Compulsory exercise 2: Group 4

TMA4268 Statistical Learning V2021

Øystein Alvestad, Lars Evje and William Scott Grundeland Olsen

18 april, 2021

Problem 1

a)

TRUE, TRUE, TRUE, FALSE

b)

Read the file:

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

Check if the dataset contains any Nans or infinite values:

```
# Check if data contains Nans
any(apply(catdat, 2, function(x) any(is.na(x))))
```

```
## [1] FALSE
```

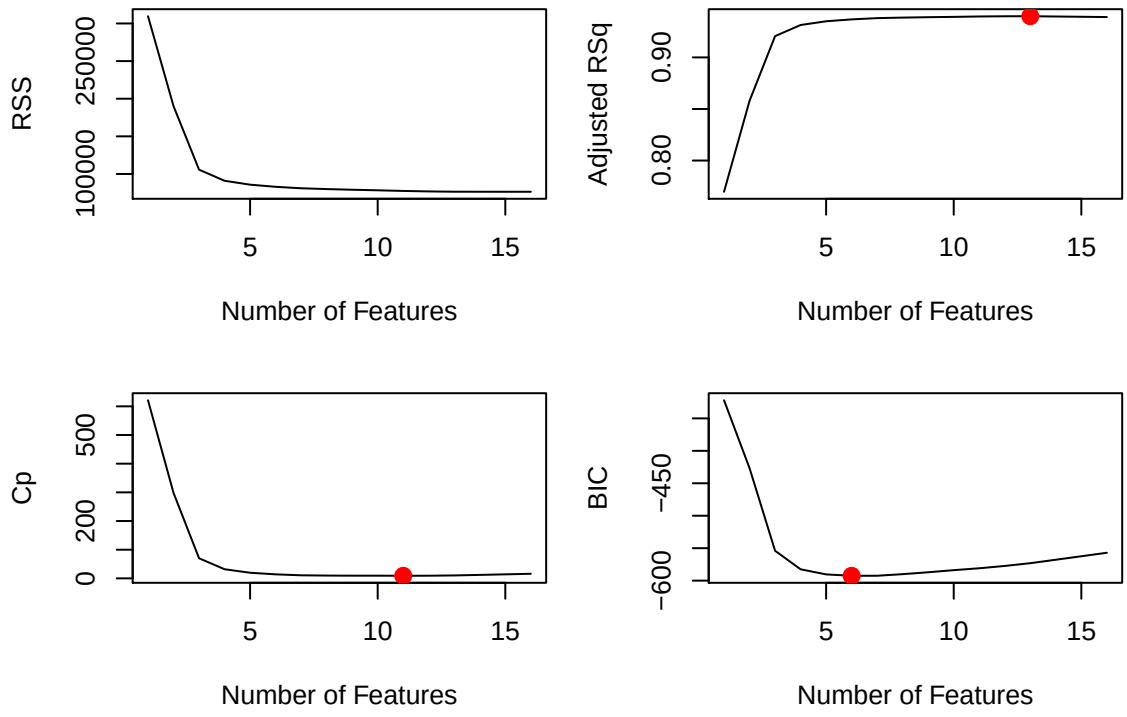
```
any(apply(catdat, 2, function(x) any(is.infinite(x))))
```

```
## [1] FALSE
```

We proceed by splitting the data into training and test sets. Afterwards we use regsubsets to search for the best subset models. To be able to decide which model is the best we look at three different model selection criteria: adjusted-R^2, Mallows' Cp and BIC. The plot shows these model selection criteria as a function of the number of features.

The red dot indicates the best value of the respective model selection criterion. We see that according to the BIC a model with 6 features is preferred, while Cp and adjusted-R^2 do well for 6 features, but even better for higher number of features. To gain robustness in the model we select a model with 6 parameters.

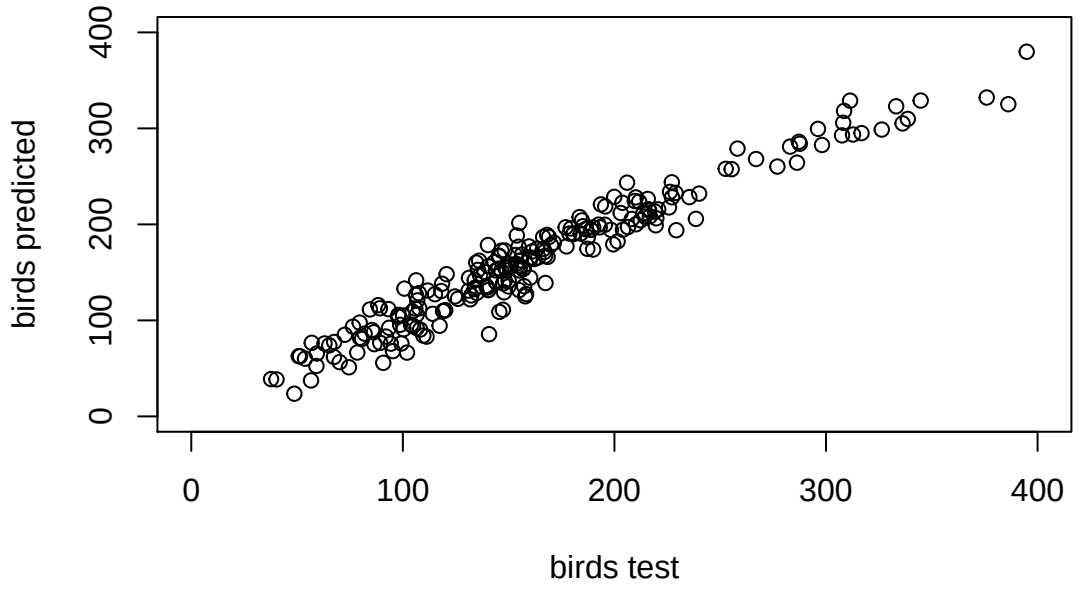
```r
# Split data set into train and test
set.seed(4268)
train.ind = sample(1:nrow(catdat), 0.5 * nrow(catdat))
catdat.train = catdat[train.ind, ]
catdat.test = catdat[-train.ind, ]

# Fit many models in search for best subset models
regfit.full = regsubsets(birds ~ ., catdat.train, nvmax = 16)

# Extract indices for best min and max values of errors
adjr2_max <- which.max(summary(regfit.full)$adjr2)
cp_min <- which.min(summary(regfit.full)$cp)
bic_min <- which.min(summary(regfit.full)$bic)

# Plot
par(mfrow = c(2, 2), mar = c(5.1, 4.1, 1, 1))
plot(summary(regfit.full)$rss, xlab = "Number of Features", ylab = "RSS", type = "l")
plot(summary(regfit.full)$adjr2, xlab = "Number of Features", ylab = "Adjusted RSq",
    type = "l")
points(adjr2_max, summary(regfit.full)$adjr2[adjr2_max], col = "red", cex = 2, pch = 20)
plot(summary(regfit.full)$cp, xlab = "Number of Features", ylab = "Cp", type = "l")
points(cp_min, summary(regfit.full)$cp[cp_min], col = "red", cex = 2, pch = 20)
plot(summary(regfit.full)$bic, xlab = "Number of Features", ylab = "BIC", type = "l")
points(bic_min, summary(regfit.full)$bic[bic_min], col = "red", cex = 2, pch = 20)
```



