Compulsory exercise 2: Group 24

TMA4268 Statistical Learning V2019

Silje Anfindsen and Clara Panchaud

30 March, 2020

Problem 1

a)

Let's find the ridge regression estimator. Remember that $\hat{\beta}_{Ridge}$ minimizes $RSS + \lambda \sum_{j=1}^{p} \beta_j^2$. Here we assume that all covariates and the response have been mean-centered, so that $\beta_0 = 0$. Let's rewrite this in matrix notation.

$$
\begin{aligned}
min_\beta\{(y - X\beta)^T(y - X\beta) + \lambda\beta^T\beta\} = & \quad \text{develop the expression} \\
min_\beta\{y^Ty - 2\beta^TX^Ty + \beta^TX^TX\beta + \lambda\beta^T\beta\} & \quad \text{take the derivative with respect to beta and set equal to 0} \\
-2X^Ty + 2X^TX\beta + 2\lambda\beta = 0 & \\
(X^TX + 2\lambda I)\beta = X^Ty & \\
\beta = (X^TX + \lambda I)^{-1}X^Ty &
\end{aligned}
$$

Therefore the estimator is $\hat{\beta}_{Ridge} = (X^TX + \lambda I)^{-1}X^Ty$.

b)

To find the expected value and the variance-covariance matrix of $\hat{\beta}_{Ridge}$ we need to remember the distribution of y, $y \sim N(X\beta, \sigma^2 I)$. Therefore we get the expected value:

$$E(\hat{\beta}_{Ridge}) = E((X^TX + \lambda I)^{-1}X^Ty) = (X^TX + \lambda I)^{-1}X^TE(y) = (X^TX + \lambda I)^{-1}X^TX\beta$$

and the variance-covariance matrix:

$$
\begin{aligned}
Var(\hat{\beta}_{Ridge}) = Var((X^TX + \lambda I)^{-1}X^Ty) = & \quad \text{by property of the variance} \\
(X^TX + \lambda I)^{-1}X^TVar(y)((X^TX + \lambda I)^{-1}X^T)^T = & \quad \text{develop the expression} \\
\sigma^2(X^TX + \lambda I)^{-1}X^TX(X^TX + \lambda I)^{-1} &
\end{aligned}
$$

c)

TRUE, FALSE, FALSE, TRUE

d)

```
library(ISLR)
library(leaps)
library(glmnet)
```

We want to work with the *College* data. First we split it into a training and a testing set.

```
set.seed(1)

#make training and testing set
train.ind = sample(1:nrow(College), 0.5 * nrow(College))
college.train = College[train.ind, ]
college.test = College[-train.ind, ]

#the structure of the data
str(College)
```

```
## 'data.frame':    777 obs. of  18 variables:
##  $ Private    : Factor w/ 2 levels "No","Yes": 2 2 2 2 2 2 2 2 2 2 ...
##  $ Apps       : num  1660 2186 1428 417 193 ...
##  $ Accept     : num  1232 1924 1097 349 146 ...
##  $ Enroll     : num  721 512 336 137 55 158 103 489 227 172 ...
##  $ Top10perc  : num  23 16 22 60 16 38 17 37 30 21 ...
##  $ Top25perc  : num  52 29 50 89 44 62 45 68 63 44 ...
##  $ F.Undergrad: num  2885 2683 1036 510 249 ...
##  $ P.Undergrad: num  537 1227 99 63 869 ...
##  $ Outstate   : num  7440 12280 11250 12960 7560 ...
##  $ Room.Board : num  3300 6450 3750 5450 4120 ...
##  $ Books      : num  450 750 400 450 800 500 500 450 300 660 ...
##  $ Personal   : num  2200 1500 1165 875 1500 ...
##  $ PhD        : num  70 29 53 92 76 67 90 89 79 40 ...
##  $ Terminal   : num  78 30 66 97 72 73 93 100 84 41 ...
##  $ S.F.Ratio  : num  18.1 12.2 12.9 7.7 11.9 9.4 11.5 13.7 11.3 11.5 ...
##  $ perc.alumni: num  12 16 30 37 2 11 26 37 23 15 ...
##  $ Expend     : num  7041 10527 8735 19016 10922 ...
##  $ Grad.Rate  : num  60 56 54 59 15 55 63 73 80 52 ...
```

Now we will apply forward selection, using *Outstate* as a response. We have 18 variables including the response so we will obtain a model including up to 17 variables.

```
nb_predictors<-17
forward<-regsubsets(Outstate~.,college.train,nvmax=17,method="forward")
sum<-summary(forward)
```

In Figure 1 we can look at the RSS and the adjusted R^2 in order to pick the number of variables that gives the optimal result. Remember that if the difference is not very significant we would rather pick the simplest model. It seems like 5 variables would be good here.

```
par(mfrow=c(1,2))
plot(sum$rss,xlab="Number of Variables",ylab="RSS",type="l")
plot(sum$adjr2,xlab="Number of Variables",ylab="Adjusted RSq",type="l")
```

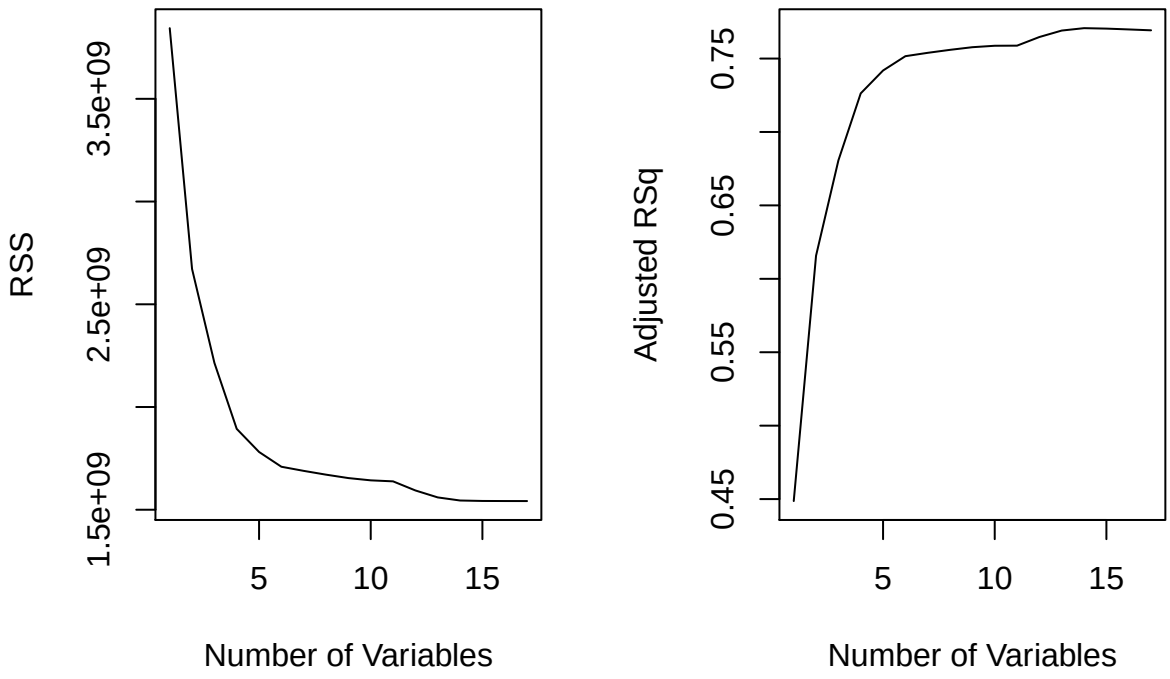


Figure 1: Comparison of models with different number of variables.

