Machine Learning: Decision Trees

Chapter 18.1-18.3

Some material adopted from notes by Chuck Dyer

What is learning?

• “Learning denotes changes in a system that ... enable a system to do the same task more efficiently the next time” – Herbert Simon
• “Learning is constructing or modifying representations of what is being experienced” – Ryszard Michalski
• “Learning is making useful changes in our minds” – Marvin Minsky

Why study learning?

• Understand and improve efficiency of human learning
  – Use to improve methods for teaching and tutoring people (e.g., better computer-aided instruction)
• Discover new things or structure previously unknown
  – Examples: data mining, scientific discovery
• Fill in skeletal or incomplete specifications in a domain
  – Large, complex systems can’t be completely built by hand & require dynamic updating to incorporate new information
  – Learning new characteristics expands the domain or expertise and lessens the “brittleness” of the system
• Build agents that can adapt to users, other agents, and their environment

AI & Learning Today

• Neural network learning was popular in the 60s
• In the 70s and 80s it was replaced with a paradigm based on manually encoding and using knowledge
• In the 90s, more data and the Web drove interest in new statistical machine learning (ML) techniques and new data mining applications
• Today, ML techniques and big data are behind almost all successful intelligent systems

Machine Learning Successes

- Sentiment analysis
- Spam detection
- Machine translation
- Spoken language understanding
- Named entity detection
- Self-driving cars
- Motion recognition (Microsoft X-Box)
- Identifying faces in digital images
- Recommender systems (Netflix, Amazon)
- Credit card fraud detection

Major paradigms of machine learning

- **Rote learning** – One-to-one mapping from inputs to stored representation. “Learning by memorization.” Association-based storage and retrieval.
- **Induction** – Use specific examples to reach general conclusions
- **Clustering** – Unsupervised identification of natural groups in data
- **Analogy** – Determine correspondence between two different representations
- **Discovery** – Unsupervised, specific goal not given
- **Genetic algorithms** – “Evolutionary” search techniques, based on an analogy to “survival of the fittest”
- **Reinforcement** – Feedback (positive or negative reward) given at the end of a sequence of steps

The Classification Problem

- Extrapolate from set of examples to make accurate predictions about future ones
- Supervised versus unsupervised learning
  - Learn unknown function f(X)=Y, where X is an input example and Y is desired output
  - **Supervised learning** implies we’re given a training set of (X, Y) pairs by a “teacher”
  - **Unsupervised learning** means we are only given the Xs and some (ultimate) feedback function on our performance.

- Concept learning or classification (aka “induction”)
  - Given a set of examples of some concept/class/category, determine if a given example is an instance of the concept or not
  - If it is an instance, we call it a positive example
  - If it is not, it is called a negative example
  - Or we can make a probabilistic prediction (e.g., using a Bayes net)
Supervised Concept Learning

• Given a training set of positive and negative examples of a concept
• Construct a description that will accurately classify whether future examples are positive or negative
• That is, learn some good estimate of function \( f \) given a training set \( \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\} \), where each \( y_i \) is either + (positive) or - (negative), or a probability distribution over +/-.

Measuring Model Quality

• How good is a model?
  – Predictive accuracy
  – False positives / false negatives for a given cutoff threshold
    • Loss function (accounts for cost of different types of errors)
    – Area under the (ROC) curve
    – Minimizing loss can lead to problems with overfitting
• Training error
  – Train on all data; measure error on all data
  – Subject to overfitting (of course we’ll make good predictions on the data on which we trained!)
• Regularization
  – Attempt to avoid overfitting
  – Explicitly minimize the complexity of the function while minimizing loss. Tradeoff is modeled with a regularization parameter

Inductive Learning Framework

• Raw input data from sensors are typically preprocessed to obtain a feature vector, \( X \), that adequately describes all of the relevant features for classifying examples
• Each \( x \) is a list of (attribute, value) pairs. For example, \( X = \{\text{Person:Sue, EyeColor:Brown, Age:Young, Sex:Female}\} \)
• The number of attributes (a.k.a. features) is fixed (positive, finite)
• Each attribute has a fixed, finite number of possible values (or could be continuous)
• Each example can be interpreted as a point in an \( n \)-dimensional feature space, where \( n \) is the number of attributes

Cross-Validation

• Holdout cross-validation:
  – Divide data into training set and test set
  – Train on training set; measure error on test set
  – Better than training error, since we are measuring generalization to new data
  – To get a good estimate, we need a reasonably large test set
  – But this gives less data to train on, reducing our model quality!
Cross-Validation, cont.

