CLASSIFICATION PROBLEMS: BIGFOOT

Project 3

Zaide Montes Dmytro Perepolkin

LUNDS UNIVERSITET

Group 1 Lund University
Lund, Sweden
https://lu.se

```r
library(MASS) # for LDA
library(tidyverse)
library(broom)
library(corrgram) # visualisation of correlations
library(lmtest)  # more linear regression tools
library(hrbrthemes) #ggplot styling
library(GGally) # Pair plot
library(ggplot2) # ggplot 2
library(hrbrthemes) # theme for ggplot
library(kableExtra)
library(leaps)
library(glmnet)
library(patchwork)
library(plotmo)
library(randomForest) # for random forest chapter
library(pls) # for principal component regression
library(rsample) # for cross-validation
library(modelr) # evaluation of models
library(splines)
library(MASS) # for LDA
library(pROC)
library(e1071)
library(ggdendro)
library(factoextra)

ggplot2::theme_set(theme_classic())

knitr::opts_chunk$set(dev = 'png')
options(device = function(file, width, height) {
  png(tempfile(), width = width, height = height)
})
```

1 Regression and tree-based models

1.1 Introduction

1.1.1 Data description

The Bigfoot Field Researchers Organization (BFRO) is an organization dedicated to investigating the bigfoot mystery, and for years they have been collecting reported sightings in a database. They manually classify their reports into

- Class A: Clear sightings in circumstances where misinterpretation or misidentification of other animals can be ruled out with greater confidence
- Class B: Incidents where a possible bigfoot was observed at a great distance or in poor lighting conditions and incidents in any other circumstance that did not afford a clear view of the subject.

However, they wonder if this can be automated and done by a classification algorithm instead. So in this task, you will set up a few different classification algorithms for this aim, and evaluate their performance.

In this task we are using a slightly simplified version of the data set where we have extracted some variables of interest and deleted observations that are missing any of these variables of interest.

```
bigfoot <- read_csv("data/bigfoot.csv", show_col_types = FALSE) %>%
  mutate(class=factor(class, levels=c(1,0), labels=c("Class A", "Class B")))
```

```
set.seed(42)
train_idx <- sample(1:nrow(bigfoot), nrow(bigfoot)*0.7)
test_idx <- (-train_idx)
#bigfoot %>% summary()
```

1.2 Logistic regression

Logistic regression is a popular and widely used statistical technique that is particularly suitable for binary classification problems, where the response variable is dichotomous

(e.g., yes/no, true/false, 1/0). In this case, the response variable is whether a bigfoot sighting can be classified as Class A or Class B. The classification of the class is based on the values of one or more predictor variables. These predictor variables can include any number of factors that might be relevant to the classification, such as the location of the sighting, the time of day, or the weather conditions.

We try a simple logistic regression model

```
log_reg_bf <- glm(class ~ .,
                  data = bigfoot, subset=train_idx,
                  family = "binomial")
summary(log_reg_bf)
```

```
Call:
glm(formula = class ~ ., family = "binomial", data = bigfoot,
    subset = train_idx)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-2.1847  -1.0050   0.4373   1.0228   2.0069

Coefficients:
             Estimate Std. Error z value Pr(>|z|)
(Intercept) -0.797520   0.423895  -1.881 0.059917 .
longitude    0.003773   0.003509   1.075 0.282365
latitude     0.028761   0.010030   2.868 0.004135 **
visibility   0.031925   0.024322   1.313 0.189318
furTRUE     -0.506978   0.139263  -3.640 0.000272 ***
howlTRUE     0.758967   0.193460   3.923 8.74e-05 ***
sawTRUE     -1.356510   0.098254 -13.806  < 2e-16 ***
heardTRUE    1.117660   0.099732  11.207  < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 2948.4  on 2126  degrees of freedom
Residual deviance: 2495.1  on 2119  degrees of freedom
AIC: 2511.1

Number of Fisher Scoring iterations: 4
```

The predictor variables `furTRUE`, `howlTRUE`, `sawTRUE`, and `heardTRUE` were all statistically significant.

There are no tuning parameters in logistic regression, so we will just attempt to predict on the validation set and measure the performance.

```
preds_glm <- predict(log_reg_bf, newdata = bigfoot[test_idx,], type="response")
preds_glm_class <- ifelse(preds_glm <= 0.5, "Class A", "Class B")
```

```
table(preds_glm_class,
      bigfoot$class[test_idx],
      dnn = c("predicted","observed" )
      ) %>% prop.table() %>% round(2)
```

```
         observed
predicted Class A Class B
  Class A    0.32    0.15
  Class B    0.17    0.35
```

Table above shows the proportions confusion matrix for the GLM model. This is not an awfully great classifier.

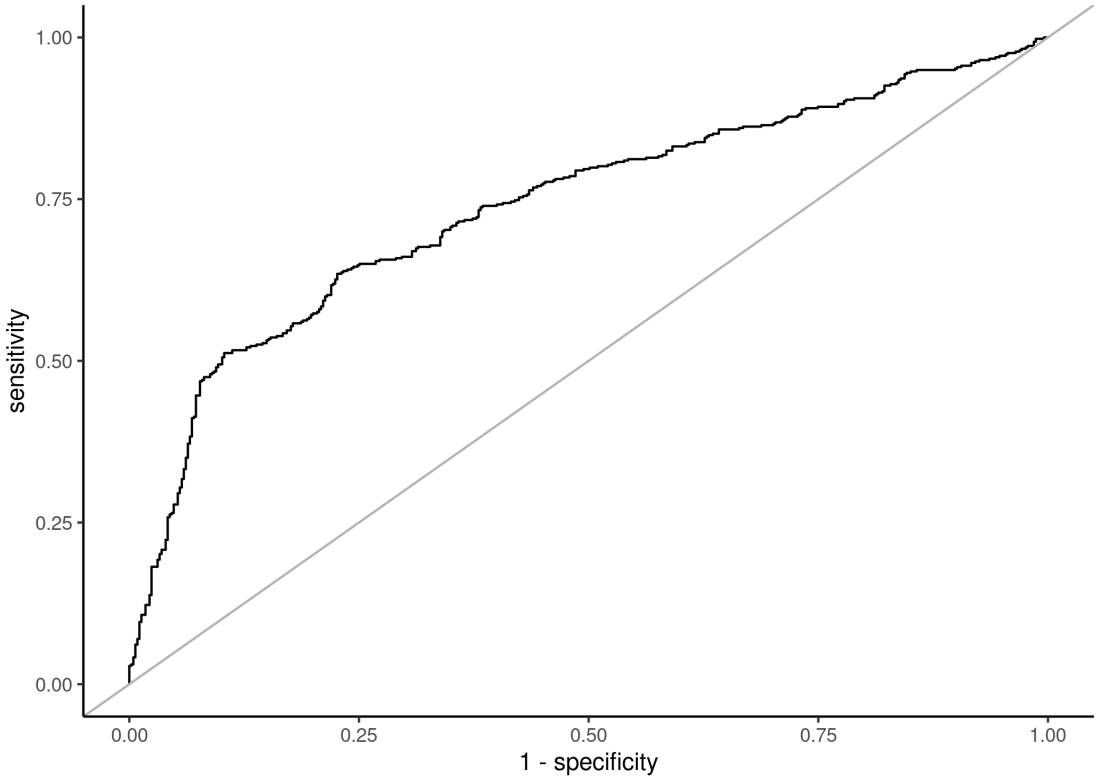
The area under the ROC curve (AUC) is a measure of the classifier's performance. It represents the probability that the classifier will rank a randomly chosen positive instance higher than a randomly chosen negative instance. An AUC of 0.5 indicates that the classifier is performing no better than chance, while an AUC of 1.0 indicates perfect classification. An area under the curve equal to 73.2% indicates that the classifier has a moderate discriminatory power. It performs better than random classification, but there is still room for improvement. A higher AUC value, closer to 1.0, indicates better classifier performance. Let's look at the ROC curve.

```
glm_roc <- roc(bigfoot$class[test_idx],
               preds_glm)
```

```
Setting levels: control = Class A, case = Class B
```

```
Setting direction: controls < cases
```

```
ggroc(glm_roc, legacy.axes = TRUE)+
  geom_abline(slope=1, intercept = 0, color="grey70")
```



