# DM & ML
# Basic Methods

**Gergely Lukács**

Pázmány Péter Catholic University

Faculty of Information Technology and Bionics

Budapest, Hungary

lukacs@itk.ppke.hu

# Simplicity first

- Different kinds of simple structures/algorithms exist:
  - One attribute
  - All attributes independently
  - A linear combination
  - An instance-based representation
- Simple algorithms often work surprisingly well
- Useful for understanding data
- Basis for more complicated algorithms

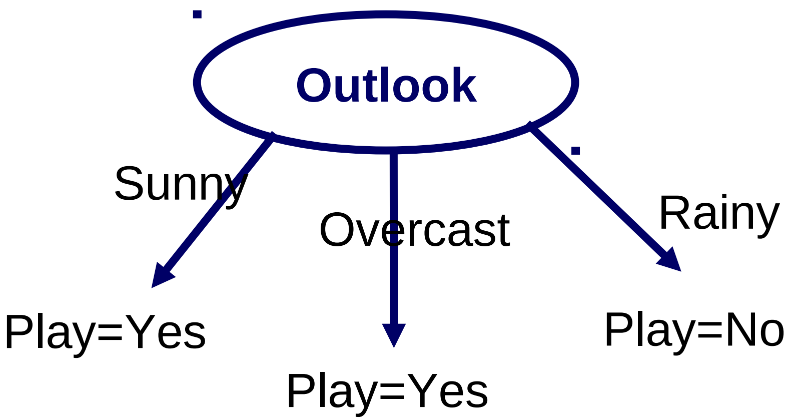
# 1R
# (One Rule: decision tree with only one level)

# 1R: Output format

- 1R: „one rule”
  - for classification
  - set of „*rules*” that all test one particular attribute (== 1-level „*decision tree*” – see later; „decision stump” )

| X | | | | Y (class) |
|---|---|---|---|---|
| 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 |

IF outlook=sunny THEN play=yes
IF outlook=overcast THEN play=yes
IF outlook=rainy THEN play=no



# 1R: Algorithm

- Basic version (assuming nominal attributes)
  - Create a ruleset for each of the attributes
    - One branch for each of the attribute’s values
    - 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 and the corresponding ruleset with lowest error rate

# 1R Example: 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 | |
| Temperature | 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 | |

**** A random choice is made between 2 equal outcomes***

# 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
```

- “missing” is treated as a separate attribute value

# Dealing with numeric attributes

- Numeric attributes are discretized: the range of the attribute is divided into a set of intervals:
  - Instances are sorted according to attribute's values
  - Breakpoints are placed where the class changes (so that the total error is minimized)
    - Equal values, different classes: majority class
- 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
```

# The problem of **overfitting**

- Discretization procedure is very sensitive to noise
  - A single instance with an incorrect class label will most likely result in a separate interval
- Simple solution: **enforce minimum number of instances in majority class per interval**
- Weather data example (with minimum set to 3):

| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 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 |

# Result of overfitting avoidance

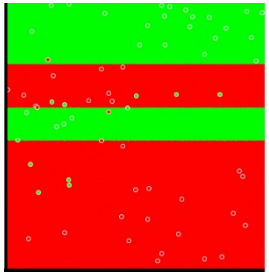
- Resulting rule sets:

| 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 | |

**** A random choice is made between 2 equal outcomes***

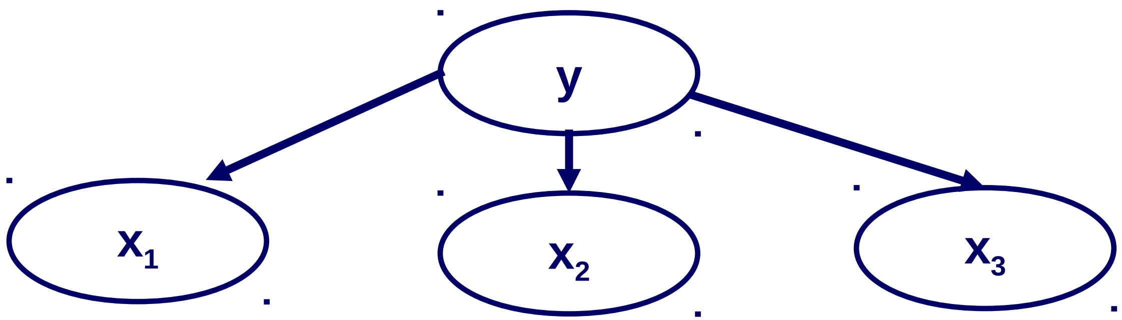
# Discussion of 1R

- 1R was described in a paper by Holte (1993)
  - Contains an experimental evaluation on 16 datasets (using *cross-validation* so that results were representative of performance on future data)
  - After some experimentation the minimum number of instances in majority class was set to 6
- Simplicity first …
- Data investigation …
  how important individual attributes are
- Part of more complexe algorithms
  (ensemble learning)
- Decision boundary
  - illustration with 2 input variables (two axes),
    binary classification (class: colour)



# Naïve Bayes

# NB: The model



(class attribute,
y, dependent/output variable)

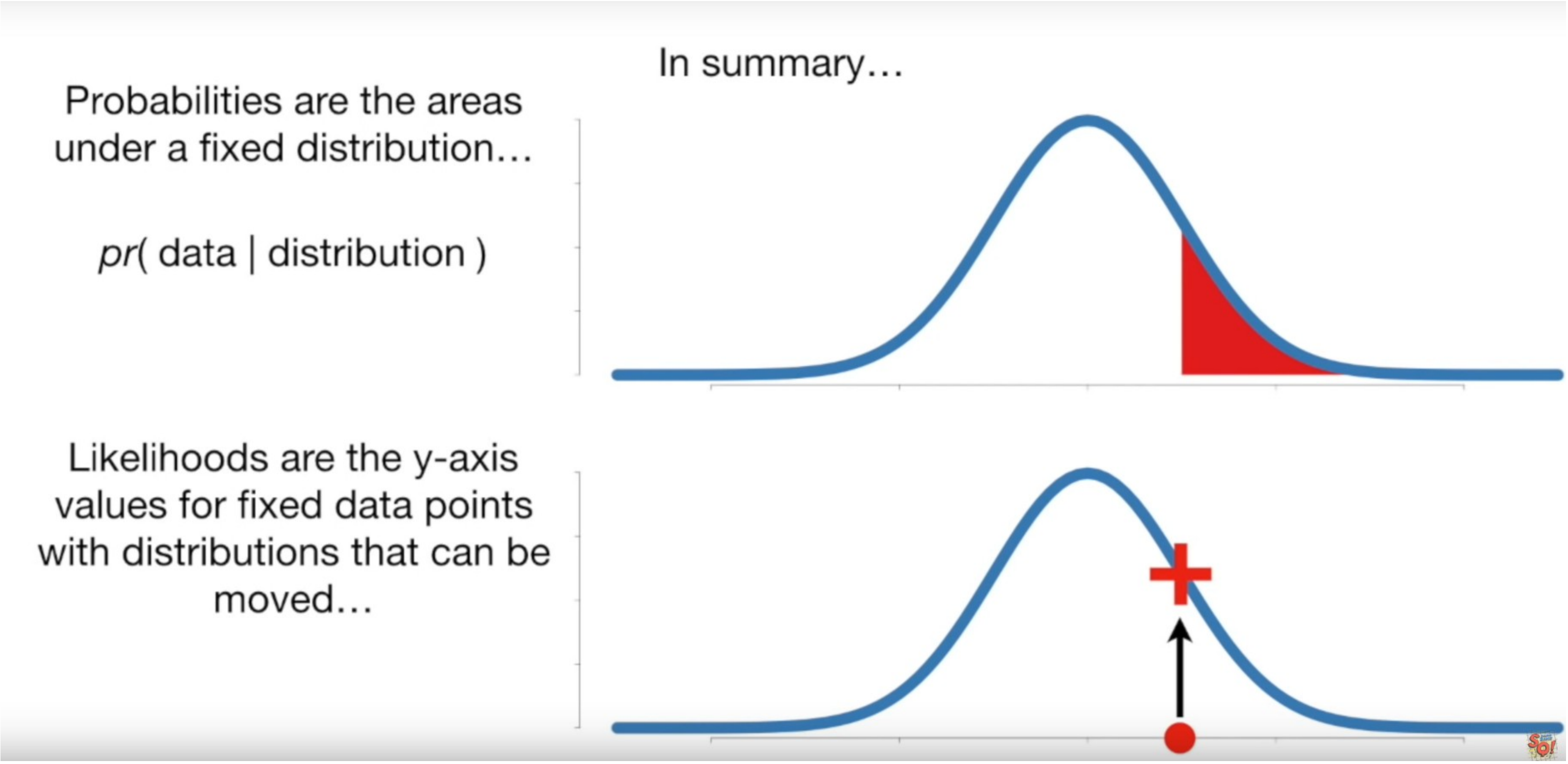
(x, independent/input variables)

- “Opposite” of 1R: use all attributes
- Assumption: Attributes are *statistically independent*, **given the class value**
    - This means that knowledge about the value of a particular attribute doesn’t tell us anything about the value of another attribute -- **if the class is known**
    - 
        - Fixed

$$\boxed{P(x|y) = P(x_1, x_2, \ldots |y) = P(x_1|y)P(x_2|y)\ldots = \prod_{i=1}^{n} P(x_i|y)}$$

- Although assumption almost never (entirely) correct, this scheme works quite often well in practice!

