Compulsory exercise 2: Group 12

TMA4268 Statistical Learning V2020

Emma Skarnes, Håkon Noren and Alexander Johan Arntzen

08 April, 2020

Problem 1

a)

From the textbook we know that the ridge regression estimator $\hat{\beta}_{Ridge}$ is given by the optimization problem

$$\min_{\hat{\beta}_0 \in \mathbb{R}, \hat{\beta}_{Ridge} \in \mathbb{R}^p} (\left\| y - \hat{\beta}_0 \mathbf{1} - X\hat{\beta}_{Ridge} \right\|_2^2 + \lambda \left\| \hat{\beta}_{Ridge} \right\|_2^2),$$

where X is the data matrix, y is the measured response and $\hat{\beta}_0 = \overline{y}$ is the intercept estimate. As $\hat{\beta}_0$ is known, we can center the columns of X and y. This results in the equivalent problem

$$\min_{\beta \in \mathbb{R}^p} (\|y - X\beta\|_2^2 + \lambda \|\beta\|_2^2),$$

where $\beta := \hat{\beta}_{Ridge}$ for ease of readability. The function to be minimized is a non-negative polynomial. Therefore it's minimal value exists and is a singular value. Thus $\hat{\beta}_{Ridge}$ must be such that

$$\frac{\partial}{\partial \beta}(\|y - X\beta\|_2^2 + \lambda \|\beta\|_2^2) = 0.$$

Calculation then yields

$$\begin{aligned} \frac{\partial}{\partial \beta}(\|y - X\beta\|^2 + \lambda \|\beta\|^2) &= \frac{\partial}{\partial \beta}[y^T y + y^T X\beta + (X\beta)^T y + (X\beta)^T X\beta + \lambda \beta^T I \beta] \\ &= -2X^T y + 2X^T X\beta + 2\lambda I\beta \end{aligned}.$$

Then inserting into the previous equation we have

$$\begin{gathered} 2X^T y + 2X^T X\beta + 2\lambda I\beta = 0 \\ \Downarrow \\ (X^T X + \lambda I)\beta = X^T y. \end{gathered}$$

Then if λ is large enough $(X^T X + \lambda I)$ will be inevitable, yielding

$$\hat{\beta}_{Ridge} = (X^T X + \lambda I)^{-1} X^T y.$$

b)

Firstly, we have by definition of the linear model that

$$\mathrm{E}[y] = \mathrm{E}[X\beta + \epsilon] = X\beta + \mathrm{E}[\epsilon] = X\beta,$$

and

$$\text{Var}[y] = \text{Var}[X\beta + \epsilon] = \text{Var}[\epsilon] = \sigma^2 I.$$

Then the expected value of $\hat{\beta}_{Ridge}$ is

$$\begin{aligned}\text{E}[\hat{\beta}_{Ridge}] &= (X^T X + \lambda I)^{-1} X^T \text{E}[y] \\ &= \boldsymbol{(X^T X + \lambda I)^{-1} X^T X \beta}.\end{aligned}$$

The covariance matrix for $\hat{\beta}_{Ridge}$ is similarly

$$\begin{aligned}\text{Var}[\hat{\beta}_{Ridge}] &= (X^T X + \lambda I)^{-1} X^T) \text{Var}[y] (X^T X + \lambda I)^{-1} X^T)^T \\ &= (X^T X + \lambda I)^{-1} X^T \sigma^2 I X (X^T X + \lambda I)^{-1} \\ &= \boldsymbol{\sigma^2 (X^T X + \lambda I)^{-1} X^T X (X^T X + \lambda I)^{-1}}\end{aligned}$$

c)

TRUE, FALSE, FALSE, TRUE

d)

```
set.seed(1)
train.ind = sample(1:nrow(College), 0.5 * nrow(College))

college.train = College[train.ind, ]
college.test = College[-train.ind, ]
selection.forward <- regsubsets(Outstate ~ ., data = college.train, method = "forward",
    nvmax = 17)

# X matrix for test data and training data
test.mat = model.matrix(Outstate ~ ., data = college.test)
train.mat = model.matrix(Outstate ~ ., data = college.train)

# Choosing model
numCoeffs = which.min(summary(selection.forward)$bic)
modelCoeffs <- coef(selection.forward, id = numCoeffs)

# Calculating test MSE
pred = test.mat[, names(modelCoeffs)] %*% modelCoeffs
test_MSE_Forward = mean((college.test$Outstate - pred)^2)
```

Using the `regsubsets` forward selection is performed on the test data set. The model with the last BIC is chosen as this penalizes models with more variables. This model has 6 coefficients , excluding the intercept, a test MSE of 3.8448572×10^6 and the following coefficients:

```
modelCoeffs
```

```
##   (Intercept)    PrivateYes    Room.Board      Terminal   perc.alumni
## -4726.8810613  2717.7019276     1.1032433    36.9990286    59.0863753
##        Expend     Grad.Rate
##     0.1930814    33.8303314
```

Note that we have not trained the model on the full dataset in order to compare the test MSE.

e)

```r
# Calulate the best lambda using 10-fold CV
cv.out = cv.glmnet(x = train.mat[, 2:18], y = college.train$Outstate, alpha = 1)
bestlam = cv.out$lambda.min
# Get coeffs of best model
coefsLasso = coef(cv.out, s = "lambda.min")

# Calulate the test MSE
lasso.pred = predict(cv.out, newx = test.mat[, 2:18], s = "lambda.min")
test_MSE_Lasso = mean((lasso.pred - college.test$Outstate)^2)
```

Using the `cv.glmnet` the value for λ is chosen by testing a sequence of λ-values with 10-fold cross validation. The λ-value with the lowest cross validation error is 10.7206997. The MSE on the test data is 3.6880607×10^6 and so it is lower than using forward selection and BIC. The variables selected are:

```r
coefsLasso[1:18, ]
```

```
##   (Intercept)    PrivateYes          Apps        Accept        Enroll
## -1.172140e+03  2.230467e+03 -2.825215e-01  6.615811e-01 -3.778631e-01
##     Top10perc     Top25perc   F.Undergrad   P.Undergrad    Room.Board
##  4.589180e+01 -1.485674e+01 -5.800132e-02 -5.713770e-02  1.088115e+00
##          Books      Personal           PhD      Terminal     S.F.Ratio
## -9.185125e-01 -3.005419e-01  4.013410e+00  2.996744e+01 -6.936391e+01
##   perc.alumni        Expend     Grad.Rate
##  4.686967e+01  1.480013e-01  2.431539e+01
```

Notice that the the lasso model gives a nonzero values for all coefficients even though it is a sparse method. It does however give small coefficients for the covariates rejected by forward selection.

Problem 2

a)

FALSE, FALSE, TRUE, TRUE

b)

The basis functions for the cubic spline with knots the quartiles q_1, q_2 and q_3 are

$$\begin{aligned} b_1(x) &= x & b_4(x) &= h(x, q_1) \\ b_2(x) &= x^2 & b_5(x) &= h(x, q_2) \\ b_3(x) &= x^3 & b_6(x) &= h(x, q_3), \end{aligned}$$

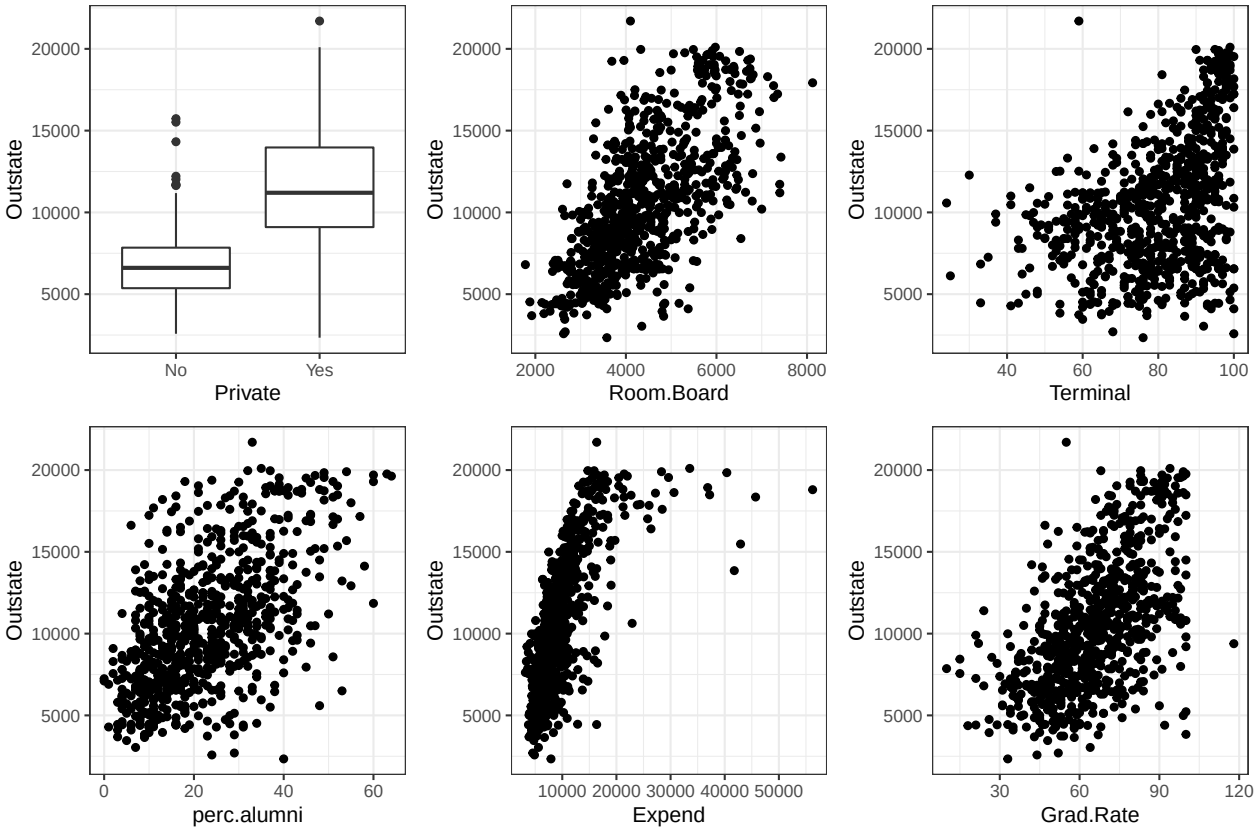
where

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

c)

```r
# scatterplots
p1 = ggplot(data = College, aes(x = Private, y = Outstate)) + geom_boxplot()
p2 = ggplot(data = College, aes(x = Room.Board, y = Outstate)) + geom_point()
p3 = ggplot(data = College, aes(x = Terminal, y = Outstate)) + geom_point()
p4 = ggplot(data = College, aes(x = perc.alumni, y = Outstate)) + geom_point()
p5 = ggplot(data = College, aes(x = Expend, y = Outstate)) + geom_point()
```

```
p6 = ggplot(data = College, aes(x = Grad.Rate, y = Outstate)) + geom_point()
ggarrange(p1, p2, p3, p4, p5, p6, ncol = 3, nrow = 2)
```



