6.4 Shrinkage Methods

The variable selection methods we already discussed work quite well in practice when it comes to data sets where we do not have too many predictors. For example, recall that the *leaps and bound* algorithms are good when $p < 50$. So, what could we do if we have too many predictors?

In this section, we are going to study three different methods we can use to **shrink** the number of predictors in order to find a *trade-off between model bias and prediction error (variance)*.

6.4.1 Principal Components Regression

The goal of principal components regression is to **reduce** the dimension in the predictors space, keeping in mind that in many cases, when many predictors are available, they might also be highly correlated. We are going to do so by searching where the variation of the data is greater and then transforming the predictors in a way that assigns higher weights to the data depending on the percentage of variance explained by them.

Take for example the design matrix $\mathbf{X}$ of predictors and center the columns of $\mathbf{X}$ to have zero mean. Consider $\mathbf{X}$ with no intercept column. Then, to find directions of greater variation in matrix $\mathbf{X}$, we take the following steps:

1. Find a vector $\mathbf{u}_1$ to maximize variance of $\mathbf{u}_1^\top \mathbf{X}$ *subject to* $\mathbf{u}_1^\top \mathbf{u}_1 = 1$.
2. Find $\mathbf{u}_2$ to maximize variance of $\mathbf{u}_2^\top \mathbf{X}$ subject to $\mathbf{u}_1^\top \mathbf{u}_2 = 0$ *and* $\mathbf{u}_2^\top \mathbf{u}_2 = 1$.
3. Continue looking for directions of greatest variation in the data which are **orthogonal** to the previous ones.

4. Continue until the total number of dimensions is exhausted.

The **principal components** are given by the columns of matrix $\mathbf{Z}$, where

$$\mathbf{Z} = \mathbf{XU}$$

$\mathbf{z}_i$ and $\mathbf{u}_i$ are the columns of $\mathbf{Z}$ and $\mathbf{U}$ respectively. U is called the **rotation** matrix. $\mathbf{Z}$ is a version of the data rotated in such a way that the resulting principal components are *orthogonal.*

Each Principal Component is a *linear combination* of the original variables $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_m$ with **weights** given by each column $\mathbf{u}_i$ of matrix $\mathbf{U}$:

$$\mathbf{z}_i = u_{1i}\mathbf{x}_1 + u_{2i}\mathbf{x}_2 + \ldots + u_{mi}\mathbf{x}_m$$

To run a PCA, we need to replace the model $Y \sim X$ by the model $Y \sim Z$, and we only need to use the **first few columns of Z as predictors**. Unfortunately, the interpretation of the PCAs as predictors might be challenging, since we need to use the values of $\mathbf{u}_i$ in the rotation matrix (also called the **loadings**) for interpretation.

How many Principal Components?

The trace of the sample variance-covariance S of $\mathbf{X}$ (total sample variance) is equal to the sum if its eigenvalues:

$$trace(S) = s_1^2 + s_2^2 + \ldots + s_m^2 = \lambda_1 + \lambda_2 + \ldots + \lambda_m$$

It has been observed that most of the total variance of a data set is concentrated in the first principal components.

- One way to determine the appropriate number of components is to look at the **cumulative percentage of variation** that is explained by the first principal components, and retain the number of PCs explaining between 70% to 90% of the total variation.

- We can also make a plot of the principal component's standard deviations ($\sqrt{\lambda_i}$) vs. the PC index i (**scree plot**). and we look for the PC index i where there is a big change in slope (the elbow) in the scree plot.
- Another way is to simply *discard* principal components such that $\lambda_i < \bar{\lambda}$ (the average λ).
- Finally, we can use the RMSPE (**Root Mean Square Prediction Error**) to decide how many components to keep. The main idea is to compute the RMSPE on the testing set using all of the components and then choose the number of components that minimize the RMSPE. Actually, the best way to select the "optimal" RMPSE is to use (10-Fold) Cross Validation.

We illustrate all these methods in the `meatspec` example below:

The `meatspec` Example

Consider the `meatspec` data set from the `faraway` library: A Tecator Infratec Food and Feed Analyzer working in the wavelength range 850 - 1050 nm by the Near Infrared Transmission (NIT) principle was used to collect data on samples of finely chopped pure meat. *215* samples were measured. For each sample, the *fat content* was measured along with a 100 channel spectrum of absorbances. Since determining the fat content via analytical chemistry is time consuming we would like to build a model to *predict* the fat content of new samples using the 100 absorbances which can be measured more easily.

PCR Steps

1. Partition the data into *training* sample and *testing* sample to test model performance.
    - Training sample: First 172 observations
    - Testing sample: Remaining 43 observations.

```
data(meatspec,package="faraway")
trainmeat<-meatspec[1:172,]
testmeat<-meatspec[173:215,]
```

2. Fit model with all predictors using the training data set and make a prediction on the testing set.

```
meat.training<-lm(fat~.,trainmeat)
#summary(meat.training)$r.sq
```

The R^2 of the model is extremely high. If we take a quick look at the p-values of the predictors, we observe that the majority of the predictors are not statistically significant.

We will use the *sample RMSE* which is defined as

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - y_i)^2}$$

for some arbitrary vectors x and y. We defined our RMSE function in `R`

```
rmse<-function(x,y) sqrt(mean((x-y)^2))
```

Now, we use it to compute the RMSE in the training and testing data sets.

```
rmse(fitted(meat.training), trainmeat$fat)
```

```
## [1] 0.6903167
```

```
rmse(predict(meat.training,testmeat),testmeat$fat)
```

```
## [1] 3.814
```

3. Select few PCs representing the 100 predictors and repeat the process (shrinkage effect). We choose to include up to 50 components and we fit a *principal components regression* (function `pcr` in the `pls` library)

```
library(pls)
meat.pcr<-pcr(fat ~ ., data=trainmeat, scale=TRUE, ncomp=50)
```

Here we chose `scale=TRUE` to standardize the data before running the PCR. If we look at the summary output of the PCR, we have that

```
summary(meat.pcr)
```

```
## Data:    X dimension: 172 100
##  Y dimension: 172 1
## Fit method: svdpc
## Number of components considered: 50
## TRAINING: % variance explained
##      1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps  8 comps
## X      98.50    99.59    99.88    99.99   100.00   100.00   100.00   100.00
## fat    22.32    26.16    65.31    88.91    93.51    94.38    94.58    94.67
##      9 comps  10 comps  11 comps  12 comps  13 comps  14 comps  15 comps
## X     100.00    100.00       100    100.00    100.00    100.00     100.0
## fat    95.19     95.29        96     96.01     96.01     96.13      96.5
##      16 comps  17 comps  18 comps  19 comps  20 comps  21 comps  22 comps
## X      100.00       100    100.00    100.00    100.00    100.00    100.00
## fat     96.83        97     97.04     97.42     97.59     97.62     97.66
##      23 comps  24 comps  25 comps  26 comps  27 comps  28 comps  29 comps
## X      100.00    100.00     100.0     100.0    100.00    100.00    100.00
## fat     97.75     97.76      97.8      97.9     97.91     98.09     98.17
##      30 comps  31 comps  32 comps  33 comps  34 comps  35 comps  36 comps
## X      100.00    100.00    100.00    100.00    100.00    100.00     100.0
## fat     98.19     98.22     98.27     98.28     98.29     98.29      98.3
##      37 comps  38 comps  39 comps  40 comps  41 comps  42 comps  43 comps
## X      100.00    100.00    100.00    100.00    100.00     100.0     100.0
## fat     98.31     98.46     98.48     98.57     98.57      98.7      98.7
##      44 comps  45 comps  46 comps  47 comps  48 comps  49 comps  50 comps
## X      100.00    100.00    100.00    100.00    100.00    100.00    100.00
## fat     98.74     98.83     98.91     98.91     98.98     98.98     98.98
```

This table tells us the percentage of the variance in the response variable explained by the principal components. We can see the following: By using just the first principal component, we can explain 98.50% of the variation in the response variable. By adding in the second

principal component, we can explain 99.59% of the variation in the response variable.

If we choose 4 components, and fit a PCR with these 4, then the RMSE in the training and testing data sets:

```
rmse(predict(meat.pcr,ncom=4),trainmeat$fat)
```

```
## [1] 4.210361
```

```
rmse(predict(meat.pcr,testmeat,ncomp=4),testmeat$fat)
```

```
## [1] 4.748898
```

Can we select the number of PCs to minimize the prediction error instead? Of course, by using *Cross-Validation*.

