Compulsory assignment 2, group 26

Jostein Aasteboel Aanes, Sindre skaugset Olderkjær, Markus Trætli

19 April, 2021

Problem 1

Problem 1a)

True, True, True, False

Problem 1b)

Below is code using best subset selection to find which covariates are most significant for predicting the number of birds killed by a cat. For choosing the covariates, one selects those that yields the model with lowest BIC.

```
catsub = regsubsets(birds~., data = catdat.train, nvmax = 17) #Do subset analysis

#Extract the model with the lowest bic score
catsub.summary=summary(catsub)
catsub.summary$which[which.min(catsub.summary$bic),]
```

```
##       (Intercept)               sex            weight           dryfood
##              TRUE             FALSE             FALSE             FALSE
##           wetfood               age      owner.income    daily.playtime
##              TRUE             FALSE             FALSE              TRUE
##       fellow.cats         owner.age        house.area       children.13
##             FALSE             FALSE             FALSE              TRUE
##             urban              bell              dogs daily.outdoortime
##              TRUE              TRUE             FALSE              TRUE
##      daily.catnip          neutered
##             FALSE             FALSE
```

Here one can see that the reduced model that was found to be the best, was the model using the predictors: wetfood, daily playtime, urban, children 13, bell and daily outdoortime. There are other candidates that have quite similar BIC, but since the best subset selection method uses cross-validation in selecting the best model, one prefers the model with lowest BIC. One may also choose a model with higher BIC in order to reduce the number of covariates if desirable. In this problem, 6 covariates is satisfactory.

```
#Fit top model. Get MSE for both train and test data set
BestSubCat.model = lm(birds ~ wetfood + daily.playtime + bell + urban + children.13 + daily.outdoortime, data
BestSubCat.testPred = predict(BestSubCat.model, catdat.test)
BestSubCat.trainPred = predict(BestSubCat.model, catdat.train)

#Calculates the test and train MSE
BestSubCat.testMSE = mean((BestSubCat.testPred - catdat.test$birds)^2)
BestSubCat.trainMSE = mean((BestSubCat.trainPred - catdat.train$birds)^2)
```

The best subset model had a test MSE of 299.88.

Problem 1c)

Now one can select a model using lasso regression instead. In order to select the optimal tuning parameter λ, one should use cross-validation and select the λ that yields the lowest cross-validation error.

```
set.seed(1)
#Cross validate for lambda
cv = cv.glmnet(x.train,y.train) #Automatically chooses values for lambda to test

#Fit the model using the best lambda from the cross validation
Lasso.model = glmnet(x.train,y.train, lambda = cv$lambda.min)

#show which coefficients that are non-zero
Lasso.model$beta[Lasso.model$beta[,1]!=0,1]
```

```
##            weight           dryfood           wetfood      owner.income
##     -2.745556e+00     -2.683814e+00     -7.729154e+00     -1.153767e-05
##    daily.playtime         owner.age        house.area       children.13
##     -1.019104e+01     -1.647091e-01      7.355691e-02      3.774737e+00
##             urban              bell              dogs daily.outdoortime
##     -8.408971e+00     -5.000957e+01      3.625879e+00      1.139876e+00
##          neutered
##     -4.519351e+00
```

```
#Predict on the test and training set and report MSE
Lasso.trainPred = predict(Lasso.model, x.train)
Lasso.trainMSE = mean((Lasso.trainPred- y.train)^2)

Lasso.testPred = predict(Lasso.model, x.test)
Lasso.testMSE = mean((Lasso.testPred- y.test)^2)
```

From the command "Lasso.model$beta[Lasso.model$beta[,1]!=0,1]", one sees that the non-zero coefficients are: weight, dryfood, wetfood, owner.income, daily.playtime, owner.age, house.area, children.13, urban, bell, dogs, daily.outdoortime, and neutered.

The optimal value for the penalty parameter was $\lambda = 0.447$ and the test MSE using the corresponding model was 298.4.

Problem 1d)

In lasso regression one aims to minimize the following expression

$$RSS + \lambda \sum_{j=1}^{p} |\beta_j|.$$

Therefore, the following holds:

1) As $\lambda \to \infty$ the regression coefficients, β_i will all shrink to zero, and one is left with just the intercept.
2) As $\lambda = 0$, the problem reduces to minimize the RSS, thus one gets a regular multivariate linear regression model.

Problem 1e)

```
intercept.model<- glm(birds ~ 1, data=catdat.train)           #Get model with only intercept
intercept.testPred <- predict(intercept.model, catdat.test)   #predict on test using only the intercept
intercept.trainPred <- predict(intercept.model, catdat.train) #predict on train using only the intercept
intercept.testMSE <- mean((intercept.testPred-y.test)^2)      #calculate test MSE for intercept only model
intercept.trainMSE <- mean((intercept.trainPred-y.train)^2)    #calculate training MSE for intercept only mod

fullMultipleLinReg.model <- glm(birds ~ ., data=catdat.train)                  #Train full model
fullMultipleLinReg.testPred <- predict(fullMultipleLinReg.model, catdat.test)   #predict using full model
fullMultipleLinReg.trainPred <- predict(fullMultipleLinReg.model, catdat.train) #predict using full model
fullMultipleLinReg.testMSE <- mean((fullMultipleLinReg.testPred-y.test)^2)      #test MSE for full model
fullMultipleLinReg.trainMSE <- mean((fullMultipleLinReg.trainPred-y.train)^2)   #train MSE for full model

#tests if the model with just the intercept is better than the ones from b) and c)
intercept.betterB <-ifelse(intercept.testMSE<BestSubCat.testMSE, "Just intercept better than best subset mode
```

```
intercept.betterC <-ifelse(intercept.testMSE<Lasso.testMSE, "Just intercept better than lasso model","Just in

#Tests if the full model is better than the ones from b) and c)
fullModel.betterB <- ifelse(fullMultipleLinReg.testMSE<BestSubCat.testMSE, "Full model better than best subse
fullModel.betterC <- ifelse(fullMultipleLinReg.testMSE<Lasso.testMSE, "Full model better than lasso model", "
```

Calling the values calculated above, the result is as following:

1) Just intercept worse than best subset model
2) Just intercept worse than lasso model
3) Full model worse than best subset model
4) Full model worse than lasso model

From this, one can see that the models from b) and c) are preferable to a model with either just the intercept or all the covariates. This is expected, as in both task b) and c) the objective is to use methods that reduce overfitting compared to the full linear model, in order to minimize the test MSE. Neither method yielded a model with either just the intercept or all the covariates.

