Statistical Programming, Unassessed Practical

Marco Scutari

November 2, 2018

Exercise 1: Parallel Computing

Plant and animal breeders have effectively used phenotypic selection to increase the mean performance in agricultural crops, trees and cattle animals. In the last decade genomic selection has emerged as a better alternative: using the genome to predict phenotypes without having to wait to observe them and thus improve the speed and precision of selection. The data we will use below is a subset of a loblolly pine population described here:

`http://www.genetics.org/content/190/4/1503.short`

The paper explores the performance different Bayesian regression models for genomic prediction; loblolly pine (*pinus taeda*) makes up a large proportions of the forests in North America and is used to produce timber, so it is interesting from both an economic and environmental perspective.

1. Load the data from the file `prepd-loblolly.txt.xz` (859 rows, 2000 columns) into a variable called `loblolly`. Make sure all columns are stored as `numeric` variables.

2. We would like to fit a ridge regression model on `T` using all other variables as explanatory variables. Ridge regression is a penalised linear regression model that minimises

$$\operatorname*{argmin}_{\boldsymbol{\beta}} \left\{ (\mathbf{y} - \mathbf{X}\boldsymbol{\beta})^T(\mathbf{y} - \mathbf{X}\boldsymbol{\beta}) + \lambda_2 \sum_{i=0}^{p} \beta_i^2 \right\} \qquad \lambda_2 \geqslant 0, \tag{1}$$

   so that the estimator for the regression coefficients becomes

$$\hat{\boldsymbol{\beta}} = (\mathbf{X}^T\mathbf{X} + \lambda_2 \mathbf{I}_p)^{-1}\mathbf{X}^T\mathbf{y}. \tag{2}$$

   Install and load the R package **penalized**. Look up the documentation of the function `penalized()` and use it to fit a ridge regression model with $\lambda_2 = 60$. Save the model in variable called `ridge`.

3. Extract the fitted values from `ridge`, and plot them against the corresponding observed values of T using the **lattice** package; if the model fits the data well, the points should cluster around the diagonal of the first quadrant. Add a red, dashed line on top of the points to represent it in the plot. (*Hint: use the* `panel` *option and* `panel.abline()`.) Also, add a grey, dashed line representing the regression line of the observed values on the fitted values.

4. Now extract the residuals from `ridge` and plot the residuals against the fitted values, adding a horizontal, grey dashed line at zero as a reference. Also add a red dashed curve representing the the running mean of the residuals. (*Hint: use* `panel.loess()` *from the* **latticeExtra** *package.*

5. Now we will apply 10-fold cross-validation to obtain a measure of how well the data predicts new observations. 10-fold cross-validation works as follows:

   (a) Split the data into 10 subsets (called "folds") of the same size (or as close as possible); you can use the `split()` function to do that.

   (b) For each fold in turn:

      i. take that fold as the *test set*;
      ii. take the rest of the data as the *training set*;
      iii. use fit a ridge regression model on the training set;
      iv. predict the response for the observations in the test set;

   (c) Collect the pairs of (observed, predicted) values for all the observations.

   (d) Compute the correlation between the (observed, predicted) pairs; this quantity is called *predictive correlation.*

   Implement this algorithm in a function called `xval()`.

6. Benchmark the running time of `xval()`.

7. Reimplement `xval()` using parallel computing and the **parallel** package. (*Hint: it may be easier to rewrite* `xval()` *to use* `lapply()` *first.*) Call the new function `parallel.xval()`. Is this algorithm embarrassingly parallel?

8. Benchmark the running time of `parallel.xval()` with 2 slaves and 4 slaves, and compute the overhead for both cases.

9. Consider the predictive correlation for 10-fold cross-validation as a function of the λ_2 parameter, and write a function that builds on `parallel.xval()` as selects the optimal value for λ_2 as follows:

   (a) start with $\lambda_2 = 2000$;

   (b) compute the predictive correlation for this value of lambda;

   (c) repeat as long as predictive correlation increases:

      i. compute $\lambda_2^* = \lambda_2/2$;
      ii. compute the predictive correlation for λ_2^*;
      iii. if it is larger than the predictive correlation for λ_2, set $\lambda_2 = \lambda_2^*$; otherwise stop.

