Data Mining

Practical Machine Learning Tools and Techniques

Slides for Chapter 4, Algorithms: the basic methods

of *Data Mining* by I. H. Witten, E. Frank, M. A. Hall and C. J. Pal

Algorithms: The basic methods

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

Simplicity first

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

Inferring rudimentary rules

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

Pseudo-code for 1R

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

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

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 $\rightarrow$ No	2/5	4/14
	Overcast $\rightarrow$ Yes	0/4	
	Rainy $\rightarrow$ Yes	2/5	
Temp	Hot $\rightarrow$ No*	2/4	5/14
	Mild $\rightarrow$ Yes	2/6	
	Cool $\rightarrow$ Yes	1/4	
Humidity	High $\rightarrow$ No	3/7	4/14
	Normal $\rightarrow$ Yes	1/7	
Windy	False $\rightarrow$ Yes	2/8	5/14
	True $\rightarrow$ No*	3/6	

* indicates a tie

Dealing with numeric attributes

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

```
 64    65    68   69   70     71   72  72     75   75     80    81    83      85
Yes  | No  | Yes  Yes  Yes  | No   No  Yes  | Yes  Yes  | No  | Yes   Yes  |  No
```

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

The problem of overfitting

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

64 65 68 69 70 71 72 72 75 75 80 81 83 85
Yes ~~|~~ No ~~|~~ Yes Yes Yes | No No Yes ~~|~~ Yes Yes | No ~~|~~ Yes Yes ~~|~~ No

64 65 68 69 70 71 72 72 75 75 80 81 83 85
Yes No Yes Yes Yes ~~|~~ No No Yes Yes Yes | No Yes Yes No

Results with overfitting avoidance

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

Attribute	Rules	Errors	Total errors
Outlook	Sunny $\rightarrow$ 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):

  Very Simple Classification Rules Perform Well on Most Commonly Used Datasets

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

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

Simple probabilistic modeling

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

Probabilities for weather data

Outlook			Temperature			Humidity			Windy			Play	
	Yes	*No*		*Yes*	*No*		*Yes*	*No*		*Yes*	*No*	*Yes*	*No*
Sunny	2	3	Hot	2	2	High	3	4	False	6	2	9	5
Overcast	4	0	Mild	4	2	Normal	6	1	True	3	3		
Rainy	3	2	Cool	3	1								
Sunny	2/9	3/5	Hot	2/9	2/5	High	3/9	4/5	False	6/9	2/5	9/ 14	5/ 14
Overcast	4/9	0/5	Mild	4/9	2/5	Normal	6/9	1/5	True	3/9	3/5		
Rainy	3/9	2/5	Cool	3/9	1/5								

Outlook	**Temp**	**Humidity**	**Windy**	**Play**
Sunny	**Hot**	**High**	**False**	**No**
Sunny	**Hot**	**High**	**True**	**No**
Overcast	**Hot**	**High**	**False**	**Yes**
Rainy	**Mild**	**High**	**False**	**Yes**
Rainy	**Cool**	**Normal**	**False**	**Yes**
Rainy	**Cool**	**Normal**	**True**	**No**
Overcast	**Cool**	**Normal**	**True**	**Yes**
Sunny	**Mild**	**High**	**False**	**No**
Sunny	**Cool**	**Normal**	**False**	**Yes**
Rainy	**Mild**	**Normal**	**False**	**Yes**
Sunny	**Mild**	**Normal**	**True**	**Yes**
Overcast	**Mild**	**High**	**True**	**Yes**
Overcast	**Hot**	**Normal**	**False**	**Yes**
Rainy	**Mild**	**High**	**True**	**No**

