15s1: COMP9417 Machine Learning and Data Mining

Classification Rule Learning

April 1, 2015

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 Kauffman, 2000. `http://www.cs.waikato.ac.nz/ml/weka`

Aims

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 and 1R approaches
- outline overfitting avoidance in rule learning using pruning
- reproduce the basic sequential covering algorithm

Relevant WEKA programs:
OneR, ZeroR, DecisionTable, DecisionStump, PART, Prism, JRip, Ridor

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
- ...

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*.

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 → 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 $\vee$
Rule2 $\vee$
. . .

Think of set of rules as a logical disjunction.

A problem: can give rise to conflicts:

Rule1: att1=red $\wedge$ att2= circle → yes

Rule2: att2=circle $\wedge$ att3= heavy → no

Instance $\langle red, circle, heavy \rangle$ 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* or OneR for "1-rule", it is a one-level decision-tree (*aka* DecisionStump) 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 ?

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

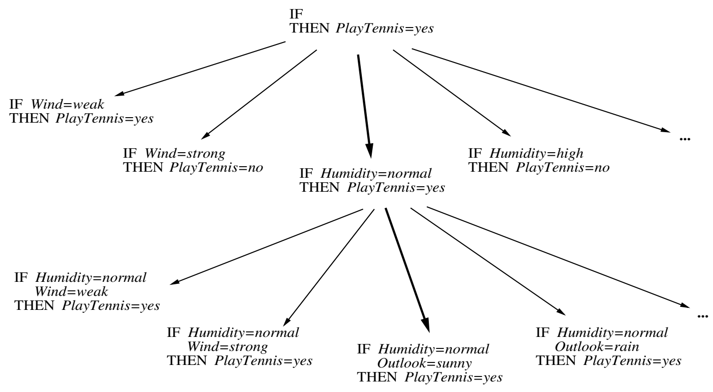
Method 2: Sequential covering algorithm:

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

Sequential Covering Algorithm

```
SEQUENTIAL-COVERING($Target\_attribute, Attributes, Examples, Threshold$)

• $Learned\_rules \leftarrow \{\}$
• $Rule \leftarrow$ LEARN-ONE-RULE($Target\_attribute, Attributes, Examples$)
• while PERFORMANCE($Rule, Examples$) $> Threshold$, do
    – $Learned\_rules \leftarrow Learned\_rules + Rule$
    – $Examples \leftarrow Examples -$ {examples correctly classified by $Rule$}
    – $Rule \leftarrow$ LEARN-ONE-RULE($Target\_attribute, Attributes, Examples$)
• $Learned\_rules \leftarrow$ sort $Learned\_rules$ accord to PERFORMANCE over
  $Examples$
• return $Learned\_rules$
```

Learn One Rule



Algorithm "Learn One Rule"

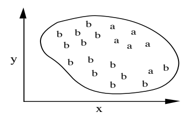
```
LEARN-ONE-RULE($Target\_attribute, Attributes, Examples$)

// Returns a single rule which covers some of the
// positive examples and none of the negatives.
$Pos$ := positive $Examples$
$Neg$ := negative $Examples$
$BestRule := \_ \to \_$
if $Pos$ do
  $NewAnte$ := most general rule antecedent possible
  $NewRuleNeg := Neg$
  while $NewRuleNeg$ do
    for $ClassVal$ in $Target\_attribute\_values$ do
      $NewCons := Target\_attribute = ClassVal$
```

```
      // Add a new literal to specialize NewAnte, i.e. possible
      // constraints of the form att = val for att ∈ Attributes
      $Candidate\_literals \leftarrow$ generate candidates
      $Best\_literal \leftarrow \text{argmax}_{L \in Candidate\_literals}$
        $Performance(SpecializeAnte(NewAnte, L) \to NewCons)$
      add $Best\_literal$ to $NewAnte$
      $NewRule := NewAnte \to NewCons$
      if $Performance(NewRule) > Performance(BestRule)$ then
        $BestRule := NewRule$
      endif
      $NewRuleNeg$ := subset of $NewRuleNeg$ that satisfies $NewAnte$
    endfor
endif
return $BestRule$
```

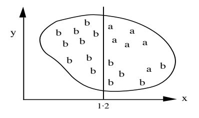
Learn One Rule

- 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 !

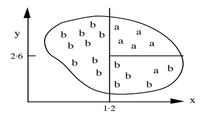
Example: generating a rule



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:
     $$\frac{n_c + mp}{n + m}$$
     – think of this as an approximation to a *Bayesian* evaluation function

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 petal-length` $\geq$ `2.45 and 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 petal-length` $\geq$ `2.45 and petal-length` $<$ `4.45 then Iris-versicolor EXCEPT`
`if 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

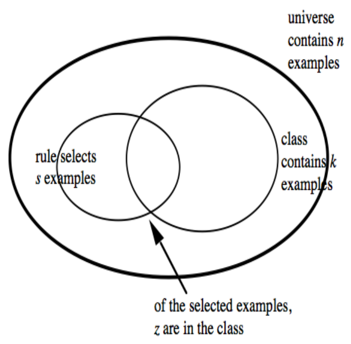
Induct-RDR

Gaines & Compton (1995)

- Learns "Ripple-Down Rules from examples
- INDUCT's significance measure for a rule:
  - Probability of completely random rule with same
- 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
  - see next slide
  - approximated by incomplete beta function
- works well if target function suits rules-with-exceptions *bias*

Induct-RDR

Hypergeometric test for rule induction
Witten & Gaines


universe contains n examples
rule selects s examples
class contains k examples
of the selected examples, z are in the class

p number of instances of that class that the rule selects;

t total number of instances that the rule selects;

p total number of instances of that class in the dataset;

t total number of instances in the dataset.

Issues for Classification 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?

Summary of Classification Rule Learning

- A major class of representations (AI, business rules, RuleML, . . .)
- Rule interpretation may need care
- Many common learning issues: search, evaluation, overfitting, etc.
- Can be related to numeric prediction by threshold functions
- Lifted to first-order representations in Inductive Logic Programming