Compulsory exercise 2: Group 4

TMA4268 Statistical Learning V2021

Øystein Alvestad, Lars Evje and William Scott Grundeland Olsen

16 april, 2021

Problem 1

a)

b)

c)

d)

e)

f)

Problem 2

a)

b)

c)

(i)

(ii)

Problem 3

a)

True, True, False, False

b)

c)

(i)

```r
id <- "1Fv6xwKLSZHldRAC1MrcK2mzdOYnbgv0E"  # google file ID
d.diabetes <- dget(sprintf("https://docs.google.com/uc?id=%s&export=download", id))

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

d.test = d.diabetes$ctest
d.test$diabetes = as.factor(d.test$diabetes)
```

```r
set.seed(1)
t.diabetes = tree(diabetes ~ ., data = d.train)

t.diabetes.pred = predict(t.diabetes, d.test, type = "class")
misclass = table(t.diabetes.pred, d.test$diabetes)
misclass
```

```
##
## t.diabetes.pred   0   1
##               0 126  28
##               1  29  49
```

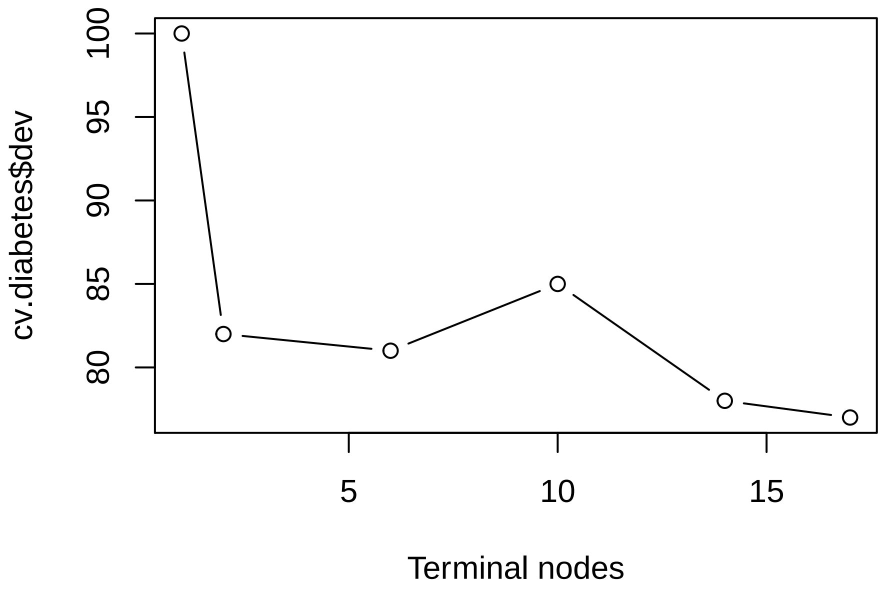
```r
1 - sum(diag(misclass)/sum(misclass))
```

```
## [1] 0.2456897
```

```r
cv.diabetes = cv.tree(t.diabetes, FUN = prune.misclass, K = 10)
cv.diabetes
```

```
## $size
## [1] 17 14 10  6  2  1
##
## $dev
## [1]  77  78  85  81  82 100
##
## $k
## [1]  -Inf  0.00  1.50  2.75  5.00 29.00
##
## $method
## [1] "misclass"
##
## attr(,"class")
## [1] "prune"         "tree.sequence"
```

```r
plot(cv.diabetes$size, cv.diabetes$dev, type = "b", xlab = "Terminal nodes")
```



```r
prune.diabetes = prune.misclass(t.diabetes, best = 3)
# plot(prune.diabetes) text(prune.diabetes, pretty = 1)

prune.diabetes.pred = predict(prune.diabetes, d.test, type = "class")
misclass.prune = table(prune.diabetes.pred, d.test$diabetes)
misclass.prune
```

```
##
## prune.diabetes.pred   0   1
##                   0 119  20
##                   1  36  57
```

```r
1 - sum(diag(misclass.prune)/sum(misclass.prune))
```

```
## [1] 0.2413793
```

The 10-fold CV cost-complexity, with FUN = prune.misclass tells us to choose the full tree. Using FUN = deviance however gives us that we should choose the three with 3 terminal nodes. The misclassification error for the full tree is 0.2457, while for the tree with three termal nodes it is 0.2414 so slightly better on the test set.

