Fitting High-dimensional Regressions using shrinkage and variable selection methods

G. de los Campos

11/29/2020

In this note we consider three approaches to build a prediction model for a problems involving a large number of predictors. The first approach selects predictors using a marginal association test (i.e., testing the association of the response and each predictor, one predictor at a time). The other two approaches (Elastic Net and Bayesian Variable selection) consider all predictors jointly.

For the example we use a data set available in the BGLR R-package. This data set provides four phenotypes (see object `wheat.Y`) for 599 wheat lines that were genotyped at 1,279 genetic markers (see object `wheat.X`).

Loading the data

This code below loads the data, center, and scales the the genotypes. While centering and scaling is not strictly needed, it is often a good practice when using penalized (e.g., Lasso) or Bayesian regressions.

```
library(BGLR)
data(wheat)
head(wheat.Y)
```

```
##               1           2           4          5
## 775   1.6716295 -1.72746986 -1.89028479  0.0509159
## 2166 -0.2527028  0.40952243  0.30938553 -1.7387588
## 2167  0.3418151 -0.64862633 -0.79955921 -1.0535691
## 2465  0.7854395  0.09394919  0.57046773  0.5517574
## 3881  0.9983176 -0.28248062  1.61868192 -0.1142848
## 3889  2.3360969  0.62647587  0.07353311  0.7195856
```

```
dim(wheat.X)
```

```
## [1]  599 1279
```

```
X=scale(wheat.X,center=TRUE,scale=TRUE)
y=wheat.Y[,2] # picks one phenotype

N<-nrow(X) ; p<-ncol(X)
```

We will compare models based on their ability to predict data that was not used to fit the models. The following code produces a training-testing partition that we will use for all mehtods.

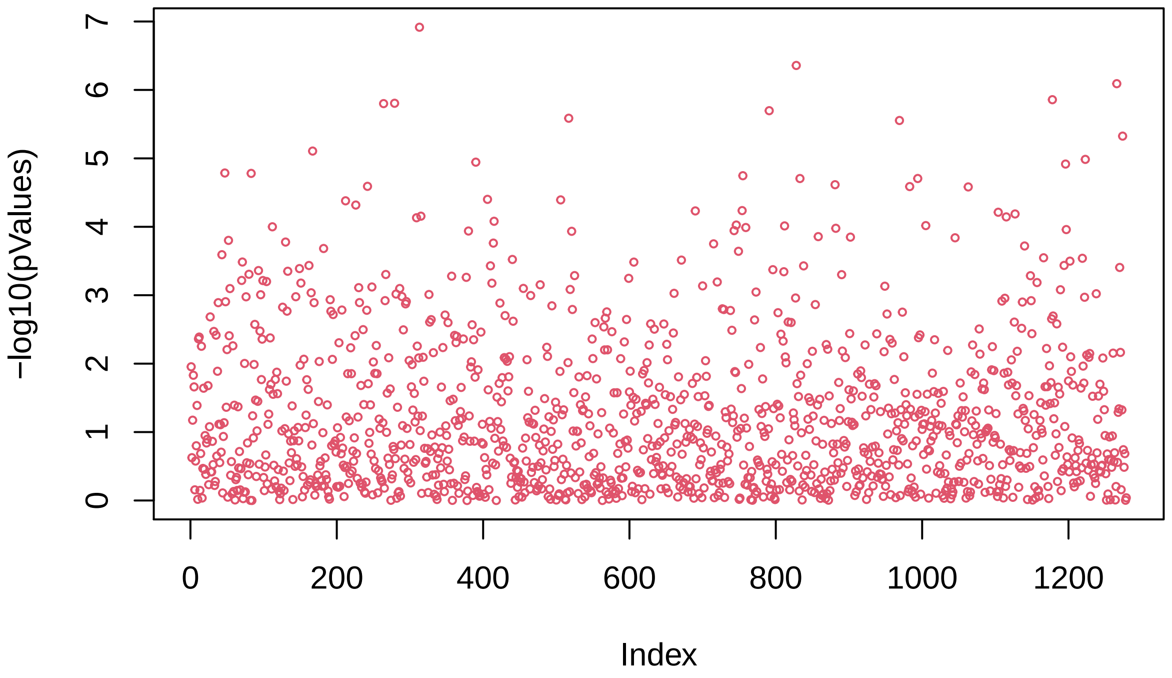
Creating a Training-Testing partition

```
set.seed(12345)
tst<-sample(1:N,size=150,replace=FALSE)
XTRN<-X[-tst,]
yTRN<-y[-tst]
XTST<-X[tst,]
yTST<-y[tst]
```

