Chapter 11

Model Building II: Shrinkage Methods

11.1 Principle Components Analysis

Sometimes data are collected on a large number of variables from a single population. Principle components analysis (PCA) seeks to find a new, smaller set of variables (or attributes, or coordinate axes) that describes most of the variation in the data. These new variables will be uncorrelated (so the corresponding axes are mutually orthogonal) and arranged so that the ith variable accounts for the ith most variation in the data. Here is an quick picture of what we mean: run the following R code:

```
set.seed(123)
x <- rnorm(100, 5, 2)
error1 <- rnorm(100, 0, 1)
y <- x + 2*error1
error2 <- rnorm(100, 0, 1)
z <- x + 2*y + 1.5 * error2
library(rgl)
library(scatterplot3d)
plot3d(x, y, z)
```

The point pattern appears elliptical, and pretty flat. Perhaps the point pattern is well-enough described with two new variables instead of the three x, y, and z. The first of these two new variables would point in the direction which describes most of the variation in the data (in this case, the vector (1.98, 3.39, 9.2), and the second would point in a perpendicular direction, accounting for as much of the remaining variation as possible (8.3, -5.57, 0) (double check these vectors....)

Example 50 In the Places Rated Almanac, Boyer and Savageau rated 329 communities according to the following nine criteria: climate and terrain, housing cost, health care & environment, crime, transportation, education, the arts, recreation, and economy.

Install and load the tourr package in R, and examine the Places Rated data:

```
library(tourr)
head(places)
attach(places)
plot(lat ~ long)   #just for fun!
```

Except for housingcost and crime, higher scores are better. Some communities are rated higher in the arts, while others are rated better in areas such as crime rate and educational opportunities.

With such a large number of variables (nine, if you do not include casenumber, longitude, latitude, population, and state number), there are $\binom{9}{2} = 36$ pairwise correlations to consider, and too many graphical displays to consider- for example, there are 84 possible three-dimensional scatterplots to study.

So we want to reduce the number of variables to a few, interpretable linear combinations of the data. Each linear combination will correspond to a *principal component*. There is another useful and related (but different) data reduction technique called *factor analysis*, which we will not address here.

11.1.1 Procedure

Consider the random column vector X of length p

$$\mathbf{X} = (X_1, X_2, \ldots, X_p)^T$$

and its covariance matrix

$$Cov(\mathbf{X}) = \mathbf{\Sigma} = \begin{pmatrix} \sigma_1^2 & \sigma_{12} & \cdots & \sigma_{1p} \\ \sigma_{21} & \sigma_2^2 & \cdots & \sigma_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ \sigma_{p1} & \sigma_{p2} & \cdots & \sigma_{pp} \end{pmatrix},$$

where $\sigma_{ij} = \text{Cov}(X_i, X_j)$ and $\sigma_i^2 = \text{Var}(X_i)$. Let w_{ij} be some weights for $i, j \in \{1, 2, \ldots, p\}$,

$$\mathbf{W} = \begin{pmatrix} w_{11} & w_{12} & \cdots & w_{1p} \\ w_{21} & w_{22} & \cdots & w_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ w_{p1} & w_{p2} & \cdots & w_{pp} \end{pmatrix} = \begin{pmatrix} \mathbf{w_1^T} \\ \mathbf{w_2^T} \\ \vdots \\ \mathbf{w_p^T} \end{pmatrix}$$

with

$$\mathbf{w_i} = (w_{i1}, w_{i2}, \ldots, w_{ip})^T.$$

Define a new, transformed random column vector Z

$$\mathbf{Z} = \mathbf{WX}$$

so that the ith component of the transformed vector is

$$Z_i = \sum_{j=1}^{p} w_{ij} X_j = \mathbf{w_i}^T \mathbf{X},$$

for each $i \in \{1, 2, \ldots, p\}$. Don't confuse the Z_i with standard normal random variables.

Of course $\mathbf{Z}$ is a random vector because $\mathbf{X}$ is, and the covariance of the ith and jth components of $\mathbf{Z}$ are

$$\begin{aligned} \text{Cov}(Z_i, Z_j) &= \text{Cov}\left(\sum_{k=1}^{p} w_{ik} X_k, \sum_{\ell=1}^{p} w_{j\ell} X_\ell\right) \\ &= \sum_{k=1}^{p}\sum_{\ell=1}^{p} w_{ik} w_{j\ell} \sigma_{k\ell} \\ &= \mathbf{w_i}^T \Sigma \mathbf{w_j}. \qquad (11.1) \end{aligned}$$

Setting $i = j$ in (11.1) gives

$$\boxed{\text{Var}(Z_i) = \mathbf{w_i}^T \Sigma \mathbf{w_i}.}$$

The goal is to find a new set of orthogonal coordinate axes along which the distribution of the random vector $\mathbf{X}$ is aligned so that the ith coordinate axis of this new system points in the direction which accounts for the ith most variance of the distribution. We would want to begin by finding the row vector $\mathbf{w_1}^T = (w_{11}, w_{12}, \ldots, w_{1p})$ of weights that maximizes the variance of $Z_1 = \mathbf{w_1}^T\mathbf{X}$ subject to the constraint that $w_i^T w_i = 1$.[1] Then Z_1 would be called the *first principle component.* Forcing the weights to obey $\mathbf{w_i}^T\mathbf{w_i} = 1$ also makes the solutions to our problem unique and also facilitates the geometrical interpretation that Z_i (the ith principle component) is the projection of X_i onto the new unit vector $\mathbf{w_i}$.

So begin by finding the first principle component:

$$\begin{gathered}\text{Find } \mathbf{w_1}^T = (w_{11}, w_{12}, \ldots, w_{1p})\\ \text{that maximizes } \mathrm{Var}(Z_1) = \mathbf{w_1}^T\mathbf{\Sigma}\mathbf{w_1}\\ \text{subject to } \mathbf{w_1}^T\mathbf{w_1} = 1.\end{gathered}$$

We want the second principal component $Z_2 = w_2 \cdot X$ to be the random variable which accounts for as much of the *remaining* variation in the distribution as possible, *after* the first principle component has been identified, under the constraint that the correlation between the first and second components is 0 (so that these first two principle components are orthogonal):

$$\begin{gathered}\text{Find } \mathbf{w_2}^T = (w_{21}, w_{22}, \ldots, w_{2p})\\ \text{that maximizes } \mathrm{Var}(Z_2) = \mathbf{w_2}^T\mathbf{\Sigma}\mathbf{w_2}\\ \text{subject to } \mathbf{w_2}^T\mathbf{w_2} = 1\\ \text{and } \mathrm{Cov}(Z_1, Z_2) = \mathbf{w_1}^T\mathbf{\Sigma}\mathbf{w_2} = 0.\end{gathered}$$

The subsequent principal components are found similarly: they determined by the w_{ij}s that account for as much of the remaining variation as possible, subject to the constraints that the sums of squared coefficients are 1 and they are uncorrelated with the other previously calculated principal components:

$$\begin{gathered}\text{Find } \mathbf{w_i}^T = (w_{i1}, w_{i2}, \ldots, w_{ip})\\ \text{that maximizes } \mathrm{Var}(Z_i) = \mathbf{w_i}^T\mathbf{\Sigma}\mathbf{w_i}\\ \text{subject to } \mathbf{w_i}^T\mathbf{w_i} = 1\\ \text{and } \mathrm{Cov}(Z_i, Z_k) = \mathbf{w_i}^T\mathbf{\Sigma}\mathbf{w_k} = 0 \text{ for all } k < i.\end{gathered}$$

[1] We want $w_i^T w_i = 1$ so that each w_i is a unit vector. Otherwise the w_{1j} values can be chosen arbitrarily large, resulting in no upper bound on the variance for Z_1.

11.1.2 Estimating the w_{ij}s

Maximizing the variances of the Z_i subject to the constraints $\mathbf{w_i}^T\mathbf{w_i} = 1$ can be accomplished with the method of Lagrange multipliers[2]. The result is that the coefficient vectors $\mathbf{w_i}$ that describe the principle components $Z_1, Z_2, \ldots$ turn out to be the eigenvectors of the covariance matrix Σ. Let $\lambda_1, \lambda_2, \ldots, \lambda_p$ denote the eigenvalues of Σ, ordered so that $\lambda_i \geq \lambda_{i+1}$. The variance of the ith principle component equals the ith eigenvalue:

$$\text{Var}(Z_i) = \mathbf{w_i}^T \boldsymbol{\Sigma} \mathbf{w_i} = \lambda_i$$

and the principle components $Z_i = \mathbf{w_i}^T \mathbf{X}$ are uncorrelated (because we chose them that way): $\text{Cov}(Z_i, Z_j) = 0$ for $i \neq j$.

The spectral decomposition theorem allows us to write the covariance matrix as a function of its eigenvalues and eigenvectors.