(ii)

```
library(randomForest)
set.seed(1)
rf.diabetes = randomForest(diabetes ~ ., data = d.train, ntree = 1000, mtry = 3,
    importance = TRUE)

rf.diabetes.pred = predict(rf.diabetes, d.test, type = "class")
misclass.rf = table(rf.diabetes.pred, d.test$diabetes)
misclass.rf
```

```
##
## rf.diabetes.pred   0   1
##                0 135  34
##                1  20  43
```

```
1 - sum(diag(misclass.rf)/sum(misclass.rf))
```

```
## [1] 0.2327586
```

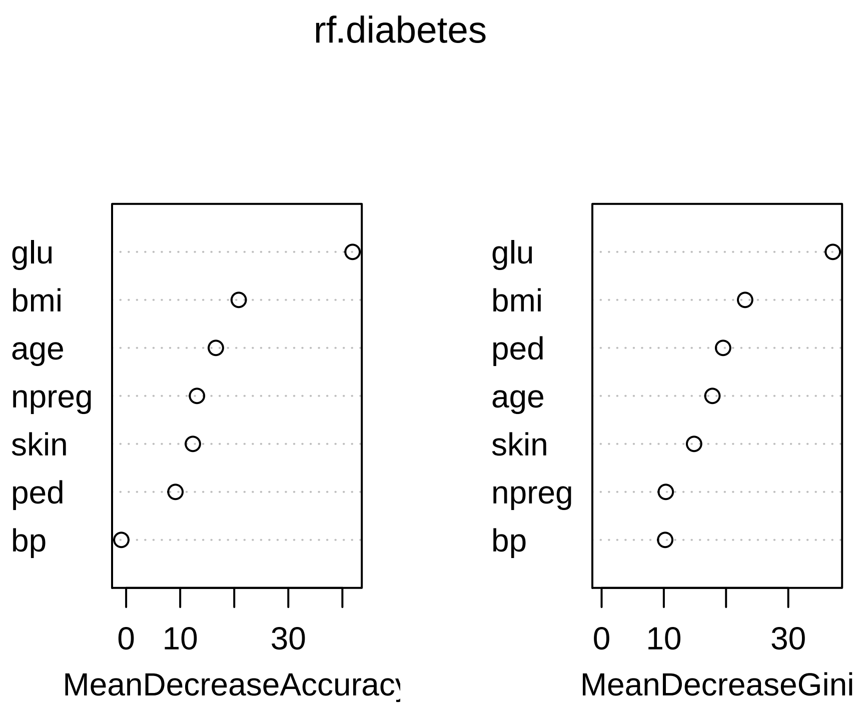
```
rf.diabetes
```

```
##
## Call:
##  randomForest(formula = diabetes ~ ., data = d.train, ntree = 1000,      mtry = 3, importance = TRUE
##                Type of random forest: classification
##                      Number of trees: 1000
## No. of variables tried at each split: 3
##
##         OOB estimate of  error rate: 20%
## Confusion matrix:
##     0  1 class.error
## 0 175 25       0.125
## 1  35 65       0.350
```

```
importance(rf.diabetes)
```

```
##               0          1 MeanDecreaseAccuracy MeanDecreaseGini
## npreg 14.968249  0.9760653           13.1023756         10.31516
## glu   33.655803 29.8766042           41.8842028         37.16251
## bp     1.995766 -3.5518898           -0.8901241         10.22150
## skin  10.536315  5.6018397           12.3438407         14.86193
## bmi   15.489014 14.4753432           20.8293113         23.06560
## ped    7.620861  4.8751738            9.1223090         19.52611
## age   15.402886  5.9624002           16.5960946         17.81053
```

```
varImpPlot(rf.diabetes)
```



We use the more advanced method Random Forest, with the tuning parameters ntree=1000, and mtry=3. From the importance plots we observe that glu and bmi is the most influential variables in the prediction of diabetes.

Problem 4

a)

False, True, False, True.

b)

(i)

A SVM tends to behave better than logistic regression when the classes are well separated, which we would assume them to be here with the genomic data. Also SVM has the ability to work in high dimensional space compared to the small number of samples.

Instead of SVM one could for example use a random forest, K-nearest neighbor, linear classifiers, or quadratic classifiers.

(ii)

The paper introduces an ensemble SVM-Recursive Feature Elimination for gene selection that follows the concept of ensemble and bagging used in random forest but adopts the backward elimination strategy which is the rationale of Recursive Feature Elimination algorithm.

(iii)

In the following code block we fit a support vector classifier with $C = 1$ on `Category` using `d.leukemia.train`.

```
set.seed(2399)

# Set-up:
id <- "1x_E8xnmz9CMHh_tMwIsWP94czPa1Fpsj"  # Google file ID
path <- "https://docs.google.com/uc?id=%s&export=download"
d.leukemia <- read.csv(sprintf(path, id), header = TRUE)

t.samples <- sample(1:60, 15, replace = FALSE)
d.leukemia$Category <- as.factor(d.leukemia$Category)
d.leukemia.test <- d.leukemia[t.samples, ]
d.leukemia.train <- d.leukemia[-t.samples, ]

# Support vector classifier:
svmfit <- svm(Category ~ ., data = d.leukemia.train, kernel = "linear", cost = 1,
    scale = TRUE)
pred_train <- predict(svmfit, d.leukemia.train)
pred_test <- predict(svmfit, d.leukemia.test)

# Confusion tables for training and testing respectively:
conf_tab_train <- table(predict = pred_train, truth = d.leukemia.train$Category)
conf_tab_test <- table(predict = pred_test, truth = d.leukemia.test$Category)

# Misclassification for training and testing respectively:
misclas_train <- 1 - sum(diag(conf_tab_train))/sum(conf_tab_train)
misclas_test <- 1 - sum(diag(conf_tab_test))/sum(conf_tab_test)
```

