CSCI 4150 Introduction to Artificial Intelligence, Fall 2003
Assignment 6 (130 points): out Monday November 10, due Thursday November 20

Problems

1. (24 points, written) You’re working for a bank, and they have some data on whether a loan
application should be approved or not:

<table>
<thead>
<tr>
<th>Example No.</th>
<th>House</th>
<th>Bills</th>
<th>Income</th>
<th>Credit</th>
<th>Approve?</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Rent</td>
<td>Late</td>
<td>Low</td>
<td>Bad</td>
<td>Yes</td>
</tr>
<tr>
<td>2</td>
<td>Rent</td>
<td>Late</td>
<td>High</td>
<td>Bad</td>
<td>Yes</td>
</tr>
<tr>
<td>3</td>
<td>Rent</td>
<td>On-time</td>
<td>Low</td>
<td>Bad</td>
<td>No</td>
</tr>
<tr>
<td>4</td>
<td>Rent</td>
<td>On-time</td>
<td>Medium</td>
<td>Bad</td>
<td>Yes</td>
</tr>
<tr>
<td>5</td>
<td>Own</td>
<td>Late</td>
<td>Medium</td>
<td>Good</td>
<td>Yes</td>
</tr>
<tr>
<td>6</td>
<td>Own</td>
<td>Late</td>
<td>Low</td>
<td>Bad</td>
<td>Yes</td>
</tr>
<tr>
<td>7</td>
<td>Own</td>
<td>On-time</td>
<td>Medium</td>
<td>Good</td>
<td>No</td>
</tr>
<tr>
<td>8</td>
<td>Own</td>
<td>On-time</td>
<td>Low</td>
<td>Bad</td>
<td>No</td>
</tr>
<tr>
<td>9</td>
<td>Own</td>
<td>On-time</td>
<td>High</td>
<td>Good</td>
<td>Yes</td>
</tr>
<tr>
<td>10</td>
<td>Own</td>
<td>On-time</td>
<td>Medium</td>
<td>Good</td>
<td>No</td>
</tr>
<tr>
<td>11</td>
<td>Own</td>
<td>On-time</td>
<td>High</td>
<td>Bad</td>
<td>Yes</td>
</tr>
<tr>
<td>12</td>
<td>Own</td>
<td>On-time</td>
<td>High</td>
<td>Good</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Suppose you start to learn a decision tree on the above data using “Approve?” as the goal
predicate. Calculate the information gains for splitting the data (at the top level) for each of
the four attributes. Which attribute provides the largest information gain (and would there-
fore be the top level attribute in a decision tree)? Show and explain your work.

2. (36 points) Write the procedure:

(learn-dtree training-data attribute-names)

which returns a decision tree learned from the training data using the algorithm covered in
class (also in the text) with the “greatest information gain” heuristic. Scheme representations
for decision trees and training data are described in the remainder of this assignment hand-
out. Support code will be provided to do some of the more mundane data manipulation
tasks.

3. (30 points) Write the procedure:

(chi^2-learn-dtree training-data attribute-names)

which learns a decision tree as for the previous problem but incorporates \( \chi^2 \) pruning as de-
scribed in the text. This procedure will be similar to your learn-dtree procedure.

4. (20 points) Write the procedure:

(missing-learn-dtree training-data attribute-names)

which learns a decision tree from training data with missing attribute values. More informa-
tion on this problem will be posted to the Assignment 6 information page.

5. (20 points, written) For this problem, you will run tests on your learn-dtree and chi^2-
learn-dtree procedures in order to analyze and compare their performance. Details of
these tests and on what you must turn in for this problem are described on the Assignment 6
web page.
Scheme representations

Training data and examples

Training data consist of a list of training examples, and a training example is a list where the first element is the value of the goal predicate (which can be any symbol, not just yes or no) and the second element is a list of the attribute values. We will also require a list of attribute names so we can refer to attributes by name.

