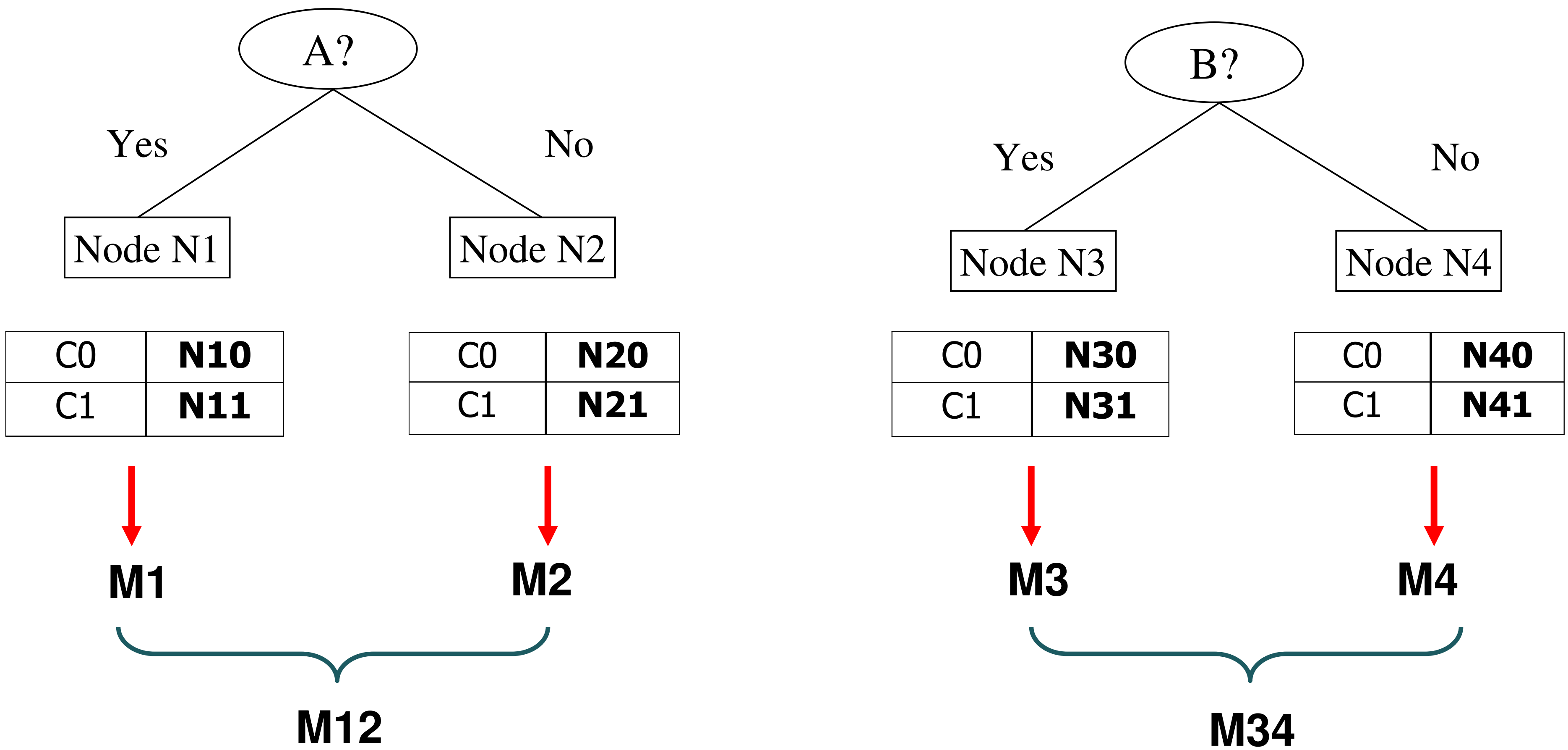
C0: 5 C1: 5	C0: 9 C1: 1
Non-homogeneous, **High degree of impurity**	**Homogeneous,** **Low degree of impurity**

How to Find the Best Split

Before Splitting:

C0	**N00**
C1	**N01**

→ **M0**



Gain = M0 – M12 vs M0 – M34

Computing information

- Given a probability distribution, the info required to predict an event is the distribution's *entropy*
- Entropy gives the information required in bits (can involve fractions of bits!)
- Formula for computing the entropy:

$$\text{entropy}(p_1, p_2, \ldots, p_n) = -p_1 \log p_1 - p_2 \log p_2 \ldots - p_n \log p_n$$

- *"High Entropy"* means X is from a uniform (boring) distribution
- *"Low Entropy"* means X is from a varied (peaks and valleys) distribution

Example: attribute *Outlook*

- **Outlook = Sunny:**
  info([2,3])=entropy(2/5,3/5)=−2/5log(2/5)−3/5log(3/5)=0.971bits

- **Outlook = Overcast:**
  info([4,0])=entropy(1,0)=−1log(1)−0log(0)=0 bits

- **Outlook = Rainy:**
  info([2,3])=entropy(3/5,2/5)=−3/5log(3/5)−2/5log(2/5)=0.971bits

- **Expected information for attribute:**
  info([3,2], [4,0], [3,2])=(5/14)×0.971+(4/14)×0+(5/14)×0.971=0.693 bits

Computing information gain

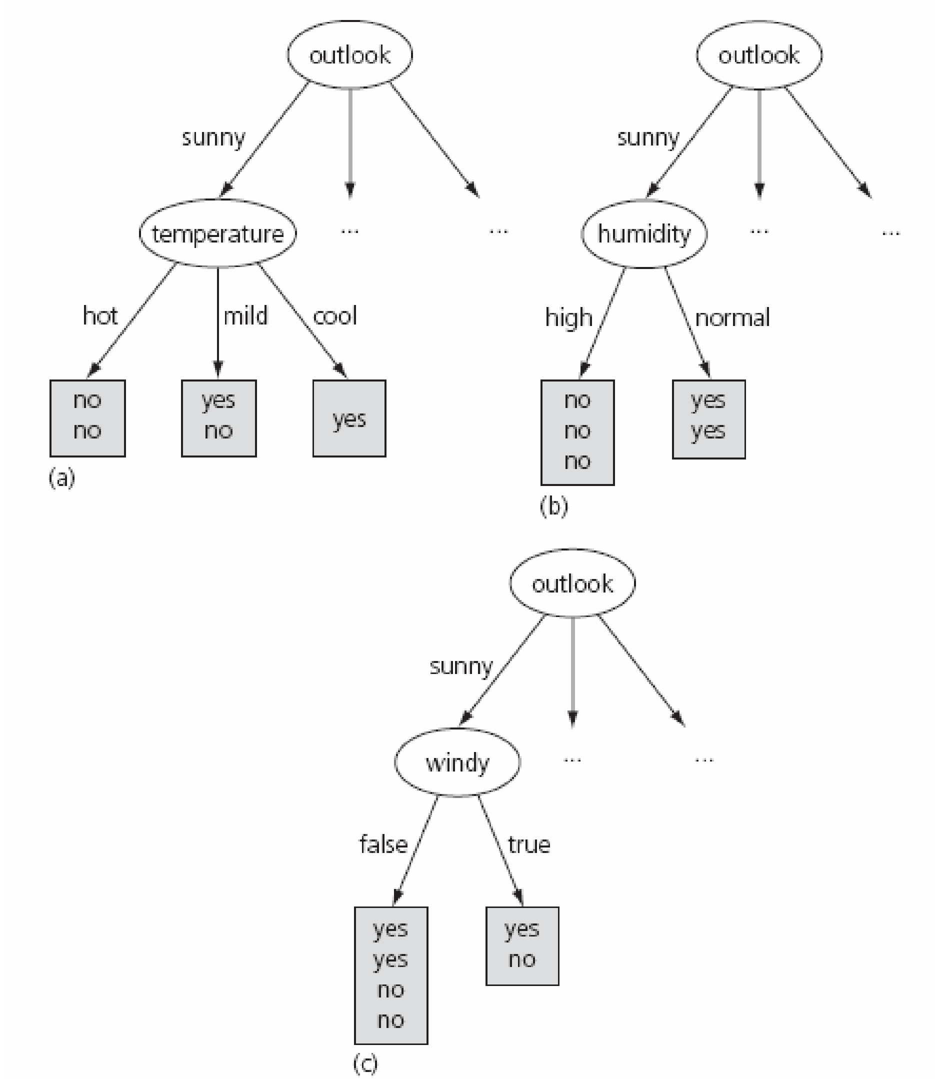
- Information gain: information before splitting – information after splitting:

  gain(*Outlook*) = info([9,5]) – info([2,3],[4,0],[3,2])
  = 0.940 – 0.693
  = 0.247 bits

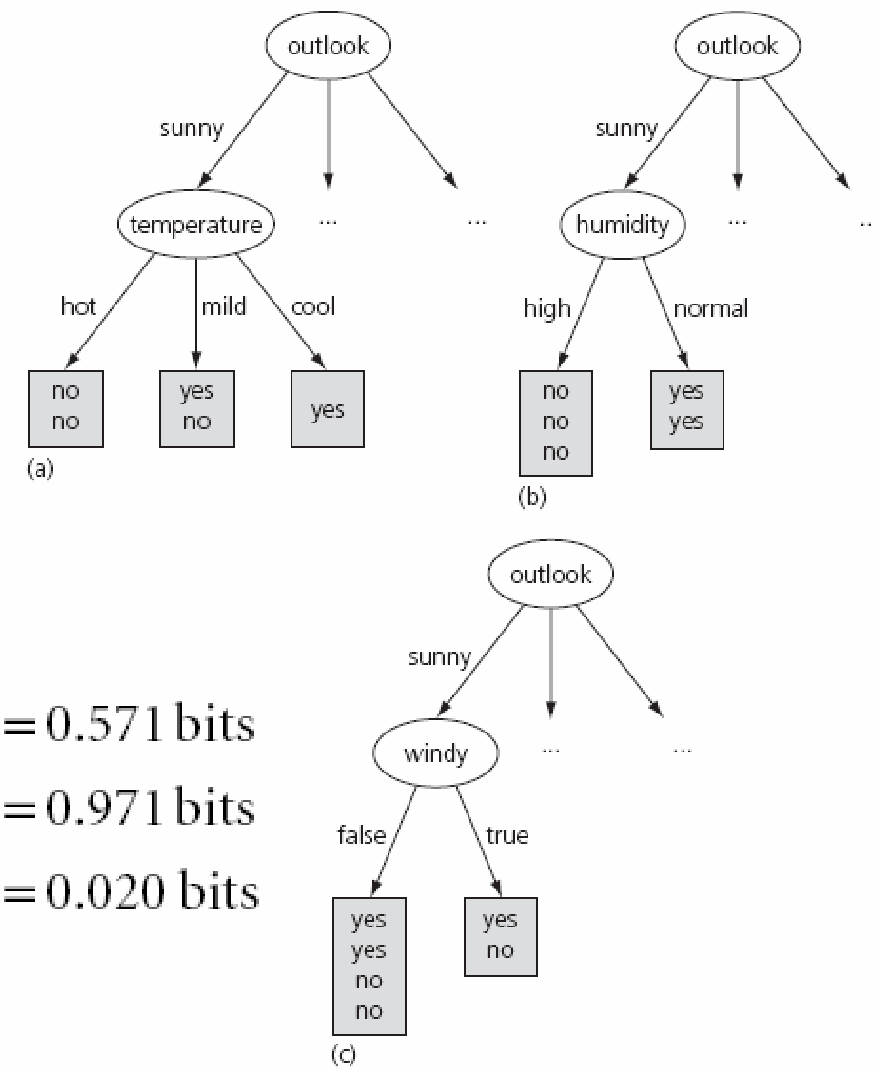
- Information gain for attributes from weather data:

  gain(*Outlook*) = 0.247 bits
  gain(*Temperature*) = 0.029 bits
  gain(*Humidity*) = 0.152 bits
  gain(*Windy*) = 0.048 bits