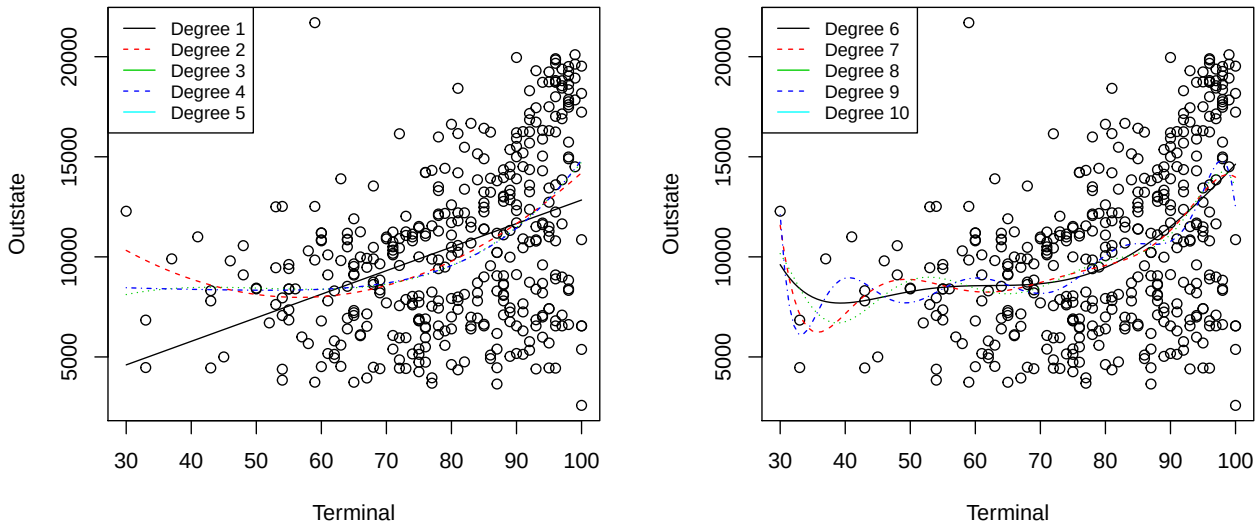
From the scatter plots above `Room.Board` and `perc.alumni` seem to have a linear relationship with `Outstate`. Conversely, `Terminal`, `Expend` and to some degree `Grad.Rate` seem to benefit form a non-linear transformation. There are also other plots that could have been used. For example Q–Q plots of standardized residual or test MSE for different degree of polynomial regression. We could also have performed ANOVA tests.

d)

i)

```
poly_MSE_train = rep(NA, 10)
poly_MSE_test = rep(NA, 10)

par(mfrow = c(1, 2))
plot(college.train$Terminal, college.train$Outstate, xlab = "Terminal", ylab = "Outstate")
for (i in 1:10) {
    # Calculate and plot
    reg.poly = lm(Outstate ~ poly(Terminal, i), data = college.train)
    x0 <- seq(min(college.train$Terminal), max(college.train$Terminal), length = 100)
    y0 = predict.lm(reg.poly, newdata = list(Terminal = x0))
    k = i%%5
    lines(x0, y0, col = k, lty = k)

    # Calculate training MSE
```

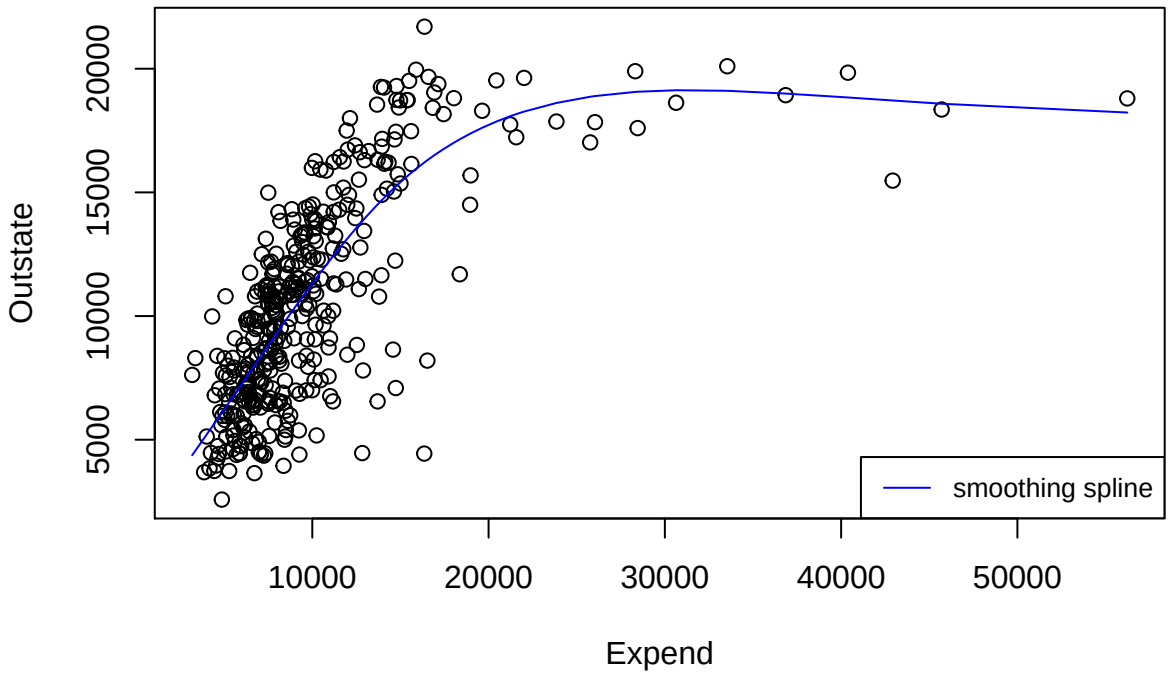
```
    pred.poly.train = predict.lm(reg.poly, newdata = list(Terminal = college.train$Terminal))
    poly_MSE_train[i] = mean((pred.poly.train - college.train$Outstate)^2)

    # Calculate test MSE
    pred.poly = predict.lm(reg.poly, newdata = list(Terminal = college.test$Terminal))
    poly_MSE_test[i] = mean((pred.poly - college.test$Outstate)^2)

    # Plot in two figures
    if (i == 5) {
        legend("topleft", legend = c("Degree 1", "Degree 2", "Degree 3", "Degree 4",
            "Degree 5"), col = c(1, 2, 3, 4, 5), lty = 1:2, cex = 0.8)
        plot(college.train$Terminal, college.train$Outstate, , xlab = "Terminal",
            ylab = "Outstate")
    }
}
legend("topleft", legend = c("Degree 6", "Degree 7", "Degree 8", "Degree 9", "Degree 10"),
    col = c(1, 2, 3, 4, 5), lty = 1:2, cex = 0.8)
```



The two plots above show regression with polynomials of different degrees.

ii)

```
reg.smooth = smooth.spline(x = college.train$Expend, y = college.train$Outstate,
    cv = TRUE)

plot(college.train$Expend, college.train$Outstate, xlab = "Expend", ylab = "Outstate")
lines(reg.smooth, col = "blue")
legend("bottomright", legend = c("smoothing spline"), col = ("blue"), lty = 1, cex = 0.8)
```




```
pred.smooth.train = predict(reg.smooth, newdata = list(Expend = college.train$Expend))
train_MSE_smooth = mean((pred.smooth.train$y - college.train$Outstate)^2)

pred.smooth = predict(reg.smooth, newdata = list(Expend = college.test$Expend))
test_MSE_smooth = mean((pred.smooth$y - college.test$Outstate)^2)
```

In the plot above shows smoothing spline regression of `Outstate` depending on `Expend`. The model degree of freedom is 4.660711 and was chosen by performing leave-one-out cross-validation, and choosing the model with least leave-one-out cross-validation error.

iii)

```
cat("Training MSEs: \n")
```

```
## Training MSEs:
```

```
for (i in seq(1, 10, by = 2)) {
    cat("deg", i, ":", poly_MSE_train[i])
    cat(", deg", i + 1, ":", poly_MSE_train[i + 1])
    cat("\n")
}
```

```
## deg 1 : 15075161, deg 2 : 14330586
## deg 3 : 14249448, deg 4 : 14247330
## deg 5 : 14231485, deg 6 : 14230392
## deg 7 : 14153207, deg 8 : 14097911
## deg 9 : 13841526, deg 10 : 13822205
```

```
cat("smooth:", train_MSE_smooth)
```

```
## smooth: 31072818
```

The training MSEs are printed above. I would have expected that polynomials regression of high order would give lower training MSEs as they are more flexible. The smoothing spline seemed to fit the data well, but had a high training MSE. This might be because of high variance in the dataset.