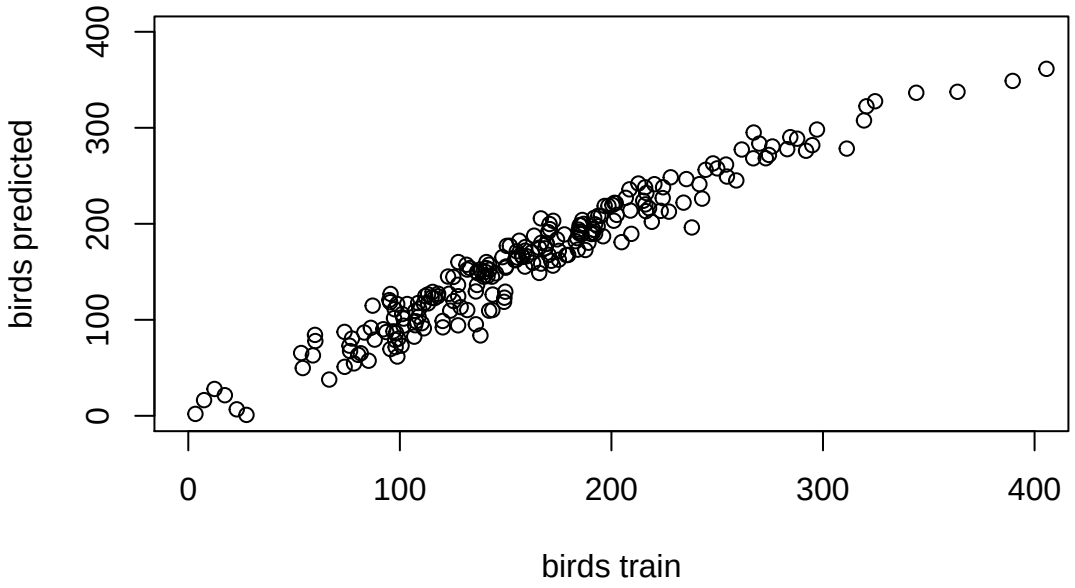
For a linear model with six features we find the six best feautres by reading off the regsubsets-summary. The features chosen are: wetfood, daily.playtime, children.13, urban, bell and daily.outdoortime. From the understanding of the domain, the selected features all seem reasonable with respect to having predictive power. We check the model qualitatively by plotting predicted values for the training set and test set against

the response in the two plots below. We see that the model performs well.

```
# Fit model with 6 parameters
lm_6 = lm(birds ~ wetfood + daily.playtime + children.13 + urban + bell + daily.outdoortime,
    catdat.train)

# Predict and calculate MSE
lm_birds_hat <- predict(lm_6, catdat.test[c("wetfood", "daily.playtime", "children.13",
    "urban", "bell", "daily.outdoortime")])
birds_mse <- mean((catdat.test[[c("birds")]] - lm_birds_hat)^2)

lm_birds_hat_train <- predict(lm_6, catdat.train[c("wetfood", "daily.playtime", "children.13",
    "urban", "bell", "daily.outdoortime")])

plot.new()
par(mfrow = c(1, 1))
plot(catdat.test[[c("birds")]], lm_birds_hat, xlim = c(0, 400), ylim = c(0, 400),
    xlab = "birds test", ylab = "birds predicted")
```




```
plot.new()
par(mfrow = c(1, 1))
plot(catdat.train[[c("birds")]], lm_birds_hat_train, xlim = c(0, 400), ylim = c(0,
    400), , xlab = "birds train", ylab = "birds predicted")
```