Continuing to split

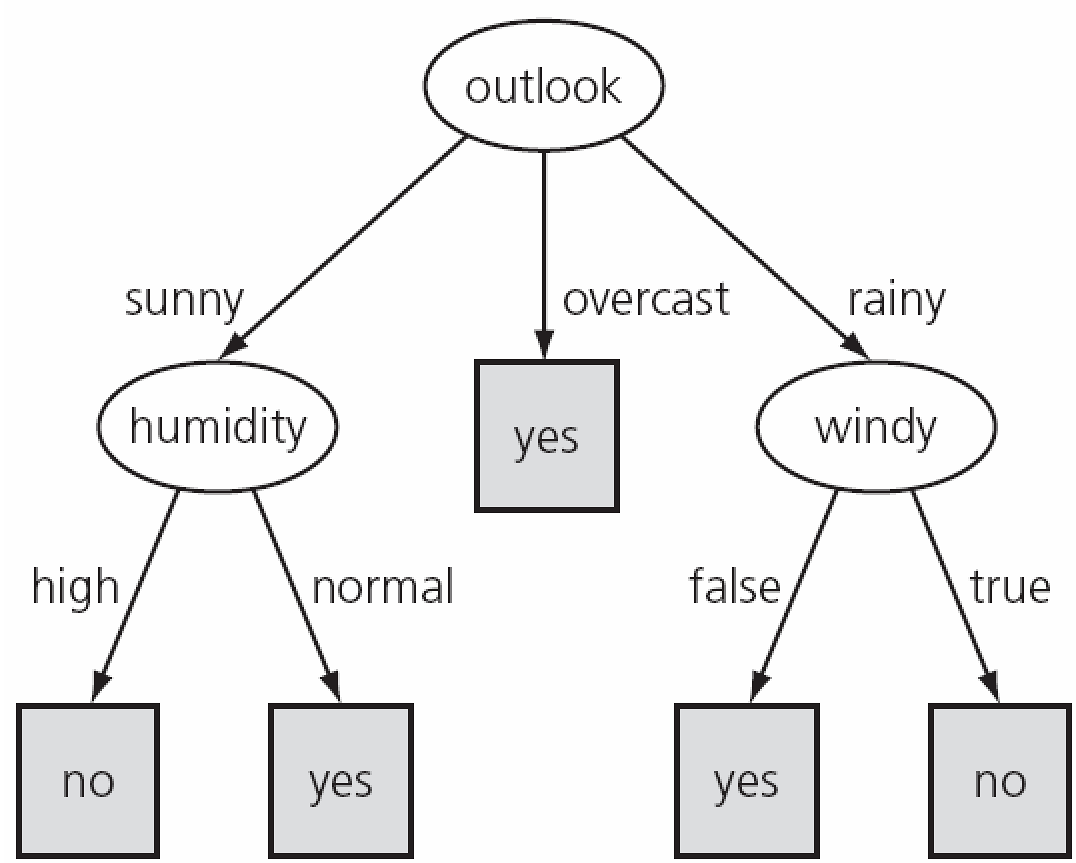


Continuing to split



$$\text{gain}(\textit{temperature}) = 0.571 \text{ bits}$$
$$\text{gain}(\textit{humidity}) = 0.971 \text{ bits}$$
$$\text{gain}(\textit{windy}) = 0.020 \text{ bits}$$

Final decision tree



- Splitting stops when data can't be split any further

Wish list for a purity measure

- Properties we require from a purity measure:
  - When node is pure, measure should be zero
  - When impurity is maximal (i.e. all classes equally likely), measure should be maximal

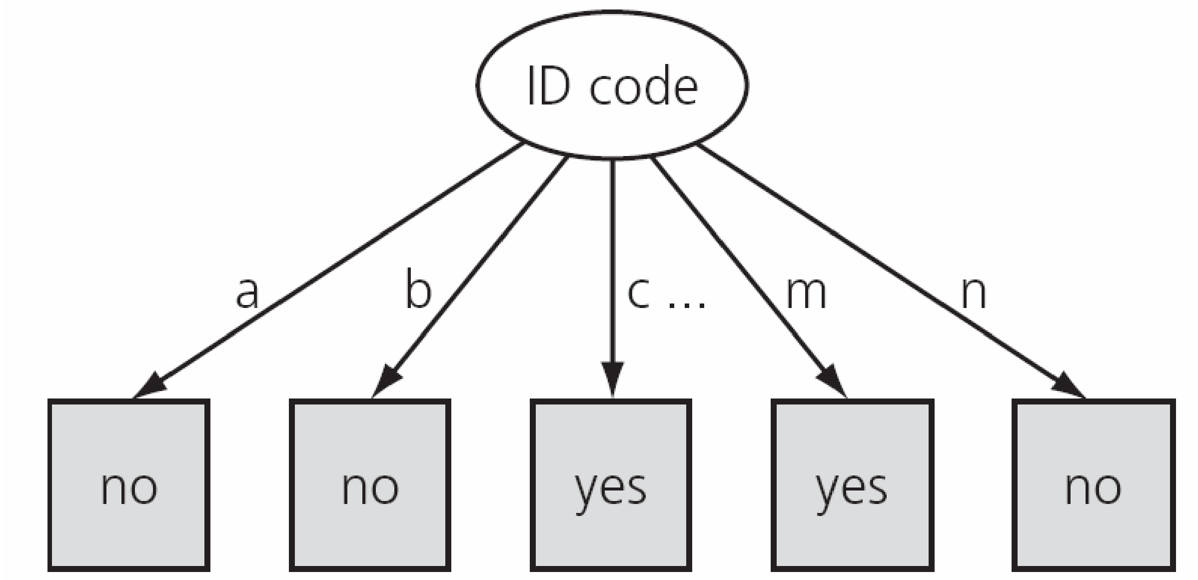
Highly-branching attributes

- Problem: attributes with a large number of values (extreme case: ID code)
- Subsets are more likely to be pure if there is a large number of values
  - Information gain is biased towards choosing attributes with a large number of values
  - This may result in selection of an attribute that is non-optimal for prediction
- Another problem: *fragmentation*

Weather data with *ID code*

ID code	Outlook	Temperature	Humidity	Windy	Play
a	sunny	hot	high	false	no
b	sunny	hot	high	true	no
c	overcast	hot	high	false	yes
d	rainy	mild	high	false	yes
e	rainy	cool	normal	false	yes
f	rainy	cool	normal	true	no
g	overcast	cool	normal	true	yes
h	sunny	mild	high	false	no
i	sunny	cool	normal	false	yes
j	rainy	mild	normal	false	yes
k	sunny	mild	normal	true	yes
l	overcast	mild	high	true	yes
m	overcast	hot	normal	false	yes
n	rainy	mild	high	true	no

Tree stump for *ID code* attribute



- Entropy of split '*ID Code*':

$$\text{info}([0,1]) + \text{info}([0,1]) + \text{info}([1,0]) + \ldots + \text{info}([1,0]) + \text{info}([0,1])$$

- Information gain is maximal for ID code (namely 0.940 bits)

Gain ratio

- *Gain ratio*: a modification of the information gain
- Gain ratio takes number and size of branches into account when choosing an attribute
  - It corrects the information gain by taking the *intrinsic information* of a split into account
- Intrinsic information: entropy of distribution of instances into branches

Computing the gain ratio

- Example: intrinsic information for *Outlook split:*

$$\text{info}([5,4,5]) = 1.577$$

- Value of attribute decreases as intrinsic information gets larger
- Gain ratio attribute = gain attribute / intrinsic info attribute
- Gain ratio ID code = 0.247 bits / 1.577 bits = 1.157

Gain ratios for weather data

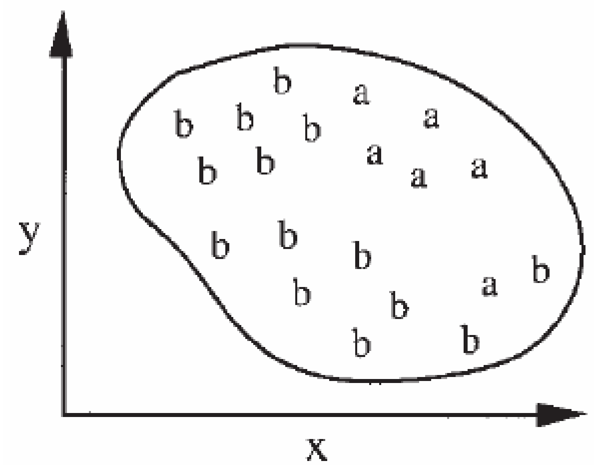
Outlook		Temperature		Humidity		Windy	
info:	0.693	info:	0.911	info:	0.788	info:	0.892
gain: 0.940–0.693	0.247	gain: 0.940–0.911	0.029	gain: 0.940–0.788	0.152	gain: 0.940–0.892	0.048
split info: info([5,4,5])	1.577	split info: info([4,6,4])	1.557	split info: info ([7,7])	1.000	split info: info([8,6])	0.985
gain ratio: 0.247/1.577	0.157	gain ratio: 0.029/1.557	0.019	gain ratio: 0.152/1	0.152	gain ratio: 0.048/0.985	0.049

4.4 PRISM method

Covering algorithms

- Convert decision tree into a rule set
  - Straightforward, but rule set very complex
- Instead, can generate rule set directly
  - for each class in turn find rule set that covers all instances in it (excluding instances not in the class)
- Called a *covering* approach:
  - at each stage a rule is identified that “covers” some of the instances

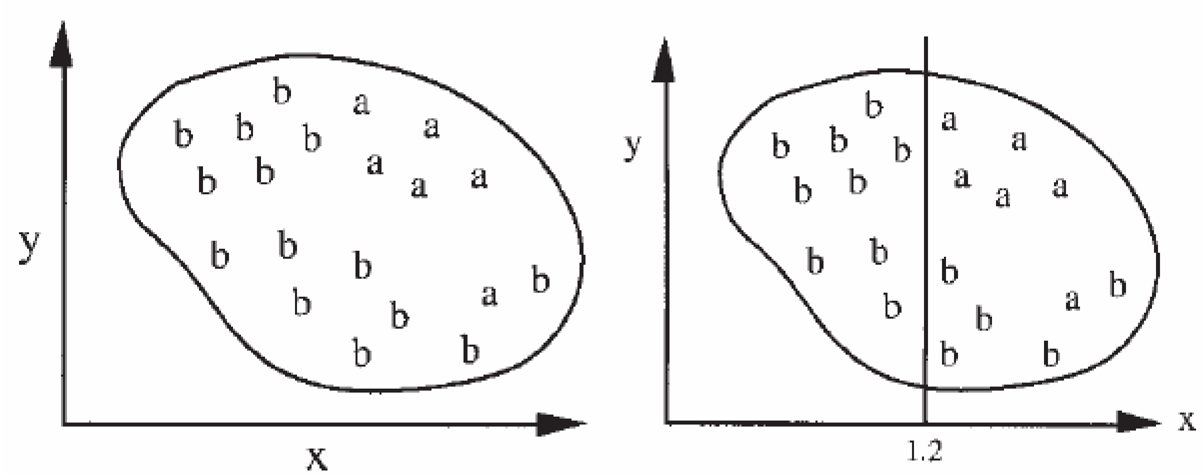
Example: generating a rule



- Possible rule set for class "a":