- Use the `prcomp` function to calculate the PCs and extract the λ's squared-roots (st. dev.) and eigenvectors (rotation) of the variance-covariance matrix:

```
meat.pca<-prcomp(trainmeat[,-101])
# we remove the last column, because it contains the response.
summary(meat.pca)
```

```
## Importance of components:
##                            PC1     PC2     PC3     PC4     PC5     PC6     PC7
## Standard deviation     5.0554 0.51114 0.28176 0.16771 0.03817 0.02457 0.01432
## Proportion of Variance 0.9857 0.01008 0.00306 0.00108 0.00006 0.00002 0.00001
## Cumulative Proportion  0.9857 0.99576 0.99882 0.99991 0.99996 0.99999 0.99999
##                            PC8      PC9     PC10     PC11     PC12     PC13
## Standard deviation     0.0108 0.004533 0.003432 0.002099 0.001759 0.000998
## Proportion of Variance 0.0000 0.000000 0.000000 0.000000 0.000000 0.000000
## Cumulative Proportion  1.0000 1.000000 1.000000 1.000000 1.000000 1.000000
##                              PC14      PC15      PC16      PC17      PC18
## Standard deviation     0.0007307 0.0005441 0.0004426 0.0003933 0.0003409
## Proportion of Variance 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
## Cumulative Proportion  1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
##                              PC19      PC20      PC21      PC22      PC23
## Standard deviation     0.0002888 0.0002046 0.0001958 0.0001904 0.0001646
## Proportion of Variance 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
## Cumulative Proportion  1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
##                              PC24      PC25      PC26      PC27      PC28
## Standard deviation     0.0001423 0.0001125 0.0001059 0.0001019 9.614e-05
## Proportion of Variance 0.0000000 0.0000000 0.0000000 0.0000000 0.000e+00
## Cumulative Proportion  1.0000000 1.0000000 1.0000000 1.0000000 1.000e+00
##                              PC29      PC30      PC31      PC32      PC33
## Standard deviation     8.896e-05 8.082e-05 7.738e-05 7.372e-05 6.861e-05
## Proportion of Variance 0.000e+00 0.000e+00 0.000e+00 0.000e+00 0.000e+00
## Cumulative Proportion  1.000e+00 1.000e+00 1.000e+00 1.000e+00 1.000e+00
##                              PC34      PC35      PC36      PC37      PC38
## Standard deviation     6.326e-05 5.761e-05 5.392e-05 4.801e-05 4.685e-05
## Proportion of Variance 0.000e+00 0.000e+00 0.000e+00 0.000e+00 0.000e+00
## Cumulative Proportion  1.000e+00 1.000e+00 1.000e+00 1.000e+00 1.000e+00
##                              PC39      PC40      PC41      PC42      PC43
## Standard deviation     4.418e-05 4.271e-05 3.768e-05 3.534e-05 3.435e-05
## Proportion of Variance 0.000e+00 0.000e+00 0.000e+00 0.000e+00 0.000e+00
## Cumulative Proportion  1.000e+00 1.000e+00 1.000e+00 1.000e+00 1.000e+00
##                              PC44      PC45      PC46      PC47      PC48
## Standard deviation     3.291e-05 3.117e-05 2.803e-05 2.662e-05 2.635e-05
```

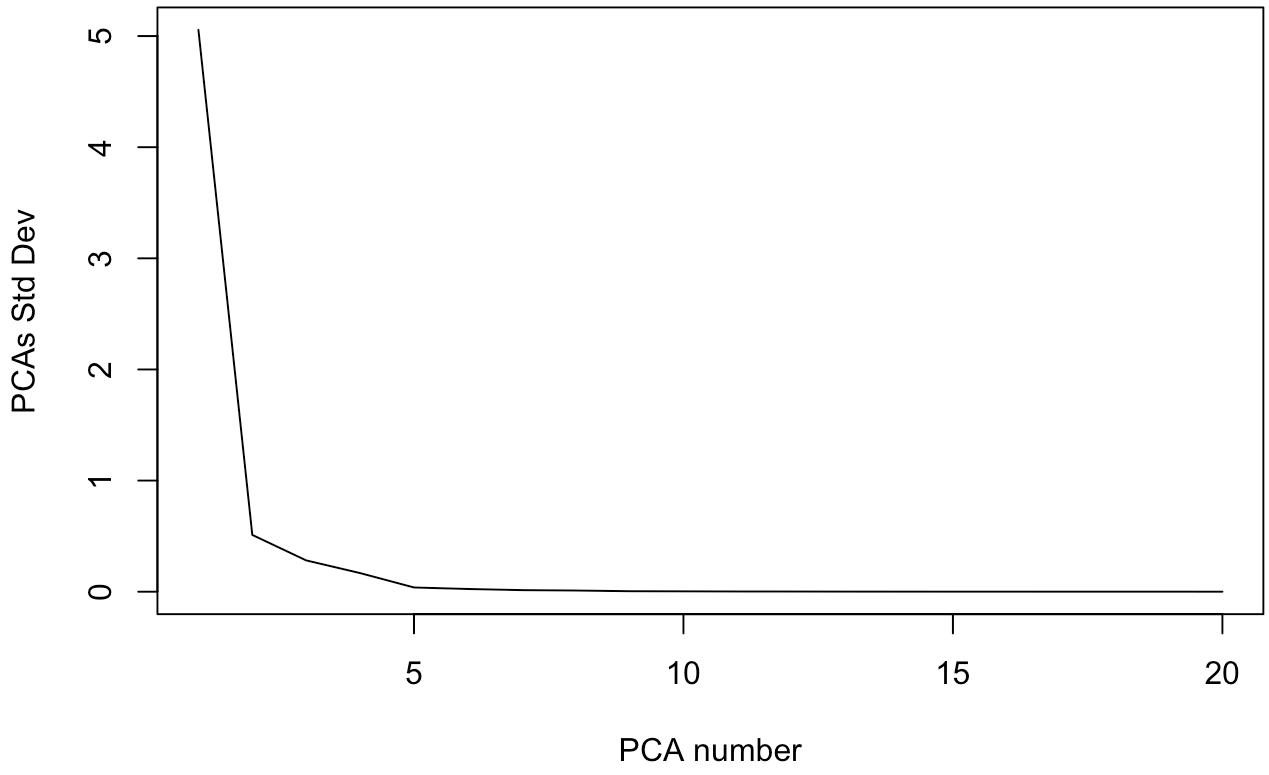
```
## Proportion of Variance 0.000e+00 0.000e+00 0.000e+00 0.000e+00 0.000e+00
## Cumulative Proportion  1.000e+00 1.000e+00 1.000e+00 1.000e+00 1.000e+00
##                            PC49      PC50     PC51      PC52     PC53     PC5
## Standard deviation     2.343e-05 2.193e-05 2.14e-05 2.082e-05 1.95e-05 1.85e-0
## Proportion of Variance 0.000e+00 0.000e+00 0.00e+00 0.000e+00 0.00e+00 0.00e+0
## Cumulative Proportion  1.000e+00 1.000e+00 1.00e+00 1.000e+00 1.00e+00 1.00e+0
##                            PC55      PC56      PC57      PC58      PC59
## Standard deviation     1.752e-05 1.746e-05 1.633e-05 1.596e-05 1.552e-05
## Proportion of Variance 0.000e+00 0.000e+00 0.000e+00 0.000e+00 0.000e+00
## Cumulative Proportion  1.000e+00 1.000e+00 1.000e+00 1.000e+00 1.000e+00
##                            PC60      PC61      PC62      PC63      PC64
## Standard deviation     1.485e-05 1.415e-05 1.358e-05 1.275e-05 1.256e-05
## Proportion of Variance 0.000e+00 0.000e+00 0.000e+00 0.000e+00 0.000e+00
## Cumulative Proportion  1.000e+00 1.000e+00 1.000e+00 1.000e+00 1.000e+00
##                            PC65      PC66      PC67      PC68      PC69
## Standard deviation     1.231e-05 1.158e-05 1.131e-05 1.092e-05 1.006e-05
## Proportion of Variance 0.000e+00 0.000e+00 0.000e+00 0.000e+00 0.000e+00
## Cumulative Proportion  1.000e+00 1.000e+00 1.000e+00 1.000e+00 1.000e+00
##                            PC70     PC71      PC72      PC73     PC74
## Standard deviation     9.678e-06 9.43e-06 9.144e-06 9.038e-06 8.83e-06
## Proportion of Variance 0.000e+00 0.00e+00 0.000e+00 0.000e+00 0.00e+00
## Cumulative Proportion  1.000e+00 1.00e+00 1.000e+00 1.000e+00 1.00e+00
##                            PC75      PC76    PC77     PC78      PC79      PC8
## Standard deviation     8.384e-06 8.112e-06 7.9e-06 7.64e-06 7.074e-06 7.007e-0
## Proportion of Variance 0.000e+00 0.000e+00 0.0e+00 0.00e+00 0.000e+00 0.000e+0
## Cumulative Proportion  1.000e+00 1.000e+00 1.0e+00 1.00e+00 1.000e+00 1.000e+0
##                            PC81      PC82      PC83      PC84      PC85
## Standard deviation     6.512e-06 6.113e-06 5.902e-06 5.721e-06 5.579e-06
## Proportion of Variance 0.000e+00 0.000e+00 0.000e+00 0.000e+00 0.000e+00
## Cumulative Proportion  1.000e+00 1.000e+00 1.000e+00 1.000e+00 1.000e+00
##                            PC86      PC87      PC88      PC89      PC90
## Standard deviation     5.047e-06 4.531e-06 4.399e-06 4.037e-06 3.989e-06
## Proportion of Variance 0.000e+00 0.000e+00 0.000e+00 0.000e+00 0.000e+00
## Cumulative Proportion  1.000e+00 1.000e+00 1.000e+00 1.000e+00 1.000e+00
##                            PC91      PC92      PC93     PC94     PC95
## Standard deviation     3.856e-06 3.664e-06 3.359e-06 2.81e-06 2.51e-06
```

```
## Proportion of Variance 0.000e+00 0.000e+00 0.000e+00 0.00e+00 0.00e+00
## Cumulative Proportion  1.000e+00 1.000e+00 1.000e+00 1.00e+00 1.00e+00
##                            PC96     PC97     PC98      PC99     PC100
## Standard deviation     2.312e-06 2.18e-06 1.96e-06 1.828e-06 1.546e-06
## Proportion of Variance 0.000e+00 0.00e+00 0.00e+00 0.000e+00 0.000e+00
## Cumulative Proportion  1.000e+00 1.00e+00 1.00e+00 1.000e+00 1.000e+00
```

Based on this output, we can say that the first two PC's explains around 90% of the total variation of meat fat content. The second line shows the proportion of variance explained by the 1st, 2nd, 3rd... principal component, while the third line shows the cumulative proportion, i.e. the 1st, then 1st and 2nd, then 1st and 2nd and 3rd ...

