6 Ridge, Lasso, etc

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).

6.1 ISLR 6.1 Subset selection

General strategy:

1. For each d ($d = 1, \ldots, p$), pick the "best" model among the models with d features using the training set.
2. Compare the p "best" models using one of the following approaches:
   - Direct estimation of test error, using a validation set or cross-validation;
   - Indirect estimation 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**: In Step 1, pick the "best" model among *all* models with d predictors.
- 6.1.2: Stepwise. In Step 1,
  - **Forward**: At $d = 1$, pick the 'best' single-predictor model. At each $d = 2, \ldots, p$, consider adding a predictor to the 'best' model at level $d - 1$, and pick the 'best' one (the one with 'best improvement'). This works even when $n < p$ (d runs up to $d = n$).
  - **Backward**: At each $d = p - 1, \ldots, 1$, consider removing a predictor from the 'best' model at level $d + 1$, and pick the 'best' one (the one with 'least degradation'). This does not work when $n < p$.
  - Both are *greedy* algorithms. But they are "guided" so that they effectively search over more models than those evaluated.

Note: 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 that *observations with any missing value have to be removed.*

- 6.1.3: Some traditional measures for model comparison. Let d be the number of *parameters* for the model being evaluated. For every model, we may calculate
  - $R^2_{adj} = 1 - \frac{RSS/(n-(d+1))}{TSS/(n-1)} = 1 - \frac{n-1}{n-(d+1)}(1 - R^2)$. *Higher is better.* (Fisher, 1924)
    - It is an adjustment of $R^2 = 1 - \frac{RSS}{TSS}$.
  - $C_p = \frac{RSS}{\hat{\sigma}^2_F} + 2d - n$. *Lower is better.* (Mallows, 1973)
    - where $\hat{\sigma}^2_F$ is from the full model and d is the number of *predictors* in the model being evaluated.
    - It is equivalent to $\frac{1}{n}(RSS + 2d\hat{\sigma}^2_F)$, which is (6.2) in ISLR.
    - When n is very small, $\hat{\sigma}^2_F$ can be underestimated, leading to a smaller penalty on large d.
  - AIC $= -2\log(\hat{L}) + 2d$. *Lower is better.* (Akaike information criterion; Akaike, 1973)
  - BIC $= -2\log(\hat{L}) + \log(n)d$. *Lower is better.* (Bayesian information criterion; Schwarz, 1978)
    - where $\hat{L}$ is the maximum likelihood.
    - AIC and BIC are penalized log-likelihoods.
    - BIC > AIC, unless $n \leq e^2 = 7.39$, which is very rare in practice.
    - BIC puts more penalty on high d and tends to favor models with a smaller d than AIC.

Intuition about AIC and BIC: Consider two nested models, F and R, with d_F and d_R parameters, respectively. Suppose the reduced model is the correct model. 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 $2(\log \hat{L}_F - \log \hat{L}_R) > 2(d_F - d_R)$. Then the probability of wrongly selecting model F is 0.16 when $d_F - d_R = 1$, 0.09 when $d_F - d_R = 4$, and 0.05 when $d_F - d_R = 7$. These probabilities are a little high.

$BIC_F < BIC_R$ is equivalent to $2(\log \hat{L}_F - \log \hat{L}_R) > \log(n)(d_F - d_R)$. At $n = 100$, $\log(100) = 4.6$, the probability of wrongly selecting model F is 0.03 when $d_F - d_R = 1$, and 0.01 when $d_F - d_R = 2$. At $n = 500$, $\log(500) = 6.2$, the probability of wrongly selecting model F is 0.01 when $d_F - d_R = 1$, and 0.002 when $d_F - d_R = 2$.]

The claim that AIC is equivalent to C_p for linear regression is valid only for a rare situation where the residuals are Gaussian and have a *known* residual variance. [In linear regresion, $-2\log(\hat{L}) = n\log(2\pi\hat{\sigma}^2_{mle}) + (RSS/\hat{\sigma}^2_{mle})$. When σ^2 is known, $-2\log(\hat{L}) = n\log(2\pi\sigma^2) + (RSS/\sigma^2)$, and AIC differs from C_p by a constant.]

```
library(ISLR)
dim(Hitters); names(Hitters)
row.names(Hitters)  # n = 322
apply(is.na(Hitters), 2, sum)  ## 59 players had missing "Salary"

group = apply(is.na(Hitters), 1, sum) > 0  ## Take a further look.
par(mfrow=c(4,5))
for(i in names(Hitters))
    boxplot(as.numeric(Hitters[[i]]) ~ group, main=i)
```

