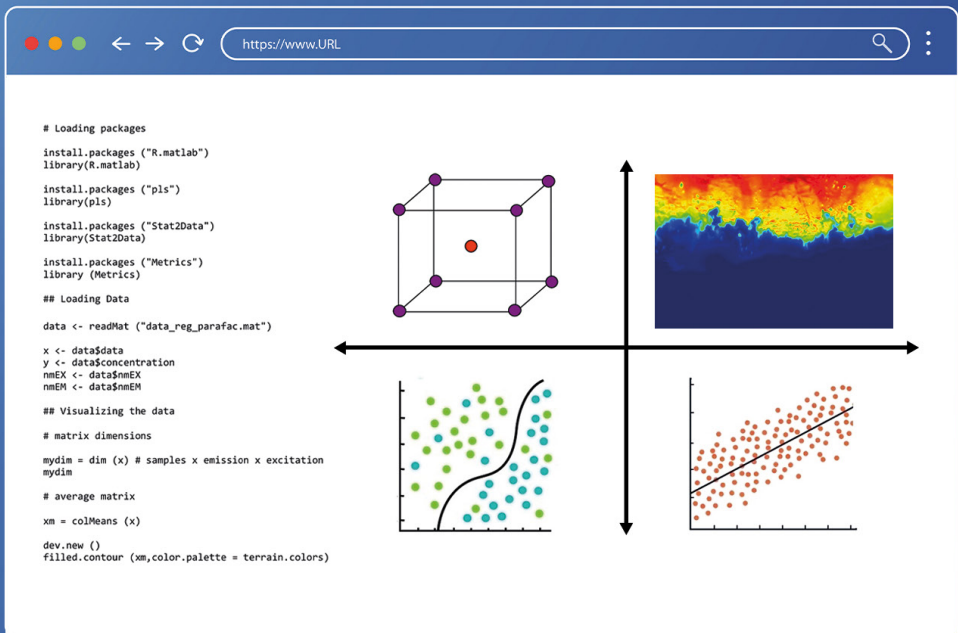


Kássio Michell Gomes de Lima
Camilo de Lelis Medeiros de Moraes
Ana Carolina de Oliveira Neves

CHEMOMETRICS

A UNIVERSITY COURSE USING
THE R LANGUAGE



```
# Loading packages
install.packages("R.matlab")
library(R.matlab)

install.packages("pls")
library(pls)

install.packages("Stat2Data")
library(Stat2Data)

install.packages("Metrics")
library(Metrics)

## Loading Data

data <- readMat("data_reg_parafac.mat")

x <- data$data
y <- data$concentration
nmEX <- data$nmEX
nmEM <- data$nmEM

## Visualizing the data

# matrix dimensions
mydim = dim(x) # samples x emission x excitation
mydim

# average matrix
xm = colMeans(x)
dev.new()
filled.contour(xm,color.palette = terrain.colors)
```

The screenshot displays an R script editor with a browser address bar showing "https://www.URL". The script includes the following code:

```
# Loading packages
install.packages("R.matlab")
library(R.matlab)

install.packages("pls")
library(pls)

install.packages("Stat2Data")
library(Stat2Data)

install.packages("Metrics")
library(Metrics)

## Loading Data

data <- readMat("data_reg_parafac.mat")

x <- data$data
y <- data$concentration
nmEX <- data$nmEX
nmEM <- data$nmEM

## Visualizing the data

# matrix dimensions
mydim = dim(x) # samples x emission x excitation
mydim

# average matrix
xm = colMeans(x)
dev.new()
filled.contour(xm,color.palette = terrain.colors)
```

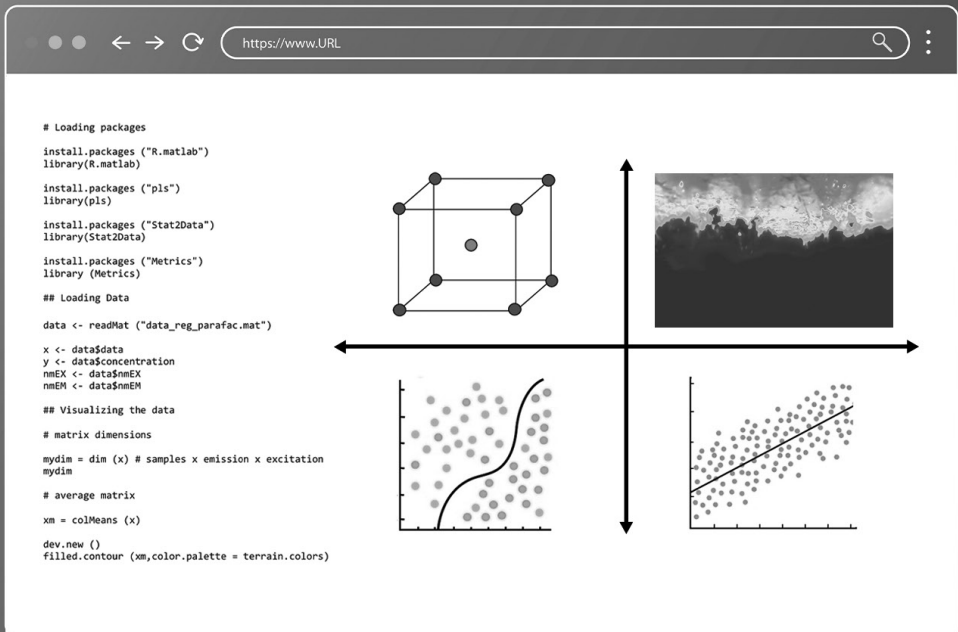
The visualization area contains four plots:

- A 3D cube with purple dots at its vertices and a red dot in the center.
- A contour plot showing a terrain-like surface with a color gradient from blue to red.
- A scatter plot with green and blue dots and a black trend line.
- A scatter plot with orange dots and a black trend line.

Kássio Michell Gomes de Lima
Camilo de Lelis Medeiros de Moraes
Ana Carolina de Oliveira Neves

CHEMOMETRICS

A UNIVERSITY COURSE USING
THE R LANGUAGE



The image shows a browser window displaying an R script on the left and four data visualization plots on the right. The browser's address bar shows "https://www.URL". The R script includes the following code:

```
# Loading packages
install.packages("R.matlab")
library(R.matlab)

install.packages("pls")
library(pls)

install.packages("Stat2Data")
library(Stat2Data)

install.packages("Metrics")
library(Metrics)

## Loading Data

data <- readMat("data_reg_parafac.mat")

x <- data$data
y <- data$concentration
nmEX <- data$nmEX
nmEM <- data$nmEM

## Visualizing the data

# matrix dimensions
mydim = dim(x) # samples x emission x excitation
mydim

# average matrix
xm = colMeans(x)
dev.new()
filled.contour(xm,color.palette = terrain.colors)
```

The four plots are:

- Top-left: A 3D cube with a central point and lines connecting the vertices.
- Top-right: A grayscale image of a landscape with a dark foreground and a bright, cloudy sky.
- Bottom-left: A scatter plot of gray dots with a black sigmoidal curve fitted to the data.
- Bottom-right: A scatter plot of gray dots with a black linear regression line fitted to the data.

Chief editor

Profª Drª Antonella Carvalho de Oliveira

Executive editor

Natalia Oliveira

Editorial assistant

Flávia Roberta Barão

Librarian

Janaina Ramos

Graphic project

Ellen Andressa Kubisty

Luiza Alves Batista

Nataly Evilin Gayde

Thamires Camili Gayde

Cover pictures

iStock

Art edition

Luiza Alves Batista

2024 by Atena Editora

Copyright © Atena Editora

Copyright of the text © 2024 The authors

Copyright of the edition © 2024 Atena Editora

Rights for this edition granted to Atena Editora by the authors.

Open access publication by Atena Editora



All content in this book is licensed under a Creative Commons Attribution License. Attribution-Non-Commercial-NonDerivatives 4.0 International (CC BY-NC-ND 4.0).

The content of the text and their data in their form, correctness and reliability are the sole responsibility of the authors, and they do not necessarily represent the official position of Atena Editora. It is allowed to download the work and share it as long as credits are given to the authors, but without the possibility of altering it in any way or using it for commercial purposes.

All manuscripts were previously submitted to blind evaluation by peers, members of the Editorial Board of this Publisher, having been approved for publication based on criteria of academic neutrality and impartiality.

Atena Editora is committed to ensuring editorial integrity at all stages of the publication process, avoiding plagiarism, fraudulent data or results and preventing financial interests from compromising the publication's ethical standards. Suspected scientific misconduct situations will be investigated to the highest standard of academic and ethical rigor.

Editorial Board**Exact and earth sciences and engineering**

Prof. Dr. Adélio Alcino Sampaio Castro Machado – Universidade do Porto

Profª Drª Alana Maria Cerqueira de Oliveira – Instituto Federal do Acre

Profª Drª Ana Grasielle Dionísio Corrêa – Universidade Presbiteriana Mackenzie

Profª Drª Ana Paula Florêncio Aires – Universidade de Trás-os-Montes e Alto Douro

Prof. Dr. Carlos Eduardo Sanches de Andrade – Universidade Federal de Goiás

Profª Drª Carmen Lúcia Voigt – Universidade Norte do Paraná

Prof. Dr. Cleiseano Emanuel da Silva Paniagua – Colégio Militar Dr. José Aluisio da Silva Luz / Colégio Santa Cruz de Araguaína/TO

Profª Drª Cristina Aledi Felseburgh – Universidade Federal do Oeste do Pará

Prof. Dr. Diogo Peixoto Cordova – Universidade Federal do Pampa, Campus Caçapava do Sul

Prof. Dr. Douglas Gonçalves da Silva – Universidade Estadual do Sudoeste da Bahia

Prof. Dr. Eloi Rufato Junior – Universidade Tecnológica Federal do Paraná

Profª Drª Érica de Melo Azevedo – Instituto Federal do Rio de Janeiro

Prof. Dr. Fabrício Menezes Ramos – Instituto Federal do Pará

Prof. Dr. Fabrício Moraes de Almeida – Universidade Federal de Rondônia

Profª Drª Glécilla Colombelli de Souza Nunes – Universidade Estadual de Maringá

Prof. Dr. Hauster Maximiler Campos de Paula – Universidade Federal de Viçosa

Profª Drª Iara Margolis Ribeiro – Universidade Federal de Pernambuco

Profª Drª Jéssica Barbosa da Silva do Nascimento – Universidade Estadual de Santa Cruz

Profª Drª Jéssica Verger Nardeli – Universidade Estadual Paulista Júlio de Mesquita Filho

Prof. Dr. Juliano Bitencourt Campos – Universidade do Extremo Sul Catarinense

Prof. Dr. Juliano Carlo Rufino de Freitas – Universidade Federal de Campina Grande

Prof. Dr. Leonardo França da Silva – Universidade Federal de Viçosa

Profª Drª Luciana do Nascimento Mendes – Instituto Federal de Educação, Ciência e Tecnologia do Rio Grande do Norte

Prof. Dr. Marcelo Marques – Universidade Estadual de Maringá

Prof. Dr. Marco Aurélio Kistemann Junior – Universidade Federal de Juiz de Fora

Prof. Dr. Marcos Vinicius Winckler Caldeira – Universidade Federal do Espírito Santo

Profª Drª Maria Iaponeide Fernandes Macêdo – Universidade do Estado do Rio de Janeiro

Profª Drª Maria José de Holanda Leite – Universidade Federal de Alagoas

Profª Drª Mariana Natale Fiorelli Fabiche – Universidade Estadual de Maringá

Prof. Dr. Miguel Adriano Inácio – Instituto Nacional de Pesquisas Espaciais

Prof. Dr. Milson dos Santos Barbosa – Universidade Tiradentes

Profª Drª Natiéli Piovesan – Instituto Federal do Rio Grande do Norte

Profª Drª Neiva Maria de Almeida – Universidade Federal da Paraíba

Prof. Dr. Nilzo Ivo Ladwig – Universidade do Extremo Sul Catarinense

Profª Drª Priscila Natasha Kinas – Universidade do Estado de Santa Catarina

Profª Drª Priscila Tessmer Scaglioni – Universidade Federal de Pelotas

Prof. Dr. Rafael Pacheco dos Santos – Universidade do Estado de Santa Catarina

Prof. Dr. Ramiro Picoli Nippes – Universidade Estadual de Maringá

Profª Drª Regina Célia da Silva Barros Allil – Universidade Federal do Rio de Janeiro

Prof. Dr. Sidney Gonçalo de Lima – Universidade Federal do Piauí

Prof. Dr. Takeshy Tachizawa – Faculdade de Campo Limpo Paulista

Chemometrics: a university course using the R language

Diagramming: Nataly Gayde
Correction: Maiara Ferreira
Indexing: Amanda Kelly da Costa Veiga
Review: The authors
Authors: Kássio Michell Gomes de Lima
Camilo de Lelis Medeiros de Morais
Ana Carolina de Oliveira Neves

International Cataloging-in-Publication Data (CIP)

L732 Lima, Kássio Michell Gomes de
Chemometrics: a university course using the R language /
Kássio Michell Gomes de Lima, Camilo de Lelis
Medeiros de Morais, Ana Carolina de Oliveira Neves. –
Ponta Grossa - PR: Atena, 2024.

Format: PDF

System requirements: Adobe Acrobat Reader

Access mode: World Wide Web

Includes bibliography

ISBN 978-65-258-2700-1

DOI: <https://doi.org/10.22533/at.ed.001241508>

1. Chemometrics. 2. Analytical chemistry. I. Lima,
Kássio Michell Gomes de. II. Morais, Camilo de Lelis
Medeiros de. III. Neves, Ana Carolina de Oliveira. IV. Title.

CDD 543.085

Prepared by Librarian Janaina Ramos – CRB-8/9166

Atena Editora

Ponta Grossa – Paraná – Brasil

Telefone: +55 (42) 3323-5493

www.atenaeditora.com.br

contato@atenaeditora.com.br

AUTHORS' DECLARATION

The authors of this work: 1. Attest that they do not have any commercial interest that constitutes a conflict of interest in relation to the published scientific article; 2. They declare that they actively participated in the construction of their manuscripts, preferably in: a) Study design, and/or data acquisition, and/or data analysis and interpretation; b) Elaboration of the article or revision in order to make the material intellectually relevant; c) Final approval of the manuscript for submission; 3. They certify that published scientific articles are completely free from fraudulent data and/or results; 4. Confirm correct citation and reference of all data and interpretations of data from other research; 5. They acknowledge having informed all sources of funding received for carrying out the research; 6. Authorize the publication of the work, which includes the catalog records, ISBN (Internacional Standard Serial Number), D.O.I. (Digital Object Identifier) and other indexes, visual design and cover creation, interior layout, as well as the release and dissemination according to Atena Editora's criteria.

PUBLISHER'S DECLARATION

Atena Editora declares, for all legal purposes, that: 1. This publication constitutes only a temporary transfer of copyright, right to publication, and does not constitute joint and several liability in the creation of published manuscripts, under the terms provided for in the Law on Rights copyright (Law 9610/98), in art. 184 of the Penal Code and in art. 927 of the Civil Code; 2. Authorizes and encourages authors to sign contracts with institutional repositories, with the exclusive purpose of disseminating the work, provided that with due acknowledgment of authorship and editing and without any commercial purpose; 3. All e-books are open access, so it does not sell them on its website, partner sites, e-commerce platforms, or any other virtual or physical means, therefore, it is exempt from copyright transfers to authors; 4. All members of the editorial board are PhDs and linked to public higher education institutions, as recommended by CAPES for obtaining the Qualis book; 5. It does not transfer, commercialize or authorize the use of the authors' names and e-mails, as well as any other data from them, for any purpose other than the scope of dissemination of this work.

The origin of this book arose when the authors saw the need to write theoretical and experimental material for the Chemometrics discipline of the undergraduate and postgraduate Chemistry course at UFRN/Brazil. The R programming language was chosen in this book because it presents a free, easily accessible work environment, rich in a variety of packages and various statistical tools capable of exploring different functionalities through different means and formats. Furthermore, with the versatility of being able to work through a command line or through menus pre-defined by the software itself, it is very accessible and practical to explore the main concepts of Chemometrics.

The main topics of Chemometrics that are explored in undergraduate and postgraduate courses at UFRN are, among them, Descriptive Statistics and their properties (Chapter 1), Design and Optimization of Experiments (Chapter 2), Pattern Recognition (Chapter 3), Higher-order Multivariate Classification (Chapter 4), Higher-order Multivariate Regression (Chapter 5) and Digital Images (Chapter 6). All chapters contain the fundamentals descriptions and examples on simulated or real data through scripts in the R language. In addition, the book presents an appendix which brings together the main scripts in the R language for importing, pre-processing and organizing matrices that are necessary for the development of multivariate models. Finally, to encourage learning, proposed exercises were created so that students can try to answer them based on theory and solved examples in each chapter.

The authors

The support, collaboration and encouragement of several people was relevant for this textbook, to whom we would like to give special thanks. Firstly, we would like to thank our dear Professor Luiz Seixas das Neves (Institute of Chemistry/UFRN) for the learning, for the opportunity to share ideas and knowledge, and above all, for the consideration and trust placed in our work throughout our academic career. Secondly, we want to thank Professor Hélio Scatena Júnior (in memory) for teaching the Contemporary Chemistry subject offered in Chemistry courses at UFRN for some years. We also want to thank the Research Group in Biological Chemistry and Chemometrics at UFRN, especially professors Edgar Perin Moraes, Davi Serradella Vieira, Fabrício Gava Menezes and Luiz Henrique da Silva Gasparotto who enriched the Chemometrics environment with discussions and applications in studies published in over more than a decade. I would also like to thank all the students and collaborators who passed through and were part of the world of Chemometrics built here at UFRN and in various parts of the world, especially the esteemed Prof. Francis L. Martin (United Kingdom).

I, professor, Kássio Lima, now want to thank my wife Katarina, who was undoubtedly a great pillar in the writing and preparation phases of this work. Always helpful, understanding and the person who always believed in my work. For all the love and friendship to whom I can only thank for understanding the number of hours put into this book, especially regarding the education of our beloved children, João Pedro and Maria Luísa. These beloved children, two graces achieved, were and will continue to be my greatest inspirations for this book.

I, professor, Camilo Morais, would like to thank my friends and family who have always believed and supported my work. In particular, I want to thank my wife and partner, Julyana, for showing me her love, dedication and friendship at all times during the writing and preparation of this book. Finally, I would like to thank the other authors, professors Kássio Lima and Ana Carolina, for the invitation, discussions, understanding and hours dedicated for preparing this book with such care.

I, professor, Ana Carolina, would like to especially thank those who have always been my biggest supporters throughout the journey that brought me here: my parents, Gladson and Maria da Conceição, and Fabrício, a great friend and life partner. Without your support, it wouldn't have been possible. I would also like to thank the other authors, professors Kássio and Camilo, for the invitation, for the rich exchange of knowledge and for the friendship.

CHAPTER 1 - DESCRIPTIVE STATISTICS AND ITS PROPERTIES.....	1
1.1 Some important concepts in descriptive statistics.....	1
1.2 Normal distribution	5
1.3 Normality test – Shapiro-Wilk.....	8
1.4 Levene's test.....	9
1.5 Standardized normal distribution.....	10
1.6 t-test	11
1.7 Analysis of variance: ANOVA	15
1.8 Post-Hoc Tests in ANOVA.....	20
1.9 Non-parametric tests	22
Proposed exercises	29
CHAPTER 2 - DESIGN OF EXPERIMENTS	31
2.1 Fundamental steps for planning experiments	31
2.2 Full Factorial Design.....	32
2.3 Factorial Design 2^2	33
2.4 Factorial Design 2^3	36
2.5 Unreplicated Factorial Design.....	41
2.6 Fractional Factorial Design.....	42
2.7 2^k Factorial Design with center-point.....	47
2.8 Box-Behnken Design	50
2.9 Multi-level Factorial Design.....	53
2.10 Nonlinear optimization for response surface	57
2.11 Simplex-lattice Design	59
2.12 Simplex-centroid Design.....	63
Proposed exercises	65
CHAPTER 3 - PATTERN RECOGNITION	67
UNSUPERVISED ANALYSIS.....	68

3.1 Cluster Analysis	68
3.2 Hierarchical Cluster Analysis (HCA)	68
3.3 K-means.....	75
3.4 Principal Component Analysis (PCA)	78
3.5 Multivariate Curve Resolution with Alternating Least Squares (MCR-ALS) ..	86
SUPERVISED ANALYSIS.....	88
3.6 KNN (K-Nearest Neighbors)	88
3.6 Linear Discriminant Analysis (LDA)	91
3.7 Quadratic Discriminant Analysis (QDA)	97
3.8 Support Vector Machines (SVM)	100
3.10 Decision Trees.....	107
Proposed exercises	109
CHAPTER 4 - HIGHER ORDER MULTIVARIATE CLASSIFICATION.....	111
4.1 Types of analytical data	112
4.2 Methods for selecting samples in multivariate classification	113
4.3 Methods for selecting variables in multivariate classification	113
4.4 Performance metrics	114
4.5 – PCA-LDA	115
4.6 SPA-LDA	127
4.7 GA-LDA	138
4.8 TUCKER3–LDA	151
4.9 PARAFAC–LDA	163
Proposed exercises	174
CHAPTER 5 - HIGHER ORDER MULTIVARIATE CALIBRATION.....	176
5.1 Univariate Calibration	177
5.2 Calibration by Least Squares – univariate model	177
5.3 Multiple Linear Regression (MLR)	190

5.4 MLR – SPA.....	196
5.5 Principal Component Regression (PCR).....	199
5.6 Partial least squares (PLS) regression	203
5.6 PARAFAC.....	207
5.7 MCR-ALS	212
5.8 UPLS/RBL	218
5.9 N-PLS.....	223
Proposed exercises	227
CHAPTER 6 - DIGITAL IMAGES	229
6.1 Digital Imaging: an overview	230
6.2 RGB to HSV and Grayscale Conversion	233
6.3 Exploratory Analysis	234
6.4 Multivariate classification.....	240
6.5 Multivariate Regression	253
Proposed exercises	261
APPENDIX A	262
A1 – Loading spectral data using R.....	262
A2 – Preprocessing spectral data in R.....	263
A3 – Loading molecular fluorescence data: excitation-emission matrix (EEM)....	267

DESCRIPTIVE STATISTICS AND ITS PROPERTIES



"It is not knowledge, but the act of learning, not possession but the act of getting there, which grants the greatest enjoyment." **Friedrich Gauss (1777-1855)**

CHAPTER IDEA

We need statistical calculations to make decisions about the quality of our experimental measurements. In this chapter, some important concepts for this decision making will be presented, including: normal distribution, parametric and non-parametric statistical tests.

Upon completing the chapter, you should be able to:

- Explain what a normal distribution is and when its application is appropriate.
- Use descriptive statistical parameters on a set of observations.
- Define parametric and non-parametric tests.
- Discuss the assumptions for parametric tests to be used.
- Determine the best statistical strategy for a real experiment.
- Propose new analytical methodologies based on statistical tests.
- Build new scripts in R language for decision making.

1. Some important concepts in descriptive statistics

In general, chemistry researchers use authentic replicas of a sample to carry out an analytical method. In this sense, some types of variables appear (qualitative, quantitative, discrete and continuous). Variable, in statistics, is the assignment of a number to each characteristic of the observation unit, that is, it is a defined mathematical function of the population.

The statistical technique to be used must be appropriate to the type of variable. In quantitative variables, we have a list of position measures (mean, median, quartiles, mode) and dispersion measures (variance, coefficient of variation, range). The equations and definitions of the main descriptive statistical parameters will be presented in **Table 1**.

Table 1: Descriptive statistical parameters.