   The function should return a numeric vector containing the optimal value of λ_2 and the corresponding predictive correlation.

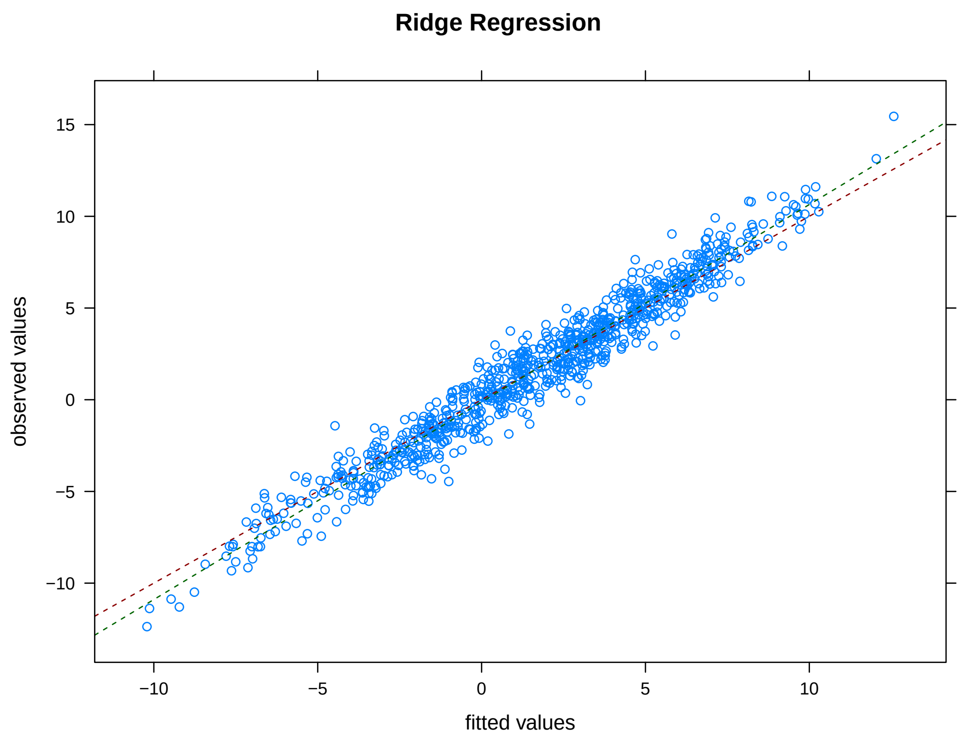
10. Is using `parallel.xval()` the best way to implement this new function? Investigate with `snow.time()` and rewrite the function as needed.