• k-fold cross-validation:
  – Divide data into $k$ folds
  – Train on $k-1$ folds, use the $k$th fold to measure error
  – Repeat $k$ times; use average error to measure generalization accuracy
  – Statistically valid and gives good accuracy estimates

• Leave-one-out cross-validation (LOOCV)
  – $k$-fold cross validation where $k=N$ (test data = 1 instance!)
  – Quite accurate, but also quite expensive, since it requires building $N$ models

Inductive learning as search

• Instance space $I$ defines the language for the training and test instances
  – Typically, but not always, each instance $i \in I$ is a feature vector
  – Features are sometimes called attributes or variables
  – $I: V_1 \times V_2 \times \ldots \times V_k, i = (v_1, v_2, ..., v_k)$

• Class variable $C$ gives an instance’s class (to be predicted)

• Model space $M$ defines the possible classifiers
  – $M: I \rightarrow C, M = \{m_1, \ldots, m_n\}$ (possibly infinite)
  – Model space is sometimes, but not always, defined in terms of the same features as the instance space

• Training data can be used to direct the search for a good (consistent, complete, simple) hypothesis in the model space

Model spaces

• Decision trees
  – Partition the instance space into axis-parallel regions, labeled with class value

• Version spaces
  – Search for necessary (lower-bound) and sufficient (upper-bound) partial instance descriptions for an instance to be in the class

• Nearest-neighbor classifiers
  – Partition the instance space into regions defined by the centroid instances (or cluster of $k$ instances)

• Associative rules (feature values $\rightarrow$ class)

• First-order logical rules

• Bayesian networks (probabilistic dependencies of class on attributes)

• Neural networks
Learning decision trees

• Goal: Build a decision tree to classify examples as positive or negative instances of a concept using supervised learning from a training set

• A decision tree is a tree where
  – each non-leaf node has associated with it an attribute (feature)
  – each leaf node has associated with it a classification (+ or -)
  – each arc has associated with it one of the possible values of the attribute at the node from which the arc is directed

• Generalization: allow for >2 classes
  – e.g., for stocks, classify into {sell, hold, buy}

Expressiveness

• Decision trees can express any function of the input attributes
• E.g., for Boolean functions, truth table row → path to leaf:

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>A xor B</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>F</td>
</tr>
</tbody>
</table>

• Trivially, there’s a consistent decision tree for any training set with one path to leaf for each example (unless f nondeterministic in x), but it probably won’t generalize to new examples
• We prefer to find more compact decision trees

Decision tree-induced partition – example

Inductive learning and bias

• Suppose that we want to learn a function f(x) = y and we are given some sample (x,y) pairs, as in figure (a)
• There are several hypotheses we could make about this function, e.g.: (b), (c) and (d)
• A preference for one over the others reveals the bias of our learning technique, e.g.:
  – prefer piece-wise functions
  – prefer a smooth function
  – prefer a simple function and treat outliers as noise
Preference bias: Ockham’s Razor

• AKA Occam’s Razor, Law of Economy, or Law of Parsimony
• Principle stated by William of Ockham (1285-1347)
  – “non sunt multiplicanda entia praeter necessitatem”
  – entities are not to be multiplied beyond necessity
• The simplest consistent explanation is the best
• Therefore, the smallest decision tree that correctly classifies all of the training examples is best
• Finding the provably smallest decision tree is NP-hard, so instead of constructing the absolute smallest tree consistent with the training examples, construct one that is pretty small

Hypothesis spaces

• How many distinct decision trees with \( n \) Boolean attributes?
  – \( = \) number of Boolean functions
  – \( = \) number of distinct truth tables with \( 2^n \) rows = \( 2^{2^n} \)
  – e.g., with 6 Boolean attributes, 18,446,744,073,709,551,616 trees
• How many conjunctive hypotheses (e.g., Hungry ∧ ¬Rain)?
  – Each attribute can be in (positive), in (negative), or out
  \( ⇒ \) \( 3^n \) distinct conjunctive hypotheses
  – e.g., with 6 Boolean attributes, 729 trees
• A more expressive hypothesis space
  – increases chance that target function can be expressed
  – increases number of hypotheses consistent with training set
  \( ⇒ \) may get worse predictions in practice

