EECS 349 (Machine Learning) Homework 2, Fall 2009

WHAT TO HAND IN

You are to submit the following things for this homework:

1. A PDF document containing answers to the homework questions.
2. The (**commented!**) source code for any software written to complete this assignment.
3. An executable for any software developed to complete this assignment (for Java, this means a .jar file, for MATLAB and Python, the source code = the executable).
4. A README.TXT file that gives the best parameters to run your program with, for the RedQueen dataset.

HOW TO HAND IT IN

To submit your lab:

1. Compress all of the files specified into a .zip file.
2. Name the file in the following manner, lastname_firstname_hw2.zip. For example, Pardo_Bryan_hw2.zip. (Please name the PDF similarly, Pardo_Bryan_hw2.pdf)
3. Submit this .zip file via email to: eecs349-hw@cs.northwestern.edu

Note: The subject line should read: "Homework 2"

DUE DATE: the start of class on Monday, Oct 26, 2009

Problem 1: AdaBoost (3 points)

The AdaBoost algorithm is described in the paper "A Brief Introduction to Boosting" by Robert Schapire. You have been provided this paper. This algorithm learns from a training set of input/output instances $\{(x_1, y_1), ..., (x_m, y_m)\}$ where X is an arbitrary set of inputs, $x_i \in X$ and $y_i \in \{-1, +1\}$. AdaBoost assumes the training set is fixed over all the rounds. Assume now that after each round, each pair in the training set has a probability q of reversing the sign of its label (the y term in the pair is the label).

A) (one point) How would the performance of AdaBoost be affected by a dataset that changes from round to round in this manner?

B) (one point) Assume that the number of rounds is fixed at some value T. Modify AdaBoost to improve its expected performance on the training set immediately after round T. Describe your modification in both text and mathematical notation.

C) (one point) What can you say about how you expect your modified AdaBoost to perform, compared to the unaltered AdaBoost? Can you give any bounds on how much better your version might do? If so, give them.

Problem 2: Make a distance measure (2 points)

A) (1 point) Define a distance measure between words composed only of lower-case alphabetic characters (strings drawn from the set {a,…..,z}). Your measure should somehow capture the relative likelihood of someone mistyping one string for another. For example, for someone using a QWERTY keyboard, the string "grog" should be closer to "frog" than "grog" is to "prog." Give your argument for why this distance measure DOES make words that are easier to mistype for each other also closer using the distance measure.

B) (1 point) Prove that your measure is or is not a metric.

Hints: Don't forget words may be of different lengths. As a starting point, you could look at Hamming Distance , Edit (Levenshtein) distance, or the band-quote distance measure described in the course lecture notes.

PROGRAMMING PROJECT

In your previous homework assignment, you used the ID3 algorithm to create a decision tree classifier. However, there are other ways to use training data to create a decision tree. In this assignment, we will explore how one might use a genetic algorithm to "evolve" a decision tree that can work as a basic game-playing AI. Here's the backstory:

There's a hot new online gambling game called RedQueen – a slightly modified variant of blackjack – that everyone's playing these days. Some of the world's best blackjack players have been thrown by it, since they are used to playing by the traditional rules, and this game's a little different. For one thing, they're not using a standard deck – you're pretty sure there are more aces than usual, and who knows how else the card distribution might be skewed. Also, there are jokers in the deck, which can count as either 0 or 5 points towards your score. Furthermore, the dealer is not playing by the traditional "keep hitting until 17 or Soft17" rules – in fact, nobody knows exactly how the dealer decides to stop taking cards.

You could strike it rich, if you could come up with an AI that can play the best possible game against this system. It's tricky though. You don't know exactly how the game works, what the odds of different cards are, nor what logic the dealer is using. However, you have managed to compile a large dataset of past games that have been played, and whether you should have chosen to "hit" or "stand" at each point in the game. The data is kind of noisy, but you're still hopeful that it can work, and you've decided to use a genetic algorithm to build a decision tree out of your data. Once you come up with something that works reasonably well on the data you have, you'll set it loose playing the online game, and fame and fortune will surely follow...

You will be given a set of (somewhat noisy) training data, which you will use to build, train, and tune a classifier. After you turn in your homework, your algorithm will be trained on the same dataset you were given, but its performance will be tested on a similar dataset with instances that have not been previously encountered.

Clarification: although you are evolving a decision tree that can function as a "game-playing AI", this homework will not be an "online" learning problem – your genetic algorithm will not interact directly with the RedQueen game itself, only with a pre-compiled dataset of old games.

