Compulsory exercise 2: Group 28

TMA4268 Statistical Learning V2020

Solfrid Hagen Johansen

02 April, 2020

Problem 1

a)

The quantity to be minimized for Rigde regression is

$$Ridge = RSS + \lambda \sum_{j=1}^{p} \beta_j^2 = \sum_{i=1}^{n} \left(y_i - \hat{y}_i\right)^2 + \lambda \sum_{j=1}^{p} \beta_j^2 = \sum_{i}^{n} \left(y_i - \boldsymbol{x}_i^T \boldsymbol{\beta}\right) + \lambda \sum_{j=1}^{p} \beta_j^2$$

Using that $\boldsymbol{Y} = \boldsymbol{X\beta} + \boldsymbol{\varepsilon}$ and $\sum \beta_j^2 = \boldsymbol{\beta}^T \boldsymbol{\beta}$, we can write this as

$$Ridge = (\boldsymbol{Y} - \boldsymbol{X}\hat{\boldsymbol{\beta}})^T (\boldsymbol{Y} - \boldsymbol{X}\hat{\boldsymbol{\beta}}) + \lambda \boldsymbol{\beta}^T \boldsymbol{\beta}$$

To minimize this to obtain the estimator, we take the derivative of the expression above with respect to $\boldsymbol{\beta}$. First, finding the derivative of the residual sum of squares term. Writing out the terms gives (omitting writing in bold and hat to save time)

$$RSS = (\boldsymbol{Y} - \boldsymbol{X}\hat{\boldsymbol{\beta}})^T (\boldsymbol{Y} - \boldsymbol{X}\hat{\boldsymbol{\beta}}) = Y^T Y - Y^T X\beta - (X\beta)^T Y + (X\beta)^T X\beta = Y^T Y - Y^T X\beta - \beta^T X^T Y + \beta^T X^T \beta X$$

Using that $\beta^T X^T Y = Y^T X \beta$, so that

$$RSS = Y^T Y - 2\beta^T X^T Y + \beta^T X^T X \beta$$

The derivative of the first term is zero, as it is independent of β. In general for a vector D_i we have

$$\frac{\partial}{\partial \beta}(\beta^T D) = \frac{\partial}{\partial \beta}\Big(\sum_i D_i \beta_i\Big) = D$$

so that

$$\frac{\partial}{\partial \beta}(2\beta^T X^T Y) = 2X^T Y.$$

For terms on the form $\beta^T D \beta$, we get

$$\frac{\partial}{\partial \beta}(\beta^T D \beta) = \frac{\partial}{\partial \beta}\Big(\sum_j^p \sum_i^p \beta_j d_{jk} \beta_k\Big) = (D + D^T)\beta.$$

Using $X^T X = X X^T$, we have that

$$\frac{\partial}{\partial \beta}(\beta^T X^T X \beta) = 2X\beta.$$

Finally, the derivative of the RSS term is

$$-2X^TY + 2(X^TX)\beta.$$

The derivative of the λ term is

$$\frac{\partial}{\partial\beta}(\lambda\beta^T\beta) = \lambda\frac{\partial}{\partial\beta}\Big(\sum_{j=1}^{N}\beta_j^2\Big) = 2\lambda\sum_{j=1}^{N}\beta_j = 2\lambda\beta,$$

so that finally the derivative of the entire expression is set to zero,

$$-2X^TY + 2(X^TX)\beta + 2\lambda\beta = 0.$$

Solving for β gives

$$(X^TX + \lambda I)\beta = X^TY$$

$$\beta = (X^TX + \lambda I)^{-1}X^TY.$$

b)

The expectation value is

$$\begin{aligned} E(\hat{\beta}) = E((X^TX+\lambda I)^{-1}X^TY) = E((X^TX+\lambda I)^{-1}X^T(X\beta+\varepsilon)) &= E((X^TX+\lambda I)^{-1}X^TX\beta) + E((X^TX+\lambda I)^{-1}\varepsilon) \\ &= (X^TX+\lambda I)^{-1}X^TX\beta + (X^TX+\lambda I)^{-1}E(\varepsilon) \\ &= (X^TX+\lambda I)^{-1}X^TX\beta, \end{aligned}$$

where I used that $E(\varepsilon) = 0$.

The variance-covariance matrix is

$$\mathrm{Var}\big[\hat{\beta}\big] = \sigma^2\big(X^TX+\lambda I\big)^{-1}X^TX\big(X^TX+\lambda I\big)^{-1}.$$

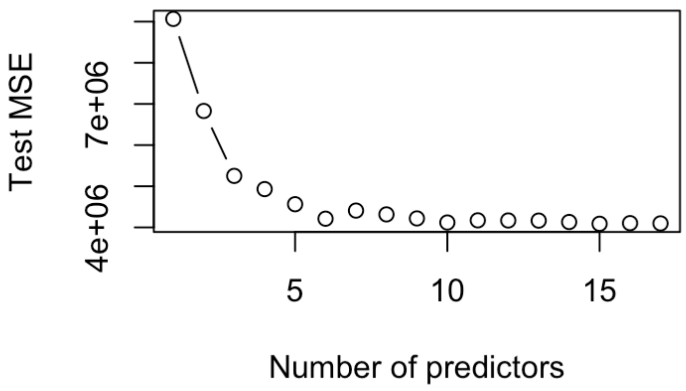
c)

Answers: True, False, False, True

d)

I will be choosing a model using 10-fold cross-validation, and choosing the model with the lowest cross-validation error. I'm choosing this method, as this will generally be more accurate than using indirect methods.

```r
library(leaps)
library(ISLR)
set.seed(1)
train.ind = sample(1:nrow(College), 0.5 * nrow(College))
college.train = College[train.ind, ]
college.test = College[-train.ind, ]

# Create a prediction function to make predictions for regsubsets with id
# predictors included
predict.regsubsets = function(object, newdata, id, ...) {
    form = as.formula(object$call[[2]])
    mat = model.matrix(form, newdata)
    coefi = coef(object, id = id)
    xvars = names(coefi)
    mat[, xvars] %*% coefi
}

# 10-fold CV
k = 10
maxNr = 17
folds = sample(1:k, nrow(College), replace = TRUE)
# To store the results
cv.errors = matrix(NA, k, maxNr, dimnames = list(NULL, paste(1:maxNr)))

# Perform cross-validation
for (j in 1:k) {
    # Fit model with all data but the part left out
    best = regsubsets(Outstate ~ ., data = College[folds != j, ], nvmax = maxNr)
    for (i in 1:maxNr) {
        # Predict for i number of predictors, for the test set for the CV
        pred = predict(best, College[folds == j, ], id = i)
        # test MSE for the ith CV for the best j variable model
        cv.errors[j, i] = mean((College$Outstate[folds == j] - pred)^2)
    }
}

# CV error, mean over the columns
mean.cv.errors = apply(cv.errors, 2, mean)

# Plot of test MSE as a function of number of predictors
plot(mean.cv.errors, type = "b", ylab = "Test MSE", xlab = "Number of predictors")
```



