STATS216 Homework 3

Darragh Hanley

Wednesday, February 11, 2015

1) Consider two curves, $\hat{g}_1$ and $\hat{g}_2$, defined by,

$$\hat{g}_1 = argmin_g\left(\sum_{i=1}^{n}(y_i - g(x_i))^2\right) + \lambda\int [g^{(3)}(x)]^2\,\mathrm{d}x)$$

$$\hat{g}_2 = argmin_g\left(\sum_{i=1}^{n}(y_i - g(x_i))^2\right) + \lambda\int [g^{(4)}(x)]^2\,\mathrm{d}x)$$

where $g^{(m)}$ represents the mth derivative of g.

(a) As λ -> ∞, will $\hat{g}_1$ or $\hat{g}_2$ have the smaller training RSS?

To answer this question, I will cite : Yue, Y. R., D. Simpson, F. Lindgren, and H. Rue (2012). Bayesian adaptive smoothing spline using stochastic differential equations. preprint arXiv:1209.2013 .

"The smoothing spline estimator of f can be defined as the solution to the following minimization problem:

$$\hat{f} = argmin_f\left(\sum_{i=1}^{n}(y_i - f(s_i))^2\right) + \lambda\int [f^{(p)}(s)]^2\,ds)$$

where λ > 0 is the smoothing parameter and f(p) is the pth derivative of f. The parameter λ controls the trade-off between fidelity to the data in terms of the residual sum of squares against smoothness of the fit in terms of the integrated squared derivative. ***The value of p is often taken to be 1 or 2, corresponding to linear and cubic smoothing splines, respectively****."*

Given the above citation, I would take $\hat{g}_1$ and $\hat{g}_2$ to be the equivalent of Quartic and Quintic splines respectively. **Given this $\hat{g}_2$ would have greater flexibility to fit the training data than $\hat{g}_1$, so as λ -> ∞, $\hat{g}_2$ will have the same or smaller training RSS as $\hat{g}_1$.**

(b) As λ -> ∞, will $\hat{g}_1$ or $\hat{g}_2$ have the smaller test RSS?

We cannot say which will have the smaller test error as the variance of a data set, caused by the residual error of g(x), could result in either model fitting better.

(c) For λ = 0, will $\hat{g}_1$ or $\hat{g}_2$ have the smaller training and test RSS?

For λ = 0, both $\hat{g}_1$ and $\hat{g}_2$ have the same function. Namely, both functions are equal to

$$\hat{g}_1 = \hat{g}_2 = argmin_g\left(\sum_{i=1}^{n}(y_i - g(x_i))^2 + 0\right)$$

Therefore both functions will have the same training and test RSS.

2) Suppose that we carry out backward stepwise, forward stepwise, and best subset all on the same data set. Each approach will yield a sequence of models with k = 0 up through k = p predictors.

(a) Which approach with k predictors will have the smallest test residual sum of squares? Explain.

We cannot say for certain which method will have the lowest test RSS, due to the potential for any model to over fit on the training data set.

(b) Which approach with k predictors will have the smallest training residual sum of squares? Explain.

The best subset will have the smallest training RSS, as every combination of features is exhaustively tested. This exhaustive check is not performed for forward and backward stepwise methods. It is possible that forward or backward setpwise will have the same training RSS as best subset (for eg., if they have the same combination of variables as best subset) however they cannot have a lower training RSS.

(c) True or False:

i. The predictors in the k-variable model identified by forward stepwise are a subset of the predictors in the (k + 1)-variable model identified by backward stepwise selection.

FALSE To take one example, if n = 10 and k = 3, forward stepwise will have three predictors {P1, P2, P3}. It is possible that of the 10 predictors, backward stepwise may have dropped one of these predictors, eg. P3, as a combination of other variables, serve to explain the predictions made by P3.

ii. The predictors in the k-variable model identified by backward stepwise are a subset of the predictors in the (k + 1)-variable model identified by forward stepwise selection.

FALSE To take one example, if n = 10 and k = 7, backward stepwise will have seven predictors {P1, P2, P3, P4, P5, P6, P7}. It is possible that forward stepwise may have chosen a predictor outside of this subset, eg. P8, as this could have been the unique predictor which decreased training RSS that max. Even though a combination of other variables, could serve to explain the prediction made by P8, hence it was dropped by backward setpwise.

iii. The predictors in the k-variable model identified by best subset are a subset of the predictors in the (k+1)-variable model identified by best subset selection.

FALSE Taking an example of n=10, and K=1. It is possible that best subset identifies the best single predictor to be {P1}, and the best combination of two predictors to be {P2, P3}. This may happen if {P2, P3} explain a significant amount of the prediction properties of P1, as well as other information on the response, not seen in the predictor P1.

iv. The predictors in the k-variable model identified by backward stepwise are a subset of the predictors in the (k + 1)-variable model identified by backward stepwise selection.

TRUE Taking for example, n=7 and k=4, then the k-variable model may have {P1, P2, P3, P4}, the k=3 variable model can only drop one of the predictors seen in the k=4 variable model, eg. P4, to make a k=3 selection {P1, P2, P3}. Therefore, the predictors in the k-variable (eg., k=3) model identified by backward stepwise are a subset of the predictors in the (k + 1)-variable model, (eg. k=4) identified by backward stepwise selection.

v. The predictors in the k-variable model identified by forward stepwise are a subset of the predictors in the (k + 1)-variable model identified by forward stepwise selection.

TRUE Taking for example, n=7 and k=3, Then the k-variable model may have {P1, P2, P3}, the k+1 variable model can only add one of the predictors seen in the k- variable model, eg. P4, to make a k+1 selection {P1, P2, P3, P4}. Therefore, the predictors in the k-variable model identified by forward stepwise are a subset of the predictors in the (k + 1)-variable model identified by forward stepwise selection.

3. For this problem I worked with Tony Wu, Sevvandi Kandanaarachchi and Andrew Beckerman. This question uses the variables dis (the weighted mean of distances to five Boston employment centers) and nox (nitrogen oxides concentration in parts per 10 million) from the Boston data. We will treat dis as the predictor and nox as the response.

(a) Use the poly() function to fit a cubic polynomial regression to predict nox using dis. Report the regression output, and plot the resulting data and polynomial fits.

First we load the package, and attached the data set.

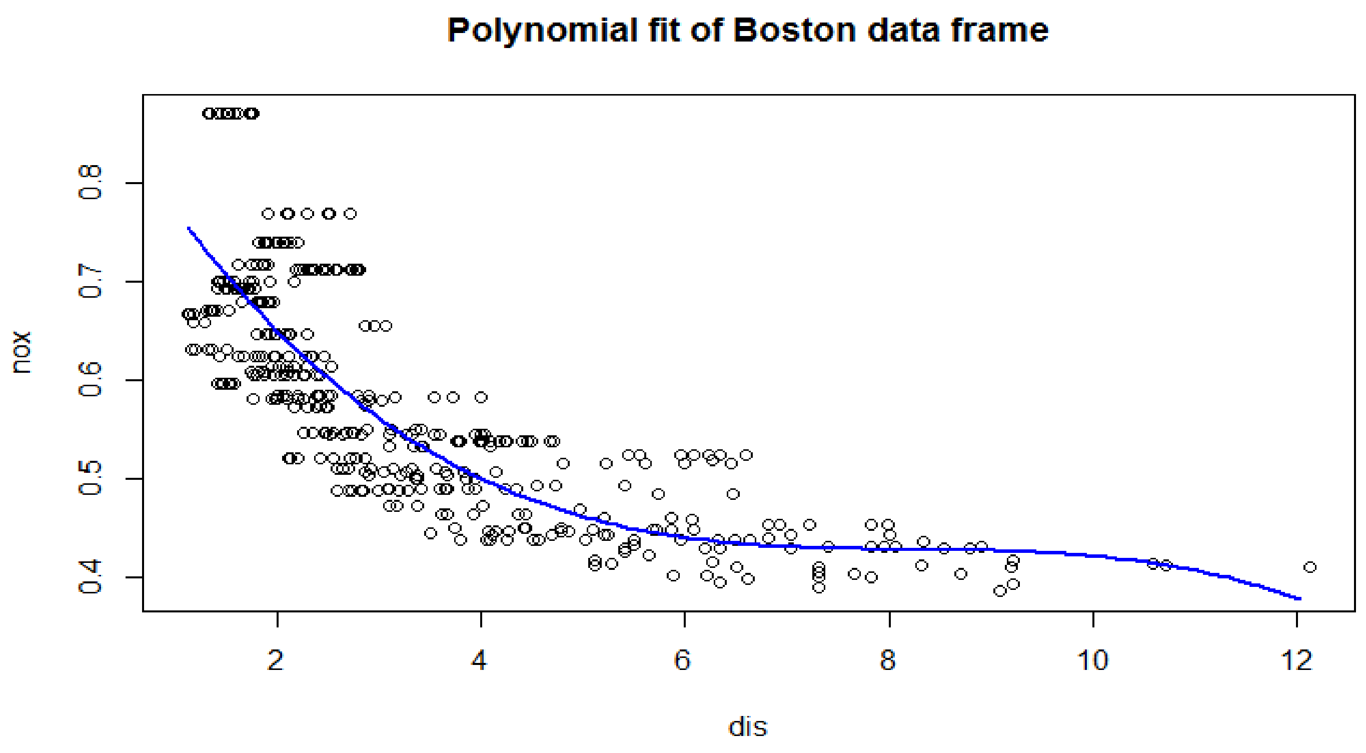
```r
library(MASS)
attach(Boston)
```

Now we fit the polynomial regression and report the regression output. Assumption is we use raw polynomials, as the basis for the fit, as opposed to orthogonal polynomials. This means we can get the direct coefficients for each degree of the fit.

```r
fit = lm(nox ~ poly(dis ,3, raw =T))
summary(fit)

##
## Call:
## lm(formula = nox ~ poly(dis, 3, raw = T))
##
## 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, 3, raw = T)1 -0.1820817  0.0146973 -12.389  < 2e-16 ***
## poly(dis, 3, raw = T)2  0.0219277  0.0029329   7.476 3.43e-13 ***
## poly(dis, 3, raw = T)3 -0.0008850  0.0001727  -5.124 4.27e-07 ***
## ---
## 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
```

So if we look at the summary, we can see that the linear, quadratic and cubic terms are significant. Lets plot the data and the polynomial fit.

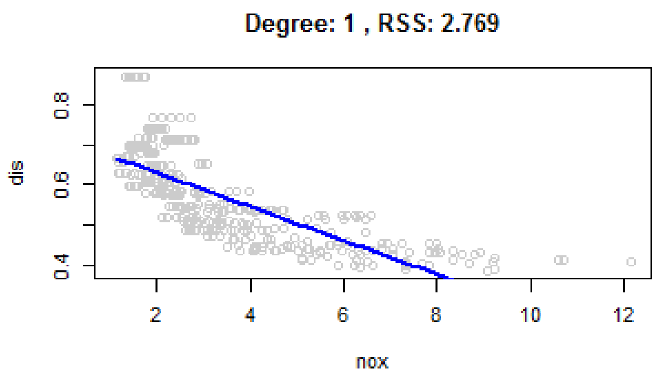
```
# get the range of the axis we want the line to follow
dislims =range(dis)
# create a grid of x-axis points we want to predict. In order to smooth the make the i
ncrements small.
dis.grid=seq(from=dislims[1], to=dislims [2], by = .1)
# predict the nox value for each of the points.
preds=predict(fit, newdata =list(dis=dis.grid), se=TRUE)
# plot our data points and the polynomial fit.
plot(dis,nox, main = "Polynomial fit of Boston data frame")
lines(dis.grid ,preds$fit ,lwd =2, col =" blue")
```

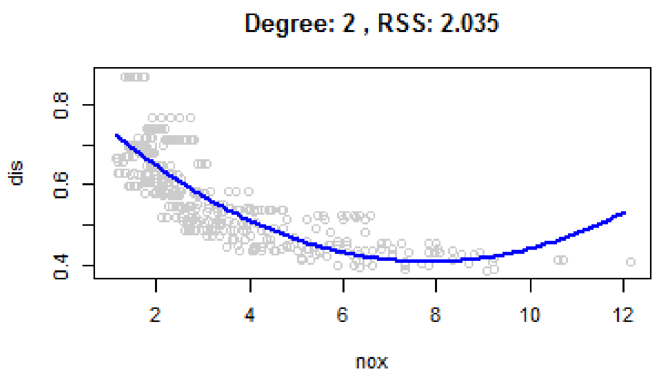


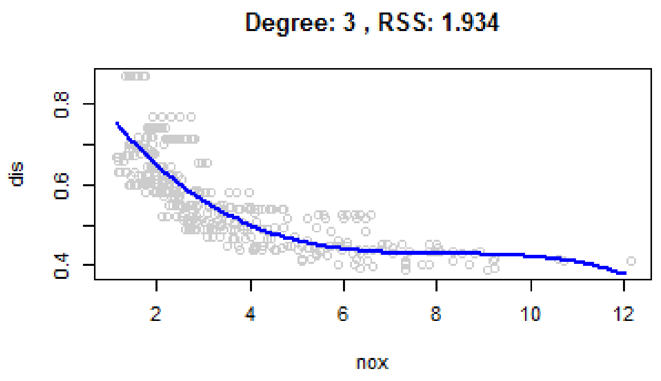
(b) Plot the polynomial fits for a range of different polynomial degrees (say, from 1 to 10), and report the associated residual sum of squares.

