Data Mining

3.3 Rule-Based Classification

Fall 2008

Instructor: Dr. Masoud Yaghini

Outline

- Using IF-THEN Rules for Classification
- Rules With Exceptions
- Rule Extraction from a Decision Tree
- 1R Algorithm
- Sequential Covering Algorithms
- PRISM Algorithm
- FOIL Algorithm
- References

Using IF-THEN Rules for Classification

Rule-Based Classifier

- Classify records by using a collection of

  IF *condition* THEN *conclusion*

- Rule: (Condition) $\rightarrow$ y
  - where
    - Condition includes one or more attribute tests
    - y is the class prediction
- LHS & RHS:
  - LHS: **rule antecedent** or **precondition**
  - RHS: **rule consequent**

Using IF-THEN rules for classification

- An example is rule $R1$:

$$\text{R1: IF } age = youth \text{ AND } student = yes \text{ THEN } buys\_computer = yes$$

  - The condition consists of one or more *attribute tests* (such as $age = youth$, and $student = yes$) that are logically ANDed
  - The rule's consequent contains a class prediction (in this case, we are predicting whether a customer will buy a computer)

- $R1$ can also be written as

$$\text{R1: } (age = youth) \wedge (student = yes) \Rightarrow (buys\_computer = yes)$$

Assessment of a Rule

- A rule *R* can be assessed by its **coverage** and **accuracy**.
- **Coverage of a rule:**
  – The percentage of instances that satisfy the antecedent of a rule (i.e., whose attribute values hold true for the rule's antecedent).
- **Accuracy of a rule:**
  – The percentage of instances that satisfy both the antecedent and consequent of a rule

Rule Coverage and Accuracy

- We can define the **coverage** and **accuracy** of rule *R* as:

$$coverage(R) = \frac{n_{covers}}{|D|}$$

$$accuracy(R) = \frac{n_{correct}}{n_{covers}}$$

- *where*
  - *D*: class labeled data set
  - $|D|$: number of instances in *D*
  - n_{covers} : number of instances covered by R
  - $n_{correct}$: number of instances correctly classified by R

Example: *AllElectronics*

RID	*age*	*income*	*student*	*credit_rating*	*Class: buys_computer*
1	youth	high	no	fair	no
2	youth	high	no	excellent	no
3	middle_aged	high	no	fair	yes
4	senior	medium	no	fair	yes
5	senior	low	yes	fair	yes
6	senior	low	yes	excellent	no
7	middle_aged	low	yes	excellent	yes
8	youth	medium	no	fair	no
9	youth	low	yes	fair	yes
10	senior	medium	yes	fair	yes
11	youth	medium	yes	excellent	yes
12	middle_aged	medium	no	excellent	yes
13	middle_aged	high	yes	fair	yes
14	senior	medium	no	excellent	no

Coverage and Accuracy

- The rule $R1$:

$$\text{R1: IF } age = youth \text{ AND } student = yes \text{ THEN } buys\_computer = yes$$

  - $R1$ covers 2 of the 14 instances
  - It can correctly classify both instances
- Therefore:
  - $Coverage(R1) = 2/14 = 14.28\%$
  - $Accuracy(R1) = 2/2 = 100\%$.

Executing a rule set

- 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

How We Can Use Rule-based Classification

- An example, We would like to classify instance of ***X*** according to *buys_computer:*

  X= (*age = youth, income = medium, student = yes, credit rating = fair*)

- If a rule is satisfied by ***X***, the rule is said to be **triggered**
- Potential problems:
  - If more than one rule is satisfied by ***X***
    - Solution: conflict resolution strategy
  - if no rule is satisfied by ***X***
    - Solution: Use a default class

Conflict Resolution

- **Conflict resolution strategy**
  - **Size ordering (**rule antecedent size ordering)
    - Assign the highest priority to the triggering rules that has the "toughest" requirement where toughness is measured by the rule antecedent *size.* (i.e., with the most attribute test)
  - **Rule Ordering**
    - **Class-based ordering:**
      - Decreasing order of most frequent
      - Most rule-based classification systems use a class-based rule-ordering strategy.
    - **Rule-based ordering** (decision list):
      - Rules are organized into one long priority list, according to some measure of rule quality such as accuracy or coverage, or by experts

Default Rule

- If no rule is satisfied by ***X,*** How can we determine the class label of ***X?***
  - In this case, a default rule can be set up to specify a default class, based on a training set.
  - This may be the class in majority or the majority class of the instances that were not covered by any rule.
  - The default rule is evaluated at the end, if and only if no other rule covers ***X.***
  - The condition in the default rule is empty.
  - In this way, the rule fires when no other rule is satisfied.