Here are the last four training examples from the first problem made into a training data set:

(define loan-names '(House Bills Income Credit))
(define loan-data-small
 '((Yes (Own On-time High Good))
  (Yes (Own On-time Medium Good))
  (No (Own On-time High Bad))
  (Yes (Own On-time High Good))))

Note that the goal predicate is not explicitly named.

Decision trees

A decision tree is either a value for the goal predicate (i.e. a symbol) or a list of the following form:

(<attribute-name> (<attribute-value-1> <decision-tree-1>)
  ...
  (<attribute-value-n> <decision-tree-n>))

For the "loan" example, a valid decision tree is:

(define loan-dtree-example
 'income (high yes)
  (low (house (rent no)
            (own yes))))

Support code

You need not use the following procedures, but you will probably find them helpful. They are organized into several different categories in the sections below.

Handling training data

• (split training-data attribute-names attribute)

This function divides the training data into groups according to the specified attribute. For example, using the training data above:

(split loan-data-small loan-names 'income)
;Value: ((medium ((yes (own on-time medium good)))))
  (high ((yes (own on-time high good))
         (no (own on-time high bad))
         (yes (own on-time high good))))
This procedure returns a list of what I refer to as \textit{splits}. Each split is a list whose first element is a value of the attribute and whose second element is a list containing a subset of the training data which all have that value for the given attribute.

Note that in the example above, there is no split generated for the income attribute value “low” because the training data do not have an example with this value. See the “Implementation notes” section for discussion of this issue.

- (tally training-data)

This procedure counts the number of examples for each value of the goal predicate. It returns a list of clauses. Each clause is a list where the first element each is the value of the goal predicate, and the second element is the number of examples with that value.

For example,

(tally loan-data-small)
;Value: ((no 1) (yes 3))

Do not assume that the goal predicate will always have the values “yes” and “no”!
Like the \texttt{split} procedure, if there are no examples with a given goal predicate value, that value will not appear in the tally.

- (pick-majority tally)

Given a tally (as returned by the \texttt{tally} procedure), this procedure returns the majority value. If there is a tie, it returns the first instance it finds. For example:

(pick-majority '((no 1) (yes 3)))
;Value: yes

Testing your decision trees

- (classify example decision-tree attribute-names default-value)

This function returns a classification for the example determined by the given decision tree. If an attribute value not in the decision tree is encountered, then it returns the default-value.

For example:

(classify '(rent late high good) loan-dtree-example loan-names 'No)
;Value: yes

- (test decision-tree training-data attribute-names)

This function takes a decision tree and a set of training data. From the training data, it creates a list of examples (i.e. just the attribute values) and a list of correct classifications. It classifies all the examples using the decision tree, compares the results to the correct classifications, and reports the results.

Implementation notes

Differences from the text’s algorithm

The basic decision tree learning algorithm you should implement for this assignment is slightly different than the algorithm in our text. The difference is in how the decision tree will handle examples that have attribute values not seen in the training data.
The algorithm in the text handles this by making a recursive call to the `learn-dtree` procedure with zero examples. This returns a decision (sub)tree that consists of a leaf node: the default classification.

The way the support code for this assignment is structured, you should never make a recursive call to your `learn-dtree` procedure when there are no examples left in a given branch. Instead, the `classify` procedure returns the default value if it encounters an attribute value not in the decision tree.

The reason for this difference is to simplify your code. In order to implement the text's algorithm, you would have to know all the values of each attribute, and they would have to be passed down from one recursive call to the next. Leaving this situation to be handled by the `classify` procedure means that only the attribute names need to be passed.

As an example, consider the decision tree in Figure 18.8 of the text. My solutions, run on the same training data produce the decision tree:

\[
(\text{learn-dtree restaurant-data restaurant-names})
\]

;Value: (patrons (none no)
  (full (hungry (no no)
    (yes (type (burger yes)
      (italian no)
      (thai (fri (no no)
        (yes yes)))))))))

(some yes))

Notice that there is no French value handled under type. This is because the French restaurants in the training data were classified under other cases of the decision tree. (One training example had some patrons; in the other, patrons was full, and hungry was no.)

