Genome Enabled Prediction Methods: Laboratory

Gustavo de los Campos

(gcampos@uab.edu)

Contents

Lab 1: Linear Models

1.1. Linear models and ordinary least squares (45 min) 2

Deriving ordinary least-squares (OLS) estimate using existing R-functions 2

Iterative procedures 4

1.2. The 'Curse' of Dimensionality (45 min) 5

1.3. Confronting the challenges posed by highly dimensional predictors(45 min) 6

Subset selection 7

Shrinkage estimation 8

References 9

LAB 2: Shrinkage Estimation

2.1. Penalized Estimates 2

2.2. Computing RR estimates 5

2.3. Effect of regularization on estimates, goodness of fit and model DF 5

2.4. The Hat Matrix of large-p with small-n genomic regressions as a local smoother 7

2.5. Bayesian View of Ridge Regression 9

2.6. G-BLUP 12

References 14

Lab 3: The Bayesian Alphabet

3.1. The Bayesian Alphabet....2

3.2. Ridge Regression Vs Bayesian Ridge Regression....9

3.3. Bayesian Lasso: fixed versus random lambda....11

3.4. Regression using markers and pedigree....13

References....14

Lab 4:Semi-parametric Genomic Regression Using Reproducing Kernel Hilbert Spaces Methods

4.1. Semi-parametric genome-enabled regression....1

4.2. Reproducing Kernel Hilbert Spaces (RKHS) regressions....2

4.3. Scatter plot smoothing with a Gaussian kernel....4

4.4. Inspecting the Hat Matrix....6

4.5. Bayesian view of RKHS....7

4.6. Genomic-Enabled Prediction Using RKHS....9

4.7. Kernel Averaging....11

4.8. Pedigree + Marker Models....14

References....16

LAB 5: Penalized Neural Networks

5.1. Introduction 2

5.2. Scatterplot smoothing using a penalized NN 5

5.3. Penalized Neural Network Using Pre-selected Markers 6

5.4. Penalized Neural Networks Using Marker-derived Basis Functions as Inputs 7

References 8

LAB 6: Validation Methods

6.1. Introduction 2

6.2. Alternative Validation Schemes 2

6.3. Between sub-population prediction 6

6.4. Across environment prediction using single-trait models 7

References 8

Statistical Methods for Genome-Enabled Prediction,

LAB 1:

Linear Models[1]

(gcampos@uab.edu)

Contents

1.1. Linear models and ordinary least squares (45 min) 2

Deriving ordinary least-squares (OLS) estimate using existing R-functions 2

Iterative procedures 4

1.2. The 'Curse' of Dimensionality (45 min) 5

1.3. Confronting the challenges posed by highly dimensional predictors(45 min) 6

Subset selection 7

Shrinkage estimation 8

References 9

[1] Suggestions made by Daniel Gianola are gratefully acknowledged.

1.1. Linear models and ordinary least squares (45 min)

Consider the following model:

$$y_i = \mu + \sum_{j=1}^{p} x_{ij}\beta_j + \varepsilon_i \qquad i = (1,...,n)$$

where: y_i is the phenotype of the i^{th} individual, μ is an effect common to all individuals (an "intercept"), x_{ij} are covariates (e.g., marker genotypes), β_j is the effect of the j^{th} covariate and ε_i is a model residual. In matrix notation the model is expressed as:

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon} \qquad [1]$$

where: $\mathbf{y} = \{y_i\}$ is a vector of phenotypes, $\mathbf{X} = \{\mathbf{1}, \mathbf{x}_1, ..., \mathbf{x}_p\}$ is an incidence matrix for the vector of regression coefficients, $\boldsymbol{\beta} = (\mu, \beta_1, ..., \beta_p)'$ and $\boldsymbol{\varepsilon} = \{\varepsilon_i\}$ is a vector of model residuals.

The ordinary least squares estimate of $\boldsymbol{\beta}$ is the solution to the following optimization problem:

$$\hat{\boldsymbol{\beta}}_{OLS} = \underset{\arg\min}{} \sum_{i} \left(y_i - \sum_{j} x_{ij}\beta_j \right)^2$$

where $\sum_{i} \left(y_i - \sum_{j} x_{ij}\beta_j \right)^2$ is a residual sum of squares. The first order conditions of [2] are satisfied by $\hat{\boldsymbol{\beta}}_{OLS} = [\mathbf{X}'\mathbf{X}]^{-1}\mathbf{X}'\mathbf{y}$.

Deriving ordinary least-squares (OLS) estimate using existing R-functions. The OLS estimate of $\boldsymbol{\beta}$ can be obtained using the function lm(), which fits a linear model by OLS. Alternatively, we can compute the solution using matrix operations. The code below simulates data for regression [1], and fits the linear model using lm().

Example 1. Deriving Ordinary Least Squares estimates using lm()

```
1   rm(list=ls())
2   ## SIMULATES DATA FOR A LINEAR MODEL
3    set.seed(12345)
4    n<-100
5    p<-6
6    set.seed(12345)
7    X<-matrix(nrow=n,ncol=p,
8              data=rbinom(n=n*p,p=.5,size=1))
9    beta<-rnorm(p,mean=0,sd=2)
10   ERROR<-rnorm(n=n,sd=1,mean=0)
11   y<-124 +X%*%beta+ERROR  # note %*% computes matrix product
12
13  ## FITS THE MODEL USING lm() ##############################
14   fm<-lm(y~X)
15   summary(fm)
16   bHat1<-fm$coeff
    #(continues below)
```

In the system of equations

$$\left[\mathbf{X}'\mathbf{X}\right]\hat{\boldsymbol{\beta}}_{OLS} = \mathbf{X}'\mathbf{y} \qquad [2]$$

we will refer to $\mathbf{C} = \left[\mathbf{X}'\mathbf{X}\right]$ as the matrix of coefficients and to $\mathbf{rhs} = \mathbf{X}'\mathbf{y}$ as the right-hand side of the system. The matrix of coefficients can be computed using `C<-t(X)%*%X`, or, equivalently, `C<-crossprod(X)`. Similarly, the right-hand-side can be computed using `rhs<-t(X)%*%y`, or, equivalently, `rhs<-crossprod(X,y)`. `crossprod()` is usually faster. The system can be solved using the function solve(), as illustrated below.

Example 2. Deriving Ordinary Least Squares Using Matrix Operations

```
1   # (continued from Example 1)
2   ## FITS LINEAR MODEL USING MATRIX OPERATIONS #################
3     X2<-cbind(1,X) ## note a vector of 1s is added type head(X)
4     C<-crossprod(X2)
5     rhs<-crossprod(X2,y)
6     bHat2<-solve(C,rhs)
7   # (continues in Example 3)
```

The matrix of coefficients is symmetric and positive definite. The cholesky decomposition of this matrix (**U**) is an upper-triangular matrix satisfying **C=U'U**. **U** can then be used to invert **C** using `chol2inv()` function (see below). This is usually faster than using function `solve()`. Other factorizations of **C,** such as the eigen-value decomposition, `eigen()`, or the QR decompositions, `qr()`, can also be used to invert **C** as well. An example using the cholesky decomposition of **C** is given below.

Example 3. Inversion of positive definite matrices using the Cholesky factorization

```
# (continued from Ex. 1 and 2)
  X2<-cbind(1,X) # note a vector of 1s is added type head(X)
  C<-crossprod(X2)
  rhs<-crossprod(X2,y)
  U<-chol(C)     # computes the Cholesky decomposition
  CInv<-chol2inv(U) # obtains the inverse from a Cholesky decomp.
  bHat3<-CInv%*%rhs
  # compare bHat1, bHat2, bHat3
  round(cbind(bHat1,bHat2,bHat3),4)
# (continues in example 4)
```

Iterative procedures. In practice, when *p* is large, the system of equation is solved using some type of iterative methods. Here is one possible algorithm. Suppose that we know all but the j^{th} regression coefficient, then, from the data-equation we can write:

$$y_i = \sum_{k=1}^{p} x_{ik}\beta_k + \varepsilon_i$$

$$y_i = \sum_{k\neq j}^{p} x_{ik}\beta_k + x_{ij}\beta_j + \varepsilon_i$$

$$y_i - \sum_{k\neq j}^{p} x_{ik}\beta_k = x_{ij}\beta_j + \varepsilon_i$$

$$\tilde{y}_{i(-j)} = x_{ij}\beta_j + \varepsilon_i \qquad [3]$$

where: $\tilde{y}_{i(-j)} = y_i - \sum_{k\neq j}^{p} x_{ik}\beta_k$ is an off-set formed by subtracting from the original phenotypes the contribution to the conditional expectation of all but the j^{th} predictor, that is $\sum_{k\neq j}^{p} x_{ik}\beta_k$. The OLS estimate of β_j in [3] is simply

$$\hat{\beta}_j = \frac{\sum_i x_{ij}\tilde{y}_{i(-j)}}{\sum_i x_{ij}^2}. \qquad [4]$$

A back-fitting algorithm can then be formed by iterating over regression coefficients using [4]. This is implemented in the following R-code.

- Run the code. How do estimates computed using the above-described algorithm compare with the exact solution?
- Change `nIter` (the number of iterations) from 2 to 30 and compare.

Example 4. Deriving Ordinary Least Squares Using Iterative Procedures

```
# Computes OLS using a back-fitting algorithm
  SSx<-colSums(X2^2)           # the diagonal elements of X'X
  nIter<-2                     # number of iterations of the algorithm
  bHat4<-rep(0,ncol(X2))       # initialvalues bj=zero
  bHat4[1]<-mean(y)            # initial values mu=mean(y)
  e<-y-mean(y)                 # initial model residuals

  for(i in 1:nIter){           # loop for iterations of the algorithm
    for(j in 1:ncol(X2)){      # loop over predictors
        yStar<-e+X2[,j]*bHat4[j]   # forming off-sets
        bHat4[j]<- sum(X2[,j]*yStar)/SSx[j]   # eq. [4]
        e<-yStar-X2[,j]*bHat4[j]   # updates residuals
    }
  }

  # compare bHat1, bHat2, bHat3, bHat4
  round(cbind(bHat1,bHat2,bHat3,bHat4),4)
```

1.2. The 'Curse' of Dimensionality (45 min)

The mean-squared error (MSE) of an estimator is: $MSE(\hat{\theta}) = E\left[(\theta - \hat{\theta})^2\right]$ where θ is the true value of the parameter and $\hat{\theta}$ is the estimator, which is a function of the data (**X** and **y** in the regression example discussed above). The expectation in the MSE formula is taken with respect to all possible samples of data. Commonly **X** is treated as fixed and the expectation is taken only with respect to possible realizations of **y** given **X**.

The MSE can be decomposed in two components: $MSE(\hat{\theta}) = \left[\theta - E(\hat{\theta})\right]^2 + Var(\hat{\theta})$, where $\left[\theta - E(\hat{\theta})\right]$ and $Var(\hat{\theta})$ are the bias and variance of the estimator.

The expectation of the OLS estimate of regression coefficients in [1] is:

$$\begin{aligned} E\left[\hat{\boldsymbol{\beta}}_{OLS} \middle| \mathbf{X}\right] &= \left[\mathbf{X}'\mathbf{X}\right]^{-1}\mathbf{X}'E[\mathbf{y}] \\ &= \left[\mathbf{X}'\mathbf{X}\right]^{-1}\mathbf{X}'E\left[\mathbf{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon}\right] \\ &= \left[\mathbf{X}'\mathbf{X}\right]^{-1}\mathbf{X}'\mathbf{X}\boldsymbol{\beta} + \left[\mathbf{X}'\mathbf{X}\right]^{-1}\mathbf{X}'E[\boldsymbol{\varepsilon}] \\ &= \boldsymbol{\beta} + \left[\mathbf{X}'\mathbf{X}\right]^{-1}\mathbf{X}'E[\boldsymbol{\varepsilon}] \end{aligned}$$

When model [1] holds, $E[\boldsymbol{\varepsilon}] = \mathbf{0}$, therefore: $E\left[\hat{\boldsymbol{\beta}}_{OLS} \middle| \mathbf{X}\right] = \boldsymbol{\beta}$. In words, if the linear model holds, OLS gives unbiased estimates of regression coefficients. The second term of the MSE formula, $Var(\hat{\theta})$, is a frequentist measure of uncertainty and reflects variability of the estimator over repeated sampling. The asymptotic (co)variance matrix of OLS estimates of regression coefficients, given **X**, is,

$Var(\hat{\boldsymbol{\beta}}) = [\mathbf{X}'\mathbf{X}]^{-1}\sigma^2$, where σ^2 is the variance of model residuals. This is also the finite-sample co-variance matrix of estimates under normality. Therefore, the MSE of the estimate of the jth regression coefficient is $C^{jj}\sigma^2$ where C^{jj} is the jth diagonal entry of the inverse of the matrix of coefficients, that is $\mathbf{C}^{-1} = [\mathbf{X}'\mathbf{X}]^{-1}$. This element decreases with sample size. In the following example we study how MSE of estimates of regression coefficients changes with n and p.

Example 5. Effects of *n* and *p* on Mean-Squared Error of OLS estimates

```
rm(list=ls())
n<-seq(from=100,to=300,by=10) # vector defining sample size
p<-seq(from=5,to=80,by=4)     # vector defining number of predictors
x<-rbinom(prob=.5,n=max(p)*max(n),size=1) # sample predictors
X<-matrix(nrow=max(n),ncol=max(p),data=x)
varE<-1
VAR<-matrix(nrow=length(n),ncol=length(p),NA)
colnames(VAR)<-p
rownames(VAR)<-n
for(i in 1:length(n)){ # loop over sample size
   for(j in 1:length(p)){ # loop over number of predictors
     tmpX<-X[1:n[i],1:p[j]]
     C<-crossprod(tmpX)
     CInv<-chol2inv(chol(C))
     VAR[i,j]<-mean(diag(CInv))*varE  #average variance of estimates
   }
}
## plot Variance (equal to MSE in this case) Vs. n and p
persp(z=VAR,x=n,y=p,xlab="Sample Size",
        ylab="Number of Predictors",zlab="MSE(bj)",col=2)
```

NOTE. When p>n, the OLS estimate is not unique because $\mathbf{X}'\mathbf{X}$ is singular. Nevertheless, predictions, $\hat{\mathbf{y}} = \mathbf{X}[\mathbf{X}'\mathbf{X}]^{-}\mathbf{X}'\mathbf{y}$, are unique; here $[\mathbf{X}'\mathbf{X}]^{-}$ is a generalized inverse of $\mathbf{X}'\mathbf{X}$. The function `ginv()` of `library(MASS)` can be used to compute a Moore-Penrose generalized inverse. The function `svd()` can be used to compute the singular value decomposition of **X** from where $\hat{\mathbf{y}}$ can also be computed.

In genomic models *p>n*, because of this, estimation methods other than OLS are required. In the following sections we consider alternative methods.

1.3. Confronting the challenges posed by highly dimensional predictors (45 min)

In this section we discuss two different approaches designed to confront the challenges posed by 'large p with small n' regressions. In the first one (**subset selection**) we design an algorithm to select k out of p (k<=p) predictors; our final model will include only these k predictors. Subset selection is a commonly used practice, and it is based on the idea that 'highly dimensional predictors are dangerous'; therefore, the approach seeks to reduce the number of predictors. The second approach (**shrinkage estimation**) uses all available predictors and confronts the challenges posed by regressions with p>n by using shrinkage estimation methods. We illustrate this approach using ridge regression. In both

examples we use a genomic dataset made available with R-package BLR ('wheat'). This dataset contains 4 phenotypes evaluated in 599 wheat lines that were genotyped for 1,279 markers. In the examples we use 450 lines for training and evaluate the prediction accuracy of each of the methods on the remaining 149 lines (testing).

Subset selection. The problem of selecting *k* out of *p* (*k<p*) predictors can be viewed as a model comparison problem. Ideally, we would fit all possible models and select the one that is best according to some model comparison criterion (e.g., AIC, Akaike Information Criterion, Akaike 1973). In practice, when p is large fitting all possible models is not feasible. Instead model search algorithms are used. A very simple search algorithm consists of regressing the response in each of the predictors one at a time ('single marker regression'). Each of these regressions yields a measure of association between markers and phenotypes (e.g., a p-value). Then, we can form our final model by using the first k predictors ranked according to the association measure. This approach is commonly used in Genome Wide Association Studies (GWAS). The following example fits models with *k* predictors (k=1,…,300) chosen based on the marginal association between markers and phenotypes. The examples use the 'wheat dataset' of the BLR package of R (G. de los Campos and Pérez 2010; Paulino Pérez et al. 2010).

Example 6. Subset selection using p-values derived from single-marker regressions

```
rm(list=ls())
##### DATA ###############################################
 library(BLR)
 data(wheat)
 objects()
 N<-nrow(X) ; p<-ncol(X)
 y<-Y[,2]
 set.seed(1235)
 tst<-sample(1:N,size=150,replace=FALSE)
 XTRN<-X[-tst,] ; yTRN<-y[-tst]
 XTST<-X[tst,] ; yTST<-y[tst]
###### SINGLE MARKER REGRESSIONS ########################
 pValues<-numeric()
 for(i in 1:p){
     fm<-lm(yTRN~XTRN[,i])
     pValues[i]<-summary(fm)$coef[2,4]
     print(paste('Fitting Marker ',i,'.',sep=''))
 }
 plot(-log(pValues,base=10),cex=.5,col=2)
####### VARIABLE SELECTION ##############################
 myRanking<-order(pValues)
 sqCor<-numeric()
 for(i in 1:300){
     tmpIndex<- myRanking[1:i]
     fm<-lm(yTRN~XTRN[,tmpIndex])
     bHat<-coef(fm)[-1] ; bHat<-ifelse(is.na(bHat),0,bHat)
     yHat<-as.matrix(XTST[,tmpIndex])%*%bHat
     sqCor[i]<-cor(yTST,yHat)^2
     print(paste('Fitting Model with ',i,' markers!',sep=''))
 }
 plot(sqCor,type='o',col=2,ylab='Squared Correlation',
       xlab='Number of markers',ylim=c(0,.28))

```

Shrinkage estimation. We have seen that when *n* is small and *p* is large OLS estimates have high variance, and therefore high MSE. In addition, when *p* is large relative to *n*, over-fitting may occur, yielding poor predictive ability. Penalized estimates of regression coefficients are designed to confront these problems. The main idea is to reduce MSE by reducing the variance of the estimator, even at the expense of introducing bias. We will cover penalized estimation procedures in more detail in Lab 2; here we briefly illustrate their performance using Ridge Regression (Hoerl and Kennard 1970). Recall that in the linear model of eq. 1

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon} \qquad [1]$$

the OLS estimates of regression coefficients are the solution to the following systems of equations

$$\left[\mathbf{X}'\mathbf{X}\right]\hat{\boldsymbol{\beta}}_{OLS} = \mathbf{X}'\mathbf{y} \qquad [2]$$

The RR estimates has a very similar form, we simply add a constant to the diagonal of the matrix of coefficients, that is:

$$\left[\ \mathbf{X}'\mathbf{X} + \lambda\mathbf{D}\ \right]\hat{\boldsymbol{\beta}}_{RR} = \mathbf{X}'\mathbf{y} \qquad [5]$$

where λ is a constant and **D** is a diagonal matrix with zero in its first diagonal entry (this, to avoid shrinking the estimate of the intercept) and ones in the remaining diagonal entries and zeroes everywhere else. When either λ equals zero, the solution to the above problem is OLS. Adding a constant to the diagonal entries of the coefficient matrix makes it non-singular and shrinks the estimates of regression coefficients other than the intercept towards zero. This induces bias but reduces the variance of the estimates; in large-p with small-n problems this may reduce MSE of estimates and may yield more accurate predictions. The following R-code computes RR estimates.

