Machine Learning Techniques for Data Mining

Eibe Frank
University of Waikato
New Zealand

PART IV

Algorithms:

The basic methods

Simplicity first

- Simple algorithms often work surprisingly well
- Many different kinds of simple structure exist:
  - One attribute might do all the work
  - All attributes might contribute independently with equal importance
  - A linear combination might be sufficient
  - An instance-based representation might work best
  - Simple logical structures might be appropriate
- Success of method depends on the domain!

Inferring rudimentary rules

- 1R: learns a 1-level decision tree
  - In other words, generates a set of rules that all test on one particular attribute
- Basic version (assuming nominal 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 with lowest error rate

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
```

- Note: “missing” is always treated as a separate attribute value

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	
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	

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 (majority) class changes (so that the total error is minimized)
- 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
- Also: *time stamp* attribute will have zero errors
- 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
```

Result of overfitting avoidance

- Final result for for *temperature* attribute:

```
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
```

- Resulting rule sets:

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

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)
  - Minimum number of instances was set to 6 after some experimentation
  - 1R's simple rules performed not much worse than much more complex decision trees
- Simplicity first pays off!

Statistical modeling

- "Opposite" of 1R: use all the attributes
- Two assumptions: Attributes are
  - *equally important*
  - *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)
- Although based on assumptions that are almost never correct, this scheme works well in practice!

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	?

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

Bayes's rule

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

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

- *A priori* probability of *H*: $\Pr[H]$
  - Probability of event *before* evidence has been seen
- *A posteriori* probability of *H*: $\Pr[H \mid E]$
  - Probability of event *after* evidence has been seen

Naïve Bayes for classification

- Classification learning: what's the probability of the class given an instance?
  - Evidence E = instance
  - Event H = class value for instance
- Naïve Bayes assumption: evidence can be split into independent parts (i.e. attributes of instance!)

$$\Pr[H \mid E] = \frac{\Pr[E_1 \mid H]\Pr[E_1 \mid H]\ldots\Pr[E_n \mid H]\Pr[H]}{\Pr[E]}$$

The weather data example

Outlook	Temp.	Humidity	Windy	Play
Sunny	Cool	High	True	?

Evidence E

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

Probability for class "yes"

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

The “zero-frequency problem”

- What if an attribute value doesn’t occur with every class value (e.g. “Humidity = high” for class “yes”)?
  - Probability will be zero! $\Pr[Humdity = High \mid yes] = 0$
  - *A posteriori* probability will also be zero! $\Pr[yes \mid E] = 0$

    (No matter how likely the other values are!)
- Remedy: add 1 to the count for every attribute value-class combination (*Laplace estimator)*
- Result: probabilities will never be zero! (also: stabilizes probability estimates)

Modified probability estimates

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

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

Sunny *Overcast* *Rainy*

- Weights don't need to be equal (if they sum to 1)

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

Missing values

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

Outlook	Temp.	Humidity	Windy	Play
?	Cool	High	True	?

Likelihood of “yes” = $3/9 \times 3/9 \times 3/9 \times 9/14 = 0.0238$

Likelihood of “no” = $1/5 \times 4/5 \times 3/5 \times 5/14 = 0.0343$

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* or *Gaussian* probability distribution (given the class)
- The *probability density function* for the normal distribution is defined by two parameters:
  - The *sample mean* μ: $$\mu = \frac{1}{n}\sum_{i=1}^{n} x_i$$
  - The *standard deviation* σ: $$\sigma = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \mu)^2$$
  - 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					
				…	…		…	…					
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:

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

Classifying a new day

- A new day:

Outlook	Temp.	Humidity	Windy	Play
Sunny	66	90	true	?