Rules With Exceptions

Rules with exceptions

- Idea: allow rules to have *exceptions*
- Example: rule for iris data

```
If petal length ≥ 2.45 and petal length < 4.45 then Iris versicolor
```

- New instance:

Sepal length (cm)	Sepal width (cm)	Petal length (cm)	Petal width (cm)	Type
5.1	3.5	2.6	0.2	*Iris setosa*

- Modified rule:

```
If petal length ≥ 2.45 and petal length < 4.45 then
   Iris versicolor EXCEPT if petal width < 1.0 then Iris setosa
```

A more complex example

- Exceptions to exceptions to exceptions …

```
Default: Iris-setosa
except if petal-length ≥ 2.45 and petal-length < 5.355
          and petal-width < 1.75
       then Iris-versicolor
            except if petal-length ≥ 4.95 and petal-width < 1.55
                   then Iris-virginica
                   else if sepal-length < 4.95 and sepal-width ≥ 2.45
                        then Iris-virginica
       else if petal-length ≥ 3.35
            then Iris-virginica
                 except if petal-length < 4.85 and sepal-length < 5.95
                        then Iris-versicolor
```

Advantages of using exceptions

- Rules can be updated incrementally
  - Easy to incorporate new data
  - Easy to incorporate domain knowledge
- People often think in terms of exceptions
- Each conclusion can be considered just in the context of rules and exceptions that lead to it
  - Locality property is important for understanding large rule sets
  - “Normal” rule sets don’t offer this advantage

More on exceptions

- Default...except if...then...
  is logically equivalent to
- if...then...else
  (where the else specifies what the default did)
- But: exceptions offer a psychological advantage
  - Assumption: defaults and tests early on apply more widely than exceptions further down
  - Exceptions reflect special cases

Rule Extraction from a Decision Tree

Building Classification Rules

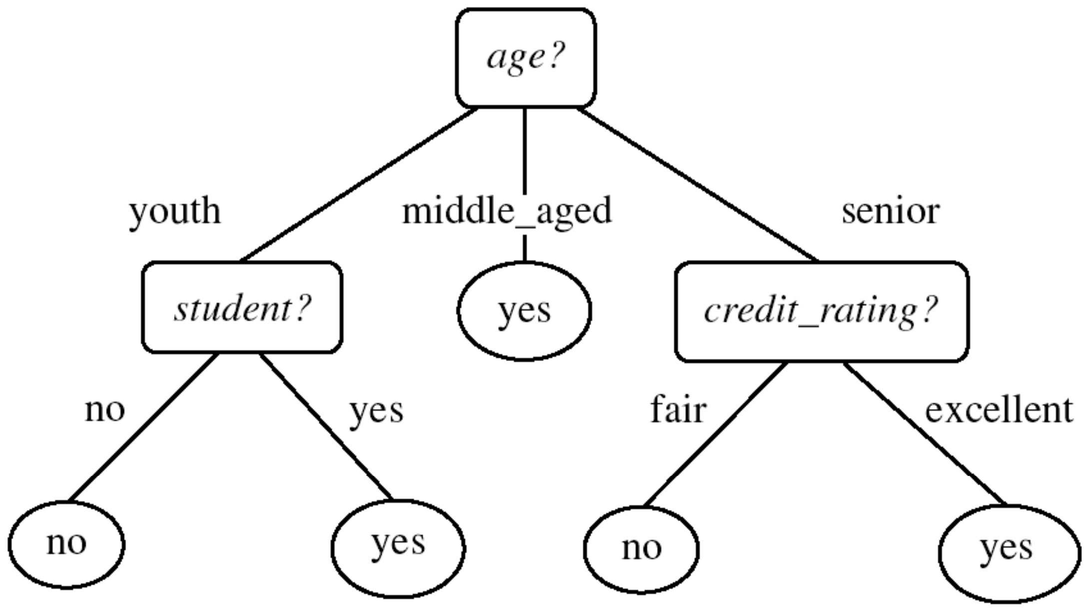
- Direct Method: extract rules directly from data
  - 1R Algorithm
  - Sequential covering algorithms
    - e.g.: PRISM, RIPPER, CN2, FOIL, and AQ

- Indirect Method: extract rules from other classification models
  - Decision trees
    - e.g: C4.5rules

Rule Extraction from a Decision Tree

- Rules are easier to understand than large trees
- One rule is created for each path from the root to a leaf
- Each attribute-value pair along a path forms a precondition: the leaf holds the class prediction
- The order of the rules does not matter
- Rules are
  - **Mutually exclusive**: no two rules will be satisfied for the same instance
  - **Exhaustive**: there is one rule for each possible attribute-value combination

Example: *AllElectronics*



$R1$: IF $age = youth$ AND $student = no$ THEN $buys\_computer = no$

$R2$: IF $age = youth$ AND $student = yes$ THEN $buys\_computer = yes$

$R3$: IF $age = middle\_aged$ THEN $buys\_computer = yes$

$R4$: IF $age = senior$ AND $credit\_rating = excellent$ THEN $buys\_computer = yes$

$R5$: IF $age = senior$ AND $credit\_rating = fair$ THEN $buys\_computer = no$

Pruning the Rule Set

- The resulting set of rules extracted can be large and difficult to follow
  - Solution: pruning the rule set
- For a given rule antecedent, any condition that does not improve the estimated accuracy of the rule can be pruned (i.e., removed)
- C4.5 extracts rules from an unpruned tree, and then prunes the rules using an approach similar to its tree pruning method

1R Algorithm

1R algorithm

- An easy way to find very simple classification rule
- 1R: rules that test one particular attribute
- Basic version
  - One branch for each value
  - 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 (*assumes nominal attributes*)
- "Missing" is treated as a separate attribute value

Pseudocode or 1R Algorithm

```
For each attribute,
  For each value of that 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.
```

Example: The weather problem

Outlook	Temperature	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

Evaluating the weather attributes

	Attribute	Rules	Errors	Total errors
1	outlook	sunny → no	2/5	4/14
		overcast → yes	0/4	
		rainy → yes	2/5	
2	temperature	hot → no*	2/4	5/14
		mild → yes	2/6	
		cool → yes	1/4	
3	humidity	high → no	3/7	4/14
		normal → yes	1/7	
4	windy	false → yes	2/8	5/14
		true → no*	3/6	