Example 7. Ridge Regression

```
MSx<-0
for(i in 1:ncol(XTRN)){ MSx<-MSx+mean((XTRN[,i]-mean(XTRN[,i]))^2)}
 h2<-0.5
 lambda<-round(MSx*(1-h2)/h2)

 TMP<-cbind(1,XTRN)
 C<-crossprod(TMP)
 rhs<-crossprod(TMP,yTRN)
 for(i in 2:ncol(C)){ C[i,i]<-C[i,i]+lambda } #adds a constant to diag
 CInv<-chol2inv(chol(C))
 bHatRR<-crossprod(CInv,rhs)
 yHatRR<-cbind(1,XTST)%*%bHatRR
 tmp<-cor(yHatRR,yTST)^2
 lines(x=c(0,30),y=rep(tmp,2),col=4,lwd=2)
 lines(x=c(150,300),y=rep(tmp,2),col=4,lwd=2)
 text(x=90,y=tmp,label=expression(paste('RR (lambda=',lambda, ')')),col=4 )
```

References

Akaike, H. 1973. Information theory and an extension of the maximum likelihood principle. In *Second international symposium on information theory*, 1:267–281.

de los Campos, G., and P. Pérez. 2010. *BLR: Bayesian linear regression. R package version 1.2*. http://cran.r-project.org/web/packages/BLR/index.html.

Hoerl, A. E, and R. W Kennard. 1970. "Ridge regression: Biased estimation for nonorthogonal problems." *Technometrics* 12 (1): 55–67.

Pérez, Paulino, Gustavo de los Campos, José Crossa, and Daniel Gianola. 2010. "Genomic-Enabled Prediction Based on Molecular Markers and Pedigree Using the Bayesian Linear Regression Package in R." *The Plant Genome Journal* 3 (2): 106-116. doi:10.3835/plantgenome2010.04.0005.

Statistical Methods for Genome-Enabled Prediction,

LAB 2:

Shrinkage Estimation[1]

(gcampos@uab.edu)

Contents

2.1. Penalized Estimates 2
2.2. Computing RR estimates 5
2.3. Effect of regularization on estimates, goodness of fit and model DF 5
2.4. The Hat Matrix of large-p with small-n genomic regressions as a local smoother 7
2.5. Bayesian View of Ridge Regression 8
2.6. G-BLUP 11
References 14

NOTE: In many examples in this lab we use Bayesian methods. In those examples we make inferences based on a relatively small number of samples and this is done due to time constraints. In practice, accurate inferences require much more samples.

[1] Suggestions made by Daniel Gianola are gratefully acknowledged.

2.1. Penalized Estimates

Ordinary least squares (OLS) and Maximum likelihood (ML) are examples of estimation methods in which estimates are derived by maximizing the fitness (as measured by the residual sum of squares or likelihood function) of the model to the training data. When the number of predictors (p) is large relative to sample size (n) this is not a good strategy: estimates can have high mean-squared error (MSE) and over-fitting may occur. Penalized estimates are obtained as the solution to an optimization problem that balances two components: how well the model fits the data and how-complex the model is. The general form of the optimization problem is:

$$\hat{\boldsymbol{\beta}} \underset{\boldsymbol{\beta}}{=} \arg\min \left\{ L(\mathbf{y},\boldsymbol{\beta}) + \lambda J(\boldsymbol{\beta}) \right\} \qquad [1]$$

where, $L(\mathbf{y},\boldsymbol{\beta})$ is a loss function that measure lack of fit of the model to the data, $J(\boldsymbol{\beta})$ is a measure of model complexity and $\lambda \geq 0$ is a regularization parameter controlling the trade-offs between fitness and model complexity.

Ridge Regression (Hoerl and Kennard 1970) is a particular case of [1] and is obtained by setting $L(\mathbf{y},\boldsymbol{\beta})$ to be a residual sum of squares $L(\mathbf{y},\boldsymbol{\beta}) = \sum_{i} \left(y_i - \sum_{j} x_{ij}\beta_j \right)^2$ and $J(\boldsymbol{\beta})$ to be the sum of square of the regression coefficients; typically, some of the regression coefficients (e.g., the intercept) are not penalized; therefore, $J(\boldsymbol{\beta}) = \sum_{j \in S} \beta_j^2$ where *S* define the set of coefficients to be penalized.

$$\hat{\boldsymbol{\beta}} \underset{\boldsymbol{\beta}}{=} \arg\min \left\{ \sum_{i} \left(y_i - \sum_{j} x_{ij}\beta_j \right)^2 + \lambda \sum_{j \in S} \beta_j^2 \right\} \qquad [2]$$

When $\lambda \to \infty$ the solution is $\hat{\boldsymbol{\beta}}_{RR} = \mathbf{0}$. On the other extreme, as $\lambda = 0$ the solution is the OLS estimates of $\boldsymbol{\beta}$. In matrix notation problem [2] can be represented as:

$$\hat{\boldsymbol{\beta}}_{RR} \underset{\boldsymbol{\beta}}{=} \arg\min \left\{ (\mathbf{y} - \mathbf{X}\boldsymbol{\beta})' (\mathbf{y} - \mathbf{X}\boldsymbol{\beta}) + \lambda \boldsymbol{\beta}'\mathbf{D}\boldsymbol{\beta} \right\}$$

where: $(\mathbf{y} - \mathbf{X}\boldsymbol{\beta})' (\mathbf{y} - \mathbf{X}\boldsymbol{\beta}) = \sum_{i} \left(y_i - \sum_{j} x_{ij}\beta_j \right)^2$ is a RSS and $\boldsymbol{\beta}'\mathbf{D}\boldsymbol{\beta} = \sum_{j \in S} \beta_j^2$ is a sum of squares of the regression coefficients. Here, $\mathbf{D}$ is a diagonal matrix whose entries are 1 for $j \in S$ and zero otherwise. The first order conditions of the above optimization problem are satisfied by the following system of linear equations:

$$[\, \mathbf{X}'\mathbf{X} + \lambda \mathbf{D} \,] \hat{\boldsymbol{\beta}}_{RR} = \mathbf{X}'\mathbf{y} \qquad [3]$$

Relative to OLS, RR adds a constant (λ) to the diagonal entry corresponding to regression coefficients that are included in S (i.e., those whose effects are penalized). When either **D** or λ equals zero, the solution to the above problem is OLS. Adding a constant to the diagonal of the matrix of coefficients shrink estimates towards zero. This induces bias but reduces the variance of the estimates. And in large-p small-n regressions this may smaller MSE than those of OLS estimates and better predictions.

A simplified example. Let us consider a simple example where each subject was assigned to one of two possible treatments (treatments 1 and 2). The treatment-means parameterization of this model is: $y_i = x_{1i}\beta_1 + x_{2i}\beta_2 + \varepsilon_i$ where y_i is the response, x_{1i} is a dummy variable indicator of treatment 1, $x_{2i} = (1 - x_{i1})$ is a dummy variable indictor of treatment 2, β_1 and β_2 the means of treatments 1 and 2, respectively, and ε_i is a model residual. The OLS estimates of regression coefficients in this model are:

$$\begin{bmatrix} \sum_i x_{1i}^2 & \sum_i x_{1i}x_{2i} \\ \sum_i x_{1i}x_{2i} & \sum_i x_{2i}^2 \end{bmatrix} \begin{pmatrix} \hat{\beta}_1 \\ \hat{\beta}_2 \end{pmatrix} = \begin{pmatrix} \sum_i x_{1i}y_i \\ \sum_i x_{2i}y_i \end{pmatrix}$$

Moreover, $\sum_i x_{1i}^2$ and $\sum_i x_{2i}^2$ equal the number of individuals in treatment 1 and 2 (denoted as n_1 and n_2 respectively), since x_{1i} and x_{1i} are orthogonal $\sum_i x_{1i}x_{2i} = 0$, and, finally, $\sum_i x_{1i}y_i$ and $\sum_i x_{2i}y_i$ are the sum of the response variable for subjects assigned to treatments 1 and 2, respectively. Therefore,

$$\begin{bmatrix} n_1 & 0 \\ 0 & n_2 \end{bmatrix} \begin{pmatrix} \hat{\beta}_1 \\ \hat{\beta}_2 \end{pmatrix} = \begin{pmatrix} \sum_{i:x_{1i}=1} y_i \\ \sum_{i:x_{2i}=1} y_i \end{pmatrix}$$

, from where we conclude that the OLS estimate of the treatment mean are simply the average of the phenotypes observed in each treatment, that is $\hat{\beta}_1 = \frac{\sum_{i:x_{1i}=1} y_i}{n_1}$ and $\hat{\beta}_2 = \frac{\sum_{i:x_{2i}=1} y_i}{n_2}$. Now, considering the RR estimates, according to [3] these will be will be

$$\begin{bmatrix} n_1 + \lambda & 0 \\ 0 & n_2 + \lambda \end{bmatrix} \begin{pmatrix} \hat{\beta}_1 \\ \hat{\beta}_2 \end{pmatrix} = \begin{pmatrix} \sum_{i:x_{1i}=1} y_i \\ \sum_{i:x_{2i}=1} y_i \end{pmatrix}$$

; therefore the RR estimates are $\hat{\beta}_1 = \frac{\sum_{i:x_{1i}=1} y_i}{n_1 + \lambda}$ and $\hat{\beta}_2 = \frac{\sum_{i:x_{2i}=1} y_i}{n_2 + \lambda}$. Therefore, adding λ to the diagonal entries of the matrix of coefficients will shrink estimates towards zero. By how much? This will depend on the relationship between λ and sample size. From here we can also see that with fix λ, the amount

of shrinkage will decrease as sample size increases. Asymptotically, if we fix λ and let the number of individuals in each treatment approach infinity, RR estimates converge to OLS estimates.

Other penalized estimators. Several alternative penalized estimation procedures have been proposed, and they differ on the choice of penalty function, $J(\boldsymbol{\beta})$. As we discussed above, in RR, the penalty is proportional to the sum of squares of the regression coefficients or L2 norm, $J(\boldsymbol{\beta}) = \sum_{j=1}^{p} \beta_j^2$. A more general formulation, known as **Bridge regression** (Frank and Friedman 1993), uses $J(\boldsymbol{\beta}) = \sum_{j=1}^{p} \left\| \beta_j \right\|^{\gamma}$ with $\gamma > 0$. RR is a particular case with $\gamma = 2$ yielding RR. **Subset selection** occurs as a limiting case with $\gamma \to 0$, this penalizes the number of non-zero effects regardless of their magnitude, $J(\boldsymbol{\beta}) = \sum_{j=1}^{p} 1(\beta_j \neq 0)$. Another special case, known as **LASSO** (Least Absolute Angle and Selection Operator, (Tibshirani 1996) occurs with $\gamma = 1$, yielding the L1 penalty: $J(\boldsymbol{\beta}) = \sum_{j=1}^{p} \left\| \beta_j \right\|$. Using this penalty induces a solution that may involve zeroing-out some regression coefficients and shrinkage estimates of the remaining effects; therefore combining in features of subset selection with shrinkage estimation. LASSO has become very popular in several fields of applications. However LASSO and subset selection approaches have two important limitations. First, by construction, in these methods the solution admits at most n non-zero estimates of regression coefficients. In GS and with complex traits, there is no reason to restrict the number of markers with non-zero effect to be limited by n (the number of observations). Second, when predictors are correlated, something which occurs in GS, methods performing variable selection such as the LASSO are usually outperformed by RR (Hastie, Tibshirani, and Friedman 2009). Therefore, in an attempt to combine the good features of RR and of Lasso in a single estimation framework (Zou and Hastie 2005) proposed to use as penalty a weighted average of the L1 and L2 norm, that is, for $0 \leq \alpha \leq 1, J(\beta) = \alpha \sum_{j=1}^{p} \left\| \beta_j \right\| + (1-\alpha) \sum_{j=1}^{p} \beta_j^2$ and termed the method the **Elastic Net** (EN), this model involves then two tuning parameters which need to be specified, the regularization parameter (λ) and α .

2.2. Computing RR estimates

In the following example we present two ways of computing ridge regression estimates. The first one implements [3] using matrix operations; the second one uses an iterative procedure. Run this last algorithm with 10 and 500 iterations.

Example 1. Alternative ways of deriving Ridge-Regression Estimates

```
rm(list=ls())
## Using Cholesky factor ######################################
  library(BLR)
  data(wheat)
  X2<-cbind(1,X)
  y<-Y[,2]
  C<-crossprod(X2)
  rhs<-crossprod(X2,y)
  MSx<-0 ; for(i in 1:ncol(X)){ MSx<-MSx+var(X[,i])}
  h2<-0.5
  lambda<-MSx*(1-h2)/h2
  for(i in 2:ncol(C)){ C[i,i]<-C[i,i]+lambda }
  CInv<-chol2inv(chol(C))
  bHatRR_1<-crossprod(CInv,rhs)

## Using an iterative procedure ##################################
diagC<-numeric()
for(i in 1:ncol(X2)){diagC[i]<-sum(X2[,i]^2)+ifelse(i==1,0,lambda) }
bHatRR_2<-rep(0,ncol(X2))
bHatRR_2[1]<-mean(y)
e<-y-mean(y)
nIter<-10
for(i in 1:nIter){
      for(j in 1:ncol(X2)){
         tmpY<-e+X2[,j]*bHatRR_2[j]
          rhs<-sum(X2[,j]*tmpY)
          bHatRR_2[j]<-rhs/diagC[j]
          e<-tmpY-X2[,j]*bHatRR_2[j]
      }
      print(i)
}
tmp<-range(c(bHatRR_1[-1],bHatRR_2[-1]))
plot(bHatRR_1[-1],bHatRR_2[-1],ylim=tmp,xlim=tmp,col=2,main="")
## Change nIter, set it equal to 500 and then equal to 1000
```

2.3. Effect of regularization on estimates, goodness of fit and model DF

In penalized estimation, the regularization parameter (λ) controls the trade-offs between model goodness of fit and model complexity. This affects parameter estimates (their value, and the statistical properties of the estimator) model goodness of fit to the training dataset and the ability of the model to predict un-observed phenotypes.

Model complexity. The complexity of a linear model can be measured by the degree of freedom of the model. In RR, predictions are computed as $\hat{\mathbf{y}} = \mathbf{X}\hat{\boldsymbol{\beta}}_{RR} = \mathbf{X}\left[\mathbf{X}'\mathbf{X} + \lambda\mathbf{D}\right]^{-1}\mathbf{X}'\mathbf{y} = \mathbf{H}_{RR}\mathbf{y}$ where $\mathbf{H}_{RR} = \mathbf{X}\left[\mathbf{X}'\mathbf{X} + \lambda\mathbf{D}\right]^{-1}\mathbf{X}'$ is the Hat matrix. If we set $\lambda = 0$ we obtain the Hat matrix of OLS: $\mathbf{H}_{OLS} = \mathbf{X}\left[\mathbf{X}'\mathbf{X}\right]^{-1}\mathbf{X}'$. In linear models degree of freedom are equal to the sum of the diagonal entries of **H**. In OLS this just equals the number of predictors (provided that **X** is full rank). In RR λ also affects DF. The following R-code fits RR over a grid of values of λ and evaluates the impact that λ has on goodness of fit to the training data, prediction accuracy, and model degree of freedom.

