The Lasso and Other Regularization Methods in Regression

Richard L. Smith

October 18, 2021

1 Introduction

The lasso method (*least absolute shrinkage and selection operator*) as we now know it was introduced by Tibshirani [11] through there were precedents for it in both the signal processing [8] and statistical [4, 1] literatures. It is conceptually similar to ridge regression [7]; in particular, both methods result in *shrinkage* of the least squares estimates, with the aim of reducing the mean squared error for both estimation and prediction by introducing some bias into the regression estimators.

We assume a standard normal-theory linear regression model,

$$y_i = \sum_{j=1}^{p} x_{ij}\beta_j + \epsilon_i,\ \epsilon_i \sim N[0,\sigma^2] \text{ (independent) }, i = 1, ..., n, \tag{1}$$

which may also be written in matrix-vector notation as

$$\mathbf{y} = X\boldsymbol{\beta} + \boldsymbol{\epsilon},\ \boldsymbol{\epsilon} \sim N_n[\mathbf{0}, \sigma^2 I_n]. \tag{2}$$

One way to characterize ridge regression is to choose the parameter estimates $\boldsymbol{\beta}$ to solve the optimization problem

$$\text{Minimize } (\mathbf{y} - X\boldsymbol{\beta})^T(\mathbf{y} - X\boldsymbol{\beta}) \text{ such that } \sum_j \beta_j^2 \le t^2 \tag{3}$$

for some specified $t > 0$. To see that (3) leads to the usual form of the ridge regression estimate, first rewrite (3) in Lagrangian form as

$$\text{Minimize } \sum_i (y_i - \sum_j x_{ij}\beta_j)^2 + \lambda \sum_j \beta_j^2 \tag{4}$$

with λ a Lagrange multiplier, then differentiate with respect to $\beta_k,\ k = 1, ..., p$ to derive the normal equations

$$\sum_i x_{ik}(y_i - \sum_j x_{ij}\beta_j) - \lambda\beta_k = 0,\ k = 1, ..., p, \tag{5}$$

which we also write in matrix notation as

$$X^T\mathbf{y} = (X^TX + \lambda I_p)\tilde{\beta}^{(\lambda)}, \tag{6}$$

equivalent to the standard formula for $\tilde{\beta}^{(\lambda)}$. In the typical case when $\lambda > 0$, λ will be chosen to exactly satisfy the constraint, i.e.

$$\mathbf{y}^T X(X^T X + \lambda I_p)^{-2} X^T \mathbf{y} \;\; = \;\; t^2, \tag{7}$$

so (7) defines the relationship between t and λ. The fundamental idea of lasso regression is to replace criterion (3) by:

$$\text{Minimize } (\mathbf{y} - X\boldsymbol{\beta})^T(\mathbf{y} - X\boldsymbol{\beta}) \text{ such that } \sum_j |\beta_j| \leq t. \tag{8}$$

The equivalent "Lagrangian" form in this case is

$$\text{Minimize } \sum_i (y_i - \sum_j x_{ij}\beta_j)^2 + \lambda \sum_j |\beta_j| \tag{9}$$

with some $\lambda > 0$.

Comparing (9) with (4), the fundamental change is to replace the ℓ_2 penalty $\lambda \sum_j \beta_j^2$ with the ℓ_1 penalty $\lambda \sum_j |\beta_j|$.

Although the paper by Tibshirani [11] is usually cited as the source paper for the lasso concept, there were a few precedents. Indeed, the basic formula (9) appears to have been first given by [8] in connection with signal processing problems in seismology, where they argued that this is an improvement on older algorithms in geophysics where both terms in (9) are based on the ℓ_1 norm. However, the generality of this approach for statistical regression problems appears not to have been recognized at the time.

Frank and Friedman [4] made a detailed comparison of ridge regression, PCR and PLS regession with ordinary least squares (OLS) regression (the usual least squares estimator without any reduction of the model) and what they called VSS (variable subset selection) regression, i.e. OLS regression with selection of variables through the techniques we have seen earlier, such as stepwise or best subsets variable selection. They also suggested the idea of *bridge regression*, in which the penalty function $\lambda \sum_j \beta_j^2$ from (4) is replaced by $\lambda \sum_j |\beta_j|^\gamma$ (the ℓ_γ penalty) for some $\gamma > 0$, pointing out that $\gamma = 2$ is ridge regression while the limit $\gamma \to 0$ is equivalent to VSS; they even suggested the possibility of optimizing both λ and γ by cross-validation or some similar technique. The special case $\gamma = 1$ is of course the lasso estimator, but Frank and Friedman did not give any particular emphasis to that case.

Another precedent noted in [11] is the *non-negative garotte* suggested by Breiman [1], in which the OLS regression coefficients $\hat{\beta}_j^{(0)}$ are shrunk by non-negative coefficients c_j to minimize

$$\sum_i (y_i - \sum_j x_{ij} c_j \hat{\beta}_j^{(0)})^2 \text{ subject to } c_j \geq 0, \; \sum_j c_j \leq t \tag{10}$$

which is clearly a similar idea to the lasso.

The bridge regression idea has also been developed further by some authors. Fu [5] proposed a general algorithm for bridge regression with $\gamma \geq 1$ but did not consider the case $\gamma < 1$ because in this case the objective function is non-convex. Much more recent work by Gorban *et al.* [6] proposed a family of optimization algorithms based on piece-wise quadratic error potentials of subquadratic growth (PQSQ potentials) which include general bridge regression as a special case of a much larger class of objective functions.