# Likelihood vs probability



In summary…

Probabilities are the areas under a fixed distribution…

*pr*( data | distribution )

Likelihoods are the y-axis values for fixed data points with distributions that can be moved…

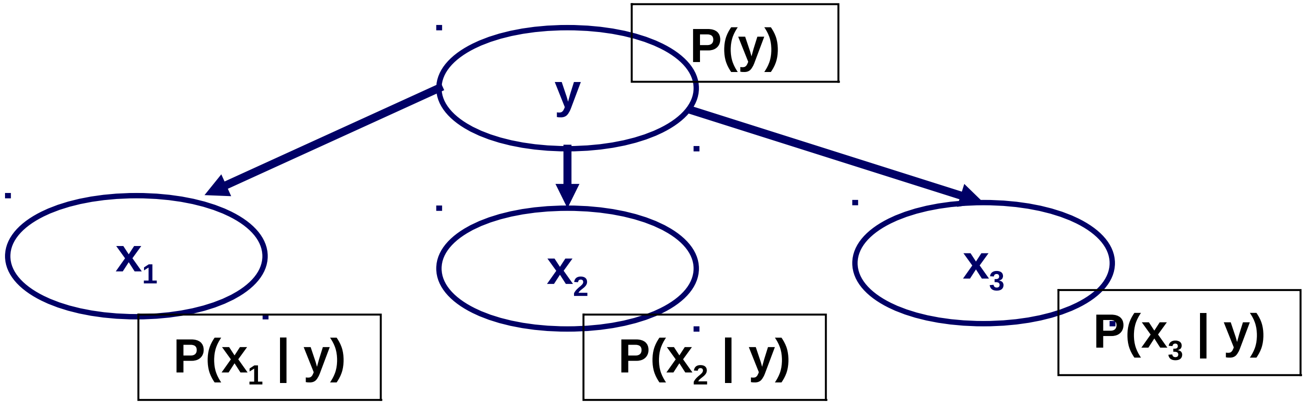
- **StatQuest: Probability vs Likelihood** **https://www.youtube.com/watch?v=pYxNSUDSFH4**

# Maximum Likelihood Estimation

- **maximum likelihood estimation (MLE)** of parameters given observations. MLE attempts to find the parameter values that maximize the likelihood function, given the observations

- Naive Bayes: learning: estimating parameter values of the model given data, based on the MLE
  (Details: http://www.datasciencecourse.org/notes/mle/)

# Naive Bayes: learning (training)



$P(y)$

$y$

$x_1$

$x_2$

$x_3$

$P(x_1 \mid y)$

$P(x_2 \mid y)$

$P(x_3 \mid y)$

- **(Relative) frequencies of values in training data**
  - For binary classifier:

$$\boxed{P(y=k) = \frac{1}{m} \times \#\{j \mid y^{(j)} = k\}} \quad \text{and} \quad \boxed{P(x_i = l \mid y = k) = \frac{\#\{j \mid y^{(j)} = k \text{ and } x_i^{(j)} = l\}}{\#\{j \mid y^{(j)} = k\}}}$$

$$k \in \{0, 1\}, l \in [\![1, L]\!]$$

# Probabilities for the weather data: Frequencies of cases

| $x_1$ | | | $x_4$ | $y$ |
|---|---|---|---|---|
| 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 |

| 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 | | | | | | | | |

# Probabilities for the weather data: probability estimations

| 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 | | | | | | | | |

P(Outlook=Sunny| Play=Yes)

(Outlook: X1, E)

P(Play=Yes)

(Play: Y, class, H)

Can you draw the Naive Bayes network,
including the (conditional and apriori) probabilites?

# NB: prediction (new/unknown instances)

- Classification learning: what’s the probability of the class given the input attributes?

# Probabilities for the weather data: Prediction task

| 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 | ? |

# The weather data example

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

***Evidence***
***E (== X)***

$$\Pr[yes \mid E] = \Pr[Outlook = Sunny \mid yes] \times$$
$$\Pr[Temperature = Cool \mid yes] \times$$
$$\Pr[Humdity = High \mid yes] \times$$
$$\Pr[Windy = True \mid yes] \times \frac{\Pr[yes]}{\Pr[E]}$$

***Cond. probability for class "yes"***

***P(Play=yes|Outlook=Sunny ^ ….)***

$$= \frac{2/9 \times 3/9 \times 3/9 \times 3/9 \times 9/14}{\Pr[E]}$$

# Probabilities for the 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 | ? |

Numerator

For “yes” = 2/9 × 3/9 × 3/9 × 3/9 × 9/14 = 0.0053

For “no” = 3/5 × 1/5 × 4/5 × 3/5 × 5/14 = 0.0206

Conversion into a probability by normalization (instead of by calculating the denominator, P(X)):

P(“yes”) = 0.0053 / (0.0053 + 0.0206) = 0.205

P(“no”) = 0.0206 / (0.0053 + 0.0206) = 0.795

# The “zero-frequency problem”

- What if an attribute value doesn’t occur with every class value (e.g. “Outlook = Overcast” for class “no”)?
  - Probability will be zero! $\Pr[Outlook = Overcast \mid no] = 0$
  - *A posteriori* probability will also be zero! $\Pr[no \mid E] = 0$

    (No matter how likely the other values are!)
- Remedy: add 1 to the count of **every** value-class combination for such problematic attribute („*Laplace estimator*”)
- Result: probabilities will never be zero

# Applying Laplace estimator

| 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 | | | Temperature | | | Humidity | | | Windy | | | Play | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | *Yes* | *No* | | *Yes* | *No* | | *Yes* | *No* | | *Yes* | *No* | *Yes* | *No* |
| Sunny | 2 | 4 | Hot | 2 | 2 | High | 3 | 4 | False | 6 | 2 | 9 | 5 |
| Overcast | 4 | 1 | Mild | 4 | 2 | Normal | 6 | 1 | True | 3 | 3 | | |
| Rainy | 3 | 3 | Cool | 3 | 1 | | | | | | | | |
| Sunny | 2/9 | 4/8 | Hot | 2/9 | 2/5 | High | 3/9 | 4/5 | False | 6/9 | 2/5 | 9/14 | 5/14 |
| Overcast | 4/9 | 1/8 | Mild | 4/9 | 2/5 | Normal | 6/9 | 1/5 | True | 3/9 | 3/5 | | |
| Rainy | 3/9 | 3/8 | Cool | 3/9 | 1/5 | | | | | | | | |

# 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 | ? |

Numerator for “yes” = 3/9 × 3/9 × 3/9 × 9/14 = 0.0238

Numerator for “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%

# Dealing with numeric attributes

- Usual assumption: attributes have a *normal*/*Gaussian* probability distribution (given the class)
- The *probability density function* for the normal distribution is defined by two parameters:
  - The *sample mean* $\mu$:

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

  - The *standard deviation* $\sigma$:

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

  - The density function *f(x)*:

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

# Statistics for the 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 | | | | | |
| | | | | .... | | | .... | | | | | | |
| 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:

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

# Probability densities

- Relationship between probability and density:

$$\Pr[a \le x \le b] = \int_a^b f(t)dt$$

$$\Pr[c - \frac{\varepsilon}{2} < x < c + \frac{\varepsilon}{2}] \approx \varepsilon \star f(c)$$

- But: this doesn’t change calculation of *a posteriori* probabilities because $\varepsilon$ cancels out

# Classifying a new day

- A new day:

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

Numerator of “yes” = 2/9 × 0.0340 × 0.0221 × 3/9 × 9/14 = 0.000036

Numerator of “no” = 3/5 × 0.0291 × 0.0380 × 3/5 × 5/14 = 0.000136

P(“yes”) = 0.000036 / (0.000036 + 0. 000136) = 20.9%

P(“no”) = 0. 000136 / (0.000036 + 0. 000136) = 79.1%

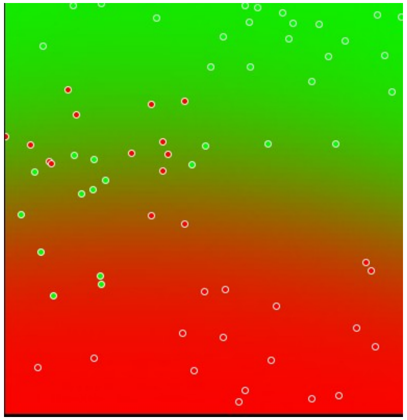
- Note also: many numeric attributes are not normally distributed
  - Known distribution: work with it
  - Unknown distribution: *kernel density estimators*
- Missing values during training: not included in calculation of mean and standard deviation

# Discussion of Naïve Bayes

- Naïve Bayes works surprisingly well (even if independence assumption is clearly violated)
- Too many ***redundant attributes*** (e.g. identical attributes) will cause problems
- ***Irrelevant attributes*** are OK, they are filtered out
- Fast to train, fast to classify
  - even for large number of attributes
  - incremental model building possible (streamed data)
- Outputs probability (?) distribution, not just a single class (+)
- (Does not learn training data perfectly (+/-))
- Typical application: text/spam detection

# Discussion of Naïve Bayes, 2

- Decision boundary….



# Divide-and-conquer
# Decision tree
# ID3

# DECISION TREE – AS KNOWLEDGE REPRESENTATION

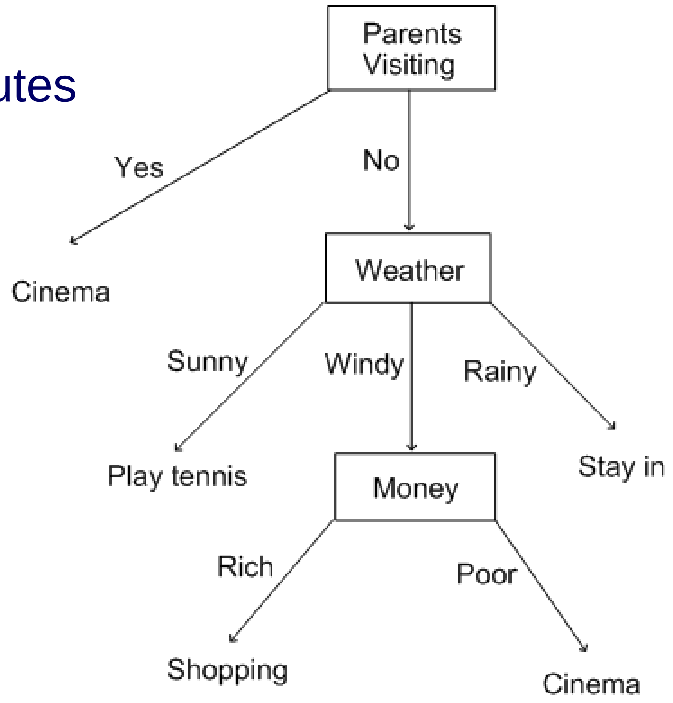
# Decision trees

## (**classification**, regression)

- Nodes involve testing a particular attribute
  - usually, attribute value is compared to constant;
  - other possibilities:
    - comparing values of two attributes
    - using a function of one or more attributes

- Leaves assign
  - class value, or
  - set of class values, or
  - probability distribution

  to instances



# Testing nominal and numeric attributes

- Nominal attribute: number of children (branches) usually equal to number values ⇒ attribute won’t get tested more than once
  - Other possibility: division into two subsets
- Numeric attribute: test whether value is greater or less than constant ⇒ attribute may get tested several times
  - Other possibility: three/multi-way split)
    - Integer: *less than, equal to, greater than*
    - Real: *below, within, above*