From the plot above, we see that the error decreases for increasing number of variables included, from 10 predictors. As the decrease is rather small, one would prefer to chose a smaller number of predictors. We also see that for 6 predictors, the error is also rather small. Thus the 6-predictor model is chosen.

```
set.seed(1)

# Fitting the model with 6 variables

reg.best = regsubsets(Outstate ~ ., data = College, nvmax = maxNr)

coef(reg.best, 6)
```

```
##   (Intercept)    PrivateYes    Room.Board      Terminal   perc.alumni
## -4127.4636290  2764.3763937     0.9282007    40.6728255    46.7490606
##        Expend     Grad.Rate
##     0.2224832    31.4546609
```

```
# Test MSE
MSE = mean.cv.errors[6]
MSE
```

```
##       6
## 4211782
```

The coefficient estimates are printed above, as well as the test MSE (4211782).

e)

The tuning parameter is selected with the built in CV (10 fold by default).

```r
set.seed(1)
library(glmnet)

test = (-train.ind)


x = model.matrix(Outstate ~ ., College)[, -1]
y = College$Outstate

grid = 10^seq(10, -2, length = 100)

lasso.mod = glmnet(x[train.ind, ], y[train.ind], alpha = 1, lambda = grid)
cv.out = cv.glmnet(x[train.ind, ], y[train.ind], alpha = 1)

bestlambda = cv.out$lambda.min

lasso.pred = predict(lasso.mod, s = bestlambda, newx = x[test, ])


# fitting with bestlamda, using entire dataset
out = glmnet(x, y, alpha = 1, lambda = grid)
lasso.coef = predict(out, type = "coefficients", s = bestlambda)[1:18, ]
lasso.coef[lasso.coef != 0]
```

```
##   (Intercept)    PrivateYes          Apps        Accept        Enroll
## -1644.4513387  2289.6634966    -0.2588852     0.7123292    -0.4079564
##     Top10perc     Top25perc   F.Undergrad    Room.Board         Books
##    22.7992777    -0.2593004    -0.1023394     0.8852958    -0.4331243
##      Personal           PhD      Terminal     S.F.Ratio   perc.alumni
##    -0.2302196    11.6855458    24.3380467   -46.9772254    41.4429087
##        Expend     Grad.Rate
##     0.1981832    23.6322860
```

```r
# Best lambda
bestlambda
```

```
## [1] 5.093201
```

```r
# Test MSE
mean((lasso.pred - y[test])^2)
```

```
## [1] 3717638
```

The coefficients are shown above. The test MSE is 3717638, which is smaller than for the forward selection case. The lambda value which was chosen using 10-fold CV was 5.093201.

Problem 2

a)

Answers: FALSE, FALSE, TRUE, TRUE

b)

For a cubic spline with three knots, we have (6 basis functions)

$$y_i = \beta_0 + \beta_1 b_1(x_i) + \beta_2 b_2(x_i) + \beta_3 b_3(x_i) + \beta_4 b_4(x_i) + \beta_5 b_5(x_i) + \beta_6 b_6(x_i),$$

where

$$\begin{aligned} b_1(x_i) &= x_i \\ b_2(x_i) &= x_i^2 \\ b_3(x_i) &= x_i^3 \end{aligned}$$

b_4, b_5 and b_6 are truncated power basis functions, which is defined as

$$h(x,\xi) = \begin{cases} (x-\xi)^3 & \text{if } x > \xi \\ 0 & \text{otherwise} \end{cases}$$

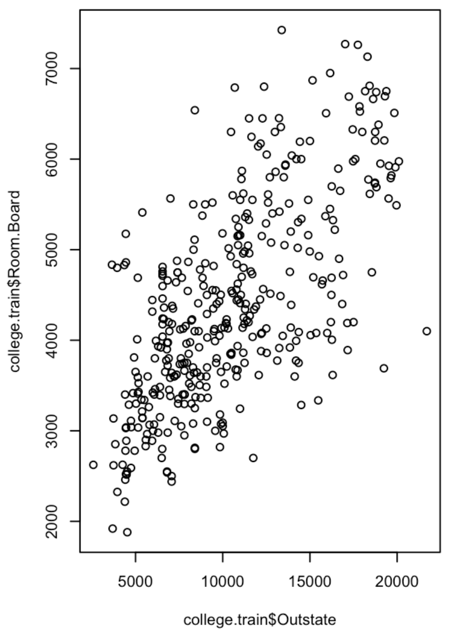
where ξ is the value of the knot (so that $x_{min} < \xi < x_{max}$).

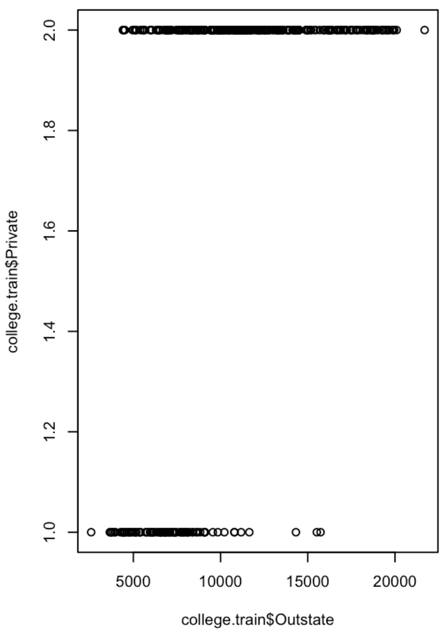
I.e.

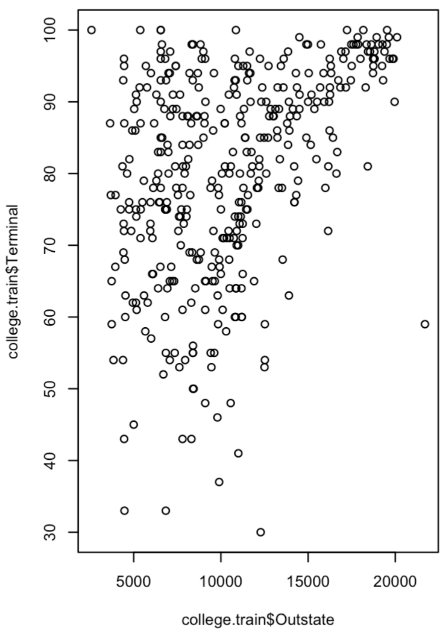
$$\begin{aligned} b_4(x_i) &= h(x_i, q_1), \\ b_5(x_i) &= h(x_i, q_2), \\ b_5(x_i) &= h(x_i, q_3). \end{aligned}$$

