# UWEO StatR301 Homework 2 Key


## Assaf Oron, May 2013


# Overview

Bayesian Model Averaging is a generic and powerful prediction method. It is also far less associated with the controversies or pitfalls of Bayesian methods, because (partial list):

1. There is no standard frequentist recipe for model selection that would be challenged by BMA. So it is perceived like most other Bayesian methods **should** be perceived (IMHO): another available option on the menu.
2. The Bayesian "machinery" involved is simple, straightforward and transparent
3. It makes sense to most people
4. It works

That being said, as a rule there is no magic solution to the model-selection problem, and BMA is not an exception to that rule.

**To understand the end result of BMA, let us examine what non-Bayesian methods it most closely resembles.** How does BMA predict?

- It takes an average of models, weighted by their calculated posterior probabilities of being the *"right"* model.
- Each of these models uses a **selection** of the available covariates.
- **The final prediction is a linear combination of covariates – with the coefficients $\tilde{\beta}$ being different than the $\hat{\beta}$'s of a model fitted directly on them, and quite possibly $\tilde{\beta} = 0$ for some covariates that do not make it to any of the nonzero-posterior models.**

So the BMA prediction is very similar in its final form, to **predictions from the Lasso/Ridge family.** The route to get there is a bit different (Lasso/Ridge work directly on the covariate estimates, while BMA works mostly on complete models). Note: PCA/PLS predictions also boil down to linear combination of covariates, but generally speaking covariates are never "turned off" as easily as with BMA and Lasso.

# 1a. Naive Comparison of `bicreg` and `bic.glm`

```
library(BMA); options(width=132)
```

```
## Warning: package 'BMA' was built under R version 3.0.1
```

```
autos=read.csv('../Datasets/autoMPGtrain.csv',as.is=TRUE)
autos$gp100m=100/autos$mpg
autos$continent=factor(autos$continent)
autos$name=tolower(autos$name)
autos$diesel=grepl('diesel',autos$name)
autos$diesel[autos$name=="mercedes-benz 240d"]=TRUE
autos$wagon=grepl('[()]sw[)]',autos$name)
autos$cylgroup=cut(autos$cyl,c(2,5.5,6.5,9))

bma1=bicreg(autos[,c(3:8,11:13)],autos$gp100m)
round(bma1$postprob,2)
```

```
##  [1] 0.24 0.23 0.15 0.10 0.04 0.03 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.01 0.01 0.01 0.01 0.01
0.01
```

```
bma1$which
```

```
##       volume    hp weight accel year continent2 continent3 dieselTRUE wagonTRUE
cylgroup.5.5.6.5. cylgroup.6.5.9.
##  [1,]  FALSE  TRUE   TRUE  TRUE TRUE      FALSE      FALSE       TRUE     FALSE
FALSE           FALSE
##  [2,]  FALSE  TRUE   TRUE FALSE TRUE      FALSE      FALSE       TRUE     FALSE
FALSE           FALSE
##  [3,]  FALSE  TRUE   TRUE  TRUE TRUE      FALSE      FALSE       TRUE     FALSE
TRUE           FALSE
##  [4,]  FALSE  TRUE   TRUE FALSE TRUE      FALSE      FALSE       TRUE     FALSE
TRUE           FALSE
##  [5,]  FALSE  TRUE   TRUE  TRUE TRUE      FALSE      FALSE       TRUE     FALSE
TRUE            TRUE
##  [6,]  FALSE  TRUE   TRUE  TRUE TRUE      FALSE       TRUE       TRUE     FALSE
FALSE           FALSE
##  [7,]  FALSE  TRUE   TRUE FALSE TRUE      FALSE       TRUE       TRUE     FALSE
FALSE           FALSE
##  [8,]  FALSE FALSE   TRUE FALSE TRUE      FALSE      FALSE       TRUE     FALSE
FALSE           FALSE
##  [9,]  FALSE  TRUE   TRUE  TRUE TRUE      FALSE      FALSE       TRUE      TRUE
FALSE           FALSE
## [10,]  FALSE  TRUE   TRUE FALSE TRUE      FALSE      FALSE       TRUE      TRUE
FALSE           FALSE
## [11,]  FALSE  TRUE   TRUE  TRUE TRUE       TRUE      FALSE       TRUE     FALSE
FALSE           FALSE
## [12,]   TRUE  TRUE   TRUE  TRUE TRUE      FALSE      FALSE       TRUE     FALSE
TRUE            TRUE
## [13,]  FALSE  TRUE   TRUE FALSE TRUE       TRUE      FALSE       TRUE     FALSE
FALSE           FALSE
## [14,]  FALSE  TRUE   TRUE FALSE TRUE      FALSE      FALSE       TRUE     FALSE
TRUE            TRUE
## [15,]   TRUE  TRUE   TRUE FALSE TRUE      FALSE      FALSE       TRUE     FALSE
FALSE           FALSE
## [16,]  FALSE  TRUE   TRUE  TRUE TRUE      FALSE      FALSE       TRUE     FALSE
FALSE            TRUE
## [17,]   TRUE  TRUE   TRUE  TRUE TRUE      FALSE      FALSE       TRUE     FALSE
FALSE           FALSE
## [18,]  FALSE  TRUE   TRUE FALSE TRUE      FALSE      FALSE       TRUE     FALSE
FALSE            TRUE
## [19,]  FALSE  TRUE   TRUE  TRUE TRUE      FALSE       TRUE       TRUE     FALSE
TRUE           FALSE
```