# Example: Labor negotiations data

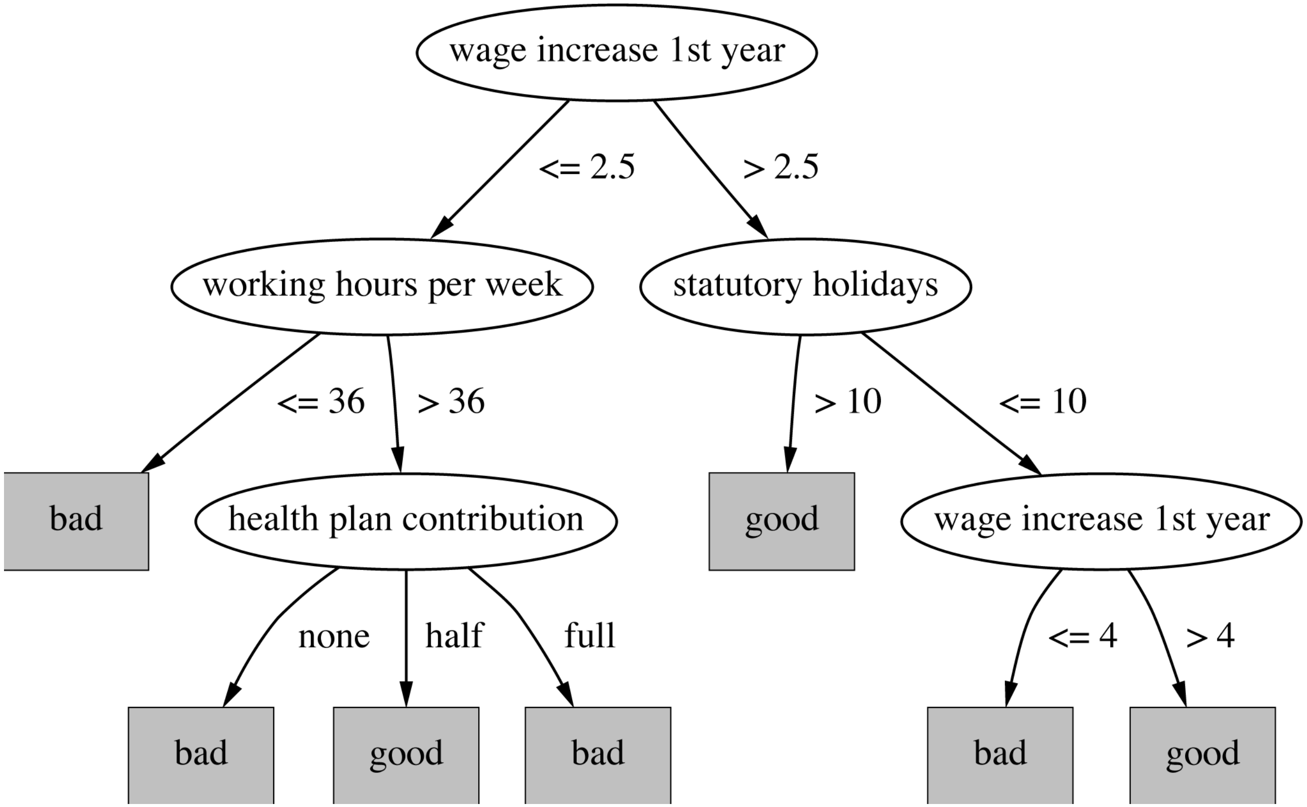
(Instances)

(Attributes)

| Attribute | Type | 1 | 2 | 3 | … | 40 |
|---|---|---|---|---|---|---|
| Duration | (Number of years) | 1 | 2 | 3 | | 2 |
| Wage increase first year | Percentage | 2% | 4% | 4.3% | | 4.5% |
| Wage increase second year | Percentage | ? | 5% | 4.4% | | 4.0% |
| Wage increase third year | Percentage | ? | ? | ? | | ? |
| Cost of living adjustment | {none,tcf,tc} | none | tcf | ? | | none |
| Working hours per week | (Number of hours) | 28 | 35 | 38 | | 40 |
| Pension | {none,ret-allw, empl-cntr} | none | ? | ? | | ? |
| Standby pay | Percentage | ? | 13% | ? | | ? |
| Shift-work supplement | Percentage | ? | 5% | 4% | | 4% |
| Education allowance | {yes,no} | yes | ? | ? | | ? |
| Statutory holidays | (Number of days) | 11 | 15 | 12 | | 12 |
| Vacation | {below-avg,avg,gen} | avg | gen | gen | | avg |
| Long-term disability assistance | {yes,no} | no | ? | ? | | yes |
| Dental plan contribution | {none,half,full} | none | ? | full | | full |
| Bereavement assistance | {yes,no} | no | ? | ? | | yes |
| Health plan contribution | {none,half,full} | none | ? | full | | half |
| Acceptability of contract | {good,bad} | bad | good | good | | good |

(Class attribute)

# Decision tree for labor negotiations data



# Missing values

- ***Does absence of value have some significance?***
- Yes ⇒ “missing” is a separate value
- No ⇒ “missing” must be treated in a special way
  - Solution A: assign instance to ***most popular branch***
  - Solution B: split instance into pieces
    - Pieces receive ***weight according to fraction of training instances*** that go down each branch

For both, we need some statistics on the training data in the tree – in addition to the previous tests

# LEARNING A DECISION TREE – ID 3 ALGORITHM

# Constructing decision trees

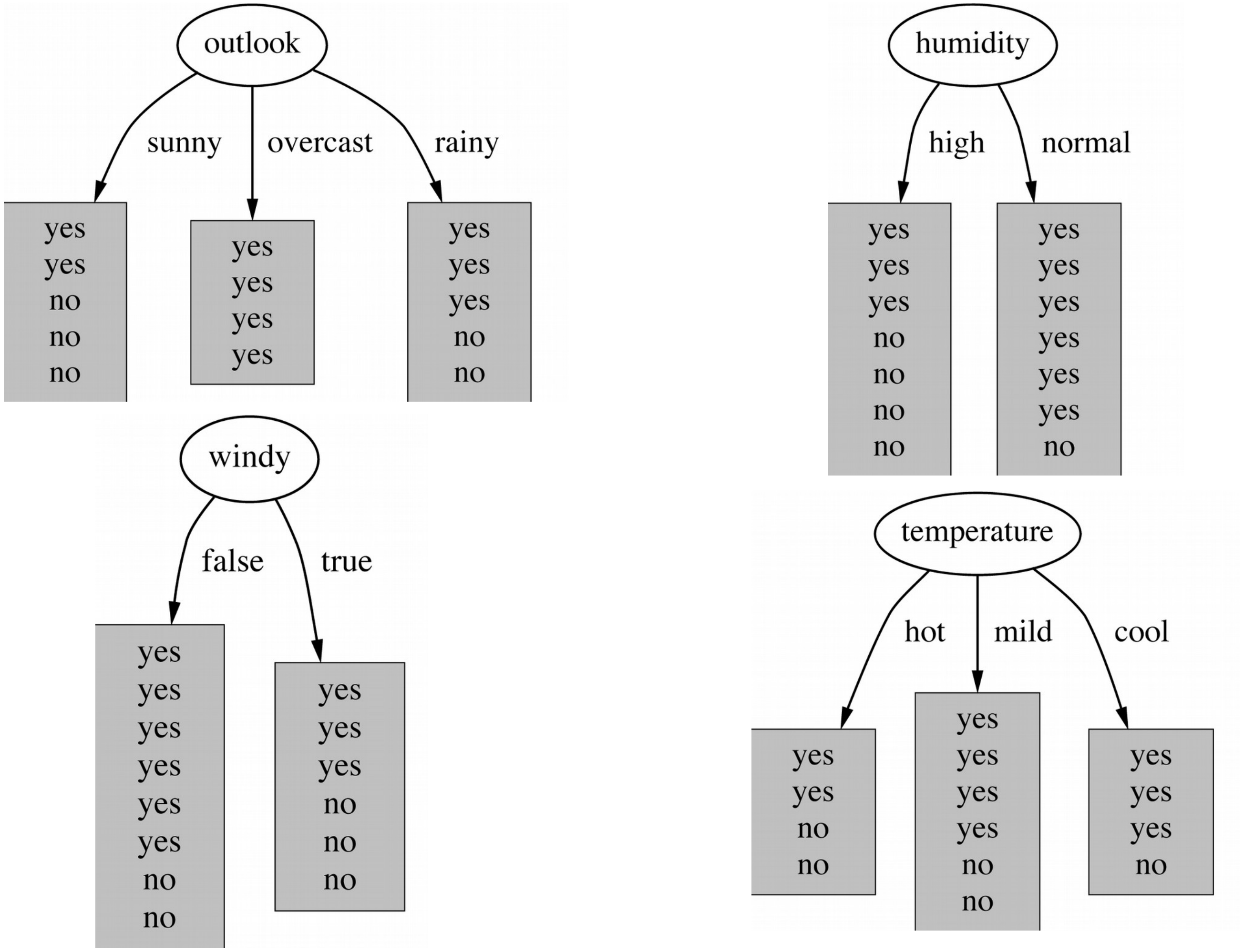
- Preferred: small decision tree (== small number of levels/attributes tested)
- Greedy algorithm
- Normal procedure: top down in recursive *divide-and-conquer* fashion
  - Attribute is selected for root node and branch is created for each possible attribute value
  - Instances are split into subsets (one for each branch extending from the node)
  - Procedure is repeated recursively for each branch, using only instances that reach the branch
- Process stops if all instances have the same class
  - or no further split makes them purer

# Example data

(class attr, y)

| 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 |

# Which attribute to select?



# A criterion for attribute selection

- Which is the best attribute?
  - The one which will result in 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 that an attribute produces
- Strategy: choose attribute that results in greatest information gain

- „purity”: entropy

# Example: attribute *Outlook*

- *Outlook* = *Sunny* :

  Info([2,3]) = entropy(2/5,3/5)=
  = $-2/5\log_2(2/5)-3/5\log_2(3/5)$=0.971 bits

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

- *Outlook* = *Rainy* :

  Info([3,2]) = entropy(3/5,2/5)=
  = $-3/5\log_2(3/5)-2/5\log_2(2/5)$=0.971 bits

- Expected information for attribute
  (weighed sum, weighs: number of instances):

  Info([2,3], [4,0], [3,2]) = 5/14 * 0.971 + 4/14 * 0 + 5/14 * 0.971 =
  = 0.693 bits

# Computing information gain

- Information gain:

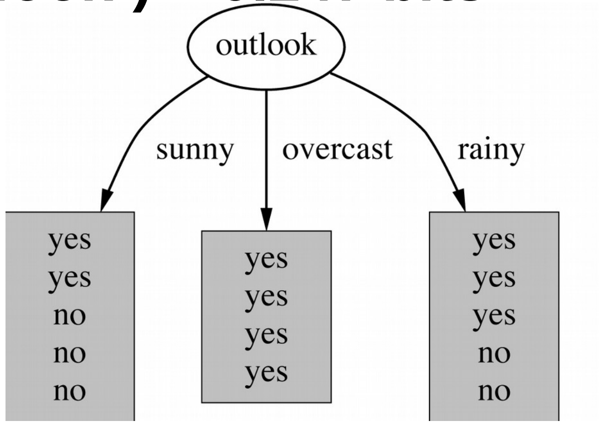
  information before splitting – information after splitting