R&N’s restaurant domain

• Develop a decision tree to model decision a patron makes when deciding whether or not to wait for a table at a restaurant
• Two classes: wait, leave
• Ten attributes: Alternative available? Bar in restaurant? Is it Friday? Are we hungry? How full is the restaurant? How expensive? Is it raining? Do we have a reservation? What type of restaurant is it? What’s the purported waiting time?
• Training set of 12 examples
• \( \sim \) 7000 possible cases

A decision tree from introspection
Attribute-based representations

<table>
<thead>
<tr>
<th>Example</th>
<th>Attributes</th>
<th>Target</th>
</tr>
</thead>
<tbody>
<tr>
<td>X_1</td>
<td>T F F T</td>
<td>T</td>
</tr>
<tr>
<td>X_2</td>
<td>T F F F</td>
<td>F</td>
</tr>
<tr>
<td>X_3</td>
<td>F F F T</td>
<td>F</td>
</tr>
<tr>
<td>X_4</td>
<td>F F F T</td>
<td>F</td>
</tr>
<tr>
<td>X_5</td>
<td>F F F T</td>
<td>F</td>
</tr>
<tr>
<td>X_6</td>
<td>F F F T</td>
<td>T</td>
</tr>
<tr>
<td>X_7</td>
<td>F F F F</td>
<td>T</td>
</tr>
<tr>
<td>X_8</td>
<td>F F F T</td>
<td>T</td>
</tr>
<tr>
<td>X_9</td>
<td>F T T F</td>
<td>T</td>
</tr>
<tr>
<td>X_10</td>
<td>F T T F</td>
<td>F</td>
</tr>
<tr>
<td>X_11</td>
<td>F T T F</td>
<td>F</td>
</tr>
<tr>
<td>X_12</td>
<td>F T T F</td>
<td>T</td>
</tr>
</tbody>
</table>

Examples described by attribute values (Boolean, discrete, continuous), e.g., situations where I will/won't wait for a table.

Classification of examples is positive (T) or negative (F).

Serves as a training set.

ID3/C4.5 Algorithm

- A greedy algorithm for decision tree construction developed by Ross Quinlan circa 1987.
- Top-down construction of decision tree by recursively selecting "best attribute" to use at the current node in tree:
  - Once attribute is selected for current node, generate child nodes, one for each possible value of selected attribute.
  - Partition examples using the possible values of this attribute, and assign these subsets of the examples to the appropriate child node.
  - Repeat for each child node until all examples associated with a node are either all positive or all negative.

Choosing the best attribute

- Key problem: choosing which attribute to split a given set of examples.
- Some possibilities are:
  - Random: Select any attribute at random
  - Least-Values: Choose the attribute with the smallest number of possible values
  - Most-Values: Choose the attribute with the largest number of possible values
  - Max-Gain: Choose the attribute that has the largest expected information gain—i.e., attribute that results in smallest expected size of subtrees rooted at its children.
- The ID3 algorithm uses the Max-Gain method of selecting the best attribute.

Choosing an attribute

Idea: a good attribute splits the examples into subsets that are (ideally) "all positive" or "all negative".

Which is better: Patrons? or Type?
Restaurant example

Random: Patrons or Wait-time; Least-values: Patrons; Most-values: Type; Max-gain: ???

<table>
<thead>
<tr>
<th>Type variable</th>
<th>French</th>
<th>Italian</th>
<th>Thai</th>
<th>Burger</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td></td>
<td>N</td>
<td></td>
<td>Y</td>
<td>Y</td>
</tr>
</tbody>
</table>

Patrons variable

Splitting examples by testing attributes

ID3-induced decision tree

Compare the two Decision Trees
Information theory 101

• Intuitions
  – Common words (a, the, dog) shorter than less common ones (parliamentarian, foreshadowing)
  – Morse code: common (probable) letters have shorter encodings
• Information is measured in minimum number of bits needed to store or send some information
• Wikipedia: The measure of data, known as information entropy, is usually expressed by the average number of bits needed for storage or communication.