We then see the confusion table for the training data below, with the misclassification error rate of 0.

```
conf_tab_train
```

```
##               truth
## predict        Non-Relapse Relapse
##   Non-Relapse           30       0
##   Relapse                0      15
```

We also have the confusion table for the test data below, with the misclassification error rate of 0.3333333.

```
conf_tab_test
```

```
##               truth
## predict        Non-Relapse Relapse
##   Non-Relapse            8       4
##   Relapse                1       2
```

The training error rate is 0, suggesting that there is an overfitting of the data. This is dependent on the cost C, and one could have done a cross validation to find a possibly better cost than $C = 1$.

The most common error in the test set is that the truth is relapse, while the prediction is non-relapse. That is, children relapse even though the prediction is that they do not. With a misclassification error rate of 0.3333333 for the test set the classification can be said to be successful. However, the false positive, which is the most common error, is worse than the false negative in this case, in our opinion.

(iv)

In the following code block we fit a support vector machine to the data using the cost $C = 1$ and the tuning parameter $\gamma = 10^{-2}$ or $\gamma = 10^{-5}$.

```
set.seed(2399)

# Support vector machine and prediction:
svmfit_gamma1 <- svm(Category ~ ., data = d.leukemia.train, kernel = "radial", cost = 1,
    gamma = 0.01, scale = TRUE)
svmfit_gamma2 <- svm(Category ~ ., data = d.leukemia.train, kernel = "radial", cost = 1,
    gamma = 1e-05, scale = TRUE)
pred_train_gamma1 <- predict(svmfit_gamma1, d.leukemia.train)
pred_test_gamma1 <- predict(svmfit_gamma1, d.leukemia.test)
pred_train_gamma2 <- predict(svmfit_gamma2, d.leukemia.train)
pred_test_gamma2 <- predict(svmfit_gamma2, d.leukemia.test)

# Confusion tables for training and testing:
conf_tab_train_gamma1 <- table(predict = pred_train_gamma1, truth = d.leukemia.train$Category)
conf_tab_test_gamma1 <- table(predict = pred_test_gamma1, truth = d.leukemia.test$Category)
conf_tab_train_gamma2 <- table(predict = pred_train_gamma2, truth = d.leukemia.train$Category)
conf_tab_test_gamma2 <- table(predict = pred_test_gamma2, truth = d.leukemia.test$Category)

# Misclassification for training and testing:
misclas_train_gamma1 <- 1 - sum(diag(conf_tab_train_gamma1))/sum(conf_tab_train_gamma1)
misclas_test_gamma1 <- 1 - sum(diag(conf_tab_test_gamma1))/sum(conf_tab_test_gamma1)
misclas_train_gamma2 <- 1 - sum(diag(conf_tab_train_gamma2))/sum(conf_tab_train_gamma2)
misclas_test_gamma2 <- 1 - sum(diag(conf_tab_test_gamma2))/sum(conf_tab_test_gamma2)
```

We then see the confusion table for the training data for $\gamma = 10^{-2}$ below, with the misclassification error rate of 0.

```
conf_tab_train_gamma1
```

```
##               truth
## predict       Non-Relapse Relapse
##   Non-Relapse          30       0
##   Relapse               0      15
```

We also have the confusion table for the test data for $\gamma = 10^{-2}$ below, with the misclassification error rate of 0.4.

```
conf_tab_test_gamma1
```

```
##               truth
## predict       Non-Relapse Relapse
##   Non-Relapse           9       6
##   Relapse               0       0
```

For $\gamma = 10^{-5}$ er have the confusion table for the training data below, with the misclassification error rate of 0.3333333.

```
conf_tab_train_gamma2
```

```
##               truth
## predict        Non-Relapse Relapse
##   Non-Relapse           30      15
##   Relapse                0       0
```

We also have the confusion table for the test data for $\gamma = 10^{-5}$ below, with the misclassification error rate of 0.4.

```
conf_tab_test_gamma2
```

```
##               truth
## predict        Non-Relapse Relapse
##   Non-Relapse            9       6
##   Relapse                0       0
```