- We can also take a look at the scree plot:

```
plot(meat.pca$sdev[1:20],ylab="PCAs Std Dev",xlab="PCA number",type="l")
```



According to the plot, **4 principal components** (elbow at 5th PC) seem adequate to represent the data.

- The `pcr` function (principal component regression) from the `pls` package has useful features for prediction and cross-validation. We can easily use it to calculate the RMSE for the training set and the testing set. In the example below, we use a 10-fold Cross-Validation to select the optimal number of components:

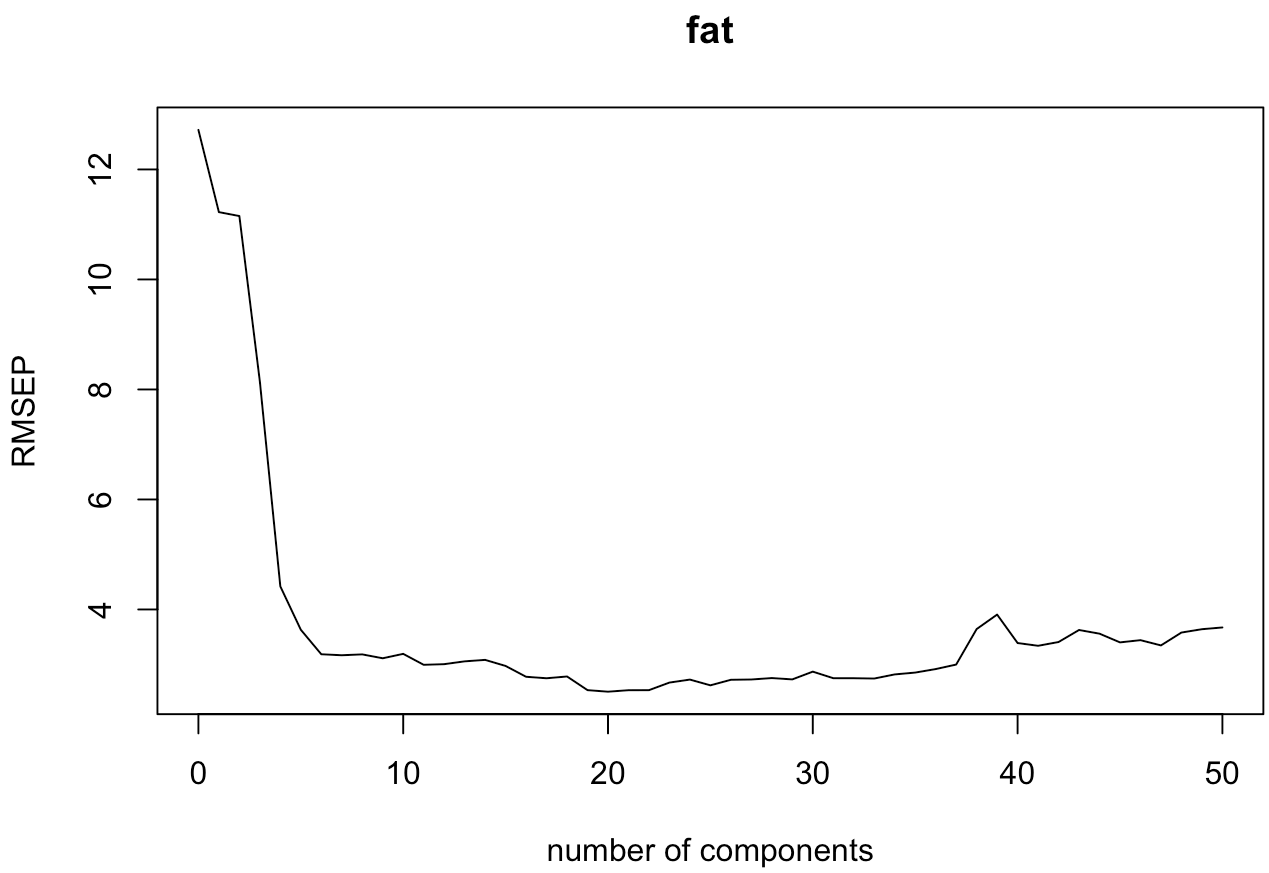
```
set.seed(135)
meat.pcrcv<-pcr(fat~.,data=trainmeat, validation="CV",ncomp=50)
```

Note that we need to set a seed (i.e. set.seed(number)) so that every time that we run the PCR with cross validation we get the same answer. Recall that the CV randomly splits the data in training and testing, therefore to be able to replicate our results every time, we fix the seed and get the same random split every time. We can now compute the training and testing RMSE using `R s RMSEP` function:

```
pcrCV<-RMSEP(meat.pcrcv,estimate="CV")
```

We plot the RMSEP values, and the number of components we select are the ones that minimize the RMSEP.

```
plot(pcrCV)
```




```
which.min(pcrCV$val)
```

```
## [1] 21
```

Here we got that 21 principal components are “optimal” according to this criterion. So, we fit a PCR with 21 components:

```
meatpred<-predict(meat.pcrcv,testmeat,ncomp=21)
rmse(meatpred,testmeat$fat)
```

```
## [1] 2.214545
```

As expected the testing error is larger than the training error.

6.4.2 Ridge Regression

Although the aim of PCR is to reduce dimensionality in the number of predictors, you still have to measure *all* the predictors since each principal component is a *linear combination* of all predictors. **Ridge regression** assumes that after normalization, some of the regression coefficients should not be very large. It is also very useful when you have collinearity and the least squares regression coefficients are unstable.

Ridge Regression

The idea of the method is to use a penalized regression by adding a **penalty** term to the LS minimization problem :

$$\text{minimize } (y - X\beta)^\top (y - X\beta) + \lambda \sum_j \beta_j^2$$

for some $\lambda \geq 0$. The penalty term is $\sum_j \beta_j^2$.

For the method to be more effective, we prefer to *standardize* the predictors first (centered by their means and scaled by their standard deviations) and center the response y as well. A big benefit of the ridge regression is that we can easily obtain closed-form solutions for the β coefficients. Indeed, solving the minimization problem we get:

$$\hat{\beta}_{\text{Ridge}} = (X^\top X + \lambda I)^{-1} X^\top y$$

Note that the extra term λI or **ridge** in the $X^\top X$ matrix. The difference with standard least squares is that the problem solution β minimizes:

$$(y - X\beta)^\top (y - X\beta) \quad \text{subject to} \quad \sum_{j=1}^{p} \beta_j^2 \leq t^2$$

The parameter λ (or t) should be chosen to have *stable* estimates of β.

Note that when $\lambda = 0$ the ridge regression estimation problem reduces to the standard least squares problem, while when $\lambda \to \infty$, $\hat{\beta} \to \mathbf{0}$. The value of λ can be also chosen using automated methods such as *Generalized Cross-Validation* (GCV) (similar to Cross-Validation). The main *disadvantage* of the ridge regression estimators is that they are **biased**.