Information theory II

• Information conveyed by distribution (aka entropy of P):
  \[ I(P) = -(p_1 \log(p_1) + p_2 \log(p_2) + \ldots + p_n \log(p_n)) \]
• Examples:
  – If P is (0.5, 0.5) then I(P) = 0.5\*\log(0.5) + 0.5\*\log(0.5) = 1
  – If P is (0.67, 0.33) then I(P) = -(2/3\*\log(2/3) + 1/3\*\log(1/3)) = 0.92
  – If P is (1, 0) then I(P) = 1*\log(1) + 0*\log(0) = 0
• The more uniform the probability distribution, the greater its information: more information is conveyed by a message telling you which event actually occurred
• Entropy is the average number of bits/message needed to represent a stream of messages

Example: Huffman code

• In 1952 MIT student David Huffman devised, in the course of doing a homework assignment, an elegant coding scheme which is optimal in the case where all symbols’ probabilities are integral powers of 1/2.
• A Huffman code can be built in the following manner:
  – Rank all symbols in order of probability of occurrence
  – Successively combine the two symbols of the lowest probability to form a new composite symbol; eventually we will build a binary tree where each node is the probability of all nodes beneath it
  – Trace a path to each leaf, noticing direction at each node
Huffman code example

If we use this code to many messages (A, B, C or D) with this probability distribution, then, over time, the average bits/message should approach 1.75.

Information for classification

If a set T of records is partitioned into disjoint exhaustive classes (C_1,C_2,...,C_k) on the basis of the value of the class attribute, then information needed to identify class of an element of T is:

\[ \text{Info}(T) = I(P) \]

where P is the probability distribution of partition (C_1,C_2,...,C_k):

\[ P = (|C_1|/|T|, |C_2|/|T|, ..., |C_k|/|T|) \]

Information for classification II

If we partition T w.r.t attribute X into sets \{T_1,T_2,...,T_n\} then the information needed to identify the class of an element of T becomes the weighted average of the information needed to identify the class of an element of T_i, i.e. the weighted average of Info(T_i):

\[ \text{Info}(X,T) = \sum |T_i|/|T| \times \text{Info}(T_i) \]

Information gain

• Consider the quantity Gain(X,T) defined as

\[ \text{Gain}(X,T) = \text{Info}(T) - \text{Info}(X,T) \]

• This represents the difference between
  – info needed to identify element of T and
  – info needed to identify element of T after value of attribute X known

• This is the gain in information due to attribute X

• Use to rank attributes and build DT where each node uses attribute with greatest gain of those not yet considered (in path from root)

• The intent of this ordering is to:
  – Create small DTs so records can be identified with few questions
  – Match a hoped-for minimality of the process represented by the records being considered (Occam’s Razor)
### Computing Information Gain

<table>
<thead>
<tr>
<th>Attribute</th>
<th>French</th>
<th>Italian</th>
<th>Thai</th>
<th>Burger</th>
<th>Empty</th>
<th>Some</th>
<th>Full</th>
</tr>
</thead>
<tbody>
<tr>
<td>I(T)</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>I (Pat, T)</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>I (Type, T)</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>N</td>
</tr>
</tbody>
</table>

Gain (Pat, T) = ?
Gain (Type, T) = ?

### Computing Information Gain

<table>
<thead>
<tr>
<th>French</th>
<th>Y</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>I(T)</td>
<td>- (.5 log .5 + .5 log .5) = .5 + .5 = 1</td>
<td></td>
</tr>
<tr>
<td>I (Pat, T)</td>
<td>2/12 (0) + 4/12 (0) + 6/12 (- (4/6 log 4/6 + 2/6 log 2/6)) = 1/2 (2/3*.6 + 1/3*1.6) = .47</td>
<td></td>
</tr>
<tr>
<td>I (Type, T)</td>
<td>2/12 (1) + 2/12 (1) + 4/12 (1) + 4/12 (1) = 1</td>
<td></td>
</tr>
</tbody>
</table>

Gain (Pat, T) = 1 - .47 = .53
Gain (Type, T) = 1 – 1 = 0

### How well does it work?

Many case studies have shown that decision trees are at least as accurate as human experts.