This was just a repeat of the chunk appearing in Lecture 3 notes. Now for the more advanced `bic.glm`:

```
bma2=bic.glm(gp100m~.,data=autos[,c(3:8,10:13)],glm.family=gaussian())
round(bma2$postprob,2)
```

```
##  [1] 0.40 0.35 0.07 0.03 0.03 0.03 0.02 0.02 0.02 0.02
```

```
colnames(bma2$which)=bma2$namesx
bma2$which
```

```
##       volume    hp weight accel year continent dieselTRUE wagonTRUE cylgroup
##  [1,]  FALSE  TRUE   TRUE  TRUE TRUE     FALSE       TRUE     FALSE    FALSE
##  [2,]  FALSE  TRUE   TRUE FALSE TRUE     FALSE       TRUE     FALSE    FALSE
##  [3,]  FALSE  TRUE   TRUE  TRUE TRUE     FALSE       TRUE     FALSE     TRUE
##  [4,]  FALSE  TRUE   TRUE  TRUE TRUE     FALSE       TRUE      TRUE    FALSE
##  [5,]  FALSE  TRUE   TRUE FALSE TRUE     FALSE       TRUE      TRUE    FALSE
##  [6,]   TRUE  TRUE   TRUE  TRUE TRUE     FALSE       TRUE     FALSE     TRUE
##  [7,]  FALSE  TRUE   TRUE FALSE TRUE     FALSE       TRUE     FALSE     TRUE
##  [8,]   TRUE  TRUE   TRUE  TRUE TRUE     FALSE       TRUE     FALSE    FALSE
##  [9,]   TRUE  TRUE   TRUE FALSE TRUE     FALSE       TRUE     FALSE    FALSE
## [10,]  FALSE FALSE   TRUE FALSE TRUE     FALSE       TRUE     FALSE    FALSE
```

As all of you have found, the `bic.glm` default is more aggressive or **parsimonious** then `bicreg`'s. Irritatingly, the `which` matrix in `bic.glm` does not have column names – but as I did above, those can be appended using the `namesx` vector (which does exist, even though the help page doesn't mention it; I guessed it might be there b/c it's there with `bicreg`).

# 1b: A More Detailed Comparison

All arguments from `bicreg` (except `x,y`) are matched by `bic.glm` arguments, and all of them retaining the same name except `drop.factor.levels` which changes to `factor.type` and also reverses its default behavior.

In addition, `bic.glm` has (besides `formula`)

- `glm.family` and `dispersion`, allowing for the standard spectrum of GLMs rather than only linear regression
- `prior.param` allowing for different prior probabilities on different covariates
- `OR.fix` allowing for tuning the relative parsimony of individual leaps-and-bounds iterations
- `factor.prior.adjust` which similarly enables fine-tuning of the treatment of multi-level factors
- `occam.window` which turns off parsimony almost completely, making the end result more similar to ridge regression than to La sso.

That being said, except for the GLM expansion none of these added knobs are really earth-shattering.

