FIT2086 Studio 8
Model Selection and Penalized Regression

Daniel F. Schmidt

September 11, 2017

Contents

1 Introduction

Studio 8 covers the topics of model selection, and in particular, the use of penalized regression for fitting regression models. To complete this Studio, you will need to install the `glmnet` package in R; remember, you can do this by typing `install.packages("glmnet")`, and can then load the package once installed using `library(glmnet)`.

During your Studio session, your demonstrator will go through the answers with you, both on the board and on the projector as appropriate. Any questions you do not complete during the session should be completed out of class before the next Studio. Complete solutions will be released on the Friday after your Studio.

2 The Variance-Bias Trade-off

A key idea underlying model selection is that we can reduce the expected squared-error of an estimator by reducing variance at the expense of increasing bias. This is the way that model selection methods like stepwise regression work: by setting a coefficient to exactly zero, they set the variance to zero at the risk of excluding an important predictor (and thus introducing bias). This is also the principle underlying methods like ridge regression, which trade bias and variance off more smoothly. We will now look at a very simple "shrinkage" estimator to get an idea of how this works. Consider n samples $Y_1, \ldots, Y_N$ from a normal population with unknown mean μ and variance σ^2. It is usual to estimate the mean using the maximum likelihood estimator

$$\hat{\mu} \equiv \bar{Y} = \frac{1}{n} \sum_{i=1}^{n} y_i$$

which is equivalent to the sample mean. Remembering that $\mathbb{E}[\hat{\mu}] = \mu$, we know the bias and variance of this estimator are (see Lectures 3 and 4)

$$\begin{aligned} \text{bias}(\hat{\mu}) &= \mathbb{E}[\hat{\mu}] - \mu = 0 \\ \text{var}(\hat{\mu}) &= \mathbb{V}[\hat{\mu}] = \frac{\sigma^2}{n}. \end{aligned}$$

From this, we know that the mean-squared error (MSE) of the estimator is

$$\begin{aligned} \text{MSE}(\hat{\mu}) &= \text{bias}^2(\hat{\mu}) + \text{var}(\hat{\mu}) \\ &= \frac{\sigma^2}{n}. \end{aligned}$$

Remember this mean-squared error is the average error in estimating μ using $\hat{\mu}$, with the average taken over all the possible samples we could potentially see from your population. We note that as $\text{MSE}(\hat{\mu}) \to 0$ as $n \to 0$, this estimator is consistent (gets closer and closer to the truth as the sample size increases.

1. Consider instead the following estimator of μ:

$$\begin{aligned} \hat{\mu}(c) &= \left(\frac{1}{n+c}\right)\left(\frac{1}{n}\sum_{i=1}^{n} y_i\right) \\ &= \left(\frac{n}{n+c}\right)\hat{\mu} \end{aligned}$$

where $c > 0$. This estimator takes the sample mean/maximum likelihood estimator of μ, $\hat{\mu}$, and multiples it by the quantity $0 < (n/(n+c)) < 1$. This is called a shrinkage estimator, as it takes the maximum likelihood estimator and shrinks it towards $\mu = 0$.

(a) Find the bias of $\hat{\mu}(c)$.

(b) Find the variance of $\hat{\mu}(c)$.

(c) Find the mean-squared error of $\hat{\mu}(c)$.

(hint: use the facts: (i) $\mathbb{E}\left[c\,x\right] = c\,\mathbb{E}\left[X\right]$ *and (ii)* $\mathbb{V}\left[c\,x\right] = c^2\,\mathbb{V}\left[x\right]$*)*

A: To find the bias we use the fact that $\mathbb{E}\left[c\,x\right] = c\,\mathbb{E}\left[x\right]$:

$$\begin{aligned}
\text{bias}\left(\hat{\mu}(c)\right) &= \mathbb{E}\left[\hat{\mu}(c)\right] - \mu \\
&= \mathbb{E}\left[\left(\frac{n}{n+c}\right)\hat{\mu}\right] - \mu \\
&= \left(\frac{n}{n+c}\right)\mathbb{E}\left[\hat{\mu}\right] - \mu \\
&= \left(\frac{n}{n+c}\right)\mu - \mu \\
&= \frac{n\mu}{n+c} - \frac{(n+c)\mu}{n+c} \\
&= \frac{n-(n+c)\mu}{n+c} \\
&= -\left(\frac{c}{n+c}\right)\mu
\end{aligned}$$

We see from this that (i) the estimator is biased downwards, in the sense that it will always underestimate $|\mu|$, and that the level of bias increases with increasing $|\mu|$. This is because we are taking an unbiased estimator of the population mean, $\hat{\mu}$, and scaling it by a quantity less than one (shrinking it towards zero). The larger (in absolute value) the population mean $|\mu|$ is, the greater this bias becomes. As the hyperparameter c increases, the degree of shrinkage increases ($n/(n+c)$ gets smaller), and the bias becomes larger. As $n \to \infty$ (we get more data) the bias goes to zero, regardless of the value of c.

