VIETNAMESE SPRING SCHOOL

IN

STATISTICS AND MACHINE LEARNING

CLASSIFICATION AND CLUSTERING

EXERCISES - PRACTICALS

Agnès Lagnoux **lagnoux@univ-tlse2.fr**

1 Worksheet No. 1 - k-Nearest Neighbors Algorithm

Exercise 1 *The goal of this exercise is to*

— *apply the k-nearest neighbors on a hand-written example;*
— *use the Euclidean and Manhattan distances;*
— *understand the importance of choosing the parameter k.*

Consider the following data table :

Number	x	y	z	Color
1	3	7	5	black
2	4	6	2	black
3	3	7	8	white
4	0	1	2	black
5	1	0	7	white
6	5	4	4	white
7	9	1	2	black
8	5	3	3	black
9	1	1	4	white
10	2	3	7	white

It contains the data of 10 points in space colored according to a certain spatial logic. We consider an eleventh point A with coordinates $x = 4$, $y = 4$ and $z = 5$ whose color we seek to predict by applying the k-nearest neighbors algorithm.

1. How many descriptors will be used to perform the prediction? What are they?
2. How many different labels are there? What are they?
3. Recall the definition of the Euclidean distance $d_{\text{Euclidean}}(A,B)$ in $\mathbb{R}^3$ between two points $A = (x_A, y_A, z_A)$ and $B = (x_B, y_B, z_B)$.
4. The Euclidean distances between the point $A(4,4,5)$ and the points in the table have been recorded in the following table. Calculate the two missing Euclidean distances.

Point	1	2	3	4	5	6	7	8	9	10
Distance	3.16	3.60	4.36	???	5.39	1.41	???	2.45	4.36	3

5. We apply the k-nearest neighbors algorithm to predict the color at point A.
   — If $k = 1$, what prediction do we get?
   — If $k = 3$, what prediction do we get?
   — If $k = 5$, what prediction do we get?
6. Why don't we take an even value for k?
7. Repeat questions 3. and 4. with the Manhattan distance :

$$d_{\text{Manhattan}}(A,B) = |x_A - x_B| + |y_A - y_B| + |z_A - z_B|.$$

Practical 1 - k-nearest-neighbors

Agnès LAGNOUX & Nicolas SAVY

Exercise 2 : ... on simulated data

The objective of this exercise is to get to grips with the R **caret** package. The aim is to illustrate, on an example of simulated data, the different issues associated with k-nn, namely:

- the issue of the number of classes
- the search for the optimal number of classes
- the issue of data pre-processing
- the issue of prediction games test data / validation / prediction

The data set is deliberately very simple. It is subdivided into three files available on IRIS in the section associated with this course. The files ***synth_train.txt*** for training the model, the file ***synth_valid.txt*** for validation and the file ***synth_test.txt*** for testing. These files contain 100 observations of the variables

- x_1 quantitative
- x_2 quantitative
- y qualitative representing the class of the observation.

  Note: These files are of the same size so that the accuracy is not affected by the sample size.

Importing the data

Load the training dataset ***synth_train.txt*** into R

- either with the command **read.table**
- or with the Rstudio import tool (Tools > Import Dataset).

Display the first data of the training sample.

```
train <- read.table(file="synth_train.txt", header=TRUE)
dim(train)
```

```
## [1] 100   3
```

```
head(train)
```

```
##   y          x1        x2
## 1 2 -0.72221141 2.0044709
## 2 2 -0.92467912 0.4836693
## 3 2 -0.76602281 0.7943289
## 4 2 -0.07328948 0.9699291
## 5 1 -1.39291198 0.9996971
## 6 2 -0.20223339 1.3503319
```

Transformation of Y into a two-modality **factor** type variable.

```
train$y = as.factor(train$y)
```

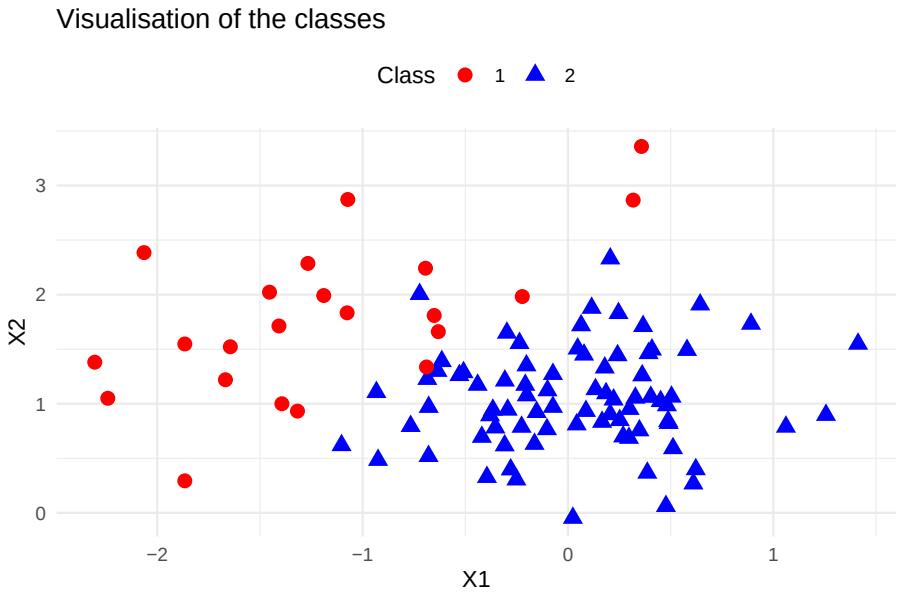
Load common packages for k-nn manipulation.

```
library(class)
library(caret)
library(FNN)
library(ggplot2)
```

Graphical representation of the data

Represent graphically data using the **ggplot2** package.

```
Xtrain <- train[,-1]          # explanatory variables
Ytrain <- train$y             # class variable

# Combine training data into a data frame
train_data <- data.frame(X1 = Xtrain[,1], X2 = Xtrain[,2], Classe = factor(Ytrain))

# Plot using ggplot2
ggplot(train_data, aes(x = X1, y = X2, shape = Classe, color = Classe)) +
  geom_point(size = 3) +
  # Choice of colors
  scale_color_manual(values = c("red", "blue"), name = "Class") +
  # Choice of shapes (triangles and rounds)
  scale_shape_manual(values = c(16, 17), name = "Class") +
  labs(title = "Visualisation of the classes", x = "X1", y = "X2") +
  theme_minimal() +
  theme(legend.position = "top")
```



Classification by k-nn with 15 neighbors

Use of the package caret for the training of a k-nn with 15 neighbors

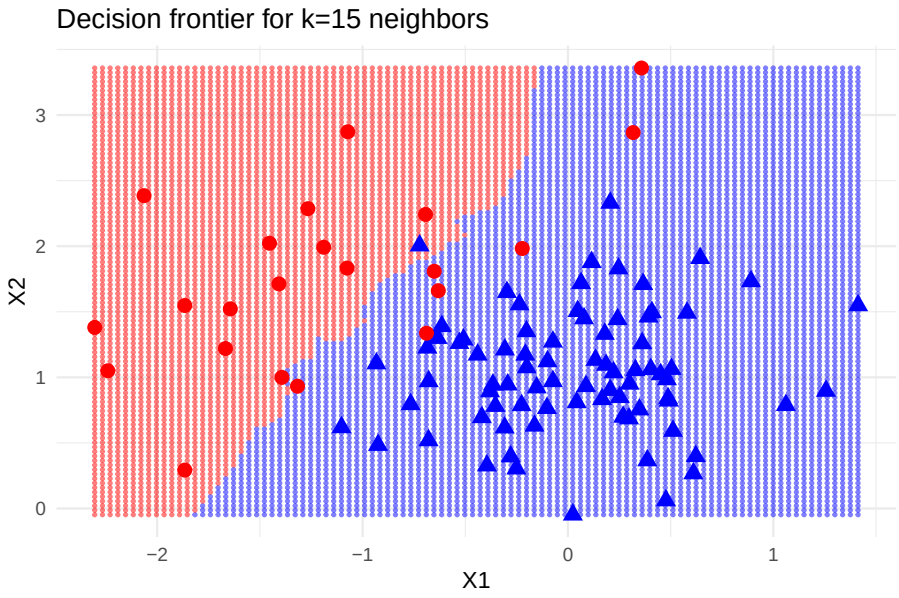
```
model_15 <- train(
  y ~ .,                                  # Formula : target variable and predictors
```

```
  data = train,                                    # Data
  method = "knn",                                  # k-nn algorithm
  tuneGrid = data.frame(k = 15)                    # Number of neighbors
)
```

Graphical representation of the decision frontier for $k = 15$ neighbors

Represent graphically the decision boundary for $k = 15$ neighbors: start by constructing a grid of points, predicting these points, then adding these points to the graph by coloring them according to their prediction.

```
a <- seq(from=min(train$x1), to=max(train$x1), length.out=100)
b <- seq(from=min(train$x2), to=max(train$x2), length.out=100)
grille <- NULL
for (i in a){
  grille <- data.frame(rbind(grille, cbind(i,b)))
}
names(grille) = c("x1","x2")

# Construction of a dataframe with the training data
train_data <- data.frame(Xtrain, Ytrain = factor(Ytrain))
pred_grille <- predict(model_15, grille, type = "raw")
# Construction of a  dataframe with the grid and the predictions
grid_data <- data.frame(grille, pred_grille = factor(pred_grille))

ggplot() +
  # Construction of the decision frintier
  geom_point(data = grid_data,
             aes(x = grille[,1], y = grille[,2], color = pred_grille),
             alpha = 0.5, size = 0.5) +
  # Positioning of the training points
  geom_point(data = train_data,
             aes(x = Xtrain[,1], y = Xtrain[,2], shape = Ytrain, color = Ytrain),
             size = 3) +
  # Adding of the legend and the titles
  scale_color_manual(values = c("red", "blue"), name = "Class") +
  scale_shape_manual(values = c(16, 17), name = "Class") +
  labs(title = "Decision frontier for k=15 neighbors", x = "X1", y = "X2") +
  theme_minimal() +
  theme(legend.position = "topright")
```



Evaluation of the performances of the classification and role of the validation dataset

The performance of a classification is measured in terms of predictive performance, i.e. the ability of the algorithm to find classes on a labeled set.

Predict the data of the training set with the model_15 classifier and compare with the real classes. Calculate the predictive performance indicators of this classifier.

```
predictions = predict(model_15, Xtrain ,type = "raw")
CM_15 = confusionMatrix(predictions, train$y)
CM_15
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  1  2
##          1 15  1
##          2  7 77
##
##                Accuracy : 0.92
##                  95% CI : (0.8484, 0.9648)
##     No Information Rate : 0.78
##     P-Value [Acc > NIR] : 0.0001699
##
##                   Kappa : 0.7416
##
##  Mcnemar's Test P-Value : 0.0770999
##
##             Sensitivity : 0.6818
##             Specificity : 0.9872
##          Pos Pred Value : 0.9375
##          Neg Pred Value : 0.9167
##              Prevalence : 0.2200
```

```
##          Detection Rate : 0.1500
##    Detection Prevalence : 0.1600
##       Balanced Accuracy : 0.8345
##
##        'Positive' Class : 1
##
```

Performance measurement should be done on a different dataset than the training dataset. Load the validation datasets ***synth_valid.txt*** into R. Display the first validation data.

```
valid <- read.table(file="synth_valid.txt", header=TRUE)
valid$y = as.factor(valid$y)
head(valid)
```

```
##   y          x1        x2
## 1 2  0.54837733 1.2213453
## 2 2 -0.51618236 1.5623959
## 3 2 -0.92877833 0.9210722
## 4 2  0.07000405 0.6197675
## 5 2  0.26702843 1.1094406
## 6 2 -0.57664073 1.0257432
```

Repeat the performance indicator calculations on the validation data and compare.

```
predictions = predict(model_15, valid ,type = "raw")
CM_15 = confusionMatrix(predictions, valid$y)
CM_15
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  1  2
##          1 27  0
##          2 10 63
##
##                Accuracy : 0.9
##                  95% CI : (0.8238, 0.951)
##     No Information Rate : 0.63
##     P-Value [Acc > NIR] : 8.883e-10
##
##                   Kappa : 0.7728
##
##  Mcnemar's Test P-Value : 0.004427
##
##             Sensitivity : 0.7297
##             Specificity : 1.0000
##          Pos Pred Value : 1.0000
##          Neg Pred Value : 0.8630
##              Prevalence : 0.3700
##          Detection Rate : 0.2700
##    Detection Prevalence : 0.2700
##       Balanced Accuracy : 0.8649
##
##        'Positive' Class : 1
##
```

Predicting the class of unannotated points

Consider the points with coordinates (0,1) and (-0.75,1.5). We can predict the class with a k-nn with $k = 15$ neighbors.

```
Xnew <- as.data.frame(matrix(c(0,1,-0.75,1.5), nrow=2, byrow=TRUE))
names(Xnew) = c("x1","x2")
# It is important that the variables have the same name as the training variables.
```

The predictions are therefore:

```
# Prediction results
predict(model_15, Xnew ,type = "raw")
```

```
## [1] 2 2
## Levels: 1 2
```

To access the k-nn indices of a given point, you need to use the FNN package:

```
KNN = get.knnx(data = Xtrain, query = Xnew, k = 15)
KNN
```

```
## $nn.index
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13] [,14]
## [1,]    4   11   34    9   73   40   29   50   94    13    63    60    80    54
## [2,]   99   31   15   19   52   25   97   22   57    44    70    39     1    30
##      [,15]
## [1,]    86
## [2,]    76
##
## $nn.dist
##            [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
## [1,] 0.07921871 0.1106247 0.1572590 0.1710396 0.1883963 0.1956980 0.2081735
## [2,] 0.17438636 0.1750355 0.1997047 0.2310871 0.2819727 0.3215701 0.3243251
##           [,8]      [,9]     [,10]     [,11]     [,12]     [,13]     [,14]
## [1,] 0.2142254 0.2252575 0.2274743 0.2357951 0.2560375 0.2696507 0.2810465
## [2,] 0.3256700 0.4347641 0.4514010 0.4656087 0.4767753 0.5052357 0.5169332
##          [,15]
## [1,] 0.2940067
## [2,] 0.5288516
```

```
Ytrain[KNN$nn.index[1,]]
```

```
##  [1] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
## Levels: 1 2
```

```
Ytrain[KNN$nn.index[2,]]
```

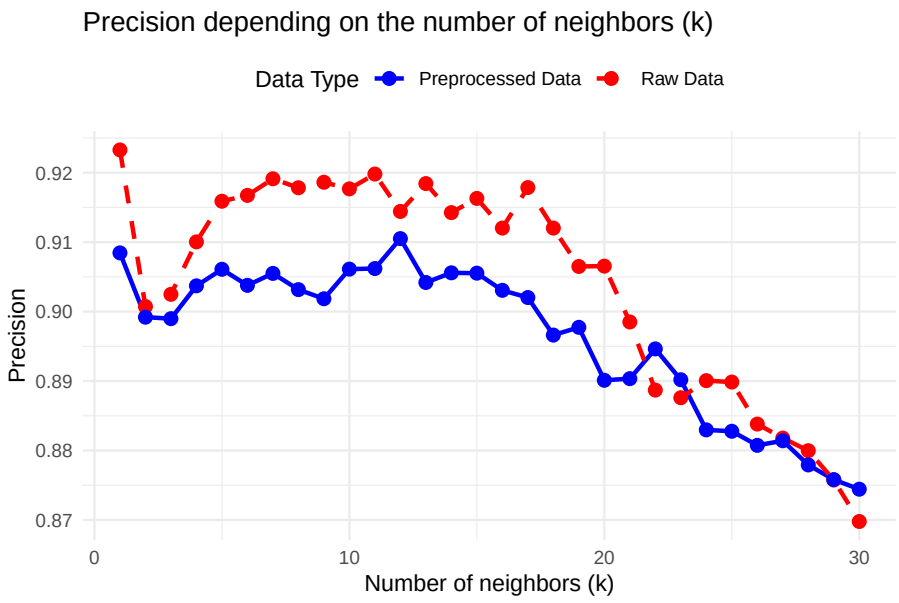
```
##  [1] 1 2 1 2 2 2 1 2 2 2 1 2 2 2 2
## Levels: 1 2
```

Role of preprocessing

We said in class that not preprocessing data can also lead to an overestimation of performance measures in k-nns. Let's illustrate this phenomenon on these data.

