# Algorithms: The basic methods

Most of these slides (used with permission) are based on the book:

*Data Mining: Practical Machine Learning Tools and Techniques*
by I. H. Witten, E. Frank, M. A. Hall, and C. J. Pal

1

## Algorithms: The basic methods

- Inferring rudimentary rules
- Simple probabilistic modeling
- Constructing decision trees
- Constructing rules
- Association rule learning
- Linear models
- Clustering

2

2

## Simplicity first

- Simple algorithms often work very well!
- There are many kinds of simple structure, e.g.:
  - One attribute does all the work
  - All attributes contribute equally & independently
  - Logical structure with a few attributes suitable for tree
  - A set of simple logical rules
  - Relationships between groups of attributes
  - A weighted linear combination of the attributes
  - Strong neighborhood relationships based on distance
  - Clusters of data in unlabeled data
  - Bags of instances that can be aggregated
- Success of method depends on the domain

3

3

## Inferring rudimentary rules

- 1R rule learner: learns a 1-level decision tree
  - A set of rules that all test one particular attribute that has been identified as the one that yields the lowest classification error
- Basic version for finding the rule set from a given training set (assumes nominal attributes):
  - For each attribute
    - Make one branch for each value of the attribute
    - To each branch, assign the most frequent class value of the instances pertaining to that branch
    - Error rate: proportion of instances that do not belong to the majority class of their corresponding branch
  - Choose attribute with lowest error rate

4

4

## Pseudo-code for 1R

```
For each attribute,
For each value of the 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
```

- 1R's handling of missing values: a missing value is treated as a separate attribute value

5

5

## Evaluating the weather attributes

| Outlook | Temp | 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 |

| Attribute | Rules | Errors | Total errors |
|---|---|---|---|
| Outlook | Sunny → No | 2/5 | 4/14 |
| | Overcast → Yes | 0/4 | |
| | Rainy → Yes | 2/5 | |
| Temp | Hot → No* | 2/4 | 5/14 |
| | Mild → Yes | 2/6 | |
| | Cool → Yes | 1/4 | |
| Humidity | High → No | 3/7 | 4/14 |
| | Normal → Yes | 1/7 | |
| Windy | False → Yes | 2/8 | 5/14 |
| | True → No* | 3/6 | |

* indicates a tie

6

6

## Dealing with numeric attributes

- Idea: discretize numeric attributes into sub ranges (intervals)
- How to divide each attribute's overall range into intervals?
  - Sort instances according to attribute's values
  - Place breakpoints where (majority) class changes
  - This minimizes the total classification error
- 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
```

| Outlook | Temperature | Humidity | Windy | Play |
|---|---|---|---|---|
| Sunny | 85 | 85 | False | No |
| Sunny | 80 | 90 | True | No |
| Overcast | 83 | 86 | False | Yes |
| Rainy | 75 | 80 | False | Yes |
| ... | ... | ... | ... | ... |

7

7

## The problem of overfitting

- Discretization procedure is very sensitive to noise
  - A single instance with an incorrect class label will probably produce a separate interval
- Also, something like a *time stamp* attribute will have zero errors
- Simple solution:
  *enforce minimum number of instances in majority class per interval*
- Example: *temperature* attribute with required minimum number of instances in majority class set to three:

```
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

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
```

8

8

## Results with overfitting avoidance

- Resulting rule sets for the four attributes in the weather data, with only two rules for the temperature attribute:

| 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 | |

9

9

## Discussion of 1R

- 1R was described in a paper by Holte (1993):

  **Very Simple Classification Rules Perform Well on Most Commonly Used Datasets**

  Robert C. Holte, Computer Science Department, University of Ottawa

  - Contains an experimental evaluation on 16 datasets (using *cross-validation* to estimate classification accuracy on fresh data)
  - Required minimum number of instances in majority class was set to 6 after some experimentation
  - 1R's simple rules performed not much worse than much more complex decision trees
- Lesson: simplicity first can pay off on practical datasets
- Note that 1R does not perform as well on more recent, more sophisticated benchmark datasets

10

10

## Simple probabilistic modeling

- "Opposite" of 1R: use all the attributes
- Two assumptions: Attributes are
  - *equally important*
  - *statistically independent* (given the class value)
    - This means knowing the value of one attribute tells us nothing about the value of another takes on (if the class is known)
- Independence assumption is almost never correct!
- But … this scheme often works surprisingly well in practice
- The scheme is easy to implement in a program and very fast
- It is known as *naïve Bayes*

11

11

## Probabilities for weather data

| 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 | | | | | | | | |

| Outlook | Temp | 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 |

12

## Probabilities for weather data

| 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 | | | | | | | | |

- A new day:

| Outlook | Temp. | Humidity | Windy | Play |
|---|---|---|---|---|
| Sunny | Cool | High | True | ? |

Likelihood of the two classes

For "yes" = $2/9 \times 3/9 \times 3/9 \times 3/9 \times 9/14 = 0.0053$

For "no" = $3/5 \times 1/5 \times 4/5 \times 3/5 \times 5/14 = 0.0206$

Conversion into a probability by normalization:

P("yes") = 0.0053 / (0.0053 + 0.0206) = 0.205

P("no") = 0.0206 / (0.0053 + 0.0206) = 0.795

13

13

## Can combine probabilities using Bayes's rule

- Famous rule from probability theory due to

**Thomas Bayes**

**Born: 1702 in London, England**
**Died: 1761 in Tunbridge Wells, Kent, England**

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

$$P(H \mid E) = P(E \mid H)P(H) / P(E)$$

- *A priori* probability of *H* : $P(H \mid E)$
  - Probability of event *before* evidence is seen
- *A posteriori* probability of *H* : $P(H)$
  - Probability of event *after* evidence is seen

14

14

## Naïve Bayes for classification

- Classification learning: what is the probability of the class given an instance?
  - Evidence *E* = instance's non-class attribute values
  - Event *H* = class value of instance
- Naïve assumption: evidence splits into parts (i.e., attributes) that are conditionally *independent*
- This means, given *n* attributes, we can write Bayes' rule using a product of per-attribute probabilities:

$$P(H \mid E) = P(E_1 \mid H)P(E_3 \mid H) \cdots P(E_n \mid H)P(H) / P(E)$$

15

15

## Weather data example

| Outlook | Temp. | Humidity | Windy | Play |
|---|---|---|---|---|
| Sunny | Cool | High | True | ? |

← ***Evidence E***

$$P(yes \mid E) = P(Outlook = Sunny \mid yes)$$
$$P(Temperature = Cool \mid yes)$$
$$P(Humidity = High \mid yes)$$
$$P(Windy = True \mid yes)$$
$$P(yes) / P(E)$$
$$= \frac{2/9 \times 3/9 \times 3/9 \times 3/9 \times 9/14}{P(E)}$$

***Probability of class "yes"***

16

16

## Missing values

- Training: instance is not included in frequency count for attribute value-class combination
- Classification: attribute will be omitted from calculation
- Example:

| Outlook | Temp. | Humidity | Windy | Play |
|---|---|---|---|---|
| ? | Cool | High | True | ? |

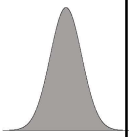
Likelihood of "yes" = 3/9 × 3/9 × 3/9 × 9/14 = 0.0238
Likelihood of "no" = 1/5 × 4/5 × 3/5 × 5/14 = 0.0343
P("yes") = 0.0238 / (0.0238 + 0.0343) = 41%
P("no") = 0.0343 / (0.0238 + 0.0343) = 59%

17

17

## Numeric attributes

- Usual assumption: attributes have a *normal* or *Gaussian* probability distribution (given the class)
- The *probability density function* for the normal distribution is defined by two parameters:
  - *Sample mean* $\mu = \frac{1}{N}\sum_{i=1}^{N} x_i$
  - *Standard deviation* $\sigma = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \mu)^2}$
  - Then the density function *f(x) is* $f(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$



18

18