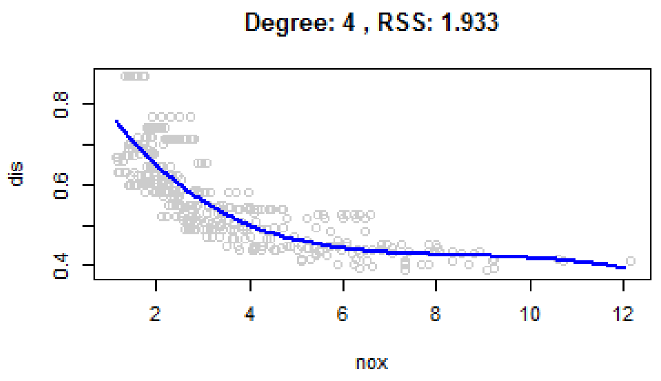
Below can be seen a plot for the polynomial fits, given the incremental degrees of freedom from 1 to 10. for each the residual some of squared, RSS, is reported in the title.

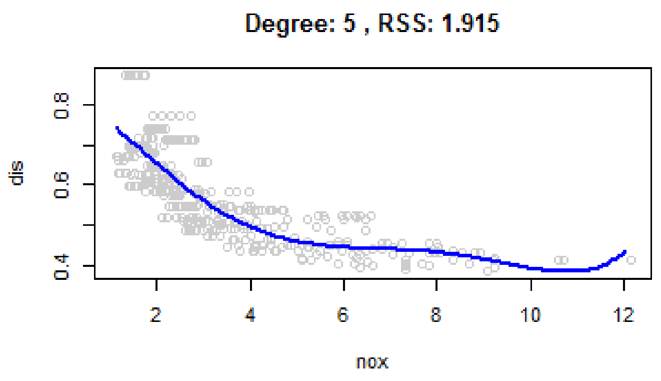
```r
par(mfrow = c(5, 2))
for(i in 1:10){
  fit = lm(nox ~ poly(dis ,i, raw =T))
  preds=predict(fit, newdata =list(dis=dis.grid), se=TRUE)
  plot(dis,nox, col="grey80",main= paste("Degree:", i, ", RSS:", round(sum(fit$residua
ls^2),3)), xlab="nox", ylab="dis")
  lines(dis.grid ,preds$fit ,lwd =2, col =" blue")
}
```

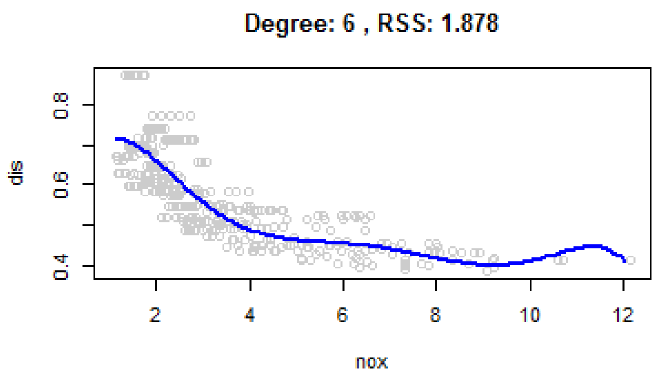

Degree: 1 , RSS: 2.769
dis
0.4
0.6
0.8
2
4
6
8
10
12
nox

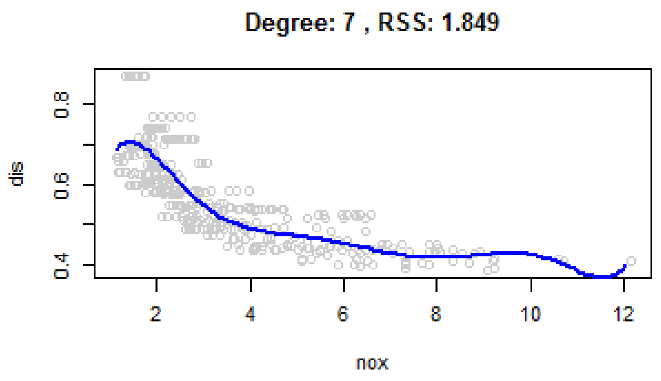

Degree: 2 , RSS: 2.035
dis
0.4
0.6
0.8
2
4
6
8
10
12
nox

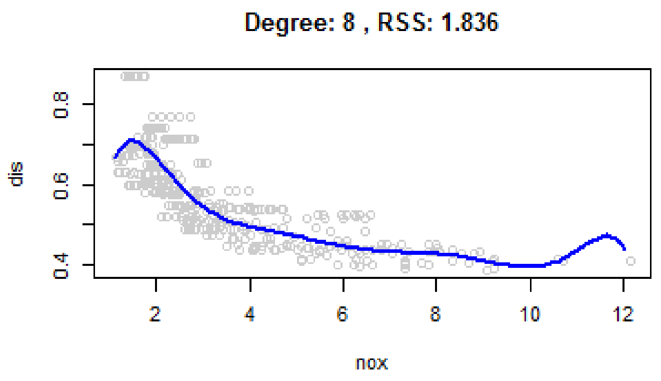

Degree: 3 , RSS: 1.934
dis
0.4
0.6
0.8
2
4
6
8
10
12
nox

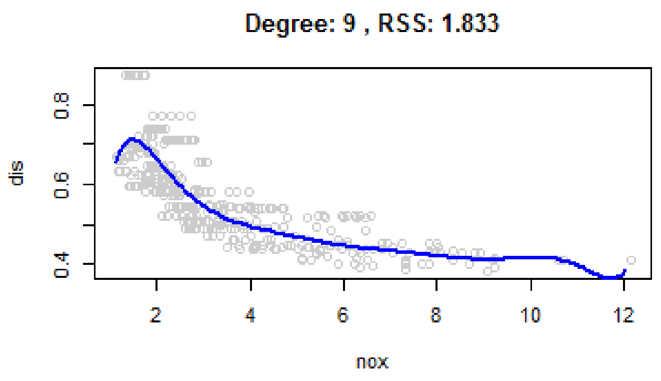

Degree: 4 , RSS: 1.933
dis
0.4
0.6
0.8
2
4
6
8
10
12
nox

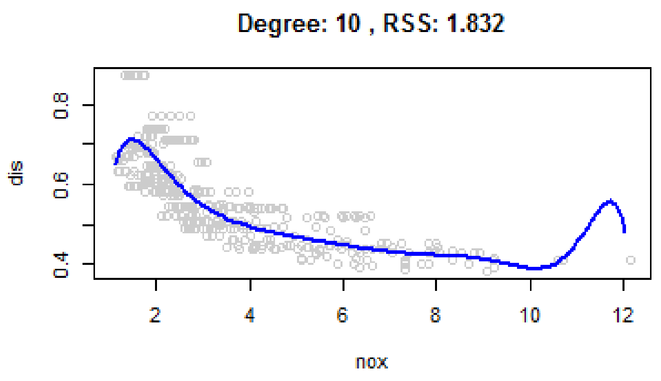

Degree: 5 , RSS: 1.915
dis
0.4
0.6
0.8
2
4
6
8
10
12
nox


Degree: 6 , RSS: 1.878
dis
0.4
0.6
0.8
2
4
6
8
10
12
nox


Degree: 7 , RSS: 1.849
dis
0.4
0.6
0.8
2
4
6
8
10
12
nox


Degree: 8 , RSS: 1.836
dis
0.4
0.6
0.8
2
4
6
8
10
12
nox


Degree: 9 , RSS: 1.833
dis
0.4
0.6
0.8
2
4
6
8
10
12
nox


Degree: 10 , RSS: 1.832
dis
0.4
0.6
0.8
2
4
6
8
10
12
nox

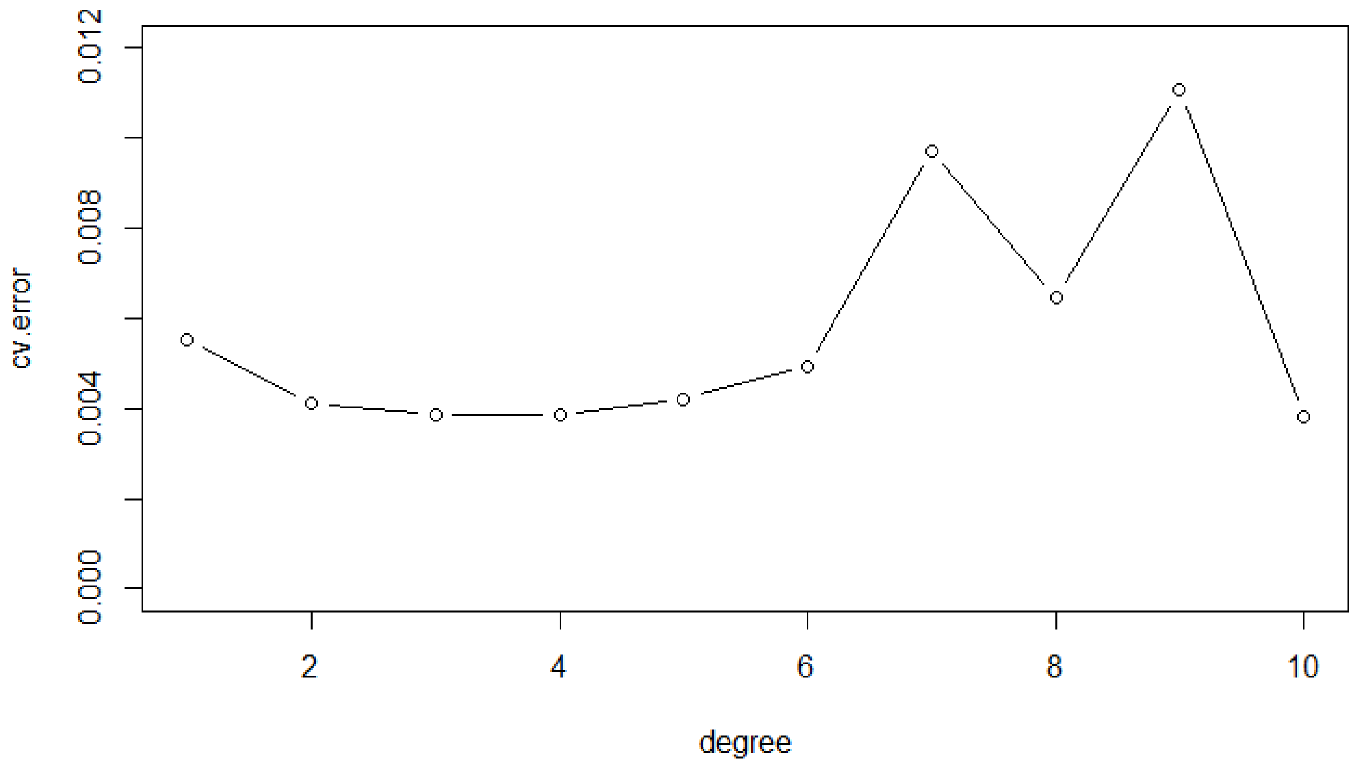
(c) Perform cross-validation or another approach to select the optimal degree for the polynomial, and explain your results.