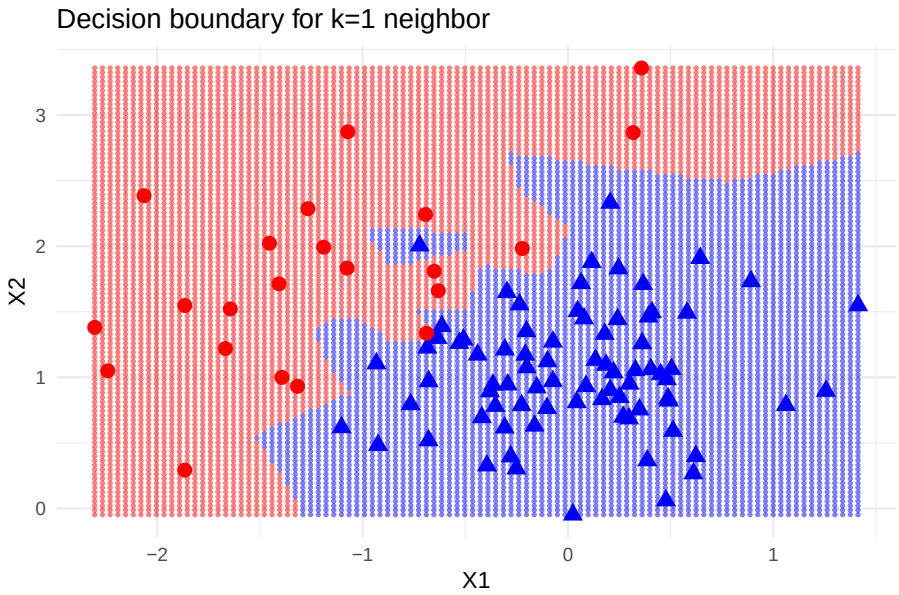
```
set.seed(123)                                # For reproducibility
grid <- expand.grid(k = seq(1, 30, by = 1)) # Grid of values of k

# Training the k-nn model without pre-processing
```

```r
knn_fit_raw <- train(
y ~ ., # Formula: target variable and predictors
data = train, # Dataset
method = "knn", # k-nn algorithm
tuneGrid = grid) # Grid of values for k

# Training the k-nn model with pre-processing
knn_fit_PP <- train(
y ~ .,              # Formula: target variable and predictors
data = train,       # Dataset
method = "knn",     # k-nn algorithm
preProcess = c("center", "scale"),
tuneGrid = grid)    # Grid of values for k

# Extract the performances of both models
k_values_raw <- knn_fit_raw$results$k
accuracy_raw <- knn_fit_raw$results$Accuracy
k_values_PP <- knn_fit_PP$results$k
accuracy_PP <- knn_fit_PP$results$Accuracy

# Construction of a dataframe collecting the data
data_raw <- data.frame(k_values = k_values_raw,
 accuracy = accuracy_raw,
 Data = "Raw Data")
data_pp <- data.frame(k_values = k_values_PP,
 accuracy = accuracy_PP,
 Data = "Preprocessed Data")
plot_data <- rbind(data_raw, data_pp)

# Plot both curves on the same graph
ggplot(plot_data, aes(x = k_values,
 y = accuracy,
 color = Data,
 linetype = Data,
 shape = Data)) +
 geom_line(size = 1) +
 geom_point(size = 3) +
 labs(
 title = "Precision depending on the number of neighbors (k)",
 x = "Number of neighbors (k)",
 y = "Precision",
 color = "Data Type",
 linetype = "Data Type",
 shape = "Data Type"
 ) +
 scale_color_manual(values = c("blue", "red")) +
 scale_linetype_manual(values = c(1, 2)) +
 scale_shape_manual(values = c(16, 16)) +
 theme_minimal() +
 theme(legend.position = "top")
```



Classification by k-nn with 1 neighbor

```
model_1 <- train(
y ~ .,                          # Formula: target variable and predictors
data = train,                   # Dataset
method = "knn",                 # k-nn algorithm
tuneGrid = data.frame(k = 1) # Number of neighbors
)

# Plot of the decision boundary for k = 1 neighbor
train_data <- data.frame(Xtrain, Ytrain = factor(Ytrain))
pred_grille <- predict(model_1, grille, type = "raw")
grid_data <- data.frame(grille, pred_grille = factor(pred_grille))

ggplot() +
geom_point(data = grid_data,
aes(x = grille[,1], y = grille[,2], color = pred_grille),
alpha = 0.5, size = 0.5) +
geom_point(data = train_data,
aes(x = Xtrain[,1], y = Xtrain[,2],
shape = Ytrain, color = Ytrain), size = 3) +
scale_color_manual(values = c("red", "blue"), name = "Class") +
scale_shape_manual(values = c(16, 17), name = "Class") +
labs(title = "Decision boundary for k=1 neighbor", x = "X1", y = "X2") +
theme_minimal() +
theme(legend.position = "topright")
```




```r
# Performance evaluation on training data
predictions = predict(model_1, Xtrain ,type = "raw")
CM_1 = confusionMatrix(predictions, train$y)
CM_1
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  1  2
##          1 22  0
##          2  0 78
##
##                Accuracy : 1
##                  95% CI : (0.9638, 1)
##     No Information Rate : 0.78
##     P-Value [Acc > NIR] : 1.62e-11
##
##                   Kappa : 1
##
##  Mcnemar's Test P-Value : NA
##
##             Sensitivity : 1.00
##             Specificity : 1.00
##          Pos Pred Value : 1.00
##          Neg Pred Value : 1.00
##              Prevalence : 0.22
##          Detection Rate : 0.22
##    Detection Prevalence : 0.22
##       Balanced Accuracy : 1.00
##
##        'Positive' Class : 1
##
```

```r
# Performance evaluation on validation data
predictions = predict(model_1, valid ,type = "raw")
CM_1_valid = confusionMatrix(predictions, valid$y)
CM_1_valid
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  1  2
##          1 30  2
##          2  7 61
##
##                Accuracy : 0.91
##                  95% CI : (0.836, 0.958)
##     No Information Rate : 0.63
##     P-Value [Acc > NIR] : 1.623e-10
##
##                   Kappa : 0.8014
##
##  Mcnemar's Test P-Value : 0.1824
##
##             Sensitivity : 0.8108
##             Specificity : 0.9683
##          Pos Pred Value : 0.9375
##          Neg Pred Value : 0.8971
##              Prevalence : 0.3700
##          Detection Rate : 0.3000
##    Detection Prevalence : 0.3200
##       Balanced Accuracy : 0.8895
##
##        'Positive' Class : 1
##
```

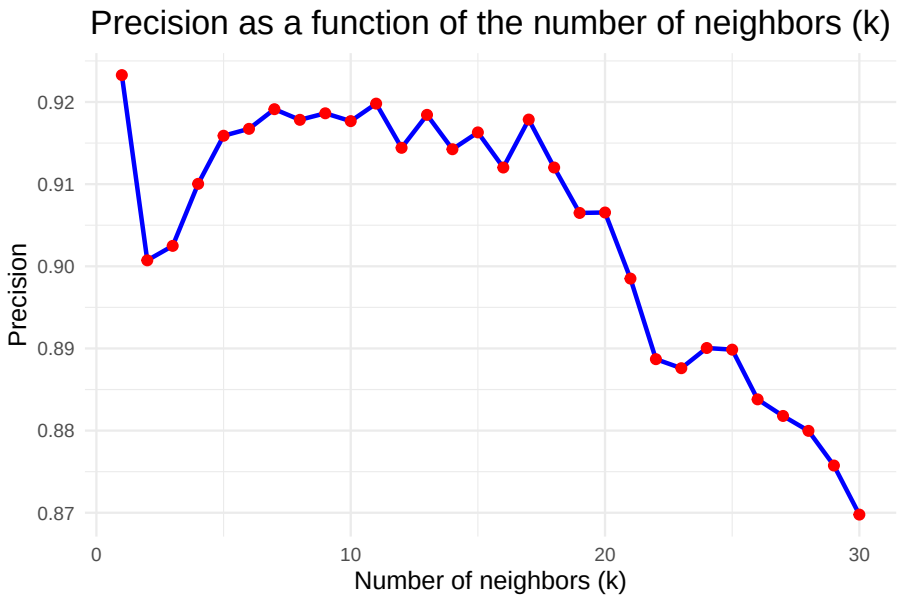
Finding the optimal number of classes by cross-validation

Measuring the impact of the number of neighbors on predictive performance

Graphical representation of the empirical error as a function of the Number of neighbors for k from 1 to 30.

```r
set.seed(123) # For reproducibility
grid <- expand.grid(k = seq(1, 30, by = 1)) # Grid of values of k

# Training the k-nn model
knn_fit <- train(
y ~ .,                  # Formula: target variable and predictors
data = train,           # Data set
method = "knn",         # k-NN algorithm
tuneGrid = grid)        # Grid of values for k

# Extract the results of the k-nn model
results <- knn_fit$results

# Creating the graph with ggplot2
ggplot(results, aes(x = k, y = Accuracy)) +
```

```
geom_line(color = "blue", size = 1) +
geom_point(color = "red", size = 2) +
labs(
title = "Precision as a function of the number of neighbors (k)",
x = "Number of neighbors (k)",
y = "Precision"
) +
theme_minimal() +
theme(
plot.title = element_text(hjust = 0.5, size = 16),
axis.title = element_text(size = 12)
)
```



Using the triptych Learning / Validation / Test

To use the knn to make predictions, 3 data sets must be used:

- ***synth_train.txt*** for learning,
- ***synth_valid.txt*** for validation and choice of the number of classes by cross-validation,
- the training of the final model is done from (***synth_train.txt*** + ***synth_valid.txt***) with the number of classes identified in the previous step,
- ***synth_test.txt*** of the unannotated data that we want to predict.

Training models with the training set

```
set.seed(123)
grid <- expand.grid(k = seq(1, 30, by = 1)) # Grid of k values
knn_model_validation <- train(
y ~ ., data = train, method = "knn",
trControl = trainControl(method = "cv", number = 10) ,
tuneGrid = grid # Test k = 1, 2, ..., 30
)
```

Measuring model performance on validation data

```
# Results on the validation set
validationPredictions <- predict(knn_model_validation, newdata = valid)
confusionMatrix(validationPredictions, valid$y)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  1  2
##          1 28  1
##          2  9 62
##
##                Accuracy : 0.9
##                  95% CI : (0.8238, 0.951)
##     No Information Rate : 0.63
##     P-Value [Acc > NIR] : 8.883e-10
##
##                   Kappa : 0.7755
##
##  Mcnemar's Test P-Value : 0.02686
##
##             Sensitivity : 0.7568
##             Specificity : 0.9841
##          Pos Pred Value : 0.9655
##          Neg Pred Value : 0.8732
##              Prevalence : 0.3700
##          Detection Rate : 0.2800
##    Detection Prevalence : 0.2900
##       Balanced Accuracy : 0.8704
##
##        'Positive' Class : 1
##
```

Identification of the best k

```
best_k <- knn_model_validation$bestTune$k
cat("Best k found :", best_k, "\n")
```

```
## Best k found : 4
```

Training of the final model on the training data + the validation data

```
finalData <- rbind(train, valid)   # Union of the datasets
set.seed(123)
final_model <- train(
  y ~ ., data = finalData,
  method = "knn",
  tuneGrid = data.frame(k = best_k), # Use the best k
  trControl = trainControl(method = "none")
)
```

Measuring the performance of the best model on test data

```
test <- read.table(file="synth_test.txt", header=TRUE)
test$y = as.factor(test$y)
head(test)
```

```
##   y          x1        x2
## 1 2 -0.30748160 1.1420242
## 2 2  0.51837239 1.0574169
## 3 2 -0.05222164 0.7100728
## 4 2 -0.08914077 1.3942948
## 5 2 -0.37621346 0.9307517
## 6 2 -0.18555218 0.7054576
```

```
testPredictions <- predict(final_model, newdata = test)
confusionMatrix(testPredictions, test$y)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  1  2
##          1 24  1
##          2  1 74
##
##                Accuracy : 0.98
##                  95% CI : (0.9296, 0.9976)
##     No Information Rate : 0.75
##     P-Value [Acc > NIR] : 1.874e-10
##
##                   Kappa : 0.9467
##
##  Mcnemar's Test P-Value : 1
##
##             Sensitivity : 0.9600
##             Specificity : 0.9867
##          Pos Pred Value : 0.9600
##          Neg Pred Value : 0.9867
##              Prevalence : 0.2500
##          Detection Rate : 0.2400
##    Detection Prevalence : 0.2500
##       Balanced Accuracy : 0.9733
##
##        'Positive' Class : 1
##
```

Back to the prediction of our new entries Xnew

Calculate the predictions of the new entries with the final model retained.

```
final_model$bestTune$k
```

```
## [1] 4
```

```
XfinalData = finalData[,-1]
YfinalData = finalData$y
KNN = get.knnx(data = XfinalData, query = Xnew, k = 4)
```

```
KNN
```

```
## $nn.index
##      [,1] [,2] [,3] [,4]
## [1,]    4  140  133   11
## [2,]  175   99   31  199
##
## $nn.dist
##            [,1]       [,2]       [,3]      [,4]
## [1,] 0.07921871 0.08300985 0.09430681 0.1106247
## [2,] 0.14421032 0.17438636 0.17503554 0.1938767
```

```
YfinalData[KNN$nn.index[1,]]
```

```
## [1] 2 2 2 2
## Levels: 1 2
```

```
YfinalData[KNN$nn.index[2,]]
```

```
## [1] 1 1 2 1
## Levels: 1 2
```

```
# Raw results
predict(final_model, Xnew ,type = "raw")
```

```
## [1] 2 1
## Levels: 1 2
```

```
# Results in terms of probability
predict(final_model, Xnew ,type = "prob")
```

```
##      1    2
## 1 0.00 1.00
## 2 0.75 0.25
```

Exercise 3. Impact of sample size on accuracy
The goal of this exercise is to illustrate the impact of sample size on precision.