We note that the misclassification error rate for the training set is 0 for $\gamma = 10^{-2}$ and 0.3333333 for $\gamma = 10^{-5}$. This can be explained by the fact that for small γ the decision boundaries are smoother than for larger γ. Thus, there may be some overfitting for $\gamma = 10^{-2}$. For the test data however, the results are the same. Comparing to the case in (iii), the results are worse, suggesting that the support vector classifier is better than the support vector machine for this dataset.

c)

The polynomial kernel of positive integer degree d has the form

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

for $\mathbf{x}, \mathbf{y} \in \mathbb{R}^p$, with elements x_i and y_i for $i = 1, \dots, p$. We assume $d = 2$ and $\mathbf{x}, \mathbf{y} \in \mathbb{R}^2$, such that

$$\begin{aligned} K(\mathbf{x}, \mathbf{y}) &= (1 + \mathbf{x}^\top \mathbf{y})^2 = 1 + 2\mathbf{x}^\top \mathbf{y} + (\mathbf{x}^\top \mathbf{y})^2 = 1 + 2(x_1 y_1 + x_2 y_2) + (x_1 y_1 + x_2 y_2)^2 \\ &= 1 + 2x_1 y_1 + 2x_2 y_2 + x_1^2 y_1^2 + x_2^2 y_2^2 + 2x_1 y_1 x_2 y_2. \end{aligned}$$

We then see that

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

by the basic definition of the inner product of two vectors, where,

$$\mathbf{h}(\mathbf{x}) = \begin{bmatrix} 1 \\ \sqrt{2}x_1 \\ \sqrt{2}x_2 \\ x_1^2 \\ x_2^2 \\ \sqrt{2}x_1 x_2 \end{bmatrix} \quad \text{and} \quad \mathbf{h}(\mathbf{y}) = \begin{bmatrix} 1 \\ \sqrt{2}y_1 \\ \sqrt{2}y_2 \\ y_1^2 \\ y_2^2 \\ \sqrt{2}y_1 y_2 \end{bmatrix}.$$

Problem 5

a)

True, False, False, False.

b)

In the following we make a random cluster of the data and compute the centroid of the two clusters we get. The clusters are color coded where one cluster is colored red, while the other is green, as seen in Figure 1. Note that the code here is not general for every K-mean clustering, but is only applicable to $K = 2$, which is the case given in the problem.

```r
set.seed(1)

x1 <- c(1, 2, 0, 4, 5, 6)
x2 <- c(5, 4, 3, 1, 1, 2)
X <- matrix(c(x1, x2), ncol = 2)

# Random cluster
X_cluster <- cbind(X, sample(c(1, 2), size = nrow(X), replace = TRUE))

# Initializing and computing the centroids:
g1_centroid <- c(0, 0)
g2_centroid <- c(0, 0)

for (i in 1:length(x1)) {
    if (X_cluster[i, 3] == 1) {
        g1_centroid[1] <- g1_centroid[1] + X[i, 1]
        g1_centroid[2] <- g1_centroid[2] + X[i, 2]
    } else {
        g2_centroid[1] <- g2_centroid[1] + X[i, 1]
        g2_centroid[2] <- g2_centroid[2] + X[i, 2]
    }
}

g1_centroid <- g1_centroid/length((X[, 1])[X_cluster[, 3] == 1])
g2_centroid <- g2_centroid/length((X[, 1])[X_cluster[, 3] == 2])


# Plotting the clusters and centroids color coded:
plot(X, col = X_cluster[, 3] + 1, main = "Random custering of the data with the centroids",
    xlab = "x1", ylab = "x2", pch = 20, cex = 2)
points(g1_centroid[1], g1_centroid[2], pch = 15, cex = 2, col = 2)
points(g2_centroid[1], g2_centroid[2], pch = 15, cex = 2, col = 3)
```

We can then measure, using the Euclidean distance, what points are closest to the respective centroids, in this case giving the correct clustering for $K = 2$. This is shown in Figure 2.

```r
dist <- function(x, y) {
    return(sqrt(sum((x - y)^2)))
}
```