  if true then class = a

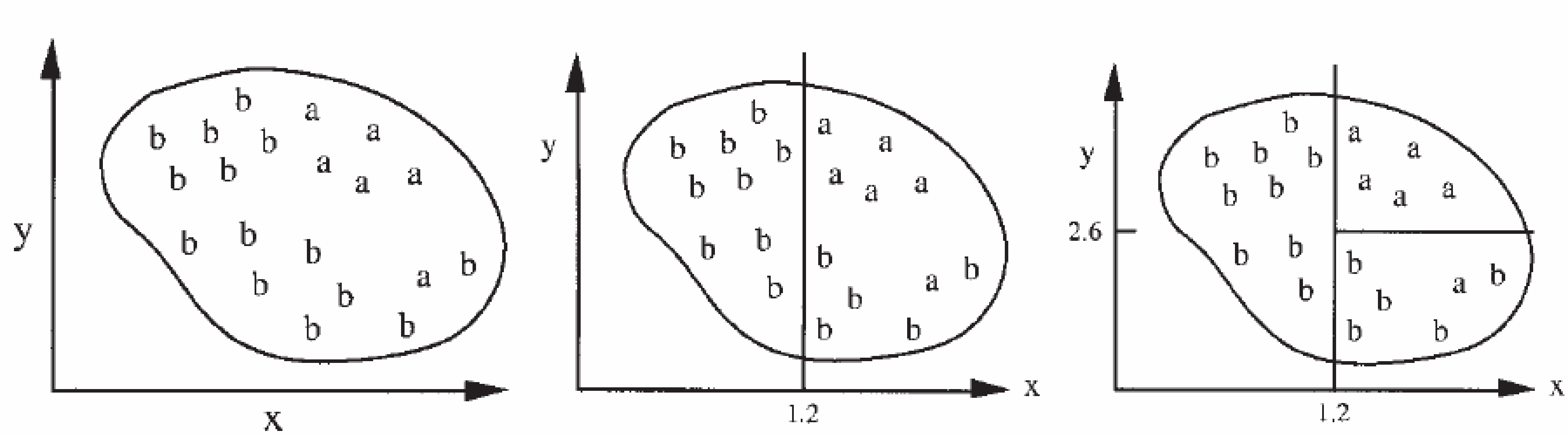
Example: generating a rule



- Possible rule set for class “a”:

```
If x > 1.2 then class = a
```

Example: generating a rule

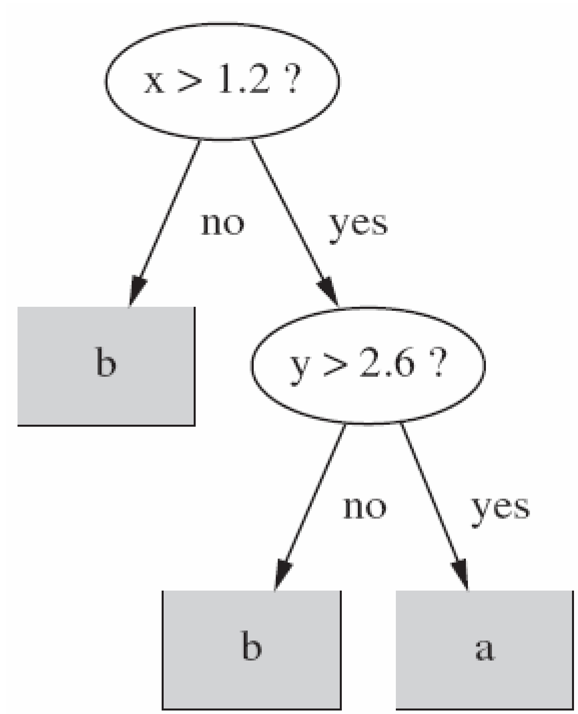


- Possible rule set for class "a":

```
If x > 1.2 and y > 2.6 then class = a
```

Decision tree for the same problem

- Corresponding decision tree: (produces exactly the same predictions)



Rules vs. trees

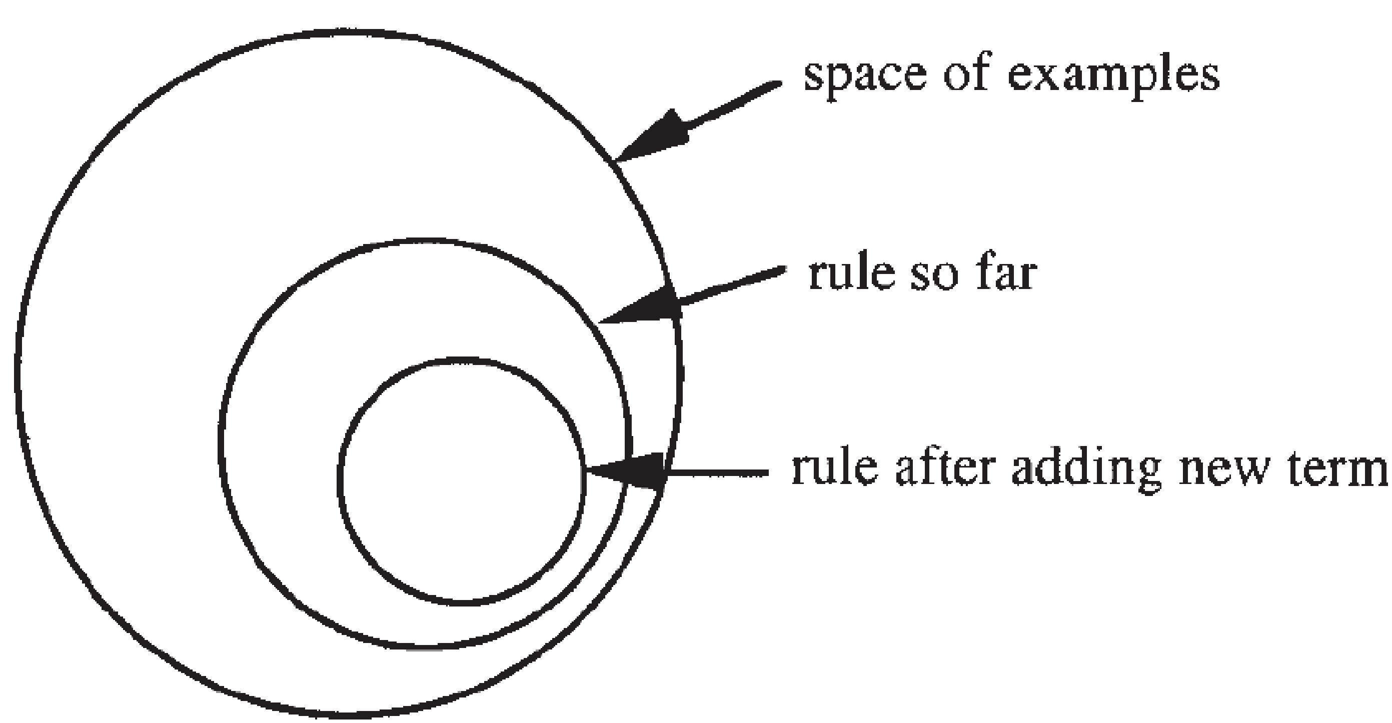
- Both methods might first split the dataset using the *x* attribute and would probably end up splitting it at the same place ($x = 1.2$)
- But: rule sets *can* be more clear when decision trees suffer from replicated subtrees
- Also: in multiclass situations, covering algorithm concentrates on one class at a time whereas decision tree learner takes all classes into account

A simple covering algorithm

- It is called ***PRISM method*** for constructing rules
- Generates a rule by adding tests that maximize rule's accuracy
- Divide-and-conquer algorithms choose an attribute to maximize the information gain
- But: the covering algorithm chooses an attribute–value pair to maximize the probability of the desired classification

A simple covering algorithm

- Each new test reduces rule's coverage:



Selecting a test

- Goal: maximize accuracy
  - t total number of instances covered by rule
  - p positive examples of the class covered by rule
  - $t - p$ number of errors made by rule
  - Select test that maximizes the ratio p/t

- We are finished when $p/t = 1$ or the set of instances can't be split any further

Example: contact lens data

Age	Spectacle prescription	Astigmatism	Tear production rate	Recommended lenses
young	myope	no	reduced	none
young	myope	no	normal	soft
young	myope	yes	reduced	none
young	myope	yes	normal	hard
young	hypermetrope	no	reduced	none
young	hypermetrope	no	normal	soft
young	hypermetrope	yes	reduced	none
young	hypermetrope	yes	normal	hard
pre-presbyopic	myope	no	reduced	none
pre-presbyopic	myope	no	normal	soft
pre-presbyopic	myope	yes	reduced	none
pre-presbyopic	myope	yes	normal	hard
pre-presbyopic	hypermetrope	no	reduced	none
pre-presbyopic	hypermetrope	no	normal	soft
pre-presbyopic	hypermetrope	yes	reduced	none
pre-presbyopic	hypermetrope	yes	normal	none
presbyopic	myope	no	reduced	none
presbyopic	myope	no	normal	none
presbyopic	myope	yes	reduced	none
presbyopic	myope	yes	normal	hard
presbyopic	hypermetrope	no	reduced	none
presbyopic	hypermetrope	no	normal	soft
presbyopic	hypermetrope	yes	reduced	none
presbyopic	hypermetrope	yes	normal	none

Example: contact lens data

- To begin, we seek a rule:

```
If ? then recommendation = hard
```

- Possible tests:

age = young	2/8
age = pre-presbyopic	1/8
age = presbyopic	1/8
spectacle prescription = myope	3/12
spectacle prescription = hypermetrope	1/12
astigmatism = no	0/12
astigmatism = yes	4/12
tear production rate = reduced	0/12
tear production rate = normal	4/12

Create the rule

- Rule with best test added and covered instances:

```
If astigmatism = yes then recommendation = hard
```

Age	Spectacle prescription	Astigmatism	Tear production rate	Recommended lenses
young	myope	yes	reduced	none
young	myope	yes	normal	hard
young	hypermetrope	yes	reduced	none
young	hypermetrope	yes	normal	hard
pre-presbyopic	myope	yes	reduced	none
pre-presbyopic	myope	yes	normal	hard
pre-presbyopic	hypermetrope	yes	reduced	none
pre-presbyopic	hypermetrope	yes	normal	none
presbyopic	myope	yes	reduced	none
presbyopic	myope	yes	normal	hard
presbyopic	hypermetrope	yes	reduced	none
presbyopic	hypermetrope	yes	normal	none

Further refinement

- Current state:

```
If astigmatism = yes and ? then recommendation = hard
```

- Possible tests:

age = young	2/4
age = pre-presbyopic	1/4
age = presbyopic	1/4
spectacle prescription = myope	3/6
spectacle prescription = hypermetrope	1/6
tear production rate = reduced	0/6
tear production rate = normal	4/6

Modified rule and resulting data

- Rule with best test added:

```
If astigmatism = yes and tear production rate = normal
    then recommendation = hard
```

- Instances covered by modified rule:

Age	Spectacle prescription	Astigmatism	Tear production rate	Recommended lenses
young	myope	yes	normal	hard
young	hypermetrope	yes	normal	hard
pre-presbyopic	myope	yes	normal	hard
pre-presbyopic	hypermetrope	yes	normal	none
presbyopic	myope	yes	normal	hard
presbyopic	hypermetrope	yes	normal	none

Further refinement

- Current state:

```
If astigmatism = yes and tear production rate = normal
   and ? then recommendation = hard
```

- Possible tests:

age = young	2/2
age = pre-presbyopic	1/2
age = presbyopic	1/2
spectacle prescription = myope	3/3
spectacle prescription = hypermetrope	1/3

- Tie between the first and the fourth test
  - We choose the one with greater coverage

The result

- Final rule:

```
If astigmatism = yes and tear production rate = normal
   and spectacle prescription = myope then recommendation = hard
```

- Second rule for recommending "hard lenses": (built from instances not covered by first rule)

```
If age = young and astigmatism = yes and
   tear production rate = normal then recommendation = hard
```

- These two rules cover all "hard lenses":
  - Process is repeated with other two classes

Pseudo-code for PRISM

```
For each class C
  Initialize E to the instance set
  While E contains instances in class C
    Create a rule R with an empty left-hand side that predicts class C
    Until R is perfect (or there are no more attributes to use) do
      For each attribute A not mentioned in R, and each value v,
        Consider adding the condition A=v to the LHS of R
        Select A and v to maximize the accuracy p/t
          (break ties by choosing the condition with the largest p)
      Add A=v to R
    Remove the instances covered by R from E
```