Below are the chosen variables when we decide to include 5 variables in the reduced model.

```
nb_selected_pred<-5
variables<-names( coef( forward,id=nb_selected_pred ) )
variables
```

```
## [1] "(Intercept)" "PrivateYes"  "Room.Board"  "perc.alumni" "Expend"
## [6] "Grad.Rate"
```

We will now find the reduced model as well as the MSE (mean squared error) on the test set.

```
#fit the reduced model
reduced.model<-lm(Outstate~Private+Room.Board+Grad.Rate+perc.alumni+Expend, data =college.train)
summary(reduced.model)
```

```
##
## Call:
```

```
## lm(formula = Outstate ~ Private + Room.Board + Grad.Rate + perc.alumni +
##     Expend, data = college.train)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -7293.1 -1537.5  -159.9  1286.7  9254.8
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -2.711e+03  5.155e+02  -5.259 2.41e-07 ***
## PrivateYes   2.250e+03  2.787e+02   8.075 8.92e-15 ***
## Room.Board   1.241e+00  1.205e-01  10.296  < 2e-16 ***
## Grad.Rate    3.855e+01  7.850e+00   4.910 1.35e-06 ***
## perc.alumni  6.446e+01  1.113e+01   5.792 1.45e-08 ***
## Expend       2.182e-01  2.317e-02   9.417  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2159 on 382 degrees of freedom
## Multiple R-squared:  0.7452, Adjusted R-squared:  0.7419
## F-statistic: 223.4 on 5 and 382 DF,  p-value: < 2.2e-16
```

The reduced model is

$$\begin{aligned}\textbf{Outstate} = &-2711.4329907 + 2250.1100562\textbf{Private} + 1.2410466\textbf{Room.Board} \\ &+ 38.5491289\textbf{Grad.Rate} + 64.4580901\textbf{perc.alumni} + 0.218216\textbf{Expend},\end{aligned}$$

```
#find test MSE
p<-predict(reduced.model,newdata=college.test)
mse_fwd <- mean(((college.test$Outstate)-p)^2)
mse_fwd
```

```
## [1] 4112680
```

The test MSE is

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - f(\hat{x}_i))^2 = 4.1126804 \times 10^6$$

e)

We will now select a model for the same dataset as in (d) but this time with the Lasso method. Again, we use both a training and testing set for the data.

```
#Make a x matrix and y vector for both the training and testing set
x_train<-model.matrix(Outstate~.,college.train)[,-1]
y_train<-college.train$Outstate
x_test<-model.matrix(Outstate~.,college.test)[,-1]
y_test<-college.test$Outstate
```

In order to select the best value for the tuning parameter λ we will use cross validation.

```r
set.seed(5555)

#perform the Lasso method and choose the best model using CV
lasso.mod = glmnet(x_train,y_train,alpha=1) #lasso method on train set
cv.lasso = cv.glmnet(x_train,y_train,alpha=1) #CV on train set
lambda.best = cv.lasso$lambda.min #select best lambda
lambda.best
```

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

```r
#find the test MSE
predictions<-predict(lasso.mod,s=lambda.best,newx=x_test)
mse_lasso <- mean((predictions-y_test)^2) #test MSE
mse_lasso
```

```
## [1] 3717020
```

%%Check if log scale!! From cross validation we can observe that the optimal tuning parameter is $\lambda = 5.0932009$ as this is the parameter that minimizes the MSE for the training set.

The test MSE is now 3.7170197×10^6, which is lower than what we found for the reduced model using forward selection in d).

The lasso yields sparse models which involves only a subset of variables. Lasso performs variable selection by forcing some of the coefficient estimates to be exactly zero. The selected variables that was not put to zero are displayed below.

```r
c<-coef(lasso.mod,s=lambda.best,exact=TRUE)
inds<-which(c!=0)
variables<-row.names(c)[inds]
variables
```

```
##  [1] "(Intercept)" "PrivateYes"  "Apps"         "Accept"       "Enroll"
##  [6] "Top10perc"   "Top25perc"   "F.Undergrad" "P.Undergrad" "Room.Board"
## [11] "Books"       "Personal"    "PhD"          "Terminal"    "S.F.Ratio"
## [16] "perc.alumni" "Expend"      "Grad.Rate"
```

Problem 2

a)

FALSE, FALSE, TRUE, FALSE

b)

The basis functions for a cubic spline with knots at each quartile q_1, q_2, q_3, of variable X are,

$$\begin{aligned} b_0(X) &= 1 & b_4(X, q_1) &= (X - q_1)_+^3 \\ b_1(X) &= X & b_5(X, q_2) &= (X - q_2)_+^3 \\ b_2(X) &= X^2 & b_6(X, q_3) &= (X - q_3)_+^3 \\ b_3(X) &= X^3 \end{aligned}$$

where the power basis functions are defined as below:

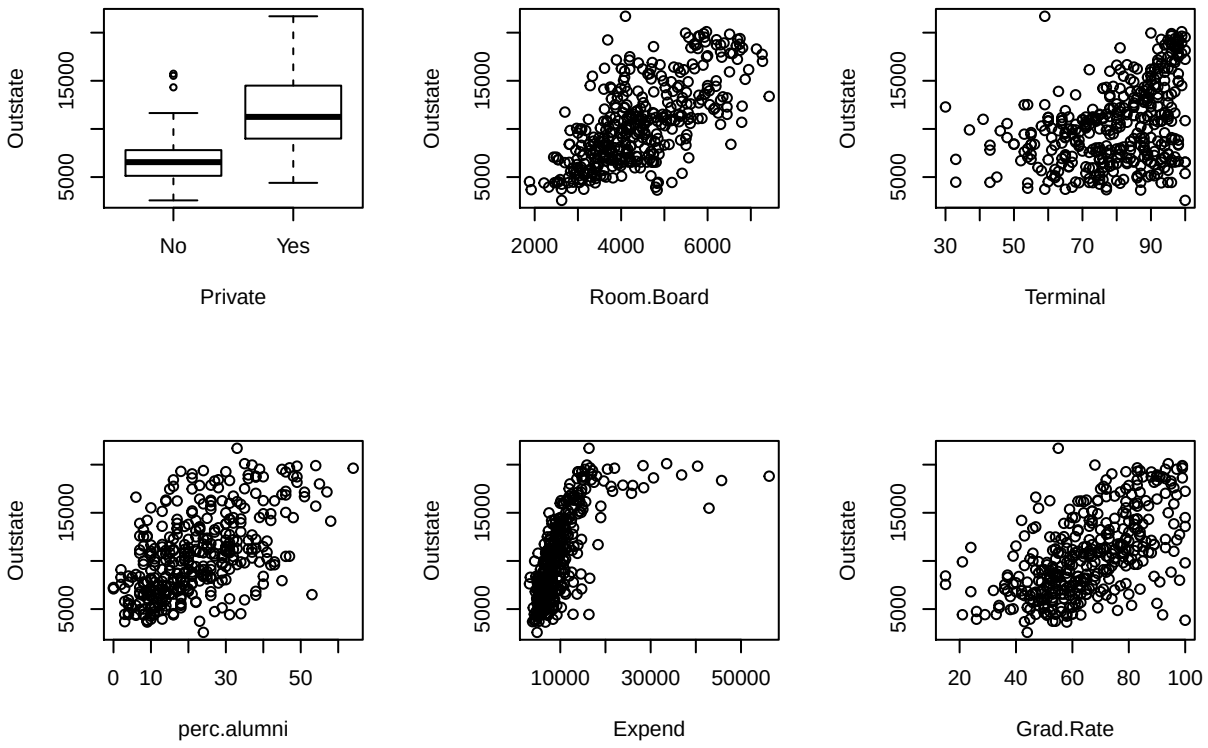
$$b(X, q_j) = (x_i - q_j)^3_+ \begin{cases} (X - q_j)^3 & \text{if } X > q_j \\ 0 & \text{otherwise} \end{cases}$$

This amounts to estimating 7 regression coefficients β_k for $k = 0, .., 7$ to the intercept and predictors.

c)

We will now investigate the realtionship between *Outstate* and the 6 of the predictors, *Private*, *Room.Board*, *Terminal*, *perc.alumni*, *Expend*, and *Grad.Rate*.

```
ds1 = college.train[c("Private", "Outstate")] #binary variable
ds2 = college.train[c("Room.Board", "Outstate")]
ds3 = college.train[c("Terminal", "Outstate")]
ds4 = college.train[c("perc.alumni", "Outstate")]
ds5 = college.train[c("Expend", "Outstate")]
ds6 = college.train[c("Grad.Rate", "Outstate")]

par(mfrow=c(2,3))
plot(ds1)
plot(ds2)
plot(ds3)
plot(ds4)
plot(ds5)
plot(ds6)
```



From each of the plots above we can conclude that at least *Terminal* and *Expend* seems to have a non-linear relationship with *Outstate*. These two variables therefore might benefit from a non-linear transformation. The others variables, *Room.Board*, *perc.alumni* and *Grad.Rate* seem to have a linear realtionship with the response variable. The binary variable *Private* is presented through a boxplot. Generally the data seem to cathegorize quite well into these two classes of private and public universites where the trend is a higher out-of-state tuition for private universites, except for some outliers for public universities where the outcome is very high and therefore can seem to belong to a private universities. Anyway, we cannot transform a binary variable.

d)

(i)

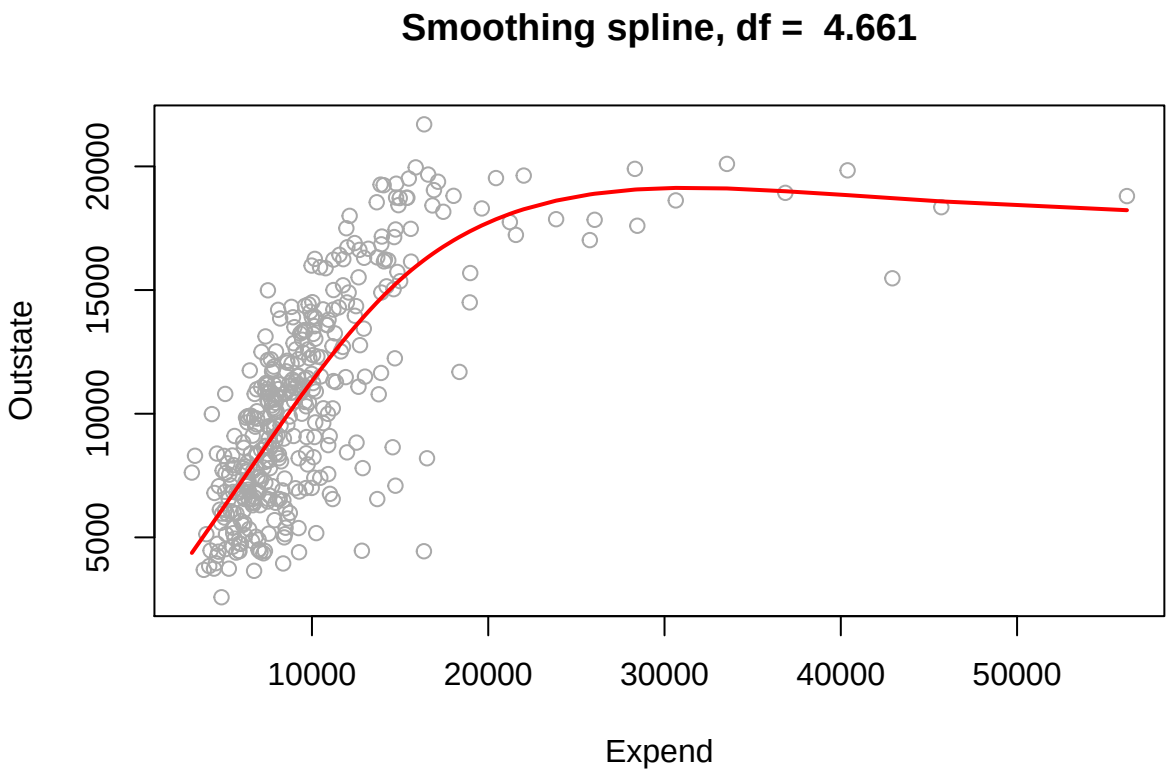
We will now fit several polynomial regression models for *Outstate* with *Terminal* as the only covariate. Each polynomial will have a degree from $d = 1, ..10$.

```
library(ggplot2)

#make a dataframe
ds = College[c("Terminal", "Outstate")]
n = nrow(ds)

# chosen degrees
deg = 1:10

#now iterate over each degree d
dat = c()  #make a empty variable to store predicted values for each degree
MSE_train_poly = c(rep(0,10))  #make a empty variable to store MSE for each degree
MSE_test_poly = c(rep(0,10))

for (d in deg) {
    # fit model with this degree
    mod = lm(Outstate ~ poly(Terminal, d), data= college.train)

    #dataframe for Terminal and Outstate showing result for each degree over all samples
    dat = rbind(dat, data.frame(Terminal = college.train$Terminal, Outstate = mod$fit,
                                degree = as.factor(rep(d,length(mod$fit)))))
    # training MSE
    MSE_train_poly[d] = mean((predict(mod, newdata=college.train) - college.train$Outstate)^2)
    MSE_test_poly[d] = mean((predict(mod, newdata= college.test) - college.test$Outstate)^2)
}

# plot fitted values for different degrees
ggplot(data = ds[train.ind, ], aes(x = Terminal, y = Outstate)) +
    geom_point(color = "darkgrey") + labs(title = "Polynomial regression")+
    geom_line(data = dat, aes(x = Terminal, y = Outstate, color = degree))
```

