5 PQHS 471 Notes Week 5

5.1 Week 5 Day 1

ISLR Chapter 6 covers various ways to "contain" linear regression, which can be too flexible when there are many features (n/p is low or even $n < p$). Interpretation may also be an issue when there are many features in the model. The techniques introduced in this chapter will be useful in other machine learning methods (e.g., regularization in deep learning).

5.1.1 ISLR 6.1 Subset selection

General strategy: For each k ($k = 1, \ldots, p$), we pick the "best" model using the training set. Then we compare the p "best" models using one of the following approaches:

1. Direct estimates of test error, using a validation set or cross-validation;
2. Indirect estimates of test error, using traditional measures such as C_p, AIC, BIC, adjusted R^2.

Here, "best" means the resulting model has the best performance among all candidate models in that step according to a measure (e.g., RSS, deviance, R^2). Different measures may lead to different "best" models.

- 6.1.1: **Best subset selection**: Pick the "best" model among *all* models with k predictors.
- 6.1.2: Stepwise.
  - **Forward**: At each $k = 1, \ldots, p$, select the predictor that, when added to the currect model, gives the "best" model (best "improvement"). Works even when $n < p$.
  - **Backward**: At each $k = p, \ldots, 1$, select the predictor that, when removed from the currect model, gives the "best" model (least "degradation").
  - These are *greedy* algorithms. But they are "guided" so that they effectively search over more models than those evaluated.

Note: At every step when comparing models to pick the "best" one, the models need to be evaluated on the same set of observations. The measures such as RSS, deviance, and R^2 are not comparable across different sets of observations. This means observations with any missing value have to be removed.

- 6.1.3: Some traditional measures for model comparison.
  - $C_p = \frac{RSS}{\hat{\sigma}_F^2} + 2d - n$. *Lower is better.* (Mallows, 1973)
    - where $\hat{\sigma}_F^2$ is from the full model and d is the number of parameters in the model being evaluated.
    - This definition is equivalent to $\frac{1}{n}(RSS + 2d\hat{\sigma}^2)$.
    - When n is very small, $\hat{\sigma}_F^2$ can be underestimated, leading to a smaller penalty on large d.
  - $AIC = -2\log(\hat{L}) + 2d$. *Lower is better.* (Akaike information criterion)
    - AIC is a penalized log-likelihood.
    - AIC is equivalent to C_p for linear regression with Gaussian errors with a *known* variance.
  - $BIC = -2\log(\hat{L}) + \log(n)d$. *Lower is better.* (Bayesian information criterion)
    - When $n > e^2 = 7.39$, which is almost always true in practice, $\log(n) > 2$ and $BIC > AIC$.
    - BIC puts more penalty on high d and tends to favor models with a smaller d than AIC.
  - $R^2_{adj} = 1 - \frac{RSS/(n-(d+1))}{TSS/(n-1)} = 1 - \frac{n-1}{n-(d+1)}(1 - R^2)$. *Higher is better.*
    - It is an adjustment of $R^2 = 1 - \frac{RSS}{TSS}$.

[Intuition about AIC: Consider two nested models where the reduced model is correct. Then approximately, $2(\log \hat{L}_F - \log \hat{L}_R) \sim \chi^2_{d_F - d_R}$. $AIC_F < AIC_R$ is equivalent to $2(\log \hat{L}_F - \log \hat{L}_R) > 2(d_F - d_R)$. If a model with d_R is the correct model, the (asymptotic) probability of wrongly selecting a richer model with d_F is 0.16 when $d_F = d_R + 1$, 0.09 when $d_F = d_R + 4$, 0.05 when $d_F = d_R + 7$.

For BIC: At $n = 100$, $\log(100) = 4.6$, if a model with d_R is the correct model, the (asymptotic) probability of wrongly selecting a richer model is 0.03 when $d_F = d_R + 1$, 0.01 when $d_F = d_R + 2$. At $n = 500$, $\log(500) = 6.2$, the probability is 0.01 when $d_F = d_R + 1$, 0.002 when $d_F = d_R + 2$.]