The attribute with the smallest number of errors

	Attribute	Rules	Errors	Total errors
1	outlook	sunny → no	2/5	4/14
		overcast → yes	0/4	
		rainy → yes	2/5	
2	temperature	hot → no*	2/4	5/14
		mild → yes	2/6	
		cool → yes	1/4	
3	humidity	high → no	3/7	4/14
		normal → yes	1/7	
4	windy	false → yes	2/8	5/14
		true → no*	3/6	

Dealing with numeric attributes

- Discretize numeric attributes
- Divide each attribute's range into intervals
  - Sort instances according to attribute's values
  - Place breakpoints where class changes (majority class)
  - This minimizes the total error

Weather data with some numeric attributes

Outlook	Temperature	Humidity	Windy	Play
sunny	85	85	false	no
sunny	80	90	true	no
overcast	83	86	false	yes
rainy	70	96	false	yes
rainy	68	80	false	yes
rainy	65	70	true	no
overcast	64	65	true	yes
sunny	72	95	false	no
sunny	69	70	false	yes
rainy	75	80	false	yes
sunny	75	70	true	yes
overcast	72	90	true	yes
overcast	81	75	false	yes
rainy	71	91	true	no

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

- Discretization involves partitioning this sequence by placing breakpoints wherever the class changes,

yes | no | yes yes yes | no no | yes yes yes | no | yes yes | no

The problem of overfitting

- Overfitting is likely to occur whenever an attribute has a large number of possible values
- This procedure is very sensitive to noise
  - One instance with an incorrect class label will probably produce a separate interval
- Attribute will have zero errors
- Simple solution: *enforce minimum number of instances in majority class per interval*

Minimum is set at 3 for temperature attribute

- The partitioning process begins

```
yes no yes yes | yes . . .
```

- the next example is also *yes,* we lose nothing by including that in the first partition

```
yes no yes yes yes | no no yes yes yes | no yes yes no
```

- Thus the final discretization is

```
yes no yes yes yes no no yes yes yes | no yes yes no
```

- the rule set

```
temperature: ≤ 77.5 → yes
             > 77.5 → no
```

Resulting rule set with overfitting avoidance

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	

Sequential Covering Algorithms

Sequential Covering Algorithms

- Convert decision tree into a rule set
  - Straightforward, but rule set very complex
- 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)
- A ***sequential covering*** **algorithm**:
  - The rules are learned *sequentially* (one at a time)
  - Each rule for a given class will ideally cover many of the instances of that class (and hopefully none of the instances of other classes).
  - Each time a rule is learned, the instances covered by the rule are removed, and the process repeats on the remaining instances.

Sequential Covering Algorithms

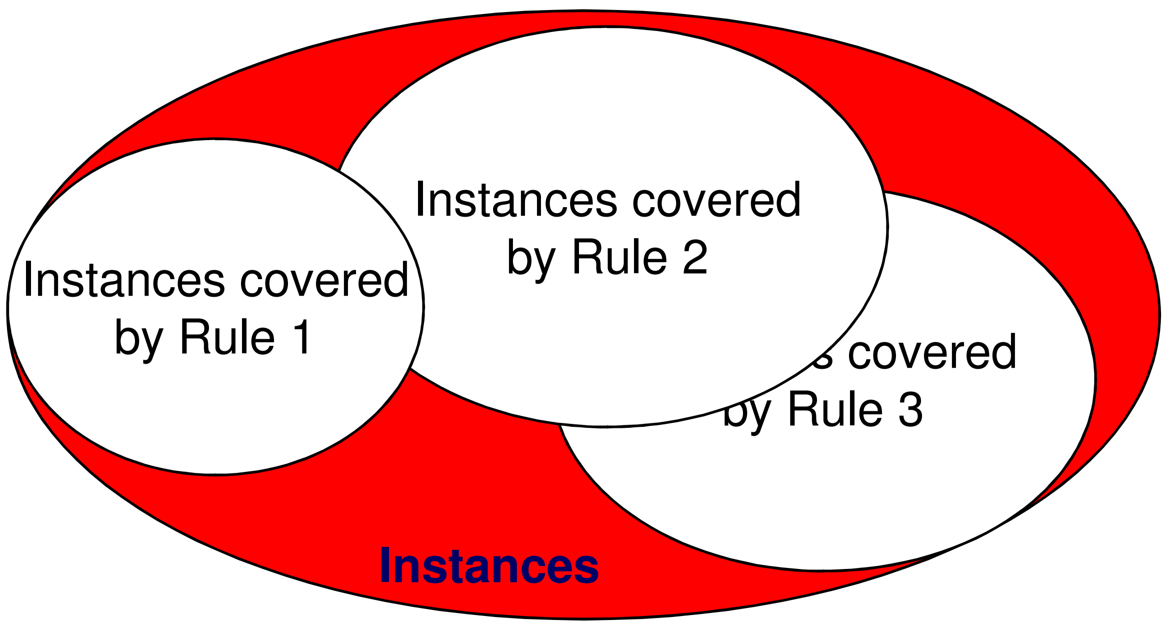
- Sequential covering algorithms are the most widely used approach to mining classification rules
- Typical algorithms: PRISM, FOIL, AQ, CN2, RIPPER
- Comparison with decision-tree induction: learning a set of rules *simultaneously*

Sequential Covering Algorithms

while (enough target instances left)

generate a rule

remove positive target instances satisfying this rule



An Alternative Approach

- A newer alternative approach, classification rules can be generated using **associative classification algorithms**
- They search for attribute-value pairs that occur frequently in the data.
- These pairs may form association rules, which can be analyzed and used in classification.