## Statistics for weather data

| Outlook | | | Temperature | | Humidity | | Windy | | | Play | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | *Yes* | *No* | *Yes* | *No* | *Yes* | *No* | | *Yes* | *No* | *Yes* | *No* |
| Sunny | 2 | 3 | 64, 68, | 65,71, | 65, 70, | 70, 85, | False | 6 | 2 | 9 | 5 |
| Overcast | 4 | 0 | 69, 70, | 72,80, | 70, 75, | 90, 91, | True | 3 | 3 | | |
| Rainy | 3 | 2 | 72, … | 85, … | 80, … | 95, … | | | | | |
| Sunny | 2/9 | 3/5 | $\mu$ =73 | $\mu$ =75 | $\mu$ =79 | $\mu$ =86 | False | 6/9 | 2/5 | 9/ 14 | 5/ 14 |
| Overcast | 4/9 | 0/5 | $\sigma$ =6.2 | $\sigma$ =7.9 | $\sigma$ =10.2 | $\sigma$ =9.7 | True | 3/9 | 3/5 | | |
| Rainy | 3/9 | 2/5 | | | | | | | | | |

- Example density value:

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

19

19

## Classifying a new day

- A new day:

| Outlook | Temp. | Humidity | Windy | Play |
|---|---|---|---|---|
| Sunny | 66 | 90 | true | ? |

Likelihood of "yes" = 2/9 × 0.0340 × 0.0221 × 3/9 × 9/14 = 0.000036
Likelihood of "no" = 3/5 × 0.0221 × 0.0381 × 3/5 × 5/14 = 0.000108
P("yes") = 0.000036 / (0.000036 + 0. 000108) = 25%
P("no") = 0.000108 / (0.000036 + 0. 000108) = 75%

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

20

20

## Naïve Bayes: discussion

- Naïve Bayes works surprisingly well even if independence assumption is clearly violated
- Why? Because classification does not require accurate probability estimates *as long as maximum probability is assigned to the correct class*
- However: adding too many redundant attributes will cause problems (e.g., identical attributes)

21

21

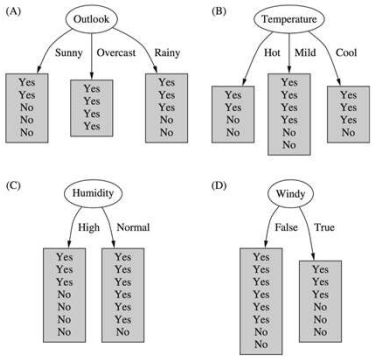
## Constructing decision trees

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

22

22

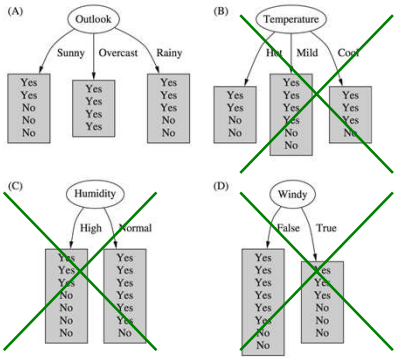
## Which attribute to select?




23

23

## Which attribute to select?




24

24

## 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 selection criterion: *information gain*
  - Information gain increases with the average purity of the subsets
- Strategy: amongst attributes available for splitting, choose attribute that gives greatest information gain
- Information gain requires measure of *impurity*
- Impurity measure that it uses is the *entropy* of the class distribution, which is a measure from information theory

25

25

## Computing information

- We have a probability distribution: the class distribution in a subset of instances
- The expected information required to determine an outcome (i.e., class value), is the distribution's *entropy*
- Formula for computing the entropy:

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

- Using base-2 logarithms, entropy gives the information required in expected *bits*
- Entropy is maximal when all classes are equally likely and minimal when one of the classes has probability 1

26

26

## Example: attribute *Outlook*

- *Outlook* = *Sunny* :

$$\text{Info}([2, 3]) = 0.971 \text{ bits}$$

- *Outlook* = *Overcast* :

$$\text{Info}([4, 0]) = 0.0 \text{ bits}$$

- *Outlook* = *Rainy* :

$$\text{Info}([3, 2]) = 0.971 \text{ bits}$$

- Expected information for attribute:

$$\text{Info}([2, 3], [4, 0], [3, 2]) = (5/14) \times 0.971 + (4/14) \times 0 + (5/14) \times 0.971$$
$$= 0.693 \text{ bits}$$

27

27

## 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

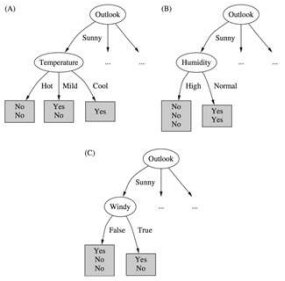
- Information gain for attributes from weather data:

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

28

28

## Continuing to split

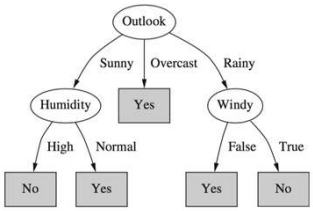



Gain(*Temperature* ) = 0.571 bits
Gain(*Humidity* ) = 0.971 bits
Gain(*Windy* ) = 0.020 bits

29

29

## Final decision tree




- Note: not all leaves need to be pure; sometimes identical instances have different classes
  - Splitting stops when data cannot be split any further

30

30

## Discussion

- Top-down induction of decision trees: ID3, algorithm developed by Ross Quinlan
  - C4.5 tree learner deals with numeric attributes, missing values, noisy data
- Similar approach: CART tree learner
  - Uses Gini index rather than entropy to measure impurity
- There are many other attribute selection criteria! (But little difference in accuracy of result)

$$Gini = 1 - \sum_{i=1}^{n} p^2(c_i)$$

$$Entropy = \sum_{i=1}^{n} -p(c_i)log_2(p(c_i))$$

where $p(c_i)$ is the probability/percentage of class $c_i$ in a node.

31

31

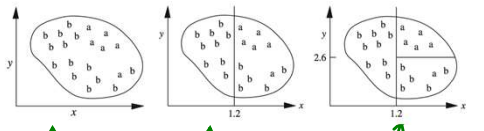
## Covering algorithms

- Can convert decision tree into a rule set
  - Straightforward, but rule set overly complex
  - More effective conversions are not trivial and may incur a lot of computation
- Instead, we can generate rule set directly
  - One approach: 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 of the algorithm, a rule is identified that "covers" some of the instances

32

32

## Example: generating a rule