Problem 1f)

Below is a table showing the test MSE for the models explored

```
Models = c("Best subset model", "Intercept only", "Lasso", "Ordinary model")

test_MSE = c(BestSubCat.testMSE, intercept.testMSE, Lasso.testMSE, fullMultipleLinReg.testMSE)

train_MSE = c(BestSubCat.trainMSE, intercept.trainMSE, Lasso.trainMSE, fullMultipleLinReg.trainMSE)

data.frame(Models, test_MSE, train_MSE)
```

```
##              Models  test_MSE train_MSE
## 1 Best subset model  299.8835  367.1225
## 2    Intercept only 4914.0694 5975.8423
## 3             Lasso  298.4001  342.1609
## 4    Ordinary model  301.9186  337.9155
```

One can see that all the models perform better than the intercept only model, on both the training data and the test data. This implies that at least some of the covariates have coefficients with a non-zero value. In addition, when looking at the training MSE, one sees that the full model outperforms the best subset model and the lasso-model. However, one also sees that the full model does worse than the best subset and lasso models on the test data. This is expected, as the full model ends up overfitting on the training data, and thus it has a higher variance than the best subset and lasso models. Since the best subset and lasso methods are regularization methods, they increase the models training error in order to obtain a lower test error.

Problem 2

Problem 2a)

True, True, False, False

Problem 2b)

Below are the basis functions for a cubic spline with knots at the quartiles q_1, q_2 for variable X.

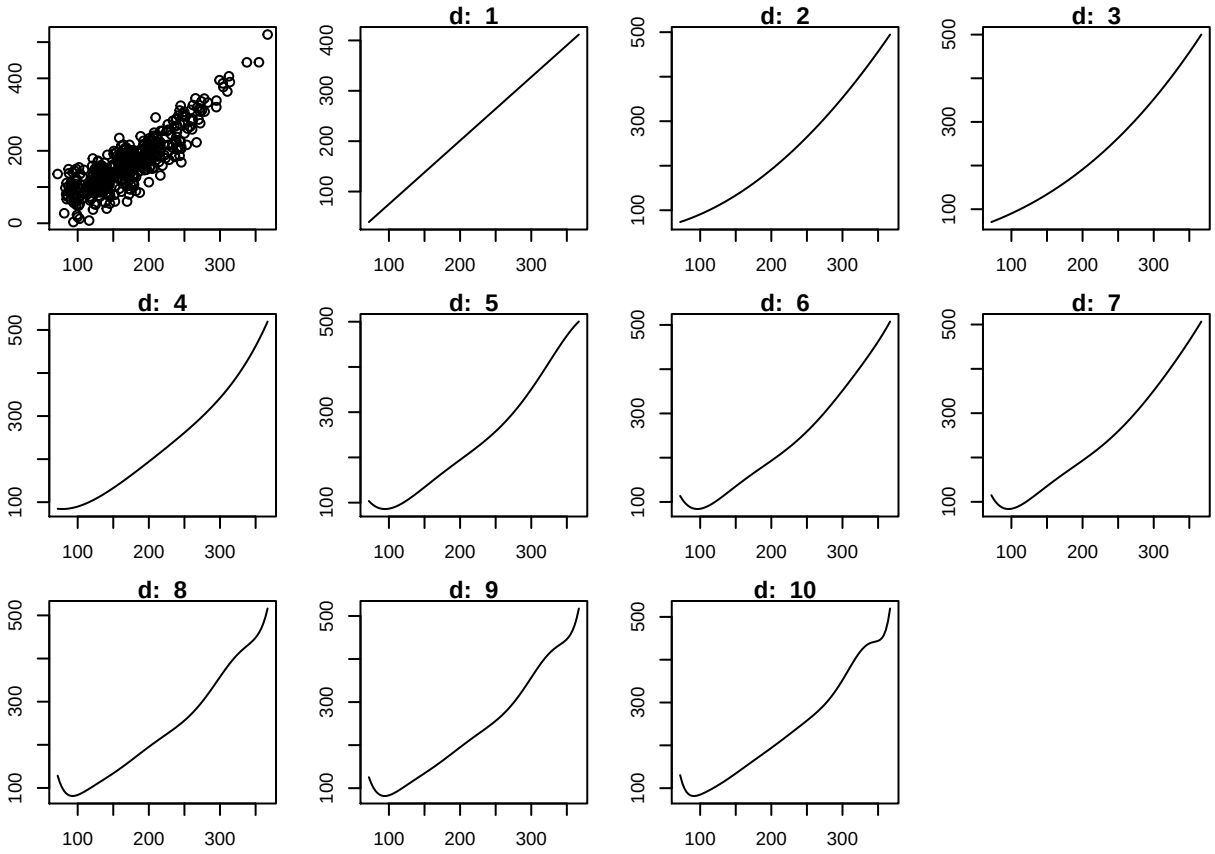
$$b_0 = 1,\ b_1(x) = x,\ b_2(x) = x^2,\ b_3(x) = x^3, b_4(x) = (x - q_1)^3_+,\ b_5(x) = (x - q_2)^3_+$$