Theorem 51 *Spectral decomposition theorem; principle axis theorem. Any real, symmetric matrix A can be factored as $A = Q\Lambda Q^T$, where Q is the matrix whose columns are eigenvectors of A and Λ is the matrix with the eigenvalues of A on its diagonal and 0s elsewhere.*

Applying this theorem to the covariance matrix gives

$$\Sigma = \sum_{i=1}^{p} \lambda_i \mathbf{w_i} \mathbf{w_i}^T \approx \sum_{i=1}^{k} \lambda_i \mathbf{w_i} \mathbf{w_i}^T$$

where the approximation with the sum of the first k matrices is useful if the higher indexed eigenvalues $\lambda_{k+1}, \ldots, \lambda_p$ are small.[3]

Define the *total variation* of $\mathbf{X}$ to be the sum of the variances of the individual variables:

$$\text{total variation of } \mathbf{X} = \text{trace}(\boldsymbol{\Sigma}) = \sum_{i=1}^{p} \sigma_i^2 = \sum_{i=1}^{p} \lambda_i.$$

Then we can define the

$$\begin{array}{c}\text{proportion of variation explained}\\ \text{by the } i\text{th principle component}\end{array} = \frac{\lambda_i}{\sum_{j=1}^{p} \lambda_j}$$

and the

$$\begin{array}{c}\text{proportion of variation explained}\\ \text{by the first } k \text{ principle components}\end{array} = \frac{\sum_{i=1}^{k} \lambda_i}{\sum_{j=1}^{p} \lambda_j}. \tag{11.2}$$

[2] You can use the method of Lagrange multipliers to maximize $\mathbf{w_i}^T \boldsymbol{\Sigma} \mathbf{w_i}$ subject to $\mathbf{w_i}^T \mathbf{w_i} = 1$ and show $\boldsymbol{\Sigma} \mathbf{w_i} = \lambda \mathbf{w_i}$ for some λ, which means $\mathbf{w_i}$ is an eigenvector of $\boldsymbol{\Sigma}$ with eigenvalue λ.

[3] This is also useful in the study of factor analysis.

When correlations exist between some of the original X_i-variables (remember this is called multicolinearity in the context of regression), the data may essentially be patterned around a line or plane in a lower number of dimensions. For example, refer to the elliptical/football shaped data pattern example at the beginning of this chapter of notes.

In reality, we will never actually know $\mathbf{\Sigma}$, let alone its eigenvectors or eigenvalues. So we estimate $\mathbf{\Sigma}$ with the sample covariance matrix $\widehat{\mathbf{\Sigma}}$:[4]

$$\widehat{\mathbf{\Sigma}} = \frac{1}{n-1}\sum_{i=1}^{n}(\mathbf{x}_i - \overline{\mathbf{x}})(\mathbf{x}_i - \overline{\mathbf{x}})^T.$$

Then compute the eigenvalues of the sample covariance matrix $\hat{\lambda}_1, \hat{\lambda}_2, \ldots, \hat{\lambda}_p$ and the corresponding eigenvectors $\hat{\mathbf{w}}_1, \hat{\mathbf{w}}_2, \ldots, \hat{\mathbf{w}}_p$. Then estimate the actual principle components using *these* eigenvectors $(\hat{\mathbf{w}}_1, \hat{\mathbf{w}}_1, \ldots)$ as the coefficients:

$$\widehat{Z}_i = \sum_{j=1}^{p} \hat{w}_{ij} X_j, \qquad i \in \{1, 2, \ldots, p\}.$$

We typically refer to the $\hat{Z}_i$s as the principle components even though they are technically estimates for the principle components of the distribution describing the behavior of the random vector $\mathbf{X}$. We typically just retain the first k principle components, where k is the smallest number so that (11.2) is "high enough".

Example 52 Refer to Figure **??**.

```
set.seed(123)
x <- rnorm(10000, 2, .3)
errors <- rnorm(10000, 0, 1)
y <- .5*x + 0.8*errors +2
plot(y ~ x)
windows()
plot(y ~ x, xlim=c(-1.5, 3.3), ylim=c(-1, 6))
xy <- cbind(x, y)
cov(xy)
```

```
           x          y
x 0.08975476 0.04632288
y 0.04632288 0.66589201
```

Get the eigenvalues and eigenvectors:

```
eigen <- eigen(cov(xy))
eigen
```

[4] Take care here: the $\mathbf{x_i}$ is the ith data vector, not the ith component of a vector. The $\overline{\mathbf{x}}$ is the mean vector of the n data vectors: $\overline{\mathbf{x}} = \frac{1}{n}\sum_{k=1}^{n}\mathbf{x_k}$.

```
$values
[1] 0.66959272 0.08605406

$vectors
           [,1]        [,2]
[1,] 0.07963564 -0.99682404
[2,] 0.99682404  0.07963564
```

Now let's draw the eigenvectors on top of the plot of the data in green and purple. Note the eigenvectors (as all vectors do) begin at the origin. We will also draw the these vectors starting at the mean or center of mass of the data (in blue and red) so we can see how they point in the two directions of the most variance. Note the eigenvectors are the same length and perpendicular, but don't appear so because of the scaling.

```
arrows(0, 0, eigen$vectors[1,1], eigen$vectors[2,1],
lwd=3, col="green")
arrows(0, 0, eigen$vectors[1,2], eigen$vectors[2,2],
lwd=3, col="purple")
meanx <- mean(x)
meany <- mean(y)
arrows(meanx, meany, meanx+eigen$vectors[1,1],
meany+eigen$vectors[2,1], lwd=3, col="blue")
arrows(meanx, meany, meanx+eigen$vectors[1,2],
meany+eigen$vectors[2,2], lwd=3, col="red")
```

Now let's try doing this the easy way with the prcomp() function in R:

```
pc <- prcomp(xy)
pc

Standard deviations:
[1] 0.8182865 0.2933497

Rotation:
         PC1         PC2
x 0.07963564 -0.99682404
y 0.99682404  0.07963564

summary(pc)

Importance of components:
                          PC1    PC2
Standard deviation     0.8183 0.2933
Proportion of Variance 0.8861 0.1139
Cumulative Proportion  0.8861 1.0000
```

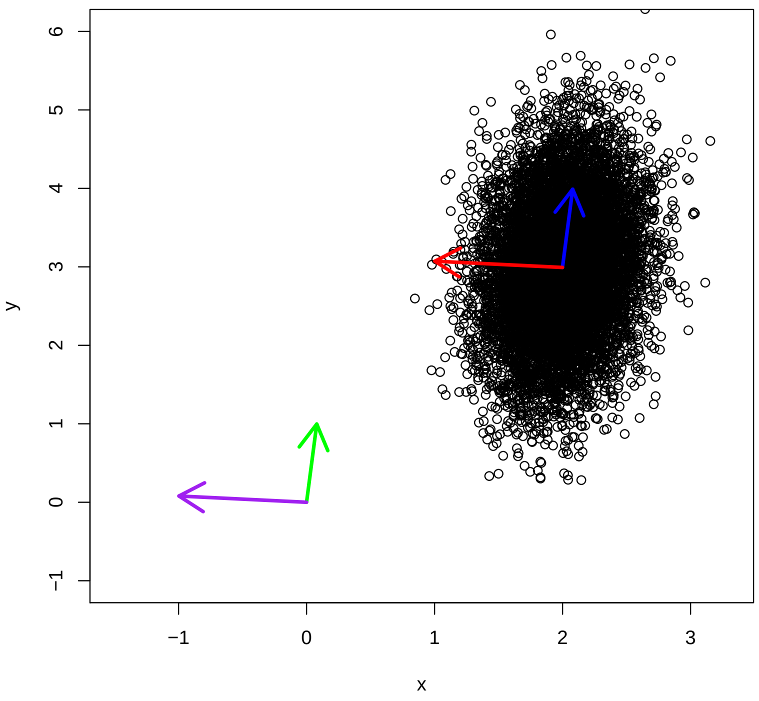

Figure 11.1: Don't be deceived- those vectors are indeed orthogonal and of the same length. They not look that way due to scaling. They are the principle components. I've copied them so that their tails originate at the center of mass of the data set so you can see that they indeed point in the directions of large variance. The eigenvalues are the variances of the data in these directions. The first principle component (in green/blue) points along the direction of most variation, and the corresponding eigenvalue/variance is the larger of the two at 0.66959272. The second eigenvalue (variance) is 0.08605406. When you run the code note your output and graph should be slightly different from this.

11.1.3 Places Rated Example

There are nine quantitative (predictor) variables of interest in columns 1 through 9. We would like to see if we can eliminate some of them by detecting some multicolinearity by using PCA. First retain columns two through ten, since the other columns contain variables we aren't interested in (like longitude and latitude). Also, the data on several variables appear right-skewed (you can make histograms to see), so we'll apply base 10 log transformations on each variable.

```
library(tourr)
head(places)
attach(places)
data <- places[,1:9]

windows()
par(mfrow=c(3,3))
hist(data[,1])
hist(data[,2])
hist(data[,3])
hist(data[,4])
hist(data[,5])
hist(data[,6])
hist(data[,7])
hist(data[,8])
hist(data[,9])

logdata <- log(data, 10)
windows()
par(mfrow=c(3,3))
hist(logdata[,1])
hist(logdata[,2])
hist(logdata[,3])
hist(logdata[,4])
hist(logdata[,5])
hist(logdata[,6])
hist(logdata[,7])
hist(logdata[,8])
hist(logdata[,9])

cov(logdata)
eigen <- eigen(cov(logdata))
eigen
```

The eigenvectors appear in the columns, and are sorted left to right in order of decreasing eigenvalues. Each eigenvector contains the loadings, or weights that are used to multiply the original coordinates of the variables to get the new ones (called *scores*) on the principle components. The eigenvalues are the variances of the mean-centered data projected onto the eigenvectors. Let's look at the first three data points in their new coordinates:

```
head(logdata)
center.scale <- function(x) {
scale(x, scale = FALSE)  ## If TRUE, rescales by st.dev
}
centered.logdata <- center.scale(logdata)
centered.logdata[1:3,] %*% eigen$vectors  #Scores.
```

Again, the numbers in the eigenvectors are the weights, or loadings for the principle components. When the mean-centered data are transformed by the loadings, the result is coordinates of the mean-centered data along the principle components, and the positions along these new axes are the scores. It is not necessary to center the data to get the covariance matrix since the covariance operation involves mean-centering. Therefore it is unnecessary to center the data to obtain the eigenvectors and eigenvalues. However, be careful when obtaining the principle component scores: to do this you must use the mean-centered data.