The R package `leaps` provides a function `regsubsets()` for subset selection.

```r
library(ISLR)
?Hitters  # n=322
dim(Hitters); names(Hitters); row.names(Hitters)
## 59 player had missing "Salary".  Take a further look.
apply(is.na(Hitters), 2, sum)
group = apply(is.na(Hitters), 1, sum) > 0
par(mfrow=c(4,5))
for(i in names(Hitters))
    boxplot(as.numeric(Hitters[[i]]) ~ group, main=i)

Hitters = na.omit(Hitters); dim(Hitters)  # n=263

library(leaps)
regfit.full = regsubsets(Salary ~ ., data=Hitters, nvmax=19)  # exhaustive search
reg.summary = summary(regfit.full)
names(reg.summary)  ## has Cp, BIC, adj R2

which.min(reg.summary$cp)    # 10 according to Cp/AIC
which.min(reg.summary$bic)   # 6 according to BIC
which.max(reg.summary$adjr2) # 11 according to adj R2
plot(reg.summary$cp, xlab="Number of Variables", ylab="AIC", type='l')
plot(reg.summary$bic, xlab="Number of Variables", ylab="BIC", type='l')
plot(reg.summary$adjr2, xlab="Number of Variables", ylab="Adj R2", type='l')

regfit.fwd = regsubsets(Salary ~ ., data=Hitters, nvmax=19, method="forward")  # forward
regfit.bwd = regsubsets(Salary ~ ., data=Hitters, nvmax=19, method="backward") # backward
```

5.1.2 ISLR 6.2 Shrinkage/regularization

In regularization (ridge, lasso, or other forms), **standardize the features** unless there is a reason not to. Not standardizing the features may allow one variable to have more influence over another. For example, if weight is recorded in kilograms and height in millimeters, then β_{weight} is the effect per kilogram increase while β_{height} is the effect per millimeter increase. As a result, β_{weight} is probably a lot bigger than β_{height}, and β_{weight} would be penalized more than β_{height} without standardization.

- 6.2.1: **Ridge regression** (also called penalized least squares, l_2 regularization):

$$\text{minimize}_{\beta_0,\beta} \sum_{i=1}^{n}(y - \beta_0 - x_i^T\beta)^2 + \lambda \sum_{j=1}^{p} \beta_j^2, \tag{6.5}$$

where $\lambda \geq 0$ and $\beta = (\beta_1, \ldots, \beta_p)$. When $\lambda = 0$, this is least squares. The penalty λ is a tuning parameter. In vector form,

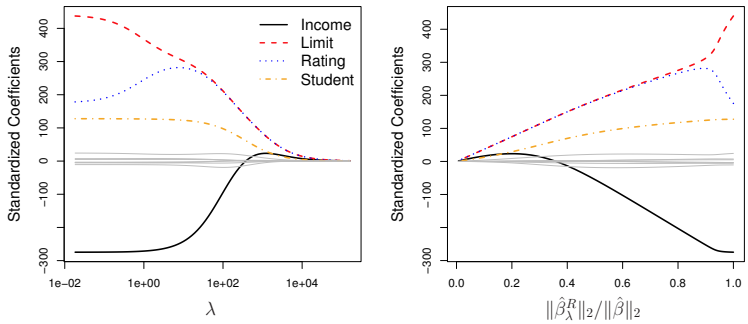
$$\text{minimize}_{\beta_0,\beta}(\|y - \beta_0 - X\beta\|^2 + \lambda\|\beta\|_2^2),$$

where $\|\beta\|_2 = \sqrt{\sum_{j=1}^{p} \beta_j^2}$ is the l_2 norm (i.e., Euclidean distance from the origin) of the vector β, and X is the $n \times p$ design matrix (without the intercept). The solution to (6.5) is $\hat{\beta}_0 = \bar{y}$ and $\hat{\beta}(\lambda) = (X'X + \lambda I)^{-1}X'y$. Note that $\hat{\beta}(\lambda) = (S + \lambda I)^{-1}S\hat{\beta}^{ls}$, where $S = X'X$ and $\hat{\beta}^{ls}$ is the least squares solution.