However, probably the most widely applied generalization of lasso and ridge regression is the *elastic net* idea which combines the two concepts,

$$\text{Minimize } \sum_i (y_i - \sum_j x_{ij}\beta_j)^2 + \lambda_1 \sum_j |\beta_j| + \lambda_2 \sum_j \beta_j^2 \tag{11}$$

which is equivalent, on writing $\alpha = \frac{\lambda_2}{\lambda_1+\lambda_2}$, to solving the constrained minimization problem

$$\text{Minimize } (\mathbf{y} - X\boldsymbol{\beta})^T(\mathbf{y} - X\boldsymbol{\beta}) \text{ such that } (1-\alpha)\sum_j |\beta_j| + \alpha \sum_j \beta_j^2 \leq t \tag{12}$$

for some specified $t > 0$.

This was introduced by Zou and Hastie [12] as a generalization of lasso that retains some of the features of ridge regression. They pointed out that ridge regression often outperforms lasso; that lasso often leads to models with very few non-zero parameter estimates which can be a disadvantage for very high-dimensional problems, especially when $p > n$, and that the elastic net estimator has the potentially desirable property of including or excluding groups of high correlated predictors together.

2 Properties of the Lasso Estimator

The simplest case, and almost the only one for which the estimator can be derived analytically, is when X is orthonormal, i.e. $X^TX = I_p$. Note that another way of writing this is $\sum_{i=1}^n x_{ij}x_{jk} = 1$ if $j = k$ and 0 if $j \neq k$. In this case the OLS estimator is $\hat{\beta}^{(0)} = X^T\mathbf{y}$. We can write the quadratic term in (9) as

$$\begin{aligned}
\sum_i (y_i - \sum_j x_{ij}\beta_j)^2 &= \sum_i (y_i^2 - 2\sum_j y_i x_{ij}\beta_j + \sum_j \sum_k x_{ij}x_{ik}\beta_j\beta_k) \\
&= \sum_i y_i^2 - 2\sum_j \hat{\beta}_j^{(0)}\beta_j + \sum_j \beta_j^2 \\
&= \sum_i y_i^2 - \sum_j (\hat{\beta}_j^{(0)})^2 + \sum_j (\beta_j - \hat{\beta}_j^{(0)})^2
\end{aligned}$$

The optimization problem (9) therefore reduces to a series of optimization problems of the form

$$\text{Minimize } \sum_j (\beta_j - \hat{\beta}_j^{(0)})^2 + \lambda|\beta_j| \tag{13}$$

It is readily checked that the solution of (13) for each j is

$$\hat{\beta}_j = \text{sign}(\hat{\beta}_j^{(0)}) \left(\hat{\beta}_j^{(0)} - \frac{\lambda}{2}\right)_+ \tag{14}$$

where the sign x_+ means the larger of x and 0. This is equivalent to the concept of "soft thresholding" known in the theory of wavelets [2].

To see (14), it is sufficient to note that if $\hat{\beta}_j^{(0)} > 0$ (the case $\hat{\beta}_j^{(0)} < 0$ is exactly symmetrical), the function $(\beta - \hat{\beta}_j^{(0)})^2 + \lambda|\beta|$ is minimized by $\beta = \hat{\beta}_j^{(0)} - \lambda/2$ if $\hat{\beta}_j^{(0)} > \lambda/2$ and 0 otherwise; see Figure 1.

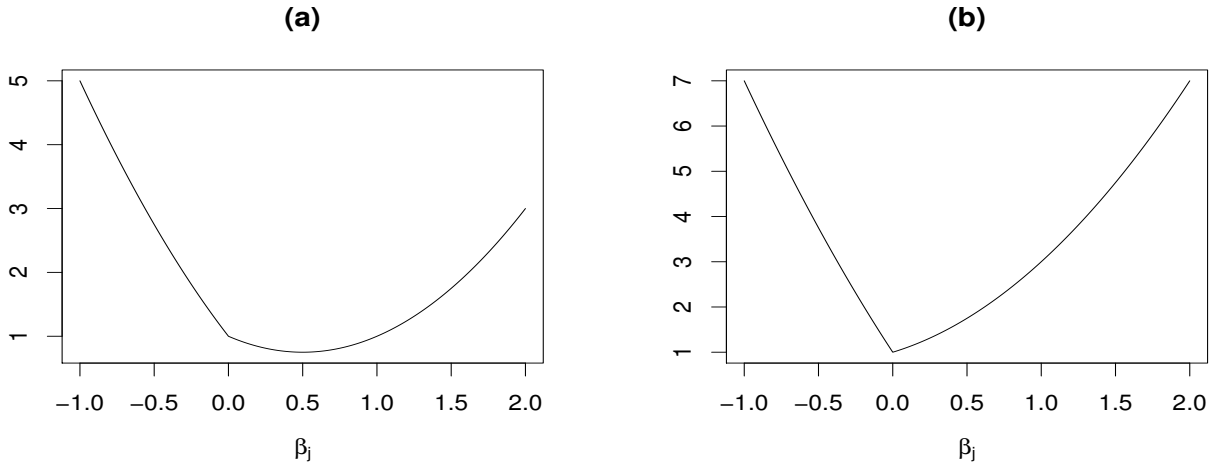

Figure 1: The function $(\beta_j - \hat{\beta}_j^{(0)})^2 + \lambda|\beta_j|$ plotted against β_j. (a) $\lambda = 1,\ \hat{\beta}_j^{(0)} = 1,\ \lambda < 2\hat{\beta}_j^{(0)}$, minimum at $\beta_j = \hat{\beta}_j^{(0)} - \lambda/2$. (b) $\lambda = 3,\ \hat{\beta}_j^{(0)} = 1,\ \lambda > 2\hat{\beta}_j^{(0)}$, minimum at $\beta_j = 0$.

Unfortunately, the non-orthogonal case cannot be derived directly from this result — unlike the case with ridge regression, the penalty function is not invariant to rotations of the X space so we cannot apply simple linear operations to make X orthonormal. Fortunately there are, by now, numerous efficient algorithms for computing the general solution of the optimization (9) and its extension (11).