R has functions `AIC()` and `BIC()`.

```
mod = lm(Salary ~ ., data=Hitters)
AIC(mod); -2*as.numeric(logLik(mod)) + 2*21  ## same value
BIC(mod); -2*as.numeric(logLik(mod)) + log(263)*21  ## same value
RSS = sum(mod$residuals^2); ssmle = RSS/263
as.numeric(logLik(mod)); -RSS/(2*ssmle) -263/2*log(2*pi*ssmle)  ## same value
```

The R package `leaps` provides a function `regsubsets()` for subset selection. First, we have to remove observations with missing data. This step is unnecessary for `regsubsets()` because it will do it internally. But it will be a good habbit to do it explicitly so that you know what you are doing.

```
library(leaps)
library(ISLR)
## REMOVE observations with missing data
Hitters = na.omit(Hitters); dim(Hitters)  # n = 263 after removal of those with any missing value
```

Note that `regsubsets()` has options such as `force.in=` and `force.out=`. The default for `nvmax` is 8.

```
regfit.full = regsubsets(Salary ~ ., data=Hitters, nvmax=19)
reg.summary = summary(regfit.full)
reg.summary   ## reg.summary$which
names(reg.summary)  ## has cp, bic, adjr2

plot(regfit.full); reg.summary$bic
```

The plot is quite informative. The default scale for the plot is BIC, which is the most useful among the available options. We can select models using any of the following criteria.

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

The function `regsubsets()` can also do forward and backward selection.

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

Out of curiosity, I check how the measures `cp` and `bic` are calculated. It turns out the `bic` reported by `summary()` (which is actually `summary.regsubsets()`) is the BIC minus the BIC for the null model with only the intercept (and plus $\log(n)$ for no obvious reason). Because `bic` is a simple shift of BIC, `bic` gives the same order for the

models as BIC does.

```
regfit.full = regsubsets(Salary ~ ., data=Hitters, nvmax=19)  # exhaustive search
reg.summary = summary(regfit.full)

## check the values for cp, which are okay
lmfull = lm(Salary ~ ., data=Hitters)
all.equal(reg.summary$cp, reg.summary$rss / summary(lmfull)$sigma^2 + 2*(2:20) - 263)  ## True

## check the values for bic for the full model
reg.summary$bic[19]   ## reported by summary()
BIC(lm(Salary ~ ., data=Hitters))  ## by BIC()
bicnull = BIC(lm(Salary ~ 1, data=Hitters))
BIC(lm(Salary ~ ., data=Hitters)) - bicnull + log(263)

ssmle = reg.summary$rss/263
all.equal(reg.summary$bic,   ## True
          reg.summary$rss/ssmle + 263*log(2*pi*ssmle) + log(263)*(3:21) - bicnull + log(263))
```

6.2 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 some variables to have more influence than others. 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 (a hyperparameter). Selection of λ is often through cross-validation.

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 ordinary least squares solution.

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

```
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!
```