The AUC for the logistic regression is 0.738827

1.3 LASSO regression

Now, for improving the interpretability of the logistic regression model, we performed a LASSO selection to identify or reduce the number of the predictors that contribute the most to explain the model.

```
x=model.matrix(class~.,data=bigfoot[train_idx,])[,-1]
y=bigfoot$class[train_idx]
lasso.cv <- cv.glmnet(x,y, alpha=1, family = "binomial")
```

```
plot(lasso.cv)
```

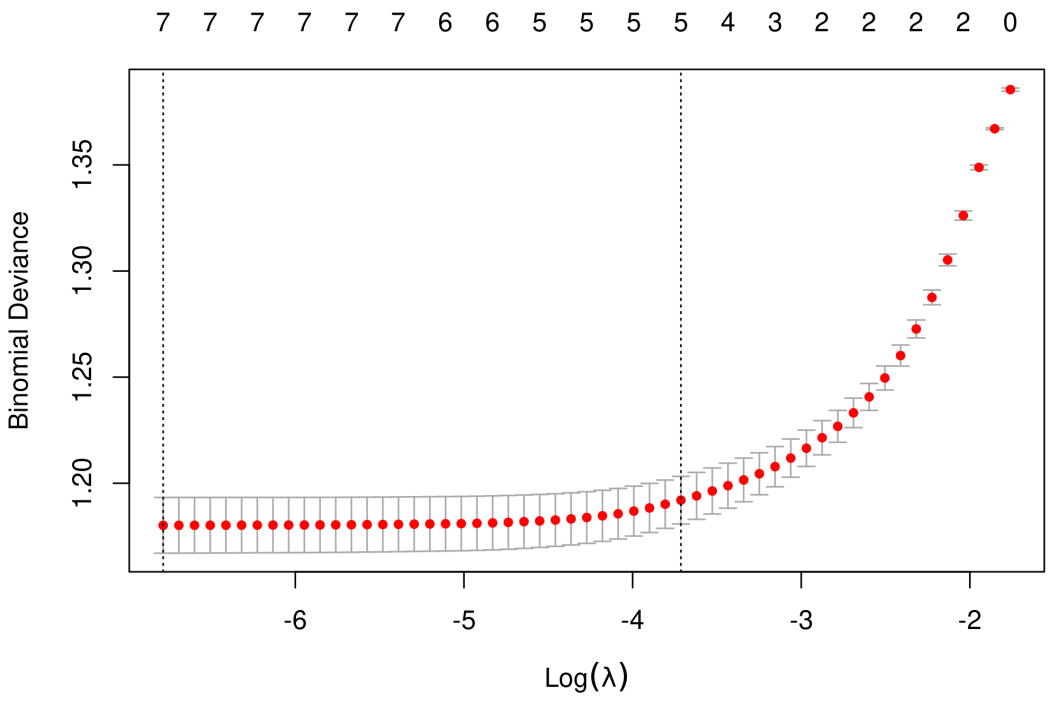


Figure 1.1: A graph of the LASSO model deviance for different lambdas.

In the @fig-lasso-plot1 the dashed lines are the $\ln(\lambda)$ values corresponding to the λ_{min} (left dashed line) and λ_{1se} (right dashed line).

The lambda value of $\lambda = 0.0243912$ is the penalty parameter that is used in LASSO to control the degree of shrinkage applied to the coefficients. A smaller lambda value results in less shrinkage and a larger lambda value results in more shrinkage.

```
lasso.fit <- glmnet(x,y,alpha=1,family = "binomial", lambda=lasso.cv$lambda.1se)
coef(lasso.fit)
```

```
8 x 1 sparse Matrix of class "dgCMatrix"
                      s0
(Intercept)  0.111330455
longitude    .
latitude     0.002567278
visibility   .
furTRUE     -0.159547910
howlTRUE     0.356749560
```

```
sawTRUE     -1.150262730
heardTRUE    0.896090758
```

Regarding the LASSO coefficients, the variables longitude and visibility with a coefficient of "." are the ones that have been shrunk to zero by the LASSO regularization, indicating that they have been excluded from the final model.

```
xval <- model.matrix(class~., data=bigfoot[test_idx,])[,-1]

lasso_preds <- predict(lasso.fit, newx = xval, type = "response")
lasso_classifier <- ifelse(lasso_preds <= 0.5, "Class A", "Class B")
```

Then, to evaluate the performance of a binary classification model like logistic regression we built a confusion matrix to assess how well the model was able to classify observations into their correct categories (Class A or Class B in this case).

```
table(lasso_classifier, bigfoot$class[test_idx],
      dnn = c("predicted","observed" )) %>% prop.table()
```

```
         observed
predicted   Class A   Class B
  Class A 0.3421053 0.1842105
  Class B 0.1567982 0.3168860
```

The diagonal cells of the table above represent the number of cases that were correctly classified. The off-diagonal cells represent the cases that were misclassified.

Finally, we use a ROC (Receiver Operating Characteristic) curve, a graphical representation of the performance of a binary classifier system as its discrimination threshold is varied. The ROC curve plots the true positive rate (TPR) against the false positive rate (FPR) for different classification thresholds.

```
lasso_roc <- roc(bigfoot$class[test_idx],     lasso_preds)
```

```
Setting levels: control = Class A, case = Class B
```

```
Warning in roc.default(bigfoot$class[test_idx], lasso_preds): Deprecated use a
matrix as predictor. Unexpected results may be produced, please pass a numeric
vector.
```

```
Setting direction: controls < cases
```

```
ggroc(lasso_roc, legacy.axes = TRUE)+
  geom_abline(slope=1, intercept = 0, color="grey70")
```