Problem 3

a)

FALSE, TRUE, TRUE, FALSE

b)

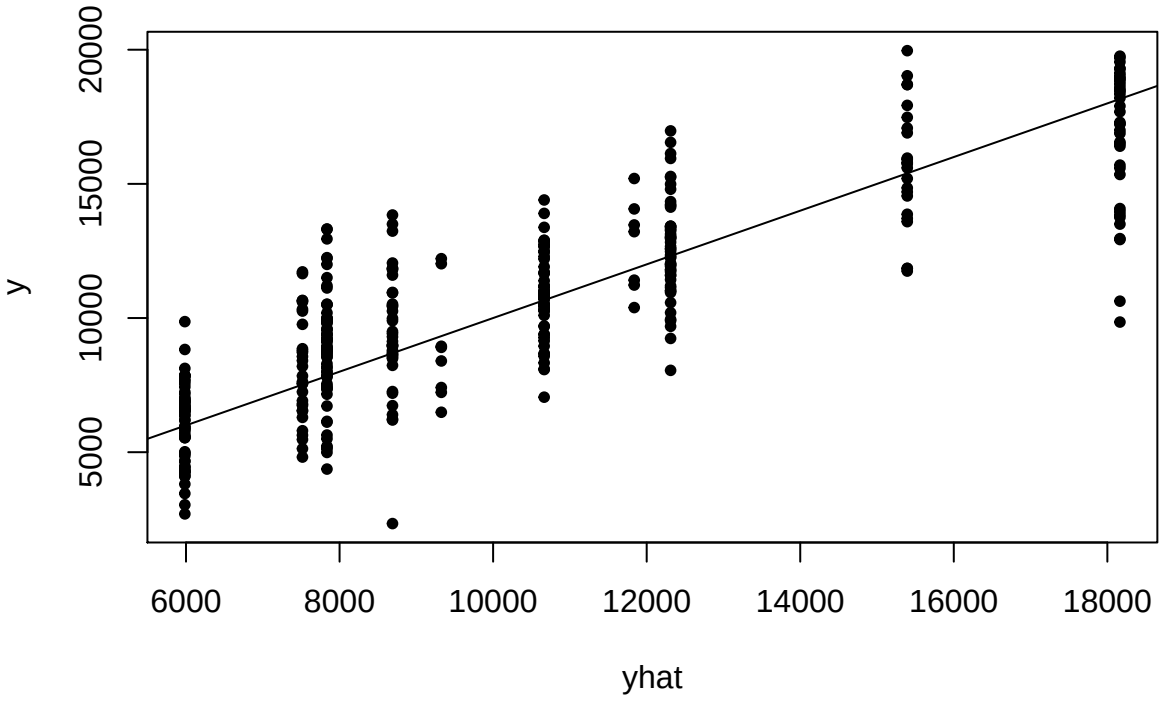
We try to make models based on a regression tree, random forest and by using boosting. The aim is to predict the variable *Outstate* by using the other variables in the *College* dataset as predictors.

We first try a regression tree and print the estimated standard deviation (root of the MSE). We also plot the estimated Outstate cost $\hat{y}$ against the Outstate cost for our testing dataset y, to get an idea of the model performance.

```
tree = tree(Outstate ~ ., data = college.train)
yhat = predict(tree, newdata = college.test)
y = college.test$Outstate
MSE_regtree = mean((yhat - y)^2)
cat("The standard deviation when using a regression tree model is:", sqrt(MSE_regtree))
```

```
## The standard deviation when using a regression tree model is: 2045.953
```

```
plot(yhat, y, pch = 20)
abline(a = 0, b = 1)
```



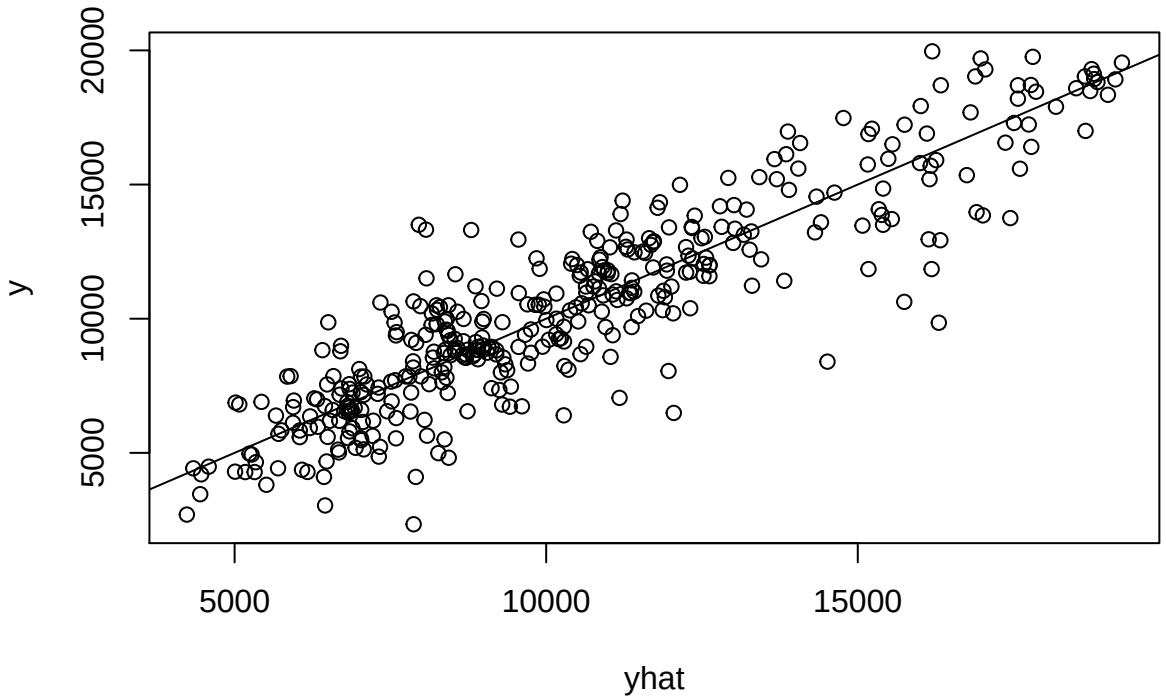
We do the same as above, only using a random forest model. On the plot below we observe how the random forest yields an estimate $\hat{y}$ that is no longer discrete as we saw for the regression tree. This follow from the fact that the random forest is an average of many different regression trees (as data is drawn by boostrapping), in our case $n_{tree} = 1000$.

```
rf = randomForest(Outstate ~ ., data = college.train, mtry = p - 1, ntree = 1000)
yhat = predict(rf, newdata = college.test)
y = college.test$Outstate
MSE.ranfor = mean((yhat - y)^2)
```

```r
cat("The standard deviation when using a random forest model is: ", sqrt(MSE.ranfor))
```

```
## The standard deviation when using a random forest model is:  1667.399
```

```r
plot(yhat, y)
abline(a = 0, b = 1)
```



Finally we use a boosting model, however in this case there are multiple tuning parameters that should be set by using cross validation. Below, we have implemented a k-fold cross validation scheme to find the optimal number of trees B, the shrinkage parameter λ and the interaction depth d.

By running a 5-fold cross validation below we found the optimal paramters $B = 500$, $\lambda = 0.01$ and $d = 6$.

In the end, we find the lowest MSE (or estimated SD in our case) for the boosting model. With the seed $s = 1234$ we find the following SD

Model	SD
Regression tree	2045
Random forest	1663
Boosting	1612

We would hence prefer to use this model, even though it is less interpretable than the regression tree. However, as seen below we could find the relative importance by how the different variables contribute to reducing the MSE. It is also expected that trees have high variance and small changes in the data can cause large changes in the final tree.

```r
set.seed(1234)

boost = gbm(Outstate ~ ., data = college.train, distribution = "gaussian", n.trees = 500,
    interaction.depth = 6, shrinkage = 0.01)
yhat = predict(boost, newdata = college.test, n.trees = 500)
y = college.test$Outstate
```

```r
MSE.boost = mean((yhat - y)^2)

cat("The standard deviation when using a boosting model is:", sqrt(MSE.boost))
```