1. Load the dataset ***dataExo3.csv***. Be careful with the data types..
2. Divide the dataset into 80% training and 20% validation sets.
   To do this, we will use the function

   ```
   createDataPartition(data$Class, p = 0.8, list = FALSE)
   ```

3. Build a function that associates the accuracy with the sample size. For example, we will take $k = 5$ neighbors.
4. Calculate the precisions associated with the sample size for a sample size ranging from 50 to 800 in steps of 50.
5. Graph the results obtained.

Exercise 4. On real data... It's your turn!
The database ***BCWDataset.csv*** was built from the Breast Cancer Wisconsin database. This database collects characteristics of cell nuclei present in breast mass imaging. These characteristics are calculated from a digitized image of a fine needle aspiration (FNA) of a breast mass. The characteristics retained here for each cell nucleus are :

— **radius** : mean radius of the cell nucleus
— **texture** : variation of pixel intensity
— **perimeter** : average cell perimeter
— **aera** : average cell area
— **smoothness** : contour smoothness (variance of gradients)
— **compactness** : ratio "perimeter^2 / (surface - 1)"
— **concavity** : depth of contour concavities
— **concave_ point** : number of concave points on the contour
— **symmetry** : degree of cell symmetry
— **fractal_ dimension** : complexity of contours measured by the fractal dimension

All feature values are recoded with four significant digits.
The database is completed by the anatopathological diagnosis of the mass :

— **diagnosis** : the diagnosis (M = malignant, B = benign)

1. *Database assembly*
   Load the dataset ***BCWDataset.csv*** into R. Display the dimension of the training set. Display the first 6 records.
2. *Learning / Validation split*
   (a) Create a training dataset of size 300 (75% of the data) and a test dataset of size 100 (25% of the data).
   (b) Train the models on the training data and calculate the error rates on the test data for k varying from 1 to 50. Plot this test error rate as a function of k (check that the abscissa of the graph starts from 0).
   (c) *Impact of the split on the result*
   Start again with another random split training / validation and represent the curve of the evolution of the test error rate on the same graph as in the previous question.
3. *Choice of k by leave-one-out (LOO) cross validation*
   Now choose the number k of neighbors using leave-one-out (LOO) cross validation and the training data.
   Calculate the accuracy of the chosen algorithm on the validation data.

4. *Training the final model on the training and validation data*

   Build the final classifier from the "training-validation" set and calculate the error rate of the initial data.

5. *Making a diagnosis on new cases*

   The database ***BCWDiagno.csv*** contains the characteristics of 169 new patients.

   Use the knn algorithm chosen previously to predict the diagnosis associated with each of these cases. The result will be provided in terms of class and in terms of probabilities of belonging to the class.

2 Worksheet No. 2 - Linear and quadratic discriminant analysis

Practical 2 - Discriminant analysis

Agnès LAGNOUX & Nicolas SAVY

Exercise 1

The objective of this exercise is to

- discover linear and quadratic discriminant analysis
- program quadratic discriminant analysis
- discover functions **lda** and **qda** of the **caret** package.

In this exercise, we will work on the simulated data from the previous sheet contained in the ***synth_train.txt*** file.

Loading libraries and importing data

Loading common packages for manipulating LDA and QDA.

```
library(caret)
library(MASS)
library(ggplot2)
```

Load the training dataset ***synth_train.txt*** into R. Display the first data of the training sample.

```
train <- read.table(file="synth_train.txt", header=TRUE)
dim(train)
```

```
## [1] 100   3
```

```
head(train)
```

```
##   y          x1        x2
## 1 2 -0.72221141 2.0044709
## 2 2 -0.92467912 0.4836693
## 3 2 -0.76602281 0.7943289
## 4 2 -0.07328948 0.9699291
## 5 1 -1.39291198 0.9996971
## 6 2 -0.20223339 1.3503319
```

Transformation of Y into a two-modality **factor** type variable.

```
train$y
```

```
##   [1] 2 2 2 2 1 2 2 1 2 2 2 2 2 2 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 1 2 1 2 2
##  [38] 2 2 2 2 2 2 2 2 2 1 2 2 2 2 2 1 2 2 1 2 1 2 2 2 2 2 2 2 1 2 2 1 1 2 2 2 2
##  [75] 2 2 2 2 2 2 1 1 2 1 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1
```

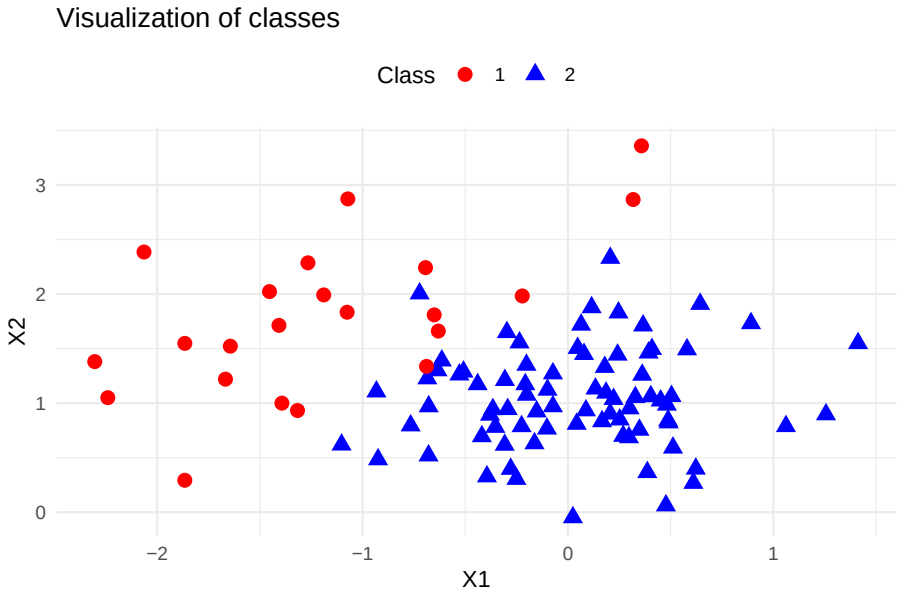
```
train$y = as.factor(train$y)
train$y
```

```
##   [1] 2 2 2 2 1 2 2 1 2 2 2 2 2 2 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 1 2 1 2 2
```

```
##  [38] 2 2 2 2 2 2 2 2 1 2 2 2 2 2 1 2 2 1 2 1 2 2 2 2 2 2 2 1 2 2 1 1 2 2 2 2
##  [75] 2 2 2 2 2 2 1 1 2 1 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1
## Levels: 1 2
```

Plot the data using the **ggplot2** package.

```
Xtrain <- train[,-1] # explanatory variables
Ytrain <- train$y # class variable

# Combine training data into a data frame
train_data <- data.frame(X1 = Xtrain[,1], X2 = Xtrain[,2], Class = factor(Ytrain))

# Plot using ggplot2
ggplot(train_data, aes(x = X1, y = X2, shape = Class, color = Class)) +
geom_point(size = 3) +
# Color selection
scale_color_manual(values = c("red", "blue"), name = "Class") +
# Shape selection (triangles and circles)
scale_shape_manual(values = c(16, 17), name = "Class") +
labs(title = "Visualization of classes", x = "X1", y = "X2") +
theme_minimal() +
theme(legend.position = "top")
```



Quadratic Discriminant Analysis by Hand

In **quadratic discriminant analysis**, we make the parametric Gaussian assumption that

$$X \mid Y = k \sim \mathcal{N}(\mu_k, \Sigma_k);$$

in other words

$$f(x \mid Y = k) = \frac{1}{(2\pi)^{p/2} |\Sigma_k|^{1/2}} \exp\left(-\frac{1}{2}(x - \mu_k)^T \Sigma_k^{-1} (x - \mu_k)\right)$$

The unknown parameters μ_k and Σ_k and the a priori probabilities

$$\pi_k = \mathbb{P}(Y = k)$$

for $k = 1.2$ are estimated by maximum likelihood by:

$$\widehat{\pi}_k = \frac{n_k}{n}, \qquad \widehat{\mu}_k = \frac{1}{n_k} \sum_{i;y_i=k} x_i, \qquad \widehat{\Sigma}_k = \frac{1}{n_k} \sum_{i;y_i=k} (x_i - \widehat{\mu}_k)^\top (x_i - \widehat{\mu}_k).$$

We will estimate these parameters on these parameters on the training data.

```
n <- nrow(Xtrain)
# class 1
ind1 <- which(Ytrain==1)
n1 <- length(ind1)
pi1 <- n1/n
mu1 <- colMeans(Xtrain[ind1,])
sigma1 <- var(Xtrain[ind1,])*(n1-1)/n1

# class 2
ind2 <- which(Ytrain==2)
n2 <- length(ind2)
pi2 <- n2/n
mu2 <- colMeans(Xtrain[ind2,])
sigma2 <- var(Xtrain[ind2,])*(n2-1)/n2
```

The Bayes decision rule (predicting the most probable class a posteriori) is then written:

$$g(x) = argmax_{k \in \{1,2\}} Q_k(x)$$

with

$$Q_k(x) = -\frac{1}{2} \log |\Sigma_k|^{-1} - \frac{1}{2}(x - \mu_k)^\top \Sigma_k^{-1}(x - \mu_k) + \log \pi_k.$$

Q_k is called **quadratic discriminant function**.

Calculate $Q_1(x)$ and $Q_2(x)$ and verify that $x = (-1, 1)$ is indeed assigned to class 2.

```
x = c(-1,1)
names(x) = c("x1","x2")
Q1 <- - 1/2*log(det(sigma1))-1/2*t(x-mu1) %*% solve(sigma1) %*% (x-mu1) +log(pi1)
Q2 <- - 1/2*log(det(sigma2)) - 1/2*t(x-mu2) %*% solve(sigma2) %*% (x-mu2) +log(pi2)
Q1
```

```
##           [,1]
## [1,] -1.940979
```

```
Q2
```

```
##            [,1]
## [1,] -0.9362387
```

We therefore have $Q_1(x) < Q_2(x)$ so we predict class 2 for x=(-1,1). This is consistent.

We also know that the posterior probabilities of the classes are calculated as follows:

$$\mathbb{P}(Y = k|X = x) = \frac{\exp(Q_k(x))}{\sum_{\ell=1}^{2} \exp(Q_\ell(x))}.$$

Estimate the posterior probabilities for $x = (-1, 1)$.

```
prob1 <- exp(Q1)/(exp(Q1)+exp(Q2))
prob1
```

```
##           [,1]
## [1,] 0.2680105
```

```
prob2 <- exp(Q2)/(exp(Q1)+exp(Q2))
prob2
```

```
##           [,1]
## [1,] 0.7319895
```

$\mathbb{P}(Y = 2/X = (-1, 1)) > \mathbb{P}(Y = 1/X = (-1, 1))$ so we predict class 2 for $x = (-1, 1)$. This is consistent and of course equivalent to the prediction with $Q_1(x)$ and $Q_2(x)$.

Quadratic discriminant analysis with caret

Now use the functions **qda** and **predict** to predict the class of the point $x = (-1, 1)$ and estimate their posterior probabilities. Check that you find the results obtained previously.

Training the model on the train data

```
qda_model <- train(y ~ .,
                   data = train,
                   method = "qda")
```

Prediction of the point $x = (-1, 1)$

```
x1 = -1
x2 = 1
x = c(x1,x2)
head(predict(qda_model, x, "raw"))
```

```
## [1] 2
## Levels: 1 2
```

```
head(predict(qda_model, x, "prob"))
```

```
##           1         2
## 1 0.2679847 0.7320153
```

Splitting the dataset into training and testing samples

```
train_index <- createDataPartition(train$y, p = 0.7, list = FALSE)
train_data <- train[train_index, ]
test_data <- train[-train_index, ]
```

Training the model on the training data

```
qda_model <- train(y ~ .,
                   data = train_data,
                   method = "qda")
```

Predictions on the test dataset

```
qda_pred <- predict(qda_model, test_data)
```

Model Evaluation - Performance Analysis on test dataset

```
qda_conf_mat <- confusionMatrix(qda_pred, test_data$y)
cat("\n### Resultts QDA ###\n")
```

```
##
## ### Resultts QDA ###
```

```
print(qda_conf_mat)
```

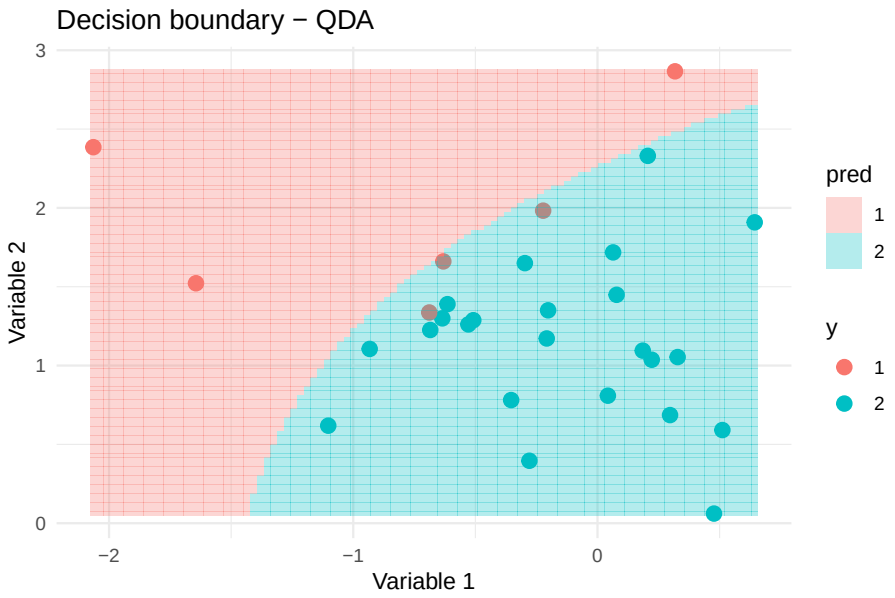
```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  1  2
##          1  3  0
##          2  3 23
##
##                Accuracy : 0.8966
##                  95% CI : (0.7265, 0.9781)
##     No Information Rate : 0.7931
##     P-Value [Acc > NIR] : 0.1217
##
##                   Kappa : 0.6133
##
##  Mcnemar's Test P-Value : 0.2482
##
##             Sensitivity : 0.5000
##             Specificity : 1.0000
##          Pos Pred Value : 1.0000
##          Neg Pred Value : 0.8846
##              Prevalence : 0.2069
##          Detection Rate : 0.1034
##    Detection Prevalence : 0.1034
##       Balanced Accuracy : 0.7500
##
##        'Positive' Class : 1
##
```