First we perform 10 fold cross validation. for this part we use the glm package linear modelling in order to leverage the cross validation function.

```
par(mar=c(5,5,5,5))
par(mfrow = c(1, 1))
library(boot)
set.seed(800)
degree=1:10
cv.error=rep(0,10)
for (i in degree){
    fit = glm(nox ~ poly(dis ,i, raw =T),data=Boston)
    cv.error[i] = cv.glm(Boston,fit, K=10)$delta[1]
}
cv.error

##  [1] 0.005537975 0.004108083 0.003865727 0.003859278 0.004217672
##  [6] 0.004933940 0.009714958 0.006465204 0.011061762 0.003799474
```

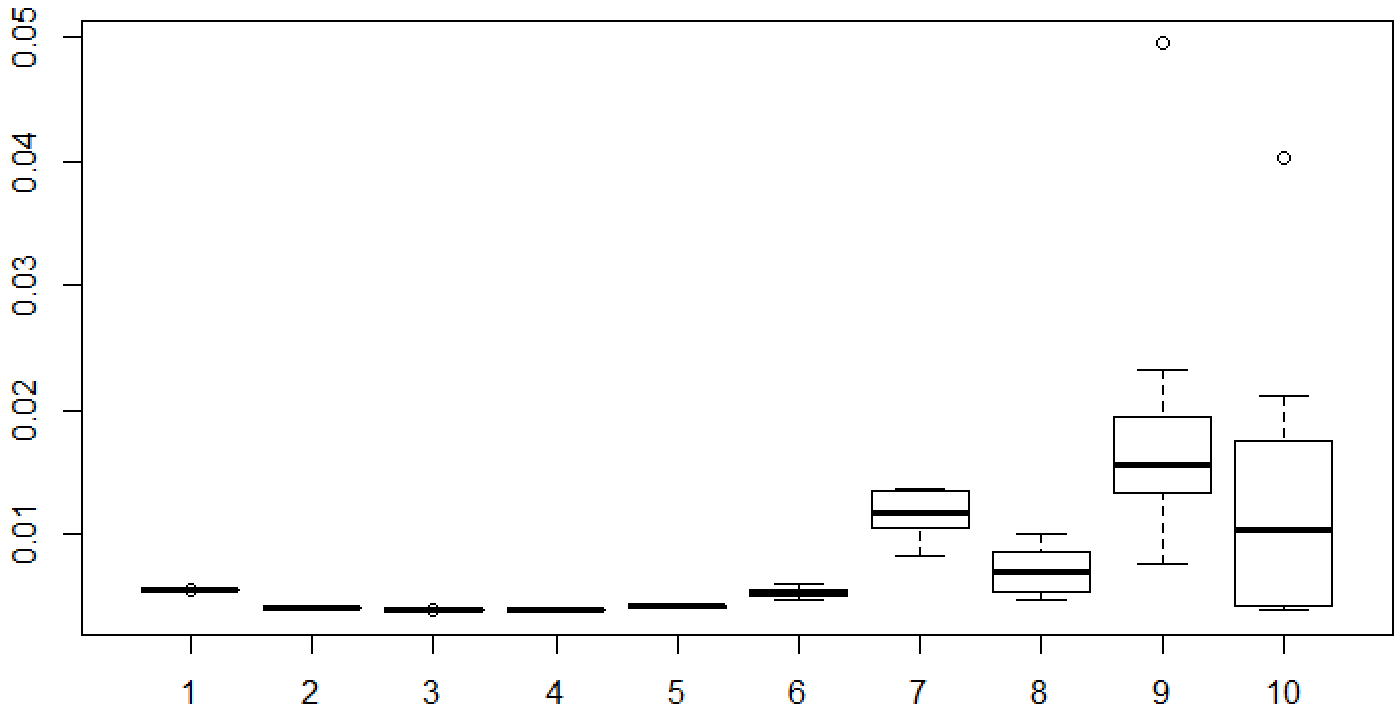
We can see that the minimum occurs at 10 degrees of freedom for this seed, with the 3rd and the 4th also performing nearly as well. However the chart seems to have a lot of fluctuation as the degrees increase beyond 6.

```
plot(degree,cv.error,type="b", ylim = c(0, 0.012))
```



Lets look at this fluctuation. We repeat the cross validation for different seeds and look at the variation in the cross validation. It can be seen that the models with larger degrees of freedom (>5) are very unstable in their results.

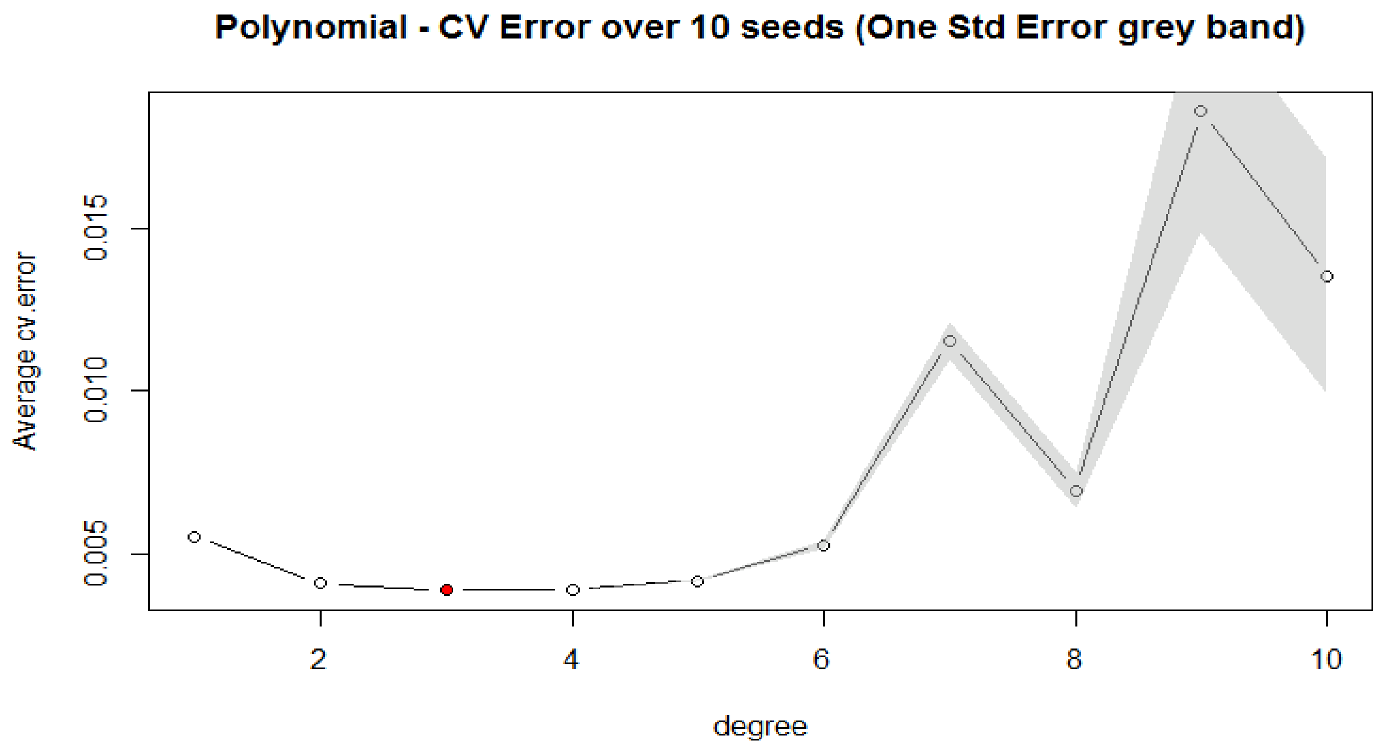
```
degree=1:10
cv.errormat = matrix(NA,nrow=10, ncol=10)
for(j in 1:10){
  set.seed(j)
  for (i in degree){
      fit = glm(nox ~ poly(dis ,i, raw =T),data=Boston)
      cv.errormat[j,i] = cv.glm(Boston,fit, K=10)$delta[1]
  }
}
boxplot(cv.errormat)
```



Below is a plot showing the average of results for 10 different seeds, along with the standard error in grey. We can see the 3rd degree marked in red which has the minimum error minimal standard error. The second degree is also quite low however is not low enough to meet the rule of one standard error from the minimum. **Therefore the optimal degree is the third.**

```
# plot the mean of the errors overs the 10 different seeds
plot(degree,colMeans(cv.errormat),type="b", ylab="Average cv.error", main="Polynomial
- CV Error over 10 seeds (One Std Error grey band)")

# Create a function to calculate the std error of the errors for each degree.
std <- function(x) sd(x)/sqrt(length(x))
# make the upper and lower threshold for SE and plot it
se.upper <- colMeans(cv.errormat) + apply(cv.errormat, 2, std)
se.lower <- colMeans(cv.errormat) - apply(cv.errormat, 2, std)
polygon(c(degree, rev(degree)), c(se.lower, rev(se.upper)),col = adjustcolor("grey",al
pha.f=0.5), border = NA)
```

```
# plot the minimum error point
degree.min <- which.min(colMeans(cv.errormat))
points(degree.min, colMeans(cv.errormat)[degree.min], pch = 20, col = "red")
```



(d) Use the bs() function to fit a regression spline to predict nox using dis. Report the output for the fit using four degrees of freedom. How did you choose the knots? Plot the resulting fit.

There are two ways this question can be interpreted. (1) The bs() package degrees of freedom = 4, which fits one knot. See documentation linked here. Or (2) the text book definition of degrees of freedom, which holds the bs() produces a cubic spline with an intercept, so df should be set at 3 in function bs(). We show both approaches below. I will assume the second choice is the correct one, but show both to be sure the question is covered

Setting df=4 – Aligned with bs() documentation

4 degrees of freedom according to function bs() in a cubic regression spline - the default setting of - indicates we should use 1 knot in the spline. In order to choose the knot, a common practice is to place knots in a uniform fashion. For this case we will use option df to produce a spline with knots at uniform intervals – for a cubic spline with one knot, the knot would be at the median of all the dis values. We can verify this below.

```
# Lets verify that one knot is placed by function bs() at the median of dis.
library(splines)

# Use attr() to find where the knot is placed with function bs().
median(dis) == attr(bs(dis,df=4), "knots")

##  50%
## TRUE
```

Now lets fit the model. As can be seen below the spline has 4 degrees of freedom.

```
# Load the package and fit cubic spline models with one knot.
fit1 = lm(nox ~ bs(dis,df=4),data=Boston)
```

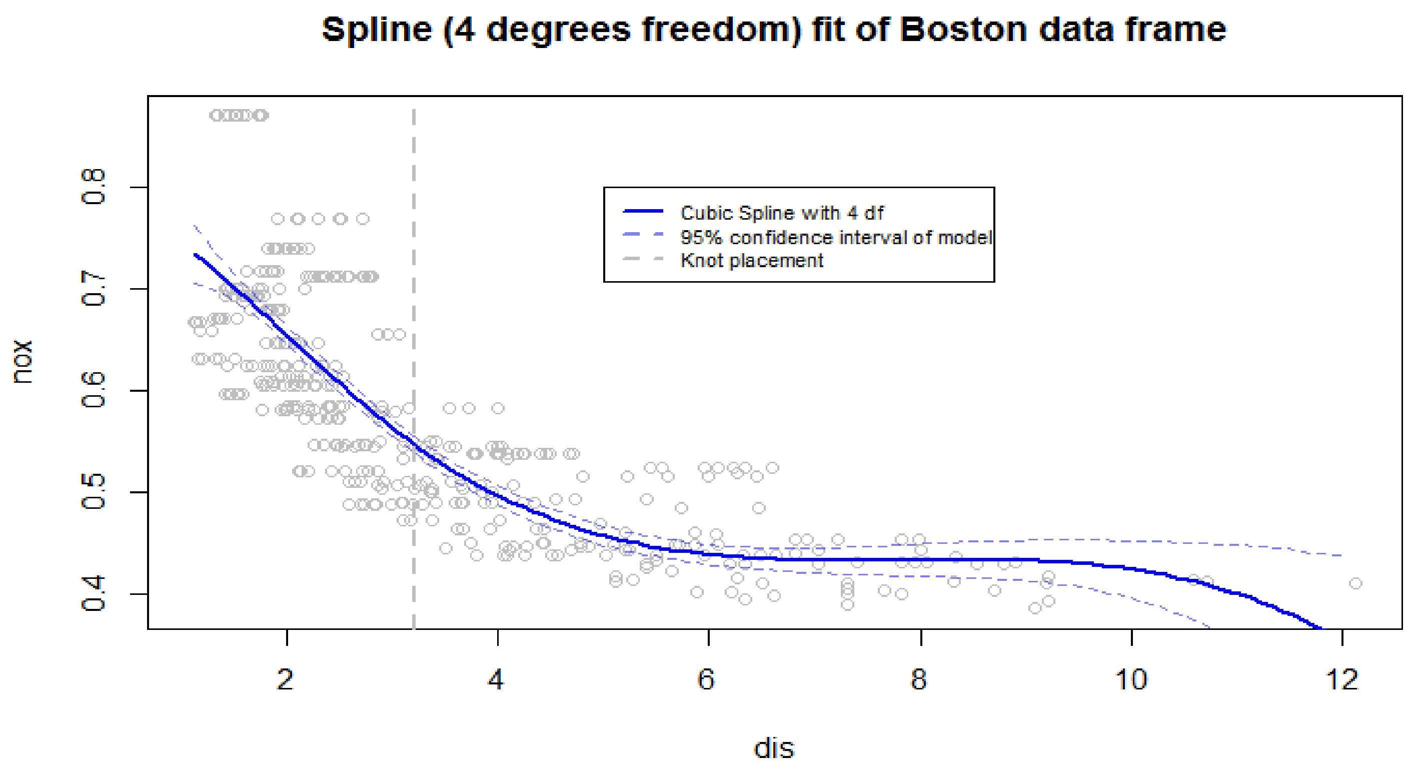
```
# Validate the degrees of freedom.
summary(fit1)$fstatistic[2]

## numdf
##     4
```

The full summary of the fit can be seen below.

```
summary(fit1)    # fit1 is the manually picked

##
## Call:
## lm(formula = nox ~ bs(dis, df = 4), data = Boston)
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -0.124622 -0.039259 -0.008514  0.020850  0.193891
##
## Coefficients:
##                  Estimate Std. Error t value Pr(>|t|)
## (Intercept)       0.73447    0.01460  50.306  < 2e-16 ***
## bs(dis, df = 4)1 -0.05810    0.02186  -2.658  0.00812 **
## bs(dis, df = 4)2 -0.46356    0.02366 -19.596  < 2e-16 ***
## bs(dis, df = 4)3 -0.19979    0.04311  -4.634 4.58e-06 ***
## bs(dis, df = 4)4 -0.38881    0.04551  -8.544  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.06195 on 501 degrees of freedom
## Multiple R-squared:  0.7164, Adjusted R-squared:  0.7142
## F-statistic: 316.5 on 4 and 501 DF,  p-value: < 2.2e-16
```

Now lets plot the fits along with the associated knot.

```
preds=predict(fit1, newdata =list(dis=dis.grid), se=TRUE)
# plot our data points and the polynomial fit.
plot(dis,nox, main = "Spline (4 degrees freedom) fit of Boston data frame", col="grey"
)
lines(dis.grid, preds$fit ,lwd =2, col =" blue")
# show the 95% confidence interval of the model
lines(dis.grid,preds$fit +1.96 * preds$se ,lty = "dashed", col= adjustcolor("blue",alp
ha.f=0.5), lwd =1.5)
lines(dis.grid ,preds$fit -1.96 * preds$se ,lty = "dashed", col= adjustcolor("blue",al
pha.f=0.5), lwd =1.5)
abline(v=attr(bs(dis,df=4), "knots"), lty=2, lwd=2, col="grey")
legend(5,.8, c("Cubic Spline with 4 df", "95% confidence interval of model", "Knot pla
cement"), col=c("blue",adjustcolor("blue",alpha.f=0.5), "grey"), lwd=2, lty = c("solid
","dashed", "dashed"), cex=.7)
```



The fit has a long tail and large confidence band at high values of dis, this is to be expected given the small number of points in this range. Besides this it is quite smooth with a slight broadening of the confidence band at the lower values of dis.