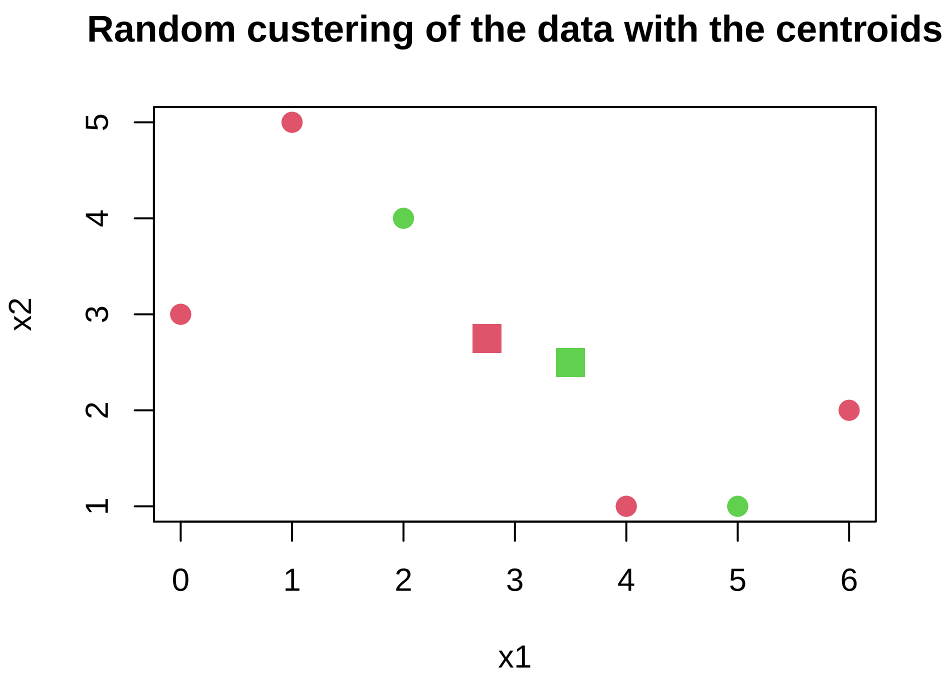


Figure 1: A random clustering of the data being the round points, and the centroids being the square points.

```r
for (i in 1:length(x1)) {
    X_cluster[i, 3] <- ifelse(dist(g1_centroid, X[i, ]) < dist(g2_centroid, X[i,
        ]), 1, 2)
}

plot(X, col = X_cluster[, 3] + 1, main = "K-means custering of the data with K = 2",
    xlab = "x1", ylab = "x2", pch = 20, cex = 2)
```

```r
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)
```

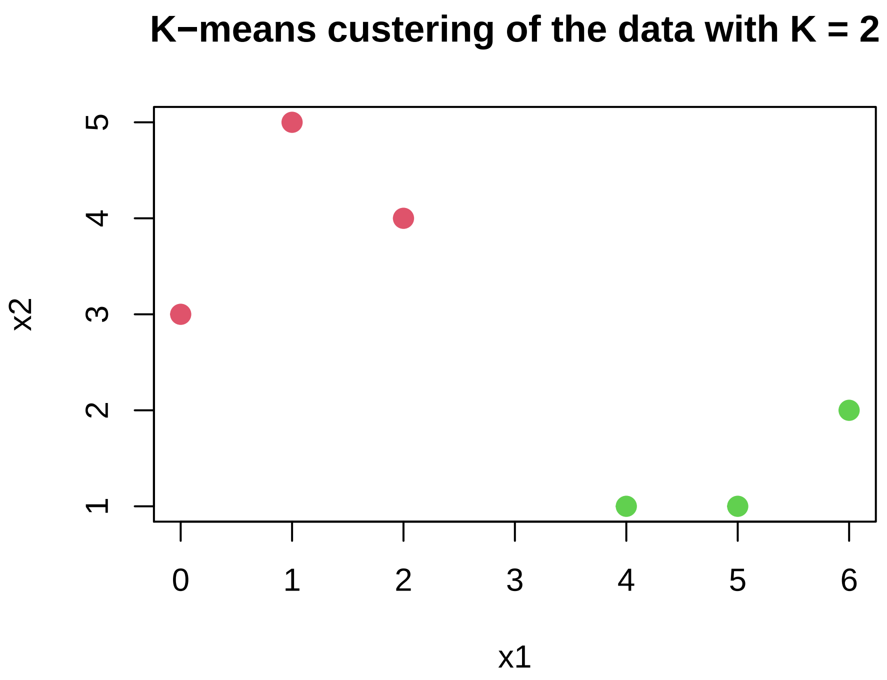
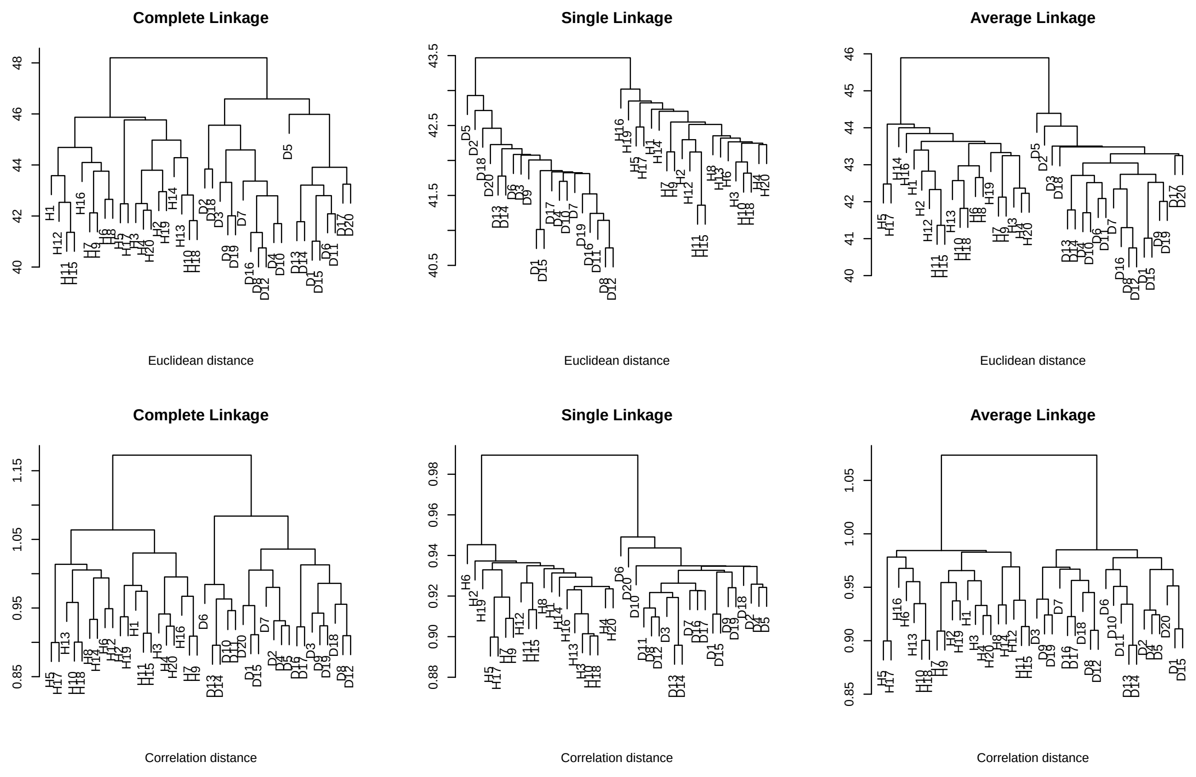


