Midterm Project (Homework 4): Decision Trees

Artificial Intelligence, Spring 2015

Ben Mitchell

program due dates: checkpoint, 11:00pm, March 25th;
final version, 11:00pm, April 1st
paper due dates: draft (partial), 11:00pm, April 3rd;
final version, 11:00pm, April 10th

Abstract

This assignment will be submitted in several stages; take careful note of what is due when, and be sure you submit each part complete and on time.

The fourth homework assignment will be a midterm project. For this assignment, you will implement and compare several different algorithms for constructing decision trees. You will implement both a traditional (information gain based) decision tree learning algorithm and an evolutionary (GA based) algorithm. You will then train both of these algorithms on a training set, and evaluate their performance on a testing set, for several data sets from the UCI machine learning repository[1] representing different types of classification problems.

The writeup for this assignment will be submitted and graded separately from the programming portion. It will consist of a conference-paper style report 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. The template will be provided as both a PDF and a set of LaTeX source files used to generate the PDF. The writeup for this assignment is ***required*** to be written in LaTeX. All submissions must include LaTeX source as well as finished PDFs.

1 Instructions

All work must be submitted through Blackboard to be considered for grading.

All portions of this assignment are to be completed by all students unless otherwise noted. Requirements that are specific to students enrolled in 600.435 will be specified in the form ***435 only:*** *implement feature x.* Students enrolled in 600.335 are not required to complete these portions of the assignment, and will not recieve extra credit for doing so. If you have questions or concerns about any of the requirements, please contact the TA or the instructor

as soon as possible. Note that the 435 portions of the assignment are interleaved with the 335 portions; do not assume that there are no more 335 portions after the first 435 portion.

1.1 Submission Contents

This homework consists of two main parts, a programming portion and a written portion. Each of these is further broken down into two parts. As the portions have separate due dates, you will be expected to submit them separately. They will also be graded separately. Both the program and the writeup should be submitted as tarballs through Blackboard. When extracted, your program tarball should produce the following directory structure. All your submitted work should be in a top-level directory named `'<lastname>.<firstname>/'`, with your name inserted, all lower case (for example, `mitchell.ben/`). That directory should contain the following subdirectories: `src/, doc/, data/, output/, [bin/], [include/], [lib/]`. The directories in brackets are optional; use them if appropriate for the language and code structure you have chosen. The proper use of these directories is described below.

`src/` should contain all your source code

`doc/` should contain all documentation and source for documentation (if applicable; eg. JavaDocs, *.tex files).

`data/` should contain all data, problem instances, or other files that are read as input by your program.

`output/` should contain sample runs showing the behavior of your program.

[`bin/`] should contain all binary executable files (it should be empty in your tarball; don't include binaries, just make sure they get put in `bin/` when they get built).

[`include/`] should contain header files (eg. if you are using C/C++, you may wish to have .h files in `include/` and .c files in `src/`).

[`lib/`] should contain any libraries your code requires to link/run. You are not expected to generate your own libraries (eg. libfoo.a) as a part of the assignment, though you may do so if you wish. Any outside libraries used must by properly cited in your README.

The top level directory should also contain an ASCII text file named `'README'`. Your README should contain your name, the name of the course, and the title of the assignment (which is also the title of this document). It should include a listing of the files in your submission, with a very brief description of what each file contains. It should describe the structure and organization of your code, as well as describing in detail how to build and run your program (ie. what arguments does it take/expect, what is the interface if the program is interactive, how to make your program use a particular problem instance, etc.). It should also list approximate runtimes for all the problem instances specified in the assignment. This is especially important for this assignment, as your writeup will not be submitted until after your program.

Your README should also have a section entitled "Reflections" at the end, which should contain a paragraph or two of text describing your thoughts about the assignment. It should contain an estimate of how much time you spent on the assignment, along with your thoughts about things like what parts you liked or didn't like, what parts you found particularly hard or easy, what you felt like you learned, or how you would change the assignment if you were in charge of creating assignments for the class. You don't have to limit yourself to these things, nor do you need to discuss every single one of them for every program; pick the ones that have interesting examples (but be sure to always include how long the homework took you to complete).

The "sample program runs" in the `output/` directory should be transcripts of your program's output. It should be clear from the output which problem instance was used for a given run. For most assignments, it should be sufficient to copy and paste the text from the terminal in which you ran your program to a text editor. Alternatively, you may wish to redirect `stdout` and/or `stderr` to a file.