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

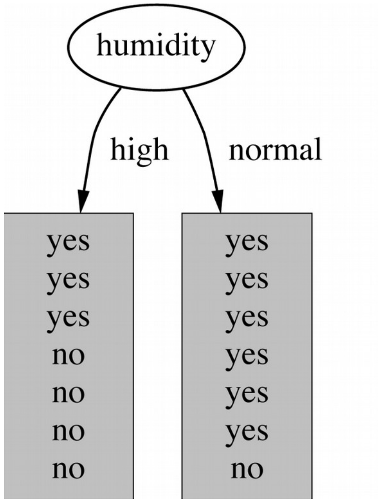
  = 0.940 – 0.693 = 0.247 bits

# Which attribute to select?

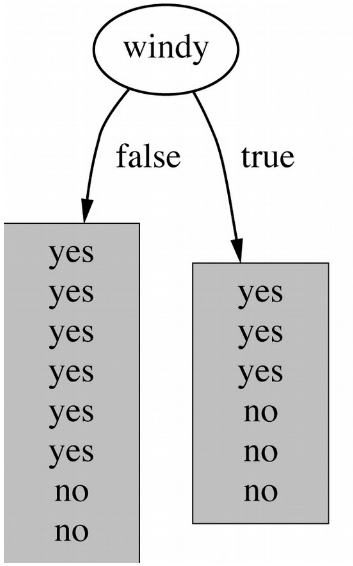
**gain(*Outlook* ) = 0.247 bits**



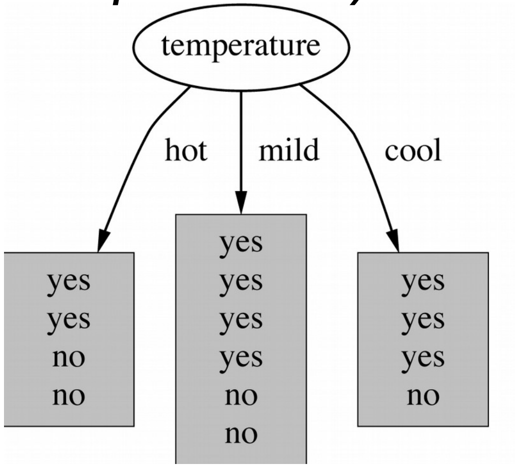
gain(*Humidity* ) = 0.152 bits



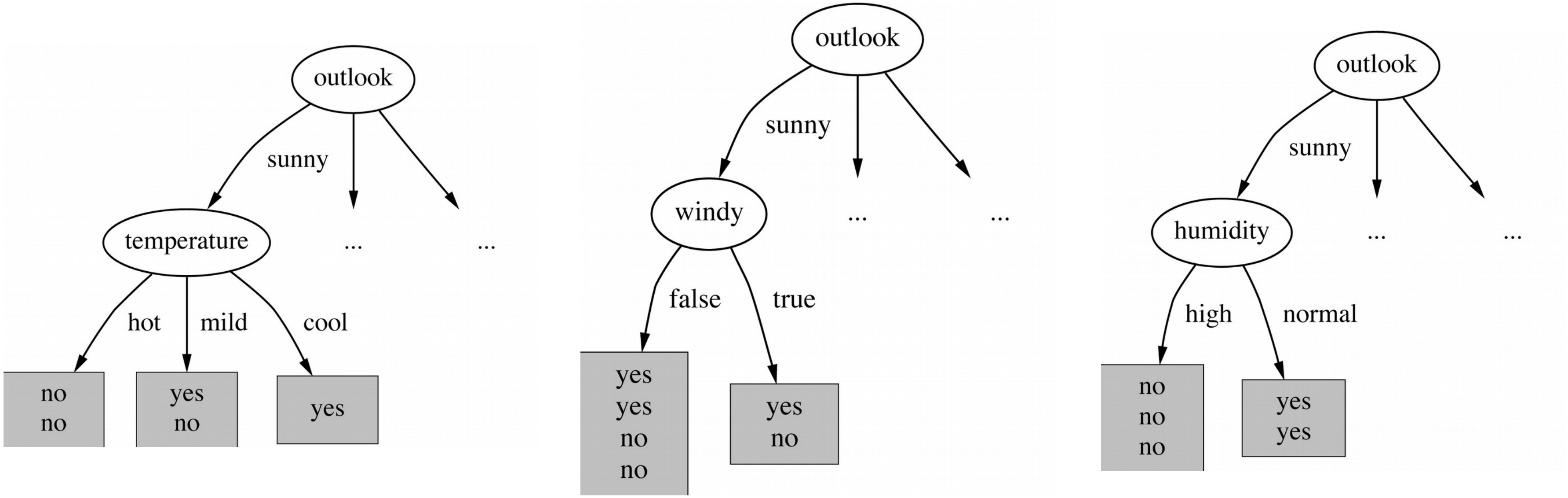
gain(*Windy* ) = 0.048 bits



gain(*Temperature* ) = 0.029 bits



# Continuing to split

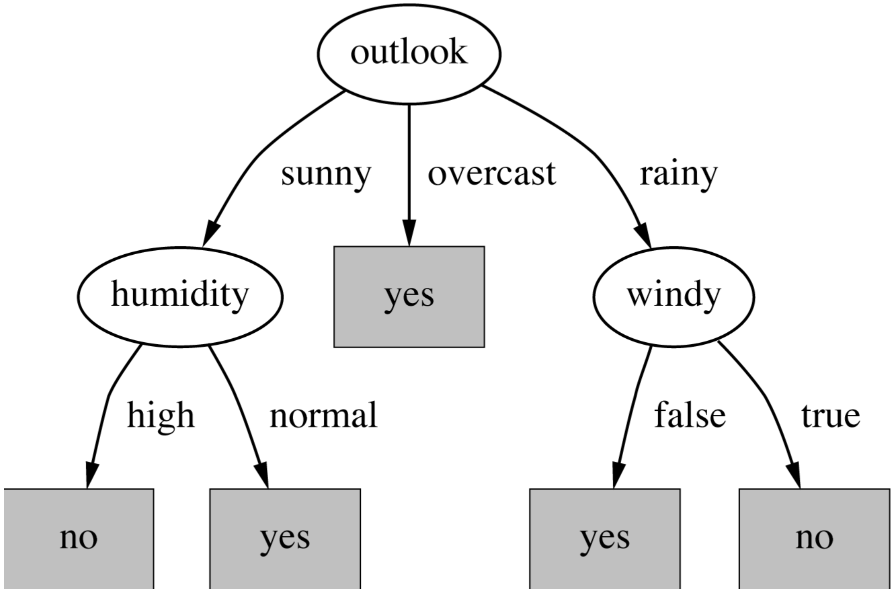


$$\text{gain}(\text{"Temperature"}) = 0.571\ \text{bits}$$

$$\text{gain}(\text{"Windy"}) = 0.019\ \text{bits}$$

$$\text{gain}(\text{"Humidity"}) = 0.971\ \text{bits}$$

# The final decision tree



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

# Wish list for a purity measure

- Properties we require from a purity measure:
  - Pure node: zero
  - Impurity is maximal (i.e. all classes equally likely): maximal (e.g., binary case: 1)
  - ***Multistage property*** (i.e. decisions can be made in several stages):

$$\text{measure}([2,3,4]) = \text{measure}([2,7]) + (7/9) \times \text{measure}([3,4])$$

- **Entropy** is the only function that satisfies all three properties!

# Computing information

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

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

# Some properties of the entropy

- The multistage property:

$$\text{entropy}(p,q,r) = \text{entropy}(p,q+r) + (q+r) \times \text{entropy}(\frac{q}{q+r},\frac{r}{q+r})$$

(where p+q+r = 1)

- Simplification of computation:

$$\text{info}([2,3,4]) = -2/9 \times \log(2/9) - 3/9 \times \log(3/9) - 4/9 \times \log(4/9)$$
$$= [-2\log 2 - 3\log 3 - 4\log 4 + 9\log 9]/9$$

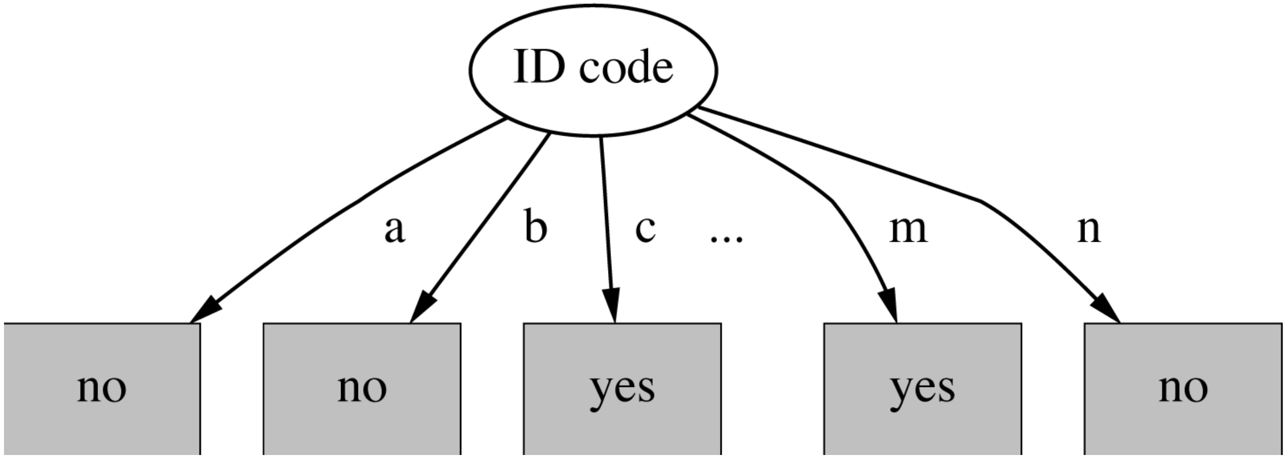
# Highly-branching attributes

- Problematic: 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 *overfitting* (selection of an attribute that is non-optimal for prediction)

# The weather data with ID code (extreme example)

| ID code | Outlook | Temp. | 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:

$$\text{info}(\text{"ID code"}) = \text{info}([0,1]) + \text{info}([0,1]) + \ldots + \text{info}([0,1]) = 0 \text{ bits}$$

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

# The gain ratio

- *Gain ratio*: a modification of the information gain that reduces its bias towards highly branching attributes
- Gain ratio takes the number and the size of branches into account when choosing an attribute (disregarding the class values!)
  - It corrects the information gain by taking the *intrinsic information* of a split into account
- Intrinsic information: entropy of distributing instances into branches (i.e. how much info do we need to tell which branch an instance belongs to)

# Computing the gain ratio

- Example: intrinsic information for ID code

$$\text{info}([1,1,\ldots,1]) = 14 \times (-1/14 \times \log 1/14) = 3.807 \text{ bits}$$

