HW3

Anish Mohan

February 23, 2016

1. Q1

- 1a.
  - g_1 has a 3rd derivative as penalty hence it will tend to favor quadratic functions as the penalty function for a quadratic polynomial will be smallest. g_2 has a 4th derivative of function as penalty and hence it will till tend to favor cubic functions.
  - Give the above statent as $\lambda -> \infty$, g_2 will have a smaller training error as it is a higher order polynomial.
- 1b.
  - Give the above statent as $\lambda -> \infty$, the question is if a quadratic or a cubic polynomial fit the test data better. This depends on the true function and the distribution of data, hence we cannot reliably answer which model will have lower test error without assumptions.
  - if the true function is quadratic with noise, then g_1 will fit better as g_2 will fit to noise. Thus g_1 will have lower test error.
  - If the true function is cubic, then g_2 should fit better and should have lower test error. However, in this case if the bias due to using g_1 is lower than the variance due to noise in data, then g_1 could have lower test error as well
- 1c.
  - With $\lambda = 0$, both functions are same and will have same test and training error

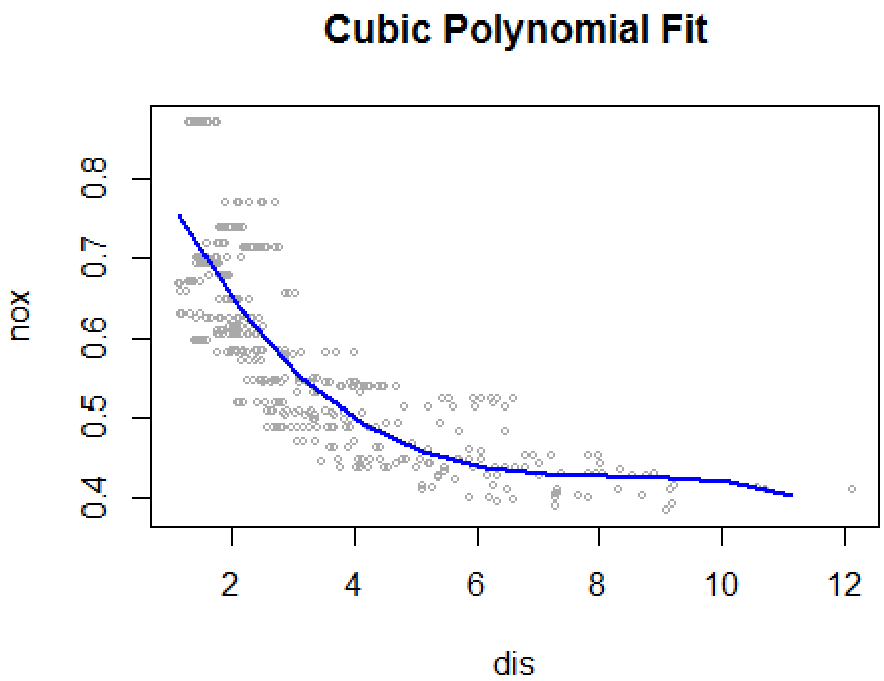
2. Q2

- 2a.

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

# Obtain the limits of the distances
dislims=range(dis)

#create grid with range of distance values
dis.grid=seq(from=dislims[1],to=dislims[2])

#fit the cubic polynomial
poly.fit=lm(nox~poly(dis,3),data=Boston)