Solution to Exercise 1

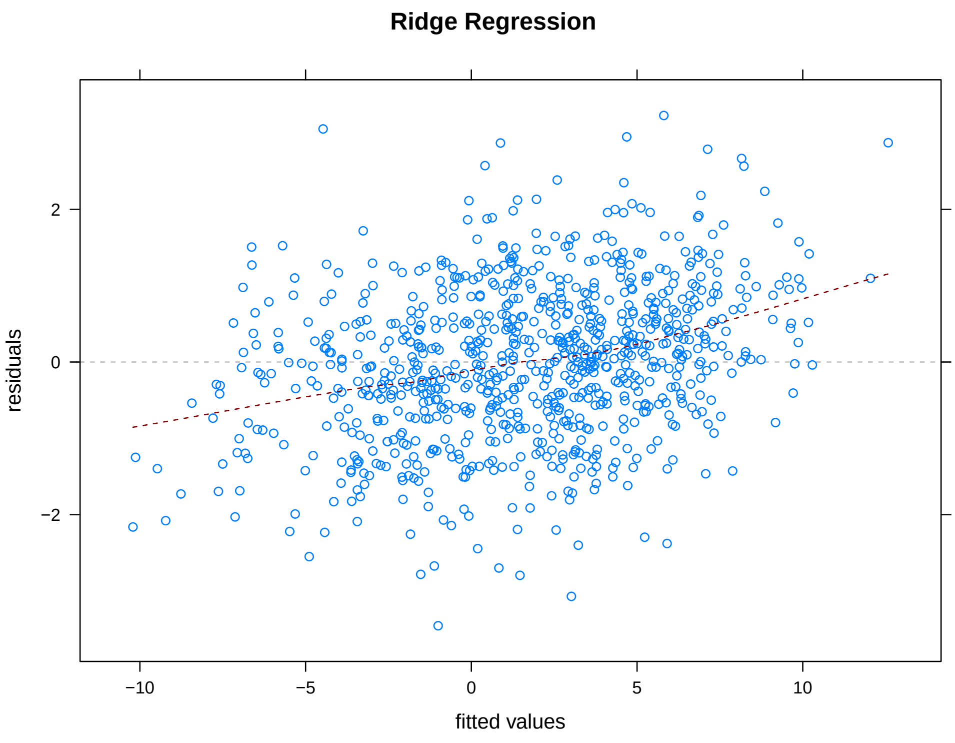
1.
   ```
   loblolly =
     read.table("prepd-loblolly.txt.xz", header = TRUE, colClasses = "numeric")
   dim(loblolly)

   ## [1]  859 2000
   ```

2.
   ```
   library(penalized)
   ridge = penalized(T, penalized = ~ . - T, data = loblolly, lambda2 = 60,
               trace = FALSE)
   ridge

   ## Penalized linear regression object
   ## 2000 regression coefficients
   ##
   ## Loglikelihood =  -1226
   ## L2 penalty =  524  at lambda2 =  60
   ```

3.
   ```
   library(lattice)
   xyplot(loblolly$T ~ fitted(ridge), main = "Ridge Regression",
     xlab = "fitted values", ylab = "observed values",
     panel = function(...) {

       panel.xyplot(...)
       panel.abline(0, 1, col = "darkred", lty = 2)
       panel.lmline(..., col = "darkgreen", lty = 2)

     })
   ```



4.
```
library(latticeExtra)
xyplot(residuals(ridge) ~ fitted(ridge), main = "Ridge Regression",
  xlab = "fitted values", ylab = "residuals",
  panel = function(...) {

    panel.abline(h = 0, col = "grey", lty = 2)
    panel.xyplot(...)
    panel.loess(..., col = "darkred", lty = 2)

  })
```



5.
```
xval = function(data, k = 10, lambda2) {

  n = nrow(data)

  folds = split(sample(n), seq_len(k))

  xval.fold = function(fold) {

    dtrain = data[-fold, ]
    dtest = data[fold, ]

    ridge.fold = penalized(T, penalized = ~ . - T, data = dtrain,
                   lambda2 = lambda2, trace = FALSE)

    pred = predict(ridge.fold, data = dtest)

    return(data.frame(PRED = pred[, "mu"], OBS = dtest$T))

  }#XVAL.FOLD

  pairs = lapply(folds, xval.fold)
  return(do.call("rbind", pairs))

}#XVAL

pairs = xval(loblolly, lambda2 = 60)
cor(pairs)

##       PRED   OBS
## PRED 1.000 0.752
## OBS  0.752 1.000
```

