Homework 3

Lin Lee Cheong

## Problem 1

1. As lambda goes to infinity, the integral term in g1 and g2 in the regularization/roughness penalty term will need to go to zero. Since g2 has more degrees of freedom it will be able to fit the data better than g1 and so g2 will have smaller training RSS.
2. Training RSS is not a good indicator for test RSS. It is not possible to know beforehand which one will be better, depending on the bias-variance tradeoff.
3. When lambda is zero, both minimization functions will be exactly the same so both functions will have the same training and test RSS.

## Problem 2 Part a)

require(ISLR)

## Loading required package: ISLR

## Warning: package 'ISLR' was built under R version 3.2.3

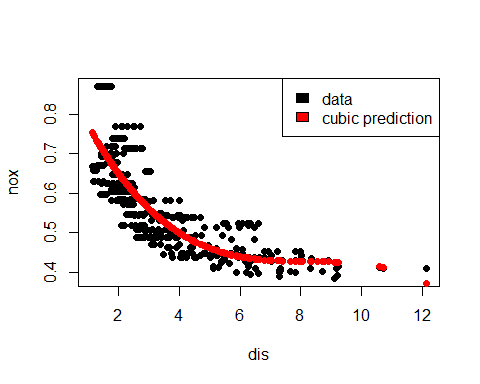
require(MASS)

## Loading required package: MASS

attach(Boston)  
  
##Part a)  
  
#cubic polynomial regression to predict NOX  
cubicModel<-lm(nox~poly(dis,degree=3,raw=TRUE))  
summary(cubicModel)

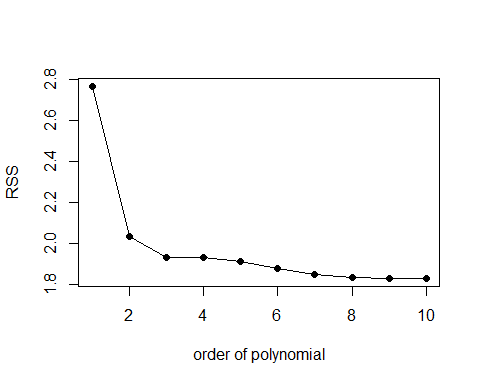
##   
## Call:  
## lm(formula = nox ~ poly(dis, degree = 3, raw = TRUE))  
##   
## Residuals:  
## Min 1Q Median 3Q Max   
## -0.121130 -0.040619 -0.009738 0.023385 0.194904   
##   
## Coefficients:  
## Estimate Std. Error t value Pr(>|t|)  
## (Intercept) 0.9341281 0.0207076 45.110 < 2e-16  
## poly(dis, degree = 3, raw = TRUE)1 -0.1820817 0.0146973 -12.389 < 2e-16  
## poly(dis, degree = 3, raw = TRUE)2 0.0219277 0.0029329 7.476 3.43e-13  
## poly(dis, degree = 3, raw = TRUE)3 -0.0008850 0.0001727 -5.124 4.27e-07  
##   
## (Intercept) \*\*\*  
## poly(dis, degree = 3, raw = TRUE)1 \*\*\*  
## poly(dis, degree = 3, raw = TRUE)2 \*\*\*  
## poly(dis, degree = 3, raw = TRUE)3 \*\*\*  
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## Residual standard error: 0.06207 on 502 degrees of freedom  
## Multiple R-squared: 0.7148, Adjusted R-squared: 0.7131   
## F-statistic: 419.3 on 3 and 502 DF, p-value: < 2.2e-16

#plot third order polynomial prediction vs actual data   
plot(dis,nox,pch=16,col='black')  
points(dis,predict(cubicModel),col='red',pch=16)  
legend('topright',legend = c('data','cubic prediction'),fill=c('black','red'))



# Part b)

orders<-seq(1:10)  
RSS<-rep(NA,length(orders))  
  
#fit model for every order, and calculate the RSS  
for (order in orders){  
 polyModel<-lm(nox~poly(dis,degree=order,raw=TRUE))  
 polyModel.pred<-predict(polyModel)  
 RSS[order]<-sum((polyModel.pred-nox)^2)  
}  
  
