Homework 2 Solutions

Alexander McLain

Question 1 (10 points) Show that the ridge regression estimator $\boldsymbol{\beta}_{RR}(k) = \{\boldsymbol{X}'\boldsymbol{X} + \lambda \boldsymbol{I}\}^{-1}\boldsymbol{X}'\boldsymbol{Y}$ can be obtained by minimizing the loss function:

$$\phi(\boldsymbol{\beta}) = \text{ESS}(\boldsymbol{\beta}) + \lambda ||\boldsymbol{\beta}||_2$$

where $\text{ESS}(\boldsymbol{\beta}) = \sum_{i=1}^n (y_i - \boldsymbol{X}_i'\beta)^2$ and $||\boldsymbol{\beta}||_2 = \sum_{j=1}^p \beta_j^2$. Assume the data are centered so that there is no intercept term.

Solution Taking the derivative of $\phi(\boldsymbol{\beta})$ with respect to $\boldsymbol{\beta}$ and setting it equal to zero gives

$$\frac{d}{d\boldsymbol{\beta}}\phi(\boldsymbol{\beta}) = -2X'Y + 2X'X\boldsymbol{\beta} + 2\lambda\boldsymbol{\beta} = 0$$

rearranging the terms we have

$$\begin{aligned} X'Y &= X'X\boldsymbol{\beta} + \lambda\boldsymbol{\beta} \\ &= (X'X + \lambda I)\beta \end{aligned}$$

as desired $\square$.

Question 2 (10 points) Complete exercise 3.6 in the online version of ESL.

In this questions your asked to show the equivalence between the ridge regression estimator and a Bayesian model where the prior distribution of the $\boldsymbol{\beta}$ coefficients is multivariate normal distribution. The solution to this can be done by multiplying the likelihood by the prior distribution and taking the log (which gives the log posterior distribution of $\boldsymbol{\beta}$). This results in

$$-\frac{1}{2\sigma^2}\sum_{i=1}^n (y_i - \boldsymbol{X}_i'\boldsymbol{\beta})^2 - \frac{1}{2\tau}\boldsymbol{\beta}^2$$

Question 3 A wine connoisseur is interested in determine what factors impact "expert" opinion on the quality of red wine. He has collected information on acidity, citric acid levels, sugar content, chlorides, sulfur dioxide, density, pH, sulphate content, and alcohol content on 1000 Portuguese red wines and obtained and average quality rating from 5 different experts. Download the "Wine" data set and:

Split the data into "learning" (50%), "validation" (25%) and "test" (25%) sets. Be sure to set the seed so your split can be reproduced.

Solution There are many ways to do this. Here's one:

```r
set.seed(8675309)
Wine_dat <- data.frame(read.csv("Wine.csv",as.is=TRUE,header=TRUE))
index <- sample(1:1000,1000,replace=FALSE)
Wine_trn <- Wine_dat[index[1:500],]
Wine_val <- Wine_dat[index[501:750],]
Wine_tst <- Wine_dat[index[751:1000],]
```

Part a (5 points) Using the learning set obtain the OLS estimates from the full model, and from 'forward' and 'backward' selection models. Calculate the EPE for full, forward and backward methods using the validation data.

Solution First, we'll fit the full model

```
full_mod <- lm(quality ~ .,data=Wine_trn)
summary(full_mod)
```

```
##
## Call:
## lm(formula = quality ~ ., data = Wine_trn)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.86369 -0.64220 -0.03782  0.58578  2.68985
##
## Coefficients:
##                    Estimate Std. Error t value Pr(>|t|)
## (Intercept)       52.468307  51.039881   1.028 0.304465
## acidity            0.082019   0.062436   1.314 0.189581
## volatile_acidity  -0.927648   0.292732  -3.169 0.001626 **
## citric_acid       -0.282029   0.353766  -0.797 0.425712
## sugar              0.040745   0.038250   1.065 0.287296
## chlorides         -3.006360   1.058487  -2.840 0.004696 **
## free_SO2          -0.001751   0.005842  -0.300 0.764571
## tot_SO2           -0.005002   0.001951  -2.563 0.010672 *
## density          -48.731600  52.006180  -0.937 0.349204
## pH                -0.020704   0.468779  -0.044 0.964790
## sulphates          1.175366   0.280453   4.191  3.3e-05 ***
## alcohol            0.222551   0.064195   3.467 0.000573 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.9073 on 488 degrees of freedom
## Multiple R-squared:  0.2534, Adjusted R-squared:  0.2366
## F-statistic: 15.06 on 11 and 488 DF,  p-value: < 2.2e-16
```

```
EPE_full <- mean((Wine_val$quality - predict(full_mod,newdata=Wine_val))^2)
```

Now, we'll do this for backward and forward selection

```
library(MASS)
bck_mod <- stepAIC(full_mod,direction="backward",trace=FALSE)
summary(bck_mod)
```

```
##
## Call:
## lm(formula = quality ~ acidity + volatile_acidity + chlorides +
##     tot_SO2 + sulphates + alcohol, data = Wine_trn)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.80168 -0.64977 -0.04076  0.57538  2.70055
##
## Coefficients:
##                   Estimate Std. Error t value Pr(>|t|)
## (Intercept)       3.875916   0.534729   7.248 1.64e-12 ***
## acidity           0.037640   0.024479   1.538 0.124781    
## volatile_acidity -0.872743   0.244245  -3.573 0.000387 ***
## chlorides        -3.194145   0.987877  -3.233 0.001306 ** 
## tot_SO2          -0.005421   0.001300  -4.170 3.59e-05 ***
## sulphates         1.109995   0.274336   4.046 6.04e-05 ***
## alcohol           0.257735   0.041914   6.149 1.61e-09 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.9048 on 493 degrees of freedom
## Multiple R-squared:  0.2499, Adjusted R-squared:  0.2407 
## F-statistic: 27.37 on 6 and 493 DF,  p-value: < 2.2e-16
```

```r
null_mod <- lm(quality ~ 1, data=Wine_trn)
frw_mod <- stepAIC(null_mod,direction="forward",scope=list(upper=full_mod,
                                                      lower=null_mod),trace=FALSE)
summary(frw_mod)
```

```
##
## Call:
## lm(formula = quality ~ alcohol + volatile_acidity + tot_SO2 + 
##     sulphates + chlorides + acidity, data = Wine_trn)
##
## Residuals:
##      Min       1Q   Median       3Q      Max 
## -2.80168 -0.64977 -0.04076  0.57538  2.70055 
##
## Coefficients:
##                   Estimate Std. Error t value Pr(>|t|)    
## (Intercept)       3.875916   0.534729   7.248 1.64e-12 ***
## alcohol           0.257735   0.041914   6.149 1.61e-09 ***
## volatile_acidity -0.872743   0.244245  -3.573 0.000387 ***
## tot_SO2          -0.005421   0.001300  -4.170 3.59e-05 ***
## sulphates         1.109995   0.274336   4.046 6.04e-05 ***
## chlorides        -3.194145   0.987877  -3.233 0.001306 ** 
## acidity           0.037640   0.024479   1.538 0.124781    
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.9048 on 493 degrees of freedom
## Multiple R-squared:  0.2499, Adjusted R-squared:  0.2407 
## F-statistic: 27.37 on 6 and 493 DF,  p-value: < 2.2e-16
```

```r
EPE_bck <- mean((Wine_val$quality - predict(bck_mod,newdata=Wine_val))^2)
EPE_frw <- mean((Wine_val$quality - predict(frw_mod,newdata=Wine_val))^2)
```

Part b (5 points) For all values k (there are 11 predictors in the data), find the subset of variables that minimizes the residual sums of squares using the training data. Then use the validation data to determine which k minimized the EPE.

```r
K_EPE_info <- NULL
best_mod <- matrix(0,11,11)
best_mod_num <- matrix(0,11,11)
for(k in 1:11){
  # First, for each value of k we'll find all possible subsets of variables (adding the
  # outcome of course)
  Win_vars <- cbind(12,t(combn(1:11,k,simplify=TRUE)))
  #Now for every possible subset combination we'll fit a model and find the RSS.
  RSS_vec <- NULL
  for(j in 1:length(Win_vars[,1])) {
    #Create a small dataset with only our variables combination and the outcome
    t_Wine_trn <- Wine_trn[,Win_vars[j,]]
    #Fit the model and get the RSS
    t_Wine_mod <- lm(quality ~ .,data=t_Wine_trn)
    RSS <- sum((t_Wine_mod$residuals)^2)
    #Store the RSS
    RSS_vec <- c(RSS_vec,RSS)
  }
  ## Now find the combination with the minimum and store the variables.
  best_mod[k,1:k] <- colnames(Wine_trn)[Win_vars[which.min(RSS_vec),-1]]
  best_mod_num[k,1:k] <- Win_vars[which.min(RSS_vec),-1]
  #Re-run the best model
  t_Wine_trn <- Wine_trn[,Win_vars[which.min(RSS_vec),]]
  best_ls_k <- lm(quality ~ .,data=t_Wine_trn)
  #Calculate the EPE based on the validation data
  EPE_sbt_k <- mean((Wine_val$quality - predict(best_ls_k,newdata=Wine_val))^2)
  #Store the EPE results
  K_EPE_info <- rbind(K_EPE_info,c(k,EPE_sbt_k))
}

# Now display k, the EPE and the best model
results <- data.frame(K_EPE_info,best_mod)
colnames(results) <- c("k","EPE","V1","V2","V3","V4","V5","V6","V7","V8","V9","V10","V11")
results[,1:8]
```