Selection of λ is often through cross-validation.

Note: Ridge regression was originally introduced to stabilize the computation of matrix inverse $(X'X)^{-1}$ in least squares. (Hoerl and Kennard, 1970)

Figure 6.4 (using the `Credit` data): Left panel shows the coefficients of the predictors, $\hat{\beta}^R_\lambda$, as the penalty λ changes. Right panel shows the coefficients as a function of the fraction of shrinkage as measured by the l_2 norm of $\hat{\beta}^R_\lambda$ relative to that of $\hat{\beta} = \hat{\beta}^{ls}$. The x-coordinates for these two plots have a one-to-one (reverse) relationship.

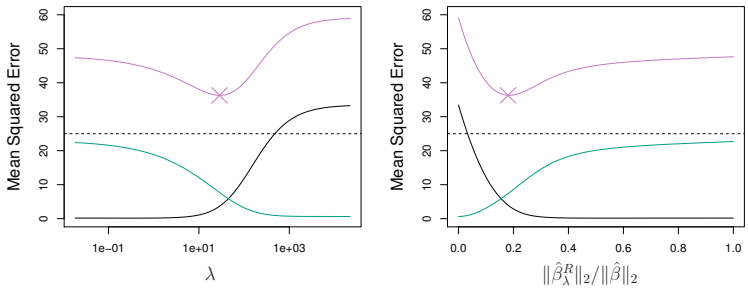



```
Credit = read.table("Credit.csv", header=T, sep=',', row.names=1)
names(Credit)
apply(is.na(Credit), 2, sum)  # no missing data

x = model.matrix(Balance ~ ., Credit)[,-1]  # remove the "intercept" column
y = Credit$Balance
x = scale(x)  # This is important!

library(glmnet)
ridge.mod = glmnet(x, y, alpha=0, lambda=exp(seq(-4, 11, 0.05)))
plot(ridge.mod, xvar='lambda')  # Figure 6.4 left panel
names(ridge.mod)
dim(ridge.mod$beta)  # (11, 301)
l2norm = function(beta) sqrt(sum(beta^2))  # function to calculate l2 norm
l2ls = with(ridge.mod, l2norm(beta[,dim(beta)[2]]))  # l2 norm for beta from LS model (approx.)
with(ridge.mod, matplot(apply(beta, 2, l2norm)/l2ls, t(beta), type='l'))  # Fig 6.4 right panel
```

Figure 6.5 shows a scenario where penalized regression gives a better prediction performance than the traditional least squares at $\lambda = 0$. (Simulation with $n = 50$, $p = 45$)



- 6.2.2: **Lasso**

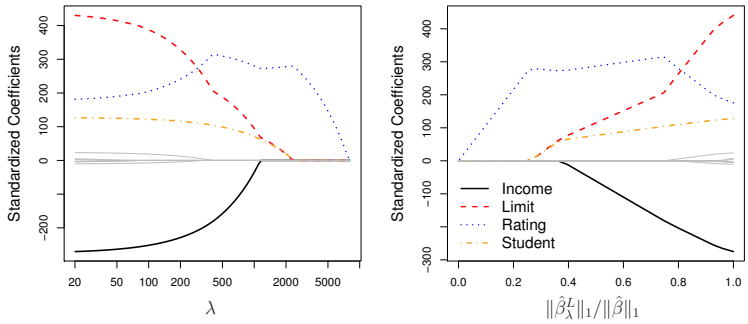
$$\text{minimize}_\beta \sum_{i=1}^{n}(y - \beta_0 - x_i^T\beta)^2 + \lambda\sum_{j=1}^{p}|\beta_j|, \tag{6.7}$$

where $\lambda \geq 0$ and $\beta = (\beta_1, \ldots, \beta_p)$. When $\lambda = 0$, it becomes the least squares. In vector form,

$$\text{minimize}_{\beta_0,\beta}(\|y - \beta_0 - X\beta\|^2 + \lambda\|\beta\|_1),$$

where $\|\beta\|_1 = \sum_{j=1}^{p}|\beta_j|$ is the l_1 norm of the vector β. l_1 norm is sometimes called the Manhattan distance.