Rules vs. decision lists

- PRISM with outer loop generates a decision list for one class
  - Subsequent rules are designed for rules that are not covered by previous rules
  - But: order doesn't matter because all rules predict the same class
- Outer loop considers all classes separately
  - No order dependence implied

Separate and conquer

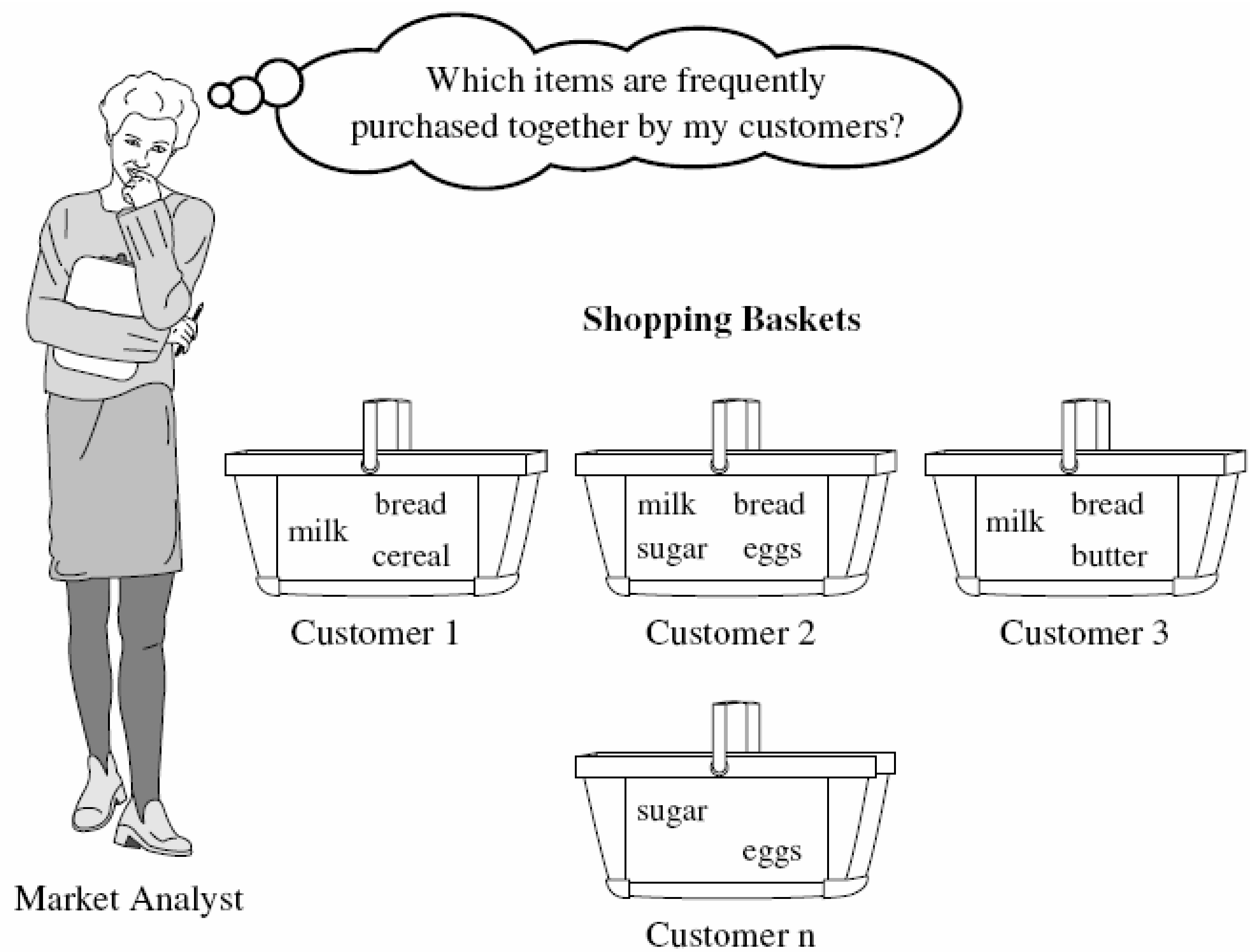
- Methods like PRISM (for dealing with one class) are *separate-and-conquer* algorithms:
  - First, identify a useful rule
  - Then, separate out all the instances it covers
  - Finally, “conquer” the remaining instances
- Difference to divide-and-conquer methods:
  - Subset covered by rule doesn’t need to be explored any further

4.5 Mining association rules

Mining association rules

- Given a set of transactions, find rules that will predict the occurrence of an item based on the occurrences of other items in the transaction
- Broad applications
  - Basket data analysis, cross-marketing, catalog design, sale campaign analysis
  - Web log (click stream) analysis, DNA sequence analysis, etc.

Market basket analysis



Market basket analysis

- **Market-Basket transactions**

TID	*Items*
1	**Bread, Milk**
2	**Bread, Diaper, Beer, Eggs**
3	**Milk, Diaper, Beer, Coke**
4	**Bread, Milk, Diaper, Beer**
5	**Bread, Milk, Diaper, Coke**

- **Example of Association Rules**

  {Diaper} $\rightarrow$ {Beer},
  {Milk, Bread} $\rightarrow$ {Eggs, Coke},
  {Beer, Bread} $\rightarrow$ {Milk},

Definitions: Item set

- *Item*: one test/attribute-value pair (e.g. Milk, Bread)
- *Item set*: A collection of one or more items (e.g. {Milk, Bread, Diaper})
- k-itemset: An itemset that contains k items
- Support count: Frequency of occurrence of an itemset
- Frequent Itemset: An itemset whose support count is greater than or equal to a *minsup*

Definition: Association Rule

- **Association Rule**
  - An implication expression of the form X → Y, where X and Y are itemsets
  - Example: {Milk, Diaper} → {Beer}
- **Rule Evaluation Metrics**
  - Support (s): Fraction of transactions that contain both X and Y
  - Confidence (c): Measures how often items in Y appear in transactions that contain X

$$support(A \Rightarrow B) = P(A \cup B)$$

$$confidence(A \Rightarrow B) = P(B|A)$$

$$confidence(A \Rightarrow B) = P(B|A) = \frac{support(A \cup B)}{support(A)} = \frac{support\_count(A \cup B)}{support\_count(A)}$$

Definition: Association Rule

- Example:

$$\{\text{Milk}, \text{Diaper}\} \Rightarrow \text{Beer}$$

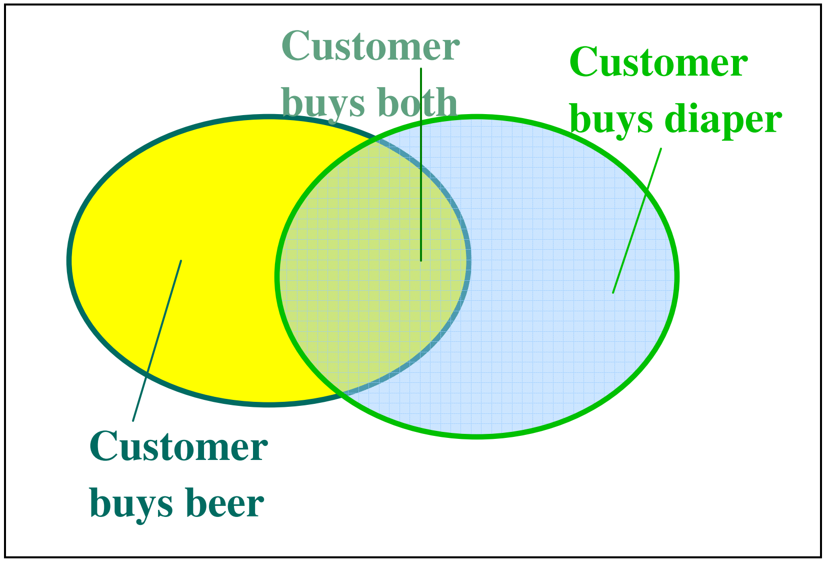
$$s = \frac{2}{5} = 0.4$$

$$c = \frac{2}{3} = 0.67$$

TID	*Items*
1	**Bread, Milk**
2	**Bread, Diaper, Beer, Eggs**
3	**Milk, Diaper, Beer, Coke**
4	**Bread, Milk, Diaper, Beer**
5	**Bread, Milk, Diaper, Coke**

Association Rules

- Itemset X={x1, …, xk}
- Find all the rules *X→Y* with min confidence and support
  - Support, *s*, probability that a transaction contains $X \cup Y$
  - Confidence, *c,* conditional probability that a transaction having X also contains *Y*.



$$\{\text{Diaper}\} \Rightarrow \text{Beer}$$

Example

Transaction-id	Items bought
10	A, B, C
20	A, C
30	A, D
40	B, E, F

- *Let min_support = 50%, min_conf = 50%:*
  - *A* → *C* (50%, 66.7%)
  - *C* → *A* (50%, 100%)

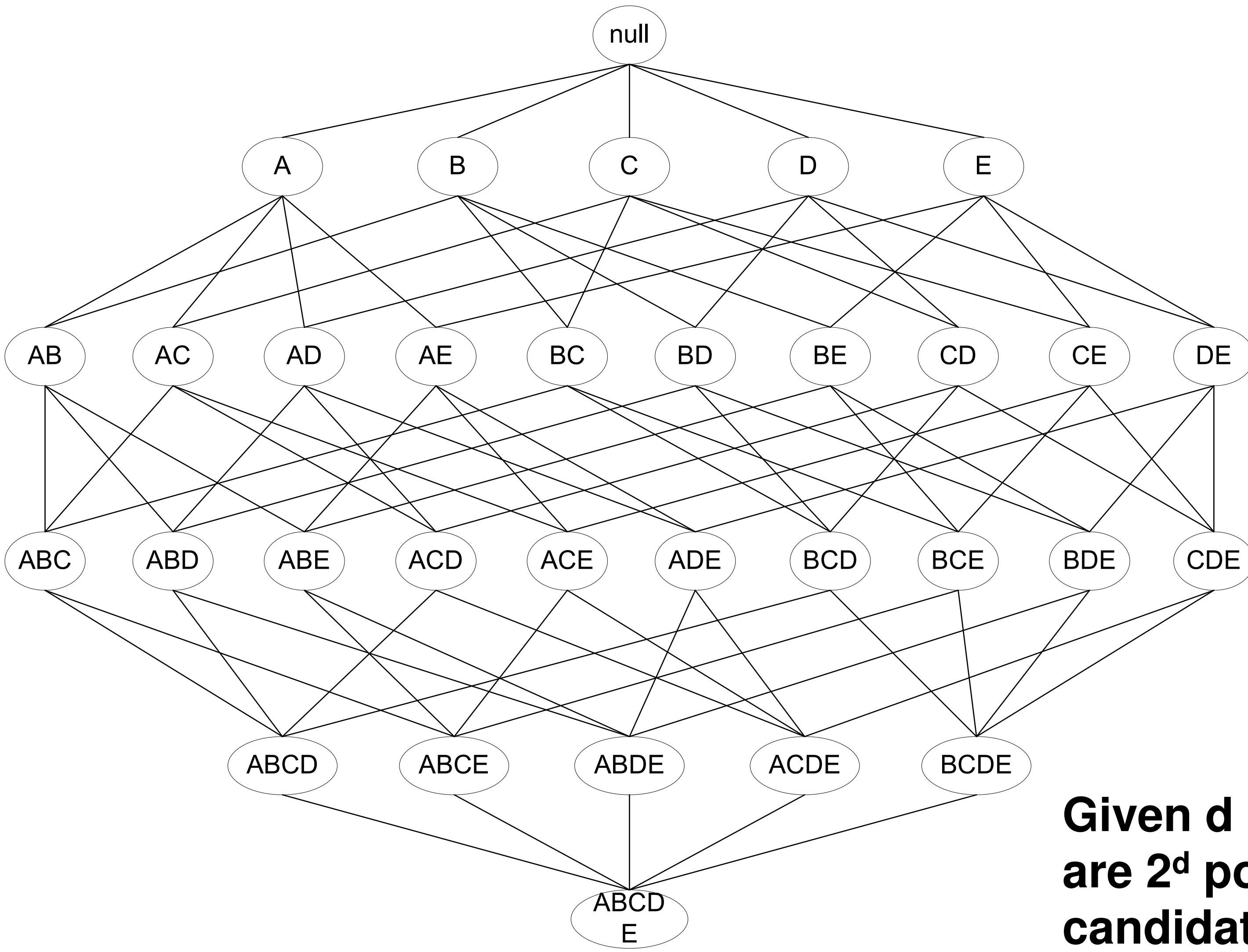
Association Rule Mining Task

- Given a set of transactions T, the goal of association rule mining is to find all rules having
  - support ≥ *minsup* threshold
  - confidence ≥ *minconf* threshold

- Brute-force approach:
  - List all possible association rules
  - Compute the support and confidence for each rule
  - Prune rules that fail the *minsup* and *minconf* thresholds

  ⇒ Problem: Computational complexity!

