MSc Long Unassessed Practical 1

Genomics Prediction

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.

Here, we will use the same dataset to test simpler models for genomic prediction. We will use ridge regression and Random Forests, using cross-validation to estimate the accuracy of each method.

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

2. We would like start by fitting a ridge regression model on the outcome variable `T` using all other variables as explanatory variables. Ridge regression is a penalised linear regression model that minimises

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

   so that the estimator for the regression coefficients becomes

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

   We will use the R package **penalized** to fit penalized ridge regression models. **penalized** uses syntax similar to a regular linear regression using **lm**. First, install and load the R package **penalized**. Then, 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. Now we want to take a look at how good the fit is. Plot the fitted values against the observed values. Use `fitted(ridge)` to extract the fitted values. If the model fits well, the points should cluster around the diagonal. Make sure to label your axes, use a colour blind palette, and otherwise adjust plotting parameters to make ensure your plot is of high quality. Additionally, add a line with slope 1 and intercept 0, so you can compare your results to the ideal fit. Finally, compute and add to your plot using `text()` the squared correlation between the fitted and observed values. Once you are happy with your plot, wrap it inside a function. We'll use it again later.

4. Residuals are defined as the difference between the observed and fitted values. Generate residuals from `ridge`. Make a new plot, again using a function, to plot the residuals, again as a high quality plot. Add a horizontal line as the ideal. Is there any structure to the

residuals? Finally make and call a function using `par(mfrow = c(1, 2))` and your two plot functions to jointly plot in the left panel the fitted versus observed values, and in the right panel the residuals.

5. Building a model, and testing it on the same data, is a biased process. To obtain an unbiased estimate of model performance, we will use cross-validation (CV). The general approach for evaluating the performance of a predictive model using CV is as follows.

   (a) Split the data into subsets (called folds) of equal size (or as close as possible)

   (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. fit a model on the training set (here the ridge regression model)

      iv. predict the response for the observations in the test set using the fitted model

   Write a function that performs 10 fold CV for ridge regression on the `loblolly` dataset. Your function should be well named, and take as arguments the dataset, the number of folds of CV you are performing, and the parameter for λ_2. Within your main function, make calls to another function that performs the steps listed above for an arbitrary fold. *You can use* `split()` *to perform the split into folds.*

6. Use the code you just wrote to perform 10 fold CV using ridge regression on the loblolly dataset. Using the plotting code you wrote earlier, plot the fit between the observed and fitted values, as well as the residuals. How does the r^2 compare to earlier, where you fit the model and tested the fit on the same data? Does this make sense?

7. Now, let's try to make this faster using parallel computing. Re-write your code, using sockets (*i.e.* not using `mclapply`), to run parallel computing, where you parallelize fitting on each fold. You should only need to modify the master function from before, *i.e.* not the one that performs the per-fold fitting.

8. Time how long it takes you to run using 1 and 2 cores. What is the overhead in this case?

9. Now, we want to try a different modelling approach, Random Forests. You can read more about the algorithm approach to how Random Forests works elsewhere, for example Wikipedia, or in any of Leo Bremain's many papers. Install and load the `randomForest` package. Modify and re-name the code you wrote earlier to perform ridge regression for a single fold, so that it can handle a user specified choice of model between ridge regression and Random Forests. Similarly, modify the code you wrote to perform model fitting for ridge regression over all folds, to perform model fitting for an arbitrary model. To handle Random Forests, write your code to allow you to specify the number of trees `ntree`, with a default value of 20.

10. Fit both Random Forests and ridge regression using your new code, using as many cores as you want. Which one gives the better fit, as judged by r^2?

11. *Optional.* Try to optimize over hyper-parameters as well, in an unbiased way. For ridge regression, try to optimize over λ_2. For Random Forests, consider optimizing over `maxnodes`, `nodesize`, and other relevant variables