1.2 Program Requirements

For the programming portion, all code must build and run on the ugrad linux machines, and must be buildable/runnable using command line tools. Where and how you develop are up to you so long as the code you turn in satisfies these requirements. You should include three shell scripts, named `compile.sh` and `run_`X`.sh` that compile and run your program, respectively, where X is one of '`traditional`' or '`evolutionary`'. If you are using a language that does not require compilation, the `compile.sh` script can simply be a placeholder that does something like ``` `echo "No compilation required"` ```, but it must still be included in your submission. These scripts should be in the top level directory. If you are unfamiliar with shellscripting, there are many resources online, and you are welcome as always to ask questions of the TA or the instructor. Additionally, you are free to discuss scripting issues on Piazza. These scripts are not considered to be a part of your implementation of an algorithm, and therefore you may discuss them in any detail you wish, including posting code or examples (if you are unsure if a given post is appropriate, send it to the instructor or TA as a private message, and we will let you know).

Your archive should contain any and all files required to build and run your program which are not standard on the ugrad network. It it ***should*** include any data sets used by your program, including those that were provided to you with the assignment, and any data sets that you modified from the versions provided.

Code will be graded based on style as well as correctness. You are expected to write well documented, well structured, readable code. The industry best practices for the language you are using is a good reference. The exact specification of what is desirable will depend on the language you are using, but in addition to good documentation, code should have good modular design and be written in a way that makes reading and understanding what it is doing as easy as possible. Note that we are not looking for state-of-the-art level accuracies from your algorithms, but this does not mean that you should not make reasonable effort to achieve good performance.

Code will also be graded based on efficiency of implementation. This does not mean that you are expected to write heavily optimized systems-level code, but rather that your implementations should be reasonably efficient. If your implementation of an $O(n)$ algorithm takes $O(2^n)$ time to run, for example, it will be penalized as a poor implementation of that algorithm. You will not be penalized for your language choice except to the extent that if your program is unable to solve the given problems before the assignment is due, you will be unable to get full credit for having completed those problems (since there is no way to prove that the answer would have been correct, and you will be unable to answer the written problems related to those solutions).

Any generated documentation related to code (eg. JavaDoc, doxygen, etc.) should be located in your `doc/` subdirectory, and described in your README.

1.3 Writeup Requirements

Your writeup for this assignment should be submitted separately from your program, as it has a different due date. Your writeup submission should consist of a tarball, which should have the standard '`<lastname>.<firstname>/`' directory; in this case it should contain only the `doc/` sub-directory (you should put your document source in the `doc/` subdirectory of your main project for this assignment in your local copy, and then just tar up only the `doc/` portion to submit it).

For this assignment, you are *required* to use LaTeX to generate your writeup. Your `doc/` directory should contain the LaTeX source files required to generate your writeup document, as well as a PDF document created from those source files. There will be a brief treatment of LaTeX in class, and a more thorough tutorial given in a section, which will take place at a time and place which will be announced. Additionally, a LaTeX template for the assignment will be distributed. You should start with this template when doing your assignment, and modify it with your own text. See the README included with the template for details about how to build the template PDF from the template source.

Links to additional information about LaTeX can be found on the instructor's website, http://cs.jhu.edu/~ben/latex/.

For this assignment, your writeup should be in the form of a scientific paper. This means that, rather than simply answering some questions, you will be expected to generate a report that describes, among other things, what experiments you did, what results they produced, and what you think can be concluded from those results. While the expectations for this class are lower than those for a peer-reviewed publication, the point of this exercise is to learn how to write for such publications. Therefore, it is expected that your writeup will have the correct form and tone for a scientific paper, be clearly and concisely written, and use correct English grammar and spelling.

The template document should help to guide you in making decisions about what is expected. Additionally, there will be an in-class discussion of scientific papers, and how to write them.

All students, though particularly those for whom English is not their first language, are

strongly encouraged to make use of JHU's writing resources, including the Writing Center (http://carey.jhu.edu/students/academic_resources/writing_center/), to get help and advice on your writing. Not only are you likely to get good advice, but if you give them the instructor's e-mail address, they will send an e-mail to him stating that you came in, and what you discussed. While you will not get points merely for going, it is likely that we will grade more leniently if it is clear that you sought help at the writing center, and followed the advice you were given.

While this is a science course, writing is emphasized because it is an important part of being a good scientist. Having brilliant ideas is useless if you are unable to communicate those ideas to others in the scientific community. Our department does not offer a course explicitly on scientific writing, so this is a chance to practice your writing skills and get feedback on your progress.

As this will be the first paper-writing experience for many students, we will grade generously, but give copious comments and suggestions. For the final project paper, we will have higher expectations, and grade accordingly. Thus, a paper that gets a good grade as a submission for the first paper might not get a good grade if it were graded as a submission for the second paper. Read the comments on your paper, and try to make the changes they suggest in your later papers.