To find the variance we use the fact that $\mathbb{V}\left[c\,x\right] = c^2\,\mathbb{V}\left[x\right]$:

$$\begin{aligned}
\text{var}\left(\hat{\mu}(c)\right) &= \mathbb{V}\left[\hat{\mu}(c)\right] \\
&= \mathbb{V}\left[\left(\frac{n}{n+c}\right)\hat{\mu}\right] \\
&= \left(\frac{n}{n+c}\right)^2\mathbb{V}\left[\hat{\mu}\right] \\
&= \left(\frac{n}{n+c}\right)^2\left(\frac{\sigma^2}{n}\right) \\
&= \frac{n\sigma^2}{(n+c)^2}
\end{aligned}$$

Compared to the variance for the sample mean, σ^2/n, the variance for the shrunken estimator is always smaller, and the larger the degree of shrinkage, the smaller the variance becomes. As

$n \to \infty$ the variance goes to zero, regardless of the value of c. The mean-squared error is

$$\begin{aligned} \mathrm{MSE}\left(\hat{\mu}(c)\right) &= \left(\mathrm{bias}\left(\hat{\mu}(c)\right)\right)^2 + \mathrm{var}\left(\hat{\mu}(c)\right) \\ &= \left(\frac{c}{n+c}\right)^2 \mu^2 + \frac{n\sigma^2}{(n+c)^2}. \end{aligned}$$

Comparing this to the MSE for the sample mean (σ^2/n), we see the MSE for the shrunken estimator depends on the value of the population mean μ. When the population mean μ is close to zero, the shrinkage towards zero improves our mean-squared error because the bias is small (bias depends on the magnitude of μ) but the variance is reduced. This is similar to the "Hitchhiker's estimate" from Studio 3, which always estimated a $\hat{\mu} = 42$ – it had a lot of bias for any $\mu \neq 42$, but zero variance. The shrinkage estimator a less extreme version of this type of estimator; in fact if we set $c = \infty$, then the shrinkage factor is $n/(n+c) = 0$, and we estimate $\hat{\mu}(\infty) = 0$ regardless of the data we observe. However, for any choice of $c < \infty$, our estimator is consistent as $n \to \infty$; for large amounts of data the shrinkage has less and less effect.

2. Using R, plot the mean-squared error over $\mu \in (-5, 5)$, using for $\sigma^2 = 1$, $n = 1$, and $c = 0.1$, $c = 1$, and $c = 5$. Compare these plots to the mean-squared error for $\hat{\mu}$. What do you see? How do the curves change as c increases?

   A: Figure **??** shows the mean-squared errors for the sample mean, and three different versions of the shrinkage estimator against different values of the population mean. We see that for all c, when the population mean is close to zero (the value which the shrinkage estimator is shrinking towards), the mean-squared error of the shrinkage estimators is smaller than the MSE for the sample mean. This is because in this region, the effects of the bias are small compared to the reduction in variance. However, outside the neighbourhood around zero, the MSE of the shrinkage estimators gets worse and worse, because the bias term grows with increasing $|\mu|$. The larger the c, the bigger the improvement near $\mu = 0$ but the worse the estimator becomes outside of the neighbourhood around $\mu = 0$. If we could choose c carefully for our problem however, there is clearly an advantage to be gained if the true population mean is near zero.

   This simple example illustrates the basic ideas behind penalized regression, in which we carefully choose the amount we shrink our parameters based on something like cross-validation. To see how this could lead to big gains, imagine the situation in which we had $p = 100$ predictors, and at the population level, all but two of these were unassociated with our target (i.e., $\beta_j = 0$ for 98 of our predictors). Then, if we shrink our estimates towards zero, for 98 of the predictors we will improve our mean-squared error by reducing the variance, while we will only incur some additional bias for 2 of our predictors, and we will make an overall gain in prediction error. Clearly, careful choice of the shrinkage parameter is crucial, and `glmnet` uses cross-validation to do this in a sensible way by directly using the available data.

