STAT 6340 Mini Project Bonus

Matthew Lynn

May 3, 2019

Section 1

Section 2 is coded in Section 1

Question 1

Consider the Caravan dataset from Mini Projects 5 and 6. Standardize the predictors and split the data into training and test sets just as in the previous projects.

```
set.seed(1)
library(ISLR)
# I standardized through the function itself, won't work for me otherwise
caravan = Caravan
test = 1:1000
train = (-test)
xtrain = caravan[train,-86]
xtest = caravan[test,]
ytrain = caravan$Purchase[train]
ytest = caravan$Purchase[test]
library(e1071)

# funcion to reduce data by 90% for testing
# toyify = function(data) {
#   set.seed(1)
#   sampler = sample(1:nrow(data), 582)
#   data[sampler,]
# }
# caravan = toyify(caravan)
# test = 1:100
# train = (-test)
# xtrain = caravan[train,-86]
# xtest = caravan[test,]
# ytrain = caravan$Purchase[train]
# ytest = caravan$Purchase[test]
```

(a)

Fit a support vector classifier to the training data with cost parameter chosen optimally using 10-fold cross-validation. Evaluate its performance on the test data. Summarize your results

```
tune.out = tune(svm, Purchase~., data = caravan[train,], scale = T, kernel = "linear",
                ranges = list(cost = c(0.001,0.01,0.1,1,5,10)))
summary(tune.out)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##   cost
##  0.001
##
## - best performance: 0.0599323
##
## - Detailed performance results:
##    cost      error  dispersion
## 1 1e-03 0.05993230 0.006651245
## 2 1e-02 0.05993230 0.006651245
## 3 1e-01 0.06034638 0.006925258
## 4 1e+00 0.06013934 0.006757994
## 5 5e+00 0.06034681 0.006354060
## 6 1e+01 0.06076132 0.006616303
```

```
bestmod = tune.out$best.model
pred = predict(bestmod, xtest)
caret::confusionMatrix(pred,ytest, positive = 'Yes')
```

```
## Confusion Matrix and Statistics
##
##          Reference
## Prediction  No Yes
##        No  941  59
##        Yes   0   0
##
##               Accuracy : 0.941
##                 95% CI : (0.9246, 0.9548)
##    No Information Rate : 0.941
##    P-Value [Acc > NIR] : 0.5346
##
##                  Kappa : 0
##  Mcnemar's Test P-Value : 4.321e-14
##
##             Sensitivity : 0.000
##             Specificity : 1.000
##          Pos Pred Value :   NaN
##          Neg Pred Value : 0.941
##              Prevalence : 0.059
##          Detection Rate : 0.000
##    Detection Prevalence : 0.000
##       Balanced Accuracy : 0.500
##
##        'Positive' Class : Yes
##
```

```
svm.lin.err = mean(pred != ytest)
```

(b)

Repeat (a) using a support vector machine with a polynomial kernel of degree two.

```
tune.out = tune(svm, Purchase~., data = caravan[train,], kernel = "polynomial", degree = 2,
                ranges = list(cost = c(0.001,0.01,0.1,1,5,10,100)))
summary(tune.out)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##   cost
##  0.001
##
## - best performance: 0.05992758
##
## - Detailed performance results:
##    cost      error dispersion
## 1 1e-03 0.05992758 0.01226739
## 2 1e-02 0.05992758 0.01226739
## 3 1e-01 0.06013505 0.01205364
## 4 1e+00 0.06262510 0.01084416
## 5 5e+00 0.06635826 0.01094048
## 6 1e+01 0.06946943 0.01073140
## 7 1e+02 0.09746785 0.00680203
```

```
bestmod = tune.out$best.model
pred = predict(bestmod, xtest)
caret::confusionMatrix(pred,ytest, positive = 'Yes')
```

```
## Confusion Matrix and Statistics
##
##          Reference
## Prediction  No Yes
##        No  941  59
##        Yes   0   0
##
##               Accuracy : 0.941
##                 95% CI : (0.9246, 0.9548)
##    No Information Rate : 0.941
##    P-Value [Acc > NIR] : 0.5346
##
##                  Kappa : 0
##  Mcnemar's Test P-Value : 4.321e-14
##
##             Sensitivity : 0.000
##             Specificity : 1.000
##          Pos Pred Value :   NaN
##          Neg Pred Value : 0.941
##              Prevalence : 0.059
##          Detection Rate : 0.000
##    Detection Prevalence : 0.000
##       Balanced Accuracy : 0.500
##
##        'Positive' Class : Yes
##
```

```
svm.poly.err = mean(pred != ytest)
```