1 Variable screening

Let's first rank predictors based on a marginal association test. Note that screening is done within the training data only/

```
pValues<-numeric()
for(i in 1:p){
   fm<-lsfit(y=yTRN,x=XTRN[,i])
   pValues[i]<-ls.print(fm,print.it=F)$coef[[1]][2,4] # extracts p-value, similar to lm() but a bit fa
}

plot(-log10(pValues),cex=.5,col=2)
```

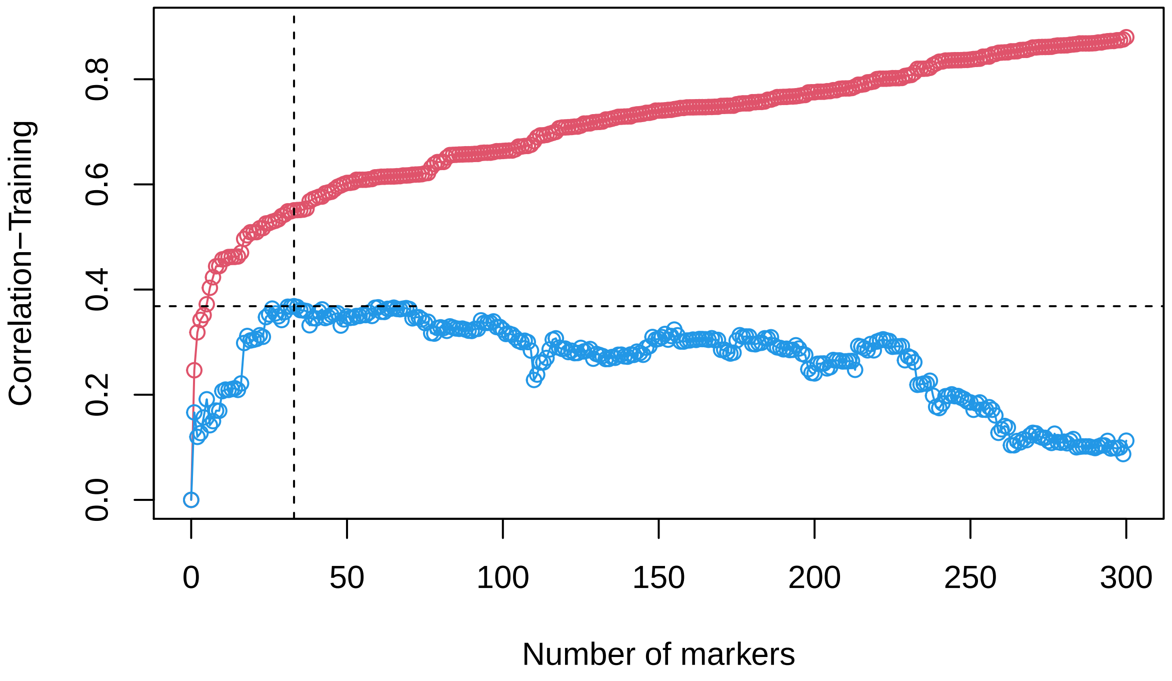


Let's now build models using the top-q (q=1,. . . ,300) markers. The script: - Ranks markers based on p-values (from smallest to largest). - Fit models using the top 1, top 2, . . . , top-q markers using data from the training set. - For each of the fitted model the script computes the correlation between phenotype and predictions within the training data and in the testing data.