The `glmnet()` function in the `glmnet` package can perform ridge regression (by setting `alpha=0`), the lasso (by setting `alpha=1`, which is the default), and elastic net. If we do not provide a sequence of `lambda` values, it will select 100 values in a "wide" range, which may not be wide enough sometimes.

```
library(glmnet)
ridge.mod = glmnet(x, y, alpha=0, lambda=exp(seq(-4, 11, 0.05)))
names(ridge.mod)
dim(ridge.mod$beta)  # (11, 301)
```

```
plot(ridge.mod, xvar='lambda')  # Figure 6.4 left panel

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 6.4 (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.

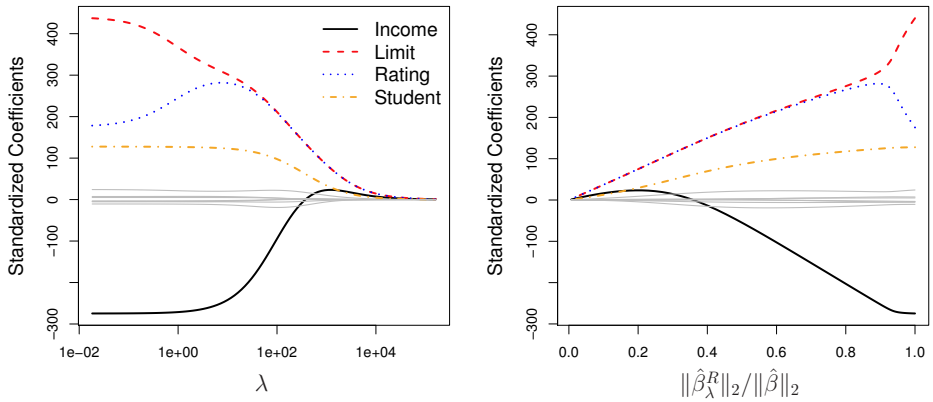
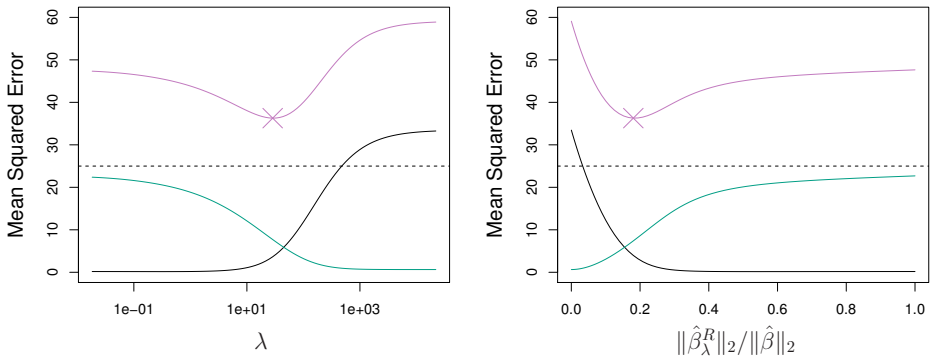


Figure 6.5 shows a scenario where penalized regression gives a better prediction performance than the traditional least squares at $\lambda = 0$. (Simulation with $n = 50$, $p = 45$)



- 6.2.2: **Lasso** (l_1 **regularization**)

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

where $\lambda \geq 0$ and $\beta = (\beta_1, \ldots, \beta_p)$. When $\lambda = 0$, it becomes the least squares. The λ in the lasso and the λ in ridge regression are on different scales, and thus they are not comparable.

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.

The range of `lambda` for the lasso can be quite different from that for ridge regression.