- Value of attribute decreases as intrinsic information gets larger
- Definition of gain ratio:

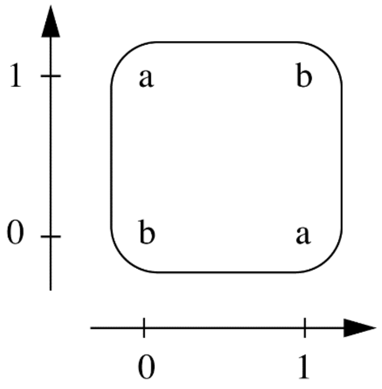
$$\text{gain\_ratio}(\text{"Attribute"}) = \frac{\text{gain}(\text{"Attribute"})}{\text{intrinsic\_info}(\text{"Attribute"})}$$

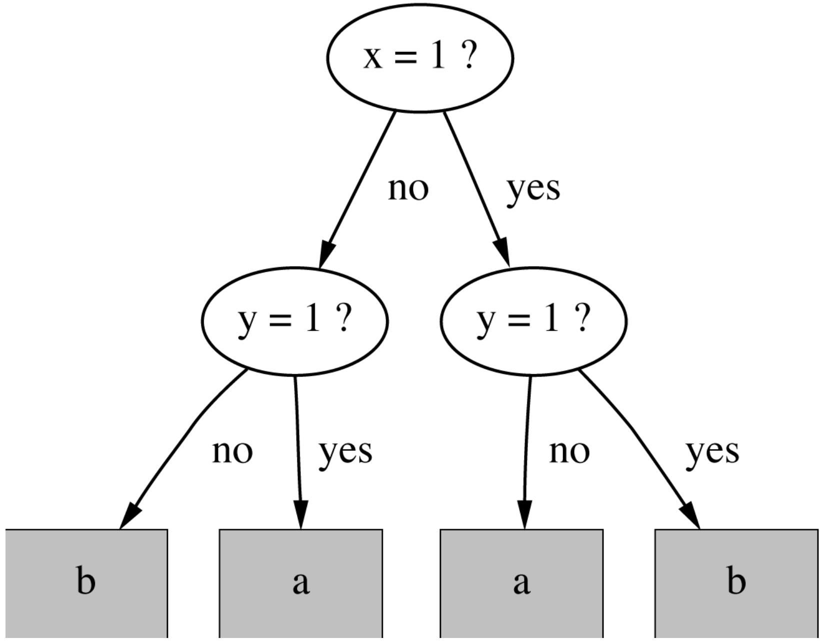
- Example: $\text{gain\_ratio}(\text{"ID\_code"}) = \dfrac{0.940 \text{ bits}}{3.807 \text{ bits}} = 0.246$

# Gain ratios for weather data

| Outlook | | Temperature | |
|---|---|---|---|
| Info: | 0.693 | Info: | 0.911 |
| Gain: 0.940-0.693 | 0.247 | Gain: 0.940-0.911 | 0.029 |
| Split info: info([5,4,5]) | 1.577 | Split info: info([4,6,4]) | 1.362 |
| Gain ratio: 0.247/1.577 | 0.156 | Gain ratio: 0.029/1.362 | 0.021 |
| Humidity | | Windy | |
| Info: | 0.788 | Info: | 0.892 |
| Gain: 0.940-0.788 | 0.152 | Gain: 0.940-0.892 | 0.048 |
| Split info: info([7,7]) | 1.000 | Split info: info([8,6]) | 0.985 |
| Gain ratio: 0.152/1 | 0.152 | Gain ratio: 0.048/0.985 | 0.049 |

# Knowledge representation: decision tree (extension): XOR





- awkward representation
- hard to learn
- XOR problem fortunately rare

# Discussion

- Algorithm for top-down induction of decision trees (“ID3”) was developed by Ross Quinlan
  - Led to development of C4.5, which can deal with numeric attributes, missing values, and noisy data
- Similar approach: Regression tree (CART: Classification and Regression Tree)
- There are many other attribute selection criteria! (But almost no difference in accuracy of result.)
- Decision Tree: „white box”
- Decision Trees: traditionally most widely used ML models!

# Classification Rule learning (Covering algorithm) (Generating rules) (PRISM)

# Knowlege Representation: Classification rules

- Popular alternative to decision trees

- *Antecedent* (pre-condition): a series of tests (just like the tests at the nodes of a decision tree)
  - tests are usually logically AND-ed together
  - but may also be general logical expressions
- *Consequent* (conclusion):
  - classes, or
  - set of classes, or
  - probability distribution assigned by rule
- Individual rules are often logically OR-ed together…

# Classification rule - example

antecedent

consequent

IF second person’s gender = female
AND first person’s parent1 = second person’s parent1
THEN sister-of = yes

Typically conjuctive antecedents

# “Nuggets” of knowledge

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

# Interpreting rules

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

# Special case: boolean class

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

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

- Order of rules is not important. No conflicts!

# From trees to rules

- Easy: converting a tree into a set of rules
  - one rule for each leaf:
    - antecedent contains a condition for every node on the path from the root to the leaf
    - consequent is class assigned by the leaf
- Produces rules that are unambiguous
  - doesn't matter in which order they are executed
- But: resulting rules are unnecessarily complex

# From rules to trees

- More difficult: transforming a rule set into a tree
  - tree cannot easily express disjunction between rules

# Linear models for regression (=predicting numeric value)

# Linear models

- Inputs (attribute values) and output are all numeric
- Output is the sum of weighted attribute values
  - $x = w_0 + w_1 a_1 + w_2 a_2 + \ldots + w_k a_k$

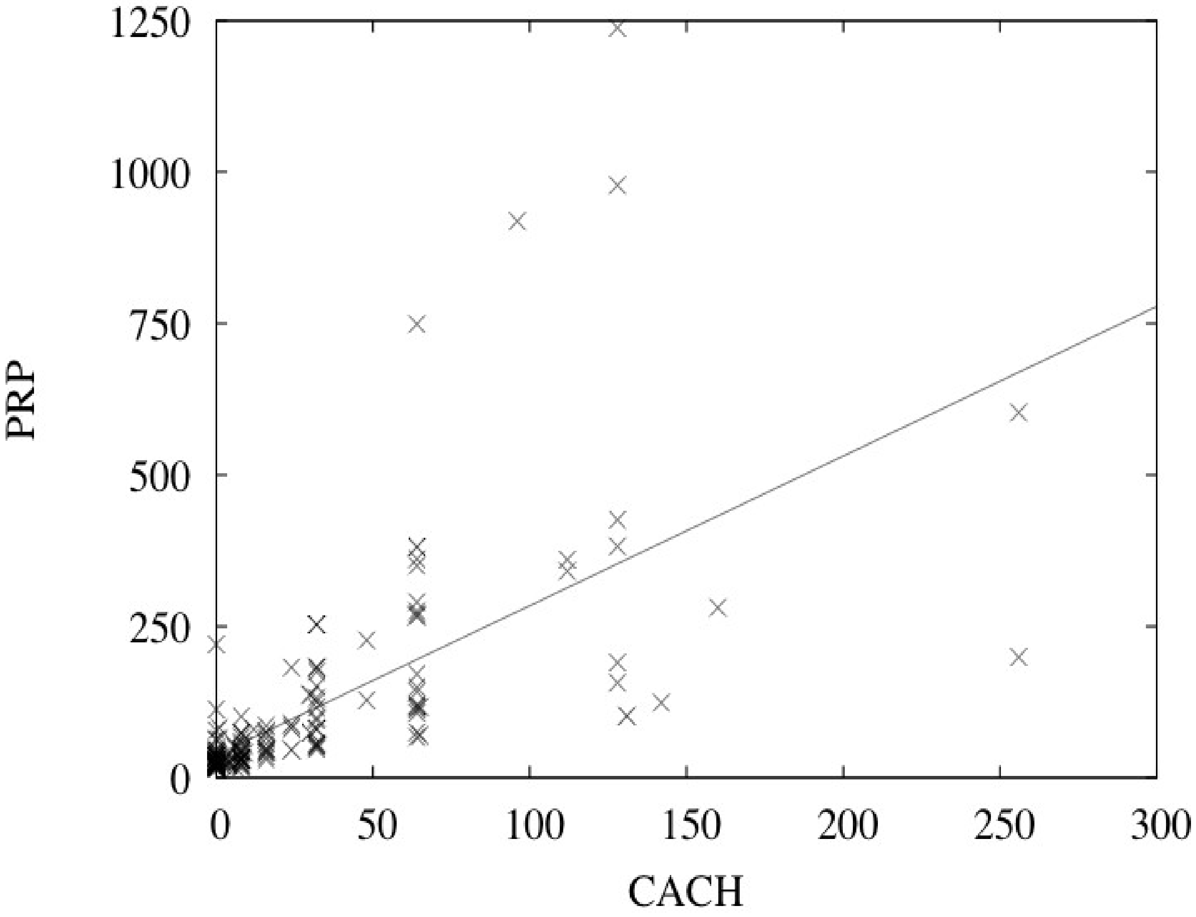
The trick is to find good values for the weights

# Example:
# Predicting CPU performance

- 209 different computer configurations

| | Cycle time (ns) | Main memory (Kb) | | Cache (Kb) | Channels | | Performance |
|---|---|---|---|---|---|---|---|
| | MYCT | MMIN | MMAX | CACH | CHMIN | CHMAX | PRP |
| *1* | 125 | 256 | 6000 | 256 | 16 | 128 | 198 |
| *2* | 29 | 8000 | 32000 | 32 | 8 | 32 | 269 |
| … | | | | | | | |
| *208* | 480 | 512 | 8000 | 32 | 0 | 0 | 67 |
| *209* | 480 | 1000 | 4000 | 0 | 0 | 0 | 45 |

# A linear regression function for the CPU performance data



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

# Linear regression

- Outcome is linear combination of attributes

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

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

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

# Goal: Minimum Mean Squared Error (MSE)

- *Mean Squared Error (MSE), or L2 loss*

$$\sum_{i=1}^{n}\left(x^{(i)} - \sum_{j=0}^{k} w_j a_j^{(i)}\right)^2$$

- *k*+1 coefficients are chosen so that MSE the training data is minimized
- Assumptions …
  - Error: Normal distribution

- MSE sensitive to outliers!
- Minimization of *absolute error* is more difficult!