Figure 2: The K-means clustering for the data, with $K = 2$.

c)



d)

```
cutree(hc.Gene.EcC, 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.Gene.EcS, 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.Gene.EcA, 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.Gene.CorrC, 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.Gene.CorrS, 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.Gene.CorrA, 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
```

Since it was given that the 20 first tissue samples was healthy, and the last 20 was damaged we observe that all of the above hierarchical clusterings managed to seperate the two groups perfectly.

e)

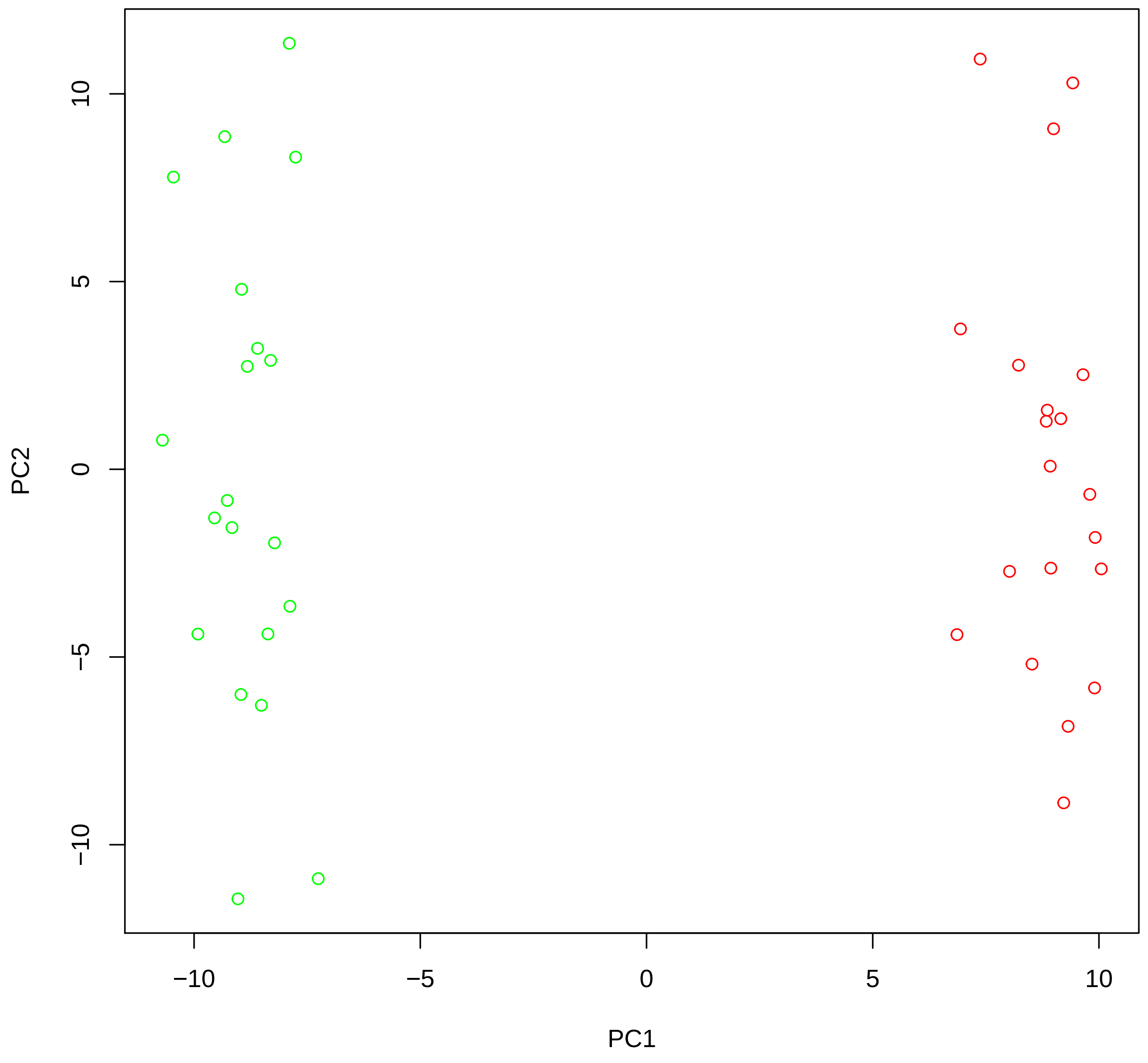
(i)

```
pc.Gene = prcomp(GeneData)
summary(pc.Gene)
```