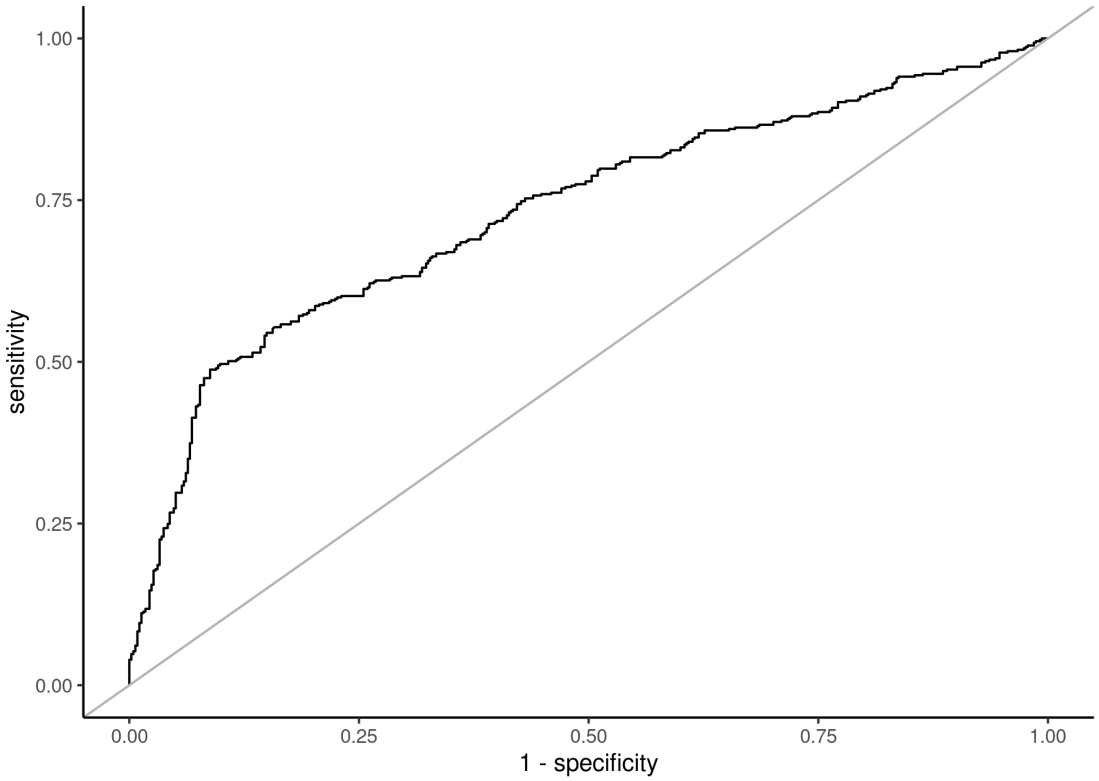


Figure 1.2: ROC curve for lasso regression

The AUC for the tuned LASSO regression is 0.7320004, which is not that much different from the logistic regression.

1.4 LDA/QDA

We fit linear discriminant analysis model to the training data with the response variable `class` modeled as a function of three predictor variables `saw`,`longitude` and `latitude`.

```
lda_mod1 <- lda(class~saw+longitude+latitude, data=bigfoot, subset = train_idx)
```

The output of the LDA function provided several pieces of information about the model. First, `prior probabilities of groups` (i.e., each level of the response variable). In

this case, there are two classes: Class A and Class B. The prior probabilities are the proportions of the training data that belong to each class. In this case, there are slightly more observations in Class B than in Class A. Second, `group means` shows the mean values of the predictor variables for each group. Finally, the `Coefficients of linear discriminants` shows how much each predictor variable contributes to the discriminant function. Positive coefficients indicate that higher values of the predictor variable are associated with Class B, while negative coefficients indicate that higher values of the predictor variable are associated with Class A. In this case, we can see that higher values of `longitude` and `latitude` are associated with Class B, while higher values of `saw` are associated with Class A.

Because this classification task includes discrimination between the two classes only, then there will be only a single set of LD values, which will separate the classes. We can now predict the class on the test (holdout) data.

```
lda_preds <- predict(lda_mod1, newdata=bigfoot[test_idx,], type = "response")$posterior[
lda_classifier <- ifelse(lda_preds <= 0.5, "Class A", "Class B")


table(
  lda_classifier, bigfoot$class[test_idx],
  dnn=c("predicted", "observed")
) %>% prop.table()
```

```
         observed
predicted   Class A   Class B
  Class A 0.3486842 0.2006579
  Class B 0.1502193 0.3004386
```

The accuracy seem to be worse than the regressions. Let's look at the AUC

```
lda_roc <- roc(bigfoot$class[test_idx],
               lda_preds)
```

```
Setting levels: control = Class A, case = Class B
```

```
Setting direction: controls < cases
```

```
ggroc(lda_roc, legacy.axes = TRUE)+
  geom_abline(slope=1, intercept = 0, color="grey70")
```

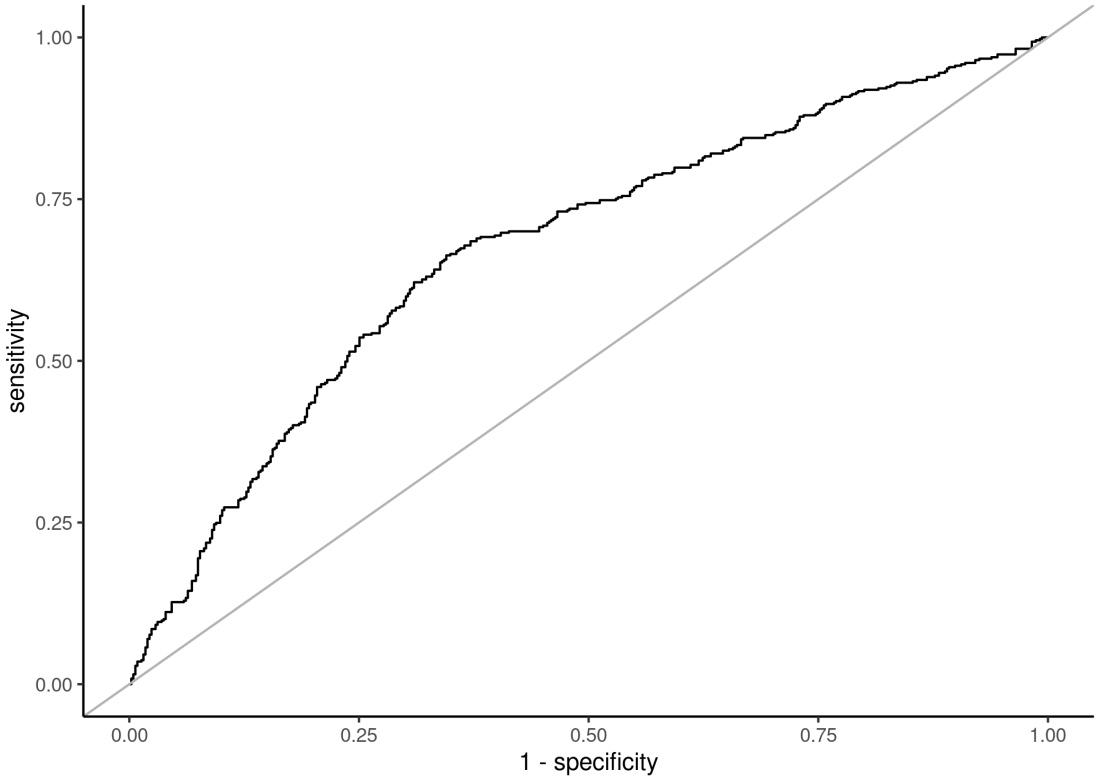


Figure 1.3: ROC for LDA

The AUC for LDA is a meager 0.6747349.