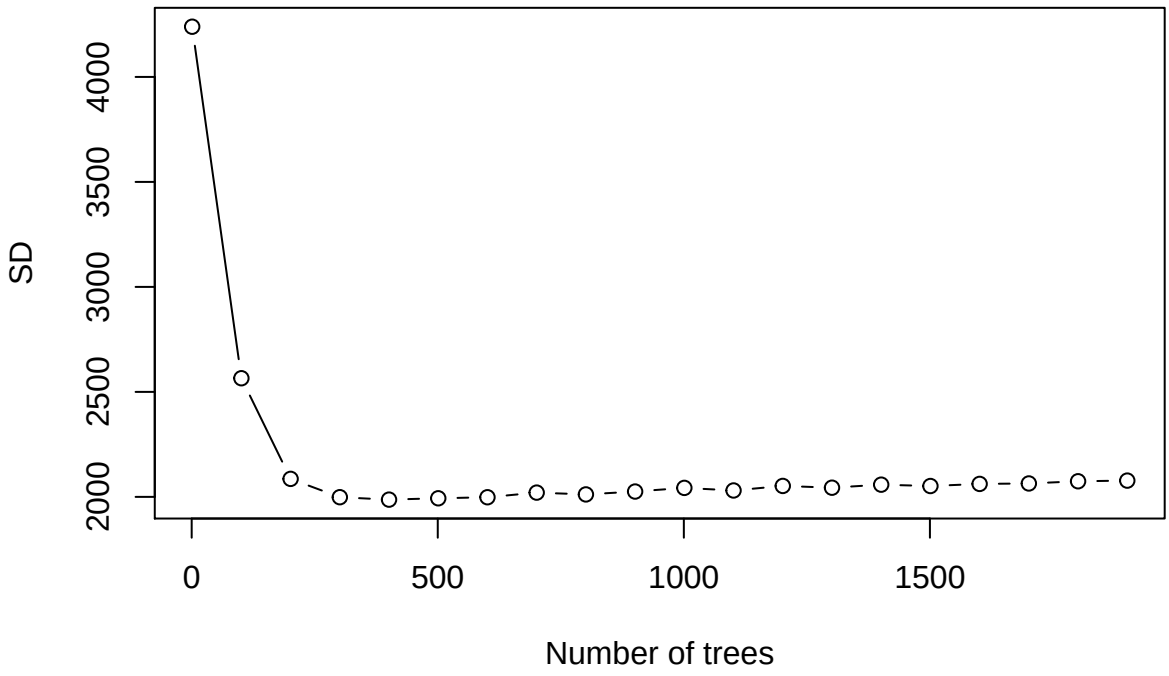
```
## The standard deviation when using a boosting model is: 1612.985
```

```r
# cross validation
k = 5

kFoldCV = function(k, data, paramInterval, param) {

    # Creating index set for CV
    n = dim(data)[1]
    allIndex = sample(1:n, n, replace = FALSE)
    # allIndex = seq(1:n)
    pSize = n%/%k
    kIndex = c(1)
    for (i in 1:k) {
        kIndex[i + 1] = pSize * i
    }
    kIndex[k + 1] = n + 1

    result = matrix(0L, ncol = 2, nrow = length(paramInterval))
    # Cross validation
    for (i in 1:k) {
        testIndex = allIndex[kIndex[i]:(kIndex[i + 1] - 1)]
        # print(i) print(testIndex)
        trainData = data[-testIndex, ]
        testData = data[testIndex, ]

        # Parameter testing
        for (j in 1:length(paramInterval)) {
            if (param == "trees") {
                boost = gbm(Outstate ~ ., data = trainData, distribution = "gaussian",
                  n.trees = paramInterval[j], interaction.depth = 6, shrinkage = 0.01)
                yhat = predict(boost, newdata = testData, n.trees = paramInterval[j])
            }
            if (param == "shrinkage") {
                boost = gbm(Outstate ~ ., data = trainData, distribution = "gaussian",
                  n.trees = 500, interaction.depth = 6, shrinkage = paramInterval[j])
            }
            if (param == "interaction") {
                boost = gbm(Outstate ~ ., trainData, distribution = "gaussian", n.trees = 500,
                  interaction.depth = paramInterval[j], shrinkage = 0.01)
            }

            if (param == "test") {
                boost = gbm(Outstate ~ ., trainData, distribution = "gaussian", n.trees = 500,
                  interaction.depth = 6, shrinkage = 0.01)
            }

            if (param != "trees") {
                yhat = predict(boost, newdata = testData, n.trees = 500)
            }
```

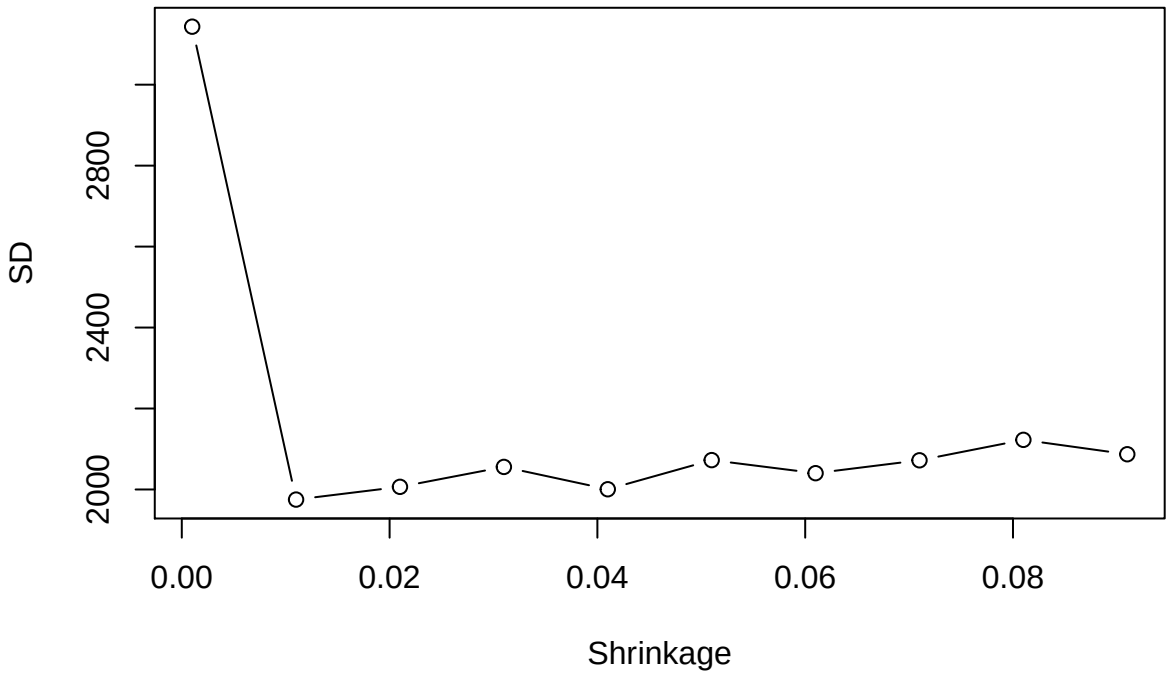
```
            y = testData$Outstate
            MSE = mean((yhat - y)^2)

            result[j, 1] = paramInterval[j]
            result[j, 2] = result[j, 2] + sqrt(MSE)/k
        }
    }
    return(result)
}
```

```
set.seed(1234)

pIntTrees = seq(1, 2000, 100)
pIntShrinkage = seq(1, 100, 10)/1000
pIntInteraction = seq(1, 10, 1)

# Tuning of boosting parameters
resT = kFoldCV(5, college.train, pIntTrees, "trees")
resS = kFoldCV(5, college.train, pIntShrinkage, "shrinkage")
resI = kFoldCV(5, college.train, pIntInteraction, "interaction")
```

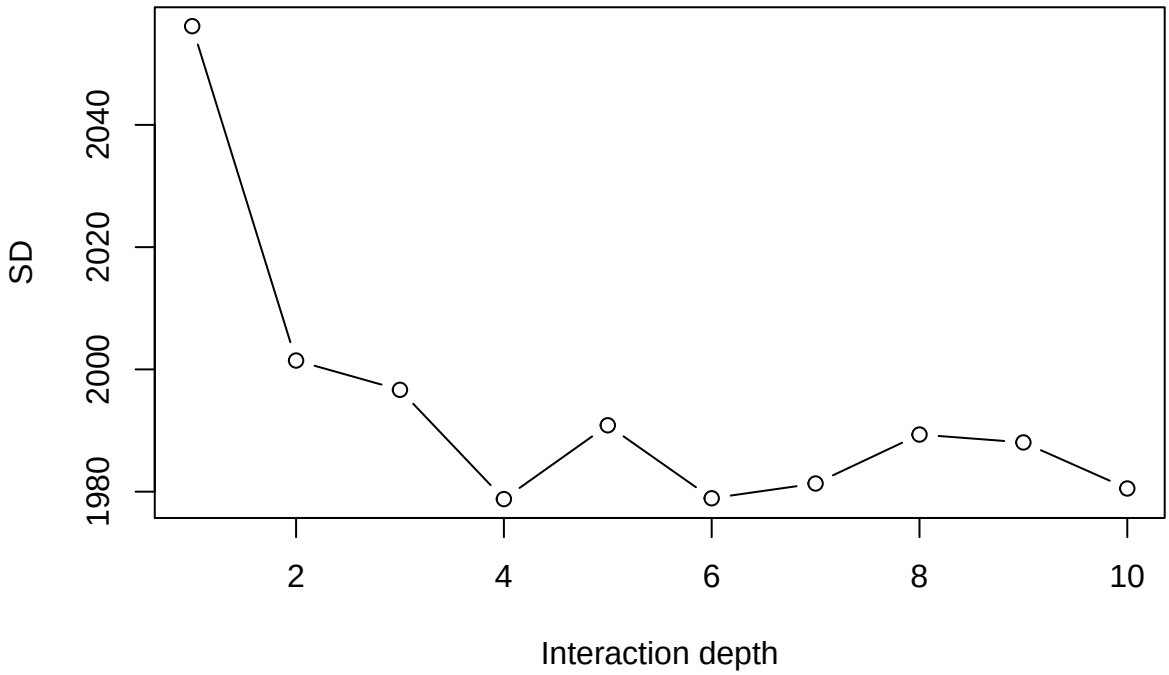
```
# Plotting SD vs Parametes from CV
plot(resT[, 1], resT[, 2], type = "b", ylab = "SD", xlab = "Number of trees")
```




```
plot(resS[, 1], resS[, 2], type = "b", ylab = "SD", xlab = "Shrinkage")
```




```
plot(resI[, 1], resI[, 2], type = "b", ylab = "SD", xlab = "Interaction depth")
```



c)

```
method.Poly = rep("Polynomial of degree ", 10)
for (i in 1:10) {
    method.Poly[i] = paste(method.Poly[i], i)
}

MSE.table1 = data.frame(Method = method.Poly, SD = sqrt(poly_MSE_test))
```

```
methods = c("Forward selection", "Lasso", "Smoothing spline", "Regression Tree",
    "Boosting")
MSEs = sqrt(c(test_MSE_Forward, test_MSE_Lasso, test_MSE_smooth, MSE_regtree, MSE.boost))
MSE.table2 = data.frame(Method = methods, SD = MSEs)

print(union(MSE.table1, MSE.table2))
```

```
##                       Method       SD
## 1   Polynomial of degree  1 3448.536
## 2   Polynomial of degree  2 3332.173
## 3   Polynomial of degree  3 3307.178
## 4   Polynomial of degree  4 3307.045
## 5   Polynomial of degree  5 3317.481
## 6   Polynomial of degree  6 3332.549
## 7   Polynomial of degree  7 4053.306
## 8   Polynomial of degree  8 3303.655
## 9   Polynomial of degree  9 9018.145
## 10 Polynomial of degree  10 4930.601
## 11        Forward selection 1960.831
## 12                    Lasso 1920.432
## 13         Smoothing spline 5355.878
## 14          Regression Tree 2045.953
## 15                 Boosting 1612.985
```

The test MSEs, are here given as test standard deviation ($\hat{SD} = \sqrt{MSE}$) in order to obtain a error measure with the same unit (and scale) as the OutState tuition, are printed above.