```
If true
   then class = a

If x > 1.2
   then class = a

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

- Possible rule set for class "b":

```
If x ≤ 1.2 then class = b
If x > 1.2 and y ≤ 2.6 then class = b
```

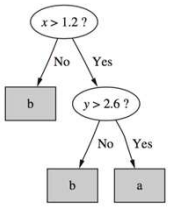
- Could add more rules, get "perfect" rule set

33

33

## Rules vs. trees

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



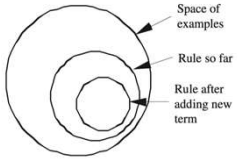

- But: rule sets *can* be more perspicuous when decision trees suffer from replicated subtrees
- Also: in multiclass situations, covering algorithm concentrates on one class at a time whereas decision tree learner takes all classes into account

34

34

## Simple covering algorithm

- Basic idea: generate a rule by adding tests that maximize the rule's accuracy
- Similar to situation in decision trees: problem of selecting an attribute to split on
  - But: decision tree inducer maximizes overall purity
- Each new test reduces rule's coverage:




35

35

## 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 cannot be split any further

36

36

## The contact lenses 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 |



37

## Example: contact lens data

- Rule we seek:

```
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
```



38

## Modified rule and resulting data

- Rule with best test added:

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

- Instances covered by modified rule:

| 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 |



39

## 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
```

40

40

## 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 |

41

41

## 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

42

42

## The final rule

- 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
```

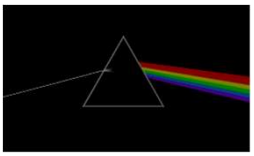
- These two rules cover all "hard lenses":
  - Process is repeated with other two classes

43

43

## 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 left-hand side 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
```



44

44

## Rules vs. decision lists

- PRISM with outer loop removed generates a decision list for one class
  - Subsequent rules are designed for rules that are not covered by previous rules
  - But: order does not matter because all rules predict the same class so outcome does not change if rules are shuffled
- Outer loop considers all classes separately
  - No order dependence implied
- Problems: overlapping rules, default rule required

45

45

## Separate and conquer rule learning

- Rule learning methods like the one PRISM employs (for each class) are called *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 a rule does not need to be explored any further

46

46

## Mining association rules

- Naïve method for finding association rules:
  - Use separate-and-conquer method
  - Treat every possible combination of attribute values as a separate class
- Two problems:
  - Computational complexity
  - Resulting number of rules (which would have to be pruned on the basis of support and confidence)
- It turns out that we can look for association rules with high support and accuracy directly

47

47

## Item sets: the basis for finding rules

- Support: number of instances correctly covered by association rule
  - The same as the number of instances covered by *all* tests in the rule (LHS and RHS!)
- *Item*: one test/attribute-value pair
- *Item set* : all items occurring in a rule
- Goal: find only rules that exceed pre-defined support
  - Do it by finding all item sets with the given minimum support and generating rules from them!

48

48

## Weather data

| Outlook | Temp | 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 |

49

49

## Item sets for weather data

| One-item sets | Two-item sets | Three-item sets | Four-item sets |
|---|---|---|---|
| Outlook = Sunny (5) | Outlook = Sunny<br>Temperature = Hot (2) | Outlook = Sunny<br>Temperature = Hot<br>Humidity = High (2) | Outlook = Sunny<br>Temperature = Hot<br>Humidity = High<br>Play = No (2) |
| Temperature = Cool (4) | Outlook = Sunny<br>Humidity = High (3) | Outlook = Sunny<br>Humidity = High<br>Windy = False (2) | Outlook = Rainy<br>Temperature = Mild<br>Windy = False<br>Play = Yes (2) |
| ... | ... | ... | ... |

- Total number of item sets with a minimum support of at least two instances: 12 one-item sets, 47 two-item sets, 39 three-item sets, 6 four-item sets and 0 five-item sets

50

50

## Generating rules from an item set

- Once all item sets with the required minimum support have been generated, we can turn them into rules
- Example 4-item set with a support of 4 instances:

```
Humidity = Normal, Windy = False, Play = Yes (4)
```

- Seven ($2^N$-1) 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 True then Humidity = Normal and Windy = False
   and Play = Yes                                         4/12
```

51

51

## Rules for weather data

- All rules with support > 1 and confidence = 100%:

| | Association rule | | Sup. | Conf. |
|---|---|---|---|---|
| 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=Cold Play=Yes | ⇒ Humidity=Normal | 3 | 100% |
| | ... | ... | ... | ... |
| 58 | Outlook=Sunny Temperature=Hot | ⇒ Humidity=High | 2 | 100% |

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

52

52

## Example rules from the same item set

- Item set:

```
Temperature = Cool, Humidity = Normal, Windy = False, Play = Yes (2)
```