1.5 Random Forest

Tree-based models are naturally suited for classification, because they create partitions of data. We build a random-forest model and tune the `mtry` parameter.

```
set.seed(42)
rf_mod <- tuneRF(x,y, mtryStart = 3, ntreeTry = 1000, doBest = TRUE, importance=TRUE)
```

```
mtry = 3  OOB error = 34.7%
Searching left ...
mtry = 2    OOB error = 32.72%
0.05691057 0.05
mtry = 1    OOB error = 32.02%
0.02155172 0.05
```

```
Searching right ...
mtry = 6    OOB error = 36.06%
-0.1020115 0.05
```

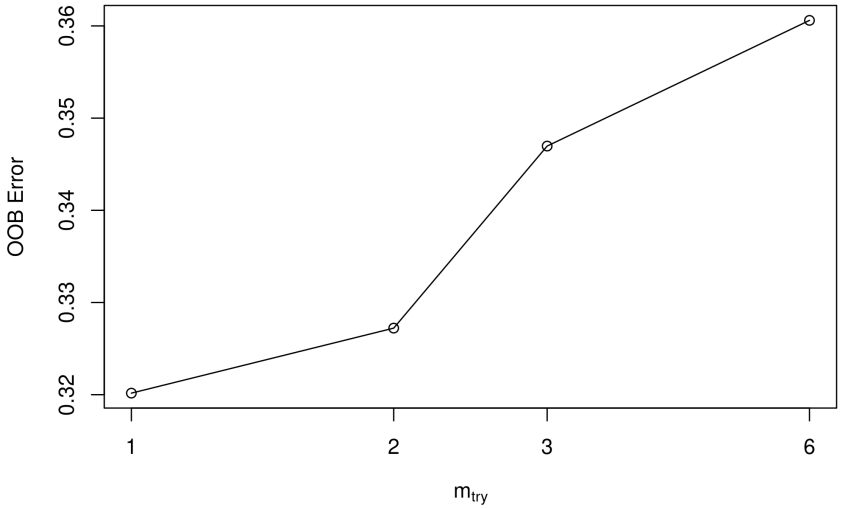


Figure 1.4: Tuning mtry parameter of random forest

```
varImpPlot(rf_mod)
```

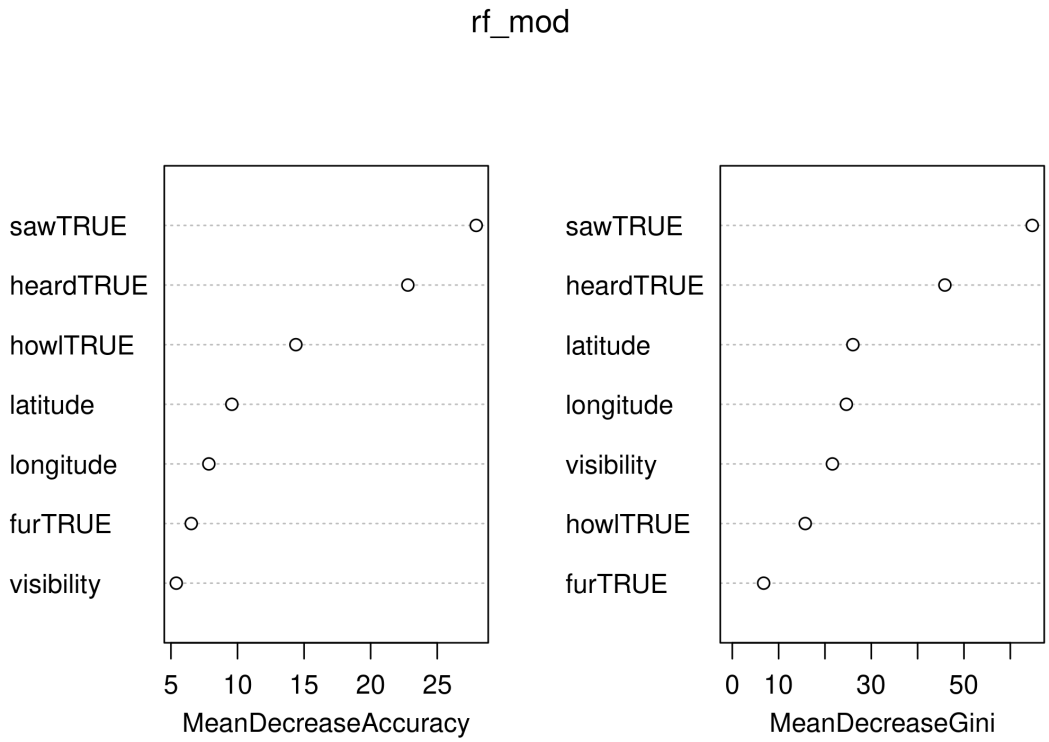


Figure 1.5: Variable importance, random forest

We can now predict on the validation set

```
rf_preds <- predict(rf_mod, newdata = xval, type="prob")[,2]
rf_classifier <- ifelse(rf_preds <= 0.5, "Class A", "Class B")

table(
  rf_classifier, bigfoot$class[test_idx],
  dnn=c("predicted", "observed")
) %>% prop.table()
```

```
          observed
predicted   Class A   Class B
  Class A 0.3958333 0.2017544
  Class B 0.1030702 0.2993421
```

The accuracy seems to be roughly on par with the regressions. Let's look at the AUC

```
rf_roc <- roc(bigfoot$class[test_idx],
              rf_preds)
```

```
Setting levels: control = Class A, case = Class B
```

```
Setting direction: controls < cases
```

```
ggroc(rf_roc, legacy.axes = TRUE)+
  geom_abline(slope=1, intercept = 0, color="grey70")
```