Example 2. Effects of regularization on goodness of fit and model DF

```
rm(list=ls())
 ##### DATA ################################################
 library(BLR)
 data(wheat)
 objects()
 N<-nrow(X) ; p<-ncol(X)
 y<-Y[,2]
 set.seed(12345)
 tst<-sample(1:N,size=150,replace=FALSE)
 XTRN<-X[-tst,]
 yTRN<-y[-tst]
 XTST<-X[tst,]
 yTST<-y[tst]

 ## FITTING MODEL OVER A GRID OF VALUES OF lambda
 lambda<-c(5,10,50,100,200,500,700,1000, 2000, 5000,20000)
 ZTRN<-cbind(1,XTRN) ; ZTST<-cbind(1,XTST)
 sqCorTRN<-numeric();  sqCorTST<-numeric(); DF<-numeric()
 BHat<-matrix(nrow=ncol(XTRN),ncol=length(lambda),NA)

 C0<-crossprod(ZTRN)
 rhs<-crossprod(ZTRN,yTRN)

 for(i in 1:length(lambda)){ #loop over values of lambda
   C<-C0
   # adds lambda to the diagonal of C (starts at 2)
   for(j in 2:ncol(C)){   C[j,j]<-C[j,j]+lambda[i]   }
   CInv<-chol2inv(chol(C))
   sol<-crossprod(CInv, rhs)
   BHat[,i]<-sol[-1]
   yHatTRN<-ZTRN%*%sol
   sqCorTRN[i]<-cor(yTRN,yHatTRN)^2
   yHatTST<-ZTST%*%sol
   sqCorTST[i]<- cor(yTST,yHatTST)^2
   H<-ZTRN%*%CInv%*%t(ZTRN)
   DF[i]<-sum(diag(H))
   print(i)
  }
  write(sqCorTST,file="sqCorTST.txt")
  write(lambda,file="lambda.txt")
# (Plots in next page)

```

Example 2. (from previous page)

```
43  ## PLOT 1: Model Degree of freedom
44    plot(DF~log(lambda),type="o",col=2,
45         xlab= expression(paste(log(lambda))),
46         ylab="DF",ylim=c(0,max(DF)));abline(h=1,lty=2)
47
48    ## PLOT 2: Estimates (shrinkage by marker)
49    marker<-1 # (choose a number between 1 and 1279)
50    plot(BHat[marker,],type="o",col=2,
51         xlab=expression(paste(log(lambda))),ylab="Estimate")
52    abline(h=0)
53    tmp<-range(BHat[,c(1,5)])
54    ## PLOT 3: Estimates (shrinkage all markers)
55    plot(BHat[,5]~BHat[,1],xlim=tmp,ylim=tmp,
56         xlab='Lambda=5',ylab='Lambda=200',col=2,cex=.5);
57    lines(x=c(-10,10),y=c(-10,10))
58
59    ## PLOT 4: Goodness of fit to TRN dataset
60    plot(sqCorTRN~log(lambda),type="o",col=2,main="Training data",
61            xlab=expression(paste(log(lambda))),ylab="Squared Corr.")
62
63    ## PLOT 5 Prediction Accuracy
64    plot(sqCorTST~log(lambda),type="o",col=2,main="Testing data",
65            xlab=expression(paste(log(lambda))),ylab="Squared Corr.")
66
```

2.4. The Hat Matrix of large-p with small-n genomic regressions as a local smoother

Above we introduce the hat matrix as applied to the training dataset, $\hat{\mathbf{y}}_{TRN} = \mathbf{X}_{TRN}\hat{\boldsymbol{\beta}}_{RR} = \mathbf{X}_{TRN}\left[\mathbf{X}'_{TRN}\mathbf{X}_{TRN} + \lambda\mathbf{D}\right]^{-1}\mathbf{X}'_{TRN}\mathbf{y}_{TRN} = \mathbf{H}_{TRN}\mathbf{y}_{TRN}$. Similarly, we can defined a hat matrix for the testing dataset, $\hat{\mathbf{y}}_{TST} = \mathbf{X}_{TST}\hat{\boldsymbol{\beta}}_{RR} = \mathbf{X}_{TST}\left[\mathbf{X}'_{TRN}\mathbf{X}_{TRN} + \lambda\mathbf{D}\right]^{-1}\mathbf{X}'_{TRN}\mathbf{y} = \mathbf{H}_{TST}\mathbf{y}_{TRN}$. In both cases, predictions are simply weighted sums of phenotypes of the training dataset,

$\hat{y}_{TRN,i} = \sum_{j\in TRN} h_{TRN,ij}y_j$ and $\hat{y}_{TST,i} = \sum_{j\in TRN} h_{TST,ij}y_j$, where $h_{.,ij}$ is the $(i,j)^{th}$ entry of either $\mathbf{H}_{TRN}$ or $\mathbf{H}_{TST}$.

The relative absolute value of each entry, $|h_{ij}|$, indicates, according to the model, how informative the *jth* phenotype of the training dataset is for estimating the conditional expectation at the *ith* point of either the training or testing dataset. The following code computes the hat matrix a training and testing dataset and plots the one of the rows of $\mathbf{H}_{TRN}$ and of $\mathbf{H}_{TST}$.

Example 3. The Hat Matrix of Ridge Regression

```
rm(list=ls())
##### DATA ################################################
  library(BLR)
  data(wheat)
  objects()
  N<-nrow(X) ; p<-ncol(X)
  y<-Y[,2]
  set.seed(1235)
  tst<-sample(1:N,size=150,replace=FALSE)
  XTRN<-X[-tst,]
  yTRN<-y[-tst]
  XTST<-X[tst,]
  yTST<-y[tst]

## FITTING THE MODEL
  lambda<-200
  ZTRN<-cbind(1,XTRN)
  ZTST<-cbind(1,XTST)

  C<-crossprod(ZTRN)
  for(j in 2:ncol(C)){   C[j,j]<-C[j,j]+lambda}
  CInv<-chol2inv(chol(C))
  TMP<-tcrossprod(CInv,ZTRN)

  HTRN<-ZTRN%*%TMP
  HTST<-ZTST%*%TMP
  yHatTRN<-HTRN%*%yTRN
  yHatTST<-HTST%*%yTRN

## Plot of row 100 of HTRN
   plot(abs(HTRN[100,]),xlab=' j (TRN)',
        ylab='h(100 , j)',col=2,main='Training dataset');abline(v=100)

## Plot of row 30 of HTST
  plot(abs(HTST[30,]),xlab=' j (TRN)',
       ylab='h(30 , j)',col=2,main='Testing dataset')
```

2.5. Bayesian View of Ridge Regression

Most penalized can be viewed as posterior modes in certain class of Bayesian models. For instance, RR estimates are equivalent to the posterior mode of the vector of regression coefficients in a Bayesian model with a Gaussian likelihood and a Gaussian prior for the vector of regression coefficients. To see this, recall that that estimates in RR are obtained as the solution to the following optimization problem:

$$\hat{\boldsymbol{\beta}}_{RR} = \underset{\arg\min}{} \left\{ (\mathbf{y} - \mathbf{X}\boldsymbol{\beta})' (\mathbf{y} - \mathbf{X}\boldsymbol{\beta}) + \lambda \boldsymbol{\beta}' \mathbf{D} \boldsymbol{\beta} \right\}$$

Multiplying the objective function by -1/2 and switching from minimization to maximization do not affect the solution; therefore,

$$\hat{\boldsymbol{\beta}}_{RR} = \underset{\arg\max}{} \left\{ -\frac{1}{2}(\mathbf{y} - \mathbf{X}\boldsymbol{\beta})'(\mathbf{y} - \mathbf{X}\boldsymbol{\beta}) - \lambda\frac{1}{2}\boldsymbol{\beta}'\mathbf{D}\boldsymbol{\beta} \right\}$$

Let $\lambda = \frac{\sigma_\varepsilon^2}{\sigma_\beta^2}$ where, σ_ε^2 and σ_β^2 are non-negative constants. Replacing above and dividing the objective function by σ_ε^2 maintains the solution unchanged, with this we get:

$$\hat{\boldsymbol{\beta}}_{RR} = \underset{\arg\max}{} \left\{ -\frac{1}{2\sigma_\varepsilon^2}(\mathbf{y} - \mathbf{X}\boldsymbol{\beta})'(\mathbf{y} - \mathbf{X}\boldsymbol{\beta}) - \frac{1}{2\sigma_\beta^2}\boldsymbol{\beta}'\mathbf{D}\boldsymbol{\beta} \right\}$$

Finally, applying the exponential function to the objective function maintains the solution unchanged, therefore:

$$\hat{\boldsymbol{\beta}}_{RR} = \underset{\arg\max}{} \left\{ \exp\left[-\frac{1}{2\sigma_\varepsilon^2}(\mathbf{y} - \mathbf{X}\boldsymbol{\beta})'(\mathbf{y} - \mathbf{X}\boldsymbol{\beta}) - \frac{1}{2\sigma_\beta^2}\boldsymbol{\beta}'\mathbf{D}\boldsymbol{\beta} \right] \right\}$$

$$= \underset{\arg\max}{} \left\{ \exp\left[-\frac{1}{2\sigma_\varepsilon^2}(\mathbf{y} - \mathbf{X}\boldsymbol{\beta})'(\mathbf{y} - \mathbf{X}\boldsymbol{\beta}) \right] \exp\left[-\frac{1}{2\sigma_\beta^2}\boldsymbol{\beta}'\mathbf{D}\boldsymbol{\beta} \right] \right\}$$

The first component of the objective function, $\exp\left[-\frac{1}{2\sigma_\varepsilon^2}(\mathbf{y} - \mathbf{X}\boldsymbol{\beta})'(\mathbf{y} - \mathbf{X}\boldsymbol{\beta}) \right]$, is proportional to a Gaussian likelihood, centered at $\mathbf{X}\boldsymbol{\beta}$ and with (co)variance matrix $\mathbf{I}\sigma_\varepsilon^2$. The second component, $\exp\left[-\frac{1}{2\sigma_\beta^2}\boldsymbol{\beta}'\mathbf{D}\boldsymbol{\beta} \right]$, is proportional a Gaussian prior for the regression coefficients, centered at zero and with (co)variance matrix $\mathbf{D}^{-1}\sigma_\beta^2$. Therefore, estimates obtained with RR are equivalent to the posterior mode of regression coefficients in the following Bayesian model.

$$\begin{cases} \text{Likelihood}: & [\mathbf{y} \mid \boldsymbol{\beta}, \sigma_\varepsilon^2] \sim N(\mathbf{X}\boldsymbol{\beta}, \mathbf{I}\sigma_\varepsilon^2) \\ \text{Prior}: & [\boldsymbol{\beta} \mid \sigma_\beta^2] \sim N(\mathbf{0}, \mathbf{D}^{-1}\sigma_\beta^2) \end{cases} \qquad [4]$$

The posterior distribution of $\boldsymbol{\beta}$ is multivariate normal with a mean (co-variance matrix) equal to the solution (inverse of the coefficient matrix) of the following system: $[\,\mathbf{X}'\mathbf{X} + \lambda\mathbf{D}\,]\hat{\boldsymbol{\beta}} = \mathbf{X}'\mathbf{y}$; this is just the RR equations. This is also the Best Linear Unbiased Predictor (BLUP) of $\boldsymbol{\beta}$ given **y**.

Recall that the ratio $\frac{\sigma_\varepsilon^2}{\sigma_\beta^2}$ is equivalent to λ in RR. I a fully-Bayesian models we assign priors to each of these variance parameters, this allow inferring these unknowns from the same training data that is used to estimate marker effects. The following example fits a Bayesian RR using the R-package BLR ('Bayesian Linear Regression'), after you run the model:

- The BLR package returns an list with posterior means and other information, type str(fm) and inspect what BLR returns
- Check the posterior mean of σ_ε^2 and σ_β^2 (fm$varE and fm$varBR, respectively), remember the ratio of these variances is interpretable as λ in RR.
- Examine trace plots
- Compare prediction accuracy of the fully-Bayesian method versus RR.

Example 4. Bayesian Ridge Regression Using BLR

```
rm(list=ls())
##### DATA (same as Example 2) ########################################
 library(BLR)
 data(wheat)
 objects()
 N<-nrow(X) ; p<-ncol(X)
 y<-Y[,2]
 set.seed(12345)
 tst<-sample(1:N,size=150,replace=FALSE)
 XTRN<-X[-tst,]
 yTRN<-y[-tst]
 XTST<-X[tst,]
 yTST<-y[tst]

## Fits the model
  prior<-list(varE=list(df=4,S=1),varBR=list(df=5,S=.01))
  fm<-BLR(y=yTRN,XR=XTRN,nIter=12000,burnIn=2000,prior=prior)

## Prediction Accuracy: Bayesian vs grid search
  x<-scan(file="lambda.txt")
  y<-scan(file="sqCorTST.txt")

  plot(y~log(x),type="o",col=2,
         xlab=expression(paste(log(lambda))),ylab="Squared Corr.",
         ylim=c(0.1,.3))

  abline(v= log(fm$varE/fm$varBR),col=4)
  abline(h=cor(yTST,XTST%*%fm$bR)^2,col=4)


## trace plots
  plot(scan("varE.dat"),type="o",col=2)
  abline(h=fm$varE,col=4)
  abline(v=200,col=4)
```

2.6. G-BLUP

Here we show the equivalence between estimates (posterior modes) derived from model [4] and the so-called G-BLUP ('Genomic Best Linear Unbiased Predictor', e.g., VanRaden, 2008). We show this using [4] and properties of the multivariate-normal density that are outlined below.

Properties of Multivariate Normal Density

Let $\boldsymbol{\theta} = (\boldsymbol{\theta}_1', \boldsymbol{\theta}_2')'$ be a multivariate normal random vector with expectation $E\begin{bmatrix}\boldsymbol{\theta}_1 \\ \boldsymbol{\theta}_2\end{bmatrix} = \begin{bmatrix}\boldsymbol{\mu}_1 \\ \boldsymbol{\mu}_2\end{bmatrix}$ and (co)variance matrix $Cov\begin{bmatrix}\boldsymbol{\theta}_1 \\ \boldsymbol{\theta}_2\end{bmatrix} = \begin{bmatrix}\boldsymbol{\Sigma}_{11} & \boldsymbol{\Sigma}_{12} \\ \boldsymbol{\Sigma}_{21} & \boldsymbol{\Sigma}_{22}\end{bmatrix}$.

(1) All ***marginal densities are also normal densities***, specifically:

$$\boldsymbol{\theta}_1 \sim MVN(\boldsymbol{\theta}_1, \boldsymbol{\Sigma}_{11}) \text{ and } \boldsymbol{\theta}_2 \sim MVN(\boldsymbol{\theta}_2, \boldsymbol{\Sigma}_{22}).$$

The ***conditional densities are also normal densities***, with mean and (co)variance matrices given by the following:

$$E[\boldsymbol{\theta}_1|\boldsymbol{\theta}_2] = \boldsymbol{\mu}_1 + \boldsymbol{\Sigma}_{12}\boldsymbol{\Sigma}_{22}^{-1}(\boldsymbol{\theta}_2 - \boldsymbol{\mu}_2) \text{ and } E[\boldsymbol{\theta}_2|\boldsymbol{\theta}_1] = \boldsymbol{\mu}_2 + \boldsymbol{\Sigma}_{21}\boldsymbol{\Sigma}_{11}^{-1}(\boldsymbol{\theta}_1 - \boldsymbol{\mu}_1). \quad [5]$$

$$Cov[\boldsymbol{\theta}_1|\boldsymbol{\theta}_2] = \boldsymbol{\Sigma}_{11} - \boldsymbol{\Sigma}_{12}\boldsymbol{\Sigma}_{22}^{-1}\boldsymbol{\Sigma}_{21} \text{ and } Cov[\boldsymbol{\theta}_2|\boldsymbol{\theta}_1] = \boldsymbol{\Sigma}_{22} - \boldsymbol{\Sigma}_{21}\boldsymbol{\Sigma}_{11}^{-1}\boldsymbol{\Sigma}_{12}. \quad [6]$$

Above, $\mathbf{B}_{21} = \boldsymbol{\Sigma}_{21}\boldsymbol{\Sigma}_{11}^{-1} = \{b_{ij}\}$ and $\mathbf{B}_{12} = \boldsymbol{\Sigma}_{12}\boldsymbol{\Sigma}_{22}^{-1} = \{b_{ij}\}$ are matrix of regression coefficients of the ith on the jth random variable of $\boldsymbol{\theta}$.

The multivariate normal density is closed under linear operations in the sense that ***linear combinations of MVN random variables*** of the form $\boldsymbol{\delta} = \boldsymbol{\alpha} + \mathbf{T}\boldsymbol{\theta}$ ***are multivariate normal*** random variables, with mean vector and (co)variance matrices given by the following:

$$E[\boldsymbol{\delta}] = \boldsymbol{\alpha} + \mathbf{T}E[\boldsymbol{\theta}] = \boldsymbol{\alpha} + \mathbf{T}\boldsymbol{\mu}, \quad [7]$$

and (co)variance matrix

$$Cov[\boldsymbol{\delta}] = \mathbf{T}Cov[\boldsymbol{\theta}]\mathbf{T}' = \mathbf{T}\boldsymbol{\Sigma}\mathbf{T}', \quad [8]$$

Best Linear Unbiased Predictor (BLUP)

We are now ready to derive the conditional expectation of marker effects and of genomic values. The **conditional expectation is the best predictor in the mean-squared error sense**. Also, as we show here, in the context of model [4] the conditional expectations of marker effects and of genomic values are linear functions of data and are un-biased. Therefore, the conditional expectations of genomic values and of marker effects from model [4] are BLUP ('Best Linear Unbiased Predictor').

For ease of notation we omit the intercept and therefore in [4] we set **D** equal to an identity matrix. The model is then described by:

$$\begin{cases} \text{Likelihood}: \left[\mathbf{y} \mid \boldsymbol{\beta}, \sigma_\varepsilon^2\right] \sim N\left(\mathbf{X}\boldsymbol{\beta}, \mathbf{I}\sigma_\varepsilon^2\right) \\ \text{Prior}: \qquad \left[\boldsymbol{\beta} \mid \sigma_\beta^2\right] \sim N\left(\mathbf{0}, \mathbf{I}\sigma_\beta^2\right) \end{cases} \qquad [4b]$$

From [4b] and using [7] and [8], we obtain that the joint density of $\mathbf{y}$ and $\boldsymbol{\beta}$:

$$\begin{bmatrix} \mathbf{y} \\ \boldsymbol{\beta} \end{bmatrix} \sim MVN\left[\mathbf{0}, \begin{bmatrix} \mathbf{X}\mathbf{X}'\sigma_\beta^2 + \mathbf{I}\sigma_\varepsilon^2 & \mathbf{X}\sigma_\beta^2 \\ \mathbf{X}'\sigma_\beta^2 & \mathbf{I}\sigma_\beta^2 \end{bmatrix}\right] \qquad [9]$$

Using [5] we get the BLUP of marker effects:

$$E\left[\boldsymbol{\beta} \mid \mathbf{y}, \sigma_\varepsilon^2\right] = \mathbf{X}'\sigma_\beta^2\left[\mathbf{X}\mathbf{X}'\sigma_\beta^2 + \mathbf{I}\sigma_\varepsilon^2\right]^{-1}\mathbf{y} = \mathbf{X}'\left[\mathbf{X}\mathbf{X}' + \lambda\mathbf{I}\right]^{-1}\mathbf{y} \qquad [10]$$

which is the posterior mean of $\boldsymbol{\beta}$. Here, $\lambda = \sigma_\varepsilon^2\sigma_\beta^{-2}$. Because of the equivalence between the posterior mode of $\boldsymbol{\beta}$ and the RR estimate, the solution given by [10] is also equivalent to the RR estimate given by [3]. Importantly, note that computing the solution using [3] requires inverting a p×p matrix. On the other hand, we can obtain the same solution using [10] with inversion of n×n matrix. Expression [10] is **linear** on data and it is **unbiased** with respect to the prior mean, $E(\boldsymbol{\beta}) = \mathbf{0}$. To see this we take expectations in [10] with respect to **y** to get $E\left\{E\left[\boldsymbol{\beta} \mid \mathbf{y}, \sigma_\varepsilon^2\right]\right\} = \mathbf{X}'\left[\mathbf{X}\mathbf{X}' + \lambda\mathbf{I}\right]^{-1}E\left[\mathbf{y}\right]$. From [9], $E\left[\mathbf{y}\right] = \mathbf{0}$; therefore: $E\left\{E\left[\boldsymbol{\beta} \mid \mathbf{y}, \sigma_\varepsilon^2\right]\right\} = \mathbf{0}$. Therefore, [10] gives the BLUP of marker effects.