#fit output
summary(poly.fit)
```

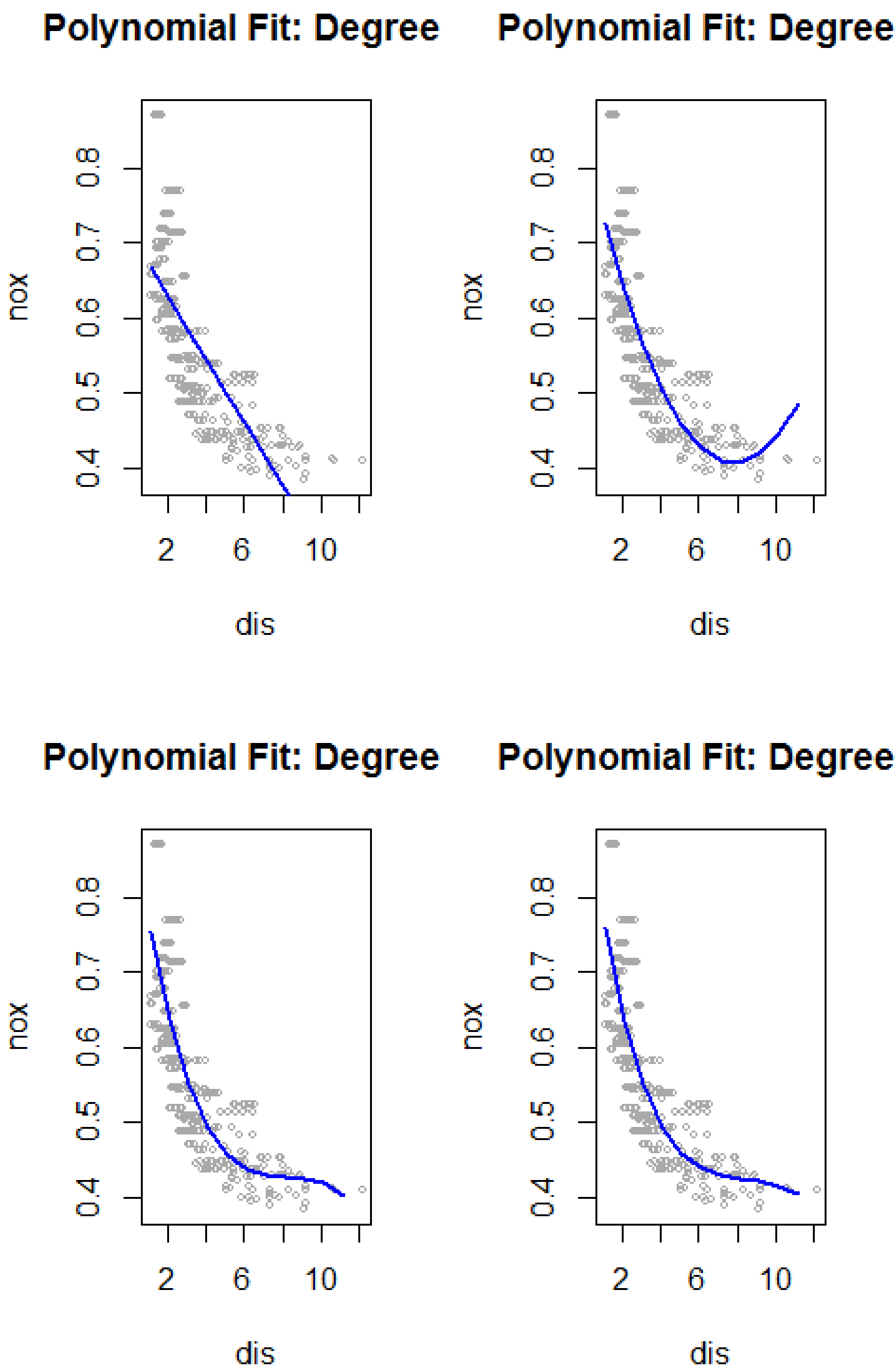
```
##
## Call:
## lm(formula = nox ~ poly(dis, 3), data = Boston)
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -0.121130 -0.040619 -0.009738  0.023385  0.194904
##
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)    0.554695   0.002759 201.021  < 2e-16 ***
## poly(dis, 3)1 -2.003096   0.062071 -32.271  < 2e-16 ***
## poly(dis, 3)2  0.856330   0.062071  13.796  < 2e-16 ***
## poly(dis, 3)3 -0.318049   0.062071  -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
```

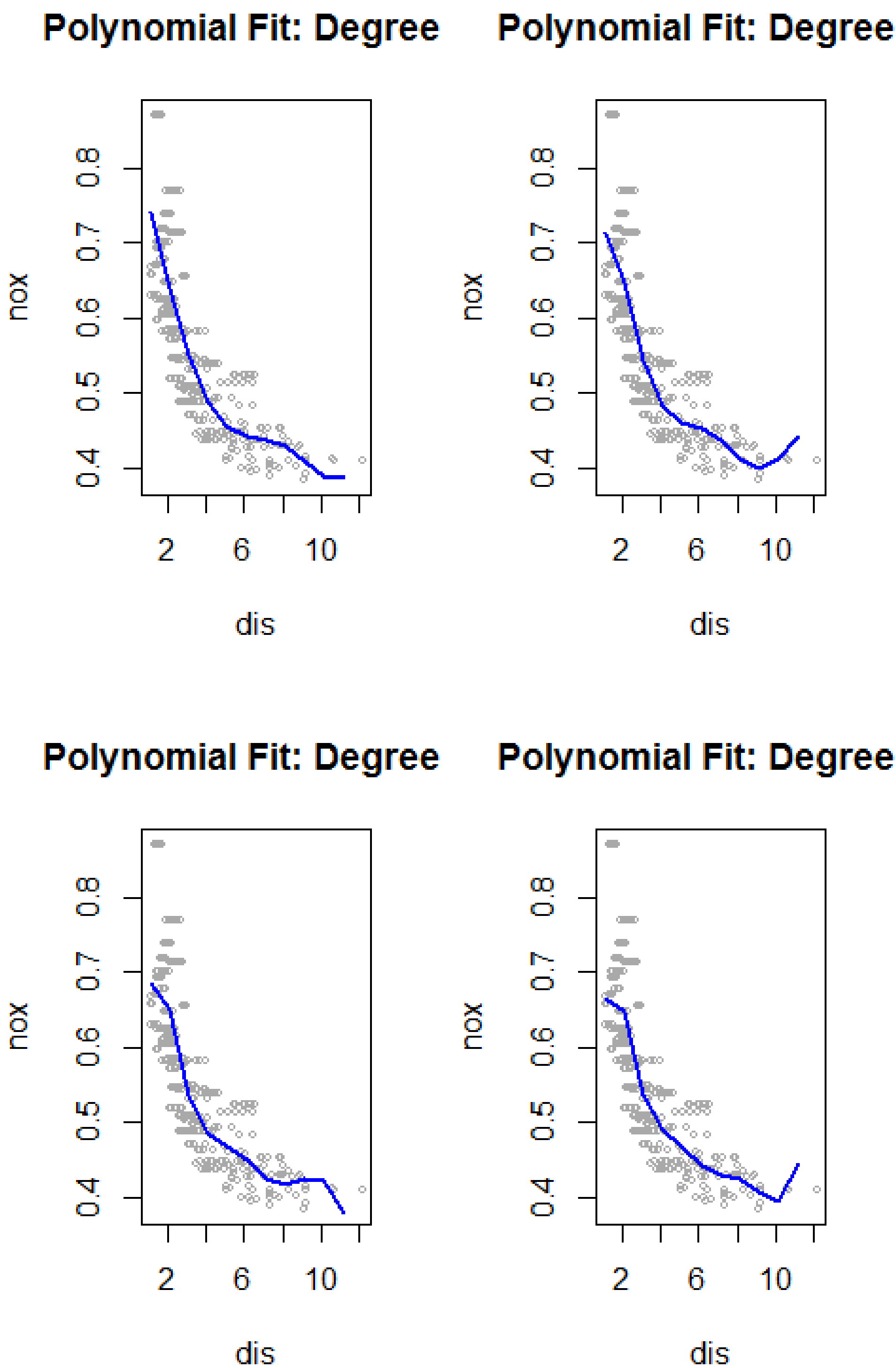
```r
#prediction and plotting
poly.pred=predict(poly.fit,newdata=list(dis=dis.grid),se=T)
plot(dis,nox,xlim=dislims,cex=0.5, col="darkgrey")
title("Cubic Polynomial Fit")
lines(dis.grid,poly.pred$fit,lwd=2, col="blue")
```

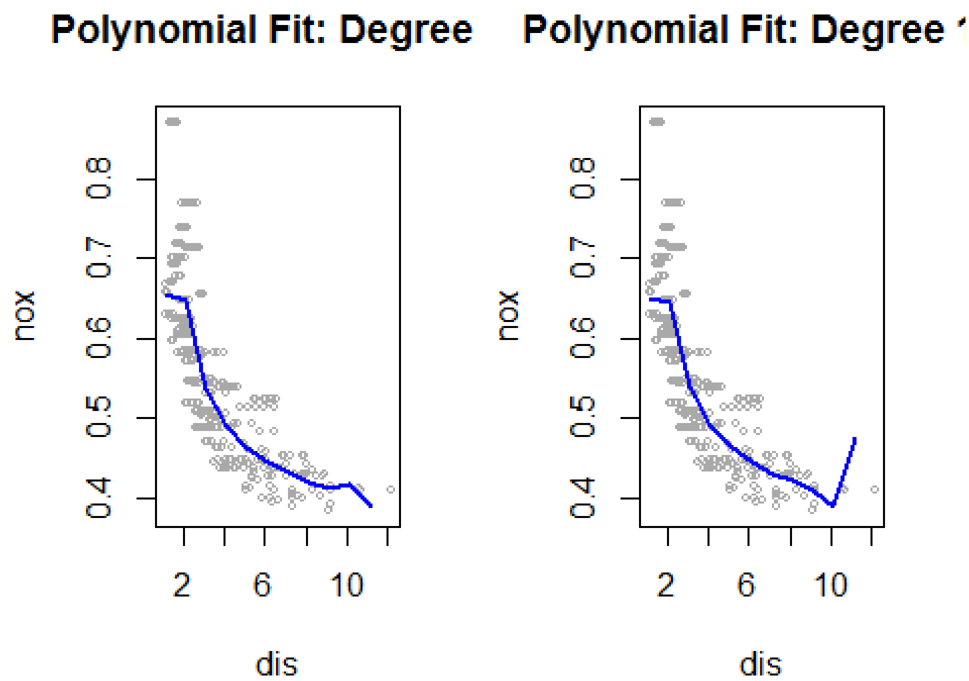


- 2b.

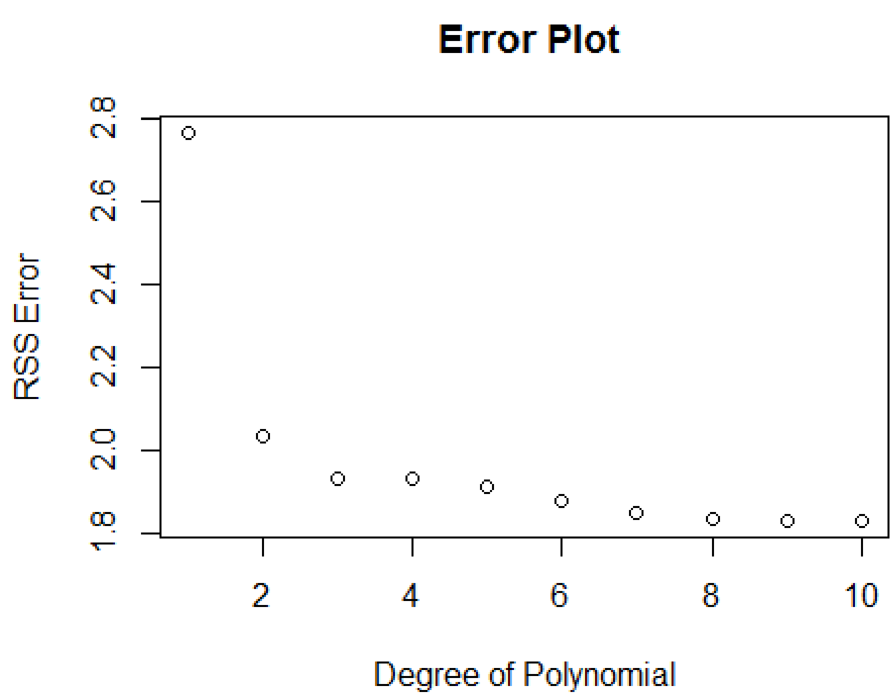
```
par(mfrow=c(1,2))
rss=rep(NA,10)

