```
# First I'll bring in the necessary libraries.
# (Note: It may be that not all of these are
# needed.)
library(class)
library(MASS)
library(Hmisc)
library(classPP)
library(klaR)
library(e1071)
library(kknn)
library(rpart)
library(boost)
library(mvtnorm)
library(multinomRob )
library(lars)
library(stats)
library(leaps)

#
# I like to set the seed when using randomly-generated
# data --- it's useful to be able to obtain the same
# samples when trouble shooting, and it's good to be
# able to have it so that others can reproduce the
# results.
set.seed(632)

#
# I'll generate the training data.
x1 <- runif(50, 0, 1)
x2 <- x1 + rnorm(50, 0, 0.25)
x3 <- (x1 + x2)/2 + runif(50, 0, 0.1)
x4 <- runif(50, 0, 1)
x5 <- (2*x4 + rnorm(50, 0, 0.25))/2 + runif(50, 0, 0.1)
x6 <- runif(50, 0, 1)
y <- (3 + x1 + x2 + 0.5*x3 + 0.75*x4 + 0.5*x5 + 0.5*x6 + rnorm(50, 0, 1))

#
# Since I'm going to use some methods for which
# standardized predictors are generally preferred,
# to keep things simpler, I'll use standardized
# predictors with all of the methods.
x <- scale( cbind(x1,x2,x3,x4,x5,x6) )
trdata <- data.frame( cbind(x,y) )
names(trdata) <- c("sx1", "sx2", "sx3", "sx4", "sx5", "sx6", "y")
attach(trdata)
cor(trdata)

#
# I'll use OLS to fit a linear model
# using all of the (standardized) predictors.
ols1 <- lm(y ~ sx1 + sx2 + sx3 + sx4 + sx5 + sx6)
summary(ols1)

#
# I'll use a backwards elimination strategy
# and fit a model with sx6 omitted.
ols2 <- lm(y ~ sx1 + sx2 + sx3 + sx4 + sx5)
summary(ols2)
#
# I'll now drop sx4.
ols3 <- lm(y ~ sx1 + sx2 + sx3 + sx5)
summary(ols3)
#
# Next, I'll drop sx2.
```

```
ols4 <- lm(y ~ sx1 + sx3 + sx5)
summary(ols4)
#
# Now, I'll drop sx3.
ols5 <- lm(y ~ sx1 + sx5)
summary(ols5)

#
# At this point I'll stop with the backwards
# elimination, and check to see what a stepwise
# procedure gives me.
ols6 <- step(ols1, direction="both")
summary(ols6)
# So, the stepwise regression based on AIC led
# to the same model as backwards elimination.
#
# If one wanted to pursue OLS models more,
# one could use
#   bestss <- regsubsets(y ~ sx1 + sx2 + sx3 + sx4 + sx5 + sx6, data=trdata)
#   summary.regsubsets(bestss)
# to see that backwards elimination did not
# even arrive at the best two variable model.
# But to deal sensibly with the output of a
# best subsets regression routine, one needs a
# way to compare the goodness of models having
# different numbers of variables.  (I'll say
# more about this later in class.)

#
# Perform lasso using lars funstion,
# noting that the syntax is different
# --- one needs to give it the design
# matrix.
las <- lars(x, y, type="lasso")
las
plot(las, plottype="coefficients")

#
plot(las, plottype="Cp")
# The C_p seems to be minimized at about 5.6.

#
# By using cross-validation with the lasso,
# a good (hopefully near-optimal) value for
# the "fraction" can be determined.
cvlas <- cv.lars(x, y, type="lasso")
cvlas
frac <- cvlas$fraction[which.min(cvlas$cv)]
frac
las.coef <- predict.lars(las, type="coefficients", mode="fraction", s=frac)
las.coef
# I got the same results when using the
# unstandardized predictors.
#
# As a check, let's see if setting the value of
# s (the fraction, in the mode being used) to 1
# yields the coefficient values from the OLS fit.
las.coef <- predict.lars(las, type="coefficients", mode="fraction", s=1)