Setting df=3 – Aligned with textbook documentation

Alternatively we can go with the text book definition where for the cubic spline with K knots, we use K+4 degrees of freedom. So this would mean the df=3 in function bs(), giving no knots.

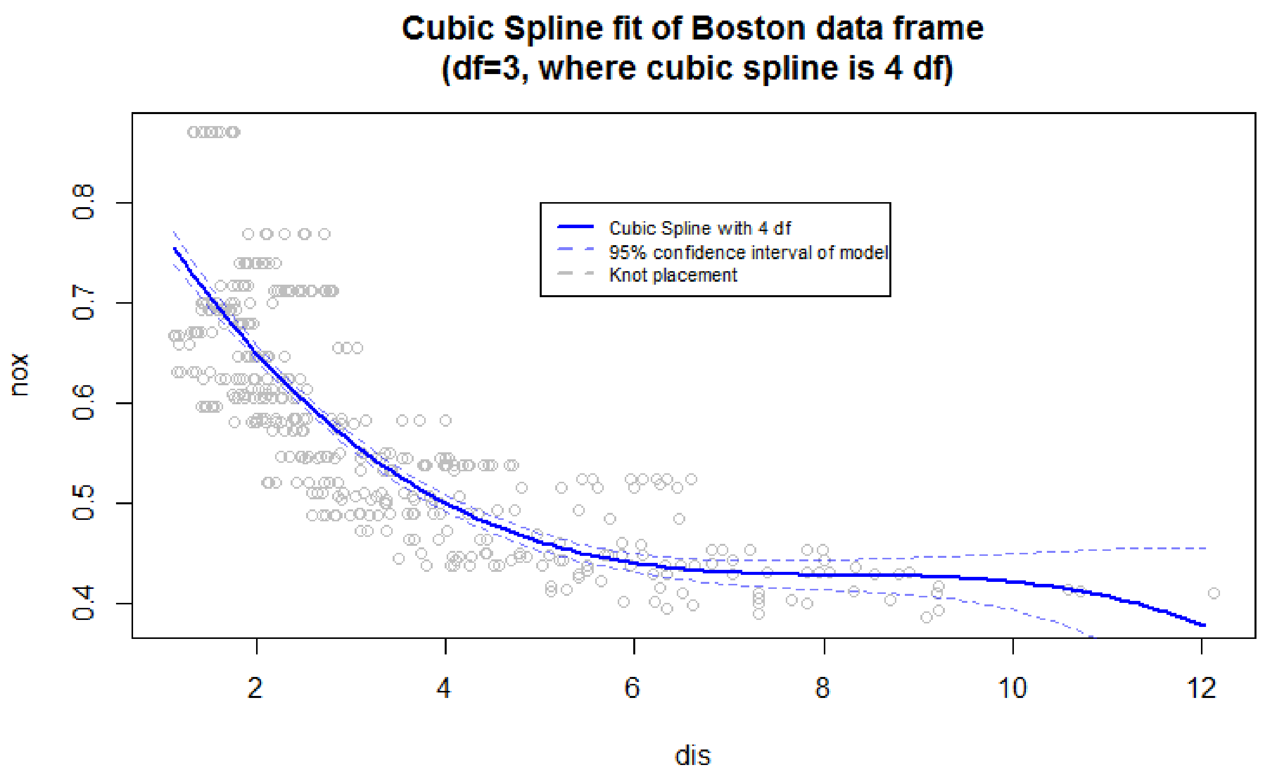
```
# Lets verify that one knot is placed by function bs() at the median of dis.
library(splines)
median(dis) == attr(bs(dis,df=3), "knots")

## logical(0)

# Load the package and fit cubic spline models with one knot.
fit1a = lm(nox ~ bs(dis,df=3),data=Boston)
```

The full summary of this fit can be seen below.

```
summary(fit1a)    # fit1 is the manually picked

##
## Call:
## lm(formula = nox ~ bs(dis, df = 3), data = Boston)
##
## 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)       0.755153   0.008283  91.168  < 2e-16 ***
## bs(dis, df = 3)1 -0.498271   0.032542 -15.312  < 2e-16 ***
## bs(dis, df = 3)2 -0.233520   0.036994  -6.312 6.05e-10 ***
## bs(dis, df = 3)3 -0.382680   0.045455  -8.419 4.00e-16 ***
## ---
## 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
```

Now lets plot the fits along with the associated knot.

```
predsa=predict(fit1a, newdata =list(dis=dis.grid), se=TRUE)
# plot our data points and the polynomial fit.
plot(dis,nox, main = " Cubic Spline fit of Boston data frame \n (df=3, where cubic spl
ine is 4 df)", col="grey")
lines(dis.grid, predsa$fit ,lwd =2, col =" blue")
# show the 95% confidence interval of the model
lines(dis.grid,predsa$fit +1.96 * predsa$se ,lty = "dashed", col= adjustcolor("blue",a
lpha.f=0.5), lwd =1.5)
lines(dis.grid ,predsa$fit -1.96 * predsa$se ,lty = "dashed", col= adjustcolor("blue",
alpha.f=0.5), lwd =1.5)
abline(v=attr(bs(dis,df=3), "knots"), lty=2, lwd=2, col="grey")
legend(5,.8, c("Cubic Spline with 4 df", "95% confidence interval of model", "Knot pla
cement"), col=c("blue",adjustcolor("blue",alpha.f=0.5), "grey"), lwd=2, lty = c("solid
","dashed", "dashed"), cex=.7)
```



This fit also has a long tail and large confidence band at high values of dis, this is to be expected given the small number of points in this range. Besides this it is quite smooth, however there is minimal broadening of the confidence band at the lower values of dis.

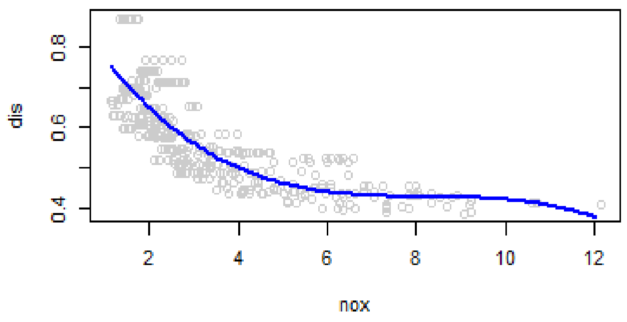
(e) Now fit a regression spline for a range of degrees of freedom, and plot the resulting fits and report the resulting RSS. Describe the results obtained.

Here we mark degrees of freedom according to the bs() documentation - https://stat.ethz.ch/R-manual/R-devel/library/splines/html/bs.html.

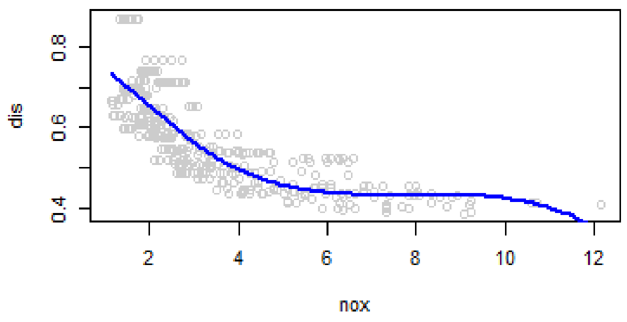
Below can be seen the plot of regression splines ranging from 3 up to 11 degrees of freedom, along with the reported RSS. We can see as the knots increase the function gets increasing complex (wiggly) to fit the training data. Up to 4 knots (7 degrees of freedom) seems reasonably smooth to the naked eye. The whiplash effect at low values of dis starts to appear at only 2 knots.

```
degrees = 3:12
RSS <- rep(0,12)

par(mfrow = c(5, 2))
for(i in degrees){
  fit = glm(nox ~ bs(dis,df=i),data=Boston)
  preds=predict(fit, newdata =list(dis=dis.grid), se=F)
  RSS[i] <- round(sum(fit$residuals^2),3)
  plot(dis, nox, col="grey80",main= paste("Degree: ", i, ", RSS: ", RSS[i], ", Knots: 
", i-3, sep=""), xlab="nox", ylab="dis")
  lines(dis.grid ,preds ,lwd =2, col =" blue")
}
```