You can also make a scree plot:

```
number <- seq(1, 9, 1)
scree <- cbind(number, eigen$values)
plot(scree, main="Scree Plot",
xlab="Component Number", ylab="eigenvalue/variance")
lines(scree)
sum(eigen$values[1:4]) /sum(eigen$values[1:9])


[1] 0.9178421


sum(eigen$values[1:5]) /sum(eigen$values[1:9])


[1] 0.9499516


sum(eigen$values[1:6]) /sum(eigen$values[1:9])


[1] 0.9728357
```

The first six principle components account for about 97.28357% of the total variation in the data. The scree plot indicates the first three or four principle components account for quite a bit of the variation. So probably four, five, or six principle components are enough to adequately describe the data.

Now let's use the prcomp() command in R to get the same result.

```
pc <- prcomp(logdata)
pc
summary(pc)
(pc$sdev)^2 #returns the eigenvalues/variances if you want them
```

Compare the above with the output of `eigen`.

You can also use the `princomp` command:

```
pc.new <- princomp(logdata)
head(pc.new)
centered.logdata[1:3,] %*% as.matrix(pc.new$loadings)
### compare with...
pc.new$scores[1:3,]
```

This gives the standard deviations along each principle component (which are the square roots of the eigenvalues), the weights (also called loadings... and careful- these are rounded quite a bit in the output of `head(pc.new)`, and the blanks are not 0s), and the scores (the new coordinates of each observation with respect to the principle components). You can see that the scores are the dot products of the loadings with the original coordinate values.

11.1.4 Interpreting Principle Components

The matrix of eigenvectors is also called the *rotation matrix*, and the values that compose them are often called the *loadings*. Let's use the first five PCs, giving a total variance of almost 95%:

```
pc$rotation  ##gives the loadings/eigenvectors/PCs
fivepcs <- as.matrix(pc$rotation[,1:5])
fivepcs
class(fivepcs)
logdatamatrix <- as.matrix(centered.logdata)
mapped <- logdatamatrix %*% fivepcs
    #This projects the centered log_10 data
        #onto just the first five principle
        #components we're keeping (based on the scree plot
        #or cumulative variance, etc.) This works since the PCs
        #are unit vectors.
```

We want to compute the correlations between the original variables (well, in our case, the base 10 logs of the original variables) and the principle components to see which of the original variables might be related, and to see which ones seem to be associated with which principle components.

```
cor(logdata, mapped)
```

```
                  PC1        PC2       PC3       PC4       PC5
climate      0.189776  0.0176671 -0.207311  0.203870 -0.453384
housingcost  0.543978  0.0197815 -0.204202 -0.256406 -0.215338
hlthcare     0.781632 -0.6052287 -0.143916 -0.016626 -0.020334
crime        0.364840  0.2944431 -0.585486  0.648373  0.010303
transp       0.585236  0.0848904 -0.234244 -0.121381  0.715004
educ         0.393516 -0.2727092 -0.027110 -0.015689  0.123308
arts         0.985400  0.1259213  0.111352  0.022550 -0.013083
recreat      0.519862  0.4016138 -0.518984 -0.506680 -0.147000
econ         0.141775  0.1500496 -0.239039  0.093590 -0.045457
```

We want to find which of the original base 10 log variables are most strongly correlated with each principle component. So maybe we decide correlations of at least 0.5 in magnitude are important enough (in practice you or someone else might decide to use a different threshold).

The first principal component is moderately correlated with four or five of the log 10 variables. It increases with increasing scores/values of log 10 Arts, Health Care, Transportation, Housing Cost and Recreation, suggesting these five criteria may tend to vary together. You can view this first principle component as a measure of Housing Cost and the quality of Arts, Health Care, Transportation, Recreation (remember high values for Housing Cost are bad, but high values for the other four variables are good). The first principal component correlates most strongly with the Arts. Based on the correlation of 0.985 we could say the first principal component primarily measures the Arts, and so that communities scoring highly along this principle component would tend to have a lot of opportunities available including theatre, orchestra, etc.

The second principal component increases with decreasing values in log 10 Health Care, and so high values of this component could correspond to low availability or satisfaction with doctors, hospitals, pharmacies, etc.

The third principal component increases as Crime and Recreation decrease. This suggests places with high (low) crime tend to have more (less) or better (worse) recreation facilities.

The fourth and fifth principle components really do not account for much of the variation in the data, and so we might choose not to include them. However, it does seem that the fourth principle component (like the third) correlates with crime and recreation (although in the opposite way), and the fifth principle component correlates with transportation.

To complete the analysis we might want to make scatterplots of the scores of the principle components. Since three of our principle components seem to account for most of the variation, we could make a 3D plot of the scores on the first three components, or a scatterplot for any pair of components:

```
library(rgl)
library(scatterplot3d)
plot3d(mapped[,1], mapped[,2], mapped[,3])
windows()
plot(mapped[,1], mapped[,2])
```

Each dot in these plots represents one community. One of the dots is far to the extreme right along the first principle component, indicating that community probably has rates well in the Arts, Health Care, Housing Cost, Transportation and Recreation categories. We could easily identify this community if the community names were in the data set (but they are not). But we can still do it:

```
places[which.max(pc.new$scores[,1]),]
```

So do an internet search for the community at latitude 33.432 and longitude -94.052 to see where it is. But note that this community may or may not necessarily have poor Arts, Health Care, etc.: it's just appearing to be among the worst of all 329 communities that were rated in the study.

The dot with second principle component value of about 0.6 indicates that community probably has bad Health Care. That community also scored low on the first principle component, and so is probably not a desirable place to live:

```
max(pc.new$scores[,2])                           # Finds the max
                                                 # score on 2nd
                                                 # prin. comp.
places[which.max(pc.new$scores[,2]),]            # Gets original
                                                 # data for this
                                                 # community.
pc.new$scores[which.max(pc.new$scores[,2]),2]    # Redundant...
pc.new$scores[which.max(pc.new$scores[,2]),1]    # Gets the first
                                                 # prin. comp.
                                                 # score for this
                                                 # community.
```

We may also wish to investigate whether physical locations of communities correspond with high or low levels of a given component. For instance, we could plot the first principle component versus the latitude and longitude:

```
plot3d(mapped[,1], places$lat, places$long)
```

Try rotating this until you see the USA. The third dimension here (along PC1) indicates more or less desirable places to live according to Arts, Health Care, etc.

Once identified, principle components are often treated as dependent variables in regression models (like in PCR, which we will do soon).

11.1.5 Standardizing Variables Using the Correlation Matrix

Running PCA on the raw data (or on transformed data in adjusting for skewness, etc.) puts more emphasis on variables with higher sample variances. That is, variables with the highest sample variances will tend to have high correlations with the first few principal components. This can be undesirable. Observe that the units of the variables can play a major role in the analysis. For example, some observed heights of women might have a sample variance of 6.25 square inches ≈ 0.0434 square feet. So the height variable might be highly correlated with the first principle component if measured in inches, but this correlation might be much lower if height is measured in feet, and even lower if measured in yards or miles. Thus many researchers say PCA should *only* be done on raw data if all variables have the same units of measure. Whatever the case, you should probably only do PCA on the raw data if you wish to give the variables with higher variances more weight in the analysis. It is important to give these things very careful thought.

If the variables either have different units of measurement (i.e., pounds, feet, gallons, etc), or if we wish each variable to receive equal weight in the analysis, then we typically standardize the variables before doing PCA according to the following:

$$z_{ij} = \frac{x_{ij} - \overline{x}_j}{s_j}$$

Then perform PCA on the standardized data (the z_{ij} values).

Note the covariance matrix for the standardized data is the same as the correlation matrix for the original data. Therefore, PCA using the covariance matrix for the standardized data is the same as PCA using the correlation matrix for the original data.

Here's an example of when you might choose to run PCA on the original data. Suppose you are making counts of different species at several locations across the region. You might want more weight on the more common species that are observed. Naturally, the more common species will likely result in higher sample variance of the counts (why?). Then running PCA on the raw data will put more emphasis on the more common species- their correlations with the first principle components will tend to be high. However, if you do PCA on the standardized counts, all species would be weighted equally regardless of how abundant they are and hence, you may find some very rare species entering in as significant contributors in the analysis, and this might not be desirable. Give careful thought whether or not standardizing the variables is a good idea before running PCA.

Example 53 *Places Rated, standardized PCA.* First the "long way"... then using the prcomp() command. We'll still use the base 10 log transform on each variable.

```
means <- colMeans(logdata)
sds <- apply(logdata, 2, sd)
zlogdata <- (logdata - t(replicate(329, means)))/
t(replicate(329, sds))
pc1 <- prcomp(zlogdata)
pc1
```

The easy way with prcomp():

```
pc2 <- prcomp(logdata, center=TRUE, scale = TRUE)
pc2
```

This tells R to center the data (subtract the mean) and rescale so the variance is 1 (divide by the standard deviations), and then run PCA. Note the outputs of the two methods are exactly the same.

The eigenvalues or variances along the principle components are

```
(pc1$sdev)^2
```

```
[1] 3.2977930 1.2135619 1.1055299 0.9072798 0.8606287
[6] 0.5621858 0.4838206 0.3180722 0.2511282
```

indicating the variation proportions accounted for by the components to be

```
(pc1$sdev)^2/sum((pc1$sdev)^2)
```

```
[1] 0.36642144 0.13484021 0.12283665 0.10080887 0.09562541
[6] 0.06246509 0.05375785 0.03534135 0.02790313
```

The first principal component accounts for about 36.6% of the variation, the second about 13.5%, etc. The first four principal components account for about 72% of the variation together, and the first five account for about 82%. You should compare these proportions with those we obtained doing PCA on the original (well, non-standardized, but base 10 log transformed) data. The current analysis requires more principle components to explain the same amount of variation as the original analysis using the covariance matrix on the non-standardized data. This is often the case because prior to standardizing, when data have their original units, one or two of the original variables can show prominently with regard to the variation.