Probabilities for weather data

Outlook			Temperature			Humidity			Windy			Play	
	Yes	*No*		*Yes*	*No*		*Yes*	*No*		*Yes*	*No*	*Yes*	*No*
Sunny	2	3	Hot	2	2	High	3	4	False	6	2	9	5
Overcast	4	0	Mild	4	2	Normal	6	1	True	3	3		
Rainy	3	2	Cool	3	1								
Sunny	2/9	3/5	Hot	2/9	2/5	High	3/9	4/5	False	6/9	2/5	9/ 14	5/ 14
Overcast	4/9	0/5	Mild	4/9	2/5	Normal	6/9	1/5	True	3/9	3/5		
Rainy	3/9	2/5	Cool	3/9	1/5								

- A new day:

Outlook	Temp.	Humidity	Windy	Play
Sunny	Cool	High	True	?

Likelihood of the two classes

For “yes” = 2/9 × 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:

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

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

Can combine probabilities using Bayes's rule

- Famous rule from probability theory due to

  Thomas Bayes

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

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

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

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

Naïve Bayes for classification

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

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

Weather data example

Outlook	Temp.	Humidity	Windy	Play
Sunny	Cool	High	True	?

$\longleftarrow$ ***Evidence E***

$$P(yes \mid E) = P(Outlook = Sunny \mid yes)$$

$$P(Temperature = Cool \mid yes)$$

$$P(Humidity = High \mid yes)$$

$$P(Windy = True \mid yes)$$

$$P(yes) / P(E)$$

Probability of class "yes"

$$= \frac{2/9 \times 3/9 \times 3/9 \times 3/9 \times 9/14}{P(E)}$$

The "zero-frequency problem"

- What if an attribute value does not occur with every class value?
  (e.g., "Humidity = high" for class "yes")
  - Probability will be zero: $P(Humidity = High \mid yes) = 0$
  - *A posteriori* probability will also be zero: $P(yes \mid E) = 0$
    (Regardless of 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
- Additional advantage: stabilizes probability estimates computed from small samples of data

Modified probability estimates

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

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

Sunny ***Overcast*** ***Rainy***

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

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

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 × 3/9 × 3/9 × 9/14 = 0.0238

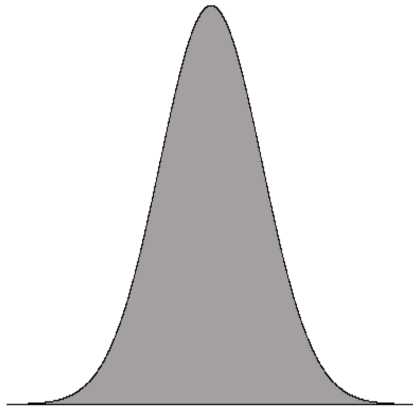
Likelihood of “no” = 1/5 × 4/5 × 3/5 × 5/14 = 0.0343

P(“yes”) = 0.0238 / (0.0238 + 0.0343) = 41%

P(“no”) = 0.0343 / (0.0238 + 0.0343) = 59%

Numeric attributes

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



Statistics for weather data

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

- Example density value:

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

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" $= 3/5 \times 0.0221 \times 0.0381 \times 3/5 \times 5/14 = 0.000108$

P("yes") $= 0.000036 / (0.000036 + 0.000108) = 25\%$

P("no") $= 0.000108 / (0.000036 + 0.000108) = 75\%$

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

Probability densities

- Probability densities *f*(*x*) can be greater than 1; hence, they are not probabilities
    - However, they must integrate to 1: the area under the probability density curve must be 1
- Approximate relationship between probability and probability density can be stated as

$$P(x-\varepsilon/2 \le X \le x+\varepsilon/2) \approx \varepsilon f(x)$$

  assuming ε is sufficiently small
- When computing likelihoods, we can treat densities just like probabilities