#The associated RSS, plotted and numbers printed out  
plot(orders,RSS,pch=16,xlab='order of polynomial')  
lines(orders,RSS)



RSS

## [1] 2.768563 2.035262 1.934107 1.932981 1.915290 1.878257 1.849484  
## [8] 1.835630 1.833331 1.832171

# Part c) CV calculations

length(nox)

## [1] 506

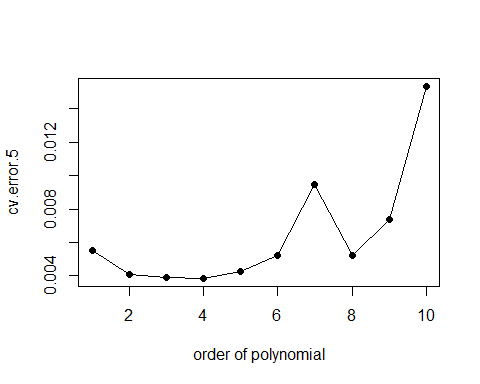
#There is 506 data points, so 5 fold cross validation is probably sufficient  
require(boot)

## Loading required package: boot

cv.error.5<-rep(NA,10)  
  
#use second delta for sample size adjusted errors  
set.seed(2016)  
for(order in orders){  
 polyModel<-glm(nox~poly(dis,degree=order,raw=TRUE))  
 cv.error.5[order]<-cv.glm(data.frame(nox=nox,dis=dis),polyModel,K=5)$delta[2]  
}  
  
#The minimum in CV suggests that a fourth order polynomial will give you the best results. Visually looking at the data though, there is almost no difference between the third and fourth order polynomial in terms of CV errors, and using the general rule of keeping it as simple as possible and 1 SE rule, second order polynomial is preferable as there is almost no difference between second to fourth orders.  
which.min(cv.error.5)

## [1] 4

plot(orders,cv.error.5,pch=16,xlab='order of polynomial')  
lines(orders,cv.error.5)



# Part d) regression spline

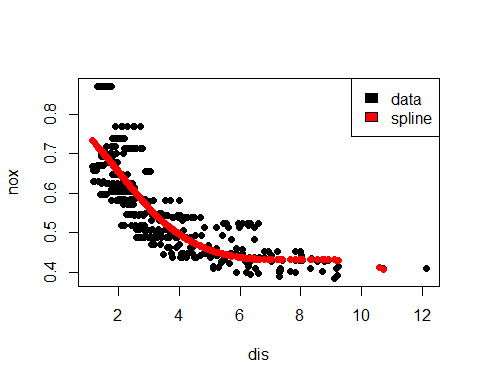
require(splines)

## Loading required package: splines

splModel<-lm(nox~bs(dis,df=4))  
#I let it pick the knots at percentiles. There is only one knot when using df=4  
attr(bs(dis,df=4),'knots')

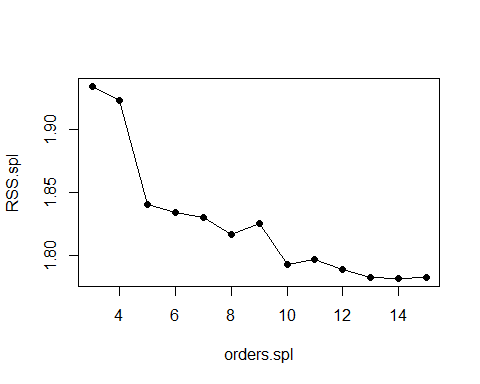
## 50%   
## 3.20745

plot(dis,nox,pch=16)  
points(dis,predict(splModel),pch=16,col='red')  
legend('topright',legend=c('data','spline'),fill=c('black','red'))



# Part e) Find RSS of spline regression

orders.spl<-seq(3,15)  
RSS.spl<-rep(NA,length(orders.spl))  
#fit model for every order, and calculate the RSS  
for (order in orders.spl){  
 splModel<-lm(nox~bs(dis,df=order))  
 splModel.pred<-predict(splModel)  
 RSS.spl[order-2]<-sum((splModel.pred-nox)^2)  
}  
  
