Homework 2

Tianyu Liu (301249861)

2023-10-23

Problem Set 7, Applications

1

(a)

```
AQ <- na.omit(airquality[,1:4])
AQ$TWcp = AQ$Temp*AQ$Wind
AQ$TWrat = AQ$Temp/AQ$Wind
head(AQ)
```

```
##   Ozone Solar.R Wind Temp  TWcp    TWrat
## 1    41     190  7.4   67 495.8 9.054054
## 2    36     118  8.0   72 576.0 9.000000
## 3    12     149 12.6   74 932.4 5.873016
## 4    18     313 11.5   62 713.0 5.391304
## 7    23     299  8.6   65 559.0 7.558140
## 8    19      99 13.8   59 814.2 4.275362
```

```
library(MASS)

###############################################################
# Ridge regression using lm.ridge()
#
#  Running sequence from 0 to 100 by a small increment may be a little expensive
#  but probably covers range.
ridge <- lm.ridge(Ozone ~ ., lambda = seq(0, 100, .05), data = AQ)
select(ridge)
```

```
## modified HKB estimator is 0.3437551
## modified L-W estimator is 1.534077
## smallest value of GCV  at 0.2
```

```
(coef.ri.best <- coef(ridge)[which.min(ridge$GCV), ])
```

```
##                      Solar.R          Wind          Temp          TWcp
## -161.63123617    0.06284069    6.82529896    2.45651021   -0.10883212
##         TWrat
##    1.64685249
```

```
coef.ri.best
```

```
##                      Solar.R          Wind          Temp          TWcp
## -161.63123617    0.06284069    6.82529896    2.45651021   -0.10883212
##         TWrat
##    1.64685249
```

(b)

```
# Compare MSPE to MSPE from lm
mod.lm <- lm(Ozone ~ ., data = AQ)
coef.ls <- coef(mod.lm)

length(which(coef.ls[2:6] - coef.ri.best[2:6] > 0)) /length(coef.ls[2:6])
```

```
## [1] 0.6
```

3 of the 5 parameter estimates (60%) are smaller than the least square estimate.

2

(a)

```
library(glmnet)
```

```
## Loading required package: Matrix
```

```
## Loaded glmnet 4.1-8
```

```
y<- AQ[,1]
x<- as.matrix(AQ[, c(2:6)])

# Fit LASSO by glmnet(y=, x=). Gaussian is default, but other families are available
#  Function produces series of fits for many values of lambda.

lasso <- glmnet(y = y, x = x, family = "gaussian")

# cv.glmnet() uses crossvalidation to estimate optimal lambda

cv.lasso <- cv.glmnet(y = y, x = x, family = "gaussian")
cv.lasso
```

```
##
## Call:  cv.glmnet(x = x, y = y, family = "gaussian")
##
## Measure: Mean-Squared Error
##
##     Lambda Index Measure    SE Nonzero
## min  0.007    89   418.4 120.3       5
## 1se  8.651    12   529.2 137.4       2
```

(b)

```
coef(cv.lasso, s = cv.lasso$lambda.min)
```

```
## 6 x 1 sparse Matrix of class "dgCMatrix"
##                        s1
## (Intercept) -183.90117408
## Solar.R        0.06354556
## Wind           8.88369468
## Temp           2.78797591
## TWcp          -0.13800612
## TWrat          1.43082832
```

```
coef(cv.lasso, s = cv.lasso$lambda.1se)
```

```
## 6 x 1 sparse Matrix of class "dgCMatrix"
##                      s1
## (Intercept) -42.3947165
## Solar.R       .
## Wind          .
## Temp          0.8586877
## TWcp          .
## TWrat         1.8785311
```

Note that the 1se model shrinked 3 of the parameters (Solar and TWcp) and they weren't included in the model, while the λ-min model shrinked only 1 (Wind)

(c)

```
  step <- step(
    object = lm(y ~ 1, data = AQ), scope = list(upper = mod.lm), direction = "both",
    k = log(nrow(AQ)), trace = 0)
coef.step <- coef(step)
coef.step
```

```
##  (Intercept)        TWrat         Temp      Solar.R
## -93.30421016   2.86326061   1.25230788   0.05959571
```

The λ-min model has 4 of the 5 variables, the 1se model has 2 of the 5, and the hybrid stepwise has 3 of the 5.