Likelihood of “yes” = $2/9 \times 0.0340 \times 0.0221 \times 3/9 \times 9/14 = 0.000036$

Likelihood of “no” = $3/5 \times 0.0291 \times 0.0380 \times 3/5 \times 5/14 = 0.000136$

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

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

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

Probability densities

- Relationship between probability and density:

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

- But: this doesn't change calculation of *a posteriori* probabilities because ε cancels out
- Exact relationship:

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

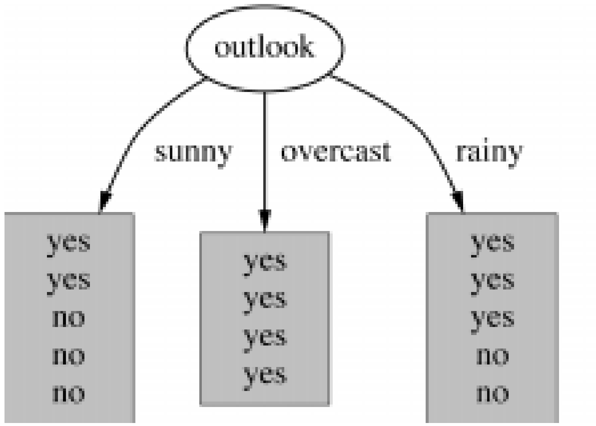
Discussion of Naïve Bayes

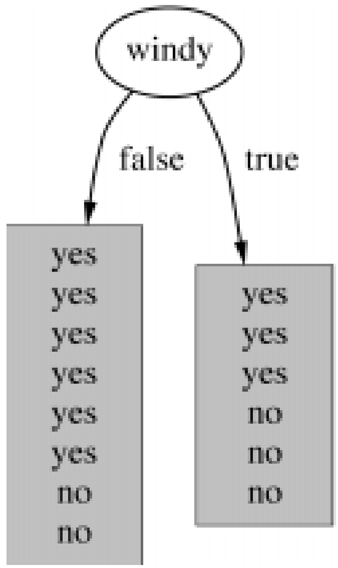
- Naïve Bayes works surprisingly well (even if independence assumption is clearly violated)
- Why? Because classification doesn't require accurate probability estimates *as long as maximum probability is assigned to correct class*
- However: adding too many redundant attributes will cause problems (e.g. identical attributes)
- Note also: many numeric attributes are not normally distributed ($\rightarrow$ *kernel density estimators*)

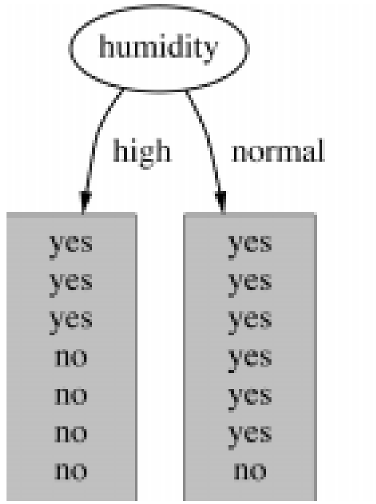
Constructing decision trees

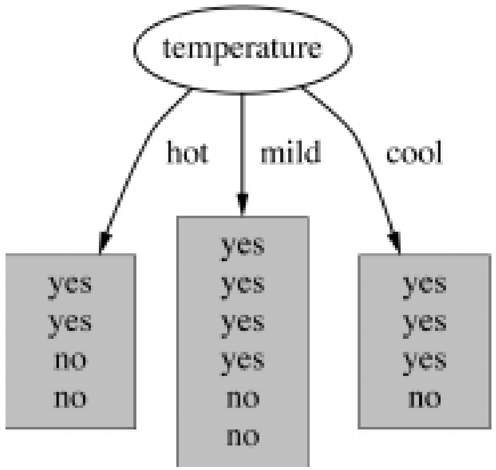
- Normal procedure: top down in recursive *divide-and-conquer* fashion
  - First: attribute is selected for root node and branch is created for each possible attribute value
  - Then: the instances are split into subsets (one for each branch extending from the node)
  - Finally: procedure is repeated recursively for each branch, using only instances that reach the branch
- Process stops if all instances have the same class

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

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:

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

Example: attribute "Outlook"

- "Outlook" = "Sunny":

$$\text{info}([2,3]) = \text{entropy}(2/5,3/5) = -2/5\log(2/5) - 3/5\log(3/5) = 0.971\text{ bits}$$

- "Outlook" = "Overcast":

$$\text{info}([4,0]) = \text{entropy}(1,0) = -1\log(1) - 0\log(0) = 0\text{ bits}$$

Note: this is normally not defined.