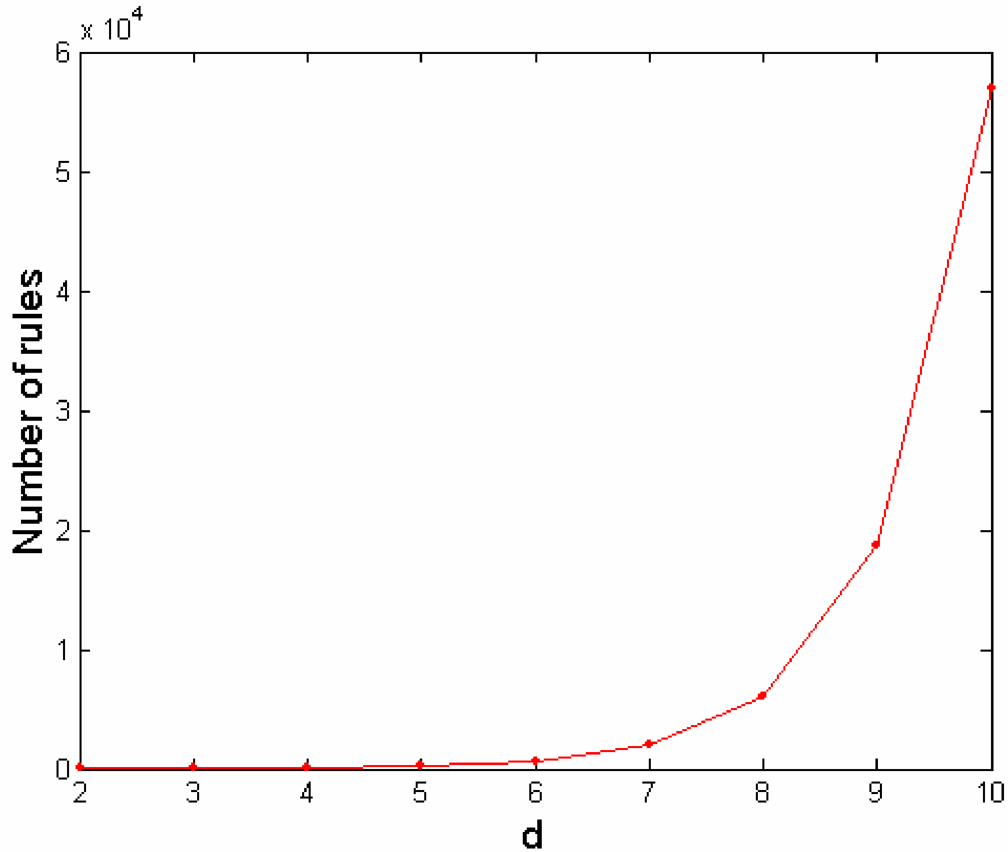
Frequent Itemset Generation



Given d items, there are 2^d possible candidate itemsets

Computational Complexity

- Given d unique items:
  - Total number of itemsets = 2^d
  - Total number of possible association rules:



$$R = \sum_{k=1}^{d-1}\left[\binom{d}{k}\times\sum_{j=1}^{d-k}\binom{d-k}{j}\right]$$
$$= 3^d - 2^{d+1} + 1$$

If d=6, R = 602 rules

Apriori Algorithm

- Let k=1
- Generate frequent itemsets of length 1
- Repeat until no new frequent itemsets are identified
  - Generate length (k+1) candidate itemsets from length k frequent itemsets
  - Prune candidate itemsets containing subsets of length k that are infrequent
  - Count the support of each candidate by scanning the dataset
  - Eliminate candidates that are infrequent, leaving only those that are frequent

The Apriori Algorithm—An Example

$Sup_{min} = 2$

Database TDB

Tid	Items
10	A, C, D
20	B, C, E
30	A, B, C, E
40	B, E

1st scan → C_1

Itemset	sup
{A}	2
{B}	3
{C}	3
{D}	1
{E}	3

→ L_1

Itemset	sup
{A}	2
{B}	3
{C}	3
{E}	3

C_2

Itemset
{A, B}
{A, C}
{A, E}
{B, C}
{B, E}
{C, E}

2nd scan → C_2

Itemset	sup
{A, B}	1
{A, C}	2
{A, E}	1
{B, C}	2
{B, E}	3
{C, E}	2

→ L_2

Itemset	sup
{A, C}	2
{B, C}	2
{B, E}	3
{C, E}	2

C_3

Itemset
{B, C, E}

3rd scan → L_3

Itemset	sup
{B, C, E}	2

The Apriori Algorithm

- Pseudo-code:

C_k: Candidate itemset of size k
L_k : frequent itemset of size k

L_1 = {frequent items};
for (k = 1; L_k !=∅; k++) **do begin**
  C_{k+1} = candidates generated from L_k;
  for each transaction t in database do
      increment the count of all candidates in C_{k+1}
    that are contained in t
  L_{k+1} = candidates in C_{k+1} with min_support
  end
return $\cup_k L_k$;

Important Details of Apriori

- How to generate candidates?
  - Step 1: self-joining L_k
  - Step 2: pruning
- How to count supports of candidates?
- Example of Candidate-generation
  - L_3={*abc, abd, acd, ace, bcd*}
  - Self-joining: L_3*L_3
    - *abcd* from *abc* and *abd*
    - *acde* from *acd* and *ace*
  - Pruning:
    - *acde* is removed because *ade* is not in L_3
  - C_4={*abcd*}

Weather data

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

Item sets for weather data

One-item sets	Two-item sets	Three-item sets	Four-item sets
outlook = sunny (5)	outlook = sunny temperature = mild (2)	outlook = sunny temperature = hot humidity = high (2)	outlook = sunny temperature = hot humidity = high play = no (2)
outlook = overcast (4)	outlook = sunny temperature = hot (2)	outlook = sunny temperature = hot play = no (2)	outlook = sunny humidity = high windy = false play = no (2)
outlook = rainy (5)	outlook = sunny humidity = normal (2)	outlook = sunny humidity = normal play = yes (2)	outlook = overcast temperature = hot windy = false play = yes (2)
……	……	……	……

- In total: 12 one-item sets, 47 two-item sets, 39 Three-item sets, 6 four-item sets and 0 five-item sets (with minimum support of two)

Generating rules from an item set

- Once all item sets with minimum support have been generated, we can turn them into rules

```
humidity = normal, windy = false, play = yes
```

- Seven potential rules:

If humidity = normal and windy = false then play = yes	4/4
If humidity = normal and play = yes then windy = false	4/6
If windy = false and play = yes then humidity = normal	4/6
If humidity = normal then windy = false and play = yes	4/7
If windy = false then humidity = normal and play = yes	4/8
If play = yes then humidity = normal and windy = false	4/9
If - then humidity = normal and windy = false and play = yes	4/12

Rules for weather data

- Rules with support > 1 and confidence = 100%:

	Association rule			Coverage	Accuracy
1	humidity = normal windy = false	⇒	play = yes	4	100%
2	temperature = cool	⇒	humidity = normal	4	100%
3	outlook = overcast	⇒	play = yes	4	100%
4	temperature = cool play = yes	⇒	humidity = normal	3	100%
5	outlook = rainy windy = false	⇒	play = yes	3	100%
6	outlook = rainy play = yes	⇒	windy = false	3	100%
7	outlook = sunny humidity = high	⇒	play = no	3	100%
8	outlook = sunny play = no	⇒	humidity = high	3	100%
9	temperature = cool windy = false	⇒	humidity = normal play = yes	2	100%

- In total: 3 rules with support four, 5 with support three, 50 with support two

Example rules from the same set

- Item set:

temperature = cool, humidity = normal, windy = false, play = yes

- Resulting rules (all with 100% confidence):

temperature = cool windy = false	⇒	humidity = normal play = yes
temperature = cool humidity = normal windy = false	⇒	play = yes
temperature = cool windy = false play = yes	⇒	humidity = normal

- Three subsets of this item set also have coverage 2:

temperature = cool, windy = false

temperature = cool, humidity = normal, windy = false

temperature = cool, windy = false, play = yes

Generating rules efficiently

- We are looking for all high-confidence rules
  - But: rough method is (2^N-1)
- Better way: building (*c* + 1) consequent rules from *c* consequent ones
  - Observation: (*c* + 1) consequent rule can only hold if all corresponding *c* consequent rules also hold
- Resulting algorithm similar to procedure for large item sets

Example

- 1 consequent rules:

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

- Corresponding 2 consequent rule:

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

4.6 Linear models

Linear regression

- Work most naturally with numeric attributes
- Standard technique for numeric prediction
- Linear regression: Data are modeled to fit a straight line
- Linear regression involves a response variable y and a single predictor variable x

  $y = w_0 + w_1 x$

  – where w_0 (y-intercept) and w_1 (slope) are regression coefficients

  – Two regression coefficients, *w* and *b*, specify the line

Linear regression

- Method of least squares: estimates the best-fitting straight line

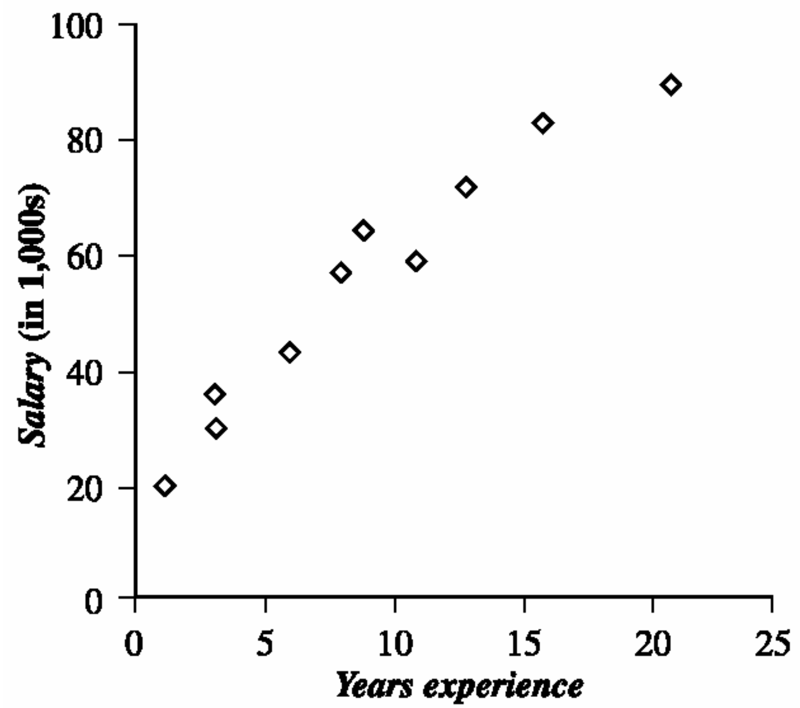
$$w_1 = \frac{\sum_{i=1}^{|D|}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{|D|}(x_i - \bar{x})^2}$$

$$w_0 = \bar{y} - w_1\bar{x}$$

- *D:* a training set consisting of values of predictor variable
- */D/* data points of the form(*x*1, *y*1), (*x*2, *y*2),…, (*x/D/*, *y/D/*).
- where *x* is the mean value of *x*1, *x*2, : : : , *x/D/*, and *y* is the mean value of *y*1, *y*2, : : : , *y/D/*.

