CSCI–4150 Introduction to Artificial Intelligence, Fall 2004
Assignment 6 (132 points), out Monday November 8, due Thursday November 18

A. Problems

1. (16 points, written) You're working for a bank, and they have some data on whether a loan application should be approved or not:

Example No.	House	Bills	Income	Credit	Approve?
1	Rent	Late	Low	Bad	No
2	Rent	Late	Low	Good	No
3	Rent	Late	High	Bad	No
4	Rent	On-time	Medium	Good	No
5	Rent	On-time	Medium	Good	Yes
6	Own	Late	Low	Bad	Yes
7	Own	Late	Low	Good	Yes
8	Own	Late	Medium	Bad	No
9	Own	Late	High	Good	No
10	Own	On-time	Medium	Good	Yes
11	Own	On-time	High	Good	No
12	Own	On-time	High	Good	No

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 therefore be the top level attribute in a decision tree)? Show and explain your work.

2. (30 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 handout. Support code will be provided to do some of the more mundane data manipulation tasks.

3. (16 points) Write the procedure:

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

which learns a decision tree from training data with missing attribute values. Your procedure should "split" examples with missing attribute values and send the appropriate fractions of each example down all branches of the tree.

4. (20 points) Write the procedure:

```
(create-binary-discretization c-attribute values training-data attribute-names)
```

which should return a procedure that takes a single example and discretizes a single continuous attribute into a binary attribute with the two values in the list `values`.

5. (20 points) Write the procedure:

```
(create-multi-discretization c-attribute values training-data attribute-names)
```

which should return a procedure that takes a single example and discretizes a single continuous attribute into a attribute with the three (or more, if you wish) values in the list `values`.

6. (30 points, written) For this problem, you will describe how you computed thresholds for the `create-multi-discretization` procedure, and you will run tests on your code using several data sets (including the one we are collecting from the class). Details of these tests and on what you must turn in for this problem will be described on the Assignment 6 information (Web) page.

B. 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
  '((No  (Own Late    High   Good))
    (Yes (Own On-time Medium Good))
    (No  (Own On-time High   Good))
    (No  (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)))))
```

C. Support code

Here are the basic procedures in the support code. See the Assignment 6 Web site for a complete list.

Handling training data

- `(split-tdata 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-tdata loan-data-small loan-names 'income)
  ;Value: ((medium ((yes (own on-time medium good))))
  ;       (high ((no (own on-time high good))
  ;              (no (own on-time high good))
  ;              (no (own late high good)))))
  ```

This procedure returns a list of what I refer to as *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-tdata 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-tdata loan-data-small)
;Value: ((yes 1) (no 3))
```

Do not assume that the goal predicate will always have the values "yes" and "no"! Also do not assume that there will always be only two values of the goal predicate!

Like the `split-tdata` procedure, if there are no examples with a given goal predicate value, that value will not appear in the tally.

This procedure will handle "weights" for each training example which will be useful for Problem 3. See the Assignment 6 information page for details.

- `(pick-majority tally)`

Given a tally (as returned by the `tally-tdata` procedure), this procedure returns the majority value. If there is a tie, it returns the first instance it finds. For example:

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

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-dtree 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.

Other

- `(log2 x)`

Takes the logarithm to base 2. Signals an error if you try to take the logarithm of a nonpositive number.

D.   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 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:

```
(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`.)

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, including the data set we are collecting from the class. See the Assignment 6 web page for details.