Building prediction models using the top-q markers

```
####### VARIABLE SELECTION ##############################
 mrk_rank<-order(pValues); corTRN<-numeric(); corTST<-numeric()
 for(i in 1:300){
    tmpIndex<- mrk_rank[1:i]
    ZTRN=XTRN[,tmpIndex,drop=F]
    ZTST=XTST[,tmpIndex,drop=F]

    fm<-lm(yTRN~ZTRN)
    bHat=coef(fm)[-1]
    bHat<-ifelse(is.na(bHat),0,bHat)

    yHatTRN=ZTRN%*%bHat
  corTRN[i]<-cor(yTRN,yHatTRN)
```

```
    yHatTST=ZTST%*%bHat
    corTST[i]<-cor(yTST,yHatTST)

}

plot(c(0,corTRN),x=0:length(corTRN),type='o',col=2,ylab='Correlation-Training',
      xlab='Number of markers',ylim=c(0,.9))

lines(x=0:length(corTST),y=c(0,corTST),col=4)
points(x=0:length(corTST),y=c(0,corTST),col=4)
abline(v=which.max(corTST),lty=2)
abline(h=corTST[which.max(corTST)],lty=2)
```



Remarks

- Goodness of fit in the training data set (`corTRN`, blue) increases with DF
- However prediction accuracy in testing data (`corTST`, red) increases, reaches a plateau, and then decreases

2: Regularized Regression

In the course we have discussed various estimators which are based on evidence conveied by that the data. In particular we have considered methods that estimate parameters by either maximizing the likelihood (ML) function or minimaizing the residual sum of squares. These methods have reasonably good properties (e.g., OLS is unbiased and has minimum variance among the class of linear unbiased estiamtors, ML is asymptotically unbiased and asymptotically efficient). However, the performace of these methods can be sub-optimal when the number of parameters is large relative to sample size. A groundbreaking paper by James & Stein (1961) show that indeed the ML estimator was, in some circumstances, indmisible; they showed that another estimator had uniformily better performance (in the sense of having a smaller Mean-Squared Error: $MSE = Variance + Bias^2$) than the ML estimator. Their estimator shrunks ML estimates towards zero. This form of 'regularization' reduces the variance of estimates and thus has potential to reduce the MSE of estimates.

There are many ways to obtain regularized estimates; two commonly used approaches are penalized and Bayesian methods. We discussed each of them next. In most cases there is a duality by which a penalized estimator can be viewed as the posterior mode from a Bayesian model.

3: Penalized regressions using glmnet

In a penalized regression, estimates are obtained by minimizing a penalized log-likelihood or, in the case of linear models, a penalized residual sum of squares:

$\hat{\beta} = argmin\{(y - X\beta)'(y - X\beta) + \lambda J(\beta)\}$

where $J(\beta)$ is a penalty function. Common choices for the penalty function are the

- L2-norm, $J(\beta) = \sum_j \beta_j^2$ (aka Ridge Regression, Hoerl and Kennard 1970),
- L-1 norm $J(\beta) = \sum_j |\beta_j|$ (aka Lasso, Tibshirani, 1996), and,
- Linear combinations of the two $J(\beta) = (1 - \alpha)\sum_j \beta_j^2 + \alpha \sum_j |\beta_j|$ (aka Elastic Net, Zhou and Hastie, 2005), for some $\alpha \in [0, 1]$.

Choosing $\lambda = 0$ leads Ordinary Least Squares estimates. Ridge regression shrunk OLS estimates towards zero, without making variable selection. Lasso and Elastic Net combine variable selection and shrinkage.

Commonly, these models are fitted over a grid of values of the regularization parameter (λ) and an optimal value is chosen by evaluating the ability of the fitted models to predict data not used to train the models. This is illustrated in the following script.

Fitting the models

```r
library(glmnet)
```

```
## Loading required package: Matrix

## Loaded glmnet 4.0-2
```

```r
# alpha 0 gives Ridge Regression
fmRR=glmnet(y=yTRN,x=XTRN,alpha=0)
dim(fmRR$beta)
```

```
## [1] 1279  100
```

```r
length(fmRR$lambda)
```

```
## [1] 100
```

```r
range(fmRR$lambda)
```