# Training: Minimizing Mean Squared Error (MSE)

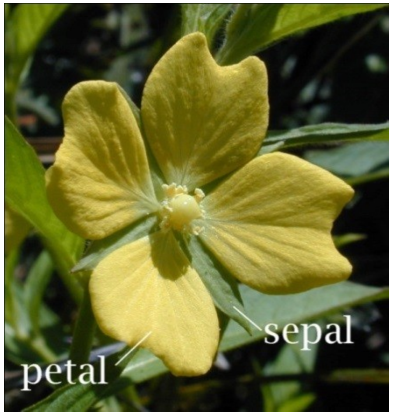
- Coefficients can be derived using matrix inversion
- Iterative solution -- gradient decent -- computationally less expensive

- Can be done if there are more instances than attributes (roughly speaking)

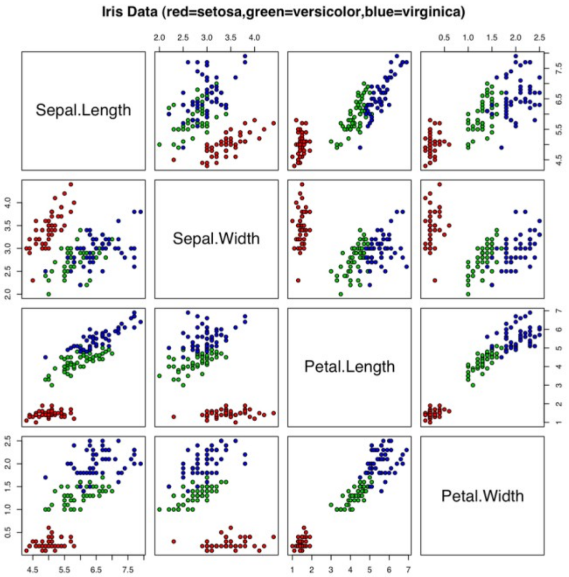
# Linear models for binary classification

- Binary classification
- Prediction is made by plugging in observed values of the attributes into the expression
  - Predict one class if output > 0 (or 0.5, …), and the other class if output < 0 (or 0.5, …)
- Decision boundary
  - defines where the decision changes from one class value to the other
  - hyperplane (high-dimensional plane)

# Example: Classifying iris flowers

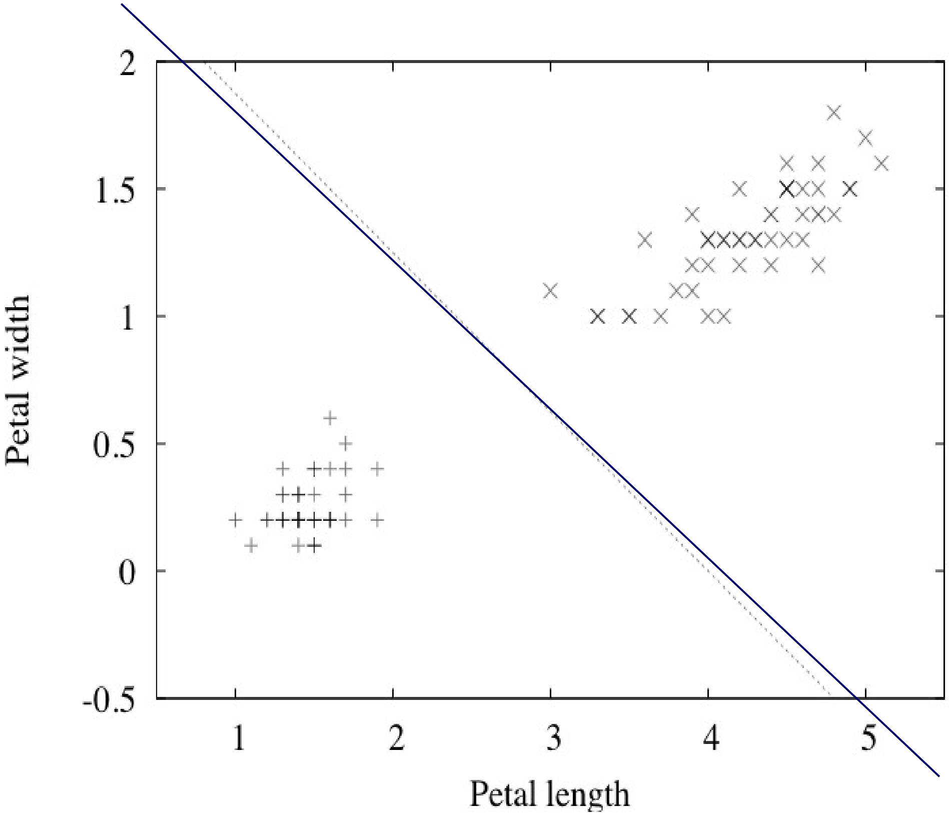


| | Sepal length | Sepal width | Petal length | Petal width | Type |
|---|---|---|---|---|---|
| *1* | 5.1 | 3.5 | 1.4 | 0.2 | Iris setosa |
| *2* | 4.9 | 3.0 | 1.4 | 0.2 | Iris setosa |
| … | | | | | |
| *51* | 7.0 | 3.2 | 4.7 | 1.4 | Iris versicolor |
| *52* | 6.4 | 3.2 | 4.5 | 1.5 | Iris versicolor |
| … | | | | | |
| | 6.3 | 3.3 | 6.0 | 2.5 | Iris virginica |
| | 5.8 | 2.7 | 5.1 | 1.9 | Iris virginica |



http://en.wikipedia.org/wiki/Iris_flower_data_set

# Separating setosas from versicolors



- **2.0 – 0.5PETAL-LENGTH – 0.8PETAL-WIDTH = 0 (2 input attributes)**

# Classification

- Problems
  - membership values are not in [0,1] range, so aren't proper probability estimates
  - assumptions used in least-square regression
    - normal distribution with same standard deviation for 0 -1 values ??!!

# Linear models: logistic regression

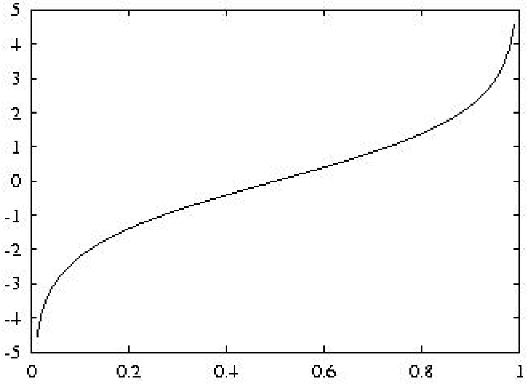
- Linear model for a transformed target variable
- Assume we have two classes
- Logistic regression replaces the target

$$P[1|a_{1,} a_{2,} \ldots., a_k]$$

- by this target

$$\log\left(\frac{P[1|a_{1,} a_{2,} \ldots., a_k]}{(1-P[1|a_{1,} a_{2,} \ldots., a_k])}\right)$$

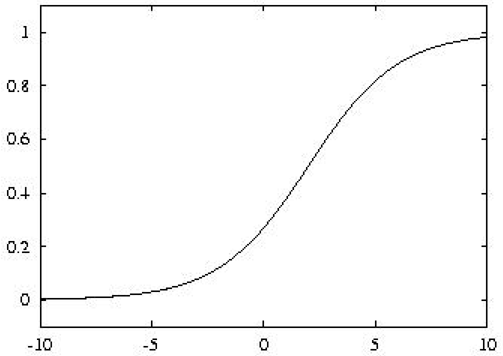
- Logit transformation maps [0,1] to (-oo , +oo )
(Inverse function of logit: sigmoid function)



# Logit transformation (Sigmoid function): Resulting model

$$Pr[1|a_{1,}a_{2,}\ldots,a_k]=\frac{1}{(1+e^{-w_0-w_1a_1-\ldots-w_ka_k})}$$

- Example
  - Model with
    - $w_0$ = 0.5 (moves curve left-right) and
    - $w_1$ = 1  (steepness)



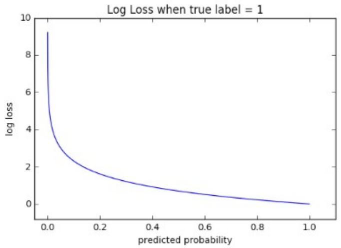
# Maximum likelihood

- Parameters are found from training data using *maximum likelihood*
- Aim: choosing parameters to maximize probability of training data („maximum likelihood”)
  - Independence: probabilities multiplied
  - Maximizing multiplied probabilites : maximizing sum of logarithms 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 $x^{(i)}$ are either 0 or 1

Log loss $\sum$ $-y_i z_i + \log(1+e^{z_i})$



# Logistic regression 2

- The independent variables should be independent of each other. That is, the model should have little or no multicollinearity.
- The independent variables are linearly related to logit function - probability.
- Logistic regression requires quite large sample sizes.

# Linear models are hyperplanes

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

$$Pr[1|a_1, a_2, \ldots, a_k] = 1/(1+\exp(-w_0 - w_1 a_1 - \ldots - w_k a_k)) = 0.5$$

- which occurs when $-w_0 - w_1 a_1 - \ldots - w_k a_k = 0$

# Linear models: the perceptron

- Don't actually need probability estimates if all we want to do is classification
- Different approach: learn separating hyperplane
- Assumption: data is *linearly separable*
- Hyperplane: $0 = w_0 a_0 + w_1 a_1 + w_2 a_2 + \ldots + w_k a_k$

  where we again assume that there is a constant attribute with value 1 (*bias*)
- If sum is greater than zero we predict the first class, otherwise the second class