for(i in 1:10){
  poly.fit=lm(nox~poly(dis,i),data=Boston)
  rss[i]=sum(poly.fit$residuals^2)
  poly.pred=predict(poly.fit,newdata=list(dis=dis.grid),se=T)
  plot(dis,nox,xlim=dislims,cex=0.5, col="darkgrey", main=paste0("Polynom
ial Fit: Degree ", i))
  #Title("Polynomial Fit")
  lines(dis.grid,poly.pred$fit,lwd=2, col="blue")
}
```


Polynomial Fit: Degree
Polynomial Fit: Degree
Polynomial Fit: Degree
Polynomial Fit: Degree
nox
dis
0.4
0.5
0.6
0.7
0.8
2
6
10


Polynomial Fit: Degree
nox
0.4
0.5
0.6
0.7
0.8
2
6
10
dis
Polynomial Fit: Degree
nox
0.4
0.5
0.6
0.7
0.8
2
6
10
dis
Polynomial Fit: Degree
nox
0.4
0.5
0.6
0.7
0.8
2
6
10
dis
Polynomial Fit: Degree
nox
0.4
0.5
0.6
0.7
0.8
2
6
10
dis




```r
par(mfrow=c(1,1))
plot(rss, xlab="Degree of Polynomial", ylab="RSS Error", main=" Error Plo
t")
```



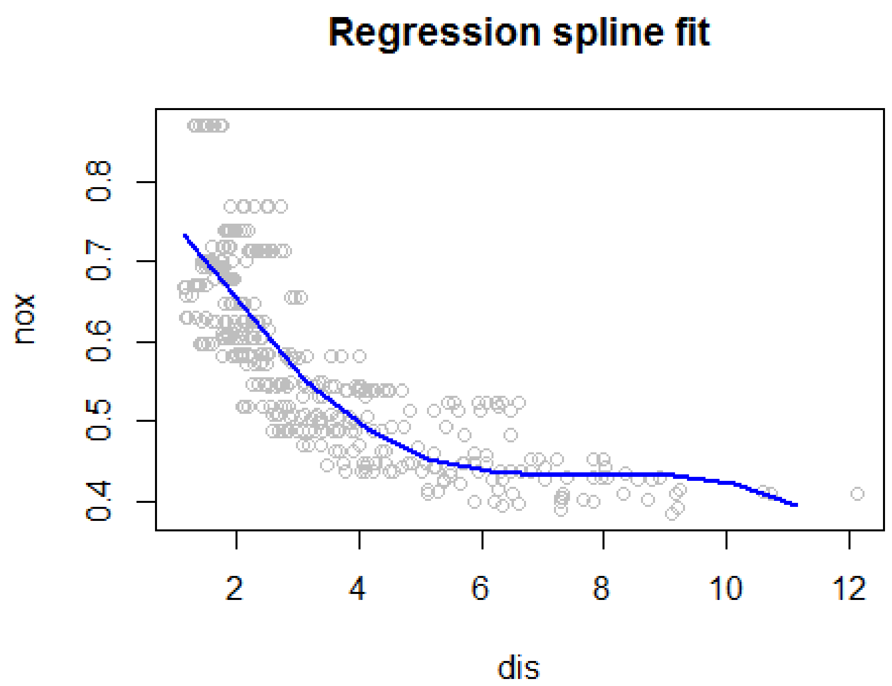
- 2c.

```
poly.fit.1=lm(nox~poly(dis,1),data=Boston)
poly.fit.2=lm(nox~poly(dis,2),data=Boston)
poly.fit.3=lm(nox~poly(dis,3),data=Boston)
poly.fit.4=lm(nox~poly(dis,4),data=Boston)
poly.fit.5=lm(nox~poly(dis,5),data=Boston)
poly.fit.6=lm(nox~poly(dis,6),data=Boston)
poly.fit.7=lm(nox~poly(dis,7),data=Boston)
poly.fit.8=lm(nox~poly(dis,8),data=Boston)
poly.fit.9=lm(nox~poly(dis,9),data=Boston)
poly.fit.10=lm(nox~poly(dis,10),data=Boston)
anova(poly.fit.1,poly.fit.2,poly.fit.3,poly.fit.4,poly.fit.5,poly.fit.6,p
oly.fit.7,poly.fit.8,poly.fit.9,poly.fit.10)

## Analysis of Variance Table
##
## Model  1: nox ~ poly(dis, 1)
## Model  2: nox ~ poly(dis, 2)
## Model  3: nox ~ poly(dis, 3)
## Model  4: nox ~ poly(dis, 4)
## Model  5: nox ~ poly(dis, 5)
## Model  6: nox ~ poly(dis, 6)
## Model  7: nox ~ poly(dis, 7)
## Model  8: nox ~ poly(dis, 8)
## Model  9: nox ~ poly(dis, 9)
## Model 10: nox ~ poly(dis, 10)
##    Res.Df    RSS Df Sum of Sq        F    Pr(>F)
## 1     504 2.7686
## 2     503 2.0353  1   0.73330 198.1169 < 2.2e-16 ***
## 3     502 1.9341  1   0.10116  27.3292 2.535e-07 ***
## 4     501 1.9330  1   0.00113   0.3040  0.581606
## 5     500 1.9153  1   0.01769   4.7797  0.029265 *
## 6     499 1.8783  1   0.03703  10.0052  0.001657 **
## 7     498 1.8495  1   0.02877   7.7738  0.005505 **
## 8     497 1.8356  1   0.01385   3.7429  0.053601 .
## 9     496 1.8333  1   0.00230   0.6211  0.431019
## 10    495 1.8322  1   0.00116   0.3133  0.575908
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

  - We use ANOVA to analyze the fits and get the best degree of freedom for the polynomial fit. The p-value when comparing a linear model to quadratic is very low ($2.2 * 10^{-16}$), hence it is a not a sufficient fit. The p-value comparing a quadratic and cubic model is very low as well. The p-value comparing cubic and quartic model is high implying that a higher degree polynomial ie. degree-4 polynomial fit is unecessary. A cubic polynomial i.e degree-3 polynomial is a good fit.