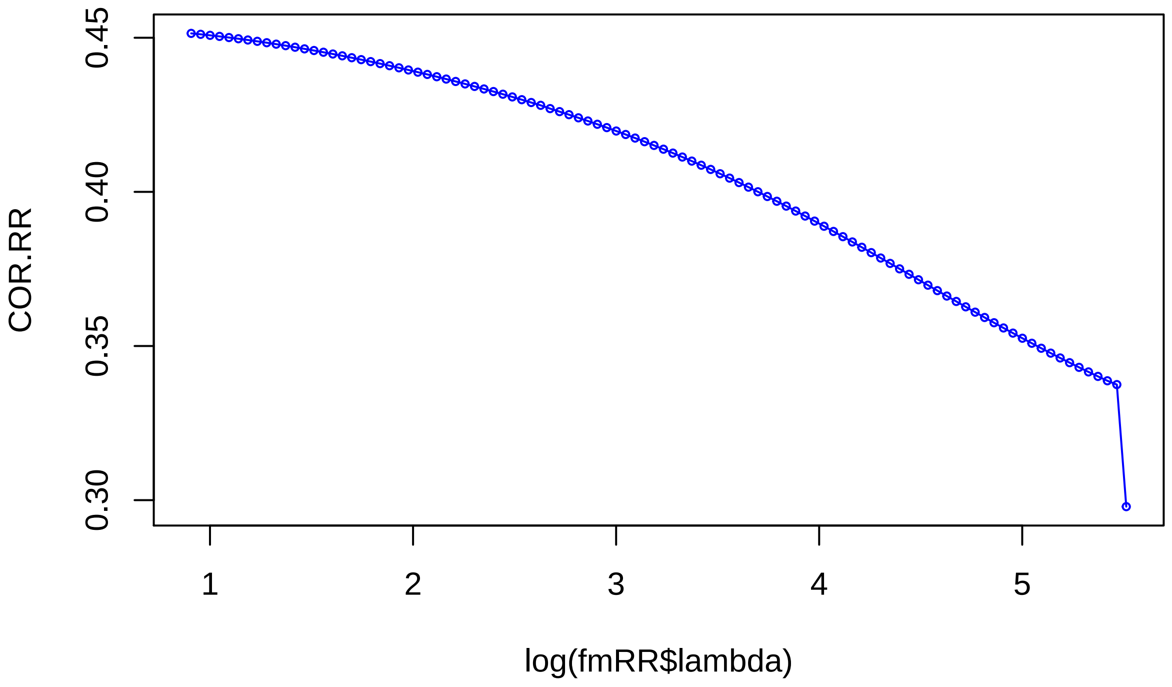
```
## [1]   2.477649 247.764898
```

```r
# alpha 1 gives Lasso
fmL=glmnet(y=yTRN,x=XTRN,alpha=1)

# alpha between 0 and 1 gives elastic net
fmEN=glmnet(y=yTRN,x=XTRN, alpha=0.5)


COR.RR=rep(NA,100)
COR.L=rep(NA,100)
COR.ENet=rep(NA,100)

# evaluating correlation in TST set
for(i in 1:100){
  COR.RR[i]=cor(yTST,XTST%*%fmRR$beta[,i])
  COR.L[i]=cor(yTST,XTST%*%fmL$beta[,i])
  COR.ENet[i]=cor(yTST,XTST%*%fmEN$beta[,i])
}
```

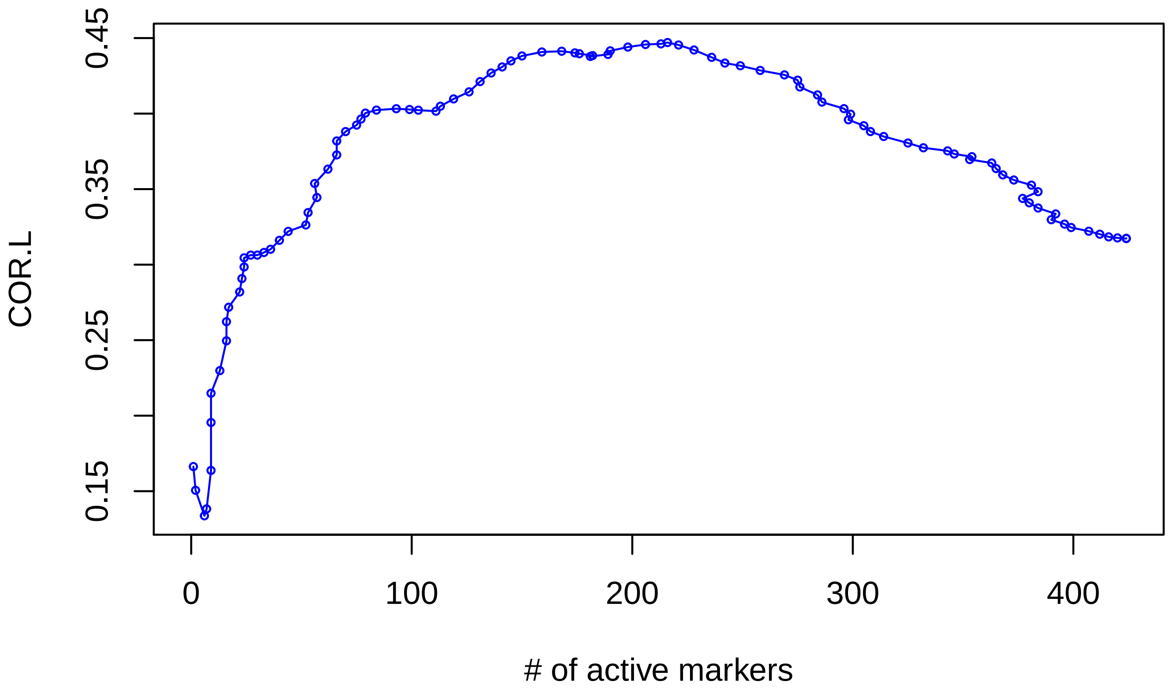
```
## Warning in cor(yTST, XTST %*% fmL$beta[, i]): the standard deviation is zero
```

```
## Warning in cor(yTST, XTST %*% fmEN$beta[, i]): the standard deviation is zero
```