#The training RSS reduces as the number of degrees of freedom increases generally as the model is able to fit the data better. There is a blip at df=9, probably because the function moves the knots around.  
plot(orders.spl,RSS.spl,pch=16)  
lines(orders.spl,RSS.spl)

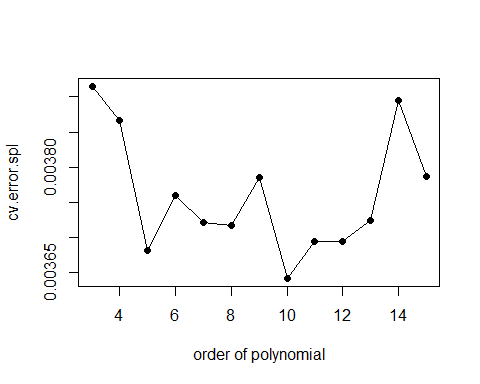


# Part f) Use 5 fold CV to find best degree of freedom on spline regression

cv.error.spl<-rep(NA,length(orders.spl))  
  
#use second delta for sample size adjusted errors  
set.seed(2016)  
for(order in orders.spl){  
 splModel<-glm(nox~bs(dis,df=order))  
 cv.error.spl[order-2]<-cv.glm(data.frame(nox=nox,dis=dis),splModel,K=5)$delta[2]  
}  
  
#Using cross-validation, it would suggest that using 10 orders of degrees of freedom will give the lowest CV error. Unlike the training RSS, continuing to increase the degrees of freedom does not always result in lower CV errors. Need to add +2 to the index of which it is minimum because I started at df=3  
which.min(cv.error.spl)

## [1] 8

plot(orders.spl,cv.error.spl,pch=16,xlab='order of polynomial')  
lines(orders.spl,cv.error.spl)

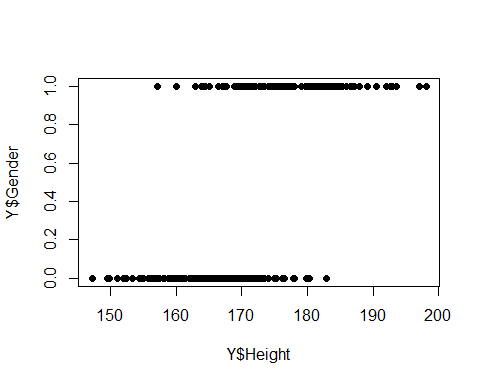


## Problem #3

# Part a) Find gender

I know in general taller people will be more likely to be men. Plotting below tells me if Gender=1, that means the person is Male as most of the gender in the plot is one when the height is above 180.

load("C:/Users/licheong/Documents/Classes\_ISLR/body.RData")  
plot(Y$Height,Y$Gender,pch=16)



# Part b)

The predictors need to be scaled to standardize each predictors. As was shown in class, the predictors (example:wrist diameter and navel girth) have very different ranges and values.

test<-sample((1:nrow(X)),200)  
train<-(1:nrow(X))[-test]  
  
#create dataframe to run pcr and plsr  
allDF<-X  
allDF$Weight<-Y$Weight  
  
require(pls)

## Loading required package: pls

## Warning: package 'pls' was built under R version 3.2.3

##   
## Attaching package: 'pls'

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

pcr.fit<-pcr(Weight~.,data=allDF,subset=train,scale=TRUE,validation='CV')  
pls.fit<-plsr(Weight~.,data=allDF,subset=train,scale=TRUE,validation='CV')

# Part c)

It looks like it takes a fair amount of components on both PCR and PLSR to explain 90% of the variance. It seems that PCR is able to explain more of the variance with the same number of components compared to PLSR, except when the total number of components is very high (>19 components). This is probably because PLSR is a supervised learning method while PCR is not, and so PLSR is probably optimizing the component choices based on Weight (which PCR is not doing).

summary(pcr.fit)