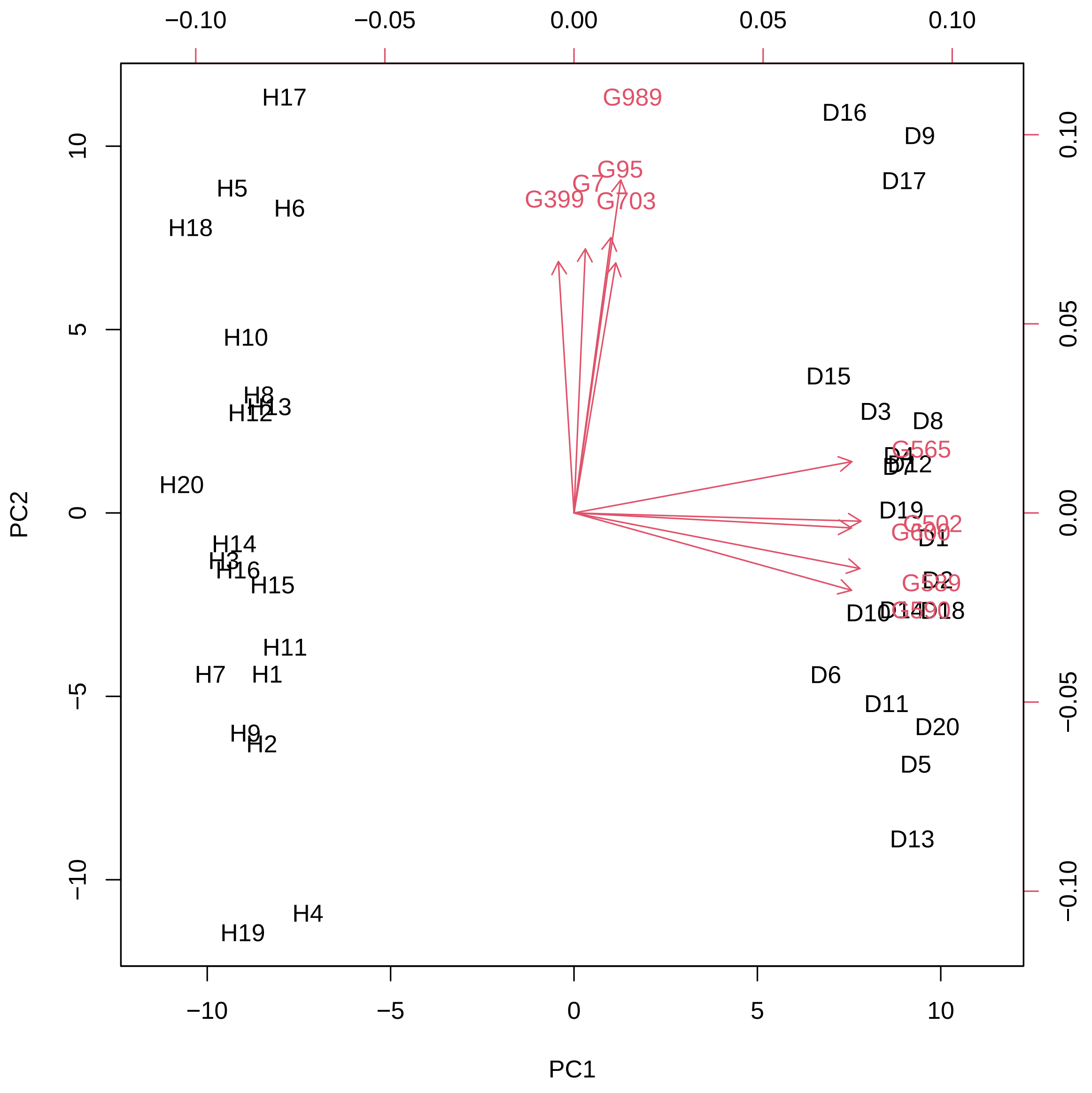
```
## Importance of components:
##                            PC1     PC2     PC3     PC4     PC5     PC6     PC7
## Standard deviation     9.00460 5.87302 5.74347 5.61806 5.55344 5.50107 5.40069
## Proportion of Variance 0.08108 0.03449 0.03299 0.03156 0.03084 0.03026 0.02917
## Cumulative Proportion  0.08108 0.11558 0.14856 0.18013 0.21097 0.24123 0.27040
##                            PC8    PC9    PC10    PC11    PC12    PC13   PC14
## Standard deviation     5.38575 5.3762 5.34146 5.31878 5.25016 5.18737 5.1667
## Proportion of Variance 0.02901 0.0289 0.02853 0.02829 0.02756 0.02691 0.0267
## Cumulative Proportion  0.29940 0.3283 0.35684 0.38513 0.41269 0.43960 0.4663
##                           PC15    PC16    PC17    PC18    PC19    PC20    PC21
## Standard deviation     5.10384 5.04667 5.03288 4.98926 4.92635 4.90996 4.88803
## Proportion of Variance 0.02605 0.02547 0.02533 0.02489 0.02427 0.02411 0.02389
## Cumulative Proportion  0.49234 0.51781 0.54314 0.56803 0.59230 0.61641 0.64030
##                           PC22    PC23    PC24    PC25    PC26    PC27    PC28
## Standard deviation     4.85159 4.79974 4.78202 4.70171 4.66105 4.64595 4.59194
## Proportion of Variance 0.02354 0.02304 0.02287 0.02211 0.02173 0.02158 0.02109
## Cumulative Proportion  0.66384 0.68688 0.70975 0.73185 0.75358 0.77516 0.79625
##                           PC29    PC30   PC31    PC32    PC33   PC34    PC35
## Standard deviation     4.53246 4.47381 4.4389 4.41670 4.39404 4.3591 4.23504
## Proportion of Variance 0.02054 0.02001 0.0197 0.01951 0.01931 0.0190 0.01794
## Cumulative