Naturally, an important question is how to find the optimal value of t or equivalently λ. Tibshirani [11] proposed three ways of doing this. The simplest (conceptually if not computationally) is cross-validation. Specifically, he proposed a five-fold cross-validation carried out at a sequence of values of the normalized parameter $s = t/\sum|\hat{\beta}_j^{(0)}|$, with the value of s (or corresponding λ) that minimizes the cross-validated mean squared prediction error taken as optimal.

The second method uses a generalized cross-validation of the form

$$GCV \quad = \quad \frac{1}{n}\frac{RSS(t)}{(1 - p(t)/n)^2} \tag{15}$$

with $RSS(t)$ the residual sum of squares associated with the lasso estimator at t, and $p(t)$ an approximation to the "effective degrees of freedom". (15) is quicker to compute than direct cross-vaidation but harder to interpret in view of the approximations involved.

The third method proposed in [11] is an application of "Stein's unbiased estimate of risk" [10] but our impression is that this is not much used, so we omit any further discussion here.

3 Computation

A very complete set of R routines is in `glmnet`, documented in detail at https://glmnet.stanford.edu/. This allows for the general elastic net formulation (12) with $\alpha \in [0, 1]$ specified by the user; the

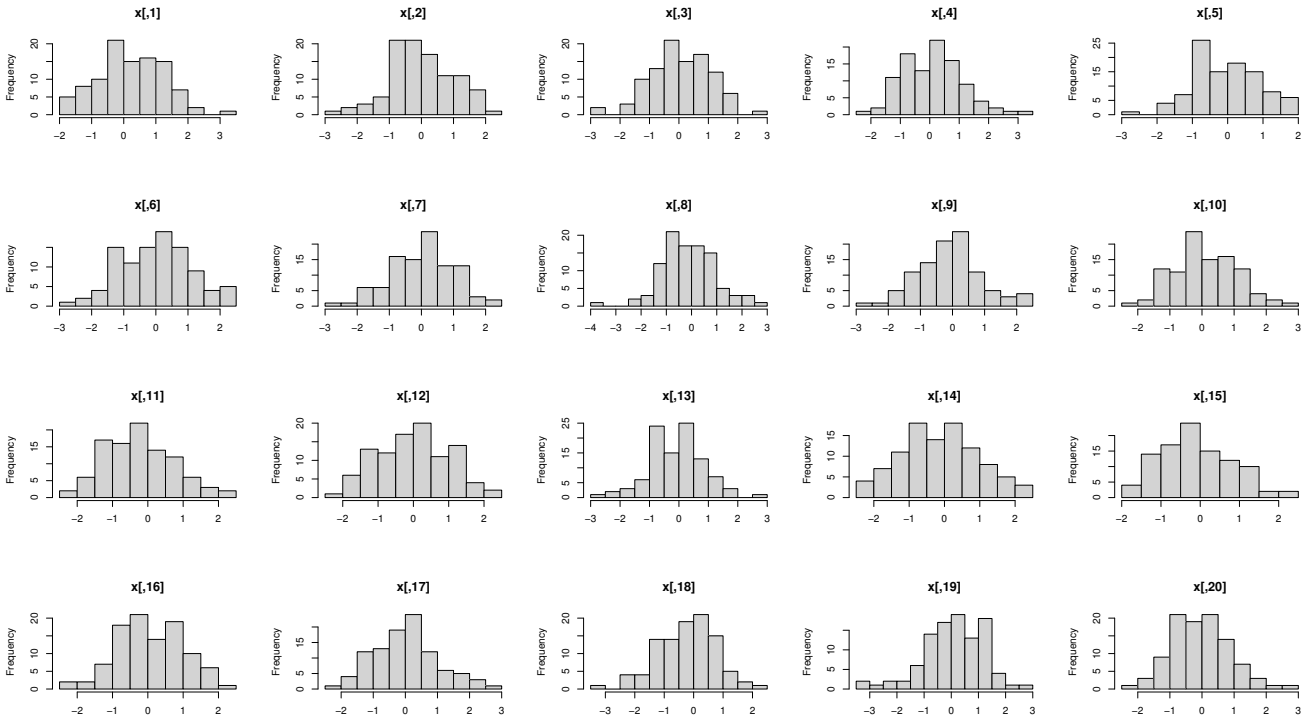

Figure 2: Histograms of the 20 covariates in the test data example.

limiting cases $\alpha = 1$ (the default) and $\alpha = 0$ corresponding to lasso regression and ridge regression respectively. The software also covers generalized linear models such as logistic or Poisson regression.

As an example, we first show results for the test dataset used to illustrate the package (https://glmnet.stanford.edu/). This can be loaded with the R commands

```
library(glmnet)
data(QuickStartExample)
```

This loads the pre-stored test datasets in two objects labelled `x` and `y`. Here `x` is a 100×20 matrix of covariates and `y` is a 100×1 vector of responses. The challenge is to find the optimal linear predictor of `y` from `x`.

Although this dataset appears to have been deliberately constructed to illustrate the workings of `glmnet`, in a typical "real" dataset this will not be the case, so some basic initial examination of the data seems appropriate.