- A study for diagnosing breast cancer had humans correctly classifying the examples 65% of the time; the decision tree classified 72% correct
- British Petroleum designed a decision tree for gas-oil separation for offshore oil platforms that replaced an earlier rule-based expert system
- Cessna designed an airplane flight controller using 90,000 examples and 20 attributes per example
Extensions of ID3

- Using gain ratios
- Real-valued data
- Noisy data and overfitting
- Generation of rules
- Setting parameters
- Cross-validation for experimental validation of performance
- C4.5 is an extension of ID3 that accounts for unavailable values, continuous attribute value ranges, pruning of decision trees, rule derivation, and so on

Using gain ratios

- The information gain criterion favors attributes that have a large number of values
  - If we have an attribute D that has a distinct value for each record, then Info(D,T) is 0, thus Gain(D,T) is maximal
- To compensate for this Quinlan suggests using the following ratio instead of Gain:
  \[
  \text{GainRatio}(D,T) = \frac{\text{Gain}(D,T)}{\text{SplitInfo}(D,T)}
  \]
- SplitInfo(D,T) is the information due to the split of T on the basis of value of categorical attribute D
  \[
  \text{SplitInfo}(D,T) = I\left(\frac{|T_1|}{|T|}, \frac{|T_2|}{|T|}, ..., \frac{|T_m|}{|T|}\right)
  \]
  where \{T_1, T_2, .., T_m\} is the partition of T induced by value of D

Real-valued data

- Select a set of thresholds defining intervals
- Each interval becomes a discrete value of the attribute
- Use some simple heuristics…
  - always divide into quartiles
- Use domain knowledge…
  - divide age into infant (0-2), toddler (3 - 5), school-aged (5-8)
- Or treat this as another learning problem
  - Try a range of ways to discretize the continuous variable and see which yield “better results” w.r.t. some metric
  - E.g., try midpoint between every pair of values

<table>
<thead>
<tr>
<th>Attribute</th>
<th>French</th>
<th>Italian</th>
<th>Thai</th>
<th>Burger</th>
<th>Empty</th>
<th>Some</th>
<th>Full</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>N Y</td>
</tr>
<tr>
<td>Gain (Pat, T)</td>
<td>.53</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Gain (Type, T)</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SplitInfo (Pat, T)</td>
<td>(-1/6 \log 1/6 + 1/6 \log 1/6 + 1/3 \log 1/3 + 1/6 \log 1/2) = 1.47)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SplitInfo (Type, T)</td>
<td>(-1/6 \log 1/6 + 1/6 \log 1/6 + 1/3 \log 1/3 + 1/3 \log 1/3) (-1/6 \log 1/6 + 1/6 \log 1/6 + 1/3 \log 1/3 + 1/3 \log 1/3) (-1/6 \log 1/6 + 1/6 \log 1/6 + 1/3 \log 1/3 + 1/3 \log 1/3) (-1/6 \log 1/6 + 1/6 \log 1/6 + 1/3 \log 1/3 + 1/3 \log 1/3)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GainRatio (Pat, T)</td>
<td>(\text{Gain}(Pat, T) / \text{SplitInfo}(Pat, T) = .53 / 1.47 = .36)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Noisy data

• Many kinds of “noise” can occur in the examples:
  • Two examples have same attribute/value pairs, but different classifications
  • Some values of attributes are incorrect because of errors in the data acquisition process or the preprocessing phase
  • The classification is wrong (e.g., + instead of -) because of some error
  • Some attributes are irrelevant to the decision-making process, e.g., color of a die is irrelevant to its outcome

Overfitting

• Irrelevant attributes, can result in overfitting the training example data
  • If hypothesis space has many dimensions (large number of attributes), we may find meaningless regularity in the data that is irrelevant to the true, important, distinguishing features
  • If we have too little training data, even a reasonable hypothesis space will ‘overfit’

Overfitting

• Fix by by removing irrelevant features
  – E.g., remove ‘year observed’, ‘month observed’, ‘day observed’, ‘observer name’ from feature vector
• Fix by getting more training data
• Fix by pruning lower nodes in the decision tree
  – E.g., if gain of the best attribute at a node is below a threshold, stop and make this node a leaf rather than generating children nodes