3

(a),(b),(c),(d)

```
set.seed(2928893)
### Let's define a function for constructing CV folds
get.folds <- function(n, K) {
  ### Get the appropriate number of fold labels
  n.fold <- ceiling(n / K) # Number of observations per fold (rounded up)
  fold.ids.raw <- rep(1:K, times = n.fold) # Generate extra labels
  fold.ids <- fold.ids.raw[1:n] # Keep only the correct number of labels

  ### Shuffle the fold labels
  folds.rand <- fold.ids[sample.int(n)]

  return(folds.rand)
}

get.MSPE <- function(Y, Y.hat) {
  return(mean((Y - Y.hat)^2))
}

### Number of folds
K <- 10

### Construct folds
n <- nrow(AQ) # Sample size
folds <- get.folds(n, K)

### Create a container for MSPEs. Let's include ordinary least-squares
### regression for reference
all.models <- c("Ridge", "LASSO-Min", "LASSO-1se")
all.MSPEs <- array(0, dim = c(K, length(all.models)))
colnames(all.MSPEs) <- all.models

### Begin cross-validation
for (i in 1:K) {
  ### Split data
  data.train <- AQ[folds != i, ]
  data.valid <- AQ[folds == i, ]
  n.train <- nrow(data.train)

  ### Get response vectors
  Y.train <- data.train$Ozone
  Y.valid <- data.valid$Ozone

  #########################################################################
  ### Let's do ridge regression. This model is fit using the    ###
  ### lm.ridge() function in the MASS package. We will need to make a ###
  ### list of candidate lambda values for the function to choose      ###
  ### from. Prediction also has some extra steps, but we'll discuss   ###
### that when we get there.                                        ###
########################################################################

### Make a list of lambda values. The lm.ridge() function will
### then choose the best value from this list. Use the seq()
### function to create an equally-spaced list.
lambda.vals <- seq(from = 0, to = 100, by = 0.05)

### Use the lm.ridge() function to fit a ridge regression model. The
### syntax is almost identical to the lm() function, we just need
### to set lambda equal to our list of candidate values.
fit.ridge <- lm.ridge(Ozone ~ .,
  lambda = lambda.vals,
  data = data.train
)

### To get predictions, we need to evaluate the fitted regression
### equation directly (sadly, no predict() function to do this for us).
### You could do this using a for loop if you prefer, but there is
### a shortcut which uses matrix-vector multiplication. The syntax
### for this multiplication method is much shorter.

### Get best lambda value and its index
### Note: Best is chosen according to smallest GCV value. We can
###       get GCV from a ridge regression object using $GCV
ind.min.GCV <- which.min(fit.ridge$GCV)
lambda.min <- lambda.vals[ind.min.GCV]

### Get coefficients corresponding to best lambda value
### We can get the coefficients for every value of lambda using
### the coef() function on a ridge regression object
all.coefs.ridge <- coef(fit.ridge)
coef.min <- all.coefs.ridge[ind.min.GCV, ]

### We will multiply the dataset by this coefficients vector, but
### we need to add a column to our dataset for the intercept and
### create indicators for our categorical predictors. A simple
### way to do this is using the model.matrix() function from last
### week.
matrix.valid.ridge <- model.matrix(Ozone ~ ., data = data.valid)

### Now we can multiply the data by our coefficient vector. The
### syntax in R for matrix-vector multiplication is %*%. Note that,
### for this type of multiplication, order matters. That is,
### A %*% B != B %*% A. Make sure you do data %*% coefficients.
### For more information, see me in a Q&A session or, better still,
### take a course on linear algebra (it's really neat stuff)
pred.ridge <- matrix.valid.ridge %*% coef.min

### Now we just need to calculate the MSPE and store it
MSPE.ridge <- get.MSPE(Y.valid, pred.ridge)
all.MSPEs[i, "Ridge"] <- MSPE.ridge
```

```
#######################################################################
### Now we can do the LASSO. This model is fit using the glmnet()   ###
### or cv.glmnet() functions in the glmnet package. LASSO also has  ###
### a tuning parameter, lambda, which we have to choose.            ###
### Fortunately, the cv.glmnet() function does CV internally, and   ###
### lets us automatically find the 'best' value of lambda.          ###
#######################################################################

### The cv.glmnet() function has different syntax from what we're
### used to. Here, we have to provide a matrix with all of our
### predictors, and a vector of our response. LASSO handles
### the intercept differently, so we want to make sure our data
### matrix does not include an intercept (then let cv.glmnet() add
### an intercept later). Unfortunately, the model.matrix() function
### gets confused if we ask it to construct indicators for our
### categorical predictors without also including an intercept.
### A simple way to fix this is to create the data matrix with an
### intercept, then delete the intercept.
matrix.train.raw <- model.matrix(Ozone ~ ., data = data.train)
matrix.train <- matrix.train.raw[, -1]

### The cv.glmnet() function creates a list of lambda values, then
### does CV internally to choose the 'best' one. 'Best' can refer to
### either the value of lambda which gives the smallest CV-MSPE
### (called the min rule), or the value of lambda which gives the
### simplest model that gives CV-MSPE close to the minimum (called
### the 1se rule). The cv.glmnet() function gets both of these
### lambda values.
all.LASSOs <- cv.glmnet(x = matrix.train, y = Y.train)

### Get both 'best' lambda values using $lambda.min and $lambda.1se
lambda.min <- all.LASSOs$lambda.min
lambda.1se <- all.LASSOs$lambda.1se

### cv.glmnet() has a predict() function (yay!). This predict function
### also does other things, like get the coefficients, or tell us
### which predictors get non-zero coefficients. We are also able
### to specify the value of lambda for which we want our output
### (remember that, with ridge, we got a matrix of coefficients and
### had to choose the row matching our lambda). Strangely, the name
### of the input where we specify our value of lambda is s.

### Get the coefficients for our two 'best' LASSO models
coef.LASSO.min <- predict(all.LASSOs, s = lambda.min, type = "coef")
coef.LASSO.1se <- predict(all.LASSOs, s = lambda.1se, type = "coef")

### Get which predictors are included in our models (i.e. which
### predictors have non-zero coefficients)
included.LASSO.min <- predict(all.LASSOs,
  s = lambda.min,
    type = "nonzero"
  )
  included.LASSO.1se <- predict(all.LASSOs,
    s = lambda.1se,
    type = "nonzero"
  )

  ### Get predictions from both models on the validation fold. First,
  ### we need to create a predictor matrix from the validation set.
  ### Remember to include the intercept in model.matrix(), then delete
  ### it in the next step.
  matrix.valid.LASSO.raw <- model.matrix(Ozone ~ ., data = data.valid)
  matrix.valid.LASSO <- matrix.valid.LASSO.raw[, -1]
  pred.LASSO.min <- predict(all.LASSOs,
    newx = matrix.valid.LASSO,
    s = lambda.min, type = "response"
  )
  pred.LASSO.1se <- predict(all.LASSOs,
    newx = matrix.valid.LASSO,
    s = lambda.1se, type = "response"
  )

  ### Calculate MSPEs and store them
  MSPE.LASSO.min <- get.MSPE(Y.valid, pred.LASSO.min)
  all.MSPEs[i, "LASSO-Min"] <- MSPE.LASSO.min

  MSPE.LASSO.1se <- get.MSPE(Y.valid, pred.LASSO.1se)
  all.MSPEs[i, "LASSO-1se"] <- MSPE.LASSO.1se
}

all.MSPEs
```

```
##           Ridge LASSO-Min LASSO-1se
##  [1,]  280.6840  314.2722  415.7364
##  [2,]  363.7170  316.8250  236.9153
##  [3,]  539.3697  586.1201  717.2987
##  [4,]  108.3192  114.6189  157.0742
##  [5,]  185.4001  190.1739  323.5662
##  [6,]  371.7604  373.4553  668.8504
##  [7,]  219.6295  217.3012  366.8750
##  [8,]  587.6590  585.7460  643.3282
##  [9,] 1042.9887 1045.5646 1125.8526
## [10,]  725.3818  650.6659  377.7573
```