las.coef
ols1
# They match!

#
# Now, I'll try ridge regression ---
# I'll give it a sequence of values
```

```
# for lambda, since it's not clear at
# this point what value to use.
lmridge <- lm.ridge(y ~ sx1 + sx2 + sx3 + sx4 + sx5 + sx6, lambda = seq(0, 10, 1))
lmridge$kHKB
lmridge$kLW
lmridge$GCV
# Two estimates of (a good) lambda are
# about 2.7 and 4.0, and generalized
# cross-validation suggests using a
# value between 6 and 8.  <I'll skip
# showing some of the work I did to
# narrow in on the interval (6.72, 6.84).>
lmridge <- lm.ridge(y ~ sx1 + sx2 + sx3 + sx4 + sx5 + sx6, lambda = seq(6.72, 6.84, 0.01))
lmridge$GCV

#
# After some searching, I determined that
# the value of lambda which minimizes the
# generalized cross-validation estimate of
# the error is about 6.8.  Two estimation
# methods produced estimated lambdas of
# about 2.7 and 4.0.  So I will fit three
# ridge regression models, using lambdas
# values of 2.7, 4.0, and 6.8.  As a check,
# I will fit a fourth model using 0 for lambda,
# which should be the same as OLS.
ridge1 <- lm.ridge(y ~ sx1 + sx2 + sx3 + sx4 + sx5 + sx6, lambda = 2.7)
ridge2 <- lm.ridge(y ~ sx1 + sx2 + sx3 + sx4 + sx5 + sx6, lambda = 4.0)
ridge3 <- lm.ridge(y ~ sx1 + sx2 + sx3 + sx4 + sx5 + sx6, lambda = 6.8)
ridge4 <- lm.ridge(y ~ sx1 + sx2 + sx3 + sx4 + sx5 + sx6, lambda = 0)
# Let's look to see if the fitted coefficients
# match those from the OLS fit.
ols1
ridge4$coef
# They are rather close, but not exactly
# the same.

#
# Now I will do principal components regression.
#
# The first step is to use the standardized training
# data to determine the principal components.
tr.pca <- princomp(~ sx1 + sx2 + sx3 + sx4 + sx5 + sx6)
# (Note: Normally, one might put cor=T, but here
# the variables have already been standardized.)
summary(tr.pca)
plot(tr.pca)
loadings(tr.pca)

#
# The next step is to compute the p.c. variable
# values from the training data.
pcomp.tr <- predict(tr.pca)
pcomp.tr

#
# Now I will "pick off" the p.c. variable vectors.
# (Note: One wouldn't have to do it this way.)
pc1 <- pcomp.tr[,1]
pc2 <- pcomp.tr[,2]
pc3 <- pcomp.tr[,3]
pc4 <- pcomp.tr[,4]
pc5 <- pcomp.tr[,5]
pc6 <- pcomp.tr[,6]
#
```

```
# And now I will do the regressions.
# I'll start by using all of the p.c's,
# which should give me a fit equivalent
# to the OLS fit based on all of the var's.
pcr1 <- lm(y ~ pc1 + pc2 + pc3 + pc4 + pc5 + pc6)
summary(pcr1)
#
# I'll use backwards elimination, and first omit pc3.
pcr2 <- lm(y ~ pc1 + pc2 + pc4 + pc5 + pc6)
summary(pcr2)
#
# I'll now omit pc5.
pcr3 <- lm(y ~ pc1 + pc2 + pc4 + pc6)
summary(pcr3)
#
# And finally I'll omit pc6.
pcr4 <- lm(y ~ pc1 + pc2 + pc4)
summary(pcr4)

#
# I'll next see what a stepwise procedure gives me.
pcr5 <- step(pcr1)
summary(pcr5)