- Resulting rules (all with 100% confidence):

```
Temperature = Cool, Windy = False ⇒ Humidity = Normal, Play = Yes
Temperature = Cool, Windy = False, Humidity = Normal ⇒ Play = Yes
Temperature = Cool, Windy = False, Play = Yes ⇒ Humidity = Normal
```

- We can establish their confidence due to the following "frequent" item sets:

```
Temperature = Cool, Windy = False                    (2)
Temperature = Cool, Humidity = Normal, Windy = False  (2)
Temperature = Cool, Windy = False, Play = Yes         (2)
```

53

53

## Generating item sets efficiently

- How can we efficiently find all frequent item sets?
- Finding one-item sets easy
- Idea: use one-item sets to generate two-item sets, two-item sets to generate three-item sets, …
  - If (A B) is a frequent item set, then (A) and (B) have to be frequent item sets as well!
  - In general: if X is a frequent *k*-item set, then all (*k*-1)-item subsets of X are also frequent
  - Compute *k*-item sets by merging (*k*-1)-item sets

54

54

## Example

- Given: five frequent three-item sets

  **(A B C), (A B D), (A C D), (A C E), (B C D)**

- Lexicographically ordered!
- Candidate four-item sets:

  **(A B C D)** **OK because of (A C D) (B C D)**

  **(A C D E)** **Not OK because of (C D E)**

- To establish that these item sets are really frequent, we need to perform a final check by counting instances
- For fast look-up, the ($k$ –1)-item sets are stored in a hash table

55

55

## Algorithm for finding item sets

```
Set k to 1
Find all k-item sets with sufficient coverage and store them in hash table #1
While some k-item sets with sufficient coverage have been found
    Increment k
    Find all pairs of (k-1)-item sets in hash table #(k-1) that differ only in
    their last item
    Create a k-item set for each pair by combining the two (k-1)-item sets
    that are paired
    Remove all k-item sets containing any (k-1)-item sets that are not in the
    #(k-1)hash table
    Scan the data and remove all remaining k-item sets that do not have
    sufficient coverage
    Store the remaining k-item sets and their coverage in hash table #k,
    sorting items in lexical order
```

56

56

## Generating rules efficiently

- We are looking for all high-confidence rules
  - Support of antecedent can be obtained from item set hash table
  - But: brute-force method is ($2^N$-1) for an N-item set
- 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

57

57

## Example

- 1-consequent rules:

```
If Outlook = Sunny and Windy = False and Play = No
   then Humidity = High (2/2)
```

```
If Humidity = High and Windy = False and Play = No
   then Outlook = Sunny (2/2)
```

- Corresponding 2-consequent rule:

```
If Windy = False and Play = No
   then Outlook = Sunny and Humidity = High (2/2)
```

- Final check of antecedent against item set hash table is required to check that rule is actually sufficiently accurate

58

58

## Algorithm for finding association rules

```
Set n to 1
Find all sufficiently accurate n-consequent rules for the k-item set and
    store them in hash table #1, computing accuracy using the hash tables
    found for item sets
While some sufficiently accurate n-consequent rules have been found
    Increment n
    Find all pairs of (n-1)-consequent rules in hash table #(n-1) whose
    consequents differ only in their last item
    Create an n-consequent rule for each pair by combining the two (n-1)-
    consequent rules that are paired
    Remove all n-consequent rules that are insufficiently accurate, computing
    accuracy using the hash tables found for item sets
    Store the remaining n-consequent rules and their accuracy in hash table
    #k, sorting items for each consequent in lexical order
```

59

59

## Association rules: discussion

- Above method makes one pass through the data for each different item set size
  - Another possibility: generate ($k$+2)-item sets just after ($k$+1)-item sets have been generated
  - Result: more candidate ($k$+2)-item sets than necessary will be generated but this requires less passes through the data
  - Makes sense if data too large for main memory
- Practical issue: support level for generating a certain minimum number of rules for a particular dataset
  - This can be done by running the whole algorithm multiple times with different minimum support levels
  - Support level is decreased until a sufficient number of rules has been found

60

60

## Linear models: linear regression

- Work most naturally with numeric attributes
- Standard technique for numeric prediction
  - Outcome is linear combination of attributes

$$x = w_0 + w_1 a_1 + w_2 a_2 + \cdots + w_k a_k$$

- Weights are calculated from the training data
- Predicted value for first training instance $\mathbf{a}^{(1)}$

$$w_0 a_0^{(1)} + w_1 a_1^{(1)} + w_2 a_2^{(1)} + \cdots + w_k a_k^{(1)} = \sum_{j=0}^{k} w_j a_j^{(1)}$$

(assuming each instance is extended with a constant attribute with value 1)