Pruning decision trees

• Pruning of the decision tree is done by replacing a whole subtree by a leaf node
  • The replacement takes place if a decision rule establishes that the expected error rate in the subtree is greater than in the single leaf. E.g.,
    – Training: one training red success and two training blue failures
    – Test: three red failures and one blue success
    – Consider replacing this subtree by a single Failure node.
  • After replacement we will have only two errors instead of five:
    | Color | Test  | Pruned |
    |-------|-------|--------|
    | red   | 1 success 1 failure | FAILURE 2 success 4 failure |
    | blue  | 0 success 2 failures | |
    | red   | 1 success 1 failure | |
    | blue  | 1 success 1 failure | |
Converting decision trees to rules

• It is easy to derive rules from a decision tree: write a rule for each path from the root to a leaf
• In that rule the left-hand side is built from the label of the nodes and the labels of the arcs
• The resulting rules set can be simplified:
  – Let LHS be the left hand side of a rule
  – LHS’ obtained from LHS by eliminating some conditions
  – Replace LHS by LHS’ in this rule if the subsets of the training set satisfying LHS and LHS’ are equal
  – A rule may be eliminated by using meta-conditions such as “if no other rule applies”
**Zoo example**

```python
aima-python> python
>>> from learning import *
>>> zoo
<DataSet(zoo): 101 examples, 18 attributes>
>>> dt = DecisionTreeLearner()
>>> dt.train(zoo)
>>> dt.predict(['shark',0,0,1,0,1,1,1,0,0,1,0,1,0,1,0,0])
'fish'
>>> dt.predict(['shark',0,0,0,0,1,1,1,0,1,0,1,0,1,0,0])
'mammal'
```

**Zoo example**

```python
>>> dt.dt.display()
Test legs
legs = 0 ==> Test fins
fins = 0 ==> Test toothed
  toothed = 0 ==> RESULT = shellfish
  toothed = 1 ==> RESULT = reptile
fins = 1 ==> Test eggs
  eggs = 0 ==> RESULT = mammal
  eggs = 1 ==> RESULT = fish
legs = 2 ==> Test hair
hair = 0 ==> RESULT = bird
hair = 1 ==> RESULT = mammal
legs = 4 ==> Test aquatic
aquatic = 0 ==> Test toothed
  toothed = 0 ==> RESULT = shellfish
  toothed = 1 ==> RESULT = amphibian
aquatic = 1 ==> RESULT = mammal
legs = 5 ==> RESULT = shellfish
legs = 6 ==> Test aquatic
aquatic = 0 ==> RESULT = insect
aquatic = 1 ==> RESULT = shellfish
legs = 8 ==> RESULT = shellfish
```

Add the shark example to the training set and retrain

```python
>>> dt.dt.display()
Test legs
legs = 0 ==> Test fins
fins = 0 ==> Test toothed
  toothed = 0 ==> RESULT = shellfish
  toothed = 1 ==> RESULT = reptile
fins = 1 ==> Test milk
  milk = 0 ==> RESULT = fish
  milk = 1 ==> RESULT = mammal
legs = 2 ==> Test hair
hair = 0 ==> RESULT = bird
hair = 1 ==> RESULT = mammal
legs = 4 ==> Test aquatic
aquatic = 0 ==> Test toothed
  toothed = 0 ==> RESULT = shellfish
  toothed = 1 ==> RESULT = amphibian
aquatic = 1 ==> RESULT = mammal
legs = 5 ==> RESULT = shellfish
legs = 6 ==> Test aquatic
aquatic = 0 ==> RESULT = insect
aquatic = 1 ==> RESULT = shellfish
legs = 8 ==> RESULT = shellfish
```
Summary: Decision tree learning

- Widely used learning methods in practice
- Can out-perform human experts in many problems
- Strengths include
  - Fast and simple to implement
  - Can convert result to a set of easily interpretable rules
  - Empirically valid in many commercial products
  - Handles noisy data
- Weaknesses include
  - Univariate splits/partitioning using only one attribute at a time so limits types of possible trees
  - Large decision trees may be hard to understand
  - Requires fixed-length feature vectors
  - Non-incremental (i.e., batch method)