Figures 6.6: Coefficients of the predictors for the fitted model, $\hat{\beta}_\lambda^L$, as λ changes (left panel), and as a function of the fraction of shrinkage as measured by the l_1 norm of $\hat{\beta}_\lambda^L$ (right panel).




```
## continue from the last code chunk
lasso.mod = glmnet(x, y, alpha=1, lambda=exp(seq(-1, 6, 0.05)))
plot(lasso.mod, xvar='lambda')  # Figure 6.6 left panel
plot(lasso.mod, xvar='norm')    # Figure 6.6 right panel
```

Figure 6.8: (Same simulation as in Figure 6.5) The left panel shows a scenario where the lasso gives a better prediction performance than the least squares at $\lambda = 0$. The right panel shows a comparison between ridge and lasso. In this simulation setting, they have similar prediction performance and similar profile, with the same "best" model, although the ridge regression model is slightly better.

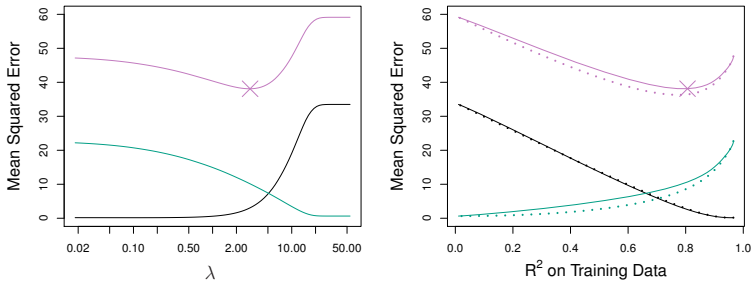
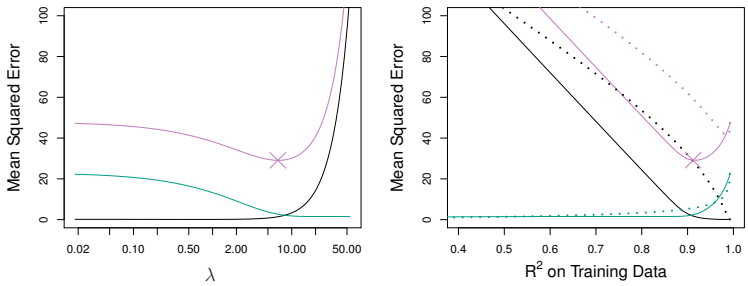


Figure 6.9: Same simulation, but with only 2 out of 45 predictors truly related to the outcome. The "best" lasso model requires a heavier penalty (a larger λ) compared to Figure 6.8 left panel. The ridge regression and the lasso now have quite different performance and profile, with the lasso model being a better one.

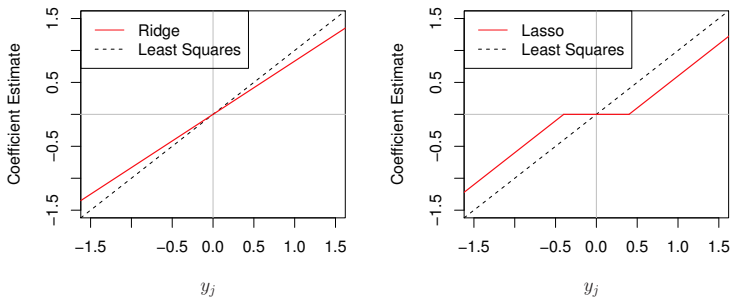


Ridge vs. lasso:

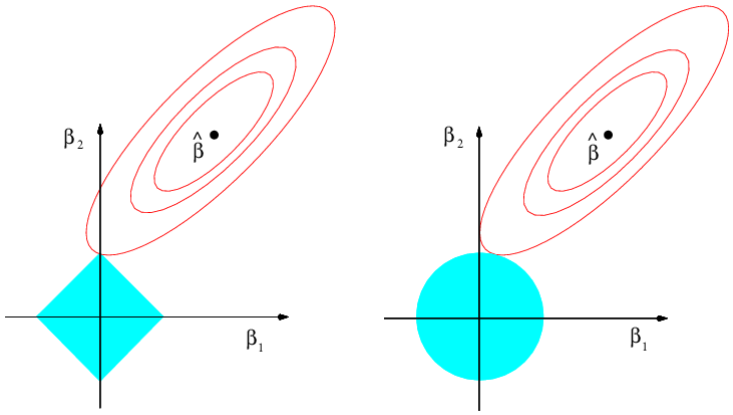
- Simulation comparisions above (Figures 6.8 and 6.9).
- Both push coefficients towards zero (a.k.a., shrinkage). The lasso models tend to have more zero coefficients (a.k.a., a more sparse model) than ridge regression models. This is because the closer a coefficient is to zero,

the less incentive ridge regression has to further push it. To the lasso, pushing a coefficient from 1 to 0.9 has the same reduction of $\sum_{j=1}^{n} |\beta_j|$ as pushing one from 0.1 to 0. But to ridge regression, the former would bring down $\sum_{j=1}^{n} \beta_j^2$ more than the latter.

- As a result, the lasso does "feature selection". This property is over-sold and can lead to misinterpretation.
- With two predictors with a very high positive correlation: Ridge regression tends to yield similar coefficients for both, while the lasso tends to push one of them out.
- Figure 6.10 shows how their coefficient estimates differ from least squares estimates:



- Figure 6.7 illustrates how ridge regression and the lasso work to minimize their respective criteria (left is the lasso, right is ridge regression):



In **elastic nets**, we minimize

$$\sum_{i=1}^{n}(y_i - \beta_0 - x_i^T\beta)^2 + \lambda\sum_{j=1}^{p}(\alpha|\beta_j| + (1-\alpha)\beta_j^2),$$

with two hyperparameters: $\lambda > 0$ and $0 < \alpha < 1$. When $\alpha = 0$, it is ridge regression; when $\alpha = 1$, it is the lasso.

The R package `glmnet` provides functions for fitting ridge/lasso/elastic net models, and for cross-validation to select hyperparameters. Try to see if standardization changes the results.

```
library(glmnet)
library(ISLR)  ## for the Hitters dataset
Hitters = na.omit(Hitters); dim(Hitters)  # 263, 20
x = model.matrix(Salary ~ ., Hitters)[,-1]  # remove the "intercept" column
y = Hitters$Salary
x = scale(x)  # standardize the predictors!

## ridge regression
ridge.mod = glmnet(x, y, alpha=0)  # over a grid of lambda values
names(ridge.mod)
plot(ridge.mod, xvar='lambda')
```

```
cv.out = cv.glmnet(x, y, alpha=0)  # default is 10-fold CV
cv.out = cv.glmnet(x, y, alpha=0, lambda=exp(seq(0, 20, 0.05)))  # widen the search space
names(cv.out)

## lambda.min: lambda that gives the smallest CV MSE (cvm) (within the search grid)
## lambda.1se: largest lambda whose CV MSE is within 1 SD of the smallest CV MSE (one-SE rule)
with(cv.out, lambda.min == lambda[which.min(cvm)])  # True
plot(cv.out)  ## with(cv.out, plot(lambda, cvm, log="x", xlog=T))
minidx = with(cv.out, which(lambda == lambda.min))  # index position for lambda.min
with(cv.out, abline(h = cvm[minidx] + cvsd[minidx], lty=2))  # show the one-SE rule

predict(ridge.mod, s=cv.out$lambda.min, type="coefficients")  # coef
ridge.pred = predict(ridge.mod, s=cv.out$lambda.min, newx=x)  # prediction

## lasso
lasso.mod = glmnet(x, y, alpha=1)
plot(lasso.mod, xvar='lambda')  # lambda as the x
plot(lasso.mod, xvar='norm')    # l1 norm as the x

cv.out = cv.glmnet(x, y, alpha=1)
plot(cv.out)
predict(lasso.mod, type="coefficients", s=cv.out$lambda.min)[1:20,]  # coef for lambda.min
predict(lasso.mod, type="coefficients", s=cv.out$lambda.1se)[1:20,]  # coef for lambda.1se
lasso.pred = predict(lasso.mod, s=lambda.1se, newx=x)  # prediction
```