k	EPE	V1	V2	V3	V4	V5	V6
1	0.7812226	alcohol	0	0	0	0	0
2	0.7506010	volatile_acidity	alcohol	0	0	0	0
3	0.7535430	volatile_acidity	tot_SO2	alcohol	0	0	0
4	0.7571852	volatile_acidity	tot_SO2	sulphates	alcohol	0	0
5	0.7696914	volatile_acidity	chlorides	tot_SO2	sulphates	alcohol	0
6	0.7716446	acidity	volatile_acidity	chlorides	tot_SO2	sulphates	alcohol
7	0.7713205	acidity	volatile_acidity	citric_acid	chlorides	tot_SO2	sulphates
8	0.7787340	acidity	volatile_acidity	sugar	chlorides	tot_SO2	density
9	0.7784136	acidity	volatile_acidity	citric_acid	sugar	chlorides	tot_SO2
10	0.7789033	acidity	volatile_acidity	citric_acid	sugar	chlorides	free_SO2
11	0.7794227	acidity	volatile_acidity	citric_acid	sugar	chlorides	free_SO2

```
results[,9:13]
```

V7	V8	V9	V10	V11
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0

V7	V8	V9	V10	V11
alcohol	0	0	0	0
sulphates	alcohol	0	0	0
density	sulphates	alcohol	0	0
tot_SO2	density	sulphates	alcohol	0
tot_SO2	density	pH	sulphates	alcohol

```
# The best of the best models is
results[which.min(results$EPE),1:(results[which.min(results$EPE),1]+2)]
```

	k	EPE	V1	V2
2	2	0.750601	volatile_acidity	alcohol

```
# or variables numbers
best_mod_num[which.min(results$EPE),1:(results[which.min(results$EPE),1])]
```

```
## [1]  2 11
```

```
#Let's now refit the best of the best so we can keep it for later
Wine_trn_best <- Wine_trn[,c(12,best_mod_num[which.min(results$EPE),
                                             1:(results[which.min(results$EPE),1])])]
best_sbt_mod <- lm(quality ~ .,data=Wine_trn_best)
summary(best_sbt_mod)
```

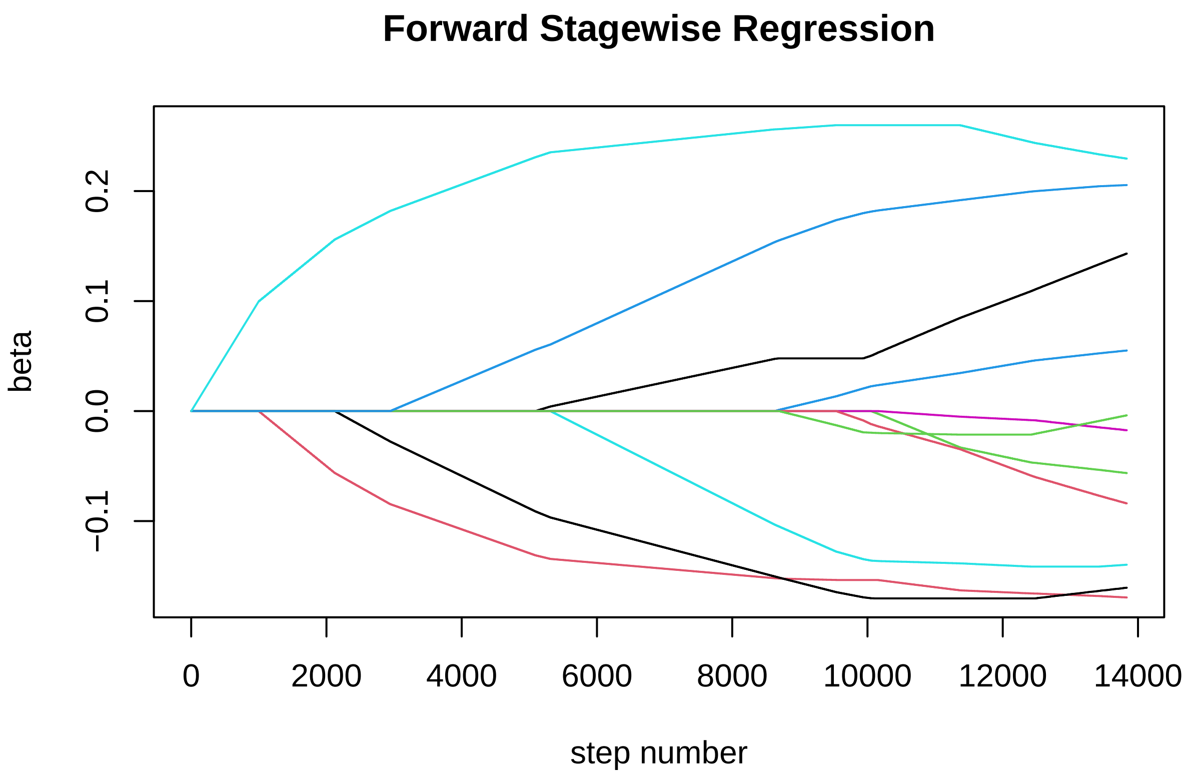
```
##
## Call:
## lm(formula = quality ~ ., data = Wine_trn_best)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.57249 -0.66471 -0.03686  0.62565  2.83827
##
## Coefficients:
##                  Estimate Std. Error t value Pr(>|t|)
## (Intercept)       3.86612    0.46056   8.394 4.91e-16 ***
## volatile_acidity -1.33982    0.23215  -5.771 1.38e-08 ***
## alcohol           0.33496    0.04125   8.120 3.69e-15 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.9361 on 497 degrees of freedom
## Multiple R-squared:  0.1906, Adjusted R-squared:  0.1874
## F-statistic: 58.52 on 2 and 497 DF,  p-value: < 2.2e-16
```

Part c (5 points) Analyze the training data using the forward stagewise algorithm. Use the R code from the example in class, not the built in function. For each step, calculate the EPE using the validation data. This may be time consuming if your step-size (i.e., eps in the example R code) is too small (might want to use eps=0.01).

