Slides for “Data Mining”
by
I. H. Witten and E. Frank
Output: Knowledge representation

- Decision tables
- Decision trees
- Decision rules
- Association rules
- Rules with exceptions
- Rules involving relations
- Linear regression
- Trees for numeric prediction
- Instance-based representation
- Clusters
Output: representing structural patterns

- Many different ways of representing patterns
  - Decision trees, rules, instance-based, ...
- Also called “knowledge” representation
- Representation determines inference method
- Understanding the output is the key to understanding the underlying learning methods
- Different types of output for different learning problems (e.g. classification, regression, ...)
Decision tables

- Simplest way of representing output:
  - Use the same format as input!

- Decision table for the weather problem:

<table>
<thead>
<tr>
<th>Outlook</th>
<th>Humidity</th>
<th>Play</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sunny</td>
<td>High</td>
<td>No</td>
</tr>
<tr>
<td>Sunny</td>
<td>Normal</td>
<td>Yes</td>
</tr>
<tr>
<td>Overcast</td>
<td>High</td>
<td>Yes</td>
</tr>
<tr>
<td>Overcast</td>
<td>Normal</td>
<td>Yes</td>
</tr>
<tr>
<td>Rainy</td>
<td>High</td>
<td>No</td>
</tr>
<tr>
<td>Rainy</td>
<td>Normal</td>
<td>No</td>
</tr>
</tbody>
</table>

- Main problem: selecting the right attributes
Decision trees

- “Divide-and-conquer” approach produces tree
- Nodes involve testing a particular attribute
- Usually, attribute value is compared to constant
- Other possibilities:
  - Comparing values of two attributes
  - Using a function of one or more attributes
- Leaves assign classification, set of classifications, or probability distribution to instances
- Unknown instance is routed down the tree
Nominal and numeric attributes

- **Nominal:**
  - number of children usually equal to number values
  - attribute won’t get tested more than once
  - Other possibility: division into two subsets

- **Numeric:**
  - test whether value is greater or less than constant
  - attribute may get tested several times
  - Other possibility: three-way split (or multi-way split)
    - Integer: less than, equal to, greater than
    - Real: below, within, above
Missing values

- Does absence of value have some significance?
  - Yes $\Rightarrow$ “missing” is a separate value
  - No $\Rightarrow$ “missing” must be treated in a special way
    - Solution A: assign instance to most popular branch
    - Solution B: split instance into pieces
      - Pieces receive weight according to fraction of training instances that go down each branch
      - Classifications from leave nodes are combined using the weights that have percolated to them
Classification rules

- Popular alternative to decision trees
- *Antecedent* (pre-condition): a series of tests (just like the tests at the nodes of a decision tree)
- Tests are usually logically ANDed together (but may also be general logical expressions)
- *Consequent* (conclusion): classes, set of classes, or probability distribution assigned by rule
- Individual rules are often logically ORed together
  - Conflicts arise if different conclusions apply
From trees to rules

- Easy: converting a tree into a set of rules
  - One rule for each leaf:
    - Antecedent contains a condition for every node on the path from the root to the leaf
    - Consequent is class assigned by the leaf

- Produces rules that are unambiguous
  - Doesn’t matter in which order they are executed

- But: resulting rules are unnecessarily complex
  - Pruning to remove redundant tests/rules
From rules to trees

- More difficult: transforming a rule set into a tree
  - Tree cannot easily express disjunction between rules
- Example: rules which test different attributes
  - If a and b then x
  - If c and d then x

- Symmetry needs to be broken
- Corresponding tree contains identical subtrees (⇒ “replicated subtree problem”)

A tree for a simple disjunction
The exclusive-or problem

- If \( x = 1 \) and \( y = 0 \) then class = a
- If \( x = 0 \) and \( y = 1 \) then class = a
- If \( x = 0 \) and \( y = 0 \) then class = b
- If \( x = 1 \) and \( y = 1 \) then class = b
A tree with a replicated subtree

If $x = 1$ and $y = 1$
then class = a
If $z = 1$ and $w = 1$
then class = a
Otherwise class = b
“Nuggets” of knowledge

- Are rules independent pieces of knowledge? (It seems easy to add a rule to an existing rule base.)
- Problem: ignores how rules are executed
- 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
Interpreting rules

- What if two or more rules conflict?
  - Give no conclusion at all?
  - Go with rule that is most popular on training data?
  - ...

- What if no rule applies to a test instance?
  - Give no conclusion at all?
  - Go with class that is most frequent in training data?
  - ...
Special case: boolean class

- Assumption: if instance does not belong to class “yes”, it belongs to class “no”
- Trick: only learn rules for class “yes” and use default rule for “no”

If \( x = 1 \) and \( y = 1 \) then class = a
If \( z = 1 \) and \( w = 1 \) then class = a
Otherwise class = b

- Order of rules is not important. No conflicts!
- Rule can be written in disjunctive normal form
Rules involving relations

- So far: all rules involved comparing an attribute-value to a constant (e.g. temperature < 45)
- These rules are called “propositional” because they have the same expressive power as propositional logic
- What if problem involves relationships between examples (e.g. family tree problem from above)?
  - Can’t be expressed with propositional rules
  - More expressive representation required
The shapes problem

- Target concept: *standing up*
- Shaded: *standing*
- Unshaded: *lying*
A propositional solution

<table>
<thead>
<tr>
<th>Width</th>
<th>Height</th>
<th>Sides</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>4</td>
<td>4</td>
<td>Standing</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>4</td>
<td>Standing</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>4</td>
<td>Lying</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td>3</td>
<td>Standing</td>
</tr>
<tr>
<td>7</td>
<td>6</td>
<td>3</td>
<td>Lying</td>
</tr>
<tr>
<td>2</td>
<td>9</td>
<td>4</td>
<td>Standing</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>4</td>
<td>Lying</td>
</tr>
<tr>
<td>10</td>
<td>2</td>
<td>3</td>
<td>Lying</td>
</tr>
</tbody>
</table>

If width ≥ 3.5 and height < 7.0 then lying
If height ≥ 3.5 then standing
A relational solution

- Comparing attributes with each other
  
  If width > height then lying
  If height > width then standing

- Generalizes better to new data
- Standard relations: =, <, >
- But: learning relational rules is costly
- Simple solution: add extra attributes
  (e.g. a binary attribute is width < height?)
Rules with variables

- Using variables and multiple relations:

  ```
  If height_and_width_of(x,h,w) and h > w 
  then standing(x)
  ```

- The top of a tower of blocks is standing:

  ```
  If height_and_width_of(x,h,w) and h > w 
  and is_top_of(x,y) 
  then standing(x)
  ```

- The whole tower is standing:

  ```
  If is_top_of(x,z) and 
  height_and_width_of(z,h,w) and h > w 
  and is_rest_of(x,y)and standing(y) 
  then standing(x)
  
  If empty(x) then standing(x)
  ```

- Recursive definition!
Inductive logic programming

- Recursive definition can be seen as logic program
- Techniques for learning logic programs stem from the area of “inductive logic programming” (ILP)
- But: recursive definitions are hard to learn
  - Also: few practical problems require recursion
  - Thus: many ILP techniques are restricted to non-recursive definitions to make learning easier