Visualizing the decision boundaries

The ***qda*** method constructs a quadratic decision boundary that can be represented graphically. Represent the decision boundary of the ***qda*** method.

```
grid <- expand.grid(x1 = seq(min(test_data$x1), max(test_data$x1), length.out = 100),
                    x2 = seq(min(test_data$x2), max(test_data$x2), length.out = 100))

# Decision boundary for QDA
grid$pred <- predict(qda_model, newdata = grid)
ggplot() +
    geom_point(data = test_data, aes(x = x1, y = x2, color = y), size = 3)   +
    geom_tile(data = grid, aes(x = x1, y = x2, fill = pred), alpha = 0.3) +
```

```
    labs(title = "Decision boundary - QDA", x = "Variable 1", y = "Variable 2") +
    theme_minimal()
```



Linear Discriminant Analysis by Hand

In **linear discriminant analysis**, the covariance matrices are assumed to be equal. The covariance matrix estimator

$$\Sigma = \Sigma_1 = \Sigma_2$$

is

$$\widehat{\Sigma} = \frac{1}{n}\sum_{k=1}^{K} n_k \widehat{\Sigma}_k \quad \text{avec} \quad \widehat{\Sigma}_k = \frac{1}{n_k}\sum_{i;y_i=k} (x_i - \widehat{\mu}_k)^\top (x_i - \widehat{\mu}_k).$$

Estimate this matrix on the training dataset.

```
n <- nrow(Xtrain)
# class 1
ind1 <- which(Ytrain==1)
n1 <- length(ind1)
pi1 <- n1/n
mu1 <- colMeans(Xtrain[ind1,])

# class 2
ind2 <- which(Ytrain==2)
n2 <- length(ind2)
pi2 <- n2/n
mu2 <- colMeans(Xtrain[ind2,])

# Sigma common
Sigma <- ((n1 - 1) * cov(Xtrain[ind1,]) + (n2 - 1) * cov(Xtrain[ind2,])) / (n1 + n2 - 2)
```

The Bayes decision rule (predicting the most probable class a posteriori) is then written:

$$g(x) = argmax_{k\in\{1,\cdots,K\}} L_k(x).$$

with

$$L_k(x) = x^\top \Sigma^{-1}\mu_k - \frac{1}{2}\mu_k^\top \Sigma^{-1}\mu_k + \log \pi_k.$$

where L_k is then called **linear discriminant function**.

We now want to predict the class of the new observation $x = (-1, 1)$ with the linear discriminant analysis method. Calculate $L_1(x)$ and $L_2(x)$ and verify that x is indeed assigned to class 2.

```
x = c(-1,1)
names(x) = c("x1","x2")
L1 <- t(x) %*% solve(Sigma) %*% mu1 - 0.5 * t(mu1) %*% solve(Sigma) %*% mu1 + log(pi1)
L2 <- t(x) %*% solve(Sigma) %*% mu2 - 0.5 * t(mu2) %*% solve(Sigma) %*% mu2 + log(pi2)
L1
```

```
##           [,1]
## [1,] 1.871951
```

```
L2
```

```
##           [,1]
## [1,] 2.755925
```

We also know that the posterior probabilities of the classes are calculated as follows:

$$\mathbb{P}(Y = k|X = x) = \frac{\exp(L_k(x))}{\sum_{\ell=1}^{K} \exp(L_\ell(x))}.$$

Estimate the a posteriori probabilities for $x = (-1, 1)$.

```
prob1 <- exp(L1)/(exp(L1)+exp(L2))
prob1
```

```
##           [,1]
## [1,] 0.2923549
```

```
prob2 <- exp(L2)/(exp(L1)+exp(L2))
prob2
```

```
##           [,1]
## [1,] 0.7076451
```

Linear discriminant analysis with caret

Now use the functions **lda** and **predict** to predict the class of the point $x = (-1, 1)$ and estimate their posterior probabilities. Check that you find the results of the previous questions.

Training the model on the train data

```
lda_model <- train(y ~ .,
                   data = train,
                   method = "lda")
```

Prediction of the point $x = (-1, 1)$

```
x1 = -1
x2 = 1
x = c(x1,x2)
head(predict(lda_model, x, "raw"))
```

```
## [1] 2
## Levels: 1 2
```

```
head(predict(lda_model, x, "prob"))
```

```
##           1         2
## 1 0.2923549 0.7076451
```

Training the model on the training data

```
lda_model <- train(y ~ .,
                   data = train_data,
                   method = "lda")
```

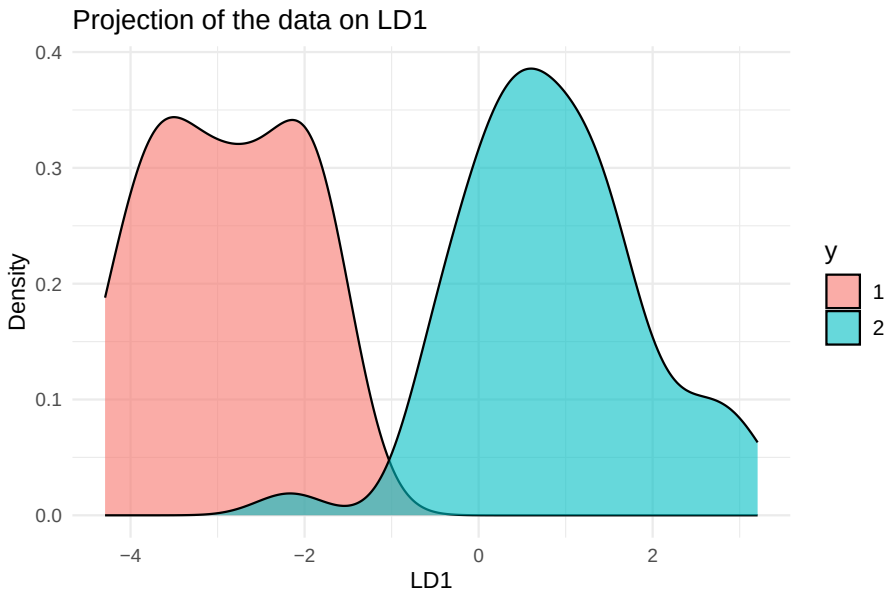
Predictions on the test dataset

```
predict(lda_model, test_data, type = "prob")
```

```
##                1            2
## 6   1.457692e-02 9.854231e-01
## 15  6.291831e-01 3.708169e-01
## 19  1.996887e-01 8.003113e-01
## 22  8.643522e-02 9.135648e-01
## 24  3.365524e-03 9.966345e-01
## 25  8.634577e-02 9.136542e-01
## 29  2.329128e-04 9.997671e-01
## 31  2.585111e-01 7.414889e-01
## 33  9.991158e-01 8.841870e-04
## 36  1.158025e-05 9.999884e-01
## 39  1.278110e-01 8.721890e-01
## 40  1.388923e-04 9.998611e-01
## 43  1.496766e-06 9.999985e-01
## 49  1.568038e-02 9.843196e-01
## 52  1.928726e-01 8.071274e-01
## 55  1.499655e-04 9.998500e-01
## 57  4.246654e-01 5.753346e-01
## 58  3.400266e-01 6.599734e-01
## 62  1.103752e-07 9.999999e-01
## 67  2.031225e-03 9.979688e-01
## 68  1.350573e-01 8.649427e-01
## 75  6.726134e-05 9.999327e-01
## 80  5.851030e-03 9.941490e-01
## 83  7.013778e-04 9.992986e-01
## 92  1.529838e-01 8.470162e-01
## 94  1.311655e-04 9.998688e-01
## 95  5.579761e-01 4.420239e-01
## 99  3.090080e-01 6.909920e-01
## 100 9.999996e-01 4.179813e-07
```

```
lda_pred <- predict(lda_model, test_data, type = "raw")
```

```
# Build the LDA model with caret
# lda_model <- train(Species ~ ., data = train_data, method = "lda")

# Project the training data onto the LDA axes
lda_final <- lda_model$finalModel
lda_projection <- predict(lda_final, train_data[,-1]) # Projections

# Organize projected data for visualization
lda_data <- data.frame(lda_projection$x) # Scores LDA1, LDA2
lda_data$y <- train_data$y  # Add the labels of the classes

ggplot(lda_data, aes(x = LD1, fill = y)) +
  geom_density(alpha = 0.6) +
  labs(title = "Projection of the data on LD1",
       x = "LD1", y = "Density") +
  theme_minimal() +
  theme(legend.title = element_text(size = 12),
        legend.text = element_text(size = 10))
```



Model evaluation and performance analysis on test data

```
lda_conf_mat <- confusionMatrix(lda_pred, test_data$y)
cat("\n### Results LDA ###\n")
```

```
##
## ### Results LDA ###
```

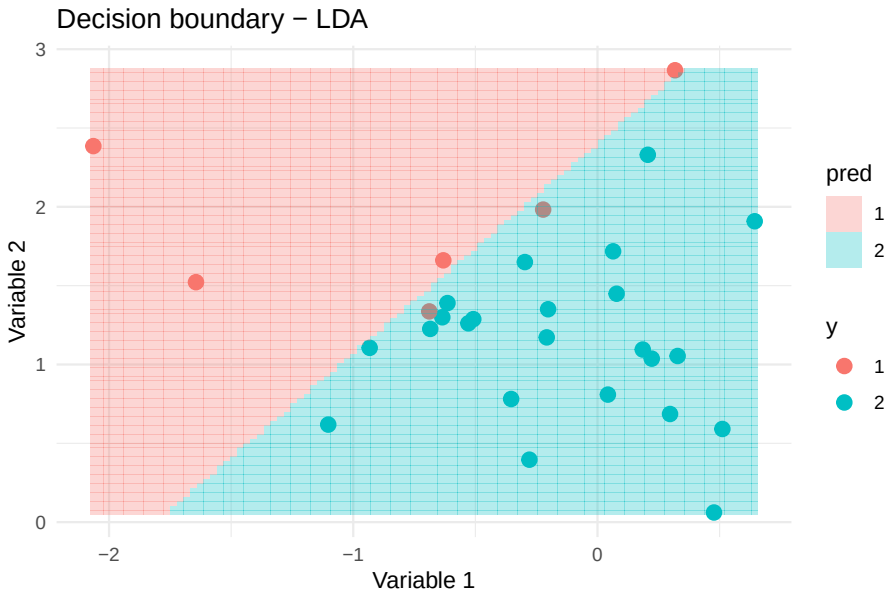
```
print(lda_conf_mat)
```

```
## Confusion Matrix and Statistics
##
##           Reference
```

```
## Prediction  1  2
##          1  4  0
##          2  2 23
##
##                Accuracy : 0.931
##                  95% CI : (0.7723, 0.9915)
##     No Information Rate : 0.7931
##     P-Value [Acc > NIR] : 0.04357
##
##                   Kappa : 0.7603
##
##  Mcnemar's Test P-Value : 0.47950
##
##             Sensitivity : 0.6667
##             Specificity : 1.0000
##          Pos Pred Value : 1.0000
##          Neg Pred Value : 0.9200
##              Prevalence : 0.2069
##          Detection Rate : 0.1379
##    Detection Prevalence : 0.1379
##       Balanced Accuracy : 0.8333
##
##        'Positive' Class : 1
##
```

Visualization of the decision boundary

```r
# Decision boundary for LDA
grid <- expand.grid(x1 = seq(min(test_data$x1), max(test_data$x1), length.out = 100),
                    x2 = seq(min(test_data$x2), max(test_data$x2), length.out = 100))
grid$pred <- predict(lda_model, newdata = grid)
ggplot() +
    geom_point(data = test_data, aes(x = x1, y = x2, color = y), size = 3)   +
    geom_tile(data = grid, aes(x = x1, y = x2, fill = pred), alpha = 0.3) +
    labs(title = "Decision boundary - LDA", x = "Variable 1", y = "Variable 2") +
    theme_minimal()
```


Decision boundary − LDA
Variable 2
Variable 1
3
2
1
0
−2
−1
0
pred
1
2
y
1
2

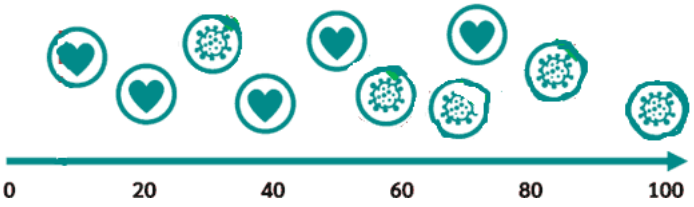
Exercise 2 *The objective of this exercise is to build the ROC curve on an example and calculate the AUC.*

We want to determine, based on screening, whether a person has cancer or not.

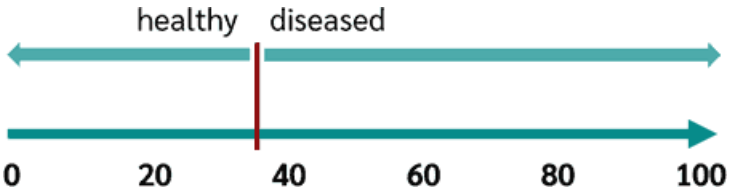
This classification is done using a certain blood value, where high values??indicate cancer. The question now is what value do we choose as the classification threshold.

☞ From what value can we predict a disease?

To do this, we obtain data from 10 people regarding the level of the blood value and the presence or absence of a disease [1].



1. For example, let's set the classification threshold to 45. Determine in this case the true positive rate (TPR) and the false positive rate (FPR).



We can now calculate, for each threshold, the true positive rate and the false positive rate. These two values??are then plotted on the ROC curve.
The true positive rate is plotted on the y-axis (ordinates) and the false positive rate on the x-axis (abscissas).

☞ Now let's plot the full ROC curve for our example!

2. Consider a value close to 0. Determine the true positive rate and the false positive rate. Plot this first point on a graph.
3. Let's consider a threshold of about 18 now. Determine the true positive rate and the false positive rate. Plot this second point on the graph.
4. Consider a threshold of about 28. Determine the true positive rate and the false positive rate. Plot this third point on the graph.
5. Consider a threshold of about 36. Determine the true positive rate and the false positive rate. Plot this fourth point on the graph.
6. Finish the ROC curve.

1. Source : https ://datatab.fr/tutorial/roc-curve.

The ROC curve allows us to compare different classification methods. The higher the curve, the better the classification model. Therefore, the larger the area under the curve (AUC), the better the classifier.
The value of the area under the curve varies between 0 and 1. The higher the value, the better the classifier.