Examination of the scree plot shows a sharp break after the first component, so perhaps the first component is enough. However, this one component only explains about 37% of the variation, so we should consider using more than one principle component.

```
windows()
nums <- seq(1, 9, 1)
eigs <- (pc1$sdev)^2
scree <- cbind(nums, eigs)
cumscree <- cbind(nums, cumeigs)
plot(scree, main="Scree Plot for Standardized PCA",
xlab="Principle Component Number", ylab="Eigenvalues / Variances")
lines(scree)
windows()
```

Or, just do

```
screeplot(pc1, type="lines",col=3)
```

We should also examine the cumulative amount of variance in the data captured by the first i principle components:

```
cumeigs <- matrix(NA, nrow=9, ncol=2)
for (i in 1:9){
cumeigs[i,1] <- sum(eigs[1:i])
cumeigs[i,2] <- cumeigs[i,1]/sum(eigs[1:9])
}
cumscree <- cbind(nums, cumeigs)
plot(cumscree, main = "Cumulative Variance Plot
for Standardized PCA", xlab="Principle Component
Number", ylab="Total Eigenvalues/Variances")
lines(cumscree)
```

A required total variation "threshold" is often specified in advance of running PCA (like 80% or 95%, etc.). This is similar to when you run a hypothesis test and the probability of type 1 error α can be set in advance of running the test, or even prior to collecting data- then compare the p-value to α and choose to reject or not based on which is greater. Anyway, if we take this "threshold" kind of approach with PCA, we simply include as many principle components as needed in order that the sum of the eigenvalues just exceeds that prespecified threshold. On the other hand, we could simply run the PCA and decide how many principle components to use based on the scree plot and the cumulative percentages of variation included (similar to not setting α in a hypothesis test and just looking at the p-value). Good judgement in these approaches often requires experience with PCA and the kinds of variables you're working. It is also important to realize what is at stake with in the situation at hand.

Next, look at the "rotation" output from our PCA analysis. These give the coefficients for the principle components:

```
pc1$rotation
```

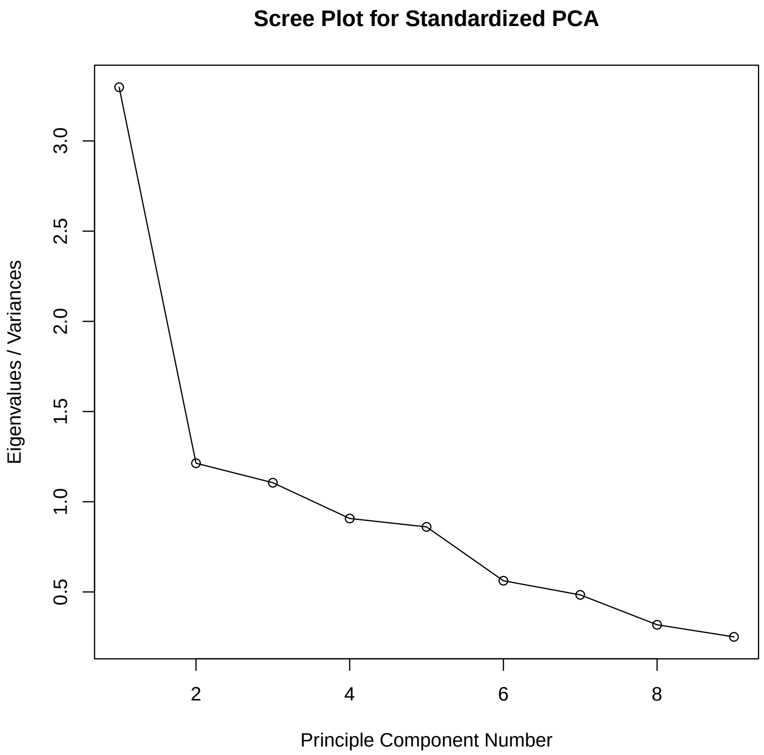

Figure 11.2: Here is a scree plot for the PCA run on standardized variables. Elbows in scree plots typically indicate where we could possibly end our search for components. Here the elbow is at 2, indicating one component *might* be enough.

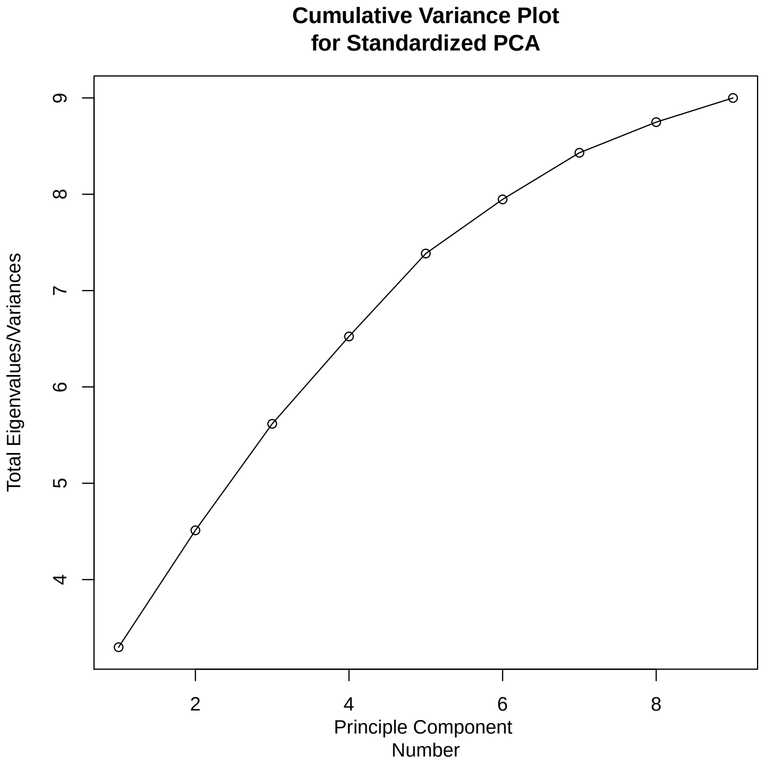

Figure 11.3: Although the elbow in Figure 11.2 indicates one component might be enough, this figure shows we would need to include at least seven components to account for 80% of the variation.

You can see the first principle component is given by

$$Y_1 = 0.15794 \times \text{climate} + 0.38441 \times \text{housingcost} + 0.40991 \times \text{hlthcare} + \cdots$$

Remember these variables (housingcost, etc.) are really standardized versions of the base 10 logs. The magnitudes of the coefficients are the contributions of each variable to that component. Since the data have been standardized, they do not depend on the variances of the corresponding variables and it is easy to compare the relative magnitudes of the variable contributions. For instance, arts makes a contribution of 0.473847 to the first principle component while the contribution due to climate is relatively less: 0.15794. Note the sum of these coefficients is not one, but the sums of the squares of the coefficients are 1s because the principle components are unit eigenvectors.

Figure 11.4 displays all the observations with coordinates in the first two principle components. Cities with high values of the first principle component seem to excel in the arts (such as observations 179, 270, and 213), whereas cities with high values of the second principle component might have a strong economy (like observations 195, 160, and 168).

```
biplot(pc1, cex=0.8, main="Observatations According to First Two
Principle Components")
abline(h=0, v=0, lty = 2, col=8)
```

The biplot is one of the primary tools for analyzing the principle components. The “bi” in biplot refers to the fact that we’re plotting (1) original variable vectors projected onto the (2) new coordinate system described by the principle components. It turns out the correlation between any two variables here is the cosine of the angle between their arrows (the *actual* arrows... not the ones projected here onto the 2D-plane. The longer the arrow, the closer it is to the 2D-plane spanned by the two principle components. Note the correlation between any two principle components is 0 since $\cos(\pi/2) = 0$, as it should, since principle components are orthogonal.

Interpretation of the principal components is based on finding which variables are most strongly correlated with each component. This can be determined by looking at the weights on the PCs, and also by looking at the lengths and directions of the variable projections onto the 2D-plane spanned by the PCs. In looking at the first PC, we might decide the hlthcare and arts weights are large (this is fairly arbitrary, and someone else might want to include additional variables as part of this first PC). If we do, then we take the first principal component as a measure of the quality of Health and the Arts, and perhaps to some extent Housing, Transportation and Recreation. Health increases with increasing values in the Arts. If any of these variables goes up, so do the remaining ones. They are all positively related as they all have positive signs.