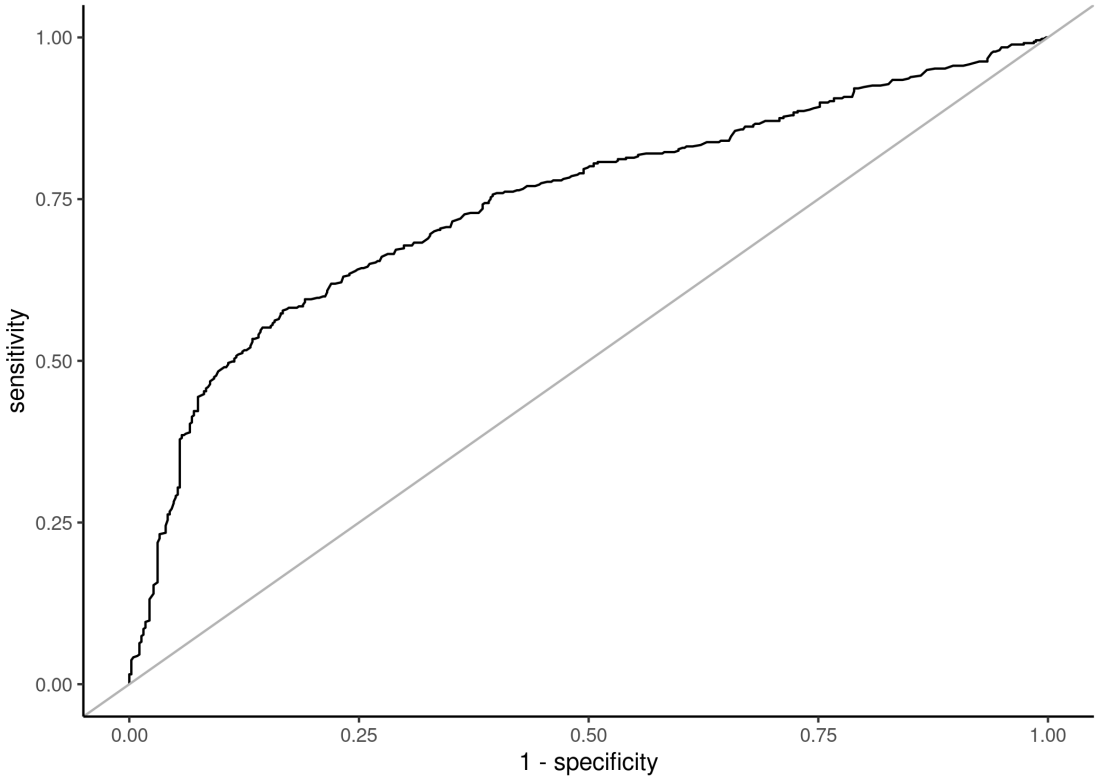


Figure 1.6: ROC for Random Forest

The AUC for the random forest model is 0.7433284

1.6 SVM

We start with linear kernel

```
svm_lin <- svm(class ~ ., data = bigfoot, subset=train_idx,
               kernel = "linear", cost = .01, scale = FALSE)
svm_lin_preds <- attributes(
  predict(svm_lin, newdata=bigfoot[test_idx,], decision.values = TRUE))$decision.values
```

```
lin_roc <- roc(
   as.numeric(bigfoot$class[test_idx]),
   as.vector(svm_lin_preds))
```

```
Setting levels: control = 1, case = 2
```

```
Setting direction: controls > cases
```

```
ggroc(lda_roc, legacy.axes = TRUE)+
  geom_abline(slope=1, intercept = 0, color="grey70")
```

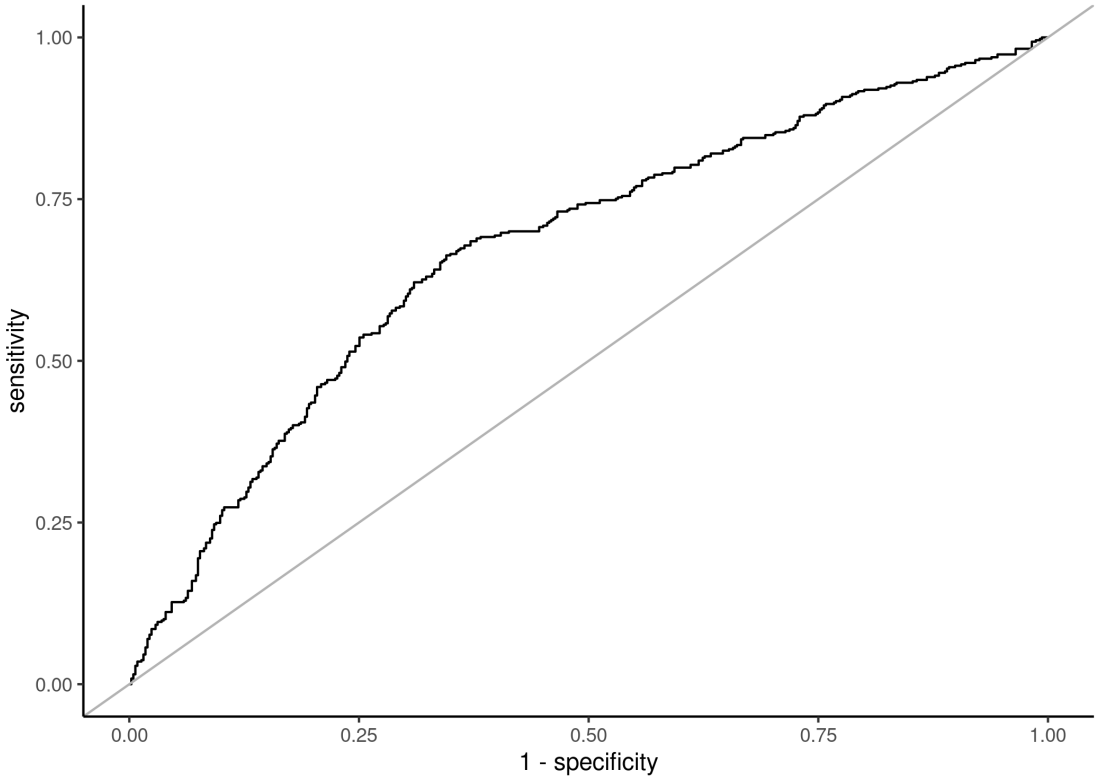


Figure 1.7: ROC for linear SVM

The AUC for the linear-kernel SVM 0.7260586

Let's try radial kernel

```
svm_rad <- svm(class ~ ., data = bigfoot, subset=train_idx,
               kernel = "radial", gamma=0.01, cost = .01, scale = FALSE)
svm_rad_preds <- attributes(
  predict(svm_lin, newdata=bigfoot[test_idx,], decision.values = TRUE))$decision.values
```

```
svm_rad_roc <- roc(
   as.numeric(bigfoot$class[test_idx]),
   as.vector(svm_lin_preds))
```

```
Setting levels: control = 1, case = 2
```

```
Setting direction: controls > cases
```

```
ggroc(svm_rad_roc, legacy.axes = TRUE)+
  geom_abline(slope=1, intercept = 0, color="grey70")
```