- "Outlook" = "Rainy":

$$\text{info}([3,2]) = \text{entropy}(3/5,2/5) = -3/5\log(3/5) - 2/5\log(2/5) = 0.971\text{ bits}$$

- Expected information for attribute:

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

$$= 0.693\text{ bits}$$

Computing the information gain

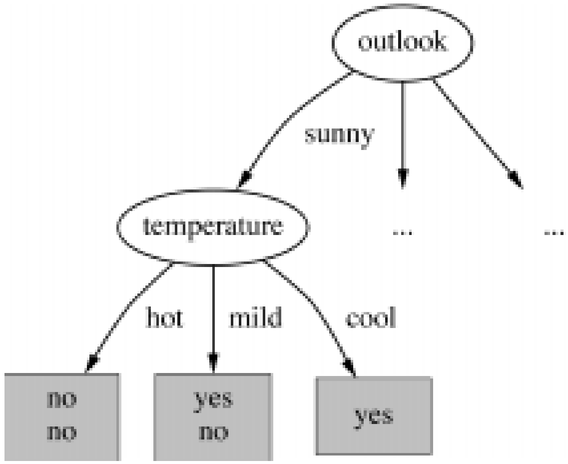
- Information gain: information before splitting – information after splitting

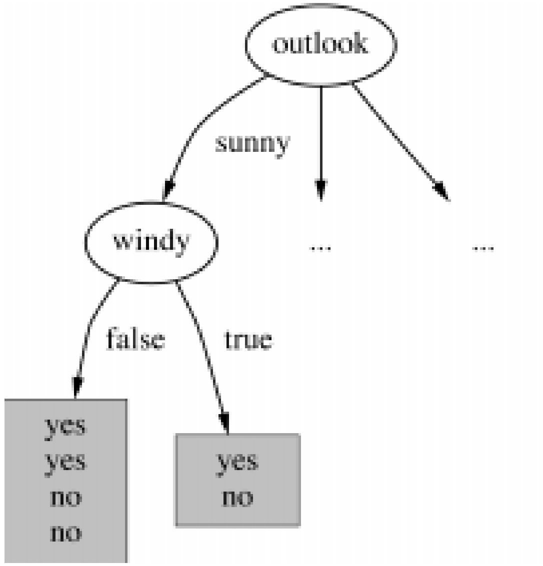
$$\text{gain}(\text{"Outlook"}) = \text{info}([9,5]) - \text{info}([2,3],[4,0,[3,2]) = 0.940 - 0.693$$
$$= 0.247 \text{ bits}$$

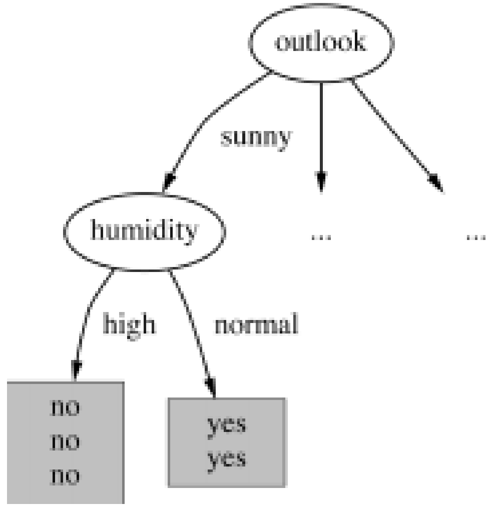
- Information gain for attributes from weather data:

$$\text{gain}(\text{"Outlook"}) = 0.247 \text{ bits}$$
$$\text{gain}(\text{"Temperature"}) = 0.029 \text{ bits}$$
$$\text{gain}(\text{"Humidity"}) = 0.152 \text{ bits}$$
$$\text{gain}(\text{"Windy"}) = 0.048 \text{ bits}$$