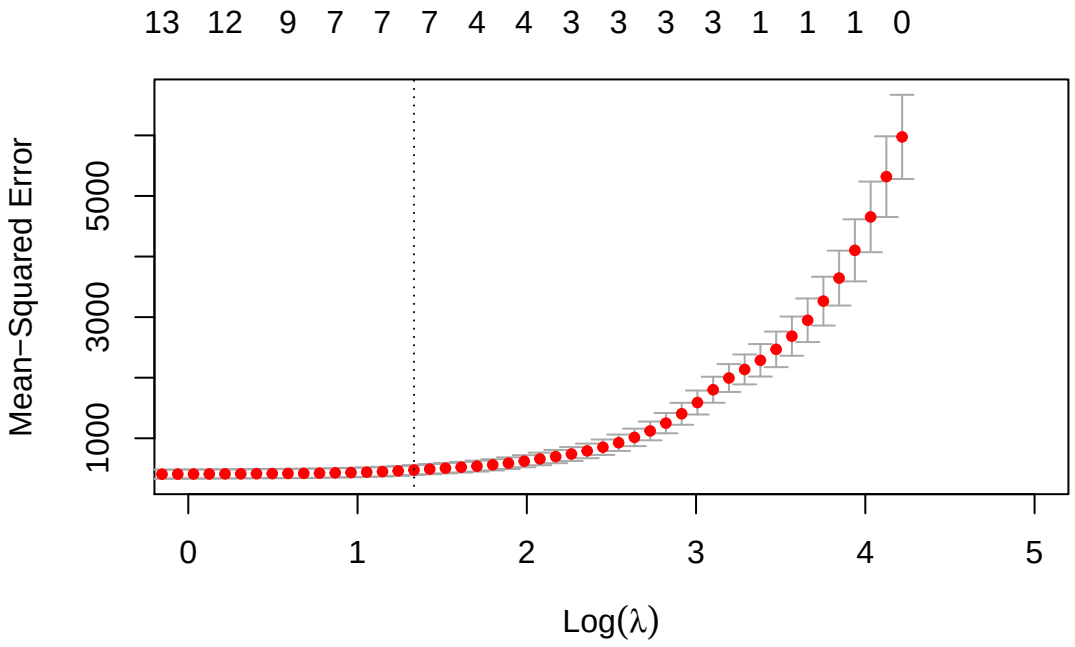
The test MSE for a linear model with 6 parameters (list above) is: 299.8835365.

c)

We use glmnet to perform Lasso regression. λ is chosen through cross validation which is offered as part of the glm-package. The plot below shows how MSE varies with different values of λ.

The non-zero coefficients are: weight, dryfood, wetfood, owner.income, daily.playtime, owner.age, house.area, children.13, urban, bell, dogs, daily.outdoortime, neutered.

```
# Split train and test into feature and response parts
x.train <- model.matrix(birds ~ ., data = catdat.train)[, -1]
y.train <- catdat.train$birds
x.test = model.matrix(birds ~ ., data = catdat.test)[, -1]
y.test = catdat.test$birds

# Find best lambda and fit model
set.seed(4268)
cv.out = cv.glmnet(x.train, y.train, alpha = 1)
plot(cv.out, xlim = c(0, 5))
```




```
bestlam = cv.out$lambda.min

lasso.mod = glmnet(x.train, y.train, alpha = 1, lambda = bestlam)
lasso.pred = predict(lasso.mod, s = bestlam, newx = x.test)
lasso_birds_mse <- mean((catdat.test[[c("birds")]] - lasso.pred)^2)

# Show coefficients of Lasso model
coef(lasso.mod, s = bestlam)
```

```
## 18 x 1 sparse Matrix of class "dgCMatrix"
##                                1
## (Intercept)         1.215859e+02
## sex                 .
## weight             -2.718865e+00
## dryfood            -2.611541e+00
## wetfood            -7.691704e+00
## age                 .
## owner.income       -1.114425e-05
## daily.playtime     -1.016884e+01
## fellow.cats         .
## owner.age          -1.578910e-01
## house.area          7.063071e-02
## children.13         3.731519e+00
## urban              -8.351642e+00
## bell               -4.986001e+01
## dogs                3.422164e+00
## daily.outdoortime   1.139824e+00
## daily.catnip        .
## neutered           -4.376866e+00
```

The test MSE for the best Lasso-model is: 298.3288862.

d)

If we let $\lambda \to \infty$ the best fit will have all coefficients $\beta_1, \beta_2, ...$ equal to zero. This is because the regularizing term in the objective will dominate the solution. Then the only coefficient left is the intercept β_0 which optimal value will be equal to the average of the response values of the training examples.

If we let $\lambda = 0$ the regularizing term disappears from the objective. Then we are left with the least squares objective from regular linear regression.

```
# Extreme lasso values
lasso.mod_lamzero = glmnet(x.train, y.train, alpha = 1, lambda = 0)
lasso.pred_lamzero = predict(lasso.mod_lamzero, s = 0, newx = x.test)
lasso_birds_mse_lamzero <- mean((catdat.test[[c("birds")]] - lasso.pred_lamzero)^2)

lasso.mod_lambig = glmnet(x.train, y.train, alpha = 1, lambda = 1e+10)
lasso.pred_lambig = predict(lasso.mod_lambig, s = 1e+10, newx = x.test)
lasso_birds_mse_lambig <- mean((catdat.test[[c("birds")]] - lasso.pred_lambig)^2)
```

The test-MSE for the Lasso-model with $\lambda = 0$ is: 302.4565944.

The test-MSE for the Lasso-model with $\lambda \to \infty$ is: 4914.0693576.

e)

The model with only intercept is much worse than both the reduced six-parameter linear regression model and the best Lasso-regression model. The difference is about 1 order of magnitude in the MSE.