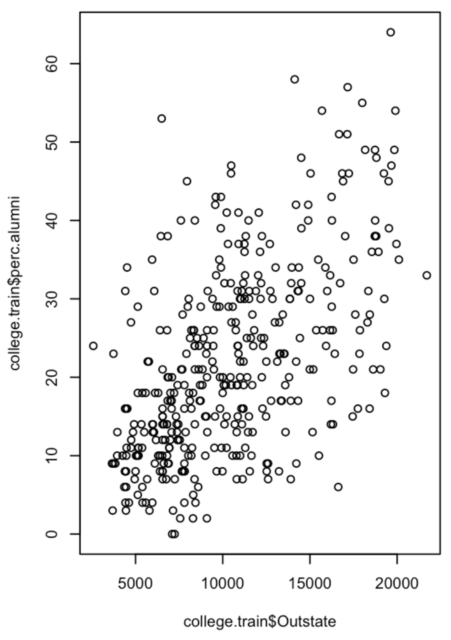
c)

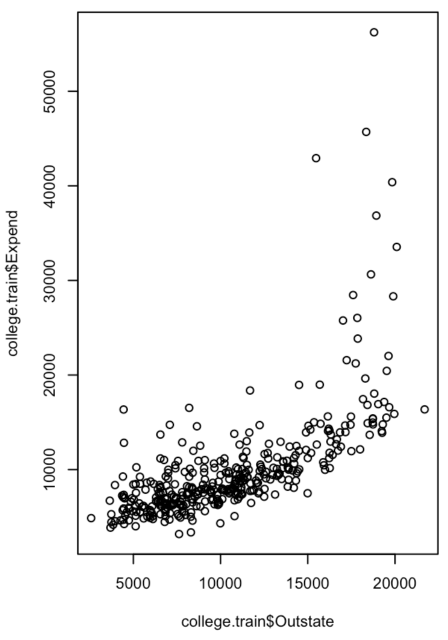
```r
par(mfrow = c(2, 3))

plot(college.train$Outstate, college.train$Room.Board)
plot(college.train$Outstate, college.train$Private)
plot(college.train$Outstate, college.train$Terminal)
plot(college.train$Outstate, college.train$perc.alumni)
plot(college.train$Outstate, college.train$Expend)
plot(college.train$Outstate, college.train$Grad.Rate)
```

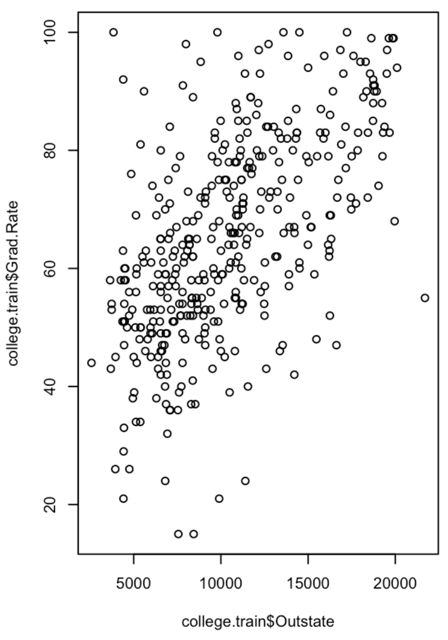












From the plot it looks like Outstate has a linear relationship with Grad.Rate. The variance also appears to be constant. This would probably not benefit from non-linear transformation.

The relationship with Room.Board looks linear, but with an increasing variance for high values. The same is the case for perc.alumni, altough this variable has a higher variance also for smaller values. Both of these would not benefit from non-linear transformation

Expend seems to have a linear relationship for up to around 20 000, where there is large variance and a sudden increase. This could benefit from non-linear tranformation, as the large values at the end will probably greatly increase the slope of the model (possibly high leverage points) so that the model is greatly impacted by these values.

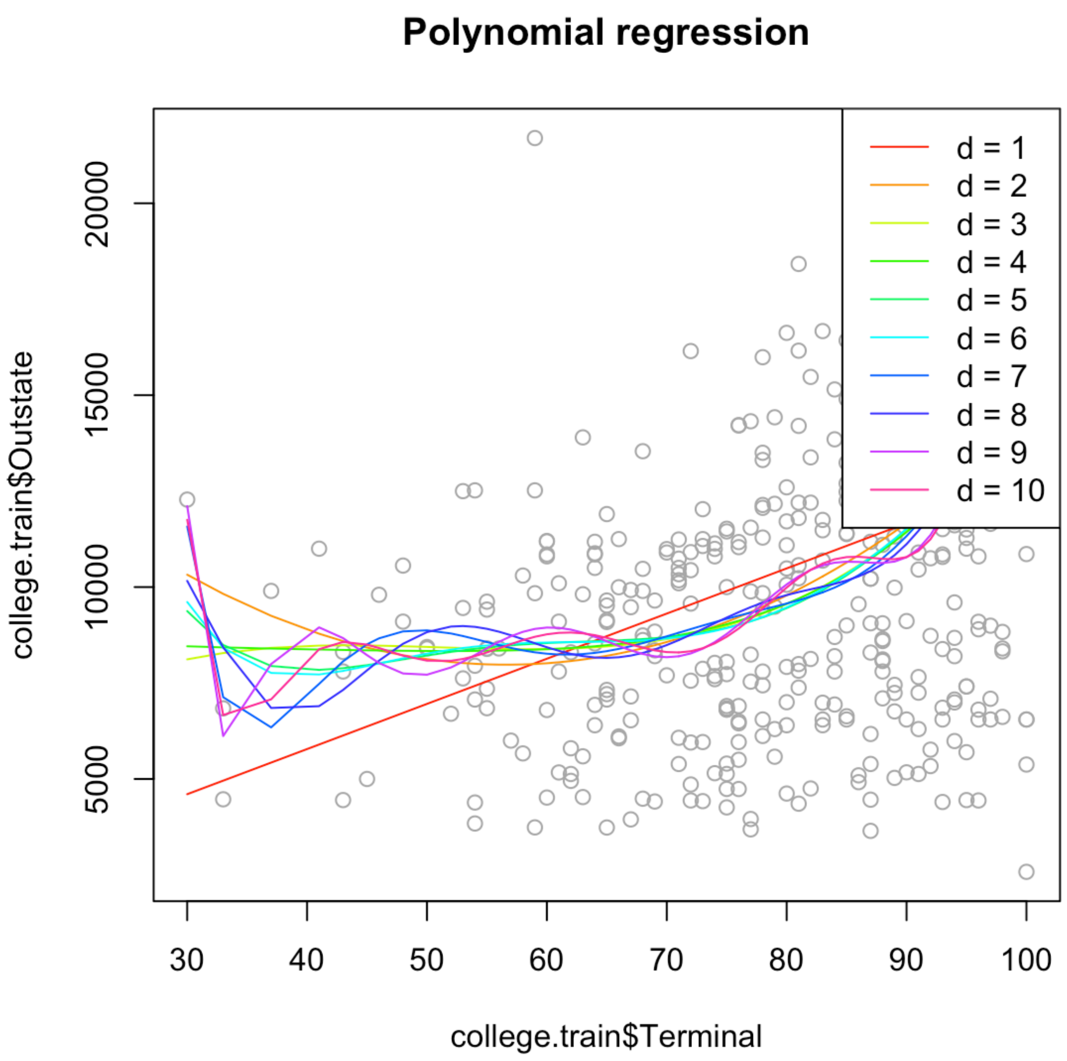
The relationship with Terminal seems to be highly non-linear, and a non-linear transformation will be needed for this.