Proportion  0.81679 0.83681 0.8565 0.87602 0.89533 0.9143 0.93226
##                          PC36    PC37   PC38    PC39    PC40
## Standard deviation     4.2184 4.12936 4.0738 4.03658 5.5e-15
## Proportion of Variance 0.0178 0.01705 0.0166 0.01629 0.0e+00
## Cumulative Proportion  0.9501 0.96711 0.9837 1.00000 1.0e+00
```

```
cols = c(rep("green", 20), rep("red", 20))


plot(pc.Gene$x[, 1:2], col = cols)
```



```
GeneLoad = pc.Gene$rotation[, 1:2]
informative_loadings = rbind(GeneLoad[order(GeneLoad[, 1], decreasing = TRUE), ][1:5,
    ], GeneLoad[order(GeneLoad[, 2], decreasing = TRUE), ][1:5, ])
biplot(x = pc.Gene$x[, 1:2], y = informative_loadings, scale = 0)
```



```r
tail(sort(abs(pc.Gene$rotation[, 1])), 20)
```

```
##        G508       G540       G564       G566       G592       G528       G535
## 0.08541371 0.08550741 0.08552071 0.08553015 0.08558608 0.08561425 0.08610094
##        G599       G570       G511       G509       G584       G538       G593
## 0.08624312 0.08626458 0.08655126 0.08661015 0.08690858 0.08745400 0.08758616
##        G551       G600       G590       G565       G589       G502
## 0.08768360 0.09167322 0.09173169 0.09183823 0.09449766 0.09485044
```

(ii)

From the summary above we read of that the first 5 principle components explain around 21.1% of the variance.

f)

From the plot above we see that the two groups can be separated by looking at the first principle component alone in this case, in addition by the properties of PCA we know that PC1 capture most of the variance (out of any PC). Therefore we look at the genes with the highest loadings in PC1, which gives us that