Chi-squared pruning

The purpose of chi-squared pruning is to test whether splitting on an attribute contributes a statistically significant amount of information. This is briefly covered in our text on pages 661–3, but you will probably find the following explanation clearer.

In the following discussion, the two classifications “positive” and “negative” are used. For Problem 3 you may assume that there are just two classifications, but you should not make any assumptions as to exactly what they are (e.g., yes, no, positive, negative, edible, poisonous, etc.)

It’s not that much more difficult to write your code so that it works with an arbitrary number of classifications.

The chi-squared test is a measurement of whether splitting on a given attribute matters. Before splitting on an attribute, we can characterize how many examples there are for each classification and see how closely each split follows these proportions. If the attribute doesn’t matter, then we’d expect that each “split” of training data (as divided by this attribute) to have approximately the same fraction of examples in each classification as the training data did before splitting on this attribute.

Suppose the current call to the learn-dtree is given a set of training data \(S\). These data have mixed classifications; the number of positive examples in \(S\) is \(p\), and the number of negative examples is \(n\). We know that \(p + n = |S|\).

Through the information analysis, we have picked an attribute that splits \(S\) into a number of subsets \(S_i\). Each of these subsets will (in general) have mixed classifications. Let \(p_i\) be the number of positive examples in \(S_i\) and \(n_i\) be the number of negative examples. We know that \(p_i + n_i = |S_i|\).
If this attribute was irrelevant, then we would expect that the number of positive examples in $S_i$ would be:

$$\hat{p}_i = \frac{p}{p + n}|S_i|$$

Likewise, we would expect the number of negative examples to be:

$$\hat{n}_i = \frac{n}{p + n}|S_i|$$

Now, we calculate how much “error” there is between the actual number of positive and negative examples in each $S_i$ and the expected number.

$$Q = \sum_i \frac{(p_i - \hat{p}_i)^2}{p_i} + \frac{(n_i - \hat{n}_i)^2}{n_i}$$

Note that this sum is over each subset created by the split on this attribute. If there are three values for this attribute, then $i$ goes from 1 to 3 in the sum above.

If $Q$ is small, that means that the attribute is not relevant — the data in each split are following the same distribution as the data before splitting on this attribute. If $Q$ is large, that means that there is a lot of “error” between what we would have expected (under the irrelevant attribute hypothesis) and the actual distribution of examples.

The $\chi^2$ distribution will determine the probability that the attribute is not relevant. I have provided a Scheme implementation of this function in the support code: `(chi^2 dof Q)` You should use $dof = (number \ of \ values \ of \ this \ attribute) - 1$. The `chi^2` procedure will return a probability between 0 and 1 which indicates that our hypothesis (that this attribute doesn’t matter) is correct. We will set a low threshold for this test: if the probability is greater than 0.001, then we will assume that the hypothesis is correct.

Although this threshold may seem low, consider the following quote from “Numerical Recipes in C”

“It is not uncommon to deem acceptable on equal terms any models with, say, $Q > 0.001$. This is not as sloppy as it sounds: truly wrong models will often be rejected with vastly smaller values of $Q$, $1e-18$, say.

If the “irrelevance hypothesis” is deemed correct, then you should not split on this attribute — instead just pick the majority classification.

Suggestions

- Because the decision tree representations, tallies, and splits can be confusing, I strongly suggest using simple accessor functions to access information from these data structures.
- Take advantage of the fact that Scheme is interpreted and test your procedures from the bottom up — make sure your lower level functions are doing the right thing before you go on to the higher level functions!
- You will probably find the MIT-Scheme `delete` procedure useful.
- Do not attempt to take the logarithm of 0!!!

Data sets

There will be several data sets available for you to test your procedures. We’ll also be collecting a data set. See the Assignment 6 web page for the files.