The Outstate variable is in general higher for when qualitative variable Private is 2 (i.e. it is a private college), which indicates a linear relationship. However, as seen above, the variance for private collages is siginficanly higher than for non-private.

d)

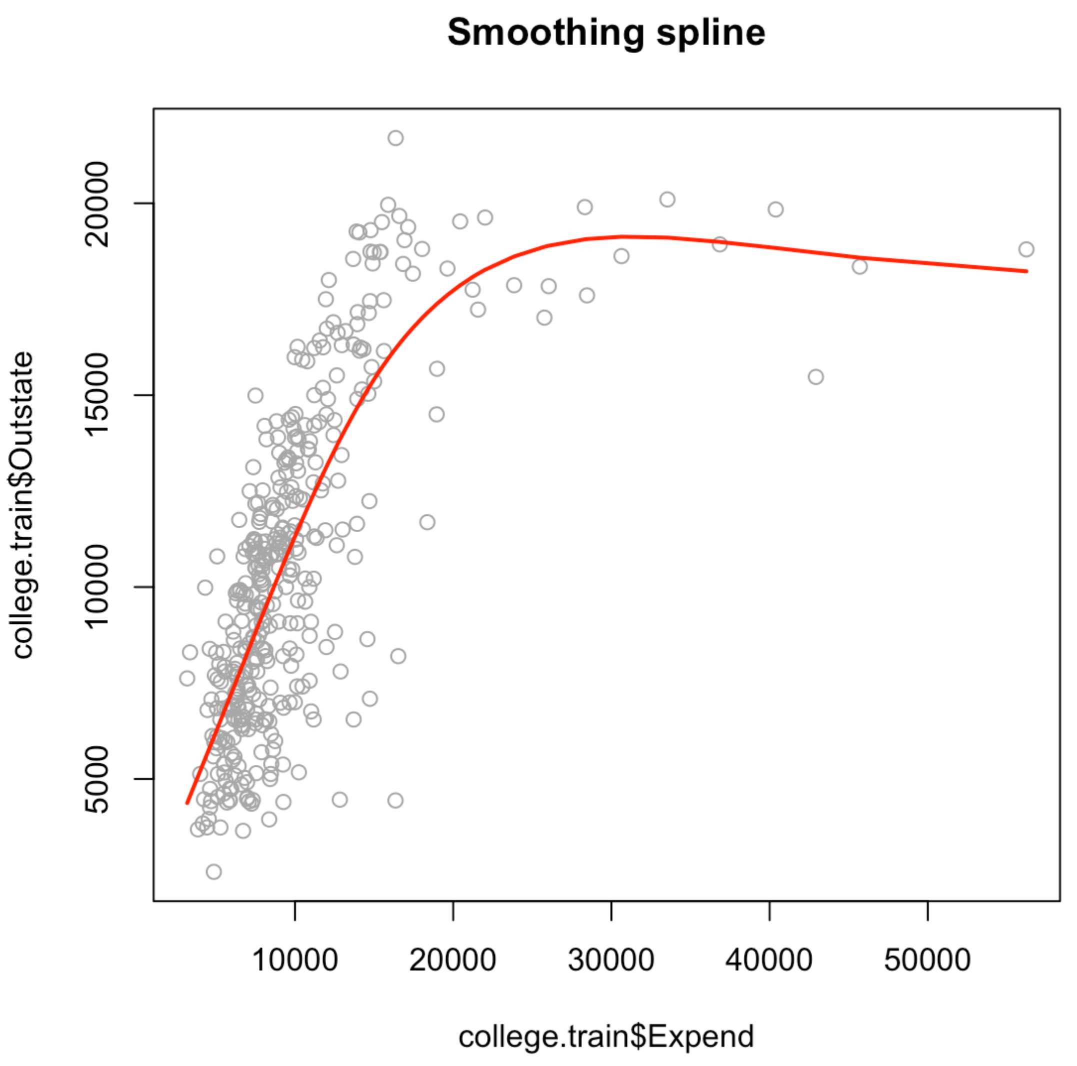
i)

```r
# Fit multiple polynomial regression models for Terminal as covariate.
set.seed(1)
deg = 1:10

# Plotting data
plot(college.train$Terminal, college.train$Outstate, col = "darkgrey", main = "Polynomial
regression")

co = rainbow(length(deg))

MSE = sapply(deg, function(d) {

    # Fitting model
    mod = lm(Outstate ~ poly(Terminal, d), data = college.train)

    # Plotting regression lines
    lines(cbind(college.train$Terminal, mod$fit)[order(college.train$Terminal), ],
        col = co[d])

    # Calculate training MSE
    mean((predict(mod, college.train) - college.train$Outstate)^2)
})

legend("topright", legend = paste("d =", deg), lty = 1, col = co)
```



ii)

```
set.seed(1)
# Choose a smoothing spline, and plot with the scatter plot
library(splines)
plot(college.train$Expend, college.train$Outstate, col = "darkgrey", main = "Smoothing sp
line")

fit = smooth.spline(college.train$Expend, college.train$Outstate, cv = FALSE)

lines(fit, col = "red", lwd = 2)
```



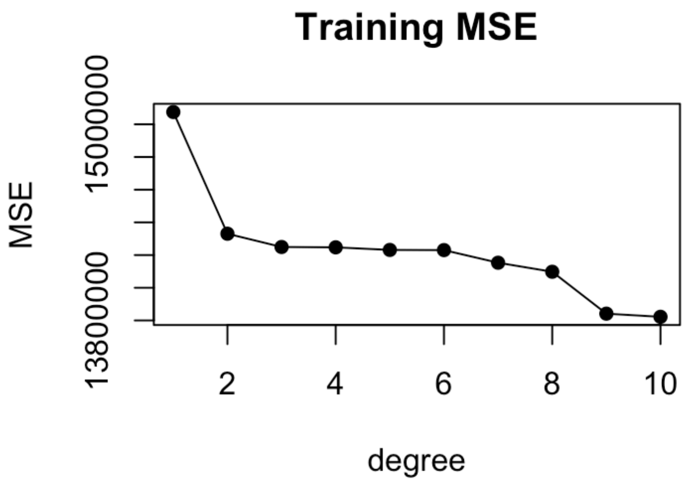
```
fit$df
```

```
## [1] 4.664187
```

Using cross-validation, λ was chosen so that the effective degrees of freedom were 4.66.

iii)

```
set.seed(1)

# Plot training MSE as a function of degree of polynomial
plot(MSE, type = "o", pch = 16, xlab = "degree", main = "Training MSE")
```



```
# Training MSE for the smoothing spline
trainMSESpline = mean((predict(fit, college.train$Expend)$y - college.train$Outstate)^2)
trainMSESpline
```

```
## [1] 6871157
```

From the plots in c) we have seen that Terminal had a much wider spread of values, and so it is expected that Expend will be able to predict Outstate better.

