UWEO StatR301 – Spring 2013: Homework 2

Due: Sunday April 28, 8 PM (grace period 4 days)

Assaf Oron, *assaf@uw.edu*

Reading related to this assignment: Lectures 2-3; Lab 2; Dobson and Barnett Chapter 12-14; Hoeting et al. 1999 (article uploaded to HW2 folder).

Instructions:

- Please submit online in the class dropbox. Either ***.pdf is accepted as the main submission (with code pasted verbatim), or *.rmd.** .
- **Starred (*) questions and question-parts are not required.** You may submit them if you choose, or do any part of them without submitting.
- **Grading is determined chiefly by effort, not by correctness.** If your submission shows evidence of independent, honest effort commensurate with the amount of homework assigned – you will receive full credit.

1. Bayesian Model Averaging with the BMA Package.

a. Download the 'autoMPGtrain.csv' file (if you don't already have it; it's the same one from StatR201). Pre-process the dataset via the code from Lecture 3, or per your own choice from StatR201 HW4.

In class I showed BMA using the `bicreg` function. That function is antiquated, not having a `formula` interface, etc. But in fact, an identical functionality – and with all modern R amenities – is available via the `bic.glm` function **on the very same package.**

Run BMA on the training dataset, using `bic.glm` and a model `formula` (tip: either specify all participating variables explicitly, or use `'~.'` and be careful to exclude those columns that shouldn't participate in the X matrix). You will need to add the argument `"glm.family=gaussian()".` Obtain the same model-selection and posterior probability summaries (of models and of variables) shown in class, and compare them with the `bicreg` results. Is the resulting suite of models identical to those obtained from `bicreg?` If not, which one seems more aggressive in excluding "bad" models?

b. As mentioned in class, the way BMA selects a small fraction of the thousands of available models, is by constructing an **"Occam's Window:"** a tolerance threshold telling how much worse than the "best" model, are the other models allowed to be in order to be selected into the average. "Best" is measured via BIC (likelihood penalized by model size). In addition, not every single possible model is fit – rather, the search in model space is carried out via **"modified leaps and bounds"** (an accelerated version of stepwise model selection; hereafter just 'leaps'). There's a variety of tuning parameters:

`prior.param`	a vector of values specifying the prior probability for each variable to be in the model (default 0.5 for all).
`OR`	The width of the final Occam's window (in likelihood units; default 20)

`occam.window`	a logical value specifying if Occam's window should be used (default `TRUE`).
`OR.fix`	How much wider than the final window, is the temporary "Occam's Window" allowed to be during leaps (default 2).
`nbest`	The (maximum?) number of models of each size returned by leaps (default 150).
`maxCol`	The size of the largest allowed model (as columns of the X matrix; default 30).
`factor.type, factor.prior. adjust`	Logical values determining how to treat factors. `factor.type=FALSE` allows including/excluding individual factor-level indicators, a.k.a. "dummy variables" (individual columns of the X matrix), while `TRUE` (default) includes/excludes **only the entire factor.** If you choose `factor.type=FALSE`, it is recommended to set `factor.prior.adjust=TRUE,` to avoid artificially upweighting the factor compared with other variables.

Look up the the help for `bicreg` and determine which of those tuning options also exist for that older function – and which not. For the common tuning parameters, are the defaults the same for both functions? Try to run both (just on the training set) making the tuning options identical. Are the results now the same?

c. Arguably, the chief tuning parameters are related to Occam's window. Split the training dataset into 5 cross-validation groups, and tune `OR` for the values 5, 10, 20 (default), 50, 100 – as well as no window at all (the table above explains how to do the last option). Then code and run a CV routine. **Note that conveniently, `bic.glm` does have a `predict` method. Use it!**

Choose the best OR value via RMSE (CV root-mean-square-error). Take the "winning" value and fit it on the entire training dataset. If this option is different than the default (OR=20) already reported in (a), then report the same summaries (if the number of participating models in the final average is large, you don't have to print out the entire '`which`' matrix – just write down its size). **Finally, download the 'autoMPGtest.csv' dataset, pre-process it, then predict and score the "winning" model (as it was fit on the training dataset), including an observed vs. predicted scatter plot.**

d*. Rather than 1D tuning, add to the mix `OR.fix` and `factor.type,` and repeat (c) doing 3D tuning. There's still probably no need for parallel processing etc., since the individual runs are really fast. Do you get a different "winning" model? Does it really improve final test-set predictions?

e*. Remember, the reason I recommended modeling the inverse to MPG rather than MPG directly, was that the latter was nonlinear in the dominant covariates. **Repeat (c), but use `mpg` as your outcome rather than `gp100m -`** and also, feed to BMA a larger X matrix that includes the inverse (or other) transformations of the main covariates (`weight`, etc.), together with the untransformed covariates (so using both simultaneously). Does BMA prefer one or the other (original or transformed covariates), or does it "mix and match"? Do the test-set predictions improve?