where

$$(x - c)^d_+ = \begin{cases} (x - c)^3 & x > c \\ 0 & x \leq c \end{cases}$$

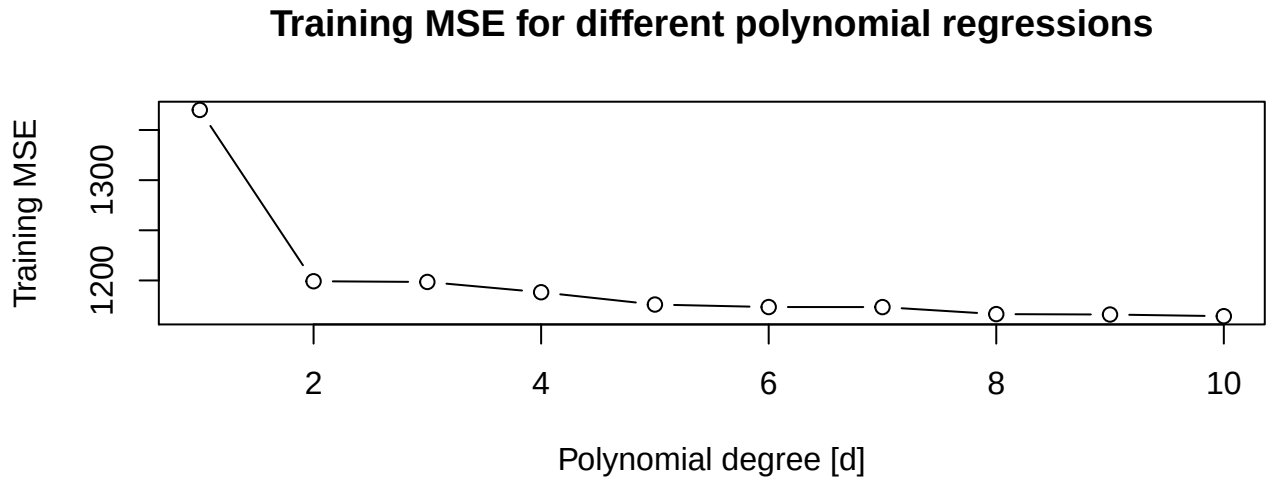
Problem 2c)

```
id <- "1iI6YaqgG0QJW5onZ_GTBsCvpKPExF30G" # google file ID
catdat <- read.csv(sprintf("https://docs.google.com/uc?id=%s&export=download", id),
header = T)

set.seed(4268)
train.ind = sample(1:nrow(catdat), 0.5 * nrow(catdat))
catdat.train = catdat[train.ind, ]
catdat.test = catdat[-train.ind, ]

par(mfrow=c(3,4), cex=0.6,mai=c(0.32,0.32,0.12,0.12),oma = c(0.12, 0.12, 0.012, 0.012))
#Plots birds against daily.outdoortime to see plausible non-linearity
plot(catdat$daily.outdoortime,catdat$birds)

#Seq of values for daily.outdoortime
x_axis_daily.outdoortime <-seq(min(catdat$daily.outdoortime),
                               max(catdat$daily.outdoortime), length.out=100)

training_MSE <- rep(0,10) #will store the training MSE for the different models

for (i in 1:10){
  #Trains model with i'th degree polynomial
  birds.lm <- lm(birds ~ poly(daily.outdoortime, i ) , data=catdat.train)

  #Predicts birds against daily outdoor time
  birds_predicted <-predict.lm(birds.lm, newdata=data.frame(daily.outdoortime=x_axis_daily.outdoortime))

  #Plots birds predicted for each model against daily.outdoortime
  plot(x_axis_daily.outdoortime, birds_predicted,
       type ="l", main=paste("d: ",as.character(i)),xlab="daily.outdoortime", ylab="birds")

  training_MSE[i] = mean(birds.lm$residuals^2) #Training MSE for current model stored
}
```



In the plots above, the first plot seems to indicate some non-linearity between birds and daily.outdoortime. In the remaining plots, one can see the prediction of birds killed in polynomial regression models with degrees ranging from 1 to 10.

```
#Plot training MSE against different polynomial degrees
plot(training_MSE, type="b", main="Training MSE for different polynomial regressions", xlab = "Polynomial deg
```



From the plot above, one can see that the training MSE decreases drastically between the model with 1st degree polynomial and the model with 2nd degree polynomial. Afterwards, the training MSE slowly decreases with higher order. This means that a polynomial regression of order 2 is significantly more accurate on this data set compared to polynomial regression of order 1, while one does not gain any significant increase in accuracy with higher order polynomial regression models. This implies that the models with polynomials of higher degree than 2, might be overfitted. This would be much more apparent when looking at the test MSE. The training MSE must clearly decrease with the order of the polynomial.

Problem 3

Problem 3a)

True, True, False, False

Problem 3b)

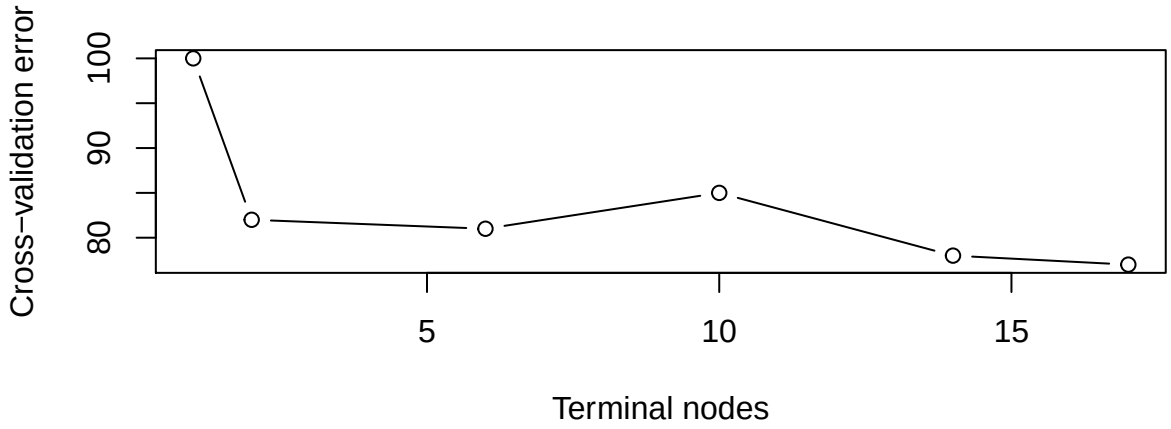
Now one can prune the tree from a) down to three leaves. In order to achieve three leaves, one must have two internal nodes. One chooses these internal nodes based on how much they reduce training RSS. Since a tree is grown by a greedy approach, where each split minimizes RSS at that step, it is obvious that the root remains. Now one has to choose between the two nodes: "age<46.5" and "country: Indonesia, Japan, Korea". In order to choose between these two, one can look at the length of the branches out from these nodes. Because the tree is a regression tree, the length of the branches is proportional to the reduction in the cost-function stated below

$$Q(T) = \sum_{m=1}^{|T|} \sum_{x_i \in R_m} (y_i - \hat{y}_{R_m})^2.$$

The cost-function $Q(T)$ is equal to training RSS, which is what one aims to minimize with the tree. Thus one chooses the internal node with the longest branches, which is "country:Indonesia, Japan, Korea". Thus, after pruning the tree from a) down to three leaves, one gets the tree with the root "age<81.5" and in the right branch a second internal node "country: Indonesia, Japan, Korea".

