Data mining techniques: decision trees

Agenda

- Decision trees
- Building a decision tree
- Rule systems
- Building rule systems
- Decision trees vs rule systems
- Quick reference
Decision trees

Set of conditions organized hierarchically in such a way that the final decision can be determined following the conditions that are fulfilled from the root of the tree to one of its leaves.

- They are easily understandable. They build a model (made up by rules) easy to understand for the user.
- They only work over a single table, and over a single attribute at a time.
- They are one of the most used data mining techniques.
Decision trees

The weather problem:

<table>
<thead>
<tr>
<th>Outlook</th>
<th>Temperature</th>
<th>Humidity</th>
<th>Windy</th>
<th>Play</th>
</tr>
</thead>
<tbody>
<tr>
<td>sunny</td>
<td>hot</td>
<td>high</td>
<td>false</td>
<td>no</td>
</tr>
<tr>
<td>sunny</td>
<td>hot</td>
<td>high</td>
<td>true</td>
<td>no</td>
</tr>
<tr>
<td>overcast</td>
<td>hot</td>
<td>high</td>
<td>false</td>
<td>yes</td>
</tr>
<tr>
<td>rainy</td>
<td>mild</td>
<td>high</td>
<td>false</td>
<td>yes</td>
</tr>
<tr>
<td>rainy</td>
<td>cool</td>
<td>normal</td>
<td>false</td>
<td>yes</td>
</tr>
<tr>
<td>rainy</td>
<td>cool</td>
<td>normal</td>
<td>true</td>
<td>no</td>
</tr>
<tr>
<td>overcast</td>
<td>cool</td>
<td>normal</td>
<td>true</td>
<td>yes</td>
</tr>
<tr>
<td>sunny</td>
<td>mild</td>
<td>high</td>
<td>false</td>
<td>no</td>
</tr>
<tr>
<td>sunny</td>
<td>cool</td>
<td>normal</td>
<td>true</td>
<td>yes</td>
</tr>
<tr>
<td>rainy</td>
<td>mild</td>
<td>normal</td>
<td>false</td>
<td>yes</td>
</tr>
<tr>
<td>sunny</td>
<td>mild</td>
<td>normal</td>
<td>true</td>
<td>yes</td>
</tr>
<tr>
<td>overcast</td>
<td>mild</td>
<td>high</td>
<td>true</td>
<td>yes</td>
</tr>
<tr>
<td>overcast</td>
<td>hot</td>
<td>normal</td>
<td>false</td>
<td>yes</td>
</tr>
<tr>
<td>rainy</td>
<td>mild</td>
<td>high</td>
<td>true</td>
<td>no</td>
</tr>
</tbody>
</table>

Decision tree that solves the problem:
Decision trees

- The decision trees are based on the strategy "divide and conquer".
- There are two possible types of divisions or partitions:
  - Nominal partitions: a nominal attribute may lead to a split with as many branches as values there are for the attribute.
  - Numerical partitions: typically, they allow partitions like "X>" and "X <a". Partitions relating two different attributes are not permitted.
- What distinguishes the different algorithms from each others are the partitions they allow, and what criteria they use to select the partitions.

Expressive power: they can only make partitions parallel to the axis:
- 2D example
Agenda

- Decision trees
- Building a decision tree
- Rule systems
- Building rule systems
- Decision trees vs rule systems
- Quick reference

Building a decision tree

The most common approach is to try to minimize the function like

$$I(s) = \sum_{j=1}^{n} p_j \cdot f(p_j^1, p_j^2, \ldots, p_j^c)$$

- $n$ is the number of partitions
- $p_j$ is the probability of falling into the node $j$
- $p_j^c$ is the proportion of elements of the class $c$ falling into the node $j$.
- $f$ is a function of impurity, thus the formula calculates the average "impurity" of the nodes.
Building a decision tree

Some functions:

- Error
- GINI (CART)
- Entropy (C4.5)
- DKM

The more entropy there is, the more information is needed to solve the problem.

Note: log are taken in base 2.

Abraham Otero
Data Mining

11/39

Building a decision tree

The weather problem: ¿Which is the best decision?

- Outlook:

info([2,3]) = 0.971 bits
info([4,0]) = 0.0 bits
info([3,2]) = 0.971 bits

info([2,3]) = \(-2\times\log(2)/5 - 3\times\log(3)/5 = 0.971 \) bits

Abraham Otero
Data Mining

12/39
Building a decision tree

- info([2,3]) = 0.971 bits
- info([4,0]) = 0.0 bits
- info([3,2]) = 0.971 bits

info([2,3], [4,0], [3,2]) = (5/14) × 0.971 + (4/14) × 0 + (5/14) × 0.971 = 0.69

But how much increase in "purity" (information) would that choice produce?
- What was the "purity" at the root of the tree?
  - info([9,5]) = 0.940 bits
- Thus, we are gaining:
  - gain(outlook) = info([9,5]) − info([2,3], [4,0], [3,2]) = 0.940 − 0.69 = 0.247 bits