We see that polynomials of high order give large error because of overfitting. However, some high order polynomials did have low test MSE indications a nonlinear relationship between `Outstate` and `Expend`. The reason for the high test MSE for the smoothing spline can be similarly explained by the relatively high degree of freedom. As polynomial and smoothing spline regression were performed with only one covariate it is expected that these models had higher test MSE than the rest.

We notice that the boosting algorithm yields the lowest test MSE. However, it is less preferable when in comes to the degree of interpretability. Here, the simple and intuitive structure of the regression tree is to be preferred, but comes at the cost of larger test error. Regression with forward selection finds the most important variables (in our case 6) and could be said to be slightly easier to interpret than the other methods except for the regression tree. In general we also see that the linear models performed worse than Boosting. This indicates that the response is not well approximated by a linear model.

Problem 4

In this problem we use the data set of diabetes from a population of women of Pima Indian heritage in the US. We split the data set into a training set of 300 observations, where 200 are non-diabetic and 100 are diabetic, and a test with of 232 observations, where 155 are non-diabetic and 77 are diabetic.

a)

TRUE, TRUE, TRUE, TRUE

b)

To fit a support vector classifier and a support vector machine to the problem, the response variable `diabetes` must first be converted into a factor variable.

We start by fitting a support vector classifier, which has a linear boundary. To find a good cost parameter, cross-validation is used. The confusion table and the misclassification error reported are for the test set.

```
svc = svm(diabetes ~ ., data = d.train, kernel = "linear", cost = 0.1, scale = FALSE)

# Find best cost for SVC
set.seed(1)
tune.cost = tune(method = "svm", diabetes ~ ., data = d.train, kernel = "linear",
    ranges = list(cost = c(0.001, 0.01, 0.1, 1, 5, 10, 100)))
summary(tune.cost)  # 0.1 is the best cost
```

```
##
## 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
```

```
svc.bestmod = tune.cost$best.model

svc.pred = predict(svc.bestmod, d.test)
svc.ct = table(predict = svc.pred, truth = d.test$diabetes)  # Confusion table
svc.mcr = 1 - sum(diag(svc.ct))/sum(svc.ct)  # Misclassification error rate
print(paste0("Confusion table: "))
```

```
## [1] "Confusion table: "
```

```
svc.ct
```

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

```
print(paste0("Misclassification error rate: "))
```

```
## [1] "Misclassification error rate: "
```

```
svc.mcr
```

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

Similarly, we now fit a support vector machine with a radial boundary. Cross-validation is now used to find the optimal combination of cost and γ parameters.

```r
svmfit = svm(diabetes ~ ., data = d.train, kernel = "radial", gamma = 0.5, cost = 1,
    scale = FALSE)

# Find the best cost and gamma for SVM
set.seed(2)
tune.costgamma = tune(method = "svm", diabetes ~ ., data = d.train, kernel = "radial",
    ranges = list(cost = c(0.1, 1, 5, 10, 100), gamma = c(0.5, 1, 2, 3, 4)))
summary(tune.costgamma)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost gamma
##     1   0.5
##
## - best performance: 0.27
##
## - Detailed performance results:
##     cost gamma     error dispersion
## 1    0.1   0.5 0.3333333 0.08606630
## 2    1.0   0.5 0.2700000 0.09222892
## 3    5.0   0.5 0.3233333 0.08322215
## 4   10.0   0.5 0.3066667 0.07665861
## 5  100.0   0.5 0.3200000 0.07568616
## 6    0.1   1.0 0.3333333 0.08606630
## 7    1.0   1.0 0.3033333 0.09222892
## 8    5.0   1.0 0.3166667 0.06712803
## 9   10.0   1.0 0.3166667 0.06712803
## 10 100.0   1.0 0.3166667 0.06712803
## 11   0.1   2.0 0.3333333 0.08606630
## 12   1.0   2.0 0.3333333 0.09686442
## 13   5.0   2.0 0.3600000 0.09660918
## 14  10.0   2.0 0.3600000 0.09660918
## 15 100.0   2.0 0.3600000 0.09660918
## 16   0.1   3.0 0.3333333 0.08606630
## 17   1.0   3.0 0.3366667 0.08527371
## 18   5.0   3.0 0.3300000 0.07927137
## 19  10.0   3.0 0.3300000 0.07927137
## 20 100.0   3.0 0.3300000 0.07927137
## 21   0.1   4.0 0.3333333 0.08606630
## 22   1.0   4.0 0.3333333 0.08606630
## 23   5.0   4.0 0.3366667 0.08527371
## 24  10.0   4.0 0.3366667 0.08527371
## 25 100.0   4.0 0.3366667 0.08527371
```

```r
svm.bestmod = tune.costgamma$best.model

svm.pred = predict(svm.bestmod, d.test)
svm.ct = table(predict = svm.pred, truth = d.test$diabetes)
svm.mcr = 1 - sum(diag(svm.ct))/sum(svm.ct)
print(paste0("Confusion table: "))
```

```
## [1] "Confusion table: "
```

```
svm.ct
```

```
##        truth
## predict   0   1
##       0 133  38
##       1  22  39
```

```
print(paste0("Misclassification error rate: "))
```

```
## [1] "Misclassification error rate: "
```

```
svm.mcr
```

```
## [1] 0.2586207
```

Based on the confusion tables and their associated misclassification error rates, we can see that the support vector classifier performs better than the support vector machine, with a misclassification error rate of 0.228 instead of 0.259 for the support vector machine. Out of these two classifiers, we thus prefer the support vector classifier, even if the difference is relatively small.
The SVC also has both a higher sensitivity and specificity. The sensitivities for the SVC and SVM, respectively, are 0.884 and 0.858. The respective specificities are 0.545 and 0.507.

c)

We now compare the performance of the two classifiers from 4b) to a classification tree. As for the SVC and SVM we fit a model, now a classification tree, to our training set, before we use the test set to find the confusion table and misclassification error rate of the method.

```
d.tree = tree(diabetes ~ ., data = d.train)
tree.pred = predict(d.tree, d.test, type = "class")
tree.ct = table(tree.pred, d.test$diabetes)
tree.mcr = 1 - sum(diag(tree.ct))/sum(tree.ct)
print(paste0("Confusion table: ", tree.ct))
```

```
## [1] "Confusion table: 126" "Confusion table: 29"  "Confusion table: 28"
## [4] "Confusion table: 49"
```

```
print(paste0("Misclassification error rate: ", tree.mcr))
```

```
## [1] "Misclassification error rate: 0.245689655172414"
```

Note that `cv.tree` automatically does 10-fold cross-validation, so we don't have to prune the tree in the same way that we had to find the optimal cost and γ parameters in Problem 4b). We observe from the confusion table and the misclassification error rate that the classification tree performed better than the support vector machine, but worse than the support vector classifier. The sensitivity of this method is 0.813, which is lower than the values for both of the classifiers from 4b). However, the specificity, 0.636, is better.

Classification trees are in general, and especially for non-staticians, much easier to interpret than other classification methods. The structure and visualization of the tree is what makes it so easy to interpret, and the method is thus more used by non-staticians than SVM. In addition, the structure of the classification tree shows which predictor is the most important, by splitting on this predictor first. Neither classification trees nor SVMs make a huge amount of assumptions, for example about the distribution of the data. Thus they are less affected by outliers, and in that matter no method is preferred over the other one. Trees are often computed quite fast, but the greedy algorithm might not be as accurate as the SVM. However, the SVM can be harder to train, and without good parameters a good performance is not guaranteed - while trees are often very good classifiers. Finally, the process where the SVM projects the feature space into a kernel space before it is projected back to the original feature space, can produce a non-linear decision boundary that performs better than the hyperrectangles of the classification trees.

d)

FALSE, FALSE, TRUE, TRUE

e)

After manipulating the logistic function a little bit, we obtain

$$log\frac{P(x_i)}{1-P(x_i)} = f(x_i)$$
$$\implies P(y_i|x;\beta) = \frac{e^{f(x_i)}}{1+e^{f(x_i)}} = \frac{1}{1+e^{-f(x_i)}}.$$

Let σ be a function such that $P(y_i = 1|x_i) = \sigma(x_i)$ and $P(y_i = -1|x_i) = 1 - \sigma(x_i) = \sigma(-x_i)$, where the last equality comes from the properties of the function and can easily be seen by rewriting the equation a little bit - that is, $\sigma(x) = \frac{1}{1+e^{-x}}$. The cumulative distribution function can then be written as $P(y_i|x_i) = \sigma(x_i)^{y_i}(1-\sigma(x_i))^{1-y_i}$, which we recognize as a binomial PMF with $p = \sigma(x_i)$.

Furthermore, the log-likelihood function for this logistic regression function is given by

$$\begin{aligned}
l(z) &= -log(\Pi_{i=1}^n P(y_i|x_i)) = -\sum_{i=1}^n log(P(y_i|x_i)) = -\sum_{i=1}^n log\Big(\sigma(x_i)^{y_i}(1-\sigma(x_i))^{1-y_i}\Big) \\
&= -\sum_{i=1}^n \Big(y_i log(\sigma(x_i)) + (1-y_i)log(1-\sigma(x_i))\Big) = -\sum_{i=1}^n \Big(y_i(log(\sigma(x_i)) - log(-\sigma(x_i))) + log(-\sigma(x_i))\Big) \\
&= -\sum_{i=1}^n \Big(y_i log(\frac{\sigma(x_i)}{1-\sigma(x_i)}) + log(\sigma(-x_i))\Big) = -\sum_{i=1}^n (y_i x_i + log(-\sigma(x_i))) \\
&= \sum_{i=1}^n log(1+e^{-y_i x_i})
\end{aligned}$$

The following formulas are used in the computation:

$$log\Big(\frac{\sigma(x_i)}{1-\sigma(x_i)}\Big) = log\Big(\frac{1+e^{x_i}}{1+e^{-x_i}}\Big) = log\Big(\frac{e^{x_i}(1+e^{-x_i})}{1+e^{-x_i}}\Big) = x_i$$
$$log(\sigma(-x_i)) = log(1-\sigma(x_i)) = log\Big(\frac{1}{1/(1+e^{-x_i})}\Big) = log(1+e^{-x_i})$$
$$-y_i x_i + log(1+e^{x_i}) = log(1+e^{-y_i x_i}) \qquad \text{Since } y_i = \pm 1.$$

Then, since $f(x_i)$ corresponds to the linear predictor in logistic regression, we can replace x_i by $f(x_i)$ in the result above, which shows that the deviance for the $y = \pm 1$ encoding in logistic regression is the same as the given loss function $log(1+e^{y_i f(x_i)})$.