```
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
```

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).

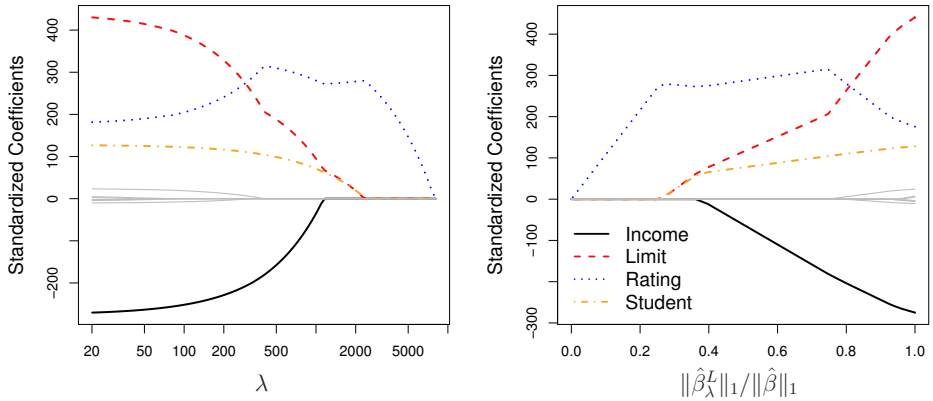


Figure 6.8: (Same simulation as in Figure 6.5) The left panel shows a scenario where the lasso gives a better prediction performance than the least squares at $\lambda = 0$. The right panel shows a comparison between ridge and lasso. In this simulation setting, the two approaches have similar prediction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

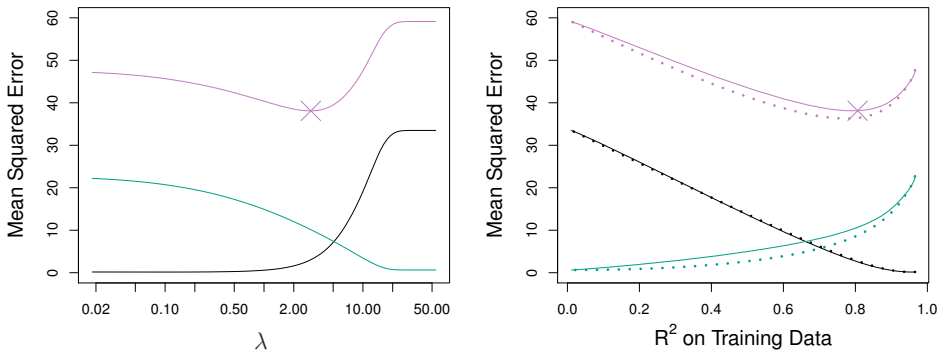
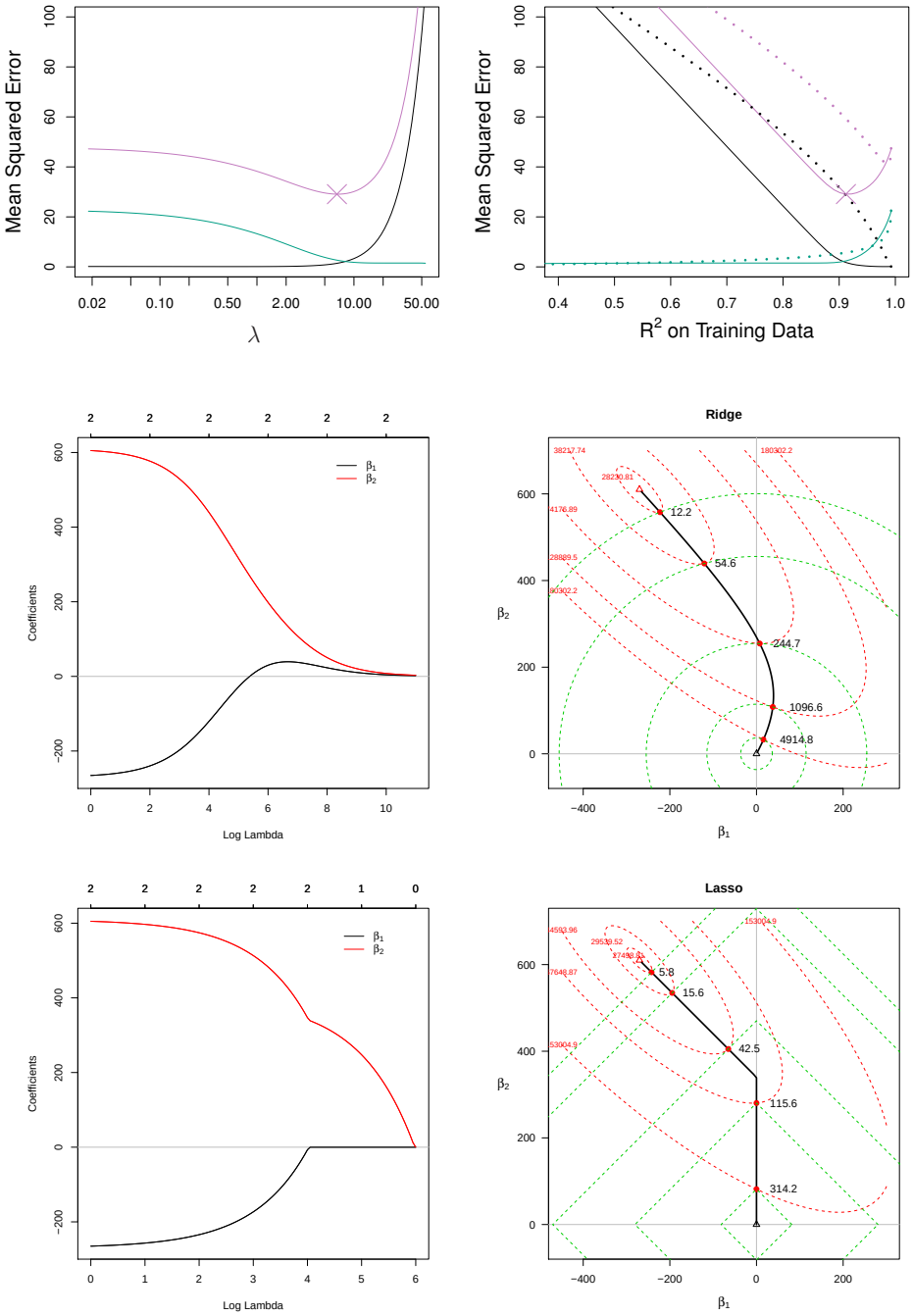


Figure 6.9: Same simulation, but with only 2 out of 45 predictors truly related to the outcome. The "best" lasso model requires a heavier penalty (a larger λ) compared to Figure 6.8 left panel. The ridge regression and the lasso now have quite different performance and profile, with the lasso model being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



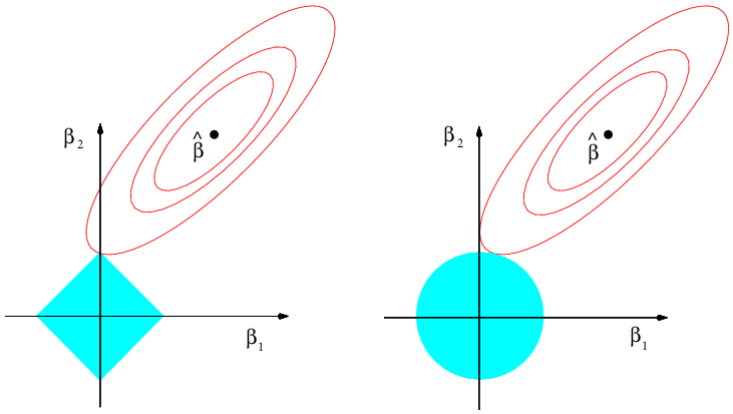
Alternative representations: Given the data, the ridge regression (6.5) can be fit by solving this optimization problem:

$$\text{minimize}_{\beta_0,\beta} \sum_{i=1}^{n}(y - \beta_0 - x_i^T\beta)^2 \quad \text{subject to} \quad \sum_{j=1}^{p}\beta_j^2 \leq s_r, \tag{6.9}$$

where s_r is a function of $\lambda > 0$; the larger λ the smaller s_r. Similarly, the lasso (6.7) can be fit by solving

$$\text{minimize}_{\beta_0,\beta} \sum_{i=1}^{n}(y - \beta_0 - x_i^T\beta)^2 \quad \text{subject to} \quad \sum_{j=1}^{p}|\beta_j| \leq s_l, \tag{6.8}$$