## Data: X dimension: 307 21   
## Y dimension: 307 1  
## Fit method: svdpc  
## Number of components considered: 21  
##   
## VALIDATION: RMSEP  
## Cross-validated using 10 random segments.  
## (Intercept) 1 comps 2 comps 3 comps 4 comps 5 comps 6 comps  
## CV 12.78 3.261 3.159 2.989 2.895 2.883 2.862  
## adjCV 12.78 3.259 3.156 2.957 2.890 2.878 2.856  
## 7 comps 8 comps 9 comps 10 comps 11 comps 12 comps 13 comps  
## CV 2.871 2.867 2.870 2.854 2.864 2.857 2.862  
## adjCV 2.866 2.859 2.863 2.847 2.856 2.848 2.854  
## 14 comps 15 comps 16 comps 17 comps 18 comps 19 comps  
## CV 2.855 2.851 2.826 2.795 2.718 2.732  
## adjCV 2.846 2.841 2.815 2.783 2.706 2.721  
## 20 comps 21 comps  
## CV 2.731 2.742  
## adjCV 2.719 2.729  
##   
## TRAINING: % variance explained  
## 1 comps 2 comps 3 comps 4 comps 5 comps 6 comps 7 comps  
## X 63.54 75.39 79.90 84.27 86.70 88.64 90.26  
## Weight 93.59 94.05 94.88 95.13 95.23 95.30 95.31  
## 8 comps 9 comps 10 comps 11 comps 12 comps 13 comps 14 comps  
## X 91.65 92.96 94.16 95.12 96.01 96.81 97.5  
## Weight 95.40 95.41 95.48 95.50 95.53 95.53 95.6  
## 15 comps 16 comps 17 comps 18 comps 19 comps 20 comps  
## X 98.08 98.55 98.97 99.35 99.62 99.83  
## Weight 95.62 95.77 95.99 96.13 96.13 96.15  
## 21 comps  
## X 100.00  
## Weight 96.15

summary(pls.fit)

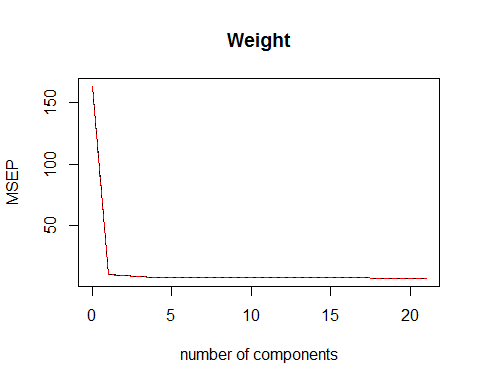
## Data: X dimension: 307 21   
## Y dimension: 307 1  
## Fit method: kernelpls  
## Number of components considered: 21  
##   
## VALIDATION: RMSEP  
## Cross-validated using 10 random segments.  
## (Intercept) 1 comps 2 comps 3 comps 4 comps 5 comps 6 comps  
## CV 12.78 3.172 2.887 2.794 2.783 2.767 2.755  
## adjCV 12.78 3.171 2.885 2.789 2.771 2.751 2.741  
## 7 comps 8 comps 9 comps 10 comps 11 comps 12 comps 13 comps  
## CV 2.740 2.738 2.740 2.743 2.743 2.743 2.743  
## adjCV 2.728 2.725 2.727 2.730 2.730 2.730 2.730  
## 14 comps 15 comps 16 comps 17 comps 18 comps 19 comps  
## CV 2.743 2.743 2.743 2.743 2.743 2.743  
## adjCV 2.730 2.730 2.730 2.730 2.730 2.730  
## 20 comps 21 comps  
## CV 2.743 2.743  
## adjCV 2.730 2.730  
##   
## TRAINING: % variance explained  
## 1 comps 2 comps 3 comps 4 comps 5 comps 6 comps 7 comps  
## X 63.53 72.98 79.53 81.25 82.9 85.75 88.31  
## Weight 93.92 95.13 95.60 95.96 96.1 96.13 96.15  
## 8 comps 9 comps 10 comps 11 comps 12 comps 13 comps 14 comps  
## X 89.78 90.94 91.71 92.62 93.64 94.66 95.47  
## Weight 96.15 96.15 96.15 96.15 96.15 96.15 96.15  
## 15 comps 16 comps 17 comps 18 comps 19 comps 20 comps  
## X 95.96 96.80 97.63 98.41 98.91 99.48  
## Weight 96.15 96.15 96.15 96.15 96.15 96.15  
## 21 comps  
## X 100.00  
## Weight 96.15