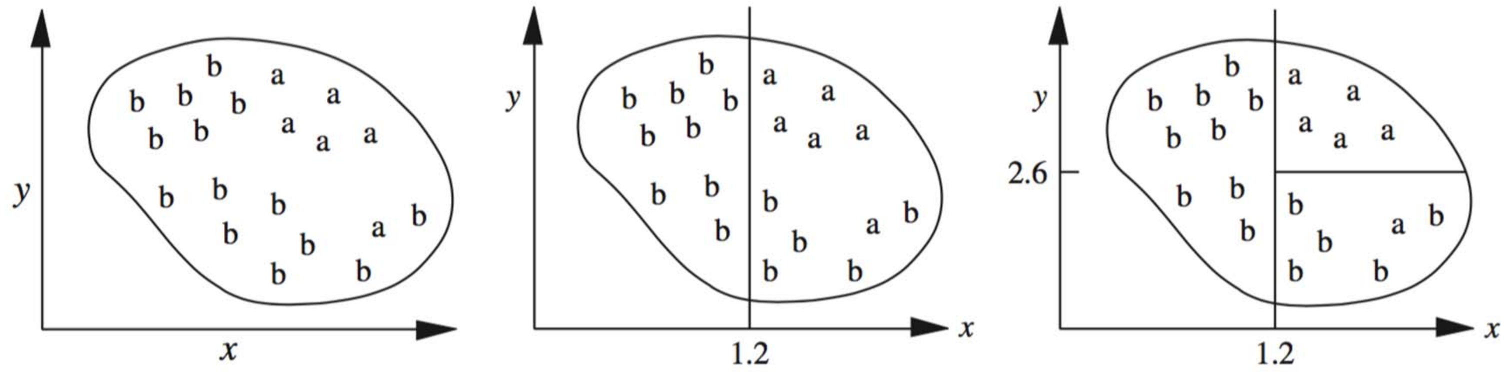
Naïve Bayes: discussion

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

Covering algorithms

- Can convert decision tree into a rule set
  - Straightforward, but rule set overly complex
  - More effective conversions are not trivial and may incur a lot of computation
- Instead, we can generate rule set directly
  - One approach: for each class in turn, find rule set that covers all instances in it
    (excluding instances not in the class)
- Called a *covering* approach:
  - At each stage of the algorithm, a rule is identified that “covers” some of the instances

Example: generating a rule



**If true
  then class = a**

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

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

- Possible rule set for class "b":