Example: Salary data

x years experience	y salary (in $1000s)
3	30
8	57
9	64
13	72
3	36
6	43
11	59
21	90
1	20
16	83



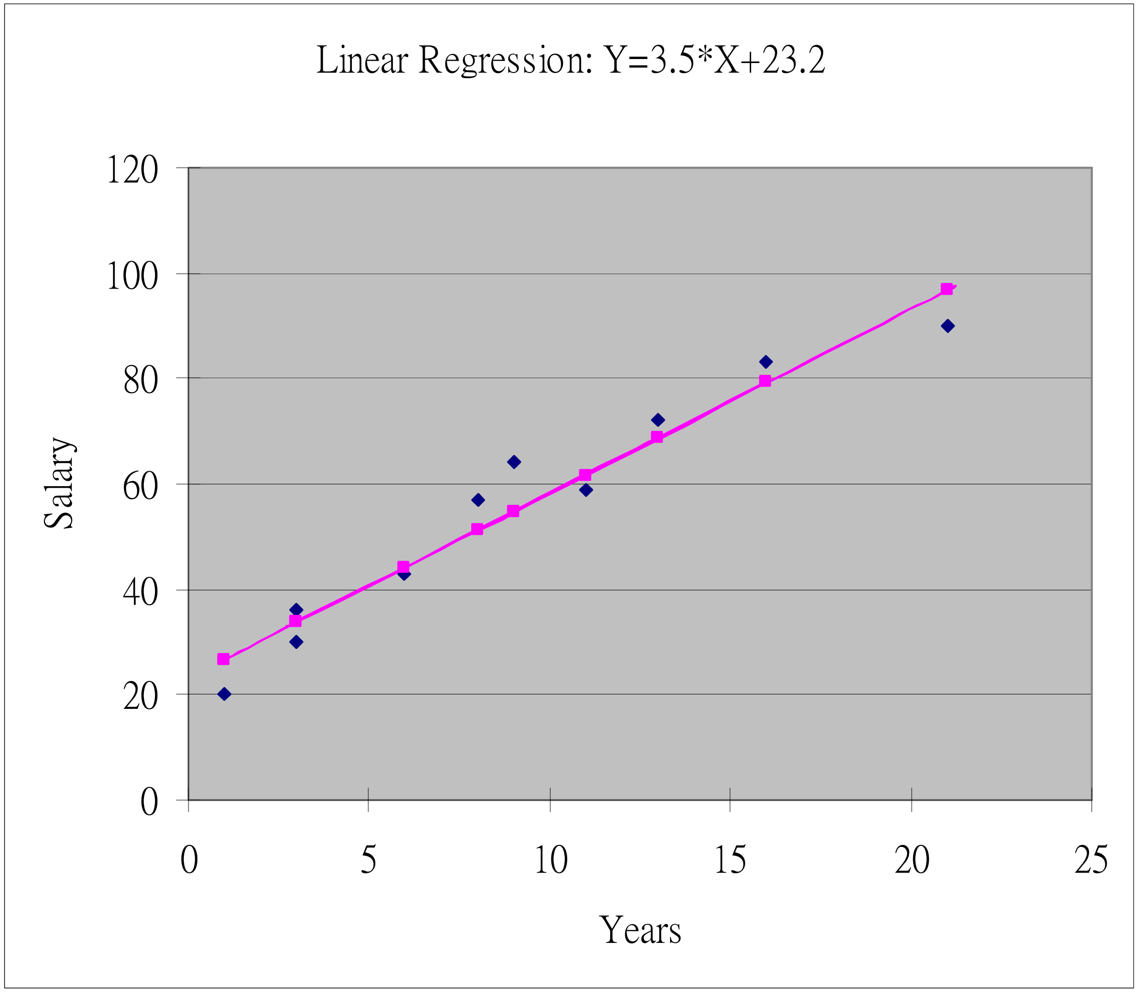
$\bar{x} = 9.1$ and $\bar{y} = 55.4$

$$w_1 = \frac{(3-9.1)(30-55.4) + (8-9.1)(57-55.4) + \cdots + (16-9.1)(83-55.4)}{(3-9.1)^2 + (8-9.1)^2 + \cdots + (16-9.1)^2} = 3.5$$

$$w_0 = 55.4 - (3.5)(9.1) = 23.6$$

$$y = 23.6 + 3.5x$$

Example: Salary data



Multiple linear regression

- Multiple linear regression involves more than one predictor variable
- Training data is of the form $(\mathbf{X_1}, y_1), (\mathbf{X_2}, y_2), \ldots, (\mathbf{X_{|D|}}, y_{|D|})$
- where the $\boldsymbol{X_i}$ are the n-dimensional training data with associated class labels, y_i
- An example of a multiple linear regression model based on two predictor attributes:

$$y = w_0 + w_1 x_1 + w_2 x_2$$

Linear Regression: CPU performance data

	Cycle time (ns) MYCT	Main memory (KB) Min. MMIN	Max. MMAX	Cache (KB) CACH	Channels Min. CHMIN	Max. CHMAX	Performance PRP
1	125	256	6000	256	16	128	198
2	29	8000	32000	32	8	32	269
3	29	8000	32000	32	8	32	220
4	29	8000	32000	32	8	32	172
5	29	8000	16000	32	8	16	132
...							
207	125	2000	8000	0	2	14	52
208	480	512	8000	32	0	0	67
209	480	1000	4000	0	0	0	45

$$\text{PRP} = -55.9 + 0.0489\ \text{MYCT} + 0.0153\ \text{MMIN} + 0.0056\ \text{MMAX} + 0.6410\ \text{CACH} - 0.2700\ \text{CHMIN} + 1.480\ \text{CHMAX}.$$

4.7 k-nearest neighbor algorithm

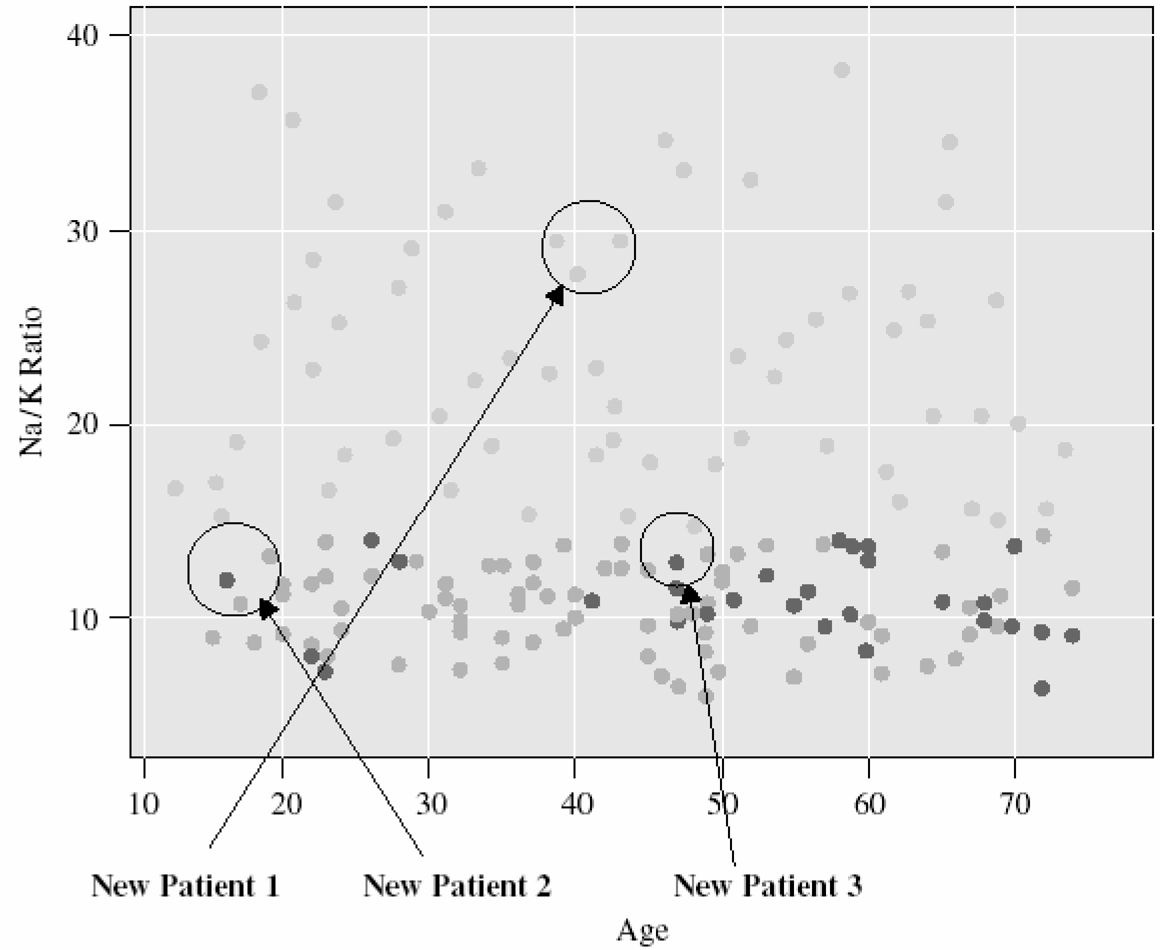
Example Problem: Face Recognition

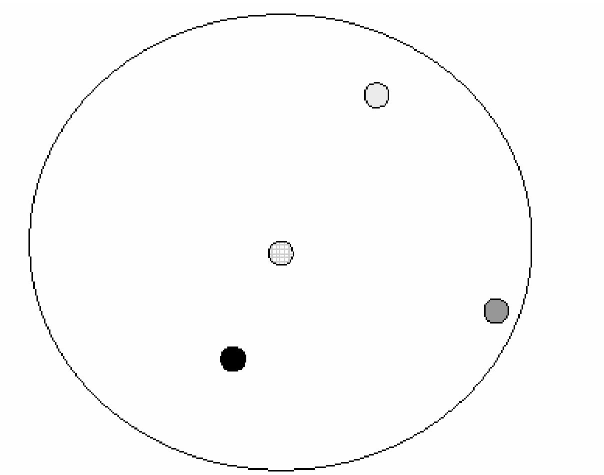
- We have a database of (say) 1 million face images
- We are given a new image and want to find the most similar images in the database
- Represent faces by (relatively) invariant values, e.g., ratio of nose width to eye width
- Each image represented by a large number of numerical features
- **Problem:** given the features of a new face, find those in the DB that are close in at least ¾ (say) of the features

k-nearest neighbor algorithm

- *k*-Nearest neighbor is an example of *instance-based learning*
- Distance function defines what's learned
- A classification for a new unclassified record may be found simply by comparing it to the most similar records in the training set
- Example:
  – We are interested in classifying the type of drug a patient should be prescribed
  – Based on the age of the patient and the patient's sodium/potassium ratio (Na/K)
  – Dataset includes 200 patients

Scatter plot



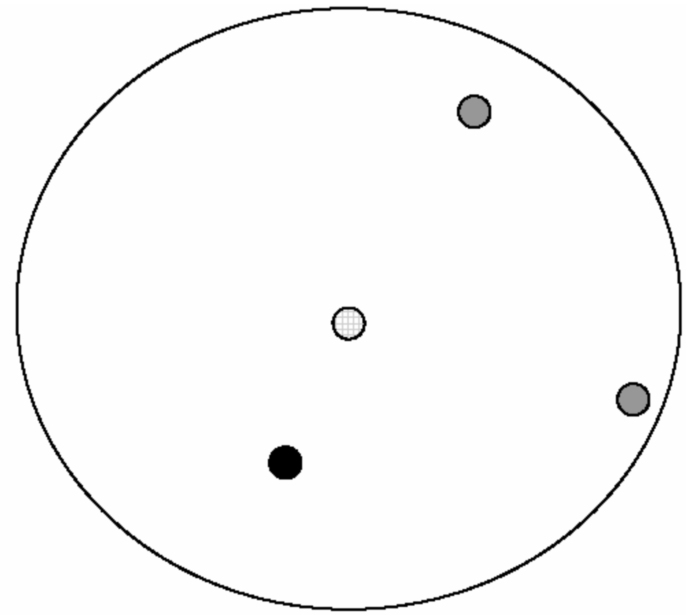


Close-up of three nearest neighbors to new patient 2.

