Basic Introduction to the Lasso

```
library(glmnet)
```

```
## Loading required package: Matrix
```

```
## Loaded glmnet 3.0-2
```

```
library(tidyverse)
```

```
## -- Attaching packages ---------------------------------------------------------------- tidyverse 1.3.0
```

```
## v ggplot2 3.3.0     v purrr   0.3.3
## v tibble  2.1.3     v dplyr   0.8.5
## v tidyr   1.0.2     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.5.0
```

```
## -- Conflicts ------------------------------------------------------------------- tidyverse_conflicts()
## x tidyr::expand() masks Matrix::expand()
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## x tidyr::pack()   masks Matrix::pack()
## x tidyr::unpack() masks Matrix::unpack()
```

Introduction

This set of notes will describe a few methods for performing variable selection with a focus on the lasso. These are in the `R` package `glmnet` so first load the package (after installation).

```
library(glmnet)
```

Recall the standard linear regression model for predicting a quantitative response variable y from a single quantitative predictor x is

$$y = \beta_0 + \beta_1 x + \epsilon.$$

We used `R` to estimate the slope and intercept and denoted these by $\hat{\beta}_0$ and $\hat{\beta}_1$. They are selected by minimizing the sum of the squared residuals. Basically, pick $\hat{\beta}_0$ and $\hat{\beta}_1$ such that the quantity below is minimized (this is just the sum of the squared residuals).

$$\sum_{i=1}^{n} \left(y_i - \beta_0 - \beta_1 x_i \right)^2$$

We then extended this model to the case where there are p explanatory variables denoted $x_1, ..., x_p$ and a response variable y. We called this multiple regression, with the model given below,

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + ... + \beta_p x_p + \epsilon.$$

We again used least squares to obtain our estimates. We pick $\hat{\beta}_0, ..., \hat{\beta}_p$ such that the quantity below is minimized

$$\sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j \right)^2.$$

Robert Tibshirani identified two reasons why the least squares estimates above leave something to be desired. The first is **prediction accuracy** and the second is **interpretability**.

The first problem is prediction accuracy. To see why this is a problem, we'll first have to discuss the bias - variance trade off.

Consider a general regression setup below.

$$y = f(x) + \epsilon$$

We want to estimate $f(x)$ using $\hat{f(x)}$ and we assume the errors have mean zero and variance σ^2. Ideally we want $\hat{f(x)}$ to be "close" to $f(x)$ both for points in our sample data and for new data from outside our sample. We formalize this notion of "closeness"" by considering the mean square error between y and $\hat{f(x)}$, $(y - \hat{f(x)})^2$.

It turns out we can decompose the expected error in a clever way.

$$\text{Expected Error} = \text{Bias}^2 + \text{Variance} + \text{Irreducible Error}$$

Here bias refers to how far our predictions for a given point are from the truth and variance refers to how much our predictions for a given point vary. The final term is the irreducible error, or the error in the true relationship. In our example this is σ^2. Looking at the expression above, it should be clear that there is a trade-off between bias and variance. Although intuitively we may want to reduce bias at the expense of variance, in practice this is usually unwise. Bias refers to the long-run average, and we are generally interested in a single realization of our dataset. Sophisticated modelers should recognize the trade-off and make appropriate and justifiable decisions regarding the balance between bias and variance.

Returning to prediction accuracy, least squares estimates generally have low bias and high variance. Note also if the predictors are highly correlated, a large positive coefficient can be cancelled by a large negative one (high variance). We can sometimes improve prediction accuracy by decreasing the variance at the expense of the bias. In a practical sense this will be done by shrinking some of the coefficients to zero.

The second problem is interpretability. A large number of predictors can be cumbersome, so we may want to examine a subset of them that have the biggest effects, excluding those predictors that have small effects.

We'll illustrate the methods using a dataset called `prostate` collected from men about to receive a radical prostatectomy. A number of clinical measures are recorded and researchers are specifically interested in prediction of the level of prostate specific antigen `lpsa`. This dataset is in the `ElemStatLearn` package so first load it into `R`.

```
prostate<-data.matrix(read_csv("Prostate.csv"))
```

You can examine the documentation and perform exploratory data analysis on your own.

Best Subset Selection

One possible solution to the two problems of **prediction accuracy** and **interpretability** is **best-subset selection**. This method finds for every possible number of predictors $(0, 1, ..., p)$, the subset that gives the smallest residual sum of squares. [There is a file on the R server *BestSubsets.R* that demonstrates this approach if you are interested.]