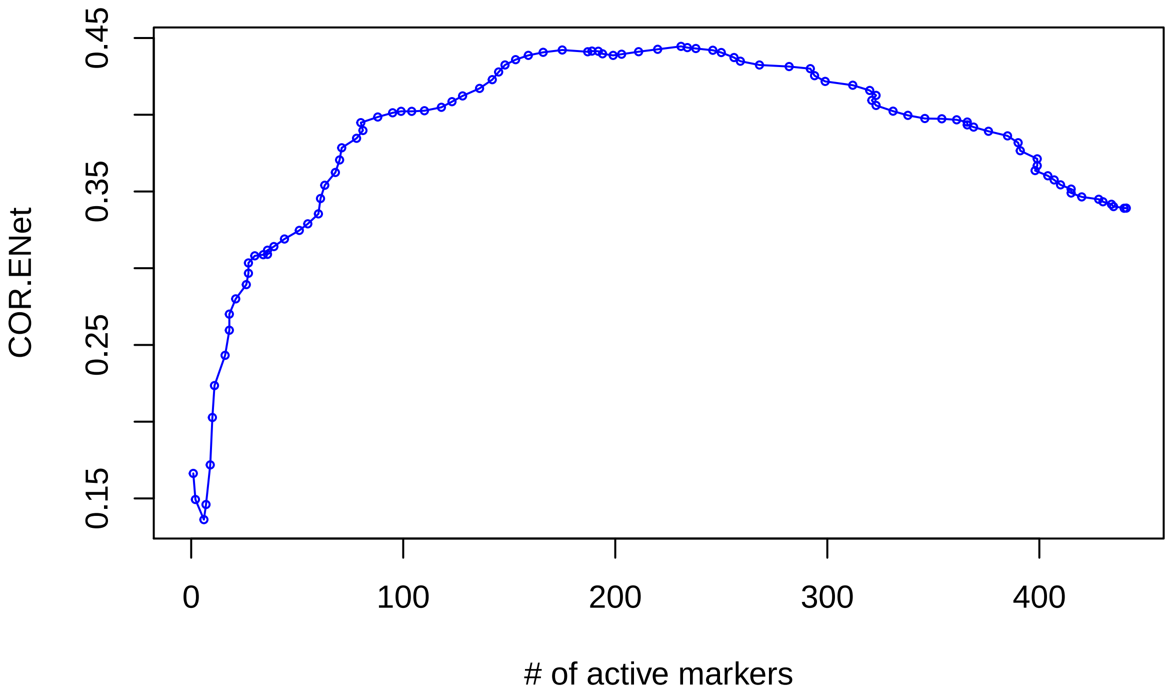
```
plot(COR.RR,x=log(fmRR$lambda),type='o',col='blue',cex=.5)
```



```
plot(COR.L,x=fmL$df,type='o',  col='blue',cex=.5,xlab='# of active markers')
```



```r
plot(COR.ENet,x=fmEN$df,type='o',col='blue',cex=.5,xlab='# of active markers')
```