Thus, the info gain for every possible branch is:

- gain(outlook) = 0.247 bits
- gain(temperature) = 0.029 bits
- gain(humidity) = 0.152 bits
- gain(windy) = 0.048 bits
Building a decision tree

Once the first decision attribute has been chosen, we repeat the operation with the other nodes:

![Diagram of a decision tree](image)

- Gain of `temperature`: 0.571 bits
- Gain of `humidity`: 0.971 bits
- Gain of `windy`: 0.020 bits

Final result:

- With real examples, a certain level of "impurity" in each leaf node is usually tolerated.
- Trying to learn the training data perfectly, will likely lead to overfitting.
Building a decision tree

But…

<table>
<thead>
<tr>
<th>ID code</th>
<th>Outlook</th>
<th>Temperature</th>
<th>Humidity</th>
<th>Windy</th>
<th>Play</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>sunny</td>
<td>hot</td>
<td>high</td>
<td>false</td>
<td>no</td>
</tr>
<tr>
<td>b</td>
<td>sunny</td>
<td>hot</td>
<td>high</td>
<td>true</td>
<td>no</td>
</tr>
<tr>
<td>c</td>
<td>overcast</td>
<td>hot</td>
<td>high</td>
<td>false</td>
<td>yes</td>
</tr>
<tr>
<td>d</td>
<td>rainy</td>
<td>mild</td>
<td>high</td>
<td>false</td>
<td>yes</td>
</tr>
<tr>
<td>e</td>
<td>rainy</td>
<td>cool</td>
<td>normal</td>
<td>false</td>
<td>yes</td>
</tr>
<tr>
<td>f</td>
<td>rainy</td>
<td>cool</td>
<td>normal</td>
<td>true</td>
<td>no</td>
</tr>
<tr>
<td>g</td>
<td>overcast</td>
<td>cool</td>
<td>normal</td>
<td>true</td>
<td>yes</td>
</tr>
<tr>
<td>h</td>
<td>sunny</td>
<td>mild</td>
<td>high</td>
<td>false</td>
<td>no</td>
</tr>
<tr>
<td>i</td>
<td>sunny</td>
<td>cool</td>
<td>normal</td>
<td>false</td>
<td>yes</td>
</tr>
<tr>
<td>j</td>
<td>rainy</td>
<td>mild</td>
<td>normal</td>
<td>false</td>
<td>yes</td>
</tr>
<tr>
<td>k</td>
<td>sunny</td>
<td>mild</td>
<td>normal</td>
<td>true</td>
<td>yes</td>
</tr>
<tr>
<td>l</td>
<td>overcast</td>
<td>mild</td>
<td>high</td>
<td>true</td>
<td>yes</td>
</tr>
<tr>
<td>m</td>
<td>overcast</td>
<td>hot</td>
<td>normal</td>
<td>false</td>
<td>yes</td>
</tr>
<tr>
<td>n</td>
<td>rainy</td>
<td>mild</td>
<td>high</td>
<td>true</td>
<td>no</td>
</tr>
</tbody>
</table>

This attribute gives us a perfect (and useless) classification!!!
- Its info gain is 0.940 bits, i.e., all the information needed to solve the problem:
Building a decision tree

- The solution to this problem is to somehow penalize the attributes that lead to a very high number of branches.
  - One option is to take into account the number and size of the children nodes, regardless of what classes they contain.
  - For our ID attribute:

\[
\text{info}([1,1,\ldots,1]) = -\frac{1}{14} \log \frac{1}{14} \times 14
\]

- This is 3.807 bits. We shall call this value “information value of the attribute”.

Instead of using "gain" to determine which attribute to use for the next branch, we shall use "gain ratio" which we shall define as the division between gain and the information value of the attribute.

- For our ID attribute: gain ratio (ID) = 0.940/3.807 = 0.247.
- For the initial branch:
Building a decision tree

The calculation method presented here is the one used in WEKA for the algorithm C4.5.
- It is one of the best algorithms, both from a theoretical point of view and in practice.
- A commercial variant (C5) implemented in Clementine has a slightly better performance.

Often, too much adjusting to the training data is counterproductive (overtraining).
One solution is to remove some branches of the tree ("pruning").
Two pruning strategies:
- Pre-pruning: the process is done during the construction of the tree. There is some criteria to stop expanding the nodes (allowing a certain level of "impurity" in each node).
- Post-pruning: the process is done after the construction of the tree. Branches are removed from the bottom up to a certain limit. It uses similar criteria to pre-pruning.
Building a decision tree

- The key to knowing when and how much to prune is to check the behavior of the tree with the test data.
- With the training data the behavior of the tree will only improve.
- Pruning is the key to dealing with noisy data.