Population average	$\mu = \frac{\sum_{i=1}^N x_i}{N}$
Sample mean	$\bar{x} = \frac{\sum_{i=1}^N x_i}{N}$
Median	It is the central value in a dataset that has been arranged in order of magnitude
Population standard deviation	$\sigma = \sqrt{\frac{\sum_{i=1}^N (x_i - \mu)^2}{N}}$
Sample standard deviation	$s = \sqrt{\frac{\sum_{i=1}^N (x_i - \bar{x})^2}{N - 1}}$
Combined standard deviation	$s = \sqrt{\frac{\sum_{i=1}^{N_1} (x_i - \bar{x})^2 + \sum_{i=1}^{N_2} (x_i - \bar{x})^2 + \sum_{i=1}^{N_3} (x_i - \bar{x})^2 + \dots}{N_1 + N_2 + N_3 + \dots - N_t}}$
Population variance	σ^2
Sample variance	s^2
Mean standard error	$S_m = \frac{s}{\sqrt{N}}$
Coefficient of Variation	$CV = \frac{s}{\bar{x}} \times 100$
Relative Standard Deviation (RSD)	$DPR = \frac{s}{\bar{x}}$
Spread or range (f)	It is the difference between the highest value and the lowest value in the set
Mode (m_o)	It is the value (or values) of maximum frequency
Absolute frequency (f_i)	It is the number of times a given value (x _i) is observed.
Relative frequency or proportion (p'_i)	It is the quotient between its absolute frequency and the total number of observations. $p'_i = \frac{f_i}{n}$ $\sum f_i = n$ $\sum p'_i = 1$
Maximum	Largest value from a set of n observations
Minimum	Smallest value from a set of n observations
Quartiles	These are values that divide a data sample into four equal parts. 1st. Quartile: the value that leaves 25% of the data below it and 75% above it. 2nd. Quartile: the value that leaves 50% of the data below it and 50% above it (median). 3rd. Quartile: the value that leaves 75% of the data below it and 25% above it.
Pearson's coefficient of Skewness (A_p)	$A_p = \frac{\bar{x} - m_o}{s}$ If $ A_p < 0.15$, symmetric distribution If $0.15 \leq A_p \leq 1$, moderately asymmetric distribution If $ A_p > 1$, strongly asymmetric distribution
Amplitude (R)	Difference between the largest and smallest value in a set of n observations

Where:

- x_i = individual values of the variable X;
- N = number of measurements for the entire population/sample;
- N_t = total number of data sets being combined.

Each of these statistical parameters will be calculated and discussed below, through Example 1.

Example 1: The following example was carried out by students of the Quantitative Analytical Chemistry course at the Federal University of Rio Grande do Norte, 2023.1, in which we have the experimental results of the calibration of a 10 mL graduated pipette. Using the data in the table below, calculate and discuss all parameters or graphs generated by this experiment.

Replicas	Volume, mL
1	9.988
2	9.993
3	9.986
4	9.980
5	9.975
6	9.982
7	9.986
8	9.982
9	9.981
10	9.990

R Script

```
##### Loading packages #####

# Installation/loading of packages if not installed

install.packages ("dplyr")
library ( dplyr )
install.packages ("psych")
library ( psych )

##### Loading the database #####

# Select the working directory ( working directory )
# Session > Set Working Directory > Choose Directory

# Load the database

pipette = c( 9.988, 9.993, 9.986, 9.980, 9.975, 9.982, 9.986, 9.982, 9.981, 9.990)
data <- data.frame (pipette) # create dataset
View (data)
glimpse (data)

##### Measures for quantitative variables #####

# Amplitude
range( pipette )
hist( pipette )
summary( pipette )
boxplot (pipette)

##### Frequency tables of categorical variables #####

# Absolute frequencies:

table (pipette)

# Relative frequencies:

prop.table ( table (pipette))

# Functions describe and describe.by (package 'psych')
describe (pipette)
```


1.2 Normal distribution

Experimental measurements present random or undetermined errors. These errors cannot be eliminated, but can be estimated from a certain number of repetitions. When we perform a very large number of experiments, we result in a bell-shaped curve known as a Gaussian curve or normal error curve, as shown in **Figure 1** .

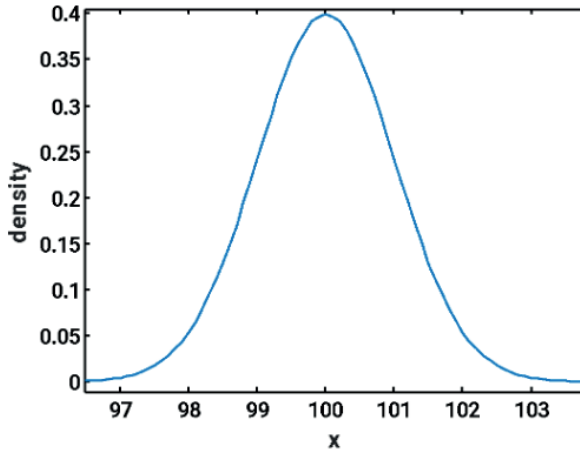


Figure 1 : Probability density function graph.

The two important parameters of the Gaussian distribution are the mean (μ), describing where the experimental values are centered, and the variance (σ^2) which describes their degree of dispersion. Depending on the parameters (μ and σ), we will have different normal distributions. Furthermore, the variable X is a continuous variable comprised between $-\infty < x < +\infty$ and the area under the curve is equal to 1. This area we call probability. The probability density function, **Equation 1**, is described as:

$$f_x(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2} \quad -\infty < x < +\infty \quad \text{Eq. 1}$$

A common way of writing whether X describes a normal distribution is to write $X \sim N(\mu, \sigma^2)$. Below, we have a practical exercise using the R language to build the probability function, histograms and check some sample statistics.

Example 2: Using the R script proposed below, we will explore the dataset by calculating some statistical parameters. We will build graphs (histograms and boxplots) and discuss statistical concepts presented here (probability density, for example).

R Script

```
# Normal distribution in R

##### Loading packages #####

install.packages ("dplyr")
library ( dplyr )
install.packages ("psych")
library ( psych )

# Normal probability density function
# Sequence for the horizontal axis

x = seq (from = 95, to = 105, length =500)

# Evaluating x in the pdf

y = dnorm (x, mean = 100, sd = 1)

# Plotting the pdf

plot (x, y, type = "l", ylab = "density")

# Random sample taken from the normal distribution
#x ~ N(100,1)

set.seed (7) # seed for randomization

# Collecting the sample

sample = rnorm ( 1000,mean = 100, sd = 1 )

# Sample histogram

hist ( sample, main = "sample, n = 1000", ylab = "frequency", col = "cadetblue")

# Sample statistics

mean (sample) # average
sd (sample) # standard deviation
summary (sample) # sample distribution measures (min; 1st Quart.; Median; Mean; 3rd
Qu.; Max)
boxplot ( sample, ylab = "sample", col = "gold")
```

Example 3: We will show, through this R script, how it is possible to calculate the probability of finding a continuous variable (x). In this case, less than 3.3, assuming that the probability function can be described as $X \sim N(3.4, 0.1^2)$. Thus, we will show how it is possible to calculate the probability of a continuous variable (x) between 3.1 and 3.7, assuming the same probability function. Additionally, we will construct a graph that presents a region of interest within a given probability function.

R Script

```
# probability calculations

##### Loading packages #####
install.packages ("dplyr")
library( dplyr )
install.packages ("psych")
library(psych)

#X ~ N( 3.4, 0.1^2)
# P( X < 3.3) = ?

pnorm ( 3.3, mean = 3.4, sd = 0.1)

# P( 3.1 < X < 3.7) = P(X < 3.7) - P (X < 3.1)?

pnorm ( 3.7, mean = 3.4, sd = 0.1) - pnorm (3.1, mean = 3.4, sd = 0.1)

# Illustrating P (X < 3.3)
# x axis sequence

x1 = seq ( from = 3, to = 3.8, length = 500)

# evaluating x1 in the pdf

y1 = dnorm ( x1, mean = 3.4, sd = 0.1)

# plotting the pdf

plot ( x1, y1, type = "l", ylab = "density")

# sequence of x to color

a = seq (3, 3.3, length = 100)

# evaluating the pdf

b = dnorm ( a, mean = 3.4, sd = 0.1)

# coloring probability of interest

polygon (c( 3,a ,3.3), c(0,b,0), col = "coral")
```

1.3 Normality test – Shapiro-Wilk

In 1965, Samuel Sanford Shapiro and Martin Bradbury Wilk [1] proposed a statistical test to evaluate whether a given data distribution is similar to the Gaussian normal distribution. As a result, the test will return the W statistic, which will have an associated significance value, the p-value.

Here, it is important to define some fundamental concepts based on hypothesis tests that will serve to understand decision making.

1. the null hypothesis indicated by H_0 states that there is no effect or variation in the population; the alternative hypothesis is indicated by H_1 and both must be mutually exclusive and exhaustive. After the test, we make one of two decisions: reject H_0 and accept H_1 ; or not reject H_0 .
2. there are two types of errors possible to occur in hypothesis testing: i) type I error (rejecting H_0 and H_0 being true); or, ii) type II error (not rejecting H_0 and H_0 being false).
3. the probability of a type I error occurring is indicated by the significance level (α). It is usually $\alpha = 5\%$
4. the probability of a type II error occurring is indicated by β .
5. the probability of occurring a value as extreme (much greater or much smaller than the value of H_0) as that obtained in the sample is indicated by P . Here, we can have two situations: i) if $P \leq \alpha$, we reject H_0 (in this case, the sample value is as or more extreme than the critical value). ii) if $P > \alpha$, we do not reject H_0 (in this case, the sample value is less extreme than the critical value).

Example 4: Using the R script below, we will show how to determine whether a given variable presents normality using the Shapiro-Wilk test.

R Script

```
# Installing packages and importing a dataset (30 obs vs 7 variables)

install.packages ("dplyr")
library( dplyr )
install.packages ("RVAideMemoire")
library( RVAideMemoire )

# Read csv file

data <- read.csv( 'Database 2.csv', sep = ';', dec = ',',
                  stringsAsFactors = T, fileEncoding = "latin1")

# View data in a separate window

View (data)

# View a summary of the data

glimpse (data)

# Checking data normality

shapiro.test(data$Height)

# Checking data normality
## Shapiro by group ( RVAideMemoire package )

#( dependent variable ~ independent variable, data)

byf.shapiro(Salary ~ Education , data)
```

1.4 Levene's test

We can consider that Levene's test [2] evaluates the equality of variances (homogeneity of variances or homoscedasticity) of a variable calculated for two or more groups. After testing, there are two possibilities:

H_0 : group variances are homogeneous $\rightarrow p > 0.05$

H_1 : group variances are not homogeneous $\rightarrow p < 0.05$

Example 5: Using the R Script below, we will show whether there is homogeneity of variances through Levene's test for a given dataset.

R Script

```
# Checking the homogeneity of variances

## Loading Packages

install.packages ("car")
library ( car )

## Loading data

data <- read.csv( 'Database 3.csv', sep = ';', dec = ',',
                 stringsAsFactors = T, fileEncoding = "latin1")

# Levene's test

# H0: group variances are homogeneous -> p > 0.05
# H1: group variances are not homogeneous -> p < 0.05

leveneTest ( Grade_Biology ~ Room_Position , data, center = mean )
```

1.5 Standardized normal distribution

Now, we can perform the calculations of the standardized normal distribution, in which the relative frequency is represented in a graphical form as a function of the quantity z . This value is basically the deviation from the mean divided by the standard deviation of the population. Here, the numbers contained in the shaded areas represent the percentage of the area over the curve, which is included among the z values. Let's look at some examples: **i**) 50% of the area of the Gaussian curve is present between -0.67σ and $+0.67 \sigma$; **ii**) 80% of the area of the Gaussian curve is contained between -1.28σ and $+1.28 \sigma$; **iii**) 90% are located between -1.64σ and 1.64σ . These previously mentioned values (50%, 80% and 90%) are also called confidence level, as they consist of the probability that the true mean is located within a certain interval, as long as we have a reasonable estimate of σ . The probability of a result being outside the confidence interval can be called the significance level.

In statistics, we can state that the confidence interval (CI) for a true value of the mean m is the range of values between which the population mean μ is expected to be contained with a certain probability. Its limits are known as confidence limits. This concept was introduced into statistics by Jerzy Neyman in 1937 [3]. When we write a set of experimental measurements in the form of 5.25 ± 0.15 , according to this concept, the true value of the mean must be contained in the range between 5.40 to 5.10, with a certain probability (95%, for example).

Example 6: Using the R script, we will show some probability calculations for standardized distributions.

R Script

```
# Standardized normal distribution

## Loading Packages

install.packages ("rstatix")
library ( rstatix )

# Z ~ N(0,1)
# phi (-1.64) = P(Z < 1.64) = to be determined

pnorm (-1.64)

# P ( z < z_alpha ) = 0.05, z_alpha = to be determined

qnorm (0.05)

# P(- z_alpha < z < z_alpha ) = 0.95, | z_alpha | = to be determined

qnorm (0.025)
abs (qnorm ( 0.025))

# P(z > z_alpha ) = 0.10, z_alpha = to be determined

qnorm (0.10, lower.tail = F)
qnorm (0.90)
```

We cannot estimate the true mean from a single observation. Typically, it is recommended to use the experimental mean of N experimental measurements as an estimate of μ . Thus, we have:

$$IC = \bar{x} \pm \frac{z\sigma}{\sqrt{N}} \quad \text{Eq. 2}$$

Equation 2 is only applied in experiments with no systematic errors and in which the values of s are a good approximation of the values of σ .

1.6 t-test

Often in Chemometrics, we have a limitation in time or in the number of samples to consider that s is a good estimate of σ . The determination of the confidence interval when σ is unknown was originally proposed by the English statistician Mr. WS Gosset, in 1908, in a classic article published in the journal *Biometrika* [4]. The curious fact was: to avoid the discovery of any commercial secret from his employer (hired by the Guinness brewery to statistically analyze the results of alcohol content determinations), Gosset published the

article under the name Student. This pseudonym "Student" and his work on the Student t-distribution became famous in the world of statistics.

Commonly, we can find three types of Student's t-test. Here they are:

- a. **Single-sample t-test:** We use this test when we want to compare the mean of a single sample with a known population mean (reference value). After testing, there are two possibilities:

H_0 : sample mean = reference value $\rightarrow p > 0.05$

H_1 : sample mean \neq reference value $\rightarrow p < 0.05$

Example 7: We will use the following R script to demonstrate how we should employ the single-sample t-test. We will discuss their results and present some graphs that confirm our observations.

R Script

```
# Performing the t-test for one sample

## Loading Packages

install.packages ("rstatix")
library( rstatix )

## Loading the data

data <- read.csv('Database 2.csv', sep = ';', dec = ',',
                 stringsAsFactors = T, fileEncoding = "latin1")

## t-test

t.test ( data$Height , mu = 167)

# Observation:
# The two-tailed test is the default; If you want one-tailed , you need to include:
# alternative = "greater" or alternative = "less"
# Example: t. test ( data$Height , mu = 167, alternative = "greater")
# In this case, the test checks whether the sample mean is greater than the tested
mean
# Visualization of data distribution

boxplot ( data$Height , ylab = "Height (cm)")
```

- b. **t for independent samples:** We use this test when we want to compare the means of two independent samples.

After testing, there are two possibilities:

H_0 : mean of group A = mean of group B $\rightarrow p > 0.05$

H_1 : mean of group A \neq mean of group B $\rightarrow p < 0.05$

Example 8: Use the following R script to calculate the independent samples t-test. We will discuss the results and evaluate your assumptions.

R Script

```
## Installing packages, if not done previously

install.packages("dplyr")
library(dplyr)
install.packages("RVAideMemoire")
library(RVAideMemoire)
install.packages("car")
library(car)

## Loading the data

data <- read.csv('Database 3.csv', sep = ';', dec = ',',
                 stringsAsFactors = T, fileEncoding = "latin1")
View(data)
glimpse (data)

# Performing the t test for independent samples

t.test ( Grade_Biology ~ Room_Position , data, var.equal =TRUE)

t.test ( Grade_Physics ~ Room_Position , data, var.equal =FALSE)

t.test ( Grade_History ~ Room_Position , data, var.equal =FALSE)

# Observation
# var.equal =FALSE does not consider the two variances as equal, between the two
groups
# The two-tailed test is the default; If you want one-tailed , you need to include:
# alternative = "greater" or alternative = "less"
# Example: t.test ( Grade_Biology ~ Room_Position , data, var.equal =TRUE,
alternative = "greater")
# In this case, the test checks whether the average of the first group is greater
than the average of the second
# OR is considering "Front" as the first group
```

```
# Visualization of data distribution

par(mfrow = c( 1,3)) # Graphs come out on the same line
boxplot (Grade_Biology ~ Room_Position , data = data, ylab = "Biology Grades", xlab
= "Position in the Room")
boxplot (Grade_Physics ~ Room_Position , data = data, ylab = "Physics Grades", xlab
= "Position in the Room")
boxplot (Grade_History ~ Room_Position , data = data, ylab = "History Grades", xlab
= "Position in the Room")
```

- c. T-test for dependent (paired) samples:** The t test for dependent samples is used when you want to compare the means of two samples that are dependent, that is, when one sample is obtained from the same population as the other.

Example 9: Using the following R script, we will show how the paired t-test and its results are determined.

R Script

```
## Load the packages that will be used

install.packages ("dplyr")
library(dplyr)
install.packages ("psych")
library(psych)

## Load the database

data <- read.csv('Database 4.csv', sep = ';', dec = ',', fileEncoding = "latin1")
View (data)
glimpse (data)

# Performing the paired t test

t.test (data$Seizures_PT, data$Seizures_S1, paired = TRUE)

# Visualization of data distribution

par(mfrow = c( 1,2)) # Graphs on the same line
boxplot (data$Seizures_PT , ylab = "Number of Seizures", xlab = "Pre-Treatment")
boxplot (data$Seizures_S1, ylab = "Number of Seizures", xlab = "1st week of
Treatment")
```

```
# Descriptive data analysis (part 1)

summary (data$Seizures_PT)
summary (data$Seizures_S1)

# Descriptive data analysis (part 2):

describe (data$Seizures_PT)
describe (data$Seizures_S1)
```

1.7 Analysis of variance: ANOVA

Basically, ANOVA consists of a statistical method for testing the equality of three or more population means, based on the analysis of sample variances. Its purpose is to understand whether there is a significant difference between the groups being compared. Experimental data must be separated into groups according to a characteristic or factor. In this case, each factor can have two or more groups.

a. One-way ANOVA or treatment with repeated measures

Here some requirements need to be met:

- I. populations must present normal distributions and with the same variance;
- II. samples must be random and mutually independent;
- III. the different samples are obtained from populations classified in just one category.

Once these requirements are met, we need to declare the null (H_0) and alternative (H_1) hypotheses for the one-way ANOVA. H_0 assumes that the mean of all populations are equal ($\mu_1 = \mu_2 = \mu_3 = \dots = \mu_k$) and the variance between groups (variation due to interaction between groups) is significantly smaller than the variance within groups (variation due to chance). On the other hand, H_1 assumes that at least one population mean is different, that is, there is an effect of factor or treatment.

Therefore,

H_0 : there are no differences between the group means $\rightarrow p > 0.05$

H_1 : there are differences between the group means $\rightarrow p < 0.05$

The basic idea of ANOVA also consists of the partition of variability, in which one part represents the variability of groups (between groups) and the other represents the variability due to other factors (within groups). A common practice for testing H_0 and H_1 is to summarize the results of the one-way ANOVA test in the form of a table, as shown below:

Table 2: One-way ANOVA.

Source of variation	Sum of Squares (SS)	Degrees of Freedom (DF)	Mean Square (MS)	F
Factor Effect (between groups)	FSS	$I - 1$	$FMS = \frac{FSS}{I-1}$	$\frac{FMS}{MSE}$
Error (within groups)	ESS	$N - 1$	$MSE = \frac{FMS}{N-1}$	
Total	TSS	$N - 1$		

Example 10: Let's look at an R script of a 1-way ANOVA with repeated measures. We will take it step by step and discuss its main results.

R Script

```
## Loading Packages

install.packages ("psych")
library(psych)
install.packages ("ggplot2")
library(ggplot2)

# Read data from a file ( import Dataset )

data <- read.table ( "data.txt" ,header=TRUE)

# Shows boxplots of the classifiers, side by side, in relation to the PCC performance
response

data$Classifiers <- as.factor ( data$Classifiers )
bp <- ggplot ( data, aes (x=Classifiers, y= PCC,fill =Classifiers)) +
  geom_boxplot ( ) +
  labs(title="PCC Boxplot by Classifiers",x="Classifiers", y = "PCC")
bp + theme_classic()

# Create the ANOVA table

data.anova <- aov ( data$PCC ~ data$Classifiers )

# Show the ANOVA table

summary ( data.anova )

# another way to perform ANOVA (using the esyanova package )

install.packages ("easyanova")
library(easyanova)

ea1(data, design = 1, alpha = 0.05)
```

b. multiple ANOVA

When an experiment involves two or more factors, then we have a multiple ANOVA. We test two-way ANOVA when we have a numerical dependent variable of two or more categorical independent variables. This test allows you to verify the effect of each of the independent variables as well as the interaction between them.

Table 2: Two-way ANOVA.

Source of variation	Sum of Squares (SS)	Degrees of Freedom (DF)	Mean Square (MS)	F
Factor A	SSA	$r - 1$	$MSA = \frac{SSA}{r-1}$	$\frac{MSA}{MSE}$
B Factor	SSB	$c - 1$	$MSB = \frac{SSB}{c-1}$	$\frac{MSB}{MSE}$
AB (interaction)	SSAB	$(r-1).(c-1)$	$MSAB = \frac{SSAB}{(r-1)(c-1)}$	$\frac{MSAB}{MSE}$
Error	EES	$rc(n' - 1)$	$MSE = \frac{SSE}{(rc)(n'-1)}$	
Total	TSS	$n - 1$		

Example 11: Below we have an R script to apply a two-factor ANOVA.

R Script

```
# Load the packages that will be used

install.packages ("dplyr")
library( dplyr )
install.packages ("car")
library(car)
install.packages ("rstatix")
library( rstatix )
install.packages ("emmeans")
library( emmeans )
install.packages ("ggplot2")
library(ggplot2)

# csv file (database 6)

data <- read.csv( 'Database 6.csv', sep = ';', dec = ',', fileEncoding = "latin1")
```

```

View (data) # View data in a separate window
glimpse (data) # View a summary of the data

data$Alcohol <- factor ( data$Alcohol ,
                        levels = c( "None",
                                     "2 Mugs",
                                     "4 Mugs"))
summary ( data$Alcohol )

# Check the 3 assumptions (normality, outliers, homogeneity) in the raw data

## 1 - Normality of groups - Shapiro test

data %>% group_by( Gender , Alcohol ) %>%
  shapiro_test ( Memory )

# Here, all the observations follow a normal distribution and the assumption has
# been met for ANOVA

## 2 - Presence of outliers per group in two ways

# form 1 - uses the quartiles calculated without the median

boxplot ( data$Memory ~ data$Gender:data$Alcohol )

# form 2 - uses the quartiles calculated with the median

data %>% group_by ( Gender, Alcohol ) %>%
  identify_outliers ( Memory )

## 3 Verification of homogeneity of variances - Levene's test ( car package )

leveneTest ( Memory ~ Gender * Alcohol , data, center = mean )

# By default, the test performed by the car package is based on the median ( median )
# Mean-based testing is more robust
# Changed to be average based

## Construction of the ANOVA model

model <- aov ( Memory ~ Gender * Alcohol , data)
model$coefficients
summary.aov(model)

# Anova in another way

install.packages ("easyanova")
library(easyanova)

```

```

ea2(data, design = 1)

## Normality test for residuals:

shapiro.test ( model$residuals )

# Ho = Null hypothesis follows normal distribution and  $p > 0.05$ 
# HA = Alternative hypothesis does not follow normal distribution and  $p < 0.05$ 

## Checking the presence of outliers among the residues:

boxplot ( model$residuals )
data$residuals <- model$residuals
data %>% group_by( Gender, Alcohol )%>%
  identify_outliers( residuals )
data %>% identify_outliers ( residuals )
## Verification of homogeneity of variances - Levene's test ( car package )

leveneTest ( residuals ~ Gender * Alcohol , data, center = mean )

# Carrying out ANOVA

## Change in contrast to match SPSS:

options (contrasts = c("contr.sum", "contr.poly"))

## Model creation:

model <- aov( Memory ~ Gender * Alcohol , data)
summary(model)
Anova(model, type = 'III')

# type III = sum of squares of residuals and does not take into account the order
of factors
# type I = depends on the order of factors at insertion time
#  $H_0 p > 0.05$ 
#  $H_a p < 0.05$ 
# Interaction plot (ggplot2 package)
## With genders with different colors

ggplot(data, aes(x = Alcohol, y = Memory, group = Gender, color = Gender)) +
  geom_line(stat = "summary", fun.data = "mean_se", size = 0.6) +
  geom_point(stat = "summary", fun = "mean") +
  geom_errorbar(stat = "summary", fun.data = "mean_se", width = 0.2)

# there is dependence on the effect of alcohol depending on gender

## With Genders with different lines

```

```

ggplot(data, aes(x = Alcohol, y = Memory, group = Gender)) +
  geom_line(stat = "summary", fun.data="mean_se", size = 0.6, aes(linetype =
Gender)) +
  geom_point(stat = "summary", fun = "mean", size = 2, aes(shape = Gender)) +
  geom_errorbar(stat = "summary", fun.data = "mean_se", width = 0.2)

# Estimated Marginal Means ( emmeans package )
# here we will analyze whether there are # statistics between genders as a function
of memory

data %>% group_by(Gender) %>%
  emmeans_test(Memory ~ Alcohol, p.adjust.method = "bonferroni")

data %>% group_by(Alcohol) %>%
  emmeans_test(Memory ~ Gender, p.adjust.method = "bonferroni")

```

In the example above, the two-way ANOVA showed that there is an effect of alcohol $F(2,42) = 20.06$; $p < 0.001$ and interaction between alcohol and gender $F(2,42) = 11.91$; $p < 0.001$ on memory. Subsequent analyzes (estimated marginal means, with Bonferroni correction) showed that alcohol consumption did not affect the memory of female individuals, but the consumption of 4 mugs decreased the memory score of male individuals, when compared to individuals of the same gender who did not consume alcohol, or only consumed 2 mugs.