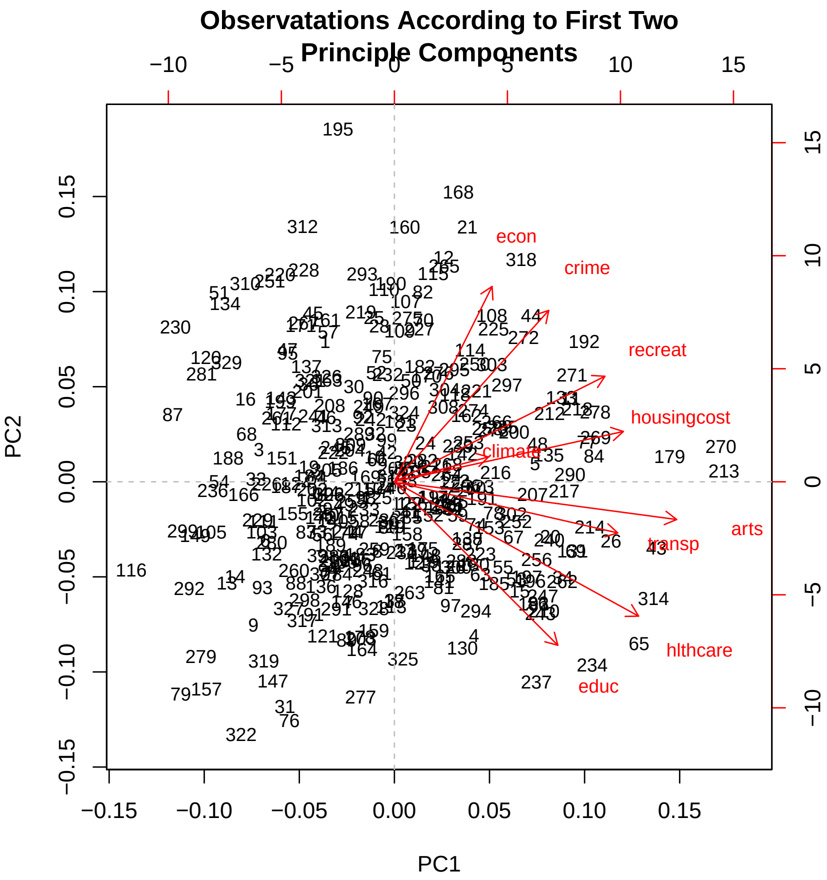
Observatations According to First Two
Principle Components
−10
−5
0
5
10
15
0.15
0.10
0.05
0.00
−0.05
−0.10
−0.15
−0.15
−0.10
−0.05
0.00
0.05
0.10
0.15
PC1
PC2
15
10
5
0
−5
−10
econ
crime
recreat
housingcost
climate
arts
transp
hlthcare
educ

Figure 11.4:

The second principal component might be a measure of the severity of crime, the quality of the economy, and the lack of quality in education. Crime and Economy increase with decreasing Education. Here we can see that cities with high levels of crime and good economies also tend to have poor educational systems.

```
###  PC3 vs PC2
biplot(pc1, choices=c(2,3), cex=0.8, main="Observations According
 to the 2nd and 3rd
Principle Components")
abline(h=0, v=0, lty = 2, col=8)

###  PC4 vs PC3
biplot(pc1, choices=c(3,4), cex=0.8, main="Observations According
 to 3rd and 4th
Principle Components")
abline(h=0, v=0, lty = 2, col=8)

###  PC5 vs PC3
biplot(pc1, choices=c(3,4), cex=0.8, main="Observations According
 to 3rd and 5th
Principle Components")
abline(h=0, v=0, lty = 2, col=8)
```

The third principal component is a measure of the quality of the climate and poorness of the economy. Climate increases with decreasing Economy. The inclusion of economy within this component will add a bit of redundancy within our results. This component is primarily a measure of climate, and to a lesser extent the economy.

The fourth principal component is a measure of the quality of education and the economy and the poorness of the transportation network and recreational opportunities. Education and Economy increase with decreasing Transportation and Recreation.

The fifth principal component is a measure of the severity of crime and the quality of housing. Crime increases with decreasing housing.

This can also be a good time to check on the correlations of the standardized data:

```
library(gclus)
windows()
something = order.single(cor(zlogdata))
cpairs(zlogdata, something)
```

11.1.6 Things to Consider

1. One of the problems with this method is the analysis is not as clean as in other methods, due to so many variables, PCs, loadings, eigenvalues, etc. For example, in the Places Rated example, economy is considered to be significant for both the second and third components. This will lead to an ambiguous interpretation in our analysis. So in addition to scree plots, biplots, and looking at the loadings, another method for deciding how many components to use is to include only the components that give unambiguous results- that is, use principle components such that no variable appears as a significant contribution in more than one of them.

2. Note the primary purpose of PCA is descriptive, not inferential. Your decision about how many components to include should typically be made based on what gives a good, concise description of the data.

3. We use PCA to make decisions about what correlations are important, not necessarily from a statistical hypothesis testing point of view, but rather from a practical one (in the case of the previous example, from an urban-sociological perspective). You must decide what is important in the context of the problem at hand, and this decision may differ across disciplines (for instance, cut-offs for correlation values). In some disciplines such as sociology and ecology the data are inherently 'noisy', and in this case you would probably expect messier or more ambiguous interpretations. For disciplines (such as engineering or science) where precision is important, you might put a higher premium on analysis, and maybe you would want to have very high correlations. To this end, PCA is perhaps more commonly used in sociological and ecological applications and in marketing research.

4. Sometimes the principal components scores will be used as explanatory variables in regression. This is called *principle components regression* (PCR). Sometimes in regression settings you might have a very large number of potential explanatory variables to work with, and you may not have much of an idea as to which ones you might think are important. What you might do is to perform a principal components analysis first and then regress the response onto the principal components. The regression coefficients will be uncorrelated with each other because the principle components are uncorrelated with each other. In this case you actually say how much of the variation in the response variable is explained by each of the individual components.

11.2 Principle Components Regression (PCR)

Principle components regression (PCR) involves performing PCA on the predictor variables, and then regressing the response variable onto the principle components. Here are the steps.

1. Standardize the observations on the response and predictor variables.
2. Perform PCA on the predictor variables, and decide how many components (M) to retain. Below, the zs are the principle components.

$$\begin{aligned} z_1 = \hat{Z}_1 &= \sum_{k=1}^{p-1} w_{1k} x_k \\ z_2 = \hat{Z}_2 &= \sum_{k=1}^{p-1} w_{2k} x_k \\ \vdots \quad \vdots \quad & \vdots \\ z_M = \hat{Z}_M &= \sum_{k=1}^{p-1} w_{Mk} x_k \end{aligned}$$

3. Regress the (standardized) response variable onto the principle components you've chosen:
$$\hat{y} = b_0 + \sum_{\ell=1}^{M} b_\ell^{pc} z_\ell.$$
4. Note the final regression model still models the response in terms of the the original (standardized) variables:
$$\begin{aligned} \hat{y} &= b_0 + \sum_{\ell=1}^{M} b_\ell^{pc} z_\ell \\ &= b_0 + \sum_{\ell=1}^{M} b_\ell^{pc} \sum_{k=1}^{p-1} w_{\ell k} x_k \\ &= b_0 + \sum_{k=1}^{p-1} x_k \sum_{\ell=1}^{M} w_{\ell k} b_\ell^{pc} \\ &= b_0 + \sum_{k=1}^{p-1} b_k x_k \end{aligned}$$
and so one can view the regression coefficients b_k for regressing the standardized response onto the standardized original variables as a weighted sum of the coefficients b^{pc} obtained when regressing the standardized response onto the principle components.

5. Finally, obtain the b_k values and write down your regression model in terms of the original standardized variables. Note these b_k coefficients are *not* the same as the ones you would get by performing least-squares regression with the original standardized predictor variables (if they were, then why would we bother doing PCA?).

There are at least three good reasons for doing PCA or PCR.

1. *Computationally less expensive.* It turns out that with large data sets, PCR is computationally less expensive than regressing onto all the predictors, even though the final model still depends on all the predictors.

2. *Exploratory data analysis.* Performing PCA on the predictors might reveal multi-collinearity among the predictors, helping to inform the statistician which predictors are most important and which could likely be discarded.

3. *Shrinkage.* The principle components are functions of the original predictor variables, so performing PCR does not actually result in a simpler model in the sense of reducing the number of original variables. However, we do consider PCR to fall into the category of *shrinkage* methods because the technique typically reduces the variation in the model coefficients (see example).

Some common misuses and misunderstandings of PCA and PCR include the following.

1. When doing PCA one typically chooses the number of principle components to be far less than the original number of predictor variables, which is why PCA falls into the category of dimensionality reduction techniques. Of course, generally speaking, models with smaller numbers of predictor variables are less likely to overfit the data. Therefore, some practitioners think of PCA as a way to prevent over-fitting. However, PCA should *not* be used as a way to prevent over-fitting. Note that PCA doesn't make use of the response variable, and so it can ignore some valuable information. To mitigate against over-fitting, one should instead consider using a regularization technique such as ridge or lasso regression.

2. Sometimes statisticians and machine learning practitioners decide to automatically run PCA before building their model from a training set, without even thinking about the data. It's actually often a good idea to build a simple model with the raw data before trying PCA, and even without standardizing data so you can more easily identify relationships between variables in their natural units. If that doesn't work well enough, then try PCA.

3. Note that each principle component is actually a function of the original data, and so unless you've used PCA to check collinearity and subsequently

removed some of the predictors, a PCR model will still depend on data from all the original predictor variables.

Example 54 *Fat in Men, adapted from Faraway's Linear Models with R, 2nd ed.* Circumference measurements were made on 252 men. Perhaps we don't need all these variables to predict the response (percentage of body fat = brozek).

```
library(faraway)
windows()
par(mfrow=c(1,3))
data(fat,package="faraway")
plot(neck ~ knee, fat)
plot(chest ~ thigh, fat)
plot(hip ~ wrist, fat)
cfat <- fat[,9:18]
prfat <- prcomp(cfat)
dim(prfat$rot)
dim(prfat$x)
summary(prfat)
round(prfat$rot[,1],2) #Makes numbers easier to read, fit page better
prfatc <- prcomp(cfat, scale=TRUE)
summary(prfatc)
prfatc
```

So after scaling, the proportion of variance in the data captured by the first PC drops to about 70.21% (prior to scaling it was 86.7%). Prior to scaling, the first three PCs accounted for more than 95% of the variation, whereas after scaling, seven PCs are required to account for at least 95% of the variation. It is common to observe scaling PCA to result in requiring more PCs to obtain a minimum required variation proportion. A scree plot suggests two PCs might be enough (two PCs account for 77.5% of the variation).

```
round(prfatc$rot[,1],2)
round(prfatc$rot[,2],2)
windows()
screeplot(prfatc, type="lines", col=3)
```

PCA is sensitive to outliers, so we must check for these. You could think in terms of regression and use the hat values h_i (which help identify high leverage values, which remember are data points that are "far away" in the predictor space), or in terms of the Mahalanobis distances d_i. Actually, these things are related:

$$\begin{aligned} d_i &:= \sqrt{(x_i-\mu)^T\Sigma^{-1}(x_i-\mu)} \\ &= (n-1)(h_i-1/n). \end{aligned}$$

Here, Σ is the covariance matrix, n the number of observations, and the x_i is the ith observation vector.