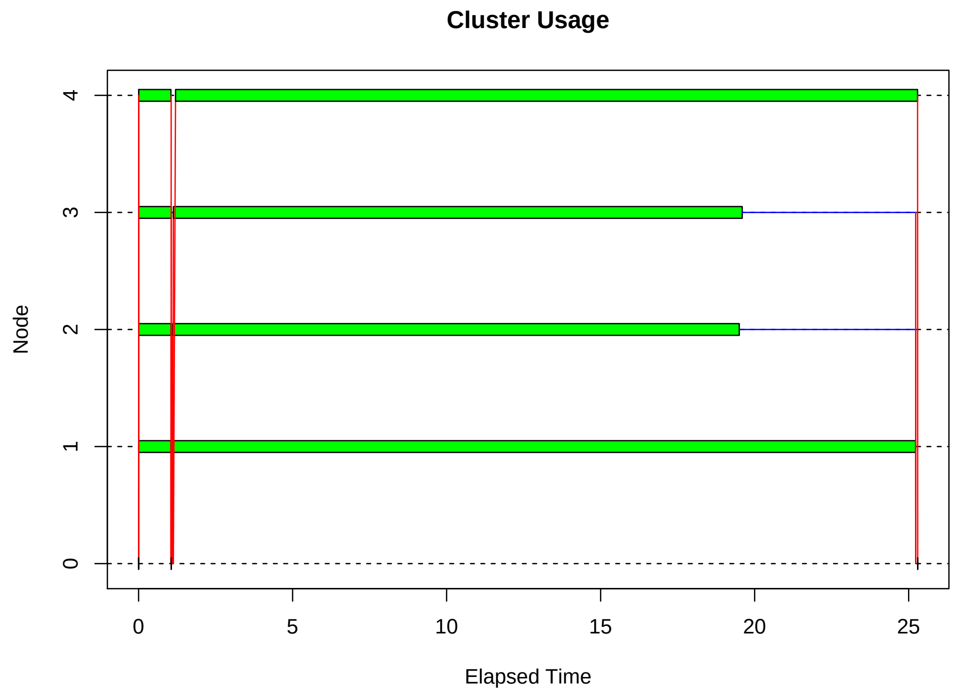
6.
```
system.time(for (i in 1:5) xval(loblolly, lambda2 = 60)) / 5

##    user  system elapsed
## 49.1988  0.0344 49.2336
```

7.
```
parallel.xval = function(data, cluster, k = 10, lambda2) {

  n = nrow(data)

  folds = split(sample(n), seq_len(k))

  xval.fold = function(fold, lambda2) {

    dtrain = data[-fold, ]
    dtest = data[fold, ]

    ridge.fold = penalized(T, penalized = ~ . - T, data = dtrain,
                   lambda2 = lambda2, trace = FALSE)

    pred = predict(ridge.fold, data = dtest)

    return(data.frame(PRED = pred[, "mu"], OBS = dtest$T))

  }#XVAL.FOLD

  clusterExport(cluster, list("data"))
  clusterEvalQ(cluster, library(penalized))

  pairs = parLapply(cluster, folds, xval.fold, lambda2 = lambda2)
  return(do.call("rbind", pairs))

}#XVAL

library(parallel)

cl = makeCluster(2)
pairs = parallel.xval(loblolly, cluster = cl, lambda2 = 60)
stopCluster(cl)

cor(pairs)


##       PRED   OBS
## PRED 1.000 0.746
## OBS  0.746 1.000
```

8.
```
library(parallel)

cl = makeCluster(2)
system.time(for (i in 1:5)
  parallel.xval(loblolly, cluster = cl, lambda2 = 60)) / 5

##    user  system elapsed
##  0.0430  0.0104 28.5324

stopCluster(cl)

cl = makeCluster(4)
system.time(for (i in 1:5)
  parallel.xval(loblolly, cluster = cl, lambda2 = 60)) / 5

##    user  system elapsed
##  0.0826  0.0230 23.8828

stopCluster(cl)
```

The ideal running time would be the `elapsed` time from the sequential `xval()` divided by the number of slaves, and the overhead will be the difference between the `elapsed` time from `parallel.xval()` and the ideal time. It will vary depending on the computer you are running the simulation on.

```
library(snow)
cl = snow::makeCluster(4)
plot(snow.time(
  parallel.xval(loblolly, cluster = cl, lambda2 = 60)))
```



```
snow::stopCluster(cl)
```