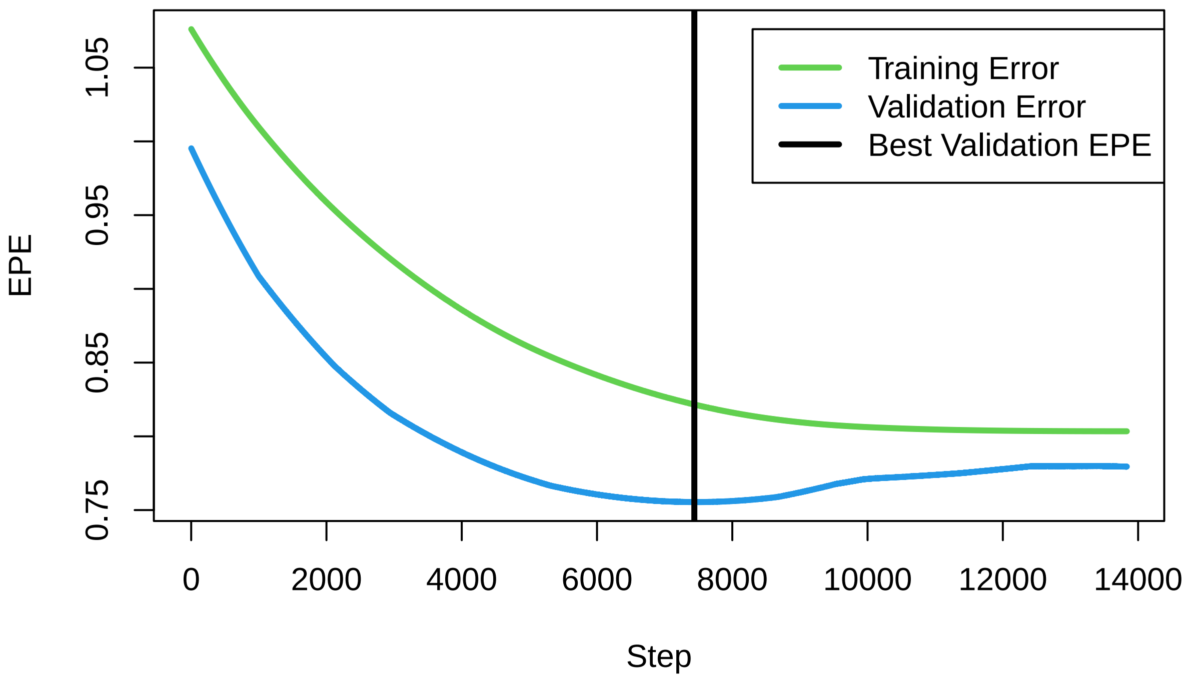
Solution We'll start with the code from class, but here I'm going to make the function run until the max absolute correlation is smaller then some value:

```r
# Load in the training and validation data.  Note that I'm standardizing both using the
# means and standard deviations from the training data
y <- Wine_trn$quality
M <- as.matrix(Wine_trn[,-12])

y_val <- Wine_val$quality
M_val <- as.matrix(Wine_val[,-12])

y_tst <- Wine_tst$quality
M_tst <- as.matrix(Wine_tst[,-12])

# Note that I'm using y and M (i.e., the training data) to standardize the data.
y_val <- y_val-mean(y)
M_val <- M_val-matrix(apply(M,2,mean),ncol=ncol(M_val),nrow=nrow(M_val),byrow=T)
M_val <- M_val/matrix(apply(M,2,sd),ncol=ncol(M_val),nrow=nrow(M_val),byrow=T)

y_tst <- y_tst-mean(y)
M_tst <- M_tst-matrix(apply(M,2,mean),ncol=ncol(M_tst),nrow=nrow(M_tst),byrow=T)
M_tst <- M_tst/matrix(apply(M,2,sd),ncol=ncol(M_tst),nrow=nrow(M_tst),byrow=T)

y <- y-mean(y)
M <- M-matrix(apply(M,2,mean),ncol=ncol(M),nrow=nrow(M),byrow=T)
M <- M/matrix(apply(M,2,sd),ncol=ncol(M),nrow=nrow(M),byrow=T)

beta <- matrix(0,ncol=ncol(M),nrow=1)
r <- y
eps <- 0.0001
# Changing the 'for' loop to a 'while' loop, which will continue until the maximum
# absolute correlation is less than or equal to kappa.
kappa <- 0.0001
max_cor = 1
steps = 0
EPE_trn = NULL
EPE_val = NULL
while(max_cor > kappa){
  # Note I change what was used in the function to using the built in Cor function.
  co <- cor(M,r)
  max_cor <- max(abs(co))
  j <- (1:ncol(M))[abs(co)==max_cor][1]
  delta <- eps*sign(co[j])
  b <- beta[nrow(beta),]
  b[j] <- b[j] + delta
  beta <- rbind(beta,b)
  r <- r - delta*M[,j]
  # For each iteration let's calculate the EPE from the training and validation data
  pred_trn <- M%*%b
  EPE_trn <- c(EPE_trn,mean((y-pred_trn)^2))

  pred_val <- M_val%*%b
  EPE_val <- c(EPE_val,mean((y_val-pred_val)^2))
  steps = steps + 1
}
```

```r
# Let's take a look at the trip taken by the beta coefficients.
matplot(beta,type="l",lty=1,xlab="step number",ylab="beta",
        main="Forward Stagewise Regression")
```



```r
# Now let's look at the training and validation EPE
plot(EPE_trn,type = "l",xlab="Step",ylab = expression(EPE),
     ylim = range(c(EPE_val,EPE_trn)),lwd=3,col=3)
lines(EPE_val,lwd=3,col=4)
legend(steps*0.6,max(c(EPE_val,EPE_trn)),
legend = c("Training Error", "Validation Error", "Best Validation EPE"), lwd=3,col=c(3,4,1))
abline(v=which.min(EPE_val),lwd=3)
```



```r
# Now, let's see the coefficients for the best model
names(M)
```

```
## NULL
```

```r
best_beta <- beta[which.min(EPE_val),]
data.frame(cbind(colnames(M),round(best_beta,5)))
```

X1	X2
acidity	0.0319
volatile_acidity	-0.1457
citric_acid	0
sugar	0
chlorides	-0.0662
free_SO2	0
tot_SO2	-0.1311
density	0
pH	0
sulphates	0.1202
alcohol	0.2487

Part d (5 points) Using the best models from (a)–(c), reestimate the EPE using the test data. Make a table that has the regression coefficients for the **best** models from (a)–(c), and the EPE estimates (from validation and test data).

```r
pred_tst <- M_tst%*%beta[which.min(EPE_val),]
Fwd_stage_EPE_tst <- mean((y_tst-pred_tst)^2)
full_mod_EPE <- mean((Wine_tst$quality- predict(full_mod,newdata = Wine_tst))^2)
bck_mod_EPE <- mean((Wine_tst$quality- predict(bck_mod,newdata = Wine_tst))^2)
fwd_mod_EPE <- mean((Wine_tst$quality- predict(frw_mod,newdata = Wine_tst))^2)
best_sbt_EPE <- mean((Wine_tst$quality- predict(best_sbt_mod,newdata = Wine_tst))^2)
```

```r
fin_res <- data.frame(c(full_mod_EPE,bck_mod_EPE,fwd_mod_EPE,best_sbt_EPE,
                        Fwd_stage_EPE_tst))
row.names(fin_res) <- c("Full Model","Backward Selection","Forward Selection",
                        "Best Subsets","Forward Stagewise")
colnames(fin_res) <- c("EPE from test data")
fin_res
```

	EPE from test data
Full Model	0.8253563
Backward Selection	0.8262375
Forward Selection	0.8262375
Best Subsets	0.8246723
Forward Stagewise	0.8007460

Part e (5 points) Comment on the results in (d) and state hich model would you recommend to use to predict wine quality?

For my results the forward stagewise model had the best fit, with the best subsets model having the second best fit. The best subsets model only had 2 parameters while the forward stagewise had 6. You have to think about whether the extra covariates (and greater model complexity) is worth the gain in EPE.

Question 4 For this exercise we'll use the "Communities and Crime Data Set" (link to info). This data set contains variables related to violent crime. The goal is to predict the variable "ViolentCrimesPerPop". The first five variables in the dataset should not be used in the regression models.

Split the data into "learning"" (50%), "validation" (25%) and "test" (25%) sets. Be sure to set the seed so your split can be reproduced.

```r
# Here, I load in the data, remove the first 5 columns.
comm <- read.csv("communities.csv", head=FALSE)
comm <- comm[,!names(comm) %in% c("V1","V2","V3","V4",'V5')]
head(comm[,90:100])
```

V95	V96	V97	V98	V99	V100	V101	V102	V103	V104	V105
0.04	0	0.12	0.42	0.50	0.51	0.64	0.03	0.13	0.96	0.17
0.00	0	0.21	0.50	0.34	0.60	0.52	NA	NA	NA	NA
0.00	0	0.14	0.49	0.54	0.67	0.56	NA	NA	NA	NA
0.00	0	0.19	0.30	0.73	0.64	0.65	NA	NA	NA	NA
0.00	0	0.11	0.72	0.64	0.61	0.53	NA	NA	NA	NA
0.00	0	0.70	0.42	0.49	0.73	0.64	NA	NA	NA	NA

```r
##only keep rows with complete data
nrow(comm)
```

```
## [1] 1994
```

```r
comm <- comm[complete.cases(comm),]
nrow(comm)
```

```
## [1] 319
```