(c)

Repeat (a) using a support vector machine with a radial kernel with both γ and cost parameter chosen optimally.

```
tune.out = tune(svm, Purchase~., data = caravan[train,], kernel = "radial",
                ranges = list(cost = c(0.001,0.01,0.1,1,5,10,100),
                gamma = c(0.1,0.5,1,2,4)))
# summary(tune.out) # long output
bestmod = tune.out$best.model
pred = predict(bestmod, xtest)
caret::confusionMatrix(pred,ytest, positive = 'Yes')
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  No Yes
##        No  941  59
##        Yes   0   0
##
##                Accuracy : 0.941
##                  95% CI : (0.9246, 0.9548)
##     No Information Rate : 0.941
##     P-Value [Acc > NIR] : 0.5346
##
##                   Kappa : 0
##  Mcnemar's Test P-Value : 4.321e-14
##
##             Sensitivity : 0.000
##             Specificity : 1.000
##          Pos Pred Value :   NaN
##          Neg Pred Value : 0.941
##              Prevalence : 0.059
##          Detection Rate : 0.000
##    Detection Prevalence : 0.000
##       Balanced Accuracy : 0.500
##
##        'Positive' Class : Yes
##
```

```
svm.rad.err = mean(pred != ytest)
```

(d)

Compare results from the above three methods and also from the method you recommended for these data in Mini Project 6. Which method would you recommend now?

Our KNN with K=9 gave us 0.05

```
## [1] "Misclassification rates are"
```

```
## [1] "SVM linear 0.059"
```

```
## [1] "SVM polynomial 0.059"
```

```
## [1] "SVM radial 0.059"
```

Question 2

Consider the business school admission data from Mini Project 3. Split the data into training and test sets just as in that project

```
admission = read.csv("admission.csv", header = T)
admission$Group <- as.factor(admission$Group)
# make test set of the last 5 obs in each Group's factor level
library(dplyr)
test <- admission %>%
  group_by(Group) %>% do(tail(.,5))
train = anti_join(admission, test)
```

(a)

Fit a support vector classifier to the training data with cost parameter chosen optimally using 10-fold cross-validation. Evaluate its performance on the test data. Summarize your results

```
tune.out = tune(svm, Group~., data = train, scale = T, kernel = "linear",
                ranges = list(cost = c(0.001,0.01,0.1,1,5,10)))
summary(tune.out)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##     1
##
## - best performance: 0.04285714
##
## - Detailed performance results:
##    cost      error dispersion
## 1 1e-03 0.62857143 0.12046772
## 2 1e-02 0.40000000 0.16218463
## 3 1e-01 0.05714286 0.09988656
## 4 1e+00 0.04285714 0.09642122
## 5 5e+00 0.05714286 0.09988656
## 6 1e+01 0.05714286 0.09988656
```

```
bestmod = tune.out$best.model
pred = predict(bestmod, test)
caret::confusionMatrix(pred,test$Group)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction 1 2 3
##          1 4 0 0
##          2 0 3 0
##          3 1 2 5
##
## Overall Statistics
##
##                Accuracy : 0.8
##                  95% CI : (0.5191, 0.9567)
##     No Information Rate : 0.3333
##     P-Value [Acc > NIR] : 0.0002851
##
##                   Kappa : 0.7
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 1 Class: 2 Class: 3
## Sensitivity            0.8000   0.6000   1.0000
## Specificity            1.0000   1.0000   0.7000
## Pos Pred Value         1.0000   1.0000   0.6250
## Neg Pred Value         0.9091   0.8333   1.0000
## Prevalence             0.3333   0.3333   0.3333
## Detection Rate         0.2667   0.2000   0.3333
## Detection Prevalence   0.2667   0.2000   0.5333
## Balanced Accuracy      0.9000   0.8000   0.8500
```

```
svm.lin.err = mean(pred != test$Group)
```