where s_l is a function of $\lambda > 0$; the larger λ the smaller s_l. Figure 6.7 illustrates how ridge regression and the lasso work to minimize their respective criteria (left is the lasso, right is ridge regression):

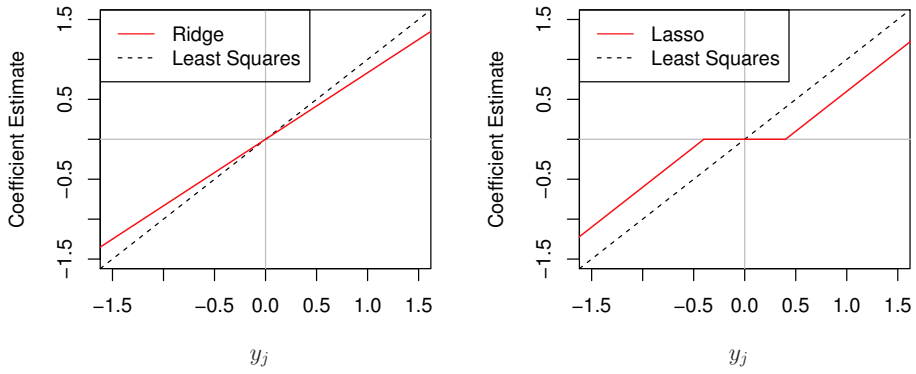


Best subset selection using squared error as the criterion with a cap K on the number of predictors is effectively solving the optimization problem (which is not easy):

$$\text{minimize}_{\beta_0,\beta} \sum_{i=1}^{n}(y - \beta_0 - x_i^T\beta)^2 \quad \text{subject to } \sum_{j=1}^{p} I(\beta_j \neq 0) \leq K. \tag{6.10}$$

Ridge vs. lasso:

- Simulation comparisions above (Figures 6.8 and 6.9).
- Both push coefficients towards zero, and thus they are also called **shrinkage** methods. The lasso models tend to have more zero coefficients (i.e., a more sparse model) than ridge regression models. This is because when a coefficient is close to zero, ridge regression has little incentive 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" to yield a more parsimonious model. However you should not over-interpret this property. You could be misled if you only focus on the "selected" features and ignore the excluded ones.
- If there are two predictors that are highly correlated: 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 in a model with n coefficients, $y_i = \beta_i + \epsilon_i$:



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. Often a large value such as $\alpha = 0.9$ is used to fit elastic net models; the resulting models would perform more like the lasso.

The R package `glmnet` provides functions for fitting ridge/lasso/elastic net models, and for cross-validation to select hyperparameters. Again, we first prepare the data.