(ii)

We will now choose a suitable smoothing spline model to predict *Outstate* as a function of *Expend* and plot the fitted function.

```
library(splines)

#perform CV in order to find optimal number of df
fit = smooth.spline(college.train$Expend, college.train$Outstate, cv=TRUE)

#plot training set for Expend as only covariate
plot(college.train$Expend, college.train$Outstate, col = "darkgrey",
    main=paste("Smoothing spline, df = ", round(fit$df,3)), xlab="Expend", ylab="Outstate")

#add fitted function from smoothing spline
lines(fit, col="red", lwd=2)
```




```
#training MSE
pred =  predict( fit, college.train$Expend)
MSE_spline_train <- mean((pred$y - college.train$Outstate )^2)

#test MSE
pred =  predict(fit, college.test$Expend)
MSE_spline_test = mean( (college.test$Outstate - pred$y)^2 )
```

In order to choose the optimal number of degrees of freedom , which is $df = 4.660711$, we did cross validation.

(iii)

We will now report the training MSE for the polynomial regresson models and the smoothing spline model. For the polynomial regression models we present the MSE for each degree of polynomial $d = 1.., 10$,

```
#MSE for polynomial regression models (1-10)
Degree <- seq(1,10,by=1)
MSE = MSE_train_poly
df <- data.frame(Degree, MSE)
kable(df)
```

Degree	MSE
1	15075161

Degree	MSE
2	14330586
3	14249448
4	14247330
5	14231485
6	14230392
7	14153207
8	14097911
9	13841526
10	13822205

For the smoothing spline the training MSE is,

$$MSE = 6.8712814 \times 10^6$$

.

If we look back at the plots from c) we can see that when we plotted *Outstate* against *Expend* we had way less noise than with $Terminal$. Therefore we expected a smaller MSE for the model that uses *Expend*, that is the smoothing spline. We can confirm this as the measured training MSE for the smoothing spline model is smaller than each of the ten polynomial regression models. The plot from d) (ii) confirms this as it indicated a pretty good fit.

From the plot in d) (i) of the polynomial regression, it seems like all degrees of polynomials were somehow similar, at least we cannot see a huge difference from the plot. So even though we normally expect the MSE to decrease with higher degrees, this remark gives a reason to expect that the MSE not will decrease too much, which also seems to be the case when looking at the table that reports the training MSE for each degree of polynomial.

Problem 3

a)

FALSE, FALSE, TRUE, FALSE

b)

In order to predict *Outstate* we will choose the tree-based method: random forests. We know that regression tress usually suffer from high variance and non-robustness. By aggregating many decision trees and averaging the resulting prediciton, random forests solves these two problems. Another advantage with random forests is the fact that the method injets more randomness than for example bagging in order to avoid strong predictors dominating the decision trees. This is done by allowing only certain number of predictors m to be selected at each node, which deacreases the correlation between each tree. Even though random forests is a flexible and accurate method, one big disadvantage is its high complexity as the method creates a lot of trees and therefore requires more computational resources than one simple regression tree.

The tuning parameter in random forrests is the number of predictors you are able to choose between at each split. It is recommended to use $m = p/3$ variables when building a regression trees. Recall that we have 17 parameters in our dataset. We will therefore pick $m = 5$.

```r
library(randomForest)

set.seed(1)

# fit a model using random forests with a sufficiently large number of trees
rf.college <- randomForest(Outstate ~ .,data=college.train, mtry=5, ntrees=500)
#predict the response using test data
yhat.college <- predict(rf.college, newdata=college.test)
#test MSE
MSE_rf <- mean((yhat.college - college.test$Outstate)^2)
```

The test MSE for the random forests is $MSE_{rf} = 2.6079851 \times 10^6$.

c)

We will now compare the test MSEs among the methods used on the data set *College* so far. That is: the two linear model selection methods, forward selection and Lasso method, non-linear methods, polynomial regression and smoothing splines and at last the tree-based method random forrests of regression trees. For the polynomial regression we just show the polynomial degree with the minimal test MSE.

```r
MSE <- c(mse_fwd,mse_lasso,min(MSE_test_poly),MSE_spline_test,MSE_rf)
Method <- c("Forward selection", "Lasso", "Polynomial regression", "Smoothing spline",
            "Random Forest")
df <- data.frame(Method, MSE)
kable(df)
```

Method	MSE
Forward selection	4112680
Lasso	3717020
Polynomial regression	10914136
Smoothing spline	28685427
Random Forest	2607985

The method performing best in terms of prediction error is the random forest of regression trees. But if the aim is to develop a interpretable model Lasso is the best choice.

Problem 4

Start by loading the data of *diabetes* from a population of women.

```r
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
```

a)

In order to answer on the Mulitple choice we will have a look at the training data.

