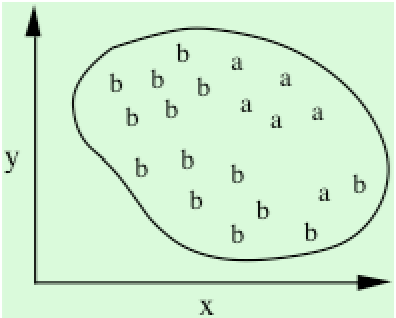
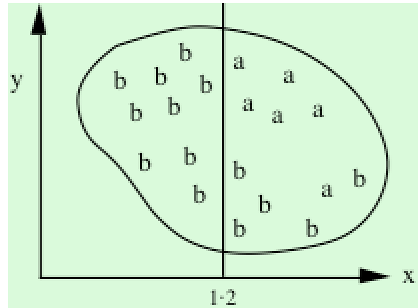
Learning from Observations – Rules

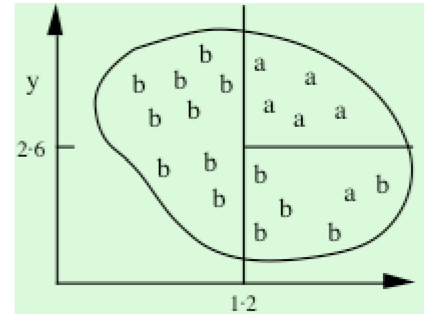
Covering Algorithms

- Can convert decision tree into a rule set
  - Straightforward, but rule set overly complex
  - More effective conversions are not trivial
- Instead, can generate rule set directly
  - For each class in turn find rule set that covers all instances in it (excluding instances not in the class)
- Called a *covering* approach:
  - At each stage a rule is identified that "covers" some of the instances

Example: Generating a Rule







**If ???
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”:

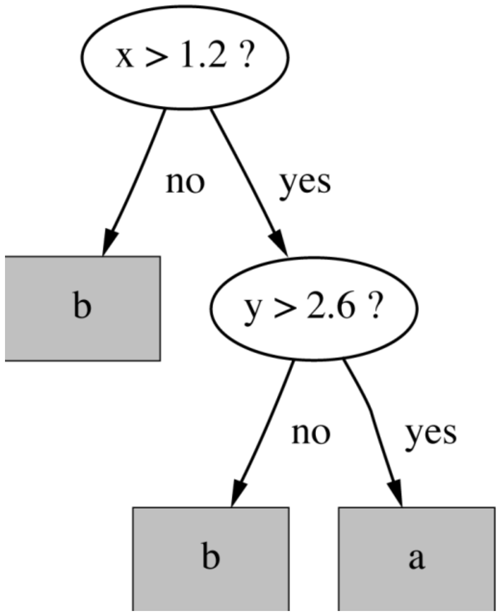
  If x ≤ 1.2 then class = b

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

- Could add more rules, get “perfect” rule set

Rules vs. Trees

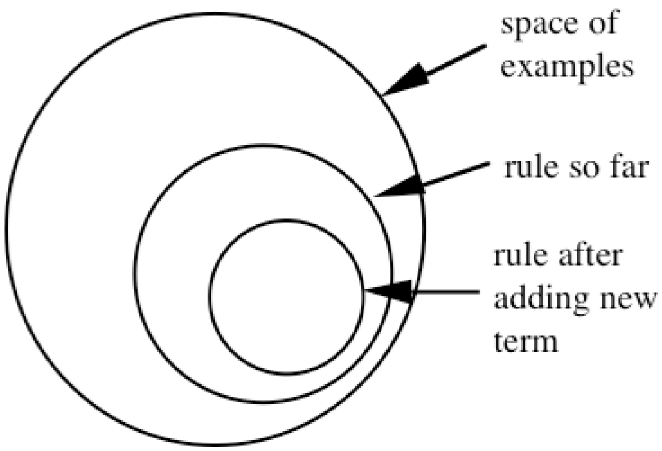
Corresponding decision tree:
(produces exactly the same predictions)