The full linear model (using all features) is about as good as the reduced linear model and the best Lasso-regression model. This is somewhat not surprising from the plots showing adjusted-R^2 and Mallows' Cp in problem 1 b) where we can see that the performance is quite stable for many features. As the BIC parameter showed (in the same plot) is however that we are better off choosing a smaller model with fewer features (which is preferable) although the performance is not much better for the smaller model.

```
# Model with only intercept
intercept_mse <- mean((y.test - mean(y.train))^2)

# Full regression model
lm_full = lm(birds ~ ., catdat.train)

# Predict and calculate MSE
lmfull_birds_hat <- predict(lm_full, catdat.test)
lmfull_birds_mse <- mean((catdat.test[[c("birds")]] - lmfull_birds_hat)^2)
```

The test-MSE for the full linear model is: 301.9185792.

The test-MSE for the only intercept model is: 4914.0693576.

f)

The table below summarizes the results from the modeling above.

Model	MSE
Full linear model	301.9185792
Reduced linear model	299.8835365
Best Lasso-regression model	298.3288862
Intercept model	4914.0693576

A pure intercept model will only perform well for problems with no signal in the data. The F-test of e.g. the reduced linear model suggests that a null hypothesis of having all coefficients $\beta_1, \beta_2, ...$ equal to zero should be rejected. On this basis it is not unsurprising that the intercept model performs worse than the two other linear models. The best Lasso-regression model (optimized over many values of λ) lies on the spectrum of the intercept model and the full linear model. Thus, the Lasso-model is expected to to better than the intercept model in this case, because all small values of λ will give something similar to a full linear model.

The reduced linear model performs better than the full liner model, but they are still within 1% of each other. So, there is not much gained with respect to. prediction to use a reduced linear model. It could however be beneficial if inference is of interest. The Lasso-regression does slightly better than the reduced linear model. So the bias in the estimation (introduced by the regularization) was dominated by the Lasso-regressions ability to search for a good reduced model. So Lasso proved slightly better than an analyst's effort. Again, with reference to the model criteria plotted, the Lasso-regression was able to search the constant interval of of e.g. adjusted-R^2 and find a better solution than the reduced linear model. However, since the interval is constant, it does not make much change in the end result.

Problem 2

a)

TRUE, TRUE, FALSE, FALSE

b)

The basis functions for a cubic spline with knots at the quartiles q_1 and q_2 of variable X are: X, X^2, X^3, $h(X, q_1)$, $h(X, q_2)$. The function $h(x, q)$ is defined as:

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

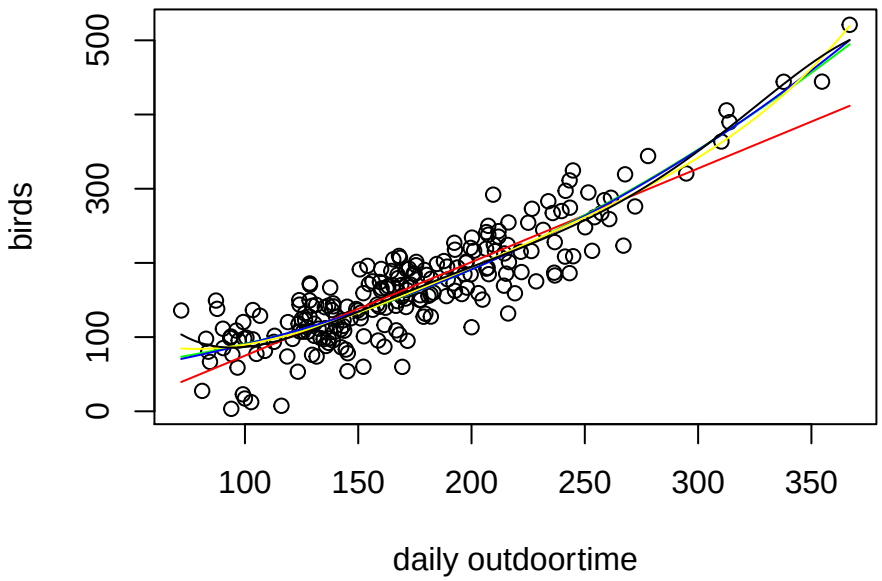
c)

We fit polynomials of degree 1, 2, ..., 10 to model the relationship between daily outdoor time and birds killed. The first plot below shows the polynomial fit of degree 1 through 5 and the second plot shows the polynomial fit of degree 6 through 10. We have used different colors in the two plots for the different polynomials:

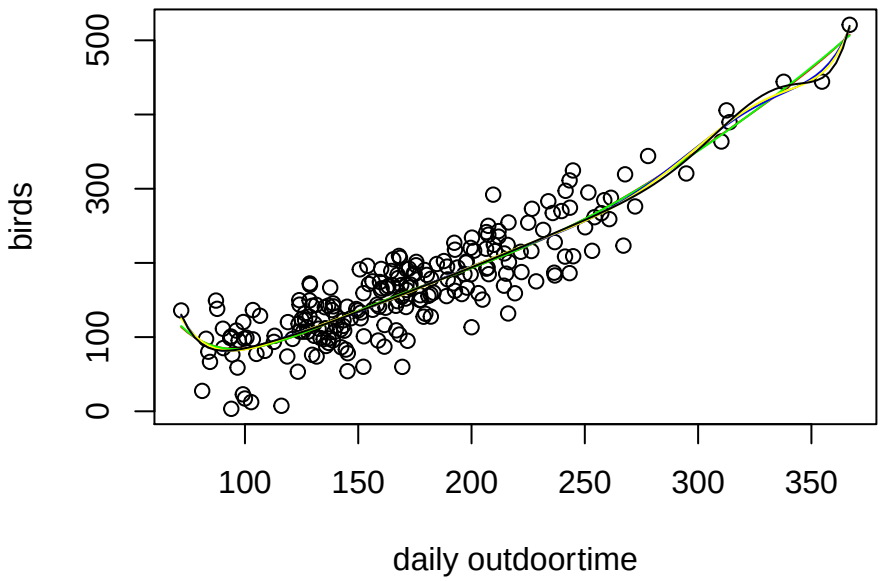
- Degree 1 and 6 are red
- Degree 2 and 7 are green
- Degree 3 and 8 are blue
- Degree 4 and 9 are yellow
- Degree 5 and 10 are black