Due to this wider spread, the training MSE will be high, as seen in the plot above. Even though the model (for high degree in polynomial) is flexible, the data is so spread out that the model will not predict very well. As expected, for the polynomial regression the training MSE decreases with increasing degree.

In contrast, the smoothing spline for Expend as seen in d) fits the data seemingly well, and the training MSE is 6.8 million, significantly less than the lowest for Terminal (13.8 million). This is to be expected due to the data ot beeing spread out much so that a model can fit it. Also, the smoothing spline is a non-linear model, so we can fit the top values decently as well.

Problem 3

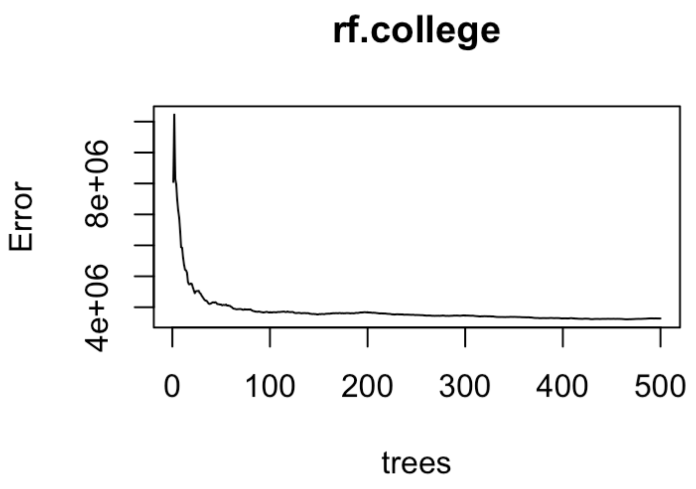
a)

Answers: FALSE, TRUE, TRUE, FALSE

b)

I chose to build a tree using random forest. A disadvantage of this is that is looses interpretability, but it generally performs better than bagging. I chose this method over boosting, due to the complication of having to select three tuning parameters. In the random forest, the parameters needed to be chosen is the size of the predictor subset, which I chose to be $m \approx \sqrt{p} = 8$ (which according to ISLR is the typical choice). The number of trees grown, B also has to be chosen, but as this is not a tuning parameter (unlike for boosting), this is selected based on the error, and observing when the error seems to converge.

```
# Build a tree using the training data, by using random forest

library(randomForest)

set.seed(1)

# Fitting tree using test set
rf.college = randomForest(Outstate ~ ., data = college.train, importance = TRUE,
    mtry = 8)
plot(rf.college)
```

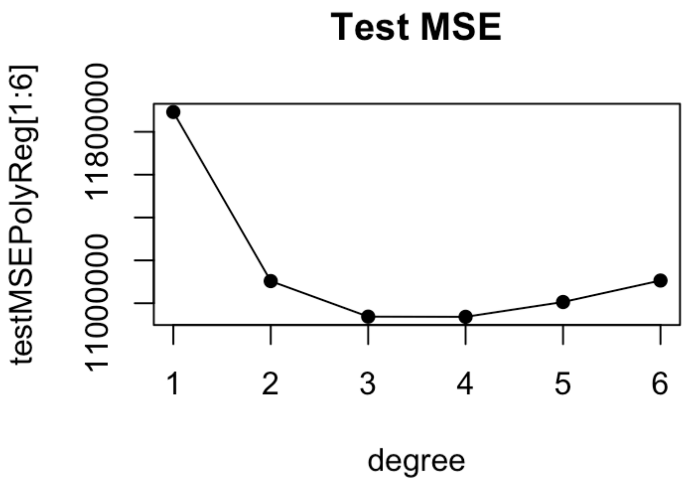


```
# Prediction values of test set
yhat.rf = predict(rf.college, newdata = college.test)

# Computing MSE, mean difference between predicted value and actual value
testError = mean((yhat.rf - college.test$Outstate)^2)
```

By default, the randomForest function uses 500 trees, but from the plot above, only about 75 trees are needed for the error to slow down decreasing.

c)

```
set.seed(1)
# Computing test MSE for polynomial regression
deg = 1:10

co = rainbow(length(deg))

testMSEPolyReg = sapply(deg, function(d) {
    # Fitting model
    mod = lm(Outstate ~ poly(Terminal, d), data = college.train)
    # Calculate training MSE
    mean((predict(mod, college.test) - college.test$Outstate)^2)
})

plot(testMSEPolyReg[1:6], type = "o", pch = 16, xlab = "degree", main = "Test MSE")
```



```
# test MSE for smoothing spline
testMSESpline = mean((predict(fit, college.test$Expend)$y - college.test$Outstate)^2)
```

Summary of error values

Forward selection: 4,2 mill

Lasso: 3,7 mill

Polynomial regression: approx. 11 mill

Smoothing spline: 5,9 mill

Random forest: approx. 4 mill

Lasso performs the best in terms of prediction error.

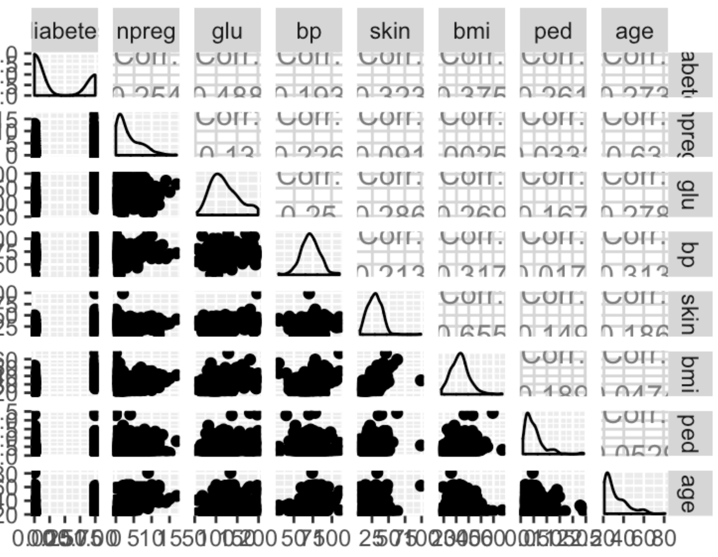
As seen in the plot above (only the lowest degrees, as the higher ones gives a large value which makes it hard to see the lowes error), which shows the test MSE for prediction Outstate from Terminal using polynomial regression, the best model is of degree 3. This gives a test MSE of 11 000 000.