- 2d.

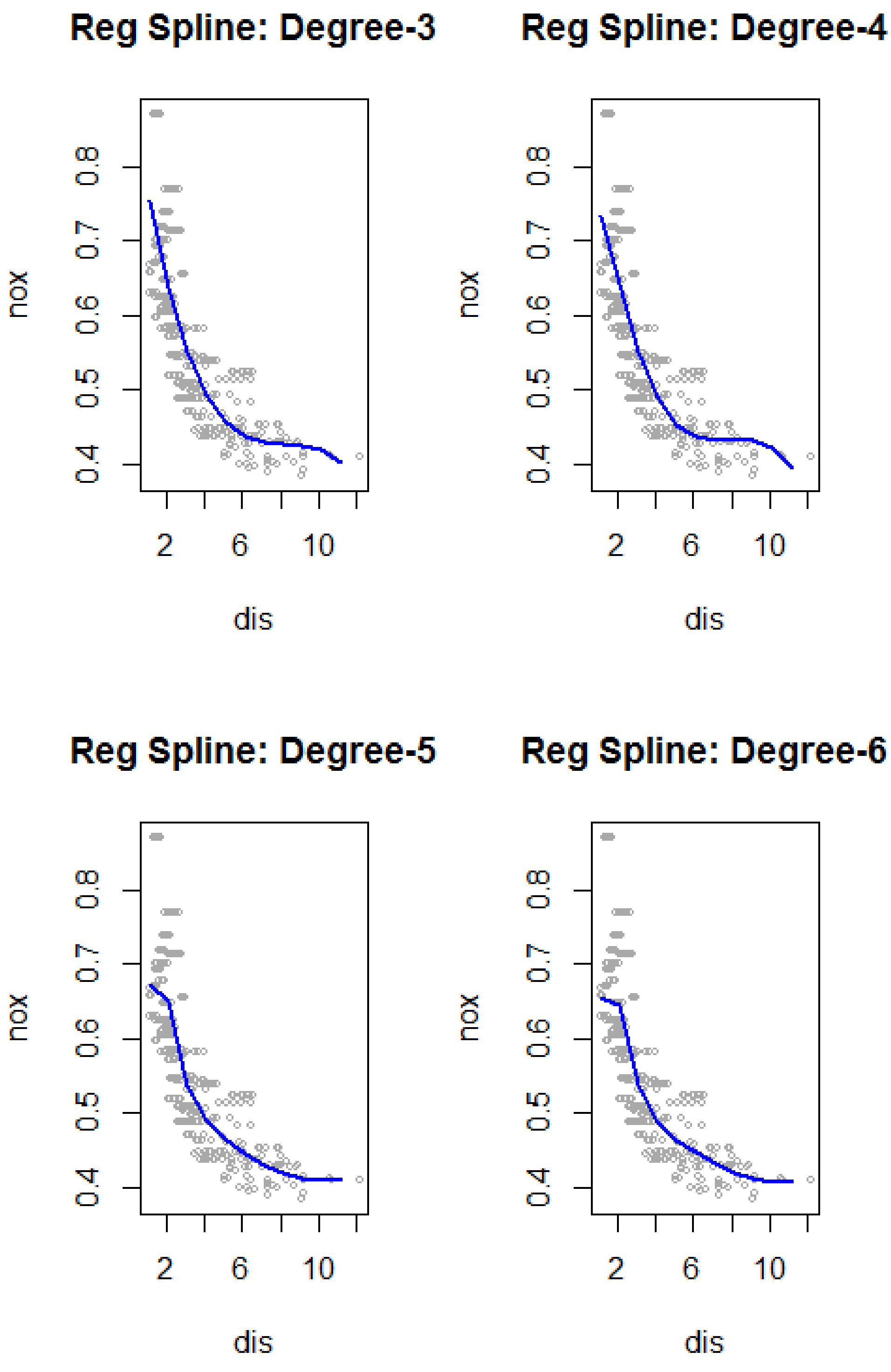
```
par(mfrow=c(1,1))
library(splines)
sp.fit=lm(nox~bs(dis,df=4),data=Boston)
sp.pred=predict(sp.fit,newdata=list(dis=dis.grid),se=T)
plot(dis,nox,col="grey",main="Regression spline fit")
lines(dis.grid,sp.pred$fit,col="blue",lwd=2)
```

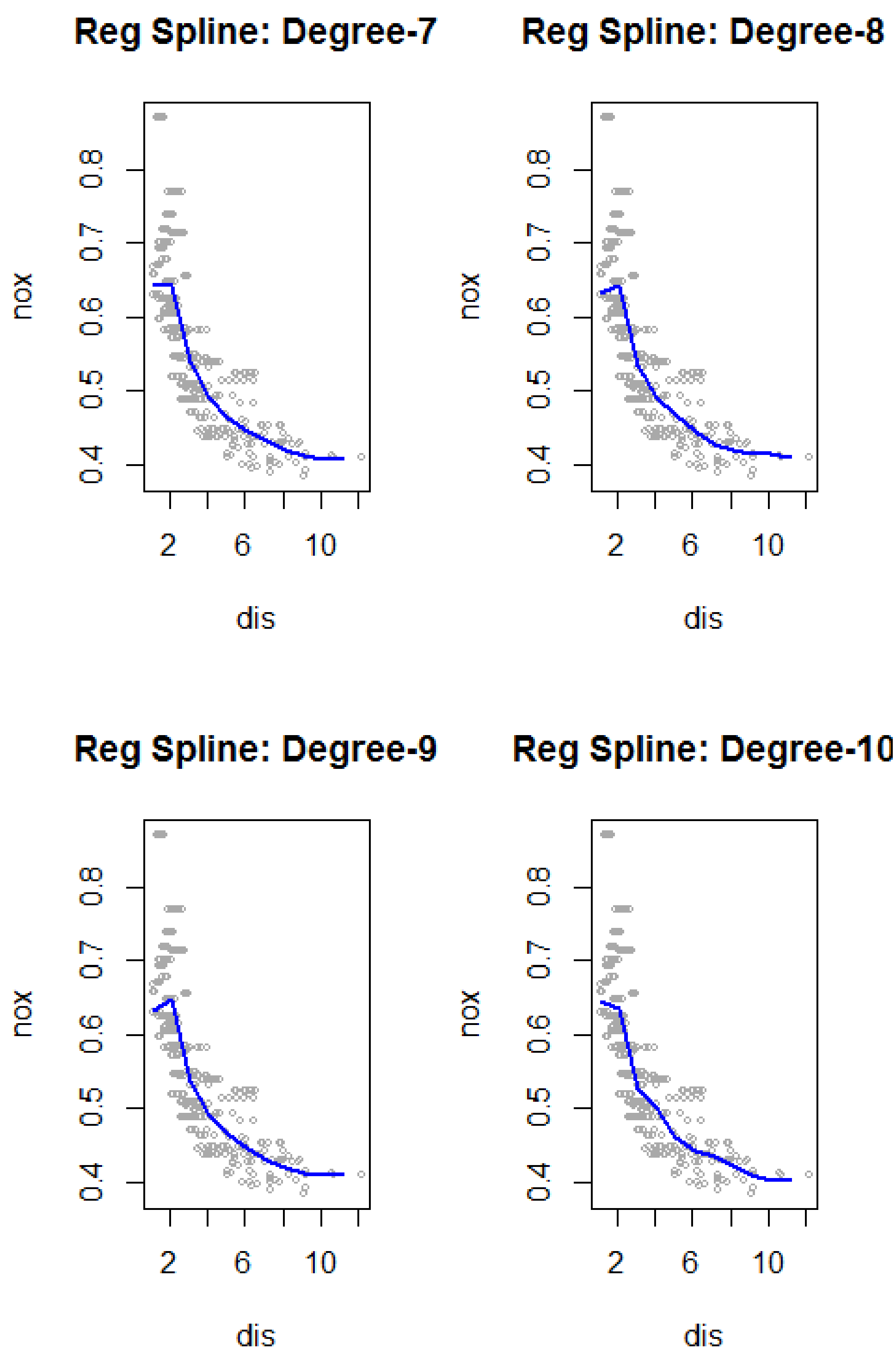


  - Equi-spaced knots were chose by the function when we just specified the degree of freedom in the equation.

