STA 414/2104, Spring 2012 — Assignment #1

Due at the start of class on February 14. Please hand it in on 8 1/2 by 11 inch paper, stapled in the upper left, with no other packaging.

This assignment is to be done by each student individually. You may discuss it in general terms with other students, but the work you hand in should be your own. In particular, you should not leave any discussion with someone else with any written notes (either on paper or in electronic form).

In this assignment, you will investigate a supervised learning method that combines linear regression on the input variables with nearest-neighbor prediction. You will implement this method in R, and try it out on two data sets that I have provided on the course web page.

The method looks at a set of n training cases, for each of which there is a vector of p inputs, x, and real-valued response, y. From the information in these training cases, predictions are made for test cases in which x is known but y is unknown. (For this assignment, you will actually have the true values of y in the tests cases, but you should not use them until the time comes to evaluate how well this method works.)

The method first fits a linear regression model for y based on x, in which y is modeled as follows:

$$y \;=\; \beta_0 \;+\; \sum_{j=1}^{p} x_j \beta_j \;+\; \text{noise}$$

This model is fit by penalized least squares, which produces estimated regression coefficients, $\widehat{\beta}_0, \widehat{\beta}_1, \ldots, \widehat{\beta}_p$, that minimize a quadratic penalty plus the sum of the squares of errors in training cases:

$$\lambda \sum_{j=1}^{p} \beta_j^2 \;+\; \sum_{i=1}^{n} \Big(y_i \;-\; \Big(\beta_0 + \sum_{j=1}^{p} x_{ij} \beta_j \Big) \Big)^2$$

The value of λ determines the magnitude of the penalty. Note that β_0 is not included in the penalty term.

Once $\widehat{\beta}_0, \widehat{\beta}_1, \ldots, \widehat{\beta}_p$ have been found, the residual for each training case can be computed. The residual for case i is defined as

$$r_i \;=\; y_i \;-\; \Big(\widehat{\beta}_0 + \sum_{j=1}^{p} x_{ij} \widehat{\beta}_j \Big)$$

When making a prediction for a test case with input vector x, the method starts by finding the K training cases with inputs nearest to x in Euclidean distance. Let the set of indexes of these K training cases be $N(x)$. The prediction for y in this test case is then computed as

$$\beta_0 \;+\; \sum_{j=1}^{p} x_j \beta_j \;+\; \frac{1}{K} \sum_{i \in N(x)} r_i$$

The effect of this is that the prediction follows the linear regression model modified by the average residual for training cases near to the test case. One might hope that this method

will do better than either simple linear regression or the simple nearest neighbor method, at least for problems in which the true relationship of y to x is close to linear, but with some local variations.

Note: If there are any ties when finding nearest neighbors, you may for simplicity break them in favour of the training case with lowest index. (This may not be the best thing to do, but ties will rarely or never arise with the data you will look at anyway.)

In order to use this method, values for λ (a non-negative real) and for K (an integer from 1 to n) must be specified. Since it isn't obvious what values will be best (and this will vary from one problem to another), we will use S-fold cross validation to pick values for λ and K, using squared error as the cross validation performance measure.

In the experiments you do, you should set S to 5 (but your program should handle any integer S from 1 to n). You may assume that the training sets you are supplied with are already in random order, so you can divide the training set into S validations sets by just taking the first n/S cases, the next n/S cases, etc. (If n/S isn't an integer, you should divide the training set as equally as possible.)

You should implement this method as a set of R functions, which should be written in good style (in particular, properly indented), and suitably documented. You should hand in a listing of these functions.

First, you should write an R function called `lm.nn` that takes as arguments a matrix of inputs in training cases, a vector of responses in training cases, a matrix of inputs in test cases, the value of K to use, and the value of λ to use. The result of this function should be a vector of predictions for y in the test cases. If the value of K is greater than n (the number of training cases), it should be adjusted to equal n. If λ is infinity (`Inf` in R), the estimated β should be the limiting values as λ goes to infinity (which you will probably have to compute specially). You should implement the penalized least squares estimation using matrix operations (not by trying to use R's built-in `lm` function). Explicitly inverting matrices is OK, even though this isn't the best method.