One question is whether there should be a transformation of any of the x variables. In Figure 2, we have shown histograms of each of the 20 potential covariates (i.e. the 20 columns of the `x` matrix), and 3 shows boxplots and a stripchart of the same 20 variables. These show that each of these variables has an approximately normal distributions and there are no obvious outliers; thus, use of untransformed `x` values seems appropriate. We also looked at a histogram of the `y` values and scatterplots of `y` against each`x` column (not shown here); these also show an approximately

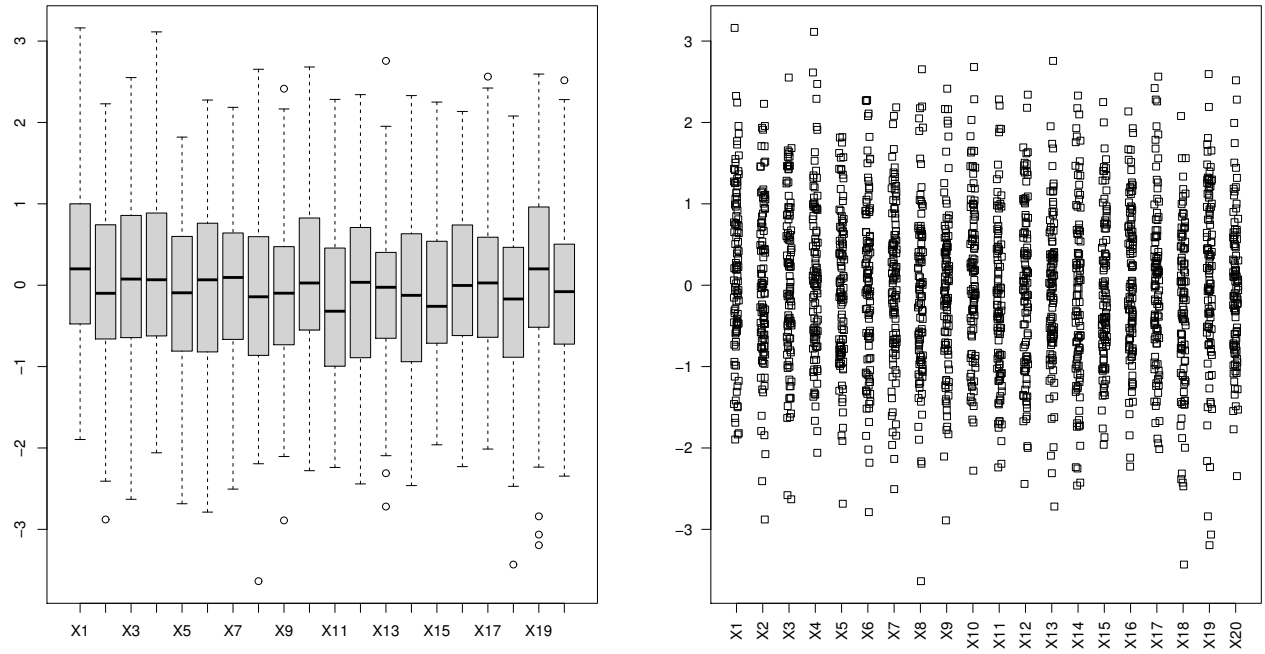

Figure 3: Boxplots and stripchart of the 20 covariates in the test data example.

normal shape for `y` and no obvious nonlinearities in the relationship between `y` and any of the `x` variables (these plots are not shown).

It should be noted that we have not done any scaling of the `x` matrix here. Another option would be to write `x=scale(x)` to scale all the covariates to mean 0 and standard deviation 1; in this example the `x` variables are *not* pre-scaled, but there are also no strong differences of scale (the 20 sample means range from –0.272 to +0.206, the 20 standard deviations from 0.887 to 1.106). In a real dataset, some initial scaling of the covariates should definitely be considered.

Another issue to consider is whether there is *multicollinearity* in the data. Write the eigenvalues of X^TX (in R notation: `t(x)% * % x`) as $\delta_1 \geq \delta_2 \geq \ldots \geq \delta_p$. The *condition number* of X^TX is defined to be

$$\kappa \quad = \quad \sqrt{\frac{\delta_1}{\delta_p}} \tag{16}$$

(see [3], p. 106). The larger κ, the more collinear the data (a value $\kappa > 30$ is considered larger). We can also consider the secondary values $\kappa_i = \sqrt{\delta_1/\delta_i}$ for $i < p$ to indicate possible further collinearities beyond the largest one; these are called *variance inflation factors* or VIFs in [3]; note that [9], Section 5.2.2, give a slightly different definition of VIFs but also define *condition indices* which are essentially the same as the VIFs given here. Whatever precise name is used, these are all measures of the instability of the X^TX matrix and therefore help assess whether shrinkage by ridge regression or lasso is likely to be helpful. Note that in the case of ridge regression, where the