Continuing to split





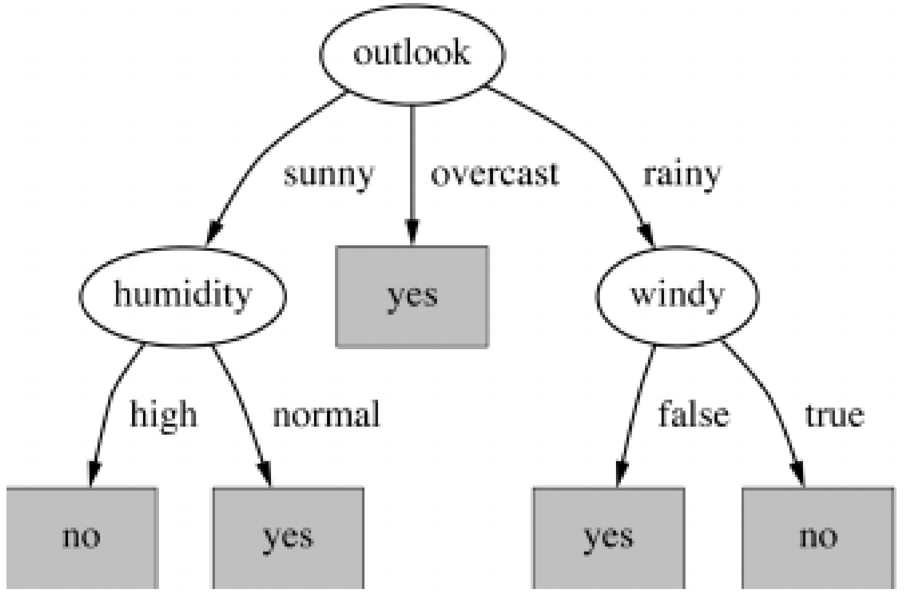


$$\text{gain}(\text{"Temperature"}) = 0.571 \text{ bits}$$

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

$$\text{gain}(\text{"Windy"}) = 0.020 \text{ bits}$$

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

Wishlist for a purity measure

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

Some properties of the entropy

- The multistage property:

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

- Simplification of computation:

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

$$= [-2\log 2 - 3\log 3 - 4\log 4 + 9\log 9]/9$$

- Note: instead of maximizing info gain we could just minimize information

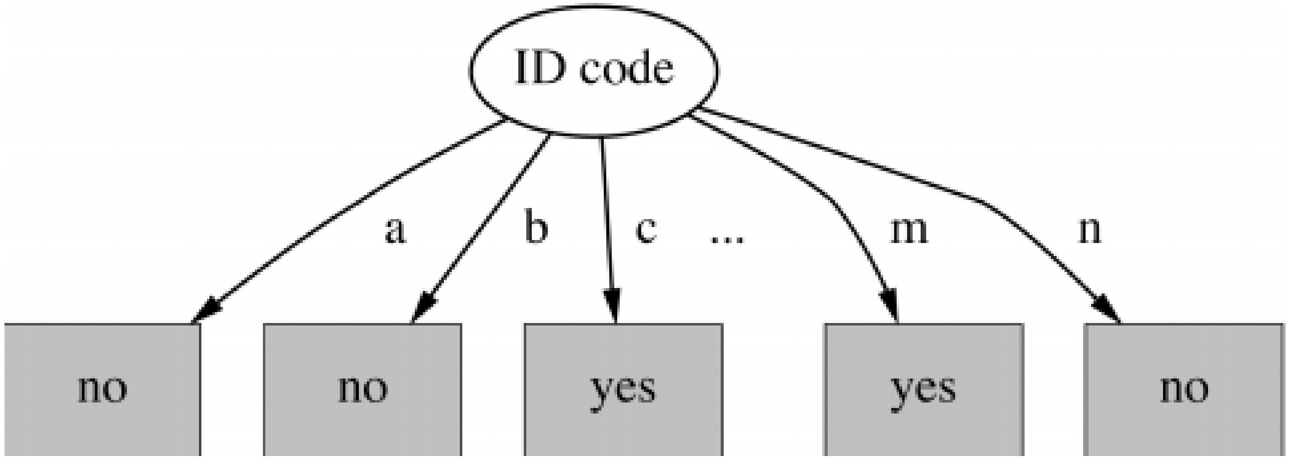
Highly-branching attributes

- 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)
- Another problem: *fragmentation*

The weather data with ID code

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:

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

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

More on the gain ratio

- “Outlook” still comes out top
- However: “ID code” has greater gain ratio
  - Standard fix: *ad hoc* test to prevent splitting on that type of attribute
- Problem with gain ratio: it may overcompensate
  - May choose an attribute just because its intrinsic information is very low
  - Standard fix: only consider attributes with greater than average information gain