3. Is the shrinkage estimator $\hat{\mu}(c)$ consistent?

   A: Yes, see above – the bias and variance (and therefore mean squared error) all go to zero as the sample size $n \to \infty$.

4. **Challenge question**: show that the shrinkage estimator $\hat{\mu}(c)$ is the solution to a ridge-regression

type penalized maximum likelihood estimation equation; i.e., prove that

$$\hat{\mu}(c) = \arg\min_{\mu} \left\{ \frac{1}{2} \sum_{i=1}^{n} (y_i - \mu)^2 + \frac{c\mu^2}{2} \right\}$$

3 Revisiting the Genetic Data, Part I

In this question we will revisit the genetic data that we analysed last week. To complete this question, you will need to install the `glmnet` package in R; remember, you can do this by typing `install.packages("glmnet")`, and can then load the package once installed using `library(glmnet)`. This package contains functions that very efficiently implement ridge, lasso and elastic-net regression, written by the people who invented some of these methods. They can be used for tens of thousands of predictors. You will also need to load the `pROC` package that we used last week into memory.

Before we use penalized regression, we will briefly look at multiple testing and the risk inflation criteria.

1. Load the files `genes.train.csv` and `genes.test.csv` into memory. Fit a logistic regression model to training data using the `glm()` function. Remember, to do this, we use:

   ```
   fullmod=glm(Disease ~ ., data=gene.train, family=binomial)
   ```

   Once the model is fitted, calculate its prediction performance on the testing data using the function `my.pred.stats()` that we used last week. To do this, load this function into memory using

   ```
   source("my.prediction.stats.R")
   ```

   and then call

   ```
   my.pred.stats(predict(fullmod,gene.test,type="response"), gene.test$Disease)
   ```

   Record the classification accuracy and AUC of this model.

2. The `glm()` function produces p-values of association for each of the predictors fitted in the model. Remember, the p-value tells you the probability of seeing an association between a predictor as strong as the one you have observed, or stronger, just by chance, even if there was no association at the population level. The smaller the p-value, the more inclined you should be to include the predictor in the model. A standard test is to accept a predictor if its p-value is less than 0.05.

   You can get access to the coefficients, standard errors and p-values for the predictors included in a model fitted using `glm()` by using the following code:

   ```
   coefficients(summary(fullmod))
   ```

   The fourth column contains the p-values; use

   ```
   pvalues = coefficients(summary(fullmod))[,4]
   ```

   to access these. We can find which of the variables in the model have p-values less than 0.05 using `pvalues < 0.05`, and can count how of these using `sum(pvalues<0.05)`. How many of these p-values are less than 0.05?

3. The multiple hypothesis testing problem (see Lecture 8, Slides 33–36) says that if we test p predictors at a level of α, then even if none of the predictors are associated at a population level, we will accept αp predictors into our model just by chance (false discoveries). For our data, if $\alpha = 0.05$, how many false discoveries might we expect to make just by chance? How does this compare to the number we have observed?

4. To "correct" this problem, we can use the more conservative Bonferroni threshold for accepting a predictor. If we have p predictors, then we can accept all predictors with p-values less than α/p. For $\alpha = 0.05$, how many predictors do we accept using the Bonferroni threshold?

5. The problem with this approach is that when we fit many predictors at once, the effect of each becomes "diluted" and the p-values of association may actually be substantially larger than if we fitted the predictors individually. This means that it becomes difficult to tease apart which predictors are associated and which are not using p-values. An alternative approach is to use stepwise selection with the risk inflation criterion (RIC) method, which was designed for the situation when there are a large number of predictors. As we have $p = 100$ predictors, this seems appropriate. To do this, run

```
fit.ric = step(fullmod,direction="both",k=2*log(ncol(gene.train)-1))
```

The RIC is actually a kind of information criterion implementation of this Bonferroni multiple hypothesis testing idea. Once this model is fitted, inspect the model – you will note that it only contains `SNP56`. Does its p-value pass the Bonferroni correction? Compute the prediction statistics for this model.