1.8 Post-Hoc Tests in ANOVA

In Latin, post hoc means "after this", that is, analyzing the experimental data later. The goal of a post-hoc analysis is to find patterns after the study is completed, and to find results that were not the main objective. In the particular case of ANOVA, after determining what differences exist between the means, post hoc range tests and multiple pairwise comparisons can determine which means differ. Range tests identify homogeneous subsets of means that are not different from each other. There are several post-hoc methods available, among them we have the "hsd", "bonferroni", "lsd", "scheffe", "newmankeuls" and "duncan" methods.

The Tukey test [5], named after John Tukey, consists of comparing all possible pairs of means and is based on the least significant difference (LSD), considering the group's percentiles. When calculating the LSD, the distribution of the studentized amplitude, the mean square of the ANOVA residuals and the sample size of the groups are also used.

Example 12: Below, there is an R script for applying Tukey's parametric test. What main results are obtained in this test?

R Script

```
## Creating the dataset

analyst1 = c(10.3, 9.8, 11.4)
analyst2 = c(9.5, 8.6, 8.9)
analyst3 = c(12.1, 13.0, 12.4)
analyst4 = c(9.6, 8.3, 8.2)
analyst5 = c(11.6, 12.5, 11.4)

data <- data.frame (analyst1, analyst2, analyst3, analyst4, analyst5)
dat <- stack (data) # creates vector in stack format

# Performing the ANOVA

anova = aov ( dat$values~dat$ind )
summary( anova )

qf ( 0.95, df1 = 4, df2 = 10) # f critical

# Tukey test
#### Who differs from whom? To do this, we need to compare the averages
#### We will perform the Tukey Test.

tk_test <- TukeyHSD (anova)
tk_test

#### For p values <  $\alpha$  we can say that the means differ
#### at a significance level of 5% ( $\alpha = 0.05$ ).
#### When  $p > \alpha$  it is not possible to say that the means differ.
#### The same result can be expressed by the test graph

plot ( tk_test )

# Finally, we can make a boxplot to better represent the data:

boxplot (data)
```

Here we can also find a statistical package in R called DescTools to perform post-hoc analysis.

R Script

```
# Post-hoc analysis (DescTools Package)
# Post hocs allowed : "hsd", "bonferroni", "lsd", "scheffe", "newmankeuls", "duncan"

install.packages ("DescTools")
library(DescTools)

#PostHocTest
# Using Duncan

PostHocTest(anova, method = "duncan")

# Using TukeyHSD

PostHocTest(anova, method = "hsd")

# Using Bonferroni

PostHocTest(anova, method = "bonferroni")
```

1.9 Non-parametric tests

Non-parametric techniques or tests encompass a series of statistical tests that have in common the absence of assumptions about distribution followed by the population from which the sample was drawn. Basically, when the assumptions of normality and homoscedasticity are not met, non-parametric tests are necessary for decision making.

Here some non-parametric tests for decision making will be presented.

a) Friedman

The Friedman test is a non-parametric statistical test developed by Milton Friedman [6-8] similar to ANOVA. It is used to detect differences in treatments in various test experiments. The procedure involves sorting each row (or block), then considering the column rank values.

Example 13: Below there is an R script to perform the Friedman test.

R Script

```
## Loading Packages

install.packages("dplyr")
library(dplyr)
install.packages("rstatix")
library(rstatix)
install.packages("reshape")
library(reshape)
install.packages("PMCMRplus")
library(PMCMRplus)
install.packages("ggplot2")
library(ggplot2)

# Load the database

# Important: select the working directory ( working directory )
# This can be done manually: Session > Set Working Directory > Choose Directory

data <- read.csv2('Database 7.csv', stringsAsFactors = T) # Loading csv file

View (data)
glimpse (data)

# Change database format from "wide" to "long" (package: reshape)
# Restructuring the database

datal <- melt ( data,
               id = "ID",
               measured = c( "Professor1", "Professor2", "Professor3", "Professor4"))

View ( datal )

# Renaming the columns of the new database

colnames( datal ) = c( "ID", "Teacher", "Grade")

# Ordering the columns by experimental subject

datal <- sort_df( datal , vars = "ID")

glimpse( datal )

# Transforming the ID variable into a factor
```

```

datal$ID <- factor( datal$ID )

# Carrying out the Friedman test

friedman.test( Grade ~ Teacher | ID, data = datal )

## Another option:
## friedman.test( datal$Grade , datal$Professor , datal$ID )

# Post-hoc testing

## Option 1: Wilcoxon with Bonferroni correction

datal %>% wilcox_test(Grade ~ Teacher, paired = TRUE, p.adjust.method = "bonferroni")

## Option 2 - post- hocs from the PMCMRplus package :
### Dunn- Bonferroni - equivalent to SPSS:
frdAllPairsSiegelTest(datal$Grade, datal$Teacher,
                      datal$ID, p.adjust.method = "bonferroni")

### Others:

frdAllPairsNemenyiTest(datal$Grade, datal$Teacher,
                      datal$ID, p.adjust.method = "bonferroni")

frdAllPairsConoverTest(datal$Grade, datal$Teacher,
                      datal$ID, p.adjust.method = "bonferroni")

# Descriptive data analysis

datal %>% group_by(Teacher) %>%
  get_summary_stats( Grade, type="median_iqr")

# Data visualization

boxplot( Grade ~ Teacher, data = datal )

# Distribution analysis

par(mfrow=c(2,2))
hist( datal$Grade [ datal$Teacher == "Professor1"],
      ylab = "Attendance", xlab = "Grades", main = "Teacher 1")
hist( datal$Grade [ datal$Teacher == "Professor2"],
      ylab = "Attendance", xlab = "Grades", main = "Teacher 2")
hist( datal$Grade [ datal$Teacher == "Professor3"],
      ylab = "Attendance", xlab = "Grades", main = "Teacher 3")
hist( datal$Grade [ datal$Teacher == "Professor4"],
      ylab = "Attendance", xlab = "Grades", main = "Teacher 4")

```

```
# Histogram with all groups, separated by color

ggplot( data1 , aes (x = Grade)) +
  geom_histogram( aes ( color = Teacher, fill = Teacher),
                 alpha = 0.3, position = "stack", binwidth = 1)
```

b) Wilcoxon

This non-parametric test developed by F. Wilcoxon in 1945 [9] is used as an alternative to the paired t-student test when samples do not follow a normal distribution. Thus, the Wilcoxon test is used to test whether the sample medians are equal in cases where the normality hypothesis is not accepted or when it is not possible to check this assumption. There are two important assumptions in this non-parametric test: i) the dependent variable must be originally numeric or categorical; ii) the independent variable must be composed of two dependent (paired) groups.

Example 14: Below there is an R script to check the Wilcoxon test on a given dataset.

R Script

```
# Load the packages that will be used

install.packages ("dplyr")
library (dplyr)
install.packages ("rstatix")
library (rstatix)

# Load the database

# Important: select the working directory ( working directory )
# This can be done manually: Session > Set Working Directory > Choose Directory

data <- read.csv('Database 4.csv', sep=";", dec=".", stringsAsFactors = T,
                fileEncoding = "latin1")

View (data)
glimpse (data)

# Wilcoxon test

wilcox.test( data$Seizures_PT , data$Seizures_S1, paired = TRUE)

# Observation:
# The two-tailed test is the default; If you want one-tailed, you need to include:
# alternative = "greater" or alternative = "less"
# Example: wilcox.test( data$Convulsoes_PT , data$Convulsoes_S1,
# paired = TRUE, alternative ="greater")
# In this case, the test will check whether the median of Seizures_PT is greater
than the median of Seizures_S1
```

c) Mann-Whitney test

In 1947, HB Mann and DR Whitney [10] generalized the technique developed by Wilcoxon (1945) to compare central tendencies of two independent samples of equal size. The Mann-Whitney test (Wilcoxon rank-sum test) is indicated for comparing two unpaired groups to verify whether or not they belong to the same population whose requirements for applying the Student's t test have not been met. Unlike the t-test, which tests the equality of means, the Mann-Whitney (U) test tests the equality of medians.

Example 15: Below there is an R script to perform the non-parametric Mann-Whitney test. Was this test helpful?

R Script

```
# Load the packages that will be used

install.packages("dplyr")
library(dplyr)
install.packages("rstatix")
library(rstatix)

# Load the database
# Important: select the working directory ( working directory )
# This can be done manually: Session > Set Working Directory > Choose Directory

data <- read.csv('Database 3.csv', sep = ';', dec = ',',
                stringsAsFactors = T, fileEncoding = "latin1") # Loading csv file

View(data) # View data in a separate window
glimpse(data) # View a summary of the data

# Performing the Mann-Whitney test

wilcox.test(Grade_Biology ~ Room_Position , data = data)
wilcox.test(Grade_Physics ~ Room_Position , data = data)
wilcox.test(Grade_History ~ Room_Position , data = data)

# Observation:
# The two-tailed test is the default; If you want one-tailed, you need to include:
# alternative = "greater" or alternative = "less"
# Example: wilcox.test( Grade_History ~ Room_Position , data = data, alternative =
"greater")
# In this case, the test checks whether the median of the first group is greater than
the median of the second
# OR is considering "Front" as the first group
```

```

# Step 4: Descriptive data analysis

data %>% group_by(Room_Position) %>%
  get_summary_stats(Grade_Biology, Grade_History, Grade_Physics, type = "median_
iqr")

# Parametric data

data %>% group_by(Room_Position) %>%
  get_summary_stats(Grade_Biology, Grade_History, Grade_Physics, type = "mean_sd")

# Distribution preview

par(mfrow=c(1,2))
hist(data$Grade_Biology[data$Room_Position == "Front"],
      ylab = "Frequency", xlab = "Grade", main = "Front Group")
hist(data$Grade_Biology[data$Room_Position == "Back"],
      ylab = "Frequency", xlab = "Grade", main = "Back Group")

```

d) Kruskal-Wallis test

The non-parametric Kruskal-Wallis test, named after William Kruskal and W. Allen Wallis [11] in 1952, is a method used to test whether samples originate from the same distribution. We typically use it to compare two or more independent samples of equal or different sizes. The parametric equivalent of this test is the F test used in one-way ANOVA.

Example 16: Below there is an R script to perform the Kruskal-Wallis non-parametric test. Comment on the results of this test.

R Script

```

# Load the packages that will be used

install.packages("dplyr")
library(dplyr)
install.packages ("rstatix")
library(rstatix)
install.packages("ggplot2")
library(ggplot2)

# Load the database

# Important: select the working directory ( working directory )
# This can be done manually: Session > Set Working Directory > Choose Directory

```

```

data <- read.csv( 'Database 5.csv', sep = ';', dec = ',',
                 stringsAsFactors = T, fileEncoding = "latin1") # Loading csv file

View (data)
glimpse (data)

# Kruskal-Wallis test

kruskal.test( BC ~ Group, data = data)
kruskal.test( Pressure ~ Group, data = data)

# Post-hoc testing
# Dunn's test with p-value adjustment

dunn_test( BC ~ Group, data = data, p.adjust.method = "bonferroni")
dunn_test( Pressure ~ Group, data = data, p.adjust.method = "bonferroni")

# Descriptive data analysis

data %>% group_by(Group) %>%
  get_summary_stats( BC, Pressure , type="median_iqr")

# Data visualization

par(mfrow=c(1,2))
boxplot(BC ~ Group, data = data)
boxplot(Pressure ~ Group, data = data)

# Distribution analysis

par(mfrow=c(1,3))
hist(data$BC[data$Group=="Placebo"],
     ylab="Frequency",xlab="bps",main="Placebo")
hist(data$BC[data$Group == "AH New"],
     ylab = "Frequency", xlab = "bps", main="AH New")
hist(data$BC[data$Group == "AH Default"],
     ylab = "Frequency", xlab = "bps", main="AH Default")

par(mfrow=c(1,3))
hist(data$Pressure[ data$Group == "Placebo"],
     ylab = "Frequency", xlab = "bps", main = "Placebo")
hist(data$Pressure[ data$Group == "AH New"],
     ylab = "Frequency", xlab = "bps", main = "AH New")
hist(data$Pressure[ data$Group == "AH Default"],
     ylab = "Frequency", xlab = "bps", main = "AH Default")

```



```
# Histogram with all groups, separated by color

ggplot(data, aes(x = BC)) +
  geom_histogram(aes(color = Group, fill = Group),
                alpha = 0.3, position = "identity", binwidth = 10)

ggplot(data, aes(x = Pressure)) +
  geom_histogram(aes(color = Group, fill = Group),
                alpha = 0.3, position = "dodge", binwidth = 10)
```

Below there is a table that summarizes the main statistical concepts discussed throughout Chapter 1 for comparison purposes.

Table 3: Summary of statistical concepts from Chapter 1

Independent Groups	Parametric	Confidence interval and limits (1 or 2 groups) Student's t (1 or 2 groups)
	Non-parametric	Mann-Whitney's U
Paired groups	Parametric	paired Student's t
	Non-parametric	Wilcoxon's test
≥ 3 independent groups	Parametric	1 or 2 factor ANOVA
	Non-parametric	Kruskall-Wallis
≥ 3 paired groups	Parametric	Anova for repeated measures
	Non-parametric	Friedman's test

PROPOSED EXERCISES

01 – Propose a chemical experiment that contains replicas and, based on the observations, present the statistical parameters described using an R script.

02 – There is a repository of univariate and multivariate data on the internet based on the R language (<https://archive.ics.uci.edu/>) in which you must choose one or more sets of data to carry out a descriptive statistical study in detail. Present your results.

03 – Propose an experiment or use a dataset from a public database (preferably) to apply the t-test for a single sample. Build an R script to demonstrate your hypotheses and main conclusions.

04 – Propose an experiment or use a dataset from a public database (preferably) to apply the t-test for independent samples. Build an R script to demonstrate your hypotheses and main conclusions.

05 – Propose an experiment or use a dataset from a public database (preferably) to apply the paired t-test. Build an R script to demonstrate your hypotheses and main conclusions.

06 – Propose an experiment or use a dataset from a public database (preferably) aiming to use 1-way ANOVA. In this exercise, use an R script so that you can test your hypotheses, make assumptions, post hoc evaluation, construct graphs and interpret results.

07 – Like the previous exercise, use an ANOVA of two or more factors for a given experiment or a dataset extracted from a public database (preferably). Show step-by-step in a R script for testing hypotheses, assumptions, post hoc, graphs and interpretation of results.

08 – During Chapter 1 of this book, some non-parametric tests were presented (Friedman, Wilcoxon, Mann-Whitney, Kruskal-Wallis). Each non-parametric test is a hypothesis test that does not require the population distribution to be characterized by certain parameters (μ and σ). Therefore, based on experiments or datasets found in a database (preferably public), present results of the four non-parametric tests mentioned and discuss their results.

REFERENCES

- [1] Shapiro, SS; Wilk, M.B. (1965). An analysis of variance test for normality (complete samples). *Biometrika* . 52 (3–4): 591–611.
- [2] Levene, Howard (1960). Robust tests for equality of variances. In: Ingram Olkin ; Harold Hotelling . *Contributions to Probability and Statistics: Essays in Honor of Harold Hotelling*. [SI]: Stanford University Press. pp. 278–292.
- [3] Neyman, J. (1937). Outline of a Theory of Statistical Estimation Based on the Classical Theory of Probability . *Philosophical Transactions of the Royal Society* . 236 : 333 – 380.
- [4] Pearson, ES (1939). Willian Sealy Gosset, "Student" as statistician. *Biometrika* , 30 (3-4): 210-250.
- [5] Tukey, John (1949). Comparing Individual Means in the Analysis of Variance. *Biometrics* . 5 (2):99–114.
- [6] Friedman, Milton (1937). The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *Journal of the American Statistical Association*. 32 (200): 675–701.
- [7] Friedman, Milton (1939). A correction: The use of ranks to avoid the assumption of normality implicit in the analysis of variance». *Journal of the American Statistical Association*. 34 (205): 109.
- [8] Friedman, Milton (1940). A comparison of alternative tests of significance for the problem of m rankings. *The Annals of Mathematical Statistics*. 11 (1): 86-92.
- [9] Wilcoxon, Frank (1945). Individual comparisons by ranking methods . *Biometrics Bulletin* . 1 (6): 80-83
- [10] Mann, HB and Whitney, DR (1947) On a Test of Whether One of Two Random Variables Is Stochastically Larger than the Other. *Annals of Mathematical Statistics*, 18, 50-60.
- [11] Kruskal, William H.; Wallis, W. Allen (1952). Use of Ranks in One-Criterion Variance Analysis . *Journal of the American Statistical Association* . 47 (260): 583–621.

DESIGN OF EXPERIMENTS

"To consult the statistician after an experiment is finished is often merely to ask him to conduct a postmortem examination. He can perhaps say what the experiment died of." **Ronald Fisher (1890-1962)**

CHAPTER IDEA

Design of Experiments (DOE) or experimental design is an ideal technique for studying the effect of a set of several factors on a response variable of interest. The main objectives of experimental planning are to reduce process time, reduce the number of tests without compromising the quality of information, reduce operational costs and increase the quality and yield of processes.

In this chapter, after analyzing the data presented in the examples, some practical conclusions from the experiment will be detailed through graphs to present the results and some confirmation tests. Upon completing the chapter, you should be able to:

- a) Understand the main terms (response variable, levels, factors or treatments, randomization and blocks) used in planning;
- b) Define the stages of experimental design (recognition and identification of the problem, goals, choice of response variable, choice of experimental design types, and others);
- c) Carry out graphic analysis for a more assertive interpretation of experiments;
- d) Represent the studied process through mathematical expressions;
- e) Carry out a selection of variables that influence the process with a reduced number of tests;
- f) Build new scripts in R language for decision making using experiment design;
- g) Understand the importance of knowing in depth the problem (system or process) you want to study.

2.1 Fundamental steps for planning experiments

Experiments can be considered a series of tests in which intentional changes are made to the input variable of a process or system so that we can analyze and identify the reasons for changes that can be observed in the response variable. When designing and carrying out experiments, one of the main goals is to ensure that the process is minimally affected by external sources of variability.

The fundamental steps of experimental design are:

- a) Recognition of the problem;
- b) Determining the goals;
- c) Choosing the response variable, factors and levels;
- d) Choosing the type of experimental design;
- e) Formulation of a hypothesis;
- f) Analysis of results;
- g) Preparation of conclusions and recommendations.

Basically, there are three techniques used to define tests in experimental planning: i) use of replicates; ii) randomization; and, iii) blocking.

The replication consists of repeating a test under pre-established conditions, obtaining an estimate of how the experimental error may affect the results of the experiments and whether these results are statistically different. Randomization is a purely statistical experimental design technique in which the sequence of tests is random, and the choice of materials that will be used in these tests is also random. Experimental blocking is a statistical technique that consists of organizing experimental units into groups (blocks) that are similar to each other. The purpose of blocking is to contain and evaluate the variability produced by the disturbing factors of the experiment. Blocking allows you to create a more homogeneous experiment and increase the precision of the responses that are analyzed. The central idea of blocking is to make the experimental units (EUs) homogeneous within the blocks.

Now, some experimental designs will be presented (full factorial, fractional factorial, Box-Behnken, mixtures, multi-levels) through examples written in the R language.

2.2 Full Factorial Design

Factorial design makes it possible to simultaneously evaluate the effect of a set of factors (each variable of the system under study) from a reduced number of experiments (trials), when compared to univariate processes. This design is very interesting when you want to study the effects (change in the response when moving from a low level (-) to a high level (+)) of two or more influencing variables, and in each attempt or replication, all the possible combinations of levels of each variable are studied.

We can present some advantages of using factorial designs: i) fewer tests without loss of information quality; ii) simultaneous determination of several factors, separating their effects; iii) selection of factors that influence a process with a reduced number of tests; and, iv) description of the experimental procedure through a mathematical model.

Factorial design is represented by b^k , in which k represents the number of factors and b the number of levels chosen. In general, if there are n_1 levels of factor 1, n_2 of factor

2, ..., and n_k of factor k , the planning will be a factorial $n_1 \times n_2 \times \dots \times n_k$. Here we can state that it does not necessarily mean that only $n_1 \times n_2 \times \dots \times n_k$ experiments will be carried out. This is the minimum number to have a complete factorial design. Factorial planning is indicated for starting the experimental procedure when there is a need to identify the influencing variables and study their effects on the chosen response variable.

2.3 Factorial Design 2^2

Factorial design with k factors and 2 levels is a particular case, called 2^k factorial design. The factors and levels are pre-determined, establishing this design as a fixed effects model. To ensure objectiveness in the analysis, the hypotheses of normality and homoscedasticity must be satisfied. This type of design is normally used in the initial stages of research, allowing the study of several factors with a reduced number of experiments.

Example 1: In this example, we will describe a step-by-step guide for creating a complete factorial design (2 factors and 2 levels), t-test, ANOVA and regression.

R Script

```
# levels
levels = c(-1,1)

# planning
plan = expand.grid(levels, levels)

#replicating the planning
plan = rbind(plan, plan)
plan

# column names
colnames(plan) = c("x1", "x2")

# Response
y = c(5.1, 8.6, 7.7, 13.5, 5.0, 8.5, 8.2, 13.9)

# adding response to planning
plan$y = y
plan

#####
# Step by step analysis
# Planning matrix
x = model.matrix(~x1*x2, data = plan [, -3])
x
# Effects
effects = crossprod(x,y)/(2*2^2/2)
effects
```

```

# Coefficients
coef = effects/2
coef

# adjusted values
fitted = x%% coef
fitted

# waste
resi = y - fitted
resi

# number of trials (N) and terms in the model (r)
N = dim(x)[1]
r = dim(x)[2]

## Sum of squares
# Waste SS
SSE = sum(resi^2)
# Total SS
SST = sum(y^2)-sum(y)^2/N

## Degrees of freedom
# of errors
DFE = N-r

# total
DFT = N-1

## Mean of squares
# of errors
MSE = SSE/DFE
# total
MST = SST/DFT

#t calculated
t0 = coef / sqrt (MSE/N)
t0

#t critical
t_critical = qt( 0.025, df = DFE, lower.tail = F )
t_critical

#p-value
pvalue = 2*pt(abs(t0), df = DFE, lower.tail = F)
pvalue

# data frame t-test summary

```

```

ttest = data.frame ( coef , rep(sqrt(MSE/N),4),t0,pvalue)
colnames ( ttest ) = c( "coef", "SE_coef", "p- value")

# coefficient of multiple determination
R2 = 1 - SSE/SST
R2_aj = 1 - MSE/MST

## The new
# Sum of squares of effects

SS_x = crossprod(x [,-1],y)^2/N

# mean of squares of effects
MS_x = SS_x /1

# F calculated
F0 = MS_x /MSE
F0

# pvalue
p = pf(F0, 1, DFE, lower.tail = F)
p

# ANOVA summary table

source = c("x1", "x2", "x1x2", "Error", "Total")
SS = c(SS_x , SSE, SST)
DF = c(rep( 1,3), DFE, DFT)
MS = c(MS_x , MSE, MST)
F0 = c(F0, NA, NA)
pvalue = c(p, NA, NA)

ANOVA = data.frame (SS, DF, MS, F0, pvalue )
rownames (ANOVA) = source
ANOVA

#####
# Matrix least squares

# Multiplying X transposed by
t(x)%*%x

#inverse of previous result
solve(t(x)%*%x)

#multiplying X transposed by y
t(x)%*%y

```

```

#getting the coefficients
beta_mat = solve(t(x)%%x)%%t(x)%%y
beta_mat

# prediction for all experimental results
y_hat = x %% beta_mat
y_hat

#prediction for specific values of x1 and x2
#x1 = 0, x2 = 0.5
x_esp = matrix(c( 1,0,0.5,0), nrow = 4, ncol = 1)
y_esp_hat = t(x_esp)%%beta_mat
y_esp_hat

```

2.4 Factorial Design 2³

A factorial design with three factors (A, B and C), each with two levels (high and low), is called a complete factorial design 2³, that is, we will have 8 trials that combine all the factors and their levels, whose matrix is presented in Table 1.