On the scatter plot; light gray points indicate drug Y; medium gray points indicate drug A or X; dark gray points indicate drug B or C

Close-up of neighbors to new patient 2

- *k*=1 => drugs B and C (dark gray)
- k=2 => ?
- K=3 => drugs A and X (medium gray)



- Main questions:
  – How many neighbors should we consider? That is, what is *k*?
  – How do we measure distance?
  – Should all points be weighted equally, or should some points have more influence than others?

Instance-based learning

- Most instance-based schemes use *Euclidean distance*:

$$\sqrt{(a_1^{(1)} - a_1^{(2)})^2 + (a_2^{(1)} - a_2^{(2)})^2 + \ldots + (a_k^{(1)} - a_k^{(2)})^2}$$

- $\mathbf{a}^{(1)}$ and $\mathbf{a}^{(2)}$: two instances with *k* attributes
- Taking the square root is not required when comparing distances
- Other popular metric: Manhattan or *city-block metric*
  – Taking absolute differences value without squaring them

Normalization and other issues

- Different attributes are measured on different scales, need to be *normalized*:

$$a_i = \frac{v_i - \min v_i}{\max v_i - \min v_i}$$

  v_i : the actual value of attribute *i*

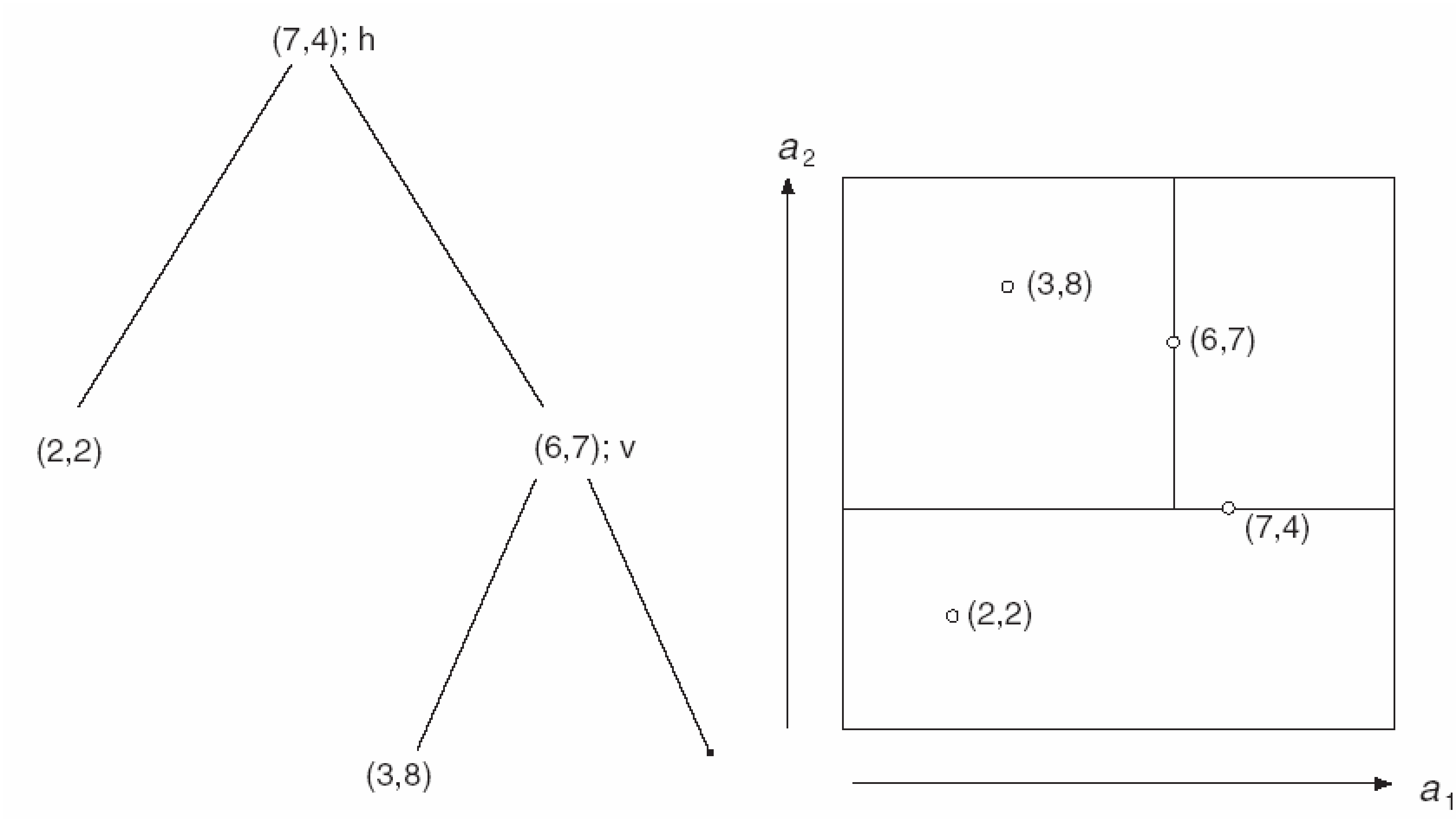
  all attribute values lie between 0 and 1

- Nominal attributes: distance either 0 or 1
- Common policy for missing values: assumed to be maximally distant (given normalized attributes)

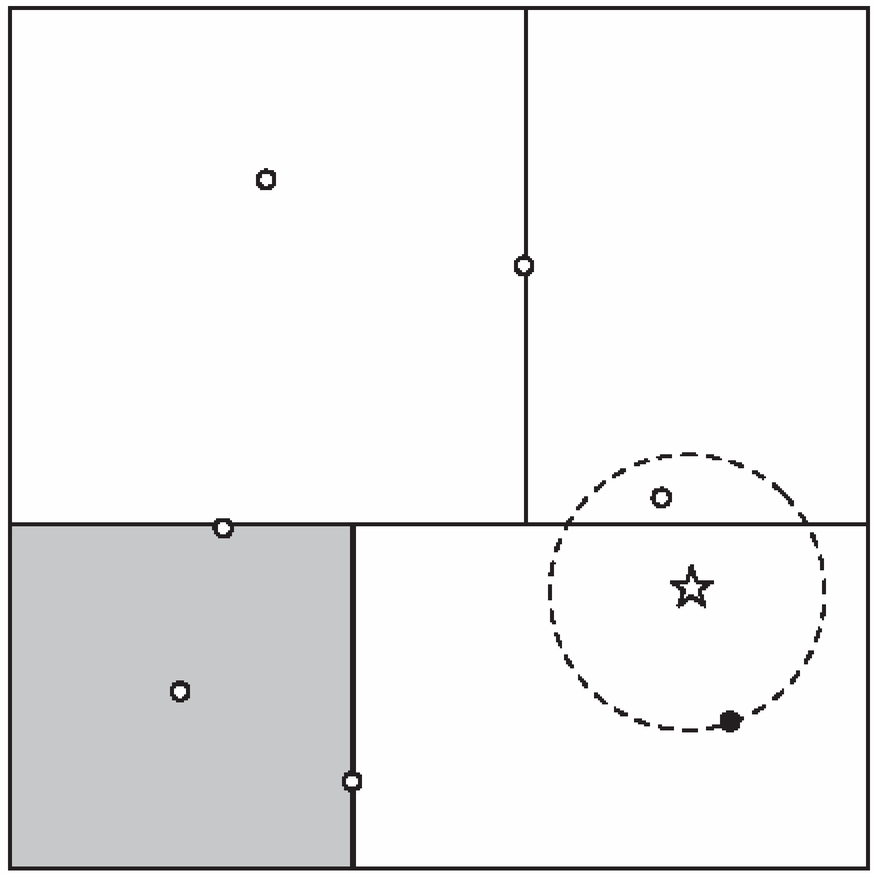
Finding nearest neighbors efficiently

- Simplest way of finding nearest neighbor: linear scan of the data
  - Classification takes time proportional to the product of the number of instances in training and test sets
- Nearest-neighbor search can be done more efficiently using appropriate data structures
- There two methods that represent training data in a tree structure:
  - *kD-trees (k-dimensional trees)*
  - *Ball trees*

*k*D-tree example



Using *k*D-trees: example



More on *k*D-trees

- Complexity depends on depth of tree, given by base 2 logarithm of number of nodes
- Amount of backtracking required depends on quality of tree
- How to build a good tree? Need to find good split point and split direction
  - Split direction: direction with greatest variance
  - Split point: median value or value closest to mean along that direction
- Can apply this recursively