```
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!
```

For ridge regression:

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

## cross-validation
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` is the lambda on the search grid that gives the smallest CV MSE (cvm). `lambda.1se` is the largest lambda whose CV MSE is within 1 SD of the smallest CV MSE (one-SE rule)

```
cv.out$lambda.min; cv.out$lambda.1se
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$lambda.min, type="coefficients")  # coef for lambda.min
predict(ridge.mod, s=cv.out$lambda.1se, type="coefficients")  # coef for lambda.1se
ridge.pred = predict(ridge.mod, s=cv.out$lambda.min, newx=x)  # prediction with a chosen lambda
```

Similarly, for the 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$lambda.min)  # coef for lambda.min
predict(lasso.mod, type="coefficients", s=cv.out$lambda.1se)  # coef for lambda.1se
lasso.pred = predict(lasso.mod, s=cv.out$lambda.1se, newx=x)  # prediction with a chosen lambda
```

6.3 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**; they reflect the contributions of variables to the PC. 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 a symmetric representation for the SVD.

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 variance; that is, $k = \arg\min_k (\sum_{j=1}^k d_j^2 \geq 0.9 \sum d_j^2)$.

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 some variables to have more influence than others. 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.
```

We now do some demonstrations. First, we 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
```

We now 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
```

We now demonstrate the differences between standardization and just centralization, and between the first PC and regression line. We first generate bivariate data (x, y), with $var(x) = 4$ and $var(y) = 1$.

```
library(MASS)  ## for mvrnorm()
set.seed(2019)
aa = mvrnorm(1000, mu=c(0,0), Sigma=matrix(c(4, 1.3, 1.3, 1), 2))
colnames(aa)=c('x','y')
```

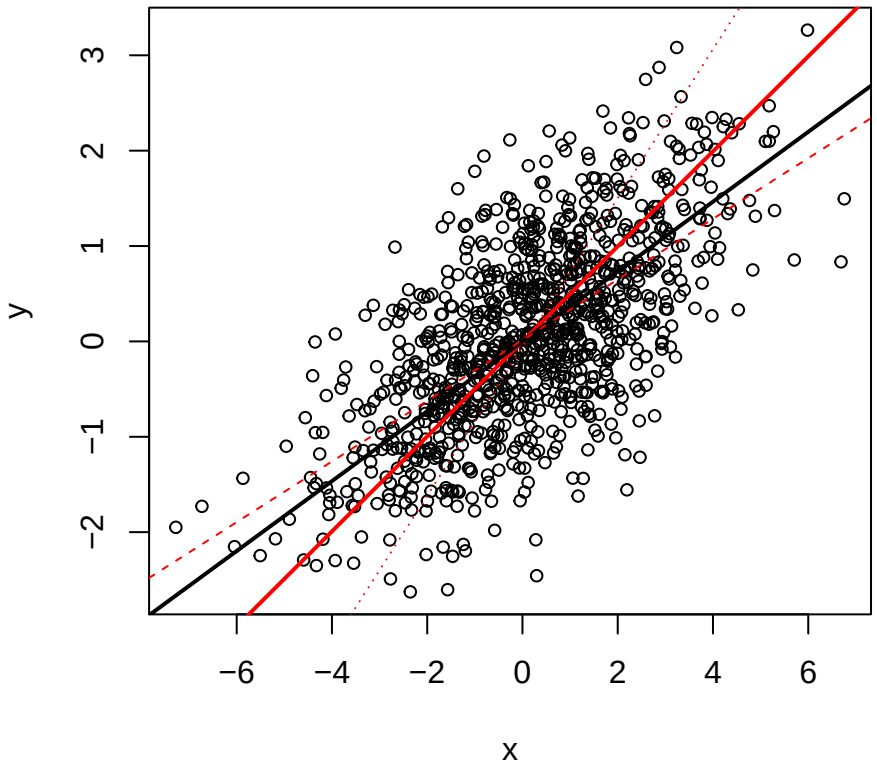
Now we plot the direction of the first PC on the original scales of x and y. The first PC for the standardized features reflects the "main direction" in data.

```
plot(aa, cex=.8)
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)  ## red solid

## Without standardization, the first PC leans towards features with "high" variances
load1b = prcomp(aa)$rotation[,1]
abline(0, load1b[2]/load1b[1], col=1, lwd=2)  ## black solid