Basic Sequential Covering Algorithm

Algorithm: Sequential covering. Learn a set of IF-THEN rules for classification.

Input:

- *D*, a data set class-labeled tuples;
- *Att_vals*, the set of all attributes and their possible values.

Output: A set of IF-THEN rules.

Method:

(1) *Rule_set* = {}; // initial set of rules learned is empty
(2) **for each** class *c* **do**
(3)   **repeat**
(4)   Rule = **Learn_One_Rule**(*D*, *Att_vals*, *c*);
(5)   remove tuples covered by *Rule* from *D*;
(6)   **until** terminating condition;
(7)   *Rule_set* = *Rule_set* + *Rule*; // add new rule to rule set
(8) **endfor**
(9) return *Rule_Set*;

Basic Sequential Covering Algorithm

- Steps:
  - Rules are learned one at a time
  - Each time a rule is learned, the instances covered by the rules are removed
  - The process repeats on the remaining instances unless *termination condition*, e.g., when no more training examples or when the quality of a rule returned is below a user-specified level

Generating A Rule

- Typically, rules are grown in a **general-to-specific manner**
- We start with an empty rule and then gradually keep appending attribute tests to it.
- We append by adding the attribute test as a logical conjunct to the existing condition of the rule antecedent.

Example: Generating A Rule

- Suppose our training set, *D,* consists of loan application data.
- Attributes regarding each applicant include their:
  - age
  - income
  - education level
  - residence
  - credit rating
  - the term of the loan.
- The classifying attribute is *loan decision,* which indicates whether a loan is accepted (considered safe) or rejected (considered risky).

Example: Generating A Rule

- To learn a rule for the class “accept,” we start off with the most general rule possible, that is, the condition of the rule antecedent is empty. The rule is:

$$\text{IF} \quad \text{THEN } \mathit{loan\_decision} = \mathit{accept}.$$

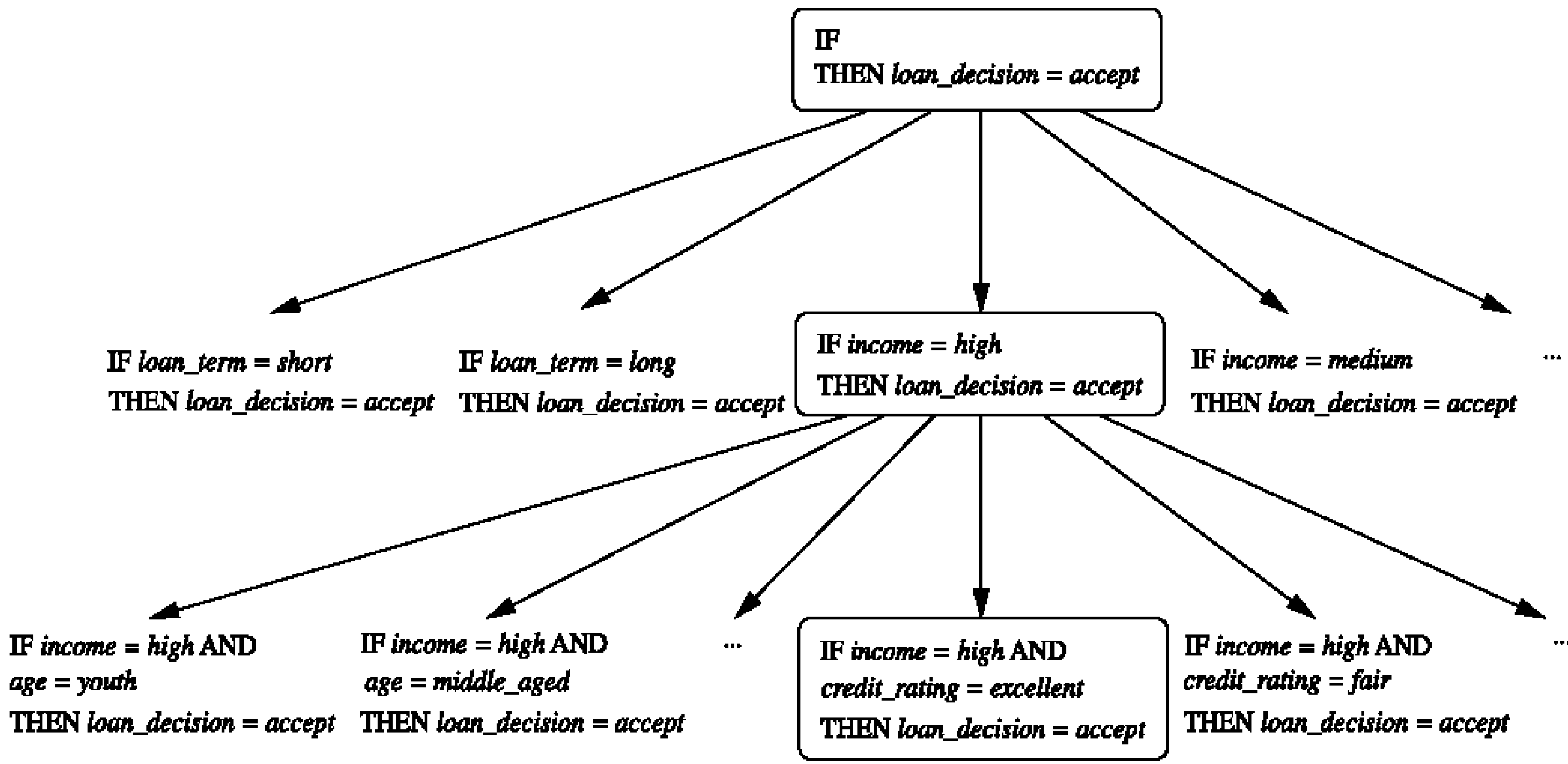
- We then consider each possible attribute test that may be added to the rule.