(b)

Repeat (a) using a support vector machine with a polynomial kernel of degree two.

```
tune.out = tune(svm, Group~., data = train, kernel = "polynomial", degree = 2,
                ranges = list(cost = c(0.001,0.01,0.1,1,5,10,100)))
summary(tune.out)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##   100
##
## - best performance: 0.3142857
##
## - Detailed performance results:
##    cost     error dispersion
## 1 1e-03 0.6857143  0.1756104
## 2 1e-02 0.6857143  0.1756104
## 3 1e-01 0.6428571  0.1934295
## 4 1e+00 0.4285714  0.1904762
## 5 5e+00 0.3428571  0.1676840
## 6 1e+01 0.3428571  0.1380131
## 7 1e+02 0.3142857  0.1621846
```

```
bestmod = tune.out$best.model
pred = predict(bestmod, test)
caret::confusionMatrix(pred,test$Group)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction 1 2 3
##          1 2 5 0
##          2 2 0 0
##          3 1 0 5
##
## Overall Statistics
##
##                Accuracy : 0.4667
##                  95% CI : (0.2127, 0.7341)
##     No Information Rate : 0.3333
##     P-Value [Acc > NIR] : 0.203
##
##                   Kappa : 0.2
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 1 Class: 2 Class: 3
## Sensitivity            0.4000   0.0000   1.0000
## Specificity            0.5000   0.8000   0.9000
## Pos Pred Value         0.2857   0.0000   0.8333
## Neg Pred Value         0.6250   0.6154   1.0000
## Prevalence             0.3333   0.3333   0.3333
## Detection Rate         0.1333   0.0000   0.3333
## Detection Prevalence   0.4667   0.1333   0.4000
## Balanced Accuracy      0.4500   0.4000   0.9500
```

```
svm.poly.err = mean(pred != test$Group)
```

(c)

> **Repeat (a) using a support vector machine with a radial kernel with both γ and cost parameter chosen optimally.**

```
tune.out = tune(svm, Group~., data = train, kernel = "radial",
                ranges = list(cost = c(0.001,0.01,0.1,1,5,10,100),
                gamma = c(0.1,0.5,1,2,4)))
# summary(tune.out) # really long data frame
bestmod = tune.out$best.model
pred = predict(bestmod, test)
caret::confusionMatrix(pred,test$Group)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction 1 2 3
##          1 4 0 0
##          2 0 3 0
##          3 1 2 5
##
## Overall Statistics
##
##                Accuracy : 0.8
##                  95% CI : (0.5191, 0.9567)
##     No Information Rate : 0.3333
##     P-Value [Acc > NIR] : 0.0002851
##
##                   Kappa : 0.7
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 1 Class: 2 Class: 3
## Sensitivity            0.8000   0.6000   1.0000
## Specificity            1.0000   1.0000   0.7000
## Pos Pred Value         1.0000   1.0000   0.6250
## Neg Pred Value         0.9091   0.8333   1.0000
## Prevalence             0.3333   0.3333   0.3333
## Detection Rate         0.2667   0.2000   0.3333
## Detection Prevalence   0.2667   0.2000   0.5333
## Balanced Accuracy      0.9000   0.8000   0.8500
```

```
svm.rad.err = mean(pred != test$Group)
```

(d)

Compare results from the above three methods and also the method you recommended for these data in Mini Project 3. Which method would you recommend now? For this method, display the data with decision boundary superimposed.