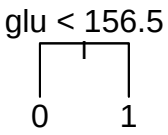
Problem 3c)

```
#Loading the data
id <- "1Fv6xwKLSZHldRAC1MrcK2mzdOYnbgv0E" # google file ID
d.diabetes <- dget(sprintf("https://docs.google.com/uc?id=%s&export=download", id))
d.train = d.diabetes$ctrain #Creating a training data set
d.test = d.diabetes$ctest #-||---------- test data set

diabetes.tree <- tree(as.factor(diabetes) ~ ., data=d.train) #Creating a classification tree

set.seed(1)
diabetes.cv <- cv.tree(diabetes.tree, FUN = prune.misclass) #Use 10-fold cv in order to find optimal number o
plot(diabetes.cv$size, diabetes.cv$dev , type = "b", xlab = "Terminal nodes" , ylab="Cross-validation error")
```



Based on the plot above, it is reasonable to prune the tree down to 2 terminal nodes. This is because the cross-validation error is only slightly bigger for the tree with 2 terminal nodes compared to the tree with 6 terminal nodes, and 2 terminal nodes gives the simplest tree.

```
diabetes.prunedtree <- prune.misclass(diabetes.tree, best = 2) #apply pruning to get tree with best performan

par(mai=c(0.2, 0.5, 0.2, 0.5))
plot(diabetes.prunedtree) #Plots the pruned tree
text(diabetes.prunedtree, pretty = 1)
```




```
diabetes.predict = predict(diabetes.prunedtree, d.test, type="class") #Predict who have diabetes from test da
diabetes.actualValues = d.test$diabetes #Extract true classifications

#Create table with predictions against actual values
diabetes.table <- table(diabetes.actualValues, diabetes.predict)

misclassification_rate <- 1 - sum(diag(diabetes.table))/sum(diabetes.table) #Calculate misclassification rate
```

The misclassification error is 56 and the misclassification rate is 0.241.

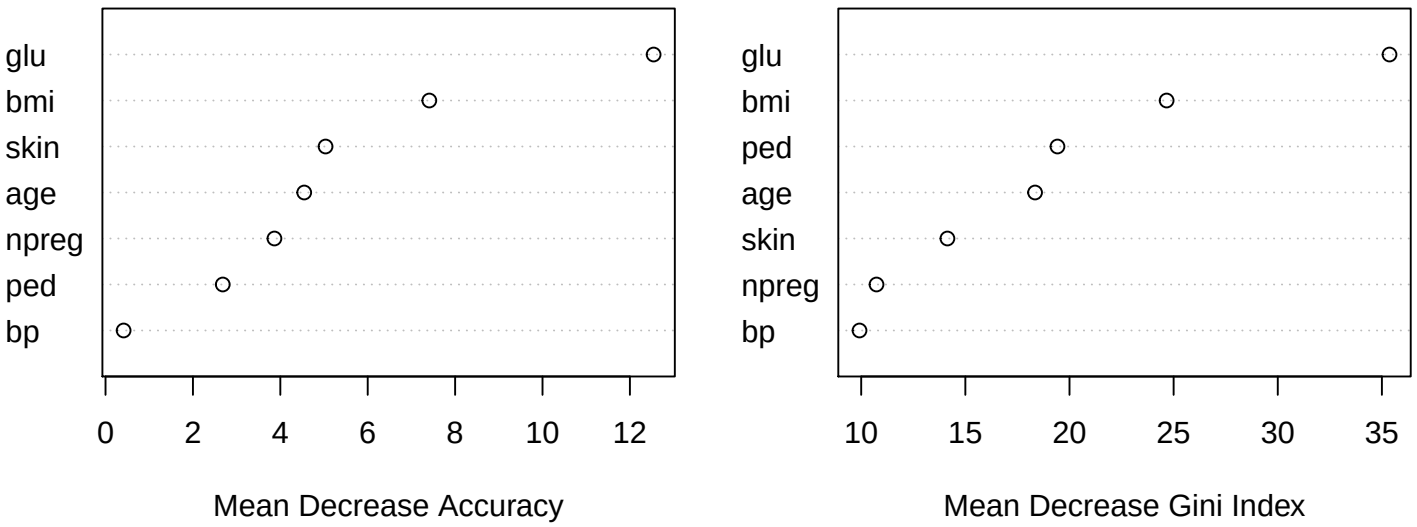
(ii)

```
numPredictors <- length(d.train[1,])-1 #Calculating the number of predictors

#Trying to find the best predictors
set.seed(1)
tune.out <- tune(randomForest, as.factor(diabetes)~., data=d.train, ranges=list(mtry=c(2,3,4), ntree=c(10,100

tune.out$best.parameters
```

```
##   mtry ntree
## 5    3   100
```

One can see that the best number of predictors per split is 3. This is the same as the square root of the number of predictors, which is an often used value for mtry when making classification random forests. The number of trees, ntree, is supposed to be better the bigger it is, but at the cost of computation time. The tune-function found that ntree= 100 is better than ntree= 1000, which is probably due to randomness. However, it implies that ntree= 1000 can only be marginally better than ntree= 100, and thus it is advantageous to choose ntree= 100.

```
#Building a random forest
diabetes.rf <- randomForest(as.factor(diabetes)~., data=d.train, mtry=3, ntree=100, importance=TRUE)

par(mfrow=c(1,2), mai=c(0.8, 0.2, 0.2, 0.2))
impToPlot <- importance(diabetes.rf, scale=T)
dotchart(sort(impToPlot[,3]), xlab="Mean Decrease Accuracy") #Plot the Mean decrease accuracy for each covari
dotchart(sort(impToPlot[,4]), xlab="Mean Decrease Gini Index") #Plot the Mean decrease gini index for each co
```



In this plot, one can see that glu is the most important predictor, which is consistent with the tree in (i). Based on the plot, the second most important predictor is bmi.

```
rf.predict <- predict(diabetes.rf, d.test, type="class") #Predict who have diabetes from the test data

rf.table <- table(diabetes.actualValues, rf.predict)
rf.misclassrate <- 1-sum(diag(rf.table))/sum(rf.table) #Calculate misclassification rate
```

The misclassification error using the random forest approach is 55, while the misclassification rate is 0.237. Compared to the error one gets using a classification tree as in (i), one gains little in accuracy. If one considers inference or computational ease, the tree from (i) is preferable to the one that the random forest approach yields.

