

Aims

09s1: COMP9417 Machine Learning and Data Mining

Rule Learning

March 25, 2009

Acknowledgement: Material derived from slides for the book
Machine Learning, Tom M. Mitchell, McGraw-Hill, 1997
<http://www-2.cs.cmu.edu/~tom/mlbook.html>

and the book Data Mining, Ian H. Witten and Eibe Frank,
Morgan Kaufman, 2000. <http://www.cs.waikato.ac.nz/ml/weka>

This lecture will enable you to describe machine learning approaches to the problem of discovering rules from data. Following it you should be able to:

- define a representation for rules
- describe the decision table approach
- define association rules
- reproduce the basic association rule discovery algorithm
- reproduce the basic sequential covering algorithm

Relevant WEKA programs:

OneR, ZeroR, DecisionTable, PART, Prism, Apriori, JRip, Ridor

COMP9417: March 25, 2009

Rule Learning: Slide 1

Introduction

Machine Learning *specialists* often prefer certain models of data

- decision-trees
- neural networks
- nearest-neighbour
- ...

Potential Machine Learning *users* often prefer certain models of data

- spreadsheets
- 2D-plots
- OLAP
- ...

COMP9417: March 25, 2009

Rule Learning: Slide 2

Introduction

In applications of machine learning, specialists may find that users:

- find it hard to understand what some representations for models mean
- expect to see in models similar types of “patterns” to those they can find using manual methods
- have other ideas about kinds of representations for models they think would help them

Message: very simple models may be useful at first to help users *understand* what is going on in the data. Later, can use representations for models which may allow for greater *predictive accuracy*.

COMP9417: March 25, 2009

Rule Learning: Slide 3

Data set for *Weather*

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

Decision Tables

Simple representation for model is to use same format as input - a *decision table*.

Just look up the attribute values of an instance in the table to find the class value.

This is *rote learning* or *memorization* - no generalization !

However, by selecting a subset of the attributes we can compress the table and classify new instances.

Decision table:

1. a **schema**, set of attributes
2. a **body**, multiset of labelled instances, each has value for each attribute and for label

A multiset is a “set” which can have repeated elements.

Learning Decision Tables

Best-first search for schema giving decision table with least error.

1. $i := 0$
2. attribute set $A_i := A$
3. schema $S_i := \emptyset$
4. **Do**
 - Find the best attribute $a \in A_i$ to add to S_i by minimising cross-validation estimation of error E_i
 - $A_i := A_i \setminus \{a\}$
 - $S_i := S_i \cup \{a\}$
 - $i := i + 1$
5. **While** E_i is reducing

LOOCV

“Leave-one-out cross-validation”.

Given a data set, we often wish to estimate the error on new data of a model learned from this data set.

What can we do ?

We can use a *holdout* set, a subset of the data set which is NOT used for training but is used in testing our model.

Often use a 2:1 split of training:test data.

BUT this means only $\frac{2}{3}$ of the data set is available to learn our model ...

So in LOOCV, for n examples, we repeatedly leave 1 out and train on the remaining $n - 1$ examples.

Doing this n times, the mean error of all the train-and-test iterations is our estimate of the “true error” of our model.

k-fold Cross-Validation

A problem with LOOCV - have to learn a model n times for n examples in our data set.

Is this really necessary ?

Partition data set into k equal size disjoint subsets.

Each of these k subsets in turn is used as the test set while the remainder are used as the training set.

The mean error of all the train-and-test iterations is our estimate of the "true error" of our model.

$k = 10$ is a reasonable choice (or $k = 3$ if the learning takes a long time).

Ensuring the class distribution in each subset is the same as that of the complete data set is called *stratification*.

We'll see cross-validation again ...

Decision Table for *play*

Best first search for feature set,
terminated after 5 non improving subsets.
Evaluation (for feature selection): CV (leave one out)

Rules:

```
=====
outlook  humidity play
=====
sunny    normal  yes
overcast normal  yes
rainy    normal  yes
rainy    high   yes
overcast high   yes
sunny    high   no
=====
```

Decision Table for *play*

Unfortunately, not particularly good at predicting *play* ...

=== Stratified cross-validation ===

Correctly Classified Instances	6	42.8571 %
Incorrectly Classified Instances	8	57.1429 %

However, on a number of real-world domains *has* been shown to give predictive accuracy competitive with C4.5 decision-tree learner *and* uses a simpler model representation.