The models using forward selection and Lasso (Problem 1), used *all* of the predictors to create the best model. The polynomial regression model only used ten of the predictors, so the test MSE for the polynomial regression will be higher. This makes it so that one cannot compare test MSE directly, as the one with all predictors used will usually have a lower test MSE. In addition, the ridge regression model was trained using only one predictor. Although the smoothing spline only uses one predictor, it has almost half of the MSE of polynomial regression, so clearly having more predictors doesn't make the model better in that case. It does from this seem like a lot of the variance in Outstate can be explained by the Expend predictor, as the smoothing spline MSE was not a huge amount greater than e.g. the Lasso error. So even though the lasso gave a smaller error, the smoothing spline

If the aim is to develop an interpretable model, a tree-based model is often a good choice. The random forest I used in this task is unfortunately not easily interpretable, but e.g. a tree build using boosting can be very easy to interpret. Polynomial regression models is also interpretable, as this gives an understanding of how the variables depend on eachother. However in this case the polynomial model only fit 10 of the predictors, and gave a high

Problem 4

```
id <- "1Fv6xwKLSZHldRAC1MrcK2mzdOYnbgv0E"  # google file ID
d.diabetes <- dget(sprintf("https://docs.google.com/uc?id=%s&export=download", id))
d.train = d.diabetes$ctrain
d.test = d.diabetes$ctest
set.seed(1)
library(GGally)

ggpairs(d.train)
```



```
max(d.train$npreg)
```

```
## [1] 17
```

a)

Answers: TRUE, TRUE, TRUE, TRUE

b)

Predict diabetes status

```
library(e1071)

d.train$diabetes = as.factor(d.train$diabetes)
d.test$diabetes = as.factor(d.test$diabetes)

set.seed(1)
tune.out = tune(svm, diabetes ~ ., data = d.train, kernel = "linear", 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
##   0.1
##
## - best performance: 0.2033333
##
## - Detailed performance results:
##    cost     error dispersion
## 1 1e-03 0.3333333 0.10061539
## 2 1e-02 0.2300000 0.09222892
## 3 1e-01 0.2033333 0.04830459
## 4 1e+00 0.2066667 0.05397759
## 5 5e+00 0.2100000 0.05454639
## 6 1e+01 0.2100000 0.05454639
## 7 1e+02 0.2100000 0.05454639
```

From the summary we see that a cost of 0.1 gave the lowest CV-error.

```
bestmod = tune.out$best.model
diabetespred = predict(bestmod, d.test)
table(predict = diabetespred, truth = d.test$diabetes)
```

```
##        truth
## predict   0   1
##       0 137  35
##       1  18  42
```

```
# Missclassification rate for SVC, i.e. proportion of incorrect classifications

miss = (18 + 35)/(137 + 18 + 35 + 42)
miss
```

```
## [1] 0.2284483
```

The misclassification rate is 22.84%.

```
# Support Vector Machine
set.seed(1)
tune.out.rad = tune(svm, diabetes ~ ., data = d.train, kernel = "radial", ranges = list(c
ost = c(0.1,
    0.5, 0.8, 1, 2, 10, 100, 1000), gamma = c(0.5, 1, 2, 3, 4)))


summary(tune.out.rad)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost gamma
##   0.5   0.5
##
## - best performance: 0.2533333
##
## - Detailed performance results:
##     cost gamma     error dispersion
## 1  1e-01   0.5 0.3333333 0.10061539
## 2  5e-01   0.5 0.2533333 0.05258738
## 3  8e-01   0.5 0.2566667 0.04981447
## 4  1e+00   0.5 0.2533333 0.05018484
## 5  2e+00   0.5 0.2933333 0.05621827
## 6  1e+01   0.5 0.3266667 0.04919099
## 7  1e+02   0.5 0.3233333 0.05676462
## 8  1e+03   0.5 0.3233333 0.05676462
## 9  1e-01   1.0 0.3333333 0.10061539
## 10 5e-01   1.0 0.3333333 0.10061539
## 11 8e-01   1.0 0.3200000 0.06885304
## 12 1e+00   1.0 0.2900000 0.05454639
## 13 2e+00   1.0 0.3066667 0.04661373
## 14 1e+01   1.0 0.3100000 0.04458312
## 15 1e+02   1.0 0.3100000 0.04458312
## 16 1e+03   1.0 0.3100000 0.04458312
## 17 1e-01   2.0 0.3333333 0.10061539
## 18 5e-01   2.0 0.3333333 0.10061539
## 19 8e-01   2.0 0.3333333 0.10061539
## 20 1e+00   2.0 0.3366667 0.10237911
## 21 2e+00   2.0 0.3433333 0.08755950
## 22 1e+01   2.0 0.3433333 0.08755950
## 23 1e+02   2.0 0.3433333 0.08755950
## 24 1e+03   2.0 0.3433333 0.08755950
## 25 1e-01   3.0 0.3333333 0.10061539
## 26 5e-01   3.0 0.3333333 0.10061539
## 27 8e-01   3.0 0.3333333 0.10061539
## 28 1e+00   3.0 0.3333333 0.10061539
## 29 2e+00   3.0 0.3400000 0.09135469
## 30 1e+01   3.0 0.3400000 0.09135469
## 31 1e+02   3.0 0.3400000 0.09135469
## 32 1e+03   3.0 0.3400000 0.09135469
## 33 1e-01   4.0 0.3333333 0.10061539
## 34 5e-01   4.0 0.3333333 0.10061539
## 35 8e-01   4.0 0.3333333 0.10061539
## 36 1e+00   4.0 0.3333333 0.10061539
## 37 2e+00   4.0 0.3400000 0.09913204
## 38 1e+01   4.0 0.3400000 0.09913204
## 39 1e+02   4.0 0.3400000 0.09913204
## 40 1e+03   4.0 0.3400000 0.09913204
```

From the summary, the values which give the lowest CV-error is cost=0.5 and gamma=0.5

```r
set.seed(1)
bestmod = tune.out.rad$best.model
diabetespred = predict(bestmod, d.test)
table(predict = diabetespred, truth = d.test$diabetes)
```

```
##        truth
## predict   0   1
##       0 141  45
##       1  14  32
```

```r
miss = (14 + 45)/(141 + 45 + 14 + 32)
miss
```

```
## [1] 0.2543103
```

The misclassification error rate is 25.43%.