We now derive the conditional expectation of genomic values given the data.

$$\begin{aligned} E\left[\mathbf{X}\boldsymbol{\beta} \mid \mathbf{y}, \sigma_\varepsilon^2\right] &= \mathbf{X}E\left[\boldsymbol{\beta} \mid \mathbf{y}, \sigma_\varepsilon^2\right] \\ &= \mathbf{X}\mathbf{X}'\left[\mathbf{X}\mathbf{X}' + \lambda\mathbf{I}\right]^{-1}\mathbf{y} \\ &= \left[\mathbf{I} + \lambda\mathbf{G}^{-1}\right]^{-1}\mathbf{y} \end{aligned} \qquad [11]$$

Where $\mathbf{G} = \mathbf{XX}'$. This is the so-called G-BLUP of genomic values. Expression [11] is the best predictor of genomic value and it is linearly on data. Also, taking expectation with respect to phenotypes $E\left\{\left[\mathbf{I}+\lambda\mathbf{G}^{-1}\right]^{-1}\mathbf{y}\right\}=\left[\mathbf{I}+\lambda\mathbf{G}^{-1}\right]^{-1}E\left\{\mathbf{y}\right\}=\mathbf{0}$; therefore [11] is the BLUP of genomic values.

The following example computes G-BLUP for the wheat datset, and illustrate the equivalence with predictions from the RR.

Example 5. Ridge Regression and G-BLUP

```
rm(list=ls())
### DATA ###################################################################
  library(BLR)
  data(wheat)
  for(i in 1:ncol(X)){X[,i]<-(X[,i]-mean(X[,i]))}
  y<-Y[,1]
  h2<-0.5
  lambda<-ncol(X)
### Computing RR estimates and prediction using eq. [3] #######
  C<-crossprod(X)
  diag(C)<-diag(C)+lambda
  CInv<-chol2inv(chol(C))
  rhs<-crossprod(X,y)
  sol<-crossprod(CInv,rhs)
  yHat_1<-X%*%sol

### GBLUP
  G<-tcrossprod(X)
  C<-chol2inv(chol(G))*lambda
  diag(C)<-diag(C)+1
  CInv<-chol2inv(chol(C))
  yHat_2<-crossprod(CInv,y)


### Comparison
  plot(yHat_2~yHat_1,col=2,xlab='Predicitons from RR equations',
                           ylab='Predicttions from GBLUP equations')
```

References

Frank, I.E., and J.H. Friedman. 1993. “A Statistical View of Some Chemometrics Regression Tools.” *Technometrics*: 109–135.

Hastie, Trevor, Robert Tibshirani, and Jerome Friedman. 2009. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Second Edition*. 2nd ed. 2009. Corr. 3rd printing 5th Printing. Springer.

Hoerl, A. E, and R. W Kennard. 1970. “Ridge Regression: Biased Estimation for Nonorthogonal Problems.” *Technometrics* 12 (1): 55–67.

Tibshirani, R. 1996. “Regression Shrinkage and Selection via the Lasso.” *Journal of the Royal Statistical Society. Series B (Methodological)* 58 (1): 267–288.

Zou, H., and T. Hastie. 2005. “Regularization and Variable Selection via the Elastic Net.” *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 67 (2): 301–320.

Statistical Methods for Genome-Enabled Prediction,

Lab 3:

The Bayesian Alphabet [1]

(gcampos@uab.edu)

Contents

3.1. The Bayesian Alphabet 2
3.2. Ridge Regression Vs Bayesian Ridge Regression 9
3.3. Bayesian Lasso: fixed versus random lambda 11
3.4. Regression using markers and pedigree 13
References 14

NOTE: In many examples in this lab we use Bayesian methods. In those examples we make inferences based on a relatively small number of samples and this is done due to time constraints. In practice, accurate inferences require much more samples.

[1] Suggestions made by Daniel Gianola are gratefully acknowledged.

3.1. The Bayesian Alphabet

In standard parametric models for genomic selection (GS) phenotypes, y_i, are regressed on marker covariates, $\{x_i\}$, using a linear model of the form $y_i = \mu + \sum_{j=1}^{p} x_{ij}\beta_j + \varepsilon_i$, where μ is an effect common to all subjects (i.e., an 'intercept'), $\{x_{ij}\}$ are marker genotypes (usually coded as 0,1,2) , $\{\beta_j\}$ are marker effects and ε_i is a model residuals. A standard practice for continuous traits is to assume that model residuals are IID normal, this yields the following likelihood function:

Likelihood: $p(\mathbf{y}|\mu,\boldsymbol{\beta},\sigma^2) = \prod_{i=1}^{n} N\left(y_i|\mu + \sum_{j=1}^{p} x_{ij}\beta_j, \sigma^2\right),$ [1]

where, $N\left(y_i|\mu + \sum_{j=1}^{p} x_{ij}\beta_j, \sigma^2\right)$ is a normal density for the random variable y_i centered at $\mu + \sum_{j=1}^{p} x_{ij}\beta_j$ and with variance σ^2.

With dense panels, the number of markers (p) vastly exceeds the number of data points (n) and because of this penalized or Bayesian shrinkage estimation methods are commonly used. In a Bayesian setting, shrinkage of estimates of effects is controlled by the choice of prior density assigned to marker effects. The joint prior density of the unknowns is commonly structured as follows:

Prior:

$$p(\mu,\boldsymbol{\beta},\sigma^2|df,S,\omega) \propto \left\{\prod_{j=1}^{p} p(\beta_j|\theta_{\beta_j},\sigma^2)p(\theta_{\beta_j}|\omega)\right\}\chi^{-2}(\sigma^2|df,S) \quad [2]$$

Above, a flat prior was assigned to the intercept, $\chi^{-2}\left(\sigma^2\middle|df,S\right)$ is a scaled-inverse Chi-squared density assigned to the residual variance and with *df* degree of freedom and scale equal to S, $p\left(\beta_j\middle|\theta_\beta,\sigma^2\right)$ denotes the prior density of the jth marker effect, θ_{β_j} is a vector of parameters indexing the prior density assigned to marker effects, $p\left(\theta_{\beta_j}\middle|\omega\right)$ is the prior density assigned to θ_{β_j} and ω are parameters indexing this density. The marginal prior density of marker effects is obtaining by integrating θ_{β_j} out, $p\left(\beta_j\middle|\sigma^2,\omega\right)=\int p\left(\beta_j\middle|\theta_{\beta_j},\sigma^2\right)p\left(\theta_{\beta_j}\middle|\omega\right)\partial\theta_{\beta_j}$. Note that, a-priori, all marker effects are assigned the same marginal prior density; therefore, contrary what it is sometimes said, in all members of the Bayesian alphabet, the prior variances of marker effects are the same for all markers.

Using Bayes rule, the posterior density of model unknowns given the data is proportional to the product of the likelihood, given in eq. [1], and the prior density, eq. [2], that is:

Posterior density:

$$p\left(\mu,\boldsymbol{\beta},\sigma^2\middle|\mathbf{y},df,S,\omega\right)\propto\prod_{i=1}^{n}N\left(y_i\middle|\mu+\sum_{j=1}^{p}x_{ij}\beta_j,\sigma^2\right)\times\left\{\prod_{j=1}^{p}p\left(\beta_j\middle|\theta_{\beta_j},\sigma^2\right)p\left(\theta_{\beta_j}\middle|\omega\right)\right\}\chi^{-2}\left(\sigma^2\middle|df,S\right), \quad [3]$$

The Bayesian Alphabet. Following the seminal contribution of Meuwissen, Hayes, and Goddard (2001) several linear Bayesian regression methods have been proposed and used for simulation and real data analysis. They differed in the choice of prior density

assigned to marker effects. In a **Bayesian Ridge** regression (BRR), the conditional prior assigned of marker effects are IID normal, $p\left(\beta_j \middle| \theta_{\beta_j}, \sigma^2\right) = N\left(\beta_j \middle| 0, \sigma^2_\beta\right)$ and $p\left(\theta_{\beta_j} \middle| \omega\right) = \chi^{-2}\left(\sigma^2_\beta \middle| df_\beta, S_\beta\right).$

A second group of models, which includes **Bayes A** (Meuwissen, Hayes, and Goddard 2001) and the **Bayesian LASSO** (BL, Park and Casella 2008) use thick tail prior densities (t in Bayes A and Double Exponential in the BL). These priors induce a different type of shrinkage than that induced by the BRR.

A third group of models, which include Bayes B (Meuwissen, Hayes, and Goddard 2001) and the spike-slab models (Ishwaran and Rao 2005) use priors that are mixtures of a peak (or a spike) of mass at (in the vicinity of) zero and of a continuous density (e.g., t, or normal). Figure 1 shows the densities of a Gaussian and Double Exponential densities and that of a mixture model with a peak of mass at zero and a Gaussian slab. The three densities have mean equal to zero and variance equal to one.

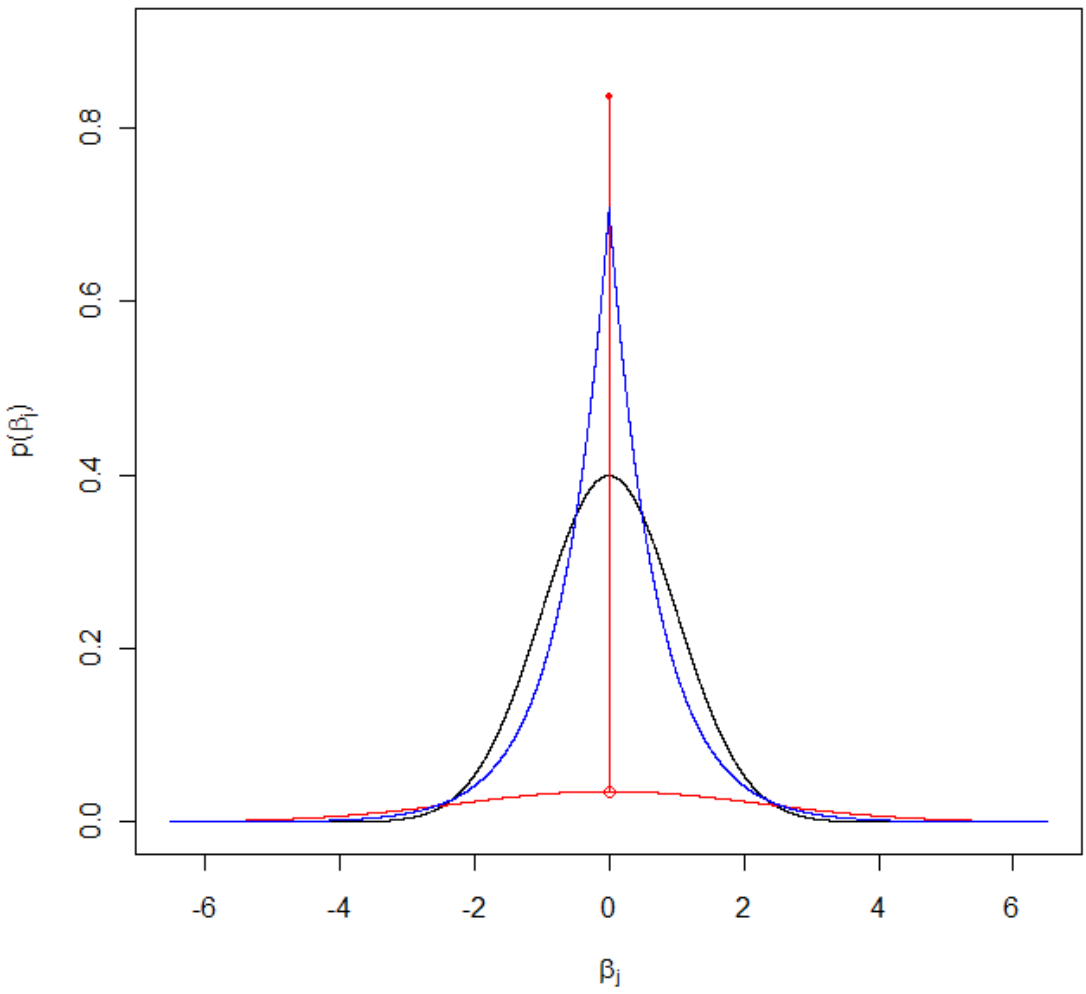

Figure 1. Density of a standard normal random variable (black), of a double-exponential random variable (blue) and of a random variable following a mixture density with a mass point at zero (with probability 0.8) and a Gaussian process with probability 0.2. All variables with zero mean and variance equal to one.

Many of the thick tail distributions, such as the t or the double-exponential densities can be represented as infinite mixtures of scaled normal densities. For instance, the t-prior density assigned to marker effects in **Bayes A** (Meuwissen, Hayes, and

Goddard 2001) can be represented as $t\left(\beta_j \middle| df_\beta, S_\beta\right) = \int N\left(\beta_j \middle| 0, \sigma^2_{\beta_j}\right) \chi^{-2}\left(\sigma^2_{\beta_j} \middle| df_\beta, S_\beta\right) \partial\sigma^2_{\beta_j}$ where df_β and S_β are prior degree of freedom and scale parameters and $\chi^{-2}\left(\sigma^2_{\beta_j} \middle| df_\beta, S_\beta\right)$ is a scaled-inverse Chi-squared density.

In the **Bayesian LASSO** (Park and Casella 2008) the Double-exponential prior density is represented as: $DE\left(\beta_j \middle| \lambda^2, \sigma^2_\varepsilon\right) = \int N\left(\beta_j \middle| 0, \sigma^2_\varepsilon \tau^2_j\right) Exp\left(\tau^2_j \middle| \frac{\lambda^2}{2}\right) \partial\sigma^2_{\beta_j}$. In the fully-Bayesian LASSO, λ^2 is treated as unknown and is assigned a Gamma prior. This prior is indexed by two parameters (rate and shape, see `help(rgamma)`) which are assumed to be known. Alternative priors for the regularization parameter are discussed in de los Campos et al. (2009).

In **Bayes B** (Meuwissen, Hayes, and Goddard 2001) marker effects are assumed to be equal to zero with probability π and with probability (1-π) the effect is assumed to be a draw form a t-distribution such as the one described in Bayes A. Model **Bayes C** (Habier et al. 2011) is similar to Bayes B but uses a Gaussian slab instead of the t-density used in Bayes B.

For infinitesimal traits, zeroing-out marker effects, such as in Bayes B or C, may harm predictive ability. Therefore, an alternative is to replace the peak of mass at zero used in Bayes B or C with a continuous density with small variance. This strategy is commonly used in what it is referred as to **Spike-Slab models** (Ishwaran and Rao 2005); for instance one can mix two Gaussian densities, one with very small variance and one with larger variance.

Choosing hyper-parameters. In the above mentioned models, the parameters indexing the prior density of marker effects play a central role in controlling the extent of

shrinkage of estimates of markers effect (similar to that of λ of the ridge regression. These parameters can be chosen in several ways, one of which is to select their values based on heritability-based rules.

Choosing Hyper parameters using heritability based rules. In linear models for genomic selection, genetic values are represented as regressions on marker covariates, that is $g_i = \sum_j x_{ij}\beta_j$. In these models, marker genotypes are fixed and marker effects are random variables drawn from an IID process; therefore:

$$Var(g_i) = \sum_j x_{ij}^2 Var(\beta_j) = \sigma_\beta^2 \sum_j x_{ij}^2$$

where σ_β^2 is the prior variance of marker effects. Summing over individuals and dividing by *n* yields

$$\frac{h^2}{1-h^2} = \frac{\sigma_\beta^2}{\sigma_\varepsilon^2} n^{-1} \sum_i \sum_j x_{ij}^2 = \frac{\sigma_\beta^2}{\sigma_\varepsilon^2} K \qquad [4]$$

where $K = n^{-1} \sum_i \sum_j x_{ij}^2$ is the average sum of square of marker genotypes in the dataset, and h^2 is the heritability of the trait. Commonly, the model uses an intercept and we measure variance at the genomic values as deviations from the center of the sample. Therefor, a common practice is to compute K after centering genotypes, that is: $K = n^{-1} \sum_i \sum_j \left(x_{ij} - 2\theta_j\right)$ where θ_j is the frequency of the allele coded as one at the jth marker. Moreover, if markers are centered and standardized to a unit variance, that is if $\tilde{x}_{ij} = \frac{x_{ij} - 2\theta_j}{\sqrt{2\theta_j\left(1-\theta_j\right)}}$ are used as marker codes in the regression, then K equals the number of markers (p).

We can now use [4] to solve for the values of the parameters controlling regularization as a function of K, h^2 and of the phenotypic variance (σ_p^2).

Ridge Regression. Recall from the Bayesian standpoint the regularization parameter of a ridge regression λ equals the ratio of the residual variance to the prior variance of marker effects, $\sigma_\varepsilon^2\sigma_\beta^{-2}$. Replacing this in [4] and solving for λ we get

$$\frac{h^2}{1-h^2}=\frac{K}{\lambda}\Rightarrow\lambda=\frac{1-h^2}{h^2}K \qquad [5]$$

Therefore, according to [5] the larger the noise-signal ratio, the strongest shrinkage of estimates should be. Also, K increases as the number of marker does; therefore, according to [5] λ should be increased as the number of markers does.

Bayesian Ridge Regression. In the Bayesian Ridge regression, instead of choosing λ we need to assign a prior to σ_β^2 and to σ_ε^2 . If these priors are scaled-inverse chi square, the prior expectations are: $E\left(\sigma_\cdot^2\middle|df_\cdot,S_\cdot\right)=\frac{S_\cdot}{df_\cdot-2}$ where (.) equals β or ε. Typically we choose $df_\cdot$ to be a small value, usually greater than 4 to guarantee finite prior variance. Then, we can solve for $S_\cdot$ as a function of $df_\cdot$, K, σ_p^2 and h^2, so that the prior expectation of each of the variance components matches the value we expect according to σ_p^2 , h^2 and [4],

specifically, equating $\sigma_p^2\left(1-h^2\right)$ to $E\left(\sigma_\varepsilon^2\middle|df,S\right)$ we get, $\sigma_p^2\left(1-h^2\right)=E\left(\sigma_\varepsilon^2\middle|df,S\right)=\frac{S_\varepsilon}{df_\varepsilon-2}$ and equating $\sigma_p^2h^2$ to $K\times E\left(\sigma_\beta^2\middle|df_\beta,S_\beta\right)$ we get

$$S_\varepsilon=\left(1-h^2\right)\sigma_p^2\left(df_\varepsilon-2\right)$$
$$S_\beta=\frac{h^2\sigma_p^2}{K}\left(df_\beta-2\right) \qquad [6]$$

Bayes A. The above formulas can also be used to define the scale parameters in Bayes B.