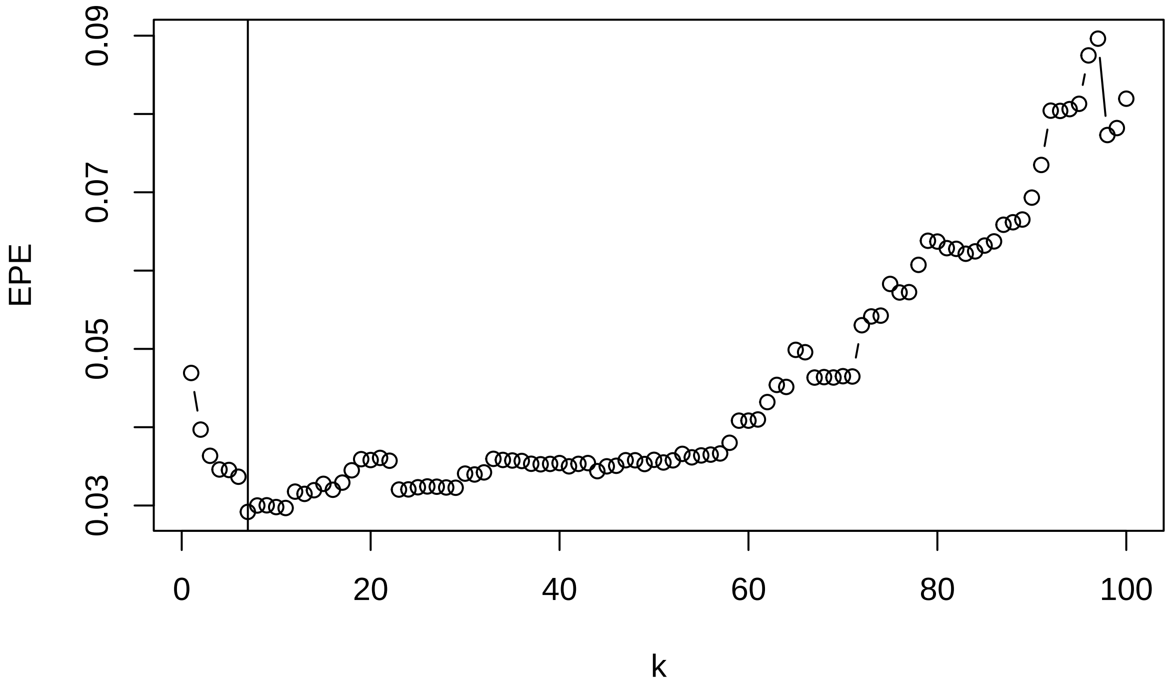
```r
##split data into learning, validation, and test sets (note: always set your seed!!)
set.seed(4)
index <- sample(c(1,2,3), nrow(comm),
                replace=TRUE,
                prob = c(1/2,1/4,1/4))
comm_tr <- comm[index==1,]
comm_va <- comm[index==2,]
comm_ts <- comm[index==3,]
```

Part a (5 points) Analyze the data using PCR. Chose which K works the best based on the validation data.

```
#PCA
X_tr <- comm_tr[,!(colnames(comm_tr)=="V128")]
X_va <- comm_va[,!(colnames(comm_tr)=="V128")]
X_ts <- comm_ts[,!(colnames(comm_tr)=="V128")]

#PCR
EPE_pcr <- NULL
for(i in 1:100){
  comm_PCR <- pcr(comm_tr$V128~as.matrix(X_tr), i, scale=TRUE, center=TRUE)
  comm_PCR_va <- predict(comm_PCR,list(X_tr = X_va))
  diff <- comm_PCR_va[,,i] - comm_va$V128
  EPE_pcr[i] <- mean(diff^2, na.rm=TRUE)
}

PCR_K <- which.min(EPE_pcr)
c(PCR_K, EPE_pcr[PCR_K])
```

```
## [1] 7.0000000 0.0291856
```

```
plot(1:100,EPE_pcr,type="b",xlab = "k",ylab = "EPE")
abline(v=PCR_K)
```



```
comm_PCR <- pcr(comm_tr$V128~as.matrix(X_tr),PCR_K, scale=TRUE, center=TRUE)
```

Here, the model with $K = 7$ PC worked the best for principal component regression.

Part b (10 points) Analyze the data using PLS. Chose which K works the best based on the validation data.

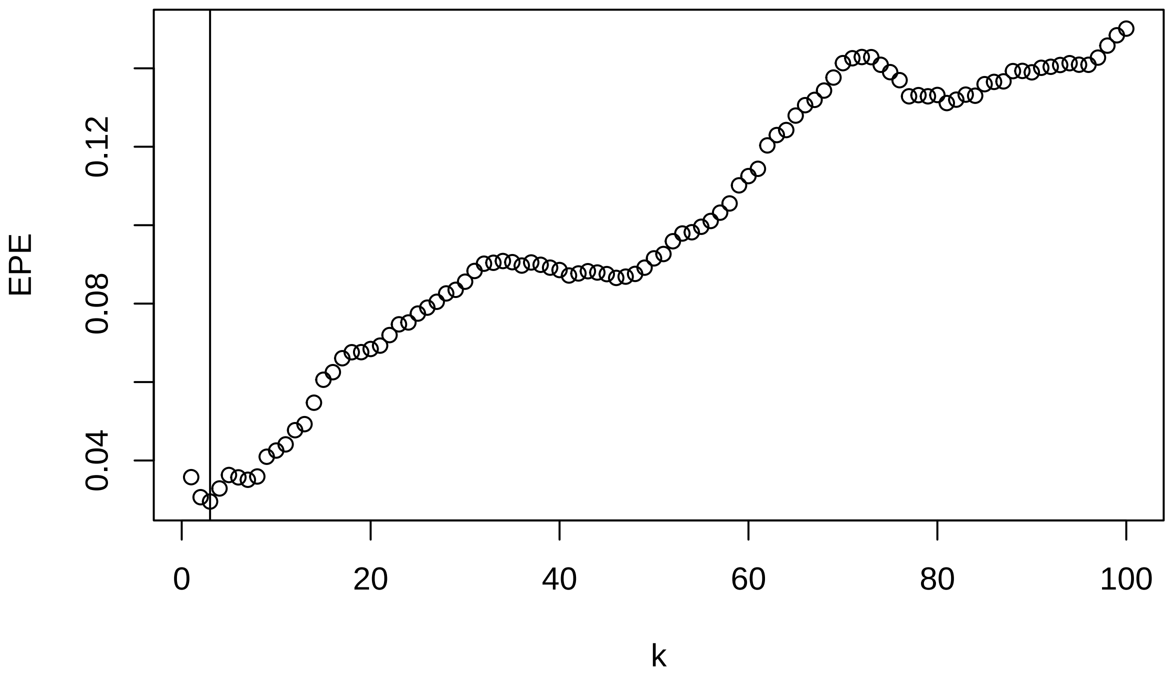
```
#PLS
EPE_plsr <- NULL
for(i in 1:100){
  comm_PLSR <- plsr(comm_tr$V128~as.matrix(X_tr), i, scale=TRUE, center=TRUE)
  comm_PLSR_va <- predict(comm_PLSR,list(X_tr = X_va))
  diff <- comm_PLSR_va[,,i] - comm_va$V128
  EPE_plsr[i] <- mean(diff^2, na.rm=TRUE)
}


PLS_K <- which.min(EPE_plsr)
EPE_plsr[PLS_K]
```

```
## [1] 0.02953941
```

```
c(PLS_K, EPE_plsr[PCR_K])
```

```
## [1] 3.00000000 0.03508526
```

```
plot(1:100,EPE_plsr,type="b",xlab = "k",ylab = "EPE")
abline(v=PLS_K)
```



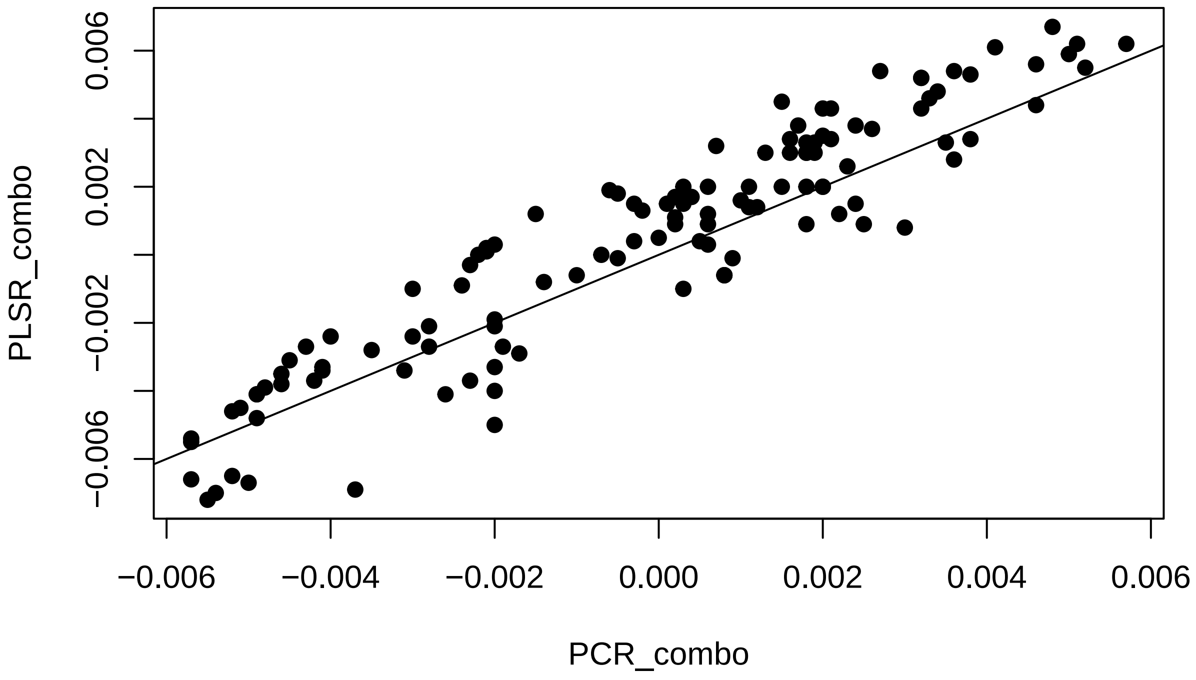
```
comm_PLSR <- plsr(comm_tr$V128~as.matrix(X_tr), PLS_K, scale=TRUE, center=TRUE)
```

Here, the model with K=3 linear combinations worked the best for parial least squares.