Table 1: Complete Factorial Design Matrix 2³

Test	Factor 1	Factor 2	Factor 3
1	-	-	-
2	+	-	-
3	-	+	-
4	+	+	-
5	-	-	+
6	+	-	+
7	-	+	+
8	+	+	+

Example 2: In this example we will describe a step-by-step guide for creating a complete factorial design (3 factors and 2 levels), ANOVA test, assumption test, construction of the Pareto chart and regression analysis. The goal of this experiment is to define the optimal conditions for determining the copper content in water samples using the ICP-OES technique [1]. In this design, we will use the FrF2 package of R software.

R Script

```
### Factors
# x1 = pH
# x2 = flow rate, vz ( mL /min)
# x3 = eluent concentration, cE (mol/L)

### Response
# y = normalized instrumental peak height

##### Planning

install.packages("FrF2")
library(FrF2)

# planning
plan = FrF2( nruns = 16,
            nfactors = 3 ,
            factor.names = list(pH = c(4,8) , # -
                                vz = c( 2.8), # mL /min
                                CE = c( 0.7,2)), # mol/L
            randomize = F)
# experiment carried out we use F, otherwise T
summary( plan )

# response - relative analytical signal
y = c(13.66, 23.60, 39.13, 95.65, 17.39, 22.36, 35.40, 100.00,
      13.04, 23.60, 35.40, 91.93, 13.66, 25.47, 33.54, 95.03)

# adding response
# add.response command from FrF2 package
plan = add.response(plan, y)
summary( plan )

##### Analysis #####
lm1 = lm( plan ) # model with 2nd order interactions
summary(lm1)

anova1 = aov(lm1 ) # anova with 2nd order interactions
summary(anova1)

lm2 = lm( y ~ pH * vz * CE, data = plan ) #complete model
summary(lm2)

anova2 = aov(lm2)
summary(lm2)

# model comparison
anova(lm1, lm2) #best complete lm2
```

```

# confidence interval for coefficients
confint(lm2)

#### Assumptions
#normality
shapiro.test(lm2$residuals)

par(mfrow = c(2,2))
plot(lm2)
par(mfrow = c( 1,1))

# homoscedasticity

install.packages("olsrr")
library(olsrr) # another package option for Breuch-Pagan test
ols_test_breusch_pagan(lm2, rhs = T, multiple = T)

##### Graphics
# via FrF2

MEPlot(lm2)

IAPlot(lm2)

cubePlot(lm2,eff1 = "pH", eff2 = "vz", eff3 = "CE", main = "")

install.packages("ggpubr")
library(ggpubr)

# effects graph
p1 = ggline( data = plan ,
             x = "pH", y = "y",
             add = c("mean_se","jitter"),
             color = "blue") + theme_bw ( )
p1

p2 = ggline( data = plan,
             x = "vz", y = "y",
             add = c("mean_se","jitter"),
             color = "green") + theme_bw ( )
p2

p3 = ggline( data = plan,
             x = "CE", y = "y",
             add = c("mean_se","jitter"),
             color = "red") + theme_bw ( )

```

```

p3

p12 = ggline( data = plan,
             x = "pH", y = "y",
             add = c("mean_se", "jitter"),
             color = "black") + theme_bw ( )

p12

ggarrange(p1,p2,p12)

## Pareto chart of standardized effects
t_critical = qt(0.025, df.residual (lm2), lower.tail = F) # t-critical

MSE = deviance(lm2)/ df.residual (lm2)
SE_coef = sqrt(MSE/16) # standard error of coefficients
t0 = lm2$coefficients/ SE_coef # t0

#data frame for t0
t_0 = data.frame(names(coef(lm2)), abs (t0))
colnames(t_0) = c("term", "t0")

# Pareto Chart - standardized effect
pPar = ggbarplot(data = t_0[-1,],
                x = "term", y = "t0",
                col = "green4",
                fill = "lightgreen",
                rotate = T,
                sort.val = "asc") + theme_bw( ) +
  geom_hline(yintercept = t_critical, col = "red")
pPar

ggarrange(p1,p2,p12,pPar)

# Contour plots

library(ggplot2)

# planning via expand.grid
plan2 = expand.grid(c(4,8),
                  c(2,8),
                  c(0.7,2))

plan2

plan2 = rbind(plan2,plan2)
colnames(plan2) = c("pH", "vz", "CE")
plan2$y = y

```

```

# model decoded via expand.grid
lm3 = lm(y ~ pH * vz * CE, plan2)
lm3 # never test significance of decoded coefficients

# mesh
grid = expand.grid(pH = seq(4,8, length = 40),
                  vz = seq( 2,8, length = 40),
                  CE = seq( 0.7,2, length = 40))

# prediction on the mesh
y_hat = predict(lm3, newdata = grid)
grid$y = y_hat

# contour plots

cp1 = ggplot(data = grid,
             mapping = aes(x = pH, y = vz, z = y)) +
  geom_line() +
  scale_fill_distiller(palette = "Spectral",
                      direction = -1) +
  geom_contour(color = "gray50") + theme_bw()
cp1

cp2 = ggplot(data = grid,
             mapping = aes(x = pH, y = CE, z = y)) +
  geom_line() +
  scale_fill_distiller(palette = "Spectral",
                      direction = -1) +
  geom_contour(color = "gray50") + theme_bw()
cp2

cp3 = ggplot(data = grid,
             mapping = aes(x = CE, y = vz, z = y)) +
  geom_line() +
  scale_fill_distiller(palette = "Spectral",
                      direction = -1) +
  geom_contour(color = "gray50") + theme_bw()
cp3

ggarrange(cp1,cp2,cp3)

##### Prediction

# Using model made via expand.grid - lm3
predict(lm3, newdata = data.frame (pH = 6,
                                  vz = 5,
                                  CE = 2))

lm3$fitted.values

```

2.5 Unreplicated Factorial Design

Example 3: This example was taken from the study [2] in which the influence of four factors on flexural strength (response) was investigated using an unreplicated factorial design. In this design, we will use the unrepx package of R software.

R Script

```
## Factors
# dc = density of the compound
# vf = fiber fraction volume
# tp = pyrolysis temperature
# ta = softening temperature of the synthesized methyl-polycarboxylane

## response
# FS = flexural strength
# https://doi.org/10.1007/s12034-017-1535-5

#####
# planning

library(FrF2)

design = FrF2(nruns = 16,
             nfactors = 4,
             factor.names = c("dc", "vf", "tp", "ta"),
             randomize = F) # generate the planning in the default order
summary(design)

# answer - flexural strength (MPa)
FS = c(425,495,450,571,374,441,409,468,
       399,525,461,592,401,489,393,487)

# adding response to planning
design$y = FS
summary(design)

#####
# Calculating the effects

X = model.matrix(~dc*vf*tp*ta, data = design[, -5])
X

N = dim(X)[1]

effects = crossprod(X,FS)/(N/2)
effects
```

```
#####
### Working with the unrepx package for unreplicated factorial
# Package authored by Professor Russell V. Lenth

install.packages("unrepx")
library(unrepx)

# Effects
effects2 = yates(FS) # responses must be in standard order
effects2

attr(effects2,"mean")

# Half normal plot
hnplot(effects2, half = TRUE, method = "Lenth", ID = ME(effects2))

# Pareto PSE plot
parplot(effects2, method = "Lenth")

# Significance analysis of effects
# t-test via Lenth 's pseudo standard error
eff.test(effects2, method = "Lenth")
```

2.6 Fractional Factorial Design

Assuming the exponential character existing in factorial design (2^k), we must consider the possibility of having full designs that are unfeasible, especially in cases of a great number of factors ($k > 4$). If we had a complete factorial design of 8 factors, we would have a composition of 256 trials ($2^8 = 256$). A more interesting alternative would be to employ a selection of the most significant factors or to carry out a fractional factorial design that uses a much smaller number of tests.

We can assume that the fractional factorial designs are determined in the form 2^{k-p} , in which k represents the number of factors and p represents the fraction of the complete scheme 2^k . For example, if we sought to halve the number of trials in a 2^4 design with 16 trials (4 factors and 2 levels), we would have $\frac{1}{2} \times 2^4 = 2^{-1} \times 2^4 = 2^{4-1} = 2^3 = 8$ trials. Table 2 presents a matrix of a fractional design 2^{4-1} (4 factors with two levels), in which the product of the column levels of factors 1, 2 and 3 corresponds to the column level of factor 4. Note that there is interaction between factors 1, 2 and 3 confused with factor 4, assuming the impossibility of distinguishing the interaction between the three factors and the fourth factor.

Table 2: Fractional Planning Matrix 2^{4-1}

Test	Factor 1	Factor 2	Factor 3	Factor 4
1	-	-	-	-
2	+	-	-	+
3	-	+	-	+
4	+	+	-	-
5	-	-	+	+
6	+	-	+	-
7	-	+	+	-
8	+	+	+	+

Example 4: This example was taken from the study [3] in which the adsorption efficiency of thiomethoxam was investigated using oxidized multi-walled carbon nanotubes (MwCNTs) as adsorbents through a fractional factorial design (2_p^{5-1}). In this design, we will use the FrF2 and unrep packages of R software.

R Script

```
## via FrF2

library(FrF2)

# design - 2^(5-1)

frac = FrF2(nruns = 16,
            nfactors = 5,
            factor.names = list(x1 = c(50,150), # initial concentration (mg/L)
                               x2 = c(25,45), # temperature (oC)
                               x3 = c(5,9), # pH
                               x4 = c(50,150), # adsorbent mass (mg)
                               x5 = c(1,3)), # time (h)
            randomize = F,
            alias.info = 3)
summary(frac)

# answer - quantity adsorbed
qty = c(24.15, 31.5, 33.62, 63.48, 19.64, 94.08, 20.34, 57.38,
        28.12, 46.99, 24.62, 44.18, 19.29, 50.83, 19.02, 61.22)
frac$y = qty

# Coded design - 2^( 5-1) main fraction
frac_p = FrF2(nruns = 16,
```

```

        nfactors = 5,
        factor.names = c("x1", "x2", "x3", "x4", "x5"),
        randomize = F,
        alias.info = 3) #3rd order information
summary(frac_p)

# Coded design - 2^( 5-1) alternative fraction
frac_alt = FrF2(nruns = 16,
               nfactors = 5,
               factor.names = c ("x1", "x2", "x3", "x4", "x5"),
               generators = "-ABCD", ### planning generator
               randomize = F,
               alias.info = 3) #3rd order information
summary(frac_alt )

# complete model
lm1 = lm(y ~ .^5, data = frac)
summary(lm1)

# model 1 confounding structure
aliases(lm1)

# 2nd order model to avoid confusion (interactions from 3rd to 5th order removed)
lm2 = lm(y ~ .^2, data = frac)
summary(lm2)

# graphics
MEPlot(lm2)
IAPlot(lm2)

# confusion lm2
aliases(lm2)

# model without interaction x2*x4
lm3 = lm(y ~ x1+x2+x3+x4+x5 + x1*x2 + x1*x3 + x1*x4 + x1*x5
        + x2*x3 + x2*x5 + x3*x4 + x3*x5 + x4*x5, data = frac)
summary(lm3)

# Pareto chart of standardized effects
library(ggpubr)

#t critical
t_critical = qt(0.025, df.residual(lm3), lower.tail = F) # t critical

# t calculated
MSE = deviance(lm3)/df.residual(lm3) # MSE, obs = deviance = sum(lm3$residuals^2)
SE_coef = sqrt(MSE/16) #standard error of coefficients
t0 = lm3$coefficients/SE_coef # t0

```



```

# data frame for t0
t_0 = data.frame(names(coef(lm3)),abs(t0))
colnames(t_0) = c("term","t0")

# Pareto chart - standardized effects
pPar = ggbarplot(data=t_0[-1,],
                 x = "term", y = "t0",
                 col = "green4",
                 fill = "lightgreen",
                 rotate = T,
                 sort.val = "asc") + theme_bw()+
  geom_hline(yintercept = t_critical , col = "red")
pPar

#####
## Via Lenth's pseudo standard error

library(unrepx)

#design matrix
X = model.matrix(~x1*x2*x3*x4*x5, data = frac[,-6])
X

# effects
effects = crossprod(X,qty)/(16/2) # here there are confusing effects
effects = effects[2:16] # only main effects and 2nd order interactions
names(effects) = c("x1", "x2", "x3", "x4", "x5", "x1x2", "x1x3", "x1x4", "x1x5",
                  "x2x3", "x2x4", "x2x5", "x3x4", "x3x5", "x4x5")

effects

#graphics
hnplot(effects, method = "Lenth", ID = ME(effects))
parplot(effects, method = "Lenth")

```

Example 5: The following example, also using fractional factorial design (2_v^{5-1}), is taken from study [4] in the development of HSLA (high-strength low-alloy) steel wire electrodeposition. In this design, we will use the FrF2 and unrep packages of R software.

R Script

```
### via FrF2

library(FrF2)

# design - 2^(5-1)

frac2 = FrF2(nruns = 16,
             nfactors = 5,
             factor.names = list(x1 = c(32,96), # pulsed time (us)
                                 x2 = c(6,9), # pulse ratio (t_off/t_on)
                                 x3 = c(3,5), # power (mu)
                                 x4 = c(30,60), # wire frequency (HZ)
                                 x5 = c(50,150)), # pulse (mu)

             randomize = F,
             alias.info = 3) # 3rd order information

summary(frac2)

# answer - average roughness
Ra = c(4.086, 6.847, 4.314, 6.185, 4.684, 5.469, 4.467, 6.52,
       4.804, 6.908, 5.073, 7.335, 5.907, 7.66, 5.688, 7.764)

frac2$y = Ra

# complete model
lm1 = lm(y ~ .^5, data = frac2)
summary(lm1)

# model 1 confounding structure
aliases(lm1)

# 2nd order model to avoid confusion (interactions from 3rd to 5th order removed)
lm2 = lm(y ~ .^2, data = frac2)
summary(lm2)

# Eliminating x2 and projecting the planning in a 2^4 full factorial
lm3 = lm(y ~ (x1+x3+x4+x5)^4, data = frac2)
summary(lm3)

# reducing previous model
lm4 = lm(y ~ (x1+x3+x4+x5)^3, data = frac2)
summary(lm4)
```

```

lm5 = step(lm4, direction = "backward")
summary(lm5)

#####
## Via Lenth 's pseudo standard error (Hypothesis test)

library(unrepX)

# design matrix
X = model.matrix(~x1*x2*x3*x4*x5, data = frac2[, -6])
X

# effects
effects = crossprod(X, Ra)/(16/2) # here there are confusing effects
effects = effects[2:16] # only main effects and 2nd order interactions
names(effects) = c("x1", "x2", "x3", "x4", "x5", "x1x2", "x1x3", "x1x4", "x1x5",
                  "x2x3", "x2x4", "x2x5", "x3x4", "x3x5", "x4x5")

effects

#graphics
hnpplot(effects, method = "Lenth", ID = ME(effects))
parplot(effects, method = "Lenth")

```

2.7 2^k Factorial Design with center-point

Central composite design is a type of experimental design that allows obtaining more detailed information about a system, with the adjustment of a useful second-order model to find the ideal conditions of the relevant factors, commonly for optimization purposes. Such design consists of three sequential parts, the first refers to the complete or fractional factorial design with factors (k) coded at two levels (+ and -) and with 2^k tests, another axial part consisting of points in all null coordinates and in coordinate α ($+\alpha$ or $-\alpha$) with $2k$ tests, and the last part is composed of n_c tests carried out at the central point with values equal to zero (minimum of 3 tests). Table 3 presents the matrix of this type of design for the case of three factors.

Table 3: Central composite design matrix for 3 factors

Test	Factor 1	Factor 2	Factor 3
1	-	-	-
2	+	-	-
3	-	+	-
4	+	+	-
5	-	-	+
6	+	-	+
7	-	+	+
8	+	+	+
9	$-\alpha$	0	0
10	α	0	0
11	0	$-\alpha$	0
12	0	α	0
13	0	0	$-\alpha$
14	0	0	α
15	0	0	0
16	0	0	0
17	0	0	0

The value of α is a new parameter that must be declared and has a value located between 1 and \sqrt{k} , where k represents the number of factors investigated. The value of α is based on the distance of the axial points in relation to the central points and is related to the shape and size of the central composite design domain, being spherical, when $\alpha=\sqrt{k}$, or cubic, when $\alpha=1$, with impact in the design rotation.

As in other experimental designs, in this design the values of coefficients used in a model are extracted using regression, calculation of coefficients significance (ANOVA), hypothesis tests or other statistical operations to extract information from the design. Finally, a mathematical model generates a response surface with the aim of optimizing the system and obtaining a graphical visualization of the optimum point.

Example 6: The following example consists of the optimization of a factorial design with a central point for determining the copper content in different water samples by ICP-OES, extracted from the study [1]. In this design, we will use the FrF2 and rsm packages of R software.

R Script

```
# Factors
# x1: pH
# x2: flow rate - vz(mL/min)

# Response
# y: relative analytical signal (resulting from normalized instrumental peak
measurements).

##
# via FrF2

library(FrF2)
plan.ctpt = FrF2(nruns = 4,
                nfactors = 2,
                ncenter = 3,
                factor.names = c("x1", "x2"),
                randomize = F)

summary(plan.ctpt)

# response - relative analytical signal
y = c(68.64, 69.82, 81.66, 85.80, 100, 99.41, 100)

plan.ctpt$y = y
summary(plan.ctpt)

# the term to evaluate liscube curvature(plan.ctpt)
lm1 = lm(y ~ x1*x2 + !liscube(plan.ctpt), data = plan.ctpt)
summary(lm1)

# ANOVA
summary(aov(lm1))

#####
## Performing design with a central point using the rsm package

install.packages("rsm")
library(rsm)
```

```

plan.ctpt2 = cube(basis = ~x1+x2,
                 n0 = 3,
                 randomize = F)

plan.ctpt2$y = y
plan.ctpt2

lm2 = lm(y ~ S0(x1,x2), data = plan.ctpt2)
summary(lm2)

#####
# Graphics for main effects and interaction

library(ggpubr)

p1 <- ggline(plan.ctpt ,
             x = "x1",
             y = "y",
             add = c("mean"),
             color = "blue") + theme_bw()

p2 <- ggline(plan.ctpt,
             x = "x2",
             y = "y",
             add = c("mean"),
             color = "green") + theme_bw()

plan.plot = plan.ctpt
plan.plot$x1 = as.factor(plan.ctpt$x1)
plan.plot$x2 = as.factor(plan.ctpt$x2)

p12 <- ggline(plan.plot,
             x = "x1",
             y = "y",
             add = c("mean", "point"),
             color = "x2") + theme_bw()

ggarrange(p1,p2,p12)

```

2.8 Box-Behnken Design

This type of design was proposed by Box and Behnken in 1960 and is characterized by second order designing to generate a response surface. Like the central composite design, this design has the disadvantages of using only three factor levels and always results in a greater number of tests than some other designs. The Box-Behnken design combines a $2k$ factorial structure with an incomplete block design, as shown in Table 4. The

result is very economical and efficient as it generates a small number of tests; in addition, it has rotational properties and constitutes an alternative to the central composite design.

Table 4: Box-Behnken construction scheme

Blocks	Factor 1	Factor 2	Factor 3
1	β	β	x
2	β	x	β
3	x	β	β

The construction of the design matrix can be exemplified by the scheme in Table 4, which considers of three blocks in the composition of the tests for three factors studied. Each β symbol, in each of the blocks, is replaced by the two-level encoded column of the corresponding factor, extracted from the matrix of a 2^2 design; and, the x is filled with a column of zeros. The procedure is repeated for each block, considering the factors that participate in the block and, at the end, at least three tests are added at the central point, resulting in 15 tests, as shown in Table 5.

Table 5: Box-Behnken Design's Example

Test	Factor 1	Factor 2	Factor 3
1	-	-	0
2	-	+	0
3	+	-	0
4	+	+	0
5	-	0	-
6	-	0	+
7	+	0	-
8	+	0	+
9	0	-	-
10	0	-	+
11	0	+	-
12	0	+	+
13	0	0	0
14	0	0	0
15	0	0	0

Example 7: The following example consists of optimizing a Box-Behnken design with four factors (F1 – layer thickness, F2 – heater energy, F3 – heater advance speed and F4 – printer advance speed) used in selective inhibition sintering of high-density polyethylene parts, extracted from the study [5]. In this design, we will use the rsm package of R software and a restricted designing optimization.

R Script

```
#####
install.packages("rsm")
library(rsm)

design = bbd(k = ~x1+x2+x3+x4,
            block = F,
            n0 = 5,
            randomize = F,
            coding = list(x1 ~ (Ac - 0.2)/0.1,
                          x2 ~ (Ea - 25.32)/3.16,
                          x3 ~ (vf_a - 3.5)/0.5,
                          x4 ~ (vf_p - 100)/20))

design

# width deviation

width = c(5.3533, 5.2615, 5.0008, 4.2712, 4.5840, 2.7470, 3.8086, 3.9839, 4.3630,
          3.5519, 4.0534,
          4.0031, 5.1495, 4.5581, 4.1959, 3.5946, 5.1642, 4.0103, 3.6354, 4.2529,
          3.5171, 4.4485,
          5.3879, 3.4132, 3.8905, 4.3263, 4.2263, 3.9451, 3.9024)

design$y = width

#####
# Analysis
rsm.bbd = rsm(y ~ SO(x1,x2,x3,x4), data = design)
summary(rsm.bbd)

#####
# Normality
shapiro.test(rsm.bbd$residuals)

#####
# Graphics

# Contour plots
par(mfrow = c(2,3))
contour(rsm.bbd , ~x1 + x2, image = TRUE)
contour(rsm.bbd , ~x1 + x3, image = TRUE)
contour(rsm.bbd , ~x1 + x4, image = TRUE)
contour(rsm.bbd , ~x2 + x3, image = TRUE)
contour(rsm.bbd , ~x2 + x4, image = TRUE)
contour(rsm.bbd , ~x3 + x4, image = TRUE)

# Perspective plots
```



```

persp(rsm.bbd , ~x1 + x2, zlab = "y [%]", col = rainbow(50), contours = "colors")
persp(rsm.bbd , ~x1 + x3, zlab = "y [%]", col = rainbow(50), contours = "colors")
persp(rsm.bbd , ~x1 + x4, zlab = "y [%]", col = rainbow(50), contours = "colors")
persp(rsm.bbd , ~x2 + x3, zlab = "y [%]", col = rainbow(50), contours = "colors")
persp(rsm.bbd , ~x2 + x4, zlab = "y [%]", col = rainbow(50), contours = "colors")
persp(rsm.bbd , ~x3 + x4, zlab = "y [%]", col = rainbow(50), contours = "colors")

#####

# Optimization restricted
optimal = steepest(rsm.bbd , dist = seq(0, sqrt(2), by = .1), descent = T)

x_ = c(optimal$x1[nrow (optimal)], optimal$x2[ nrow (optimal)], optimal$x3[ nrow
(optimal)], optimal$x4[nrow(optimal)])
names(x_) = c("x1", "x2", "x3", "x4")

par(mfrow = c(2,3))
contour(rsm.bbd , ~x1+x2, col = "black", decode = F, at = x_)
points(x2 ~ x1, data = optimal , col = "blue", pch = "*")

contour(rsm.bbd , ~x1+x3, col = "black", decode = F, at = x_)
points(x3 ~ x1, data = optimal , col = "blue", pch = "*")

contour(rsm.bbd , ~x1+x4, col = "black", decode = F, at = x_)
points(x4 ~ x1, data = optimal , col = "blue", pch = "*")

contour(rsm.bbd , ~x2+x3, col = "black", decode = F, at = x_)
points(x3 ~ x2, data = optimal , col = "blue", pch = "*")

contour(rsm.bbd , ~x2+x4, col = "black", decode = F, at = x_)
points(x4 ~ x2, data = optimal , col = "blue", pch = "*")

contour(rsm.bbd , ~x3+x4, col = "black", decode = F, at = x_)
points(x4 ~ x3, data = optimal , col = "blue", pch = "*")

```

2.9 Multi-level Factorial Design

Multi-level factorial design, also known as generalized factorial design, is a method used when the control factors are qualitative. In this method, each factor can have several distinct levels. For example, Factor A can have two levels, Factor B can have three levels, and Factor C can have five levels. Experimental trials include all combinations of these factor levels.

The number of experiments in this design is the number of replicates times the number of levels of each factor. For example, if one factor has four levels, another has three and another has two, the number of combinations is $4 \times 3 \times 2 = 24$. The simplest case of factorial design is one in which each factor is present in only two levels. In this case, for an experiment with k factors and two levels, $2 \times 2 \times \dots \times 2$ (k times) = 2^k observations of the response variable are carried out.