5.1.3 Assignment

1. Reading for next lecture: ISLR 6.3–6.4; HOML Chapter 4
2. ISLR Chapter 6 R Labs

5.2 Week 5 Day 2

5.2.1 ISLR 6.3 Dimension reduction

General strategy: Instead of fitting a model with $p+1$ coefficients,

$$y = \beta_0 + \beta_1 x_1 + \ldots + \beta_p x_p + \epsilon,$$

we fit a model with $M+1$ coefficients ($M \ll p$),

$$y = \beta_0 + \beta_1 z_1 + \ldots + \beta_M z_M + \epsilon.$$

Here $z_j = f_j(x_1, \ldots, x_p)$ is a summary feature of the original predictors. The functions f_j may be nonlinear. This is also called *feature extraction* in machine learning.

This section describes two approaches. Both approaches use linear functions for f_j's.

Principal components (PCs) can be defined in multiple ways. The most intuitive definition is the following: Imagine we project data onto any direction in R^p. The direction that retains the largest variance is the first PC. Then we project all data onto a subplane that is perpenticular to the first PC. On the subplane, repeat the process to identify the direction that retains the largest variance as the second PC. And so on. Every PC is a linear combination of the input variables. The coefficients are called the **loadings**. For example, the first PC can be represented as $\sum_{j=1}^{p} v_{1j} x_j$, with v_{1j} being the loading for x_j. Here, $v_1 = (v_{11}, \ldots, v_{1p})$ is called the loading vector for PC1, and $\sum_{j=1}^{p} v_{1j}^2 = 1$.

PCs can be calculated through the **singular value decomposition** (SVD) of a column centered data matrix. The SVD of an $n \times p$ centralized matrix X is

$$X = UDV^T,$$

where $U_{n\times p}$ and $V_{p\times p}$ have orthonormal columns, and $D_{p\times p}$ is a diagonal matrix. (When $n < p$, $U_{n\times n}$, $D_{n\times n}$, and $V_{p\times n}$.) Let $U = (u_1, \ldots, u_p)$, $V = (v_1, \ldots, v_p)$, and $D = diag(d_1, \ldots, d_p)$ with $d_1 \geq \cdots \geq d_p \geq 0$. Then $X = UDV^T = \sum d_j u_j v_j^T$, which gives the SVD a symmetric appearance.

The **j-th PC** is $Xv_j = d_j u_j$. The elements of v_j are the *loadings* of the input variables for the j-th PC. The *standard deviation* of the j-th PC is $s_j = d_j/\sqrt{n-1}$. The *total variance* is $\sum s_j^2 = \frac{1}{n-1}\sum d_j^2$.

Fraction of total variance explained by the first k PCs is $\sum_{j=1}^k d_j^2 / \sum_{j=1}^p d_j^2$. We may select the first k PCs that account for most (say, 90%) of the total variation; that is, $k = \arg\min_k(\sum_{j=1}^k d_j^2 \geq 0.9 \sum d_j^2)$. The selected first k PCs are then used as the predictors in PC regression.

Note that PCs can also be calculated through the eigen approach using the covariance or correlation matrix of the input variables. R function `princomp()` uses this approach. The R function `prcomp()` uses the SVD approach. `prcomp()` is recommended for PC calculation.