Part c (5 points) Compare the first linear combination for PCR and PLS. Plot the results and comment on the differences.

```
#first linear combination for PCR
PCR_combo <- (round(comm_PCR$coefficients[1:122, ,1 ], 4))
PLSR_combo <- (round(comm_PLSR$coefficients[1:122, ,1 ], 4))
```

```r
plot(PCR_combo, PLSR_combo, pch=19)
lines(c(-1,1),c(-1,1))
```



There's a strong linear relationship between the first components of the PCR and the PLS linear combinations.

Part d (5 points) Using the best models from (a) and (b), compare the results using the test data.

```r
#best model
#PCR
comm_PCR_ts <- predict(comm_PCR,list(X_tr = X_ts))
diff <- comm_PCR_ts[,,PCR_K] - comm_ts$V128
EPE_pcr_best <- mean(diff^2, na.rm=TRUE)

#PLSR
comm_PLSR_ts <- predict(comm_PLSR,list(X_tr = X_ts))
diff <- comm_PCR_ts[,,PLS_K] - comm_ts$V128
EPE_plsr_best <- mean(diff^2, na.rm=TRUE)

res <- cbind(c(PCR_K,PLS_K),c(EPE_pcr[PCR_K],EPE_plsr[PLS_K]),c(EPE_pcr_best, EPE_plsr_best))
colnames(res) <- c("K","Validation EPE", "Test EPE")
row.names(res) <- c("PCR","PLS")
res
```

	K	Validation EPE	Test EPE
PCR	7	0.0291856	0.0315580
PLS	3	0.0295394	0.0391953

```r
EPE_plsr_best/EPE_pcr_best
```

```
## [1] 1.242007
```

Part e (5 points) Which model do you think should be used? You should base this on the results and interpretability.

The PCR model bests the PLS model by a notable amount. We do want simpler models, and the PLS has fewer linear combinations, but here the difference is substantial enough for me to choose the PCR, which has EPE that is over 20% lower.

Question 5 This dataset is composed of a range of biomedical voice measurements from 42 people with early-stage Parkinson's disease recruited to a six-month trial of a telemonitoring device for remote symptom progression monitoring.

Each row corresponds to one of 5,875 voice recording from these individuals. The main aim of the data is to predict the motor and total UPDRS scores ('motor_UPDRS' and 'total_UPDRS') from the 16 voice measures. Here we'll focus on total UPDRS score.

Part a (2 points) Center and scale the 16 voice measurements. Then for each voice measurement create a squared and cubic terms. Thus each variable will then have 3 columns (X, X^2, X^3). The total design matrix will have $48 = 3 \times 16$ columns.

```r
data.in <- read.csv("parkinsons_updrs.txt",as.is=TRUE,header=TRUE)

names(data.in) <- c(
   "subject",
   "age",
   "sex",
   "test_time",
   "motor_UPDRS",
   "total_UPDRS",
   "Jitter",
   "Jitter.Abs",
   "Jitter.RAP",
   "Jitter.PPQ5",
   "Jitter.DDP",
   "Shimmer",
   "Shimmer.dB",
   "Shimmer.APQ3",
   "Shimmer.APQ5",
   "Shimmer.APQ11",
   "Shimmer.DDA",
   "NHR",
   "HNR",
   "RPDE",
   "DFA",
   "PPE"
)

data1 <- as.data.frame(apply(data.in[,7:22],2,scale))

sqr <- function(x) { x^2 }
data1.2 <- as.data.frame(apply(data1,2,sqr))

names(data1.2) <- c(
   "Jitter2",
   "Jitter.Abs2",
   "Jitter.RAP2",
   "Jitter.PPQ52",
   "Jitter.DDP2",
   "Shimmer2",
   "Shimmer.dB2",
   "Shimmer.APQ32",
   "Shimmer.APQ52",
   "Shimmer.APQ112",
   "Shimmer.DDA2",
   "NHR2",
   "HNR2",
   "RPDE2",
   "DFA2",
   "PPE2"
)

cbe <- function(x) { x^3 }
data1.3 <- as.data.frame(apply(data1,2,cbe))

names(data1.3) <- c(
   "Jitter3",
   "Jitter.Abs3",
   "Jitter.RAP3",
   "Jitter.PPQ53",
   "Jitter.DDP3",
   "Shimmer3",
   "Shimmer.dB3",
   "Shimmer.APQ33",
   "Shimmer.APQ53",
   "Shimmer.APQ113",
   "Shimmer.DDA3",
   "NHR3",
   "HNR3",
   "RPDE3",
   "DFA3",
   "PPE3"
)

data.all <- cbind(UPDRS=scale(data.in$total_UPDRS),data1,data1.2,data1.3)
```

```
head(data.all[,c(1,2,18,34)])
```

UPDRS	Jitter	Jitter2	Jitter3
0.5027024	0.0828982	0.0068721	0.0005697
0.5490563	-0.5607457	0.3144357	-0.1763185
0.5953167	-0.2389237	0.0570846	-0.0136389
0.6346615	-0.1553567	0.0241357	-0.0037496
0.6874638	-0.4985149	0.2485171	-0.1238895
0.7337243	-0.4665105	0.2176321	-0.1015277

```
set.seed(8675309)
index <- sample(1:5875,5875,replace=FALSE)
park_tr <- data.all[index[1:2937],]
park_va <- data.all[index[2938:4406],]
park_ts <- data.all[index[4407:5875],]
```

Part a (5 points) Develop a ridge regression model of these data for six different values of λ. The six values of λ should be chosen so that $df(\lambda) \approx 8, 16, 24, 32, 40, 48$. Then chose the best value of λ based on the validation data.

To do this we first need to solve to find the appropriate value of λ which will result in the desired degrees of freedom. To this end, first write a function to evaluate the following

$$df(\lambda) = \text{tr}\{\boldsymbol{X}(\boldsymbol{X}'\boldsymbol{X} + \lambda\boldsymbol{I})^{-1}\boldsymbol{X}'\} = \text{tr}(\boldsymbol{H}_\lambda)$$

where $\boldsymbol{X}$ is the design matrix of the ridge regression model.

```
df <- function(lam,X,p) {
  H_lam <- X %*% solve(t(X)%*%X + lam*diag(p)) %*%t(X)
  return(sum(diag(H_lam)))
}
```

Now we need to get a function to solve $df(\lambda) = k$ for λ. Do do this, I'll create a new function that will evaluate $L_k(\lambda) = \{df(\lambda) - k\}^2$. Then I'll use the `optimize` function to minimize $L_k(\lambda)$ with respect to λ.

```
L <- function(lam,k,X,p) {
  (df(lam,X,p) - k)^2
}
```

Now we'll use `optimize` to get the values for λ for $k = 8, 16, 24, 32, 40, 48$ (note: optimize is not the best optimization function, but in this case the function is monotone so it's an easy optimization problem).

```
#Get X using the model.matrix function.
park_tr_lm <- lm(UPDRS ~ .,data=park_tr)
X <- model.matrix(park_tr_lm)
p <- dim(X)[2]
#First I tested out the function to get a maximum value I knew would contain lambda.
df(1000000,X,p)
```

```
## [1] 6.145443
```

```
L_opt <- optimize(L,c(0,1000000),X=X,p=p,k=8)
lambda_8 <- L_opt$minimum
L_opt <- optimize(L,c(0,1000000),X=X,p=p,k=16)
lambda_16 <- L_opt$minimum
L_opt <- optimize(L,c(0,1000000),X=X,p=p,k=24)
lambda_24 <- L_opt$minimum
L_opt <- optimize(L,c(0,1000000),X=X,p=p,k=32)
lambda_32 <- L_opt$minimum
L_opt <- optimize(L,c(0,1000000),X=X,p=p,k=40)
lambda_40 <- L_opt$minimum
L_opt <- optimize(L,c(0,1000000),X=X,p=p,k=48)
lambda_48 <- L_opt$minimum
data.frame(k = c(8,16,24,32,40,48), lambda = c(lambda_8, lambda_16, lambda_24, lambda_32, lambda_40, la
```

k	lambda
8	3.376765e+05
16	1.127155e+04
24	1.300101e+03
32	1.758414e+02
40	9.457064e+00
48	5.230000e-05