Example 8: The following example consists of the use of a multi-level factorial design applied to turning ABNT 1045 steel, extracted from study [6]. In this design, we will use the DoE.base package and phia of R software and a restricted designing optimization.

R Script

```

library(DoE.base)

# Planning decoded via DoE .base package
design = fac.design(factor.names = list(CB = c("PF", "PM", "QM", "KR"),
                                       f = c(0.16, 0.24, 0.32),
                                       vc = c( 310,380)),
                  replications = 3,
                  randomize = FALSE)

design

# Planning encoded

design2 =fac.design(factor.names = list(CB = c(-1, -0.33, 0.33, 1),
                                       f = c(-1,0,1),
                                       vc = c(-1,1)),
                  replications = 3,
                  randomize = FALSE)

design2

# Cutting force - Fc

Fc <- c(902.30, 877.31, 845.72, 991.03, 1287.40, 1198.23, 1166.36,
        1399.52, 1724.08, 1544.10, 1510.12, 1712.83, 870.88, 888.42,
        857.49, 955.14, 1280.49, 1200.65, 1161.42, 1342.05, 1674.23,
        1522.24, 1508.66, 1687.77, 912.67, 882.97, 835.37, 974.67,
        1309.27, 1205.55, 1194.54, 1370.80, 1721.67, 1528.56, 1545.90,
        1727.58, 894.77, 880.27, 846.80, 959.81, 1280.55, 1227.36,
        1169.19, 1330.01, 1676.26, 1501.24, 1509.74, 1650.20, 916.21,
        880.05, 850.29, 1008.54, 1311.10, 1209.18, 1200.13, 1356.21,
        1697.23, 1566.89, 1525.19, 1683.15, 885.12, 875.53, 838.02,
        979.96, 1286.59, 1193.36, 1164.99, 1350.14, 1686.84, 1503.87,
        1536.91, 1690.23)

design$Fc = Fc
design2$Fc = Fc

##### #
## Analysis####

# ANOVA for Fc
res.Fc = aov(Fc ~ CB*f* vc , data = design)
summary(res.Fc )

```

```

lm.Fc = lm(Fc ~ CB*f* vc , data = design2)
summary(lm.Fc )

##### Assumptions

# Normality
shapiro.test(res.Fc $residuals )

par(mfrow = c(2,2))
plot(res.Fc )

# Homoscedasticity

library(car)
leveneTest(Fc ~ CB*f* vc , data = design)

# multiple comparisons test

library(emmeans)
Tukey.Fc = emmeans(res.Fc, # Tukey
                  ~ CB|f)
plot(Tukey.Fc)

install.packages("ScottKnott")
library(ScottKnott)

sk1 <- with(design,
            SK(x = res.Fc, # Scott-Knott
              y = Fc,
              model = 'Fc ~ CB*f',
              which = 'f:CB',
              fl1 = 1))

sk2 = with(design,
           SK(x = res.Fc, # Scott-Knott
             y = Fc,
             model = 'Fc ~ CB*f',
             which = 'f:CB',
             fl1 = 2))

sk3 = with(design,
           SK(x = res.Fc, # Scott-Knott
             y = Fc,
             model = 'Fc ~ CB*f',
             which = 'f:CB',
             fl1 = 3))

```

```

summary(sk1) # equal characters, equal statistical averages
summary(sk2)
summary(sk3)

par(mfrow=c(1,3))
plot(sk1)
plot(sk2)
plot(sk3)

##### Effects graph

install.packages("phia")
library(phia)
IM = interactionMeans(res.Fc)
IM
plot(IM)

library(ggpubr)

p1 <- ggline(design,
             x = "CB",
             y = "Fc",
             add = c("mean", "jitter"),
             color = "blue") + theme_bw()

p2 <- ggline(design,
             x = "f",
             y = "Fc",
             add = c("mean", "jitter"),
             color = "red") + theme_bw()

p3 <- ggline(design,
             x = "vc",
             y = "Fc",
             add = c("mean", "jitter"),
             color = "green") + theme_bw()

ggarrange(p1,p2,p3)

p12 <- ggline(design,
             x = "f",
             y = "Fc",
             add = c("mean", "jitter"),
             color = "CB") + theme_bw()

p13 <- ggline(design,
             x = "vc",
             y = "Fc",

```

```

        add = c("mean", "jitter"),
        color = "CB") + theme_bw()

p23 <- ggline(design,
             x = "f",
             y = "Fc",
             add = c("mean", "jitter"),
             color = "vc") + theme_bw()

ggarrange(p12,p13,p23)

```

2.10 Nonlinear optimization for response surface

Response surface methodology (RSM) is a collection of statistical and mathematical techniques that can be used to optimize processes. RSM is useful for modeling and analyzing problems where the response variable is influenced by multiple factors. RSM can be used to optimize extractive processes for one or more bioactive compounds, among others.

In the optimization process, the first part consists of finding a suitable approximation for the relationship between the response and the factors, generally using low-degree polynomials. RSM can also be used to combine multiple responses into a single response using a mathematical function. The response surface obtained can be used to calculate the optimal values for each variable, to simultaneously satisfy all responses considered.

When we are at a point on the surface far from the optimum, there will be small curvature and a first-order model is adequate. The aim is to get as close as possible to this optimum and once it has been found, we can use a more elaborate analysis (second order model, for example).

Normally in the optimization process, the steepest ascending method is used to find the maximum increase direction of the response. The steepest upward path consists of a line that passes through the center of the region of interest and is perpendicular to the contours of the fitted surface.

Example 9: The following example consists of the non-linear optimization for the response surface of the components of a catalyst, taken from study [7]. In this design, we will use the rsm and phia package of R software and a rigid analysis of the design.

R Script

```
#####

library(rsm)

# designing with central points via RSM

plan = ccd(basis = ~x1+x2+x3,
           n0 = c(0,3),
           randomize = F,
           alpha = "rotatable",
           coding = list(x1 ~ (Co - 10)/2, # Co mass
                        x2 ~ (W - 1.5)/0.5, # W mass
                        x3 ~ (Ce - 4)/1)) # Ce mass

plan

y = c(88.36, 93.40, 89.22, 92.02, 91.28, 92.02, 89.62, 88.92, # factorials points
      85.98, 89.72, 91.43, 88.53, 95.66, 94.63, # axial points
      94.38, 94.53, 94.08) # central points

plan$y = y

#####
## Analysis via RSM

res.rsm = rsm( y ~ SO(x1,x2,x3), data = plan )
summary(res.rsm )

##### Assumptions

# Normality
shapiro.test(res.rsm$residuals )

#### Graphics
# Contour and Surface
par(mfrow = c(2,3))
contour(res.rsm , ~x1 + x2, image = TRUE)
contour(res.rsm , ~x1 + x3, image = TRUE)
contour(res.rsm , ~x2 + x3, image = TRUE)
persp (res.rsm , ~x1 + x2, zlab = "y", col = rainbow(50), contours = ("colors"))
persp (res.rsm , ~x1 + x3, zlab = "y", col = rainbow(50), contours = ("colors"))
persp (res.rsm , ~x2 + x3, zlab = "y", col = rainbow(50), contours = ("colors"))

##### Optimization

radius = (2^ 3)^ 0.25 # radius of the CCD (alpha) or root 4 of 2k
```

```

# Rigid Analysis
optimum = steepest(res.rsm , dist = seq (0, radius, by =.1), descent = F)
# maximize response
optimum

par(mfrow = c(1,3))
contour(res.rsm , ~x1 + x2, col = "black", decode = F)
points(c(-1, 1,- 1,1,-radius,radius,0,0,0), c(-1,-1,1,1,0,0,-radius,radius,0), col
= "blue", pch = 19)
points(x2 ~ x1, data = optimum , col = "magenta", pch = "*")

contour(res.rsm , ~x1 + x3, col = "black", decode = F)
points(c(-1, 1,- 1,1,-radius,radius,0,0,0), c(-1,-1,1,1,0,0,-radius,radius,0), col
= "blue", pch = 19)
points(x3 ~ x1, data = optimal , col = "magenta", pch = "*")

contour(res.rsm , ~x2 + x3, col = "black", decode = F)
points(c(-1, 1,- 1,1,-radius,radius,0,0,0), c(-1,-1,1,1,0,0,-radius,radius,0), col
= "blue", pch = 19)
points(x3 ~ x2, data = optimal , col = "magenta", pch = "*")

```

2.11 Simplex-lattice Design

A simplex-lattice planning $\{q,m\}$ is a set of uniformly spaced points in a simplex consisting of a q -component design that supports up to a polynomial of degree m . The $m+1$ points are equally spaced, observing that the proportion of the i th component and all possible combinations (mixtures) of the equation below are used:

$$x_i = 0, \frac{1}{m}, \frac{2}{m}, \dots, 1 \quad i = 1, 2, \dots, q \quad \text{Eq. 1}$$

For a three-component mixture fitting a polynomial of degree 2 ($q = 3, m = 2$), the simplex-lattice consists of the six points:

$$(x_1, x_2, x_3) = (1, 0, 0), (0, 1, 0), (0, 0, 1), (\frac{1}{2}, \frac{1}{2}, 0), (\frac{1}{2}, 0, \frac{1}{2}) \text{ and } (0, \frac{1}{2}, \frac{1}{2}) \text{ eq. (2.1)}$$

For this simplex-lattice $\{3,2\}$, the three vertices $(1,0,0)$, $(0,1,0)$ and $(0,0,1)$ are pure mixtures, and the points $(\frac{1}{2}, \frac{1}{2}, 0)$, $(\frac{1}{2}, 0, \frac{1}{2})$ and $(0, \frac{1}{2}, \frac{1}{2})$ are binary mixtures, located at the midpoints of the three edges of the triangle in Fig. 2.1.

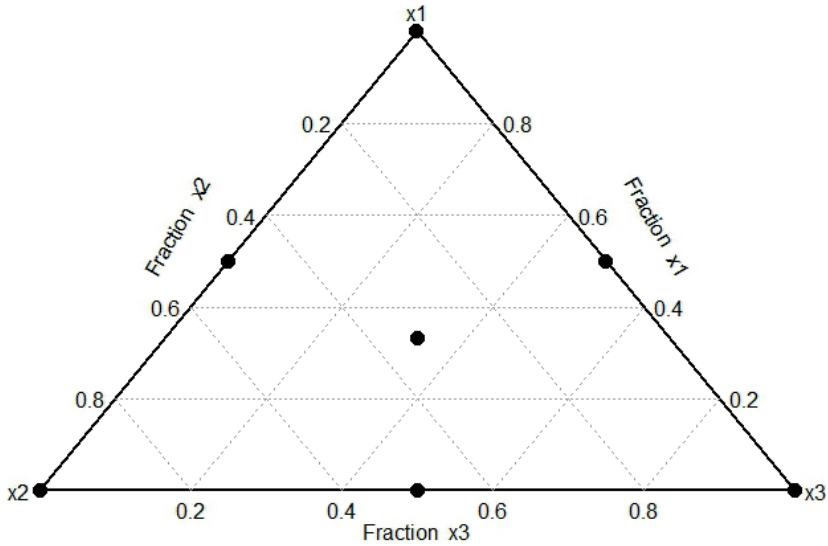


Fig. 2.1: Simplex-lattice design for three components.

The simplex-lattice design is an experimental design where the points are located on the edges or boundaries of the simplex. It is represented by six points, with the pure components being the vertices of the triangle and the binary mixtures. Simplex-lattice design was presented by Scheffé (1958) at the beginning of studies on experiments with mixtures.

Example 10: The following example consists of using simplex-lattice mixture design to optimize the natural fiber composition of a fiber-reinforced composite in 3 factors (F1 – sisal fiber, F2 – jute fiber, F3 – coconut fiber), extracted from the study [8]. In this design, we will use the mixexp and NlcOptim package of R software.

R Script

```
install.packages("mixexp")
library(mixexp)

# Simplex-lattice design with 3 components and grade 2
plan.simplex = SLD(3,2)
plan.simplex

# Planning was replicated 3 times

plan.simplex = rbind(plan.simplex, plan.simplex, plan.simplex)
plan.simplex

# Design planning
DesignPoints((plan.simplex))
```



```

# answer - specific breaking stress (SBS)
y = c(28.56, 21.73, 26.38, 33.71, 24.22, 22.93,
      29.58, 20.98, 25.9, 32.98, 23.98, 21.79,
      29.26, 21.23, 26.65, 34, 23.15, 22.17)

plan.simplex$SBS = y

##### Analysis

# complete model

res.composite = MixModel(frame = plan.simplex,
                        response = "SBS",
                        mixcomps = c("x1", "x2", "x3"),
                        model = 2)

summary(res.composite)

# reduced model - lm command
res.composite.reduced = lm(SBS ~ -1 + x1 + x2 + x3 + x1*x2 + x1*x3,
                          data = plan.simplex )

summary(res.composite.reduced )

##### Graphics

# Contour full model plot

ModelPlot(model = res.composite,
          dimensions = list( x1="x1", x2="x2", x3="x3"),
          contour = T,
          fill = T,
          axislabs = c("sisal", "juta", "coconut"),
          color.palette = cm.colors ,
          colorkey = T)

# Effects chart

ModelEff(nfac = 3,
        mod = 2,
        dir = 2,
        nproc = 0,
        ufunc = res.composite )

### Assumptions

```

```

#normality
shapiro.test(res.composite$residuals )
par(mfrow = c(2,2))
plot(res.composite)

### Non-linear optimization - rigid analysis

install.packages("Nlcoptim")
library(Nlcoptim)

# Objective function to be optimized
obj = function(x){
  y_hat = predict(res.composite , newdata = data.frame (x1 = x[1],
                                                         x2 = x[2],
                                                         x3 = x[3]))

  return(-y_hat)
}

# equality constraint
cons_eq = function(x){
  g = x[1] + x[2] + x[3] -1
  return(list( ceq = g,c = NULL))
}

# initial x

x0 = c(1/3, 1/3, 1/3)

# test objective function and constraint

obj(x0)
cons_eq(x0)

# optimization

opt = solnl(X = x0, objfun = obj, confun = cons_eq, lb = rep (0,3), ub = rep(1,3))

# Optimal proportions
x_optim = opt$par
x_optim

# Great answer
y_ = opt$fn
y_

```

2.12 Simplex-centroid Design

Simplex-centroid is a design where the points are located on the edges or borders of the simplex, with the exception of the central point (centroid). Simplex-centroid allows you to reduce the number of coefficients in a model. It is an alternative to the simplex-lattice design. The difference between the two designs is that the simplex-centroid creates additional points always aligned to the centroid, while in the simplex-lattice, the points cover the entire internal space.

Example 11: The following example consists of using simplex-centroid mixture design to optimize the composition of biodiesel based on vegetable oil and animal fat (factors: soybean oil, beef tallow, poultry fat; response: induction period), extracted from study [9]. In this design, we will use the `mixexp` and `NlcOptim` package of R software.

R Script

```
install.packages("mixexp")
library(mixexp)

# Simplex-centroide q = 3
plan.centroide = SCD(3)

# replicating the central points
plan.centroide = rbind(plan.centroide , plan.centroide [7,], plan.centroide [7,],
plan.centroide [7,])
plan.centroide

# Drawing planning
DesignPoints((plan.centroide))

# Answer - Induction period
IP = c(3.76, 9.57, 9.77, 8.19, 7.92, 12.92,
      10.04, 9.27, 10.07, 9.35)

# adding response to planning
plan.centroide$y = IP
plan.centroide

##### Analysis

# complete model

res.centroide = MixModel(frame = plan.centroide,
                        response = "IP",
                        mixcomps = c( "x1", "x2", "x3"),
                        model = 4) #special cube = 4

summary(res.centroide)
```

```

##### Graphics

# Contour full model plot

ModelPlot(model = res.centroide ,
           dimensions = list( x1="x1", x2="x2", x3="x3"),
           contour = T,
           fill = T,
           axislabs = c( "oleo_soy", "tallow_beef", "fat_birds"),
           color.palette = terrain.colors ,
           colorkey = T)

# Effects chart
ModelEff(nfac = 3,
         mod = 4,
         dir = 2,
         nproc = 0,
         ufunc = res.centroide )

### Assumptions

# Normality
shapiro.test(res.centroide$residuals)

### Non-linear optimization - rigid analysis

library(Nlcoptim)

# Objective function to be optimized
obj = function(x){
  y_hat = predict(res.centroide , newdata = data.frame (x1 = x[1],
                                                         x2 = x[2],
                                                         x3 = x[3]))

  return(-y_hat)
}

# equality constraint
cons_eq = function(x){
  g = x[1] + x[2] + x[3] -1
  return( list( ceq = g,c = NULL))
}

# initial x
x0 = c(1/3, 1/3, 1/3)

```

```

# test objective function and constraint

obj(x0)
cons_eq(x0)

# optimization

opt = solnl(X = x0, objfun = obj, confun = cons_eq, lb = rep (0,3), ub = rep(1,3))

# Optimal proportions
x_optim = opt$par
x_optim

# Great answer
y_ = opt$fn
y_

```

PROPOSED EXERCISES

01 – Propose a complete factorial design (2 levels) using a script in R language and present your main conclusions.

02 – There are multivariate data repositories on the internet (web of science, Science Direct, and others) in which you must choose a full 3-level factorial design to perform a statistical study in detail. Present the main results and conclusions.

03 – Present a fractional factorial design through a real example containing an R script to demonstrate your hypotheses and main conclusions.

04 – Propose a Box-Behnken experiment and perform a statistical interpretation presenting its main conclusions.

05 – Present a simplex-lattice design of mixtures and your main conclusions. Also, perform non-linear optimization or rigid analysis of the experiment.

06 – In the same way as the previous exercise, present a simplex-centroid design of mixtures and your main conclusions. Also, perform non-linear optimization or rigid analysis of the experiment.

07 – Present a multi-level factorial design and its main conclusions using the R language.

REFERENCES

1 – Escudero , LA; Cerutti, S.; Olsina , RA; Salonia, JA; Gasquez , J. A. (2010). Factorial design optimization of experimental variables in the on-line separation/preconcentration of copper in water samples using solid phase extraction and ICP-OES determination. *Journal of Hazardous Materials*. 183 (1-3): 218-223.

- 2 – Kumar, S.; Bablu, M.; Janghela, S. *et al.* (2018). Factorial design, processing, characterization and microstructure analysis of PIP-based C/ SiC composites. *Bull Mater Sci* . 41 (17): 1:13.
- 3 – X, Hu.; J, Xu.; C, Wu *et al.* (2017). Ethylenediamine grafted to graphene oxide@Fe₃O₄ for chromium(VI) decontamination: Performance, modeling, and fractional factorial design. *PLoS One* . 12(10): 1:14.
- 4 – Azam, M., Jahanzaib, M., Abbasi, JA *et al.* (2016). Parametric analysis of recast layer formation in wire-cut EDM of HSLA steel. *Int J Adv Manuf Technol* 87 :713–722.
- 5 – Azhikannickal, Elizabeth & Uhrin, Aaron. (2019). Dimensional Stability of 3D Printed Parts: Effects of Process Parameters. *The Ohio Journal of Science*. 119(2): 9-16.
- 6 – Pereira, RBD, Braga, DU, Nevez, FO *et al.* (2013). Analysis of surface roughness and cutting force when turning AISI 1045 steel with grooved tools through Scott–Knott method. *Int J Adv Manuf Technol* 69, 1431–1441.
- 7 – Li, S.; Qin, X.; Zhang, G. *et al.* (2020). Optimization of content of components over activated carbon catalyst on CO₂ reforming of methane using multi-response surface methodology. *International Journal of Hydrogen Energy* . 45(16): 9695-9709.
- 8 – D, Das.; S, Mukhopadhyay.; H, Kaur.; (2012). Optimization of fiber composition in natural fiber-reinforced composites using a simplex lattice design. *Journal of Composite Materials* . 46(26):3311-3319.
- 9 – Orives, JR; Galvan, D.; Coppo, R.L.; *et al.* (2014). Multiresponse optimization on biodiesel obtained through a ternary mixture of vegetable oil and animal fat: Simplex-centroid mixture design application. *Energy Conversion and Management* . 398-404.

PATTERN RECOGNITION



" There is no short cut to truth, no way to gain a knowledge of the universe except through the gateway of scientific method " **Karl Pearson (1857-1936)**

CHAPTER IDEA

The term pattern can be defined as the opposite of chaos or a loosely defined entity that can be given a name. Formally, pattern recognition is the area of science that aims to classify objects (patterns) into a number of categories or classes. Thus, for a given set of c classes, $\omega_1, \omega_2, \dots, \omega_c$, and unknown pattern x , a pattern recognizer will be a system that, aided by pre-processing, feature extraction and selection, associates x to label i of a class ω_i .

We can find pattern recognition techniques in several applications, such as: i) analysis of genome sequences, in microarray applications and technology (bioinformatics); ii) data mining; iii) medical diagnosis; iv) biometric recognition; v) remote sensing using multispectral images; and, vi) speech recognition.

Basically, pattern recognition techniques are based on three main steps: i) data acquisition for extraction and selection of the most informative features; ii) data representation; and, iii) construction of a classifier or descriptor for decision making.

Generally, the classifier used in pattern recognition techniques learns how to map the feature data space. Thus, we can group pattern classification techniques, according to the type of learning, into two forms: i) unsupervised analysis (they use patterns that do not have defined class labels); and, (ii) supervised analysis (the patterns belong to a pre-defined class).

Upon completing the chapter, you should be able to:

- Apply the main unsupervised analysis algorithms (HCA, K-means and PCA) to a set of real or simulated data, seeking to identify patterns in the analyzed data;
- Apply the main supervised analysis algorithms (LDA, QDA, KNN, SVM and decision trees) to real or simulated data sets in the construction of classification models;
- Understand the stages of building multivariate classification models and evaluating the models (confusion matrix and ROC curve);
- Build new scripts in R language for decision making using the pattern recognition technique;
- Propose new applications in chemistry or related areas of pattern recognition techniques.

UNSUPERVISED ANALYSIS

3.1 Cluster Analysis

The term Cluster Analysis was first used by Tyron in 1939 [1] in an attempt to organize observed data into structures that make sense or in the construction of taxonomies capable of classifying samples into different classes. In Biology, Zoologists use taxonomy in an attempt to classify observed specimens (samples) into groups. In Chemistry, for example, there are many situations in which Cluster Analysis appears, and throughout the text, they will be presented and discussed in detail.

The purpose of grouping (*clustering*) is to define intrinsic groups in a set of data that does not have labels, so that the objects in each group are similar according to some pre-established criteria. In other words, we have a statistical tool with which it is possible to form groups with homogeneity within the grouping, and heterogeneity between them. The presence of personal computers has made routine evaluation of complex data sets (with thousands of variables and samples) possible. Currently, there are several tools available to extract useful information from complex data using the detection and evaluation of patterns in your dataset.

Generally speaking, cluster analysis does not require any initial assumptions about the structure of the data. The search for a natural grouping structure in the data itself is an important exploratory technique. The aim, therefore, is to find natural groupings and classify samples characterized by the values of a set of variables into groups. This technique aims to partition the elements of a data set into two or more groupings based on their similarity founded on a set of variables. We must remember that the cluster solution is not generalizable because it is totally dependent on the variables used as a basis for the similarity measure. Finally, we can identify three common applications in cluster analysis: i) classification of elements (taxonomy); ii) data simplification; and, iii) identification of relationships between elements.

3.2 Hierarchical Cluster Analysis (HCA)

A hierarchical cluster is a sequence of partitions in which each partition is allocated to its neighboring partition in the sequence. The aim of this technique is to classify samples using similarity measures. We can understand the similarity between elements as an empirical measure of correspondence (distance) or similarity between the elements to be grouped. The smaller the distance between samples in n-dimensional space, the greater the similarity. In summary, distance measures are actually dissimilarity measures, that is, higher values indicate greater dissimilarity between the variables. From the results, and using the inverse relationship, it is possible to identify the similarity measure.

In HCA, metric distances are calculated between the samples (objects) that form the data set, and these are grouped according to the degree of similarity presented. HCA comprises agglomerative and divisive ways of forming clusters. In agglomerative procedures (most common), we start with the instances forming disjoint unitary groups (*singletons*), that is, each of the n instances in the data set will be assigned to a group (cluster). In divisive hierarchical analysis, the process occurs in the opposite order to the agglomerative one. The results provided by HCA are called dendrograms (Figure 3.1), which graphically express the distance (similarity) between the samples.