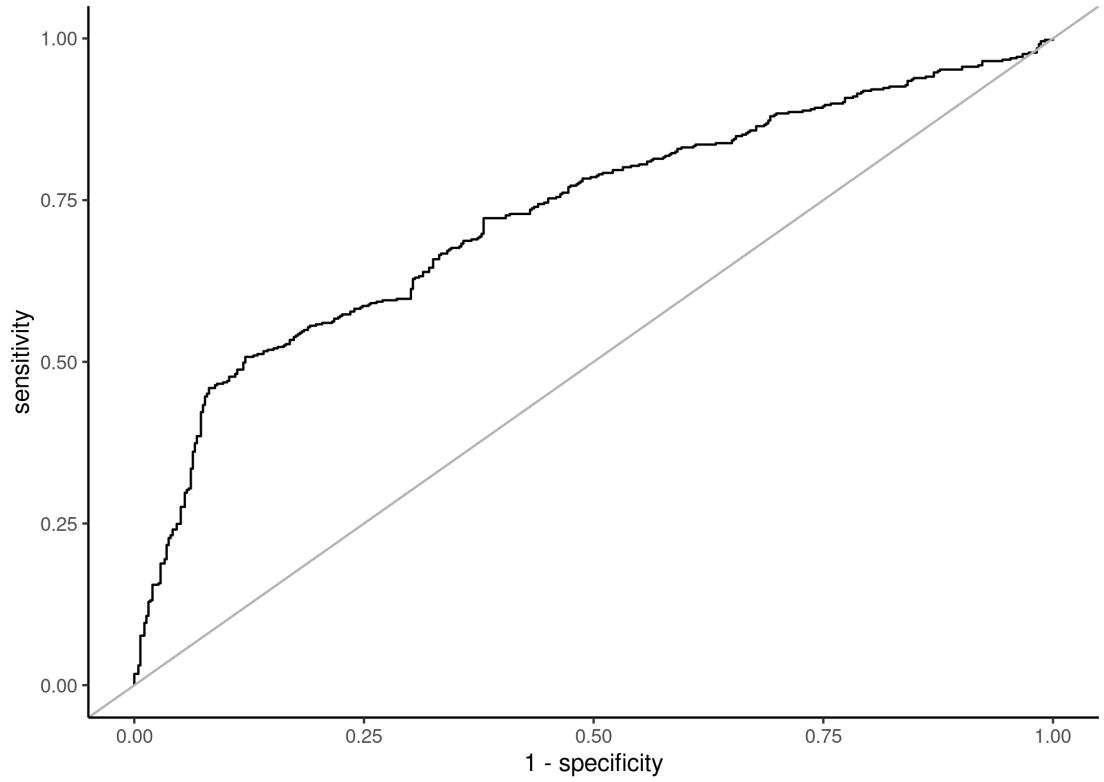


Figure 1.8: ROC for radial SVM

The AUC for the radial-kernel SVM 0.7260586. We can tune the SVM parameters (both `gamma` and `cost`).

```
set.seed(42)
svm_rad_tune <- tune("svm",
                  class~.,
                  data=bigfoot[train_idx, ],
                  kernel="radial",
                  ranges = list(
                    cost = c(0.01, 0.05, 0.1, 0.5, 1),
                    gamma = c(0.01, 0.05, 0.1, 0.5, 1)),
                  tunecontrol = tune.control(cross=3))
svm_bestrad_mod <- svm_rad_tune$best.model
```

```
svm_bestrad_preds <- attributes(
  predict(svm_bestrad_mod, newdata=bigfoot[test_idx,], decision.values = TRUE))$decision
```

```
svm_bestrad_roc <- roc(
   as.numeric(bigfoot$class[test_idx]),
   as.vector(svm_bestrad_preds))
```

```
Setting levels: control = 1, case = 2
```

```
Setting direction: controls > cases
```

```
ggroc(svm_bestrad_roc, legacy.axes = TRUE)+
  geom_abline(slope=1, intercept = 0, color="grey70")
```

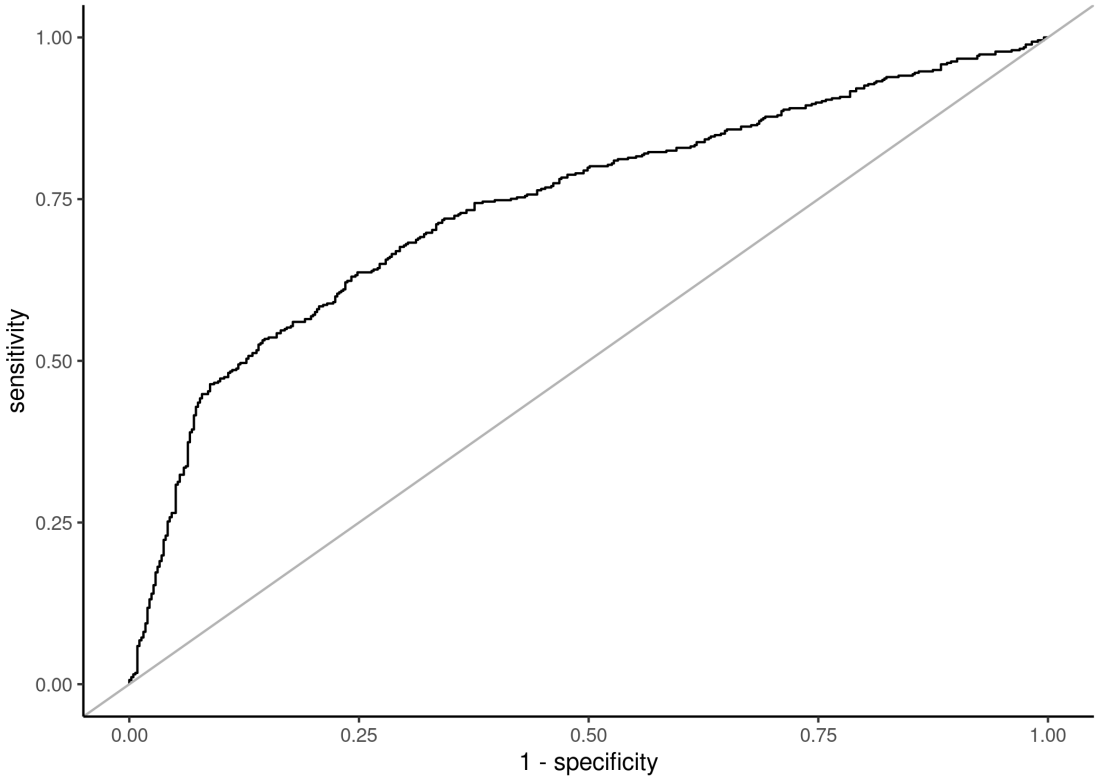


Figure 1.9: ROC for tuned radial SVM

The AUC for the tuned radial SVM is 0.7398851

1.7 Clustering

The GeneEx.csv data contains gene-expressions for 1000 genes for 40 samples. The 20 first samples come from healthy patients (controls) while the 20 last come from a diseased group (cases). The gene expression of 1000 different genes are collected in the 1000 columns (named ”G1” to ”G1000”).