Example: Generating A Rule

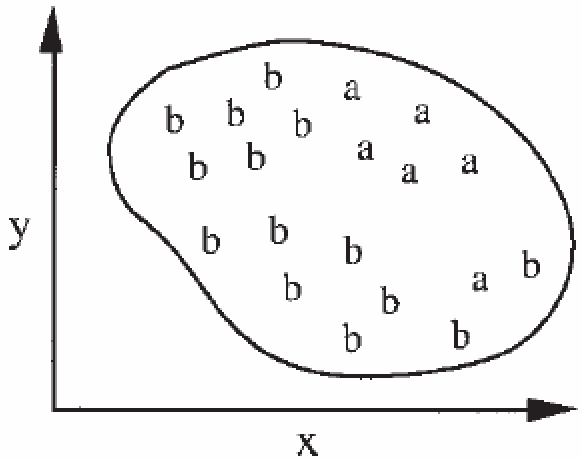
- Each time it is faced with adding a new attribute test (conjunct) to the current rule, it picks the one that most improves the rule quality, based on the training samples.
- The process repeats, where at each step, we continue to greedily grow rules until the resulting rule meets an acceptable quality level.

Example: Generating A Rule

- A general-to-specific search through rule space



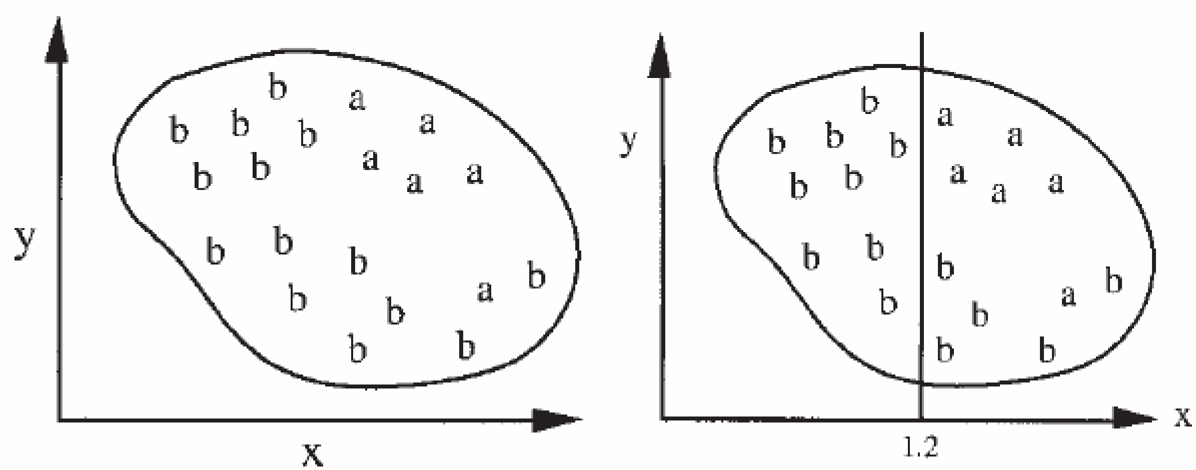
Example: Generating A Rule



- Possible rule set for class "a":