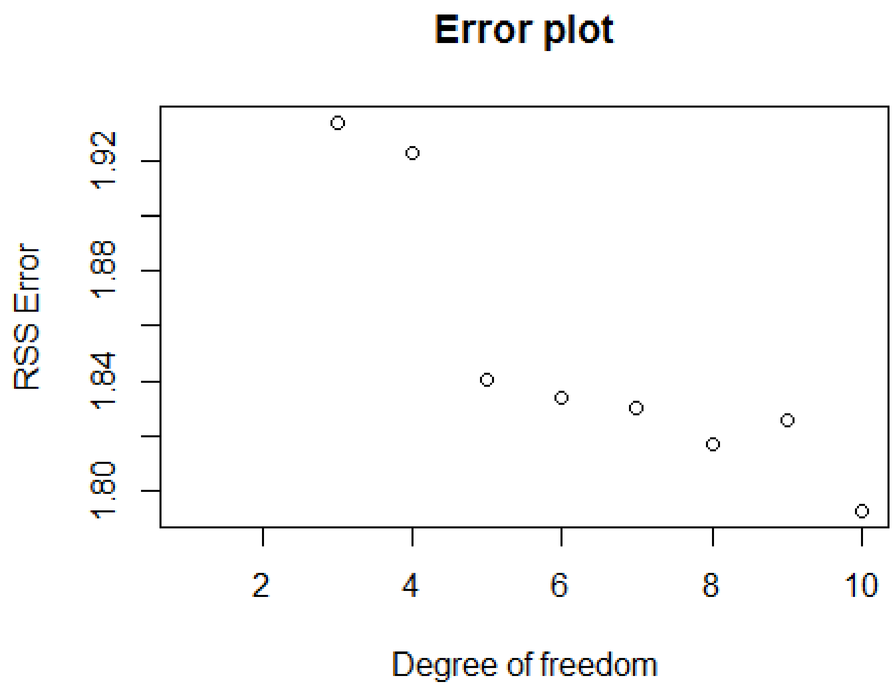
- 2e.

```
rss=rep(NA,8)
par(mfrow=c(1,2))
for(i in 3:10){
  bs.fit=lm(nox~bs(dis,df=i),data=Boston)
  rss[i]=sum(bs.fit$residuals^2)
  bs.pred=predict(bs.fit,newdata=list(dis=dis.grid),se=T)
  plot(dis,nox,xlim=dislims,cex=0.5, col="darkgrey", main=paste0("Reg Spl
ine: Degree-", i))
  lines(dis.grid,bs.pred$fit,lwd=2, col="blue")
}
```


Reg Spline: Degree-3
Reg Spline: Degree-4
Reg Spline: Degree-5
Reg Spline: Degree-6
nox
dis
0.4
0.5
0.6
0.7
0.8
2
6
10




```
par(mfrow=c(1,1))
plot(rss, xlab="Degree of freedom", ylab="RSS Error", main="Error plot")
```



- The RSS error plot shows that the error decreases with with additional degree of freedom. Reviewing the plots, a cubic or quartic polynomial seem to be the smoothest fits. High order regression splines have sharp boundaries especially at the limits of the data range.

- 2f.

```
bs.fit.3=lm(nox~bs(dis,df=3),data=Boston)
bs.fit.4=lm(nox~bs(dis,df=4),data=Boston)
bs.fit.5=lm(nox~bs(dis,df=5),data=Boston)
bs.fit.6=lm(nox~bs(dis,df=6),data=Boston)
bs.fit.7=lm(nox~bs(dis,df=7),data=Boston)
bs.fit.8=lm(nox~bs(dis,df=8),data=Boston)
bs.fit.9=lm(nox~bs(dis,df=9),data=Boston)
bs.fit.10=lm(nox~bs(dis,df=10),data=Boston)
anova(bs.fit.3,bs.fit.4,bs.fit.5,bs.fit.6,bs.fit.7,bs.fit.8,bs.fit.9,bs.f
it.10)

## Analysis of Variance Table
##
## Model 1: nox ~ bs(dis, df = 3)
## Model 2: nox ~ bs(dis, df = 4)
## Model 3: nox ~ bs(dis, df = 5)
## Model 4: nox ~ bs(dis, df = 6)
## Model 5: nox ~ bs(dis, df = 7)
## Model 6: nox ~ bs(dis, df = 8)
## Model 7: nox ~ bs(dis, df = 9)
```

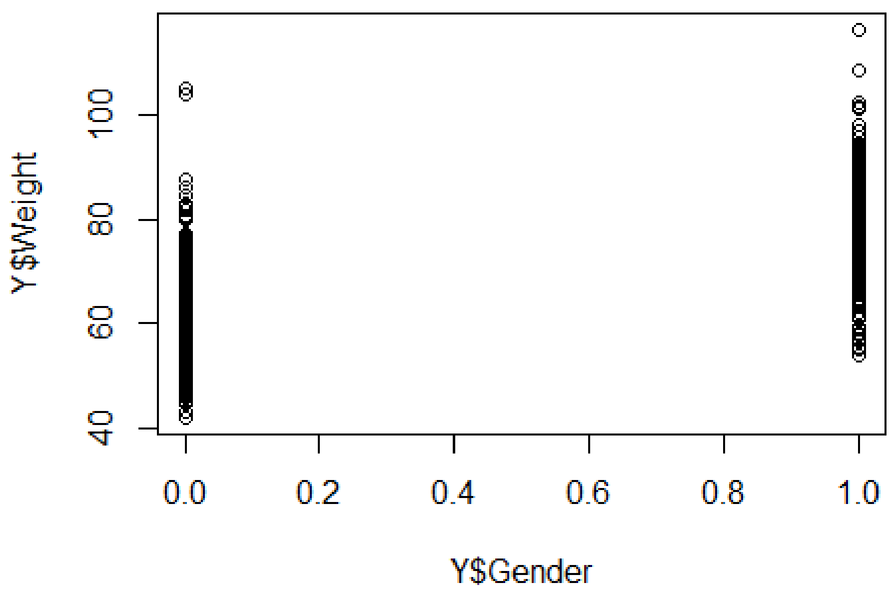
```
## Model 8: nox ~ bs(dis, df = 10)
##   Res.Df    RSS Df Sum of Sq       F    Pr(>F)
## 1    502 1.9341
## 2    501 1.9228  1  0.011332  3.1292  0.077517 .
## 3    500 1.8402  1  0.082602 22.8102 2.359e-06 ***
## 4    499 1.8340  1  0.006207  1.7140  0.191074
## 5    498 1.8299  1  0.004081  1.1271  0.288918
## 6    497 1.8170  1  0.012889  3.5593  0.059796 .
## 7    496 1.8256  1 -0.008657
## 8    495 1.7925  1  0.033118  9.1453  0.002623 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

- A regression spline with df=3 is the best fit. The regression splines function could not fit df=1/2 splines and defaulted to df=3 splines. The ANOVA function shows that a regression spline with df=3 is a good fit. A df=4 spline has a p-value of 0.07 hence not necessary or a much better fit than a df=3 regression spline.

3. Q3

- 3a.