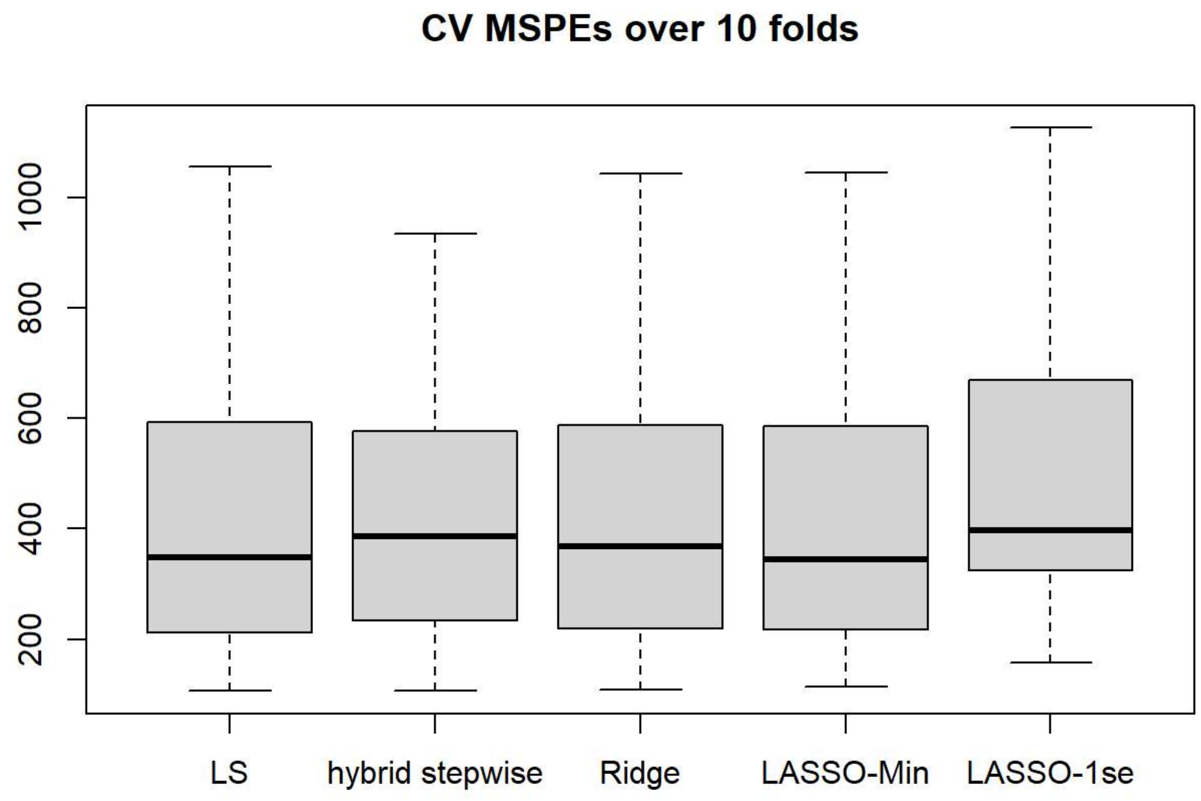
```
avg.MSPEs = colMeans(all.MSPEs)
avg.MSPEs
```

```
##     Ridge LASSO-Min LASSO-1se
##  442.4909  439.4743  503.3254
```