7. Calculate the AUC.

Exercise 3

1. Load the data from Exercise 4 of Worksheet No. 1.
2. Create a training dataset of size 300 (75% of the data) and a test dataset of size 100 (25% of the data).
3. Resume the processing of this learning problem by k-nn.
   We will take care to collect the predictive performances of the selected classifier as well as the predicted values??in dataframes in order to compare the methods.
4. We now want to evaluate the ability of a score to discriminate sick individuals from healthy individuals. To do this, we will build the ROC curve associated with the score obtained with **knn** and evaluate the area under this curve with the **ROCR** package.

   (a) Install the **ROCR** package.

   (b) Calculate the probabilities with the command :

   ```
   prob <- predict(model_knn, BCWDataset ,type = "prob")
   head(prob)
   score_knn <- prob[,2]
   head(score_knn)
   ```

   As a reminder, if we set a threshold $s \in [0,1]$, we obtain a vector of predictions by assigning the observations that have a score greater than or equal to the threshold s to the positive class (here M=malignant) and the others to the negative class (here B=benign). Until now, we have always used a threshold (cutoff) s of 0.5 (by default) to obtain the predictions. The ROC curve provides more information than a single prediction vector obtained with $s = 0.5$. Indeed, it is built from a threshold grid (grid of values??in [0, 1]). For each threshold s in this grid, the false positive rate

   $$\text{FPR} = \frac{\text{false positives}}{\text{false positives + true negatives}} = \frac{FP}{FP+TN}.$$

   and the true positive rate

   $$\text{TPR} = \frac{\text{true positives}}{\text{true positives + false negatives}} = \frac{TP}{TP+FN}.$$

   of the prediction obtained with this threshold give the abscissa and ordinate of a point of the ROC curve. There will be as many points as thresholds in the grid.
   The prediction function of the ROCR package returns many results based on a grid of thresholds.

   ```
   pred <- prediction(score_knn, BCWDataset$diagnosis, label.ordering=c("B","M"))
   # label.ordering: indicates the label of the negative then positive class
   ```

   pred is an object of class S4. The @ symbol is therefore used to retrieve the results. We can thus retrieve the grid of threshold values??(cutoffs).

```
pred@cutoffs[[1]]
```

These thresholds provide contingency tables obtained by crossing the true classes and the predicted classes. With the code below, we retrieve the number of true positives (TP) and the number of false positives (FP) associated with each threshold.

```
pred@fp[[1]]
pred@tp[[1]]
```

With threshold 0, the 400 individuals are predicted to be malignant (class M). However, there are 227 benign individuals and 173 malignant ones.

```
table(BCWDataset$diagnosis)
```

Threshold 0 therefore gives 227 false positives and 173 true positives. This corresponds to a true positive rate of 1 (TVP) and a false positive rate (FPR) of 1. This will therefore be the point (1, 1) (top right) of the ROC curve for this score.

The threshold 0.11 gives 110 false positives and 173 true positives. In other words, 173 (out of 173) malignant individuals are predicted to be malignant and 110 (out of 227) benign individuals are predicted to be malignant. The threshold 0.11 therefore corresponds to a true positive rate (TPR) of 173/173=1 and a false positive rate (FPR) of 110/227=0.48. This threshold will therefore correspond to the point (0.48, 1) on the ROC curve.

We understand that when the threshold increases, the points on the ROC curve shift to the left. Let's now plot this curve using the performance function to obtain the TFPs and TVPs that will form the abscissas and ordinates of this curve.

```
perf <- performance(pred, "tpr", "fpr")
perf@x.values[[1]] # fpr on the abscissa
perf@y.values[[1]] # tpr on the ordinate
```

We can clearly see the points (0.48,1) and (1,1) of the last two thresholds.

The performance function can be used to obtain many performance measures of a score. The code below can be used to retrieve the true positive rates and the false positive rates that will be the abscissas and ordinates of the points of the ROC curve.

The **plot** method then allows to plot the ROC curve.

```
plot(perf, main="ROC curve of knn")
abline(a=0, b=1, col=2)
```

The first bisector symbolizes the ROC curve that would be obtained with a very bad score (classes randomly mixed along the score). This situation corresponds to an area under the ROC curve of 0.5. If this curve passed through the point (0,1) (top left), there would be a threshold allowing to have 100% true positives (all malignant individuals predicted malignant) and 0% false positives (no benign individuals predicted malignant). A ROC curve passing through this point would correspond to a perfect score and an area under the curve of 1.

The code below allows you to retrieve the value of the area under the ROC curve.

```
perf <- performance(pred, "auc")
auc <- perf@y.values[[1]]
auc
```

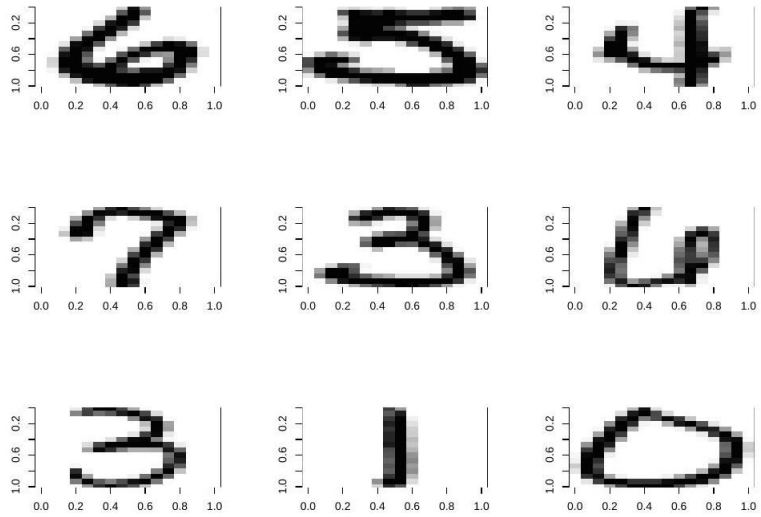
The area under the ROC curve of the knn method (on test data) is 0.96. A value of 1 would correspond to a perfect score capable of perfectly predicting the data. A value of 0.5 would correspond to the worst case (absolutely non-discriminating score).

5. Address this learning problem by Linear Discriminant Analysis.
   Plot the ROC curve and calculate the AUC.
6. Address this learning problem by Quadratic Discriminant Analysis.
   Plot the ROC curve and calculate the AUC.
7. Compare the predictive performances of the classifiers as well as the predicted values.

Exercise 4 *The objective of this exercise is to do automatic recognition of handwritten characters.*

Retrieve the datasets ***numbers_train.txt*** and ***numbers_test.txt***. Each file contains 500 images of dimension 16×16 and each image represents a handwritten character (a number between 0 and 9). We therefore have $Y \in \{0,\ldots,9\}$ and $X = (X_1,\ldots,X_{256}) \in \mathbb{R}^{256}$. This is a grayscale image where each pixel takes a value between 0 (black) and 1 (white).

1. Import the 500 images from the file ***numbers_train.txt***.
2. View the first nine images.

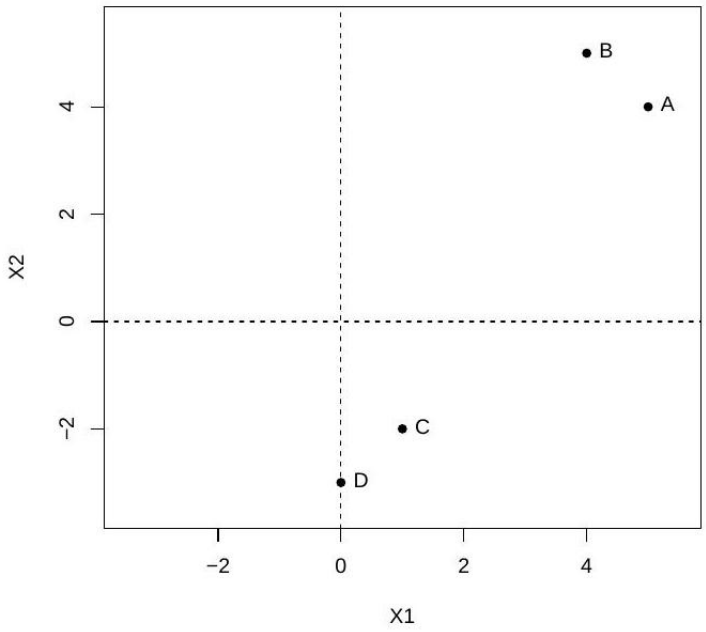


3. Predict with the `lda` method the classes of the 500 images of the training set and calculate the learning error rate.
4. Import the 500 images from the file ***numbers_test.txt***.
5. Now predict the classes of the 500 images of the test set and calculate the test error rate.
6. Now try to use the `qda` method. What do you observe? Explain why.
7. Finally, estimate the error of the `lda` method by LOO (Leave One Out) cross-validation and then by 5-fold cross-validation. Comment on your results.

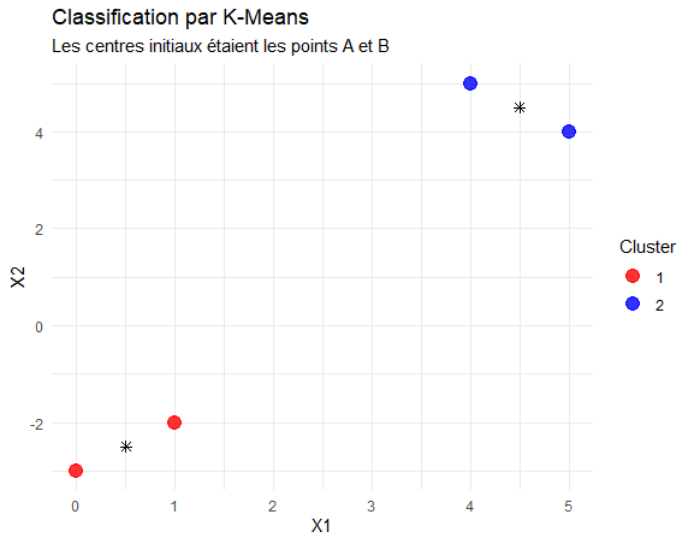
3 Worksheet No. 3 - Partitioning with the k-means method

Exercise 1 We consider the data table where 4 individuals (here points) A, B, C and D are described on two variables (X1 and X2) :

	X1	X2
A	5	4
B	4	5
C	1	-2
D	0	-3



1. Recall the definition of a partition.
2. Which quality criterion is optimized by the k-means method?
3. Is this criterion optimized locally or globally? Why?
4. Propose a bad (in the sense of this criterion) partition into two classes of $\Omega = \{A, B, C, D\}$.
5. Apply the k-means method by hand, taking as initial centers the points A and B. What partition into two classes is obtained?
6. Determine the quality of the partition. (We will calculate the intra-class inertia, the total inertia and deduce the percentage of inertia explained by this partition.)
7. With R, the k-means method taking as initial centers the points A and B leads to the partition into two following classes :



and the following output :

```
> kmeans_result$totss
[1] 67
```

```
> kmeans_result$tot.withinss
[1] 2
```

8. What do the outputs `totss`, `tot.withinss` correspond to?
9. Deduce the percentage of inertia explained by this partition.

Remarque.
— Here the weights of the individuals are all $w_i = 1$. In this case, we speak of total sum of squares rather than inertia
— In English we speak of total, within, between sum of squares
— We can verify that the total inertia (T) is the sum of the intra inertia (W) and the inter inertia (B). We deduce that the percentage of inertia explained by the partition is B/T or 1-W/T.

TP3 - k means

Agnès LAGNOUX & Nicolas SAVY

Exercise 2 : on simulated data

Load the required librairies

```
library(cluster)      # Pour l'algorithme k-means
library(factoextra)   # Pour la visualisation
library(ggplot2)      # Pour la visualisation
```

Generation of a simulated dataset with 3 clusters

```
set.seed(123)
n <- 150
data <- data.frame(
  x = c(rnorm(n, mean = 2, sd = 0.5), rnorm(n, mean = 6, sd = 0.5), rnorm(n, mean = 10, sd = 0.5)),
  y = c(rnorm(n, mean = 2, sd = 0.5), rnorm(n, mean = 6, sd = 0.5), rnorm(n, mean = 10, sd = 0.5))
)
```

```
head(data$x)
```

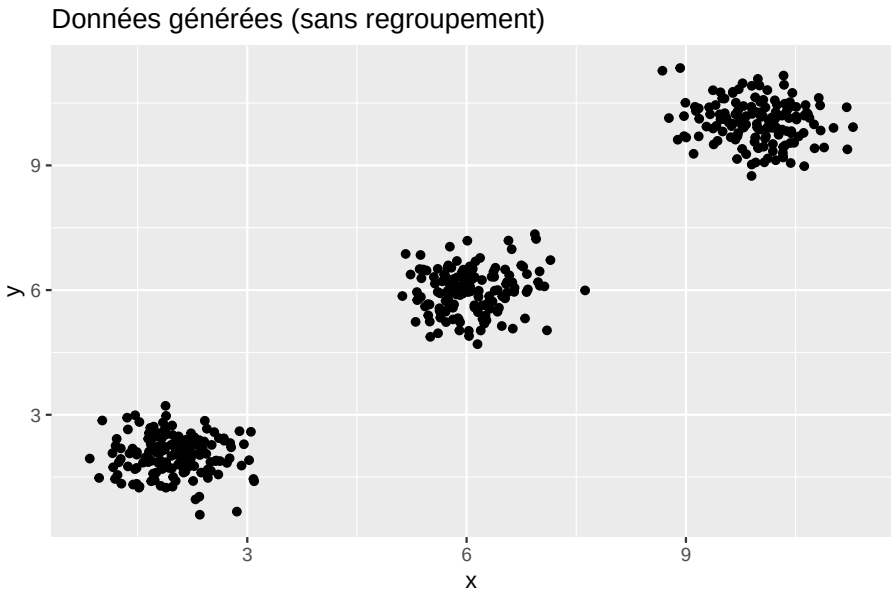
```
## [1] 1.719762 1.884911 2.779354 2.035254 2.064644 2.857532
```

```
length(data$x)
```

```
## [1] 450
```

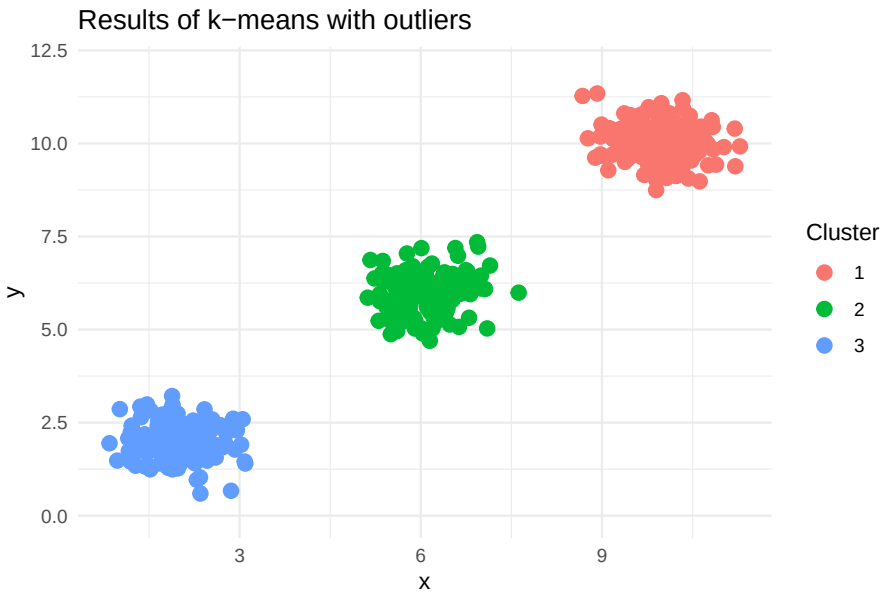
Data visualization

```
ggplot(data, aes(x, y)) +
  geom_point() +
  ggtitle("Données générées (sans regroupement)")
```



Clustering with 3 clusters

```
set.seed(123)
kmeans_result_1 <- kmeans(data, centers = 3)
data$kmeans_result_1 <- as.factor(kmeans_result_1$cluster)

# Visualization of the results
ggplot(data, aes(x = x, y = y, color = kmeans_result_1)) +
  geom_point(size = 3)  + ylim(0,12) +
  labs(title = "Results of k-means with outliers", color = "Cluster") +
  theme_minimal()
```



Importance of renormalization

The k-means method relies on distance calculations (usually the Euclidean distance) to group points into clusters. If the variables have different scales, those with larger amplitudes will have a disproportionate influence on the calculation of distances, which can distort the clustering results.