Representing Rules

General form of a rule:

Antecedent \rightarrow Consequent

- *Antecedent* (pre-condition) is a series of tests or constraints on attributes (like the tests at decision tree nodes)
- *Consequent* (post-condition or conclusion) gives class value or probability distribution on class values (like leaf nodes of a decision tree)
- Rules of this form (with a single conclusion) are classification rules
- Antecedent is true if logical conjunction of constraints is true
- Rule "fires" and gives the class in the consequent

Also has a procedural interpretation: If antecedent Then consequent

Sets of Rules

Rule1 ∨
Rule2 ∨
...

Think of set of rules as a logical disjunction.

A problem: can give rise to conflicts:

Rule1: $\text{att1}=\text{red} \wedge \text{att2}=\text{circle} \rightarrow \text{yes}$

Rule2: $\text{att2}=\text{circle} \wedge \text{att3}=\text{heavy} \rightarrow \text{no}$

Instance $\langle \text{red}, \text{circle}, \text{heavy} \rangle$ classified as both yes and no !

Either give no conclusion, or conclusion of rule with highest coverage.

Another problem: some instances may not be covered by rules:

Either give no conclusion, or majority class of training set.

Rules vs. Trees

Can solve both problems on previous slide by using ordered rules with a default class, e.g. *decision list*.

If Then Else If Then ...

However, essentially back to trees (which don't suffer from these problems due to fixed order of execution)

So why not just use trees ?

Rules can be modular (independent "nuggets" of information) whereas trees are not (easily) made of independent components.

Rules can be more compact than trees – see lecture on "Decision Tree Learning".

Rules vs. Trees

How would you represent these rules as a tree if each attribute w , x , y and z can have values 1, 2 or 3?

If $x = 1$ and $y = 1$ Then class = a

If $z = 1$ and $w = 1$ Then class = a

Otherwise class = b

1R

A simple rule-learner which has nonetheless proved very competitive in some domains.

Called *1R* for "1-rule", it is a one-level decision-tree expressed as a set of rules that test **one** attribute.

For each attribute a

For each value v of a make a rule:

count how often each class appears

find most frequent class c

set rule to assign class c for attribute-value $a = v$

Calculate error rate of rules for a

Choose set of rules with lowest error rate

1R on play

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	

1R on play

Two rules tie with the smallest number of errors, the first one is:

```
outlook:
  sunny   -> no
  overcast -> yes
  rainy   -> yes
(10/14 instances correct)
```

1R on play

More complicated with missing or numeric attributes:

- treat “missing” as a separate value
- *discretize* numeric attributes by choosing breakpoints for threshold tests

However, too many breakpoints causes overfitting, so parameter to specify minimum number of examples lying between two thresholds.

```
humidity:
  < 82.5  -> yes
  < 95.5  -> no
  >= 95.5 -> yes
(11/14 instances correct)
```

ZeroR

What is this ?

Simply the 1R method but testing *zero* attributes instead of one.

What does it do ?

Predicts majority class in training set (mean if numerical prediction).

What is the point ?

Use a baseline for comparing classifier performance.

Stop and think about it ...

... it is a *most-general* classifier, having no constraints on attributes. Usually, it will be too general (e.g. “always play”). So we could try 1R, which is less general (more specific) ...

What does this process of moving from ZeroR to 1R resemble ?

Data Mining for Associations

Consider supermarket where database records items bought by customer as a transaction.

Managers interested in finding *associations* between sets of items in all customer transactions, e.g.

“90% of all transactions that purchase bread and butter also purchase milk”.

This can be express as the rule:

$\text{purchase}(\text{bread}) \text{ and } \text{purchase}(\text{butter}) \rightarrow \text{purchase}(\text{milk})$

The antecedent is the conjunction of $\{\text{purchase}(\text{bread}), \text{purchase}(\text{butter})\}$ and the consequent is $\text{purchase}(\text{milk})$.

90% is the confidence factor of the rule.

Usually, interested in discovering or “mining” sets of rules from the data which satisfy some initial specifications.

More Associations

- Find all rules that have “Diet Coke” as a consequent
- Find all rules that have “bagels” in the antecedent
- Find all rules that have “sausage” in the antecedent and “mustard” in the consequent
- Find all rules relating items located on aisles 9 and 10
- Find “best” k rules that have “bagels” in the consequent

Note that “best” can be in terms of confidence or support of rules.