Bayesian Lasso. In this model, as originally formulated by (Park and Casella 2008), marker effects are assigned IID double-exponential priors with rate parameter, $\frac{\lambda^2}{\sigma_\varepsilon^2}$ (note, λ here is a different parameter than that of the ridge regression). The prior variance of marker effects is: $Var\left(\beta_j \middle| \lambda^2, \sigma_\varepsilon^2\right) = \sigma_\beta^2 = 2\frac{\sigma_\varepsilon^2}{\lambda^2}$; therefore, $\frac{\sigma_\beta^2}{\sigma_\varepsilon^2} = \frac{2}{\lambda^2}$. Using this in [4] we get: $\frac{h^2}{1-h^2} = \frac{2}{\lambda^2}K$ or

$$\lambda = \sqrt{2\frac{1-h^2}{h^2}K} \qquad [7]$$

For the scale parameter of the residual variance we can use formula [6].

Note. The regularization parameter of the Bayesian Lasso is a function of the noise-signal ratio, and also of the number of markers. Specifically we expect K at a rate proportional to the square-root of the number of markers. The same occurs in RR (see [5]).

Bayes B and C. Here, the prior variance of marker effects are $\sigma_\beta^2 = \frac{\sigma_{\beta_1}^2}{1-\pi}$ where π is the proportion of marker effects coming from the zero-state of the mixture and $\sigma_{\beta_1}^2$ is the variance of the 'slab' (a Gaussian density in Bayes C and a t in Bayes B); therefore we can use the following formulas to chose the scale parameters as functions of *df*, K, σ_p^2 h^2 and π,

$$S_\varepsilon = \frac{\left(1-h^2\right)\sigma_p^2}{df_\varepsilon - 2},\ S_\beta = \frac{h^2\sigma_p^2}{K\left(df_\beta - 2\right)}\frac{1}{(1-\pi)} \qquad [8]$$

3.2. Ridge Regression Vs Bayesian Ridge Regression

In this section we compare estimates of marker effects derived from a ridge regression using lambda from eq. [5] with those obtained with a Bayesian Ridge Regression using

hyper-parameters chosen according to [6]. For the BRR we use the BLR package. Here, the prior is provided as a list. There is one component in the list for each of the variance parameters. In each component you need to provide prior degree of freedom and scale. For more details refer to `help(BLR)` or see (Pérez et al. 2010).

Example 1. Ridge regression Vs Bayesian Ridge Regression

```
rm(list=ls())
library(BLR)
data(wheat)
y<-Y[,2]
h2<-.2
df0<-5
for(i in 1:ncol(X)){ X[,i]<-(X[,i]-mean(X[,i]))/sd(X[,i]) }

K<-ncol(X) # after standardization, K=# of markers
lambda<-K*(1-h2)/h2
Se<-(1-h2)*var(y)*(df0-2)
Sb<-h2*var(y)*(df0-2)/K
round(Se/Sb,5)==lambda

## Ridge Regression
X2<-cbind(1,X)
C<-crossprod(X2)
for(i in 2:ncol(C)){ C[i,i]<- C[i,i]+lambda }
CInv<-chol2inv(chol(C))
rhs<-crossprod(X2,y)
bHat_RR<-crossprod(CInv,rhs)
yHat_RR<-X2%*%bHat_RR

## Bayesian Ridge Regression
library(BLR)
prior<-list(varE=list(df=df0,S=Se) , varBR=list(df=df0,S=Sb))
fmBRR<-BLR(y=y,XR=X,prior=prior,
           nIter=13000,burnIn=3000, saveAt='BRR_')

fmBRR$varE/fmBRR$varBR
lambda

tmp<-range(c(bHat_RR[-1],fmBRR$bR))
plot(fmBRR$bR ~bHat_RR[-1],xlim=tmp,
     ylim=tmp, ,main='Estimates of Marker Effects',
     xlab='Ridge Regression', ylab='Bayesian Ridge Regression')
lines(x=c(-1,1),y=c(-1,1),col=2)

tmp<-range(c(yHat_RR,fmBRR$yHat))
plot(fmBRR$yHat~yHat_RR,xlim=tmp,ylim=tmp,main='Predictions',
     xlab='Ridge Regression', ylab='Bayesian Ridge Regression')
lines(x=c(-10,10),y=c(-10,10),col=2,lwd=2)
## Change the prior scale (e.g., double it) and evaluate the
## in inferences
```

3.3. Bayesian Lasso: fixed versus random lambda

In this example we fit the Bayesian LASSO using BLR. The prior for parameter lambda of the BL has four arguments: `type, value, rate` and `shape`. If `type='fixed'` lambda is set equal to `value` and kept fixed. If `type='random'` lambda is treated as unknown; in this case a gamma prior is assigned to λ^2 as described in Park and Casella (2008). For more details type `help(BLR)` in R or see Pérez et al. (2010). We chose values of the rate and shape parameters of the gamma prior so that the prior is flat in the neighborhood of the value of lambda we derive from eq. [4]. The following code displays the prior, run it and evaluates sensitivity with respect to rate and shape.

Example 2. Displaying prior of lambda of the BL

```
  h2<-0.5
  lambda0<-sqrt(2*K*(1-h2)/h2)
  lambda<-seq(from=0,to=250,by=1)
  dLambda<-2*lambda*dgamma(x=lambda^2,rate=1e-5,shape=0.53)
  plot(dLambda~lambda, type='l')
  abline(v=lambda0,col=2)

# change rate and shape and evaluate sensitivity of the prior
```

Now we fit the BL with fix and random lambda.

Example 3. Bayesian Lasso with fixed and random

```
  rm(list=ls())
  library(BLR)
  data(wheat)
  y<-Y[,2] ;  h2<-.5
  df0<-5
  for(i in 1:ncol(X)){ X[,i]<-(X[,i]-mean(X[,i]))/sd(X[,i]) }

  Se<-(1-h2)*var(y)*(df0-2)
  lambda0<-sqrt(2*(1-h2)/h2*ncol(X))

## Bayesian Lasso fixed lambda ###################################
  prior<-list(varE=list(df=df0,S=Se) ,
              lambda=list(value=lambda0,
                     type='fixed',rate=1e-5,shape=.53))

  fmBL_fixed<-BLR(y=y,XL=X,prior=prior,
                  nIter=12000,burnIn=2000,saveAt='BL_fixed_')

  fmBL_fixed$lambda
  lambda0

  tmp<-range(c(bHat_RR[-1],fmBL_fixed$bL))
  plot(fmBL_fixed$bL ~bHat_RR[-1],xlim=tmp,ylim=tmp)
  lines(x=c(-1,1),y=c(-1,1),col=2)

  tmp<-range(c(yHat_RR,fmBL_fixed$yHat))
  plot(fmBL_fixed$yHat~yHat_RR,xlim=tmp,ylim=tmp)
  lines(x=c(-10,10),y=c(-10,10),col=2,lwd=2)

  ## Now: change the value of lambda (e.g., 30 and 200) and
   ##      evaluate the impact on shrinkage of estimates

## Bayesian Lasso random lambda ##################################
  prior$lambda$type='random'

  fmBL_rand<-BLR(y=y,XL=X,prior=prior,
                   nIter=12000,burnIn=2000,saveAt='BL_rand_')

  fmBL_rand$lambda
  lambda0

  tmp<-range(fmBL_rand$bL,fmBL_fixed$bL)
  plot(fmBL_rand$bL ~fmBL_fixed$bL,xlim=tmp,ylim=tmp)
  lines(x=c(-1,1),y=c(-1,1),col=2)

  tmp<-range(c(fmBL_rand$yHat,fmBL_fixed$yHat))
  plot(fmBL_rand$yHat~fmBL_fixed$yHat,xlim=tmp,ylim=tmp)
  lines(x=c(-10,10),y=c(-10,10),col=2,lwd=2)
```

3.4. Regression using markers and pedigree

So far we have regressed phenotypes on markers only. The following code gives an example of models with and without pedigree. In the wheat dataset, matrix A is an additive relationship matrix computed from the pedigree.

Example 4. Bayesian Lasso with & without pedigree

```
##### DATA ############################################
 rm(list())
 library(BLR)
 data(wheat)
 objects()
 y<-Y[,2]
 set.seed(1235)
 tst<-sample(1:599,size=150,replace=FALSE)
 yNA<-y
 yNA[tst]<-NA

 ## Markers model
 prior<-list(varE=list(df=df0,S=Se) ,
            lambda=list(value=lambda0,type='random',
                        rate=1e-5,shape=.53))

 ## Model with only markers
  fmM<-BLR(y=yNA,XL=X,prior=prior,
                nIter=12000,burnIn=2000,saveAt='BL_M_')

  prior$varU=list(df=df0,S=Se/3)
  fmPM<-BLR(y=yNA,XL=X,prior=prior,GF=list(A=A,ID=1:599),
                nIter=12000,burnIn=2000,saveAt='BL_PM_')

  fmPM$varE/fmM$varE
  fmPM$lambda/fmM$lambda

  cor(y[tst],fmM$yHat[tst])
  cor(y[tst],fmPM$yHat[tst])

  tmp<-range(c(fmM$bL,fmPM$bL))
  plot(fmM$bL ~fmPM$bL,xlim=tmp,ylim=tmp)
  lines(x=c(-1,1),y=c(-1,1),col=2)

  tmp<-range(c(fmPM$yHat,fmM$yHat))
  plot(fmPM$yHat~fmM$yHat,xlim=tmp,ylim=tmp)
  lines(x=c(-10,10),y=c(-10,10),col=2,lwd=2)
```

References

Habier, D., R. Fernando, K. Kizilkaya, and D. Garrick. 2011. “Extension of the Bayesian Alphabet for Genomic Selection.” *BMC Bioinformatics* 12 (1): 186.

Ishwaran, H., and J. S Rao. 2005. “Spike and Slab Variable Selection: Frequentist and Bayesian Strategies.” *The Annals of Statistics* 33 (2): 730–773.

Meuwissen, T H, B J Hayes, and M E Goddard. 2001. “Prediction of Total Genetic Value Using Genome-wide Dense Marker Maps.” *Genetics* 157 (4) (April): 1819-1829.

Park, T., and G. Casella. 2008. “The Bayesian Lasso.” *Journal of the American Statistical Association* 103 (482): 681–686.

Pérez, Paulino, Gustavo de los Campos, José Crossa, and Daniel Gianola. 2010. “Genomic-Enabled Prediction Based on Molecular Markers and Pedigree Using the Bayesian Linear Regression Package in R.” *The Plant Genome Journal* 3 (2): 106-116. doi:10.3835/plantgenome2010.04.0005.

Statistical Methods for Genome-Enabled Prediction,

Lab 4:

Semi-parametric Genomic Regression Using Reproducing Kernel Hilbert Spaces Methods[1]

(gcampos@uab.edu)

Contents

4.1. Semi-parametric genome-enabled regression 2
4.2. Reproducing Kernel Hilbert Spaces (RKHS) regressions 3
4.3. Scatter plot smoothing with a Gaussian kernel 5
4.4. Inspecting the Hat Matrix 7
4.5. Bayesian view of RKHS 8
4.6. Genomic-Enabled Prediction Using RKHS 9
4.7. Kernel Averaging 12
4.8. Pedigree + Marker Models 15
References 17

NOTE: In many examples in this lab we use Bayesian methods. In those examples we make inferences based on a relatively small number of samples and this is done due to time constraints. In practice, accurate inferences require much more samples.

[1] Suggestions made by Daniel Gianola are gratefully acknowledged.

4.1. Semi-parametric genome-enabled regression

In a standard regression model, the response, y_i, is expressed as the sum of a conditional expectation function, $g(\mathbf{x}_i)$, and a model residual, ε_i, that is $y_i = g(\mathbf{x}_i) + \varepsilon_i$. In previous labs we have focused on the case where $g(\mathbf{x}_i)$ is a linear function of marker genotypes, that is $g(\mathbf{x}_i) = \sum_{j=1}^{p} x_{ij}\beta_j$. Departures from the linear model could theoretically be captured by extending the regression formula with addition of contrasts between marker genotypes, for instance dominance (i.e., within-loci interaction of alleles) could be modeled using dummy variables of the form $d_{ij} = \{1 \text{ if } x_{ij} = 1 ; 0 \text{ ow}\}$, and similar contrasts could be used to model interaction of alleles at different loci (i.e., epitasis). However, with large p the number of possible interaction terms needed to model even modest degree of interactions (e.g., 1s order epistatic interactions) is extremely large and the problem becomes intractable.

Alternatively, we could try to capture departures from the linear model using semi-parametric procedures. This was first suggested in the context of Genomic Selection (GS) by Gianola, Fernando, and Stella (2006) who propose implementing GS using various semi-parametric procedures. Since then, several existing semi parametric procedures have been evaluated in GS. In this lab we focus on Reproducing Kernel Hiblert Spaces (RKHS). Penalized Neural Networks are introduced in LAB 5.

4.2. Reproducing Kernel Hilbert Spaces (RKHS) regressions

Reproducing kernel Hilbert spaces (RKHS) methods are used for semi-parametric modeling in different areas of application such as scatter-plot smoothing (e.g., smoothing spline, Wahba, 1990; spatial smoothing (e.g., Kriging, Cressie 1988); classification problems (e.g., support vector, Vapnik 1998), just to mention a few. Gianola, Fernando, and Stella (2006) suggested using this methodology for semi-parametric genomic enabled prediction. Since then, several authors have discussed and evaluated this methodology in a genomic context.

Estimates in RKHS can be motivated as solution to a penalized optimization problem in a RKHS of real-valued functions or, simply, as posterior modes in certain class of Bayesian models. Next, we provide an overview of the methodology. Detailed discussions of RKHS regressions in the context of genome-enabled prediction can be found in Gianola and van Kaam (2008), de los Campos, Gianola, and Rosa 2009) and de los Campos et al. (2010).

Penalized Regression in Reproducing Kernel Hilbert Spaces

In RKHS regressions we define the set of functions, or space, in which we perform the regression by choosing a reproducing kernel (RK). Technically, the RK can be any positive definite function[2] mapping from pairs of points in input space onto the

[2]For $K(\mathbf{x}_i,\mathbf{x}_{i'})$ to be positive semi definite it must satisfy $\sum_i\sum_{i'}\alpha_i\alpha_{i'}K(\mathbf{x}_i,\mathbf{x}_{i'})K(\mathbf{x}_i,\mathbf{x}_{i'})\geq 0$ for every non-null sequence $\{\alpha_i\}$.

real line, that is $K\left(\mathbf{x}_i,\mathbf{x}_{i'}\right):\left\{\left(\mathbf{x}_i,\mathbf{x}_{i'}\right)\rightarrow\Re\right\}$. For reasons that we will discuss later in this handout you can also think $K\left(\mathbf{x}_i,\mathbf{x}_{i'}\right)$ as a co-variance function. For example, if the input space consists of a pedigree additive relationships $K\left(ID_i,ID_{i'}\right)=a\left(ID_i,ID_{i'}\right)$ constitute a valid RK.

In RKHS regressions the evaluations of functions are expressed as linear combinations of the basis functions provided by the reproducing kernel, RK, $K\left(\mathbf{x}_i,\mathbf{x}_{i'}\right)$, that is $g\left(\mathbf{x}_i\right)=\sum_{i'}K\left(\mathbf{x}_i,\mathbf{x}_{i'}\right)\alpha_{i'}$, and the squared of the norm of the function is given by $\left\|g\right\|^2=\sum_i\sum_{i'}K\left(\mathbf{x}_i,\mathbf{x}_{i'}\right)\alpha_{i'}$.

Stacking the evaluations of the function into a vector yields: $\mathbf{g}=\mathbf{K}\boldsymbol{\alpha}$ and $\left\|\mathbf{g}\right\|^2=\boldsymbol{\alpha}'\mathbf{K}\boldsymbol{\alpha}$, where $\mathbf{g}=\left\{g_i\right\}$, $\mathbf{K}=\left\{K_{ii'}=K\left(\mathbf{x}_i,\mathbf{x}_{i'}\right)\right\}$ and $\boldsymbol{\alpha}=\left\{\alpha_i\right\}$.

Estimates in RKHS are usually obtained as the solution to the following penalized residual sum of squares (intercept and non-maker effects omitted for ease of notation):

$$\hat{\boldsymbol{\alpha}}=\underset{\arg\min}{\left\{\left(\mathbf{y}-\mathbf{K}\boldsymbol{\alpha}\right)'\left(\mathbf{y}-\mathbf{K}\boldsymbol{\alpha}\right)+\lambda\boldsymbol{\alpha}'\mathbf{K}\boldsymbol{\alpha}\right\}} \qquad [1]$$

above, $\left(\mathbf{y}-\mathbf{K}\boldsymbol{\alpha}\right)'\left(\mathbf{y}-\mathbf{K}\boldsymbol{\alpha}\right)$ is a residual sum of squares, $\boldsymbol{\alpha}'\mathbf{K}\boldsymbol{\alpha}$ is a penalty on model complexity, which is taken to be the square of the norm of the function and λ is a regularization parameters.

The solution to the above optimization problem can be shown to be:

$$\hat{\boldsymbol{\alpha}}=\left[\mathbf{K}'\mathbf{K}+\lambda\mathbf{K}\right]^{-1}\mathbf{K}'\mathbf{y}. \qquad [2]$$

Predictions are then obtained as follows:

$$\mathbf{K}\hat{\boldsymbol{\alpha}}=\mathbf{K}\left[\mathbf{K}'\mathbf{K}+\lambda\mathbf{K}\right]^{-1}\mathbf{K}'\mathbf{y}=\left[\mathbf{I}+\lambda\mathbf{K}^{-1}\right]^{-1}\mathbf{y}; \qquad [3]$$

therefore, $\mathbf{K}\left[\mathbf{K}'\mathbf{K}+\lambda\mathbf{K}\right]^{-1}\mathbf{K}'=\left[\mathbf{I}+\lambda\mathbf{K}^{-1}\right]^{-1}$ is the Hat matrix of RKHS.