# Perceptron learning rule

```
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 situation where instance *a* pertaining to the first class has been added:

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

  This means output for *a* has increased by:

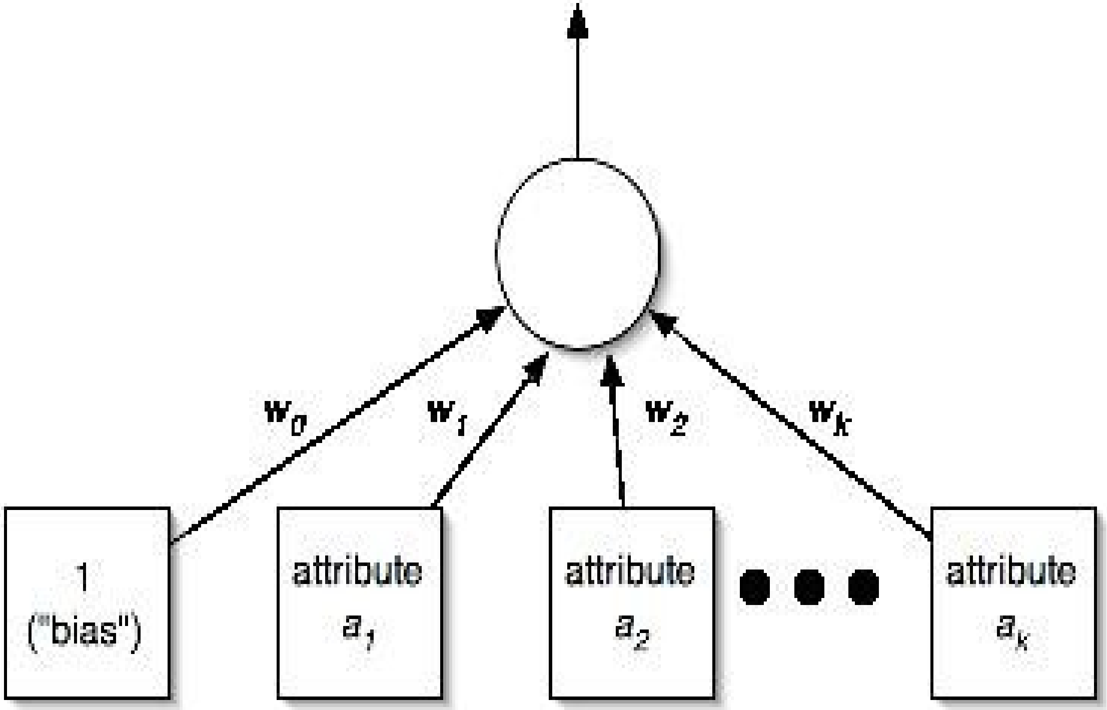
  This number is

  $$a_0a_0+a_1a_1+a_2a_2+\ldots+a_ka_k$$

  has moved into the correct direction (and we can show output decreases for instances of other class)
- Convergence guaranteed -- if data linearly separable --, upper bound on number of steps

# Perceptron as a neural network

Output layer

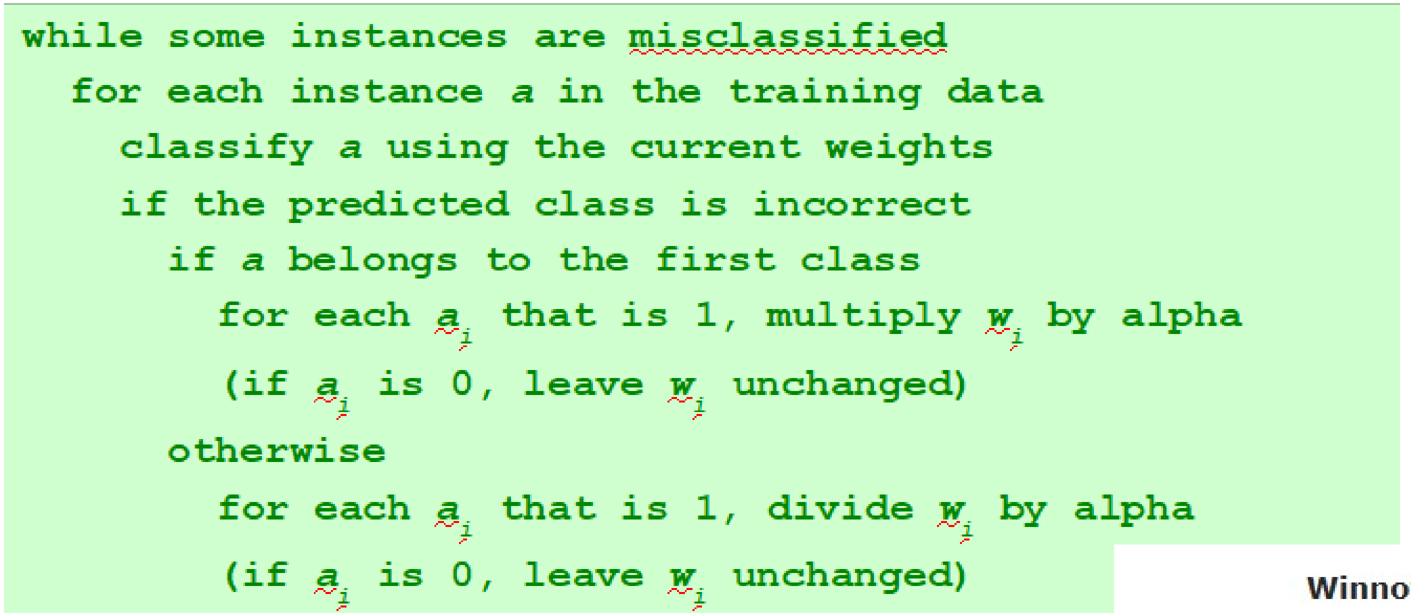


Input layer

# Linear models: Winnow

- Binary classification
- Mistake-driven
- **Multiplicative updates** (perceptron: additive updates):
  - Weights are multiplied (or divided) by a user-specified parameter (alpha > 1)
- User-specific threshold (perceptron: 0)
  - Predict first class, if: $w_0a_0+w_1a_1+w_2a_2+\ldots+w_ka_k>\theta$

# Winnow learning rule



```
while some instances are misclassified
  for each instance a in the training data
    classify a using the current weights
    if the predicted class is incorrect
      if a belongs to the first class
        for each $a_i$ that is 1, multiply $w_i$ by alpha
        (if $a_i$ is 0, leave $w_i$ unchanged)
      otherwise
        for each $a_i$ that is 1, divide $w_i$ by alpha
        (if $a_i$ is 0, leave $w_i$ unchanged)
```

- Effectively focuses on relevant features

Winnowing

# Instance-based learning

# Instance-based representation

- Simplest form of learning: *rote learning*
  - Training instances are searched for instance that most closely resembles new instance
  - The instances themselves represent the knowledge
    - ***Lazy learning!***
  - Also called *instance-based* learning
- Similarity function

# The distance function

- Simplest case: one numeric attribute
  - Distance is the difference between the two attribute values involved (or a function thereof)
- Several numeric attributes: typically Euclidean distance is used and attributes are **normalized**

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

- Takin[illegible] hen comparing distances
- Other popular metric: *city-block metric*
  - Adds differences without squaring them

# The distance function

- Nominal attributes: distance is set to 1 if values are different, 0 if they are equal
- Are all attributes equally important?
  - – weighting the attributes might be necessary

# Discussion of nearest-neighbor learning

- Statisticians have used *k*-NN since early 1950s
- Often very accurate

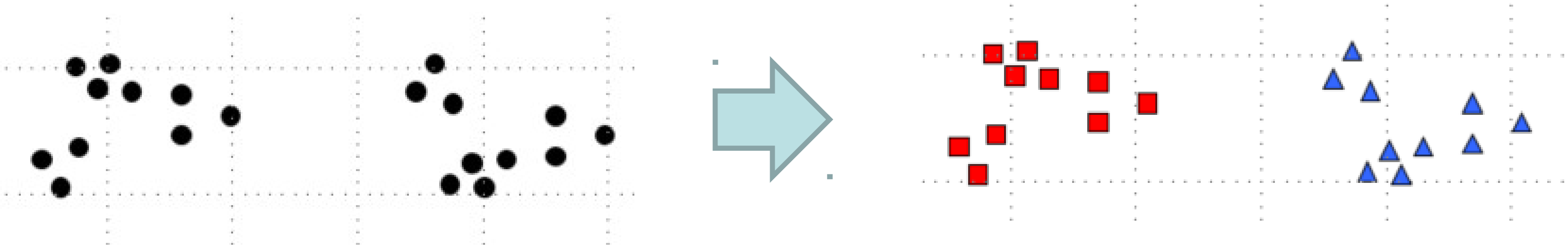
Limitations (of the basic algorithm)

- White box, generalisation ?!
- Speed (of prediction!!)
  - E.g. 1E6 instances, 100 attributes (dimensions)
- Noisy data ?!
- Domains of attributes different ?!
  - comparing e.g. [0,1] with [-5000,5000] ?
- Irrelevant attributes (only noise, with no relevance)

# Clustering

# Clustering: Knowledge Representation

- Divide instances into “natural” groups (clusters)
  - Instances in a group (cluster) similar to each other, in some or the other way
  - No class attribute, unlabelled data
  - Unsupervised learning

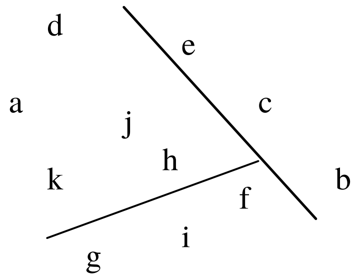


Cluster 1

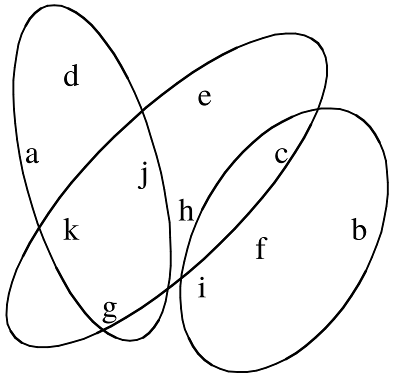
Cluster 2

# Representing clusters I

Simple 2-D representation



Venn diagram



**disjoint vs. overlapping**

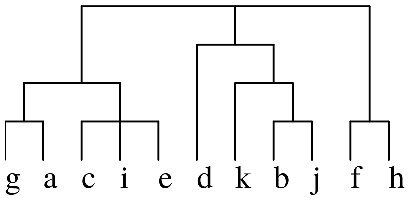
# Representing clusters II

## Probabilistic assignment

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

**probabilistic**
**(vs. deterministic)**

## Dendrogram



NB: dendron is the Greek word for tree

**hierarchical**
**(vs. flat)**

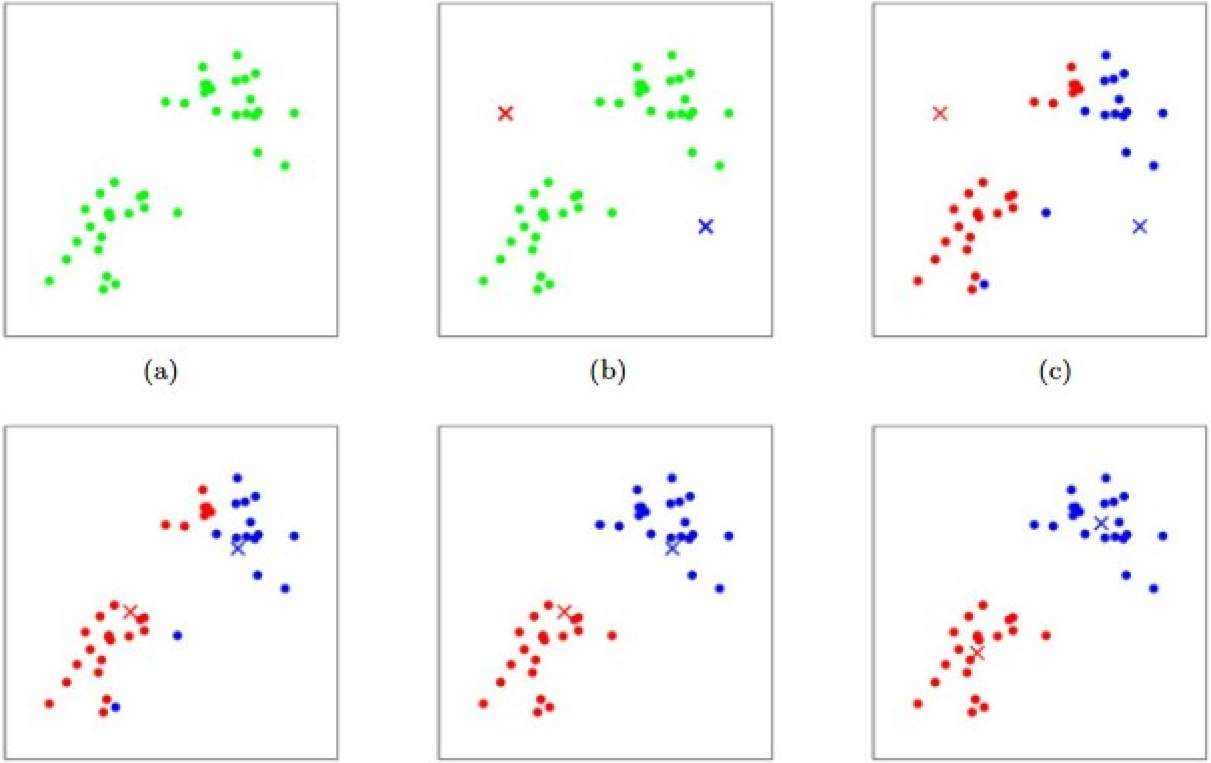
# The *k* -means algorithm

- disjoint, deterministic, and flat clusters
- number of clusters (k) preselected

1. Choose *k* cluster centers (e.g. randomly)
2. Repeat until convergence
    1. assign instances to clusters (based on distance; closest cluster center)
    2. compute *centroids* of clusters

(convergence: assignment of instances to clusters do not change)

# K-means: illustration



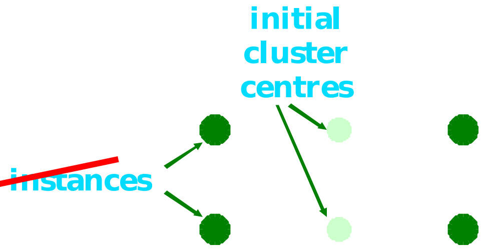
(a) (b) (c)

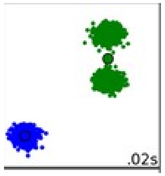
(d) (e) (f)

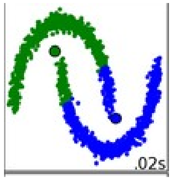
# Discussion

- Algorithm minimizes squared distance to cluster centers
- Converges
  - distortion function (==sum of squared errors) converges
  - typically also cluster centers and assignment of instances to clusters)
- Result can vary significantly
  - based on initial choice of seeds

    (restarting with different random seeds)
- Can get trapped in local minimum
  (in practice not typical)
- Number of clusters?
- Distance?
- Speed of distance calculations…
- Probabilistic, hierarhical, overlapping…







# Mining Association Rules
# (Apriori Algorithm)

# Association rules (knowledge representation)

- Form of rules
  If a and b and c occurs in the set, then t is also part of the set