61

61

## Minimizing the squared error

- Choose $k$ +1 coefficients to minimize the squared error on the training data
- Squared error: $\sum_{i=1}^{n} \left( x^{(i)} - \sum_{j=0}^{k} w_j a_j^{(i)} \right)^2$
- Coefficients can be derived using standard matrix operations
- Can be done if there are more instances than attributes (roughly speaking)
- Minimizing the *absolute error* is more difficult

62

62

## Classification

- *Any* regression technique can be used for classification
  - Training: perform a regression for each class, setting the output to 1 for training instances that belong to class, and 0 for those that don't
  - Prediction: predict class corresponding to model with largest output value (*membership value*)
- For linear regression this method is also known as *multi-response linear regression*
- Problem: membership values are not in the [0,1] range, so they cannot be considered proper probability estimates
  - In practice, they are often simply clipped into the [0,1] range and normalized to sum to 1

63

63

## Linear models: logistic regression

- Can we do better than using linear regression for classification?
- Yes, we can, by applying logistic regression
- Logistic regression builds a linear model for a transformed target variable
- Assume we have two classes
- Logistic regression replaces the target

$$\Pr[1|a_1, a_2, \ldots, a_k]$$

by this target

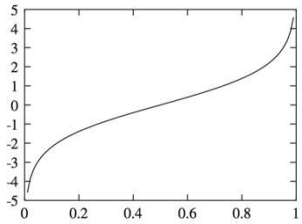
$$\log[\Pr[1|a_1, a_2, \ldots, a_k]/(1 - \Pr[1|a_1, a_2, \ldots, a_k])$$

- This *logit transformation* maps [0,1] to (-∞ , +∞ ), i.e., the new target values are no longer restricted to the [0,1] interval

64

64

## Logit transformation




- Resulting class probability model:

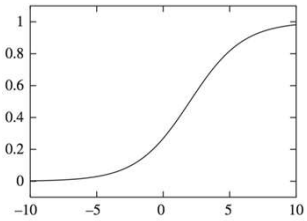
$$\Pr[1|a_1, a_2, \ldots, a_k] = 1/(1 + \exp(-w_0 - w_1 a_1 - \cdots - w_k a_k))$$

65

65

## Example logistic regression model

- Model with $w_0$ = -1.25 and $w_1$ = 0.5:




- Parameters are found from training data using *maximum likelihood*

66

66

## Maximum likelihood

- Aim: maximize probability of observed training data with respect to final parameters of the logistic regression model
- We can use logarithms of probabilities and maximize conditional *log-likelihood* instead of product of probabilities:

$$\sum_{i=1}^{n}(1-x^{(i)})\log(1-\Pr[1|a_1^{(i)},\ a_2^{(i)},\ \ldots,\ a_k^{(i)}])+x^{(i)}\log(\Pr[1|a_1^{(i)},a_2^{(i)},\ldots,a_k^{(i)}])$$

  where the class values $x^{(i)}$ are either 0 or 1
- Weights $w_i$ need to be chosen to maximize log-likelihood
- A relatively simple method to do this is *iteratively re-weighted least squares* but other optimization methods can be used

67

67

## Linear models are hyperplanes

- Decision boundary for two-class logistic regression is where probability equals 0.5:

$$\Pr[1|a_1,a_2,\ldots,a_k]=1/(1+\exp(-w_0-w_1a_1-\cdots-w_ka_k))=0.5$$

  which occurs when $-w_0-w_1a_1-\cdots-w_ka_k=0$
- Thus logistic regression can only separate data that can be separated by a hyperplane

68

68

## Linear models: the perceptron

- Observation: we do not actually need probability estimates if all we want to do is classification
- Different approach: learn separating hyperplane directly
- Let us assume the data is *linearly separable*
- In that case there is a simple algorithm for learning a separating hyperplane called the *perceptron learning rule*
- Hyperplane: $w_0a_0+w_1a_1+w_2a_2+\cdots+w_ka_k=0$
  where we again assume that there is a constant attribute with value 1 (*bias*)
- If the weighted sum is greater than zero we predict the first class, otherwise the second class

69

69

## The algorithm

```
Set all weights to zero
Until all instances in the training data are classified correctly
  For each instance I in the training data
    If I is classified incorrectly by the perceptron
      If I belongs to the first class add it to the weight vector
      else subtract it from the weight vector
```

- Why does this work?
  Consider a situation where an instance *a* pertaining to the first class has been added:

$$(w_0 + a_0)a_0 + (w_1 + a_1)a_1 + (w_2 + a_2)a_2 + \cdots + (w_k + a_k)a_k$$

  This means the output for *a* has increased by:

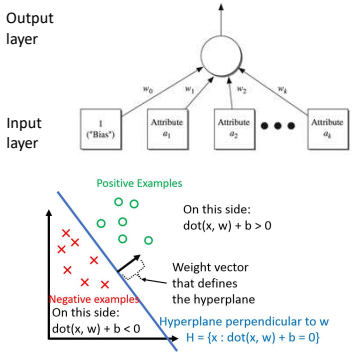
$$a_0 \times a_0 + a_1 \times a_1 + a_2 \times a_2 + \cdots + a_k \times a_k$$

  This number is always positive, thus the hyperplane has moved into the correct direction (and we can show that output decreases for instances of other class)
- It can be shown that this process converges to a linear separator if the data is linearly separable

70

70

## Perceptron as a neural network




71

71

## Clustering

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

72

72

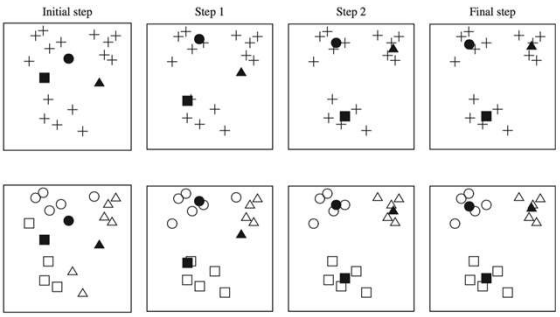
## The *k*-means algorithm

- Step 1: Choose *k* random cluster centers
- Step 2: Assign each instance to its closest cluster center based on Euclidean distance
- Step 3: Recompute cluster centers by computing the average (aka *centroid*) of the instances pertaining to each cluster
- Step 4: If cluster centers have moved, go back to Step 2
- This algorithm minimizes the squared Euclidean distance of the instances from their corresponding cluster centers
  - Determines a solution that achieves a *local* minimum of the squared Euclidean distance
- Equivalent termination criterion: stop when assignment of instances to cluster centers has not changed

73

73

## The *k*-means algorithm: example

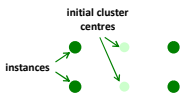



74

74

## Discussion

- Algorithm minimizes squared distance to cluster centers
- Result can vary significantly
  - based on initial choice of seeds
- Can get trapped in local minimum
  - Example:




- To increase chance of finding global optimum: restart with different random seeds
- Can we applied recursively with *k* = 2

75

75

## Choosing the number of clusters

- Big question in practice: what is the right number of clusters, i.e., what is the right value for *k*?
- Cannot simply optimize squared distance on training data to choose k
  - Squared distance decreases monotonically with increasing values of *k*
- Need some measure that balances distance with complexity of the model, e.g., based on the MDL principle (minimum description length)
- Finding the right-size model using MDL becomes easier when applying a recursive version of *k*-means (*bisecting k-means*):
  - Compute A: information required to store data centroid, and the location of each instance with respect to this centroid
  - Split data into two clusters using 2-means
  - Compute B: information required to store the two new cluster centroids, and the location of each instance with respect to these two
  - If A > B, split the data and recurse (just like in other tree learners)

76

76

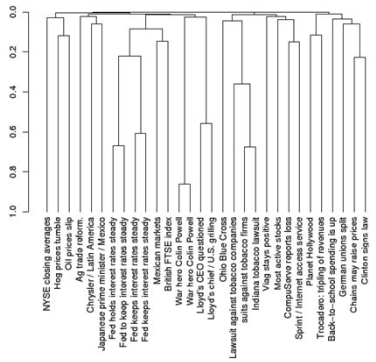
## Hierarchical clustering

- Bisecting *k*-means performs hierarchical clustering in a top-down manner
- Standard hierarchical clustering performs clustering in a bottom-up manner; it performs *agglomerative* clustering:
  - First, make each instance in the dataset into a trivial mini-cluster
  - Then, find the two closest clusters and merge them; repeat
  - Clustering stops when all clusters have been merged into a single cluster
- Outcome is determined by the distance function that is used:
  - *Single-linkage* clustering: distance of two clusters is measured by finding the two closest instances, one from each cluster, and taking their distance
  - *Complete-linkage* clustering*:* use the two most distant instances instead
  - *Average-linkage* clustering: take average distance between all instances
  - *Centroid-linkage* clustering: take distance of cluster centroids
  - *Group-average* clustering: take average distance in merged clusters
  - *Ward's method*: optimize *k-means* criterion (i.e., squared distance)

77

77

## Example: hierarchical clustering




78

78

## Incremental clustering

- Heuristic approach (COBWEB/CLASSIT)
- Forms a hierarchy of clusters incrementally
- Start:
  - tree consists of empty root node
- Then:
  - add instances one by one
  - update tree appropriately at each stage
  - to update, find the right leaf for an instance
  - may involve restructuring the tree using *merging* or *splitting* of nodes
- Update decisions are based on a goodness measure called *category utility*