EECS 349 (Machine Learning) Homework 2, Fall 2009

About RedQueen

You don't need to know the rules of RedQueen to do this homework problem, but they are given here in case you are interested.

THE CARD DECK

The game is played with the same cards that are found in a standard USA fifty-two playing card deck. (see here for an illustration: http://en.wikipedia.org/wiki/Playing_card#Anglo-American) However, the exact distribution of cards in the deck is unknown (there may be many more 5s than 6s, for example). IN ADDITION, there are several "jokers" in the deck, which are scored as shown below.

WHAT CARDS ARE WORTH

Each card with a number on it is worth its face value. Except for the Joker, each card with a face (King, Queen, Jack) is worth 10 points.

You may treat Aces as either 1 or 11 points (whichever serves you better).

You may treat Jokers as either 0 or 5 (whichever serves you better).

THE STARTING POINT

You start with two cards drawn from the deck, and your opponent (the dealer) also starts with two cards. You can see the value of both of your own cards, but only one of the dealer's two cards is visible to you.

HOW TO PLAY

You start the game by deciding to either "hit" (draw a card) or "stand" (stop with your current set of cards).

You want the sum of your cards to be larger than the sum of the dealer's cards, but if your sum goes over 21, you "bust" and lose.

If you haven't "busted", then after you have decided to "stand", the dealer may proceed to take several more cards. The rules by which the dealer decides whether or not to take cards are deterministic, and not based on the cards in your hand, but the dealer's rules are unknown to you.

You win if the dealer busts or if your final sum is higher than the dealer's final sum.

ADDITIONAL NOTES

RedQueen does not contain any of the Blackjack intricacies such as "doubling", "splitting", "insurance", and "surrendering", so don't worry about such things.

Input File Format

The input file format is almost identical to that of the data files from the previous homework assignment. An input file (either training or testing) is expected to be a tab-delimited ASCII file, where the first line is a list of *j* attribute names, followed by the classification label name (e.g. "ShouldHit?" for the RedQueen data file). Each subsequent line consists of a fixed number of tab-delimited strings, each line corresponding to a single example. The *i*th string in each line must contain the Boolean value for the *i*th attribute. Value *j*+1 will be a Boolean value that specifies how the target concept function would classify the example defined by these attribute values. You may assume that all attributes will be Boolean.

You have been given an example data file "RedQueen.csv" which contains instances from past RedQueen games, describing features of your and the dealer's hands. You may wish to split this file into separate training and testing sets, or you may test on the same dataset as you are training on. When we grade your assignment, we will be partially assessing the effectiveness of your design by training with the RedQueen.csv file that we have given you, but then testing it on a secret validation set containing similar data.

An example of the data format (with most of the 43 columns omitted) is shown below:

```
IhaveNoJokers   … minHandValue<9      … bestHandValue<13     … dealer<7      … ShouldHit?
True            … False               … True                 … False         … True
False           … False               … False                … True          … False
...
```

`minHandValue` *refers to the value of your hand if all aces are treated as 1s, and all jokers as 0s.*

`bestHandValue` *refers to the value of your hand that is closest to (but still less than) 21.*

`dealer` *refers to the value of the one card from the dealer's hand that you can see.*

Problem 3: GA Design (3 points)

You must design a genetic algorithm (GA) that evolves decision tree classifiers based on training data. To do this, there are a number of important design questions that must be addressed. Please describe your design choices using between 500 and 1000 words (in *total*, not per question). Illustrations and math formulae may also be useful.

- Define how a decision tree will be encoded (genotype). Traditionally GA's use a linear string of bits or linear vector of numbers for the genotype. However, if you wish to use a tree-based genotype structure – more akin to *genetic programming* than genetic algorithms -- that is acceptable, though you will need to specify appropriate genetic mutation and crossover operators (such as swapping subtrees).[1]
- Explain how your encoding maps into a decision tree. (How genotype maps to phenotype).
- Describe the hypothesis space enabled by this encoding. Compare the possible decision trees that can be represented by your encoding to decision trees created by the ID3 algorithm.
- Define a fitness function

    Explain EXACTLY how you will calculate fitness for a member of the population. This will presumably involve the provided training data somehow.

- Define how mutation works

  Precisely define your probability of mutation formula.

    What kinds of mutation operators will you use? Bit flips? Swapping of positions? Incrementing/decrementing? Gaussian-based mutation? Something else?