  if true then class = a

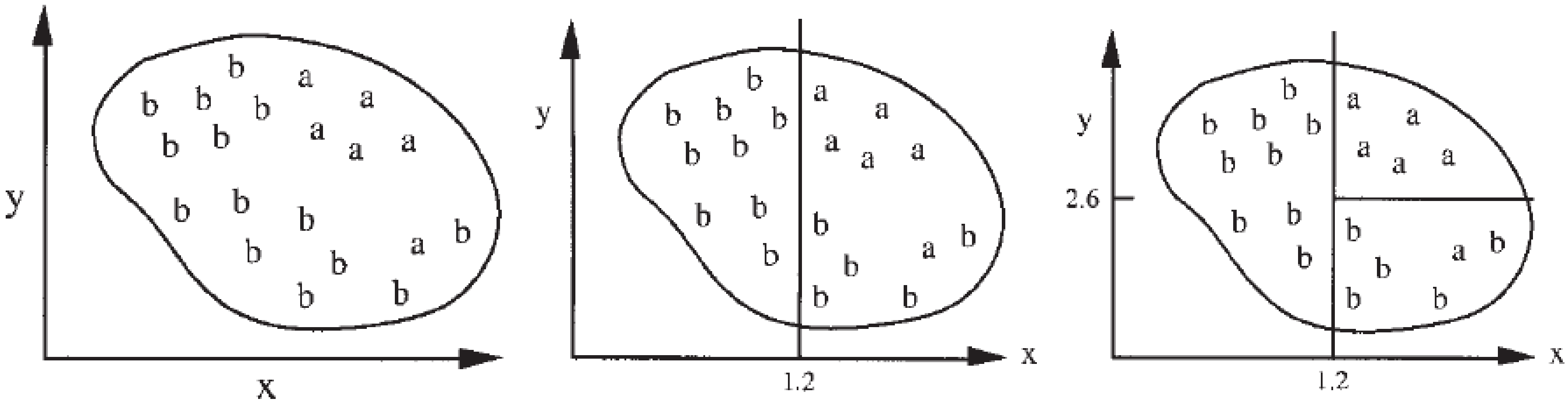
Example: Generating A Rule



- Possible rule set for class “a”:

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

Example: Generating A Rule

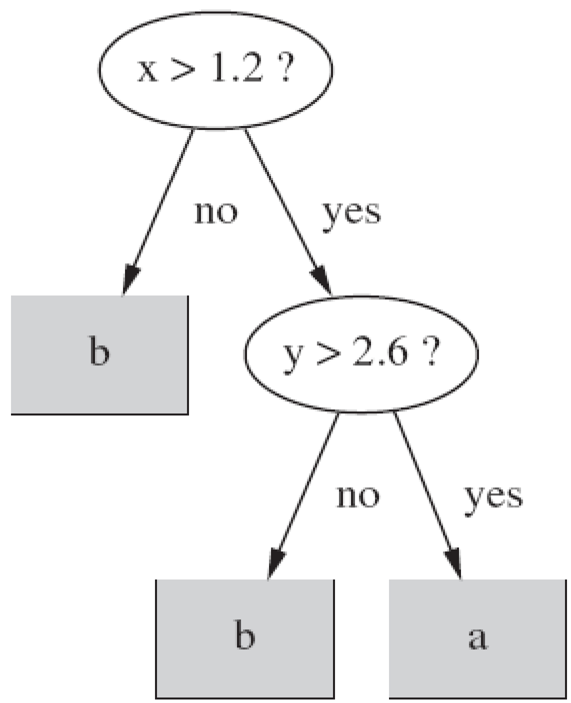


- Possible rule set for class “a”:

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

Decision tree for the same problem

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



Rules vs. trees

- Both methods might first split the dataset using the *x* attribute and would probably end up splitting it at the same place ($x = 1.2$)
- But: rule sets *can* be more clear 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

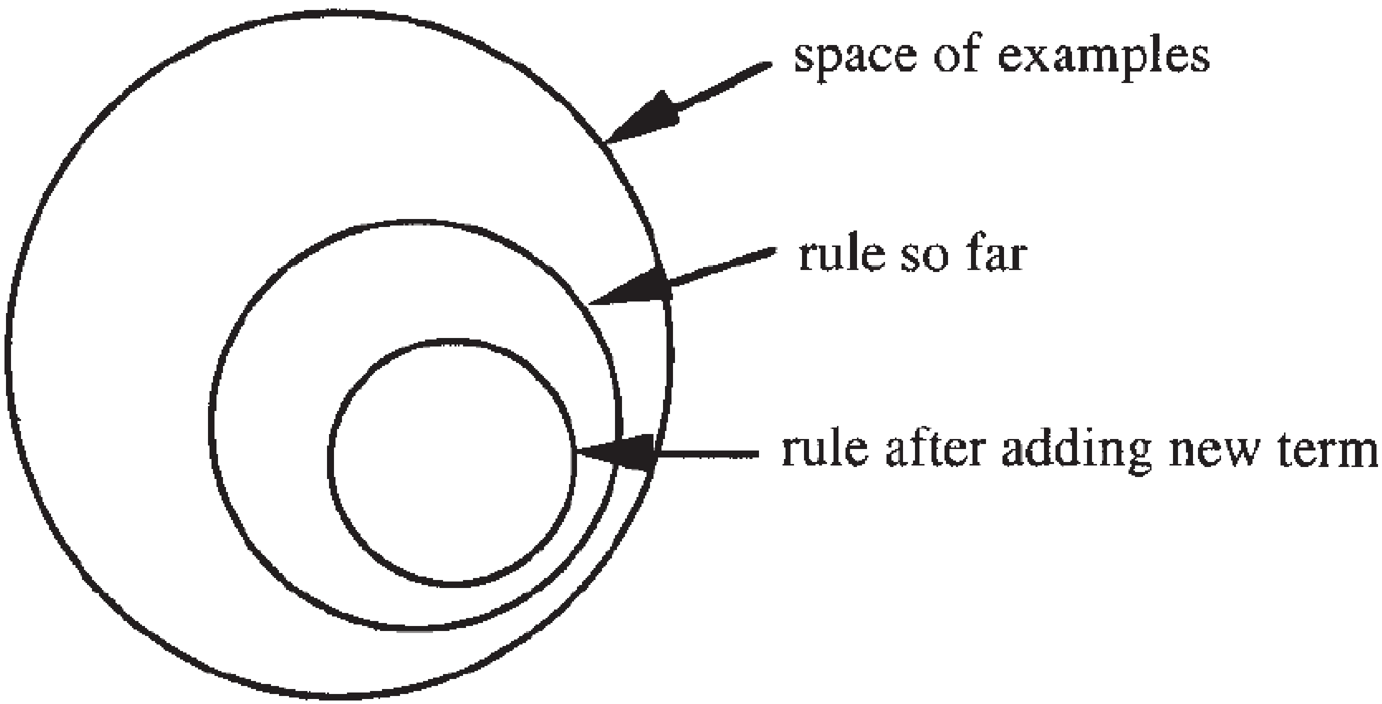
PRISM Algorithm

PRISM Algorithm

- **PRISM method** generates a rule by adding tests that maximize rule's accuracy
- Divide-and-conquer vs. covering algorithms
  - Divide-and-conquer algorithms choose an attribute to maximize the information gain
  - But: the covering algorithms choose an attribute–value pair to maximize the probability of the desired classification