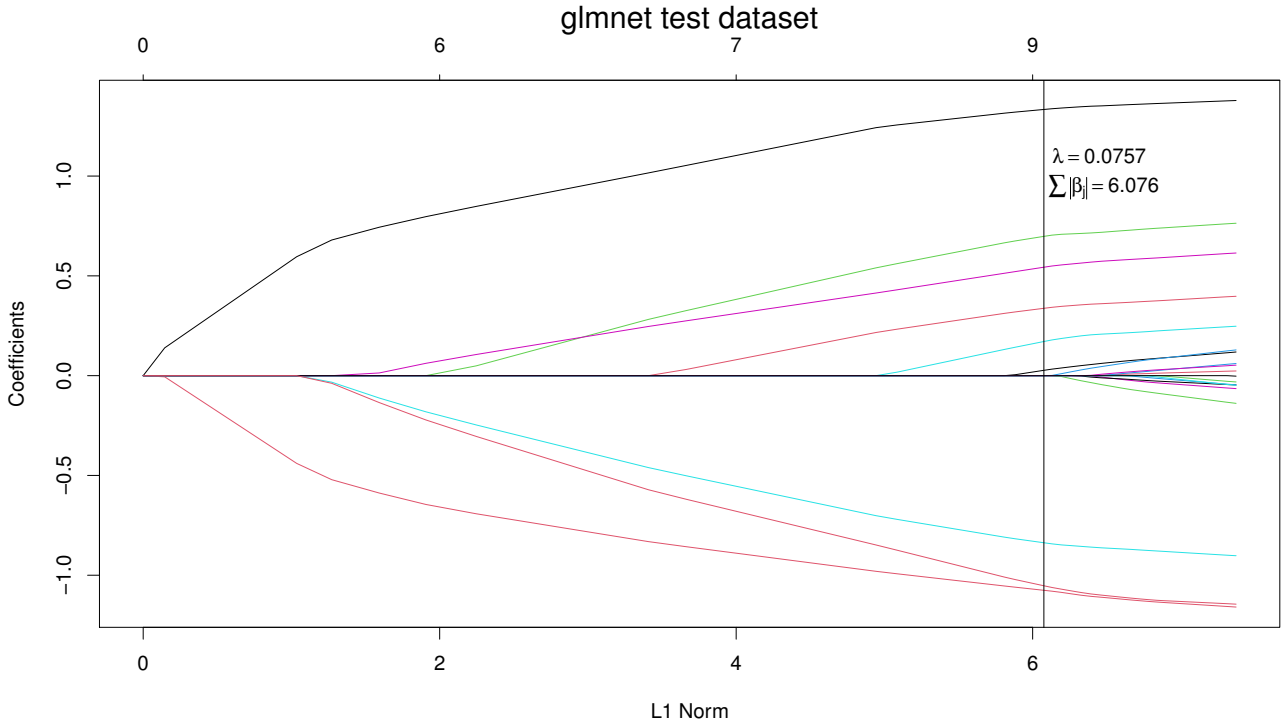

Figure 4: Result of `plot fit` for the test dataset. The vertical bar and accompanying test were added to the main plot and shows the optimal value of λ (0.0757) and $\sum_j |\beta_j|$ (6.076) as determined by leave-one-out cross-validation.

matrix $X^T X$ is replaced by $X^T X + \lambda I_p$ before inverting, the condition number (16) is replaced by $\kappa_\lambda = \sqrt{\delta_1 + \lambda/\delta_p + \lambda}$ which could be much smaller if κ is large. This argument could also be used to give some idea of the appropriate range of λ to consider; at a minimum, it needs to be large enough that κ_λ is substantially smaller than κ.

All these calculations are preliminary to the main analysis with this example; the eigenvalues decrease from 188.3 to 36.8, for $\kappa = 2.3$, which indicates no major issue with multicollinearity, though of course there may still be advantages in applying the lasso or ridge regression methods.

We now turn to `glmnet` itself; a good starting point is the commands

```
fit=glmnet(x,y)
plot(fit)
```

which produces the plot in Figure 4, except for the vertical line near the right hand side, which will be explained later.

The interpretation of the plot is as follows. Reading from left to right, starting from $t = \sum_j |\beta_j| = 0$, we initially have precisely one non-zero β_j. As $t = \sum |\beta_j|$ ("L1 Norm" in the plot) increases, we see that the number of non-zero parameter estimates increases, until at the right hand end of the plot, all 20 parameters are present. These estimates are given in the `fit$beta` matrix (with 20 rows, 67 columns) that correspond to 67 values of λ, given by `fit$lambda`. For example,

the parameters in column 10 are given by

```
> fit$beta[,10]
         V1          V2          V3          V4          V5          V6          V7
 0.84797353  0.00000000  0.04902347  0.00000000 -0.24689473  0.10589979  0.00000000
         V8          V9         V10         V11         V12         V13         V14
 0.00000000  0.00000000  0.00000000  0.00000000  0.00000000  0.00000000 -0.69202504
        V15         V16         V17         V18         V19         V20
 0.00000000  0.00000000  0.00000000  0.00000000  0.00000000 -0.30436207
```

with six non-zero parameter estimates, that corresponds to

```
> fit$lambda[10]
[1] 0.705918
```

One point that should be made about all these model fits: although we have not distinguished between the intercept and the other regression parameters in our theoretical discussion, all the models discussed so far include an intercept (in other words, the intercept is not included in $\sum |\beta_j|$). The vector of 67 intercepts is given by

```
> fit$a0
       s0        s1        s2        s3        s4        s5        s6        s7
0.6607581 0.6312350 0.5874616 0.5475769 0.5112354 0.4781224 0.4457983 0.4134041
...
      s64       s65       s66
0.1112088 0.1110190 0.1108457
```

It makes sense that this is the default option, because otherwise a simple shift of all the y values could radically change the model fit; however, `glmnet` does contain an option to omit the intercept, given by `intercept=F`.

Now let's talk about how to determine the optimal value of λ. One point should be made clear up front: there is no optimal value. It depends, in part, on the loss function used to measure fit, but the default in `glmnet` is `type.measure="deviance"`, which uses mean squared error for Gaussian models, also represented as `type.measure="mse"`. The `cv.glmnet` function *estimates* the optimal λ by cross-validation. A key option here is `nfolds`, given the number of "folds" (i.e. partitions of the data) used for cross-validation. The extreme case here is to set `nfolds` equal to the number of observations in the dataset, which is also known as "leave-one-out cross-validation": for each candidate value of λ, the model fit is repeated with one observation left out at a time, and the resulting prediction errors combined into an overall mean squared error for prediction; the λ with the lowest MSE is declared optimal. It should be noted that the default value of `nfolds` is 10, and the `glmnet` documentation explicitly recommends against setting `nfolds` equation to n, the total sample size, presumably because the computation time would be excessive when n s large. However, taking `nfolds` equal to 10 does have the slightly disconcerting feature that the result of `cv.glmnet` is (slightly) different each time the function is applied, presumably because the function is randomly choosing the 10-fold splits, so here we do use the leave-one-out method which avoids that inconvenience:

```
cvfit=cv.glmnet(x,y,nfolds=100)
cvfit$lambda.min
which(fit$lambda==cvfit$lambda.min)
```

which produces outpout

```
> cvfit=cv.glmnet(x,y,nfolds=100)
Warning message:
Option grouped=FALSE enforced in cv.glmnet, since < 3 observations per fold
> cvfit$lambda.min
[1] 0.07569327
> which(fit$lambda==cvfit$lambda.min)
[1] 34
```