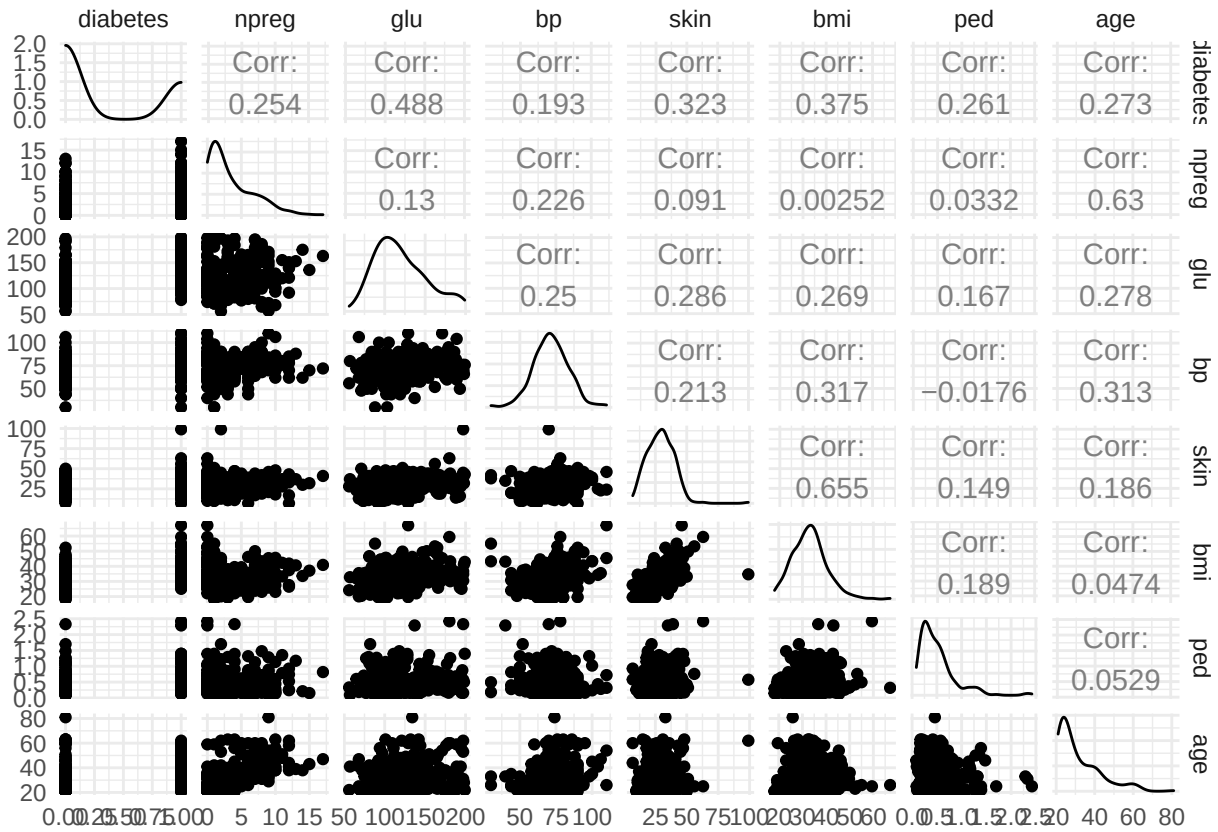
```
library(GGally)
library(gridExtra)

max(d.train$npreg)#max nr of pregnancies
```

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

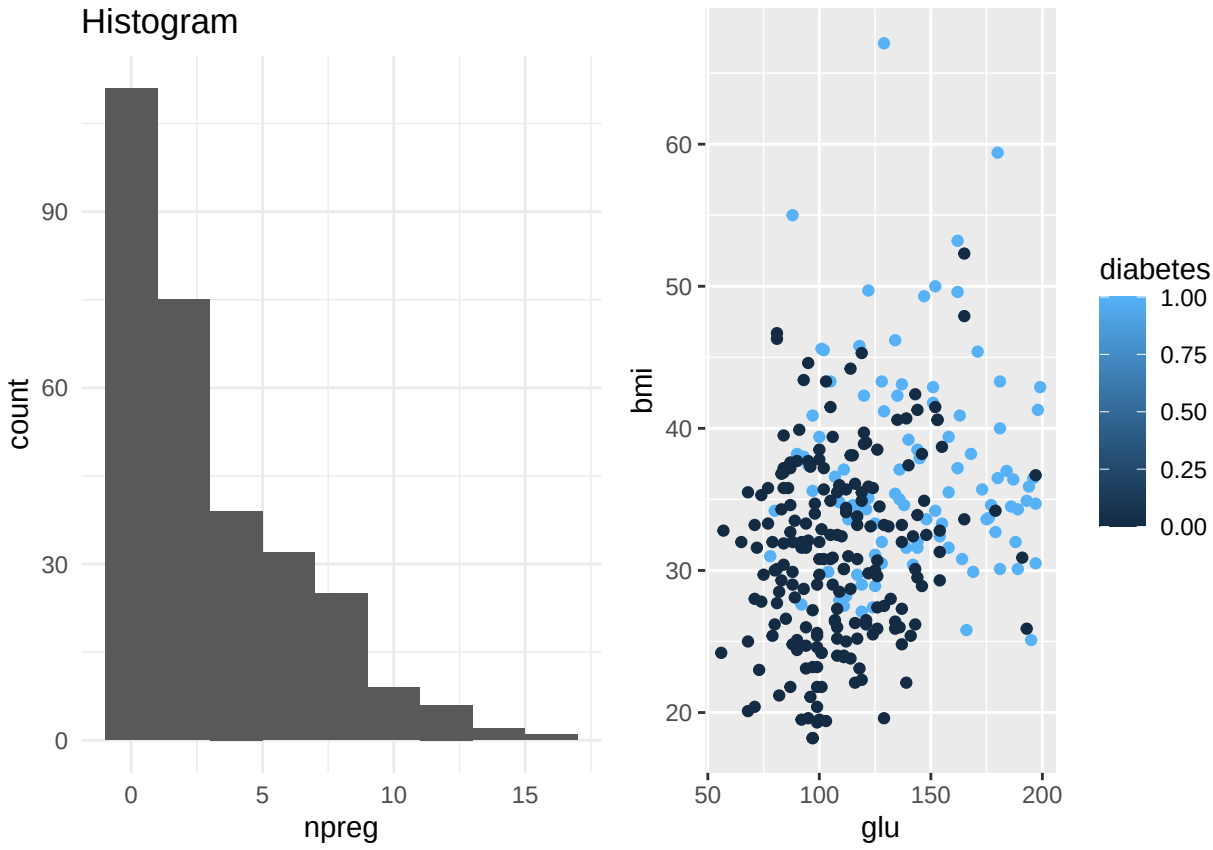
```
head(d.train) #overview of data
```

```
##     diabetes npreg glu bp skin  bmi   ped age
## 339        1     7 109 80   31 35.9 1.127  43
## 270        1     7 152 88   44 50.0 0.337  36
## 210        1     9 119 80   35 29.0 0.263  29
## 117        1     9 112 82   32 34.2 0.260  36
## 390        1     8 151 78   32 42.9 0.516  36
## 217        1     2  90 68   42 38.2 0.503  27
```

```
#plots
ggpairs(d.train) + theme_minimal() #look at correlation between variables
```



```
#plot2 <- ggplot(d.train, aes(x=npreg, y=diabetes))+geom_point()
plot2 <- ggplot(d.train, aes(npreg)) + geom_histogram(binwidth = 2) + labs(title = "Histogram") +
    theme_minimal()
plot3 <- ggplot(d.train,aes(x=glu,y=bmi,color=diabetes))+geom_point()
grid.arrange(plot2, plot3, ncol=2, nrow = 1)
```



TRUE, TRUE, TRUE, TRUE

b)