PRISM Algorithm

- 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

Age	Spectacle prescription	Astigmatism	Tear production rate	Recommended lenses
young	myope	no	reduced	none
young	myope	no	normal	soft
young	myope	yes	reduced	none
young	myope	yes	normal	hard
young	hypermetrope	no	reduced	none
young	hypermetrope	no	normal	soft
young	hypermetrope	yes	reduced	none
young	hypermetrope	yes	normal	hard
pre-presbyopic	myope	no	reduced	none
pre-presbyopic	myope	no	normal	soft
pre-presbyopic	myope	yes	reduced	none
pre-presbyopic	myope	yes	normal	hard
pre-presbyopic	hypermetrope	no	reduced	none
pre-presbyopic	hypermetrope	no	normal	soft
pre-presbyopic	hypermetrope	yes	reduced	none
pre-presbyopic	hypermetrope	yes	normal	none
presbyopic	myope	no	reduced	none
presbyopic	myope	no	normal	none
presbyopic	myope	yes	reduced	none
presbyopic	myope	yes	normal	hard
presbyopic	hypermetrope	no	reduced	none
presbyopic	hypermetrope	no	normal	soft
presbyopic	hypermetrope	yes	reduced	none
presbyopic	hypermetrope	yes	normal	none

Example: contact lens data

- To begin, we seek a rule:

```
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

Create the rule

- Rule with best test added and covered instances:

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

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 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 LHS 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 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

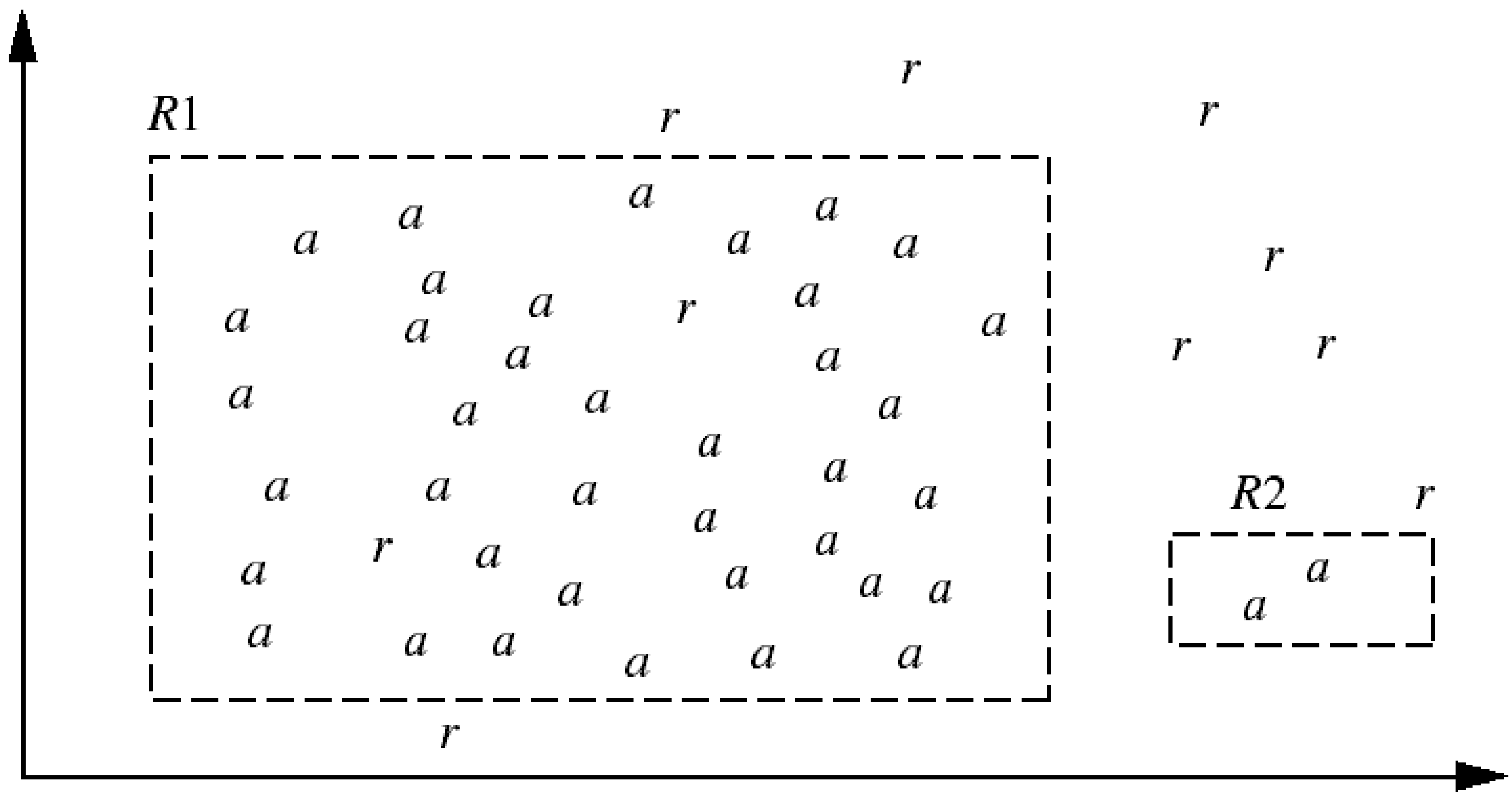
Separate and conquer

- 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

FOIL Algorithm

(First Order Inductive Learner Algorithm)

Coverage or Accuracy?



Coverage or Accuracy?

- Consider the two rules:
  - *R*1: correctly classifies 38 of the 40 instances it covers
  - *R2*: covers only two instances, which it correctly classifies
- Their accuracies are 95% and 100%