The `meatspec` Example

We start by running a Ridge Regression in `R`. This is done using the `lm.ridge` function in the `MASS` package:

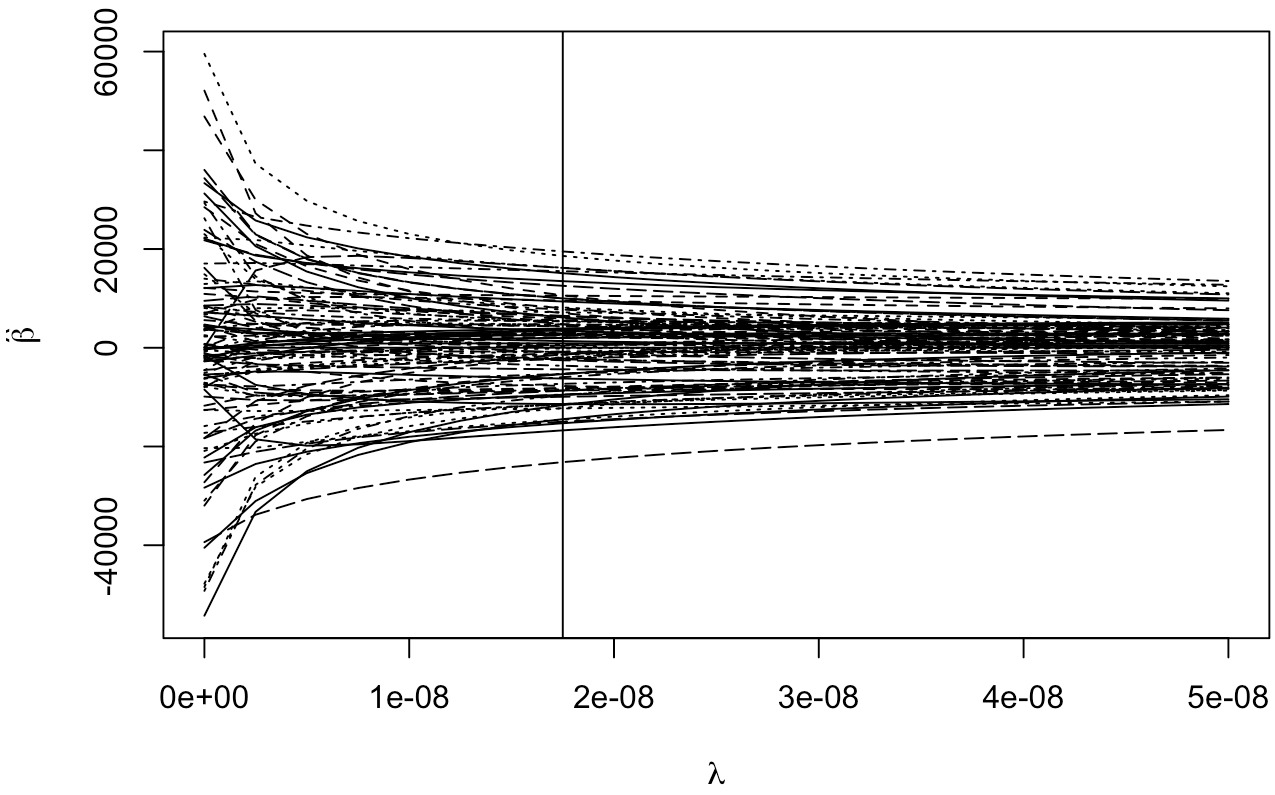
```
require (MASS)
meat.ridge <- lm.ridge(fat~., trainmeat, lambda=seq(0, 5e-8, len=21))
```

Then, we construct a ridge trace plot to determine the appropriate range of λ, using **generalized cross validation (GCV)**:

```
matplot(meat.ridge$lambda, coef(meat.ridge), type="l", xlab=expression(lambda), y
which.min(meat.ridge$GCV)
```

```
## 1.75e-08
##        8
```

```
abline(v=1.75e-08 )
```



We compute the training sample performance

```
fat.pred.train <- cbind(1, as.matrix(trainmeat[,-101]))%*% coef(meat.ridge)[8,]
rmse(fat.pred.train, trainmeat$fat)
```

```
## [1] 0.8024395
```

and the test sample performance:

```
fat.pred.test <- cbind(1, as.matrix(testmeat[,-101]))%*% coef(meat.ridge)[8,]
rmse(fat.pred.test, testmeat$fat)
```

```
## [1] 4.101066
```

As we can see the ridge prediction does not perform well in this example.

Let's take a look at another example as well.

The `fat` Data Example

Age, weight, height, and 10 body circumference measurements are recorded for 252 men. Each man's percentage of body fat was accurately estimated by an underwater weighing technique. The `fat` data set contains 252 observations on 18 variables. `brozek` and `siri` are the Percent body fat measure by the Brozek's equation and Siri's equation and can be used as response variables (one of them).

```
library(faraway)
data(fat)
dim(fat)
```

```
## [1] 252  18
```

```
head(fat)
```

```
##   brozek siri density age weight height adipos  free neck chest abdom   hip
## 1   12.6 12.3  1.0708  23 154.25  67.75   23.7 134.9 36.2  93.1  85.2  94.5
## 2    6.9  6.1  1.0853  22 173.25  72.25   23.4 161.3 38.5  93.6  83.0  98.7
## 3   24.6 25.3  1.0414  22 154.00  66.25   24.7 116.0 34.0  95.8  87.9  99.2
## 4   10.9 10.4  1.0751  26 184.75  72.25   24.9 164.7 37.4 101.8  86.4 101.2
## 5   27.8 28.7  1.0340  24 184.25  71.25   25.6 133.1 34.4  97.3 100.0 101.9
## 6   20.6 20.9  1.0502  24 210.25  74.75   26.5 167.0 39.0 104.5  94.4 107.8
##   thigh knee ankle biceps forearm wrist
## 1  59.0 37.3  21.9   32.0    27.4  17.1
## 2  58.7 37.3  23.4   30.5    28.9  18.2
## 3  59.6 38.9  24.0   28.8    25.2  16.6
## 4  60.1 37.3  22.8   32.4    29.4  18.2
## 5  63.2 42.2  24.0   32.2    27.7  17.7
## 6  66.0 42.0  25.6   35.7    30.6  18.8
```

Many of the variables are highly correlated:

```
cor(fat[,3:18])
```

```
##             density         age      weight      height      adipos        fre
## density  1.00000000 -0.27763721 -0.59406188  0.09788114 -0.71473204 -0.0057487
## age     -0.27763721  1.00000000 -0.01274609 -0.17164514  0.11885126 -0.2379053
## weight  -0.59406188 -0.01274609  1.00000000  0.30827854  0.88735216  0.7921951
## height   0.09788114 -0.17164514  0.30827854  1.00000000 -0.02489094  0.4877984
## adipos  -0.71473204  0.11885126  0.88735216 -0.02489094  1.00000000  0.5471900
## free    -0.00574871 -0.23790534  0.79219519  0.48779841  0.54719009  1.0000000
## neck    -0.47296636  0.11350519  0.83071622  0.25370988  0.77785691  0.6791180
## chest   -0.68259865  0.17644968  0.89419052  0.13489181  0.91179865  0.5929571
## abdom   -0.79895463  0.23040942  0.88799494  0.08781291  0.92388010  0.4956522
## hip     -0.60933143 -0.05033212  0.94088412  0.17039426  0.88326922  0.7034810
## thigh   -0.55309098 -0.20009576  0.86869354  0.14843561  0.81270609  0.6766805
## knee    -0.49504035  0.01751569  0.85316739  0.28605321  0.71365983  0.7036243
## ankle   -0.26489003 -0.10505810  0.61368542  0.26474369  0.50031664  0.5829460
## biceps  -0.48710872 -0.04116212  0.80041593  0.20781557  0.74638418  0.6492953
## forearm -0.35164842 -0.08505555  0.63030143  0.22864922  0.55859425  0.5502771
## wrist   -0.32571598  0.21353062  0.72977489  0.32206533  0.62590659  0.6733589
##                neck      chest       abdom         hip      thigh        knee
## density -0.4729664 -0.6825987 -0.79895463 -0.60933143 -0.5530910 -0.49504035
## age      0.1135052  0.1764497  0.23040942 -0.05033212 -0.2000958  0.01751569
## weight   0.8307162  0.8941905  0.88799494  0.94088412  0.8686935  0.85316739
## height   0.2537099  0.1348918  0.08781291  0.17039426  0.1484356  0.28605321
## adipos   0.7778569  0.9117986  0.92388010  0.88326922  0.8127061  0.71365983
## free     0.6791180  0.5929571  0.49565221  0.70348104  0.6766805  0.70362435
## neck     1.0000000  0.7848350  0.75407737  0.73495788  0.6956973  0.67240498
## chest    0.7848350  1.0000000  0.91582767  0.82941992  0.7298586  0.71949640
## abdom    0.7540774  0.9158277  1.00000000  0.87406618  0.7666239  0.73717888
## hip      0.7349579  0.8294199  0.87406618  1.00000000  0.8964098  0.82347262
## thigh    0.6956973  0.7298586  0.76662393  0.89640979  1.0000000  0.79917030
## knee     0.6724050  0.7194964  0.73717888  0.82347262  0.7991703  1.00000000
## ankle    0.4778924  0.4829879  0.45322269  0.55838682  0.5397971  0.61160820
## biceps   0.7311459  0.7279075  0.68498272  0.73927252  0.7614774  0.67870883
## forearm  0.6236603  0.5801727  0.50331609  0.54501412  0.5668422  0.55589819
## wrist    0.7448264  0.6601623  0.61983243  0.63008954  0.5586848  0.66450729
##               ankle      biceps     forearm      wrist
```

```
## density -0.2648900 -0.48710872 -0.35164842 -0.3257160
## age     -0.1050581 -0.04116212 -0.08505555  0.2135306
## weight   0.6136854  0.80041593  0.63030143  0.7297749
## height   0.2647437  0.20781557  0.22864922  0.3220653
## adipos   0.5003166  0.74638418  0.55859425  0.6259066
## free     0.5829460  0.64929534  0.55027717  0.6733590
## neck     0.4778924  0.73114592  0.62366027  0.7448264
## chest    0.4829879  0.72790748  0.58017273  0.6601623
## abdom    0.4532227  0.68498272  0.50331609  0.6198324
## hip      0.5583868  0.73927252  0.54501412  0.6300895
## thigh    0.5397971  0.76147745  0.56684218  0.5586848
## knee     0.6116082  0.67870883  0.55589819  0.6645073
## ankle    1.0000000  0.48485454  0.41904999  0.5661946
## biceps   0.4848545  1.00000000  0.67825513  0.6321264
## forearm  0.4190500  0.67825513  1.00000000  0.5855883
## wrist    0.5661946  0.63212642  0.58558825  1.0000000
```