Agenda

- Decision trees
- Building a decision tree
- Rule systems
- Building rule systems
- Decision trees vs rule systems
- Quick reference
Rule systems

In rule systems the rules are not organized hierarchically, nor must they provide a classification for all the examples.
- There may be instances that are not covered by the rule set.
- Instead of building the rules by "divide and conquer" they use an approach based on coverage.
  - They try to create rules that "cover" all instances of the training data set.

Example:

If $x \leq 1.2$ then class = b
Rule systems

Example:

If $x \leq 1.2$ then class = $b$

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

If $x > 1.2$ and $y \leq 2.6$ then class = $b$

Agenda

- Decision trees
- Building a decision tree
- Rule systems
- Building rule systems
- Decision trees vs rule systems
- Quick reference
Building rule systems

Another example: we have to determine the recommended type of contact lens.

Some of the data:

<table>
<thead>
<tr>
<th>Age</th>
<th>Spectacle prescription</th>
<th>Astigmatism</th>
<th>Tear production rate</th>
<th>Recommended lenses</th>
</tr>
</thead>
<tbody>
<tr>
<td>young</td>
<td>myope</td>
<td>yes</td>
<td>reduced</td>
<td>none</td>
</tr>
<tr>
<td>young</td>
<td>myope</td>
<td>yes</td>
<td>normal</td>
<td>hard</td>
</tr>
<tr>
<td>young</td>
<td>hypermetrope</td>
<td>yes</td>
<td>reduced</td>
<td>none</td>
</tr>
<tr>
<td>pre-presbyopic</td>
<td>myope</td>
<td>yes</td>
<td>reduced</td>
<td>none</td>
</tr>
<tr>
<td>pre-presbyopic</td>
<td>hypermetrope</td>
<td>yes</td>
<td>normal</td>
<td>hard</td>
</tr>
<tr>
<td>pre-presbyopic</td>
<td>hypermetrope</td>
<td>yes</td>
<td>reduced</td>
<td>none</td>
</tr>
<tr>
<td>presbyopic</td>
<td>myope</td>
<td>yes</td>
<td>normal</td>
<td>hard</td>
</tr>
<tr>
<td>presbyopic</td>
<td>hypermetrope</td>
<td>yes</td>
<td>reduced</td>
<td>none</td>
</tr>
<tr>
<td>presbyopic</td>
<td>hypermetrope</td>
<td>yes</td>
<td>normal</td>
<td>hard</td>
</tr>
</tbody>
</table>

Building rule systems

All the data:

- Column 2, age of the patient: (1) young, (2) pre-presbyopic, (3) presbyopic.
- Column 3, spectacle prescription: (1) myope, (2) hypermetrope.
- Column 4, astigmatic: (1) no, (2) yes
- Column 5, tear production rate: (1) reduced, (2) normal.
- Column 6, contact lenses: (1) hard, (2) soft, (3) no contact lenses.
Building rule systems

- We start by looking for conditions that will enable us to determine whether the lenses should be hard:

  If ? then recommendation = hard

- To do this, we calculate the number of instances for which, given the value of an attribute, we can adequately predict that the contact lenses should be hard.

- We then select the attribute that covers a higher percentage of instances that meet the desired criteria (hard contact lenses recommended).

Selection process:

- If ? then recommendation = hard

  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

If astigmatism = yes then recommendation = hard
Building rule systems

- We now try to properly classify the other 8 instances covered by the rule:

  If astigmatism = yes and some condition then recommendation = hard
  - 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

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

Building rule systems

- And we try to properly classify the other 2 instances covered by the rule:

  - age = young: 2/2
  - age = pre-presbyopic: 1/2
  - age = presbyopic: 1/2
  - spectacle prescription = myope: 3/3
  - spectacle prescription = hypermetrope: 1/3

- We now have a rule that classifies perfectly the three instances it covers.

  Now we start the process again, and so on until we have rules that cover all (or most) instances.
Decision trees vs rule systems

- The decision trees are often more efficient because each partition generates two rules.
- Pruning of trees is often simpler than the pruning of rules systems (although the same criteria are used in both cases).
- Rule systems permit non-exhaustive coverage, which is interesting if an attribute has many values, but not all are relevant.
- Rule systems are less voracious, so they tend to fall less in local minima.
Agenda

- Decision trees
- Building a decision tree
- Rule systems
- Building rule systems
- Decision trees vs rule systems
- Quick reference

Quick reference

- They are used mainly for classification (supervised learning), although it is possible to use them for regression, clustering and estimation of probabilities.
- They are relatively efficient and capable of working with large volumes of data.
- They work with numeric and nominal attributes.
- They only work over a single table and a single attribute at a time. They do not allow relations between two attributes (for efficiency).
- They are tolerant to noise, not significant attributes and missing values.
- They build a model (rules) that is easy to understand and interpret by the end user.
Quick reference

Visual comparison of their expressive power:

Rule systems

Decision trees