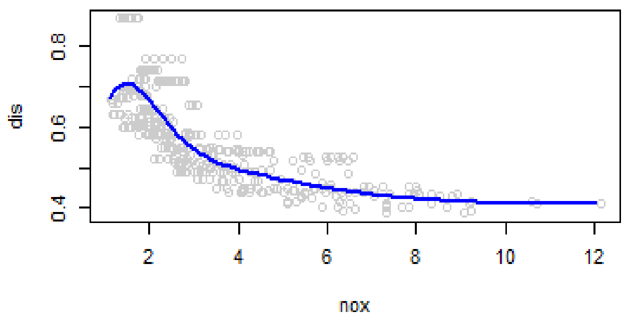
Degree: 3, RSS: 1.934, Knots: 0



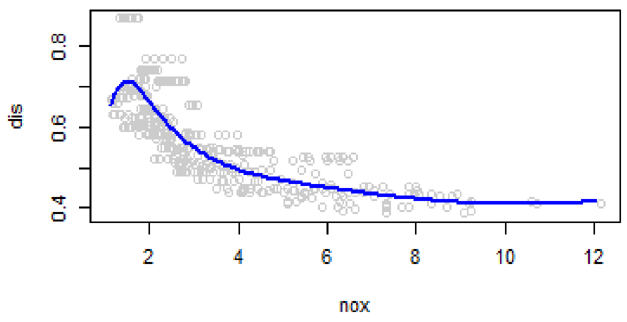
Degree: 4, RSS: 1.923, Knots: 1



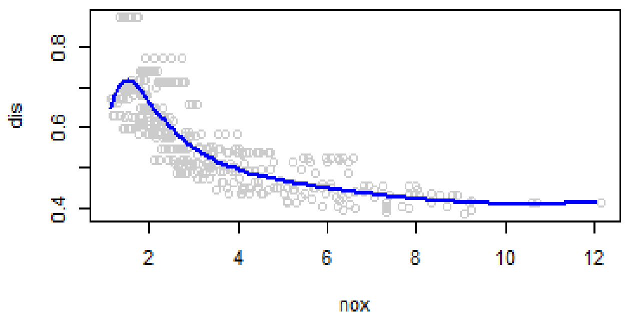
Degree: 5, RSS: 1.84, Knots: 2



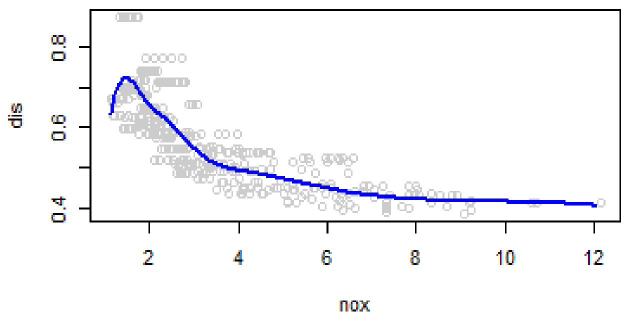
Degree: 6, RSS: 1.834, Knots: 3



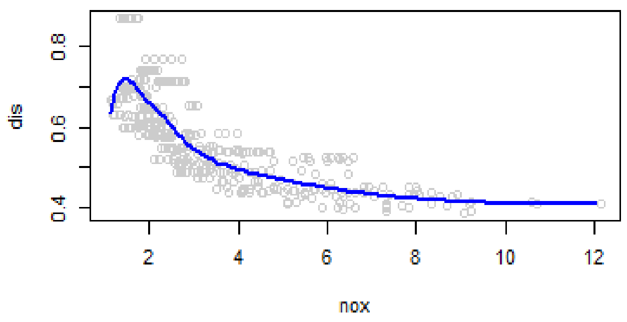
Degree: 7, RSS: 1.83, Knots: 4



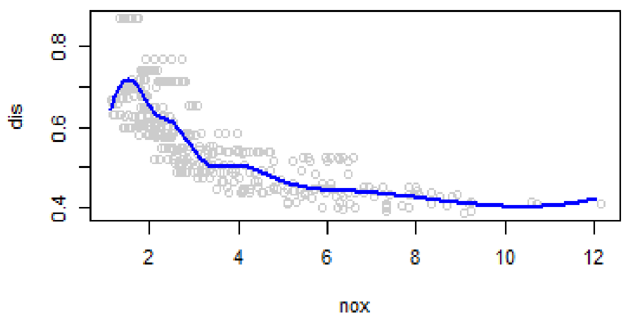
Degree: 8, RSS: 1.817, Knots: 5



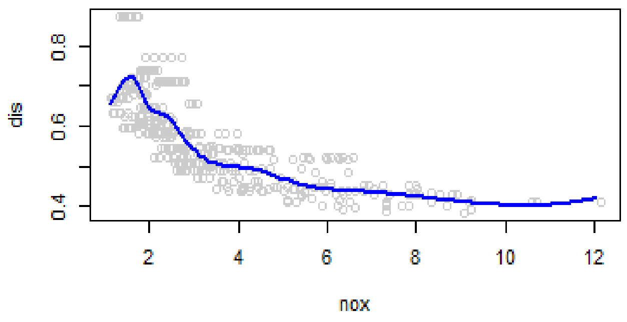
Degree: 9, RSS: 1.826, Knots: 6



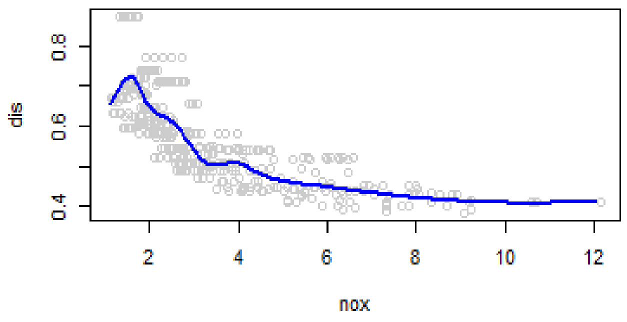
Degree: 10, RSS: 1.793, Knots: 7



Degree: 11, RSS: 1.797, Knots: 8

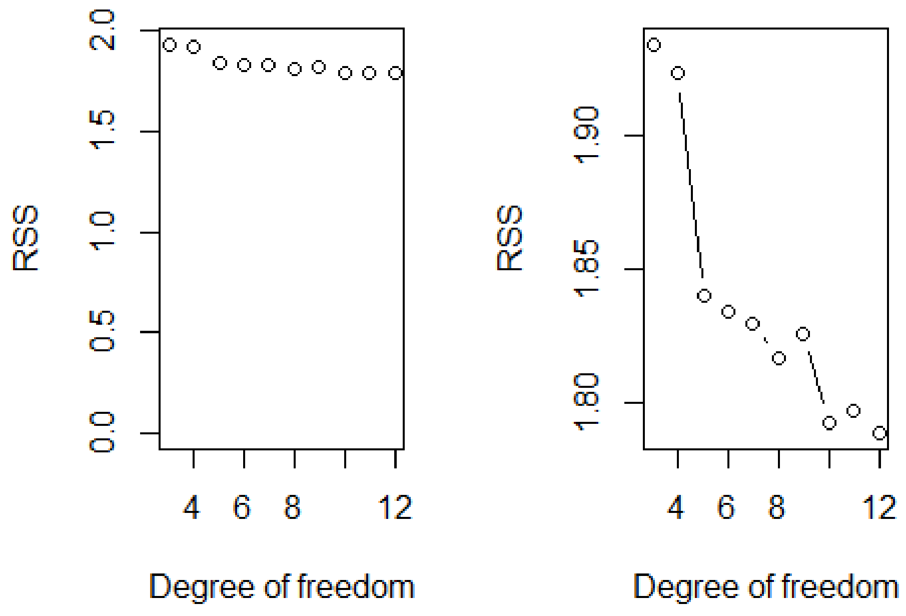


Degree: 12, RSS: 1.789, Knots: 9



As can be seen on the 0 scale chart on the left below, there is an initial drop in RSS from 4 to 5 df, however the drop is slower after that point, as the function gradually fits the data better. The right chart zoomed in on the y-axis shows a nearly steady decline in RSS from 5 to 12 df.

```
par(mfrow = c(1, 2))
plot(degrees,RSS[3:12],type="b", ylim=c(0,max(RSS)), xlab="Degree of freedom", ylab="R
SS")
plot(degrees,RSS[3:12],type="b", xlab="Degree of freedom", ylab="RSS")
```



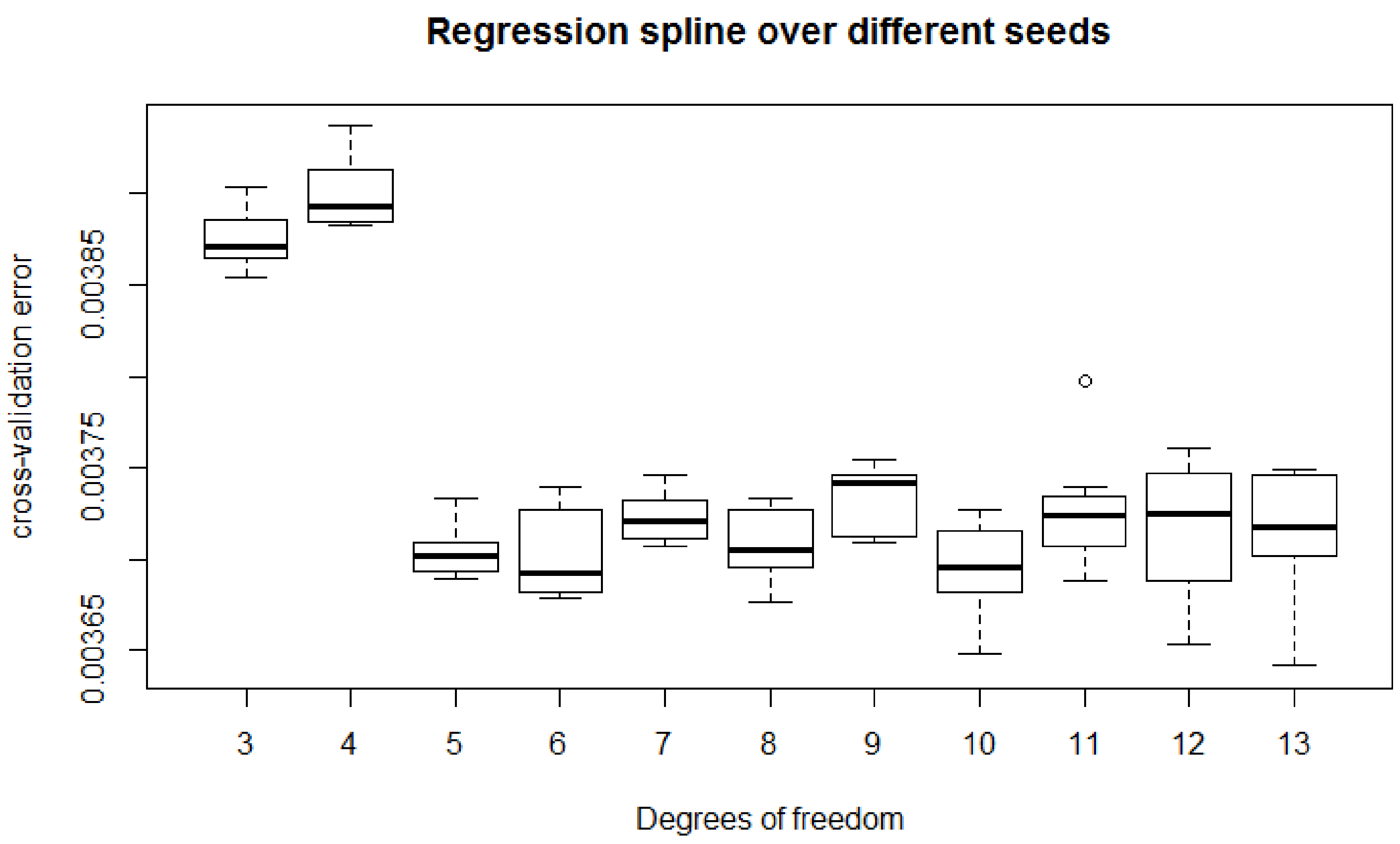
(f) Perform cross-validation or another approach in order to select the best degrees of freedom for a regression spline. Describe your results.

Here we interpret degrees of freedom according to the bs() documentation - https://stat.ethz.ch/R-manual/R-devel/library/splines/html/bs.html.

Here we run a number of cross validations for each degree and box plot the results. Of interest is that the median cv.error results from df=5 to df=13 are quite similar. However the df=5 appears to have the least variance.

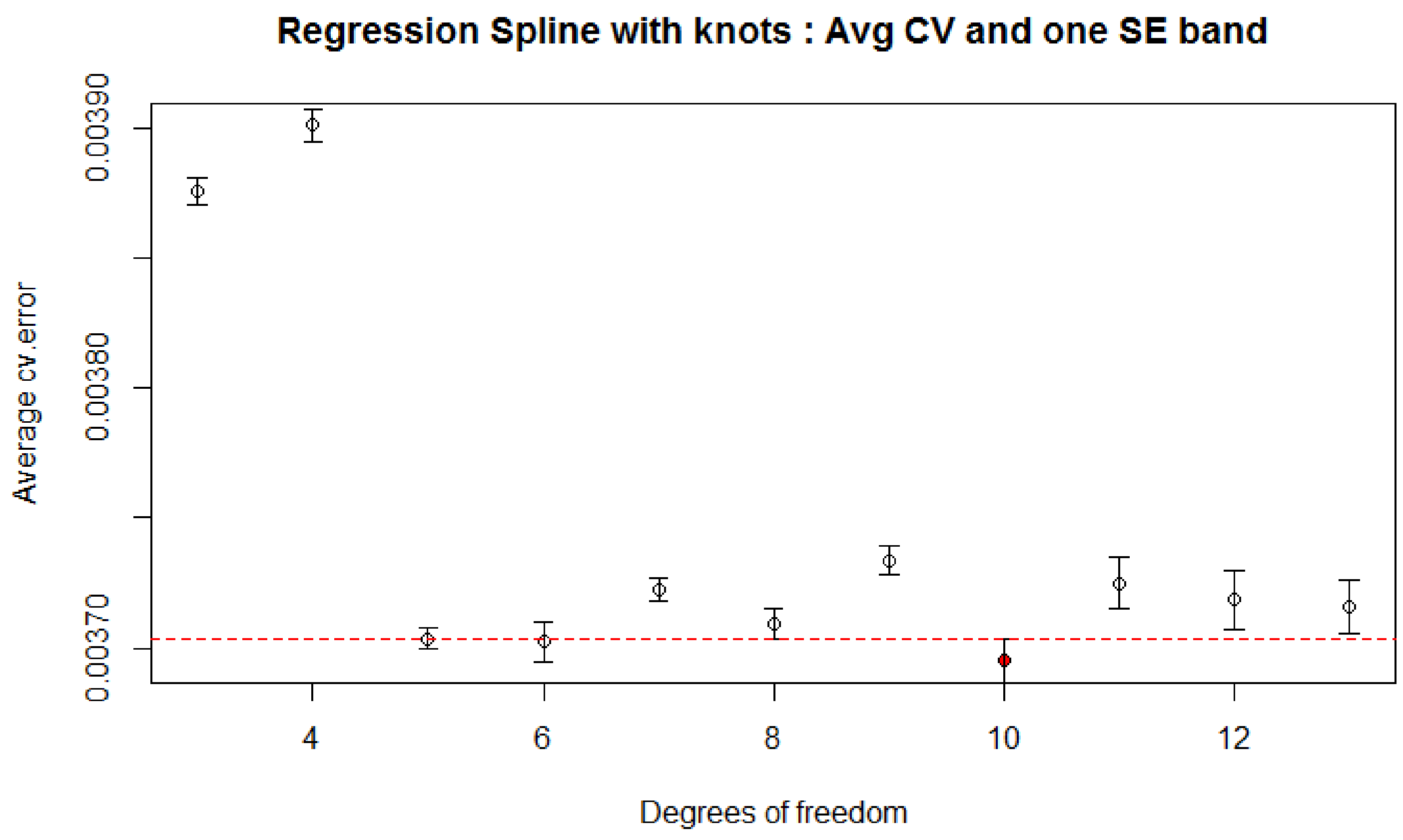
```
cv.errormat = data.frame(matrix(NA,nrow=10, ncol=10))
degree.k <- 3:13
for(j in 1:10){
  set.seed(j)
  for (i in degree.k){
      fit2 = glm(nox ~ bs(dis,df=i),data=Boston)
      cv.errormat[j,i-2] = cv.glm(Boston,fit2, K=10)$delta[1]
  }
}

colnames(cv.errormat) <- degree.k
```

```r
boxplot(cv.errormat, xlab="Degrees of freedom", ylab="cross-validation error", main="R
egression spline over different seeds")
```



