COMP 652 - Homework 4

Assigned: Oct 22, 2008
Due: Nov 3, 2008
Late: Not accepted late!

1. Weighted nearest neighbor for regression (10 pts.)

Consider the data in the files "HW4_Q1_X.txt" and "HW4_Q1_Y.txt". These two data files decribe a univariate regression problem with 100 data points, which we will solve by using a weighted nearest neighbor approach. Recall that in weighted nearest neighbor, we compute an output as $h(x) = \sum_{i=1}^{m} w_i(x) y_i$, where $w_i(x)$ is the weight that training example i contributes to the prediction for query point x. We will use Gaussian weights, of the form:

$$w_i(x) = \frac{\exp(-c(x - x_i)^2)}{\sum_{j=1}^{m} \exp(-c(x - x_j)^2)}$$

The numerator in the above formula depends on the distance between x and x_i, and the denominator ensures that the weights, w_i, sum to one. Notice that the weight formula has a parameter c in it. Normally, for a Gaussian weighting function, we would write $1/2\sigma^2$ instead of c. However, for some of the math below, it is more convenient to write it this way.

A. (4 pts.) Implement the weighted nearest neighbor regressor described above. Make a plot of the data along with the curve $h(x)$ for three different choices of c: 0.1, 1 and 10. Also, turn in your code.

B. (3 pts.) Implement a cross-validation approach to find an optimal value for c. In particular, separate the data set into its first half and its second half. Use the first half for "training" the regressor[1]. Use the second half for validation. In particular, for any choice of c, you can compute the mean squared error on the validation set of the weighted nearest neighbor regressor "trained" on the training set. Use this approach to determine an optimal (or near optimal) c. For example, you could simply test all c between 0.1 and 10 in increments of 0.1, and see which is best. Report the optimal c, the mean square error on the validation set for that c, and turn in your code.

C. (3 pts.) The approach described above is perfectly fine for optimizing c. However, if we were operating in higher dimensions, we might want to have not just a single parameter c, but perhaps different scale factors for each dimension or possibly an entire covariance matrix. In such a situation, we can not exhaustively test all possible parameter settings. Although we will stay in one dimension for this homework, such considerations motivate the following problem. Consider again the data set which has been split into training and validation halves. The mean squared error of the validation set can be viewed as a function of c. We'll denote this by $\mathcal{E}(c)$. Moreover, this error function is differentiable with respect to c. Derive a formula for $\frac{\partial \mathcal{E}(c)}{\partial c}$, and simplify this formula as much as possible.

[1] I put training in quotes, because for a nearest neighbor regressor there isn't any training really. One just stores the data.

D. (Extra credit–do if you're interested) If you want, you can use your part C results to implement a gradient descent approach to optimize c. That is, starting from some initial value of c, repeatedly update as $c \leftarrow c - \alpha \frac{\partial \hat{\mathcal{E}}(c)}{\partial c}$, where α is a step size parameter. (For convenience, you can probably choose some constant stepsize like $\alpha = 0.1$, though I haven't actually tested this yet.) Keep taking gradient steps until the derivative is essentially zero, and report the final value of c found. Does it correspond to the value found in part B? As usual, turn in your code.

2. Decision Trees (10 pts.)

Recall that decision trees are usually constructed incrementally from the top down. At the root we choose a *test* that appears best according to some criterion. In this exercise, we will explore some ideas related to tests.

A. (3pts.) Let $p_1, p_2, \ldots, p_k$ define a probability distribution for a random variable with k possible outcomes. That is, $0 \leq p_i \leq 1$ for all i, and $\sum_i p_i = 1$. In the lecture, we described the entropy of a distribution, $\sum_i p_i \log p_i$, and its use in choosing tests. Toward the end of the lecture, however, we also mentioned the Gini index $1 - \sum_i p_i^2$. What is the maximum value of the Gini index, and for which value or values of the p_i is that maximum attained? What is the minimum value of the Gini index, and for which value or values of the p_i is that minimum attained? Prove your claims.

B. (3pts.) Now, consider the data in the files "HW4_Q2_X.txt" and "HW4_Q2_Y.txt". The second file is just our usual Wisconsin data with the binary recurrence / nonrecurrence of each patient. The first file has the feature values for each patient, but binarized. Each feature value is either zero or one, depending on whether the original (real-valued) feature value was greater than or less than the midpoint between the min and max values for the feature. If we were constructing a tree, it would be natural to have one test for each feature, branching on the two possible values of the feature. Write code to evaluate the *information gain* of each feature, for predicting Y. Report the information gain of each feature, and state which feature has the greatest information gain.

C. (4pts.) This part is similar to the previous one, but we will use the Gini index instead. Using information gain as an analogy, describe how you would use the Gini index to quantitatively measure the goodness of each feature as a potential test at the root. Write down a formula for the quantity you suggest to compute, and whether a minimal or maximal value determines the best feature. Then, implement (i.e., code) your suggestion and use it on the data set mentioned in part B. Report the score of each feature and which feature is best. And turn in your code.

3. Decision Lists (10 pts.)

A decision list is an ordered sequence of tests and predictions: $T_1, h_1, T_2, h_2, \ldots, T_k, h_k$. Each test T_i is binary, evaluating to true or false. Each h_i is a prediction. To make a prediction for query point $\mathbf{x}$, the tests are applied to $\mathbf{x}$ in order until we find the first test that evaluates to true. We then return the prediction corresponding to that test.

In this problem, we will implement one method for learning a decision list. We will use the data "HW4_Q3_X.txt" and "HW4_Q3_Y.txt". This is a binary classification problem with two real-valued inputs. It is the same artificial data set as was used in the decision tree lecture.

Consider tests of the form $x_1 > \tau$, $x_1 < \tau$, $x_2 > \tau$ and $x_2 < \tau$. For thresholds τ, take the midpoints between observed values in the data set. As there are 100 data points, there should thus be 99 different thresholds for x_1 and 99 thresholds for x_2, leading to a total of 4*99 = 396 possible tests.

We will use a greedy approach to construct the decision list. For a given data set, let the score of a test be as determined as follows:

> Find all training examples for which the test evaluates to true. If these example include both classes ($y = 0$ and $y = 1$), then give the test a score of zero. If all examples are of the same class, then give the test a score equal to the number of examples passing the test.

Evaluate each test a find the one (or a one) with maximum score. Take this as the first test in the decision list. The corresponding prediction is either $y = 0$ or $y = 1$, depending on which type of training examples passed the test. Then, remove from the training data all the examples that pass the test, and repeat! That is: evaluate each test again (on the reduced training data), add the best test to the second spot on the decision list, and remove the training data it correctly classifies. Etc.

Turn in the decision list obtained, the score of each test as it was added to the list, your code, and if at all possible a visualization of the tests similar to the one shown in class for decision trees (i.e., showing the cuts of the input space).