LDA got 0.2
QDA got 0.13
I would still go with LDA due to its simplicity. It additionally has more interpretability than SVMs.

```
print(paste("Misclassification rates are"))
```

```
## [1] "Misclassification rates are"
```

```
print(paste("SVM linear", svm.lin.err))
```

```
## [1] "SVM linear 0.2"
```

```
print(paste("SVM polynomial", svm.poly.err))
```

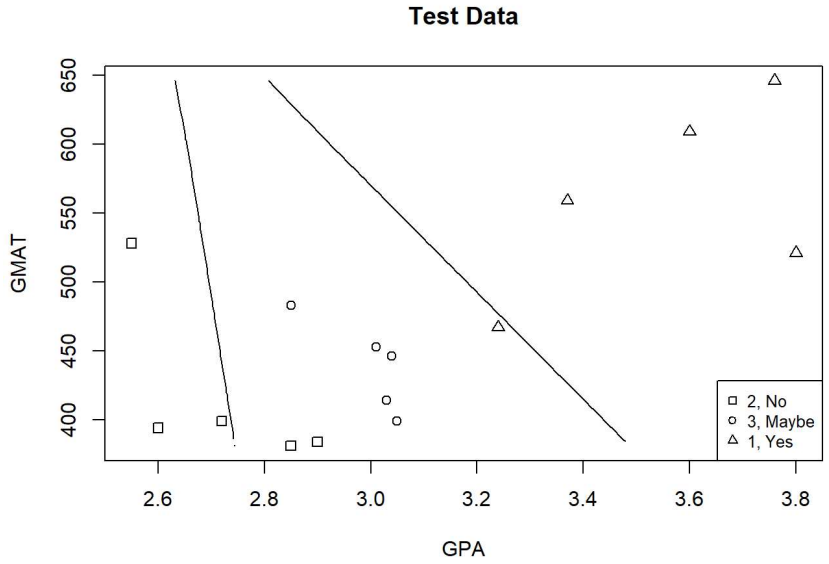
```
## [1] "SVM polynomial 0.533333333333333"
```

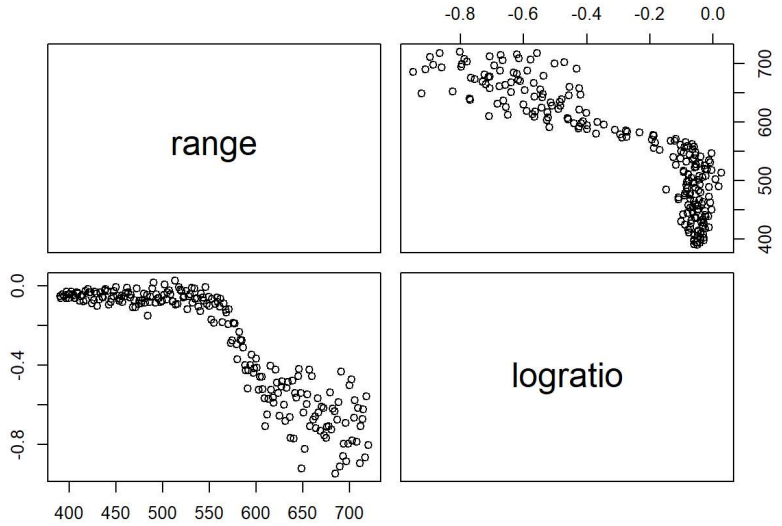
```
print(paste("SVM radial", svm.rad.err))
```

```
## [1] "SVM radial 0.2"
```

```
# now we can perform LDA and superimpose the decision boundary!
library(MASS)
lda.fit <- lda(Group ~ GPA + GMAT, data = train)
lda.pred <- predict(lda.fit, test)
n.grid <- 50
x1.grid <- seq(f = min(test[, 1]), t = max(test[, 1]), l = n.grid)
x2.grid <- seq(f = min(test[, 2]), t = max(test[, 2]), l = n.grid)
grid <- expand.grid(x1.grid, x2.grid)
colnames(grid) <- colnames(test[,1:2])
pred.grid <- predict(lda.fit, grid)

model <- lda(Group ~ GPA+GMAT, data=train)
# simple and pretty version for graphing
# plot on test set with decision boundaries
prob1 <- matrix(pred.grid$posterior[, 1],
                nrow = n.grid, ncol = n.grid, byrow = F)
prob2 <- matrix(pred.grid$posterior[, 2],
                nrow = n.grid, ncol = n.grid, byrow = F)
plot(test[,1:2], pch = ifelse(test$Group != 1,
                              ifelse(test$Group == 2, 0, 1), 2))
contour(x1.grid, x2.grid, prob1, levels = 0.5,
        labels = "", xlab = "", ylab = "", main = "", add = T)
contour(x1.grid, x2.grid, prob2, levels = 0.5,
        labels = "", xlab = "", ylab = "", main = "", add = T)
legend("bottomright", legend=c("2, No", "3, Maybe", "1, Yes"),
       pch=c(0, 1, 2), cex=0.8, bg="transparent")
title("Test Data")
```