In the daily outdoortime interval from 130 to 260 all polynomials are roughly linear. Approaching the lower and upper limits of the range the polynomials start to display non-linear behaviour.

```
# Create fits d=1,2,...,10
fit1 <- lm(birds ~ poly(daily.outdoortime, 1, raw = T), data = catdat.train)
fit2 <- lm(birds ~ poly(daily.outdoortime, 2, raw = T), data = catdat.train)
fit3 <- lm(birds ~ poly(daily.outdoortime, 3, raw = T), data = catdat.train)
fit4 <- lm(birds ~ poly(daily.outdoortime, 4, raw = T), data = catdat.train)
fit5 <- lm(birds ~ poly(daily.outdoortime, 5, raw = T), data = catdat.train)
fit6 <- lm(birds ~ poly(daily.outdoortime, 6, raw = T), data = catdat.train)
fit7 <- lm(birds ~ poly(daily.outdoortime, 7, raw = T), data = catdat.train)
fit8 <- lm(birds ~ poly(daily.outdoortime, 8, raw = T), data = catdat.train)
fit9 <- lm(birds ~ poly(daily.outdoortime, 9, raw = T), data = catdat.train)
fit10 <- lm(birds ~ poly(daily.outdoortime, 10, raw = T), data = catdat.train)

# Create grid and predictions d=1,2,...,10
pred_grid <- data.frame(daily.outdoortime = seq(min(catdat.train[[c("daily.outdoortime")]]),
    max(catdat.train[[c("daily.outdoortime")]]), length.out = 100))

pred1 <- predict(fit1, newdata = pred_grid)
pred2 <- predict(fit2, newdata = pred_grid)
pred3 <- predict(fit3, newdata = pred_grid)
pred4 <- predict(fit4, newdata = pred_grid)
pred5 <- predict(fit5, newdata = pred_grid)
pred6 <- predict(fit6, newdata = pred_grid)
pred7 <- predict(fit7, newdata = pred_grid)
pred8 <- predict(fit8, newdata = pred_grid)
pred9 <- predict(fit9, newdata = pred_grid)
pred10 <- predict(fit10, newdata = pred_grid)

# Plot polynomials degree 1 through 5
{
    plot(catdat.train[[c("daily.outdoortime")]], catdat.train[[c("birds")]], xlab = "daily outdoortime"
        ylab = "birds")
    lines(x = pred_grid[["daily.outdoortime"]], y = pred1, col = "red")
    lines(x = pred_grid[["daily.outdoortime"]], y = pred2, col = "green")
    lines(x = pred_grid[["daily.outdoortime"]], y = pred3, col = "blue")
    lines(x = pred_grid[["daily.outdoortime"]], y = pred4, col = "yellow")
    lines(x = pred_grid[["daily.outdoortime"]], y = pred5)
}
```




```
# Plot polynomials degree 6 through 10
{
    plot(catdat.train[[c("daily.outdoortime")]], catdat.train[[c("birds")]], xlab = "daily outdoortime"
        ylab = "birds")
    lines(x = pred_grid[["daily.outdoortime"]], y = pred6, col = "red")
    lines(x = pred_grid[["daily.outdoortime"]], y = pred7, col = "green")
    lines(x = pred_grid[["daily.outdoortime"]], y = pred8, col = "blue")
    lines(x = pred_grid[["daily.outdoortime"]], y = pred9, col = "yellow")
    lines(x = pred_grid[["daily.outdoortime"]], y = pred10)
}
```



We see a clear improvement in going from a 1st degree polynomial to a 2nd degree polynomial. The second degree polynomial catches the increase in birds killed at high values of daily outdoor time. Further increases in the polynomial degree leads to a slow decrease in MSE, but no notable improvements. Higher degree polynomials can do no worse in their predictions than lower degree polynomials, but whatever approvement in MSE above degree 2 is probably only overfitting.

Model	MSE training
1st degree	1369.9965211
2nd degree	1199.2015957
3rd degree	1198.4987697
4th degree	1188.2270371
5th degree	1175.9807085
6th degree	1173.4847242
7th degree	1173.4371218
8th degree	1166.4368012
9th degree	1166.0188244
10th degree	1164.3676256

Problem 3

a)

True, True, False, False

b)

We are left with the tree split at age<81.5, and age<46.5.

