Self Organizing Maps and Boosting

This problem set contains a few short programming problems and some by-hand exercises on boosting intended for final exam practice. Expect problem 1 to take up to 4 hours, depending on your comfort level with Scheme, problem 2 to be quick, and problem 3 to take up to 2 hours, depending on your comfort level with boosting. Ask questions about the coding section as early as possible.

Problem 1: SOM Core Functions

For this problem, you will implement some utility functions that will be used by an SOM framework we have given you. The specifications for each function are described in the attached som-doc.txt file, and the public test cases should help you resolve any confusion. For convenience, the three functions you are to implement are named below:

1. get-k-neighbors
2. metric-find-best-match
3. make-simple-decay

Please note that when get-k-neighbors returns, it needs to return the nodes in order of decreasing distance; i.e., the furthest one should be first in the list.

Problem 2: SOM Experiment

Following the example in som-doc.txt, perform the following SOM experiment. Construct a 10x10 grid SOM, initialized with vectors randomly sampled (uniformly) from [0,1) in each component. Use euclidean metric and refinement.

1. Train this SOM for 1000 iterations. Has it converged to a good representation of the distribution the inputs are being sampled from? (Answer yes or no.)

2. Train the SOM for 1000 more iterations (2000 iterations total). Has it converged to a good representation now? (Answer yes or no.)

Afterwards, think about the refinement function used in this example and the examples Prof. Winston presented in lecture.
Problem 3: Boosting

As you know, AdaBoost is a strategy for combining weak learners to produce strong(er) ones. For this problem, you will simulate AdaBoost’s operation on a simple (but somewhat nonstandard) weak learner, to gain insight into AdaBoost’s operation. The weak learner operates as follows:

It takes as input a weighted training set - that is, a list of example vectors $x_i$ (of dimension $d$), a list of the corresponding class values for these examples $y_i \in \{1, -1\}$, and a weight for example $p_i$. Call $P = \{i | y_i > 0\}$ the set of positive examples, and $N = \{i | y_i < 0\}$ the set of negative examples.

For each dimension $j = 1...d$, compute the weighted average coordinate of the positive examples and the weighted average coordinate of the negative examples (where $P_{\text{tot}}$ is the total weight of the positive examples, $N_{\text{tot}}$ is the total weight of the negative examples, and $x^j_i$ is the $j$th component of example vector $i$):

$$POS^j = \frac{1}{P_{\text{tot}}} \sum_{i \in P} p_i * x^j_i$$

$$NEG^j = \frac{1}{N_{\text{tot}}} \sum_{i \in N} p_i * x^j_i$$

The learner now has the weighted centers of the components of the positive and negative examples for each dimension. The learner gets to choose among splits at the midpoint of these centers, for each dimension $j$. The splits are located at $MID^j = 0.5 * (POS^j + NEG^j)$, with the positive side of the split hyperplane determined by $POS^j - NEG^j$. In other words, if $d = 3$, the learner can choose between splitting at the midpoint of the $x$ coordinates of the weighted centers, the $y$ coordinates of the weighted centers, or the $z$ coordinates of the weighted centers.

The training set error formula for split $j$, then, is:

$$\epsilon^j = \sum_{i} p_i * ((y_i \text{sign}[(POS^j - NEG^j)(x^j_i - MID^j)]) \leq 0)$$

The expression $(... \leq 0)$ is 1 if the sign of the quantity $...$ is negative, and 0 otherwise.

The learner picks the decision boundary with the lowest error according to this formula. Note that this is NOT THE SAME as the minimum entropy split criterion. (If you are confused, draw the pictures corresponding to the examples below to help you work out the vectors $POS$ and $NEG$, etc.)

1. Consider the training set containing the examples (1,1), (-1,1), (-1,-1), (1,-1) with class labels 1,-1,1, and -1 respectively. Can you run an iteration of AdaBoost with the stumps described above on this training set? (Answer yes or no.)

2. Consider the training set with examples (1,1), (-1,1), (-1,-1), (.9,-1), (.2,.3) and class labels 1, -1, 1, -1, -1 respectively. Compute two iterations of AdaBoost. Report the following:

   • The weights for each example (at each iteration)
   • The $\alpha$ values for each iteration
• The $\epsilon$ values for each iteration

Report this information as a list. This list should have the form

\[
\begin{align*}
(1 & \text{ (<weight for example 1> <weight for ex 2> ... ) } \\
& \text{ (<err for split on dim 1> <err for split on dim 2>)} \\
& \text{ <alpha_1> <epsilon_1> }
\end{align*}
\]

\[
\begin{align*}
(2 & \text{ (<weight for example 1> <weight for ex 2> ... ) } \\
& \text{ (<err for split on dim 1> <err for split on dim 2>)} \\
& \text{ <alpha_2> <epsilon_2> }
\end{align*}
\]

That is, each iteration has a list containing the iteration number, the list of weights at that iteration, the alpha value for that iteration and the epsilon value for that iteration. Think about what you observe and why.

3. Imagine you had a one-dimensional training set containing (1), (2), (3), and (4) with class values 1, 1, -1 and 1. Imagine the first decision stump AdaBoost chooses is “if $x > 2.5$ then the class value is -1”. Could the reweighting (that is, the weights for each example) for the next step be (1/8, 1/8, 1/8, 5/8)? Answer yes or no.

---

**Problem 4: Ensuring A Nonzero Final Score**

Make sure you test your code on the public test cases from a freshly started run of the official scheme. One way to do this is to log in to a dialup, add scheme, run scheme, load the tester, then test your submission.