4 Revisiting the Pima Indians Data with `glmnet`

As we have seen there are a myriad of different ways we can try and select good predictors using `step()` and p-values. We will now examine a different approach based on penalized regression. In particular, we will look at how to use `glmnet()` to fit logistic regression models to this data using ridge or lasso regression in place of stepwise regression. Begin by ensuring that the `glmnet` library has been installed and loaded. We will also load a couple of extra commands that have been provided to make using `glmnet()` a little easier – by default, it does not support R's formula interface for specifying models, so I have provided some wrappers. Load these using

```
source("wrappers.R")
```

Then load the `pima.train.csv` and `pima.test.csv` data.

1. Let us begin by fitting a model using `glmnet()`. Use the command

   ```
   lasso.fit = glmnet.f(DIABETES ~ ., pima.train, family="binomial", lambda = 0)
   ```

   (note: `glmnet` *requires the* `binomial` *to be in quotes, unlike* `glm()`*).*

   This fits a model using the Lasso (see Lecture 8, slides 58–60), but sets $\lambda = 0$; the `lambda` option tells `glmnet` how much penalisation to use, and setting this to zero tells it to use no penalisation. In this case, the fit is essentially the same as using maximum likelihood to fit the model, i.e., using `glm()`. To calculate the prediction statistics for a model fitted using `glmnet` we use

   ```
   my.pred.stats(predict.glmnet.f(lasso.fit, pima.test, type="response"), pima.test$DIABETES)
   ```

   The prediction stats are essentially the same as the prediction stats we obtained when fitting a logistic regression model to all the Pima predictors using `glm()`.

2. Of course, the case of $\lambda = 0$ (i.e., no penalisation) is not why we want to use `glmnet`. We actually want to use `glmnet` to find estimates when there is penalisation; to try a few different levels of penalisation, we can varying the values of λ, i.e., try

   ```
   lasso.fit = glmnet.f(DIABETES ~ .,  pima.train, family="binomial", lambda = 0.02)
   coefficients(lasso.fit)
   sum(coefficients(lasso.fit) != 0)
   ```

   The first line of code fits the model using `glmnet`, the second line lists the coefficients of the logistic regression model fitted using `glmnet`and the third counts the number of non-zero coefficients in the model (i.e., the number of predictors included in the model). A "`.`" indicates the variable is not included in the model (i.e, has a coefficient of zero). Try the above with `lambda` = 0.02, `lambda` = 0.03 and `lambda` = 0.05. What happens to the model and the number of non-zero coefficients as you increase λ?

3. Instead of varying λ by hand, we can actually ask `glmnet` to try a large number of different values of λ automatically. To do this, just run the command without the `lambda` option:

   ```
   lasso.fit = glmnet.f(DIABETES ~ .,  pima.train, family="binomial")
   ```

   By default, this will try 100 different values of λ that are cleverly chosen by the `glmnet` package. If we run

   ```
   coefficients(lasso.fit)
   ```

we will see the coefficients are now a matrix, with one column for each value of λ `glmnet` has tried. We can visual the "path" the coefficients take, as λ varies, using the nice plotting functionality provided in the package:

```
plot(lasso.fit)
```

The plot shows how the coefficients change as the λ changes.

4. The question is then: which λ should we use? The `glmnet` package has the ability to automatically select a good value for λ, and therefore, a good model, by using cross-validation (see Lecture 8, slides 42–44). Basically, it divides the data into two groups, tries all the different values of on one data group, and sees how well the resulting models predict on the other data group. The value of λ with the smallest cross-validated prediction error is chosen. Run:

   ```
   lasso.fit = cv.glmnet.f(DIABETES ~ ., pima.train, family="binomial")
   plot(lasso.fit)
   ```

   The `plot()` function, when used after using `cv.glmnet.f()`, displays a nice graph showing the cross-validation prediction error as the value of (log) λ changes. The package chooses the value that has the smallest cross-validation error as the best value. You can find the cross-validation error of the best model that was tried using

   ```
   min(lasso.fit$cvm)
   ```

5. To see which coefficients have been chosen using cross-validation and the lasso, run

   ```
   coefficients(lasso.fit)
   ```

   We can also compute the prediction stats using

   ```
   my.pred.stats(predict.glmnet.f(lasso.fit, pima.test, type="response"), pima.test$DIABETES)
   ```

   Write down the regression equation for Diabetes estimated by the lasso.

6. We can compare the Lasso model to the model estimated by using stepwise regression and BIC that we found last week:

   ```
   fit.bic = step(glm(DIABETES ~ .,pima.train,family="binomial"),k=log(668),direction="both")
   summary(fit.bic)
   ```

   How do the two models (BIC stepwise and lasso using CV) compare?