Problem 4

Problem 4a)

True, True, False, True

Problem 4b)

i)

By using a SVM method, one is guaranteed to find a separating hyperplane due to $n < p + 1$. Logistic regression is unstable for $p \gg n$. One could use feature selection in combination with logistic regression, but since a vast majority of the covariates are then excluded, one may lose a lot of predictive power. Thus, a SVM is more suitable for the problem compared to a logistic regression approach. Due to $p \gg n$, many regression models will struggle as the predictor coefficients cannot be uniquely determined, since there are too few degrees of freedom. As a result, comparison based models like SVM or KNN are preferable for similar problems. Alternatively, one could reduce the dimensionality of the problem by using methods like principal component regression, forward subset selection, lasso regression or ridge regression.

ii)

In the paper, they suggest to use an ensemble-SVM method combined with bagging to decide which genes to use for singular value decomposition to reveal clusters in the data. The clusters would then be used to figure out which genes would place a patient in risk of relapsing with leukemia.

iii)

In the code below, the SVM model with linear boundary and $C = 1$ has been fitted on the training set and used to predict on both the training and test data sets.

```
#fit the model on the training data using C=1 and linear kernel. cost = 1/C, thus cost=1 -> C=1
svm.model = svm(Category~., data = d.leukemia.train, cost = 1, kernel = "linear", scale = T)

#User the model to predict on the training set and test set
svm.predtrain = predict(svm.model, d.leukemia.train)
svm.predtest = predict(svm.model, d.leukemia.test)
```

The confusion table for the training set is

```
#Confusion table and misclassification rate
svm.trainerr = mean(svm.predtrain != d.leukemia.train$Category) #Calculates the misclassification rate
svm.traincomf = confusionMatrix(as.factor(svm.predtrain), as.factor(d.leukemia.train$Category), positive = "R
svm.traincomf$table
```

```
##              Reference
## Prediction    Non-Relapse Relapse
##   Non-Relapse          30       0
##   Relapse               0      15
```

with misclassification rate 0. This is unsurprising since we have a separating hyperplane.

The confusion table for the test set is

```
#Confusion table and missclassification rate
svm.testerr = mean(svm.predtest != d.leukemia.test$Category) #Calculates the misclassification rate
```

```
svm.testcomf = confusionMatrix(as.factor(svm.predtest), as.factor(d.leukemia.test$Category), positive = "Rela
svm.testcomf$table
```

```
##              Reference
## Prediction    Non-Relapse Relapse
##   Non-Relapse           8       4
##   Relapse               1       2
```

with misclassification rate 0.333.

From the table, given that relapse equals positive and non-relapse equals negative, the model results in 4 false-negatives and 1 false-positive. The model can be considered a success on the grounds of predictive ability. However, considering the high rate of false negatives, the model is not a success. The aim of this model is to find out which patients needs closer follow up after ended treatment, so that one can early discover if the patient has a relapse of leukemia. Therefore, it is preferable that the model sacrifices some predictive power, in order to decrease the number of false negatives.

iv) Below, two models are fitted to the data. One model with $\gamma = 10^{-2}$ and with $\gamma = 10^{-5}$. Afterwards, the models are used to predict on both the training and test data sets.

```
#Fit the two models
svm.modelg1 = svm(Category~., data = d.leukemia.train, cost = 1, kernel = "radial", gamma = 10^(-2), scale =
svm.modelg2 = svm(Category~., data = d.leukemia.train, cost = 1, kernel = "radial", gamma = 10^(-5), scale =

#Use the models to predict on the training set and test set
svm.predtraing1 = predict(svm.modelg1, d.leukemia.train)
svm.predtestg1 = predict(svm.modelg1, d.leukemia.test)

svm.predtraing2 = predict(svm.modelg2, d.leukemia.train)
svm.predtestg2 = predict(svm.modelg2, d.leukemia.test)
```

Now the first model with $\gamma = 10^{-2}$, the confusion table for the training predictions are as follows:

```
#Confusion table and misclassification rate
svm.trainerrg1 = mean(svm.predtraing1 != d.leukemia.train$Category)
svm.traincomfg1 = confusionMatrix(as.factor(svm.predtraing1), as.factor(d.leukemia.train$Category), positive
svm.traincomfg1$table
```

```
##              Reference
## Prediction    Non-Relapse Relapse
##   Non-Relapse          30       0
##   Relapse               0      15
```

Here the training misclassification rate is 0. This makes sense, as high γ makes the boundary more flexible. Hence, it will fit well with the training data. The confusion table from the test set is as follows:

```
svm.testerrg1 = mean(svm.predtestg1 != d.leukemia.test$Category)
svm.testcomfg1 = confusionMatrix(as.factor(svm.predtestg1), as.factor(d.leukemia.test$Category), positive = "
svm.testcomfg1$table
```

```
##              Reference
## Prediction    Non-Relapse Relapse
##   Non-Relapse           9       6
##   Relapse               0       0
```

The test misclassification rate of this model is 0.4. A possible explanation could be that the high value of γ makes the boundary too stringent. Therefore, the area where the points would get assigned to relapse could then become too small. Hence, all the test points are classified as non-relapsed.

For the second model with $\gamma = 10^{-5}$, the result is as follows:

```
#Confusion table and misclassification rate
svm.trainerrg2 = mean(svm.predtraing2 != d.leukemia.train$Category)
svm.traincomfg2 = confusionMatrix(as.factor(svm.predtraing2), as.factor(d.leukemia.train$Category), positive
svm.traincomfg2$table
```

```
##              Reference
## Prediction    Non-Relapse Relapse
##   Non-Relapse          30      15
##   Relapse               0       0
```

The training misclassification rate on the training set is 0.333. The low value of γ makes the boundary too smooth. Therefore the boundary encapsulates all the data points, and assigns them to the most common class, which is "Non-relapse".

Using the model on the test set, one gets the following:

```
svm.testerrg2 = mean(svm.predtestg2 != d.leukemia.test$Category)
svm.testcomfg2 = confusionMatrix(as.factor(svm.predtestg2), as.factor(d.leukemia.test$Category), positive = "
svm.testcomfg2$table
```

```
##              Reference
## Prediction    Non-Relapse Relapse
##   Non-Relapse           9       6
##   Relapse               0       0
```

The test misclassification rate is 0.4. This is consistent with the result from the training, where the boundary encapsulated all of the training data points. Therefore it is expected that all or most of the test data will be assigned to the class which is most common, in accordance with the training data.

```
model <- c("linear kernel", "radial kernel, high gamma", "radial kernel, low gamma")
training.mr <- c(svm.trainerr,svm.trainerrg1,svm.trainerrg2)
test.mr <- c(svm.testerr,svm.testerrg1, svm.testerrg2)
data.frame(model,training.mr , test.mr)
```

```
##                        model training.mr   test.mr
## 1             linear kernel   0.0000000 0.3333333
## 2 radial kernel, high gamma   0.0000000 0.4000000
## 3  radial kernel, low gamma   0.3333333 0.4000000
```

From the table above, one observes that both the model with linear kernel and the model with radial kernel and high γ manage to fully separate the training data, whereas the model with radial kernel and low γ does not achieve this. However, when observing the test errors of the models, one sees that both models using a radial kernel performs equally poorly. The model with a linear kernel outperforms them both. This implies that a radial kernel is not appropriate for this problem, and that one should use a model with a linear kernel.