We read in the data and calculated distances between observations for the hierarchical clustering. We measure both Euclidean distance and correlation-based distance.

```
genexp_df <- read.csv("data/GeneEx.csv", header=TRUE)
lbls <- rep(c("controls", "cases"), each=20)
genexp_m <- genexp_df %>% as.matrix()
genexp_scaled_m <- scale(genexp_m)
```

```
genexp_dist <- dist(genexp_scaled_m)
genexp_cor_dist <- as.dist(1 - cor(t(genexp_scaled_m)))
```

For k-means clustering we want to make sure we can identify the optimal number of clusters in the data. We fit 1 to 10 clusters and compare mean within squared error per cluster.

```
set.seed(42)
N <- 10
toterr <- vector("numeric", N)
for(i in seq(N))
  toterr[i] <- kmeans(genexp_scaled_m,  i,  nstart = 20, iter.max = 100)$tot.withinss/i
plot(seq(N), toterr, type="l")
```

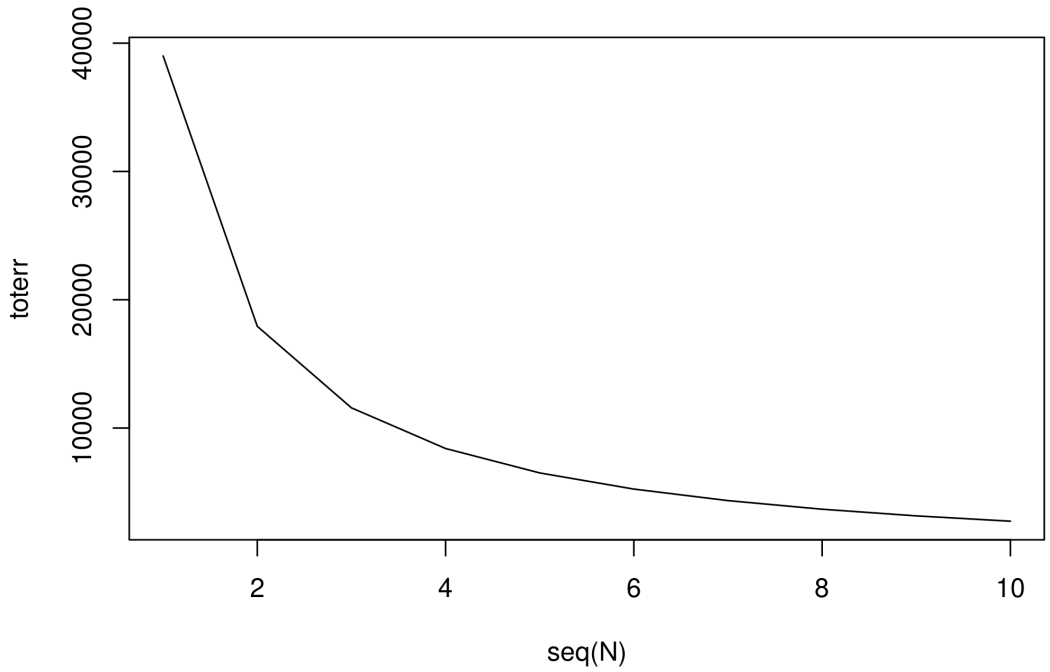


Figure 1.10: Validation of number of clusters by mean within-cluster error

It seems by the look of Figure 1.10 like the elbow is at 2 clusters. Lets re-fit the kmeans with 2 clusters. We will also check if all of the cases and controls have been properly allocated.

```
km_clusters2 <- kmeans(genexp_scaled_m,  2,  nstart = 20, iter.max = 100)

tibble::tibble(clstr = km_clusters2$cluster,
               lbl=lbls) %>% table()
```

```
     lbl
clstr cases controls
    1     0       20
    2    20        0
```

Yes it seems like all controls are in one cluster and all cases are in the other cluster.

For hierarchical clustering we will start with the complete linkage algorithm using the Euclidean distance.

```
hc_out <- hclust(genexp_dist, method = "complete")
ggdendrogram(hc_out)
```

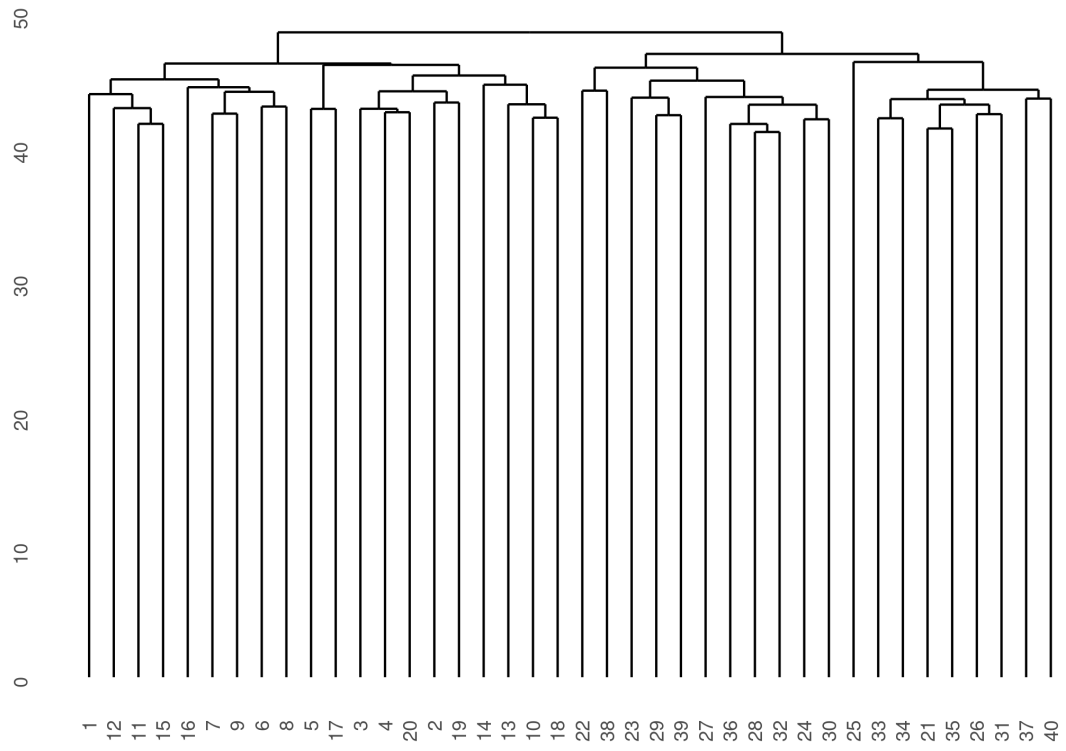


Figure 1.11: Hierarchical clustring. Euclidean distance, complete linkage

```
hc_cor_out <- hclust(genexp_cor_dist, method = "complete")
ggdendrogram(hc_cor_out)
```