We use the cov.rob function to obtain robust estimators for the mean vector μ and covariance matrix Σ. That is, the mean and covariance are estimated using only "nice" data values... see the cov.rob documentation for more detail, but this function estimates the mean vector and covariance matrix after removing data points that are "far from the others". We want to do this because we want to measure the distances of points from the middle of the main cluster of points. Potential outliers could bias the mean and covariance estimates that we compare those same outliers to, making them appear less strange than they actually are.

```
require(MASS)
robfat <- cov.rob(cfat)
md <- mahalanobis(cfat, center=robfat$center, cov=robfat$cov)
n <- nrow(cfat);p <- ncol(cfat)
plot(qchisq(1:n/(n+1),p), sort(md),
    xlab=expression(paste(chi^2," quantiles")),
    ylab="Sorted Mahalanobis distances", main="Chi-Square
    Probability Plot")
abline(0,1)
```

The mahalanobis() function returns squared distances, and so if the data actually come from a multivariate normal distribution, the values in md should have approximately a χ^2 with degrees of freedom equal to the number of dimensions. This probability plot indicates about eleven points might be outliers. Removing them will change the PCs:

```
mahal <- order(md)
mahal     # So the largest three md values
          # correspond to observations 31, 86, 39.
newmd <- md[-c(39, 86, 31, 175, 159, 206, 41, 36, 54, 216, 106)]
newn <- n - 11    # update the number of rows
plot(qchisq(1:newn/(newn+1),p), sort(newmd))
```

This plot looks much better. Now PCA...

```
newfatc <- cfat[-c(39, 86, 31, 175, 159, 206, 41, 36, 54,
    216, 106),]
newprfatc <- prcomp(newfatc, scale=T)
summary(newprfatc)
```

You can indeed see the PCs have changed some, and for the better in the sense that they begin accounting for more variation in the data quicker (they're more efficient in this sense). From here on we will use the newfatc and newprfatc for our analysis, whereas the Faraway text uses the data and PCA results without removing outliers (well, high leverage values... they're outliers in the predictor space). You should compare the outputs of the two models. First though, here is the model when we use all ten predictors, high leverage values removed, without PCA:

```
newbrozek <- fat$brozek[-c(39, 86, 31, 175, 159, 206, 41, 36,
    54, 216, 106)]
our.lmoda <- lm(newbrozek ~ ., data=newfatc)
sumary(our.lmoda)
```

Note that removing high leverage points in this case actually resulted in a slight reduction in R^2. However, it is likely that this model has better predictive ability than the one built using the high leverage values. Also, there are clear signs of collinearity here, as R^2 is not small, but many of the predictors show as statistically insignificant. Notice that, as with the output in the Faraway text, that larger abdom corresponds to more body fat, while larger hip measurement corresponds to less body fat. Why would this be the case? Let's now regress newbrozek onto the first two principle components:

```
our.lmodpcr <- lm(newbrozek ~ newprfatc$x[,1:2])
sumary(our.lmodpcr)
```

Note these predictors are highly statistically significant, but R^2 has fallen to 50%. But there is no collinearity (remember PCs are always orthogonal). Now look at the loadings of these two PCs:

```
newprfatc$rot
```

The loadings on the first PC indicate, perhaps, that men with larger attributes tend to have more body fat. The loadings on the second PC indicate men with proportionally more weight in their extremities (neck, ankle, biceps, forearm, wrist) tend to be leaner.

Also note that we haven't really reduced the dimensionality of our data... that is, we have not really gone from ten predictors to two: remember each PC is a linear combination of all ten original predictor variables, so we still may need all this information to model the response well. Also, we have to rely on our experience (in this case, prior experience observing how men carry their weight) to help interpret the PCs. You can decide to throw our all but a few predictors that seem to correlate with the first few PCs. In this case, maybe choose to keep only hip (since it has the largest loading in PC1) and the difference between thigh and wrist (since their respective loadings in PC2 have opposite sign and are largest in magnitude):[5]

```
our.lmodr <- lm(newbrozek ~ scale(newfatc$hip) +
    I(scale(newfatc$thigh)-scale(newfatc$wrist)))
sumary(our.lmodr)
```

Faraway's final model uses abdom and ankle - abdom. If we use these variables, we get an R^2 that is higher than that of the model above, and higher than

[5] Be sure to compare these choices with those from Faraway's text: they're difference! Remember we removed eleven observations due to high leverage.

that of Faraway's model (again, remember we threw out leverage points). Also the coefficients appear more statistically significantly different from 0 (small p-values):

```
our.lmodr.star <- lm(newbrozek ~ scale(newfatc$abdom) +
     I(scale(newfatc$ankle)-scale(newfatc$abdom)))
sumary(our.lmodr.star)
```

So it might be a good idea to use R to build *all* possible models that include one original variable and the difference between two original variables, and sort them according to R^2 and p-values.

Example 55 *Fat and Light, adapted from Faraway's Linear Models with R, 2nd ed.* A Tecator Infratec Food and Feed Analyzer working in the wavelength range of 850 to 1050 nm by the near-infrared transmission principle was used to collect data on 215 observations of finely chopped pure meat. For each observation, the fat content was measured along with a 100-channel spectrum of absorbances. Because determination of the fat content via analytical chemistry is time consuming, it would be nice to build a model to predict the fat content of new observations using the 100 absorbances which can be measured more easily.

```
data(meatspec, package="faraway")
head(meatspec)
```

You can see there are 100 predictor variables (one for each channel). Let's partition the data into a training set (80%) and test set (20%). Then we can build models on the training data and use the test set to compare their predictive abilities with something like SSE_{test} or $RMSE_{\text{test}}$:

```
trainmeat <- meatspec[1:172,]
testmeat <- meatspec[173:215,]
modlm <- lm(fat ~ ., trainmeat)
summary(modlm)
```

This is a lot of output! You can extract R^2:

```
summary(modlm)$r.squared
```

Faraway uses the root-mean-squared-error (RMSE), which he defines to be

$$RMSE = \sqrt{\sum_i (\hat{y}_i - y_i)^2 / n}.$$

Note some people will divide by $n-1$ instead of n, but it really doesn't matter which if you're using this measure to compare models. Also, note that

$$RMSE = \sqrt{SSE/n},$$

so larger SSE gives larger $RMSE$. Dividing by n allows you to compare among models built on different sized data sets. So let's get $RMSE_{\text{train}}$ and $RMSE_{\text{test}}$:

```
rmse <- function(x,y) sqrt(mean((x-y)^2))
rmse(fitted(modlm), trainmeat$fat)
rmse(predict(modlm,testmeat), testmeat$fat)
```

It's not likely all 100 predictors are needed. The step() command does stepwise regression according to AIC. Refer to the online documentation, but omitting the direction argument defaults to backward stepwise (meaning it initializes with all 100 variables in the model):

```
modsteplm <- step(modlm)
summary(modsteplm)
length(modsteplm$coef)
```

So the final iteration of step() reduced the number of predictor variables to 72 (there are 73 coefficients: one is the bias/intercept term). Now get $RMSE_{\text{train}}$ and $RMSE_{\text{test}}$ for this smaller model:

```
rmse(modsteplm$fit, trainmeat$fat)
rmse(predict(modsteplm,testmeat), testmeat$fat)
```

Note the improvement in $RMSE$. Now let's apply PCR.

```
meatpca <- prcomp(trainmeat[,-101]) # Remove response.
<- round(meatpca$sdev,3)    # Easier to look at.
```

Getting a scree plot and cumulative scree plot"

```
windows()
screeplot(meatpca, type="lines", col=3)
cumeigs <- matrix(NA, nrow=100, ncol=2)
nums <- seq(1, 100, 1)
eigs <-(meatpca$sdev)^2
sumeigs <- sum(eigs[1:100])

for (i in 1:100){
cumeigs[i,1] <- sum(eigs[1:i])
cumeigs[i,2] <- cumeigs[i,1]/sumeigs
}

cumscree <- cbind(nums, cumeigs)
```

```
plot(cumscree, main = "Cumulative Variance Plot
for Standardized PCA", xlab="Principle Component
Number", ylab="Total Eigenvalues/Variances")
lines(cumscree)

cumeigs[,2]
```

The first PC accounts for more than 98% of the total variation in the data. Note Faraway says, "...ten times...", but is basing his observation on the standard deviations. We can plot the loadings of the first three PCs vs. the frequency with the matplot() command from the pls package.[6] Note the typo in Faraway: the dashed curve is PC2 and the dotted curve is PC3.

```
matplot(1:100, meatpca$rot[,1:3], type="l", xlab="Frequency", ylab="", col=1)
```

The first PC is nearly uniformly composed of all the frequencies. The second PC indicates contrast between the higher and lower frequencies, and the third PC is more difficult to interpret. Higher numbered PCs are typically more difficult to interpret, but are also typically less important. We can do PCR using what we have done here, or just use the pcr command from the pls package.

```
install.packages("pls")
require(pls)
pcrmod <- pcr(fat ~ ., data=trainmeat, ncomp=50) #stipulate the number of PCs
```

So let's use the first four PCs to predict the response (fat content):

```
rmse(predict(pcrmod, ncomp=4), trainmeat$fat)
windows()
par(mfrow=c(1,3))
plot(modlm$coef[-1],xlab="Frequency",ylab="Coefficient",type="l")
coefplot(pcrmod, ncomp=4, xlab="Frequency",main="")
plot(meatpca$sdev[1:10],type="l",ylab="SD of PC", xlab="PC number")
```

The first plot on the left graphs the coefficients of the original linear model vs. each of the 100 variables (the frequencies). These coefficient values seem to be varying wildly.