We will now fit a support vector classifier with linear boundary and a support vector machine with radial boundary to find good functions that predict the diabetes status of a patient.

```
library(e1071)

set.seed(10111)

d.train$diabetes <- as.factor(d.train$diabetes)
d.test$diabetes <- as.factor(d.test$diabetes)


#Fit a support vector classifier (linear boundary)
svm.linear = svm(diabetes~.,
                 data = d.train,
                 kernel = 'linear')
```

```r
#fit a support vector machine (radial boundary)
svm.radial = svm(diabetes~.,
                 data = d.train,
                 kernel = 'radial')

#CV to find the best parameters for each model
cv.linear <- tune(svm, diabetes ~ ., data=d.train, kernel = "linear",
                  ranges = list(cost = c(.001, .01 , .1, 1, 10, 100 ) ) )

cv.radial <- tune(svm, diabetes ~., data = d.train, kernel = "radial",
                  ranges = list(cost = c(0.1,1,10,100,1000), gamma = c(0.5,1,2,3,4) ))

#fit new models with the optimalized parameters from CV
bestmod.linear = cv.linear$best.model
bestmod.radial = cv.radial$best.model

# Predict the response for the test set
pred.lin = predict(bestmod.linear, newdata = d.test)
pred.rad = predict(bestmod.radial, newdata = d.test)
# Confusion tables (0: no diabetes, 1: diabetes)
#for SVC (linear)
table(Prediction = pred.lin, Truth = d.test$diabetes)
```

```
##           Truth
## Prediction   0   1
##          0 137  35
##          1  18  42
```

```r
#for SVM (radial)
table(Prediction = pred.rad, Truth = d.test$diabetes)
```

```
##           Truth
## Prediction   0   1
##          0 133  38
##          1  22  39
```

From the two confusion tables above we can find the missclassification rates. The left rate belongs to the support vector classifier (linear boundary), and the right rate to the support vector machine (radial boundary).

$$\frac{35+18}{137+35+18+42} = 0.2284 \quad \frac{38+22}{133+38+22+39} = 0.2586$$

We know that the missclassification error rate is a good performance measure for classification rules. So by looking at these rates the support vector classifier with a linear boundary seems to make the best classification rule for the prescence of diabetes for our data. This is because the number of missclassifications is remarkable lower for this classifier. We also observe that the number of false positive and false negative findings are both lower for this chosen model compared to the support vector machine with radial boundary.

c)

We will now compare the performance of the support vector classifier and support vector machine with a new method: logistic regression. As we have observed above, our data seems to performe well when

being classified with a linear decision boundary. Logistic regression models the probability that the response belongs to one of the two classes, producing a linear decision boundary. Therefore this method seems to be a good choice for our data.

```
#fit a logistic regression model using training data
glm.fit <- glm(diabetes ~ .,data=d.train, family="binomial")

#predict the response using testing data
glm.probs <- predict(glm.fit, newdata=d.test, type="response")

#sort the probabilities for whether the observations are < or > than p = 0.5
glm.pred = rep("0",length(d.test$diabetes)) #create vector of nr. of elements = dataset
glm.pred[glm.probs>0.5]="1"

#confusion table
table(Prediction = glm.pred, Truth = d.test$diabetes)
```

```
##            Truth
## Prediction   0   1
##          0 136  32
##          1  19  45
```

The missclassification rate for logistic regression is

$$\frac{32+19}{136+45+32+19} = 0.2198$$

We notice that the missclassification rate for logistic regression is even lower than for the two SVMs. This indicates that logistic regression could be a even better choice in order to predict the prescense of diabetes based on our data.

Both logistic regression and support vector machines solve classification problems. In logistic regression all observations contribute to the decision boundary, while for SVMs, only the support vectors (the points closest to the decision boundary) contribute to the margin. A consequence of this is that LR is more sensitive to outliers than SVM.

For classes that are well separeted SVM tend to perform better than LR, while in more overlapping regimes we usually prefer LR. We also know that logistic regression produces probabilistic values, while SVM produces binary values. This can be an advantage if we want an estimation rather than just the resulting class for each observation.

d)

FALSE, FALSE, TRUE, TRUE

e)

We will now show the connection between hinge loss and logistic loss for the $y = -1, 1$ encoding. Let's find the loss function of the logistic regression model in this case. We can see that with this encoding we can write that $P(Y_i = y_i|\boldsymbol{x_i}) = g(y_i\beta^T\boldsymbol{x_i})$ where $g(\boldsymbol{x_i})$ is the logistic function. We will use the notation $f(\boldsymbol{x_i}) = \beta^T\boldsymbol{x_i}$. We verify that this corresponds to the model:

$$P(Y_i = 1|\boldsymbol{x_i}) = \frac{1}{1+e^{-f(\boldsymbol{x_i})}}$$

$$P(Y_i = -1|\boldsymbol{x_i}) = \frac{1}{1+e^{f(\boldsymbol{x_i})}} = 1 - \frac{1}{1+e^{-f(\boldsymbol{x_i})}} = 1 - P(Y_i = 1|\boldsymbol{x_i})$$

So indeed that is a correct way to write the model. We will now calculate the contribution off one observation to the deviance. Remember that the deviance is $-2Log(Likelihood)$, but we will ignore the 2 here.

$$-log(g(y_i\beta^T \boldsymbol{x_i})) =$$
$$-log(\frac{1}{1+e^{-y_i f(\boldsymbol{x_i})}}) = log(1+e(-y_i f(\boldsymbol{x_i})))$$

We found the desired result.

Problem 5

```
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 = "")
GeneData<-t(GeneData)
```