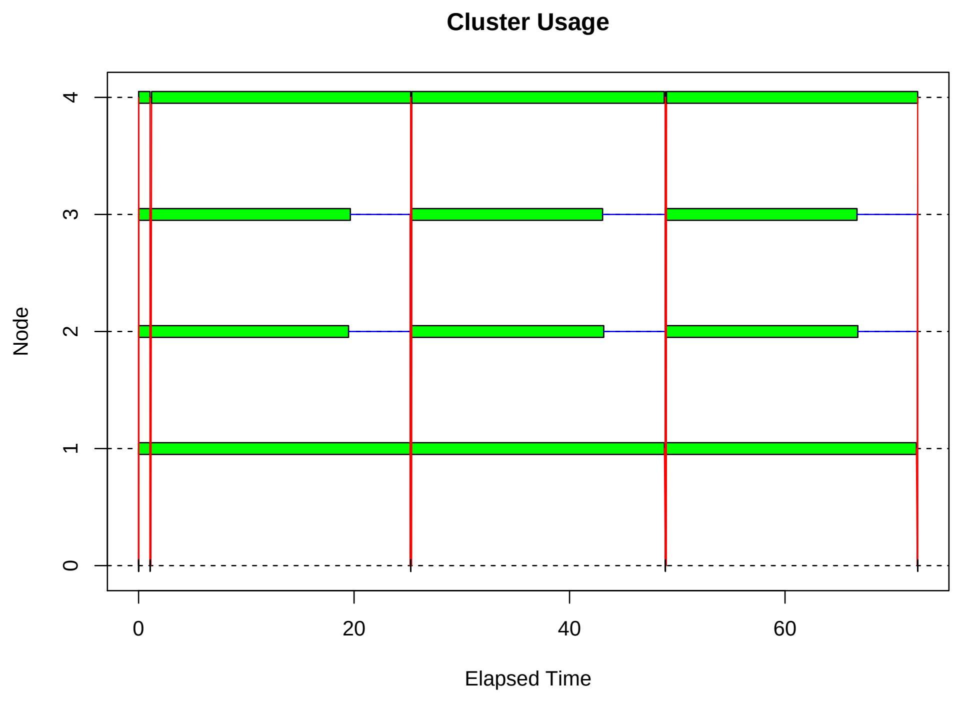
Working with 4 slaves, two take more time than the others because there are 10 to process, $4 + 4 + 2$.

9.
   ```
   library(parallel)

   tune.ridge = function(data, cluster, k, start.lambda2) {

     cur.lambda2 = start.lambda2
     pairs = parallel.xval(loblolly, cluster = cluster, k = k,
               lambda2 = cur.lambda2)

     cur.predcor = cor(pairs$PRED, pairs$OBS)

     repeat {

       new.lambda2 = cur.lambda2 / 2
       pairs = parallel.xval(data, cluster = cluster, k = k,
                 lambda2 = new.lambda2)
       new.predcor = cor(pairs$PRED, pairs$OBS)

       if (new.predcor >= cur.predcor) {

         cur.lambda2 = new.lambda2
         cur.predcor = new.predcor

       }#THEN
       else {

         break

       }#ELSE

     }#REPEAT

     return(c(cur.predcor = cur.predcor, cur.lambda2 = cur.lambda2))

   }#TUNE.RIDGE

   cl = makeCluster(4)
   tune.ridge(loblolly, cluster = cl, k = 10, start.lambda2 = 2000)

   ## cur.predcor cur.lambda2
   ##       0.788    1000.000

   stopCluster(cl)
   ```

10. Using `parallel.xval()` is not optimal because it copies the data to the slave processes over and over, which adds to the overhead. Still, the plot generated by `snow.time()` looks nice.

Two possible improvements are exporting the data to the slaves just once at the beginning of `tune.ridge()`; and to use either 2 or 5 slaves.

```r
library(snow)

cl = snow::makeCluster(4)
plot(snow.time(
  tune.ridge(loblolly, cluster = cl, k = 10, start.lambda2 = 2000)))
```



```r
snow::stopCluster(cl)
```