Model specification in RKHS regression is defined by two main elements[3]: the choice of the reproducing kernel, this functions provide the basis functions and the inner product which define the Hilbert Space, and λ which, as in ridge regression, represents a shrinkage parameter.

4.3. Scatter plot smoothing with a Gaussian kernel

In the following example we will use a RKHS regression to estimate a conditional expectation function non-parametrically. In the example, there is a single predictor, $x_i \in [0,2\pi]$ and the true conditional expectation function is $g(x_i)=120+\sin(x_i)$. Data was generated as $y_i=120+\sin(x_i)+\varepsilon_i$ where $\varepsilon_i \overset{IID}{\sim} N(0,1)$. With this setting, approximately 1/3 rd of the variance of the response is explained by the conditional expectation function and 2/3rd by model residuals.

In this example we use the Gaussian kernel,

$$K(x_i,x_{i'})=\exp\{-h\times d(x_i,x_{i'})\}$$

where: $d(x_i,x_{i'})$ is a distance function which in this example we set to be a squared-Euclidean distance, $d(x_i,x_{i'})=(x_i-x_{i'})^2$, and h is a bandwidth parameter controlling

[3] A third element pertains to the choice of the function used to measure model goodness/lack of fit to the training data. Here we focus on the case where lack of fit is measured by the residual sum of squares; other common choices are the negative of the log-likelihood, this allows modeling continuous, binary and other types of outcomes. For binary outcomes another popular choice is the hinge function, the support vector machine (Vapnik 1998) is a special case of RKHS where the loss-function is chosen to be a hinge function (Wahba 1990).

how fast the kernel decay as the two points, $(x_i, x_{i'})$, get further apart. In the example we evaluate the effects of h (which defines the RK) and of λ.

- Run the code with the values of h and λ given in the example.
- Set h=1/1000, this makes the kernel extremely global, and run the code.
- Set h=50, this makes the kernel extremely local, and run the code.
- Now fix h=1 and change lambda, evaluate ʎ=200, then ʎ=1/100, evaluate results.

Example 1. Scatter-plot smoothing with a Gaussian kernel

```
### SIMULATION##############################################
  set.seed(12345)
  N<-200
  x<-seq(from=0,to=2*pi,length=N)
  signal<-sin(x)
  error<-rnorm(N)
  y<-signal+error
  h<-1
  lambda<-10
### DISTANCE FUNCTION AND REPRODUCING KERNEL #######
  D<-as.matrix(dist(x,method="euclidean"))^2
  K<-exp(-h*D)
  diag(K)<-diag(K) +.001

### FITTING THE MODEL ##############################
  yStar<-y-mean(y)
  KInv<-chol2inv(chol(K))
  C<-KInv*lambda
  diag(C)<-diag(C)+1
  H<-chol2inv(chol(C))  # the Hat matrix
  uHat<-H%*%(y-mean(y))

  plot(y~x, main=paste("lambda=",lambda," h=",h,sep=""))
  lines(x=x,y=signal,col=2,lwd=2)
  lines(x=x,y=uHat+mean(y),col=4,lwd=2)

## want to make the function less local? set h=1/1000,
## want to make it extremely local? set h=100
## Now fix h=1 and change lambda = 200 then lambda= 1/100
```

4.4. Inspecting the Hat Matrix

From eq. [3] predictions are obtained as $\hat{\mathbf{y}} = \left[\mathbf{I} + \mathbf{K}^{-1}\lambda\right]\mathbf{y} = \mathbf{H}\mathbf{y}$, where, $\mathbf{H} = \{h_{ij}\} = \left[\mathbf{I} + \mathbf{K}^{-1}\lambda\right]^{-1}$, therefore, $\hat{g}_i = \sum_j h_{ij} y_j$. The following code displays the entries of the hat matrix of Example 1. You can evaluate the impact of the bandwidth parameter on the weights by changing (in Example 1) *h*.

Example 2. Displaying the entries of the Hat matrix in RKHS

```
### SIMULATION#######################################
  rm(list=ls())
  set.seed(12345)
  N<-200
  x<-seq(from=0,to=2*pi,length=N)
  signal<-sin(x)
  error<-rnorm(N)
  y<-signal+error
  h<-1
  lambda<-10
### DISTANCE FUNCTION AND REPRODUCING KERNEL #######
  D<-as.matrix(dist(x,method="euclidean"))^2
  K<-exp(-h*D)
  diag(K)<-diag(K) +.001

### Hat Matrix  ####################################
  yStar<-y-mean(y)
  KInv<-chol2inv(chol(K))
  C<-KInv*lambda
  diag(C)<-diag(C)+1
  H<-chol2inv(chol(C))  # the Hat matrix
### Plotts the ith row of H ########################
  row<-50
  plot(H[row,]~x, main="",xlab="x(j)",
       type="l", ylab="h(i,j)",col=2)
  abline(v=x[row],col=4) ; abline(h=0)
```

4.5. Bayesian view of RKHS

The solution to the penalized RKHS regression (see eq. [1]) can be shown to be the same than the posterior mode of the vector of regression coefficients in the following Bayesian model:

$$\begin{cases} \mathbf{y} = \mathbf{K}\boldsymbol{\alpha} + \boldsymbol{\varepsilon} \\ \begin{pmatrix} \boldsymbol{\varepsilon} \\ \boldsymbol{\alpha} \end{pmatrix} \Big| \sigma^2_\varepsilon, \sigma^2_g \sim N\left[\mathbf{0}, \begin{pmatrix} \mathbf{I}\sigma^2_\varepsilon & \mathbf{0} \\ \mathbf{0} & \mathbf{K}^{-1}\sigma^2_\alpha \end{pmatrix} \right] \end{cases}$$

[4]

where $\lambda = \sigma^2_\varepsilon \sigma^{-2}_\alpha$. The proof of the equivalence between the posterior mode of $\boldsymbol{\alpha}$ in the Bayesian model described in [4] and the solution given in [2] can be obtained following the same steps used in section 2.5 of LAB 2.

Further, changing variables in [4] from $\mathbf{K}\boldsymbol{\alpha}$ to $\mathbf{g} = \mathbf{K}\boldsymbol{\alpha}$, and noting from the properties of the MVN density (see section 2.6 of LAB 2) that $\mathbf{g} \sim MVN\left(\mathbf{0}, \mathbf{K}\sigma^2_g\right)$, where $\sigma^2_\alpha = \sigma^2_g$, we obtain an equivalent representation of [4],

$$\begin{cases} \mathbf{y} = \mathbf{g} + \boldsymbol{\varepsilon} \\ \begin{pmatrix} \boldsymbol{\varepsilon} \\ \mathbf{g} \end{pmatrix} \Big| \sigma^2_\varepsilon, \sigma^2_g \sim N\left[\mathbf{0}, \begin{pmatrix} \mathbf{I}\sigma^2_\varepsilon & \mathbf{0} \\ \mathbf{0} & \mathbf{K}\sigma^2_g \end{pmatrix} \right] \end{cases}$$

[5]

Therefore, from the Bayesian perspective, the evaluations of functions at points in the input space, $\mathbf{g} = \{g(\mathbf{x}_i)\}$ are viewed as realizations from Gaussian process satisfying: $Cor[g(\mathbf{x}_i), g(\mathbf{x}_{i'})] = \frac{K(\mathbf{x}_i, \mathbf{x}_{i'})}{\sqrt{K(\mathbf{x}_i, \mathbf{x}_i)K(\mathbf{x}_{i'}, \mathbf{x}_{i'})}}$. Here, the RK $K(\mathbf{x}_i, \mathbf{x}_{i'})$ is viewed as a (co)variance function whcih defines a notion of smoothens of the function with respect to points in the input space (genotypes in our case). A high value of $Cor[g(\mathbf{x}_i), g(\mathbf{x}_{i'})]$ implies that, a-priori, we expect the function to behave smoothly when we jump from $\mathbf{x}_i$ to $\mathbf{x}_{i'}$. At the same time, this means y_i is informative about $g(\mathbf{x}_{i'})$ and that $y_{i'}$ informs us something about $g(\mathbf{x}_i)$.

Special cases. Certain parametric models appear as special cases of RKHS regression. For instance, if our information set consists of a pedigree and **K** is a matrix of additive relationship matrix, the model defined by [1] is equivalent to the infinitesimal additive model, the so-called **Animal Model**. The Bayesian ridge regression and GBLUP (see section 2.6 of LAB 2) is another example of a parametric model that can be represented as a RKHS, this is obtained by setting $\mathbf{K} = \mathbf{XX}'$. These are examples where the RK is chosen so as to represent the types of patterns expected under a parametric model. Another alternative is to choose kernels based on their performance (e.g., predictive ability). In this lab we will focus on this second approach.

4.6. Genomic-Enabled Prediction Using RKHS

In this section we use the Gaussian kernel for genomic-enabled prediction. To this end, we replace the distance function by a genomic-distance. For instance, we can set

$d(\mathbf{x}_i, \mathbf{x}_{i'}) = \sum_j (x_{ij} - x_{i'j})^2$; the Gaussian kernel becomes: $K(x_i, x_{i'}) = \exp\{-h \times d(\mathbf{x}_i, \mathbf{x}_{i'})\}$.

The function `dist()` of R takes tow arguments: x which should be a numeric vector or matrix, and methods, which should be a string indicating the method fro computing distances. By default the Euclidean distance is computed. Type `help(dist)` for further details. The function returns an object, which can be converted to an n×n matrix, containing pairwise distance between the rows of **X**.

The example below fits the model over a grid of values of the bandwidth parameter (*h*) and evaluates the effect of it on goodness of fit, model complexity and predictive ability.

- Run the code;
- Evaluate how goodness of fit and predictive ability changes with *h*
- How does $\lambda = \frac{\sigma_\varepsilon^2}{\sigma_g^2}$ changes with *h*?

Example 3. RKHS for Genomic Prediction

```
 rm(list=ls())
 setwd('~/Dropbox/Armidale/')
 load("PROGRAMS/RKHS/RKHS.rda")
 library(BLR)
 data(wheat)

### DISTANCE MATRIX ##############################
 D<-as.matrix(dist(X,method="euclidean"))^2
 D<-D/mean(D)
 h<-c(1e-2,.1,.4,.8,1.5,3,5)

### GENERATES TESTING SET ########################
 set.seed(12345)
 tst<-sample(1:599,size=100,replace=FALSE)
 y<-Y[,4]
 yNA<-y
 yNA[tst]<-NA

### FITS MODELS ##################################
 PMSE<-numeric() ; VARE<-numeric(); VARU<-numeric() ;
 pD<-numeric(); DIC<-numeric()
 fmList<-list()
 for(i in 1:length(h)){
   print(paste('Working with h=',h[i],sep=''))
   # COMPUTES THE KERNEL
   K<-exp(-h[i]*D)
   # FITS THE MODEL
   prefix<- paste(h[i], "_",sep="")
   fm<-RKHS(y=yNA,K=list(list(K=K,df0=5,S0=2)),
          nIter=5000,burnIn=1000,df0=5,S0=2,saveAt=prefix)
   fmList[[i]]<-fm
   PMSE[i]<-mean((y[tst]-fm$yHat[tst])^2)
   VARE[i]<-fm$varE
   VARU[i]<-fm$K[[1]]$varU
   DIC[i]<-fm$fit$DIC
   pD[i]<-fm$fit$pD
 }
 R2<-1-PMSE/mean((y[tst]-mean(y[-tst]))^2)

### PLOTS ########################################
 plot(VARE~h,xlab="Bandwidth", ylab="Residual Variance",type="o",col=4)

 plot(I(VARE/VARU)~h,xlab="Bandwidth",
       ylab="variance ratio (noise/signal)",type="o",col=4)

 plot(pD~h,xlab="Bandwidth", ylab="pD",type="o",col=2)

 plot(DIC~h,xlab="Bandwidth", ylab="DIC",type="o",col=2)

 plot(R2~h,xlab="Bandwidth", ylab="R-squared",type="o",col=2)

```

4.7. Kernel Averaging

The choice of the RK (its functional form and the values of parameters such as the bandwidth) constitutes the central element of model specification in RKHS regressions. There are several ways of choosing a kernel. In **parametric models**, the RK is chosen to represent the type of patterns expected under a particular parametric model (e.g., additive infinitesimal, **K**=**A**; linear model, **K**=**XX**'). Form a **non-parametric** perspective one can choose kernels based on the performance of the model, e.g., predictive ability; an illustration of this was provided in the previous example where a validation set was used to evaluate predictive ability of RKHS using a Gaussian kernel, over a grid of values of the bandwidth parameter.

A third way is by **inferring the kernel** from the data. For instance, in a Bayesian context one could assign a prior to the bandwidth parameter and infer this parameter jointly with other unknowns. While this is appealing, it is computationally demanding for at least two reasons: (a) the RK must be re-computed every time a new value of the bandwidth parameter is sampled; (b) mixing may be poor. This occurs because, usually, variance parameters and the bandwidth parameter are highly correlated at the posterior distribution. An alternative which we consider here is to offer the algorithm all candidate kernels jointly. For instance, we can make the conditional expectation to be a sum of several random effects, $\{\mathbf{g}_1,...,\mathbf{g}_{N_k}\}$, each of which has its own (co)variance function, the model becomes:

$$\begin{cases} \mathbf{y} = \mathbf{1}\mu + \sum_{k=1}^{N_k} \mathbf{g}_k + \boldsymbol{\varepsilon} \\ p\left(\boldsymbol{\varepsilon}, \mathbf{g}_1, ..., \mathbf{g}_{N_k} \middle| \sigma_\varepsilon^2, \sigma_{g_1}^2, .., \sigma_{g_{N_k}}^2\right) = N\left(\boldsymbol{\varepsilon} \middle| \mathbf{0}, \mathbf{I}\sigma_\varepsilon^2\right) \prod_{k=1}^{N_k} N\left(\mathbf{g}_k \middle| \mathbf{0}, \mathbf{K}_k \sigma_{g_k}^2\right) \end{cases}$$

It can be shown that, conditional on variance parameters, the above model is equivalent to one with a single random effect, **g**, whose prior distribution is $N\left(\mathbf{g} \middle| \mathbf{0}, \overline{\mathbf{K}}\sigma_g^2\right)$ where: $\overline{\mathbf{K}} = \mathbf{K}_1\alpha_1 + \mathbf{K}_2\alpha_2 + ... + \mathbf{K}_{N_k}\alpha_{N_k}$ is a weighted sum of the candidate kernels with weight given by $\alpha_k = \frac{\sigma_{g_k}^2}{\sigma_g^2}$ and $\sigma_g^2 = \sum_k \sigma_{g_k}^2$. Variance parameter here can then be seen as weights associated to each kernel which can be inferred from the data. The larger the variance associated to a given kernel the larger the contribution of that random effect to the conditional expectation We refer to this approach as kernel averaging (KA, de los Campos et al., 2010).

The following example illustrates the use of KA; the sequence of kernels was generated using the Gaussian kernel and the values of the bandwidth parameter used in our previous example.

- Run the code below.
- What Kernel gets higher weight?
- Is that the Kernel that gave highest predictive ability in our previous example?
- Compare the predictive ability of KA with that of models fitted in our previous example (i.e., single kernel with fixed bandwidth).

Example 4. Kernel Averaging

```
  rm(list=ls())
  setwd('~/Dropbox/Armidale/') ; load("PROGRAMS/RKHS/RKHS.rda")

  library(BLR)
  data(wheat)
  D<-as.matrix(dist(X,method="euclidean"))^2
  D<-D/mean(D)
  h<-c(1e-2,.1,.4,.8,1.5,3,5)

### GENERATES TESTING SET ######################
  set.seed(12345)
  tst<-sample(1:599,size=100,replace=FALSE)
  y<-Y[,4]
  yNA<-y
  yNA[tst]<-NA

### FITS MODELS ################################
  PMSE<-numeric()
  VARE<-numeric()
  KList<-list()
  for(i in 1:length(h)){
    KList[[i]]<-list(K=exp(-h[i]*D),df0=5,S0=.5)
  }

## Displays entries of different kernels
  plot(KList[[1]]$K[100,],ylim=c(0,1),col=2);abline(v=100)

  plot(KList[[5]]$K[100,],ylim=c(0,1),col=2);abline(v=100)

  fmKA<-RKHS(y=yNA,K=KList,thin=10,
           nIter=25000,burnIn=5000,df0=5,S0=1,saveAt="KA_")

  VARG<-numeric()
  for(i in 1:length(KList)){  VARG[i]<-fmKA$K[[i]]$varU }
  weights<-round(VARG/sum(VARG),5)

  PMSE<-mean((y[tst]-fmKA$yHat[tst])^2)
  R2_KA<-1-PMSE/mean((y[tst]-mean(y[-tst]))^2)

  # compare with results obtained in the previous example
  # take a look at the trace plots of variance parameters


```

4.8. Pedigree + Marker Models

The following code compares the entries of a pedigree-based additive relationship matrix versus that of two marker-based genomic relationships. The first one (`XX'`, denoted as `XXt`) is the co-variance structure corresponding to a linear regression on marker-covariates with IID normal marker effects (what we have called the Bayesian Ridge Regression). The second one (`denoted as K`) is a Gaussian kernel.