7. Next, we are going to fit some linear models to the data using ridge regression. The `glmnet` package includes ridge regression (see Lecture 8, slides 55–57) as well as lasso regression, as both are actually a special case of the elastic net regression procedure which it also implements (see Lecture 8, slides 61–62). Just like lasso, ridge regression has a parameter λ that controls how strong the penalisation is, and just like lasso, when $\lambda = 0$, there is no penalty and the resulting model is (virtually) the same as the one estimated by `glm()`. Lets try ridge regression for a few different values of λ:

   ```
   ridge.fit = glmnet.f(DIABETES ~ ., pima.train, family="binomial", lambda = 0.05, alpha=0)
   coefficients(ridge.fit)
   ```

   The extra `alpha=0` option tells `glmnet` that we want to use ridge regression. Additionally, try `lambda` = 0.15 and `lambda` = 0.5. How do the coefficients change, and how do they differ from the lasso coefficients?

8. You also use cross-validation to pick a good value of λ for ridge regression:

   ```
   ridge.fit = cv.glmnet.f(DIABETES ~ ., pima.train, family="binomial", alpha=0)
   coefficients(ridge.fit)
   my.pred.stats(predict.glmnet.f(ridge.fit, pima.test, type="response"), pima.test$DIABETES)
   ```

   How does this model compare to the models selected by BIC and lasso?

9. Finally, we can use the Lasso to fit a model when we include all the transformations that we looked at last week (logs of variables, squares of variables, interaction between variables) plus the cubes of all the variables. Look at the code in the provided file `studio8.R`; note that I have provided a function `my.make.formula()` that can be used to quickly create the "formula" statement for a linear/logistic model that includes all interactions, logs, squares and cubes of variables; the function `glmnet.tidy.coef()` in `wrappers.R` can be used to display only the coefficients of a lasso fit that are non-zero, which is useful when the number of potential predictors is large.

   Step through the code for this question; we can learn two things: (i) the lasso can be substantially faster to run than stepwise regression, and (ii) when n is large compared to the number of predictors, the lasso does not necessarily outperform step-wise selection, although it does not perform substantially worse.

10. When there is a lot of data, most methods perform reasonably well. The real strength of the lasso is the fact that it does not degrade dramatically when the sample size is small compared to the number of predictors, which can be the case with methods like stepwise selection. To explore this, we can swap the training and testing data, i.e.,

    ```
    pima.train = read.csv("pima.test.csv")
    pima.test = read.csv("pima.train.csv")
    ```

    Now we have only $n = 100$ data points available for training, and have 668 available for testing. Using this data:

    (a) Fit a model to the new training data using only the variables in the dataset, and calculate the prediction statistics for this model

    (b) Fit a model to the new training data using the variables plus their interactions, logs, squares and cubes, and calculate the prediction statistics for this model.

    How do these models perform?

11. It is clear that as we are fitting $p = 60$ variables (after variable transformations) to $n = 100$ samples, we are very likely to be overfitting. Try using backwards stepwise regression with the BIC to try and prune out unimportant variables. How many variables does it retain? Calculate prediction statistics for this model − has stepwise selection improved our predictions much in this case?

12. Now, we will use `glmnet` to fit models with all transformations using the lasso and ridge with cross-validation.

    (a) Fit a model using the lasso. What is the final model selected by the lasso? (you can examine the retained variables and their estimates using the `glmnet.tidy.coef()` function). Write out the regression equation for this model. Calculate the prediction statistics for this model. How does it perform?

    (b) Fit a model using ridge and cross-validation. Which model is better (ridge or lasso) in terms of prediction onto future data? You can examine which model cross-validation would prefer by comparing the minimum cross-validation errors for the two different models.

5 Additional Questions

You have seen how to fit models and use `glmnet` to fit ridge and lasso regressions. Some further work you can undertake includes:

1. Return to the genetics data we examined above, and use both ridge and lasso to fit models to that data. Compare them to the models selected by BIC and RIC, and calculate their prediction statistics.

2. Revisit the wine quality data from Lecture 6. You can use `glmnet` with continuous predictors by simply omitting the `family="binomial"` option when calling the `glmnet` functions. See if you can use the lasso and the `my.make.formula()` function to try fitting a model using all interactions, logs, squares and cubes of the variables in the dataset. You can swap the training and testing data when doing this so you have more data to learn your model on.