Assignment 5

CSG220, Spring 2007
Due: Thursday, March 29

1. Consider a 2-class classification problem where the input space is the real line and the two classes are labeled +1 and -1. Suppose the training data is the set of 3 points $\{(1, +1), (3, -1), (5, +1)\}$.

a. Design by hand a simple decision tree that classifies this training data perfectly. In this tree each test should be based on comparing the given input value x against some threshold value, yielding two possible outcomes. (This is how decision trees are built when some attributes are real-valued.)

b. Now suppose AdaBoost is run on this same training data using as a base learner decision stumps, based on the same tests you used in your decision tree. Do a pencil-and-paper simulation of the first few rounds of AdaBoost on this data. Just as in the example done in class, for simplicity, use the weighted misclassification rate as the criterion for selecting the best stump in each round rather than the information gain. Also, you may find it convenient to use the (equivalent) weight update variant discussed in class in which only the misclassified examples have their weights changed and the weights remain unnormalized. (Whether you do this or not is entirely up to you, but you should state explicitly which variant you are using.)

When you pick the best decision stump in each round, if there is a tie for which stump gives the best value according to the minimum-weighted-error criterion, you may choose arbitrarily between any of the equally best possibilities. Also, you should arbitrarily specify whether +1 or -1 will be the default value assigned to any leaf having equal weight for both labels and apply this consistently whenever appropriate.

Show the data weights, decision stumps, and confidence weights α_t for each of the first three rounds of boosting with AdaBoost (and show all intermediate calculations as well). Verify that the ensemble classifier after these three rounds classifies all the training data correctly.

c. Compute the margins of the training examples following each of these three rounds. List these numbers in some organized way, and, for each round, draw a cumulative distribution function plot showing the margin distribution after that round.

2. Consider a ± 1 classification problem where the input consists of attribute vectors where each attribute has a finite number of distinct values. An example is the PlayTennis data set, where we assume that the target classification is the value of PlayTennis: +1 for yes and -1 for no.

Suppose that we have an ensemble classifier consisting of a weighted majority of base classifiers, each of which is a simple decision stump testing exactly one attribute in these attribute vectors. Prove that this ensemble classifier has no more representational power than a simple perceptron using a one-out-of-k encoding for each k-valued attribute. In particular, given a set of such decision stumps and a corresponding set of confidence weights, show explicitly how to construct the weights for a simple perceptron that gives the same classifications. Make your proof as rigorous as possible. In addition, give a nontrivial example illustrating your construction of the perceptron weights from a given set of decision stumps and confidence weights.

Note: Writing down the complete formal argument involves the careful use of notation that makes clear various dependencies. For this reason, it is suggested that you make use of the following observations and notation:

First, in a one-out-of-k input encoding for k-valued attributes, there is a separate input node u for each (attribute, value) pair. Let $u(a, v)$ denote the value (0 or 1) taken on by the input node corresponding to the attribute a and value v. Note that this can be defined using indicator function notation as $u(a, v) = I(a = v)$. For example, in the PlayTennis data set, $u(Outlook, Sunny)$ should take on the value 1 iff $Outlook = Sunny$. Let A represent the set of attributes and, for any given attribute $a \in A$, let $V(a)$ represent the set of possible values attribute a can have. For example, in the PlayTennis data, $A = \{Outlook, Temperature, Humidity, Wind\}$ and $V(Outlook) = \{Sunny, Overcast, Rain\}$.

Next, let the decision stumps be indexed by $t = 1, 2, \ldots, T$. The t^{th} stump produces a ± 1 for each possible value of a particular attribute. Let $a_t \in A$ denote the particular attribute tested by the t^{th} stump, and let $y_t(v)$ denote the classification (± 1) appearing at the leaf of this stump on the branch for value $v \in V(a_t)$. Finally, let α_t denote the confidence weight corresponding to the t^{th} stump.

(Note that this suggested notation adheres consistently to the convention that dependencies on attributes and values is given by functional arguments, while dependencies on which stump is involved use a subscript.)

Hint: Write the output of the t^{th} stump as a sum involving the appropriate u and y values.

Additional hint: Given a set of decision stumps indexed by $t \in \{1, 2, \ldots, T\}$ and any particular attribute $a \in A$ you may find it useful to define $\mathcal{T}(a) = \{t \mid a_t = a\}$. (Remember that multiple stumps in the ensemble may test the same attribute.)

3. Download and unzip the Lisp Q-learning program found by following the appropriate links on the course web page. This code implements a Q-learning agent learning an optimal path from the start state to the goal state in variations of the maze discussed in the lecture.

a. Start with all the default values as initialized in the code. The most important of these are: learning rate (`*alpha*`) = 1, discount factor (`*gamma*`) = 1, and probability of random outcomes (`*perturb-prob*`) = 0. How many trials does it take to learn the optimal path from start to goal? What optimal path is learned? Give a simple description of the objective this agent learns to optimize.

b. This time set the learning rate `*alpha*` to 0.5. Now how many trials does it take to learn the optimal path? What optimal path was learned this time? Explain the difference between your results here and your results in part (a).

c. Now reset `*alpha*` to 1 (making sure all other global parameters have their initial values as well). Then add the 4-cell "quicksand" area to the maze as discussed in lecture. You'll have to modify the source code slightly for this. (Where to make this change is clearly indicated in the program.) [Note: Lisp does not reinitialize global variables to the values specified in the code when a file is reloaded. They'll retain whatever values they already have in the Lisp environment.] Now how many trials does it take to learn the optimal path? What optimal path is learned? How does this result compare with your result in part (a)?

d. Using the same maze (including "quicksand") as in part (c), this time set `*alpha*` to 0.1 and `*perturb-prob*` to 0.2. This time how many trials does it take to learn the optimal path? (An approximate number is good enough here.) What optimal path is learned this time? How do your results in this experiment compare with the results in parts (a) and (c)? What accounts for the differences?

e. Now go back to using the original maze (without "quicksand") and set all parameters back to their initial values. (In particular, make sure you set `*alpha*` to 1 and `*perturb-prob*` to 0.) But this time set `*goal-reward*` to 100 (and keep `*explore-prob*` at 0). After some number of trials,

you'll find that the agent consistently follows the same path to the goal. Is this path optimal? Explain why this occurs.

f. Keep everything just as in part (e), including `*goal-reward*` = 100, but this time experiment with some nonzero values for `*explore-prob*` (like 0.1 or 0.3). Describe what happens and explain why it happens.

For this problem collect all your discussion and answers to the questions in one concise location in what you turn in. Also provide, where appropriate, a limited amount of dribble output demonstrating the relevant results you obtain running the program. Feel free to edit out extraneous program output. (For example, it prints to the screen a separate line for each trial. In any experiment that runs for several hundred trials, it is unnecessary to print out hardcopy showing every line of output produced.) But it will be helpful if your dribble file shows the values of relevant parameters together with the corresponding output as you vary them.

Unless you make substantial modifications to the code (or write your own), no source code listings are expected.