As the Support vector classifier gives a lower misclassification error, this is prefered. Also, this model is simpler and thus possibly easier to interpret.

c)

Comparing with linear discriminant analysis (LDA), as as seen above, it seems that a linear decision boundary outperforms a non-linear boundary. Thus it is interesting to compare with another linear model.

```r
library(MASS)
set.seed(1)
# Fitting model
lda.fit = lda(diabetes ~ ., data = d.train)

# Confusion table

lda.pred = predict(lda.fit, d.test)
lda.class = lda.pred$class
table(lda.class, d.test$diabetes)
```

```
##
## lda.class   0   1
##         0 137  34
##         1  18  43
```

```r
miss = (34 + 18)/(137 + 34 + 18 + 43)
miss
```

```
## [1] 0.2241379
```

This model has a misclassification rate of 22.41%. This is slightly lower than for the support vector classifier. LDA assumes that the observations are drawn from a Gaussian distribution, which can be a disadvantage. SVM makes no assumption on the data, which makes it more flexible. However, it can make it less interpretable in general. An advantage with LDA is that it is easy to handle cases where the classification is non-binary, while the SVM has no "natural" extension to this. The ways of adding more classes for SVM can be computationally expensive, so LDA could be prefered in that case.

d)

Answers: FALSE, FALSE, FALSE, TRUE

e)

The deviance of a binary distribution is

$$\text{deviance} = -2\log(Pr(y_i|\boldsymbol{x_i}, \boldsymbol{\beta}))$$

To compute the above expression, a formula for the probability is needed. As stated in the task, a $y_i = -1, 1$ encoding is to be used, so we cannot employ the common form of the Bernoulli distribution (which has a $y_i = 1, 0$ encoding). As the only probabilities needed to be computed is that of $y_i = -1, 1$, the requirement for such a function is that $Pr(y_i = 1) = p$ and $Pr(y_i = -1) = p$. Inspired by the form of the loss function which is the one to show that is equal to, a possible choice is

$$\Pr(y_i|\boldsymbol{x}_i, \boldsymbol{\beta}) = \frac{1}{1 + \exp(-y_i f(\boldsymbol{x}_i))}$$

as this function gives

$$\Pr(y_i = 1|\boldsymbol{x}_i, \boldsymbol{\beta}) = \frac{1}{1 + \exp(-f(\boldsymbol{x}_i))} = \frac{\exp(f(\boldsymbol{x_i}))}{1 + \exp(f(\boldsymbol{x_i}))} = p$$

and

$$\Pr(y_i = -1|\boldsymbol{x}_i, \boldsymbol{\beta}) = \frac{1}{1 + \exp(f(\boldsymbol{x}_i))} = \frac{\exp(-f(\boldsymbol{x_i}))}{1 + \exp(-f(\boldsymbol{x_i}))} = 1 - p$$

By now inserting the expression into the deviance equation, we get

$$\text{deviance} = -2\log\left(\frac{1}{1 + \exp(-y_i f(\boldsymbol{x}_i))}\right) \sim \log(1 + \exp(-y_i f(\boldsymbol{x}_i))),$$

which is equal to the expression that was to be found.