Problem 4c)

Given the polynomial kernel on the following form

$$K(\mathbf{x}, \mathbf{y}) = (1 + \sum_{i=1}^{p} x_i y_i)^d$$

along with the input-vector $\mathbf{x} = [x_1, x_2]^T$ and $\mathbf{y} = [y_1, y_2]^T$. One wants to find an appropriate 6-dimensional function $h(\mathbf{x}) = [h_1(\mathbf{x}), ..., h_6(\mathbf{x}))]^T$ so that one can represent the above kernel with $d = 2$ as

$$K(\mathbf{x}, \mathbf{y}) = \langle h(\mathbf{x}), h(\mathbf{y}) \rangle.$$

One can begin by simply multiplying out the kernel for the given $d, \mathbf{x}, \mathbf{y}$. Since $\mathbf{x}$ and $\mathbf{y}$ are 2-dimensional, one requires that $p = 2$. Thus one obtains

$$K(\mathbf{x}, \mathbf{y}) = (1 + \sum_{i=1}^{2} x_i y_i)^2 = 1^2 + 2\sum_{i=1}^{2} x_i y_i + (\sum_{i=1}^{2} x_i y_i)^2$$

$$= 1^2 + \sum_{i=1}^{2} (\sqrt{2}x_i)(\sqrt{2}y_i) + (x_1^2)(y_1^2) + (x_2^2)(y_2^2) + (\sqrt{2}x_1 x_2)(\sqrt{2}y_1 y_2).$$

For the above equation to be true, it is required that the components of the 6-dimensional function $h(\mathbf{x})$ are

$$h_1(\mathbf{x}) = 1,\ h_2(\mathbf{x}) = \sqrt{2}x_1,\ h_3(\mathbf{x}) = \sqrt{2}x_2,\ h_4(\mathbf{x}) = x_1^2,\ h_5(\mathbf{x}) = x_2^2,\ h_6(\mathbf{x}) = \sqrt{2}x_1 x_2.$$

What has been shown is that by enlarging the feature space using the derived $h(\mathbf{x})$, the polynomial kernel function can be expressed as an inner product, which costs less to calculate.

Problem 5

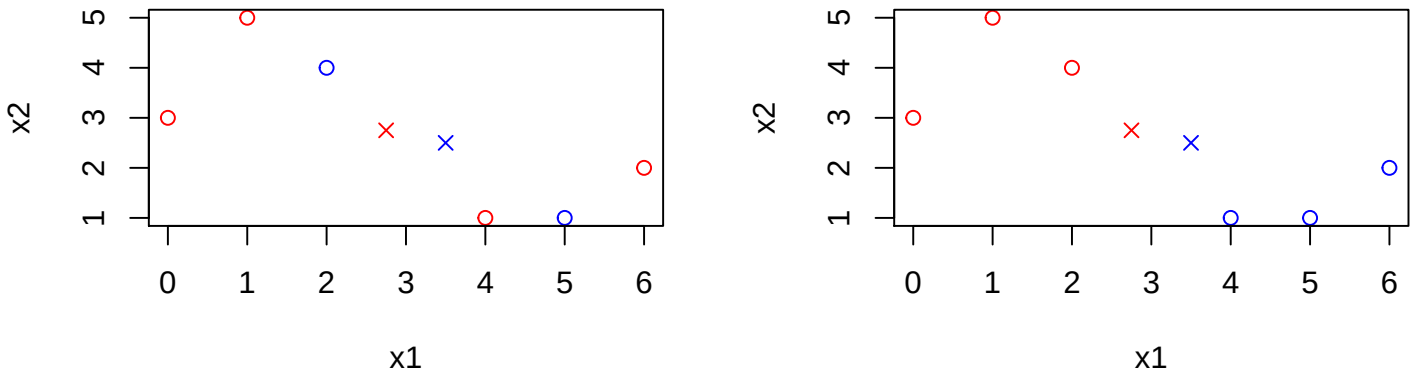
Problem 5a)

True, False, False, False

Problem 5b)

```r
#Data
x1 <- c(1, 2, 0, 4, 5, 6)
x2 <- c(5, 4, 3, 1, 1, 2)

#We have two klusters, K=2
d.dataKlust = data.frame(x1, x2) #creates a dataframe s.t. all is combined

#Randomly assigns each point to a cluster and plots the points with the color describing their cluster (red =
assign_to_cluster_initial <- function(d.data, K=2){
  n=length(d.data[,1])
  clusterByIndices <- sample(1:K, n, replace=TRUE) #Samples randomly to different clusters by indices
  return(clusterByIndices)
}
#Assigns the points to their nearest cluster. Returns a list of 1,2 where index is the index of the point and
assign_to_cluster <- function(d.data, cluster1, cluster2){

  #Calculating the distance from cluster1 to the points
  difference_cluster1_squared <- t(apply(d.dataKlust, 1, function(x){
    return( (x-cluster1)**2)}))
  distance_to_cluster1 <- sqrt(difference_cluster1_squared[,1]+
                               difference_cluster1_squared[,2])

  #Calculating the distance from cluster2 to the points
  difference_cluster2_squared <- t(apply(d.dataKlust, 1, function(x){
    return( (x-cluster2)**2)}))
  distance_to_cluster2 <- sqrt(difference_cluster2_squared[,1]+
                               difference_cluster2_squared[,2])

  #Assigns points to their respective cluster
  clusterByIndices <- as.numeric(distance_to_cluster2<distance_to_cluster1)+1

  return(clusterByIndices)
}
set.seed(1)

#Initializing random clusters to the points
clusterByIndices = assign_to_cluster_initial(d.dataKlust)

#Calculating the cluster centroids from the initial cluster-assignments of the points
cluster1 <- c(
  mean(x1[clusterByIndices==1]),
  mean(x2[clusterByIndices==1]))

cluster2 <- c(
  mean(x1[clusterByIndices==2]),
  mean(x2[clusterByIndices==2]))

par(mfrow = c(1,2))
#Plotting the initial clusterassignments and the
plot(d.dataKlust, col = c("Red", "Blue")[clusterByIndices])
points(cluster1[1],cluster1[2], col="Red", pch=4)
```

```
points(cluster2[1],cluster2[2], col="Blue", pch=4)

#Finding the new cluster-assignments of the points
clusterByIndices <- assign_to_cluster(d.data, cluster1, cluster2)

plot(d.dataKlust, col = c("Red", "Blue")[clusterByIndices])
points(cluster1[1],cluster1[2], col="Red", pch=4)
points(cluster2[1],cluster2[2], col="Blue", pch=4)
```