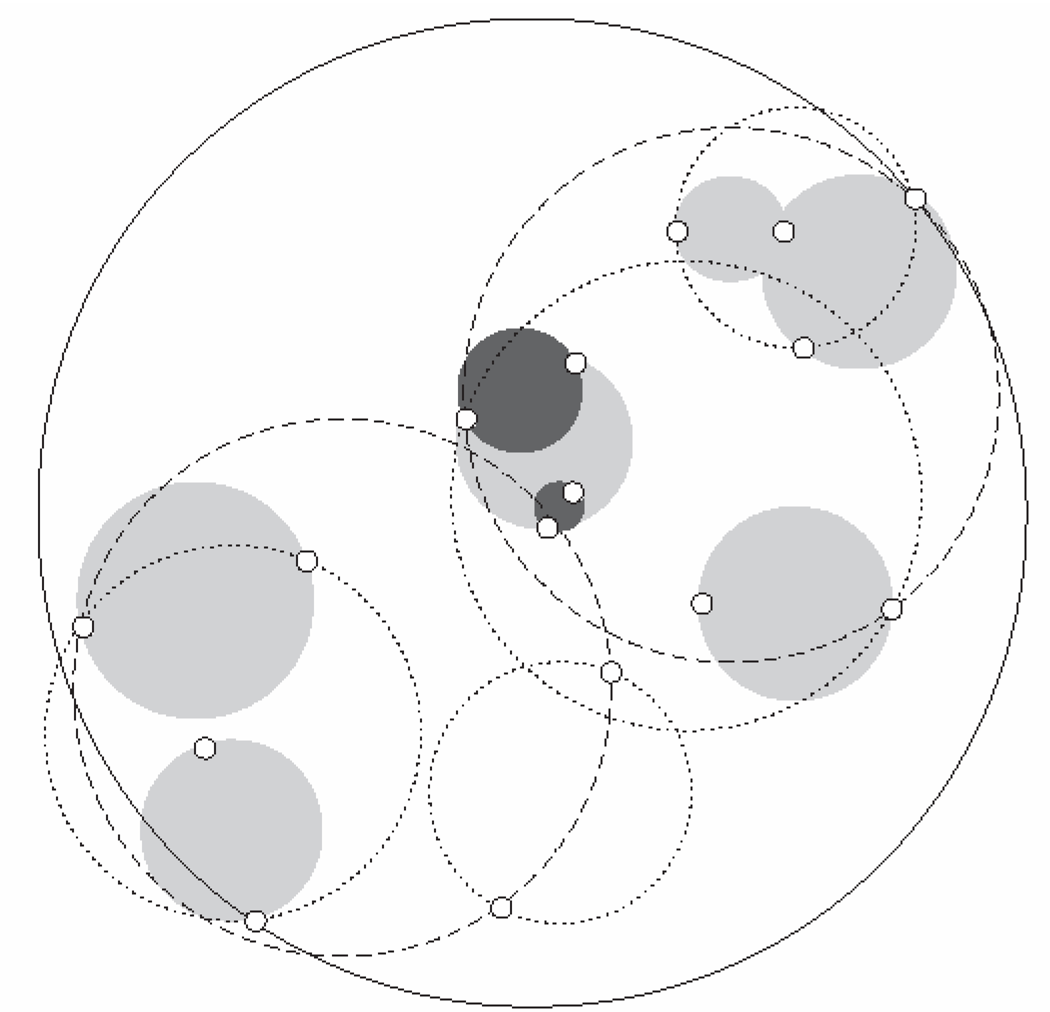
Building trees incrementally

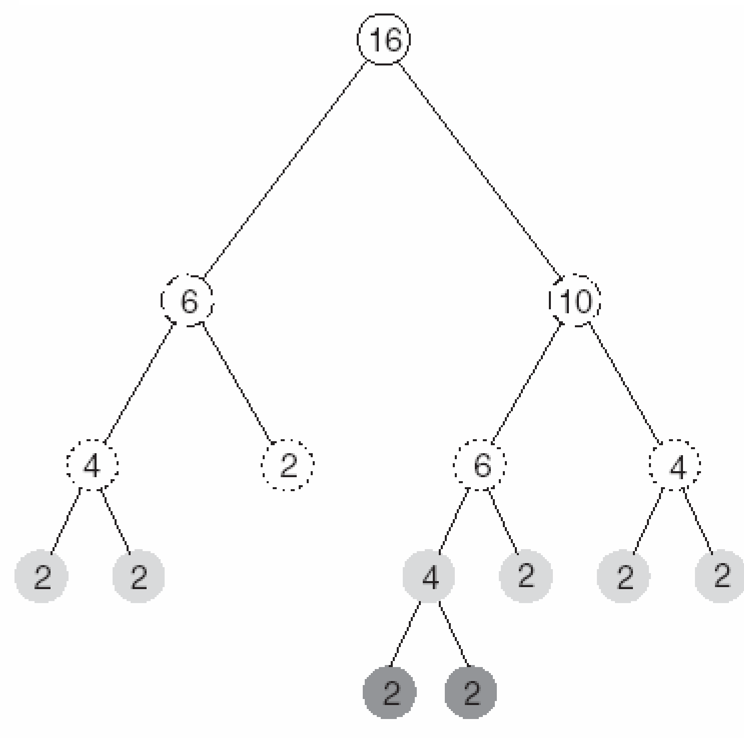
- Big advantage of instance-based learning: classifier can be updated incrementally
  - Just add new training instance!
- We can do the same with *k*D-trees
- Heuristic strategy:
  - Find leaf node containing new instance
  - Place instance into leaf if leaf is empty
  - Otherwise, split leaf
- Tree should be rebuilt occasionally

Ball trees

- Problem in *k*D-trees: corners
- Can use balls (hyperspheres) instead of hyperrectangles
  - no need to make sure that regions don't overlap
  - A *ball tree* organizes the data into a tree of *k*-dimensional hyperspheres
  - Normally allows for a better fit to the data and thus more efficient search

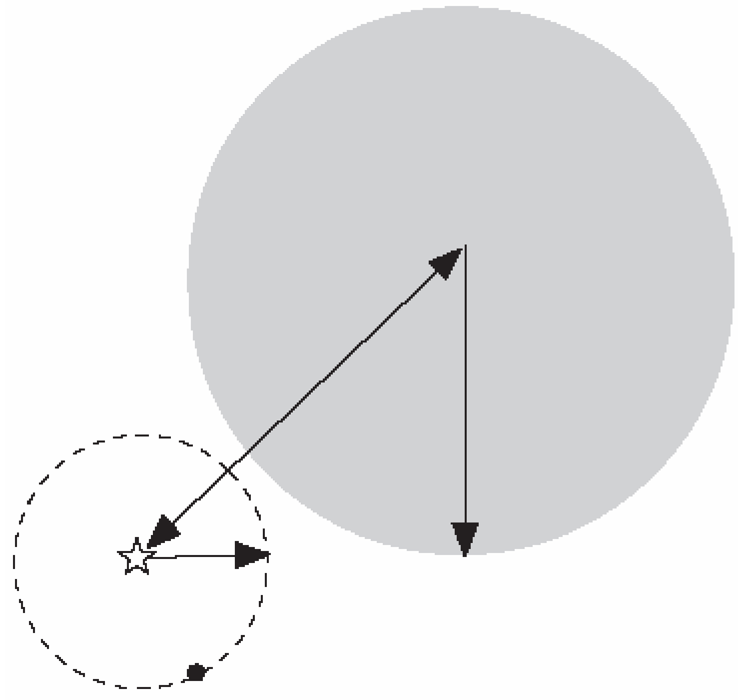
Ball tree for 16 training instances





Using ball trees

- Nearest-neighbor search is done using the same backtracking strategy as in *k*D-trees
- Ball can be ruled out from consideration if: distance from target to ball's center exceeds ball's radius plus current upper bound



Building ball trees

- Ball trees are built top down (like *k*D-trees)
- Don't have to continue until leaf balls contain just two points: can enforce minimum occupancy (same in *k*D-trees)
- Basic problem: splitting a ball into two
- Simple (linear-time) split selection strategy:
  - Choose point farthest from ball's center
  - Choose second point farthest from first one
  - Assign each point to these two points
  - Compute cluster centers and minimum radius based on the two subsets to get two balls

4.8 Clustering: k-means method

Example: Clustering Documents

- Represent a document by a vector $(x_1, x_2, \ldots, x_k)$, where $x_i = 1$ if the i^{th} word (in some order) appears in the document.
- Documents with similar sets of words may be about the same topic.

Clustering

- Clustering techniques apply when there is no class to be predicted
- Aim: divide instances into "natural" groups
- As we've seen clusters can be:
  - disjoint vs. overlapping
  - deterministic vs. probabilistic
  - flat vs. hierarchical
- We'll look at a classic clustering algorithm called *k-means*
  - *K-means* clusters are disjoint, deterministic, and flat

Examples of Clustering Applications

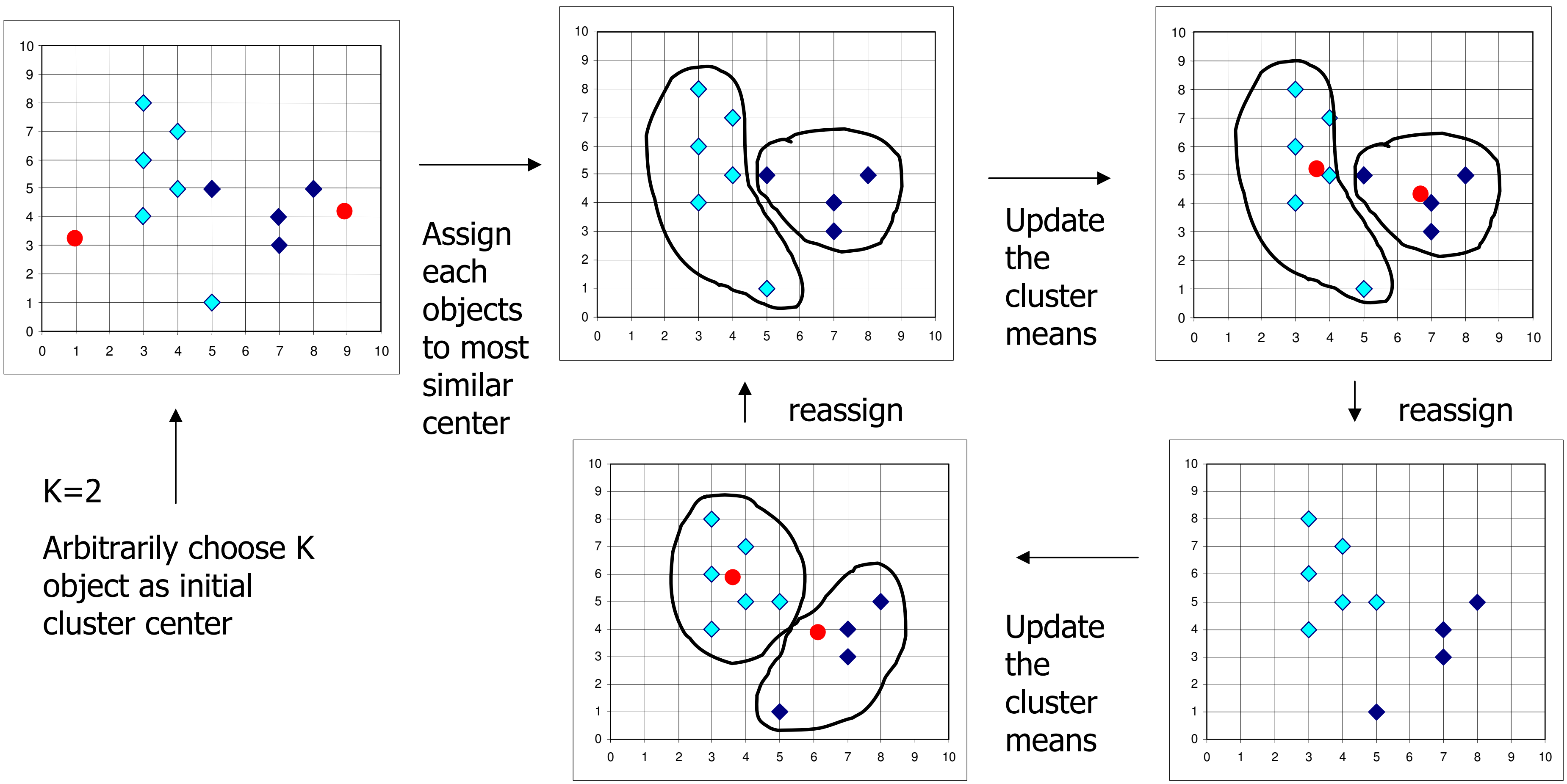
- Marketing: Help marketers discover distinct groups in their customer bases, and then use this knowledge to develop targeted marketing programs
- Land use: Identification of areas of similar land use in an earth observation database
- Insurance: Identifying groups of motor insurance policy holders with a high average claim cost
- City-planning: Identifying groups of houses according to their house type, value, and geographical location
- Documenting: with similar sets of words may be about the same topic

The *k*-means algorithm

To cluster data into *k* groups:
(*k* is predefined)

1. Choose *k* cluster centers
   – e.g. first time at random, then mean point
2. Assign instances to clusters
   – based on distance to cluster centers with the nearest point
3. Compute *centroids or mean* of clusters and they are taken to be new center values
4. Go to step 1
   – until convergence

Example: The *K-Means* Clustering Method



The criterion function

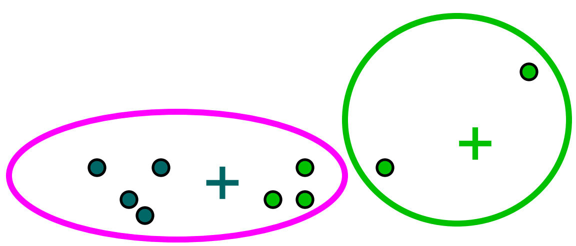
- The square-error criterion

$$E = \sum_{i=1}^{k} \sum_{p \in C_i} |\boldsymbol{p} - \boldsymbol{m_i}|^2$$

  – where *E* is the sum of the square error for all objects in the data set;
  – ***p*** is the point in space representing a given object; and
  – $\boldsymbol{m_i}$ is the mean of cluster *Ci* (both ***p*** and $\boldsymbol{m_i}$ are multidimensional)

Weakness of K-means method

- Often terminate at a local optimum, The *global optimum* may be found using techniques such as: *deterministic annealing* and *genetic algorithms*
- Applicable only when *mean* is defined, then what about categorical data?
- Need to specify *k,* the *number* of clusters, in advance
- Unable to handle noisy data and *outliers*



The end of

Chapter 4: Algorithms: The Basic Methods