```
library(ISLR)
library(pls)

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

bodyR=load("body.RData")
plot(Y$Gender,Y$Weight)
```



  - Here is a simple visualization showing the distribution of male and female. Assuming that the data is from an average human population, then usually mena are heavier than woman. Given that information, we see that the distribution of weights (also mean and median) of the category with Class=0 has lower wieghts than category with Class=1. From here we can find out that Class=1 are the Males and Class=0 are the females

- 3b.

```
set.seed(1)
train=sample(507,307)
```

```
test=-train
X.train=X[train,]
X.test=X[test,]
Y.test=Y[test,"Weight"]
Y.train=Y[train,"Weight"]


set.seed(1)
#PCR Fit
pcr.fit=pcr(Y.train~.,data=X.train, scale=TRUE, validation="CV")
#PLS Fit
set.seed(1)
pls.fit=plsr(Y.train~.,data=X.train, scale=TRUE, validation="CV")
```

- – The variables measured here have different range of measurements depending on the body port. Some measurements like Wrist diameter or girth are going to inherently smaller than othe measurments like Hip Girth or Diameter. To ensure that magnitude of these measurement do not impact the principal components, we choose to standardize so that measurements are interms of how many sds are the measurement from their mean.

- 3c.

```
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 com
ps
## CV           13.34    3.428    3.266    3.000    2.969    2.963    2.9
40
## adjCV        13.34    3.426    3.264    2.977    2.966    2.960    2.9
37
##        7 comps  8 comps  9 comps  10 comps  11 comps  12 comps  13 com
ps
## CV       2.956    2.922    2.940     2.921     2.930     2.923     2.9
13
## adjCV    2.953    2.918    2.937     2.916     2.926     2.916     2.9
08
##        14 comps  15 comps  16 comps  17 comps  18 comps  19 comps
## CV        2.906     2.859     2.792     2.769     2.788     2.804
## adjCV     2.898     2.852     2.782     2.758     2.777     2.793
##        20 comps  21 comps
## CV        2.808     2.808
## adjCV     2.797     2.796
```

```
##
## TRAINING: % variance explained
##          1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps
## X          63.08    75.20    79.96    84.46    86.77    88.89    90.37
## Y.train    93.46    94.12    95.18    95.20    95.27    95.32    95.38
##          8 comps  9 comps  10 comps  11 comps  12 comps  13 comps
## X          91.80    93.08     94.19     95.17     96.05     96.82
## Y.train    95.47    95.47     95.52     95.54     95.60     95.61
##          14 comps  15 comps  16 comps  17 comps  18 comps  19 comps
## X           97.54     98.10     98.57     98.97     99.34     99.60
## Y.train     95.71     95.88     96.08     96.20     96.20     96.21
##          20 comps  21 comps
## X           99.82    100.00
## Y.train     96.21     96.23
```

```
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 com
ps
## CV           13.34    3.324    2.991    2.863    2.816    2.801    2.7
92
## adjCV        13.34    3.322    2.989    2.859    2.806    2.789    2.7
81
##        7 comps  8 comps  9 comps  10 comps  11 comps  12 comps  13 com
ps
## CV       2.796    2.802    2.807     2.810     2.810     2.809     2.8
09
## adjCV    2.785    2.790    2.795     2.798     2.798     2.797     2.7
97
##        14 comps  15 comps  16 comps  17 comps  18 comps  19 comps
## CV        2.808     2.808     2.808     2.808     2.808     2.808
## adjCV     2.796     2.796     2.796     2.796     2.796     2.796
##        20 comps  21 comps
## CV        2.808     2.808
## adjCV     2.796     2.796
##
## TRAINING: % variance explained
##          1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps
## X          63.06    73.25    79.60    81.27    82.80    85.27    88.37
## Y.train    93.88    95.17    95.67    96.07    96.19    96.21    96.22
##          8 comps  9 comps  10 comps  11 comps  12 comps  13 comps
## X          89.55    91.07     92.05     92.80     93.66     94.67
## Y.train    96.23    96.23     96.23     96.23     96.23     96.23
```

```
##          14 comps  15 comps  16 comps  17 comps  18 comps  19 comps
## X           95.56     96.35     97.19     97.77     98.57     99.01
## Y.train     96.23     96.23     96.23     96.23     96.23     96.23
##          20 comps  21 comps
## X           99.66    100.00
## Y.train     96.23     96.23
```

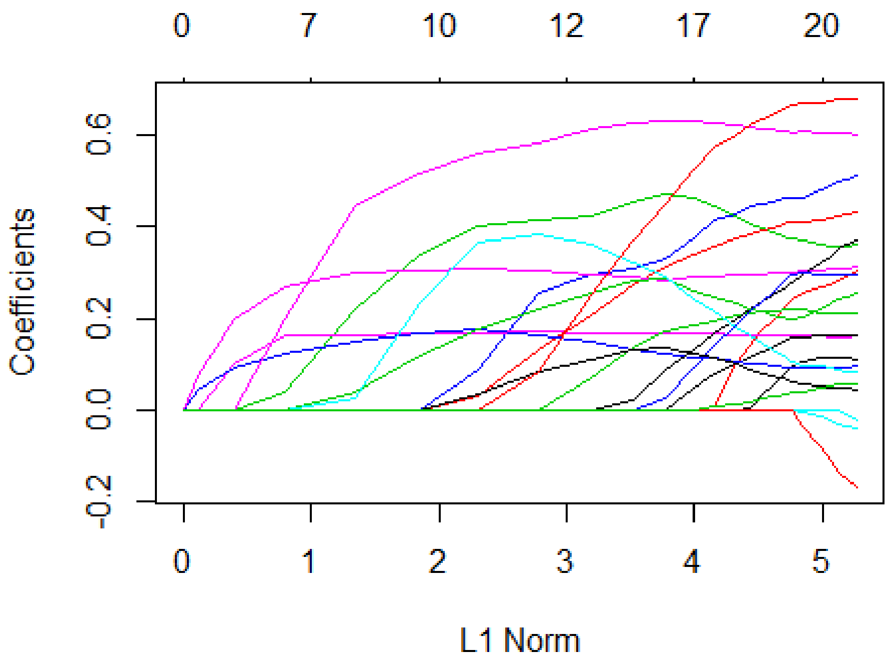
- The % of training variance explained by PCR and PLS are very similar. This is not surprising as both the process depend on finding the Principal components first. Principal components generally do capture the maximum variation in the input data. PLS regresses the values of Y on principal components and hence is expected to do have higher % of variance explained than PCR, which it does albiet with the improvement is minor.

- 3d.
  - We can choose the number of components by reviewing the CV error and the % of variance explained and choose the simplest model that has a reasonable fit. In this example, CV error for PCR and PLS reduces significantly adding first few principal components but after that the reduction error is marginal. For e.g N=3 seems be a reasonable fit to have a low CV error (<3.0) and about 95% of the variance of the data explained.

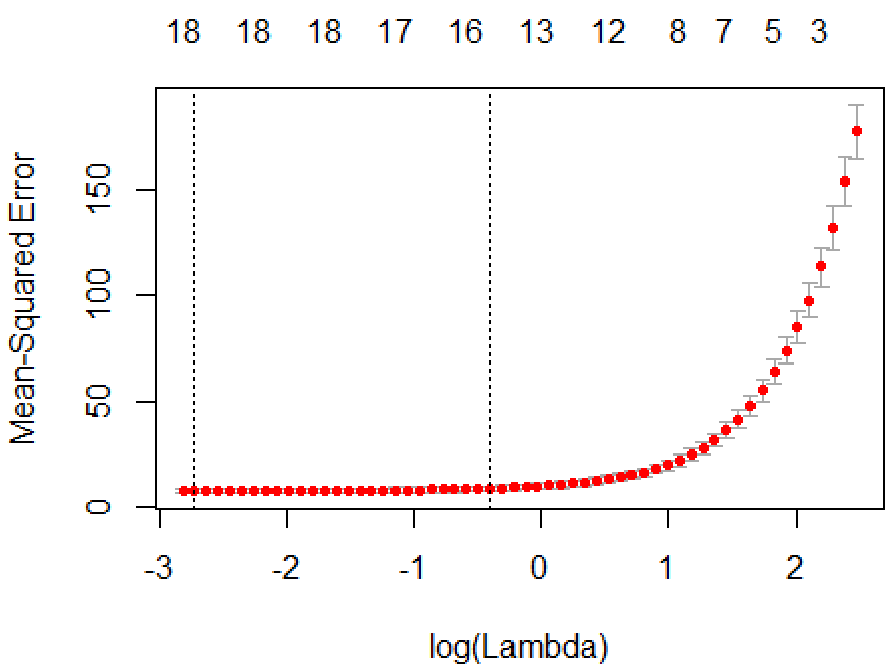
```
pcr.pred=predict(pcr.fit,X[test,], ncomp=3)
mean((pcr.pred-Y.test)^2)