[1] *Students interested in prior research on this topic, might like to peruse this conference paper (published in 2000), which used a tree-based genotype structure to evolve decision trees, and reported several favorable results compared against C4.5.*
http://www.gatree.com/data/GATreICTAI00.pdf

- Define how your crossover method works

  One point? Multi-point? Uniform? Something else? Give an example.

- Define how selection works:

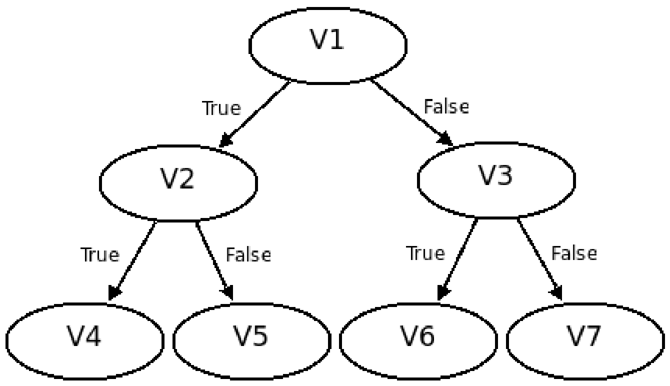
  How will you determine who reproduces?
  Roulette wheel/fitness proportional? Rank-based? Tournament-selection?

- Define the lifespan of population members

  Do parents survive to the next generation to compete with children?

- Define a termination condition

  When do you decide that the learner is finished? After a fixed number of generations? After a fixed percentage of the population reaches a certain fitness? When the diversity of the population decreases?



Because genotype-phenotype mapping can be a challenge, here is one possible encoding you may use if you wish, you may also design your own:

A decision tree of maximum depth D can be encoded as a fixed-length vector **V** (1-D array) of integers that is of length N, where N = (2^D – 1), by using a bit of clever array indexing.

V = $V_1 \ldots V_N$, where V*i* has child nodes V_{2i} and V_{2i+1}, and it's parent node is $V_{i/2}$ (if you use integer division). This results in a tree that looks like the figure shown at right.

Each node (V_i) can be assigned an integer value of 0 (“return classification=false”), 1 (“return classification=true”), or one of the integers 2...K+1, where K is the number of features in your dataset. Each of 2...(K+1) represents a possible feature to split on. There is one subtle point about this representation: at the bottom layer of your tree further splitting must not be allowed, so you will want to convert any V_i value into either 0 or 1 (false or true classification), perhaps by value modulo 2.

Problem 4: Programming (4 points)

Create a genetic algorithm that implements the specifications you laid out in Problem 3. Your program execution should go through two separate phases: training and testing. In the training phase, you should use the specified training data file to evolve a good decision tree. In the testing phase, you should take the best individual decision tree that was found in the course of the genetic algorithm, and apply it to each instance found in the specified testing data file.

EECS 349 (Machine Learning) Homework 2, Fall 2009

The Executable

Your program must be written in Java, Python, or MatLab. Executable requirements for the varying languages are outlined below.

A Java Executable

If your program is written in Java, it must be a JAR file (that will run on **Java 1.6**) corresponding to the following usage spec:

```
java -jar GA.jar <trainingFile> <testingFile> <maxGen> <popSize> <mutRate>
       <maxTreeDepth> <verbose>
```

```
Example: java -jar GA.jar RedQueenTraining.csv RedQueenTesting.csv 500 50 0.05 6 TRUE
```

A MATLAB Executable

If your program is written in MATLAB, you must hand in a folder containing one or more .m files that implement the decision tree. The program must run in **MATLAB R2008a**. There should be no need to alter any paths. You should **not** use any of the extra toolboxes (such as the genetic algorithms toolbox). Your folder MUST contain a file named "GA.m" that can be called according to the following usage spec:

```
Usage: GA(<trainingFile> <testingFile> <maxGen> <popSize> <mutRate>
              <maxTreeDepth> <verbose>)
```

```
Example: GA('RedQueenTraining.csv', 'RedQueenTesting.csv', 500, 50, 0.05, 6, TRUE);
```

A PYTHON Executable

If your program is written in Python, you must hand in a folder containing one or more .py files that implement the decision tree. The program must run in **Python 2.5.2**. Please refrain from using any non-standard libraries. Your folder MUST contain a file named "GA.py" that can be run with the following usage spec:

```
python GA.py <trainingFile> <testingFile> <maxGen> <popSize> <mutRate>
      <maxTreeDepth> <verbose>
```

```
Example: python GA.py RedQueenTraining.csv RedQueenTesting.csv 500 50 0.05 6 TRUE
```