lm1 = lm(aa[,'y'] ~ aa[,'x'])
```

```
abline(lm1, col=2, lty=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=3)  ## regression line when regressing x on y
```

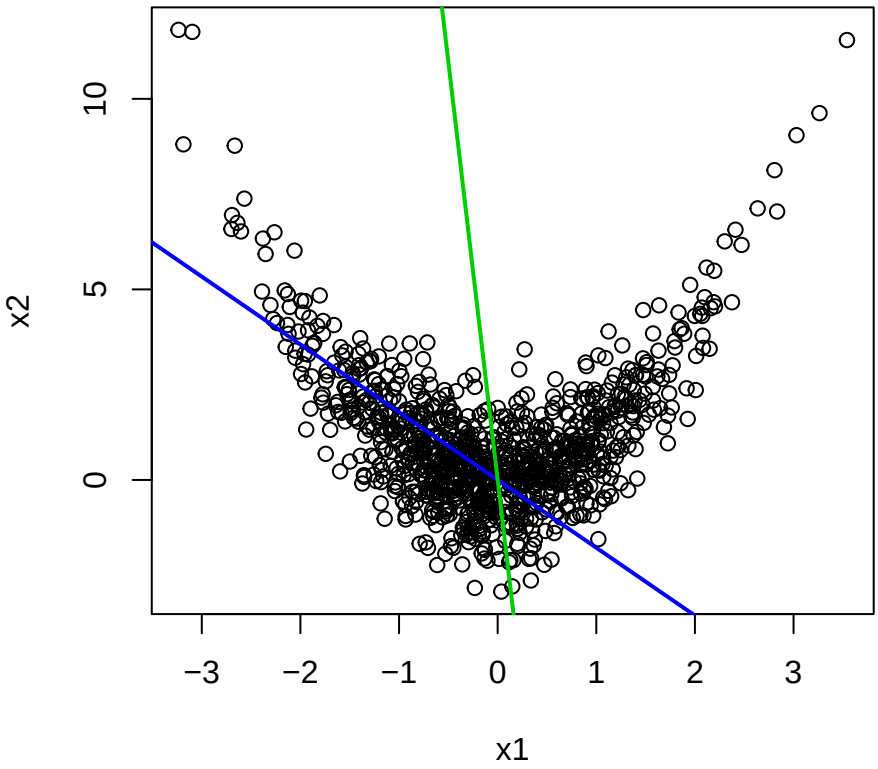


The last few lines of code demonstrate the effect of "regression to the mean". When regressing x on y, the fitted line $x = a + by$ is equivalent to $y = -\frac{a}{b} + \frac{1}{b}x$. It is different than the regression line of y on x! And they both differ from the first PC!

By definition, principal components only reflect linear variation. **They cannot capture nonlinear patterns in the data.** An example is below:

```
set.seed(2019)
x1 = rnorm(1000)
x2 = x1^2 + rnorm(1000)
bb = cbind(x1, x2)

plot(x1, x2)
load1a = prcomp(bb, scale=T)$rotation[,1]
abline(0, (load1a[2]*sd(bb[,2]))/(load1a[1]*sd(bb[,1])), col=4, lwd=2)  ## PC1, blue
load1b = prcomp(bb)$rotation[,1]
abline(0, load1b[2]/load1b[1], col=3, lwd=2)  ## PC1 without standardization is even worse. green
```



For data with weak correlation among the features, PCs are not useful. For example,

```
library(MASS)  ## for mvrnorm()
x = diag(10); x = 1 * 0.1^abs(row(x)-col(x))  ## 10-dim autoregressive correlation
aa = mvrnorm(1000, mu=rep(0,10), Sigma=x)

bb = prcomp(aa, scale=T)
cumsum((bb$sdev)^2) / sum((bb$sdev)^2)  ## slow accumulation of variance
```

6.4 ISLR 6.3 (Cont'd)

- 6.3.1: Principal components regression (PCR): Use the first M PCs of the input variables as the predictors.

$$y = \beta_0 + \beta_1 \text{PC}_1 + \cdots + \beta_M \text{PC}_M.$$

  - The hyperparameter M can be chosen with cross-validation, or with a pre-determined threshold for the fraction of total variance explained by the first M PCs.
  - The PCs are determined solely on X. The outcome variable is not involved in PC calculation.
  - PCR is not feature selection. It is more like feature extraction.
  - There is no guarantee that the directions that best explain the features will also be the best directions to use for predicting the outcome.
  - It performs well when the first few PCs capture most of the variance in the features.
  - It can be viewed as a discrete version of ridge regression.

The R `pls` package provides a function `pcr()` for PC regression. The output is of class `mvr`. To look for the documents for `summary()` and `predict()`, look at `summary.mvr()` and `predict.mvr()`.

```
require(pls)
library(ISLR)
Hitters = na.omit(Hitters)

pcr.fit = pcr(Salary ~ ., data=Hitters, scale=T)
```

```
coef(pcr.fit, ncomp=5)  ## coefficients for the standardized PREDICTORs for the model using 5 PCs
pcr.pred = predict(pcr.fit, Hitters, ncomp=5)  ## prediction using 5 PCs
```

Using all PCs results in the same model as the full data linear model. We check the coefficients and the fitted values.