We create a data.frame containing all the circumference measurements

```
cfat<-fat[,9:18]
```

We obtain the principal components decomposition using the `prcomp` function to extract the principal components:

```
pcfat<-prcomp(cfat)
names(pcfat)
```

```
## [1] "sdev"     "rotation" "center"   "scale"    "x"
```

```
dim(pcfat)
```

```
## NULL
```

```
summary(pcfat)
```

```
## Importance of components:
##                           PC1     PC2     PC3     PC4     PC5     PC6     PC7
## Standard deviation     15.990 4.06584 2.96596 2.00044 1.69408 1.49881 1.30322
## Proportion of Variance  0.867 0.05605 0.02983 0.01357 0.00973 0.00762 0.00576
## Cumulative Proportion   0.867 0.92304 0.95287 0.96644 0.97617 0.98378 0.98954
##                           PC8     PC9    PC10
## Standard deviation     1.25478 1.10955 0.52737
## Proportion of Variance 0.00534 0.00417 0.00094
## Cumulative Proportion  0.99488 0.99906 1.00000
```

The first principal component (PC1), explains **86.7%** of the total data variation, while the first three components explain *more than 95% of the total variation*.

Since each principal component is a linear combination of the original data, we can inspect the elements of the rotation matrix to understand the PC meaning. This can be challenging in some cases. In particular the first column u_1 is:

```
round(pcfat$rot[,1],2)
```

```
##    neck   chest   abdom     hip   thigh    knee   ankle  biceps forearm   wris
##    0.12    0.50    0.66    0.42    0.28    0.12    0.06    0.15    0.07    0.0
```

This PC weights indicate that the first PC is a combination of all body measures with more weights on central body measurements `abdom`, `chest` and `hip`, and less weight on extremity measurements.

Since the central body measurements are larger and the PC decomposition depends on the variables scale, we should re-scale all variables before applying the PC method:

```
pcfatr<-prcomp(cfat,scale=TRUE)
summary(pcfatr)
```

```
## Importance of components:
##                           PC1     PC2     PC3     PC4     PC5     PC6     PC7
## Standard deviation     2.6498 0.85301 0.81909 0.70114 0.54708 0.52831 0.45196
## Proportion of Variance 0.7021 0.07276 0.06709 0.04916 0.02993 0.02791 0.02043
## Cumulative Proportion  0.7021 0.77490 0.84199 0.89115 0.92108 0.94899 0.96942
##                            PC8     PC9   PC10
## Standard deviation     0.40539 0.27827 0.2530
## Proportion of Variance 0.01643 0.00774 0.0064
## Cumulative Proportion  0.98586 0.99360 1.0000
```

```
round(pcfatr$rot[,1],2)
```

```
##    neck   chest   abdom     hip   thigh    knee   ankle  biceps forearm   wris
##    0.33    0.34    0.33    0.35    0.33    0.33    0.25    0.32    0.27    0.3
```

Now all weights are very similar, and the first principal component represents an approximate average of all original variables explaining 70.2% of the total variation.

The remaining principal components will be *orthogonal* to the first one. The second principal component has negative values for the central body measurements and positive values for the extremity measurements, which indicates a contrats between these two groups of measurements. This will result in high score values (the x in `pcrfat`) for observations with high extremity measures and low central body measurements; and low score values for observations with low extremity measurements and high central body measurements.

```
round(pcfatr$rot[,2],2)
```

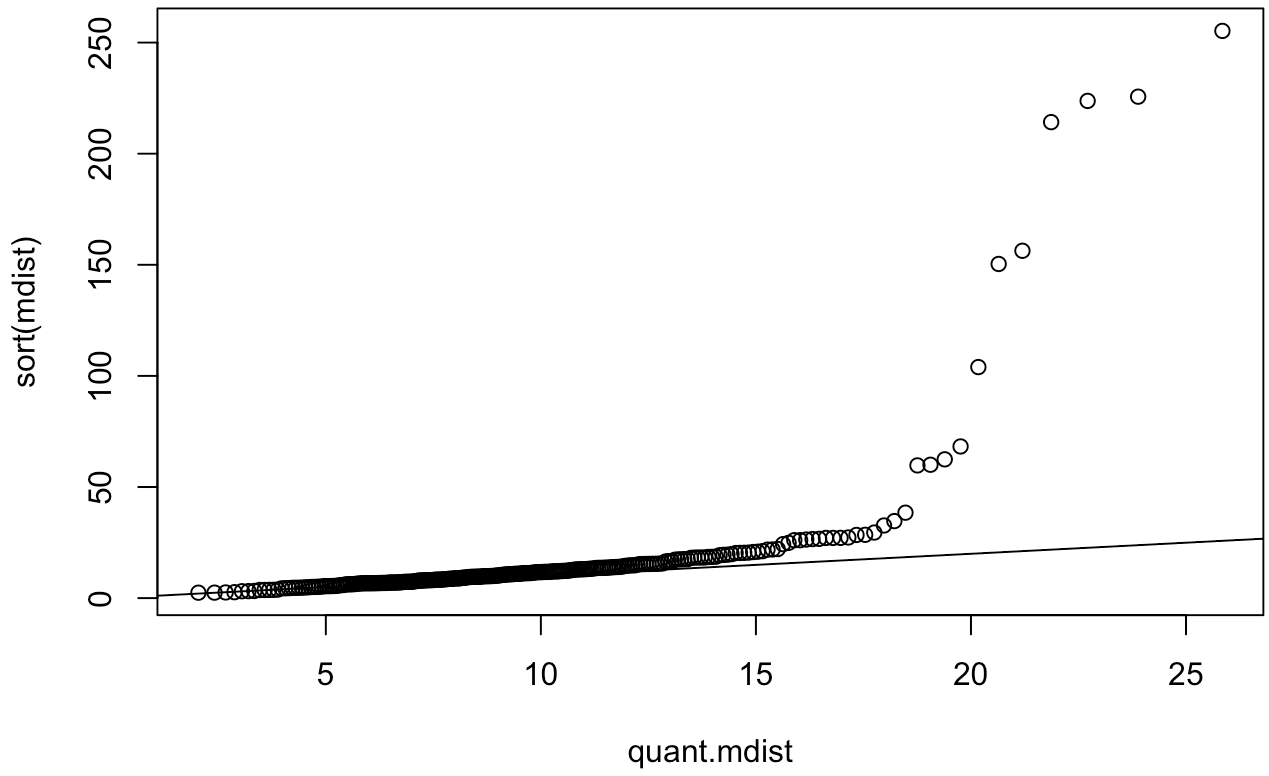
```
##    neck   chest   abdom     hip   thigh    knee   ankle  biceps forearm   wris
##    0.00   -0.27   -0.40   -0.25   -0.19    0.02    0.62    0.02    0.36    0.3
```

```
t(pcfatr$rot[,2])%*%pcfatr$rot[,1] # This product is zero
```

```
##               [,1]
## [1,] -9.714451e-17
```

PCA can be very sensitive to outliers. However, detecting outliers in multiple dimensions is harder. If the m-dimensional data has a multivariate normal distribution, we can use the fact that the `mahalanobis distance` $d^2 = (x-\mu)^\top \Sigma^{-1}(x-mu) \sim \chi^2_m$. We can compare the theoretical quantiles of the χ^2 with the observed mahalanobis distance to the center mean μ for each observation x_i. This can be calculated as the chi-square cdf inverse evaluated in $i/(n+1)$ where i is the order of the distance d_i. We can use the function `cov.rob` from the `MASS` package to estimate the covariance matrix.

```
library(MASS)
sigfat<-cov.rob(cfat)
mdist<-mahalanobis(cfat,center=sigfat$center, cov=sigfat$cov)
n<-dim(cfat)[1]
p<-dim(cfat)[2]
quant.mdist<-qchisq(1:n/(n+1),p)
plot(quant.mdist,sort(mdist)); abline(0,1)
```



We could repeat the PCA after removing these points (you might like to try this).

Now, we will use these PCA results in the regression. First we fit a model to `brozek` using the 10 original predictors, and fit a second model using the first two PCs. Note that the first model fit might be unstable since many predictors are correlated.

