Programming Assignment 2: Decision Trees

CS 436 – Artificial Intelligence – Fall 2008

Due: November 17, 2008

Abstract

For the second programming assignment, you will implement and compare several different algorithms for constructing decision trees. You will implement both a traditional decision tree learning algorithm and an evolutionary algorithm. You will then train both of these algorithms on a training set, and evaluate their performance on a testing set, using several data sets from the UCI Machine Learning Repository [1] representing different types of classification problems.

The writeup for this assignment will consist of a conference-style paper reporting on your findings. For this assignment, you will be provided with a template which will demonstrate the appropriate layout, and give suggestions and examples for what should go in each section.

This assignment is individual work; you are not allowed to collaborate with anyone on this assignment in any way. All submitted code must be written by you. You may only discuss this assignment with the instructor. Any references used while working on this assignment must be properly cited.

1 Traditional Algorithms for Decision Tree Learning

The standard method for decision tree learning is to evaluate all the possible ways to split the data, choose one based on some fitness criterion, and then repeat on each of the split populations. This is described in Section 18.3 of the text (Russell & Norvig, p. 653). The most commonly used selection criterion is known as information gain or entropy minimization (the two are equivalent). This technique is described on pages 659–660 of the text.

2 Evolutionary Algorithms for Decision Tree Learning

One alternative to learning a decision tree iteratively is to use an evolutionary technique to find a decision tree that, taken as a whole, performs well according to some fitness function. This has the advantage of allowing the fitness function to be applied globally to the entire tree, rather than locally and independently at each node. This has the potential to result in better overall performance. A discussion of genetic algorithms is given in section 4.3 of the text (Russell & Norvig, p. 116).

The key issues to think about for a GA are encoding and fitness evaluation. The fitness function must properly reward "good" individuals, and it must do so even when the performance of all individuals is poor on an absolute scale. This is because, near the beginning of the search no individual is going to do well, and your fitness function still needs to be able to drive the search towards more productive parts of the search space.

The selection of a fitness function is closely tied to the selection of an encoding. The encoding must not only be able to represent solutions to the problem in a way that can be evaluated by the fitness function, but it must also be designed so that mutation and crossover are actually useful. For example, poorly designed encodings often have the problem that one or both of mutation and crossover are likely to produce encodings that cannot be mapped to valid solutions to the problem (e.g., for the travelling salesman problem, if mutation allowed for the generation of a "solution" that did not visit each city exactly once, it would be generating genomes that did not encode a valid solution to the TSP). Therefore, your encoding should be designed so that only valid solutions can be generated. Each genome must encode an "individual" that represents a valid decision tree.

3 Program Requirements

3.1 Submission Requirements

You are welcome to develop your programs on any computer, using any "traditional" language. All code should be well structured and well commented. "Self-documenting" coding practices are encouraged, but are almost never sufficient to replace comments entirely.

Your program should be submitted as an archive, either a zip file or a tarball, with the filename consisting of your first and last name and the assignment name, delimited by underscores, all lower case. For example, '`jane_doe_pa2.tar.gz`'. You should submit this archive via email to the instructor at `john.sheppard@cs.montana.edu`.

Your archive should contain any and all files required to build and run your program. It should *not* include the provided data set(s), but it *should* include any data sets used by your program that were identified as part of the assignment. It should also contain a README file, in which you list the files in your archive and a brief description of what each file contains. Your README should also contain any relevant notes on the functioning of your program, such as known bugs, parameters that result in excessive runtime or poor behavior, etc.

Your archive should also contain a `./data/` subdirectory which should contain output from sample runs, performance information, and other information as requested.

3.2 Feature Requirements

3.2.1 Input/Output Requirements

Your program should allow specification of data set file names at runtime. It should output classification accuracy, precision, and recall for both the training set and the test set. Recall is the number of true positives divided by the total number of positive examples. Precision is the number of true positives divided by the sum of the number of true positives and the

number of false positives. The minimal data sets you will use are discussed below. Accuracy is the percentage of the examples classified correctly.

3.2.2 Traditional Decision Tree Requirements

Your program must learn a decision tree from the given training data using the decision tree learning algorithm described in the text. It should use maximum information gain as its selection criterion. It must then evaluate the learned tree on the test set. You must also have a version that uses gain ratio to deal with data sets that have multivariate features (see R&N p. 663). You are not required to implement pruning in this assignment.

3.2.3 Evolutionary Decision Tree Requirements

Your evolutionary program must learn a decision tree from the given training data using a process similar to a genetic algorithm. The choice of encoding and fitness function are yours, and you will be expected to describe and justify your choices in the writeup.

Due to the nature of the problem, you will most likely need to use a variable length encoding, rather than the fixed-length encodings discussed in the book. In addition, it is unlikely that a binary encoding will work well. The basic algorithm is unchanged, but your fitness function must be able to process whatever encoding you decide to use. Additionally, you will need to modify the mutation and crossover functions so that they make sense with your encoding. Mutation, for example, should have a chance not only to change a unit of the encoding, but also to add or delete a unit. Crossover should also handle variable length encodings. You are required to implement one-point crossover, but not multi-point crossover. Remember to ensure that mutation and crossover produce valid encodings (eg. encodings that can be translated into a decision tree).

Be sure to experiment with different values for the tunable parameters, like population size, replacement policy, etc. Record the results of these experiments, as you will have to justify the final parameter choices you make in your writeup. Perform experiments with at least two different selection schemes (eg. fitness proportionate, tournament selection, rank based, etc.).