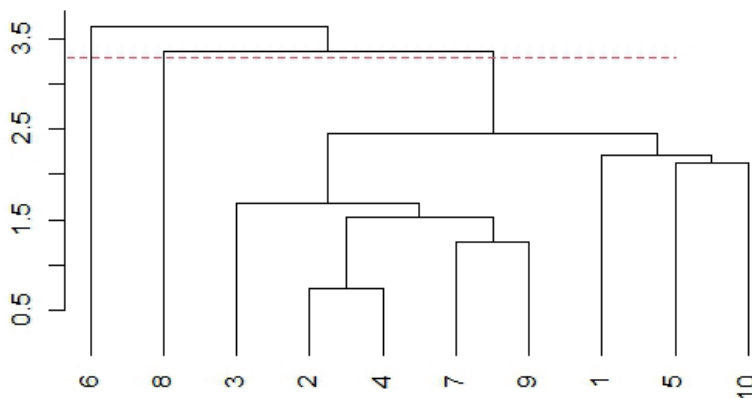


Figure 3.1 : Representation of a dendrogram or binary tree.

Several criteria can be adopted to choose the number of clusters. The desired number of clusters can be known in advance or a predetermined distance value is used as a criterion to separate the number of clusters. The number of clusters is chosen based on observation of the dendrogram, based on knowledge of the data.

For Cluster Criteria, the distance from an object to cluster k can be calculated as the average distance of objects A and B to object i , in several ways:

- I. Single Link (KNN) – The shortest distance between clusters is calculated. This procedure is also known as KNN (Kth Nearest Neighbor);
- II. Complete connection – Based on the greatest distance between objects in opposite groupings. In general, small, compact, spherical and well-separated clusters tend to form;
- III. Centroid link (k-means) – A centroid is calculated as the average of the objects in a cluster. Spatial distortion of the grouping is avoided and tends to preserve groups of small importance in relation to larger ones;
- IV. Ward's method – The clusters are aggregated in such a way as to minimize the sum of squares of the deviations of each centroid in relation to the group itself.

Example 1: In this example, an HCA is performed for a simulated dataset using the `dist` function in the R language.

R Script

```
# Simulated data

A=c(9.60, 8.40, 2.40, 18.20, 3.90, 6.40)
B=c(28, 31, 42, 38, 25, 41)

data= cbind(A,B)

# Distance calculation

? dist

DE= dist(data, method = "euclidean", diag = TRUE, upper = TRUE, p =2)
dendo = hclust ( DE,method = "average")

# Dendrogram

plot(dendo)
```

Example 2: In this example, an HCA is performed for an ICP-OES dataset using the R packages *factoextra* and *NbClust*.

R Script

```
# Loading the dataset

D1= read.table("ICPOES.txt")
D2= D1[,-1]

# Detection of atypical variables
# Calculating Mahalanobis distance
p.cov <- var(scale(D2)) # standardize first
p.cov <- var(D2)
p.mean <- apply(D2,2,mean)
p.mah <- mahalanobis(D2, p.mean, p.cov)
View(p.mah)

# Analyzing variance
# Variables with different scales and different variances can distort the analysis
apply(D2, 2, var)

#Standardizing variables
DP <- scale(D2)
apply(DP, 2, mean)
apply(DP, 2, var)
```

```

#-----
# 2. Selection of grouping criteria
# Database rows must represent observations (samples)
# that you want to group.
# Columns must be formed by variables.
#-----
# Select the similarity (or dissimilarity) criterion that will determine
# which observations are similar, and should be grouped into a given group, and
# which are not
# similar, and must be in different groups.
# For a similarity measure, the lower its value, the more similar two observations
# are.
# Calculating Euclidean distance
d.eucl <- dist(DP, method = "euclidean")

#Viewing the Euclidean distance rounding 1 decimal place:
round(as.matrix( d.eucl)[1:4, 1:4], 1)

#-----
# 3. Selection of clustering algorithm
# Hierarchical x Non-hierarchical
#-----

#Hierarchical method of Ward's minimum variance or mean distance

res.hc <- hclust(d = d.eucl, method = "ward.D2")

# Calculating the cophenetic matrix
## Compares the distances actually observed between objects and
## the distances predicted from the grouping process.
res.coph <- cophenetic(res.hc)

# Correlation between the cophenetic distance and the original distance
cor(d.eucl, res.coph)

#Comparing with the average link method
hc.m <- hclust(d.eucl, method = "average")

# Correlation between the cophenetic distance and the original distance
cor(d.eucl, cophenetic(hc.m))

#-----
# 4. Defining the number of clusters
#-----
#Loading the factoextra package

install.packages("factoextra")
library(factoextra)

```

```

# Obtaining the dendrogram
fviz_dend(hc.m, cex = 0.5)

# Some indicators can be used to help choose the number of groupings.
#To calculate these indices we must install the NbClust package
install.packages("NbClust")
library(NbClust)

# Note: We can install more than one package at a time using
pkgs <- c("factoextra", "NbClust")
install.packages(pkgs)

nb <- NbClust(DP, distance = "euclidean", min.nc = 2,
              max.nc = 10, method = "average", index = "all")

#method = NULL must be replaced by the grouping algorithm used ("ward.D",
# "ward.D2", "single", "complete", "average", "kmeans", etc.)

#Getting the indicators
nb[["All.index"]]

#-----
#For just one indicator, use the help
?NbClust

#For only the ccc index

nb.c <- NbClust(DP, distance = "euclidean", min.nc = 2,
                max.nc = 10, method = "ward.D2", index = "ccc")

fviz_nbclust(nb.c)

#-----
#For pseudo-f('ch')
nb.i <- NbClust(DP, distance = "euclidean", min.nc = 2,
                max.nc = 10, method = "ward.D2", index = "ch")

fviz_nbclust(nb.i)

#-----
-----
#5. Interpretation and validation of groupings
#-----
-----
#Getting the groupings
g <- cutree(hc.m,k =3)

```

```

#Number of members in each group
table(g)

#We can visualize the clustering result in the dendrogram
fviz_dend(hc.m, k = 3, # Cut in four groups
          cex = 0.5, # label size
          k_colors = c("#2E9FDF", "#00BB0C", "#E7B800", "#FC4E07"),
          color_labels_by_k = TRUE, # color labels by groups
          rect = TRUE # Add rectangle around groups
)

```

Example 3: In this example, an HCA is performed for a dataset containing 8 physicochemical properties of 89 chemical elements using the R packages *factoextra* and *NbClust*.

R Script

```

# load the TP dataset (89 x 9)
D1 = read.table("TP.txt")
D2 = D1[,-1]

# Detection of atypical variables
# Calculating Mahalanobis distance
p.cov <- var(scale(D2)) # standardize first
p.cov <- var(D2)
p.mean <- apply(D2,2,mean)
p.mah <- mahalanobis(D2, p.mean, p.cov)
View(p.mah)

#Variables with different scales and different variances can distort
## the analysis
apply(D2, 2, var)

#Standardizing variables
DP <- scale(D2)
apply(DP,2,mean)
apply(DP,2,var)

#-----
# 2. Selection of similarity or dissimilarity criterion
# For a similarity measure, the lower its value, the more similar two observations
are.
# Calculating Euclidean distance
d.eucl <- dist(DP, method = "euclidean")

#Viewing the Euclidean distance rounding 1 decimal place:
round(as.matrix(d.eucl)[89:8, 89:8], 1)

```

```

#-----
# 3. Selection of clustering algorithm
# Hierarchical x Non-hierarchical
#-----

#Hierarchical method of Ward's minimum variance or mean distance
res.hc <- hclust(d = d.eucl, method = "ward.D2")

# Calculating the cophenetic matrix
## Compares the distances actually observed between objects and
## the distances predicted from the grouping process.
res.coph <- cophenetic(res.hc)

# Correlation between the cophenetic distance and the original distance
cor(d.eucl, res.coph)

#Comparing with the average link method
hc.m <- hclust(d.eucl, method = "average")

#cophenetic distance and the original distance
cor(d.eucl, cophenetic(hc.m))

#-----
# 4. Defining the number of clusters
#-----

#Loading the factoextra package
install.packages("factoextra")
library(factoextra)

# Obtaining the dendrogram
fviz_dend(hc.m, cex = 0.5)

# Some indicators can be used to help choose the number of groupings.
# To calculate these indices we must install the NbClust package
install.packages("NbClust")
library(NbClust)

# Note: We can install more than one package at a time using
pkgs <- c("factoextra", "NbClust")
install.packages(pkgs)

nb <- NbClust(DP, distance = "euclidean", min.nc = 2,
             max.nc = 10, method = "average", index = "all")

# method = NULL must be replaced by the grouping algorithm used ("ward.D",
# "ward.D 2", "single", "complete", "average", "kmeans", etc.)

#Getting the indicators
nb[["All.index"]]

```

```

#-----
#For just one indicator, use the help
?NbClust

#For only the ccc index

nb.c <- NbClust(DP, distance = "euclidean", min.nc = 2,
                max.nc = 10, method = "ward.D2", index = "ccc")

fviz_nbclust(nb.c)

#-----
#For pseudo-f('ch')
nb.i <- NbClust(DP, distance = "euclidean", min.nc = 2,
                max.nc = 10, method = "ward.D2", index = "ch")

fviz_nbclust(nb.i)

#-----
-----
#5. Interpretation and validation of groupings.
#-----
-----
#Getting the groupings
g <- cutree(hc.m,k=4)
#Number of members in each group
table(g)

#We can visualize the clustering result in the dendrogram
fviz_dend(hc.m, k = 4, # Cut in four groups
          cex = 0.5, # label size
          k_colors = c("#2E9FDF", "#00BB0C", "#E7B800", "#FC4E07"),
          color_labels_by_k = TRUE, # color labels by groups
          rect = TRUE # Add rectangle around groups
)

```

3.3 K-means

K-means is partitional (non-hierarchical) center-based technique, that is, the groups formed by this technique are represented by a centroid (a central point in the group). K-means was proposed in a pioneering work by S. Lloyd in 1957, however, it was only published in 1982 [2]. For Lloyd, the centroid was chosen as the point that minimizes the sum of the square of the Euclidean distance, d_E , between itself and each point in the set, according to the equation:

$$c_i = \operatorname{argmin} \sum_{p_j \in x_i} d_E^2(c_i, p_j) \quad \text{Eq. 1}$$

The aim of the K-means algorithm is to minimize the sum of squared error over all k groups:

$$\sum_{j=1}^k \sum_{p_i \in x_i} d(p_i, c_j)^2 \quad \text{Eq. 2}$$

Basically, K-means performs five main steps: i) selects k instances (randomly) to be the initial centroids of the groups; ii) assigns all instances to the closest centroid; iii) recalculates the centroid for each group; iv) calculates the averages of all instances of the group; v) repeat steps (ii and iii) until the centroids do not change.

A critical point in the use of K-means that determines its performance is the choice of initial centroids. The choices, despite being random, in general, can lead to a local minimum. However, we can find the use of K-means in various applications, such as: data mining, statistics, engineering, machine learning, medicine, marketing, administration and biology.

Example 4: In this example the K-means algorithm is applied to an ICP-OES dataset using the R *stats* and *cluster* packages.

R Script

```
# loading data
D1= read.table ("ICPOES.txt")
D2=D1[,-1]

# kmeans(x, centers, iter.max = 10, nstart = 1)
## Defining a seed. This allows the result to be
## reproducible, since the seed interferes with the final result
set.seed(123)

DP <- scale(D2)
km.res <- kmeans(DP, 3, nstart = 25)
print(km.res)

#Interpretation and validation of clusters
#-----
#Adding the k-means cluster column to the original data
DPK <- cbind(D2, Groups=km.res$cluster)
DPK

#Calculating the group average for the original data
aggregate(D2, by=list(cluster= km.res$cluster), mean)

round(aggregate(D2, by=list(cluster= km.res$cluster), mean),1)
```



```

# vizualizing the clusters

fviz_cluster (km.res, data = D2,
              palette = c( "#2E9FDF", "#00BB0C", "#E7B800", "#FC4E07"),
              ellipse.type = "euclid", # Concentration ellipse
              star.plot = TRUE, # Add segments from centroids to items
              repel = TRUE, # Avoid label overplotting (slow)
              ggtheme = theme_minimal())

## vizualizing the clusters in 2D

install.packages("cluster")
library(cluster)
clusplot (DP, km.res$cluster , main='Two-dimensional cluster representation',
          color=TRUE, shade=TRUE,
          labels=2, lines=0)

```

Example 5: In this example the K-means algorithm is applied to an ICP-OES dataset using the *FactoMineR*, *factoextra*, *cluster*, *ggplot2* and *xlsx* packages.

R Script

```

install.packages("FactorMineR")
install.packages("factoextra")
install.packages("cluster")
install.packages("ggplot2")

library(FactoMineR)
library(factoextra)
library(cluster)
library(ggplot2)

# importing data

D1= read.table("Matrix 54x7.txt", head=T)
D2=D1[,-1]
DP=apply(D2, 2, function(x)(x-mean(x))/sd (x))

#Bloxplot
boxplot(D2$Ca)

#Scaling
Data = scale(D2)

#Defining optimal number of cluster
fviz_nbclust(DP, kmeans, method = "gap_stat")

#Generate kmeans
data_kmeans = kmeans(DP,5)

```

```
#vizualizing the clusters
fviz_cluster(data_kmeans, data = DP, ellipse.type = "t")

#Creating a list of clusters

list = data_kmeans$cluster
#Grouping data into a table
general_data = cbind(D2, list)
general_data
```

3.4 Principal Component Analysis (PCA)

Principal component analysis technique (PCA), originally proposed in 1901 by Karl Pearson [3], is a mathematical procedure that uses an orthogonal transformation or orthogonalization of vectors to convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables called principal components. PCA is a tool used to reduce the dimensionality of a set of variables by creating a new base, whose components are linearly independent and fewer in number. These components are ordered in order to maintain the largest portion of the original variance in the first components.

One of the ways to calculate the PCA is through the covariance method. This method presents some main steps: i) organize the data set in the form of an $n \times m$ matrix (n is the number of observations or samples and m is the number of measured variables); ii) if necessary, carry out some pre-treatment (normalization, correlation or autoscaling); iii) calculate the covariance matrix; iv) calculate the eigenvalues and eigenvectors associated with the covariance matrix; v) order the eigenvectors according to the associated eigenvalues - the first eigenvector is the first principal component and so on; vi) calculate the percentage of the original variance from the associated eigenvalues. Another way to perform PCA is through singular value decomposition (SVD). The SVD method of matrix \mathbf{X} is $\mathbf{X} = \mathbf{W}\mathbf{\Sigma}\mathbf{V}^T$, where the $m \times m$ matrix \mathbf{W} is the eigenvector matrix of the covariance matrix $\mathbf{X}\mathbf{X}^T$, the matrix $\mathbf{\Sigma}$ is $m \times n$ and is a rectangular diagonal matrix with non-negative real numbers on the diagonal, and the $n \times n$ matrix \mathbf{V} is the eigenvector matrix of $\mathbf{X}^T\mathbf{X}$.

In a principal components analysis, the grouping of samples defines the structure of the data through graphs of scores and loadings whose axes are principal components onto which the data is projected. The scores provide the composition of the PCs in relation to the samples, while the loadings provide this same composition in relation to the variables. As the principal components are orthogonal, it is possible to examine the relationships between samples and variables through scores and loadings graphs.

Example 6: In this example, the PCA algorithm is applied to an ICP-OES dataset using the *FactoMineR* and *factoextra* packages.

R Script

```
#loading the ICP-OES dataset (54 x 7)

D1 = read.table("ICPOES.txt")
D2 = D1[,- 1]

#Calculating the covaraince matrix var-cov(X)
cov.c <- cov(D2)

#Checking the type of the cov.c object
class(cov.c)

##Viewing only some elements of the var-cov(X) matrix
cov.c[1:5,1:4]

##Calculating the total variance:
sum(diag(cov.c))

##Calculating the generalized variance:
det(cov.c)

#-----
#Calculating the correlation matrix (X)
cor.c <- cor(D2)

#Viewing the cor.c matrix
cor.c

##Calculating the total variance
var.total <- sum(diag(cor.c))
var.total

#-----
#Calculating eigenvalues and eigenvectors for cor.c
ev <- eigen(cor.c)

#Viewing the data stored in the ev objective
ev

# Extracting the eigenvalues
c.values <- ev$values

#Extracting the eigenvectors
c.vectors <- ev$vectors

##Calculating the percentage explained by each component (Yi=ei1x1+ei2x2+...+ eipxp)
per.var <- c((c.values/var.total)* 100)
per.var
```

```

#####
#Principal component analysis.
install.packages(c("FactoMineR", #for analysis
                  "factoextra" #to plot the principal components
))

library(FactoMineR)
library(factoextra)

res.pca <- PCA(D2, graph = T)

print(res.pca)

#See the components included in the res.pca object
eig.val <- get_eigenvalue(res.pca)
eig.val

#Determining the number of components to maintain
fviz_eig(res.pca)
fviz_eig(res.pca , addlabels = TRUE, xlab = "PCs", ylab = "Percent of explained
variance", ylim = c(0, 80))

#Correlation circle
fviz_pca_var ( res.pca , #name of the object that saved the results
              col.var = "black" ,
              repel = TRUE, # Prevent text from overlapping
              title = "Correlation circle, variables x PC" #title
)

# Quality of representation is measured by "cos2":

install.packages("corrplot")
library(corrplot)

corrplot(res.pca$var$cos2, is.corr =TRUE)

#bar chart for "cos2"

fviz_cos2(res.pca, choice = "var", axes = 2:3,
          title =("Bar chart for cos2"))

#We can represent variables in a colorful and gradient way

fviz_pca_var(res.pca, col.var = "cos2",
             gradient.cols = c("#00AFBB", "#E7B800", "#FC4E07"),
             repel = TRUE) # Avoid text overlapping

```

```

#Biplot graph (scores and loadings)

fviz_pca_biplot(res.pca , title = "PCA Plot")

#-----
#Variables that are correlated with PC1 (Dim1) and PC2 (Dim2)
head(res.pca$var$contrib, 4)

corrplot(res.pca$var$contrib, is.corr=FALSE)

# Contribution of variables to PC1

fviz_contrib(res.pca, choice = "var", axes = 1, top = 10,
             title ="Variables contribution to PC 1")

# Contribution of variables to PC2

fviz_contrib(res.pca, choice = "var", axes = 2, top = 10,
             title ="Variables contribution to PC 2")

#Contribution of variables to PC1 and PC2
fviz_contrib(res.pca, choice = "var", axes = 1:2, top = 10)

# We can use the function dimdesc() [in FactoMineR], to identify
## the variables associated with the greater significance with the component

res.desc <- dimdesc(res.pca, axes = c(1,2,3), proba = 0.05)
print(res.desc)

#-----
#To export the results, we use

write.infile(res.pca, "pca.csv", sep = ";")

#Saving the value of components
cp <- res.pca$ind$coord

#Exporting to a .csv file named cp

library( FactoMineR )
write.infile(cp, "cp.csv", sep = ";")

```

Example 7: In this example, the PCA algorithm is applied to the Iris dataset (150 x 5) using the *FactoMineR* and *factoextra*. Note that PCA will be performed both for the original dataset and for the same one after pre-processing (done using the "scale" function).

R Script

```
# loading dataset

D = iris
D1 = data.frame(D[,c(1:4)]) # original dataset
D2 = scale(D1) # pre-processed dataset

#-----
#calculating the var-cov matrix (X)
cov.c = cov(D1)
cov.c2 = cov(D2)

#calculating the total variance
sum(diag(cov.c))
sum(diag(cov.c2))

#calculating the generalized variance
det(cov.c)
det(cov.c2)

#-----
#calculating the correlation matrix
cor.c = cor(D1)
cor.c2 = cor(D2)

#calculating the total variance
var.total = sum(diag(cor.c))
var.total2 = sum(diag(cor.c2))
#-----

#Calculating eigenvalues and eigenvectors for cor.c
ev =eigen(cor.c)
ev2=eigen(cor.c2)

#visualizing the data stored in the eigenvalues
ev
ev2

#extracting the eigenvalues
c.values = ev$values
c.values2 = ev2$values

#extracting the eigenvectors
c.vectors = ev$vectors
c.vectors2 = ev2$vectors
```

```

#calculating the percentage explained by each component (Yi=ei1x1+ei2x2+... eipxp)
per.var = c((c.values/var.total)*100)
per.var
per.var2 = c((c.values2/var.total2)*100)
per.var2

#####
#Principal component analysis

#Installing the packages
install.packages("FactoMineR")
install.packages("factoextra")
install.packages("ggplot2")

library(FactoMineR)
library(factoextra)
library(ggplot2)

res.pca = PCA(D1, graph = T)
res.pca2 = PCA(D2, graph=T)

#Biplot graph (scores and loadings)

fviz_pca_biplot(res.pca, title = "PCA Iris Plot")

fviz_pca_biplot(res.pca2, title = "PCA Iris Plot")

#The values for each of the components are stored in
# indi$coord

eig.val = get_eigenvalue(res.pca)
eig.val2 = get_eigenvalue(res.pca2)

##Determining the number of components maintained
eig.val
eig.val2

#Determining the number of components to maintain

fviz_eig(res.pca, addlabels = TRUE, xlab = "PCs", ylab = "Percent of explained
variance", ylim =c(0,100))

fviz_eig(res.pca2, addlabels = TRUE, xlab = "PCs", ylab = "Percent of explained
variance", ylim =c(0,100))

```

```

#-----
#Correlation circle
fviz_pca_var ( res.pca ,
               col.var = "black",
               repel = TRUE,
               title = "Correlation circle, variables X PC") #title

fviz_pca_var ( res.pca2 ,
               col.var = "black",
               repel = TRUE,
               title = "Correlation circle, variables X PC") #title

#Quality of representation is measured by cos2

install.packages("corrplot")
library(corrplot)

corrplot(res.pca$var$cos2, is.corr = FALSE)
corrplot(res.pca2$var$cos2, is.corr = FALSE)

#We can represent variables in a colorful and gradient way

fviz_pca_var(res.pca, col.var = "cos2",
             gradient.cols= c("#00AFBB", "#E7B800", "#FC4E07"),
             repel = TRUE)

fviz_pca_var(res.pca2, col.var = "cos2",
             gradient.cols= c("#00AFBB", "#E7B800", "#FC4E07"),
             repel = TRUE)

#-----
#Variables that are correlated with PC1 (Dim1) and PC2 (Dim2)

head(res.pca$var$contrib,4)
corrplot(res.pca$var$contrib , is.corr = FALSE)

head(res.pca2$var$contrib,4)
corrplot(res.pca2$var$contrib, is.corr = FALSE)

#contribution of variables to PC1

fviz_contrib (res.pca , choice = "var", axes =1, top = 10,
              title ="contribution of variables to Dim1")

```



```

fviz_contrib (res.pca2, choice = "var", axes = 1, top = 10,
              title ="contribution of variables to Dim1")

#contribution of variables to PC2

fviz_contrib (res.pca , choice = "var", axes = 2, top = 10,
              title ="contribution of variables to Dim2")

fviz_contrib (res.pca2, choice = "var", axes = 2, top = 10,
              title ="contribution of variables to Dim2")

#contribution of variables to PC1 and PC2

fviz_contrib (res.pca , choice = "var", axes = 1:2, top = 10)

fviz_contrib (res.pca2, choice = "var", axes = 1:2, top = 10)

# We can use the function dimdesc () [in FactoMiner ], to identify
# the variables associated with greater significance with the component

res.desc = dimdesc(res.pca, axes = c(1,2,3), proba = 0.05)
print(res.desc)

res.desc = dimdesc(res.pca2, axes = c(1,2,3), proba = 0.05)
print(res.desc)

#to export the results, we use

write.infile(res.pca, "pca.csv", sep = ";")

write.infile(res.pca2, "pca2.csv", sep = ";")

#saving the value of components

cp = res.pca$ind$coord

cp2 = res.pca2$ind$coord

#Exporting to a .csv file named cp

library(FactoMineR)

write.infile(cp, "cp.csv", sep =";")
write.infile(cp, "cp2.csv", sep =";")

```

3.5 Multivariate Curve Resolution with Alternating Least Squares (MCR-ALS)

In 1971, Lawton and Sylvestre [4] presented to the scientific community the emergence of methodologies called Curve Resolution with the study entitled "Self Modeling Curve Resolution ". Basically, these researchers present a method for determining the forms of two overlapping functions $f_1(x)$ and $f_2(x)$ from an observed set of additive mixtures, $\{\alpha_i f_1(x) + \beta_i f_2(x); i = 1, \dots, n\}$, of the two functions.

From this initial study, there was an evolution of Multivariate Curve Resolution (MCR) methods for analytical signal processing whose goal is to resolve mixtures of non-selective signals originating from an instrument (D) into real contributions from the pure components in the system (represented by the concentration profiles in **C** and spectral profiles in **S^T**), as exemplified in the equation below:

$$\mathbf{D} = \mathbf{CS}^T \quad \text{Eq. 3}$$

where **D** is the instrumental response matrix, **C** is the relative concentration matrix, and **S** is a matrix of pure spectra.