Example:

- If one variable is measured in kilometers and another in meters, the first will dominate the calculations.
- Renormalizing (often using standardization or normalization) allows to put all variables on the same scale, preventing some variables from biasing the distances.

```
# Adding a variable with a different scale
# We now consider points of R^3 and no more of R^2

data$z <- c(rnorm(2 * n, mean = 50, sd = 5), rnorm(n, mean = 100, sd = 5))
head(data)
```

```
##          x         y kmeans_result_1        z
## 1 1.719762 2.7152012               3 57.79354
## 2 1.884911 2.5233144               3 50.35254
## 3 2.779354 2.2176445               3 50.64644
## 4 2.035254 2.3575892               3 58.57532
## 5 2.064644 2.4585875               3 52.30458
## 6 2.857532 0.6695386               3 43.67469
```

```
# Comparison with and without renormalisation
# k-means without renormalization
kmeans_no_scaling <- kmeans(data, centers = 3, nstart = 25)
data$cluster_no_scaling <- as.factor(kmeans_no_scaling$cluster)

# k-means with renormalization
data_scaled <- scale(data[,c(1,2,4)])
kmeans_scaling <- kmeans(data_scaled, centers = 3, nstart = 25)
data$cluster_scaling <- as.factor(kmeans_scaling$cluster)

# Data visualization
p1 <- ggplot(data, aes(x, y, color = cluster_no_scaling)) +
  geom_point() +
  ggtitle("Clustering without renormalization") +
  theme(legend.position = "none")

p2 <- ggplot(data, aes(x, y, color = cluster_scaling)) +
  geom_point() +
  ggtitle("Clustering avec renormalisation") +
  theme(legend.position = "none")

library(gridExtra)
grid.arrange(p1, p2, ncol = 2)
```


```
head(data)
```

```
##          x         y kmeans_result_1        z cluster_no_scaling
## 1 1.719762 2.7152012               3 57.79354                   1
## 2 1.884911 2.5233144               3 50.35254                   2
## 3 2.779354 2.2176445               3 50.64644                   2
## 4 2.035254 2.3575892               3 58.57532                   1
## 5 2.064644 2.4585875               3 52.30458                   1
## 6 2.857532 0.6695386               3 43.67469                   2
##   cluster_scaling
## 1               2
## 2               2
## 3               2
## 4               2
## 5               2
## 6               2
```

```
head(data_scaled)
```

```
##               x          y          z
## [1,] -1.2976579 -0.9980443 -0.3731645
## [2,] -1.2476335 -1.0560598 -0.6834974
## [3,] -0.9767027 -1.1484768 -0.6712402
## [4,] -1.2020939 -1.1061656 -0.3405596
## [5,] -1.1931917 -1.0756295 -0.6020860
## [6,] -0.9530221 -1.6165348 -0.9620025
```

```
dim(data)
```

```
## [1] 450   6
```

```
dim(data_scaled)
```

```
## [1] 450   3
```

```
data_scaled[400,]
```

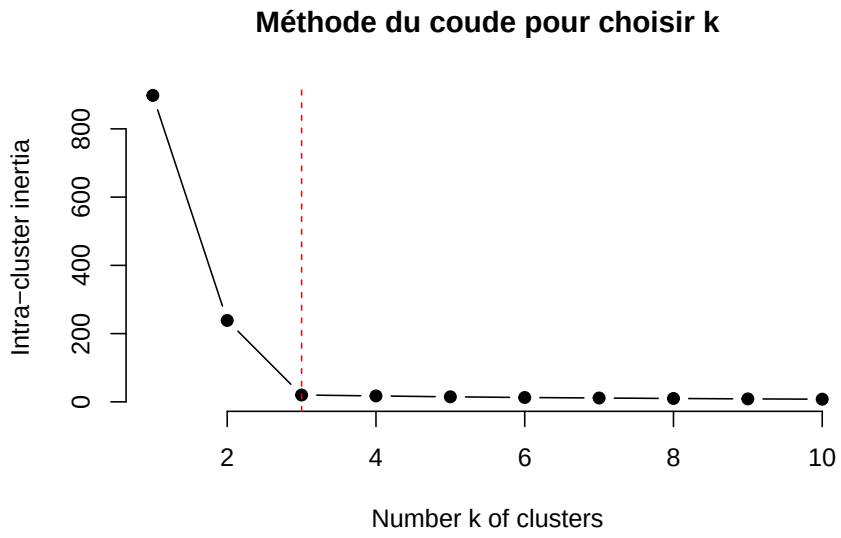
```
##        x        y        z
## 1.156768 1.199970 1.143392
```

Choosing k: elbow method

The elbow method aims to determine the optimal number of clusters (k) by visualizing the relationship between the number of clusters and the total intra-cluster inertia (tot.withinss).

- **Observation**: We generally observe a rapid decrease in inertia when k increases, followed by a slowdown.
- **Elbow Point**: The "elbow" of the graph corresponds to the point where adding an additional cluster no longer significantly improves the reduction of inertia. This point indicates the optimal number of clusters. In the given example, the elbow is observed at k=3, which suggests that 3 clusters are appropriate for the data... this is a bit of the expected result...

```r
# Calculating intra-cluster inertia for different numbers of clusters
set.seed(123)
inertias <- sapply(1:10, function(k) {
  kmeans(scale(data[,c(1,2)]), centers = k, nstart = 25)$tot.withinss
})

# Visualization of the elbow criterion
plot(1:10, inertias, type = "b", pch = 19, frame = FALSE,
     xlab = "Number k of clusters", ylab = "Intra-cluster inertia",
     main = "Méthode du coude pour choisir k")
abline(v = 3, lty = 2, col = "red")
```



Clusters Parameter

Once the choice of number of clusters is made, an important parameter is the center of the clusters.

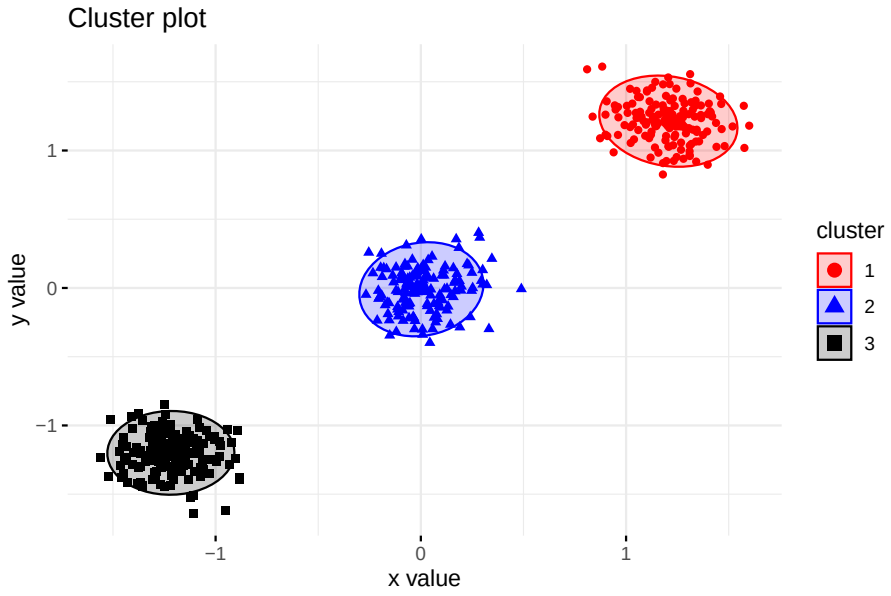
```r
kmeans_result_1 <- kmeans(data[,c(1,2)], centers = 3)
kmeans_result_1$centers
```

```
##          x        y
## 1 9.976992 10.031090
```

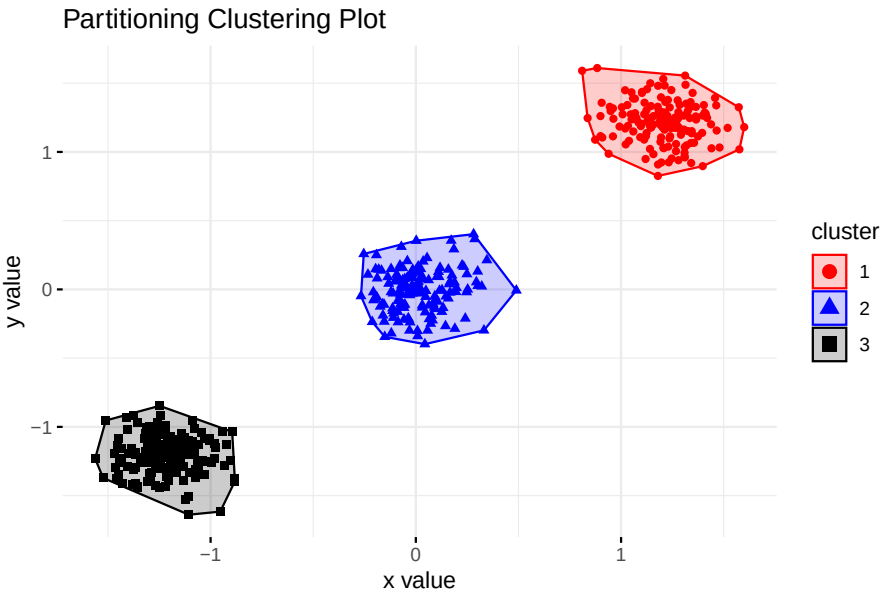
```
## 2 6.046623  5.985517
## 3 1.987819  2.032118
```

A relevant visualization is the graph of clusters enriched with the center and confidence ellipses or the envelope of the data by ***xcluster***.

```
fviz_cluster(kmeans_result_1, data = data[,c(1,2)],
             labelsize = 0,
             palette=c("red", "blue", "black"),
             ellipse.type = "t",
             star.plot = F,
             repel = F,
             ggtheme = theme_minimal())
```




```
fviz_cluster(kmeans_result_1, data = data[,c(1,2)],
             labelsize = 0,
             palette=c("red", "blue", "black"),
             ggtheme = theme_minimal(),
             star.plot = F,
             repel = F,
             main = "Partitioning Clustering Plot"
             )
```



Performance measures

The silhouette index measures the quality of clustering for each point. It is calculated from two distances:

- $a(i)$: average distance between a point i and the other points of its own cluster.
- $b(i)$: average distance between a point i and the points of the nearest neighboring cluster.

The silhouette for a point i is given by:

$$s(i) = \frac{b(i) - a(i)}{\max(a(i) - b(i))}$$

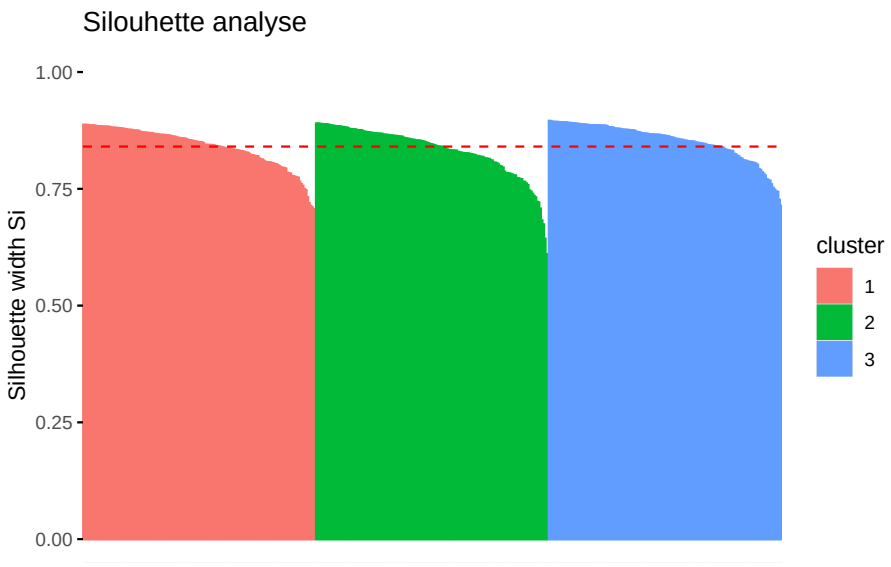
$s(i)$ is between -1 and 1.

- $s(i) \approx 1$: i is well assigned to its cluster.
- $s(i) \approx 0$: i is close to the boundary between two clusters.
- $s(i) \approx -1$: i is probably misassigned.

Global interpretation : The average silhouette index gives an overall measure of the quality of the clustering. The closer the average index is to 1, the better the clustering. A value lower than 0.5 indicates that the cluster structure is weak or poorly defined.

```
# Silhouette computation
set.seed(123)
final_kmeans <- kmeans(scale(data[,c(1,2)]), centers = 3, nstart = 25)
silhouette_scores <- silhouette(final_kmeans$cluster, dist(scale(data[,c(1,2)])))

# Silouhette visualization
fviz_silhouette(silhouette_scores) +
  ggtitle("Silouhette analyse")
```

```
##   cluster size ave.sil.width
## 1       1  150          0.84
## 2       2  150          0.83
## 3       3  150          0.85
```




```
# Displaying the average silhouette index
mean_silhouette <- mean(silhouette_scores[, 3])
cat("Average silhouette index:", round(mean_silhouette, 2))
```

```
## Average silhouette index: 0.84
```

4 Worksheet No. 4 - Hierarchical ascending classification

Exercise 1 We consider the following data table where 4 individuals (here points) A, B, C and D are described on two variables (X1 and X 2) :

	X 1	X 2
A	5	4
B	4	5
C	1	-2
D	0	-3

1. Recall the definition of the maximal link and that of the index h (height).
2. Construct the dendrogram of the maximal link of the 4 individuals.
3. Recall the definition of Ward's distance.
4. Construct the Ward dendrogram by weighting the individuals by 1.
5. Calculate the inter-class inertia of the two-class partition from the Ward dendrogram.
6. Calculate the total inertia and deduce the percentage of inertia explained by this partition.