On average, ridge regression produces the smallest prediction error.

(e)

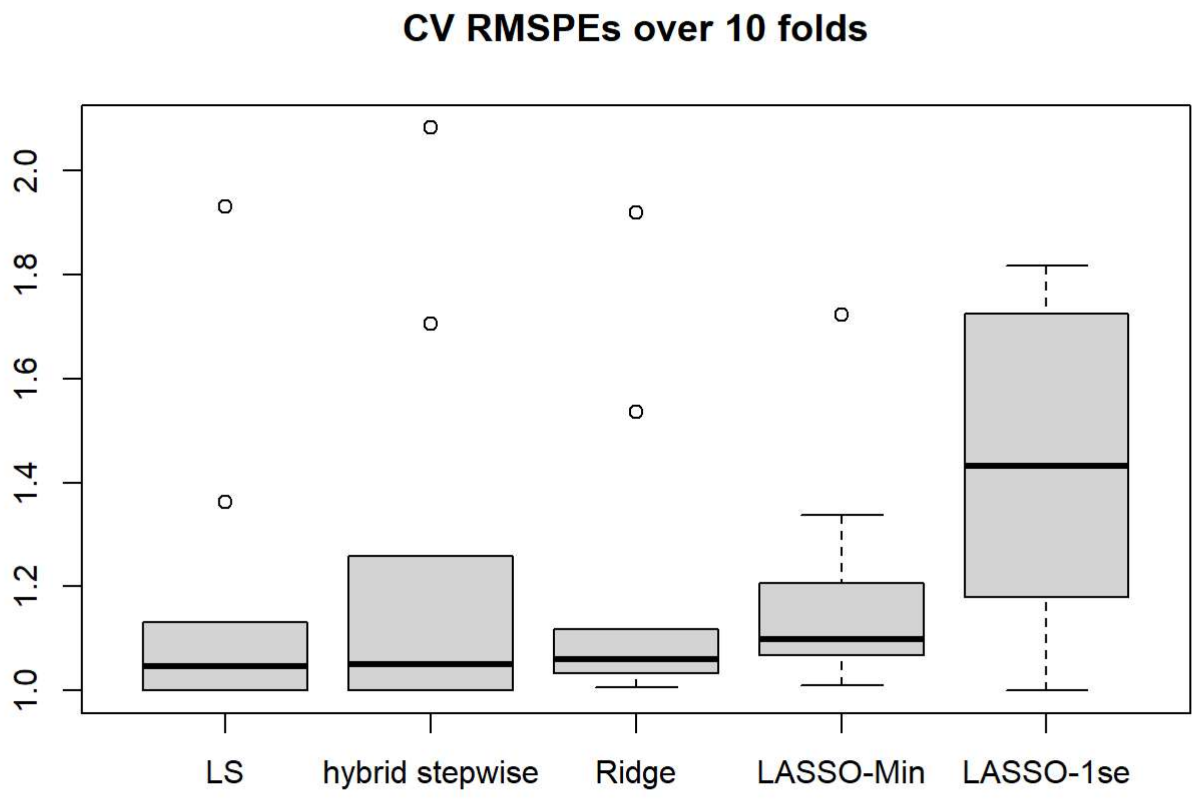
```r
library(stringr)

all.MSPEs.LS.step <- array(0, dim = c(K, 2))
colnames(all.MSPEs.LS.step) <- c("LS", "hybrid stepwise")

coefs.sw <- matrix(NA, nrow = K, ncol = 11)

for (i in 1:K) {
  ### Split data
  data.train <- AQ[folds != i, ]
  data.valid <- AQ[folds == i, ]
  n.train <- nrow(data.train)

  ### Get response vectors
  Y.train <- data.train$Ozone
  Y.valid <- data.valid$Ozone

  ###################################################################
  ### First, let's quickly do LS so we have a reference point for ###
  ### how well the other models do.                               ###
  ###################################################################

  fit.ls <- lm(Ozone ~ ., data = data.train)
  pred.ls <- predict(fit.ls, newdata = data.valid)
  MSPE.ls <- get.MSPE(Y.valid, pred.ls)
  all.MSPEs.LS.step[i, "LS"] <- MSPE.ls

  step <- step(
    object = lm(Ozone ~ 1, data = data.train), scope = list(upper = fit.ls), direction = "both",
    k = log(nrow(data.train)), trace = 0)
  pred.sw <- predict(step, as.data.frame(data.valid))
  MSPE.sw <- get.MSPE(Y.valid, pred.sw)
  all.MSPEs.LS.step[i, "hybrid stepwise"] <- MSPE.sw
}

all.MSPEs.all = cbind(all.MSPEs.LS.step, all.MSPEs)
boxplot(all.MSPEs.all, main = paste0("CV MSPEs over ", K, " folds"))
```