It is important to point out that the MCR method does not require *a priori information* about the contribution of different factors to the overall response. However, for the MCR method to be successful in the analysis, two requirements are fundamental: i) the analytical signal must obey a relationship similar to the Beer-Lambert law (linear relationship with concentration); ii) the rank of the matrix must be equal to the number of species that produce analytical signal present in the mixtures (i.e., the number of vectors that cannot be written as a linear combination of the others).

The calculation performed by MCR-ALS uses alternating least squares (ALS) to seek the result that presents the best fit through a process called "optimization". This process allows the recovery of individual concentration profiles and species signals that best explain the variance of the observed data based on knowledge of the signals or concentrations of pure components present in the data matrix.

Iterative optimization, with constraints, via ALS can be described in two main steps:

$$\mathbf{C} = \mathbf{DS}(\mathbf{S}^T\mathbf{S})^{-1} \quad \text{Eq. 4}$$

$$\mathbf{S}^T = (\mathbf{C}^T\mathbf{C})^{-1}\mathbf{C}^T\mathbf{D} \quad \text{Eq. 5}$$

In this process, a matrix D, reconstructed from the product of the \mathbf{CS}^T matrices, in which C or \mathbf{S}^T comes from the initial estimate, is calculated and compared with the original matrix D. Iterative optimization continues until the convergence criterion is met. Figure 3.2 shows a graphical representation of the MCR-ALS method.

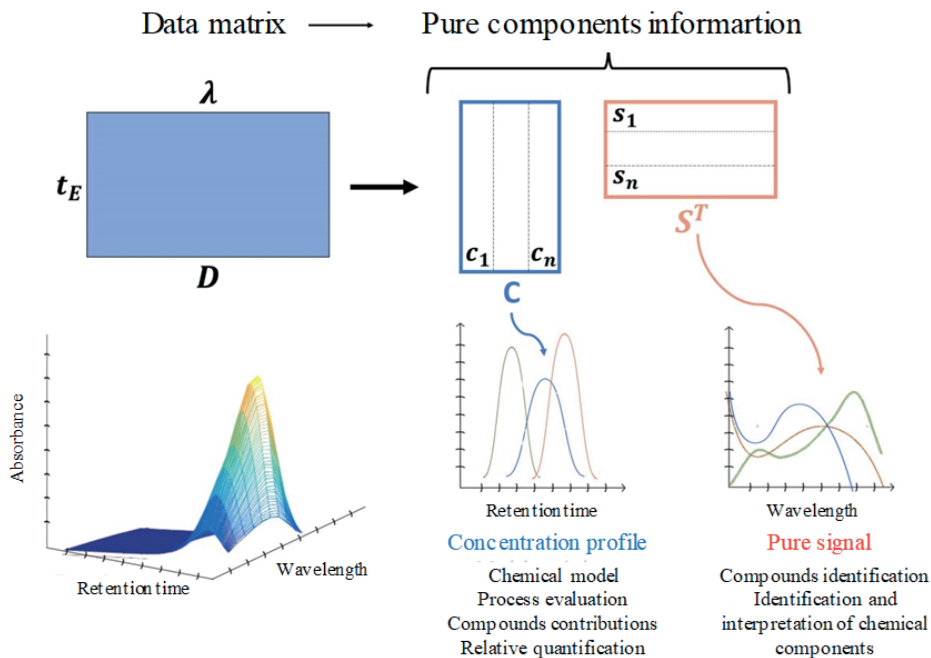


Figure 3.2: Graphical representation of the MCR-ALS method.

Example 8: In this example, the MCR-ALS algorithm is applied to a dataset of chromatographic data (elution time x mass spectrum) using the ALS package, in which the components recovered by the MCR-ALS model in the mixture are similar to the values of a known standard.

R Script

```
## Loading Packages

install.packages("ALS")
library(ALS)

## Chromatography elution profiles - 2 components (2 replicates)

data(multiex)

matplot(x, Cstart1, type="l", xlab = "Elution time", ylab = "Intensity", main =
"Chromatographic components")

matplot(x, Cstart2, type="l", add =TRUE)
```

```

## MCR-ALS

dimS = dim(S) # dimensions of mass spectra array

mcr <- als(CList=list(Cstart1,Cstart2), S=matrix(1, nrow=dimS[1], ncol=2),
PsiList=list(d1,d2), x=x, x2=x2, uniC=TRUE, normS =0)

# MCR-ALS with unimodality constraint (uniC = TRUE) and normalization (normS = 0)

## Plot of recovered mass components

plots(mcr$S,x2)

## comparing the mass spectrum of the known standard S with the recovered profiles
mcr$S (fit rate)

matchFactor(S[,1], mcr$S[,1])
matchFactor(S[,2], mcr$S[,2])

# Copt values (estimated elution values)

matplot(x, mcr$CList[[1]], type="l")

matplot(x, mcr$CList[[2]], type="l", add = TRUE)

```

SUPERVISED ANALYSIS

Supervised pattern recognition is a machine learning method that uses techniques to identify similarities and differences between different types of samples. Pattern recognition techniques are based on the following assumptions: i) Samples of the same type are similar; and, ii) The classes are defined by the system designer.

In general, supervised algorithms present a classifier, also called a model, that will be able to predict the class of a new set of data. The classifier produced can also be described as a function f , which takes a given x and provides a prediction y .

3.6 KNN (K-Nearest Neighbors)

In 1951, Evelyn Fix and Joseph Hodges [5] described the fundamentals of classifying unknown data points based on classes of nearest points. In 1967, Thomas Cover and Peter Hart [6] explored and addressed the concept of the K-Nearest Neighbors (KNN) technique through the parameter K (selection of the number of neighbors) and the choice of distance metrics to measure the proximity between data points.

The KNN algorithm basically uses a training set made up of n -dimensional vectors and each element of this set represents a point in an n -dimensional space. The KNN

methodology is based on three fundamental steps: i) determining the distance between a new sample and the other samples in the training set; ii) identify the K closest samples or with the most similar characteristics; iii) with the k known elements of k-nearest neighbors, the closest class will be assigned to the class of the unknown element.

Example 9: Analysis of a data set to perform KNN extracted from the *DAAG* and *Caret* packages.

R Script

```
# reading data

install.packages("DAAG")
library(DAAG)

data("leafshape")
?leafshape
data = leafshape
data = na.omit(data)
data = data[,-c(1:3,9)]

set.seed(2)

#### K-fold cross validation via caret #####

install.packages("caret")
library(caret)

# defining the number of partitions ( folds )
trControl = trainControl(method = "CV",
                          number = 10)

# KNN CV
knn.cv = train(as.factor(arch) ~ .,
               method = "knn",
               tuneGrid = expand.grid(k=seq(5,95, by = 10)), ## k = 1:100),
               trControl = trControl,
               metric = "Accuracy",
               data = data)

knn.cv
plot(knn.cv)

# plotting
pairs(data, col = rainbow(2)[as.factor(data$arch)])

#####

# separating training and testing data
set.seed(13)
```

```

tr = round(0.5* nrow (data))
training = sample(nrow(data), tr, replace = F)

data.training = data[training,]
data.test = data[-training,]

x.training = data.training[,-5]
x.test = data.test[,-5]
y.training = data.training[,5]
y.test = data.test[,5]

#####
##### knn for k = 5 neighbors

library(class)

knn5 = knn(x.training, x.test, y.training , k = 5)

# confusion matrix
table(knn5, y.test)

mean(knn5 == y.test)

### Preview

grid = expand.grid(logwid = seq(min(data$logwid),
                               max(data$logwid), length = 200),
                  logpet = seq(min(data$logpet),
                               max(data$logpet), length = 200))
grid$class = knn (x.training [,2:3], grid, y.training , k = 5)

### Training

ggplot ( ) +
  geom_raster ( aes (x= grid$logwid , y = grid$logpet , fill = grid$class ),
              alpha = 0.3, interpolate = T) +
  geom_point ( aes (x = data.training$logwid , y = data.training$logpet ,
                   color = as.factor ( data.training$arch ),
                   shape = as.factor ( data.training$arch )), size = 2) +
  labs( x = "logwid", y = "logpet",
        col = "arch", shape = "arch", fill = "arch") + theme_bw ( )

#### Test

```

```

ggplot ( ) +
  geom_raster ( aes ( x= grid$logwid , y = grid$logpet , fill = grid$class ),
               alpha = 0.3, interpolate = T) +
  geom_point ( aes ( x = data.test$logwid , y = data.test$logpet ,
                   color = as.factor ( data.test$arch ),
                   shape = as.factor ( data.test$arch )), size = 2) +
  labs( x = "logwid", y = "logpet",
        col = "arch", shape = "arch", fill = "arch") + theme_bw ( )

```

3.6 Linear Discriminant Analysis (LDA)

Linear discriminant analysis (LDA) was introduced in its initial form by Ronald Fisher in 1936 [6] for two classes, as a method employed for solving classification problems, dimensionality reduction and data visualization. In 1948, CR Rao [7] proposed a generalization to multiple classes. Basically, the LDA algorithm tries to find a linear transformation by maximizing the inter-class distance and minimizing the intra-class distance. The method seeks to find the best direction, so that when data are projected onto a plane, classes can be separated.

For a univariate case, we have two classes: class 1: $x \sim N(\mu_1, \sigma_2)$ and class 2: $x \sim N(\mu_2, \sigma_2)$. The probability ratio $\lambda(x)$ that indicates the density ratio to classify a given sample (x) into one of the classes can be written as:

$$\lambda_x = \frac{f_1(x)}{f_2(x)} \tag{Eq. 6}$$

Assuming that f_1 and f_2 are normal distribution densities, we can rewrite the probability ratio as:

$$\lambda_x = \exp \left\{ -\frac{1}{2} \left[\left(\frac{x-\mu_1}{\sigma} \right)^2 - \left(\frac{x-\mu_2}{\sigma} \right)^2 \right] \right\} \tag{Eq. 7}$$

The quality of discrimination will depend on the degree of intersection of the two densities. Therefore, the functions $\lambda(x)$ and $-2\log\lambda(x)$ are called discriminant functions and have the following properties, as exemplified in Table 1 below:

Table 3.1: Properties of LDA discriminant functions.

$\lambda(x)$	$-2 \log \lambda(x)$	Situation
>1	<0	x closest to μ_1
<1	>0	x closest to μ_2
$=1$	$=0$	x equally close to μ_1 and μ_2

For a multivariate case, in which we have p variables, we call $X \sim N(\mu_1, \Sigma_1)$ for class 1 and $X \sim N(\mu_2, \Sigma_2)$ for class 2, we can rewrite the discriminant function as being:

$$-2 \log \lambda_{(x)} = \log \left[\frac{(2\pi)^{p/2} |\Sigma_1|^{-1/2} \exp\{-\frac{1}{2}(x-\mu_1)^T \Sigma_1^{-1} (x-\mu_1)\}}{(2\pi)^{p/2} |\Sigma_2|^{-1/2} \exp\{-\frac{1}{2}(x-\mu_2)^T \Sigma_2^{-1} (x-\mu_2)\}} \right] \quad \text{Eq. 8}$$

where Σ represents the covariance matrix.

Equation 3.8 can be rewritten as:

$$-2 \log \lambda_{(x)} = (x - \mu_1)^T \Sigma_1^{-1} (x - \mu_1) - (x - \mu_2)^T \Sigma_2^{-1} (x - \mu_2) + \log |\Sigma_1| - \log |\Sigma_2| \quad \text{Eq. 9}$$

Thus, we can classify x in class 1 if $-2 \log \lambda_{(x)} < 0$ and in class 2 if $-2 \log \lambda_{(x)} > 0$. The classification rule for the Fisher discriminant function is met when $\Sigma_1 = \Sigma_2 = \Sigma$. For this reason we have:

$$fd(x) = (\mu_1 - \mu_2^T) \Sigma^{-1} x - \frac{1}{2} (\mu_1 - \mu_2^T) \Sigma^{-1} (\mu_1 + \mu_2) \quad \text{Eq. 10}$$

A sample element with observation vector x would be classified in class 1 if $fd(x) > 0$ and would be classified in class 2 if $fd(x) < 0$.

Example 10: Analysis of a data set to perform linear discriminant analysis (LDA) using R packages called *datasetsICR*, *dplyr*, *GGally*, *DFA.CANCOR*, *heplots*, *MVM* and *MASS*.

R Script

```
# Loading packages and data

install.packages("datasetsICR")
library(datasetsICR)

data("seeds")
?seeds

data = seeds

head(data)
levels(data$variety)
library(dplyr)
glimpse(data)
#####
```



```

# plotting
pairs(data, col = rainbow(3)[data$variety])

library(ggplot2)
install.packages("GGally")
library(GGally)

ggpairs(data, aes(color = variety, alpha = 0.5)) + theme_bw ()

#####

### Separating training and validation data

set.seed(1)

tr = round(0.7* nrow(data))
training = sample(nrow(data), tr, replace = F)
training

data.training = data[training,]
data.test = data[-training,]

#####

#Assumptions

# Homogeneity of variance/covariance matrices

install.packages("DFA.CANCOR")
library(DFA.CANCOR)

HOMOGENEITY(data.training,group='variety',
             variables = c('area','perimeter'))
HOMOGENEITY(data.test,group='variety',
             variables = c('area', 'perimeter', 'compactness','length of kernel',
                           'width of kernel',
                           'asymmetry coefficient', 'length of kernel groove'))

# H0  p>0.05
# H1  p<0.05

# Another homogeneity test
library(heplots)
boxM(data.training [,1:7], data.training$variety )

# Multivariate normality

library(MVN)

```

```

mvn(data.training, subset = "variety")

#####

#Linear discriminant analysis (LDA)

library(MASS)

#Model 1
fit.lda1 = lda(variety ~ compactness+perimeter, data.training)
fit.lda1

#prediction for training data
lda.pred1= predict(fit.lda1)

#Graphics
plot(fit.lda1)
pairs(fit.lda1)

ldahist(lda.pred1$x[,1], g = lda.pred1$class)

# Confusion matrix
cm1 = table(data.training$variety, lda.pred1$class)
cm1

#Prediction ability
mean(data.training$variety == lda.pred1$class)

#plotting classes in new directions
d.plot = data.frame(Class = data.training$variety, lda = lda.pred1$x)

library(ggplot2)

grid = expand.grid (compactness = seq(min(data$compactness),
                                     max(data$compactness), length = 200),
                  perimeter = seq(min(data$perimeter),
                                   max(data$perimeter), length = 200))

grid$class = predict(fit.lda1,grid)$class

ggplot() +
  geom_raster(aes(x=grid$compactness, y = grid$perimeter, fill = grid$class),
             alpha = 0.3, interpolate = T) +
  geom_point(aes(x = data.training$compactness, y = data.training$perimeter,
                color = as.factor(data.training$variety),
                shape = as.factor(data.training$variety)), size = 2) +
  labs(x = "compactness", y = "perimeter",
       col = "variety", shape = "variety", fill = "variety") + theme_bw()

d.plot = data.frame(Class = data.training$variety, lda = lda.pred1$x)

```

```

ggplot(d.plot, aes(lda.LD1, lda.LD2, group = Class)) +
  geom_point(aes(col = Class), size = 2.5) +
  stat_ellipse(aes(fill = Class), geom = "polygon", alpha = .3) +
  theme_bw()

### Model Test 1

# prediction for test data
lda.pred1.t = predict(fit.lda1, data.test)

# confusion matrix
cm1.t = table(data.test$variety, lda.pred1.t$class)
cm1.t

# prediction ability
mean(data.test$variety == lda.pred1.t$class)

# plotting classes in new directions

d.plot.t = data.frame(Class = data.test$variety, lda = lda.pred1.t$x)

ggplot(d.plot.t, aes(lda.LD1, lda.LD2, group = Class)) +
  geom_point(aes(col = Class), size = 2.5) +
  stat_ellipse(aes(fill = Class), geom = "polygon", alpha = .3) +
  theme_bw()

ggplot() +
  geom_raster(aes(x=grid$compactness, y = grid$perimeter, fill = grid$class),
             alpha = 0.3, interpolate = T) +
  geom_point(aes(x = data.test$compactness, y = data.test$perimeter,
                 color = as.factor(data.test$variety),
                 shape = as.factor(data.test$variety)), size = 2) +
  labs(x = "compactness", y = "perimeter",
       col = "variety", shape = "variety", fill = "variety") + theme_bw()

#####

# Training model 2

# Model 2
fit.lda2 = lda( variety ~., data.training )
fit.lda2

# prediction for training data
lda.pred2 = predict(fit.lda2)

plot(fit.lda2)
pairs(fit.lda2)

```

```

# Confusion matrix
cm2 = table(data.training$variety, lda.pred2$class)
cm2

#Prediction ability
mean(data.training$variety == lda.pred2$class)

# plotting classes in new directions

library(ggplot2)
d.plot2 = data.frame(Class = data.training$variety, lda = lda.pred2$x)
ggplot(d.plot2, aes(lda.LD1, lda.LD2, group = Class)) +
  geom_point(aes(col = Class), size = 2.5) +
  stat_ellipse(aes(fill = Class), geom = "polygon", alpha = .3) +
  theme_bw()

#####

install.packages ("klaR")
library (klaR)

data.training2 = data.training
colnames(data.training2) = c("x1", "x2", "x3", "x4", "x5", "x6", "x7", "variety")
dev.new ( )
partimat(variety ~., data = data.training2, method = "lda")

#####

# Test model 2

# prediction for test data
lda.pred2.t = predict(fit.lda2, data.test)

# confusion matrix
cm2.t = table(data.test$variety, lda.pred2.t$class)
cm2.t

# prediction ability
mean(data.test$variety == lda.pred2.t$class)

# prediction
head(lda.pred2.t$class,10)

# predicted probabilities for classes
head(lda.pred2.t$posterior,10)

# linear discriminants
head(lda.pred2.t$x,10)

#####

```

3.7 Quadratic Discriminant Analysis (QDA)

In quadratic discriminant analysis (QDA), there is no assumption that classes have equal covariance matrices. As in linear discriminant analysis, an observation is classified into the group with the smallest squared distance. However, the squared distance does not result in a linear function, hence the name quadratic discriminant analysis.

In QDA, for each of the classes y , the covariance arrangement is given by:

$$\Sigma_y = \frac{1}{N_y - 1} \sum_{y_i=y} (x_i - \mu_y)(x_i - \mu_y)^T \quad \text{Eq. 11}$$

Taking the log for both sides of the above equation, the quadratic discriminant function will be given by:

$$fd(x) = \log \pi - \frac{1}{2} \mu^T \Sigma^{-1} \mu + x^T \Sigma^{-1} \mu - \frac{1}{2} x^T \Sigma^{-1} x - \frac{1}{2} \log |\Sigma| \quad \text{Eq. 12}$$

A sample element with observation vector x would be classified in class 1 if $fd(x) > 0$ and would be classified in class 2 if $fd(x) < 0$.

Example 11: Analysis of a data set to perform quadratic discriminant analysis (QDA) using R packages called *datasetsICR*, *dplyr*, *GGally*, *DFA.CANCOR*, *heplots*, *MVM* and *MASS*.

R Script

```
## Loading packages and data

install.packages("datasetsICR")
library(datasetsICR)
install.packages("klaR")
library(klaR)

# Dataset - seeds
data("seeds")
?seeds

data = seeds
```

```

head(data)
levels(data$variety)
library(dplyr)
glimpse (data)
#####

# plotting
pairs(data, col = rainbow(3)[data$variety])

library(ggplot2)
install.packages ("GGally")
library(GGally)

ggpairs(data, aes(color = variety, alpha = 0.5)) + theme_bw ()

#####

### Separating training and validation data

set.seed (1)

tr = round(0.7* nrow (data))
training = sample(nrow (data), tr , replace = F)
training

data.training = data[training,]
data.test = data[-training,]

#####
#Assumptions

# Homogeneity of variance/covariance matrices

install.packages("DFA.CANCOR")
library(DFA.CANCOR)

HOMOGENEITY(data.training,group='variety',
             variables = c('area','perimeter'))
HOMOGENEITY(data.training,group='variety',
             variables = c('area', 'perimeter', 'compactness','length of kernel',
                           'width of kernel',
                           'asymmetry coefficient', 'length of kernel groove'))

# H0  p>0.05
# H1  p<0.05

# Another homogeneity test
library(heplots)
boxM(data.training[,1:7], data.training$variety)

```

```

# Multivariate normality

library(MVN)

mvn(data.training, subset = "variety")

#####

#Quadratic linear discriminant analysis (QDA)

library(MASS)

#training model 1
fit.qda1 = qda(variety~compactness+perimeter, data.training )
fit.qda1

#prediction for training data
qda.pred1=predict(fit.qda1)

# Confusion matrix
cm1 = table(data.training$variety, qda.pred1$class)
cm1

# prediction ability
mean(data.training$variety == qda.pred1$class)

# plotting classes in new directions

library(ggplot2)

grid = expand.grid(compactness = seq(min(data$compactness),
                                     max(data$compactness), length = 200),
                  perimeter = seq(min(data$perimeter),
                                   max(data$perimeter), length = 200))

grid$class = predict(fit.qda1,grid)$class

ggplot() +
  geom_raster(aes(x=grid$compactness, y = grid$perimeter, fill = grid$class),
             alpha = 0.3, interpolate = T) +
  geom_point(aes(x = data.training$compactness, y = data.training$perimeter,
                 color = as.factor(data.training$variety),
                 shape = as.factor(data.training$variety)), size = 2) +
  labs(x = "compactness", y = "perimeter",
       col = "variety", shape = "variety", fill = "variety") + theme_bw()

#####
# Test model 1

```

```

# prediction for test data
qda.pred1.t = predict(fit.qda1, data.test)

# confusion matrix
cm1.t = table(data.test$variety, qda.pred1.t$class)
cm1.t

# prediction ability
mean(data.test$variety == qda.pred1.t$class)

grid$class = predict(fit.qda1, grid)$class

ggplot() +
  geom_raster(aes(x=grid$compactness, y = grid$perimeter, fill = grid$class),
             alpha = 0.3, interpolate = T) +
  geom_point(aes(x = data.training$compactness, y = data.training$perimeter,
                color = as.factor(data.training$variety),
                shape = as.factor(data.training$variety)), size = 2) +
  labs(x = "compactness", y = "perimeter",
       col = "variety", shape = "variety", fill = "variety") + theme_bw()

#####

```

3.8 Support Vector Machines (SVM)

Support Vector Machines (SVM) originated in the studies of Vapnik and Chervonenkis [8] in 1971. Essentially, the SVM is responsible for finding the best possible separation boundary between classes for a given data set that are linearly separable. For SVM, the various possible separation boundaries that are capable of completely separating classes are called hyperplanes. In this way, SVM seeks to find the best hyperplane for a given data set whose classes are linearly separable.

Immediately, we can observe that the dimensionality of the hyperplane is directly proportional to the dimension of the data set (n):

$$h_{dim} = n - 1 \qquad \text{Eq. 13}$$

Therefore, in a two-dimensional dataset, the hyperplane is a straight line. In a three-dimensional dataset, the hyperplane is in fact a plane. And so on. The hyperplane is located at the midpoint between the two groups of classes, providing a characteristic of symmetry in the classification, in which the closest point of each class is at a distance d from the hyperplane, in order to minimize classification errors and problems of model bias (overfitting). The points closest to the hyperplane are called support vectors, giving

the algorithm its name, as it is from them that the model will be mathematically developed, trained and optimized. The distance between the support vectors and the hyperplane is called the margin.

The equation of a hyperplane is presented in equation 3.14, in which $\mathbf{w} \cdot \mathbf{x}$ is the dot product between the vectors \mathbf{w} and \mathbf{x} , $\mathbf{w} \in X$ is the normal vector to the described hyperplane and $\frac{b}{\|\mathbf{w}\|}$ corresponds to the distance of the hyperplane in relation to the origin, with $b \in \mathfrak{R}$.

$$f(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} + b = 0 \quad \text{Eq. 14}$$

A signal function $g(x) = \text{sgn}(f(x)) = \begin{cases} +1 & \text{se } \mathbf{w} \cdot \mathbf{x} + b > 0 \\ -1 & \text{se } \mathbf{w} \cdot \mathbf{x} + b < 0 \end{cases}$ is finally written in obtaining classifications by the SVM algorithm.

Example 12: Analysis of a data set to perform SVM using R packages called *mvtnorm* and *e1071*.