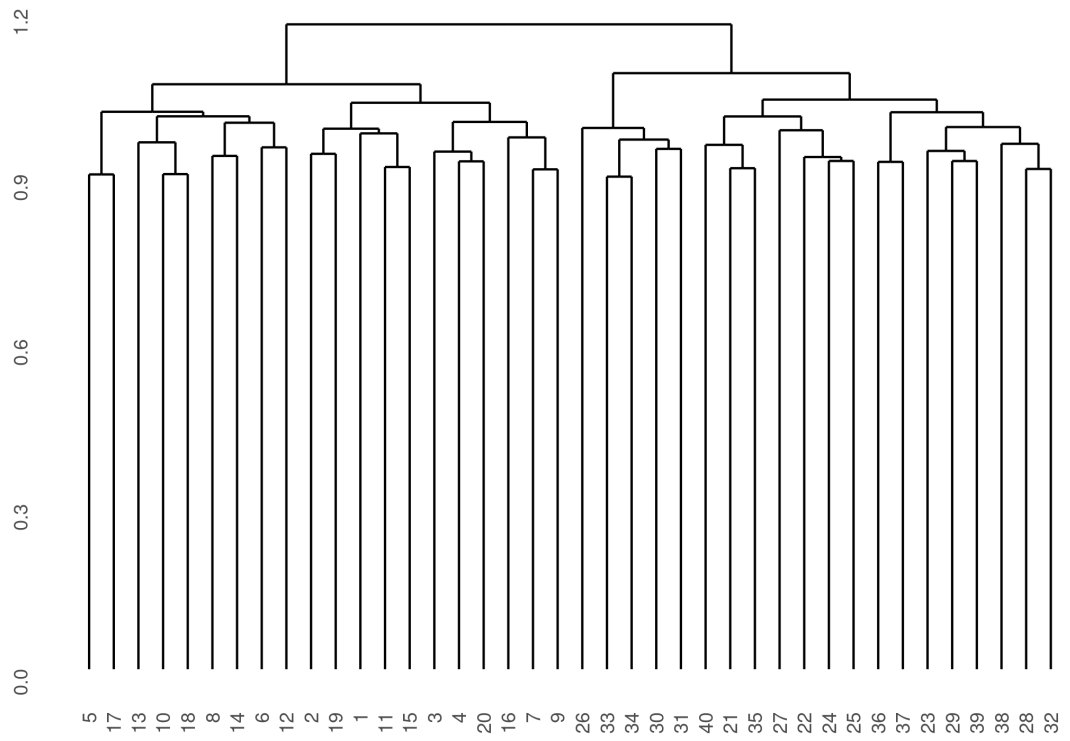


Figure 1.12: Hierarchical clustring. Correlation distance, complete linkage

Finally we compare if our kmeans-based algorithm identified the same clusters as the hierarchical clustering.

```
hc_clusters <- cutree(hc_out, 2)
hc_cor_clusters <- cutree(hc_cor_out, 2)
table(km_clusters2$cluster, hc_clusters)
```

```
   hc_clusters
     1  2
  1 20  0
  2  0 20
```

```
table(km_clusters2$cluster, hc_cor_clusters)
```

```
   hc_cor_clusters
     1  2
  1 20  0
  2  0 20
```

There's a complete match of clusters between the algorithms.

1.8 Appendix A. Data cleaning script

The `bigfoot` data for this assignment was cleaned using the following script

```
bigfoot_original <- readr::read_csv("https://raw.githubusercontent.com/rfordatascience/t

bigfoot <- bigfoot_original %>%
# Select the relevant covariates:
select(classification , observed , longitude , latitude , visibility) %>%
# Remove observations of class C ( second - or third hand accounts ):
filter(classification != "Class C") %>%
# Turn into 0/1, 1 = Class A, 0 = Class B:
mutate(class = ifelse(classification == "Class A", 1, 0)) %>%
# Create new indicator variables for some words from the description:
mutate(fur = grepl("fur", observed),
howl = grepl("howl", observed),
saw = grepl("saw", observed),
heard = grepl("heard", observed )) %>%
# Remove unnecessary variables:
dplyr::select(-c("classification", "observed")) %>%
# Remove any rows that contain missing values:
drop_na()

write_csv(bigfoot, "data/bigfoot.csv")
```

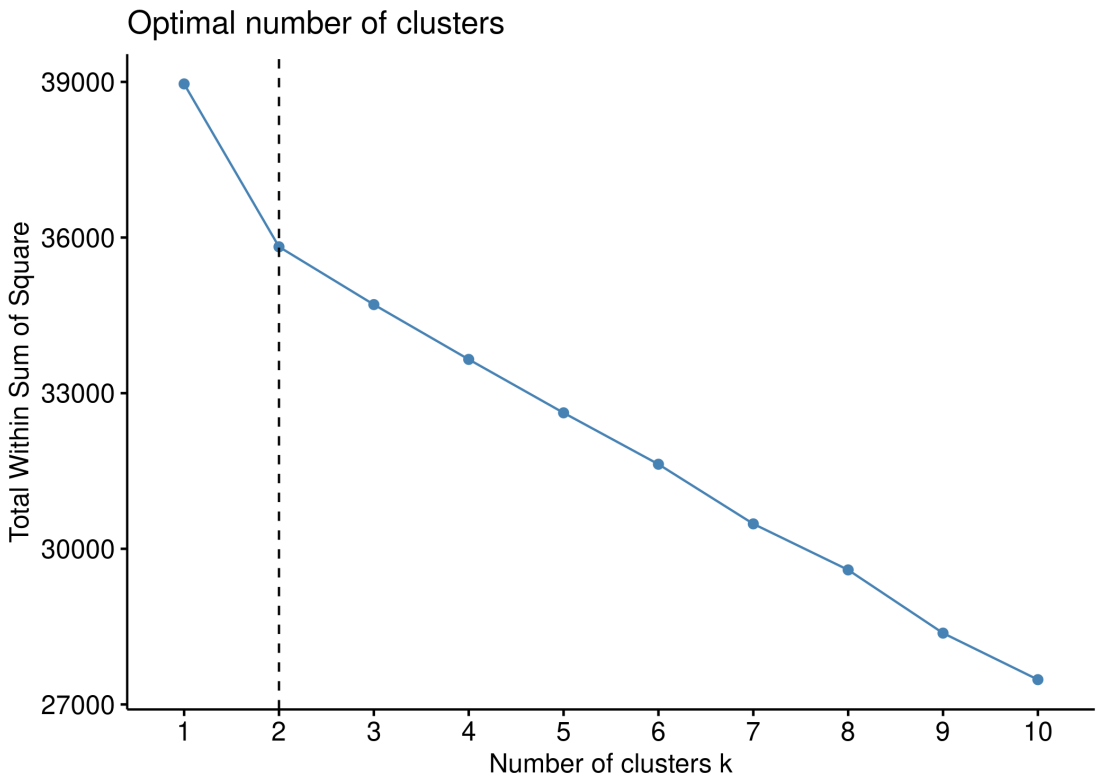
1.9 K-means

Since the data contains gene expression values for 1000 genes, it is important to preprocess the data before clustering. One common preprocessing step is to normalize the data so that each gene has a mean of 0 and a standard deviation of 1.

```
data_norm <- scale(genexp_df[,2:1000])
```

To perform K-Means Clustering, we need to first determine the optimal number of clusters. One way to do this is to use the elbow method. The method consists of plotting the explained variation as a function of the number of clusters and picking the elbow of the curve as the best number of clusters.

```
fviz_nbclust(data_norm, kmeans, method = "wss") +
  geom_vline(xintercept = 2, linetype = "dashed")
```



Based on the elbow plot, we can see that the optimal number of clusters is 2.

```
set.seed(123)
kmeans_res <- kmeans(data_norm, centers = 2)
```

We can visualize the clusters using a scatter plot. Since the data has 1000 dimensions, we need to reduce the dimensionality of the data to 2 dimensions using principal component analysis (PCA) or t-SNE. Here, we will use PCA.

```
pca <- prcomp(data_norm)
data_pca <- data.frame(PC1=pca$x[,1], PC2=pca$x[,2], Cluster=kmeans_res$cluster)
```

```
ggplot(data_pca, aes(x=PC1, y=PC2, color=as.factor(Cluster))) +
  geom_point() +
  labs(title="K-Means Clustering Results (K=2)", x="PC1", y="PC2") +
  theme_bw()
```