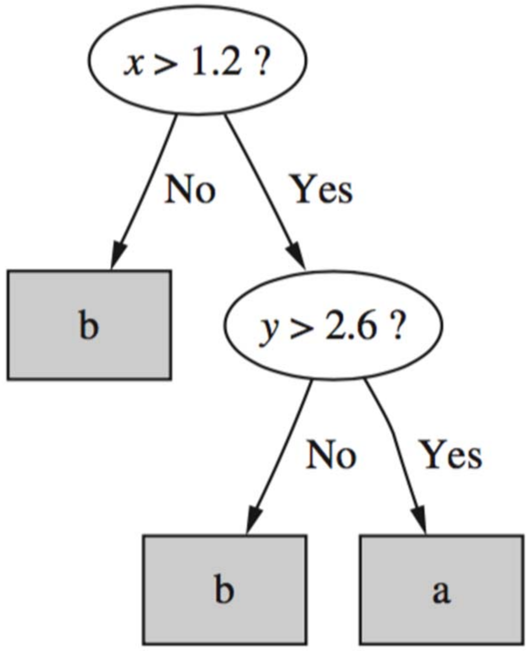
  If x ≤ 1.2 then class = b

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

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

Rules vs. trees

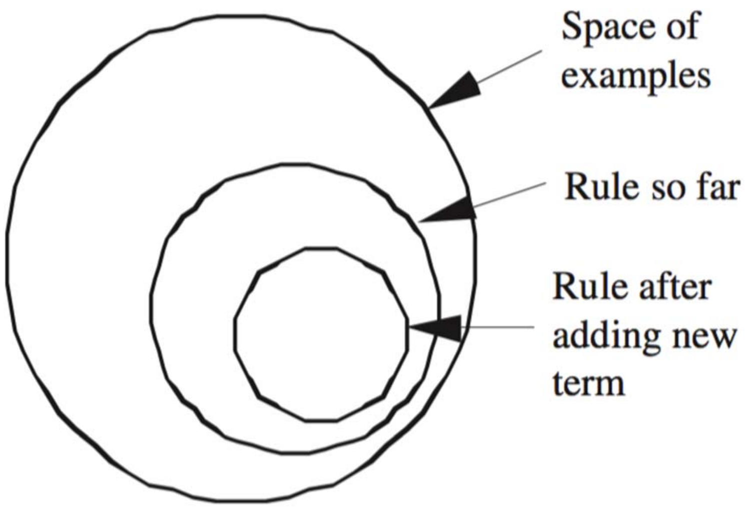
- Corresponding decision tree: (produces exactly the same predictions)



- 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

Simple covering algorithm

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



Selecting a test

- Goal: maximize accuracy
    - *t* total number of instances covered by rule
    - *p* positive examples of the class covered by rule
    - *t* – *p* number of errors made by rule
    - Select test that maximizes the ratio *p/t*
- We are finished when *p/t* = 1 or the set of instances cannot be split any further

Example: contact lens data

- Rule we seek:

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

- Possible tests:

Age = Young	2/8
Age = Pre-presbyopic	1/8
Age = Presbyopic	1/8
Spectacle prescription = Myope	3/12
Spectacle prescription = Hypermetrope	1/12
Astigmatism = no	0/12
Astigmatism = yes	4/12
Tear production rate = Reduced	0/12
Tear production rate = Normal	4/12

Modified rule and resulting data

- Rule with best test added:

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

- Instances covered by modified rule:

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

Further refinement

- Current state:

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

- Possible tests:

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

Modified rule and resulting data

- Rule with best test added:

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

- Instances covered by modified rule:

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

Further refinement

- Current state:

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

- Possible tests:

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

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

The final rule

- Final rule:

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

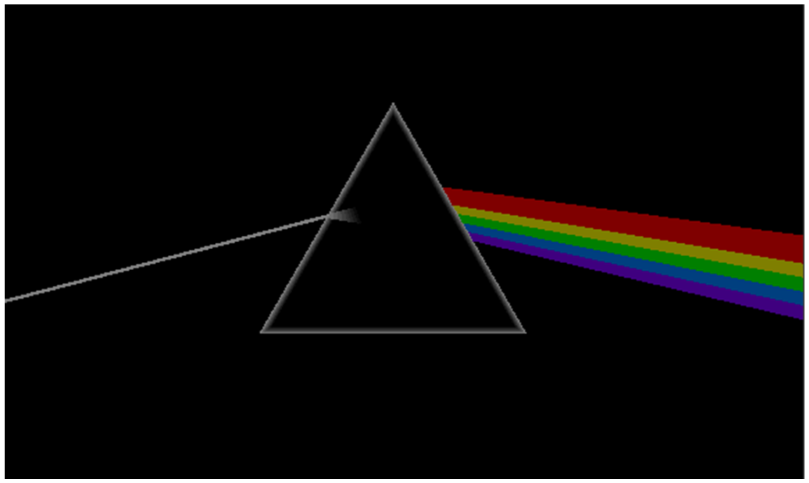
- Second rule for recommending “hard lenses”:
  (built from instances not covered by first rule)

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

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

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 does not matter because all rules predict the same class so outcome does not change if rules are shuffled
- Outer loop considers all classes separately
  - No order dependence implied
- Problems: overlapping rules, default rule required

Separate and conquer rule learning

- Rule learning methods like the one PRISM employs (for each class) are called *separate-and-conquer* algorithms:
  - First, identify a useful rule
  - Then, separate out all the instances it covers
  - Finally, “conquer” the remaining instances
- Difference to divide-and-conquer methods:
  - Subset covered by a rule does not need to be explored any further