STA 414/2104, Spring 2011 — Assignment #2

Due at the start of class on March 22. 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 paper or electronic).

In this assignment, you will extend the R functions for neural network training from the course web page so that training minimizes minus the log likelihood plus a quadratic penalty. You will then apply this program, as well as R's `glm` function for doing logistic regression, to data from a problem in high-energy physics.

The physics problem is to determine whether or not there is good evidence for some new phenomenon, such as a new particle, in the data from collisions in a particle accelerator. Physicists have devised ways of condensing the many measurements obtained from a collision into a smaller number of relevant numbers. For the data we will look at, there are 50 such numbers describing each collision event.

The physicists also have a simulation program that randomly generates these 50 numbers on the assumption that the new phenomenon either does not exist, or did not occur in the particular collision observed, as well as a simulation program that randomly generates these 50 numbers on the assumption that the new phenomenon did occur. From a training set of these "background" and "signal" events, your task is to train a MLP network to distinguish background from signal events. Given a vector of 50 numbers describing an event, the trained network will produce a probability that this is a signal event (on the assumption that backround and signal events occur equally often, as will be true in the simulated data I give you).

Such a network would not be used to directy declare that an observed event actually shows the new phenomenon, since one might not trust the network to produce correct probabilities. Instead, the network would be used to condense the 50 numbers describing an event down to just one number (the supposed probability that the event is a signal event). The distribution of this one number for simulated background and signal events can then be estimated (eg, by a histogram). We could then see whether the distribution for actual events can be explained as only background, or whether some non-zero fraction of signal events is required to adequately explain it. For this assignment, however, you will just train the network; you don't have to worry about exactly how it would be used later.

The simulated data used for this assignment was obtained from the UCI repository of machine learning datasets. I have eliminated some garbage records with all numbers equal to -999, which may have been cases where the simulation program crashed. I randomly selected 4000 events for use as an estimation set, 2000 events for use as a validation set, and 10000 events for use as a test set (which you should look at only at the very end to see how well you did). All these data sets contain half background events and half signal events (though this may not be the proportions for real events).

The 50 variables have diverse offsets and scales, so centering and scaling them is necessary to avoid numerical problems, and to prevent the variables that happen to have the largest scales from dominating. R commands to read the data, rescale it, and combine the estimation and validation sets can be found on the course web page.

You should modify the R functions for MLP training from the course web page to allow for a penalty of the form λ_1 times the sum of the squares of the input-to-hidden parameters (weights),

excluding the `w1_0` weights, plus λ_2 times the sum of the squares of the hidden-to-output weights, excluding `w2_0`.

You should then use the `mlp.cv` function to train several networks, using the estimation and validation sets provided. (Note that `mlp.cv` expects to receive a single training set, along with a vector of indexes for the validation cases, so you should combine the estimation and validation cases, as is done in the script for reading the data mentioned above.) Each training run results in a single set of network parameters, chosen to be those from the iteration with the best performance on the validation set.

You will need to choose suitable learning rates so that the learning is stable (the log probability of estimation should always go up). You may need a quite small learning rate. You will also need to decide how many iterations to train for. You should aim to train for long enough that it is clear that the log probability for the validation cases will go down with further training (or that this seems to have stablized, and will not change much with further training).

The results you get from a training run will depend on the random number seed used to generate the initial weights. You should try at least two random number seeds for each type of run, which you can set with commands such as `set.seed(7)`, to see how much the results vary with the random initial weights.

You should first try training networks with no penalty ($\lambda_1 = \lambda_2 = 0$) using 2, 4, 8, and 16 hidden units, and identify which number of hidden units seems to work best. You should then try setting both λ_1 and λ_2 to 1, 3, 9, and 27, with whatever number of hidden units seemed best with no penalty. Once you have the results from these runs, you can try other possibilities, looking for the best results possible on the validation set. You could use different numbers of hidden units than mentioned above, or different values for the penalty, including perhaps setting λ_1 to be different form λ_2, and different learning rates, including perhaps using different learning rates for the two groups of weights.

From these explorations, you should choose a few networks that seem like they work best. Only after this should you look at the test data, to see how well these networks actually perform.

As a point of comparison, you should also see how well maximum likelihood logistic regression does, when fit to the entire training set (estimation and validation sets together). You can fit a logistic regression model with the R command

```
m <- glm (y ~ X, family="binomial")
```

where `y` is the vector of responses (0 or 1) and `X` is the matrix of input variables (one variable per column). You can then make predictions based on the estimated coefficients, which you can get with `coef(m)`. Do not use R's built-in `predict` function for this.

Finally, you should discuss your results. Does using a penalty help? If the penalty is chosen well, does early stopping help? What else of interest have you seen? You should hand in your R commands, the results you obtained, your discussion, and any output or plots that contribute to your discussion.