STA 414/2104, Spring 2014 — Assignment #3

Due at the start of class on April 3. 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 use a combination of Principal Components Analysis (PCA) and a multilayer perceptron (MLP) neural network to classify whether images included in webpages are advertisements or not.

The data is from a 1999 paper by Nicholas Kusmerick on "Learning to remove Internet advertisements", and is available from the UC Irving Machine Learning Repository, from the URL `archive.ics.uci.edu/ml/datasets/Internet+Advertisements`. I have processed the data to remove three covariates that are often missing, to remove a few cases in which another covariate is missing, to remove a few covariates that are zero in all but nine or fewer cases, to randomly reorder the cases, and to divide the cases into a training set of 1300 cases and a test set of 1964 cases.

After this processing, there are 1521 covariates, all of which are binary (with values of 0 or 1). One of these covariates indicates whether the URL for the image is for the same server as the main webpage. Others indicate whether or not certain words occur in text associated with the image. Most of these covariates are 0 for most cases — in particular, 1281 of the covariates have the value 1 in less than 1% of the training cases).

The data is available from the course web page in the following files:

`a3trnx`	covariate values for the training cases
`a3trny`	class labels for training cases
`a3tstx`	covariate values for the test cases
`a3tsty`	class labels for test cases

The class label is 1 if the image is an advertisement and 0 if the image is not an advertisement. You should of course look at the labels for test cases only at the very end, to see how well you did. You can read the covariate files with `read.table`, with the `head=FALSE` option, and then convert the data to a matrix with `as.matrix`. You can read the labels with `scan`.

For both computational and statistical reasons, it may be desirable to reduce the dimensionality of the covariates from 1521 to a much smaller number. You will do this using PCA, keeping the first 10, 20, or 40 principal components. You will then see how well you can predict the class using only these 10, 20, or 40 principal components as covariates. You should first try predicting the class using maximum likelihood logistic regression, and then try using a multilayer perceptron network with 10 hidden units. You should try training the MLP network by gradient descent on minus the log likelihood alone, and on minus the log likelihood plus a quadratic penalty on the input-to-hidden weights (only).

To make predictions for the test cases, you will need to make choices of whether to use 10, 20, or 40 principal components, of what learning rate to use, of the number of gradient descent iterations to do, and of whether or not to use a penalty and if so what its magnitude should be. You should make these choices by splitting the 1300 training cases into an estimation set (the first 1000 training cases) and a validation set (the last 300 training cases). You should find principal component directions and fit logistic regression models using only the data in the estimation set. When fitting the multilayer perceptron models, you should do gradient descent using the gradient computed from only the cases in the estimation set. You should compute the average log probability of the cases in the validation set according to the parameters found for the logistic regression and MLP models in order to choose among them, and for the MLP models, to choose how many iterations to train for, and what penalty magnitude to use. (You could also choose the learning rate this way, but you may instead choose it so that minus the log likelihood from the estimation set decreases steadily.)

You should find the principal component vectors using the `pca.vectors` function from the PCA example (for week 10) on the course web page, which you can use without modification. You should find these vectors using the estimation set of 1000 cases, *not* the full training set, or the test set. Once you have found the principal component vectors, you can compute the projections of the estimation, validation, and test cases on these vectors using the `pca.proj` function. You will use these projections on the first 10, 20, or 40 principal component vectors as covariates in your classification models.

You can fit logistic regression models by maximum likelihood using R's `glm` function, with the option `family="binomial"`. For example, `glm(y~X,family="binomial")` models the vector of binary responses `y` in terms of the matrix of covariates `X`.

You should fit multilayer perceptron models using a modified version of the example functions provided on the course web page (for week 10). You will need to modify these functions to model a binary response, replacing the squared error for a training case by minus the log probability of the response. You will also need to modify the network training function so that it can minimize the sum of minus the log likelihood over training cases plus a penalty proportional to the sum of the squares of the weights on input to hidden connections, with the penalty factor being an argument of the training function (set to zero for no penalty). In terms of the week 10 lecture titled "Avoiding Overfitting Using a Penalty", only the first penalty term (sum of squares of $w_{kj}^{(1)}$ multiplied by λ_1) is used, with λ_2 fixed at zero. Note that there is no penalty on the "bias" parameters, $w_{0j}^{(1)}$.

You will need to write a function that computes the average log probability of the responses for the 300 validation cases for all values of the parameters found in a network training run. You can use this to select which set of parameters found during training seems best (which will not necessarily be the parameters from the last iteration). You can also compare the best validation performance for different training runs, that use different numbers of principal components, or different penalty magnitudes. You will also need to compute the average log probability of responses in the validation set using the logistic regression coefficients found using `glm`, using different numbers of principal components. Using these figures on validation performance, you can select which model (and set of model parameters) to use for making predictions for test cases.

For the model and parameter set that you select, you should compute both the average log probability of the responses for test cases, and the classification error rate, when classifying a test case as being an advertisement (label 1) when the model says its probability of being an advertisement is greater than 0.5. These figures indicate how well you would have done if this were a real problem, in which you have to make predictions for test cases before knowing their true values.

You should also find the average log probability of the response and the classification error rate on test cases for all the other models and training runs that you did (using the best parameter set from a run as chosen using the validation set). These performance figures are of interest in seeing how reliable the performance on the validation set was as an indication of performance on the test set.

You should hand in your modified MLP functions, the other functions you wrote, the R script you used to read the data, fit models, and report the results, the output of this R script, and a discussion of the results. In your discussion, you might comment on how easy or hard it was to "tune" the network training (selecting a suitable learning rate, and a suitable number of gradient descent iterations), how fast or slow the training was, whether training with a penalty helped or not, how reliable performance on the validation set was as an indicator of which model was best, and what the effect was of using more or fewer principal components.