```
pcr.coef = coef(pcr.fit, ncomp=19) / apply(data.matrix(Hitters[,-19]), 2, sd)
pcr.pred = predict(pcr.fit, Hitters, ncomp=19)

mod = lm(Salary ~ ., data=Hitters)  ## full data linear model
mod.coef = coef(mod)[-1]            ## coefficients
mod.pred = predict(mod, Hitters)    ## fitted values

all.equal(mod.coef, as.numeric(pcr.coef), check.attributes = F)  ## True
all.equal(mod.pred, as.numeric(pcr.pred), check.attributes = F)  ## True
```

We can perform cross-validation on PC regression.

```
pcr.fit = pcr(Salary ~ ., data=Hitters, scale=T, validation="CV")
summary(pcr.fit)  ## summary.mvr()
validationplot(pcr.fit, val.type="RMSE")
```

- 6.3.2: Partial least squares (PLS):

1. Standardize all the features. Then let $y_0 = y$, $x_{0,j} = x_j$ $(j = 1, \ldots, p)$.
2. For $m = 1, \ldots, M$:
   a. Perform simple linear regression of y_{m-1} on $x_{m-1,j}$ to obtain slope $\hat{\beta}_{mj}$ $(j = 1, \ldots, p)$.
   b. Compute $Z_m = \sum_j \hat{\beta}_{mj} x_j$.
   c. Perform simple linear regression of y_{m-1} on Z_m to obtain residual y_m, and simple linear regression of $x_{m-1,j}$ on Z_m to obtain residual $x_{m,j}$ $(j = 1, \ldots, p)$.
3. Perform the final regression:
$$y = \beta_0 + \beta_1 Z_1 + \cdots + \beta_M Z_M.$$
   - The hyperparameter M can be chosen with cross-validation.
   - Z_m can be viewed as a composite score over individual contributions from the predictors.

The R `pls` package provides a function `plsr()` for partial least squares regression.

```
pls.fit = plsr(Salary ~ ., data=Hitters, scale=T)
coef(pls.fit, ncomp=5)
pls.pred = predict(pls.fit, Hitters, ncomp=5)
```

Using all p Z-scores leads to the same model as the full data linear model.

```
pls.coef = coef(pls.fit, ncomp=19) / apply(data.matrix(Hitters[,-19]), 2, sd)
pls.pred = predict(pls.fit, Hitters, ncomp=19)

all.equal(mod.coef, as.numeric(pls.coef), check.attributes = F)  ## True
all.equal(mod.pred, as.numeric(pls.pred), check.attributes = F)  ## True
```

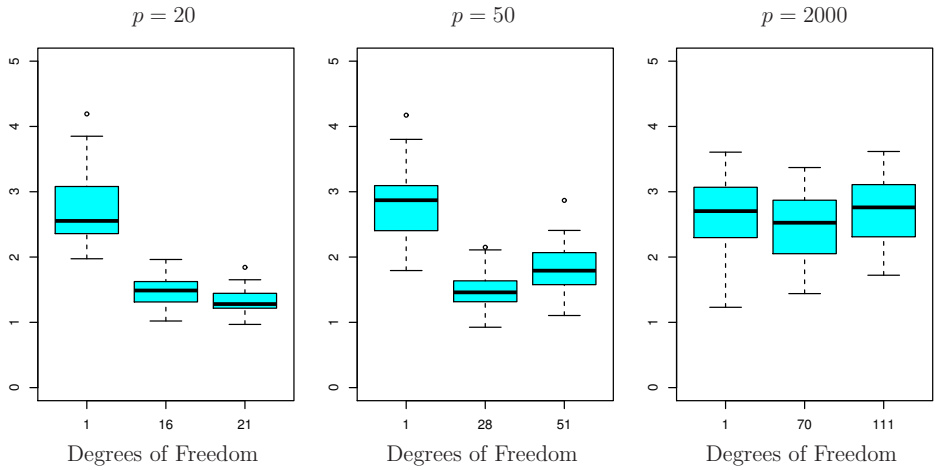
We can perform cross-validation on PLS regression.

```
pls.fit = plsr(Salary ~ ., data=Hitters, scale=T, validation="CV")
summary(pls.fit)
validationplot(pls.fit, val.type="MSEP")
```

6.5 ISLR 6.4 High dimensional data

- 6.4.1: High dimensional data have a large p. The authors are too strict by using $n < p$ as the definition.

- 6.4.2: Traditional measures such as C_p, AIC, and BIC are not suitable for high dimensional data.
- 6.4.3: Curse of dimensionality. Figure 6.24. $n = 100$ and only 20 features are associated with the outcome.



- 6.4.4: Do not over-interpret the results.

6.6 Assignment

1. **Homework**: ISLR Chapter 6 Exercises 9, 11