Second, you should write an R function called `lm.nn.val` that takes as arguments a matrix of inputs in training cases, a vector of responses in training cases, a vector of indexes of validation cases, a vector of values for K to try, and a vector of values for λ to try (which might include 0 or `Inf`). This functions should assess all combinations of the values for K and λ to be tried using the set of validation cases specified, fitting to the remaining cases (the estimation set), and return an array of validation squared errors for these combinations. The `lm.nn.val` function should of course call the `lm.nn` function with the "training cases" and "test cases" for this call of `lm.nn` actually being the estimation and validation subsets of the full training set.

Third, you should write an R function called `lm.nn.cross.val` that takes as arguments a matrix of inputs in training cases, a vector of responses in training cases, the value of S, a vector of values for K to try, and a vector of values for λ to try (which might include 0 or `Inf`). This function should assess all combinations of the values for K and λ to be tried using S-fold cross validation, and return a list with elements `K` and `lambda` giving the best combination found. It should also print a table giving the cross-validation results (square root of average squared error) for all combinations of K and λ. The `lm.nn.cross.val` function

should work by calling the `lm.nn.val` function S times.

To make predictions for a set of test cases in a real application, the `lm.nn.cross.val` function would first be used to find the "best" K and λ, and these values for K and λ would then be used as arguments in a call of `lm.nn` that is given the full training set and the actual matrix of inputs in test cases.

For this assignment, however, you should write a function called `lm.nn.test.err` that takes as arguments a matrix of inputs for training cases, a vector of responses for training cases, a matrix of inputs for test cases, a vector of responses for test cases, a vector of values for K, and a vector of values for λ, and prints a table giving the predictive performance (square root of average squared error) of `lm.nn` on the test cases for each combination of values for K and λ. This table shows how good each combination of values for K and λ actually is (on these test cases), and can be compared with the table printed by `lm.nn.cross.val`. (Of course, in a real application you wouldn't know the true values the responses in the test cases at the time when you need to make predictions.)

You should try out this method on two data sets provided on the course web page, handing in the output of `lm.nn.cross.val` and `lm.nn.test.err` for each. For both data sets, you should try values for K of 1, 2, 4, 8, 16, 32, and `Inf` (which will set K to the number of items in the training/estimation set), and values for λ of 0, 0.1, 1, 10, 100, 1000, and `Inf`. You should hand in the R script that runs the functions you wrote on these data sets, and the output of your script.

The first data set is the one used as an example in lectures, with a single input. The fifty training case supplied in the file `art-train.dat` are the same as in the lecture examples. There are 1000 test cases in the file `art-test.dat`. You should read both files using `read.table` with the `head=FALSE` option. The response variable is first, followed by the input variable.

The second data set was used as an example in Hastie, Tibshirani, and Friedman's book, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. The objective with this data is to predict ozone levels from nine input variables. I have randomly reordered the cases, and selected 100 for a test set, in the file `ozone-test.dat`, with the other 230 used for training in the file `ozone-train.dat`. (Note that this random split makes sense only if one assumes that the cases are independent, which is open to question since they are from (mostly) consecutive days in one year, but we'll ignore this issue.) In both files, the response variable (ozone) is first, with the nine input variables following. Both files should be read using `read.table` with the `head=TRUE` option.

You should first try predicting ozone using the inputs as given. You should then standardize the inputs to have mean zero and standard deviation one (the R `scale` function is useful for this), and try predicting ozone using these standardized inputs.

Last, you should hand in your discussion of the method and results. To begin, you should consider what the method does in the limits as λ goes to 0 or to infinity, and as K goes to the number of training/estimation cases (explain and justify your reasoning about this). You can then discuss the results you obtained for the two data sets, and the effect of standardizing inputs.