## [1] 8.778749

pls.pred=predict(pls.fit,X[test,], ncomp=3)
mean((pls.pred-Y.test)^2)

## [1] 8.370953
```

- 3e.
  - We show that you can do some variable selection with Lasso and more variable selection with Forward Selection and get comparable error rate to PCR and PLS with a better model interpretability

```
#Running Lasso
library(glmnet)

## Loading required package: Matrix
## Loading required package: foreach
## Loaded glmnet 2.0-3

body=data.frame(Weight=Y$Weight,X)
xl=model.matrix(Weight~.,body)[,-1]
yl=body$Weight

grid=10^seq(10,-2, length=100)
lasso.mod=glmnet(xl[train,], yl[train],alpha=1,lambda=grid)
plot(lasso.mod)
```




```
set.seed(1)
cv.out=cv.glmnet(xl[train,],yl[train],alpha=1)
plot(cv.out)
```



```r
bestlam=cv.out$lambda.min

lasso.pred=predict(lasso.mod,s=bestlam, newx=xl[test,])
lasso.coef=predict(cv.out,type="coefficients",s=bestlam)
lasso.coef

## 22 x 1 sparse Matrix of class "dgCMatrix"
##                                1
## (Intercept)         -100.45764492
## Wrist.Diam             0.09285864
## Wrist.Girth            0.23600867
## Forearm.Girth          0.37524915
## Elbow.Diam             0.46117144
## Bicep.Girth            .
## Shoulder.Girth         0.16318198
## Biacromial.Diam        0.15389404
## Chest.Depth            0.40779724
## Chest.Diam             0.21823430
## Chest.Girth            0.09231997
## Navel.Girth            .
## Waist.Girth            0.30016716
## Pelvic.Breadth         0.27411671
## Bitrochanteric.Diam    .
## Hip.Girth              0.19807114
## Thigh.Girth            0.29043305
## Knee.Diam              0.11038084
## Knee.Girth             0.60539062
## Calf.Girth             0.06393019
## Ankle.Diam             0.65893472
## Ankle.Girth            0.03846260

print(paste0("Unfortunately with the best value of lambda, most of the va
riables are selected."))

## [1] "Unfortunately with the best value of lambda, most of the variable
s are selected."

#Variable Selection
library(leaps)
regfit.fwd=regsubsets(Y.train~., data=X.train, method="forward", nv=20)
summary(regfit.fwd)

## Subset selection object
## Call: regsubsets.formula(Y.train ~ ., data = X.train, method = "forwar
d",
##     nv = 20)
## 21 Variables  (and intercept)
##                      Forced in Forced out
## Wrist.Diam               FALSE      FALSE
## Wrist.Girth              FALSE      FALSE
```

```
## Forearm.Girth            FALSE      FALSE
## Elbow.Diam               FALSE      FALSE
## Bicep.Girth              FALSE      FALSE
## Shoulder.Girth           FALSE      FALSE
## Biacromial.Diam          FALSE      FALSE
## Chest.Depth              FALSE      FALSE
## Chest.Diam               FALSE      FALSE
## Chest.Girth              FALSE      FALSE
## Navel.Girth              FALSE      FALSE
## Waist.Girth              FALSE      FALSE
## Pelvic.Breadth           FALSE      FALSE
## Bitrochanteric.Diam      FALSE      FALSE
## Hip.Girth                FALSE      FALSE
## Thigh.Girth              FALSE      FALSE
## Knee.Diam                FALSE      FALSE
## Knee.Girth               FALSE      FALSE
## Calf.Girth               FALSE      FALSE
## Ankle.Diam               FALSE      FALSE
## Ankle.Girth              FALSE      FALSE
## 1 subsets of each size up to 20
## Selection Algorithm: forward
##            Wrist.Diam Wrist.Girth Forearm.Girth Elbow.Diam Bicep.Girth
## 1  ( 1 )  " "        " "         " "           " "        " "
## 2  ( 1 )  " "        " "         "*"           " "        " "
## 3  ( 1 )  " "        " "         "*"           " "        " "
## 4  ( 1 )  " "        " "         "*"           " "        " "
## 5  ( 1 )  " "        " "         "*"           " "        " "
## 6  ( 1 )  " "        " "         "*"           " "        " "
## 7  ( 1 )  " "        " "         "*"           " "        " "
## 8  ( 1 )  " "        " "         "*"           " "        " "
## 9  ( 1 )  " "        " "         "*"           " "        " "
## 10  ( 1 ) " "        " "         "*"           " "        " "
## 11  ( 1 ) " "        " "         "*"           "*"        " "
## 12  ( 1 ) " "        " "         "*"           "*"        " "
## 13  ( 1 ) " "        " "         "*"           "*"        " "
## 14  ( 1 ) " "        "*"         "*"           "*"        " "
## 15  ( 1 ) " "        "*"         "*"           "*"        " "
## 16  ( 1 ) " "        "*"         "*"           "*"        " "
## 17  ( 1 ) " "        "*"         "*"           "*"        "*"
## 18  ( 1 ) " "        "*"         "*"           "*"        "*"
## 19  ( 1 ) " "        "*"         "*"           "*"        "*"
## 20  ( 1 ) " "        "*"         "*"           "*"        "*"
##            Shoulder.Girth Biacromial.Diam Chest.Depth Chest.Diam
## 1  ( 1 )  " "            " "             " "         " "
## 2  ( 1 )  " "            " "             " "         " "
## 3  ( 1 )  " "            " "             " "         " "
## 4  ( 1 )  " "            " "             " "         " "
## 5  ( 1 )  " "            " "             " "         " "
## 6  ( 1 )  "*"            " "             " "         " "
## 7  ( 1 )  "*"            " "             "*"         " "
```