What the Parameters Do

Here are the specifications for what the parameters mean:

```
<trainingFile>  - the fully specified path to the training data file.

<testingFile>  - the fully specified path to the testing data file.

<maxGen>         - an integer specifying the maximum number of allowed before the learner
                   terminates (you can stop earlier if the termination condition
                   you specify in Problem 3 is met)

<popSize>        - an integer specifying how many members of the population will exist

<mutRate>        - a real value that specifies the probability of mutation.

<maxTreeDepth>   - an integer that specifies the maximum depth of the decision trees.

<verbose>        - a string that must be either 'TRUE' or 'FALSE'
                 If verbose is 'TRUE' the output will output mean fitness of the
                 population for each generation
```

EECS 349 (Machine Learning) Homework 2, Fall 2009

What the Executable Must Do

When run, your executable should perform the genetic algorithm search that you designed in Problem 3, and generate the output described below.

Output Format

Each run of your program should create an output on the command line that contains the following information:

- The values for the parameters specified on the command line
- Data from training:
  1. The median, min and max fitness of the population in the first generation
  2. When <verbose> = TRUE you must provide the following information for each trial: The median, min and max fitness of the population in each generation
  3. The median, min and max fitness of the population in the final generation
  4. A textual representation of the "fittest" decision tree returned by your learner
- Data from testing
  1. The confusion matrix from running the fittest decision tree on the TESTING set.
  2. The accuracy of the tree on the TESTING set
     (# of instances correctly classified / total # of instances)

No specific layout is required. That said, if we have ANY problem finding the information specified above in your output file, you WILL lose points. Your output format should be CLEAR.

Include a README.TXT

In addition to the executable, please submit a README.TXT file which specifies the command line parameters that you would like the grader to use when running your program on the validation dataset. An example README.TXT would look something like:

```
Hi grader, please run my program with these options:

python GA.py RedQueenTraining.csv RedQueenValidation.csv 200 100 0.05 5 TRUE
```

IMPORTANT: Do not choose unreasonably large values for maxGen or popSize, in an attempt to improve performance. With the README.TXT settings you provide, your program should complete execution in less than 60 seconds, otherwise the grader may become bored and/or cranky...

Your Coding Style

On this assignment, your code must be clear, modular and very well commented. If your code is not clear, modular and well commented, you may lose points even if your code functions. For Java, this means we should be able to run the javadoc generator on your code and get documentation that gives the input and output specifications for every method, explanations of what every class is about, etc. For MATLAB, this means every function should be documented. Each MATLAB function should return usage and explanations when help is called for that function name. Similarly, Python functions should have associated "docstrings" and also be well commented.

EECS 349 (Machine Learning) Homework 2, Fall 2009

Problem 5: Analysis (3 points)

Evaluate the performance of your system. Describe the strengths and weaknesses of your approach, or of evolving decision trees in general, or perhaps the limitations of the RedQueen dataset (note that 100% accuracy is impossible, even on the training set, since the data is noisy). Use data generated from multiple runs of your system to support your analysis. This analysis should take between 300 and 600 words, taking no more than two pages including figures. Some example questions you might discuss in your analysis are…

How does population size affect how well/quickly your system learns?

How does mutation rate affect results?

How does changing/removing operators (mutation, sexual reproduction, etc) affect how well the system learns?

If you look at a graph of the worst, best, and population-average fitness in each generation, over time, what trends do you see? At what point does each of these performance metrics stop increasing, and why?

How do results on testing data compare with your results on training data? Do you think overfitting is occurring? Why or why not? What (if anything) might you do to help reduce overfitting when evolving a tree with a genetic algorithm?

If you look at the best decision trees that your system evolves, are they human-readable/understandable? How parsimonious are they compared to the trees generated on this dataset by Weka's ID3? Weka's J48?

Looking at one of the best trees you evolved, do the branching choices seem sensible? Are they intuitive in terms of how you might (as a human) decide to either "hit" and "stand" if you were playing RedQueen? Why or why not?

One very important criterion for determining whether you are gaining anything from a "genetic algorithm" approach is whether it outperforms other simpler methods. In particular, you might want to compare your performance to "random search" (which simply generates one random tree after another, and keeps track of the best one it found) or a "hill climber" (which only looks for local improvements).

If you had more time, what might you do in the future to potentially improve your results?