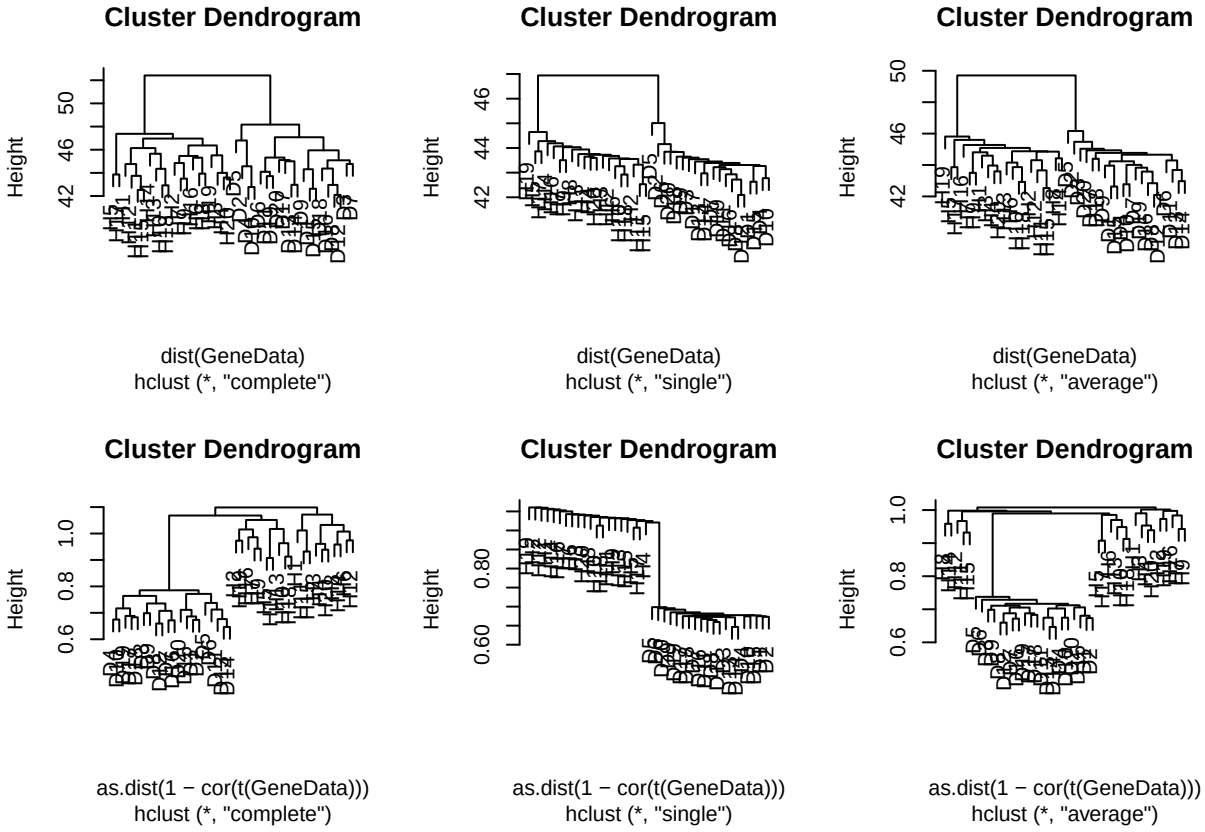
Problem 5

```
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)

In this task we study the measurements of 1000 genes from 40 tissue samples. We know that the first 20 tissue samples come from healthy patients and the remaining come from patients with disease. We will use different clustering method and principal component analysis to try to separate the two groups and study the data.

```
gene_complete = hclust(dist(GeneData), method = "complete")
gene_complete_column = hclust(dist(t(GeneData)), method = "complete")
gene_single = hclust(dist(GeneData), method = "single")
gene_average = hclust(dist(GeneData), method = "average")

gene_complete_cor = hclust(as.dist(1 - cor(t(GeneData))), method = "complete")
gene_single_cor = hclust(as.dist(1 - cor(t(GeneData))), method = "single")
gene_average_cor = hclust(as.dist(1 - cor(t(GeneData))), method = "average")


par(mfrow = c(2, 3))
plot(gene_complete)
plot(gene_single)
plot(gene_average)
plot(gene_complete_cor)
plot(gene_single_cor)
plot(gene_average_cor)
```



Above you will find six clusterings, where the plots columnwise uses the different linkages: complete, single and average, and the plots rowwise uses the different distance measures: Euclidean and correlation.

b)

«««< HEAD Below we compare the classification of the different clustering methods and find that all clusters using euclidian distance all classify the tissue correctly with $k = 2$ cuts.Furthermore we find that none of the correlation based hierarchical clustering methods obtain a successful separation between the two classes with two cuts. However, by cutting the tree with correlation based distance and complete linkage such that we get 5 clusters, we have get one cluster with cancer tissue and the rest with healthy tissue. =======

Below we compare the classification of the different clustering methods and find that all clusters using euclidian distance all classify the tissue correctly with $k = 2$ cuts. Furthermore we find that none of the correlation based hierarchical clustering methods obtain a successful separation between the two classes with two cuts. However, by cutting the tree with correlation based distance and complete linkage such that we get 5 clusters, we have get one cluster with cancer tissue and the rest with healthy tissue. »»»> 49dda26ae4f6c3c62a8c64c1c9f40cd52d4fdb25

Furthermore, we see that the Euclidean distance with complete has the most balanced tree, hence this could be preferred.

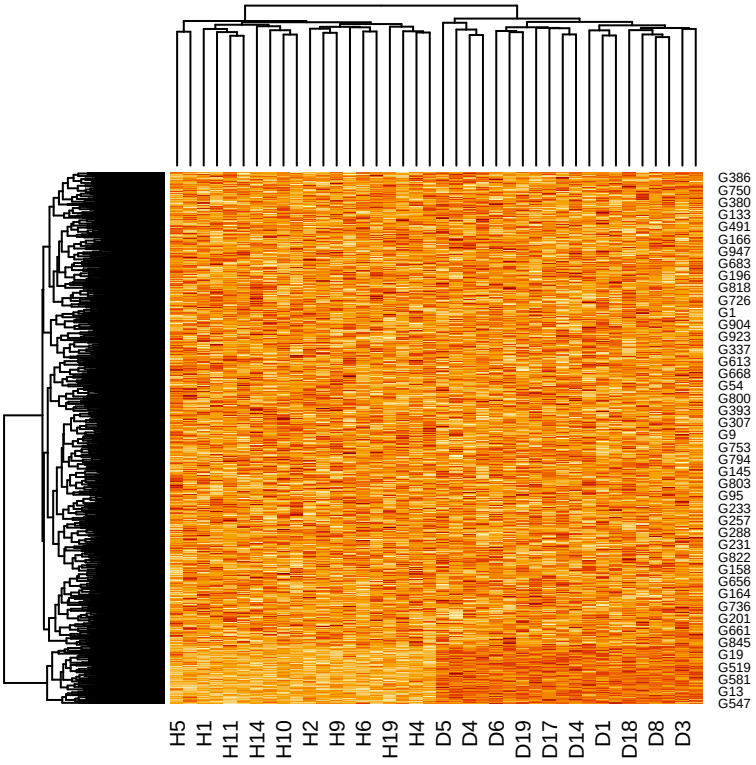
```
dc = cutree(gene_complete, k = 2)
ds = cutree(gene_single, k = 2)
da = cutree(gene_average, k = 2)
cc = cutree(gene_complete_cor, k = 2)
cs = cutree(gene_single_cor, k = 2)
ca = cutree(gene_average_cor, k = 2)

cbind(dc, ds, da, cc, cs, ca)
```

```
##     dc ds da cc cs ca
## H1   1  1  1  1  1  1
## H2   1  1  1  2  1  1
## H3   1  1  1  1  1  1
## H4   1  1  1  1  1  1
## H5   1  1  1  2  1  2
## H6   1  1  1  1  1  2
## H7   1  1  1  2  1  1
## H8   1  1  1  1  1  2
## H9   1  1  1  2  1  1
## H10  1  1  1  2  1  2
## H11  1  1  1  1  1  2
## H12  1  1  1  1  1  2
## H13  1  1  1  2  1  2
## H14  1  1  1  1  1  2
## H15  1  1  1  1  1  2
## H16  1  1  1  2  1  1
## H17  1  1  1  2  1  2
## H18  1  1  1  2  1  2
## H19  1  1  1  2  2  1
## H20  1  1  1  1  1  1
## D1   2  2  2  2  1  2
## D2   2  2  2  2  1  2
## D3   2  2  2  2  1  2
## D4   2  2  2  2  1  2
## D5   2  2  2  2  1  2
## D6   2  2  2  2  1  2
## D7   2  2  2  2  1  2
## D8   2  2  2  2  1  2
```