Stepwise Selection

Another possible solution is **stepwise selection** (either forwards or backwards). Forwards and backwards selection are both described in your textbook.

Both best subset selection and stepwise selection have problems. Since they are discrete processes, they can sometimes exhibit high variance. A continuous set of methods using shrinkage don't suffer from high variance.

Shrinkage Methods

Ridge Regression

Ridge regression begins with the standard residual sum of squares, but penalizes using the sum of squares of the parameters.

The ridge regression coefficient estimates are determined by finding the $\hat{\beta}_1, ..., \hat{\beta}_p$ such that

$$\sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j \right)^2$$

is minimized, subject to the constraint

$$\sum_{j=1}^{p} \beta_j^2 \leq t.$$

Lasso Regression

Lasso stands for Least Absolute Shrinkage and Selection Operator. It is similar to ridge regression with a few important differences.

The lasso coefficient estimates are determined by finding the $\hat{\beta}_1, ..., \hat{\beta}_p$ such that

$$\sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j \right)^2$$

is minimized, subject to the constraint

$$\sum_{j=1}^{p} |\beta_j| \leq t$$

We can examine shrinkage methods by considering a case with two predictors. The point represents the ordinary least squares estimate, the constraint region is the shaded square and circle and the ellipses are the contours of the least squares error function. Both ridge regression and the lasso find the first point at which the contour hits the constraint region. We can notice an important difference between ridge regression and the lasso. Lasso will shrink coefficients to zero while this is unlikely for ridge regression (do you see why?).

In both ridge regression and the lasso, the amount of shrinkage is determined by the parameter t, called a tuning parameter. A tuning parameter of zero yields the linear regression estimates for both ridge regression and the lasso and larger values of t mean more shrinkage. For the lasso, as t increases, more coefficients are set to zero. The effect of the tuning parameter t can be seen in the plots below for both ridge regression and the lasso.

We'll use the function `glmnet` from the `glmnet` package. This function requires a matrix of predictors as well as the response. We'll note the matrix of predictors `X` and the response `y`. Since we generally want the model to not depend on units (ie height in centimeters or height in inches), we standardize both our predictors and the response variable. This is handled internally in `R`.

```
X <- apply(prostate[,-c(9,10)], 2, as.numeric)
y <- prostate[,9]
```

Then we'll fit the model. For lasso we use `alpha = 1` and for ridge regression we use `alpha = 0`. The default is `alpha = 1`, so the argument below isn't actually necessary.

```
lasso.model <- glmnet(X, y, alpha = 1)
```

We can show the number of nonzero entries as well as the tuning parameter (Lambda).