Since `abdom` has the largest weight in the first principal component, and `ankle` and `abdom` have the highest weights in the second component, we can select a combination of these two variables.

```
moda<-lm(fat$brozek~.,data=cfat)
summary(moda) #Using all predictors
```

```
##
## Call:
## lm(formula = fat$brozek ~ ., data = cfat)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -9.3159 -2.7435 -0.1584  2.8388 10.5150
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  7.228749   6.214309   1.163  0.24588
## neck        -0.581947   0.208580  -2.790  0.00569 **
## chest       -0.090847   0.085430  -1.063  0.28866
## abdom        0.960229   0.071582  13.414  < 2e-16 ***
## hip         -0.391355   0.112686  -3.473  0.00061 ***
## thigh        0.133708   0.124922   1.070  0.28554
## knee        -0.094055   0.212394  -0.443  0.65828
## ankle        0.004222   0.203175   0.021  0.98344
## biceps       0.111196   0.159118   0.699  0.48533
## forearm      0.344536   0.185511   1.857  0.06450 .
## wrist       -1.353472   0.471410  -2.871  0.00445 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.071 on 241 degrees of freedom
## Multiple R-squared:  0.7351, Adjusted R-squared:  0.7241
## F-statistic: 66.87 on 10 and 241 DF,  p-value: < 2.2e-16
```

```
modpcr<-lm(fat$brozek~pcfatr$x[,1:2]) #Using the first two PCs
summary(modpcr)
```

```
##
## Call:
## lm(formula = fat$brozek ~ pcfatr$x[, 1:2])
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -17.6966  -3.6115  -0.1938   3.4381  20.8732
##
## Coefficients:
##                    Estimate Std. Error t value Pr(>|t|)
## (Intercept)         18.9385     0.3291  57.542   <2e-16 ***
## pcfatr$x[, 1:2]PC1   1.8420     0.1245  14.800   <2e-16 ***
## pcfatr$x[, 1:2]PC2  -3.5505     0.3866  -9.184   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 5.225 on 249 degrees of freedom
## Multiple R-squared:  0.5492, Adjusted R-squared:  0.5456
## F-statistic: 151.7 on 2 and 249 DF,  p-value: < 2.2e-16
```

Some of the coefficients signs for `moda` do not make sense. If we use the two PCs as predictors, the signs of the parameters can be interpreted in terms of the meaning of the PCs.

```
pcfatr$rot[,1:2]
```

```
##               PC1         PC2
## neck    0.3272162 -0.00277082
## chest   0.3385170 -0.27329761
## abdom   0.3341249 -0.39848795
## hip     0.3477307 -0.25464325
## thigh   0.3327963 -0.19141446
## knee    0.3288933  0.02161851
## ankle   0.2465960  0.62464601
## biceps  0.3221654  0.02159095
## forearm 0.2701136  0.36276204
## wrist   0.2988210  0.37724388
```

```
modcon<-lm(fat$brozek~scale(fat$abdom)+ I(scale(fat$ankle)-scale(fat$abdom)), dat
summary(modcon)
```

```
##
## Call:
## lm(formula = fat$brozek ~ scale(fat$abdom) + I(scale(fat$ankle) -
##     scale(fat$abdom)), data = cfat)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -16.134  -3.390  -0.074   3.107  14.873
##
## Coefficients:
##                                     Estimate Std. Error t value Pr(>|t|)
## (Intercept)                          18.9385     0.2794  67.789  < 2e-16 ***
## scale(fat$abdom)                      5.7629     0.3284  17.548  < 2e-16 ***
## I(scale(fat$ankle) - scale(fat$abdom))  -0.9950     0.3140  -3.169  0.00172 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.435 on 249 degrees of freedom
## Multiple R-squared:  0.6752, Adjusted R-squared:  0.6726
## F-statistic: 258.8 on 2 and 249 DF,  p-value: < 2.2e-16
```

This model does a good job when compared with the full model with 10 predictors. The first predictor represents PC1 and the second predictor represents PC2. Scaled predictors work better in this case.

6.4.3 LASSO Regression

LASSO Regression is similar to the Ridge regression in the sense that it minimizes the least squares criterion *subject to a penalty term*. However, the penalty term is different in the case of LASSO.

LASSO Regression

$\hat{\beta}_{\text{LASSO}}$ minimizes:

$$\text{minimize } (y - X\beta)^{\top}(y - X\beta) + \lambda \sum_{j} |\beta_j|$$

for some $\lambda \geq 0$. The penalty term is $\sum_j |\beta_j|$ (L_1 constraint).

In two-dimensions the constraint defines a square, while in higher dimensions it defines a polytope. LASSO is useful when the response can be explained by *few* predictors with zero effect on the remaining predictors (LASSO is similar to a variable selection method). When $\beta_j = 0$ the corresponding predictor is eliminated which is not the case for ridge regression. Therefore, we use LASSO when the effect of predictors is **sparse**. This means that only few predictors will have an effect on the response (e.g. gene expression data) or when number of predictors is large ($p > n$). We typically select t in the constraint $\sum_{j=1}^{p} |\beta|_j \leq t$ by using Cross-Validation (CV). As t increases, the number of predictors increases.

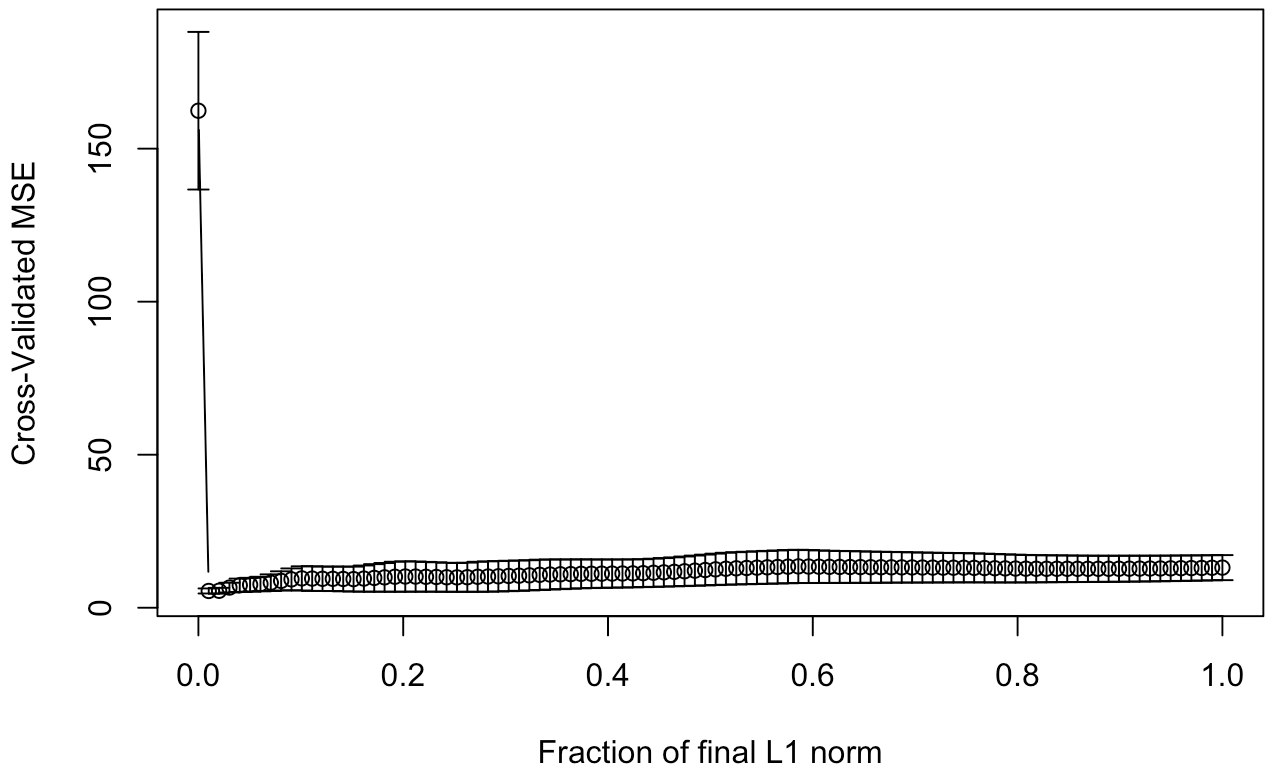
The `meatspec` Example

We start by fitting a LASSO Regression for this data set using the `lars` function in the `lars` library:

```
train.y<-trainmeat$fat
train.x<-as.matrix(trainmeat[,-101])

library(lars)


## Loaded lars 1.2


meatlasso<-lars(train.x,train.y)
```

We will select the parameter t using *cross-validation*:

```
set.seed(123)
cv.ml<-cv.lars(train.x,train.y)
```




```
which.min(cv.ml$cv)
```