79

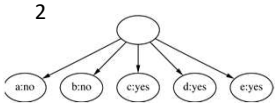
79

## Clustering the weather data I

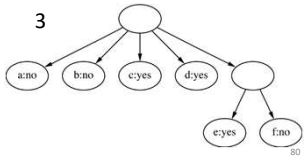
| ID | Outlook | Temp. | Humidity | Windy |
|---|---|---|---|---|
| A | Sunny | Hot | High | False |
| B | Sunny | Hot | High | True |
| C | Overcast | Hot | High | False |
| D | Rainy | Mild | High | False |
| E | Rainy | Cool | Normal | False |
| F | Rainy | Cool | Normal | True |
| G | Overcast | Cool | Normal | True |
| H | Sunny | Mild | High | False |
| I | Sunny | Cool | Normal | False |
| J | Rainy | Mild | Normal | False |
| K | Sunny | Mild | Normal | True |
| L | Overcast | Mild | High | True |
| M | Overcast | Hot | Normal | False |
| N | Rainy | Mild | High | True |

1


2




3



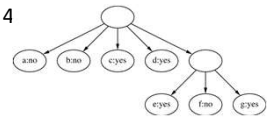

80

80

## Clustering the weather data II

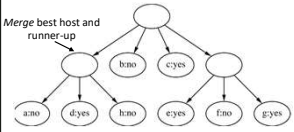
| ID | Outlook | Temp. | Humidity | Windy |
|---|---|---|---|---|
| A | Sunny | Hot | High | False |
| B | Sunny | Hot | High | True |
| C | Overcast | Hot | High | False |
| D | Rainy | Mild | High | False |
| E | Rainy | Cool | Normal | False |
| F | Rainy | Cool | Normal | True |
| G | Overcast | Cool | Normal | True |
| H | Sunny | Mild | High | False |
| I | Sunny | Cool | Normal | False |
| J | Rainy | Mild | Normal | False |
| K | Sunny | Mild | Normal | True |
| L | Overcast | Mild | High | True |
| M | Overcast | Hot | Normal | False |
| N | Rainy | Mild | High | True |

4




5

*Merge* best host and runner-up

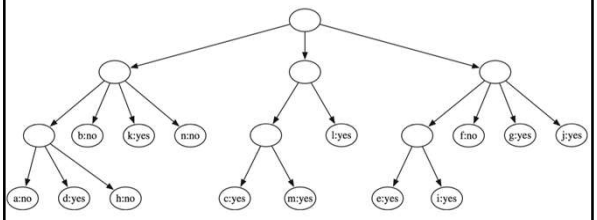



Consider *splitting* the best host if merging does not help

81

81

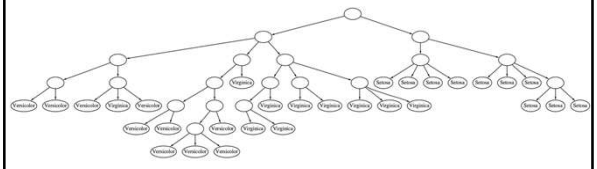
Final clustering




82

82

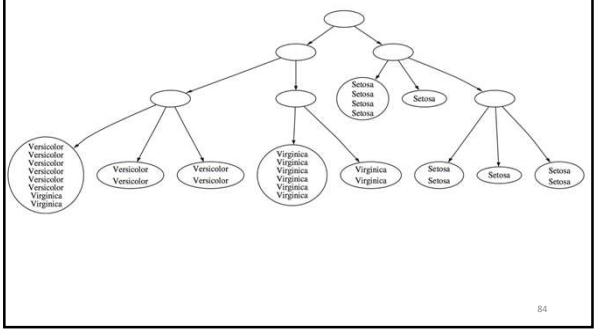
Example: clustering a subset of the iris data



83

83

Example: iris data with cutoff




84

84

## The category utility measure

- Category utility: quadratic loss function defined on conditional probabilities:

$$CU(C_1, C_2, ..., C_k) = \frac{\sum_l \mathrm{P}(C_l) \sum_i \sum_j (\mathrm{P}(a_i = v_{ij} \mid C_l)^2 - \mathrm{P}(a_i = v_{ij})^2)}{k}$$

- Every instance in a different category ⇒ numerator becomes

$$m - \mathrm{P}(a_i = v_{ij})^2 \longleftarrow \text{maximum}$$

number of attributes

85

85

## Some final comments on the basic methods

- Bayes' rule stems from his "Essay towards solving a problem in the doctrine of chances" (1763)
  - Difficult bit in general: estimating prior probabilities (easy in the case of naïve Bayes)
- Extension of naïve Bayes: Bayesian networks (which we will discuss later)
- The algorithm for association rules we discussed is called APRIORI; many other algorithms exist
- Minsky and Papert (1969) showed that linear classifiers have limitations, e.g., can't learn a logical XOR of two attributes
  - But: combinations of them can (this yields multi-layer neural nets)

86

86