This is because when we prune the tree we remove the splits that have the least impact, while favoring smaller and smaller trees. Furthermore the pruning happens in a nested fashion, so for each time we find a new subtree we choose an internal node and snip away all the subbranches extending from this node. We see from the cv deviance plot that we first prune away one split, and then we prune away two splits at once. Then since the deviance stays the same at both times, we immediately know that the first pruning happens at (country: indonesia, japan, Korea), and the next at (sex:female), since if the first had happened at (age<48.5), then the next would have to be at (age<46.5). However then the pruning algorithm would rather go directly from the full tree to pruning at (age<46.5).

c)

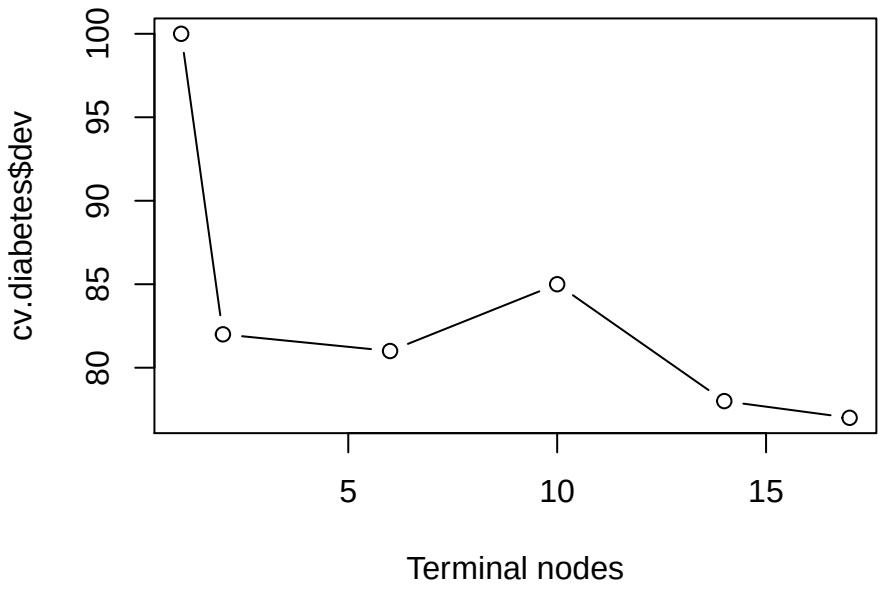
(i)

```
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)
```

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

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"
```

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




```
prune.diabetes = prune.misclass(t.diabetes, best = 14)

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
##                   0 126  28
##                   1  29  49
```

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

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

We apply 10-fold cost-complexity pruning and observe that the CV error is lowest for the full tree, i.e. 17 terminal nodes. However by pruning to 14 terminal nodes, the CV error only increases slightly (from 77 to 78) and we therefore choose the simpler tree with 14 terminal nodes. This gives us a misclassification error on the test set of about 0.246.

(ii)

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

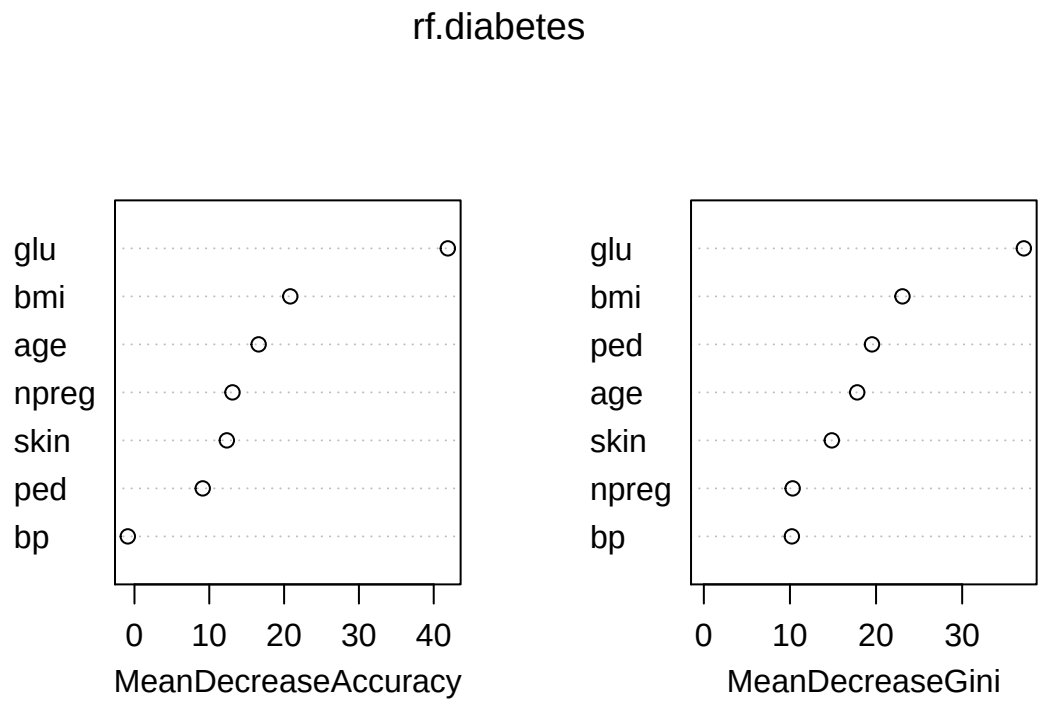
```
rf.diabetes.pred = predict(rf.diabetes, d.test, type = "class")
misclass.rf = table(rf.diabetes.pred, d.test$diabetes)
misclass.rf
```

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

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

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

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



We use the more advanced method Random Forest, with the parameters ntree=1000, and mtry=3. We use mtry=3 since we have p=7 predictors and the recommended mtry for classification is $\sqrt{p} = \sqrt{7} \approx 3$. For the ntree parameter we have simply set ntree high enough that the OOB error do not decrease substantially when increasing ntree anymore (Do not have to worry about overfitting here). The test set error is then about 0.233 a slight increase from the single tree. 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 $i = 1, \ldots, p$. 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)))
}

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 = "")
```