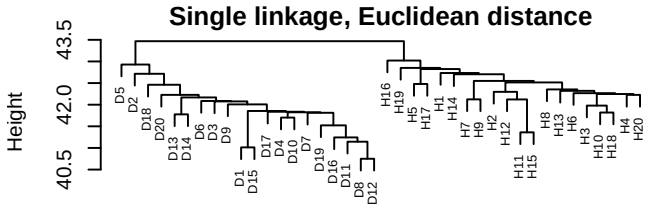


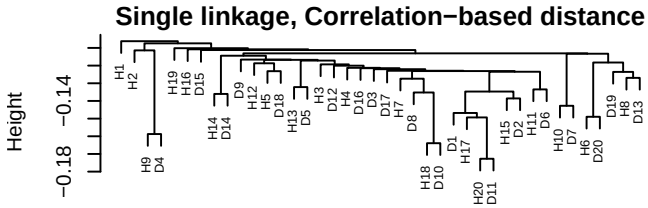
In the plots above, circular points represents data points, and crosses represents the centroids.

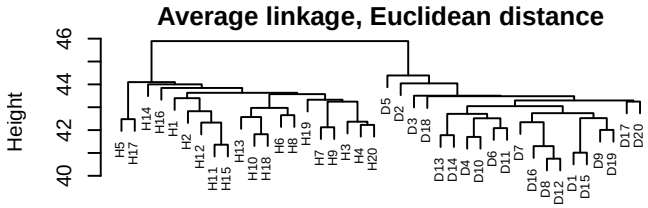
Problem 5c)

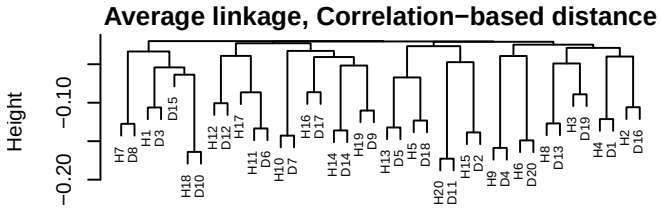
```
#40 tissue samples with measurements of 1,000 genes
#The first 20 tissues come from healthy patients and the remaining 20 come from a diseased patient group
id <- "1VfVCQvWt121UN39NXZ4aR9Dmsbj-p9OU" # google file ID
GeneData <- read.csv(sprintf("https://docs.google.com/uc?id=%s&export=download",
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)
GeneData <- scale(GeneData)

par(mfrow=c(3,2), mai=c(0.2, 0.5, 0.2, 0.5))
#Single linkage, Euclidean distance
sled <- hclust(dist(GeneData), method = "single")
plot(sled, cex=0.6, main = "Single linkage, Euclidean distance")

#Single linkage, Correlation-based distance
slcd <- hclust(as.dist(cor(t(GeneData), use = "pairwise.complete.obs",
                           method = "pearson")), method = "single")
plot(slcd, cex=0.6, main = "Single linkage, Correlation-based distance")

#Average linkage, Euclidean distance
aled <- hclust(dist(GeneData), method = "average")
plot(aled, cex=0.6, main="Average linkage, Euclidean distance")

#Average linkage, Correlation-based distance
alcd <- hclust(as.dist(cor(t(GeneData),use = "pairwise.complete.obs",
                           method = "pearson" )), method = "average")
plot(alcd, cex=0.6, main = "Average linkage, Correlation-based distance")

#Complete linkage, Euclidean distance
cled <- hclust(dist(GeneData), method="complete")
```

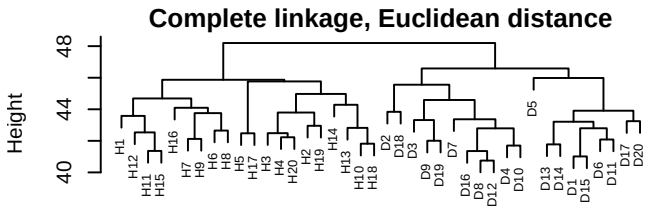
```r
plot(cled, cex=0.6, main = "Complete linkage, Euclidean distance")

#Complete linkage, Correlation-based distance
clcd <- hclust(as.dist(cor(t(GeneData),use = "pairwise.complete.obs",
                           method = "pearson" )), method="complete")

plot(clcd, cex=0.6, main = "Complete linkage, Correlation-based distance")
```