- But rule sets *can* be clearer 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

- 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: Maximize Accuracy
  - *t* total number of instances covered by rule
  - *p* positive examples of the class covered by rule
  - *t* – *p* number of errors made by rule

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

Example: Contact Lens Data

- 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

- Current state:

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

Further Refinement

- Possible tests:

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

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

The Result

- Final rule:

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

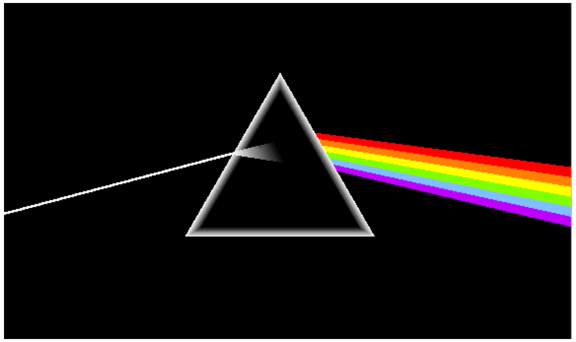
- Second rule for recommending "hard lenses":
(built from instances not covered by first rule)

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

- These two rules cover all "hard lenses":
  - The process is then 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, identify a useful rule
  - Then, separate out all the instances it covers
  - Finally, "conquer" the remaining instances
- Difference to divide-and-conquer methods:
  - Subset covered by rule doesn't need to be explored any further

Missing Values, Numeric Attributes

- Common treatment of missing values: *for any test, they fail*
  - Algorithm must either
    - Use other tests to separate out positive instances
    - Leave them uncovered until later in the process
- In some cases it's better to treat "missing" as a separate value
- Numeric attributes are treated just like they are in decision trees

Pruning Rules

- Two main strategies:
  - *Incremental* pruning
  - *Global* pruning
- Other difference: pruning criterion
  - Error on hold-out set (*reduced-error pruning*)
  - Statistical significance
  - MDL principle

Using a Pruning Set

- For statistical validity, must evaluate measure on data not used for training:
  - This requires a *growing set* and a *pruning set*
- *Reduced-error pruning* :
  - Build full rule set and then prune it
- *Incremental reduced-error pruning* :
  - Simplify each rule as soon as it is built
    - Can re-split data after rule has been pruned
- Stratification advantageous

Variations

- Generating rules for classes in order
  - Start with the smallest class
  - Leave the largest class covered by the default rule
- Stopping criterion
  - Stop rule production if accuracy becomes too low
- Rule learner RIPPER:
  - Uses MDL-based stopping criterion
  - Employs post-processing step to modify rules guided by MDL criterion

Using Global Optimization

- RIPPER: *Repeated Incremental Pruning to Produce Error Reduction* (does global optimization in an efficient way)
  - Classes are processed in order of increasing size
  - Initial rule set for each class is generated using IREP
- An MDL-based stopping condition is used
- Once a rule set has been produced for each class, each rule is re-considered and two variants are produced
  - One is an extended version, one is grown from scratch
  - Chooses among three candidates according to *DL*
- Final clean-up step greedily deletes rules to minimize DL

PART

- Avoids global optimization step used in C4.5rules and RIPPER
- Builds a *partial* decision tree to obtain a rule
- Uses C4.5's procedures to build a tree

Notes on PART

- Make leaf with maximum coverage into a rule
- Treat missing values just as C4.5 does
    - i.e. split instance into pieces
- Time taken to generate a rule:
    - Worst case: same as for building a pruned tree
        - Occurs when data is noisy
    - Best case: same as for building a single rule
        - Occurs when data is noise free

Rules with Exceptions

1. Given: a way of generating a single good rule
2. Then it’s easy to generate rules with exceptions
3. Select default class for top-level rule
4. Generate a good rule for one of the remaining classes
5. Apply this method recursively to the two subsets produced by the rule (i.e. instances that are covered/not covered)