Example 5. Pedigree Vs marker based relationship matrices

```
rm(list=ls())
library(BLR)
setwd('~/Dropbox/Armidale/') ; load("PROGRAMS/RKHS/RKHS.rda")
data(wheat) ; for(i in 1:ncol(X)){ X[,i]<-(X[,i]-mean(X[,i]))/sd(X[,i]) }

D<-as.matrix(X,method='euclidean')^2
D<-D/mean(D)
K<-exp(-2*D)
G<-tcrossprod(X)/ncol(X)

## plot of entries of XXt versus A
tmpX<-as.vector(A)
tmpY<-as.vector(G)
tmp<-range(c(tmpX,tmpY))
plot(tmpY~tmpX,xlab='A',ylab='G',cex=0.3,col=2,xlim=tmp,ylim=tmp)
```

Example 6. RKHS with markers and pedigree

```
 rm(list=ls())
 library(BLR)
 setwd('~/Dropbox/Armidale/') ; load("PROGRAMS/RKHS/RKHS.rda")
 data(wheat) ; for(i in 1:ncol(X)){ X[,i]<-(X[,i]-mean(X[,i]))/sd(X[,i]) }

### Generates Testing Sets ######################
    set.seed(12345)
    tst<-sample(1:599,size=100,replace=FALSE)
    y<-Y[,4] ; yNA<-y; yNA[tst]<-NA; KList<-list()

### First the pedigree-model ####################
    KList[[1]]<-list(K=A,df0=5,S0=.2)
    fmP<-RKHS(y=yNA,K=KList,thin=10,
           nIter=6000,burnIn=1000,df=5,S0=1,saveAt="P_")
    PMSE<- mean((y[tst]-fmP$yHat[tst])^2)
    R2_P<-1-PMSE /mean((y[tst]-mean(y[-tst]))^2)

### Now Markers #################################
    G<-tcrossprod(X)/ncol(X)
    KList[[1]]<-list(K=G,df0=5,S0=.2)
    fmM<-RKHS(y=yNA,K=KList,thin=10,
              nIter=6000,burnIn=1000,df=5,S0=1,saveAt="M_")
    PMSE<- mean((y[tst]-fmM$yHat[tst])^2)
    R2_M<-1-PMSE /mean((y[tst]-mean(y[-tst]))^2)

### Now Markers and pedigree ####################
    KList[[1]]<-list(K=A,df0=5,S0=.1)
    KList[[2]]<-list(K=G,df0=5,S0=.1)

    fmPM<-RKHS(y=yNA,K=KList,thin=10,
           nIter=6000,burnIn=1000,df=5,S0=1,saveAt="PM_")
    PMSE<- mean((y[tst]-fmPM$yHat[tst])^2)
    R2_PM<-1-PMSE /mean((y[tst]-mean(y[-tst]))^2)

## Now Lets add XXt#XXt ###################
    KList[[1]]<-list(K=A,df0=5,S0=.1)
    KList[[2]]<-list(K=G,df0=5,S0=.05)
    KList[[3]]<-list(K=I(G^2),df0=5,S0=.05)

    fmPM2<-RKHS(y=yNA,K=KList,thin=10,
           nIter=15000,burnIn=5000,df=5,S0=1,saveAt="PM2_")
    PMSE<- mean((y[tst]-fmPM2$yHat[tst])^2)
    R2_PM2<-1-PMSE /mean((y[tst]-mean(y[-tst]))^2)

    library(graphics)
    barplot(height=c(R2_P,R2_M,R2_PM,R2_PM2),
            names.arg=c('P','M','PM','PM2'), ylab='R-sq. TRN set',col=2)
## Take a look at trace plots of variance parameters
```

References

de los Campos, G., D. Gianola, G. J. M. Rosa, K. A Weigel, and J. Crossa. 2010. "Semi-parametric Genomic-enabled Prediction of Genetic Values Using Reproducing Kernel Hilbert Spaces Methods." *Genetics Research* 92 (04): 295–308.

de los Campos, G., D. Gianola, and G. J.M Rosa. 2009. "Reproducing Kernel Hilbert Spaces Regression: a General Framework for Genetic Evaluation." *Journal of Animal Science* 87 (6): 1883.

Cressie, N. 1988. "Spatial Prediction and Ordinary Kriging." *Mathematical Geology* 20 (4): 405–421.

Gianola, D., and J. B van Kaam. 2008. "Reproducing Kernel Hilbert Spaces Regression Methods for Genomic Assisted Prediction of Quantitative Traits." *Genetics* 178 (4): 2289.

Gianola, Daniel, Rohan L. Fernando, and Alessandra Stella. 2006. "Genomic-Assisted Prediction of Genetic Value With Semiparametric Procedures." *Genetics* 173 (3) (July 1): 1761-1776. doi:10.1534/genetics.105.049510.

Vapnik, V. N. 1998. "Statistical Learning Theory."

Wahba, G. 1990. "Spline Methods for Observational Data." *SIAM: Philadelphia*.

Statistical Methods for Genome-Enabled Prediction,

LAB 5:

Penalized Neural Networks[1]

(gcampos@uab.edu)

Contents

5.1. Introduction 2

5.2. Scatterplot smoothing using a penalized NN 5

5.3. Penalized Neural Network Using Pre-selected Markers 7

5.4. Penalized Neural Networks Using Marker-derived Basis Functions as Inputs 8

References 9

[1] Suggestions made by Paulino Pérez are gratefully acknowledged.

5.1. Introduction

In **linear regression** models the conditional expectation is represented as a weighted sum of input variables, $E\left(y_i \middle| \mathbf{x}_i\right) = \sum_{j=1}^{p} x_{ij}\beta_j$. Many **non-linear patterns** can be represented linearly by appropriate choice of **basis functions:** $E\left(y_i \middle| \mathbf{x}_i\right) = \sum_{m=0}^{M} \phi\left(\mathbf{x}_i\right) w_m$ where, $\left\{ \phi_m\left(\mathbf{x}_i\right) \right\}_{m=1}^{M}$ are the basis functions, which map from the input variables onto the real line. An example of these are the polynomial basis functions: $\Phi = \left\{ \varphi_m\left(x_i\right) = x_i^m \right\}_{m=0}^{M}$. For instance, if M=2 we have the 2nd degree polynomial basis functions, $\Phi = \left\{1, x_i, x_i^2 \right\}$; therefore, $E\left(y_i \middle| \mathbf{x}_i\right) = \beta_0 + \beta_1 x_i + \beta_2 x_i^2$. Other common examples of non-linear basis functions are the power, logarithm and exponential functions. With this types of basis functions each of the regression coefficients affect the behavior of the conditional expectation in the entire input space, and this may limit the ability of a model to capture the local behavior of the conditional expectation.

Local basis functions can be used to model a conditional expectation within certain regions of the input space. **Splines** represent an example of this. In a spline**,** polynomial basis functions are used to represent the regression function within boundaries defined by a set of knots. The **Gaussian kerne**l discussed in LAB4 is another example of a local basis function, here $\varphi_m\left(\mathbf{x}_i, \mathbf{t}_m, h\right) = e^{-h\left\|\mathbf{x}_i - \mathbf{t}_m\right\|^2}$ where $\mathbf{t}_m$ is a focal point and h is a bandwidth parameter which controls how fast the basis function decay as $\mathbf{x}_i$ gets further apart from the focal point. Model specification in this case pertains to the choice of focal points (how many and where in input space should be placed) and of the bandwidth parameter. In the RKHS regressions of LAB4, the strategy was to 'offer' the model a large set of basis functions (one per subject in the sample) generated by setting $\mathbf{t}_1 = \mathbf{x}_1, \mathbf{t}_2 = \mathbf{x}_2, \ldots, \mathbf{t}_n = \mathbf{x}_n$; therefore $E\left(y_i \middle| \mathbf{x}_i\right) = \sum_{i'=1}^{n} \alpha_{i'} \times e^{-h\left\|\mathbf{x}_i - \mathbf{x}_{i'}\right\|^2}$. This strategy may induce over-fitting and this was confronted by using shrinkage estimation procedures. This is approach is also used in smoothing spline (Craven and Wahba 1978; Wahba 1991).

Non-linear basis functions such as the ones described above offer great potential for capturing potentially complex patterns between input and output variables; however, the set of basis functions needs to be defined a-priori. In Neural Networks (NN) the basis functions used for regression are inferred (i.e., are data driven), this gives NN great potential for capturing potentially complex patterns.

One of the simplest NNs is the ***single hidden layer feed-forward NN***. This NN can be thought as non-linear regressions consisting of two steps (Hastie, Tibshirani, and Friedman 2009): in the first one (or hidden layer) the basis functions are inferred, and in the second one (or output layer) the output, y_i, is regressed on the basis function inferred in the hidden layer. A graphical representation of such NN is given

in Figure 1. The term feed-forward is used to highlight that in these NNs information flows from inputs (the $\mathbf{x}_i$'s) to output (the y_I's), other NN allow feedbacks.

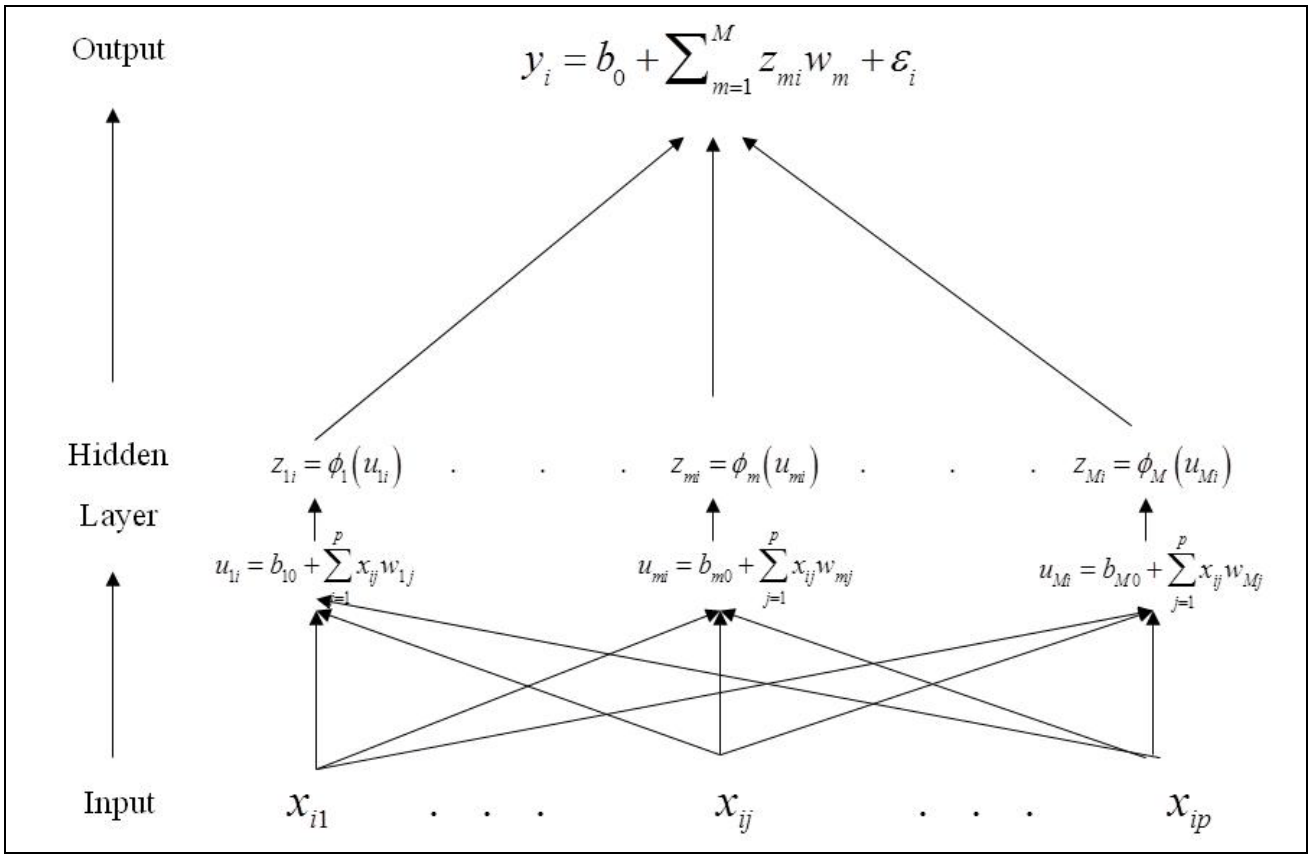

Figure 1. Graphical Representation of Single Hidden Layer Feed-Forward Neural Network for a Continuous Response (y_i) and p predictor variables ($x_{1i},\ldots,x_{ip}$). The network contains M neurons. At each neuron, linear combinations of the predictors ($u_{mi} = b_{m0} + \sum_{j=1}^{p} x_{ij} w_{mj}$) are inferred and subsequently activated $z_{mi} = \phi_m(u_{mi})$. These basis functions are then used in the output layer to regress the output variable using a linear model ($y_i = b_0 + \sum_{m=1}^{M} z_{mi} w_m + \varepsilon_i$).

As illustrated in Figure 1, in the **hidden layer** M basis functions, $\varphi_m\left(b_{m0} + \sum_{j=1}^{p} x_{ij} w_{mj}\right)$, are inferred (one at each **neuron)**. Each of these basis functions consist of a linear score, $u_{mi} = b_{m0} + \sum_{j=1}^{p} x_{ij} w_{mj}$, activated by a non-linear **activation function**, $\varphi_m(.)$.

In the **output layer**, the outcome, y_i, is regressed on the basis functions using an additive model. The example of Figure 1 is for a continuous response; in many applications with NN the outcome is either binary or polychotomous. In those cases an additional activation functions are added in the output layer. Note that, if the activation function of the hidden and output layers are identity functions (i.e., $\varphi_m\left(u_{im}\right)=u_{im}$ the model of Figure 1 becomes a standard multiple linear regression model. Moreover, if we set the $\varphi_m\left(.\right)$ to be the basis functions of a reproducing kernel (see LAB4), the NN of Figure 1 becomes the RKHS regression. Therefore, we can view the NN of figure 1 as a general framework that includes the linear model and the RKHS as special cases.

The **activation functions** of the hidden layers map from the real line onto the [0,1] interval, and a common choice is to set this to be a sigmoid function. For instance we could use $\phi_m\left(z_{mi}\right)=\frac{1}{1+e^{-\theta\times z_{mi}}}$ for some $\theta>0$.

Architecture of a Neural Network. The elements that define model specification in NN are: (a) the choice of input variables, (b) the type of network (e.g., feed-forward), (c) the number of layers, (d) the number of neurons per layer, and (d) the choice of activation functions. In general the term 'architecture' of the network is used to referred to the choices made in (b)-(d).

Penalized Neural Networks. The set of parameters to be estimated in the NN of Figure 1 include: all the intercepts and regression coefficients at each neurons, the parameters of the activation functions, and the intercept and regression coefficients of the output layer. With large *p,* and with several neurons, the total number of parameters to be estimated can be huge. This, together with the intrinsic flexibility of the NN, can easily yield over-fitting and poor predictive performance. To prevent this, a common strategy is to fit the neural network using penalized methods such as those discussed in LAB2. Therefore, in a penalized NN, parameters are estimated by minimizing an objective function consisting of a lack-of fit function (e.g., a residual sum of squares) plus a penalty on model complexity. Any of the penalties discussed in LAB 2 can be used; however, a common choice is to set the penalty to be the of regression coefficients (usually intercepts are not penalized).

In what remains of the lab we illustrate the use of penalized NN using a beta version of the R-package `trainbr`. This package was developed and kindly shared by Paulino Perez.

5.2. Scatterplot smoothing using a penalized NN

The following example illustrates the use of penalized NN for scatter-plot smoothing.

Example 1: Scatter-plot smoothing Using a Neural Network

```
  rm(list=ls());library(trainbr) ; library(splines)
### SIMULATION (same as the one used in Ex. 1 of LAB4) #####
  set.seed(12345)
  N<-200
  x<-seq(from=0,to=2*pi,length=N)
  signal<-sin(x)
  error<-rnorm(N)
  y<-signal+error

  # for train-br the oucome variable needs to be standardized to [0,1]
  yStd<-normalize(y)
  signalStd<-2*(signal-min(y))/(max(y)-min(y))-1

## Various parametric models
  lm1<-lm(y~x)
  poly3<-lm(yStd~x+I(x^2)+I(x^3))
## Natural spline with 4 knots
   X<-ns(x=x,df=4)
  fmNS<-lm(yStd~X)
## Neural Networks with 1,2,3 and 5 nuerons
  NN1<-trainbr(y=yStd,X=as.matrix(x),neurons=1)
  yHatNN_1<-predictions.nn(X=as.matrix(x),theta=NN1$theta, neurons=1)

  NN2<-trainbr(y=yStd,X=as.matrix(x),neurons=2)
  yHatNN_2<-predictions.nn(X=as.matrix(x),theta=NN2$theta, neurons=2)

  NN3<-trainbr(y=yStd,X=as.matrix(x),neurons=3)
  yHatNN_3<-predictions.nn(X=as.matrix(x),theta=NN3$theta, neurons=3)

  NN4<-trainbr(y=yStd,X=as.matrix(x),neurons=4)
  yHatNN_4<-predictions.nn(X=as.matrix(x),theta=NN4$theta, neurons=4)

  NN5<-trainbr(y=yStd,X=as.matrix(x),neurons=5)
  yHatNN_5<-predictions.nn(X=as.matrix(x),theta=NN5$theta, neurons=5)

#(continues next page)
```

Example 1: Scatter-plot smoothing Using a Neural Network

```
# (FROM PREVIOUS PAGE)
## R-Squared ##############################################################
  R2_lm<-1-mean((signalStd-predict(lm1))^2)/var(signalStd)
  R2_ply3<-1- mean((signalStd-predict(poly3))^2)/var(signalStd)
  R2_NS<-1- mean((signalStd-predict(fmNS))^2)/var(signalStd)
  R2_NN<-numeric()
  R2_NN[1]<-1-mean((signalStd-yHatNN_1)^2)/var(signalStd)
  R2_NN[2]<-1-mean((signalStd-yHatNN_2)^2)/var(signalStd)
  R2_NN[3]<-1-mean((signalStd-yHatNN_3)^2)/var(signalStd)
  R2_NN[4]<-1-mean((signalStd-yHatNN_5)^2)/var(signalStd)
  R2_NN[5]<-1-mean((signalStd-yHatNN_5)^2)/var(signalStd)

## Plots ##################################################################
  plot(yStd~x,col=1,cex=.5)
  lines(x=x,y=signalStd,lwd=2,col=2)
  lines(x=x,y=yHatNN_3,col=4,lwd=4,lty=2)

  plot(R2_NN~I(1:5),
       xlab='Number of Neurons',ylab= 'R2(Pred. vs signal',type='o'
       , col=4)
  abline(h=R2_NS,col=4,lty=2)

```

Example 1 illustrates the flexibility that NNs have in terms of capturing complex patters: starting from a single predictor, the NN generated complexity by inferring multiple basis functions which were able to capture the non-linear patterns between inputs and outputs very well. The example uses a single predictor, but as illustrated in Figure 1 the method could also be applied to multiple-predictors. However, with large p and with multiple neurons, the computational requirements increase substantially.

5.3. Penalized Neural Network Using Pre-selected Markers

In Example 2 we first select the top p markers from single marker regressions and subsequently offer these markers to a NN with 3 neurons.