There isn’t any big differences. The least-square and ridge models have lower MSPE but higher variance, whereas the LASSO-1se model has the lowest variance (but has a clear outlier).

(f)

```
all.RMSPEs <- apply(all.MSPEs.all, 1, function(W) {
  best <- min(W)
  return(W / best)
})
all.RMSPEs <- t(all.RMSPEs)
boxplot(all.RMSPEs, main = paste0("CV RMSPEs over ", K, " folds"))
```



The least-square and ridge models behave the best still.

Problem Set 8, Applications (Ozone Data)

1

(a),(b),(c)

```
library(pls)
```

```
##
## Attaching package: 'pls'
```

```
## The following object is masked from 'package:stats':
##
##     loadings
```

```r
all.MSPEs.pls <- array(0, dim = c(K, 1))
colnames(all.MSPEs.pls) <- c("PLS")

n_comps = array(0, dim = c(K,1))

for (i in 1:K) {
  ### Split data
  data.train <- AQ[folds != i, ]
  data.valid <- AQ[folds == i, ]
  n.train <- nrow(data.train)

  ### Get response vectors
  Y.train <- data.train$Ozone
  Y.valid <- data.valid$Ozone

  ### Now, let's do PLS using the plsr() function. The syntax is
  ### very similar to lm(). If we set validation = "CV", the plsr()
  ### function will do its own internal CV, and give MSPEs for each
  ### number of components. We can then use this to choose how many
  ### componenets to keep when doing prediction on the validation
  ### fold. We can use an optional input called segments to specify
  ### how many folds we want plsr() to use for its internal CV
  ### (default is 10).
  fit_pls <- plsr(Ozone ~ .,
    data = data.train, validation = "CV",
    segments = 5
  )

  ### Investigate the fitted PLS model. Comment out the next two
  ### lines when running a CV loop

  ### The summary function gives us lots of information about how
  ### errors change as we increase the number of components
  # summary(fit.pls)

  ### The validationplot() function shows how MSPE from the internal
  ### CV of plsr() changes with the number of included components.
  # validationplot(fit.pls)

  ### Get the best model from PLS. To do this, we need to find the model
  ### that minimizes MSPE for the plsr() function's internal CV. It
  ### takes a few steps, but all the information we need is contained
  ### in the output of plsr().
  CV_pls <- fit_pls$validation
  pls_comps <- CV_pls$PRESS
  n_comps[i] <- which.min(pls_comps)

  ### Get predictions and calculate MSPE on the validation fold
  ### Set ncomps equal to the optimal number of components
  pred.pls <- predict(fit_pls, data.valid, ncomp = n_comps[i])
  MSPE.pls <- get.MSPE(Y.valid, pred.pls)
  all.MSPEs.pls[i, "PLS"] <- MSPE.pls
}

n_comps
```

```
##       [,1]
##  [1,]    5
##  [2,]    3
##  [3,]    3
##  [4,]    3
##  [5,]    5
##  [6,]    4
##  [7,]    3
##  [8,]    3
##  [9,]    5
## [10,]    3
```

```
all.MSPEs.pls
```

```
##             PLS
##  [1,]  260.4455
##  [2,]  139.8033
##  [3,]  513.6342
##  [4,]  137.6674
##  [5,]  192.1326
##  [6,]  364.8359
##  [7,]  265.6754
##  [8,]  655.6349
##  [9,] 1055.8045
## [10,]  699.0148
```