  -
    ```
    if Milk = true and
       Bread = True
    then Eggs = true  and
     Coke = True
    ```
- Typically boolean attributes
  - Compact notation: **if** `Milk` and `Bread` **then** `Eggs` and `Coke`
- No selected class attribute
- Multiple attributes on consequent side possible/typical
- Rules interpreted individually!

  -
    ```
    if Milk and True then Eggs and Coke
    ```

# Association rule mining, applications

- Application areas
  - Market basket analysis
  - ~Sequence mining

MARKET BASKET ANALYSIS
98% of people who purchased items A and B also purchased item C

| *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** |

{Diaper} → {Beer},
{Milk, Bread} → {Eggs,Coke},
{Beer, Bread} → {Milk},

# Association rule mining

Immense number of potential rules…

a,**b,c,d,**e

**b,c,d**

a,**b,c,d**

a,**b,c,d,**e

a,c,e,f

d,c,e,f

a,**b,c,d,**f

→

Sufficiently **general** rules

**b,c,d** co-occur

in 71% of the cases

→

Sufficiently **correct/ precise** rules

**If b and c then d**
in 100% of the cases

# Support and confidence of a rule

- (Rule: X -> Y)
- *Support/coverage*: number of instances predicted correctly
  - Supp(X and Y) „*Itemset*" – no matter, whether an item (attribute) is on the left or on the right hand side of the rule
  - Can also be calculated as a relative number (relative to the number of all instances)
- *Confidence/accuracy*: number of correct predictions, as proportion of all instances that rule applies to
  - confidence=Supp(X and Y)/Supp(X)
- *Lift*: How popular is Y, when X, **considering also how popular Y is**
  - lift= Confidence of Rule / Relative support of Y
  - 1: no association,
    >1 Y is more likely when X,
    <1  Y is less likely when X

# Association rules
# Example for support & confidence

| 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 |

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

support = 4
confidence = 100%

```
If temperature = cool
then play= yes
```

support = 3
confidence = 75%

# Interpreting association rules

Interpretation is not obvious:

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

is *not* the same as the two rules

**If** windy = false
    **and** play = no
**then** outlook = sunny

**If** windy = false
    **and** play = no
**then** humidity = high

It means that the following **also** holds:

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

Why?

# Interpreting association rules

Interpretation is not obvious:

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

is *not* the same as the two rules

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

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

It means that the following **also** holds:

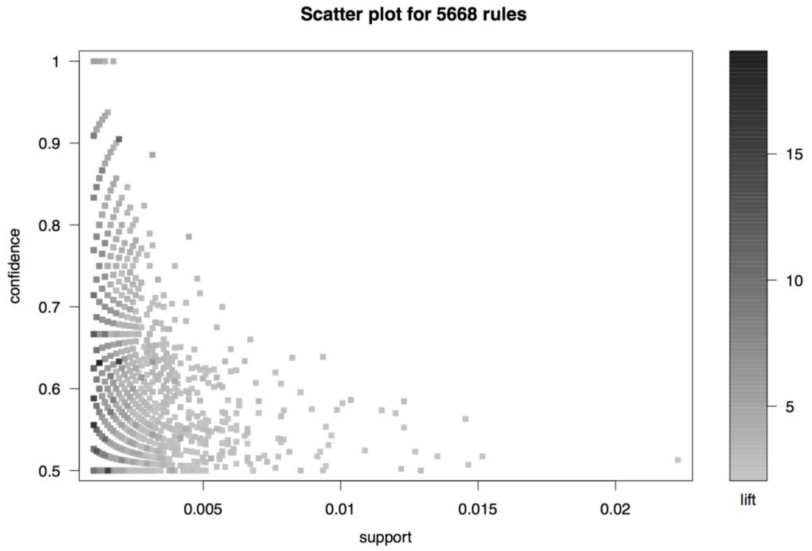
```
If humidity = high
    and windy = false
    and play = no
then outlook = sunny
```

Why?
Support…
Confidence…

# ASSOCIATION RULES: APRIORI ALGORITHM

# Mining association rules

- Normally: minimum support and confidence pre-specified (e.g. for weather data: support ≥ 2 and confidence ≥ 95%, -> 58 rules)
- Brute force: All rules, filtering on the bases of support and confidence
- Immense number of possible associations
- Computational complexity!



# Association rules

- But: we can look for high support rules (itemsets) and high confidence rules directly!
- Two steps
  1. High support (frequent item sets)
  2. High confidence

# Frequent Item sets

- 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 (that is, attribute-value pair)
- *Item set*: all items occurring in a rule (LHS and RHS !!)
- Goal: only rules that exceed pre-defined support

  ⇒ We can do it by finding all item sets with the given minimum support (=frequent item sets) and generating rules from them!

# Frequent Item Sets for weather data

| One-item sets | Two-item sets | Three-item sets | Four-item sets |
|---|---|---|---|
| Outlook = Sunny (5) | Outlook = Sunny<br>Temperature = Mild (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) |
| … | … | … | … |

- Number of frequent item sets (with minimum support of two) :
  - 12 with 1 item,
  - 47 with 2 items,
  - 39 with 3 items,
  - 6 with 4 items
  - 0 with 5 items

# Generating item sets efficiently

- How can we efficiently find all frequent item sets (item sets with support value higher than the threshold for support)?

- If (A B) is frequent item set, then (A) and (B) have to be frequent item sets as well!
  - Necessary, but not sufficient condition:
    if (A) and (B) are frequent item sets, (A B) is not necessaryily frequent, too.
- In general: if X is frequent *k*-item set, then all (*k*-1)-item subsets of X are also frequent

# Generating item sets efficiently

- Find frequent (support larger than threshold) one-item sets
- Compute candidate frequent *k*-item sets by merging frequent (*k*-1)-item sets
- Check (real) support of candidate frequent k-items sets
  by reading through the whole database

- Remark: The algorithm reduces the number of k-item sets, the support of which has to be checked by reading through the whole database.

# An 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 frequent four-item sets:

  `(A B C D)     candidate`

  `(B C D E)     no candidate, e.g., because of (C D E)`

- Check real support by counting instances in dataset!

# Generating rules from an item set

- Once all item sets with minimum support have been generated, we can turn them into rules
- Example: `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 | 4/14 |

Rules with same itemset: all k-1 consequent rules have at least the same confidence, as k consequent rules

# Generating rules efficiently

- We are looking for all high-confidence rules
  - Which items on the RHS of the rule?
  - But: brute-force method is ($2^N$-1)

- Observation: (*c* + 1)-consequent rule can only have a confidence above the threshold if all corresponding *c*-consequent rules also have

- Building (*c* + 1)-consequent rules from *c*-consequent ones!

# 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 candidate 2-consequent rule:

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

- Final check of antecedent against data!

# Discussion of association rules

- Above method makes one pass through the data for each different size item set
  - Other possibility: generate (*k*+2)-item sets just after (*k*+1)-item sets have been generated
    - Result: more (*k*+2)-item sets than necessary will be considered but less passes through the data
- Practical issue: generating a certain number of rules (e.g. by incrementally reducing min. support)

# Summary/concepts/questions

- 1R
  - Model
  - Algorithm
  - Unknown values
  - Numeric attributes
- Naive Bayes
  - Model
  - Algorithm (Learning, predicting)
  - Laplace estimator
  - Unknown values
  - Numeric attributes
- ID3 decision tree
  - Level of surprise, entropy, information gain, algorithm
- Rules: problem with interpretation

# Summary/concepts/questions - 2

- Linear models
  - Form of linear model for regression
  - Form of error to be minimized
- Linear models for classification
  - Logistic regression
  - Perceptron
  - Winnow
  - Instance based (Nearest neighbour) learning
  - „Knowledge representation”
  - Algorithm
  - Problems
- Clustering
  - Knowledge representation
  - K-means: algorithm + limitations
- Association rule mining
  - Knowledge representation
  - Item, itemset, support, confidence, lift
  - Apriori algorithm