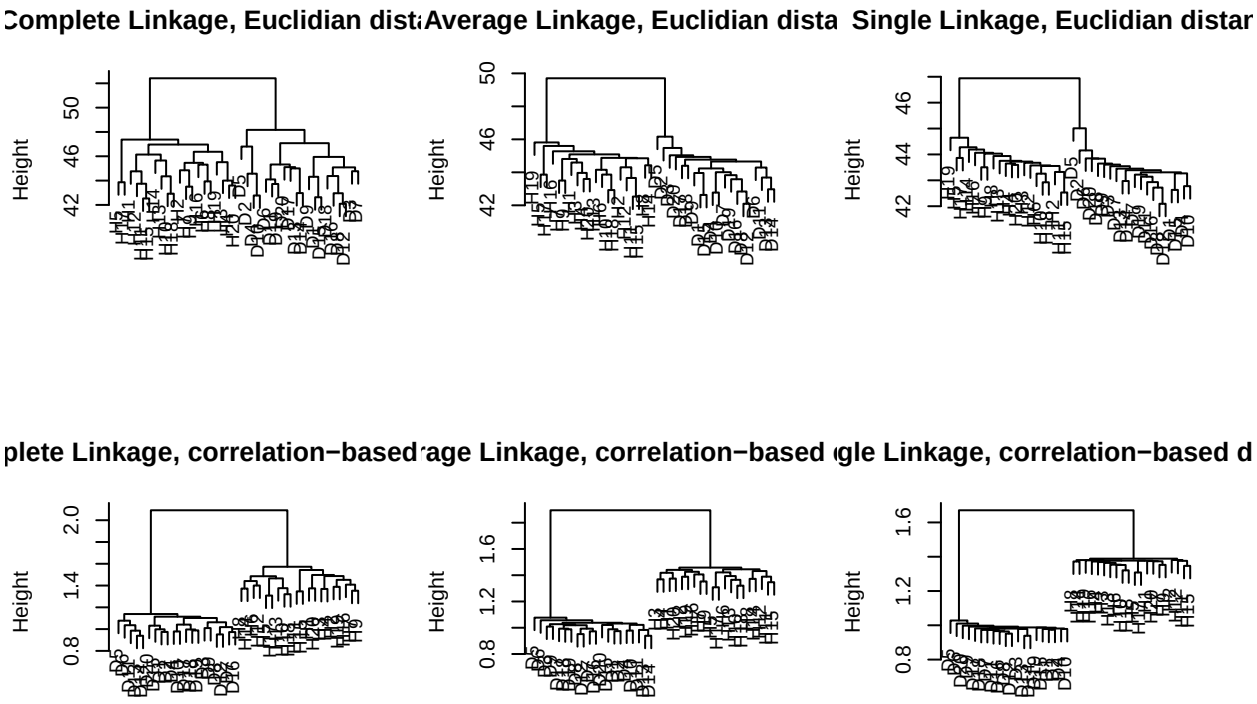
a)

We start by performing hierarchical clustering on the dataset. We try the complete, single and average linkage for both the Euclidian and correlation-based distance. We transposed the data in order to cluster the tissues into the two patient groups.

```
hc.eucl.complete=hclust(dist(GeneData,method="euclidian"), method="complete")
hc.eucl.average=hclust(dist(GeneData,method="euclidian"), method="average")
hc.eucl.single=hclust(dist(GeneData,method="euclidian"), method="single")
```

```
correlation<-dist(cor(t(GeneData)))
hc.corr.complete=hclust(correlation, method="complete")
hc.corr.average=hclust(correlation, method="average")
hc.corr.single=hclust(correlation, method="single")
```

```
par(mfrow=c(2,3))
plot(hc.eucl.complete,main="Complete Linkage, Euclidian distance", xlab="", sub="", cex=.9)
plot(hc.eucl.average, main="Average Linkage, Euclidian distance", xlab="", sub="", cex=.9)
plot(hc.eucl.single, main="Single Linkage, Euclidian distance", xlab="", sub="", cex=.9)
plot(hc.corr.complete,main="Complete Linkage, correlation-based distance", xlab="", sub="", cex=.9)
plot(hc.corr.average, main="Average Linkage, correlation-based distance", xlab="", sub="", cex=.9)
plot(hc.corr.single, main="Single Linkage, correlation-based distance", xlab="", sub="", cex=.9)
```



The dendograms seem to recognize that there are two different groups.

b)

We now use the dendograms to cluster the tissues into two groups.

```
cutree(hc.eucl.complete, 2)
```

```
##  H1  H2  H3  H4  H5  H6  H7  H8  H9 H10 H11 H12 H13 H14 H15 H16 H17 H18 H19 H20
##   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
##  D1  D2  D3  D4  D5  D6  D7  D8  D9 D10 D11 D12 D13 D14 D15 D16 D17 D18 D19 D20
##   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2
```

```
cutree(hc.eucl.average, 2)
```

```
##  H1  H2  H3  H4  H5  H6  H7  H8  H9 H10 H11 H12 H13 H14 H15 H16 H17 H18 H19 H20
##   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
##  D1  D2  D3  D4  D5  D6  D7  D8  D9 D10 D11 D12 D13 D14 D15 D16 D17 D18 D19 D20
##   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2
```

```
cutree(hc.eucl.single, 2)
```

```
##  H1  H2  H3  H4  H5  H6  H7  H8  H9 H10 H11 H12 H13 H14 H15 H16 H17 H18 H19 H20
##   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
```

```
##  D1  D2  D3  D4  D5  D6  D7  D8  D9 D10 D11 D12 D13 D14 D15 D16 D17 D18 D19 D20
##   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2
```

```
cutree(hc.corr.complete, 2)
```

```
##  H1  H2  H3  H4  H5  H6  H7  H8  H9 H10 H11 H12 H13 H14 H15 H16 H17 H18 H19 H20
##   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
##  D1  D2  D3  D4  D5  D6  D7  D8  D9 D10 D11 D12 D13 D14 D15 D16 D17 D18 D19 D20
##   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2
```

```
cutree(hc.corr.average, 2)
```

```
##  H1  H2  H3  H4  H5  H6  H7  H8  H9 H10 H11 H12 H13 H14 H15 H16 H17 H18 H19 H20
##   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
##  D1  D2  D3  D4  D5  D6  D7  D8  D9 D10 D11 D12 D13 D14 D15 D16 D17 D18 D19 D20
##   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2
```

```
cutree(hc.corr.single, 2)
```

```
##  H1  H2  H3  H4  H5  H6  H7  H8  H9 H10 H11 H12 H13 H14 H15 H16 H17 H18 H19 H20
##   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
##  D1  D2  D3  D4  D5  D6  D7  D8  D9 D10 D11 D12 D13 D14 D15 D16 D17 D18 D19 D20
##   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2
```

We know that the first 20 tissues come from healthy patients and the last 20 from diseased ones. Therefore it seems like all linkage and distance measures perform perfectly.

c)