```
## D9   2  2  2  2  1  2
## D10  2  2  2  2  1  2
## D11  2  2  2  2  1  2
## D12  2  2  2  2  1  2
## D13  2  2  2  2  1  2
## D14  2  2  2  2  1  2
## D15  2  2  2  2  1  2
## D16  2  2  2  2  1  2
## D17  2  2  2  2  1  2
## D18  2  2  2  2  1  2
## D19  2  2  2  2  1  2
## D20  2  2  2  2  1  2
```

As an additional analysis we use the heatmap-function in R to plot dendrograms applied to both the rows and columns in the GeneData. This displays in a beautiful manner that we have a group of genes that seems to separate the patients with cancer and without cancer. By cutting the tree classifying the different genes in two we can find the genes that seem to predict cancer.

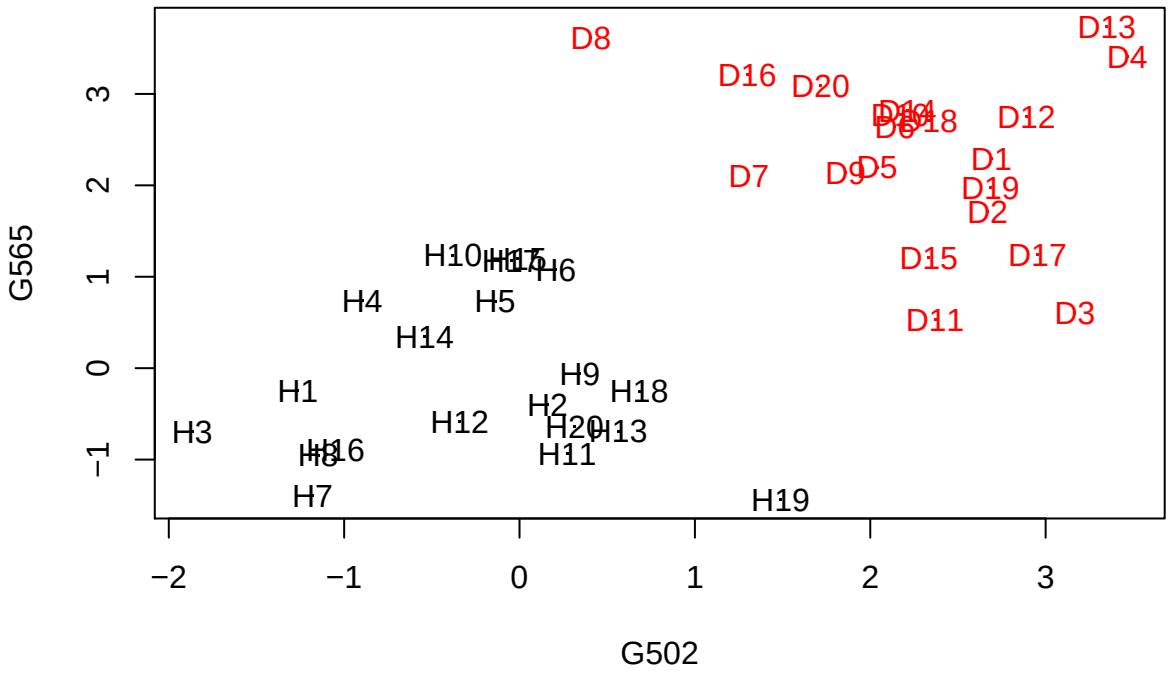
```
heatmap(as.matrix(t(GeneData)), Rowv = as.dendrogram(gene_complete_column), Colv = as.dendrogram(gene_c
```




```
important_genes_clust = which(cutree(gene_complete_column, k = 2) == 2)
```

Finally we also plot the tissue with regards to the measure of two different genes that both are identified as "important" from the above analysis. We mark them with colour based on the hierarchical clustering with Euclidean distance and complete linkage.

```
plot(GeneData[, 502], GeneData[, 565], col = cutree(gene_complete, 2), pch = ".",
    ylab = "G565", xlab = "G502")
text(GeneData[, 502], GeneData[, 565], rownames(GeneData), col = cutree(gene_complete,
    2))
```



c)

There are multiple ways to motivate Principal Component Analysis, but recalling the spectral theorem from linear algebra provides a useful framework for understanding and explanation:

> **Spectral Theorem**
>
> Given a matrix $A \in \mathbb{R}^{n\times n}$ there exists a diagonal matrix $\Lambda = \text{diag}\{\lambda_1, \cdots, \lambda_n\}$, where λ_i are the eigenvalues of A, and a unitary matrix U (meaning $U^T = U^{-1}$) such that
>
> $$A = U\Lambda U^T$$
>
> if and only if $A^T A = AA^T$, meaning A is normal.

Given a datamatrix $X \in \mathbb{R}^{n\times p}$, $X = [x_1, \cdots, x_n]^T$, meaning we have n observations x_i of p variables, lets first assume that X is normalized, meaning the mean of every column (multiple observations of a given variable) is zero and the sample variance is equal to one. This is easily achieved with the Mahalanobis transformation. This yields the following estimate of the covariance matrix

$$S = \frac{X^T X}{n}$$

We note that S is in fact symmetric and hence normal, allowing for the spectral decomposition described above,

$$S = \frac{1}{n} U\Lambda U^T$$

Where the columns of $U = [u_1, \cdots, u_n]$ are the eigenvectors of $X^T X$ and the diagonal entries $\Lambda = \text{diag}\{\lambda_1, \cdots, \lambda_n\}$ the corresponding eigenvalues. Notice that as U is unitary, the eigenvectors are orthonormal, meaning $u_i^T u_j = \delta_{i,j}, i, j = 0, \cdots, n$. Finally, lets also assume that Λ and U is permutated in a way such that $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n$.

We now notice that as $X^T X = U\Lambda U^T$ we have that

$$U^T X^T X U = \Lambda$$

And by dividing the expression above with n we see that the transformation $Z = XU$ yield the very orderly estimated covariance matrix $\frac{1}{n}\Lambda$ which is diagonal with decreasing variances $\frac{\lambda_1}{n}, \cdots, \frac{\lambda_n}{n}$.

The columns of Z are then the principal components $z_m, m = 0, \cdots, n$ with decreasing variance $\frac{\lambda_m}{n}$. For some reason unknown to the author, the elements of the principal components are also called scores and hence the scores of the first principal components would be given as

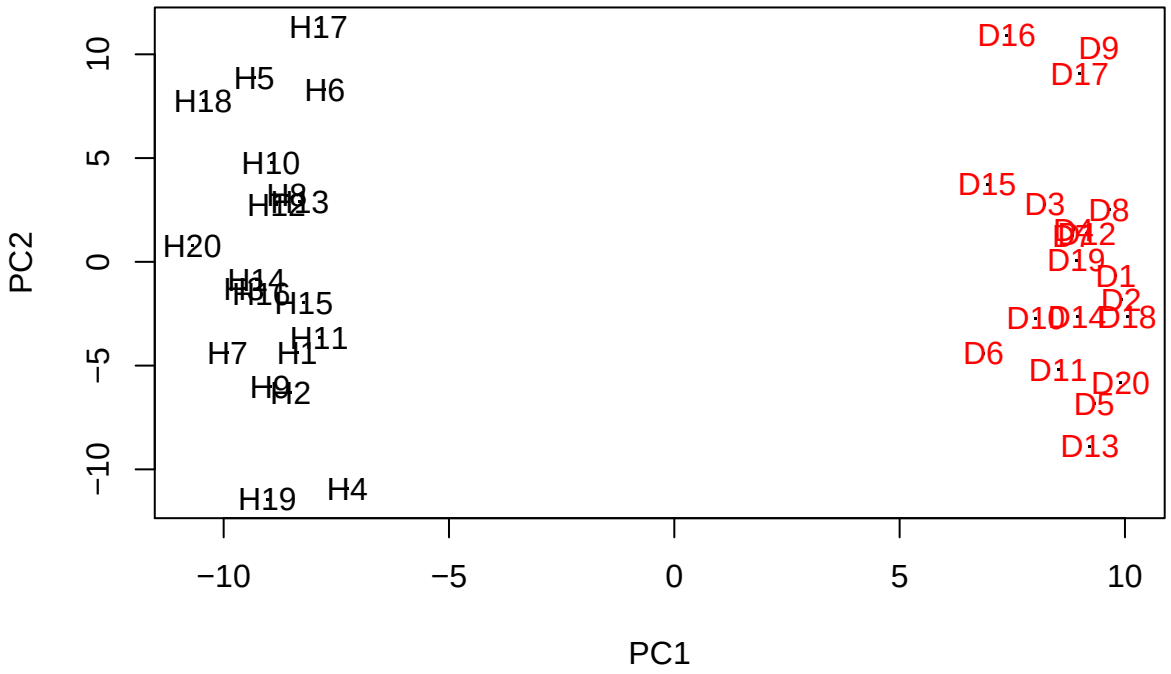
$$z_1 = Xu_1$$

As we have chosen to derive the PCA in a different way that what is done in the lecture notes, there are some differences in the notation. p, n, X are the same, while we denote the matrix ϕ as U. Meaning we have the first column in U as $u_1 = \phi_1$.

d)

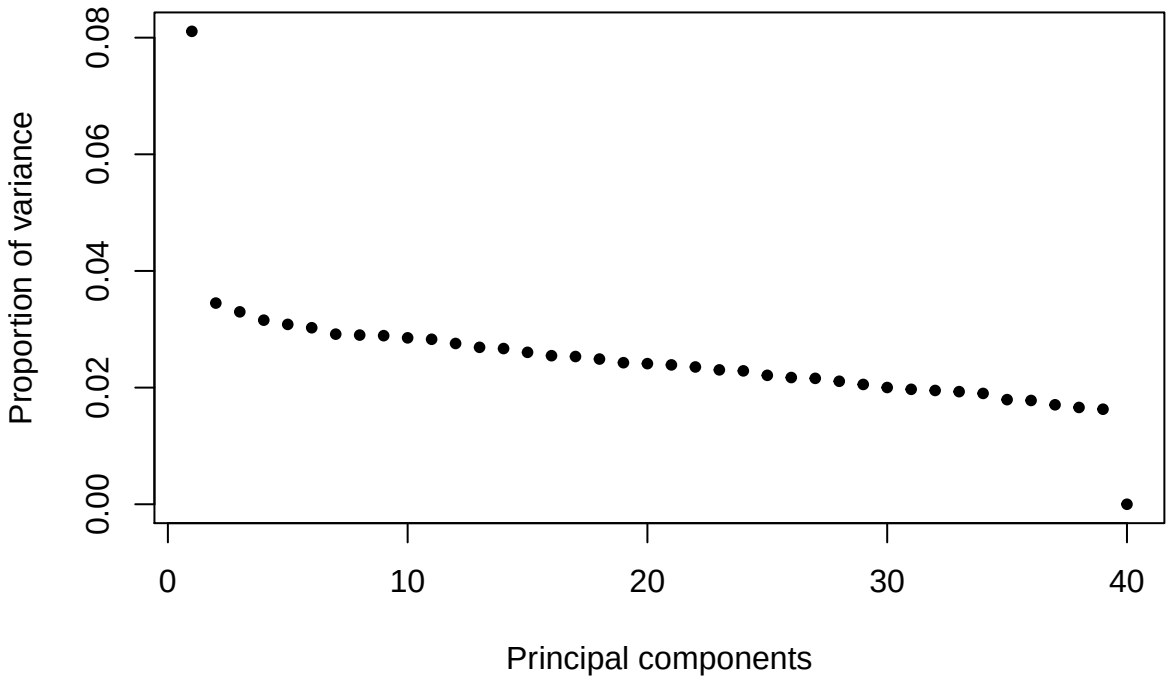
We will now perform a principal component analysis on the gene data.