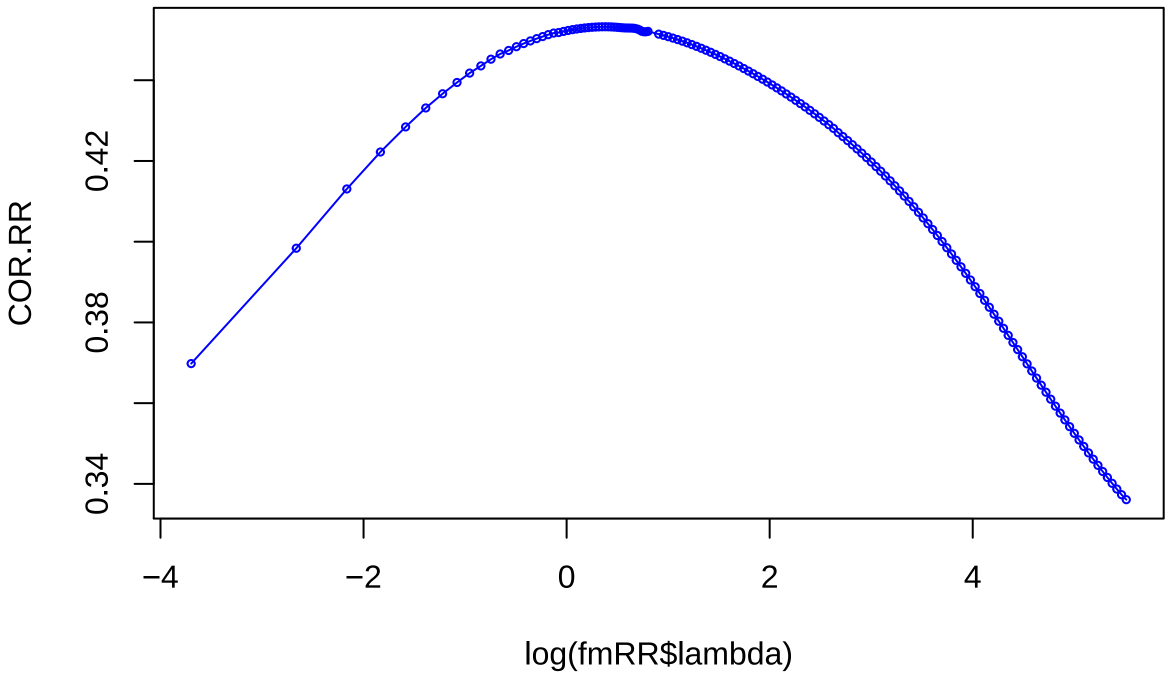


Remarks

- The `glmnet` function fits each of the models over a grid of values of λ, the rules used to choose those values are described in the following study by Friedman, Hastie, and Tibshirani; by default a grid of 100 values.
- The matrix `$beta` has the solutions (estimated effects) obtained for each value of λ‘
- After fitting the model we evaluate prediction accuracy by correlating the testing phenotypes `yTST` with predictions (see loop above).

- In the case of Lasso and Elastic Net the default values of lambda led to an internal maxima, i.e., an internal region with maximum correlation. This is not the case for the Ridge Regression, the grid in this case may need to use smaller values of λ. The following example produce new fits using a user-provided grid of values for the regularization parameter.

```
lambda=c(fmRR$lambda,min(fmRR$lambda)*seq(from=0.9,to=0.01,length=50))
fmRR=glmnet(y=yTRN,x=XTRN,alpha=0,lambda=lambda)

# evaluating correlation in TST set
COR.RR=rep(NA,length(fmRR$lambda))
for(i in 1:length(fmRR$lambda)){
 COR.RR[i]=cor(yTST,XTST%*%fmRR$beta[,i])
}
plot(COR.RR,x=log(fmRR$lambda),type='o',col='blue',cex=.5)
```



- The three models achieve correlations of about 0.45 (Ridge Regression does slightly better)
- This is much better than what we obtained selecting markers based on their marginal association (correlation ~0.37, Example 1).
- Remember that estimates of accuracy, such as the ones discussed above, are point estimates subject to sampling variability; there is sampling variance emerging from the sampling of training and testing data. In the In-class assignment you will be asked to repeat the examples using many training-testing partitions; we will use those results to assess sampling variability on these estimates and also to get a more precise estimate.