Finally we will plot the results along with there standard error, and display the standard error to attempt to use the one standard error rule.

```r
## plot the mean of the errors overs the 10 different seeds
plot(degree.k,colMeans(cv.errormat),xlab="Degrees of freedom", ylab="Average cv.error"
, main="Regression Spline with knots : Avg CV and one SE band")
## make the upper and lower threshold for SE and plot it
se.upper <- colMeans(cv.errormat) + apply(cv.errormat, 2, std)
se.lower <- colMeans(cv.errormat) - apply(cv.errormat, 2, std)
## plot the minimum error point
degree.min <- which.min(colMeans(cv.errormat))
points(degree.k[degree.min], colMeans(cv.errormat)[degree.min], pch = 20, col = "red")
arrows(degree.k, se.lower, degree.k, se.upper,length=0.05, angle=90, code=3)
abline(h=se.upper[degree.min] ,col="red",lty=2)
```



It appears like some of the results at lower degrees of freedom meets the standard error rule of being within one standard error of the minimum. We validate this below. 6 is the lowest df to meet the one SE rule, therefore **we choose 6 as the best degrees of freedom for a regression spline with this data.**

```
# Check which results are within the one SE of the minimum.
colMeans(cv.errormat) < se.upper[degree.min]

##     3     4     5     6     7     8     9    10    11    12    13
## FALSE FALSE FALSE  TRUE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE
```

4) For this problem I worked with Tony Wu, Sevvandi Kandanaarachchi and Andrew Beckerman. This problem works with the body dataset used in the in class session from Feb 11. The goal of this problem is to perform and compare Principal Components Regression and Partial Least Squares on the problem of trying to predict someones weight. While you can use any R tools at your disposal to complete the problem, library(pls) and Lab 3 from ISLR will probably be very helpful and the problem set was written with these approaches in mind. If you have not already downloaded the data, please go to the class coursework page and do so. More information about this dataset can be found at http://www.amstat.org/publications/jse/v11n2/datasets.heinz.html.

(a) Read the body dataset into R using the load() function. This dataset contains:

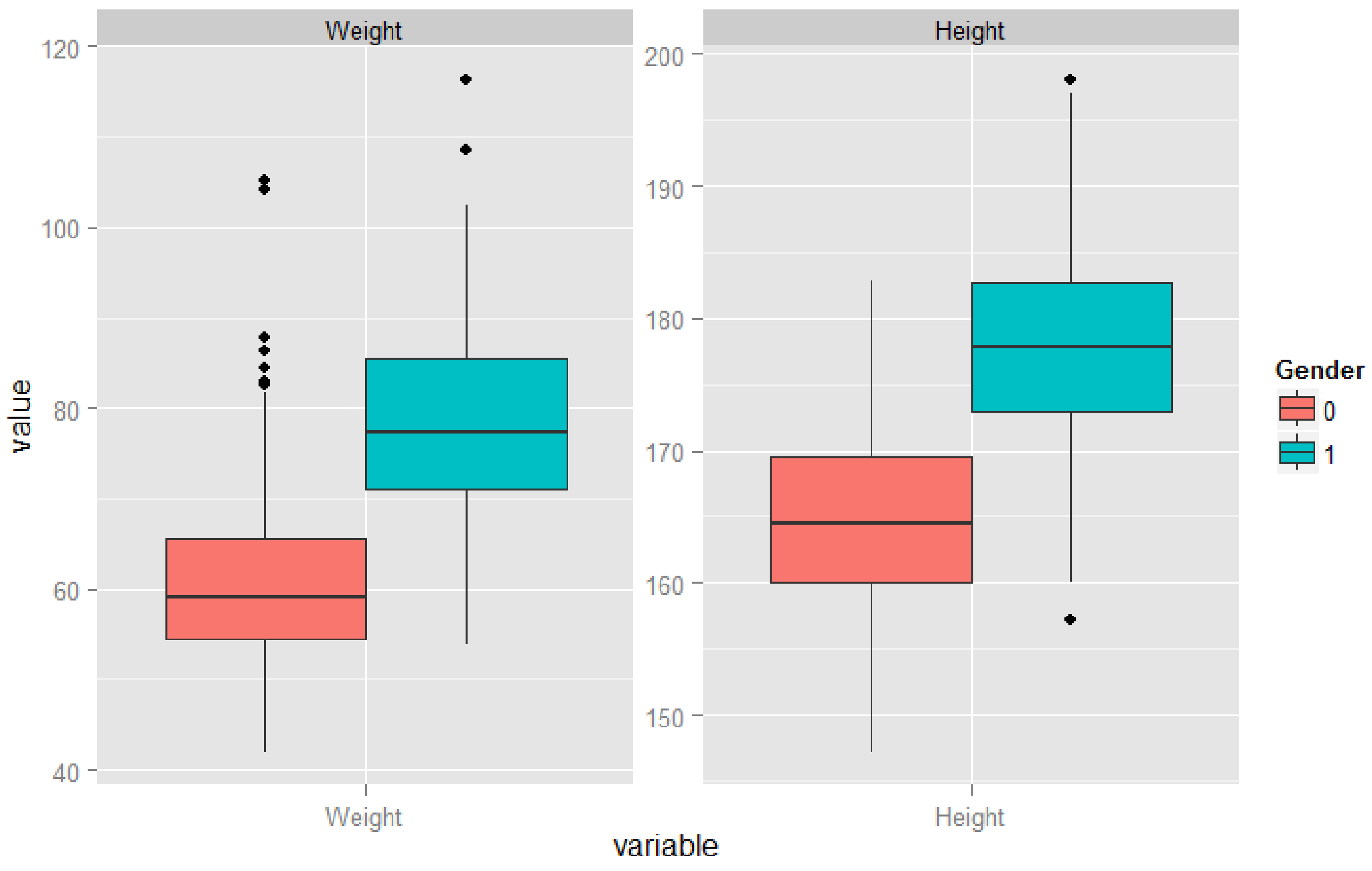
X: A dataframe containing 21 different types of measurements on the human body.

Y: A dataframe that contains the age, weight (kg), height (cm), and the gender of each person in the sample.

Let's say we forgot how the gender is coded in this dataset. Using a simple visualization, explain how you can tell which gender is which.

First we set our working directory and load up the dataset. Below it can be seen clearly that female is 0, and male is 1, from the height and weight difference between the two codes.

```
# Place the data set in this directory
setwd("C:/Users/Think/Google Drive/stats216/Feb12th")
load("body.RData")

# Use package reshape2() to reorganize the data to allow for a ggplot boxplot
library(reshape2)
library(ggplot2)
dat.m <- melt(Y[,-1], id=c("Gender"))

# For ggplot() boxplot convert Gender variable to factor
dat.m$Gender <- as.factor(dat.m$Gender)
ggplot(data = dat.m, aes(x=variable, y=value)) + geom_boxplot(aes(fill=Gender)) +facet
_wrap( ~ variable, scales="free")
```



(b) Reserve 200 observations from your dataset to act as a test set and use the remaining 307 as a training set. On the training set, use both pcr and plsr to fit models to predict a person's weight based on the variables in X. Use the options scale = TRUE and validation=`CV'. Why does it make sense to scale our variables in this case?

Below we load our data and the packages, and create a data frame containing predictors and response. Note when performing PCR, we scale our X variables. We do this because some variables have smaller scaled values, however they may be good indicators of weight, and we would like to treat them equally. For example, people with lower bone mass, may have narrower wrist size, so wrist size may be a key indicator even though the measurement are small compared to other variable like chest.Diam.

```
# Set our seed, load our library and sample the 200 test data point, the rest we put i
n train.
set.seed(1000)
library(pls)
# create the index for our test and train set
test = sort(sample(1:nrow(X), 200))
train = (1:nrow(X))[-test]
# Create a data frame with the predictors and the weight, which is the response.
mydf <- data.frame(Y$Weight, X)
# Fit the Principal Component Regreession
pcr.fit  <- pcr(Y.Weight ~ .,data=mydf[train,], scale=TRUE, validaton="CV")
# Fit the Partial Least Squares Regreesion
plsr.fit <- plsr(Y.Weight ~ .,data=mydf[train,], scale=TRUE, validaton="CV")
```

(c) Run summary() on each of the objects calculated above, and compare the training % variance explained from the pcr output to the plsr output. Do you notice any consistent patterns (in comparing the two)? Is that pattern surprising? Explain why or why not.

From looking at the two summaries below we see they are broadly similar. However it can be noted that PCR generally explains more of the variance in X, the predictors, while PLSR explains more variance in Weight, the response.

We know from the book, that PCR explains the predictors, however there is no guarantee that the directions that best explain the predictors will also be the best directions to use for predicting the response. PLS on the other hand identifies new features in a supervised way - that is, it makes use of the response Y in order to identify new features that not only approximate the old features well, but also that are related to the response.

Therefore the behavior below is to be expected, where PCR explains predictors better, while PLSR explain the response better.

```
summary(pcr.fit)

## Data:    X dimension: 307 21
##  Y dimension: 307 1
## Fit method: svdpc
## Number of components considered: 21
## TRAINING: % variance explained
##           1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps
## X           63.05    75.30    80.14    84.23    86.51    88.61    90.21
## Y.Weight    93.09    93.85    94.74    94.79    95.01    95.05    95.12
##           8 comps  9 comps  10 comps  11 comps  12 comps  13 comps
## X            91.63    92.89     94.06     95.04     95.93     96.72
## Y.Weight     95.15    95.15     95.16     95.19     95.19     95.26
##           14 comps  15 comps  16 comps  17 comps  18 comps  19 comps
## X            97.44     98.06     98.54     98.95     99.35     99.60
## Y.Weight     95.31     95.68     95.74     95.75     96.01     96.01
##           20 comps  21 comps
## X            99.82    100.00
## Y.Weight     96.01     96.02
```

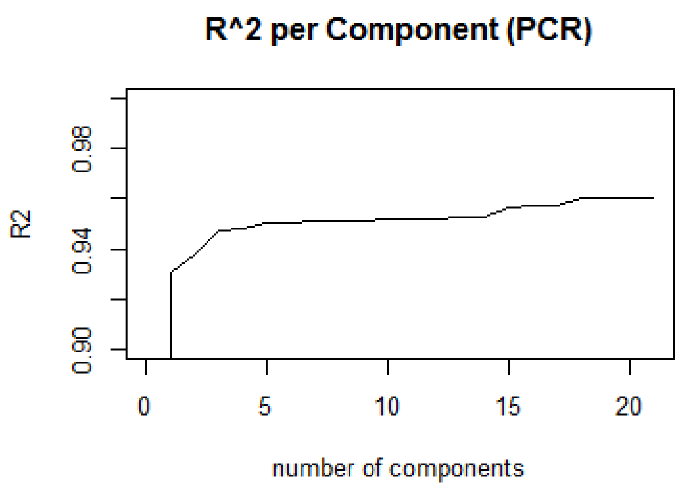
```r
summary(plsr.fit)
```

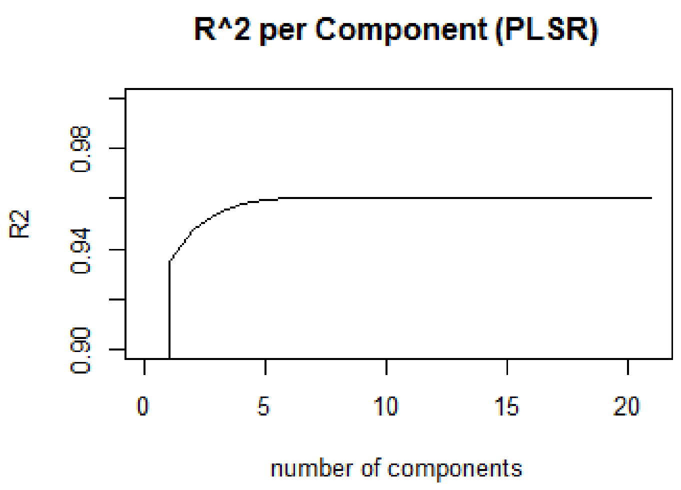
```
## Data:    X dimension: 307 21
##  Y dimension: 307 1
## Fit method: kernelpls
## Number of components considered: 21
## TRAINING: % variance explained
##           1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps
## X           63.03    73.66    79.60    81.63    83.37    86.66    88.24
## Y.Weight    93.53    94.83    95.39    95.80    95.97    96.00    96.01
##           8 comps  9 comps  10 comps  11 comps  12 comps  13 comps
## X           89.27    90.69     91.97     92.81     93.72     94.59
## Y.Weight    96.02    96.02     96.02     96.02     96.02     96.02
##           14 comps  15 comps  16 comps  17 comps  18 comps  19 comps
## X            95.33     96.03     96.60     97.65     98.32     98.70
## Y.Weight     96.02     96.02     96.02     96.02     96.02     96.02
##           20 comps  21 comps
## X            99.41    100.00
## Y.Weight     96.02     96.02
```

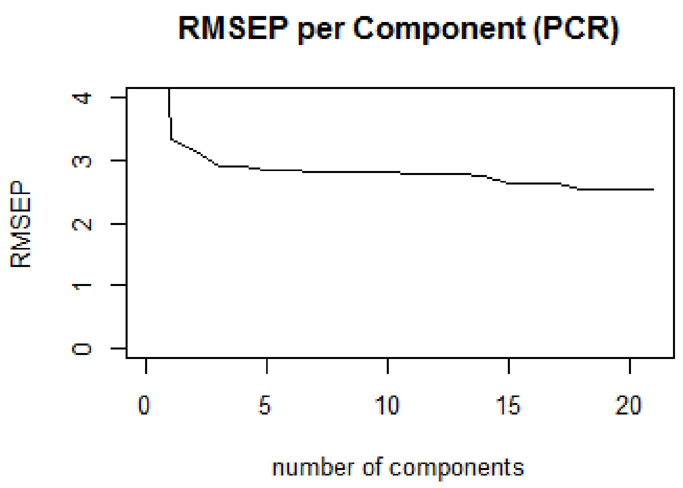
(d) For each of models, pick a number of components that you would use to predict future values of weight from X. Please include any further analysis you use to decide on the number of components.