Question 3

Consider the lidar data available in lidar.csv file on eLearning. This dataset has 221 observations from a light detection and ranging (LIDAR) experiment on two variables — range, indicating distance travelled before the light is reflected back to its source, and logratio, indicating logarithm of the ratio of received light from two laser sources. Here logratio serves as response and range serves as predictor.

```
lidar = read.csv("lidar.csv", header = T)
```

(a)

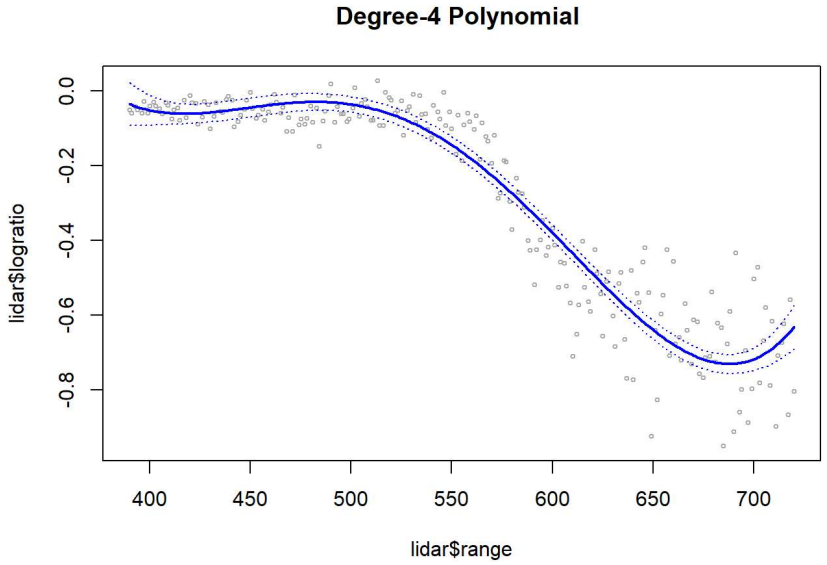
Draw a scatterplot of the data. Comment on the relationship between the two variables.

```
pairs(lidar)
```

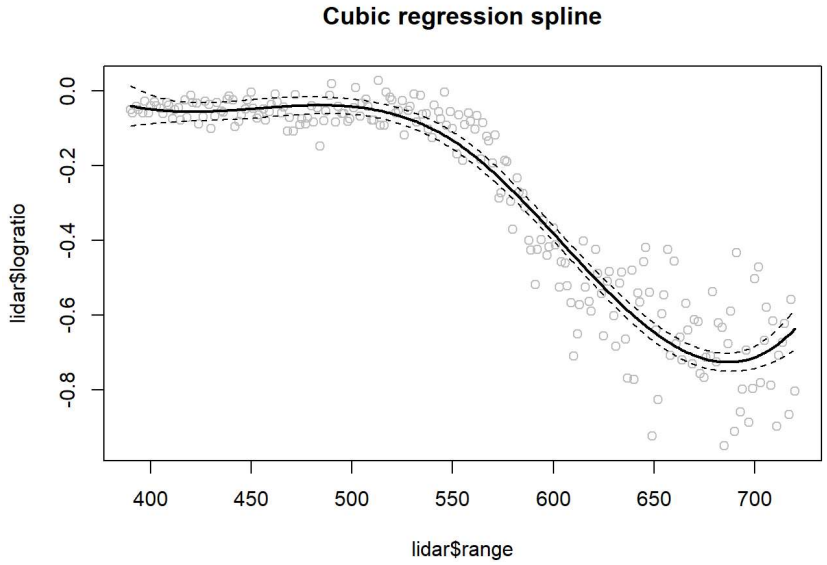
(b)