4: Bayesian Regressions

In a Bayesian model, estimates are obtained from the posterior distribution of the model uknowns (e.g., regression coefficients). Recall that from Baye's rule we have that $p(A|B) = p(A, B)/p(B) = p(B|A)P(a)/p(B)$. Taking A to be the model parameters (e.g., β in a regression model), and B to be the data (y), we have that

$p(\beta|y) = p(y|\beta)p(\beta)/p(y)$

Above,

- $p(\beta|y)$ is the posterior distribution of the parameters given the data (the object we use to summarize knowledge and uncertainty about model parameters),
- $p(y|\beta)$ is the conditional distribution of the data given the parameters, the likelihood function when viewed as a function of β,
- $p(\beta)$ is the prior distribution of the model uknowns, the object we use to summarize *prior knowledge*, and
- $p(y) = \int p(y|\beta)p(\beta)\, d\beta$

We show here how to fit various Bayesian models using the BGLR R-package.

For additional examples and a description of the methods implemented you can check the following GitHub repository (include multiple examples) and the following manuscript.

2a) Model fitting

```
 library(BGLR)
 nIter=6000 # I set this to small value that way it will run quickly, for more serious analyses use lo
 burnIn=1000 # and longer burnin

# Gaussian prior ("Bayesian Ridge-Regression")
 LP=list( list(X=XTRN,model='BRR') ) # 2-level list, allows specifying different types of random and f
 fmBRR=BGLR(y=yTRN,ETA=LP,nIter=nIter,burnIn=burnIn,saveAt='BRR_',verbose=FALSE)

 # Scaled-t
  LP[[1]]$model='BayesA'
  fmBA=BGLR(y=yTRN,ETA=LP,nIter=nIter,burnIn=burnIn,saveAt='BA_',verbose=FALSE)

 # Double-Exponential
  LP[[1]]$model='BL'
  fmBL=BGLR(y=yTRN,ETA=LP,nIter=nIter,burnIn=burnIn,saveAt='BL_',verbose=FALSE)

 # Spike-slab (Gaussian)
  LP[[1]]$model='BayesC'
  fmBC=BGLR(y=yTRN,ETA=LP,nIter=nIter,burnIn=burnIn,saveAt='BC_',verbose=FALSE)

 # Spike-slab (Scaled-t)
  LP[[1]]$model='BayesB'
  fmBB=BGLR(y=yTRN,ETA=LP,nIter=nIter,burnIn=burnIn,saveAt='BB_',verbose=FALSE)
```