Supervised vs. Unsupervised Learning

Supervised learning pre-defined class

- learn to predict class (a classifier)
- concept learning, decision trees

Unsupervised learning no pre-defined class

- find attributes which can be grouped together in characteristic “patterns”
- clustering, association rules

Association Rules

Like classification rules except

- no pre-defined class, any attribute can appear in the consequent
- consequent can contain ≥ 1 attributes

Can generate *very* many association rules . . .

However, we can use restrictions on the coverage and accuracy of rules, plus syntactic restrictions, to dramatically reduce the number of “interesting” and “potentially useful” rules generated.

Association Rules

Suppose we are given a set of items \mathcal{I} , such as the set of items for sale at a retail outlet, the set of attribute-value pairs used to describe a dataset, etc.

Definition 1. Itemset. An itemset $I \subseteq \mathcal{I}$ is some subset of items.

Definition 2. Transaction. A transaction is a pair $T = (tid, I)$, where tid is the transaction identifier and I is an itemset.

A transaction $T = (tid, I)$ is said to *support* an itemset X if $X \subseteq I$.

Definition 3. Transaction database. A transaction database \mathcal{D} is a set of transactions such that each transaction has unique identifier.

Definition 4. Cover. The cover of an itemset X in \mathcal{D} consists of the set of transaction identifiers of transactions in \mathcal{D} that support X :

$$\text{cover}(X, \mathcal{D}) := \{tid \mid (tid, I) \in \mathcal{D}, X \subseteq I\}$$

Association Rules

Definition 6. Confidence. The confidence of a rule of the form $X \rightarrow Y$ in \mathcal{D} is the proportion of transactions in the cover of X in \mathcal{D} which are also in the cover of $X \wedge Y$ in \mathcal{D} :

$$\text{confidence}(X \rightarrow Y, \mathcal{D}) := \frac{|\text{cover}(X \wedge Y, \mathcal{D})|}{|\text{cover}(X, \mathcal{D})|}$$

We can relate these terms from database mining to the machine learning terms *coverage* and *accuracy* as follows:

- support (coverage) number of transactions (instances) for which rule predicts correctly
- confidence (accuracy) number of transactions (instances) predicted correctly as a proportion of all instances rule applies to

Association Rules

Definition 5. Support. The support of an itemset X in \mathcal{D} is the number of transactions in the cover of X in \mathcal{D} :

$$\text{support}(X, \mathcal{D}) := |\text{cover}(X, \mathcal{D})|$$

An itemset is called *frequent* (or *large*) in \mathcal{D} if its support in \mathcal{D} exceeds some minimum support threshold.

Note that support is often defined as a *fraction*, namely the ratio of transactions in the cover of X to the total number of transactions in \mathcal{D} . See:

Agrawal, R., Imielinski, T. and Swami, A. (1993)
"Mining association rules between sets of items in large databases".
Proc. ACM SIGMOD Conference 1993, pp 26–28.

Either can be used, usually clear from the context.

Item sets

Back to the supermarket for a moment ...

... in *market basket analysis* each transaction (instance) contains the set of *items* purchased by the customer.

An item is therefore a (Boolean) attribute, true if the customer bought the article in the transaction, false otherwise.

Now think of the number of possible items you could buy in your local supermarket ...

... then think of the number of possible *item sets* we can form from those items ...

... and then think of the number of possible *association rules* we can form between items in each item set ...

Item sets with minimum support 2 for *play*

one-item sets	two-item sets	three-item sets	four-item sets
outl = sunny (5)	outl = sunny temp = mild (2)	outl = sunny temp = hot humi = high (2)	outl = sunny temp = hot humi = high play = no (2)
...
temp = mild (6)	outl = sunny wind = true (2)	outl = sunny humi = high play = no (3)	outl = rainy humi = normal wind = false play = yes (2)
...
	humi = high wind = true (3)		

COMP9417: March 25, 2009

Rule Learning: Slide 28

From item sets to association rules

- individual items (attribute-value constraints) form one-item sets
- two-item sets formed from pairs of one-item sets (except pairs of same attribute with different values), and so on ...
- all item sets have a support (coverage) on the data set
- any item sets below *minimum support* are discarded, leaving so-called "large" item sets
- generate association rules with a certain *minimum confidence* from each item set

COMP9417: March 25, 2009

Rule Learning: Slide 29

From item sets to association rules

For example, a 3-item set with support 4:

humidity = normal, windy = false, play = yes

leads to seven potential rules:

Rule	Conf
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
Then humidity = normal and windy = false and play = yes	4/14

COMP9417: March 25, 2009

Rule Learning: Slide 30

Generating association rules efficiently

Two stage algorithm:

- generate item sets with specified minimum support (coverage)
- generate association rules with specified minimum confidence (accuracy) from each item set

COMP9417: March 25, 2009

Rule Learning: Slide 31

Algorithm schema for generating item sets

Key idea: a k -item set can only have minimum support if all of its $k-1$ -item subsets have minimum support (are "large").