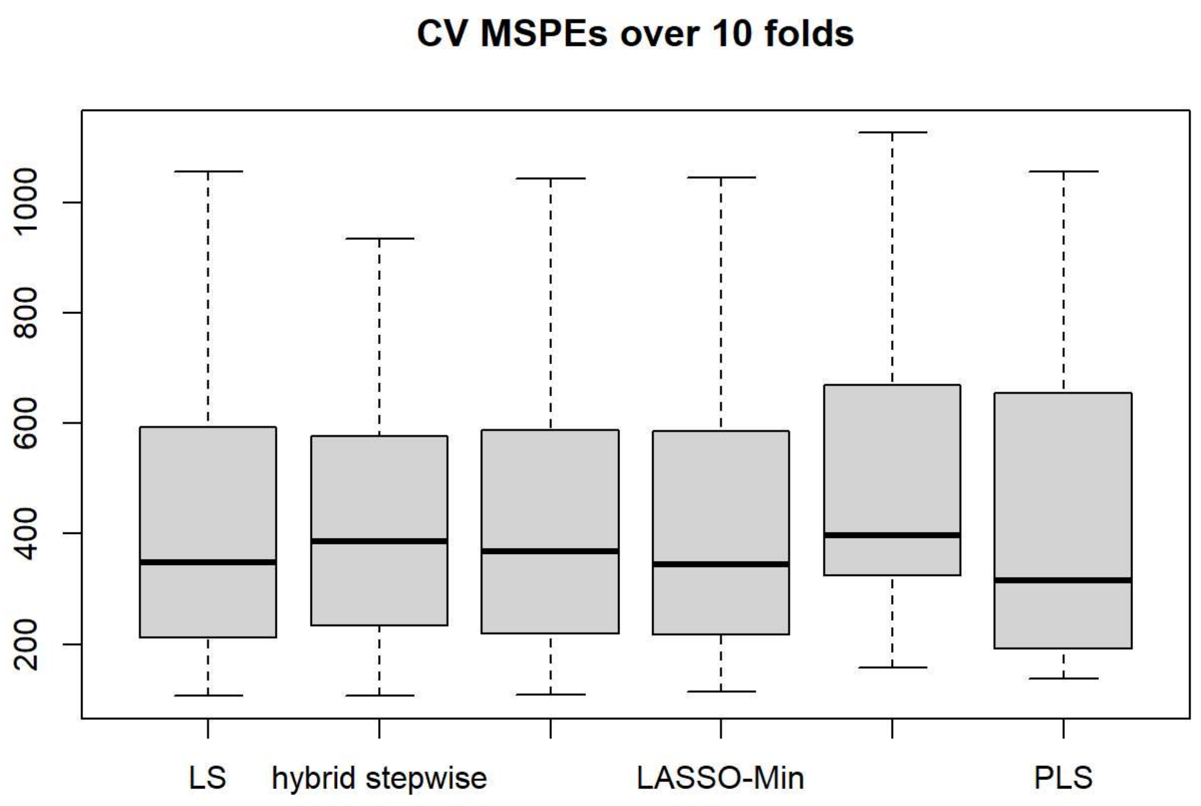
```
mean(all.MSPEs.pls)
```

```
## [1] 428.4649
```

The optimal number of components for each of the 10 folds range from 3 to 5 (5 just means that each variable is its own component). The average prediction error is 433.