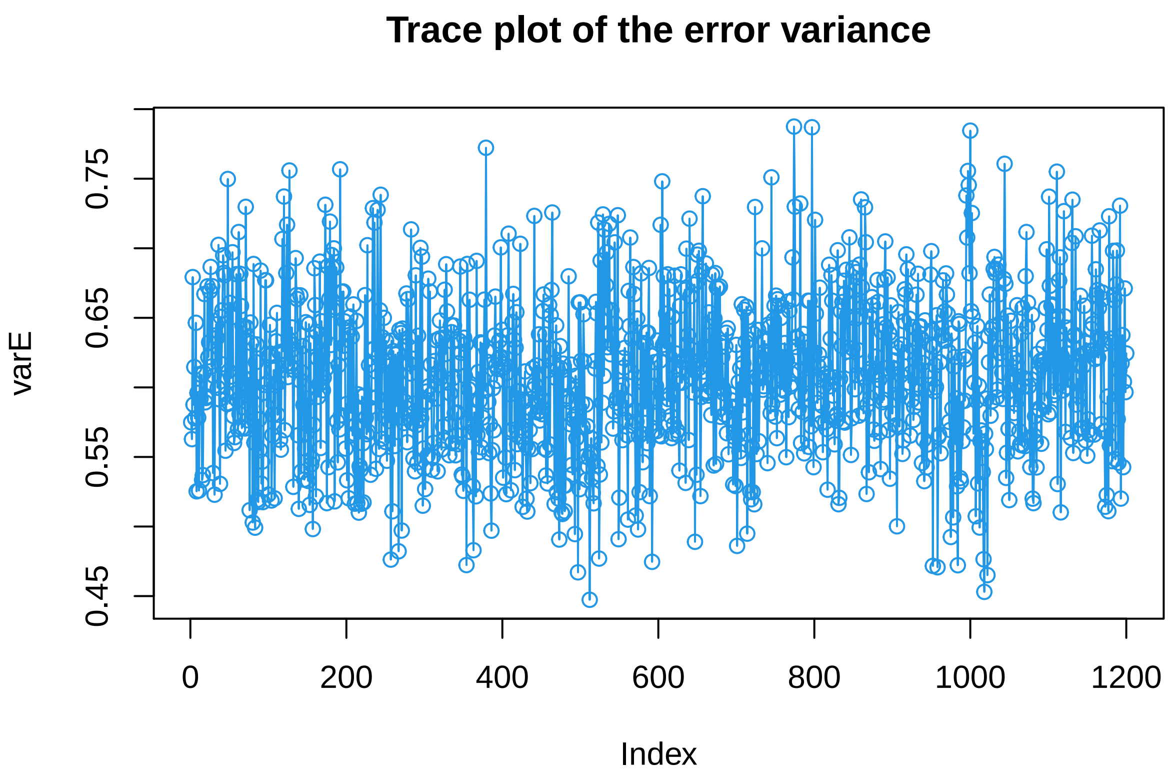
2b) Retrieving samples and estimates

```
 bayes=c(
    'BRR'   =cor( yTST, XTST%*%fmBRR$ETA[[1]]$b),
    'BL'    =cor( yTST, XTST%*%fmBL$ETA[[1]]$b),
    'BayesA'=cor( yTST, XTST%*%fmBA$ETA[[1]]$b),
    'BayesB'=cor( yTST, XTST%*%fmBB$ETA[[1]]$b),
    'BayesC'=cor( yTST, XTST%*%fmBC$ETA[[1]]$b)
 )
 round(bayes,3)
```

```
##    BRR     BL BayesA BayesB BayesC
##  0.447  0.450  0.449  0.450  0.454
```

2c) Retrieving samples and estimates

```
##         775         2166         2465         3881         3889         4248
## -0.76171511  0.40209810  0.10023304 -0.43888630  0.40867352 -0.05090429
## [1] -0.00379885

## [1] 0.6082153

## $logLikAtPostMean
## [1] -468.6342
##
## $postMeanLogLik
## [1] -524.0605
##
## $pD
## [1] 110.8525
##
## $DIC
## [1] 1158.973

## [1] 6000
```



```
##      wPt.0538      wPt.8463      wPt.6348      wPt.9992      wPt.2838
## -0.0006041365 -0.0022878006 -0.0005568744 -0.0003905224 -0.0022636344
##      wPt.8266
##  0.0018327024

##   wPt.0538   wPt.8463   wPt.6348   wPt.9992   wPt.2838   wPt.8266
## 0.01833970 0.01781971 0.01779866 0.01828983 0.01830182 0.01780537

##      wPt.0538      wPt.8463      wPt.6348      wPt.9992      wPt.2838
##  0.0001383523 -0.0033504450 -0.0013478428 -0.0006640690 -0.0014151997
##      wPt.8266
##  0.0005890544

##      wPt.0538      wPt.8463      wPt.6348      wPt.9992      wPt.2838
##  0.0012237992 -0.0024724112 -0.0015929326 -0.0002319299 -0.0009103907
##      wPt.8266
##  0.0001716467
```

```
##      wPt.0538      wPt.8463      wPt.6348      wPt.9992      wPt.2838
##  1.515197e-03 -2.465375e-03 -8.622074e-04  4.229876e-05 -4.897002e-04
##      wPt.8266
##  7.868874e-04
```

```
## [1] 0.468 0.437 0.434 0.478 0.449 0.454
```

```
##      wPt.0538      wPt.8463      wPt.6348      wPt.9992      wPt.2838
##  0.0001496895 -0.0023849425 -0.0007173022  0.0009612699 -0.0021898622
##      wPt.8266
##  0.0002630655
```

```
## [1] 0.448 0.463 0.460 0.450 0.444 0.456
```