We can find the corresponding regression coefficients (which *includes* the intercept) by typing

```
> coef(cvfit, s = "lambda.min")
21 x 1 sparse Matrix of class "dgCMatrix"
                      1
(Intercept)  0.14867414
V1           1.33377821
V2           .
V3           0.69787701
V4           .
V5          -0.83726751
V6           0.54334327
V7           0.02668633
V8           0.33741131
V9           .
V10          .
V11          0.17105029
V12          .
V13          .
V14         -1.07552680
V15          .
V16          .
V17          .
V18          .
V19          .
V20         -1.05278699
```

and we can find the corresponding L1 norm (*without* the intercept) by

```
> sum(abs(coef(cvfit, s = "lambda.min")[2:21]))
[1] 6.075728
```

This is the t value that determines the vertical straight line in Figure 4. Alternatively, since we have already seen that `which(fit$lambda==cvfit$lambda.min)` produces the answer 34, we could find the same coefficients with the commands `fit$a0[34]` and `fit$beta[,34]`.

One thing that seems to be lost here is that there is nothing corresponding to the standard error of an individual β_j estimate. However, since the whole procedure is predicated on mean squared prediction error, it may be that this is the most meaningful way to evaluate uncertainty. The function `cvfit$cvm` gives estimates of the mean squared prediction error at each evaluated value of λ, and `cvfit$cvsd` is an estimate of the standard deviation of that estimate (presumably, based on the variation in the `nfolds` individual estimates of prediction error). In this case, we note that

```
> cvfit$cvm[34]
     s33
1.021887
> cvfit$cvsd[34]
      s33
0.1269369
```

so at the supposed optimal λ, the estimated MSPE is 1.02 with a standard deviation of 0.13.

Before leaving this example, we want to discuss one other thing: what about other values of α (the parameter that determines the mix between lasso and ridge regression). For example, setting $\alpha = 0.2$ (the default in `glmnet` is 1) is a lot closer to ridge regression. In that case we enter

```
fit=glmnet(x,y,alpha=0.2)
cvfit=cv.glmnet(x,y,alpha=0.2,nfolds=100)
cvfit$lambda.min
coef(cvfit, s = "lambda.min")
which(fit$lambda==cvfit$lambda.min)
```

In that case we find

```
> cvfit$lambda.min
[1] 0.1638291
> coef(cvfit, s = "lambda.min")
21 x 1 sparse Matrix of class "dgCMatrix"
                       1
(Intercept)  0.15802821
V1           1.30156479
V2           0.01665947
V3           0.69691040
V4           .
V5          -0.84135177
V6           0.57430374
V7           0.07950233
V8           0.35196779
V9           .
V10          0.04885985
V11          0.21204443
V12         -0.02439286
V13         -0.01500440
V14         -1.07433517
```

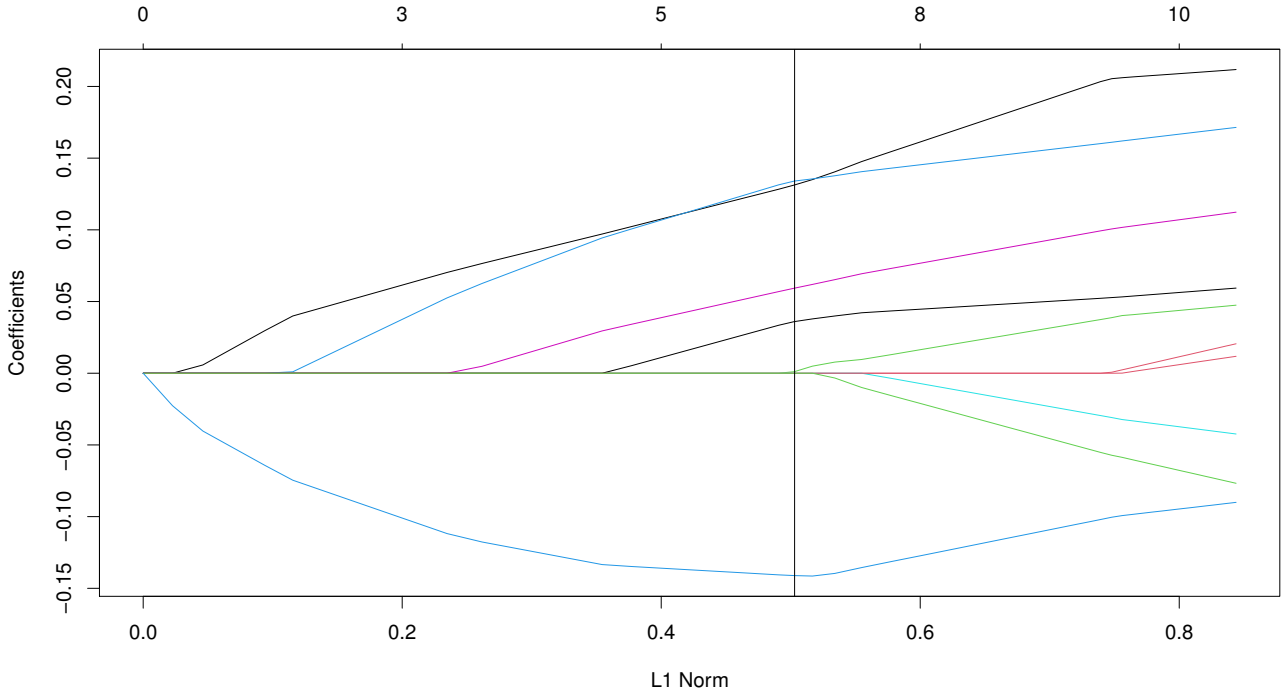

Figure 5: Result of `glmnet` for the nuclear power data

```
V15         -0.05465064
V16          .
V17          .
V18          0.02529988
V19          .
V20         -1.06601593
> which(fit$lambda==cvfit$lambda.min)
[1] 43
> cvfit$cvm[43]
    s42
1.09767
```

so there are indeed many more non-zero parameter estimates than in the lasso case, but the final mean squared prediction error (1.097..) is larger than the 1.02 we found with lasso so it looks as though lasso really is better, in this case. However, we could of course try other values of α.

