Homework 8: Additive Models and Backfitting

36-350, Fall 2009

Due at 5pm, Monday, 16 November 2009

In this assignment, you will build a simple function to fit additive models through backfitting, and apply it to a simulated data set (where you know what it should give you), and a real one (where you don't). This will build character. In principle, you could write something at least as powerful as `mgcv::gam` or `gam::gam`, but I don't want to build *that* much character.

Whenever giving numerical values, do not use more precision (significant figures) than you can justify.

1. Write a function which takes as inputs two vectors, say `x` and `y`, centers `y`, and returns a smoothing of the centered `y` values over `x`. You can call any nonparametric smoothing method you like; however, your function must *automatically* adjust any control settings (e.g., bandwidth); fixed, data-independent control settings will only get you partial credit.

   Check that your centered smoothing function is working by running it with `x=seq(-1,1,length.out=100)` and three different responses:

   (a) `y=rnorm(100,1,0.1)`

   (b) `y=x^2+rnorm(100,0,0.1)`

   (c) `y=tanh(5*x)+0.1*rt(100,3)`

   For each case, check that the output is centered, and that it has the proper shape. Include plots comparing your smoothing curves to what they should be.

   Debugging hint: Some smoothing functions re-order the data; this helps for drawing curves on scatterplots, but is not what we want. Make sure that your vector of returned fitted values is in the same order as the original data.

2. Important Note: In the first version of the handout for lecture 21, and I believe in lecture, I said that the partial residuals should be kept fixed through each iteration of back-fitting. This is wrong; I must have been having a (prolonged) senior moment. The lecture notes online have been corrected.

(a) Write a function to do one iteration of backfitting. It should take as arguments an $n \times p$ matrix of input values, a length-n vector of residuals, and an $n \times p$ matrix of partial response values. It should cycle through the p variables, calculating the partial residuals for each and smoothing them against the corresponding input features. It should return the new $n \times p$ matrix of partial responses.

(b) Write a function, `am`, for fitting additive models. It should take as inputs an $n \times p$ matrix of `x` values, a length-n vector of y values, a numerical tolerance, and maximum number of backfitting iterations. Your function should halt and return either when none of the fitted values changes by more than the tolerance, or when it reaches the maximum number of backfitting iterations. It should return a list with the following components:

- a vector of fitted response values
- a vector of residuals
- the over-all mean
- an $n \times p$ matrix of partial responses
- the $n \times p$ matrix of input features
- a flag indicating whether it converged or ran out of iterations.

You may include other elements in the return value if they seem helpful.

(c) i. Generate a 100×3 matrix where the first column is uniform on $[-1, 1]$, the second column is standard Gaussian, and the third column is exponential with rate 2. Generate a length 100 vector as $\tanh x_1 + x_2^2 + 0.1\eta$, where η has a t distribution with three degrees of freedom. Fit an additive model to this data.

ii. Plot the partial response functions. Add curves showing what the partial response functions ought to be. How well do they match?

iii. Test that the partial response matrix does not change appreciably under further backfitting.

iv. Fit the same data with `gam` from `mgcv`, and plot the those partial response curves as well; how well do the results match?

3. Write a function, `predict.am`, for predicting responses to new inputs from an additive model. Your function should take two arguments: a list of the kind produced by your `am` function, and an $m \times p$ matrix of new input values. Your function should return a vector of length m of predicted response values.

   You *cannot* assume that the new input values match old ones. You *must* interpolate (or extrapolate).

   Make sure your function checks that the dimensions of the new data are compatible with those of the fitted model.

(a) Check that when the new data is equal to the `x` argument of the training data, the new predictions match the fitted responses. (Inserting a special test to check for old data will result in your getting no credit for this part.) If you are unable to reproduce the fitted values *exactly*, explain why, and how to make sure the discrepancy is kept under control.

(b) Check that when given a sequence of testing points between two training points, the predicted responses smoothly interpolate between the fitted values at the end-points.

(c) Check that when given test points outside the training region, but not far outside, the predictions extrapolate reasonably.

4. The data set `ozone` (in the CRAN package `faraway`, among other places) records the level of ozone (O_3), one of the toxic components of smog, and some covariates, as recorded daily over 1976 in Los Angeles. (Some days with missing data have been edited out.) The features are as follows:

`O3`	Daily maximum ozone concentration (parts per billion)
`vh`	Height above sea-level at which pressure equaled 500 millibars (meters)
`wind`	Wind speed (miles per hour)
`humidity`	Humidity (percentage)
`temp`	Daily high temperature (degrees Farenheit)
`ibh`	Height of the bottom of the inversion layer over LAX airport (feet)
`dpg`	Difference in atmospheric pressure between LAX and Daggett, California (mm Hg)
`ibt`	Temperature at the base of the inversion layer over LAX (degrees Farenheit)
`vis`	Visibility at LAX (miles)
`doy`	Day of the year

(Look up "inversion layers" if you don't know what they are or what they have to do with smog.)

(a) Divide the data at random into a training set and a validation set, the latter containing 10% of the data.

(b) Use your code to fit an additive model to the training set, regressing `O3` on `temp`, `ibh` and `ibt`. Plot the partial response functions, and describe, in words, the relationships they imply between the three input features and the response. How big is the root mean squared error? Plot the residuals (not partial residuals) against the input features: do they look independent? Include all the commands you run.

(c) Use your code to fit another additive model for `O3`, regressing it on all the other features except `doy`. Repeat the analysis for the previous part.

(d) Which model does a better job of predicting the testing data?

5. Extra credit Modify your fitting function so its returned object has the class `"am"`. (See `help(class)`.)

   (a) Write `fitted.am` and `residuals.am` functions. These should take as arguments `am`-class objects, and return the appropriate vectors. Verify that `fitted` and `residuals` work properly when called on `am` objects.

   (b) Write `print.am` and `summary.am` functions. You should decide what information they give; look at the corresponding functions for `lm` and `npreg`.

   (c) Write a `plot.am` function to automate plotting of the partial response functions. Ideally, the user should be able to plot any arbitrary subset of the partials, with the function setting up the graphics device appropriately. (Look at the plots for GAMs in lecture 21, thought it doesn't have to look exactly like that.)

   (d) Re-write your `am` function so that it stores a list of p partial response *functions*, in addition to the $n \times p$ partial response *values*. Re-write `predict.am` so that it uses these functions. (You may want to re-write `centered.smoothing` as well.) Verify that it still works.