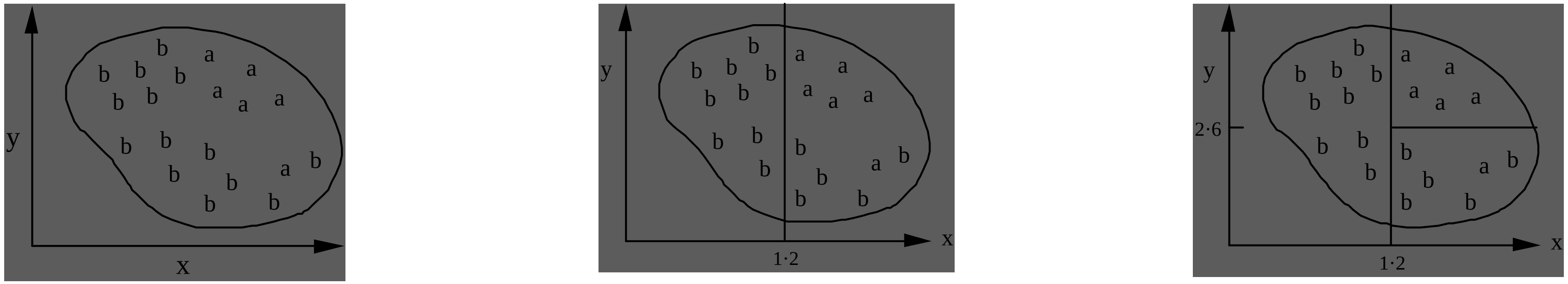
Discussion

- Algorithm for top-down induction of decision trees ("ID3") was developed by Ross Quinlan
  - Gain ratio just one modification of this basic algorithm
  - Led to development of C4.5, which can deal with numeric attributes, missing values, and noisy data
- Similar approach: CART
- There are many other attribute selection criteria! (But almost no difference in accuracy of result.)

Covering algorithms

- Decision tree can be converted into a rule set
  - Straightforward conversion: rule set overly complex
  - More effective conversions are not trivial
- Strategy for generating a rule set directly: for each class in turn find rule set that covers all instances in it (excluding instances not in the class)
- This approach is called a *covering* approach because at each stage a rule is identified that covers some of the instances

Example: generating a rule



If true then class = a

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

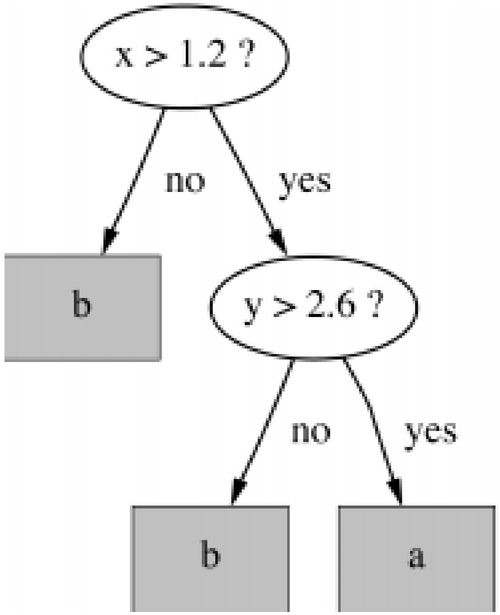
If x > 1.2 then class = a

- Possible rule set for class "b":

  If x ≤ 1.2 then class = b

  If x > 1.2 and y ≤ 2.6 then class = b
- More rules could be added for "perfect" rule set

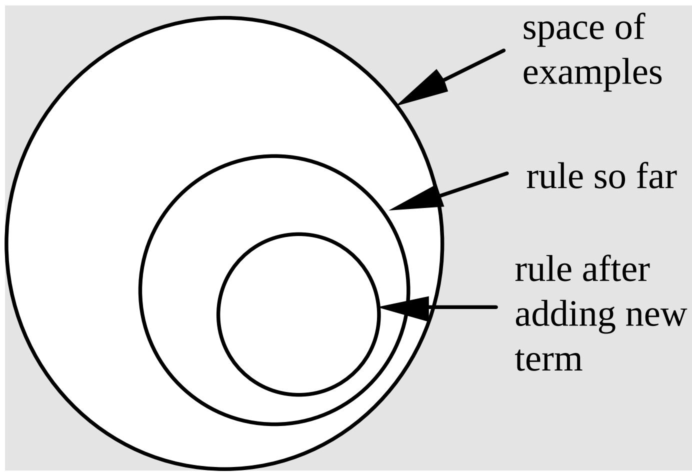
Rules vs. trees



- 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

A simple covering algorithm

- Generates a rule by adding tests that maximize 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:



Selecting a test

- Goal: maximizing accuracy
  - *t*: total number of instances covered by rule
  - *p*: positive examples of the class covered by rule
  - *t-p*: number of errors made by rule

  ⇒ Select test that maximizes the ratio *p/t*
- We are finished when *p/t* = 1 or the set of instances can't be split any further

Example: contact lenses 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

Modified rule and resulting data

- Rule with best test added:

```
If astigmatics = 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

Further refinement

- Current state:

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

- Possible tests:

Age = Young	2/4
Age = Pre-presbyopic	1/4
Age = Presbyopic	1/4
Spectacle prescription = Myope	3/6
Spectacle prescription = Hypermetrope	1/6
Tear production rate = Reduced	0/6
Tear production rate = Normal	4/6

Modified rule and resulting data

- Rule with best test added:

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

- Instances covered by modified rule:

Age	Spectacle prescription	Astigmatism	Tear production rate	Recommended lenses
Young	Myope	Yes	Normal	Hard
Young	Hypermetrope	Yes	Normal	hard
Pre-presbyopic	Myope	Yes	Normal	Hard
Pre-presbyopic	Hypermetrope	Yes	Normal	None
Presbyopic	Myope	Yes	Normal	Hard
Presbyopic	Hypermetrope	Yes	Normal	None

Further refinement

- Current state:
- Possible tests:

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

Age = Young	2/2
Age = Pre-presbyopic	1/2
Age = Presbyopic	1/2
Spectacle prescription = Myope	3/3
Spectacle prescription = Hypermetrope	1/3

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

The result

- Final rule:

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

- Second rule for recommending “hard lenses”:

  (built from instances not covered by first rule)

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

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

Pseudo-code for PRISM

```
For each class C
  Initialize E to the instance set
  While E contains instances in class C
    Create a rule R with an empty left-hand side that predicts class C
    Until R is perfect (or there are no more attributes to use) do
      For each attribute A not mentioned in R, and each value v,
        Consider adding the condition A = v to the 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
```

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 doesn't matter because all rules predict the same class
- Outer loop considers all classes separately
  - No order dependence implied
- Problems: overlapping rules, default rule required

Separate and conquer

- Methods like PRISM (for dealing with one class) are *separate-and-conquer* algorithms:
  - First, a rule is identified
  - Then, all instances covered by the rule are separated out
  - Finally, the remaining instances are "conquered"
- Difference to divide-and-conquer methods:
  - Subset covered by rule doesn't need to be explored any further

Mining association rules

- Naïve method for finding association rules:
  - Using the standard separate-and-conquer method, treating 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)
- But: we can look for high support rules directly!

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/attribute-value pair
- *Item set*: all items occurring in a rule
- Goal: only rules that exceed pre-defined support

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

Item sets for weather data

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

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

Generating rules from an item set

- Once all item sets with minimum support have been generated, we can turn them into rules
- 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  4/12
```

Rules for the weather data

- 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, and 50 with support two

Example rules from the same 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
```

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)
```

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 frequent item set, then (A) and (B) have to be frequent item sets as well!
  - In general: if X is frequent *k*-item set, then all (*k*-1)-item subsets of X are also frequent

  $\Rightarrow$ Compute *k*-item set by merging (*k*-1)-item sets

An example

- Given: five 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 (B C D)
(A C D E)        Not OK because of (C D E)
```

- Final check by counting instances in dataset!
- (*k*-1)-item sets are stored in hash table

Generating rules efficiently

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

Example

- 1-consequent rules:

```
If 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 hash table!