3.1 Nuclear power data

Now let's go back to our old friend, the nuclear power data. Recall that in our earlier analysis of the data frame `nukes` we found

```
> lm4=lm(LC~D+LT1+LT2+LS+PR+NE+CT+BW+LN+PT,nukes)
```

```
> lm5=lm(LC~D+LS+NE+CT+LN+PT,nukes)
> anova(lm4,lm5)
  Res.Df     RSS Df Sum of Sq      F Pr(>F)
1     21 0.56803
2     25 0.63374 -4 -0.065714 0.6074 0.6617
> summary(lm5)
             Estimate Std. Error t value Pr(>|t|)
(Intercept) -13.26031    3.13950  -4.224 0.000278 ***
D             0.21241    0.04326   4.910 4.70e-05 ***
LS            0.72341    0.11882   6.088 2.31e-06 ***
NE            0.24902    0.07414   3.359 0.002510 **
CT            0.14039    0.06042   2.323 0.028582 *
LN           -0.08758    0.04147  -2.112 0.044891 *
PT           -0.22610    0.11355  -1.991 0.057490 .
Residual standard error: 0.1592 on 25 degrees of freedom
Multiple R-squared:  0.8569,    Adjusted R-squared:  0.8225
```

in which the original ten covariates in model `lm4` were reduced by backward selection to six in `lm5`, and an anova test of one model against the other produced a p-value of 0.66, indicating that no significant effects were lost in reducing to the 6-covariate model. Now, let's see what lasso has to say about this dataset.

Although in our earlier analysis we already had a fair amount of discussion of possible transformations in the dataset, it is worth repeating some of the checks that were done with the test dataset, since lasso and ridge regression analyses are more sensitive to scaling issues than standard least squares regression. In fact, for this dataset the means of the `x` variables range from 0.187 to 66.58, and the standard devations from 0.17 to 1.01, which does suggest the need for some rescaling. Moreover, the eignvalues of $X^T X$ turn out to be

```
> eigen(t(x) %*% x)$val
 [1] 1.528427e+05 3.055580e+01 1.030780e+01 7.141386e+00 6.321334e+00 4.694459e+00
 [7] 2.689490e+00 1.972372e+00 6.079432e-01 3.352477e-01
```

which implies a condition number of 675, not good! However, a simple scaling seems to resolve this:

```
> x=scale(x)
> eigen(t(x) %*% x)$val
 [1] 88.724641 51.202706 40.648161 38.845486 27.545258 23.451593 21.328418  8.045346
 [9]  6.785769  3.422623
>
> sqrt(eigen(t(x) %*% x)$val[1]/eigen(t(x) %*% x)$val[10])
[1] 5.091463
```

which is much better! As a side comment, sometimes problems with multicollinearity and high condition numbers can be resolved with simple rescaling operations, but the estimates could still be improved with shrinkage methods.

Let's repeat the lasso steps for this dataset:

```
fit=glmnet(x,y)
cvfit=cv.glmnet(x,y,nfolds=32)
plot(cvfit)
coef(cvfit, s = cvfit$lambda.min)
sum(abs(coef(cvfit, s = cvfit$lambda.min)[2:11]))
plot(fit)
lines(c(0.503,0.503),c(-10,10))
```

where the optimal L1 Norm of 0.503 comes from

```
> sum(abs(coef(cvfit, s = cvfit$lambda.min)[2:11]))
[1] 0.5027698
```

and the coefficients of the lasso estimator are

```
> coef(cvfit, s = cvfit$lambda.min)
11 x 1 sparse Matrix of class "dgCMatrix"
                       1
(Intercept)  6.067176261
V1           0.131257531
V2           .
V3           0.001075573
V4           0.134011975
V5           .
V6           0.059304260
V7           0.036004694
V8           .
V9           .
V10         -0.141115757
```

The two plots produced here are in Figure 5 and Figure 6.

In contrast, the OLS fit with the same covariates is

```
> summary(lm6)
                          Estimate Std. Error t value Pr(>|t|)
(Intercept)                6.06718    0.02964 204.707  < 2e-16 ***
x[, c(1, 3, 4, 6, 7, 10)]1  0.17396    0.03827   4.546 0.000121 ***
x[, c(1, 3, 4, 6, 7, 10)]2  0.04452    0.03557   1.251 0.222351
x[, c(1, 3, 4, 6, 7, 10)]3  0.15117    0.03212   4.706 7.98e-05 ***
x[, c(1, 3, 4, 6, 7, 10)]4  0.08890    0.03194   2.784 0.010087 *
x[, c(1, 3, 4, 6, 7, 10)]5  0.05723    0.03209   1.783 0.086688 .
x[, c(1, 3, 4, 6, 7, 10)]6 -0.14480    0.03789  -3.822 0.000781 ***
Residual standard error: 0.1677 on 25 degrees of freedom
Multiple R-squared:  0.8413,    Adjusted R-squared:  0.8032
```

which shows, in every case, a smaller estimate (in absolute value) under the lasso estimate than the least squares estimate, except for the intercept which stays the same.