Now we can find the estimates using

$$\hat{\boldsymbol{\beta}}^{\text{ridge}} = (\boldsymbol{X}'\boldsymbol{X} + \lambda\boldsymbol{I})^{-1}\boldsymbol{X}'\boldsymbol{Y}$$

```r
Y <- as.matrix(park_tr[,1],drop=FALSE)
beta_8  <- solve((t(X)%*%X+lambda_8*diag(p)))%*%t(X)%*%Y
beta_16 <- solve((t(X)%*%X+lambda_16*diag(p)))%*%t(X)%*%Y
beta_24 <- solve((t(X)%*%X+lambda_24*diag(p)))%*%t(X)%*%Y
beta_32 <- solve((t(X)%*%X+lambda_32*diag(p)))%*%t(X)%*%Y
beta_40 <- solve((t(X)%*%X+lambda_40*diag(p)))%*%t(X)%*%Y
beta_48 <- solve((t(X)%*%X+lambda_48*diag(p)))%*%t(X)%*%Y
res <- data.frame(round(cbind(beta_8,beta_16,beta_24,beta_32,beta_40,beta_48),5))
colnames(res) <- c("8", "16", "24", "32", "40", "48")
res
```

	8	16	24	32	40	48
(Intercept)	-0.00011	0.00899	0.05956	0.10287	0.10916	0.11600
Jitter	0.00047	0.00554	0.01726	0.02099	-0.04546	-0.17507
Jitter.Abs	0.00044	0.00443	0.00852	-0.00935	-0.04610	-0.03592
Jitter.RAP	0.00038	0.00396	0.00826	0.00955	0.04210	-0.57177
Jitter.PPQ5	0.00037	0.00412	0.01641	0.05681	0.26383	0.42163
Jitter.DDP	0.00038	0.00396	0.00824	0.00940	0.03943	0.66691
Shimmer	0.00064	0.00712	0.01215	0.01025	0.05562	0.18459
Shimmer.dB	0.00070	0.00831	0.01600	-0.00363	-0.11028	-0.18114
Shimmer.APQ3	0.00056	0.00509	-0.00646	-0.07258	-0.11208	35.94746
Shimmer.APQ5	0.00057	0.00596	0.00768	-0.01418	-0.31255	-0.69561
Shimmer.APQ11	0.00086	0.01376	0.06155	0.22182	0.63265	0.79785
Shimmer.DDA	0.00056	0.00509	-0.00647	-0.07265	-0.11345	-36.04106
NHR	0.00035	0.00054	-0.01058	-0.08830	-0.30467	-0.30727
HNR	-0.00118	-0.01656	-0.05624	-0.11625	-0.17540	-0.18324
RPDE	0.00112	0.01072	0.01661	-0.00661	-0.02605	-0.03256
DFA	-0.00104	-0.01919	-0.06119	-0.12982	-0.19476	-0.20240
PPE	0.00122	0.02149	0.09580	0.19979	0.22999	0.22557
Jitter2	-0.00008	-0.00232	-0.00188	-0.00808	0.00148	0.03595
Jitter.Abs2	-0.00045	-0.00822	-0.03698	-0.07397	-0.08960	-0.10550
Jitter.RAP2	-0.00005	-0.00114	0.00271	0.01555	0.02335	-23.94534
Jitter.PPQ52	-0.00015	-0.00487	-0.01469	-0.04375	-0.10938	-0.14822
Jitter.DDP2	-0.00006	-0.00114	0.00265	0.01529	0.02145	23.98393
Shimmer2	-0.00009	-0.00208	-0.00088	0.01392	0.09665	0.06488
Shimmer.dB2	-0.00002	-0.00112	0.00586	0.01491	-0.00099	0.01946
Shimmer.APQ32	-0.00018	-0.00204	-0.00318	0.00641	0.00543	2.54738
Shimmer.APQ52	-0.00019	-0.00282	-0.00251	0.01184	0.08159	0.18123
Shimmer.APQ112	0.00004	0.00259	0.02135	0.00348	-0.09747	-0.13644
Shimmer.DDA2	-0.00018	-0.00204	-0.00320	0.00624	0.00225	-2.57443
NHR2	0.00006	-0.00419	0.01517	0.07458	0.12846	0.13068
HNR2	-0.00160	-0.02998	-0.11193	-0.18374	-0.19016	-0.19222
RPDE2	-0.00111	-0.01164	-0.01743	-0.01777	-0.01278	-0.01058
DFA2	-0.00084	-0.00113	0.01372	-0.00416	-0.02762	-0.02907
PPE2	0.00017	0.00703	0.05555	0.11875	0.13936	0.14257
Jitter3	0.00013	-0.00093	-0.00103	0.00105	0.00411	0.00203
Jitter.Abs3	-0.00030	-0.00004	0.00263	0.00674	0.00899	0.01082
Jitter.RAP3	0.00016	0.00069	0.00063	0.00070	0.02804	5.31313
Jitter.PPQ53	-0.00034	-0.00002	0.00132	0.00412	0.00760	0.01055

	8	16	24	32	40	48
Jitter.DDP3	0.00016	0.00063	0.00000	-0.00361	-0.03474	-5.32016
Shimmer3	0.00044	0.00436	0.00698	0.00361	-0.00900	-0.01011
Shimmer.dB3	0.00053	-0.00321	-0.00554	-0.00470	-0.00065	0.00003
Shimmer.APQ33	-0.00016	-0.00006	-0.00020	0.00015	0.01563	12.64637
Shimmer.APQ53	-0.00064	-0.00106	-0.00087	-0.00192	-0.00588	-0.01228
Shimmer.APQ113	0.00025	-0.00059	-0.00327	-0.00289	0.00397	0.00678
Shimmer.DDA3	-0.00016	-0.00008	-0.00039	-0.00148	-0.01542	-12.64318
NHR3	-0.00010	0.00045	-0.00052	-0.00436	-0.00721	-0.00741
HNR3	-0.00275	-0.00883	-0.00273	0.00788	0.01349	0.01699
RPDE3	0.00424	0.02010	0.01596	0.01346	0.01545	0.01679
DFA3	-0.00274	-0.04137	-0.07308	-0.06871	-0.05877	-0.05795
PPE3	0.00254	0.00800	-0.00487	-0.02455	-0.03271	-0.03215

Now we'll estimate the EPE on the validation data for each model.

```r
park_va_lm <- lm(UPDRS ~ .,data=park_va)
X_va <- model.matrix(park_va_lm)
Y_va <- as.matrix(park_va[,1],drop=FALSE)
Y_hat_va <- X_va%*%beta_8
EPE <- mean((Y_va - Y_hat_va)^2)

Y_hat_va <- X_va%*%beta_16
EPE <- c(EPE,mean((Y_va - Y_hat_va)^2))
Y_hat_va <- X_va%*%beta_24
EPE <- c(EPE,mean((Y_va - Y_hat_va)^2))
Y_hat_va <- X_va%*%beta_32
EPE <- c(EPE,mean((Y_va - Y_hat_va)^2))
Y_hat_va <- X_va%*%beta_40
EPE <- c(EPE,mean((Y_va - Y_hat_va)^2))
Y_hat_va <- X_va%*%beta_48
EPE <- data.frame(t(c(EPE,mean((Y_va - Y_hat_va)^2))))
colnames(EPE) <- c("8", "16", "24", "32", "40", "48")
EPE
```

8	16	24	32	40	48
0.9747262	0.9196438	0.8805554	0.8630346	0.869043	0.8947519

The model with $df = 32$ performs the best. Now let's estimate the EPE from the test data on this model.

```r
park_ts_lm <- lm(UPDRS ~ .,data=park_ts)
X_ts <- model.matrix(park_ts_lm)
Y_ts <- as.matrix(park_ts[,1],drop=FALSE)
Y_hat_ts <- X_ts%*%beta_32
EPE_RR_ts <- mean((Y_ts - Y_hat_ts)^2)
```

Part c (5 points) Develop a LASSO model of these data for six different values of λ. The six values of λ should be chosen so that s (as defined in the notes) is $s \approx 0.15, 0.3, 0.45,\ 0.6, 0.75, 0.9$. Then chose the best value of λ based on the validation data.

For this problem, I'll use a similar strategy to the previous model.