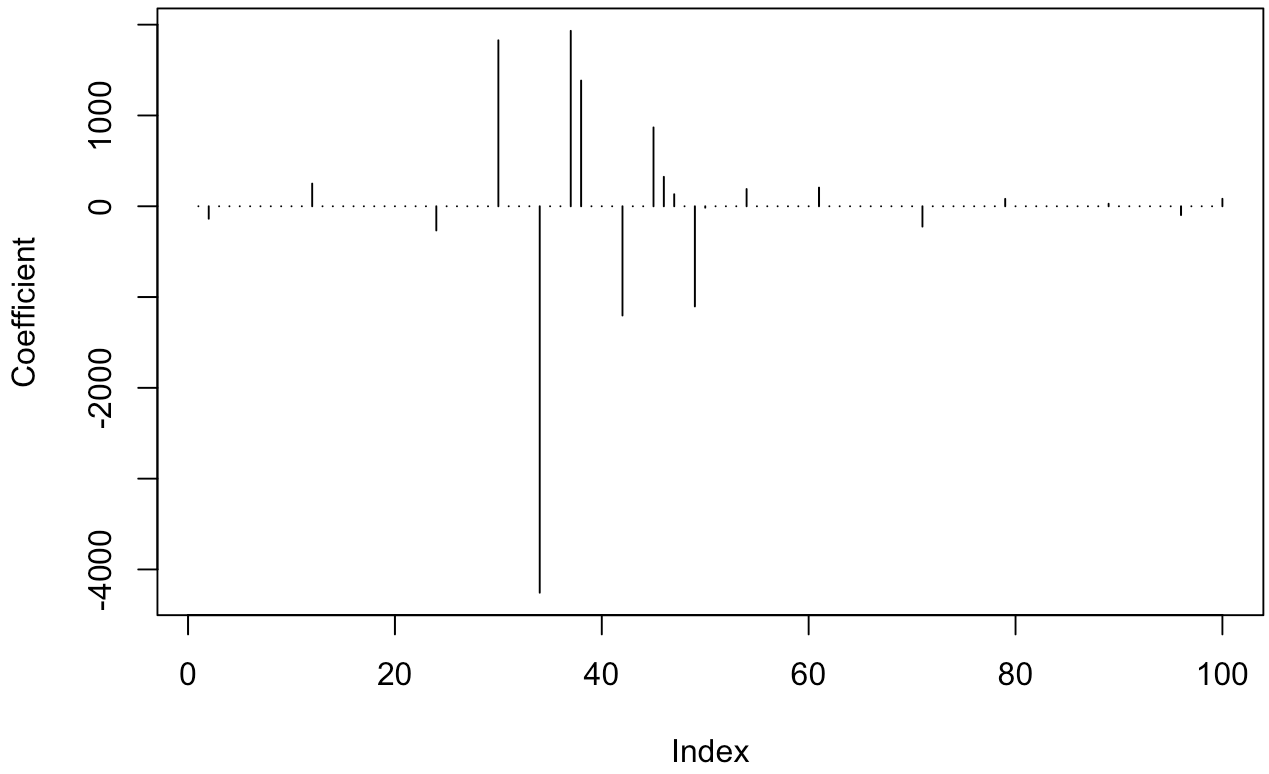
```
## [1] 2
```

```
svm<-cv.ml$index[which.min(cv.ml$cv)]
svm
```

```
## [1] 0.01010101
```

For this choice of t we compute the predicted value of the test data:

```
testx<-as.matrix(testmeat[,-101])

predlasso<-predict(meatlasso, testx, s=svm, mode="fraction")
rmse(testmeat$fat, predlasso$fit)


## [1] 2.132223


predlasso<-predict(meatlasso, s=svm, type="coef", mode="fraction")
plot(predlasso$coef,type="h",ylab="Coefficient")
```



From the coefficients plot we can see that only few frequencies are used to predict the response, which is equivalent to a variable selection method result.

Another example in `R` is also

The `meatspec` Example

We have data from *50 states*:

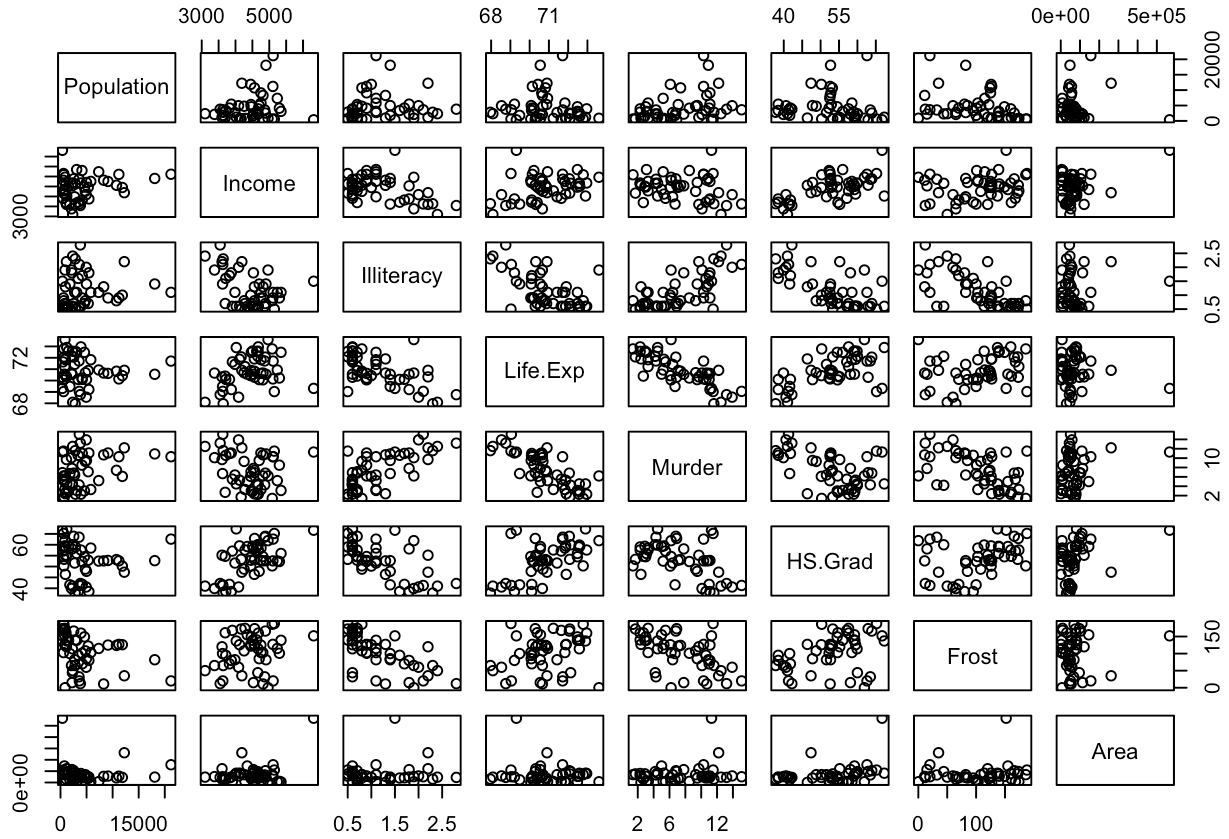
- `Population` : population estimate as of July 1, 1975
- `Income` : per capita income (1974)
- `Illiteracy` : illiteracy (1970, percent of population)
- `Life Exp` : life expectancy in years (1969–71)
- `Murder` : murder and non-negligent manslaughter rate per 100,000 population (1976)
- `HS Grad` : percent high-school graduates (1970)
- `Frost` : mean number of days with minimum temperature below freezing (1931–1960) in capital or large city
- `Area` : land area in square miles

We want to predict *Life Expectancy*. To do so, we use the function `lars` from library `lars` to fit a LASSO regression:

```
library(lars)
data(state)
statedata<-data.frame(state.x77,row.names = state.abb)
head(statedata)
```

```
##    Population Income Illiteracy Life.Exp Murder HS.Grad Frost   Area
## AL       3615   3624        2.1    69.05   15.1    41.3    20  50708
## AK        365   6315        1.5    69.31   11.3    66.7   152 566432
## AZ       2212   4530        1.8    70.55    7.8    58.1    15 113417
## AR       2110   3378        1.9    70.66   10.1    39.9    65  51945
## CA      21198   5114        1.1    71.71   10.3    62.6    20 156361
## CO       2541   4884        0.7    72.06    6.8    63.9   166 103766
```

```
plot(statedata)
```