Example 2: Penalized Neural Network Applied to Pre-selected Markers

```
1  rm(list=ls())
2  ### DATA ###############################################
3   library(BLR) ; library(trainbr) ;  data(wheat)
4   N<-nrow(X) ; p<-ncol(X)
5   y<-Y[,4]
6   y<-normalize(y)
7   set.seed(1235)
8   tst<-sample(1:N,size=150,replace=FALSE)
9   XTRN<-X[-tst,] ; yTRN<-y[-tst]
10  XTST<-X[tst,] ;  yTST<-y[tst]
11 ### SINGLE MARKER REGRESSIONS ########################
12  pValues<-numeric()
13  for(i in 1:p){
14       fm<-lm(yTRN~XTRN[,i])
15       pValues[i]<-summary(fm)$coef[2,4]
16       print(paste('Fitting Marker ',i,'!',sep=''))
17  }
18  nMarkers<-75
19  selSNPs<-order(pValues)[1:nMarkers]
20  XTRN<-XTRN[,selSNPs]
21  XTST<-XTST[,selSNPs]
22
23 ### Neural Network ###################################
24   NN<-trainbr(y=yTRN,X=XTRN,neurons=4, epochs=100)
25   yHatNN<-predictions.nn(X=XTST,theta=NN$theta, neurons=4)
26  cor(yHatNN,y[tst])
27
28 ## Change the number of pre-selected markers (line 22) and number of
29 ## Neurons (lines 28 and 29) and experiment.
```

5.4. Penalized Neural Networks Using Marker-derived Basis Functions as Inputs

In Example 2 we pre-selected markers, another strategy consist of first mapping the input information into some basis functions (e.g., using a reproducing kernel or using genomic relationships) and then applying the NN to these basis functions. For instance, Gianola et al. (2011) suggested using the additive relationships as basis functions, by so doing we reduce the number of input variables of the NN from p to n. In Example 3 we illustrate this approach by using as inputs to the NN marker-derived principal components.

Example 3: Penalized Neural Network Applied to Marker-derived Principal Components

```
rm(list=ls())
### DATA ##############################################
 library(BLR) ;library(trainbr) ; data(wheat)
 for(i in 1:ncol(X)){ X[,i]<-X[,i]-mean(X[,i])}
 N<-nrow(X) ; p<-ncol(X)
 y<-Y[,4]
 y<-normalize(y)
 ## Pcs
 SVD<-svd(X,nu=599,nv=0)
 PC<-SVD$u ; for(i in 1:ncol(PC)){ PC[,i]<-PC[,i]*SVD$d[i] }
 plot(PC[,1:2],col=4)
 set.seed(1235)
 tst<-sample(1:N,size=150,replace=FALSE)
 yTRN<-y[-tst]
 yTST<-y[tst]
 PCTrn<-PC[-tst,]

 PCTst<-PC[tst,]

 nPC<-300
 NN<-trainbr(y=yTRN,X=PCTrn[,1 :nPC],neurons=3, epochs=150)
 yHatNN<-predictions.nn(X=PCTst[,1:nPC],theta=NN$theta,
             neurons=c(length(NN$theta)-1))
 cor(yHatNN,yTST)
```

References

Craven, P., and G. Wahba. 1978. “Smoothing Noisy Data with Spline Functions.” *Numerische Mathematik* 31 (4): 377–403.

Gianola, D., H. Okut, K. Weigel, and G. Rosa. 2011. “Predicting Complex Quantitative Traits with Bayesian Neural Networks: a Case Study with Jersey Cows and Wheat.” *BMC Genetics* 12 (1): 87.

Hastie, Trevor, Robert Tibshirani, and Jerome Friedman. 2009. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Second Edition*. 2nd ed. 2009. Corr. 3rd printing 5th Printing. Springer.

Wahba, G. 1991. “Spline Functions for Observational Data.” *SIAM, Philadelphia, PA*.

Statistical Methods for Genome-Enabled Prediction,

LAB 6:

Validation Methods[1]

(gcampos@uab.edu)

Contents

6.1. Introduction 2

6.2. Alternative Validation Schemes 3

6.3. Between sub-population prediction 7

6.4. Across environment prediction using single-trait models 8

References 9

NOTE: In many examples in this lab we use Bayesian methods. In those examples we make inferences based on a relatively small number of samples and this is done due to time constraints. In practice, accurate inferences require much more samples.

[1] Suggestions made by Daniel Gianola are gratefully acknowledged.

6.1. Introduction

Prediction is a central problem in plant and animal breeding and in many other domains. It is natural to compare models based on their ability to predict future outcomes. Validation methods aim at estimating the distribution (or features of it, e.g., the variance) of prediction errors.

Prediction error. Let $S_{TRN} = \{y_i, \mathbf{x}_i\}$ denote the available training data, M a model (or algorithm) and $\{y_{new}, \mathbf{x}_{new}\}$ an un-observed data point that we want to predict. The algorithm processes the training sample and derives a prediction: $\hat{y}_{new}(\mathbf{x}_{new}, M, S_{TRN})$. Example: using training data, S_{TRN}, and a linear model (M) we estimate marker effects and then we use the estimated marker effects and the genotypes of candidates of selection ($\mathbf{x}_{new}$) to derive predictions. The prediction error is $\hat{\varepsilon}_{new} = y_{new} - \hat{y}_{new}$. Model performance can then be assessed using features of the distribution of prediction errors.

Validation methods. Deriving a closed form for the distribution of prediction errors requires making assumptions about the true data generating process. In practice we do not know such process and models are, at best, good approximations. However, if we are able to draw a large number of samples from the desired prediction errors $\{\hat{\varepsilon}_{new,i}\}$, we can then estimate features of the density of prediction errors using Monte Carlo methods. For instance, given a large number of sample of prediction errors we could estimate the proportion of variance of future phenotypes accounted for by predictions using an R-squared type statistic: $R^2_{TST} = 1 - \dfrac{\sum_i \hat{\varepsilon}^2_{new,i}}{\sum_i \left(y_{new,i} - \bar{y}_{new}\right)^2}$.

In practice we have only a finite sample of data and most validation methods emulate the sampling process by sampling data points using some type of resampling method. There are different types of prediction errors, and the design of the validation scheme will determine what type of prediction errors are we describing.

Conditional error. Typically, we want to estimate the distribution of the prediction error given the training sample, that is, $p(\hat{\varepsilon}_{new}|S_{TRN})$. Here, prediction errors are random variables because they are functions of yet-to-be-observed genotypes and phenotypes. Intuitively, we can obtain draws from the distribution of conditional errors by first fitting the model (only once) to the available TRN sample and subsequently evaluating the prediction accuracy of the model we derived by sampling testing samples.

Marginal prediction errors are obtained by averaging the density of conditional errors over all possible realizations of the training sample: $p(\hat{\varepsilon}_{new}) = E\left[p(\hat{\varepsilon}_{new}|S_{TRN})\right] = \int p(\hat{\varepsilon}_{new}|S_{TRN})p(S_{TRN})\partial S_{TRN}$. Intuitively we can estimate the marginal distribution of prediction error with re-sampling of both raining and testing datasets.

In most applications, our interest is to estimate the density of conditional errors; however this density is difficult to estimate and most of the methods we will see estimate $p(\hat{\varepsilon}_{new})$ (Hastie, Tibshirani, and Friedman 2009).

6.2. Alternative Validation Schemes

Training-Testing (TRN-TST) Validation

If sample size is large we can simply assign some individuals for training (TRN) and some for testing (TST). We use TRN to fit the model and derive prediction errors from TST. We have done so in previous labs by partitioning at random the wheat dataset into TRN and TST. If the prediction problem of interest has certain structure (e.g., ancestors will be used for training with the goal of predicting performance of progeny) the partition of the data into TRN and TST should reflect such structure. This has been done, for instance for validation of methods for genomic selection in dairy cattle. Unfortunately we can't do this with the wheat dataset because we lack a pedigree.

Cross-validation (CV)

One disadvantage of the TRN/TST design above described is that individuals are either used for training or testing. When the total sample size is large this is not a problem; however, with small sample size one would like to use all individuals both for training and testing CV allows this. In CV individuals are randomly assigned to disjoint sets using an index, for example, in 2-fold CV each individual is assigned to either 1^{st} or 2^{nd} fold. Then, a TRN/TST evaluation is done for every fold. In those evaluations, individuals assigned to that fold are regarded as TST set and the remaining ones as TRN set. The following R-code implements a 5-fold CV using the wheat dataset.

Example 1: 5-fold CV

```
###  LOADS DATA  ############################
  rm(list=ls());  library(BLR);  data(wheat)
  y<-Y[,4]
  for(i in 1:ncol(X)){ X[,i]<-(X[,i]-mean(X[,i]))/sd(X[,i]) }
  h2<-0.5 ; lambda<-(1-h2)/h2*ncol(X)
### ASSIGNMENT TO FOLDS (5-fold CV) ###########
  set.seed(124292)
  sets<-sample(1:5,size=nrow(X),replace=TRUE)
  yHatCV_RR<-rep(NA,length(y))
  yHatCV_0<- rep(NA,length(y))
  varE<-numeric()
  indexH<-rep(NA,length(y))
  for(fold in 1:5){
    tst<-which(sets==fold) # here we partition the data
    C<-crossprod(X[-tst,])
    for(j in 1:ncol(C)){ C[j,j]<- C[j,j]+lambda }

    CInv<-chol2inv(chol(C))
    H<-X[tst,]%*%CInv%*%t(X[-tst,])
    indexH[tst]<-rowSums(abs(H)>.15) # count entries > 0.15 in H
    yHatCV_RR[tst]<- H%*%y[-tst]
    yHatCV_0[tst]<-mean(y[-tst])
     print(fold)
  }

 sqErrorRR<-(y-yHatCV_RR)^2
 sqError0<-(y-yHatCV_0)^2

 PMSE_RR<-tapply(X=sqErrorRR,FUN=mean,INDEX=sets)
 PMSE_0<-tapply(X=sqError0,FUN=mean,INDEX=sets)
 R2<-1-PMSE_RR/PMSE_0 # compare to cor(y,yHatCV)^2
 sqrt(R2)

## Three different ways of computing R2: discuss!
  cor(y,yHatCV_RR)^2
  1-var(y-yHatCV_RR)/var(y)
  1-sum((y-yHatCV_RR)^2)/sum((y-yHatCV_0)^2)

## Relationships between entries of hat matrix and pred. errors
tapply(FUN=mean,X=sqErrorRR,INDEX=indexH)

plot(sqErrorRR~indexH,ylab='Sq.Error',xlab='Index',col=2,cex=.5)
```

NOTE 1. While CV is commonly used in statistics and computer science, one needs to be aware that CV is not always an appropriate validation design. For instance, as previously mentioned, in breeding applications the prediction problem usually consists of inferring genetic values of candidates to selection. This prediction problem involves a generational order that is not considered in a standard CV with random assignment of individuals to folds. This may or may not induce biases, but one needs to be aware that CV is not the solution to any validation problem.

NOTE 2. The observed the variability in PMSE and R-squared across partitions of the CV reflects uncertainty associated to the sampling of TRN and TST sets. Evaluating such uncertainty is very important, especially when the number of records in the TRN and/or TST set is small. Note however, that ideally we would like to hold the training data fixed and evaluate the uncertainty associated to sampling of un-observed data (i.e., TST) only.

NOTE 3. We also observed that sq.-error diminishes as 'local sample size', measured, for example using the entries of the hat matrix, increases.

Replicated Training-Testing

In CV the number of folds affects the size of the training and testing datasets and the number of replicates of estimates of prediction accuracy. For instance, in a 5-fold CV the size of the TRN (TST) datasets is 80% (20%) of that of the available data and we only obtain 5 estimates of prediction accuracy (one per fold), this is a very small number if we wish to construct a confidence interval on estimates of prediction accuracy. An alternative is to replicate TRN-TST experiments a large number of times, each time re-assigning at random subjects into TRN and TST samples. The following R-code illustrates this with 30 replicates (example in next page).

Example 3: Replicated TRN-TST partitions

```
rm(list=ls())
##### DATA ################################################
 library(BLR)
 data(wheat)
 N<-nrow(X) ; p<-ncol(X)
 for(i in 1:ncol(X)){ X[,i]<-(X[,i]-mean(X[,i]))/sd(X[,i]) }
 y<-Y[,2]
 nTst<-150
 nRep<-30
 set.seed(1235)
 COR<-matrix(nrow=nRep,ncol=3,NA)
 colnames(COR)<-c('lambda=10', 'lambda=1279', 'lambda=5000')
 lambda<-c(10,1279,10000)

 for(i in 1:nRep){
     print(paste('TRN-TST Replicate ',i,sep=''))
     tst<-sample(1:N,size=nTst,replace=FALSE)
     XTRN<-X[-tst,]
     yTRN<-y[-tst]
     XTST<-X[tst,]
     yTST<-y[tst]
     ZTRN<-cbind(1,XTRN)
     ZTST<-cbind(1,XTST)
     rhs<-crossprod(ZTRN,yTRN)
     C0<-crossprod(ZTRN)
      for(j in 1:3){
       C<-C0
       for(k in 2:ncol(C)){ C[k,k]<-C[k,k]+lambda[j] }
       CInv<-chol2inv(chol(C))
       sol<- CInv%*%rhs
       yHatTST<- ZTST%*%sol
       COR[i,j]<-cor(yTST,yHatTST)
     }
 }
## Plots in next page
### PLOTS (Results from previous page)
## One way of looking at the problem (not quite correct)
  x<-rep(lambda,nRep)
  boxplot(as.vector(COR)~x,xlab=expression(paste(lambda)),
          ylab='Correlation')

## A better way
  plot(y=COR[,2],x=COR[,1],xlim=range(COR),ylim=range(COR),
       xlab=expression(paste(lambda[10])),
       ylab=expression(paste(lambda[1279])),main='Correlation',col=2)
  abline(a=0,b=1,col=4)

  plot(y=COR[,3],x=COR[,2],xlim=range(COR),ylim=range(COR),
     xlab=expression(paste(lambda[1279])),
     ylab=expression(paste(lambda[10000])),main='Correlation',col=2)
  abline(a=0,b=1,col=4)
```

6.3. Between sub-population prediction

So far we have assigned lines from training and testing completely at random. In this example we explore the impacts of training and validating in different subpopulations.

Example 3: Across sub-population prediction

```
 rm(list=ls())
 ##### DATA ################################################
  library(BLR)
  data(wheat) ;
  for(i in 1:ncol(X)){ X[,i]<-(X[,i]-mean(X[,i]))/sd(X[,i])}

## Clustering based on q principal components
  q<-2 # number of PCs used for clustering
  for(i in 1:ncol(X)){X[,i]<-X[,i]-mean(X[,i])}
  SVD<-svd(X,nu=q,nv=0)
  myClusters<-kmeans(x=SVD$u%*%diag(SVD$d[1:q]),centers=2)

## Ploting principal components
  tmp<-which(myClusters$cluster==1)
  plot(x=SVD$u[tmp,1],y=SVD$u[tmp,2], ylim=range(SVD$u[,2]),
       xlim=range(SVD$u[,1]), col=2, xlab='1st PC', ylab='2nd PC'  )
  points(x=SVD$u[-tmp,1],y=SVD$u[-tmp,2],col=4)

## Fitting models
  prior=list(varE=list(df=5,S=1),
             lambda=list(type='random',value=20,rate=1e-5,shape=.53))

  group1<-myClusters$cluster==1
  y<-Y[,4]
  yNA1<-y
  yNA1[which(group1)]<-NA
  yNA2<-y
  yNA2[which(!group1)]<-NA

## Training in sub-population 1
  fm1<-BLR(y=yNA1,XL=X,nIter=7000,burnIn=2000,prior=prior,saveAt='1_')

  # training in sub-population 2
  fm2<-BLR(y=yNA2,XL=X,nIter=7000,burnIn=2000,prior=prior,saveAt='2_')

## Across group prediction
   cor(X[which(group1),]%*%fm1$bL,y[which(group1)])
   cor(X[which(!group1),]%*%fm2$bL,y[which(!group1)])

## Estimates of marker effects
plot(fm1$bL~fm2$bL,col=2)
```

6.4. Across environment prediction using single-trait models

In this example we address the problem of across environment (or trait prediction), this appear, for example when we want to select individuals based on expected performance in an environment in which these genotypes have not been evaluated. Most of the models we have discussed so far can be extended to accommodate multiple traits. Here, we explore the problem of prediction across correlated environments using single-trait models alone or combined using an ad-hoc procedure. A fully multi-environment evaluation of genome-enabled prediction methods for this dataset is presented in Burgueño, de los Campos, and Crossa (2012).

Example 4: Across environment prediction

```
rm(list=ls())
##### DATA ###############################################
  library(BLR)
  data(wheat)
  for(i in 1:ncol(X)){ X[,i]<-(X[,i]-mean(X[,i]))/sd(X[,i])}
 round(cor(Y),3) #

  prior=list(varE=list(df=5,S=1),
             lambda=list(type='random',value=20,rate=1e-5,shape=.53))

## Training models in environments 1-4
  fm<-list()
  for(i in 1:4){
     fm[[i]]<-BLR(y=Y[,i],XL=X,nIter=7000,burnIn=2000,
               prior=prior,saveAt=paste('E_',i,sep=''))

  }

## 1st strategy
  COR<-matrix(nrow=4,ncol=4,NA)
  colnames(COR)<-paste('TRN_',1:4,sep='')
  rownames(COR)<-paste('TST_',1:4,sep='')
  for(i in 1:4){
      for(j in 1:4){
          if(i!=j){ COR[i,j]<-cor(Y[,i],fm[[j]]$yHat)         }
      }
  }
## 2nd strategy (a bit of cheating)
  covP<-cov(Y)
  W<-matrix(ncol=4,nrow=4,0)
  wCor<-rep(NA,4)
  for(i in 1:4){
      W[i,-i]<-covP[i,-i]%*%solve(covP[-i,-i])
      TMP<-cbind(fm[[1]]$yHat,fm[[2]]$yHat,fm[[3]]$yHat,fm[[4]]$yHat)
      wCor[i]<-cor(Y[,i],TMP%*%W[i,])
 }
## compare COR & wCor
```

References

Burgueño, J., G. de Los Campos, and J. Crossa. “Genomic Prediction of Breeding Values When Modeling Genotype × Environment Interaction Using Pedigree and Dense Molecular Markers.” *Crop Science* In Press. doi:doi: 10.2135/cropsci2011.06.0299.

Hastie, Trevor, Robert Tibshirani, and Jerome Friedman. 2009. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Second Edition*. 2nd ed. 2009. Corr. 3rd printing 5th Printing. Springer.