```
gene_pca = prcomp(GeneData, scale = TRUE)
plot(gene_pca$x[, 1:2], pch = ".")
text(gene_pca$x[, 1:2], rownames(GeneData), col = cutree(gene_complete, 2))
```



Above we have used the first and second principal component to plot the tissue samples, and have colored them by using the correct hierachical tree from the above exercise. Below we find the proportion of variance explained by the different principal components, and specifically the five first. We also plot this.

```
var = gene_pca$sdev^2
prop_var = var/sum(var)
plot(prop_var, pch = 20, xlab = "Principal components", ylab = "Proportion of variance")
```




```
print("Proportion of variance explained by the 5 first PCs:")
```

```
## [1] "Proportion of variance explained by the 5 first PCs:"
```

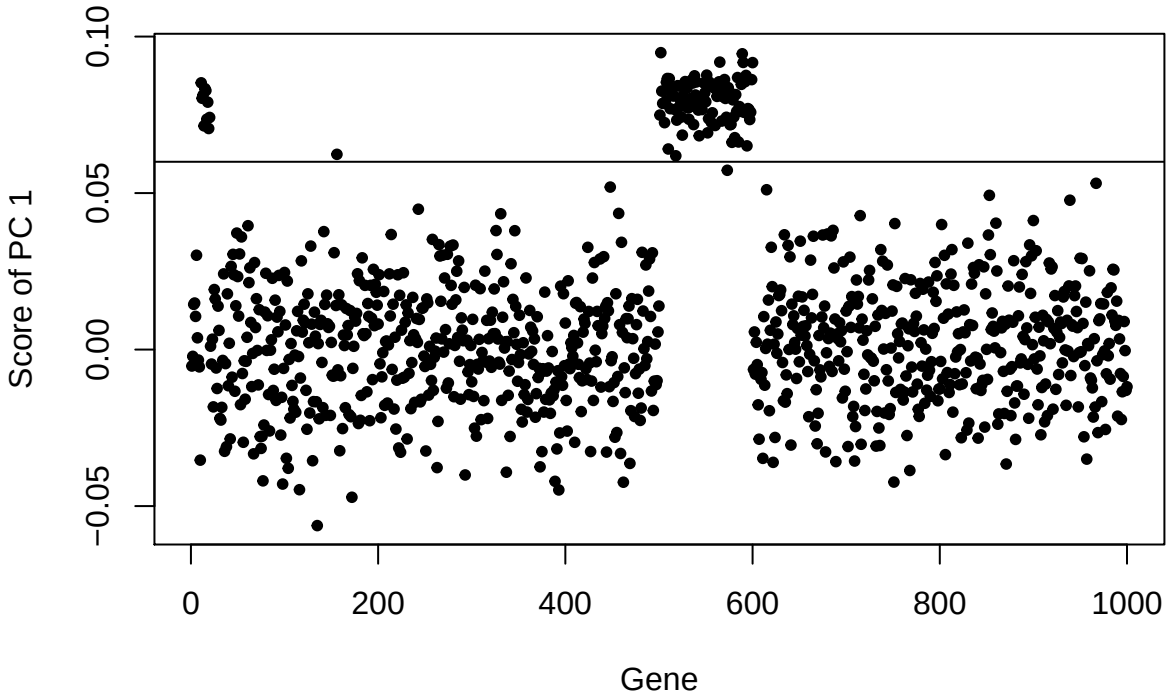
```
print(sum(prop_var[1:5]))
```

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

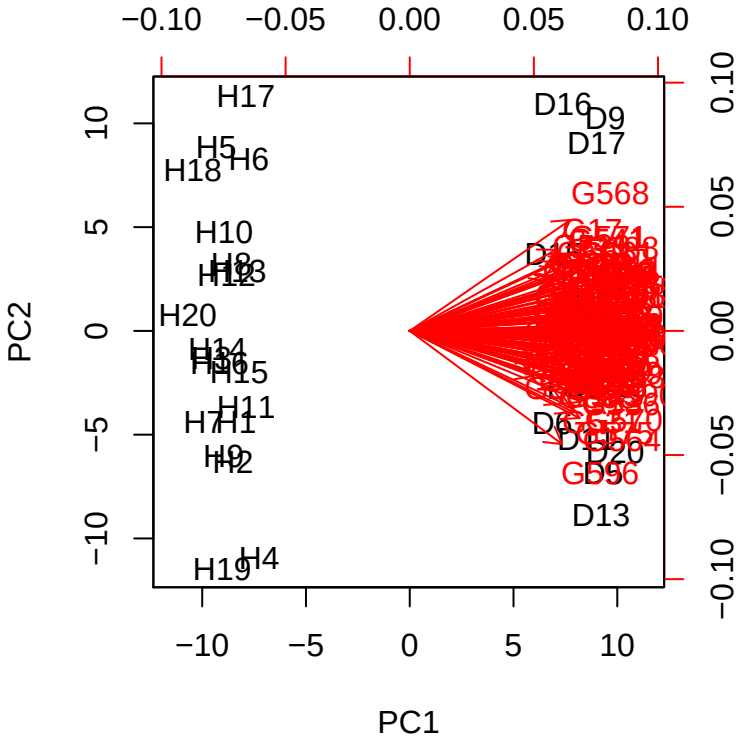
e)

Below we plot the scores, or the weights of the different elements of the first principal component. Here we clearly see that there is a group of genes with a larger score than the others and find a cutoff value at 0.06 to separate the two. We also identified genes that vary greatly between the different groups from the heatmap above. It could be seen that the two groups are almost identical. On the bottom plot we have a biplot with the genes identified.

```
gene_PC_1 = gene_pca$rotation[, 1]
plot(gene_PC_1, pch = 20, ylab = "Score of PC 1", xlab = "Gene")
cutoff = 0.06
abline(a = cutoff, b = 0)
```



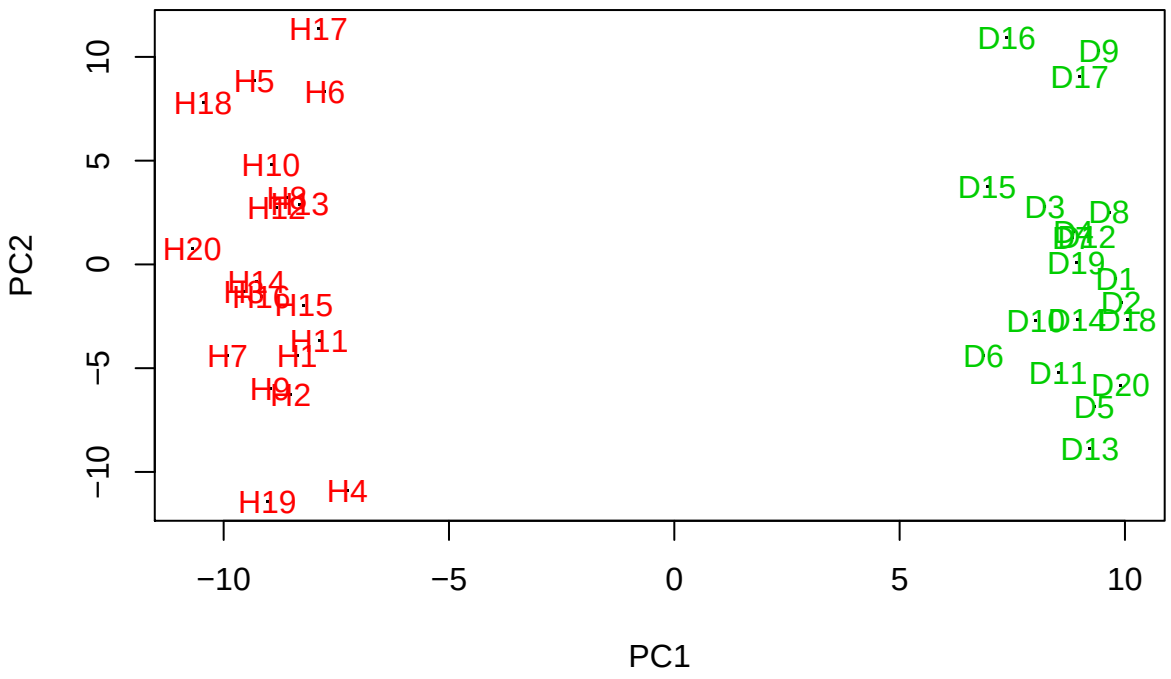

```
important_genes = which(gene_PC_1 > cutoff)
biplot(gene_pca$x[, 1:2], gene_pca$rotation[important_genes, ])
```



f)

Finally we use K-means clustering to separate the two groups of tissue and see that in fact, this algorithm is also fully successfully, having an error rate of zero.

```
gene_km = kmeans(as.matrix(GeneData), 2, nstart = 20)
plot(gene_pca$x[, 1:2], pch = ".")
text(gene_pca$x[, 1:2], rownames(GeneData), col = gene_km$cluster + 1)
```




```
gene_km$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
```