- *R*2 has greater accuracy than *R*1, but it is not the better rule because of its small coverage
- Accuracy on its own is not a reliable estimate of rule quality
- Coverage on its own is not useful either

Consider Both Coverage and Accuracy

- If our current rule is *R*:
  IF *condition* THEN *class* = *c*
- We want to see if logically ANDing a given attribute test to *condition* would result in a better rule
- We call the new condition, *condition'*, where *R'*:
  IF *condition'* THEN *class* = *c*
  - is our potential new rule
- In other words, we want to see if *R'* is any better than *R*

FOIL Information Gain

- FOIL_Gain (in FOIL & RIPPER): assesses info_gain by extending condition

$$FOIL\_Gain = pos' \times \left(\log_2 \frac{pos'}{pos' + neg'} - log_2 \frac{pos}{pos + neg}\right)$$

- where
  - *pos* (*neg*) be the number of positive (negative) instances covered by *R*
  - *pos'* (*neg'*) be the number of positive (negative) instances covered by *R'*
- It favors rules that have high accuracy and cover many positive instances

Rule Generation

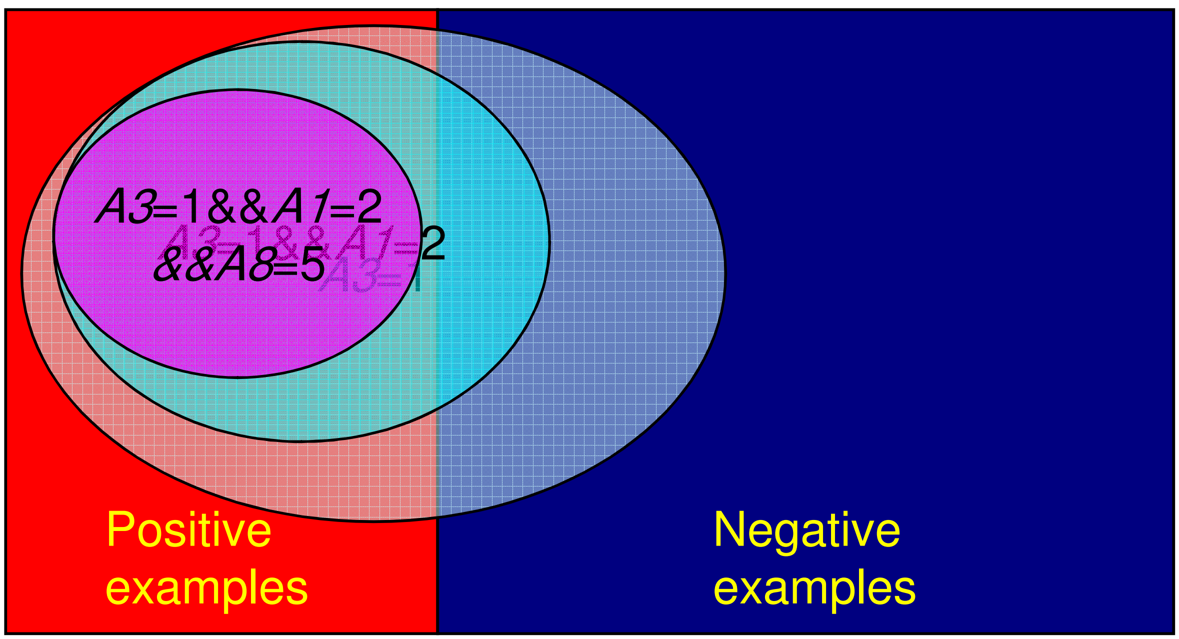
- To generate a rule

  while(true)

  find the best predicate *p*

  if foil-gain(*p*) > threshold **then** add *p* to current rule

  else break



Rule Pruning: FOIL method

- Assessments of rule quality as described above are made with instances from the training data
- Rule pruning based on an independent set of test instances

$$FOIL\_Prune(R) = \frac{pos - neg}{pos + neg}$$

- We calculate *FOIL_Prune* for *FOIL_Prune*
- If *FOIL_Prune* is higher for the pruned version of R, prune R

References

References

- J. Han, M. Kamber, **Data Mining: Concepts and Techniques**, Elsevier Inc. (2006). (Chapter 6)

- I. H. Witten and E. Frank, **Data Mining: Practical Machine Learning Tools and Techniques**, 2nd Edition, Elsevier Inc., 2005. (Chapter 6)

The end