CISC 481/681 Intro to Artificial Intelligence
Program 3

1 Introduction

This project requires you to implement a decision tree learning algorithm, run experiments on real datasets, and write a report explaining your experimental results. For full credit your program will have to handle data sets with an arbitrary number of (continuous) numeric-valued features, an arbitrary number of output classes, and an arbitrary number of example instances. Your program should be able to interpret the data format specified below, produce a human readable version of the decision tree formed from a data set, and be able to use the decision tree to classify instances (producing, of course, interesting statistics on the produced trees such as accuracy and size).

If you use Lisp you may want to briefly review Chapter 4 (especially 4.6) of the Lisp book on data structures, and Chapter 7 on I/O. If you don't use Lisp (or, presumably, Java), beware that the memory requirements for the big data sets may require careful memory management.

2 Software

Your algorithm must:

- Use the information gain splitting rule. This is an information theoretic metric that selects the feature to split on based on the gain in information (measured in bits) obtained by that split. In particular, if T is the set of training data (containing $|T|$ instances) representing a concept with k classes, and $\mathit{freq}(C_j, T)$ is the frequency of class C_j occurring in that set, then the expected information needed to identify a given class in T is:

$$\mathit{info}(T) = -\sum_{j=1}^{k} \frac{\mathit{freq}(C_j, T)}{|T|} \times \log_2 \left(\frac{\mathit{freq}(C_j, T)}{|T|} \right)$$

  bits.

  For example, in this assignment you will look at hospital patient data for diagnosing hypothyroid conditions. Here $k = 2$ (two classes, hypothyroid and not-hypothyroid), so class 1 (C_1) would be for example the cases with hypothyroids and class 2 (C_2) would be the cases without hypothyroids.

  When a feature, X, with n values, has been selected as a test feature, then the expected information needed to identify a class under that test is:

$$\mathit{info}_X(T) = \sum_{i=1}^{n} \frac{|T_i|}{|T|} \times \mathit{info}(T_i)$$

where T_i is the subset of T all of whose instances possess value i for feature X.

For example, in the hypothyroid problem there are 23 features (age, sex, on-thyroxine, etc.) These features are CONTINUOUS. We have already coded the boolean features into "0/1" so you can assume that all features are continuous. You should be aware that testing continuous numeric values in a decision tree involves partitioning a continuous variable into a Boolean range attribute ($X > a_i$), where a_i is a feature value in the observed range for X. To accomplish this in your code, it will be necessary to sort the instances on a feature value in order to consider split points falling between each pair of data values. You must, however, retain the relationship between the feature values and the associated class. There's a code example later in this document. In any case, then, for this assignment, the splits will always be BOOLEAN. Thus $n = 2$ in the formula above. You need to consider *all possible boolean split points for all possible features.* Unlike the simple algorithm in the book, then, you cannot discard a feature once you have split on it (instead, you might have to split on that feature—at a different split point—again later in the tree.

The information gain, then, is simply the difference between the expected information needed to identify a class with and without the test on feature X:

$$gain(T) = info(T) - info_X(T)$$

Thus, the feature yielding the highest information gain is selected as the current split. Note that when splitting at a node of the tree other than the root, the set of instances under examination, T, will be only a subset of the whole set of training instances (in particular, it is the subset of training instances that have propagated down through the nodes of the tree that are parents of the node under consideration).

As an example of how you might undertake this, however, here are code fragments implementing how to sort on a feature value in LISP (using the Common LISP function `sort`):

LISP:

```
(defstruct fvstruct
  (feature-vals nil)
  (class-val nil))

(defun get-nth-feature-val (n struct)
   (nth n (fvstruct-feature-vals struct)))

(defun number-of-features (struct)
  "Compute the number of features, i.e. the length of the FV slot"
  (length (fvstruct-feature-vals struct)))

(defun same-class-p (fvl)
  "takes an fvstruct list (fvl) and sees if they all have the same class"
  ;; First, grab the class of the first fvstruct just once (efficiency hack)
```

```
  (let ((test-class (fvstruct-class-val (first fvl))))
    ;; Now, see if anyone is different from the first one (find-if-not eql)
    ;; If we can't find any who are not equal, then this will return NIL
    ;; and so our same-class-p function will return T.
    (not (find-if-not #'(lambda (class) (eql class test-class))
                      (cdr fvl)
                      :key #'fvstruct-class-val))))

(defun sort-on-feature (n fvl)
  "Common Lisp SORT is fast, efficient, and is DESTRUCTIVE!!!
   (i.e. the linked list fvl WILL BE MODIFIED). Hence you should
   at least SETF the result (setf fvl (sort-on-feature 4 fvl))
   and if you care at all about keeping your original list order,
   then you should make a copy first (setf fvlcopy (copy-list fvl))."
  (sort fvl
        #'(lambda (s1 s2)
            (< (get-nth-feature-val n s1)
               (get-nth-feature-val n s2)))))

(defun read-data-file (filename)
  "This function reads in a data file and returns a list of fvstructs."
  (with-open-file (stream filename :DIRECTION :INPUT)
    (loop for item = (read stream nil nil)
        until (null item)
        ;; do (format t "~&LINE: ~A" item)      ;debugging only!!!
        collect (make-fvstruct :feature-vals (butlast item)
                               :class-val (first (last item))))))

(defun frequency (class fvl)
  (loop for fv in fvl
        counting (eql class (fvstruct-class-val fv))))

(defun find-majority-class (fvl cl)
  (let ((best-class NIL)
        (max-frequency 0))
    (loop for class in cl
          for freq = (frequency class fvl)
          do (when (> freq max-frequency)
               (setf best-class class)
               (setf max-frequency freq)))
    (values best-class)
    ))

(defun majority-class-accuracy (fvl cl)
  (/ (frequency (find-majority-class fvl cl) fvl)
     (length fvl))
  )
```

You are free to construct whatever user interface for your program you like, but you must *fully document* your interface. This code, and some other stuff, is in the file learning.lisp in the data set directory for this assignment (see the pathnames given later in the document).