You will also be required to turn in a draft version of a few key sections of your paper; this will give you a chance to get feedback and make improvements before you turn in your paper.

The template will be available separately from this document on the course website assignment page. More detailed instructions on writeup requirements will be included in the template.

2 Decision Tree Learning Algorithms

2.1 Traditional Algorithms for Decision Tree Learning

The standard method for decision tree learning is to evaluate all the possible ways to split the data, greedily 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 of the 2nd edition, and p. 697 of the 3rd edition). The most commonly used selection criterion is known as information gain or entropy minimization (the two are equivalent). This technique is described in the text on pages 659-660 in the second edition and on pages 703-704 in the third edition. There are also a number of good online references; you are free to use such references, so long as you cite them properly.

2.2 Evolutionary Algorithms for Decision Tree Learning

One alternative to learning a decision tree iteratively from the top down 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 2nd edition, and section 4.14 of the 3rd edition, though the book's coverage of this topic is somewhat lacking. Again, many good resources are available online, and you are free to use them so long as they are cited properly.

The key issues to think about for a GA are encoding, and fitness function. 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 by providing a useful gradient.

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 (eg. 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.

You are encouraged to use an encoding resembling the one that is described as follows. You are not required to do so, as the choice of encoding is up to you, but the following encoding is a standard one for learning tree-based structures with GAs.

The basic idea behind a tree-structured encoding is to directly encode and manipulate the tree. The tree will consist of a root node, which will have a set of nodes as children; these may be leaf nodes, or they may have their own children. Because we are dealing with decision trees, each node will have a decision variable associated with it, and each child will have a decision outcome associated with it. There should be one child per possible outcome (i.e. possible value the associated variable can take on); be sure that there are not outcomes that don't lead to children, or children that are not associated with a unique outcome.

This will be a variable length encoding, as the length of the encoding will depend on the depth/branching factor of the tree. Crossover using this encoding is fairly straightforward; simply select one node from each tree, and swap those nodes (and their sub-trees). The fact that entire sub-trees can be swapped allows for large changes in behavior, but can still preserve good behaviors generated by a sub-tree.

Mutation is a bit less straightforward; there are essentially two types of mutation that

can be performed in this encoding. The first is to randomly change the decision variable assigned to a node. So long as the new decision variable has the same number of outcomes as the old one, the tree will still be well-formed. The second type of mutation is to randomly add or delete a node. Adding a node only really makes sense at the leaves; a leaf node can be turned into a decision node by associating it with a decision variable and creating a set of children. Node deletion is easiest to implement as simply deleting the entire subtree below a node, and making that node into a leaf node; however, this can wind up pruning rather agressively if it happens high up in the tree, so you may want to restrict it to nodes close to a leaf. Inserting or removing arbitrary nodes does not make sense, because the number of children that any given node can have is exactly fixed by the number of allowed outcomes of the decision variable with which it is associated. Again, caution must be used to ensure that the new encoding generated by reproduction is still a valid decision tree. You can also use both of these mutation operators. Keep in mind that while you want *valid* solutions, they don't have to be *good* solutions. For example, if the same attribute is used as the decision criterion twice in a given path through the tree, that's fine. If the number of children doesn't match the number of values of the associated attribute, that's a problem. For things that are valid but not desireable, you can use the fitness function to discourage them; a small penalty based on tree size, for example, might help discourage redundant decisions.

This type of tree-structured encoding is commonly used in Genetic Programming, particularly in conjunction with languages like Lisp that are inherently tree-structured in their execution. This encoding is well suited to learning decision trees. A fixed-length bitstring encoding, while possible, would likely be much more difficult to implement and evolve successfully.

Some suggested things to experiment with are as follows:

Fitness function Try different fitness functions. What happens if you use precision? What about recall? What if you use some combination of the two? What happens if, rather than evaluating on the entire train set each time, each time the fitness function is called it random chooses a sub-set of the traning vectors to evaluate on? This should speed things up, but what effect does it have on performance? Does the ability to run for more generations outweigh the loss of accuracy in the fitness function?

Population size What is the optimal population size? How does the performance of your GA vary if you use a population of 10? 100? 1000?

Selection strategy What is the best way to select individuals for reproduction? Fitness-proportionate is the most commonly used, but has been known lead to early convergence in some problems. Does rank-based selection work better? If the population is large, these may be time-consuming; does a tournament-based combined fitness function/selection strategy offer better performance?