When it came to making the two BMA flavors as identical as possible (really, in order to examine whether they in fact call the exact same routine or not) – the class split into two camps. So let's check it out together. Here's the closest I got:

```
bma12=bicreg(autos[,c(3:8,11:13)],autos$gp100m,drop.factor.levels=FALSE)
round(bma12$postprob,2)
```

```
##  [1] 0.24 0.23 0.15 0.10 0.04 0.03 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.01 0.01 0.01 0.01 0.01
0.01
```

```
round(bma12$postprob-bma1$postprob,3) # no change
```

```
##  [1] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

```
table(bma12$which==bma1$which)
```

```
##
## TRUE
##  209
```

Aha! The `drop.factor.levels` switch doesn't work: the function still returns models with some factor levels in and others out. So we cannot make `bicreg` behave like the `bic.glm` default. As they say, it's not a feature, it's a bug.

Anyway, let's try the other direction:

```
bma22=bic.glm(gp100m~.,data=autos[,c(3:8,10:13)],glm.family=gaussian(),factor.type=FALSE,nbest=10)

round(bma22$postprob,2) # here we increase from 10 to 19 models,..
```

```
##  [1] 0.25 0.22 0.15 0.10 0.04 0.03 0.02 0.02 0.02 0.02 0.02 0.02 0.02 0.01 0.01 0.01 0.01 0.01
0.01
```

```
round(bma22$postprob-bma1$postprob,3) # but not quite identical results...
```

```
##  [1]  0.006 -0.008  0.005  0.001  0.000  0.001  0.000  0.000 -0.001  0.000 -0.001 -0.001
-0.001  0.000  0.000  0.000  0.000  0.000
## [19]  0.000
```

```
table(bma22$which==bma1$which) # some models might be switched in order?
```

```
##
## FALSE  TRUE
##    32   177
```

My speculation is that the two functions still call the same fitting algorithm, but that there are some hard-coded tuning knobs (especially in the older `bicreg`) that make the results a bit different no matter how hard we try to make them identical. I would definitely recommend using `bic.lm` over `bicreg`.

Overall, this package is 1990s vintage and not very well documented. On the other hand, the methodology is straightforward and therefore the performance is stable. Of course, there are newer BMA options in R town and there will be still newer ones as time goes by; just look them up when you actually need to use this tool.

Trivia bit: one of the chief authors of the BMA package, Chris Volinsky, is co-leader of the team that won the Netflix predictive-modeling challenge a few years ago.

# 1c: Tuning `bic.glm`

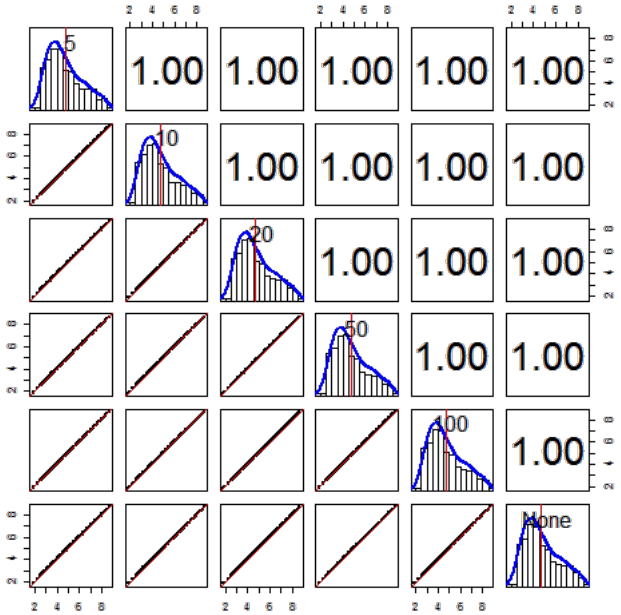
First, some prep - a core CV function, as well as this simple trick to set equal-sized CV groups:

```
n=dim(autos)[1]
cvgroups=sample(rep(1:5,60),size=n)
table(cvgroups)
```

```
## cvgroups
##  1  2  3  4  5
## 60 60 60 59 59
```

```
bicglmCV=function(formula,dataset,cvg,...)
{
  n=dim(dataset)[1]
  k=unique(cvg)
  outpreds=rep(NA,n)
  for (b in k)
  {
    tmpfit=bic.glm(f=formula,data=dataset[cvg!=b,],...)
    outpreds[cvg==b]=predict(tmpfit,newdata=dataset[cvg==b,],...)
  }
  return(outpreds)
}
```

```
tunepreds1=matrix(NA,nrow=n,ncol=6)
orvals=c(5,10,20,50,100)
colnames(tunepreds1)=c(orvals,"None")
source('../Code/enhancedGraphics.r')
for (a in 1:5)
tunepreds1[,a]=bicglmCV(gp100m~.,dataset=autos[,c(3:8,10:13)],glm.family=gaussian(),OR=orvals[a],cvg=cvg

tunepreds1[,6]=bicglmCV(gp100m~.,dataset=autos[,c(3:8,10:13)],glm.family=gaussian(),occam.window=FALSE,c

pairsPlus(tunepreds1)
```




```
rmse=function(x,ref) sqrt(mean((x-ref)^2))
round(apply(tunepreds1,2,rmse,ref=autos$gp100m),4)
```