(d)

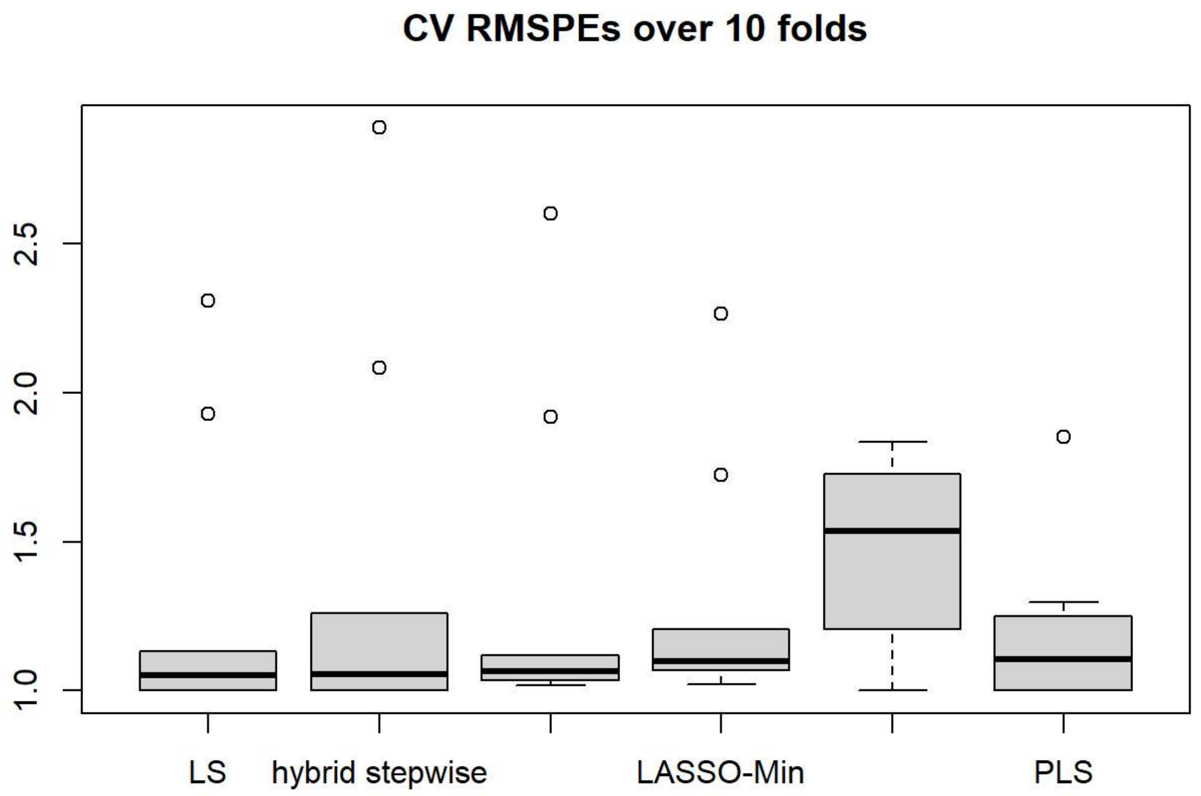
```
boxplot(cbind(all.MSPEs.all, all.MSPEs.pls), main = paste0("CV MSPEs over ", K, " folds"))
```



Compared with other models, the partial least squares estimate has the lowest MSPE and variance, but seems to be heavily skewed.

(e)

```
all.RMSPEs.all <- apply(cbind(all.MSPEs.all, all.MSPEs.pls), 1, function(W) {
  best <- min(W)
  return(W / best)
})
all.RMSPEs.all <- t(all.RMSPEs.all)
boxplot(all.RMSPEs.all, main = paste0("CV RMSPEs over ", K, " folds"))
```



The

PLS prediction outperforms every other model, according to the relative prediction error. The LS and ridge models are also comparable to PLS.

Problem Set 9, Concepts

1

(a)

β_o is the intercept value when the explanatory variable, X, is in region 0 of the step function (region used as the baseline) in the regression model.

(b)