```
## 8  ( 1 )  "*"            " "             "*"          " "
## 9  ( 1 )  "*"            " "             "*"          " "
## 10  ( 1 ) "*"            " "             "*"          "*"
## 11  ( 1 ) "*"            " "             "*"          "*"
## 12  ( 1 ) "*"            "*"             "*"          "*"
## 13  ( 1 ) "*"            "*"             "*"          "*"
## 14  ( 1 ) "*"            "*"             "*"          "*"
## 15  ( 1 ) "*"            "*"             "*"          "*"
## 16  ( 1 ) "*"            "*"             "*"          "*"
## 17  ( 1 ) "*"            "*"             "*"          "*"
## 18  ( 1 ) "*"            "*"             "*"          "*"
## 19  ( 1 ) "*"            "*"             "*"          "*"
## 20  ( 1 ) "*"            "*"             "*"          "*"
##           Chest.Girth Navel.Girth Waist.Girth Pelvic.Breadth
## 1  ( 1 )  " "         " "         "*"         " "
## 2  ( 1 )  " "         " "         "*"         " "
## 3  ( 1 )  " "         " "         "*"         " "
## 4  ( 1 )  " "         " "         "*"         " "
## 5  ( 1 )  " "         " "         "*"         " "
## 6  ( 1 )  " "         " "         "*"         " "
## 7  ( 1 )  " "         " "         "*"         " "
## 8  ( 1 )  " "         " "         "*"         " "
## 9  ( 1 )  " "         " "         "*"         "*"
## 10  ( 1 ) " "         " "         "*"         "*"
## 11  ( 1 ) " "         " "         "*"         "*"
## 12  ( 1 ) " "         " "         "*"         "*"
## 13  ( 1 ) "*"         " "         "*"         "*"
## 14  ( 1 ) "*"         " "         "*"         "*"
## 15  ( 1 ) "*"         "*"         "*"         "*"
## 16  ( 1 ) "*"         "*"         "*"         "*"
## 17  ( 1 ) "*"         "*"         "*"         "*"
## 18  ( 1 ) "*"         "*"         "*"         "*"
## 19  ( 1 ) "*"         "*"         "*"         "*"
## 20  ( 1 ) "*"         "*"         "*"         "*"
##           Bitrochanteric.Diam Hip.Girth Thigh.Girth Knee.Diam Knee.Gir
th
## 1  ( 1 )  " "                 " "       " "         " "       " "
## 2  ( 1 )  " "                 " "       " "         " "       " "
## 3  ( 1 )  " "                 "*"       " "         " "       " "
## 4  ( 1 )  " "                 "*"       " "         " "       "*"
## 5  ( 1 )  " "                 "*"       " "         " "       "*"
## 6  ( 1 )  " "                 "*"       " "         " "       "*"
## 7  ( 1 )  " "                 "*"       " "         " "       "*"
## 8  ( 1 )  " "                 "*"       "*"         " "       "*"
## 9  ( 1 )  " "                 "*"       "*"         " "       "*"
## 10  ( 1 ) " "                 "*"       "*"         " "       "*"
## 11  ( 1 ) " "                 "*"       "*"         " "       "*"
## 12  ( 1 ) " "                 "*"       "*"         " "       "*"
## 13  ( 1 ) " "                 "*"       "*"         " "       "*"
## 14  ( 1 ) " "                 "*"       "*"         " "       "*"
```

```
## 15  ( 1 ) " "                    "*"       "*"         " "       "*"
## 16  ( 1 ) "*"                    "*"       "*"         " "       "*"
## 17  ( 1 ) "*"                    "*"       "*"         " "       "*"
## 18  ( 1 ) "*"                    "*"       "*"         " "       "*"
## 19  ( 1 ) "*"                    "*"       "*"         "*"       "*"
## 20  ( 1 ) "*"                    "*"       "*"         "*"       "*"
##           Calf.Girth Ankle.Diam Ankle.Girth
## 1  ( 1 )  " "        " "        " "
## 2  ( 1 )  " "        " "        " "
## 3  ( 1 )  " "        " "        " "
## 4  ( 1 )  " "        " "        " "
## 5  ( 1 )  " "        "*"        " "
## 6  ( 1 )  " "        "*"        " "
## 7  ( 1 )  " "        "*"        " "
## 8  ( 1 )  " "        "*"        " "
## 9  ( 1 )  " "        "*"        " "
## 10  ( 1 ) " "        "*"        " "
## 11  ( 1 ) " "        "*"        " "
## 12  ( 1 ) " "        "*"        " "
## 13  ( 1 ) " "        "*"        " "
## 14  ( 1 ) " "        "*"        " "
## 15  ( 1 ) " "        "*"        " "
## 16  ( 1 ) " "        "*"        " "
## 17  ( 1 ) " "        "*"        " "
## 18  ( 1 ) "*"        "*"        " "
## 19  ( 1 ) "*"        "*"        " "
## 20  ( 1 ) "*"        "*"        "*"

#apply linear regression with variables selected from forward selection
lm.fit=lm(Y.train~Forearm.Girth+Waist.Girth+Hip.Girth, data=X.train)
lm.pred=predict(lm.fit,X.train)
mean((Y.train-lm.pred)^2)

## [1] 12.73465

lm.testpred=predict(lm.fit,newdata=X.test)
summary(lm.fit)

##
## Call:
## lm(formula = Y.train ~ Forearm.Girth + Waist.Girth + Hip.Girth,
##     data = X.train)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -9.8061 -2.4854 -0.2277  2.1819 16.0035
##
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)   -74.53249    3.36911  -22.12   <2e-16 ***
```

```
## Forearm.Girth   2.01695    0.10868   18.56   <2e-16 ***
## Waist.Girth     0.48661    0.03325   14.64   <2e-16 ***
## Hip.Girth       0.55700    0.04161   13.39   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3.592 on 303 degrees of freedom
## Multiple R-squared:  0.928,  Adjusted R-squared:  0.9273
## F-statistic:  1301 on 3 and 303 DF,  p-value: < 2.2e-16
```

- 3f.

```
  print(paste0("PCR Error (N=3):",mean((pcr.pred-Y.test)^2)))

## [1] "PCR Error (N=3):8.77874910783801"

  print(paste0("PLS Error (N=3):", mean((pls.pred-Y.test)^2)))

## [1] "PLS Error (N=3):8.37095343541857"

  print(paste0("LASSO Error:",  mean((lasso.pred-yl[test])^2)))

## [1] "LASSO Error:7.96214425943955"

  print(paste0("Variable Selection Error:",mean((Y.test-lm.testpred)^2)))

## [1] "Variable Selection Error:13.1441842901964"
```

  - Comparing the results on test data, PCR and PLS with (N=3) have bit better error rate than Variable Selection with 3 variables. But with Variable selection it is very clear which 3 variables are the most critical one. Lasso has a low error rate but the best lamda with lowest CV error ends up choosing most of the variables.