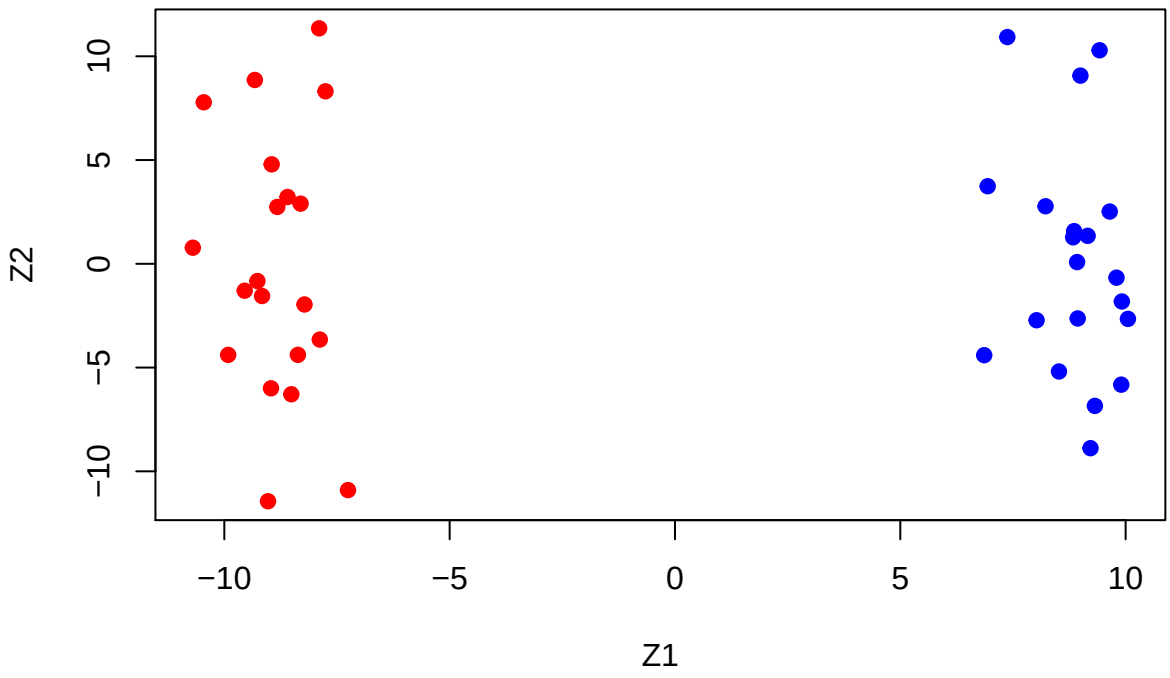
The elements of the vector ϕ are called loadings and define a direction in the feature space along which the data varies the most. The data is a $n \times p$ matrix X.

For the first principal component we want $Z_1 = \phi_1 X$ subject to $||\phi_1||_2 = 1$. We want Z_1 to have the highest possible variance $V(Z_1) = \phi_1^T \Sigma \phi_1$, where Σ is the covariance matrix of X. The first principal component scores are then the column eigenvector corresponding to the largest eigenvalue of Σ.

d)

```
color<-c(rep(1,20),rep(2,20))
```

```
pca_gene=prcomp(GeneData, scale=TRUE)
plot(pca_gene$x[,1:2], col=c("red","blue")[color],pch=19,xlab="Z1",ylab="Z2")
```



Now we calculate the proportion of variance explained (PVE) by the 5 first components.

```
pve=100*pca_gene$sdev^2/sum(pca_gene$sdev^2)
cumsum(pve)[5]
```

```
## [1] 21.09659
```

About 21 percent of the variance is explained by first 5 PCs.

e)

We now will use the results from PCA to find which genes vary the most accross the two groups. For this, we refer to the score vectors obtained by PCA. The genes that vary the most accross the two groups are the ones that have the highest scores. We will display the 5 genes that vary most for the first principal components and the same for the second principal components.

```
gene_loading = pca_gene$rotation[, 1:2]
informative_loadings = rbind(head(gene_loading[order(gene_loading[, 1], decreasing = TRUE),
                                               ]), head(gene_loading[order(gene_loading[, 2], decreasing
informative_loadings
```

```
##               PC1          PC2
## G502  0.094850438 -0.002729035
## G589  0.094497659 -0.018373912
```

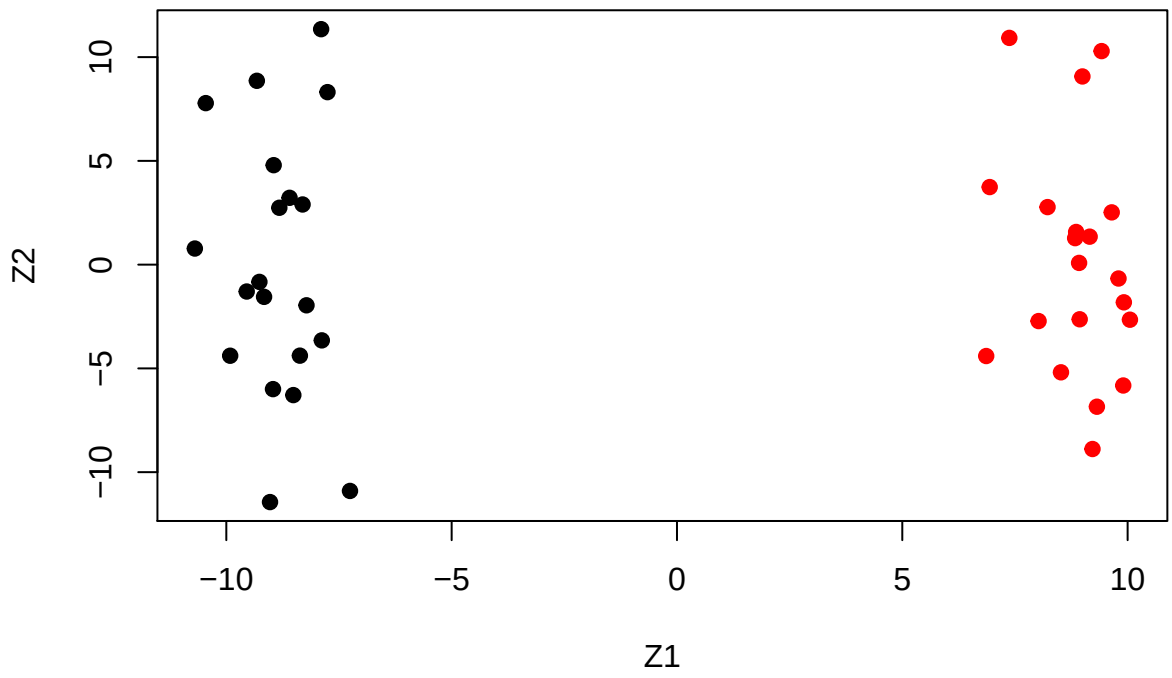
```
## G565  0.091838234  0.016964824
## G590  0.091731695 -0.025556766
## G600  0.091673220 -0.004948422
## G551  0.087683596  0.013883967
## G989  0.015472152  0.110029421
## G95   0.012194885  0.090990473
## G7    0.003764118  0.087251726
## G399 -0.005157145  0.083070538
## G703  0.013810197  0.082616817
## G474 -0.021381601  0.076298114
```

f)

```
km.out = kmeans(GeneData,2, nstart = 20)
km.out$cluster
```

```
##  H1  H2  H3  H4  H5  H6  H7  H8  H9 H10 H11 H12 H13 H14 H15 H16 H17 H18 H19 H20
##   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
##  D1  D2  D3  D4  D5  D6  D7  D8  D9 D10 D11 D12 D13 D14 D15 D16 D17 D18 D19 D20
##   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2
```

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



References

James, G., D. Witten, T. Hastie, and R. Tibshirani. 2013. An Introduction to Statistical Learning with Applications in R. New York: Springer.