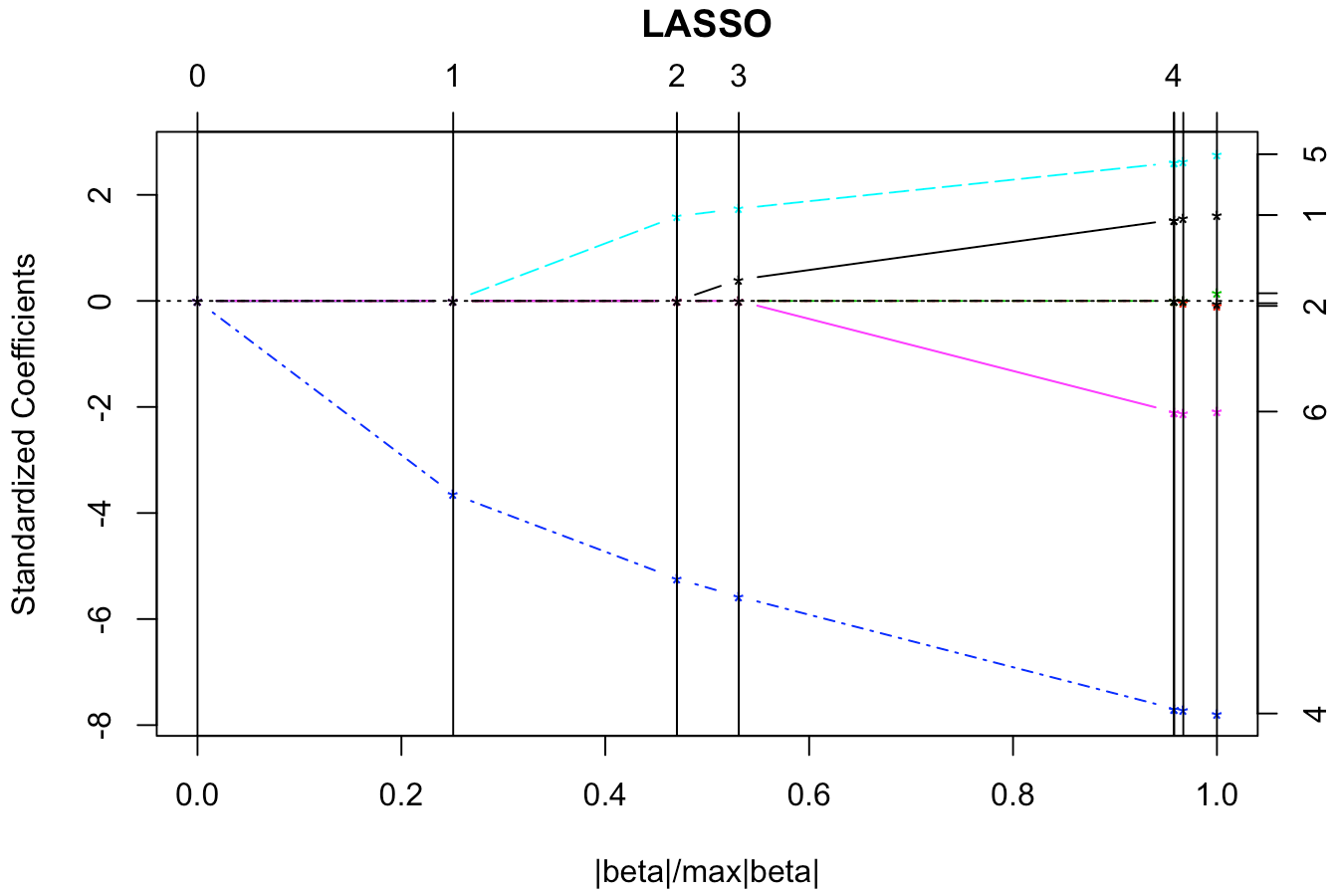
Fit a full model using LS:

```
mod.full<-lm(Life.Exp~.,data=statedata)
summary(mod.full)
```

```
##
## Call:
## lm(formula = Life.Exp ~ ., data = statedata)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.48895 -0.51232 -0.02747  0.57002  1.49447
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  7.094e+01  1.748e+00  40.586  < 2e-16 ***
## Population   5.180e-05  2.919e-05   1.775   0.0832 .
## Income      -2.180e-05  2.444e-04  -0.089   0.9293
## Illiteracy   3.382e-02  3.663e-01   0.092   0.9269
## Murder      -3.011e-01  4.662e-02  -6.459 8.68e-08 ***
## HS.Grad      4.893e-02  2.332e-02   2.098   0.0420 *
## Frost       -5.735e-03  3.143e-03  -1.825   0.0752 .
## Area        -7.383e-08  1.668e-06  -0.044   0.9649
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.7448 on 42 degrees of freedom
## Multiple R-squared:  0.7362, Adjusted R-squared:  0.6922
## F-statistic: 16.74 on 7 and 42 DF,  p-value: 2.534e-10
```

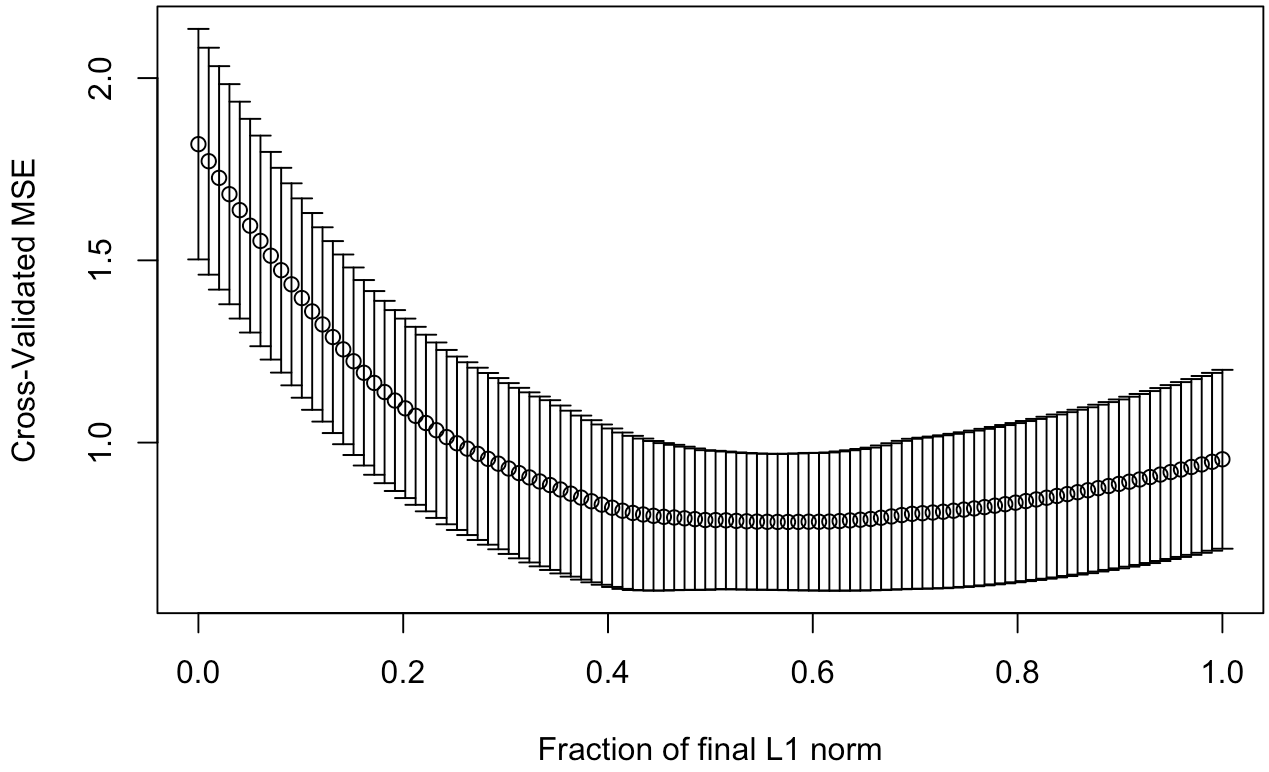
Fit a LASSO regression:

```
modlasso<-lars(as.matrix(statedata[,-4]),statedata$Life.Exp)
plot(modlasso) #standardized coefficients vs t (restriction upper limit)
```



The x axis is the value of the *L1* norm of the coefficients relative to the norm of the LS solution t. As this value increases more predictors enter into the model. For small values of t only `murder` stays active. For larger values of t, variables `HS.Grad` `Population` and `Frost` enter into the model. The optimal value of t can be selected by *Cross-Validation*:

```
set.seed(253)
cvmodlasso<-cv.lars(as.matrix(statedata[,-4]),statedata$Life.Exp)
```




```
sv<-cvmodlasso$index[which.min(cvmodlasso$cv)]
sv
```

```
## [1] 0.5757576
```

We can also extract the LASSO coefficients:

```
#Lasso coefficients
predict(modlasso,s=sv, type="coef", mode="fraction")$coef
```

```
##    Population        Income    Illiteracy        Murder       HS.Grad
##  1.660212e-05  0.000000e+00  0.000000e+00 -2.244076e-01  3.238973e-02
##         Frost          Area
## -6.044026e-04  0.000000e+00
```

```
#use help(predict.lars) for more details
```

and the LS coefficients to compare:

```
coef(lm(Life.Exp~Population+Murder+HS.Grad+Frost,data=statedata))
```

```
##   (Intercept)    Population        Murder       HS.Grad         Frost
##  7.102713e+01  5.013998e-05 -3.001488e-01  4.658225e-02 -5.943290e-03
```

Observe that the LASSO coefficients are **shrunk** with respect to the LS coefficients.

Comparing Ridge Regression and LASSO

LASSO selects a sub-set of predictors (some coefficients equal to zero).Ridge regression performs better when the response is a function of many predictors with coefficients around the same size. LASSO will perform better when a relatively small number of predictors have large coefficients and the rest are very small or equal to zero.Since the number of predictors is never known *a priori*, cross-validation can be used to decide which approach is better for a particular data set.