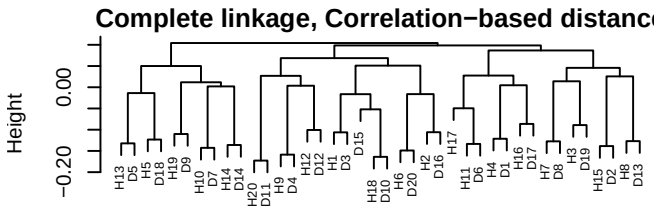






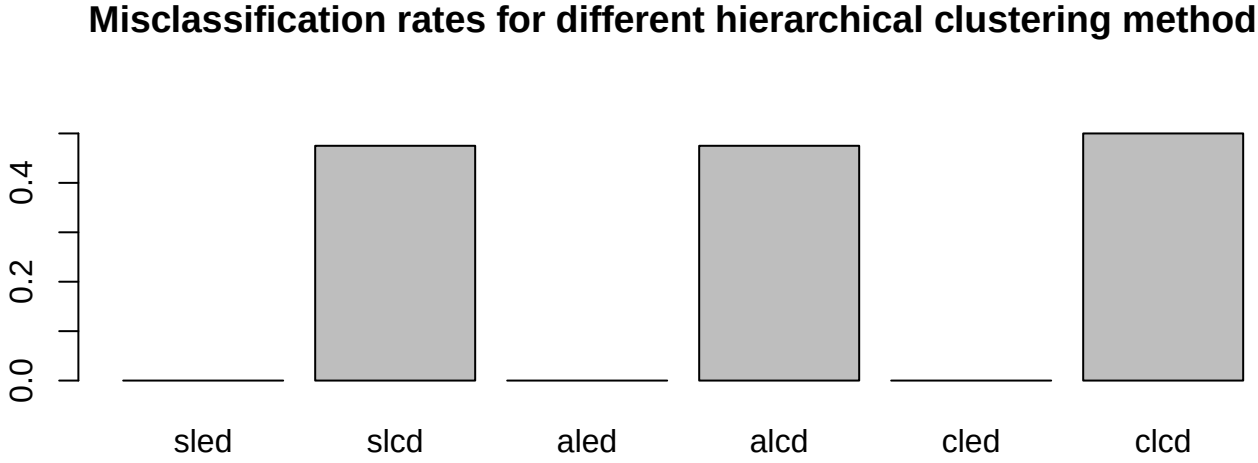






Problem 5d)

```r
#Hierarchical clustering finds clusters, but doesn't give these choose the cluster value based on the respons
solution_1 <- c(rep(1,20),rep(2,20))
solution_2 <- c(rep(2,20),rep(1,20))

cut_sled <- cutree(sled, k=2)
cut_slcd <- cutree(slcd, k=2)
cut_aled <- cutree(aled, k=2)
cut_alcd <- cutree(alcd, k=2)
cut_cled <- cutree(cled, k=2)
cut_clcd <- cutree(clcd, k=2)

cuts_list <- list(cut_sled, cut_slcd, cut_aled, cut_alcd, cut_cled, cut_clcd)
misclassification_rate = sapply(cuts_list, function(x){
  return(min(mean(abs(x-solution_1)),mean(abs(x-solution_2))))})

par(mai=c(0.5, 0.5, 1.2, 0.2))
barplot(misclassification_rate, main="Misclassification rates for different hierarchical clustering methods",
```



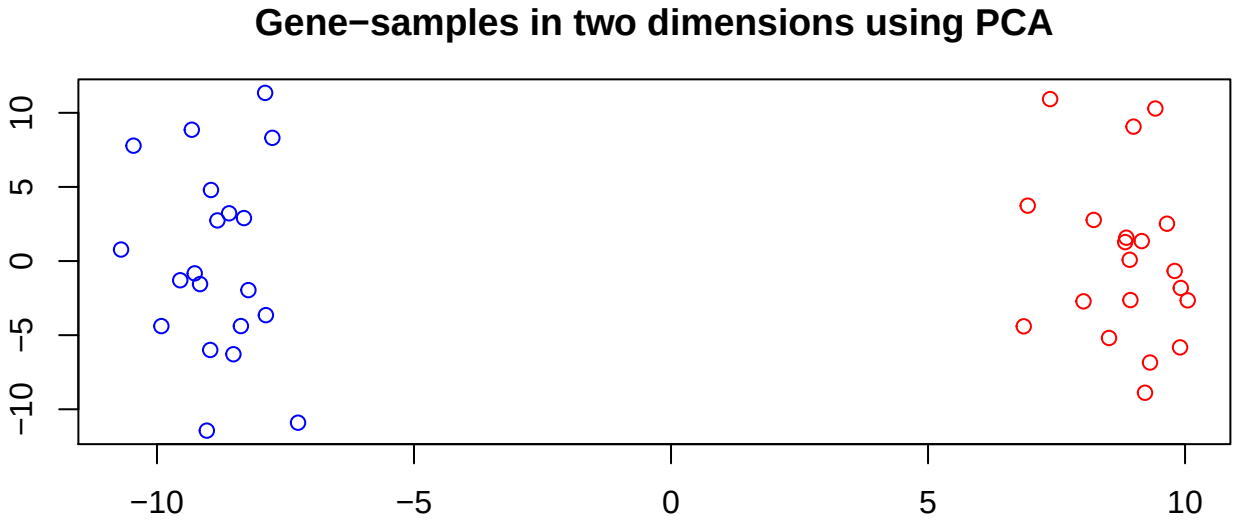
Above one can see a bar plot for the rate of which the gene-samples was clustered in the correct group, for simplicity this is called misclassification rate, for different linkages and distance-functions. The two first characters describes which linkage is used in the model, where "sl" is "single linkage", "al" is "average linkage" and "cl" is "complete linkage". The two last characters describes which distance function is used, where "ed" is "euclidean distance" and "cd" is "correlation-based distance". Then the tree is cut into two clusters.

Clearly all the clusters made from hierarchical clustering using euclidean distance correctly placed healthy and diseased tissues in their repsective clusters, when the true state of the tissue is known. The linkage had no apparent effect as the misclassification rate was zero for all euclidean distance-based models. The clusters made using correlation-based distance on the other hand gave large misclassification rates for all linkages.

Problem 5e)

i)

```
pca <- prcomp(GeneData, center=T)

data_pc1 <- GeneData%*% pca$rotation[,1]
data_pc2 <- GeneData%*% pca$rotation[,2]

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

par(mai=c(0.5, 0.5, 0.6, 0.5))
plot(data_pc1, data_pc2, col=c("blue","red")[color],main="Gene-samples in two dimensions using PCA", xlab="PC
```



ii)

```
eig_val <- get_eigenvalue(pca) #From the factoextra library
var_exp_5first <- sum(eig_val$eigenvalue[1:5])/
```

```
  sum(eig_val$eigenvalue)
var_exp_5first
```

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

The fraction of the total variance explained by the five first principal components is given by:

$$R^2 = \frac{\sum_{i=1}^{5} \lambda_i}{\sum_{i=1}^{40} \lambda_i}.$$

The above code calculates this fraction and yields 0.211. Hence, 21.1% of the variance is explained by the five first principal components.

Problem 5f)

The plot in 5e)(i) clearly shows that the first principal component perfectly separates the two groups of gene-samples. It is therefore plausible to assume that the covariates with the biggest coefficients in PC1 are the genes that wary the most across the two groups.

```
largest_coeff_gene <- which.max(abs(pca$rotation[,1]))
largest_coeff <- pca$rotation[largest_coeff_gene,1]

num_relatively_big_coeff <- sum( abs(pca$rotation[,1])>
                                   0.9*largest_coeff )

relatively_big_coeff <- c(1:1000)[abs(pca$rotation[,1])>
                                   0.9*largest_coeff ]
```

The largest coefficient in PC1 belongs to gene 502 and is 0.0948504. It therefore seems like this gene varies the most across the two groups. The number of coefficients that are maximum 10% smaller than the largest coefficient is 21. Since so many genes have such similar coefficient in PC1, it is very difficult to choose a specific subset of genes that are the most important. For the sake of illustration, here are every gene with coefficients which are maximum 10% smaller than the largest coefficient. These genes are: 502, 508, 509, 511, 528, 535, 538, 540, 551, 554, 564, 565, 566, 570, 584, 589, 590, 592, 593, 599, 600.