```
##      5     10     20     50    100   None
## 0.5652 0.5674 0.5669 0.5670 0.5672 0.5672
```

# 1d: Multi-Dimensional Tuning

So here's the problem: at least for this dataset, `bic.glm` doesn't appear to be very tunable. I tried various combinations of tuning parameters, such as this one:

```
tunepreds3=matrix(NA,nrow=n,ncol=12)
ort=expand.grid(c(5,20,80),c(TRUE,FALSE),c(TRUE,FALSE))
for (a in 1:12)
tunepreds3[,a]=bicglmCV(gp100m~.,dataset=autos[,c(3:8,10:13)],glm.family=gaussian(),OR=ort[a,1],factor.t

round(apply(tunepreds3,2,rmse,ref=autos$gp100m),4)
```

```
##  [1] 0.5652 0.5669 0.5672 0.5646 0.5661 0.5665 0.5652 0.5669 0.5672 0.5636 0.5653 0.5657
```

For tuning to work properly, we need to see an impact upon the 1st significant digit, not the 3rd. The variability between various tuning settings here, is demonstrably smaller than the variability between different CV group randomizations. How do I know? **Because different students got different answers for the "best" setting, running essentially the same code. I've seen that too by re-running the same code several time.**

The lesson here goes beyond the narrow HW problem: don't just take the tuning output (or any numerical output, for that matter) and run to the next step.

In our case, I'm not sure what's the leading cause here -

- BMA itself?
- This particular implementation of BMA?
- Or just the dataset that is a bit funky?

  …but we've seen greater variations in performance when tuning various stepwise/CV/Lasso methods. If you consider using BMA in the future, this is something you need to inspect. Note that we haven't actually touched the more **Bayesian** tuning elements, such as the prior (the default is a $50\%$ prior probability for each column of $\mathbf{X}$).

All that being said, the second tuning round suggests that setting `factor.type=FALSE` (these are parameter combinations 4-6 and 10-12 above; recalling 1b, that's the `bicreg` default) – is a bit more effective for this dataset than not.

So I'll use that to predict.

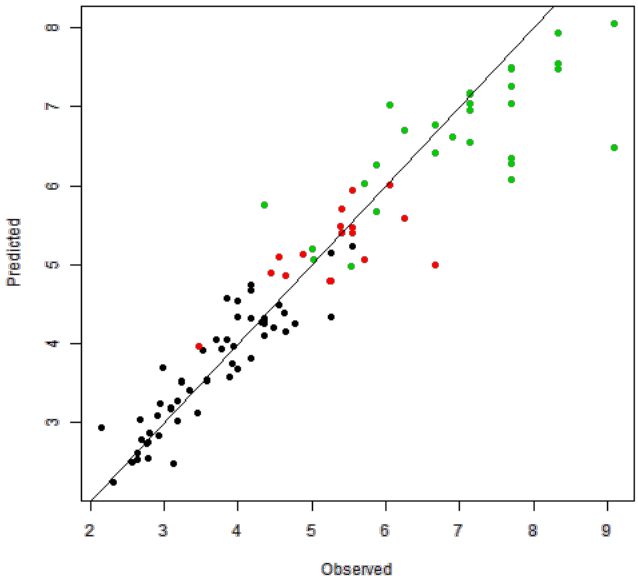
```
autest=read.csv('../Datasets/autoMPGtest.csv',as.is=TRUE)
autest$gp100m=100/autest$mpg
autest$continent=factor(autest$continent)
autest$name=tolower(autest$name)
autest$diesel=grepl('diesel',autest$name)
autest$diesel[autest$name=="mercedes benz 300d"]=TRUE
autest$wagon=grepl('[()]sw[)]',autest$name)
autest$cylgroup=cut(autest$cyl,c(2,5.5,6.5,9))

bma3=bic.glm(gp100m~.,data=autos[,c(3:8,10:13)],glm.family=gaussian(),factor.type=FALSE)
dim(bma3$which)
```

```
## [1] 19 11
```

```
bmapreds=predict(bma3,newdata=autest)
rmse(bmapreds,autest$gp100m)
```

```
## [1] 0.5677
```

```
plot(autest$gp100m,bmapreds,xlab="Observed",ylab="Predicted",pch=19,col=autest$cylgroup)
abline(0,1)
```




Again, decent predictions overall, but an apparent lack of effective tuning options – at least for this dataset. I would inspect a couple of other datasets before ruling that this version of BMA is less tunable than other prediction methods.