- make a function to calculate s,

- make a function that will be zero when the s for a given value of λ is equal to the desired s,
- minimize the previous function for all desired values of s, and
- caluclate the regression coefficients.

```
fit.l0 <- glmnet(X,Y,alpha=1,family="gaussian", lambda=0,standardize=FALSE,intercept=TRUE)
t0 <- sum(abs(coef(fit.l0)))

#calculates s given a value of lambda
s_lam <- function(lam,X,Y,t0) {
  fit.g <- glmnet(X,Y,alpha=1,family="gaussian", lambda=lam,standardize=FALSE,intercept=TRUE)
  t1 <- sum(abs(coef(fit.g)))
  return(t1/t0)
}
L <- function(lam,s,X,Y,t0) {
  (s_lam(lam,X,Y,t0) - s)^2
}
#Get X using the model.matrix function.
park_tr_lm <- lm(UPDRS ~ .,data=park_tr)
X <- model.matrix(park_tr_lm)
#First I tested out the function to get a maximum value I knew would contain lambda.
s_lam(0.001,X,Y,t0)
```

```
## [1] 0.7028721
```

```
L_opt <- optimize(L,c(0,1),X=X,Y=Y,s=0.15,t0=t0)
lambda_015 <- L_opt$minimum
L_opt <- optimize(L,c(0,1),X=X,Y=Y,s=0.3,t0=t0)
lambda_03 <- L_opt$minimum
L_opt <- optimize(L,c(0,1),X=X,Y=Y,s=0.6,t0=t0)
lambda_06 <- L_opt$minimum
L_opt <- optimize(L,c(0,1),X=X,Y=Y,s=0.75,t0=t0)
lambda_075 <- L_opt$minimum
L_opt <- optimize(L,c(0,1),X=X,Y=Y,s=0.9,t0=t0)
lambda_09 <- L_opt$minimum
c(lambda_015,lambda_03,lambda_06,lambda_075,lambda_09)
```

```
## [1] 2.652695e-02 6.238105e-03 1.726811e-03 6.437663e-04 7.229992e-05
```

```
data.frame(s = c(0.15,0.3,0.6,0.75,0.9), lambda = c(lambda_015,lambda_03,lambda_06,lambda_075,lambda_09)
```

s	lambda
0.15	0.0265270
0.30	0.0062381
0.60	0.0017268
0.75	0.0006438
0.90	0.0000723

Now we'll analyze the data with these values

```
Y <- as.matrix(park_tr[,1],drop=FALSE)
fit.g <- glmnet(X,Y,alpha=1,family="gaussian", lambda=lambda_015,
                standardize=FALSE,intercept=TRUE)
beta_015  <- fit.g$beta
fit.g <- glmnet(X,Y,alpha=1,family="gaussian", lambda=lambda_03,
                standardize=FALSE,intercept=TRUE)
```

```
beta_03  <- fit.g$beta
fit.g <- glmnet(X,Y,alpha=1,family="gaussian", lambda=lambda_06,
               standardize=FALSE,intercept=TRUE)
beta_06  <- fit.g$beta
fit.g <- glmnet(X,Y,alpha=1,family="gaussian", lambda=lambda_075,
               standardize=FALSE,intercept=TRUE)
beta_075  <- fit.g$beta
fit.g <- glmnet(X,Y,alpha=1,family="gaussian", lambda=lambda_09,
               standardize=FALSE,intercept=TRUE)
beta_09  <- fit.g$beta
res <- round(cbind(beta_015,beta_03,beta_06,beta_075,beta_09),5)
colnames(res) <- c("s=0.15", "s=0.3", "s=0.6", "s=0.75", "s=0.9")
res
```

```
## 49 x 5 sparse Matrix of class "dgCMatrix"
##                   s=0.15    s=0.3    s=0.6   s=0.75    s=0.9
## (Intercept)      .         .        .        .        .
## Jitter           .         .        .        .       -0.03744
## Jitter.Abs       .         .       -0.01522 -0.04530 -0.04554
## Jitter.RAP       .         .        0.07670  0.05858  0.10284
## Jitter.PPQ5      .         .        0.11379  0.23330  0.24829
## Jitter.DDP       .         .        .        .       -0.01048
## Shimmer          .         .        .        .       -0.02756
## Shimmer.dB       .         .       -0.03816 -0.11540 -0.14854
## Shimmer.APQ3     .       -0.03011  .        .       -0.03140
## Shimmer.APQ5     .         .       -0.19039 -0.41245 -0.59679
## Shimmer.APQ11   0.12641  0.28072  0.55608  0.71804  0.82958
## Shimmer.DDA      .       -0.12699 -0.25220 -0.14947 -0.00662
## NHR              .         .       -0.24614 -0.29421 -0.32239
## HNR            -0.00395 -0.07517 -0.15177 -0.17028 -0.18419
## RPDE             .         .       -0.02173 -0.02603 -0.02967
## DFA            -0.06251 -0.12446 -0.17693 -0.19097 -0.19817
## PPE             0.18078  0.22152  0.22359  0.22326  0.22886
## Jitter2          .         .       -0.04142 -0.05367 -0.05980
## Jitter.Abs2    -0.01820 -0.06053 -0.08166 -0.08396 -0.08684
## Jitter.RAP2      .        0.00097  0.00196 -0.00246 -0.00191
## Jitter.PPQ52     .       -0.00400 -0.04034 -0.07042 -0.06905
## Jitter.DDP2      .        0.00429  0.03942  0.06468  0.06744
## Shimmer2         .        0.01050  0.08504  0.12490  0.14015
## Shimmer.dB2      .         .        0.00099  0.00889  0.01752
## Shimmer.APQ32    .        0.01745  0.01385  .       -0.01255
## Shimmer.APQ52    .         .        0.02204  0.07310  0.12215
## Shimmer.APQ112   .         .       -0.07152 -0.11488 -0.14122
## Shimmer.DDA2     .         .        0.01101 -0.00571 -0.02390
## NHR2            0.00593  0.06192  0.11467  0.12437  0.13324
## HNR2           -0.13344 -0.19946 -0.19092 -0.19090 -0.19223
## RPDE2          -0.00744 -0.01478 -0.01218 -0.01069 -0.00991
## DFA2             .         .       -0.02069 -0.02699 -0.03005
## PPE2            0.02984  0.11562  0.13513  0.13964  0.14238
## Jitter3        -0.00022  .        0.00439  0.00645  0.00712
## Jitter.Abs3     0.00086  0.00532  0.00782  0.00841  0.00873
## Jitter.RAP3     0.00082  0.00076 -0.00085 -0.00164 -0.00223
## Jitter.PPQ53    0.00000  0.00147  0.00368  0.00520  0.00506
## Jitter.DDP3      .       -0.00112 -0.00361 -0.00511 -0.00497
## Shimmer3        0.00298  0.00460 -0.00458 -0.01019 -0.01174
## Shimmer.dB3     .        -0.00256 -0.00311 -0.00310 -0.00386
## Shimmer.APQ33  -0.00103 -0.00164 -0.00125 -0.00004  0.00079
## Shimmer.APQ53  -0.00080 -0.00147 -0.00141 -0.00431 -0.00777
## Shimmer.APQ113 -0.00119 -0.00289  0.00220  0.00531  0.00708
## Shimmer.DDA3    .        .       -0.00020  0.00087  0.00211
## NHR3           -0.00001 -0.00398 -0.00671 -0.00703 -0.00747
## HNR3           -0.00588  0.00593  0.01110  0.01340  0.01642
## RPDE3           0.01437  0.00999  0.01415  0.01558  0.01688
## DFA3           -0.07894 -0.07052 -0.06277 -0.05973 -0.05769
## PPE3           -0.00321 -0.02595 -0.03055 -0.03218 -0.03297
```

Now we'll estimate the EPE on the validation data for each model.

```r
park_va_lm <- lm(UPDRS ~ .,data=park_va)
X_va <- model.matrix(park_va_lm)
Y_va <- as.matrix(park_va[,1],drop=FALSE)

Y_hat_va <- X_va%*%beta_015
EPE <- mean((Y_va - Y_hat_va)^2)

Y_hat_va <- X_va%*%beta_03
EPE <- c(EPE,mean((Y_va - Y_hat_va)^2))
Y_hat_va <- X_va%*%beta_06
EPE <- c(EPE,mean((Y_va - Y_hat_va)^2))
Y_hat_va <- X_va%*%beta_075
EPE <- c(EPE,mean((Y_va - Y_hat_va)^2))
Y_hat_va <- X_va%*%beta_09
EPE <- data.frame(t(c(EPE,mean((Y_va - Y_hat_va)^2))))
colnames(EPE) <- c("s=0.15", "s=0.3", "s=0.6", "s=0.75", "s=0.9")
EPE
```

s=0.15	s=0.3	s=0.6	s=0.75	s=0.9
0.9065897	0.8897472	0.8843354	0.8907512	0.8910528

The model with $s = 0.6$ performs the best. Now let's estimate the EPE from the test data on this model.

```r
park_ts_lm <- lm(UPDRS ~ .,data=park_ts)
X_ts <- model.matrix(park_ts_lm)
Y_ts <- as.matrix(park_ts[,1],drop=FALSE)
Y_hat_ts <- X_ts%*%beta_06
EPE_lasso_ts <- mean((Y_ts - Y_hat_ts)^2)
```

Part d (5 points) Analyze the data using a grouped LASSO. Here you will have 16 groups based on the steps in (a). Describe how you identified the appropriate choice cutoff.