Practical 4 - Hierarchical ascending classification

Agnès LAGNOUX & Nicolas SAVY

Exercise 2

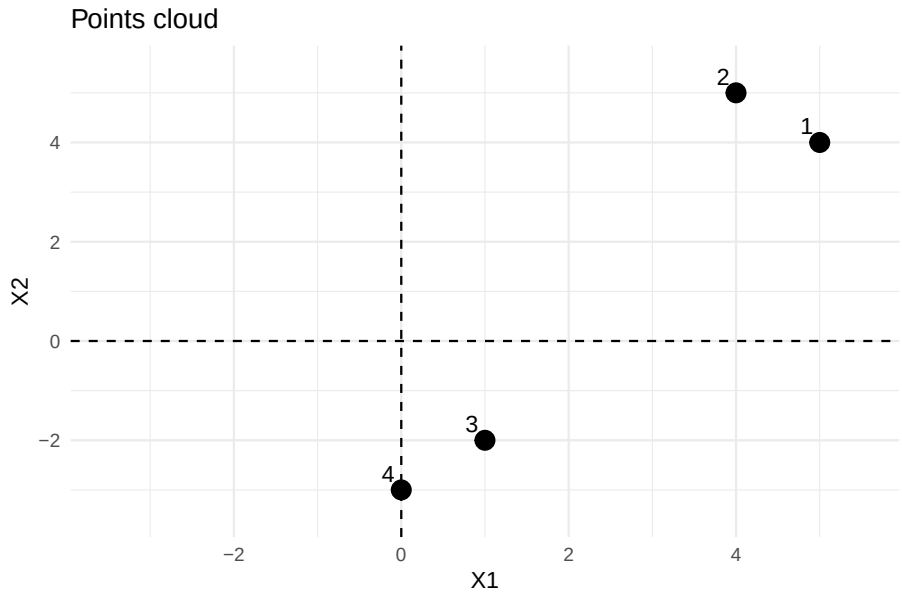
The goal is to find with R the results of Exercise 1.

1. Create a matrix X containing the data.

```
X <- matrix(c(5,4,4,5,1,-2,0,-3),4,2,byrow=TRUE)
colnames(X) <- c("X1","X2")
rownames(X) <- c("1", "2","3","4")
```

2. Visualize the dataset.

```
# Loading of ggplot2
library(ggplot2)

# Generation of the data
X <- data.frame(
  X1 = c(5, 4, 1, 0),
  X2 = c(4, 5, -2, -3),
  label = factor(1:4)
)

# Graphical reprezentation with ggplot2
ggplot(X, aes(x = X1, y = X2)) +
  geom_point(size = 4) +  # Points
  geom_text(aes(label = label), vjust = -0.5, hjust = 1.5) +  # Add the labels
  geom_hline(yintercept = 0, linetype = "dashed") +           # Horizontal line
  geom_vline(xintercept = 0, linetype = "dashed") +           # Vertical line
  theme_minimal() +                                           # Minimal theme
  xlim(-3.5, 5.5) +                                           # x-axis limits
  ylim(-3.5, 5.5) +                                           # y-axis limits
  labs(title = "Points cloud", x = "X1", y = "X2")
```



3. Construct the hierarchy with the maximal link.

```
d <- dist(X)                          # compute the Euclidean distances between points
treeC <- hclust(d, method="complete") # all the results are stored in the object named tree
```

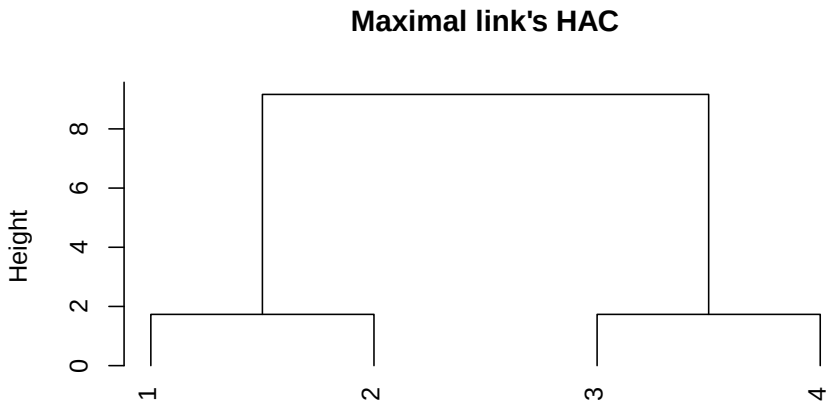
4. Determiner the heights of the classes.

```
treeC$height # give the height of the classes in the dendogram
```

```
## [1] 1.732051 1.732051 9.165151
```

5. Represent the dendrogram of such hierarchy.

```
plot(treeC, main="Maximal link's HAC", xlab ="", sub="", hang=-1)
```



6. Cut the dendrogram to get the partition in two classes of the maximal link's HAC.

```
cutree(treeC, k=2 )  # allow to obtain the partition in two classes
```

```
## [1] 1 1 2 2
```

```
# here, the two classes are C1={1,2} et C2={3,4}
```

7. Construct now Ward's hiérarchy and retrieve the lengths of Exercise 1.

```
treeW <- hclust(d, method="ward.D2")
sum(treeW$height)
```

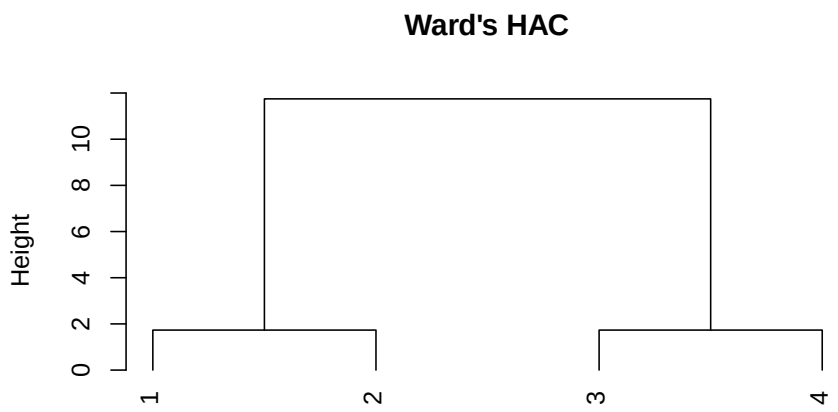
```
## [1] 15.21144
```

NB1: in this exercise, the weights of the individuals are all equal to 1.

NB2: with this setting, we find that the sum of the heights of the tree is worth 67, that is to say the total inertia (here the sum of the total squares) calculated by hand and found previously with the k-means.

8. Represent the dendrogram of such hierarchy.

```
plot(treeW, main="Ward's HAC", xlab ="", sub="",hang=-1)
```




```
cutree(treeW,k = 2) # we find again the two classes C1={1,2} et C2={3,4}
```

```
## [1] 1 1 2 2
```

9. Find from the heights of this dendrogram:

- the total inertia,
- the inter-class inertia of the partition into two classes.

Deduce the share of the total variance explained by this partition.

```
# Calculation of the share of inertia explained by Ward partitions

# Total inertia = sum of heights
Tot <- sum(treeW$height)

# Inter-class inertia = sum of the k-1 greatest heights (k=number of classes)
```

```
h <- sort(treeW$height, decreasing=TRUE)
K <- 2
B <- sum(h[1:K-1])

# Part of inertia explained by the partition = inter inertia/total inertia
B/Tot
```

```
## [1] 0.77227
```

Practical 4 - Hierarchical ascending classification

Agnès LAGNOUX & Nicolas SAVY

Exercise 3 : Hierarchical ascending classification with R

Objective

In this practical, we will explore hierarchical ascending clustering (HAC) in R. We will cover the main steps:

1. Data preparation
2. Performing the HAC
3. Visualizing the results with ggplot2
4. Interpreting the results (quantitative and qualitative analysis)
5. Finding the optimal number of clusters and cross-validation
6. Discussing the hypotheses and performance of the classification.

Step 1: Data preparation

Loading the necessary libraries

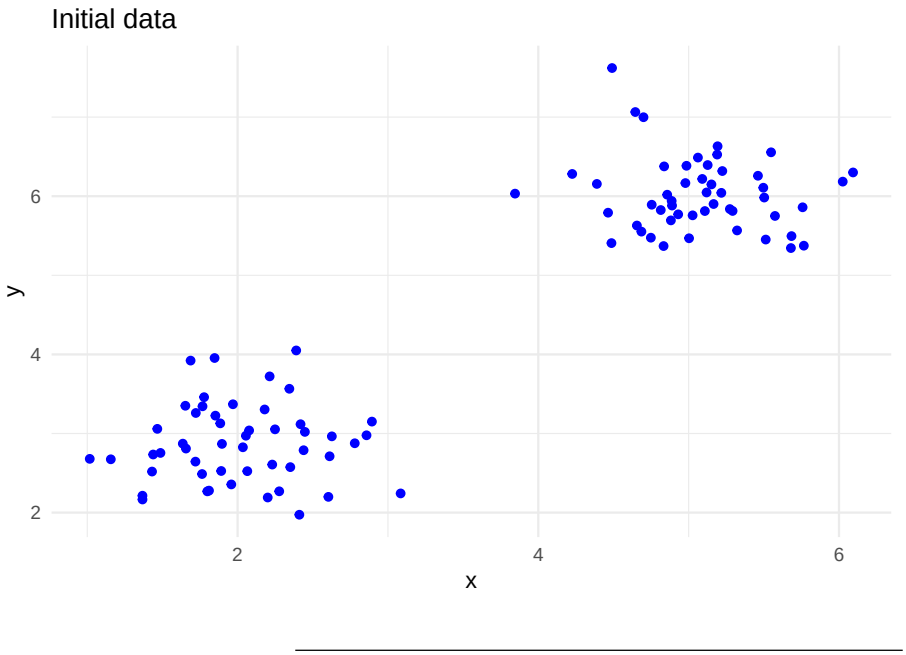
```
library(ggplot2)
library(cluster)
library(dendextend)
library(factoextra)
```

Generation of the simulated data

```
set.seed(123)
data <- data.frame(
  x = c(rnorm(50, mean = 2, sd = 0.5), rnorm(50, mean = 5, sd = 0.5)),
  y = c(rnorm(50, mean = 3, sd = 0.5), rnorm(50, mean = 6, sd = 0.5))
)
```

Visualization of the raw data

```
ggplot(data, aes(x = x, y = y)) +
  geom_point(color = 'blue') +
  theme_minimal() +
  ggtitle("Initial data")
```



Step 2: Realization of the HAC

Computation of the distance matrix

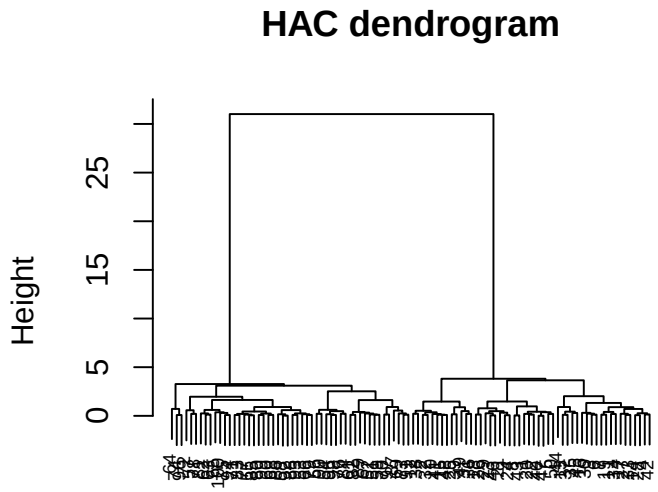
```
distance_matrix <- dist(data, method = "euclidean")
```

Hierarchical ascending classification

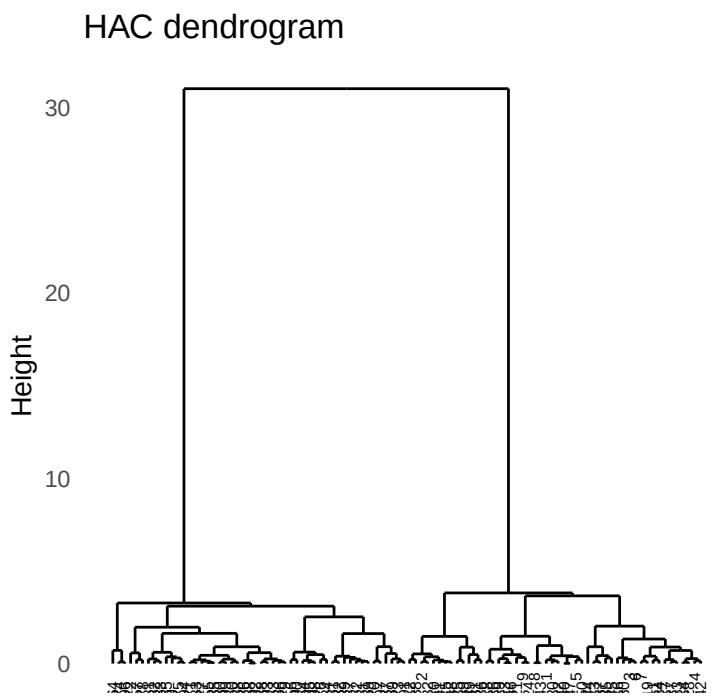
```
cah <- hclust(distance_matrix, method = "ward.D2")
```

Display of the basic dendogram

```
# Direct method
plot(cah, main = "HAC dendrogram", xlab = "", sub = "", cex = 0.6)
```

