CS478 Machine Learning
Spring 2000
Assignment 1

Due electronically by Tuesday, February 22, 11a.m.

Problem Set Portion (30 pts)

1. Exercise 1.2, in *Mitchell* (5 pts).
2. Exercise 2.2, in *Mitchell* (10 pts).
3. Exercise 2.4, in *Mitchell* (15 pts). You don't need to turn in the hypotheses drawn on the diagram.

Programming Portion (70 pts)

Implement the ID3 decision tree algorithm as described in Sections 3.1–3.4 of the *Mitchell* text. You may use C, C++, Java, Lisp, or Scheme to implement ID3. If you would like to use any other language, contact Prof. Cardie for approval first.[1] Your system only needs to handle binary classification tasks (i.e. each instance will belong to one of two classes). However, your system must be able to handle multi-valued attributes. The attribute names will be specified at the beginning of the training file, but you will need to determine the possible values for each attribute dynamically by reading the training instances. (You can assume that all possible values are represented in the training data and that there are no missing values in the training or test data.) It may be easiest to treat each attribute value as a string. You do not need to implement pruning or deal with continuous-valued inputs, etc.

You may assume that each data file will contain no more than 1000 instances, the instance representations will use no more than 200 attributes, and each attribute will have no more than 20 possible values. When building a decision tree, if you reach a leaf node but still have examples that belong to different classes, then choose the most frequent class (among the instances at the leaf node). If you reach a leaf node in the decision tree and have no examples left or the examples are equally split among multiple classes, then choose the class that's most frequent in the *entire* training set.

Your program should prompt the user for a training file and a test file or allow both to be specified in the command line invocation of your program. This is important because we will be running your program on additional data sets for grading purposes! There should be no graphical user interface (GUI) of any kind.

You will need to turn in a **Trace File** showing the performance of your decision tree program on various training and test sets.

Generating a Trace File

1. Build a decision tree using the training instances and print the first four levels of the decision tree in a depth-first fashion. For each node, print the attribute name associated with the node. For each branch, print the attribute value associated with the branch. For each leaf node, print the classification of the instances at that node followed by the number of instances. The exact format that you should use is illustrated on the next page with an example. For

[1] For C and C++ prorammers, we encourage the use of *gcc*, the GNU compiler. For Java programmers, we encourage the use of SUN's JDK 1.2.

simplicity, in this example we assumed that there are exactly two instances at each *mammal* leaf node and three instances at each *notmammal* leaf node. Of course, in a real decision tree the number of instances at each leaf node can be highly variable!

```
Sample decision tree:
                    DOMESTIC
                  yes/   \no
                    /     \
                mammal     \
                           LEGS
                         /    | \
                     two/ four|  \eight
                       /      |   \
                      /       |    \
                notmammal   TAIL   notmammal
                            / \
                       yes /   \ no
                          /     \
                      mammal  notmammal
```

```
Sample trace for the decision tree above:

LEVEL1: DOMESTIC
BRANCH: yes
LEVEL2: mammal (2)
BRANCH: no
LEVEL2: LEGS
BRANCH: two
LEVEL3: notmammal (3)
BRANCH: four
LEVEL3: TAIL
BRANCH: yes
LEVEL4: mammal (2)
BRANCH: no
LEVEL4: notmammal (3)
BRANCH: eight
LEVEL3: notmammal (3)
```

2. Use the learned decision tree to classify the **training** instances. Print the accuracy of the tree. (In this case, the tree has been trained and tested on the same data set.) The accuracy should be computed as the percentage of examples that were correctly classified. For example, if 86 of 90 examples are classified correctly, then the accuracy of the decision tree would be 95.6%.

```
Accuracy on training set (90 instances):  95.6%
```

3. Use the learned decision tree to classify the **test** instances. Print the accuracy of the tree. (In this case, the decision tree has been trained and tested on different data sets.)

```
Accuracy on test set (10 instances):      60.0%
```

Grading Criteria

The programming portion of this assignment will be graded on both correctness (50 points) and documentation (20 points).

Correctness

50 points will be based on the correctness of your decision tree program and the trace output. We will likely run your program on a new data set to test your code, so we encourage you to do the same!

Documentation

20 points will be based on the documentation accompanying your source code. We expect each source file to contain a paragraph or two at the beginning to describe the contents of that file. The main program file should describe the functionality of the program: the type of input it expects, the type of output it produces, and the function that it performs. The data structures used in the program must also be clearly described. The code should be modular. This implies that functions (procedures, methods) should be SHORT. Each function (procedure, method) should also be documented, and in-line comments should explain any code that is unusual or potentially confusing. With your source code, please include a README file that gives us instructions for running your program.

Files to Turn In

1. The **source code** and **executable(s)** for the decision tree program. Include all files that go with it, such as .h files. If you are using C or C++, please include a **Makefile**. The extension of the source files should indicate the programming language in which the code was written (e.g. dtree.c or dtree.scheme).

2. The **README** file.

3. A trace of your **dtree** program on the two data sets listed below. The data sets can be downloaded from the CS478 web page.

   (a) tictactoe.train
   tictactoe.test

   (b) mushroom.train
   mushroom.test

   Trace files should be named with a ".trace" extension.

4. Your answers to the problem set. The answers to the questions may be turned in as an ASCII text file (**assign1.txt**), a postscript file (**assign1.ps**), or a pdf file (**assign1.pdf**). Other file formats will not be accepted.

Electronic Submission

All of the files for this assignment should be submitted electronically as a single **zip archive** or **tar** file. Electronic submission is done through an interface available at the CS478 home page. Any additional instructions for electronic submission will be posted there.

Data File Format

The data files will have the following format. The file will begin with the keyword "attributes" followed by the list of attribute names and then the keyword "instances" followed by a list of instances. Each instance will consist of a name, a list of attribute values, and a classification, i.e. a class value.

The following is a sample data file. This sample file lists six attributes and eight instances. Each instance has a name (e.g. "dog"), six values for each of the attributes respectively (e.g. "no no four yes yes no"), and the correct classification for that instance (e.g. "mammal").

```
attributes feathers fins legs tail domestic aquatic
instances
dog no no four yes yes no mammal
cat no no four yes yes no mammal
sparrow yes no two yes no no notmammal
elephant no no four yes no no mammal
mouse no no four yes no no mammal
dolphin no no zero yes no yes mammal
spider no no eight no no no notmammal
worm no no zero no no no notmammal
```

The training and test files will both have the same format. But note that the classifications are included for different purposes. During training, the classifications are essential to build the decision tree. During testing, the classifications should be used only to determine whether the decision tree correctly classified each instance or not.