Mutation/Crossover probability What effect do different values of these probabilities have? Note that standard GAs often use a very low probability of mutation (usually only a few percent; sometimes less than 1), and a crossover probability on the order of

30%-80%. These values are highly encoding specific, so you these "standard" values may or may not be good choices for your GA.

Replacement strategy What replacement strategy is the most effective? Does some form of elitism improve performance, or do you get the best results with complete replacement? Be sure to try a non-elitist version.

Termination criterion How long do you need to run your GA? How quickly does it seem to converge? Does the performance still increase after 100 generations? 1,000? 10,000?

3 Programming Assignment

For this assignment, your program must learn decision trees from data. You must implement both a traditional decision tree learning method and an evolutionary decision tree learning method. Details for each are given below.

For the checkpoint, you need to fully implement the traditional, information-gain based learning algorithms, along with the basic framework required to run and test them. While this submission will not be graded with the same attention to detail as a full assignment submission, we will be checking to see if you have implemented what was required, and whether it works or not. It will still be worth points (if not as many as a full assignment), and if you don't get it done on time you will also likely have difficulty finishing the rest of the assignment before it is due, so do your best to treat it as a regular assignment submission deadline.

Be sure to include your README, your sample runs, and everything else you would include in any other program submission.

For the final version, you need to also fully implement the GA-based evolutionary learning algorithms. Once again, be sure your README, sample runs, and all other files are up to date and include material relevant to both parts of the assignment.

3.1 Feature Requirements

3.1.1 Input/Output Requirements

Your program should allow specification of data set file names at runtime as a command line argument. It should output classification accuracy, precision, and recall for both the train 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. Accuracy is the percentage of correct predictions, or the number of true positives plus the number of true negatives divided by the total size of the data set. The data sets you will use are discussed below.

3.1.2 Part A: Traditional Decision Tree Requirements

Your program must learn a decision tree from the given training data using the `Decision-Tree-Learn` 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).

CS 435 Only *You must also implement a version of your program that does pruning, as described in the text (p.662), to combat overfitting. You should provide two versions of your program, one with pruning, one without, and perform experiments with both to compare their performance.*

3.1.3 PartB: Evolutionary Decision Tree Requirements

Your evolutionary program must learn a decision tree from the given training data using 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.

See the earlier section for some suggestions; 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.

CS 435 Only *Perform experiments with at least two different selection schemes (eg. fitness proportionate, tournament selection, rank based, etc.). Perform experiments with at least two different replacement methods (eg. complete replacement vs. elitism)*

3.1.4 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 (eg. CPU power, other users/processes, etc.), but relative timing can still be interesting to observe. Other useful measures of speed are 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 details 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 are provided in the form available from the University of California, Irvine (UCI) machine learning database. You may want to do some pre-processing before using them (eg. replacing string class labels with numeric ones). If you do so, provide both the scripts used to do the processing and their output with your submission.

You are also responsible for deciding how to split the data sets into testing and training sets (and a development set, if your program requires one). You will have to explain and justify this choice in your writeup.

Any scripts or programs used for pre-processing or splitting the data should be included in your submission, and their use and purpose should be described in your README.

CS 435 Only *In addition to comparing accuracy scores of your different algorithms, perform statistical significance tests on your results. Report how significant the differences between your algorithms are in the text file describing your algorithms' performance.*

3.2 Data Descriptions

The data for this assignment comes from the UCI machine learning database, http://www.ics.uci.edu/~mlearn/MLRepository.html. Several data sets have been chosen for you to test your algorithms on.

3.2.1 Congressional Voting Data

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 http://archive.ics.uci.edu/ml/datasets/Congressional+Voting+Records. For your task, however, the true meaning of the data is more or less irrelevant.

This data set was chosen for several reasons. Firstly, 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.'

3.2.2 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. Since they are designed to be hard problems, you are likely to see somewhat lower absolute performance on these problems; however, you should still try to get the best scores you can.

This data set is a bit more challenging than the previous one, because some features can take on more values than others. 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.

Note that there are several distinct data sets here; each represents a completely different problem. Do not combine these data sets, as the result of doing so is meaningless data. You should evaluate the performance of your algorithms on each MONK's problem separately.

3.2.3 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 challenging because some of the features can take on a wide range of values. All variables are still discrete, however, and you don't need to worry about missing data (one feature must be removed from the original data set to achieve this; all other features are always present).

Again, run your algorithm with pure information gain, and with gain ratio.

3.2.4 *CS 435 Only* - 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. Only the students in CS 435 need to report results on this data set.

4 Written Questions

There are no questions; your writeup will be a paper, as described above. The writeup template will be available separately from this document through Blackboard; it will contain more detailed instructions.

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.