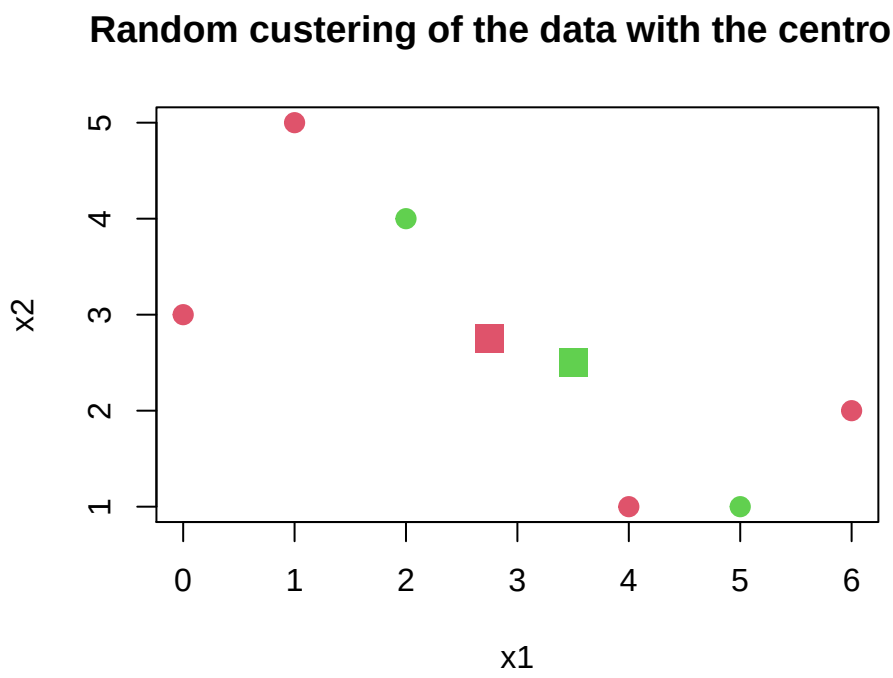


Figure 1: A random clustering of the data being the round points, and the centroids being the square points.

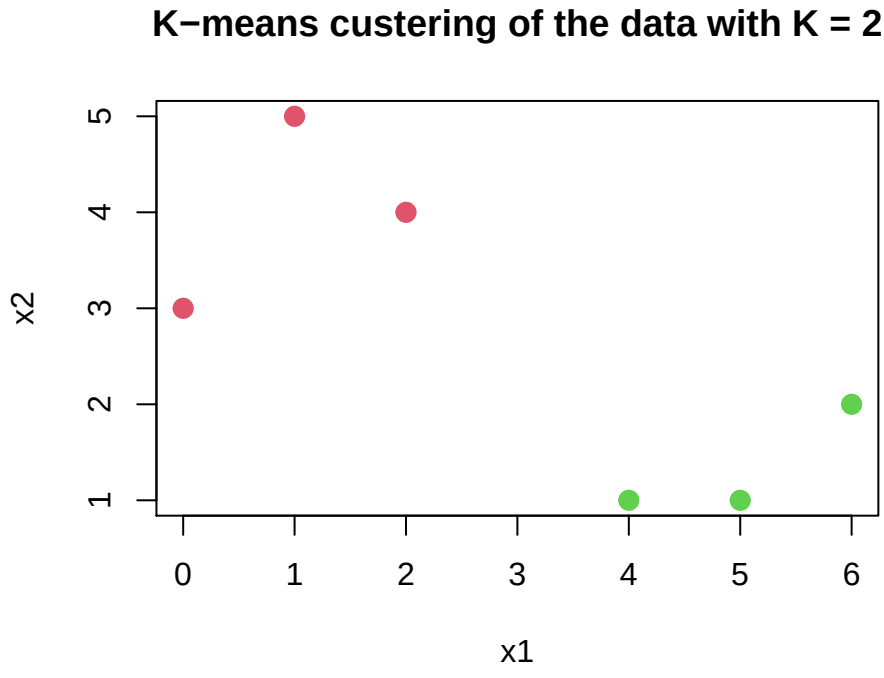


Figure 2: The K-means clustering for the data, with $K = 2$.

```
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)
```

c)

```
par(mfrow = c(2, 3))
gene.dist.ec = stats::dist(GeneData, method = "euclidean")
gene.dist.corr = as.dist(1 - cor(t(GeneData)))

hc.Gene.EcC = hclust(gene.dist.ec, method = "complete")
hc.Gene.EcS = hclust(gene.dist.ec, method = "single")
hc.Gene.EcA = hclust(gene.dist.ec, method = "average")
hc.Gene.CorrC = hclust(gene.dist.corr, method = "complete")
hc.Gene.CorrS = hclust(gene.dist.corr, method = "single")
hc.Gene.CorrA = hclust(gene.dist.corr, method = "average")

plot(hc.Gene.EcC, main = "Complete Linkage", xlab = "Euclidean distance", ylab = "",
    sub = "")
plot(hc.Gene.EcS, main = "Single Linkage", xlab = "Euclidean distance", ylab = "",
    sub = "")
plot(hc.Gene.EcA, main = "Average Linkage", xlab = "Euclidean distance", ylab = "",
    sub = "")
plot(hc.Gene.CorrC, main = "Complete Linkage", xlab = "Correlation distance", ylab = "",
    sub = "")
plot(hc.Gene.CorrS, main = "Single Linkage", xlab = "Correlation distance", ylab = "",
    sub = "")
plot(hc.Gene.CorrA, main = "Average Linkage", xlab = "Correlation distance", ylab = "",
    sub = "")
```