- First, generate all 1-item sets by making a pass through the data set and storing all those above min. support in a hash table
- Next, generate all 2-item sets from all pairs of 1-item sets from the hash table
- Then pass through data set counting the support of the 2-item sets, discarding those below min. support
- The same process is iterated to generate k -item sets from $k-1$ -item sets, until no more large item sets are found

Example: generating item sets

Suppose we have five three-item sets: (ABC) , (ABD) , (ACD) , (ACE) and (BCD) where A is a feature like `outlook = sunny`.

Let $(ABCD)$ be the union of the first two three-item sets.

It is a *candidate* four-item set since all of its three-item subsets are large. What are they ?

(ABC) , (ABD) , (ACD) and (BCD) .

Assuming the three-item sets are sorted in lexical order, we only have to consider pairs with the same two first items; otherwise the resulting item set would have more than four items !

These are (ABC) and (ABD) plus (ACD) and (ACE) , and we can forget about (BCD) .

The second pair leads to $(ACDE)$.

Example: generating item sets

BUT all of the three-item subsets of $(ACDE)$ are not large ...

What are they ?

(ACD) and (ACE) are large, but (ADE) and CDE are not large.

Therefore, we only have one candidate four-item set, $(ABCD)$, which must be checked for *actual* minimum support on the data set.

From item-sets to association rules

Key idea: say we have a rule

$$A \wedge B \rightarrow C \wedge D$$

with greater than minimum support and minimum confidence c_1 .

This is a *double-consequent* rule.

Now consider the two *single-consequent* rules

$$A \wedge B \wedge C \rightarrow D$$

$$A \wedge B \wedge D \rightarrow C$$

These must also hold with greater than minimum support and confidence.

From item-sets to association rules

To see why, first consider support: this must be greater than minimum support since it is the same for each single-consequent as for the double-consequent rule.

Let the rule $A \wedge B \wedge C \rightarrow D$ have confidence c_2 .

This confidence $c_2 \geq c_1$, because

- the denominator (coverage) for c_1 must be \geq that for c_2 , since
- the antecedent of the double-consequent rule has fewer conditions than the single-consequent rule, therefore
- must cover more instances than the antecedent of the second rule.

What is another way of explaining this ?

Antecedent $A \wedge B$ is **more general than** antecedent $A \wedge B \wedge C$.

From item-sets to association rules

Now consider the converse, for a given item-set:

If any of the single-consequent rules which can form a double-consequent rule **do not** have better than minimum confidence, then do not consider the double-consequent rule, since it **cannot** have better than minimum confidence.

This gives a basis for an efficient algorithm for constructing rules by building up from candidate single-consequent rules to candidate double-consequent rules, etc.

Algorithm schema for generating association rules

- First, generate confidences for all single-consequent rules
- Discard any below minimum confidence
- Build up candidate double-consequent rules from retained single-consequent rules
- Iterate as for construction of large item sets

Apriori in Weka

Apriori
=====

Minimum support: 0.05
Minimum metric <confidence>: 0.9
Number of cycles performed: 17

Generated sets of large itemsets:

Size of set of large itemsets L(1): 12

Size of set of large itemsets L(2): 47

Size of set of large itemsets L(3): 39

Size of set of large itemsets L(4): 6

Best rules found:

1. humidity=normal windy=FALSE 4 ==> play=yes 4 conf:(1)
2. temperature=cool 4 ==> humidity=normal 4 conf:(1)
3. outlook=overcast 4 ==> play=yes 4 conf:(1)
4. temperature=cool play=yes 3 ==> humidity=normal 3 conf:(1)
5. outlook=rainy windy=FALSE 3 ==> play=yes 3 conf:(1)
6. outlook=rainy play=yes 3 ==> windy=FALSE 3 conf:(1)
7. outlook=sunny humidity=high 3 ==> play=no 3 conf:(1)
8. outlook=sunny play=no 3 ==> humidity=high 3 conf:(1)
9. temperature=cool windy=FALSE 2 ==> humidity=normal play=yes 2
conf:(1)
10. temperature=cool humidity=normal windy=FALSE 2 ==> play=yes 2
conf:(1)