It's worth checking out the cross-validated MSPE:

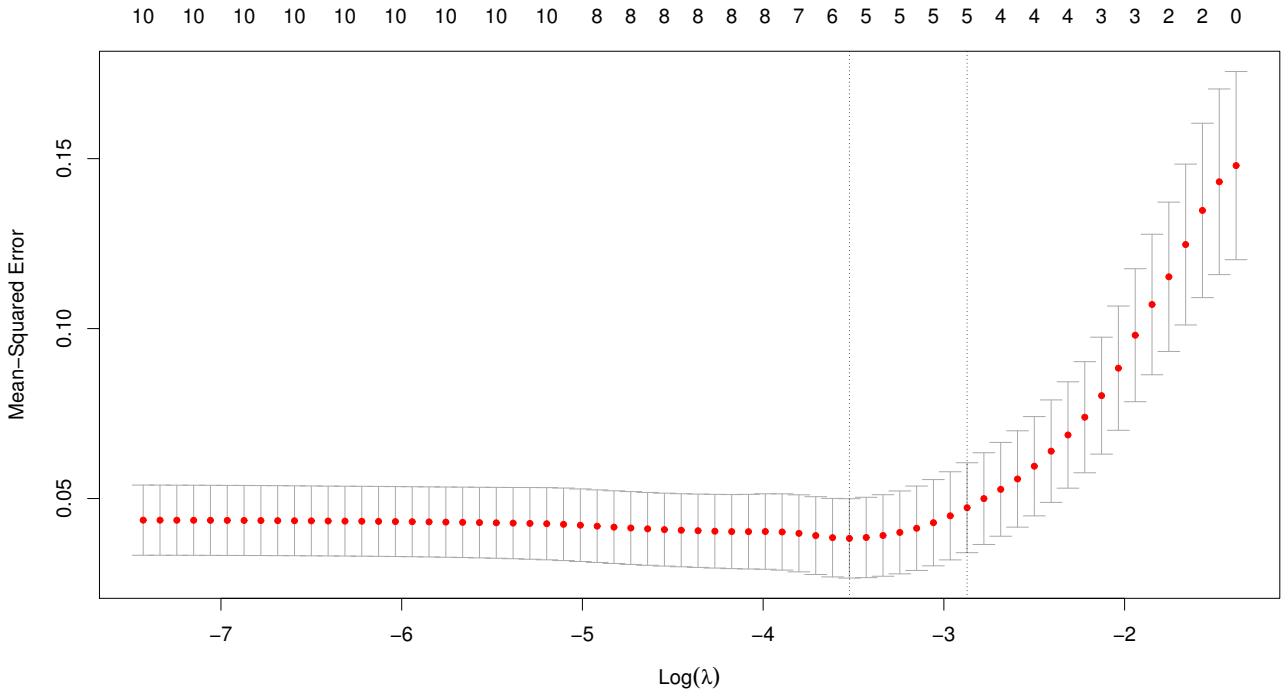

Figure 6: Result of `cv.glmnet` for the nuclear power data

```
> cvfit$cvm[which(fit$lambda==cvfit$lambda.min)]
0.03827464
```

How does this compare with the least squares estimates? We repeat the lasso analysis with just the six selected covariates:

```
fit1=glmnet(x[,c(1,3,4,6,7,10)],y)
cvfit1=cv.glmnet(x[,c(1,3,4,6,7,10)],y,nfolds=32)
```

In this case, the model is evaluated at 60 values of λ, and if we print out the values for the smallest λ (nearest to the least squares estimates) we find

```
fit1$beta[,60]
         V1          V2          V3          V4          V5          V6
 0.17244622  0.04298682  0.15056627  0.08785952  0.05648051 -0.14468237
> cvfit1$cvm[60]
0.03783014
```

nearly but not exactly the same as the least squares parameter estimates, and a very slightly smaller value of the cross-validated MSPE (0.0378 compared with 0.0383 for the lasso estimator).

So maybe the least squares variable selection procedure is better after all!

To be continued...

References

[1] Leo Breiman. Better subset regression using the nonnegative garrote. *Technometrics*, 37(4):373–384, 1995.

[2] D. Donoho and I. Johnstone. Ideal spatial adaptation by wavelet shrinkage. *Biometrika*, 81:425–455, 1994.

[3] J.J. Faraway. *Linear Models with R. Second Edition.* Chapman and Hall/CRC Press, Boca Raton, Florida, 2014.

[4] I. Frank and J. Friedman. A statistical view of some chemometrics regression tools (with discussion). *Technometrics*, 35(2):109–148, 1993.

[5] Wenjiang J. Fu. Penalized regressions: The bridge versus the lasso. *Journal of Computational and Graphical Statistics*, 7(3):397–416, 1998.

[6] A.N. Gorban, E.M. Mirkes, and A. Zinovyev. Piece-wise quadratic approximations of arbitrary error functions for fast and robust machine learning. *Neural Networks*, 84:28–38, 2016.

[7] Arthur E. Hoerl and Robert W. Kennard. Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, 12(1):55–67, 1970.

[8] Fadil Santosa and William W. Symes. Linear inversion of band-limited reflection seismograms. *SIAM Journal on Scientific and Statistical Computing*, 7(4):1307–1330, 1986.

[9] R.L. Smith and K.D.S. Young. *Linear Regression.* Course Notes, University of North Carolina, 20??

[10] C. Stein. Estimation of the mean of a multivariate normal distribution. *Annals of Statistics*, 9:1135–1151, 1981.

[11] Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, 58(1):267–288, 1996.

[12] Hui Zou and Trevor Hastie. Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society. Series B (Statistical Methodology)*, 67(2):301–320, 2005.