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
```

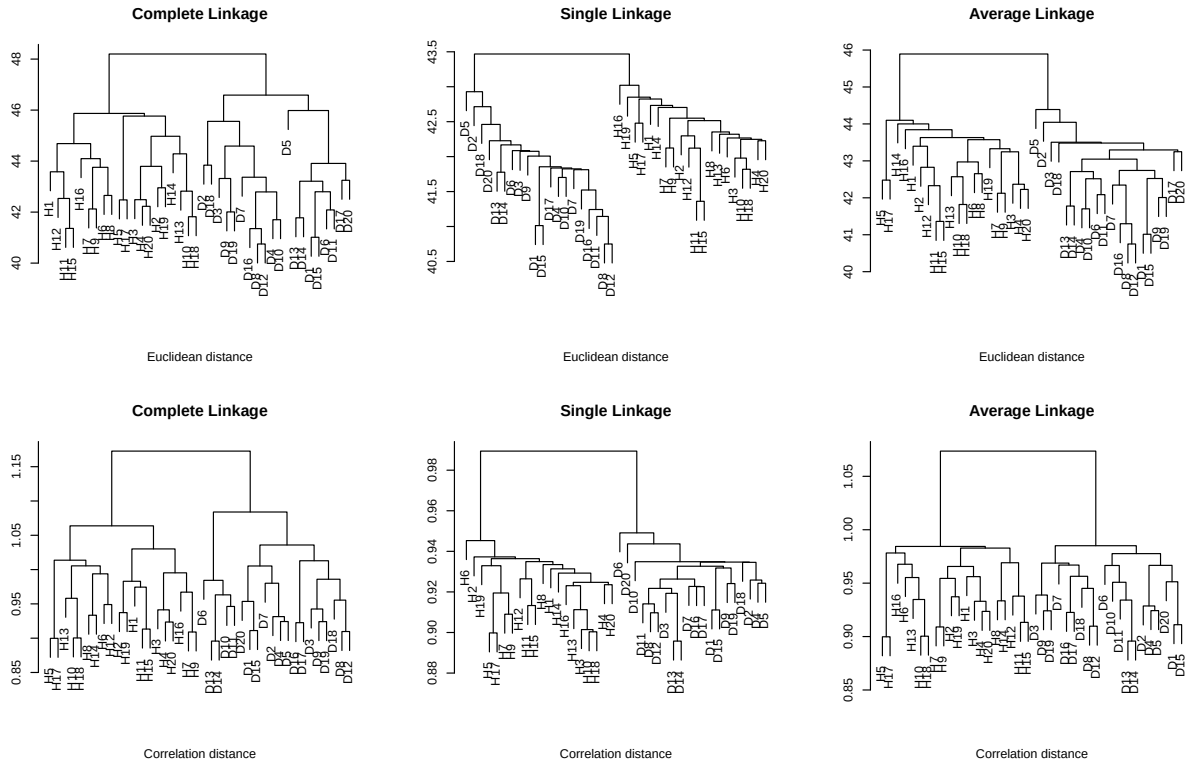


Figure 3: Hierarchical clustering with Euclidean and Correlation based distances.

```
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
```

We know that the first 20 tissue samples are healthy and the last 20 are diseased. From above we observe that all the linkage and distance measures in the hierarchical clustering method manage to separate the two groups perfectly.

e)

(i)

```
pc.Gene = prcomp(GeneData)

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

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

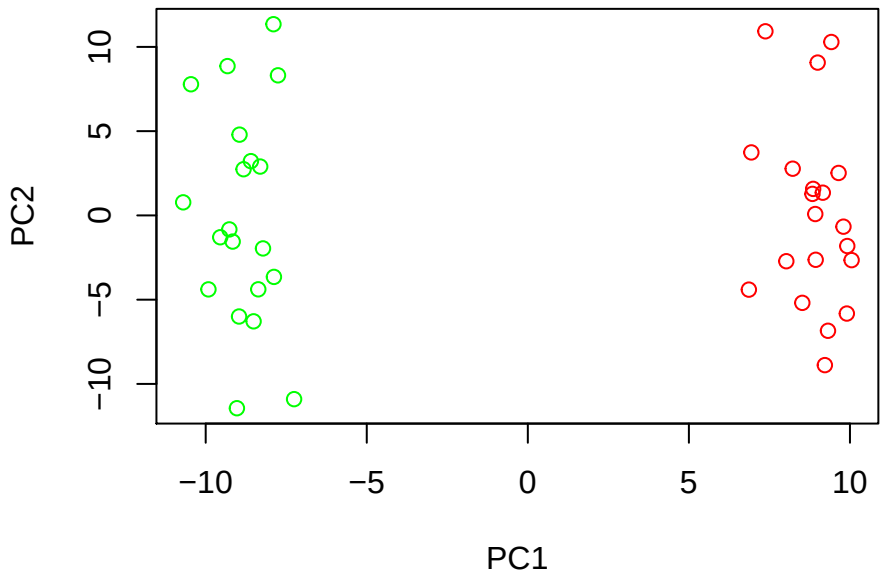


Figure 4: PC1 and PC2 plotted against each other for the Gene Data. The green dots are healthy tissue samples, whilst the red dots are diseased tissue samples.

(ii)

```
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
```

From the summary printout of the PCA we find that the first 5 PCs explains about 21.1% of the variance.

f)

```
tail(sort(abs(pc.Gene$rotation[, 1])), 10)
```

```
##       G509       G584       G538       G593       G551       G600       G590
## 0.08661015 0.08690858 0.08745400 0.08758616 0.08768360 0.09167322 0.09173169
##       G565       G589       G502
## 0.09183823 0.09449766 0.09485044
```

We observe in figure 4 that PC1 alone is enough to seperate the two groups, moreover we know that PC1 explains the most variance out of all the PC. Thus we find the genes that vary the most across the two groups by extracting the genes with the highest loadings (in absolute value) from PC1. From above we get that G502, G589, G565, G590, G600 vary the most.