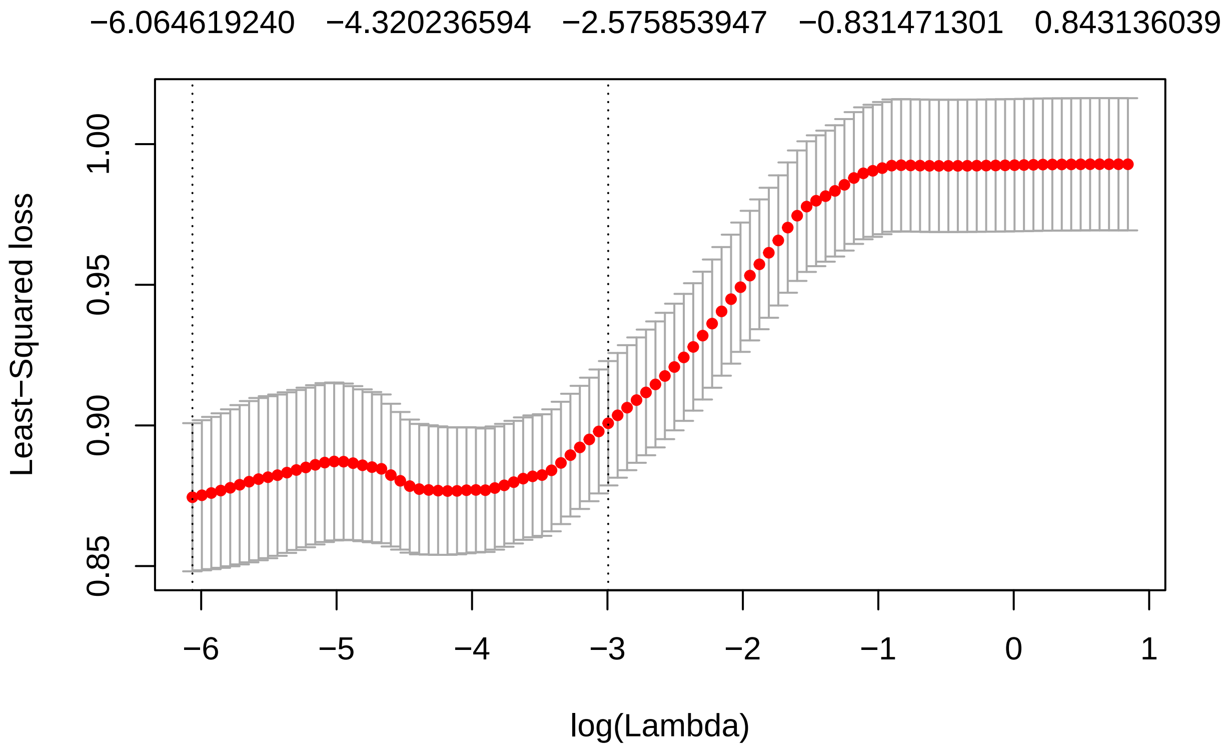
Here I'm going to fit the group lasso model and choose the best λ based on 5-fold cross validation of the data. We could use the validation to choose λ, I'm just switching it up for a change. A lovely feature of gglasso is that "Groups must be consecutively numbered 1,2,3,.." so first we'll have to rearrange the data.

```r
X_new <- NULL
for(k in 1:16){
  X_new <- cbind(X_new,X[,c(k+1,k+16+1,k+2*16+1)])
}
X_new_ts <- NULL
```

```
for(k in 1:16){
  X_new_ts <- cbind(X_new_ts,X_ts[,c(k+1,k+16+1,k+2*16+1)])
}
```

Now that we have the data arranged properly, we'll run the function. If you don't specify the λ values what `cv.gglasso` works effectively (in this example). If it doesn't work well (all λ values are too big or too small) you can use the `lambda.factor` function to fix this. The default `lambda.factor` depends on the relationship between n and p (the number of predictors). If n $\geq$ p, the default is 0.001, close to zero. If $n < p$, the defaultis 0.05. This can be increased or decreased if there is an issue.

```
group <- rep(1:16,each=3)
cv_glasso <- cv.gglasso(x = X_new, y = Y, group = group, loss = "ls",
                        pred.loss = "L2", nfolds = 5)

plot(cv_glasso)
```



Now let's look at the values λ which worked the best. And the corresponding coefficients.

```
#group lasso cutoff
cv_glasso$lambda.min
```

```
## [1] 0.002323643
```

```
cv_glasso$lambda.1se
```

```
## [1] 0.05006136
```

```
data.frame(Group = c(0, group), Beta.1se = c(coef(cv_glasso, s = "lambda.1se")),
           Beta.min = c(coef(cv_glasso, s = "lambda.min")))
```

Group	Beta.1se	Beta.min
0	0.0818200	0.1151194

Group	Beta.1se	Beta.min
1	0.0000000	0.0007158
1	0.0000000	-0.0000755
1	0.0000000	0.0017589
2	0.0002028	-0.0168775
2	-0.0004290	-0.0741992
2	-0.0005912	0.0070331
3	0.0000987	0.0131557
3	-0.0000400	0.0122199
3	0.0004141	-0.0010434
4	0.0000481	0.0809911
4	-0.0000645	-0.0526391
4	-0.0001753	0.0045815
5	0.0000000	0.0147209
5	0.0000000	0.0136513
5	0.0000000	-0.0023783
6	0.0001706	-0.0030687
6	0.0000532	0.0080978
6	0.0003123	0.0022841
7	0.0000000	-0.0040180
7	0.0000000	0.0033111
7	0.0000000	-0.0016398
8	0.0000000	-0.1153233
8	0.0000000	0.0206540
8	0.0000000	0.0001002
9	0.0002578	-0.0462219
9	-0.0000194	0.0291562
9	-0.0005644	-0.0034520
10	0.0002844	0.4013276
10	0.0001277	-0.0349822
10	0.0001906	-0.0004858
11	0.0000000	-0.1445785
11	0.0000000	0.0254932
11	0.0000000	-0.0033273
12	0.0000000	-0.1549757
12	0.0000000	0.1014286
12	0.0000000	-0.0061488
13	-0.0717507	-0.1382739
13	-0.0880424	-0.1955982
13	-0.0114719	0.0104498
14	0.0088189	-0.0160859
14	-0.0109233	-0.0156607
14	0.0166994	0.0126596
15	-0.0340409	-0.1560496
15	-0.0055078	-0.0169061
15	-0.0589728	-0.0653546
16	0.0384152	0.2280775
16	0.0122771	0.1334994
16	0.0066494	-0.0302581

```
pred_gglasso_ts <- predict(cv_glasso,newx = X_new_ts,s="lambda.min",type = "link")
EPE_gglasso_ts <- mean((c(Y_ts) - c(pred_gglasso_ts))^2)
```

Part e (5 points) Compare the findings of all model to each other and the OLS estimates from the full model (using all 48 predictors). What are the similarities and differences.

```r
#First predict for the full model
pred <- predict(park_va_lm, park_ts)
diff <- Y_ts - pred
EPE_full <- mean(diff^2)
```

```r
data.frame(EPE_RR_ts,EPE_lasso_ts,EPE_gglasso_ts,EPE_full)
```

EPE_RR_ts	EPE_lasso_ts	EPE_gglasso_ts	EPE_full
0.897852	0.9074537	0.8958459	0.9471158

The grouped lasso works the best (by a small amount). Overall, each of the penalization methods works better than the full least squares model.

Part f (5 points) What biomedical voice measures appear to have some impact on the outcome? Which model gives the 'best' answer to this question?

"Which voice measures appear to have some impact on the outcome" is a "feature selection" question. That is, which features appear to impact the outcome. The best model for this question is the Lasso model, particularly the grouped lasso model since this will give us "feature selection" on the level of the voice measures (regular lasso would be on the level of the individual voice measures and their transformations separately). For this I'll use the `lambda.1se` criteria. Let's see which measure have at least one non-zero coefficient.

```r
#Get the beta coefficients, removing the intercept
beta_vals <- coef(cv_glasso, s = "lambda.1se")[-1]
#which groups have at least one non-zero coefficient
unique(rep(1:16,each=3)[beta_vals!=0])
```

```
##  [1]  2  3  4  6  9 10 13 14 15 16
```

```r
#now let's see the variable names
variable_names <- names(park_tr)[2:17]
variable_names[unique(rep(1:16,each=3)[beta_vals!=0])]
```

```
##  [1] "Jitter.Abs"    "Jitter.RAP"    "Jitter.PPQ5"   "Shimmer"
##  [5] "Shimmer.APQ5"  "Shimmer.APQ11" "HNR"           "RPDE"
##  [9] "DFA"           "PPE"
```

```r
#The voice measurements that don't matter are
variable_names[unique(rep(1:16,each=3)[beta_vals==0])]
```

```
## [1] "Jitter"       "Jitter.DDP"   "Shimmer.dB"   "Shimmer.APQ3" "Shimmer.DDA"
## [6] "NHR"
```

Note that this isn't using any of the post-selection methods we discussed. Those would be better suited to this type of question.