Fit a polynomial of degree 4 to the data. Superimpose the resulting fit on the scatterplot. Comment on the appropriateness of the fit.

```
fit = lm(logratio~poly(range, 4), data = lidar)
rangelims =range(lidar$range)
range.grid=seq (from=rangelims [1], to=rangelims [2])
pred = predict(fit, newdata = list(range=range.grid),se=TRUE)
lidar.se.bands=cbind(pred$fit+2*pred$se.fit, pred$fit-2*pred$se.fit)
plot(lidar$range, lidar$logratio, xlim=rangelims, cex =.5, col =" darkgrey ")
title ("Degree-4 Polynomial ")
lines(range.grid, pred$fit ,lwd =2, col =" blue")
matlines (range.grid, lidar.se.bands, lwd =1, col =" blue",lty =3)
```




```
poly.pred = pred
```




```
cubic.pred = pred
```

(c)

Repeat (b) by using a cubic regression spline with 4 degrees of freedom. How were the knots chosen? Report the knot locations.

```
library(splines)
# knots chosen by quantiles, here we have just 1
attr(bs(lidar$range, df = 4), "knots")
```

```
## 50%
## 555
```

```
cubic=lm(logratio~bs(range, df = 4), data=lidar)
pred=predict(cubic, newdata =list(range=range.grid),se=T)
plot(lidar$range, lidar$logratio, col =" gray ")
title ("Cubic regression spline")
lines(range.grid ,pred$fit ,lwd =2)
lines(range.grid, pred$fit +2* pred$se ,lty ="dashed")
lines(range.grid ,pred$fit -2* pred$se ,lty ="dashed")
```

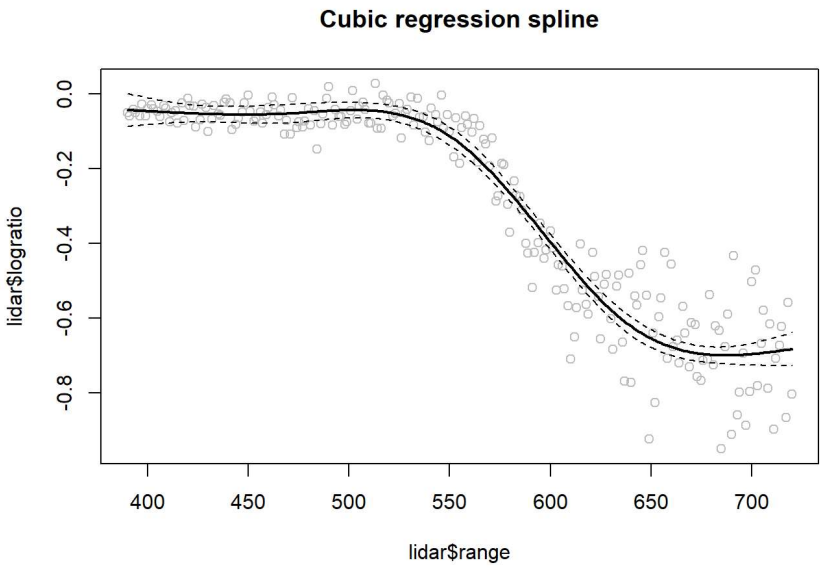
(d)