The second plot shows the new regression coefficients (but on the original 100 variables) vs. the frequencies from PCR (remember when you do PCA, even if you use only one PC, you still use all the original variables, as the PCs are linear combinations of the original variables and vice-versa). This second plot shows the magnitudes of the new coefficients do not vary nearly as much: now they go from about -7 to 12, whereas before they varied from about -5000 to 6000.

[6] Remember, the 100 frequencies are the original predictor variables, so you could replace the word "frequency" here with "variable". It happens that in this example these 100 variables are linearly ordered.

This is one reason PCR is called a shrinkage method: it shrinks the magnitudes of the model coefficients (another reason might be as in the previous example where inspection of the loadings lead us to choose a small subset of the original variables for a model). Also note the smoothness of this curve is nice.

The third curve is a scree plot with standard deviations along the vertical axis (our previous one used variances). Both plots are useful: remember the total variance partitions as sums of variances along orthogonal axes, and so the first scree plot can be useful in examining overall proportion of the variance explained. However, using the standard deviations along the vertical axis can reveal elbows not visible in the other plot. In this case, we might decide five PCs is the right number. Let's stick with four so that our results match those of Faraway.

Let's check our $RMSE_{\text{test}}$ using PCR with four PCs:

```
rmse(predict(pcrmod, testmeat, ncomp=4), testmeat$fat)
```

You should expect $RMSE_{\text{test}}$ to exceed $RMSE_{\text{train}}$, and it does. We can use the RMSEP() command to help determine a good number of PCs to use: it computes $RMSE_{\text{test}}$ for each number of PCs.

```
windows()
par(mfrow=c(1,2))
pcrmse <- RMSEP(pcrmod, newdata=testmeat)
plot(pcrmse,main="Test RMSE vs. Number of PCs \n using Training and Test Sets")
which.min(pcrmse$val)
pcrmse$val[28]
```

We can also try cross-validation:

```
set.seed(123)
pcrmod <- pcr(fat ~ ., data=trainmeat, validation="CV", ncomp=50)
pcrCV <- RMSEP(pcrmod, estimate="CV")
plot(pcrCV,main="Test RMSE vs. Number of PCs \n by Cross-Validation")
which.min(pcrCV$val)
ypred <- predict(pcrmod, testmeat, ncomp=18)
rmse(ypred, testmeat$fat)
```

In any case, about 20 or 30 PCs seems to give the smallest RMSE, so that is what we would use.

11.3 Ridge Regression

$$\sqrt{\frac{p(1-p)}{n}}$$

Recall that least squares regression estimates the $\beta_0, \beta_1, \ldots, \beta_p$ with the respective values $b_0, b_1, \ldots, b_p$ that minimize

$$\begin{aligned} Q = SSE &= \sum_{i=1}^{n} \left(y_i - b_0 - \sum_{j=1}^{p} b_j x_{ij} \right)^2 && (11.3) \\ &= (Y - X\beta)^T (Y - X\beta), && (11.4) \end{aligned}$$

where X is the $n \times (p+1)$ model matrix with 1s in the first column. *Ridge regression* is done by estimating the model parameters with the b-values that minimize

$$\begin{aligned} Q &= SSE + \lambda \sum_{j=1}^{p} b_j^2 && (11.5) \\ &= (Y - X\beta)^T (Y - X\beta) + \lambda \beta^T I^* \beta, \end{aligned}$$

where

$$I^* = \begin{pmatrix} 0 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \end{pmatrix}.$$

Ridge regression gets its name from the λI^*, which adds a ridge along the diagonal of the $X^T X$ matrix. The summation term in (11.5) is a *shrinkage penalty*, which gets small as the $\beta_1, \beta_2, \ldots, \beta_p$ values get small. The idea is to attempt to obtain a simpler model where the coefficients are not so large, and hopefully many (or most) of them are very small. The *tuning parameter* $\lambda \geq 0$ controls the impact of the shrinkage penalty on the minimization. The value of λ can be determined with cross validation. The sum of the squared parameter estimates is just the square of the L2 norm of the parameter estimates, a natural way to measure size. The lasso method (coming up) uses the L1 norm.

Note the shrinkage penalty is not applied to the estimate of β_0. This is because doing so would shrink the estimate of the mean response at the "center of mass" of the explanatory variable values. Hence the 0 in the top-left entry of the I^* matrix.

When $\lambda = 0$, ridge regression reduces to least-squares. As λ increases from 0, the model coefficients $\beta_i, i \geq 1$ tend toward 0. This is useful because it can be the case that many of these tend to 0, while the ones that are more important for predicting the response do not. This can give clues as to which predictor variables might be more useful in predicting the response, and we might be able to exclude some that are not so useful. If λ gets too high, the model can suffer from high bias.

It is not too difficult to see that setting the derivative of Q with respect to the βs to zero gives the following solution:

$$\hat{\beta} = (X^T X + \lambda I^*)^{-1} X^T Y.$$

Also, it is worth noting that while $X^T X$ might not always be invertible, $(X^T X + \lambda I^*)$ is always invertible for $\lambda > 0$. This can be a way to get around multicollinearity.

Ridge regression (and lasso coming next, as well as other methods) require the glmnet package in R.

Example 56 *Hitters.*

```
library(ISLR)
Hitters <- na.omit(Hitters)
head(Hitters)
x <- model.matrix(Salary~., Hitters)[,-1]
y <- Hitters$Salary
head(x)
```

The function model.matrix() creates an "X" matrix from the data. Note we've then left out the Salary column in the same step. Note also this function converts categorical variables to coded/indicator ones. This is important because glmnet() only accepts quantitative values.

```
library(glmnet)
grid <- 10^seq(10, -2, length = 100)
ridge.mod <- glmnet(x,y, alpha = 0, lambda=grid)
```

When alpha = 0, glmnet() performs ridge regression; alpha = 1 triggers lasso. Also, the glmnet() function standardizes each variable by default so they are on the same scale. You can turn this option off with standardize = FALSE. A vector of ridge regression coefficients is stored for each value of λ.

```
dim(coef(ridge.mod))
```

Large tuning parameter values should correspond to smaller parameter estimates (so their L2 norm is smaller). For example, let's compare the parameter estimates and L2 norms for $\lambda = 46415.89$ and $\lambda = 2848.036$ (these are the 45th and 55th λ values in the grid, respectively).

```
ridge.mod$lambda[45]
ridge.mod$lambda[55]
coef(ridge.mod)[,45]
sqrt(sum(coef(ridge.mod)[-1,45]^2)) # L2 norm
coef(ridge.mod)[,55]
sqrt(sum(coef(ridge.mod)[-1,55]^2)) # L2 norm
```

Split the set of observations into a training set and a test set in order to estimate the ridge regression test error:

```
set.seed(1)
train=sample(1:nrow(x), nrow(x)/2)
test=(-train)
y.test=y[test]
ridge.mod=glmnet(x[train,], y[train], alpha=0, lambda=grid, thresh=1e-12)
ridge.pred=predict(ridge.mod, s=4, newx=x[test,])
mean((ridge.pred-y.test)^2)   # This is the test MSE when
                    #lambda = 4.
mean((mean(y[train])-y.test)^2)  #This gets the test MSE
            # for a model with only an intercept term
            # built from the training set.
ridge.pred=predict(ridge.mod, s = 1e10, newx=x[test,]) #alternatively...
mean((ridge.pred-y.test)^2)
```

So $\lambda = 4$ results in lower test MSE than a model with just the intercept term. We can also compare to the full model by setting $\lambda = 0$:

```
ridge.pred=predict(ridge.mod, s=0, newx=x[test,])
    #May need ‘‘exact=T’’ else R interpolates
mean((ridge.pred-y.test)^2)
lm(y~x, subset=train)
predict(ridge.mod, s=0, exact=T, type = "coefficients")[1:20,]
```

We should really use cross-validation to choose the tuning parameter value (instead of just guessing a value like $\lambda = 4$). The cv.glmnet() function performs ten-fold cross-validation by default, which you can change with folds=9, etc.

```
set.seed(1)
cv.out = cv.glmnet(x[train,], y[train], alpha=0)
plot(cv.out)
bestlam=cv.out$lambda.min
bestlam
```

The test MSE associated with this error is

```
ridge.pred <- predict(ridge.mod, s = bestlam, newx=x[test,])
mean((ridge.pred-y.test)^2)
```

Finally, refit the ridge regression model on the full data set using λ chosen by cross validation and examine the coefficient estimates. Note that none of them are zero.

```
out=glmnet(x, y, alpha=0)
predict(out, type="coefficients", s = bestlam)[1:20,]
```

11.4 Lasso

Unlike ridge regression, the *lasso* can lead to sparse models, having fewer than $p-1$ predictors. This is because the lasso coefficients $\hat{\beta}^L_\lambda$ are chosen to minimize

$$SSE + \lambda \sum_{j=1}^{p-1} |\beta_j|, \tag{11.6}$$

which often results in $\hat{\beta}^L_\lambda$ values of exactly 0. The only difference between the objective functions (11.5) and (11.6) is the L2 norm has been replaced with the L1 norm in the penalty term.