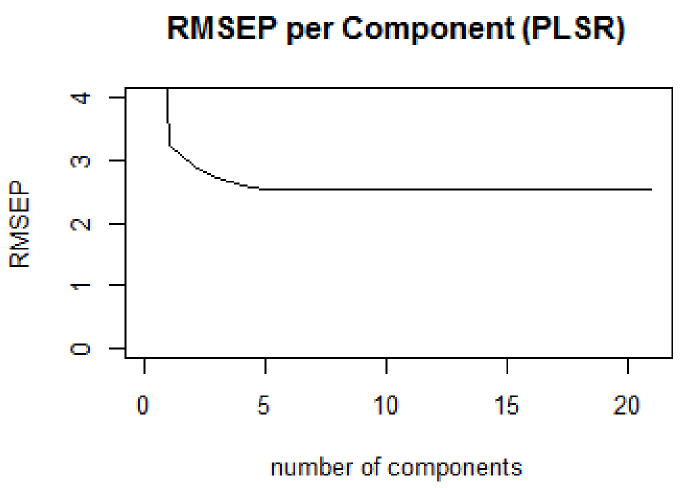
First let's take a look at the R^2 value and root mean squared error of prediction for each model. This can be seen below, in order to compare we zoomed into the area of interest on the y-axis. It is apparent that PLSR reaches close to the max in predicting weight, much sooner than PCR.

```r
par(mfrow=c(2,2))
plot(R2(pcr.fit), ylim=c(.9,1), main="R^2 per Component (PCR)")
plot(R2(plsr.fit), ylim=c(.9,1), main="R^2 per Component (PLSR)")
plot(RMSEP(pcr.fit), ylim=c(0,4), main="RMSEP per Component (PCR)")
plot(RMSEP(plsr.fit), ylim=c(0,4), main="RMSEP per Component (PLSR)")
```



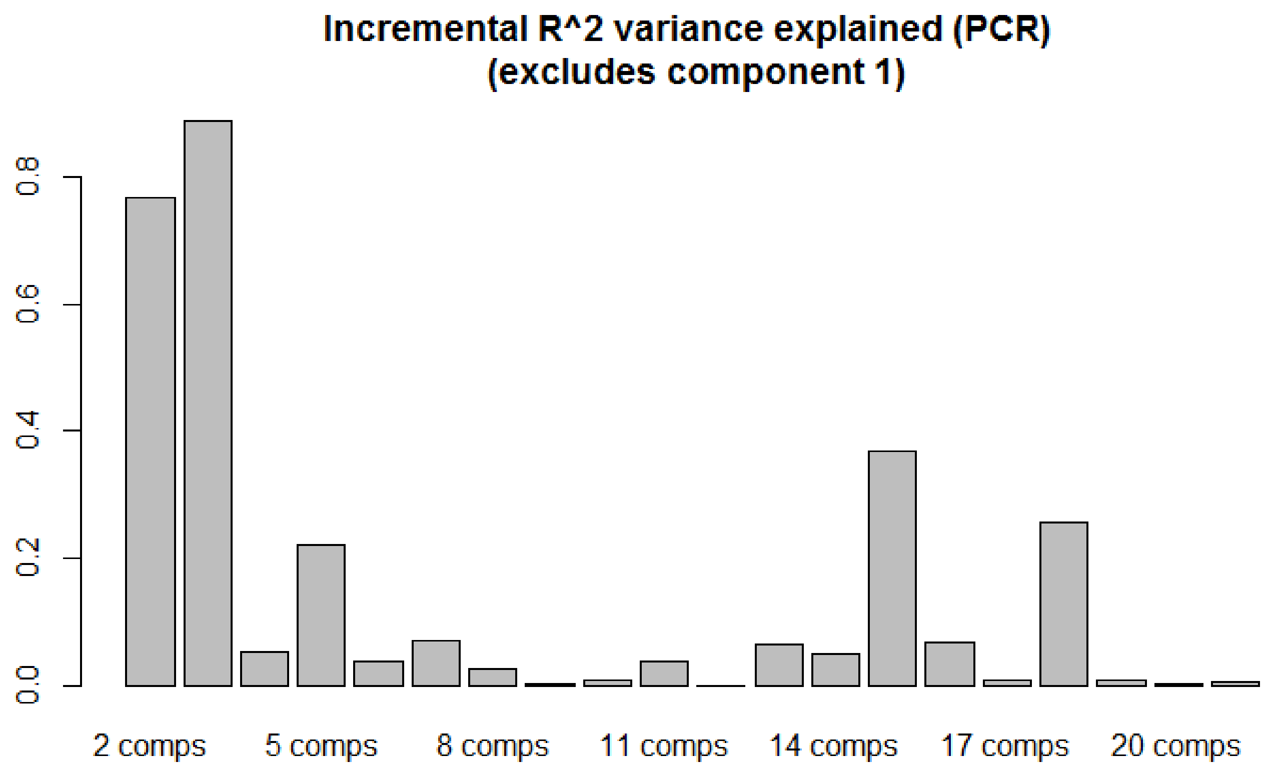




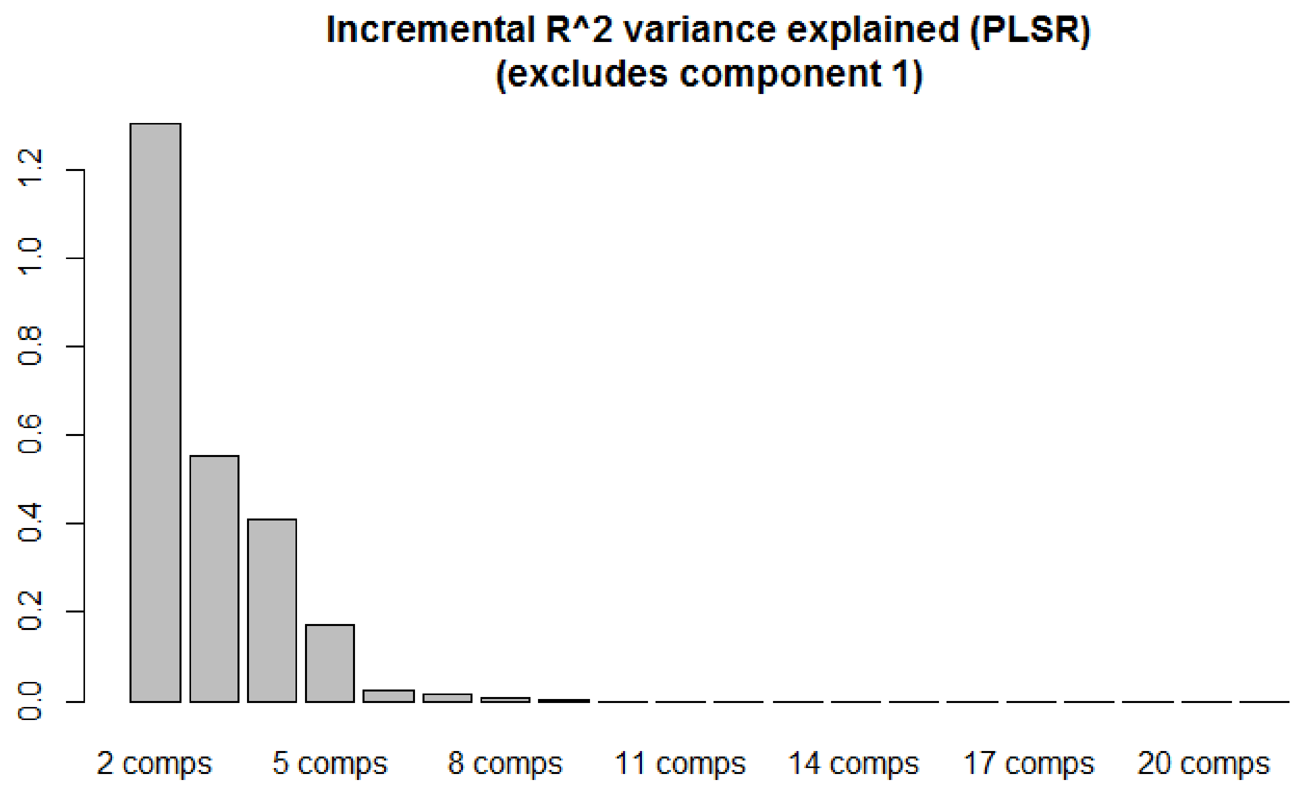


Obviously we know we should include the first component, but how about the following components. To answer this, let's look at the percentage of incremental variance explained by adding each component, after the first component. We will do this considering the R^2.

```
# first we pull out the responses R2 coefficients from the fits
pcr.comps <- 100 * drop(R2(pcr.fit, estimate = "train", intercept = FALSE)$val)
plsr.comps <- 100 * drop(R2(plsr.fit, estimate = "train", intercept = FALSE)$val)
# Next we plot the incremental change in R2 for every subsequent component after comp
1
barplot(diff(pcr.comps), main="Incremental R^2 variance explained (PCR) \n (excludes c
omponent 1)")
```



```
barplot(diff(plsr.comps), main="Incremental R^2 variance explained (PLSR)\n(excludes c
omponent 1)")
```



From the above chart for PLSR, I would take the first three components, seen in the in the chart above as well as the first component. **So in total, for PLSR, I would choose the first four components.**

For PCR the second and third comp explain nearly 1% of the variance so should be chosen. since the question implicitly assumes the standard PCR convention, which is to select the first k components, **for PCR, I would choose the first three components.**

(e) Practically speaking, it might be nice if we could guess a person's weight without measuring 21 different quantities. Do either of the methods performed above allow us to do that? If not, pick another method that will and fit it on the training data.

To answer the first question, let's look at number of the 21 features included in the first component of both PCR and PLSR. We see below that in both cases, PCR and PLSR, for the principal component 1 we would need all features - therefore this method does not serve our case for subset selection.