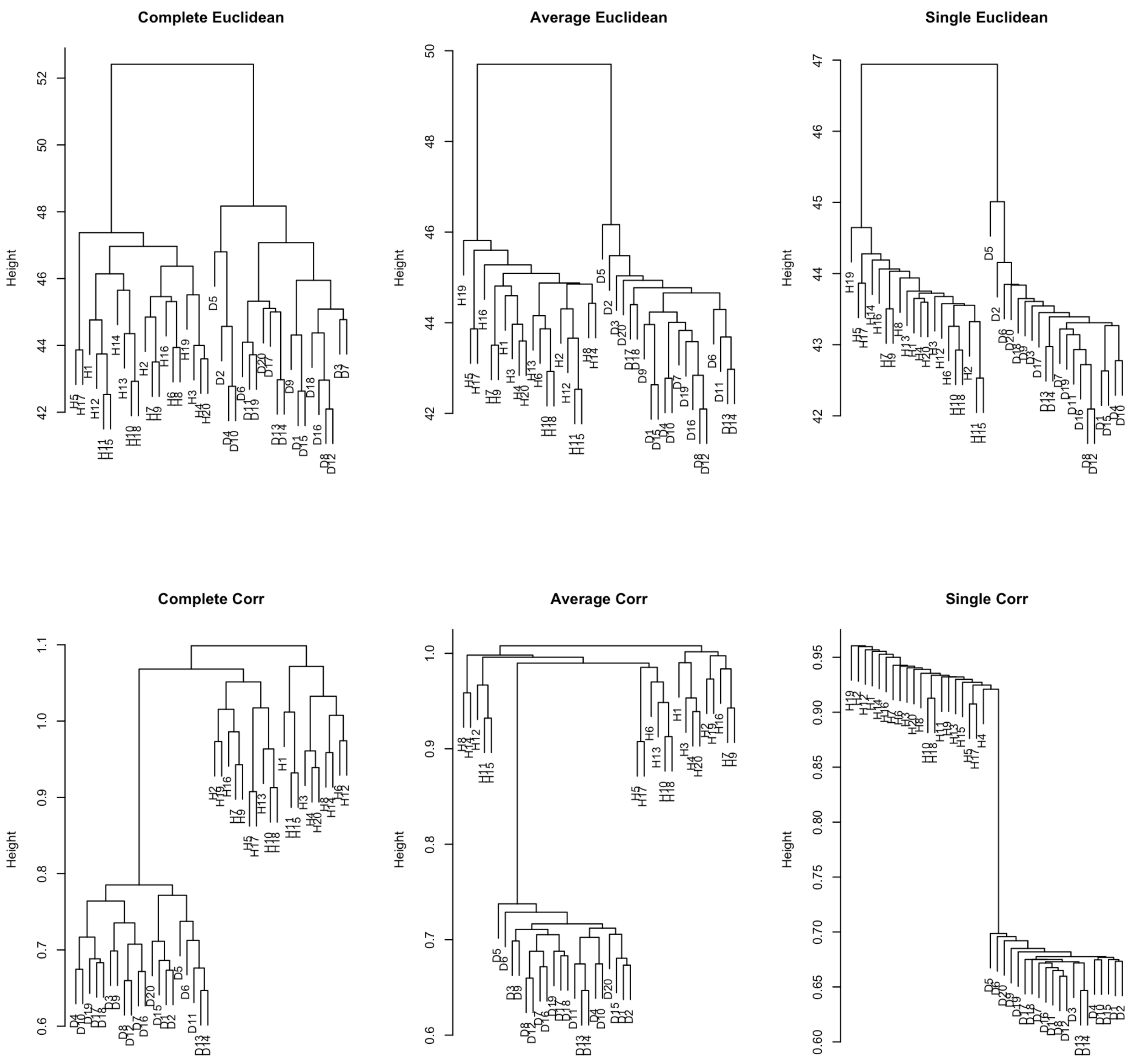
Problem 5

a)

```
id <- "1VfVCQvWt121UN39NXZ4aR9Dmsbj-p9OU"  # google file ID
GeneData <- read.csv(sprintf("https://docs.google.com/uc?id=%s&export=download",
    id), header = F)
colnames(GeneData)[1:20] = paste(rep("H", 20), c(1:20), sep = "")
colnames(GeneData)[21:40] = paste(rep("D", 20), c(1:20), sep = "")

row.names(GeneData) = paste(rep("G", 1000), c(1:1000), sep = "")
```

```r
# Transforming matrix
GeneData = t(GeneData)
```

```r
set.seed(1)
par(mfrow = c(2, 3))
# Euclidean distance
hc.complete = hclust(dist(GeneData), method = "complete")
hc.average = hclust(dist(GeneData), method = "average")
hc.single = hclust(dist(GeneData), method = "single")

plot(hc.complete, main = "Complete Euclidean", xlab = "", sub = "", cex = 0.9)
plot(hc.average, main = "Average Euclidean", xlab = "", sub = "", cex = 0.9)
plot(hc.single, main = "Single Euclidean", xlab = "", sub = "", cex = 0.9)

dd = as.dist(1 - cor(t(GeneData)))
hc.corr.complete = hclust(dd, method = "complete")
hc.corr.average = hclust(dd, method = "average")
hc.corr.single = hclust(dd, method = "single")

plot(hc.corr.complete, main = "Complete Corr", xlab = "", sub = "", cex = 0.9)
plot(hc.corr.average, main = "Average Corr", xlab = "", sub = "", cex = 0.9)
plot(hc.corr.single, main = "Single Corr", xlab = "", sub = "", cex = 0.9)
```


Complete Euclidean
Average Euclidean
Single Euclidean
Complete Corr
Average Corr
Single Corr
Height

b)

```r
# two clusters of tissue samples
nrClusters = 2
set.seed(1)
# Splits into two clusters, 1 and 2, for Euclidean distance
completeE = cutree(hc.complete, 2)
averageE = cutree(hc.average, 2)
singleE = cutree(hc.single, 2)

# Correlation-based distance
completeC = cutree(hc.corr.complete, 2)
averageC = cutree(hc.corr.average, 2)
singleC = cutree(hc.corr.single, 2)

# Comparing with true state, performs best i.e. lowest missclassification rate
# Know that the first 20 samples are healthy, last 20 are unhealthy
true = c(rep(1, 20), rep(2, 20))
completeEMissClass = mean(completeE != true)
averageEMissClass = mean(averageE != true)
singleEMissClass = mean(singleE != true)

completeCMissClass = mean(completeC != true)
averageCMissClass = mean(averageC != true)
singleCMissClass = mean(singleC != true)

c(completeEMissClass, averageEMissClass, singleEMissClass, completeCMissClass, averageCMi
ssClass,
    singleCMissClass)
```

```
## [1] 0.000 0.000 0.000 0.250 0.275 0.525
```

We see that all of the Euclidean distance linkages gives zero missclassifications, and thus they perform equally well.

c)

ϕ_{j1} are the loadings, which determine how large each term in the linear combination is (normalized to one). Each feature has a (possible zero) loading, so there are p of these.

p is the number of features.

n are the number of observations.

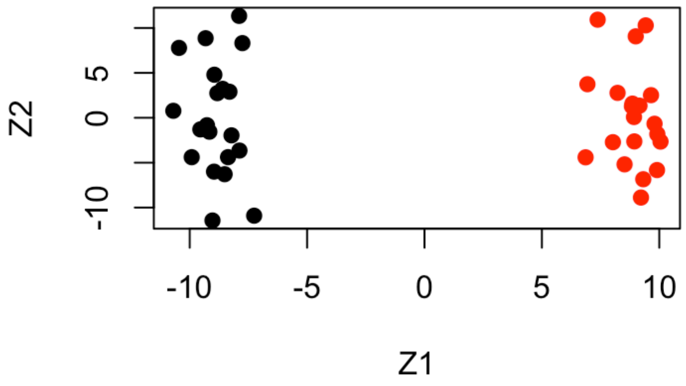
x_{ij} is the value of the ith observation and jth feature.

The formula for the first principal component scores is

$$z_{i1} = \phi_{11}x_{i1} + \phi_{21}x_{i2} + \ldots + \phi_{p1}x_{ip}.$$

d)

```
# Performing PCA, and scaling the data (also centers variables by default)
pr.out = prcomp(GeneData, scale = TRUE)
set.seed(1)
true = c(rep(1, 20), rep(2, 20))

plot(pr.out$x[, 1:2], col = true, pch = 19, xlab = "Z1", ylab = "Z2")
```



```
# Could also show for more using different principle axis, but chose not to do
# this here
```

```
# Finding how much of the variance that is explained by the 5 first PCs, i.e. the
# cummulative proportion of variance explained by each principal component

# Standard deviations squares
pr.var = pr.out$sdev^2

# PVE
pve = pr.var/sum(pr.var)

# Cummulative PVE

cum_pve = cumsum(pve[1:5])
cum_pve[5]
```

```
## [1] 0.2109659
```

The variance explained by the first 5 principle components is 0.2109659, i.e. 21.1%.

e)

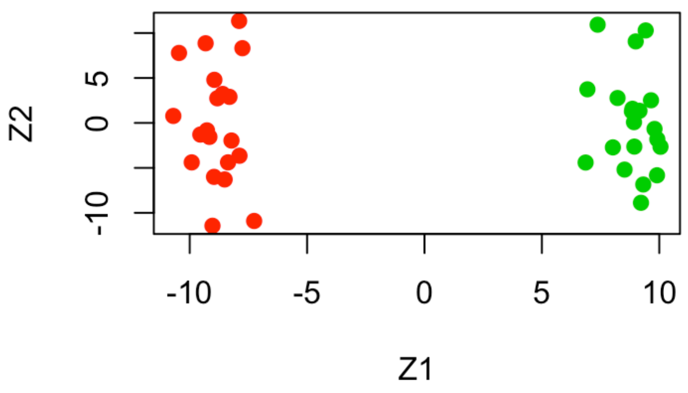
The genes that vary the most will have the largest components in the first principal compent. Thus, the genes listed below are the 3 genes which varies the most.

```
genes = pr.out$rotation[, 1]

ordered_genes = sort(abs(genes), decreasing = TRUE)
ordered_genes[1:3]
```

```
##        G502       G589       G565
## 0.09485044 0.09449766 0.09183823
```

f)

```
# k-means with two clusters, choosing 20 random sets
set.seed(3)
km.out = kmeans(GeneData, 2, nstart = 20)

# Plotting the values in two-dim space using PCA
plot(pr.out$x[, 1:2], col = (km.out$cluster + 1), pch = 19, xlab = "Z1", ylab = "Z2")
```



In the plot above, the tissue samples are split into two groups.

```
# Error rate

true = c(rep(1, 20), rep(2, 20))
kmeansError = mean(km.out$cluster != true)
kmeansError
```

```
## [1] 0
```

The error rate of K-means is zero.