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
  - Makes sense if data too large for main memory
- Practical issue: generating a certain number of rules (e.g. by incrementally reducing min. support)

Other issues

- ARFF format very inefficient for typical *market basket data*
  - Attributes represent items in a basket and most items are usually missing
- Instances are also called *transactions*
- Confidence is not necessarily the best measure
  - Example: milk occurs in almost every supermarket transaction
  - Other measures have been devised (e.g. lift)

Linear models

- Work most naturally with numeric attributes
- Standard technique for numeric prediction: 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 $\mathbf{a}^{(1)}$

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

Minimizing the squared error

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

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 is known as *multi-response linear regression*

Theoretical justification

Observed target value (either 0 or 1)

Model

Instance

The scheme minimizes this

$$E_y\{(f(X)-Y)^2 \mid X=x\}$$

True class probability

$$= E_y\{(f(X)-P(Y=1 \mid X=x)+P(Y=1 \mid X=x)-Y)^2 \mid X=x\}$$

$$= (f(x)-P(Y=1 \mid X=x))^2 + 2\times(f(x)-P(Y=1 \mid X=x))\times$$
$$E_y\{P(Y=1 \mid X=x)-Y \mid X=x\} + E_y\{(P(Y=1 \mid X=x)-Y)^2 \mid X=x\}$$

$$= (f(x)-P(Y=1 \mid X=x))^2 + 2\times(f(x)-P(Y=1 \mid X=x))\times$$
$$(P(Y=1 \mid X=x)-E_y\{Y \mid X=x\}) + E_y\{(P(Y=1 \mid X=x)-Y)^2 \mid X=x\}$$

$$= \underbrace{(f(x)-P(Y=1 \mid X=x))^2}_{\text{We want to minimize this}} + \underbrace{E_y\{(P(Y=1 \mid X=x)-Y)^2 \mid X=x\}}_{\text{Constant}}$$

Pairwise regression

- Another way of using regression for classification:
  - A regression function for every *pair* of classes, using only instances from these two classes
  - An output of +1 is assigned to one member of the pair, an output of –1 to the other
- Prediction is done by voting
  - Class that receives most votes is predicted
  - Alternative: “don’t know” if there is no agreement
- More likely to be accurate but more expensive

Logistic regression

- Problem: some assumptions violated when linear regression is applied to classification problems
- *Logistic* regression: alternative to linear regression
  - Designed for classification problems
  - Tries to estimate class probabilities directly
    - Does this using the *maximum likelihood* method
  - Uses the following linear model:

$$\log(P/(1-P)) = w_0 a_0 + w_1 a_1 + w_2 a_2 + \ldots + w_k a_k$$

Class probability

Discussion of linear models

- Not appropriate if data exhibits non-linear dependencies
- But: can serve as building blocks for more complex schemes (i.e. model trees)
- Example: multi-response linear regression defines a *hyperplane* for any two given classes:

$$(w_0^{(1)} - w_0^{(2)})a_0 + (w_1^{(1)} - w_1^{(2)})a_1 + (w_2^{(1)} - w_2^{(2)})a_2 + \ldots + (w_k^{(1)} - w_k^{(2)})a_k > 0$$

- Obviously the same for pairwise linear regression

Instance-based learning

- Distance function defines what's learned
- Most instance-based schemes use *Euclidean distance*:

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

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

Normalization and other issues

- Different attributes are measured on different scales $\Rightarrow$ they need to be *normalized*:

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

  v_i: the actual value of attribute i
- Nominal attributes: distance either 0 or 1
- Common policy for missing values: assumed to be maximally distant (given normalized attributes)

Discussion of 1-NN

- Often very accurate but also slow: simple version scans entire training data to derive a prediction
- Assumes all attributes are equally important
  - Remedy: attribute selection or weights
- Possible remedies against noisy instances:
  - Taking a majority vote over the *k* nearest neighbors
  - Removing noisy instances from dataset (difficult!)
- Statisticians have used *k*-NN since early 1950s
  - If $n \rightarrow \infty$ and $k/n \rightarrow 0$, error approaches minimum

Comments on basic methods

- Bayes' rule stems from his "Essay towards solving a problem in the doctrine of chances" (1763)
  - Difficult bit: estimating prior probabilities
  - Prior-free analysis generates confidence intervals
- Extension of Naïve Bayes: Bayesian Networks
- Algorithm for association rules is called APRIORI
- Minsky and Papert (1969) showed that linear classifiers have limitations, e.g. can't learn XOR
  - But: combinations of them can ($\rightarrow$Neural Networks)