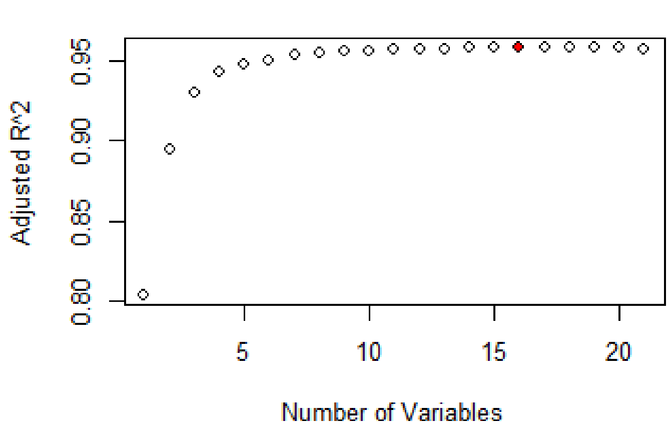
```
# Extract the loadings
comp1_pcr.load <- loadings(pcr.fit)[,1]
comp1_plsr.load <- loadings(plsr.fit)[,1]
# Print out the loading for both models.
cbind(comp1_pcr.load, comp1_plsr.load)
```

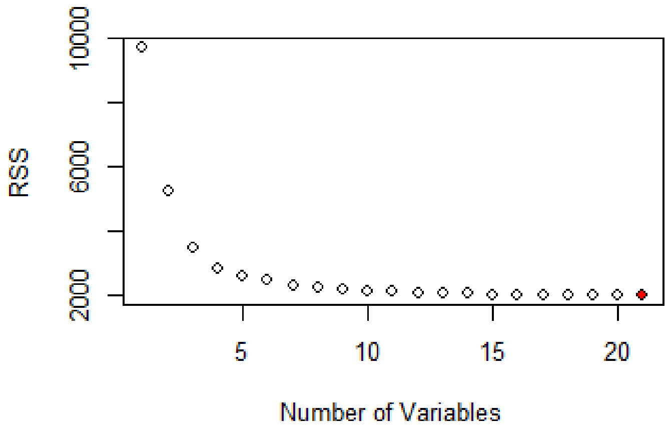
```
##                       comp1_pcr.load comp1_plsr.load
## Wrist.Diam                 0.2299843       0.2282916
## Wrist.Girth                0.2425125       0.2409802
## Forearm.Girth              0.2509594       0.2500627
## Elbow.Diam                 0.2363242       0.2346344
## Bicep.Girth                0.2467189       0.2464543
## Shoulder.Girth             0.2477947       0.2472734
## Biacromial.Diam            0.2112658       0.2096356
## Chest.Depth                0.2084904       0.2092505
## Chest.Diam                 0.2333174       0.2330909
## Chest.Girth                0.2500837       0.2501094
## Navel.Girth                0.1882279       0.1913650
## Waist.Girth                0.2399852       0.2411381
## Pelvic.Breadth             0.1311312       0.1334551
## Bitrochanteric.Diam        0.1881341       0.1900003
## Hip.Girth                  0.1948859       0.1983368
## Thigh.Girth                0.1290396       0.1329272
## Knee.Diam                  0.2230365       0.2225800
## Knee.Girth                 0.2187423       0.2202094
## Calf.Girth                 0.2156401       0.2167254
## Ankle.Diam                 0.2189421       0.2172635
## Ankle.Girth                0.2232451       0.2229910
```

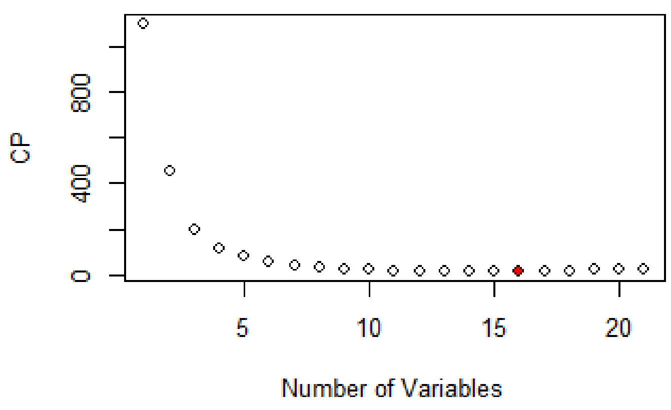
As we have only have 21 features (< 40, which is the approx. threshold computationally for best subset), we can use best subset selection for this task. Package leaps(), performs an exhaustive search for the best subsets of the variables in x for predicting y in linear regression, using an efficient branch-and-bound algorithm. Looking at the results of different measures in terms of rss (Residual sum of squares for each model), adjr2 (Adjusted r-squared), cp (Mallows' Cp) and bic (Schwartz's information criterion, BIC), based on the training data, we see most of the variance is explained using 5 variables.

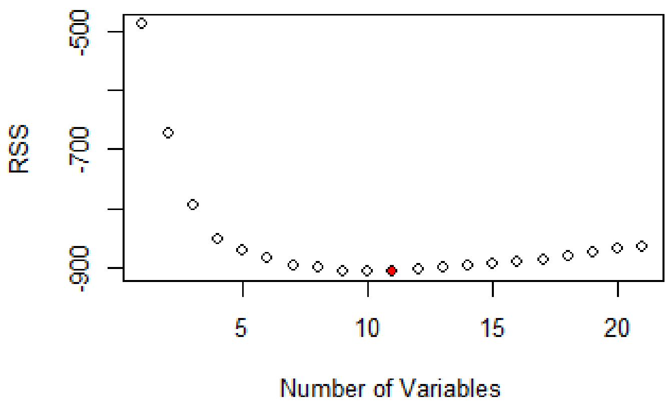
```
# load the package
library(leaps)

# fit the linear regression based best subset selection, default method is "exhaustive
"
regfit.full = regsubsets(Y.Weight ~ ., data = mydf[train,], nvmax = 21)
# output the summary and plot some of the measurement parameters
reg.summary = summary(regfit.full)
par(mfrow=c(2,2))
```

```r
# Plot the different error measurements per variable count.
plot(reg.summary$adjr2, xlab = "Number of Variables", ylab = "Adjusted R^2")
# Add a red point where the minimum is found.
points(which.max(reg.summary$adjr2), reg.summary$adjr2[which.max(reg.summary$adjr2)],
pch = 20, col = "red")

plot(reg.summary$rss, xlab = "Number of Variables", ylab = "RSS")
points(which.min(reg.summary$rss), reg.summary$rss[which.min(reg.summary$rss)], pch =
20, col = "red")

plot(reg.summary$cp, xlab = "Number of Variables", ylab = "CP")
points(which.min(reg.summary$cp), reg.summary$cp[which.min(reg.summary$cp)], pch = 20,
col = "red")

plot(reg.summary$bic, xlab = "Number of Variables", ylab = "RSS")
points(which.min(reg.summary$bic), reg.summary$bic[which.min(reg.summary$bic)], pch =
20, col = "red")
```



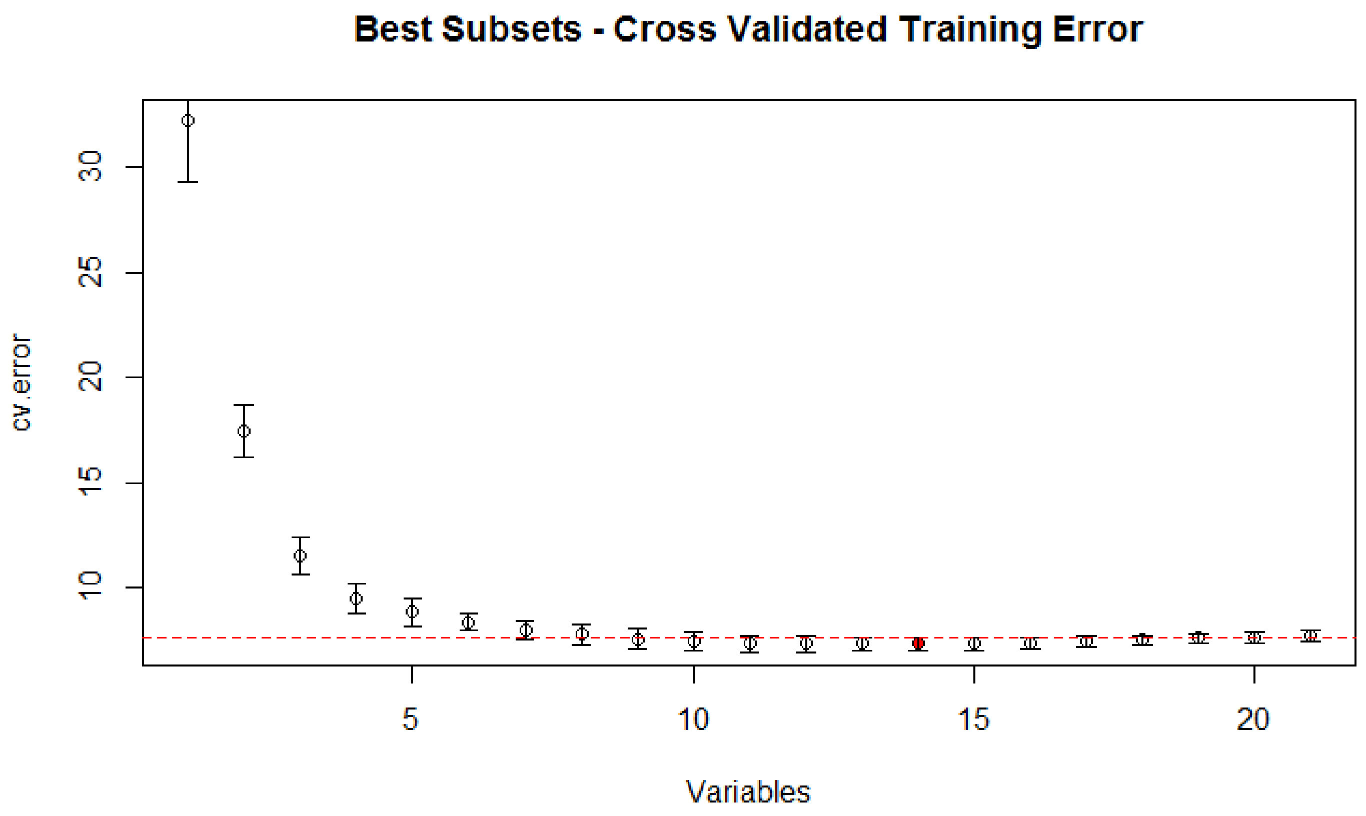






As can be seen we get variable results for the best number of predictors when using adjustments to the training error to account for the bias due to overfitting. Therefore we shall try cross validation on the training set. We shall use the one standard error rule to choose the minimum number of predictors.

```r
# Perform 5 fold cross validation on the training set to indicate where we have the lo
west error
cv.error <- rep(0,21)
cv.se <- rep(0,21)
cv.inner <- rep(0,5)
folds = sample(rep(1:10, length = length(train)))

# Create a function to calculate the std error of the errors for each degree.
std <- function(x) sd(x)/sqrt(length(x))

# Loop for each variable count of best subsets
for(i in 1:21){
  # within each variable count, loop 5 times to perform cross validation
  for(j in 1:10){
    # Break the training data into 5 folds for CV.
    trn <- train[folds!=j]
    tst <- train[folds==j]
    # create the model on the 4 folds
    bsubs.mod = glm(mydf$Y.Weight[trn] ~ ., data = mydf[trn,reg.summary$which[i,]])
    # Calculate the RSS based of the test set within the training data
    cv.inner[j] = mean((mydf$Y.Weight[tst]-predict(bsubs.mod, mydf[tst,reg.summary$whi
ch[i,]]))^2)
  }
  # Get the mean RSS accross all folds for each variable count.
  cv.error[i]=mean(cv.inner)
  cv.se[i]=std(cv.inner)
}
par(mfrow=c(1,1))
plot(1:21,cv.error, xlab="Variables", main="Best Subsets - Cross Validated Training Er
ror")

# Create a variable to store the feature count where we achieved the lowest cv error
feat.min <- which.min(cv.error)
points(feat.min, cv.error[feat.min], pch = 20, col = "red")
arrows(1:21, cv.error + cv.se, 1:21, cv.error - cv.se,length=0.05, angle=90, code=3)
abline(h=cv.error[feat.min]+cv.se[feat.min] ,col="red",lty=2)
```



It can be seen below that the minimum number of features which meet the one standard error rule is 9.

```
# check which values are one SE from the minimum
cv.error < cv.error[feat.min]+cv.se[feat.min]

##  [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE
## [12]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE FALSE
```

Lets take a look at which features are selected using 9 variables.

```
# Output the features which are chosen, when subsetting to 5 features using best subse
ts
features <- subset(reg.summary$which[9,], reg.summary$which[9,] == TRUE)
names(features)[-1]

## [1] "Forearm.Girth"   "Elbow.Diam"      "Shoulder.Girth"  "Biacromial.Diam"
## [5] "Chest.Depth"     "Waist.Girth"     "Hip.Girth"       "Thigh.Girth"
## [9] "Knee.Girth"
```

(f) Compare all 3 methods in terms of performance on the test set. Keep in mind that you should only run one version of each model on the test set. Any necessary selection of parameters should be done only with the training set.

We already have defined our PCR and PLSR model fitted in section (b) of the question, below we run it again using the chosen number of components. We still must fit our linear regression using features found in best subset.

```
# Fit the linear regression using the best subset selection
bsubs.mod <- glm(mydf$Y.Weight[train] ~ ., data = mydf[train,reg.summary$which[9,]])
```

```r
pcr.mod   <- pcr(Y.Weight ~ .,ncomp = 3, data = mydf[train,], scale=TRUE)
plsr.mod  <- plsr(Y.Weight ~ .,ncomp = 4, data = mydf[train,], scale=TRUE)
```

Not let us predict all methods on the test set, and use RSS to determine which model has the best fit.

```r
# Calculate the Residual Sum of Squares for each model
RSS.bsubs <- mean((mydf$Y.Weight[test]-predict(bsubs.mod, mydf[test,]))^2)
RSS.pcr   <- mean((mydf$Y.Weight[test]-predict(pcr.mod, mydf[test,]))^2)
RSS.plsr  <- mean((mydf$Y.Weight[test]-predict(plsr.mod, mydf[test,]))^2)
# Create a vector to output the results of each model
RSS.comparison <- c(RSS.bsubs, RSS.pcr, RSS.plsr)
names(RSS.comparison) <- c("Best subset RSS (9 Features)","PCR RSS (3 components)","PL
SR RSS (4 components)")
```

As can be seen below, the models performed relatively close. However, given the selected components and variables for all models, the best performing model, is PLSR. This is very closely followed by best subsets. With all models we have significantly reduced the number of features used.

```
RSS.comparison

## Best subset RSS (9 Features)       PCR RSS (3 components) 
##                     9.444664                    10.298720 
##      PLSR RSS (4 components) 
##                     9.441792
```