Example 57 *Hitters, ISLR.*

```
lasso.mod=glmnet(x[train,], y[train], alpha=1, lambda=grid)
plot(lasso.mod)
```

Apply cross-validation to arrive at a value for the tuning parameter λ:

```
set.seed(1)
cv.out <- cv.glmnet(x[train,], y[train], alpha=1)
windows()
plot(cv.out)
bestlam = cv.out$lambda.min
lasso.pred = predict(lasso.mod, s=bestlam, newx=x[test,])
mean((lasso.pred - y.test)^2)
```

This MSE is lower than that of the null and least-squares models, and similar to that of the one chosen by ridge regression with cross validation. A major advantage of lasso over ridge regression is several of the coefficients are 0:

```
out = glmnet(x, y, alpha=1, lambda=grid)
lasso.coef=predict(out, type = "coefficients", s=bestlam)[1:20,]
lasso.coef
```

11.5 Partial Least-Squares

The PCR approach uses an unsupervised approach in determining the principle components, meaning it identifies the directions that best describe the variation of the data in the predictor space, without using the observations of the response variable Y. This is a shortcoming of PCR because although the principle components point in the orthogonal directions of most variation in the predictor space we cannot assume these directions are also the best ones for predicting the response. Partial least squares (PLS) is a supervised learning alternative to PCR.

As with PCR, the first step is to standardize the $p-1$ predictor variables (remember we use $p =$ number of parameters) and also standardize the response. The first PLS component Z_1 is computed by setting each ϕ_{j1} in

$$Z_1 = \sum_{j=1}^{p-1} \phi_{j1} X_j \tag{11.7}$$

to the corresponding coefficient in the simple linear regression of the response Y onto the individual predictor X_j. As you know, the ϕ_{j1} is proportional to the correlation between X_j and Y[7]. Thus the PLS procedure puts more weight on predictors in (11.8) that are more correlated with the response.

To obtain the second PLS direction Z_2, each (standardized) predictor variable is regressed onto the first component Z_1, and the residuals for each of these $p-1$ models are computed. These residuals represent the information in the predictor variables that is not accounted for by Z_1. Likewise, the (standardized) response is regressed onto the first component Z_1, and these residuals represent the remaining information that the first PLS component neglects to explain about the response. Then these residuals (instead of the original predictors and response) are used to obtain the weights for the second component Z_2 the same way the weights were obtained for Z_1: each set of predictor residuals is regressed onto the residuals for the response, and the resulting model coefficients are the weights for the second component: $\{\phi_{j2},\ j \in \{1, 2, \ldots, p-1\}\}$:

$$Z_2 = \sum_{j=1}^{p-1} \phi_{j2} X_j \tag{11.8}$$

The process is repeated to obtain M PLS components $(Z_1, Z_2, \ldots, Z_M)$. Then use least squares to fit a linear model for predicting the original (standardized) response Y with the $Z_1, Z_2, \ldots, Z_M$, as we do with PCR. The number of components M is chosen by cross-validation. The interested reader should refer to

[7] Recall the slope in simple linear regression equals the correlation coefficient multiplied by the ratio of the standard deviation of the response over the standard deviation of the predictor. If the variables are standardized so they have standard deviation 1, then these slopes exactly equal the linear correlation coefficients.

[8] for a more detailed treatment of this procedure.

Partial least squares tends to explain the response better than PCR, but does not fit the predictor variables as well as PCR. In practice, PLS often performs no better than PCR or ridge regression. Although the PLS procedure can result in models with lower bias (bias in the sense of under-fitting), these models may suffer from higher variance (over-fitting).

Example 58 *Meat spectroscopy, Faraway.*

```
set.seed(123)
plsmod <- plsr(fat ~ ., data=meatspec[1:172,], ncomp=50,
    validation="CV")
coefplot(plsmod, ncomp=4, xlab="Frequency")
plsCV <- RMSEP(plsmod, estimate="CV")
plot(plsCV,main="")
ypred <- predict(plsmod,ncomp=15)
rmse(ypred, trainmeat$fat)
ytpred <- predict(plsmod, testmeat, ncomp=15)
rmse(ytpred, testmeat$fat)
```

11.6 Model Building Strategies

There are many approaches to model building. Here is some advice.

1. Decide the reason for building a regression model. Will the main goal be to

   (a) use the model to predict a response based on future observations on a set of predictor variables, and determine those predictor variables?

   (b) build a model based on "known" theoretical relationships between a response and predictor variables?

   (c) control a response by adjusting predictor variable values?

   (d) explore relationship strengths between a response and some predictors variables?

   (e) summarize a large data set by a single equation?

2. Decide to run a pilot study, if appropriate and time-permitting.

3. Decide what your predictor variables and response variable should be, and collect and clean the appropriate data.

4. Explore the data. Identify outliers and influential observations, potential errors, and missing values. Study bivariate relationships to identify other potential outliers or influential observations and to suggest possible transformations. Identify possible multi-collinearities. Use PCA. This step is very important. Failure to exercise due diligence on data exploration on the front end can lead to wasted effort, money, etc.

5. Use training/validation approaches when possible. Split the original data set into a training and validation set, or prepare to run some sort of cross-validation (like k-fold).

6. Use training/validation or cross-validation approaches along with any of the following to identify several good models that meet your needs:

   (a) best-subsets regression;

   (b) stepwise regression (if there are too many potential predictor variables for best-subsets);

   (c) ridge-regression;

   (d) lasso.

7. Always remember to use diagnostics and transformations to check on and remediate for violations in model assumptions. If none of the models provide a good fit, or assumptions are violated, try other things, like collecting more data, identifying different predictors, applying other transformations, adding higher order terms, weighted-least-squares, or formulating a different type of model.

8. Remember, there is not necessarily one "best" model for a given data set.

11.7 Homework

1. Perform PCA on the Iris data. http://www.instantr.com/2012/12/18/performing-a-principal-component-analysis-in-r/ To view the dataset, simply type `iris` at the R prompt. We will not attempt to build a regression model for this dataset because the response (Species) is categorical, and so linear regression won't work. Let's just explore the nature of the relationships between the predictor variables. Run PCA (not PCR) on the variables Sepal.Length, Sepal.Width, Petal.Length, and Petal.Width.

   (a) List the eigenvalues in order from highest to lowest, along with the percentage of variation captured by each principle component.

   (b) What is the total variation captured by the first component? What is the total variation captured by the first two components? The first three? All four?

   (c) Make a scree plot. How many principle components do you think are enough to adequately describe the variation in the data?

   (d) What do the loadings for the components indicate? Be specific.

   (e) What do the scores for the observations tell you?

2. Another way to describe the lasso method is that it estimates the regression coefficients by choosing them to be the values of the $b_j,\ j \in \{0, 1, \ldots, p-1\}$ by minimizing

$$\sum_{i=1}^{n}\left(y_i - b_0 - \sum_{j=1}^{p-1} b_j x_{ij}\right)^2 \quad \text{subject to} \quad \sum_{k=1}^{p-1} |\beta_k| \leq s$$

   for some number s. For parts (a) through (f), indicate which of the following occurs and justify your answer.

   i. remain constant.

   ii. monotonically increase.

   iii. monotonically decrease.

   iv. initially increase, then decrease.

   v. initially decrease, then increase.

   (a) As s increases from 0, the training SSE will...

   (b) As s increases from 0, the training R^2 will...

   (c) As s increases from 0, the test or validation SSE will...

   (d) As s increases from 0, the test or validation R^2 will...

   (e) As s increases from 0, the squared bias will...

   (f) As s increases from 0, the variance will...

3. Consider estimating regression coefficients by choosing the $b_j,\ j \in \{0, 1, \ldots, p-1\}$ that minimize

$$\sum_{i=1}^{n}\left(y_i - b_0 - \sum_{j=1}^{p-1} b_j x_{ij}\right)^2 + \lambda \sum_{j=1}^{p-1} b_j^2$$

for fixed λ. For parts (a) through (f), indicate which of the following occurs and justify your answer.

    i. remain constant.
    ii. monotonically increase.
    iii. monotonically decrease.
    iv. initially increase, then decrease.
    v. initially decrease, then increase.

    (a) As λ increases from 0, the training SSE will...
    (b) As λ increases from 0, the training R^2 will...
    (c) As λ increases from 0, the test or validation SSE will...
    (d) As λ increases from 0, the test or validation R^2 will...
    (e) As λ increases from 0, the squared bias will...
    (f) As λ increases from 0, the variance will...

4. Load and read the documentation for the College data set from the ISLR package. We want to build a model to predict the number of applications received using the other variables.

    (a) Split the data set into a training set and a validation/test set, approximately 70%, 30%, respectively.
    (b) Fit a linear least-squares regression model on the training set. Compute the test MSE and test R^2.
    (c) Fit a ridge regression model on the training set. Use cross-validation to choose the tuning parameter λ. Give the test MSE and test R^2.
    (d) Fit a lasso regression model on the training set. Use cross-validation to choose the tuning parameter λ. Give the test MSE and test R^2.
    (e) Fit a principle components regression model on the training set, and use cross-validation to choose the number of principle components. Give the test MSE and test R^2, and the number of principle components.
    (f) Fit a partial least squares regression model on the training set, and use cross-validation to choose the number of new model features. Give the test MSE and test R^2, and the number of new features used in the model.

(g) Compare the five models. Which ones seem better? Is there much difference between the test R^2 and test MSE values? How well do these models predict the number of college applications?

5. Prove the form of the ridge regression coefficients:

$$\hat{\beta} = (X^T X + \lambda I^*)^{-1} X^T Y.$$