Standardize the features unless there is a reason not to. Not standardizing the features may allow one variable to have more influence over another. For example, if weight is recorded in kilograms and height in millimeters, the `weight` variable has values mostly between 50 and 150 while the `height` variable has values mostly between 1000 and 2000. The first PC would be dominated by `height` without standardization. Unfortunately, standardization is not the default in R function `prcomp()`!

```
prcomp(x, retx = TRUE, center = TRUE, scale. = FALSE,
       tol = NULL, rank. = NULL, ...)

scale.: a logical value indicating whether the variables should be
        scaled to have unit variance before the analysis takes place.
        The default is 'FALSE' for consistency with S, but in general
        scaling is advisable.
```

```r
## simulate three variables
N=100
x1 = 1 + rnorm(N, 0, 1)
x2 = x1*.2 + 5 + rnorm(N, 0, .4)
x3 = x1*.1 + x2*.2 + 5 + rnorm(N, 0, .2)

M = cbind(x1,x2,x3)      # original
Mcen = scale(M, scale=F) # centered
Mstd = scale(M)          # standardized
t(Mstd) %*% Mstd  ## the diagonal is N-1

## visualize the data
library(rgl)
myrglplot = function(M) {
    x1 = M[,1]; x2 = M[,2]; x3 = M[,3]
    l1 = min(x1,x2,x3); if(l1>0) l1=0
    u1 = max(x1,x2,x3); if(u1<0) u1=0
    plot3d(x1,x2,x3, xlim=c(l1,u1), ylim=c(l1,u1), zlim=c(l1,u1))
    arrow3d(c(0,0,0), c(u1,0,0), barblen=.05, width=.1, type='rotation', col=1)
    arrow3d(c(0,0,0), c(0,u1,0), barblen=.05, width=.1, type='rotation', col=2)
    arrow3d(c(0,0,0), c(0,0,u1), barblen=.05, width=.1, type='rotation', col=3)
}

clear3d(); myrglplot(M)
clear3d(); myrglplot(Mcen)
clear3d(); myrglplot(Mstd)
```

To calculate the PCs in R, use `prcomp(x, scale.=T)`!

```
a2 = prcomp(M, scale=T)  ## same as prcomp(Mstd)
names(a2)
dim(a2$x)    # a2$x are the PCs, with the same dimensions as the input
a2$rotation  # loadings of the PCs
a2$sdev      # SDs of the PCs
sum((a2$sdev)^2)  ## sum of PC variances is p
cumsum((a2$sdev)^2) / sum((a2$sdev)^2)  ## fraction of total variance explained by the first PCs
```

The information can also be obtained through SVD:

```
s2 = svd(Mstd)
names(s2)   ## d, u, v
dim(s2$u); dim(s2$v); length(s2$d)
s2$d/sqrt(N-1)            # SDs
s2$v                      # loadings
dim(s2$u %*% diag(s2$d))  # PCs
head(a2$x, 4); head(s2$u %*% diag(s2$d), 4)  ## check the first few rows
```

The following simulation demonstrates the differences between first PC and regression line, and between standardization and just centralization.

```
library(MASS)  ## for mvrnorm()
aa = mvrnorm(1000, mu=c(0,0), Sigma=matrix(c(4,1.3,1.3,1),2))
colnames(aa)=c('x','y')
plot(aa)

## Plot the direction of the first PC on the original scales of x and y.
## The first PC for standardized features recovers the "true" direction in data.
load1a = prcomp(aa, scale=T)$rotation[,1]  ## loadings of the first PC
abline(0, (load1a[2]*sd(aa[,2]))/(load1a[1]*sd(aa[,1])), col=2, lwd=2)
## But the first PC for non-standardized features does not.
load1b = prcomp(aa)$rotation[,1]
abline(0, load1b[2]/load1b[1], col=1, lwd=2)

lm1 = lm(aa[,'y'] ~ aa[,'x'])
abline(lm1, col=2)  ## regression line when regressing y on x
coef2 = lm(aa[,'x'] ~ aa[,'y'])$coef
abline(-coef2[1]/coef2[2], 1/coef2[2], col=2, lty=2)  ## regression line when regressing x on y
```

The last few lines of code demonstrate the effect of "regression to the mean". When regressing x on y, the fitted line $x = a + by$ is equivalent to $y = -\frac{a}{b} + \frac{1}{b}x$. It is different than the regression line of y on x! And they both differ from the first PC! This is because both models regress to the mean.

5.2.2 Assignment

1. **Homework**: ISLR Chapter 6 Exercises 9, 11
2. Reading for next lecture: Rest of ISLR 6; HOML 4.