# So, the stepwise regression based on AIC led
# to the same model as backwards elimination.
# So, I'll go with th emodel based on the 1st,
# 2nd, and 4th p.c's, and use the one based on
# all of the p.c's as a check (below).

#
# I will generate a large data set that can be
# used to estimate the generalization errors.
gx1 <- runif(5000, 0, 1)
gx2 <- gx1 + rnorm(5000, 0, 0.25)
gx3 <- (gx1 + gx2)/2 + runif(5000, 0, 0.1)
gx4 <- runif(5000, 0, 1)
gx5 <- (2*gx4 + rnorm(5000, 0, 0.25))/2 + runif(5000, 0, 0.1)
gx6 <- runif(5000, 0, 1)
gy <- (3 + gx1 + gx2 + 0.5*gx3 + 0.75*gx4 + 0.5*gx5 + 0.5*gx6 + rnorm(5000, 0, 1))

#
# To use this data with the previously
# fit models to get sensible predictions,
# it has to be standardized in the same
# way --- the sample means and sample
# variances from the training data need
# to be used!
sgx1 <- (gx1 - mean(x1))/sqrt(var(x1))
sgx2 <- (gx2 - mean(x2))/sqrt(var(x2))
sgx3 <- (gx3 - mean(x3))/sqrt(var(x3))
sgx4 <- (gx4 - mean(x4))/sqrt(var(x4))
sgx5 <- (gx5 - mean(x5))/sqrt(var(x5))
sgx6 <- (gx6 - mean(x6))/sqrt(var(x6))
gx <- cbind(sgx1,sgx2,sgx3,sgx4,sgx5,sgx6)
gendata <- data.frame( cbind(gx, gy) )
names(gendata) <- c("sx1", "sx2", "sx3", "sx4", "sx5", "sx6", "y")
detach(trdata)
attach(gendata)

#
# I will plug the new data values into the
# p.c. formulas determined above.
pcomp.gen <- predict(tr.pca, newdata=gendata)
cor(pcomp.gen)
```

```
#
# Now I will estimate the generalization errors.
ols1.mspe <- mean( (gy - predict(ols1, newdata=gendata))^2 )
ols5.mspe <- mean( (gy - predict(ols5, newdata=gendata))^2 )
predlas <- predict.lars(las, newx=gx, type="fit", mode="fraction", s=frac)
las1.mspe <- mean( (gy - predlas$fit)^2 )
predlas <- predict.lars(las, newx=gx, type="fit", mode="fraction", s=0.56)
las2.mspe <- mean( (gy - predlas$fit)^2 )
predlas <- predict.lars(las, newx=gx, type="fit", mode="fraction", s=1)
las3.mspe <- mean( (gy - predlas$fit)^2 )
# I can't find a function which obtains predicted values
# using an lm.ridge object, so I will compute them as
# shown below.  (Note: The ym component's value is the
# sample mean of the training sample response values,
# which (see the top of p. 60 of HTF) is the proper
# estimate of the intercept if the predictors were centered.)
ridge1.mspe <- mean( (gy - (gx %*% ridge1$coef + ridge1$ym))^2 )
ridge2.mspe <- mean( (gy - (gx %*% ridge2$coef + ridge2$ym))^2 )
ridge3.mspe <- mean( (gy - (gx %*% ridge3$coef + ridge3$ym))^2 )
ridge4.mspe <- mean( (gy - (gx %*% ridge4$coef + ridge4$ym))^2 )

#
genpcdata <- data.frame(pcomp.gen)
names(genpcdata) <- c("pc1", "pc2", "pc3", "pc4", "pc5", "pc6")
detach(gendata)
attach(genpcdata)
pcr1.mspe <- mean( (gy - predict(pcr1, newdata=genpcdata))^2 )
pcr4.mspe <- mean( (gy - predict(pcr4, newdata=genpcdata))^2 )

#
# Here are the estimated mean squared prediction errors.
# (I'll start by giving the ones which are equiv. to the
# OLS fit based on all of the predictors.)
ridge4.mspe
### ridge (lambda for OLS equiv.)
las3.mspe
### lasso (fraction for OLS equiv.)
pcr1.mspe
### prin. comp. reg. (using all p.c's --- equiv. to ols1)
ols1.mspe
### OLS (all variables)
ols5.mspe
### OLS (2 var's from backwards elimination (also stepwise))
pcr4.mspe
### prin. comp. reg. (using 3 p.c's)
las1.mspe
### lasso (fraction by c-v)
las2.mspe
### lasso (fraction by C_p)
ridge1.mspe
### ridge (lambda by HKB)
ridge2.mspe
### ridge (lambda by L-W)
ridge3.mspe
### ridge (lambda by gen. c-v)
# The last value is the smallest of them all.
# While the prediction error was only descreased
# by about 8%, it should be kept in mind that
# there is a relatively large irreducible error
# (due to the variance of the error term).
# If the mean irreducible error, which is equal
# to 1, is subtracted from all of the estimated
# errors, then this last result indicates that
# the reducible error was decreased by about 50%.
```