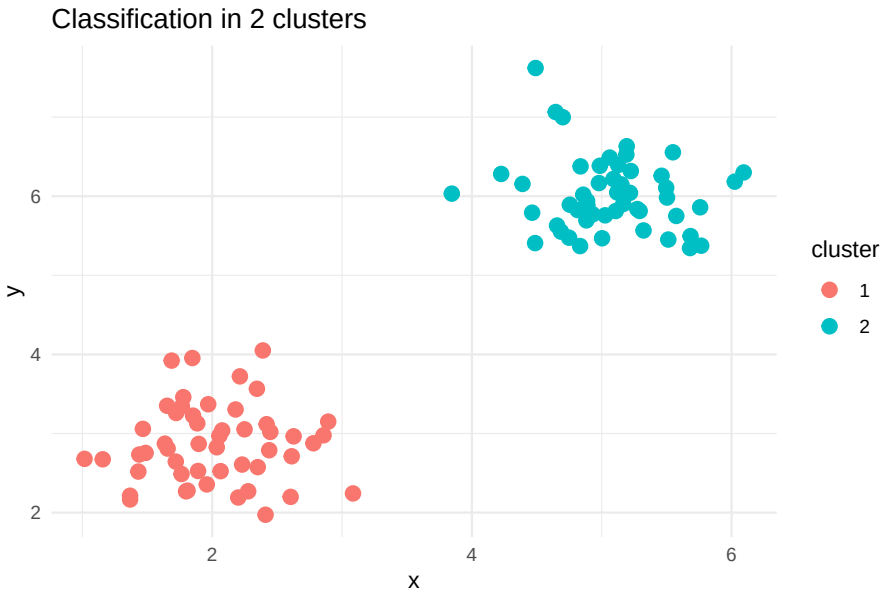



```
# Method with ggplot2
library(ggdendro)

# Conversion into format ggplot2
dendro_data <- ggdendro::dendro_data(cah)

# Extraction of the labels of the leaves
label_data <- dendro_data$labels

# Display of the basic dendogram with ggplot2
ggplot() +
  geom_segment(data = dendro_data$segments, aes(x = x, y = y, xend = xend, yend = yend)) +
  geom_text(data = label_data, aes(x = x, y = y, label = label),
            hjust = 2,
            angle = 90,
            size = 2) +
  labs(title = "HAC dendrogram", x = "", y = "Height") +
  theme_minimal() +
  theme(axis.text.x = element_blank(),
        axis.ticks.x = element_blank(),
        panel.grid = element_blank())
```

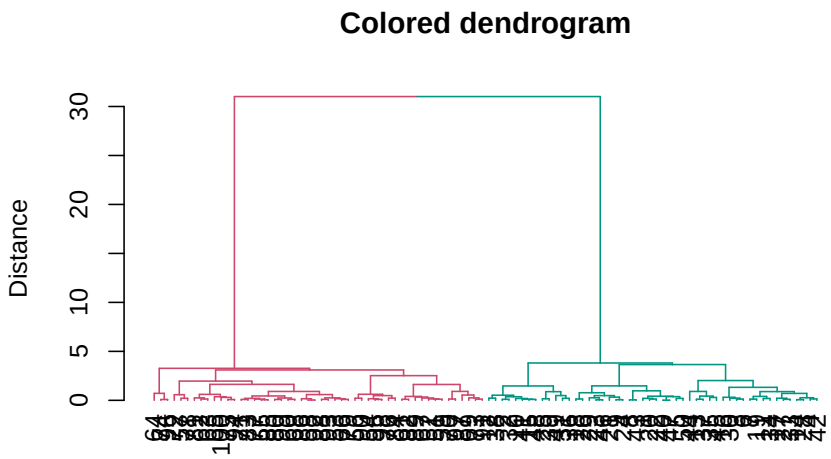


Step 3: Visualization of the results with ggplot2

```
# Cutting in clusters
k <- 2  # Number of desired clusters
clusters <- cutree(cah, k = k)
data$cluster <- as.factor(clusters)

# clusters visualization
ggplot(data, aes(x = x, y = y, color = cluster)) +
  geom_point(size = 3) +
  theme_minimal() +
  ggtitle(paste("Classification in", k, "clusters"))
```




```
# Dendrogramme enhanced
dend <- as.dendrogram(cah)
dend <- color_branches(dend, k = k)
plot(dend, main = "Colored dendrogram", ylab = "Distance")
```



Step 4: Results interpretation

Quantitative analysis of the clusters

1. **Clusters sizes**

```
# Size of each cluster
table(clusters)
```

```
## clusters
##  1  2
## 50 50
```

2. **Clusters centers**

```
# Computation of the centers
centres <- aggregate(. ~ cluster, data = data, mean)
print(centres)
```

```
##   cluster        x        y
## 1       1 2.017202 2.873050
## 2       2 5.073204 6.019403
```

3. **Intra- and inter inertias**

```
# Intra-cluster inertia
intra_inertia <- sum(sapply(unique(clusters), function(k) {
  cluster_data <- data[clusters == k, 1:2]
  sum(rowSums((cluster_data - colMeans(cluster_data))^2))
}))

# Total inertia
total_inertia <- sum(rowSums((data[1:2] - colMeans(data[1:2]))^2))

# Inter-cluster inertia
inter_inertia <- total_inertia - intra_inertia
cat("Intra-cluster inertia:", intra_inertia, "\n")
```

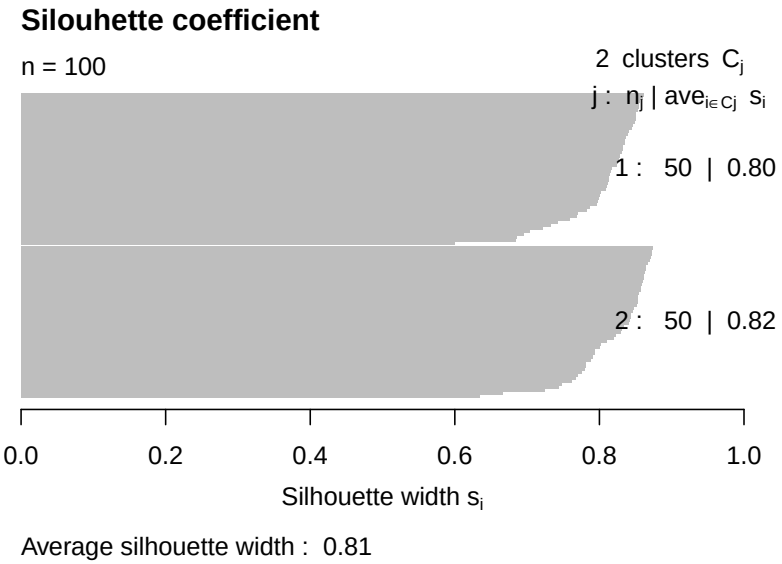
```
## Intra-cluster inertia: 125.814
```

```
cat("Inter-cluster inertia:", inter_inertia, "\n")
```

```
## Inter-cluster inertia: 480.7845
```

4. **Silouhette coefficient**

```
# Computation of the silouhette coefficient
sil <- silhouette(clusters, distance_matrix)
plot(sil, main = "Silouhette coefficient")
```



Qualitative cluster analysis

1. **Grouping structure**

- **Visual observation**: examine the spatial distribution of points within clusters using visualizations such as those created with ggplot2. Check whether each cluster appears well defined or whether there are significant overlaps between them.
- **Internal homogeneity**: check whether points within the same cluster are close to each other. Compact clusters indicate good internal cohesion.
  - **Anomalies**: identify isolated or misclassified points. These anomalies can reveal complex structures or outliers in your dataset.

2. **Intra- and inter-cluster Distances**

- **Intra-cluster distances**: assess the proximity of points within each cluster by calculating the average of the distances between points within the same cluster.

```
intra_distances <- sapply(unique(clusters), function(k) {
      cluster_data <- data[clusters == k, 1:2]
      mean(dist(cluster_data))
    })
    names(intra_distances) <- paste("Cluster", unique(clusters))
    print(intra_distances)
```

```
## Cluster 1 Cluster 2
## 0.8610846 0.8019067
```

- **Inter-cluster distances**: measure the average distance between cluster centers to assess their separation.
- Low intra-cluster inertia and high inter-cluster inertia indicate well-separated clusters.

```
cluster_centers <- aggregate(. ~ cluster, data = data, mean)
    inter_distances <- dist(cluster_centers[, -1])
    print(as.matrix(inter_distances))
```

```
##          1        2
## 1 0.000000 4.386193
## 2 4.386193 0.000000
```

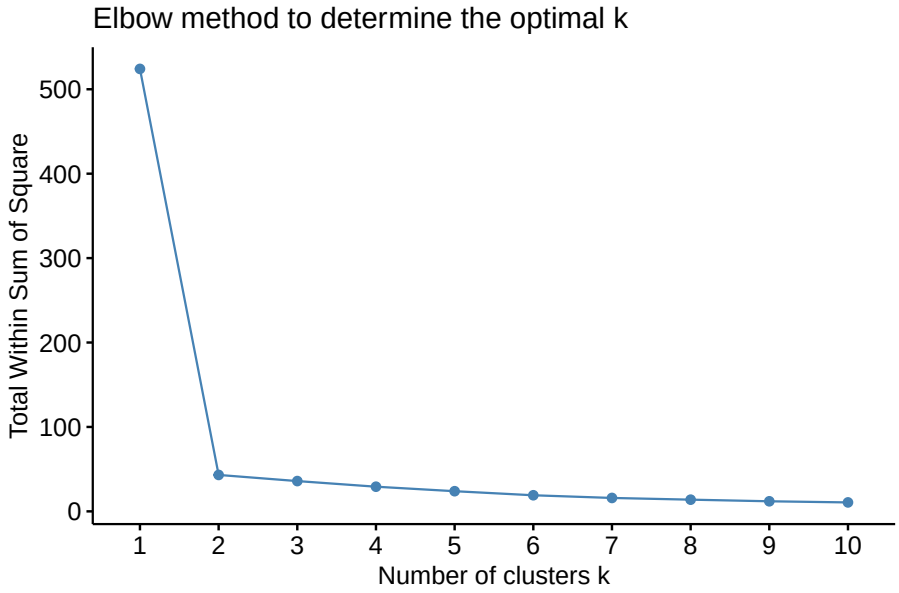
- **Consistency with assumptions**: compare these distances with your initial expectations about the structure of the data. If the clusters are too close, this may indicate an inadequate number of clusters or significant similarities between some classes.

Step 5: Search of the optimal number of clusters and cross validation

Search of the optimal number of clusters

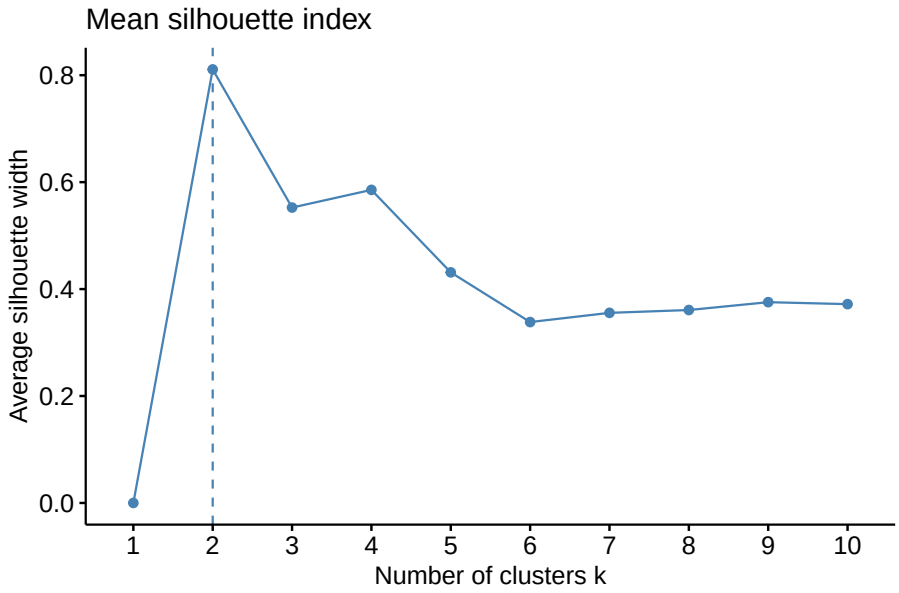
1. **Elbow method**

```r
# Use factoextra to plot the elbow curve
fviz_nbclust(data[1:2], FUN = hcut, method = "wss") +
  ggtitle("Elbow method to determine the optimal k")
```



2. **Mean silhouette index**

```r
# Plot the mean silhouette for any number of clusters
fviz_nbclust(data[1:2], FUN = hcut, method = "silhouette") +
  ggtitle("Mean silhouette index")
```




```
# Cross validation with division of the data
set.seed(123)
train_indices <- sample(1:nrow(data), size = 0.7 * nrow(data))
train_data <- data[train_indices, 1:2]
test_data <- data[-train_indices, 1:2]

# HAC on the training dataset
train_dist <- dist(train_data)
train_cah <- hclust(train_dist, method = "ward.D2")
train_clusters <- cutree(train_cah, k = 2)

# Prediction on the test dataset
# Note: HAC is not directly predictive, but we can qualitatively assess
# the consistency of groupings between train and test
test_dist <- dist(rbind(train_data, test_data))
cat("Qualitative cross-validation performed")
```

Cross validation

```
## Qualitative cross-validation performed
```

Exercise 4 *The objective of this exercise is to compare the 3 clustering methods studied : k-means, k-medoids and CHA on a real data set.*

We use the data ***USArrests.csv*** where the 50 American states are described by 3 variables indicating the number (per 100,000 inhabitants)

— of arrests for murder (Murder),
— of assaults (Assault),
— of rape (Rape).

As well as a variable indicating the proportion of urban population (UrbanPop)
We want to find classes of states that are similar on the 3 crime variables.

1. Data preparation : loading data, descriptive data analysis, data visualization and data normalization.
2. Apply the k-means method to perform a clustering into 4 clusters. Add the results to the dataset. Visualize the results in each of the planes. Visualize the results on the first 2 principal components.
3. Same questions with the k-medoids method
4. Same questions with the CAH method
5. Visually compare the results obtained.
6. For each method, calculate the average silhouette index, the Dunn index and the calculation time.
7. Compare the performance of the methods.
8. For the method that seems most relevant to you, give an interpretation of the clustering of the standardized data via a normalized PCA.

Exercise 5 *The objective of this exercise is to do clustering after dimension reduction.*

We will use the dataset ***protein.rda***.

The idea is to reduce the dimension before applying clustering to automatically find groups of countries with the same protein consumption profiles. To do this, we can perform a PCA and keep q principal components (q to choose) or perform a clustering of variables in J classes and keep the J synthetic variables of the classes (first principal component of the PCA of the variables of the class).

1. We first reduce the dimension by normalized PCA before doing the clustering. We must therefore choose the number q of principal components to retain.
   (a) What is the maximum number r of principal components for this data? Check that the Ward dendrogram obtained with the standardized data is identical to that obtained with the r principal components (all).
   (b) We decide to keep $q = 4$ principal components. What is the percentage of information retained by these new variables?
   (c) Then compare the Ward dendrogram obtained with these first 4 principal components to that obtained with the standardized data.
2. We now reduce the dimension by clustering variables. We must therefore choose the number J of clusters to retain.
   (a) Use the code below to obtain a hierarchical tree of the nine variables. The objective of variable clustering is to find clusters of variables that are correlated with each other (positively or negatively). When looking at this tree, we choose to retain $J = 4$ cluster of variables.

```
library(ClustOfVar)
treevar <- hclustvar(protein)
plot(treevar, main="Clustering of variables")
rect.hclust(treevar, k=4)
```

(b) Then compare the dendrogram obtained with the 4 synthetic variables to that obtained with the standardized data. What could be the advantage of this dimension reduction method?