Interpretation is not obvious:

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

is *not* the same as:

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

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

However, it means that the following also holds:

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

Association rules summary

- the main contribution from *database mining*
- concerned with *scalability*
- simple basic algorithm
- generalisation lattice
- many extensions
- numerical, temporal, spatial, negation, closed & free itemsets, . . .
- sparse instance representation
- rule “interestingness” measures

Learning Disjunctive Sets of Rules

Method 1: Learn decision tree, convert to rules

- can be slow for large and noisy datasets
- improvements: e.g. C5.0, Weka PART

Method 2: Sequential covering algorithm:

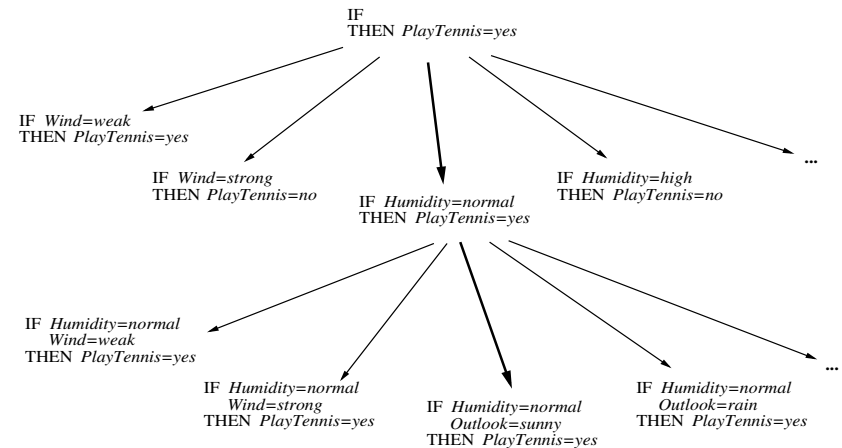
1. *Learn one rule* with high accuracy, any coverage
2. Remove positive examples covered by this rule
3. Repeat

Sequential Covering Algorithm

SEQUENTIAL-COVERING(*Target_attribute*, *Attributes*, *Examples*, *Threshold*)

- $Learned_rules \leftarrow \{\}$
- $Rule \leftarrow \text{LEARN-ONE-RULE}(Target_attribute, Attributes, Examples)$
- while PERFORMANCE(*Rule*, *Examples*) > *Threshold*, do
 - $Learned_rules \leftarrow Learned_rules + Rule$
 - $Examples \leftarrow Examples - \{\text{examples correctly classified by } Rule\}$
 - $Rule \leftarrow \text{LEARN-ONE-RULE}(Target_attribute, Attributes, Examples)$
- $Learned_rules \leftarrow \text{sort } Learned_rules \text{ accord to PERFORMANCE over } Examples$
- return *Learned_rules*

Learn One Rule



Algorithm “Learn One Rule”

LEARN-ONE-RULE(*Target_attribute*, *Attributes*, *Examples*)

// Returns a single rule which covers some of the
// positive examples and none of the negatives.

Pos := positive Examples
Neg := negative Examples
BestRule := $_ \rightarrow _$

if *Pos* **do**

NewAnte := most general rule antecedent possible
 NewRuleNeg := *Neg*

while *NewRuleNeg* **do**

for *ClassVal* in *Target_attribute_values* **do**
 NewCons := *Target_attribute* = *ClassVal*

Algorithm “Learn One Rule”

// Add a new literal to specialize *NewAnte*, i.e. possible
// constraints of the form *att* = *val* for *att* ∈ *Attributes*

Candidate_literals ← generate candidates

Best_literal ← $\text{argmax}_{L \in \text{Candidate_literals}}$

 Performance(*SpecializeAnte*(*NewAnte*, *L*) → *NewCons*)

add *Best_literal* to *NewAnte*

NewRule := *NewAnte* → *NewCons*

if Performance(*NewRule*) > Performance(*BestRule*) **then**

BestRule := *NewRule*

endif

NewRuleNeg := subset of *NewRuleNeg* that satisfies *NewAnte*

endfor

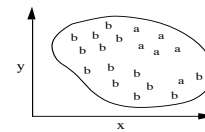
endif

return *BestRule*

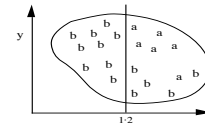
Learn One Rule

- Called a covering approach because at each stage a rule is identified that covers some of the instances
- the evaluation function $Performance(Rule)$ is unspecified
- a simple measure would be the number of negatives not covered by the antecedent, i.e. $Neg - NewRuleNeg$
- the consequent could then be the most frequent value of the target attribute among the examples covered by the antecedent
- this is sure not to be the best measure of performance !