3.2.4 Experimental Requirements

For each learning algorithm and data-set, you should perform an experiment in which the algorithm is trained on a training set, and evaluated on a test set. Under no circumstances should you train any algorithm using data from the test set; the test set should be used only after all training has finished and the decision tree is fixed. Record the precision, recall, and accuracy of the different algorithmic variations. Also make note of how long the training took; precise time taken is generally not an accurate way of comparing algorithms, because of external factors (e.g., CPU power, other users/processes, etc.). Therefore, you should use some other measures of speed such as the number of generations needed to reach good

convergence in a GA (population size and other parameters are also relevant), and total number of "next decision" choices evaluated by the traditional learning method. The results of your experiments will be discussed in detail in your writeup, but you should include a short text file describing the performance of your algorithms with your program submission.

The data sets described below can be obtained from the University of California, Irvine (UCI) machine learning repository. After downloading the data files, you may want to do some pre-processing before using them (e.g., replacing string class labels with numeric ones). If you do so, be sure to explain what pre-processing you did and how you did it.

You are also responsible for deciding how to split the data sets into testing and training sets (and a validation set, if your program requires one). You will need to run at least 10 trials for each configuration, so you might want to consider either cross-validation or multiple hold out trials. You will have to explain and justify this choice in your writeup. In addition to comparing the resulting scores of your different algorithms, perform statistical significance tests on your results (for example, using the Student's t-test). Report how significant the differences between your algorithms are in the text file describing your algorithms' performance.

3.3 Data Descriptions

The data for this assignment comes from the UCI Machine Learning Repository, `http://www.ics.uci.edu/~mlearn/MLRepository.html`. Several data sets have been chosen for you for testing your algorithms, but you should download the data sets yourself from the UCI repository [1].

3.3.1 Congressional Voting Data

`http://archive.ics.uci.edu/ml/datasets/Congressional+Voting+Records`

The first data set is a binary classification problem: classify members of Congress as Democrats or Republicans based on their voting record (for a particular period). For a full description of the data, along with the original data files, see the README that comes with the data set. For your task, however, the true meaning of the data is more or less irrelevant.

This data set was chosen for several reasons. First, the UCI database is a good place to get machine learning data to test algorithms because you can compare your results to those others have achieved with the same data. Second, this data set was chosen because it was small enough (both in terms of number of features per data point and number of data points in the set) to be computationally manageable, allowing you to test and refine your code quickly. Third, the attributes all represent the same information, and can all take on exactly 3 values. Most of the UCI data sets have more complex data, including integers and/or real numbers mixed with multivariate discreet variables. While it is possible to build decision trees that can handle this type of data, it is beyond the scope of this assignment.

Note that for this data set, you should interpret the '?' as being a third value that each feature can take on, rather than being missing data. This interpretation is consistent

with the meaning of the underlying data, in which the '?' indicates that no opinion was recorded (due to a "present" vote, for example). The three values are essentially 'yes,' 'no,' and 'abstain.'

3.3.2 MONK's Problems Data

`http://archive.ics.uci.edu/ml/datasets/MONK%27s+Problems`

The MONK's Problem data set comes from a machine learning competition, and therefore is a good benchmark for your algorithms. This data set is a bit more challenging than the previous one, because some features can take on more values than others. In addition, notice that there are actually three different problems. All features are still discrete and take on a relatively small number of values. Additionally, the number of features is small. The fact that different variables take on different numbers of possible values will have important consequences for the Information Gain heuristic, as described in the text (p. 663). You should modify your algorithm to account for this by using the ratio of information gain to the number of values the variable can take on. Record the performance of your algorithm with and without this modification. For these data sets, you should combine the training and test sets and then apply the same experimental method for the other problems (e.g., multiple hold-outs or cross-validation).

3.3.3 Mushroom Data

`http://archive.ics.uci.edu/ml/datasets/Mushroom`

This data set contains information about the physical properties of mushroom specimens; the goal is to classify each mushroom as being safe to eat or unsafe to eat. This data is again more challenging, because some of the features can take on a wide range of values. All variables are still discrete, however, and there is no missing data (one feature was removed from the original data set to achieve this; all other features were always present). Again, run your algorithm with pure information gain, and with gain ratio.

3.3.4 Splice Junction

`http://archive.ics.uci.edu/ml/datasets/Molecular+Biology+%28Splice-junction+Gene+Sequences%29`

This data set contains data from a molecular biology problem. The goal is to recognize boundaries between introns and exons (portions of the genome which do and do no code for proteins, respectively). This data set is larger and has a larger number of features than the others.

4 Writeup Requirements

While this is a computer science course, writing is emphasized because it is an important part of being a good scientist or engineer. Having brilliant ideas is useless if you are unable to communicate those ideas to others in the scientific community. For this assignment, your writeup will take the form of a scientific paper. A template will be provided for you to use. The template will include the expected formatting, as well as some suggestions and examples of what kind of content you should include. The written assignment will be graded on proper formatting, quality of writing, and quality of content.

The template will be available in LaTeX source and as a PDF generated from that source. If you use LaTeX for your report, you are welcome to use the source directly; otherwise, it is your responsibility to create a document that fits the template using the program of your choice. As before, the writeup you turn in should be a PDF.

The template itself will be posted separately, and will contain more detailed information on what is expected in the writeup.

References

[1] Asuncion, A. & Newman, D.J. (2007). UCI Machine Learning Repository [http://www.ics.uci.edu/∼mlearn/MLRepository.html]. Irvine, CA: University of California, School of Information and Computer Science.