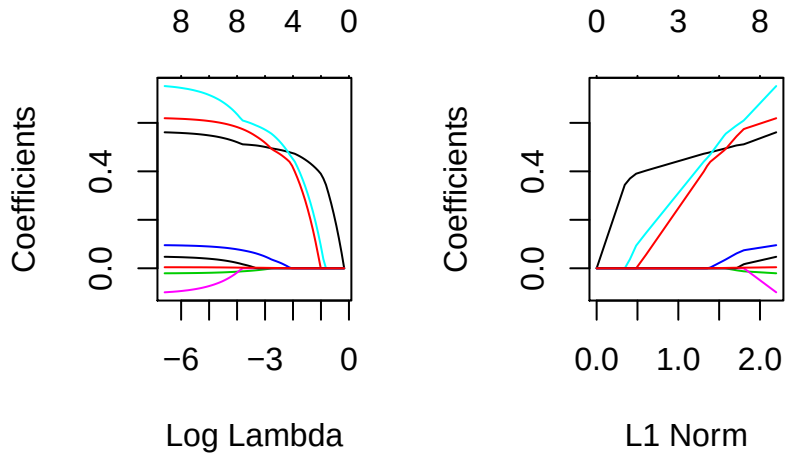
```
print(lasso.model)
```

```
##
## Call:  glmnet(x = X, y = y, alpha = 1)
##
##    Df    %Dev  Lambda
## 1   0 0.00000 0.84340
## 2   1 0.09159 0.76850
## 3   1 0.16760 0.70020
## 4   1 0.23070 0.63800
## 5   1 0.28320 0.58130
## 6   1 0.32670 0.52970
## 7   1 0.36280 0.48260
## 8   1 0.39280 0.43980
## 9   2 0.42210 0.40070
## 10  2 0.44900 0.36510
## 11  3 0.48010 0.33270
## 12  3 0.50660 0.30310
## 13  3 0.52850 0.27620
## 14  3 0.54680 0.25160
## 15  3 0.56190 0.22930
## 16  3 0.57450 0.20890
## 17  3 0.58490 0.19040
## 18  3 0.59360 0.17350
## 19  3 0.60080 0.15800
## 20  3 0.60670 0.14400
## 21  4 0.61230 0.13120
```

```
## 22  5 0.61750 0.11960
## 23  5 0.62240 0.10890
## 24  5 0.62640 0.09926
## 25  5 0.62980 0.09044
## 26  5 0.63250 0.08240
## 27  5 0.63490 0.07508
## 28  5 0.63680 0.06841
## 29  6 0.63890 0.06234
## 30  6 0.64210 0.05680
## 31  6 0.64480 0.05175
## 32  6 0.64700 0.04715
## 33  6 0.64880 0.04297
## 34  6 0.65030 0.03915
## 35  7 0.65160 0.03567
## 36  7 0.65270 0.03250
## 37  7 0.65360 0.02961
## 38  7 0.65440 0.02698
## 39  7 0.65500 0.02459
## 40  7 0.65550 0.02240
## 41  8 0.65670 0.02041
## 42  8 0.65780 0.01860
## 43  8 0.65880 0.01695
## 44  8 0.65950 0.01544
## 45  8 0.66020 0.01407
## 46  8 0.66070 0.01282
## 47  8 0.66120 0.01168
## 48  8 0.66160 0.01064
## 49  8 0.66190 0.00970
## 50  8 0.66210 0.00884
## 51  8 0.66230 0.00805
## 52  8 0.66250 0.00734
## 53  8 0.66270 0.00668
## 54  8 0.66280 0.00609
## 55  8 0.66290 0.00555
## 56  8 0.66300 0.00506
## 57  8 0.66300 0.00461
## 58  8 0.66310 0.00420
## 59  8 0.66320 0.00382
## 60  8 0.66320 0.00348
## 61  8 0.66320 0.00318
## 62  8 0.66330 0.00289
## 63  8 0.66330 0.00264
## 64  8 0.66330 0.00240
## 65  8 0.66330 0.00219
## 66  8 0.66330 0.00199
## 67  8 0.66330 0.00182
## 68  8 0.66330 0.00166
## 69  8 0.66340 0.00151
## 70  8 0.66340 0.00138
```

We can also visualize the model. The left figure shows how the coefficients shrink as the penalty becomes more and more important (Log Lambda on the x-axis), and the right figure shows the same but for the L-1 norm.

```
par(mfrow = c(1, 2))
plot(lasso.model, xvar="lambda")
plot(lasso.model, xvar="norm")
```



We have a series of models each fit at a different value of the tuning parameter t. The question now becomes how do we select an appropriate model? There are a number of possible methods, but we'll focus on cross validation.

k-Fold Cross Validation

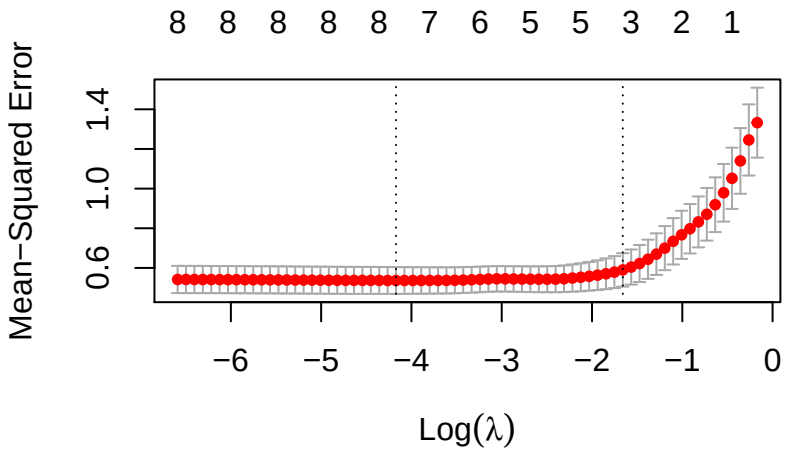
k-Fold Cross Validation works by splitting the data into k equal parts (called folds). The method is fit on $k-1$ of the folds using a range of values of the tuning parameter, and we calculate the prediction error for the remaining unused part of the data. We do this for each of the k folds and take the average of these prediction errors, calling this the cross-validation error.

$$e_i(t) = \sum_{j \text{ in Fold i}} (y_i - \hat{y}(t))^2$$

Note the dependence of our prediction on the tuning parameter is made explicit in the expression above. If $k = n$, this is called leave-one-out cross validation (see why?), but common choices for k are 5 or 10.