3 Data

You'll be examining the behavior of your DT algorithm on four data sets (actually, it's just variations on two). The data sets are all represented in a standard format, consisting of four files per data set. The first file, `<dataset>.names`, describes the categories and features of the dataset, and is formatted as follows:

```
(cat1 cat2 cat3 ... catn)
feature1
feature2
        .
        .
featurek
```

Where `cat1 ... catn` are the n names of the output categories, and `feature1 ... featurek` are the names of the input features. For this assignment, all features will be numeric-valued, taking on continuous values.

The other files of each data set, `<dataset>.data` (this is the training data), `<dataset>.prune`, and `<dataset>.test`, contain the actual data instances, formatted at one instance per line, as follows:

```
(i1fv1 i1fv2 ... i1fvk i1cat)
(i2fv1 i2fv2 ... i2fvk i2cat)
                .
                .
                .
(iNfv1 iNfv2 ... iNfvk iNcat)
```

Where `i1fv1` represents the value of the first feature for the first instance, and `i1cat` is the category value of the first instance. You can assume that there will be no missing feature values.

For each problem we have provided training, pruning and test sets. Training is 60% of the data, pruning is 20% of the data and testing is 20% of the data.

The data sets you will be examining are:

BALLOONS These are four very very small files you can use to test your program on. They are simple enough that you will know the correct answer (see balloon.info) and each feature is binary and there are only two classes. There are no test or pruning sets, you may make some up.

HYPOTHYROID This is a medical database representing the classification of 2278 patients into those with and without hypothyroid conditions.

There are two different versions of this data set. One in hypothyroid and the other in hypothyroid2.

REMOTE This is a satellite remote-sensing database representing the categorization of locations by the various types of ground cover. The features represent indices of "greenness" for a given location at different times during the year. The training data for this set has been further partitioned into `remote_10 ... remote_60` each representing a fraction of the available training data. This will allow you to generate a "happy graph" (hopefully) demonstrating that your algorithm does, in fact, learn the concept better as it is presented with more training data.

REMOTE2 This is the same data set as REMOTE, but with the middle six features removed.

REMOTE3 This is also the same data as REMOTE, but this time with every other feature removed.

Each data set (and a complete Lisp solution "missing" some of the definitions :-) can be found in a subdirectory of [on CIS] `~decker/courses/681/data/` or [on the Composers] `~decker/Class/CISC681/data/`.

4 Your Mission...

You will need to hand in both a writeup report and mail code to the TA. Deliverables for this project are:

- Code to implement the decision tree learning algorithm for the data file formats given above.
- A README file, with simple, clear instructions on how to run your code.
- Testing statistics for the application of your learning algorithm to each of the provided data sets. At a minimum you should provide training set accuracy, test set accuracy, size of the decision tree. If you prune (extra credit, see below), you should provide these numbers for trees that were pruned and for trees that were not pruned.
- A report analyzing the behavior of your algorithm on each data set, including any unusual or anomalous (in your opinion) behavior. In the report you should answer the following questions:

General Questions [10 points]:

1. For each of the data sets was the decision tree algorithm able to memorize the training data (i.e., get 100% accuracy when used to classify the training data)? If not, can you give at least one reason that you think it was not able to?
2. Could you use your system to, say, buy stocks or bet on horses? Why or why not?

Balloon [30 points]:

1. Demonstrate your program on these small data sets. Draw out the trees for the 4 balloon examples. Note that there are no testing or pruning files—we assume your tree will be perfect in these very simple examples.

Hypothyroid [30 points]:

1. What is the majority class accuracy for each of the two test sets?
2. How does you tree's test accuracy compare to the majority accuracy?
3. What does the top 2 levels (the root and the level just below) look like?

Remote [30 points]:

1. Generate graphs for each of the three version of remote that show how having more training data affects: test set accuracy, training set accuracy, and tree size.
2. The test set of accuracy of remote2 and remote3 should be lower than for remote. Can you explain why this is so? Should this be true in general for decision trees applied to datasets with a reduced set of features?
3. Which of remote2 or remote3 gets higher test accuracy? Since they test exactly the same number of features, please explain what you think the reason for this is.
4. Even for remote, the accuracy is far below 100%. Give at least one reason that you think precludes obtaining higher accuracy with this dataset.

- **EXTRA CREDIT [20 points]:** Use reduced-error pruning. This is a pruning criterion that replaces a subtree with a single leaf when that leaf (which classifies the instance as the majority class of all instances seen at that leaf during pruning) represents a lower error rate than does the subtree. The pruning procedure thus has the following steps:
  - Grow the tree fully on the training data.
  - In a postorder recursive fashion traverse the decision tree, replacing a non-leaf node with a leaf node only when the error rate of the subtree node exceeds the error rate of the leaf node. Measure the error rate on a separate set of instances (called the pruning data).
- **EXTRA CREDIT [20 points]:** Explore the effect of Boosting or Bagging. Choose the data set for which you performed the WORST, and see if you can improve your results via either boosting or bagging.