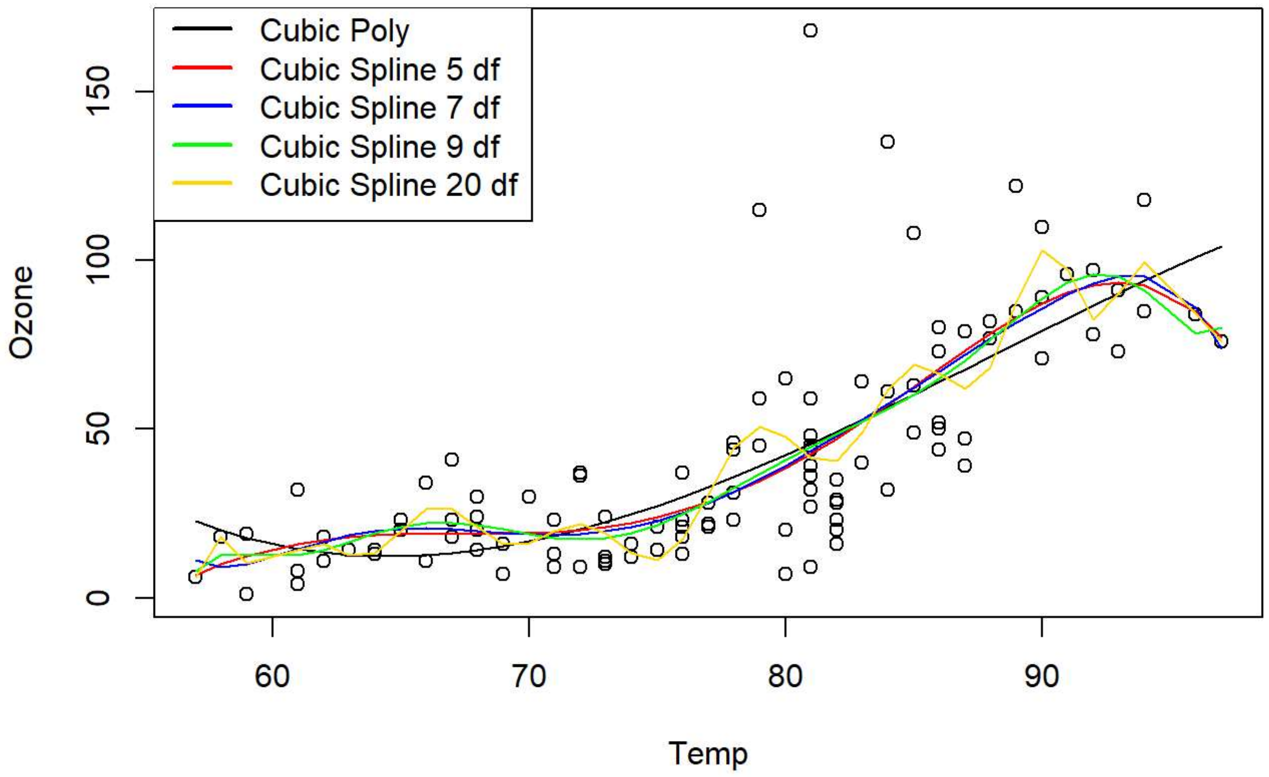
β_k is the difference of intercept values when X is in the last region (region k) and β_o, the baseline region.

Problem Set 10, Applications

1

(a)

```r
### Fit polynomial regression models
fit.poly.3 <- lm(Ozone ~ poly(Temp, degree = 3), data = AQ)

### Fit basis splines
library(splines)

fit.basis.5 <- lm(Ozone ~ bs(Temp, degree = 5), data = AQ)
fit.basis.7 <- lm(Ozone ~ bs(Temp, degree = 7), data = AQ)
fit.basis.9 <- lm(Ozone ~ bs(Temp, degree = 9), data = AQ)
fit.basis.20 <- lm(Ozone ~ bs(Temp, degree = 20), data = AQ)

### Predicting
Temp.sort <- data.frame(Temp = sort(AQ$Temp))
pred.poly.3 <- predict(fit.poly.3, Temp.sort)
pred.basis.5 <- predict(fit.basis.5, Temp.sort)
pred.basis.7 <- predict(fit.basis.7, Temp.sort)
pred.basis.9 <- predict(fit.basis.9, Temp.sort)
pred.basis.20 <- predict(fit.basis.20, Temp.sort)

### Plots
with(AQ, plot(Temp, Ozone))
lines(Temp.sort$Temp, pred.poly.3)
lines(Temp.sort$Temp, pred.basis.5, col = 'red')
lines(Temp.sort$Temp, pred.basis.7, col = 'blue')
lines(Temp.sort$Temp, pred.basis.9, col = 'green')
lines(Temp.sort$Temp, pred.basis.20, col = 'gold')
legend(x = 55, y = 180, legend = c(
    "Cubic Poly", "Cubic Spline 5 df",
    "Cubic Spline 7 df", "Cubic Spline 9 df", "Cubic Spline 20 df"
  ),
  lty = "solid", col = c('black', 'red', 'blue', 'green', 'gold'), lwd = 2
)
```



(b)

Looking at the graph, the cubic polynomial seems to have the most bias.

(c)

The 9-df and 20-df splines seem to overfit, because there is some fluctuation towards the left end of the graph where Temp is low. Also for the 20-df spline there is too much fluctuation in the middle.

(d)

I would choose the 5-df cubic spline. It seems to capture the data relatively well with no clear overfitting, and no random end-point fluctuations.