Using this method, we obtain an estimate of the prediction error at a number of values of the tuning parameter. Let's examine this for the `prostate` data below using 10 fold cross validation.

```
cv.lasso.model <- cv.glmnet(X, y, nfolds = 10)
par(mfrow = c(1,1))
plot(cv.lasso.model)
```



What do you notice? What seems like a reasonable choice of t? One possibility is to choose the t that minimizes the cross validation error, however this tends to not eliminate enough predictors. Another possibility is to take the largest t such that cross validation error curve is within one standard error of the minimum. These correspond to the left and rich vertical dashed lines.

We can also obtain these from `cv.lasso.model`.

```
cv.lasso.model$lambda.min
```

```
## [1] 0.01544095
```

```
cv.lasso.model$lambda.1se
```

```
## [1] 0.1903632
```

Finally, we can make a prediction. Let's try predicting the first 5 observations using our value of t based on the 1 standard error rule.

```
predict(cv.lasso.model, newx = X[1:5,],  s = cv.lasso.model$lambda.1se)
```

```
##             1
## [1,] 1.249898
## [2,] 1.234077
## [3,] 1.256734
## [4,] 1.126995
## [5,] 2.064470
```

Let's also examine our coefficients using the tuning parameter within 1 standard error of the minimum.

```
coef(cv.lasso.model, s = "lambda.1se")
```

```
## 9 x 1 sparse Matrix of class "dgCMatrix"
##                      1
```

```
## (Intercept) 0.6435469
## lcavol      0.4553889
## lweight     0.3142829
## age         .
## lbph        .
## svi         0.3674270
## lcp         .
## gleason     .
## pgg45       .
```

Or the coefficients using the tuning parameter using the minimum.

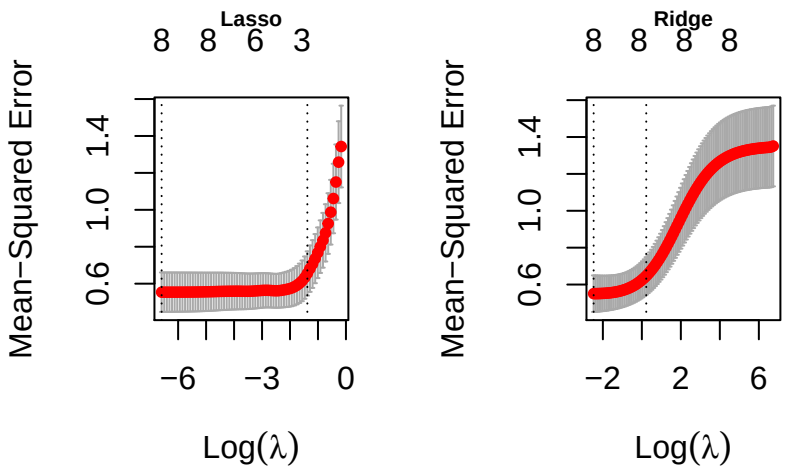
```
coef(cv.lasso.model, s = "lambda.min")
```

```
## 9 x 1 sparse Matrix of class "dgCMatrix"
##                        1
## (Intercept)  0.187557481
## lcavol       0.527313407
## lweight      0.588813499
## age         -0.015155012
## lbph         0.081126027
## svi          0.655172782
## lcp         -0.031389445
## gleason      0.026618793
## pgg45        0.003064293
```

We can also compare the lasso to ridge regression. First, made the folds

```
folds <- sample(1:10,size=length(y), replace=TRUE)
lasso.cv.1 <- cv.glmnet(X,y, foldid = folds, alpha = 1)
lasso.cv.0 <- cv.glmnet(X,y, foldid = folds, alpha = 0)
```

```
par(mfrow=c(1,2))
plot(lasso.cv.1, main = "Lasso", cex.main = .7)
plot(lasso.cv.0, main = "Ridge", cex.main = .7)
```



```
par(mfrow = c(1,1))
plot(log(lasso.cv.1$lambda),lasso.cv.1$cvm,pch=19,col="red",xlab="log(Lambda)",ylab=lasso.cv.1$name, xli
points(log(lasso.cv.0$lambda),lasso.cv.0$cvm,pch=19,col="blue")
legend("topleft",legend=c("alpha= 1","alpha 0"),pch=19,col=c("red","blue"), bty ="n")
```