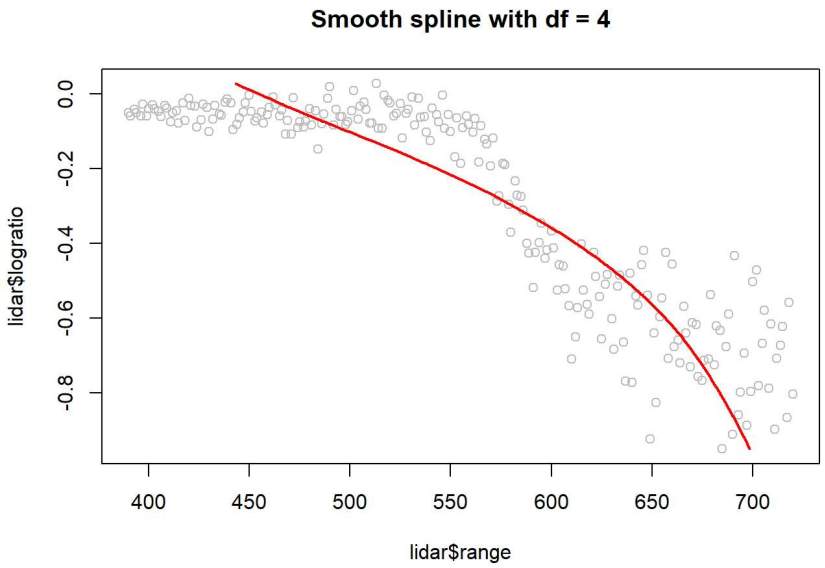
Repeat (b) by using a natural cubic regression spline with 4 degrees of freedom. How were the knots chosen? Report the knot locations.

```
# knots chosen by quantiles, here we have 3
attr(ns(lidar$range, df = 4), "knots")
```

```
## 25% 50% 75%
## 472 555 637
```

```
natural=lm(logratio~ns(range, df = 4), data=lidar)
pred=predict(natural, newdata =list(range=range.grid),se=T)
plot(lidar$range, lidar$logratio, col =" gray ")
title ("Cubic regression spline")
lines(range.grid ,pred$fit ,lwd =2)
lines(range.grid, pred$fit +2* pred$se ,lty ="dashed")
lines(range.grid ,pred$fit -2* pred$se ,lty ="dashed")
```






```
natural.pred = pred
```

(e)

Repeat (b) by using a smoothing spline with 4 effective degrees of freedom. How were the knots chosen? Report the knot locations.

```
# knots are somewhere but we don't define it for smoothing
smooth = smooth.spline(lidar$logratio, lidar$range, df=4)
plot(lidar$range, lidar$logratio, col ="gray")
title ("Smooth spline with df = 4")
lines(smooth$y, smooth$x, col ="red", lwd = 2)
```

(f)

(f) Compare the four fits. Which method would you recommend? For this method, additionally provide a scatterplot of the data superimposed with the fit and the associated approximate 95% confidence interval. Comment on the result.

It appears that the Natural spline fits better than the others, it handles the right side boundary better as expected.

```
plot(lidar$range, lidar$logratio, col ="gray")

# Poly fit
lines(range.grid, poly.pred$fit ,lwd =1, col =" blue", lty = 3)
# matlines (range.grid, lidar.se.bands, lwd =1, col =" blue",lty =3)

# cubic fit
lines(range.grid ,cubic.pred$fit ,lwd =1, col = "darkgreen", lty = 2)
# lines(range.grid, cubic.pred$fit +2* cubic.pred$se ,lty ="dashed")
# lines(range.grid ,cubic.pred$fit -2* cubic.pred$se ,lty ="dashed")

# natural spline
lines(range.grid ,natural.pred$fit ,lwd =1, col = "purple", lty = 1)
lines(range.grid, natural.pred$fit +2* natural.pred$se ,col = "purple", lty ="dashed")
lines(range.grid ,natural.pred$fit -2* natural.pred$se ,col = "purple", lty ="dashed")

# smooth spline
lines(smooth$y, smooth$x, col ="red", lwd = 1)

legend("bottomleft", legend = c("Poly", "Cubic", "Natrual", "Smooth"),
       col = c("blue", "darkgreen", "purple", "red"),
       lty = c(3, 2, 1, 1))
```