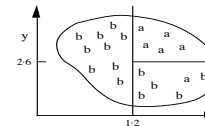
Example: generating a rule



If true then class = a



If $x > 1.2$ then class = a



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

Subtleties: Learn One Rule

1. May use *beam search*
2. Easily generalizes to multi-valued target functions
3. Choose evaluation function to guide search:
 - Entropy (i.e., information gain)
 - Sample accuracy:

$$\frac{n_c}{n}$$

where n_c = correct rule predictions, n = all predictions

- m estimate:

$$\frac{n_c + mp}{n + m}$$

Aspects of Sequential Covering Algorithms

- Sequential Covering learns rules singly. Decision Tree induction learns all disjuncts simultaneously.
- Sequential Covering chooses between *all att-val pairs* at each specialisation step (i.e. between subsets of the examples covered). Decision Tree induction only chooses between *all attributes* (i.e. between partitions of the examples w.r.t. the added attribute).
- Assuming final rule-set contains on average n rules with k conditions, sequential covering requires $n \times k$ primitive selection decisions. Choosing an attribute at the internal node of a decision tree equates to choosing att-val pairs for the conditions of all corresponding rules.
- If data is plentiful, then the greater flexibility for choosing att-val pairs might be desired and might lead to better performance.

Aspects of Sequential Covering Algorithms

- If a general-to-specific search is chosen, then start from a single node. If a specific-to-general search is chosen, then for a set of examples, need to determine what are the starting nodes.
- Depending on the number of conditions expected for rules relative to the number of conditions in the examples, most general rules may be closer to the target than most specific rules.
- General-to-specific sequential covering is a generate-and-test approach. All syntactically permitted specialisations are generated and tested against the data. Specific-to-general is typically example-driven, constraining the hypotheses generated.
- Variations on performance evaluation are often implemented: entropy, m -estimate, relative frequency, significance tests (e.g. likelihood ratio).

Rules with exceptions

Idea: allow rules to have exceptions

Example: rule for iris data

If $\text{petal-length} \geq 2.45$ and $\text{petal-length} < 4.45$ then Iris-versicolor

New instance:

Sepal length	Sepal width	Petal length	Petal width	Type
5.1	3.5	2.6	0.2	Iris-setosa

Modified rule:

If $\text{petal-length} \geq 2.45$ and $\text{petal-length} < 4.45$ then Iris-versicolor EXCEPT if $\text{petal-width} < 1.0$ then Iris-setosa

Exceptions to exceptions to exceptions ...

```
default: Iris-setosa
except if petal-length  $\geq$  2.45 and petal-length  $<$  5.355
  and petal-width  $<$  1.75
  then Iris-versicolor
    except if petal-length  $\geq$  4.95 and petal-width  $<$  1.55
      then Iris-virginica
        else if sepal-length  $<$  4.95 and sepal-width  $\geq$  2.45
          then Iris-virginica
    else if petal-length  $\geq$  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

Advantages of using exceptions

Default...except if...then...

is logically equivalent to

if...then...else

where the else specifies the default.

But: exceptions offer a psychological advantage

- Assumption: defaults and tests early on apply more widely than exceptions further down
- Exceptions reflect special cases

Induct-RDR

Gaines & Compton (1995)

- Learns “Ripple-Down Rules from examples
- INDUCT s significance measure for a rule:
 - Probability of completely random rule with same coverage performing at least as well
- Random rule R selects t cases at random from the data set
- How likely is it that p of these belong to the correct class ?
- Probability given by *hypergeometric* distribution
- approximate by incomplete beta function
- works well if target function suits rules-with-exceptions *bias*

Variants of Rule Learning Programs

- *Sequential* or *simultaneous* covering of data?
- General \rightarrow specific, or specific \rightarrow general?
- Generate-and-test, or example-driven?
- Whether and how to post-prune?
- What statistical evaluation function?