# Part d)

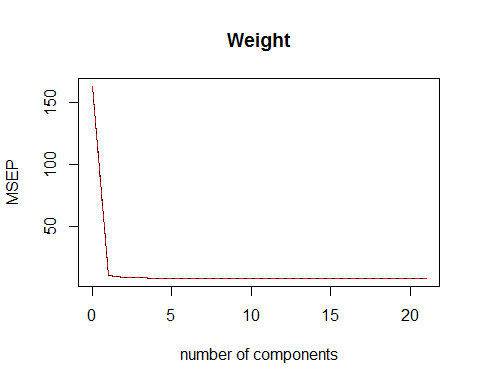
Use cross-validation data to determine number of components used, using validationplot() as described in the textbook. For PCR, there is a shallow gradual slope from using 1 components and seems to taper off around N=5. After N=5 there is still a very graduate decrease in errors but does not seem worth the massive increase in number of components.

This similar trend is observed with PLSR as well, although it seems to taper off quicker, around N=4.

validationplot(pcr.fit,val.type='MSEP')



validationplot(pls.fit,val.type='MSEP')



# Part e)

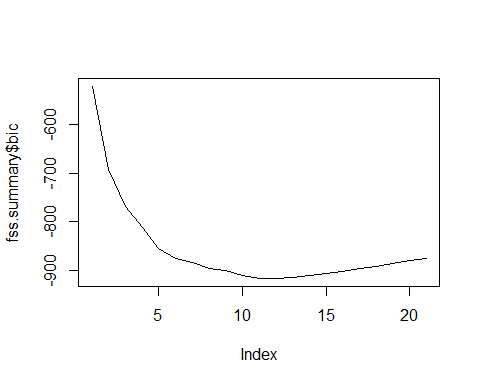
Both methods are not feature selection methods. To get to a principle component requires the linear combination of all the 21 different quantities. To do feature selection, one could use forward stepwise selection.

require(leaps)

## Loading required package: leaps

## Warning: package 'leaps' was built under R version 3.2.3

set.seed(2016)  
fss.fit<-regsubsets(Weight~.,data=allDF,subset=train,nvmax=21,method='forward')  
fss.summary<-summary(fss.fit)  
  
#select best model using bic (tends towards smaller models), which requires 12 degrees of freedom  
plot(fss.summary$bic,type='l')



which.min(fss.summary$bic)

## [1] 12

coef(fss.fit,12)

## (Intercept) Forearm.Girth Elbow.Diam Shoulder.Girth   
## -95.8405253 0.4602406 0.6685463 0.1964815   
## Biacromial.Diam Chest.Depth Chest.Girth Waist.Girth   
## 0.3249556 0.3400383 0.0860628 0.2901980   
## Pelvic.Breadth Hip.Girth Thigh.Girth Knee.Girth   
## 0.3462196 0.1935760 0.3476647 0.4508070   
## Ankle.Diam   
## 0.6434532

# Part f)

For PCR, use the first 5 principle components. For PLSR, use the first 4 principle components. For FSS, use only the features listed in part e).

pcr.pred<-predict(pcr.fit,allDF[test,],ncomp=5)  
pls.pred<-predict(pls.fit,allDF[test,],ncomp=4)  
  
#Annoyingly no predict function for forward stepwise  
coef12<-coef(fss.fit,12)  
test.mat<-model.matrix(Weight~.,data = allDF[test,])  
fss.pred<-test.mat[,names(coef12)]%\*%coef12  
  
#Compare performance in terms of mean error in predictions  
#Average error for PCR  
mean((Y$Weight[test]-pcr.pred)^2)

## [1] 9.999427

#Average error for PLS  
mean((Y$Weight[test]-pls.pred)^2)

## [1] 9.218095

#Average error for FSS  
mean((Y$Weight[test]-fss.pred)^2)

## [1] 8.987679

The test error performance above shows that PLSR performed the best, with average error of 8.6, while forward stepwise selection and PCR performed roughly the same at around 9.2.