R Script

```
#### Simulating data
### Parameters to simulate data
library(mvtnorm)
set.seed(14)
m= c(0,0) # vector of means
S= matrix(c(1, 0.2, 0.2,1),2) # covariance matrix

# Simulating data
data = rmvnorm(1000, mean = m, sigma = S)

# Calculating distance to separate classes
S2 = matrix(c( 1,-0.5,-0.5,1),2)
dist = mahalanobis(data, c(0,0), S2)
dist

# transforming into data.frame
data = data.frame(data)
colnames(data) = c("x1", "x2")

#defining column with classes
data$y = ifelse(dist + rnorm(nrow(data), sd = 0.3) < 1.7, 1, -1)
data$y = as.factor((data$y))

#plotting data
library(ggplot2)
ggplot(data, aes( x = x1, y = x2, group = y ))+
  geom_point( aes (color=y) ) + theme_bw ( )
```

```

summary( data$y )

#####
#Separating training and testing data

set.seed(1)

tr = round( 0.7* nrow (data))
training = sample(nrow (data), tr , replace = F)

data.training = data[training,]
data.test = data[-training,]

#####
#Library for SVM and other ML methods
library(e1071)

# support vector classifier 1 (linear)

svc1= svm(y~., data = data.training, kernel = "linear", cost = 10, scale = FALSE)

plot(svc1,data.training)
svc1$index # support vectors
summary(svc1) # analysis summary

# cross validation to define parameter c (cost - limit for slack variables)
set.seed(1)
tune.out = tune(svm , y~., data = data.training, kernel = "linear",
               ranges = list (cost = c( 0.001, 0.01, 0.1, 1, 10, 100, 1000)))
summary( tune.out )

# support vector classifier 1 (linear)

svc1 = svm( y~ ., data = data.training, kernel = "linear", cost = 100, scale = FALSE)

# confusion matrix - test data
cm = table(true = data.test[, "y"], pred = predict( tune.out$best.model , newdata =
data.test ))
cm

# test classification ratio
(cm[ 1,1] + cm[2,2])/sum(cm)

#####

### Preview
grid = expand.grid (x1 = seq(min(c(data.training$x1, data.test$x1)),
                           max(c(data.training$x1, data.test$x1)), length = 200),
                  x2 = seq(min(c(data.training$x2, data.test$x2)),
                           max(c(data.training$x2, data.test$x2)), length = 200))

```

```

grid$class = predict(svc1, grid)

##Training
ggplot() +
  geom_raster(aes(x=grid$x1, y=grid$x2, fill=grid$class),
             alpha = 0.3, interpolate = T) +
  scale_fill_brewer(palette = "Dark2", drop = FALSE)+
  geom_point(aes(x = data.training$x1, y = data.training$x2,
                color = data.training$y,
                shape = data.training$y), size = 2) +
  scale_color_brewer(palette = "Dark2") +
  labs(x = "x1", y = "x2", col = "class",
       shape = "class", fill = "class") + theme_bw()

## Test
ggplot() +
  geom_raster(aes(x=grid$x1, y=grid$x2, fill=grid$class),
             alpha = 0.3, interpolate = T) +
  scale_fill_brewer(palette = "Dark2", drop = FALSE)+
  geom_point(aes(x = data.test$x1, y = data.test$x2,
                color = data.test$y,
                shape = data.test$y), size = 2) +
  scale_color_brewer(palette = "Dark2") +
  labs(x = "x1", y = "x2", col = "class",
       shape = "class", fill = "class") + theme_bw()

#####

# SVM with Radial Kernel
svm1 = svm(y~ .,data = data.training, kernel = "radial", cost = 10, gamma = 0.1,
scale = FALSE)

plot(svm1, data.training)
svm1$index
summary(svm1)

# cross validation to define ce gamma
set.seed(1)
tune.out = tune(svm , y~., data = data.training, kernel = "radial",
               ranges = list (cost = c( 0.001, 0.01, 0.1, 1, 10, 100),
                             gamma = c( 0.5, 1,2,3,4)))
summary(tune.out)

# Model with greater gamma
svm1 = svm( y~ ., data = data.training, kernel = "radial", cost = 10, gamma = 0.5,
scale = FALSE)

```

```

# Confusion matrix - test

cm = table(true = data.test[,"y"], pred = predict(tune.out$best.model, newdata =
data.test))
cm

# test classification ratio

(cm[ 1,1] + cm[2,2])/sum(cm)

#####

### Preview
grid = expand.grid (x1 = seq (min(c(data.training$x1, data.test$x1)),
                        max ( c(data.training$x1, data.test$x1)), length = 200),
                  x2 = seq ( min(c(data.training$x2, data.test$x2)),
                        max ( c(data.training$x2, data.test$x2)), length = 200))

grid$class = predict(svm1, grid)

##Training
ggplot() +
  geom_raster(aes(x=grid$x1, y=grid$x2, fill=grid$class),
             alpha = 0.3, interpolate = T) +
  scale_fill_brewer(palette = "Set1", drop = FALSE)+
  geom_point(aes(x = data.training$x1, y = data.training$x2,
                color = data.training$y,
                shape = data.training$y), size = 2) +
  scale_color_brewer(palette = "Set1") +
  labs(x = "x1", y = "x2", col = "class",
       shape = "class", fill = "class") + theme_bw()

## Test
ggplot() +
  geom_raster(aes(x=grid$x1, y=grid$x2, fill=grid$class),
             alpha = 0.3, interpolate = T) +
  scale_fill_brewer(palette = "Set1", drop = FALSE)+
  geom_point(aes(x = data.test$x1, y = data.test$x2,
                color = data.test$y,
                shape = data.test$y), size = 2) +
  scale_color_brewer(palette = "Set1") +
  labs(x = "x1", y = "x2", col = "class",
       shape = "class", fill = "class") + theme_bw()

#####

# ROC curve
install.packages("ROCR")
library(ROCR)

```

```

##### ROC model SVC
fitted1 = attributes(predict(svc1, data.test, decision.values = T))$decision.values
pred1 = prediction(fitted1, data.test$y )
perf1 = performance(pred1, "tpr", "fpr")

plot( perf1,
      avg = 'vertical',
      lwd = 3, main = "ROC curve model scv1 - test data",
      col = 'blue')

##### ROC model SVM
fitted2 = attributes(predict(svm1, data.test, decision.values = T))$decision.values
pred2 = prediction(fitted2, data.test$y)
perf2 = performance(pred2, "tpr", "fpr")

plot ( perf2,
      avg = 'vertical',
      lwd = 3, main = "ROC curve model svm1 - test data" ,
      col = 'green3')

#####

# SVM with polynomial kernel of degree 2

svm2 = svm (y~.,data = data.training, kernel = "polynomial", cost = 10, degree =
2, scale = FALSE)

plot( svm2, data.training )
svm2$index
summary(svm2)

# cross validation to define c
set.seed(1)
tune.out = tune( svm, y~., data = data.training, kernel = "polynomial", degree = 2,
                ranges = list(cost = c( 0.001, 0.01, 0.1, 1, 10, 100)))
summary( tune.out )

# Model with great c
svm2 = svm( y~ ., data = data.training, kernel = "polynomial", cost = 100, degree
= 2, scale = FALSE)

# Confusion matrix - test

cm = table(true = data.test[, "y"], pred = predict(tune.out$best.model, newdata =
data.test))
cm

```

```

# test classification ratio

(cm[ 1,1] + cm[2,2])/sum(cm)

#####

### Preview
grid = expand.grid (x1 = seq (min(c(data.training$x1, data.test$x1)),
                             max(c(data.training$x1, data.test$x1)), length = 200),
                  x2 = seq (min(c(data.training$x2, data.test$x2)),
                             max(c(data.training$x2, data.test$x2)), length = 200))

grid$class = predict(svm2, grid)

##Training
ggplot() +
  geom_raster(aes(x=grid$x1, y=grid$x2, fill=grid$class),
             alpha = 0.3, interpolate = T) +
  scale_fill_brewer(palette = "Set1", drop = FALSE)+
  geom_point(aes(x = data.training$x1, y = data.training$x2,
                color = data.training$y,
                shape = data.training$y), size = 2) +
  scale_color_brewer(palette = "Set1") +
  labs(x = "x1", y = "x2", col = "class",
       shape = "class", fill = "class") + theme_bw()

## Test
ggplot() +
  geom_raster(aes(x=grid$x1, y=grid$x2, fill=grid$class),
             alpha = 0.3, interpolate = T) +
  scale_fill_brewer(palette = "Set1", drop = FALSE)+
  geom_point(aes(x = data.test$x1, y = data.test$x2,
                color = data.test$y,
                shape = data.test$y), size = 2) +
  scale_color_brewer(palette = "Set1") +
  labs(x = "x1", y = "x2", col = "class",
       shape = "class", fill = "class") + theme_bw()

#####

##### ROC model SVM2
fitted3 = attributes(predict(svm2, data.test, decision.values = T))$decision.values
pred3 = prediction(fitted3, data.test$y )
perf3 = performance(pred3, "tpr", "fpr")

plot ( perf3,
      avg = 'vertical',
      lwd = 3, main = "ROC curve model svm2 - test data" ,
      col = 'green3')

#####

```

3.10 Decision Trees

The decision tree concept was developed by J. Ross Quinlan in 1975 [9]. Decision trees learning is one of the predictive modeling approaches that can be found in statistics, data mining, or machine learning. Basically, a decision tree is used (as a predictive model) to verify observations about an item (represented in the branches) and obtain conclusions about the target value of the item (represented in the leaves). In other words, a decision tree is a tree in which each internal (non-leaf) node is labeled with an input feature. Arcs coming from a node labeled with an input feature are labeled with each of the possible values of the target feature, or the arc leads to a subordinate decision node on a different input feature. Each leaf of the tree is labeled with a class or probability distribution over classes, which means that the data set has been classified by the tree into a specific class or a specific probability distribution (which, if the decision tree is well constructed, is biased towards certain subsets of classes).

The way the algorithm will know how to build the tree is based on conditions that minimize entropy and increase information gain. Entropy is the measure that tells us how disorganized and mixed the original data is. The higher the entropy, the lower the information gain and vice versa. The entropy value of a data can be calculated using the following equation:

$$H(x) = - \sum_{i=1}^n p(x_i) \log_2 p(x_i) \quad \text{Eq. 15}$$

Entropy usually varies between 0 and our number of classes -1, assuming its maximum value when the probabilities of each class occur. The goal with a decision tree is to achieve the lowest entropy possible.

Example 13: Analysis of a data set to create the decision tree using R packages called *rpart.plot*, *caret*, *Amelia*, *pROC*.

R Script

```
# Loading and Separating training and testing data
train <- read.csv("trainTitanic.csv", header = T)
test <- read.csv("testTitanic.csv", header = T)

# Tidying up the dataset
train$X <- NULL
test$X <- NULL

# Check the encoding of variables
str( train )
str( test )
```

```

# Transforming the variables that need to be categorical
train$Survived <- as.factor( train$Survived )
test$Survived <- as.factor( test$Survived )

train$Pclass <- as.factor( train$Pclass )
test$Pclass <- as.factor( test$Pclass )

train$Sex <- as.factor( train$Sex )
test$Sex <- as.factor( test$Sex )

train$Embarked <- as.factor( train$Embarked )
test$Embarked <- as.factor( test$Embarked )

# Variables
# Survived : 0 = No, 1 = Yes
# SibSp : Number of siblings/spouses on board
# Parch : Number of parents/children on board
# Fare: Fare
# Embarked : Port of embarkation C = Cherbourg , Q = Queenstown , S = Southampton
# Pclass : Ship class

### Parameter Adjustment ###

# Loading packages
install.packages("caret")
install.packages("Amelia")
install.packages("pROC")
library(caret)
library(Amelia)
library(pROC)

# Defining the seed
set.seed(123)

# Let's use a 10-fold cross-validation
ctrl <- trainControl ( method = "cv",
                      number = 10,
                      summaryFunction = twoClassSummary ,
                      classProbs = TRUE)

# We have to change the variable - when we use twoClassSummary
levels( train$Survived ) <- c("M", "S")
levels( test$Survived ) <- c("M", "S")

dtFit <- train( Survived ~ .,
              method = "rpart2", # uses depth maximum
              tuneLength = 20,
              trControl = ctrl,
              metric = "ROC",
              data = train)

dtFit
plot(dtFit)

```



```

# Tree drawing
install.packages("rpart.plot")
library(rpart.plot)
rpart.plot ( dtFit$finalModel,
             cex = 0.7,
             extra = 4,
             type = 1,
             box.palette = "RdYlGn")

### Predictions ###

predtdt <- predict(dtFit, test, type = "prob")
resultdt <- as.factor(ifelse(predtdt [,2] > 0.5, "S", "M"))

### Model performance ###

# Confusion matrix and measurements
library(caret)
confusionMatrix(resultdt, test$Survived , positive = "S")

# ROC curve and AUC
library(pROC)
aucdt <- roc(test$Survived, predtdt[,2])
plot.roc(aucdt, print.thres = T) # find the cutoff point that provides the best sum
of S and E

# Using the new cutoff point
resultdt2 <- as.factor(ifelse(predtdt[,2] > 0.393, "S", "M"))
confusionMatrix(resultdt2, test$Survived, positive = "S")

```

PROPOSED EXERCISES

01 – Propose the application of the HCA algorithm through a script in the R language on an experimental data set, presenting your hypotheses and conclusions.

02 – There are multivariate data repositories on the internet (web of science, science direct and others) in which you must choose a dataset to use the K-means algorithm in the R language and carry out a statistical study in detail. Present the main results and conclusions.

03 – Present a script in the R language for the PCA algorithm for a dataset and demonstrate your hypotheses and main conclusions.

04 – Propose a script in the R language for the KNN algorithm using a given dataset and perform a statistical interpretation presenting its main conclusions.

05 – Present a script for the LDA and QDA algorithms in the R language for a given dataset and present your main results.

06 – Like the previous exercise, present a script in the R language to build a decision tree algorithm for a dataset. Present your conclusions.

REFERENCES

- 1 – Tryon, RC (1939), Cluster analysis: correlation profile and orthometric (factor) analysis for the isolation of units in mind and personality, Ann Arbor, Mich: Edwards brothers, inc., lithoprinters and publishers.
- 2 – Lloyd, Stuart P. (1982). Least squares quantization in PCM. IEEE Transactions on Information Theory. 28 (2): 129–137.
- 3 – Pearson, K. (1901). On Lines and Planes of Closest Fit to Systems of Points in Space. Philosophical Magazine. 2(6):559–572.
- 4 – Lawton, WH; Sylvestre, EA; (1971). Self modeling curve resolution. Technometrics , 13: 617-633.
- 4 – Fix, E.; Jr., J. L. H. (1951). Discriminatory Analysis. Nonparametric Discrimination: Consistency Properties. International Statistical Review. 57(3): 238-247.
- 5 – Cover, T.; Hart, P. (1967). Nearest neighbor pattern classification. IEEE Transactions on Information Theory , 13(1): 21-27.
- 6 – Fisher, RA (1936). The Use of Multiple Measurements in Taxonomic Problems. Annals of Eugenics. 7 (2): 179–188.
- 7 – Rao, CR (1948) The Use of Multiple Measurements in Problems of Biological Classification. Journal of the Royal Statistical Society: Series B, 10, 159-203.
- 8 – Vapnik , VN; Chervonenkis , A. Y. (1971). On the uniform convergence of relative frequencies of events to their probabilities. Theory of Probability and its Applications, 16(2):283–305.
- 9 – Quinlan, JR (1975). Machine Learning, vol. 1, no 1.

HIGHER ORDER MULTIVARIATE CLASSIFICATION



"Model building is the art of selecting those aspects of a process that are relevant to the question being asked. As with any art, this selection is guided by taste, elegance, and metaphor; it is a matter of induction, rather than deduction. High science depends on this art." John Henry Holland (1929-2015)

CHAPTER IDEA

According to the amount of information generated per sample, analytical data can be categorized into 0th order, 1st order, 2nd order, 3rd order and 4th order. We obtain for each sample analyzed in 0th order (a scalar), in 1st order (a vector), in 2nd order (a matrix), in 3rd order (3 ways) and 4th order (4 ways). In this chapter, we will explore some multivariate classification algorithms that are typically employed on 1st and 2nd order data in real datasets. In addition, some methods for selecting samples and variables in multivariate classification will be presented.

Upon completing the chapter, you should be able to:

- a) Apply the main 1st order multivariate classification algorithms coupled with variable selection methods (PCA-LDA, SPA-LDA, GA-LDA) to a set of real data seeking to build models and analyze them.
- b) Incorporate QDA models into PCA, SPA and GA algorithms.
- c) Understand the sample and variable selection algorithms used in multivariate classification models.
- d) Apply the main 2nd order multivariate classification algorithms (Turkey-3 and PARAFAC) using the sample selection algorithms (KS and MLM) and the two classifiers (LDA and QDA) on real or simulated data sets seeking to build models and analyze them.
- e) Investigate the stages of building multivariate classification models when applied to variable selection algorithms and model performance assessment (figures of merit).
- f) Build new scripts in R language for decision making using 1st and 2nd order multivariate classification.
- g) Propose new applications in chemistry or related areas of multivariate classification techniques.

4.1 Types of analytical data

According to the amount of information generated per sample [1], the analytical data can be divided as shown in Figure 4.1 below:

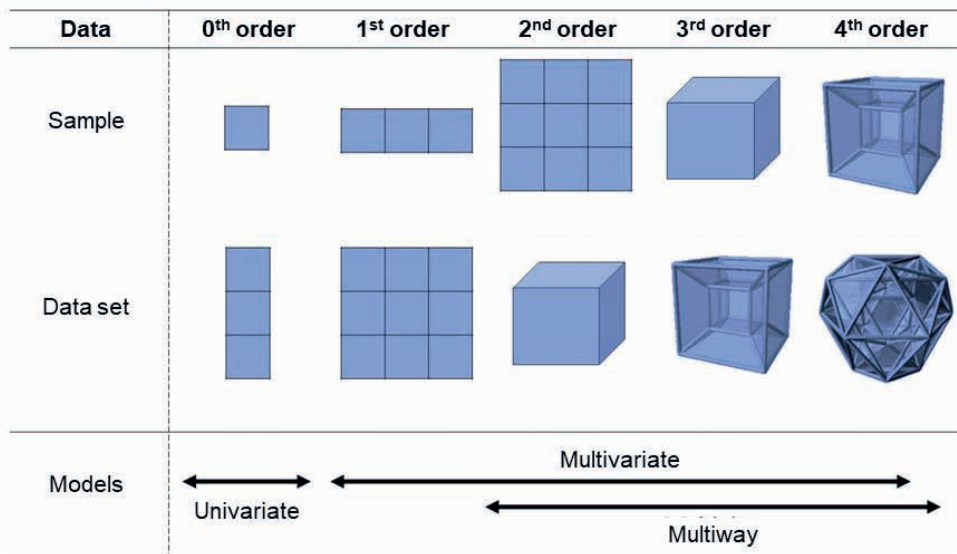


Figure 4.1: Types of analytical data

The 1st order or 2-way data results in a vector of information per sample resulting in a two-dimensional matrix \mathbf{X} . In this type of data arrangement, we have the 1st order advantage which consists of the ability to build multivariate models in the presence of interferers as long as they are present in the training samples. The algorithms used in 1st order data are based on bilinear models that consist of interpreting an instrumental response function $X(r_1, r_2)$ as a product of two independent functions $X_1(r_1)$ and $X_2(r_2)$. Bilinear models are obeyed when the mathematical rank is equal to the chemical rank.

In 2nd order or 3-way data, we have as a response a matrix (second order tensor) of data for each sample and when the data matrices are placed side by side they generate a parallelepiped. Here we have the 2nd order advantage which consists of the possibility of quantifying the analyte in the presence of interferers even if these interferers are not present in the training set. Furthermore, 2nd order algorithms use a reduced number of samples in the training set since potential interferents do not need to be modeled. The algorithms used in 2nd order models use the concept of trilinearity, which consists of a generalization of bilinearity for a three-way arrangement ($l \times j \times k$). Rank deficiency due to the presence of highly correlated spectral profiles can be a source of trilinearity breakdown .

4.2 Methods for selecting samples in multivariate classification

One of the limitations in building multivariate classification models (1st or 2nd order) is the appropriate choice of samples that represent the greatest variance in the analyzed data, improving their predictive capacity.

In 1969, Kennard -Stone (KS) [2] proposed a sample selection algorithm dividing the original data set into two sub-sets (training and validation) so that each sub-set of samples maintains maximum data variability. This algorithm uses Euclidean distances for each pair (p, q) of samples to select the samples that will compose the training subset, according to equation 4.1:

$$d_x(p, q) = \sqrt{\sum_{j=1}^j [x_p(j) - x_q(j)]^2} \quad p, q \in [1, N] \quad \text{Eq. 1}$$

Where j represents the number of covariates and N corresponds to the sample size.

To ensure the uniformity of distribution of each subset throughout the instrumental response space, KS follows a stepwise procedure, in which a new selection is made in regions of space far from the already selected samples. With each subsequent iteration, the algorithm selects the sample that exhibits the greatest minimum distance from an already selected sample. This procedure is repeated until the number of samples specified by the analyst is reached.

The Morais-Lima-Martim (MLM) algorithm for sample selection was proposed in 2018 [3] and consists of a modification of the Kennard-Stone (KS) algorithm, in which a random mutation factor is inserted into the latter. That is, after executing the KS algorithm, some training samples are randomly selected and transferred to the validation set and *vice versa*. Usually, this number of transferred samples is 20%, that is, 80% of the validation set remains with the samples selected by KS, and the rest are samples randomly chosen from the training set. MLM proved to be superior to the KS algorithm, showing that there is a synergistic effect when combining the KS deterministic process with a small randomness when selecting training and validation samples.

4.3 Methods for selecting variables in multivariate classification

Variable selection is an important step for data analysis. This step identifies the most informative subsets of variables for building more accurate models. Basically, three main approaches for variable selection methods can be found in the literature: i) filter; ii) wrapper ; iii) embedded.

In filter-based methods, variables are evaluated considering the characteristics of their nature and normally use statistical tests of significance previously applied to a

classification algorithm. Correlation-based feature selection (CFS), Minimum Redundancy Maximum Relevance (MRMR), Information Gain and Joint Mutual Information (JMI) are some examples of filter-based variable selection methods. For wrapper-based methods, subsets of variables are evaluated using learning algorithms to find the subset that performs best. Genetic algorithm (GA), successive projection algorithm (SPA), Particle Swarm Optimization (PSO), and Simulated Annealing are some examples of variable selection methods in the wrapper approach. Finally, embedded-based methods are those that select the subset of variables during the classification model construction process itself. Least Absolute Selection (LAS), Shrinkage Operator (LASSO) and deep learning are some examples of the embedded method.

4.4 Performance metrics

Normally the performance of multivariate classification techniques is evaluated through error. The efficiency of a classification model is the ability to correctly classify samples into their respective classes. Generally, the results of the error rate made by the classifier are organized in the form of a table or confusion matrix. If a sample classified as positive by the reference or gold standard method is correctly classified by the classification model, it is considered true positive (TP). However, if it is classified as negative, we have a false negative (FN) or Type II error. In the case of a sample classified as negative by the reference method and the classification model calculates as negative, we have a true negative (TN). However, if a sample is classified as positive, it is counted as a false positive (FP), or Type I error. Table 4.1 exemplifies a confusion matrix based on the concepts of TP, FN, FP and TN for two classes.

Table 4.1 : Confusion matrix for multivariate classification models

		true class	
		A	B
predicted class	A	TP	FP
	B	FN	TN

On the other hand, there are other performance metrics that indicate the efficiency of classification models, as shown in **Table 4.2** .

Metric	Calculation
Correct classification	$\frac{TP + TN}{sample\ size}$
Sensitivity (sens)	$\frac{TP}{TP + FN}$
Specificity (spec)	$\frac{TN}{FP + TN}$
Positive predictive value (PPV)	$\frac{TP}{TP + FP}$
Negative predictive value (NPV)	$\frac{TN}{FN + TN}$
Negative likelihood ratio	$\frac{1 - sens}{spec}$
Accuracy	$\frac{TP + TN}{TP + TN + FP + FN} \times 100$
F-score (FS)	$\frac{2 \times sens \times spec}{sens + spec}$

In addition to these metrics already described, there is a statistical tool that allows evaluating the performance of a classification system: the ROC curve (Receiver Operating Characteristic). A ROC curve is a two-dimensional line graph that represents the relationship between the sensitivity and specificity of a classification model. The index that evaluates the accuracy of these graphs is the area under the curve (AUC), and the larger the area, the greater the performance of the system in question. An ideal test is one whose area under the ROC curve is equal to 1.

4.5 – PCA-LDA

Mathematically, the multivariate classification algorithm PCA-LDA (Principal Component Analysis with Linear Discriminant Analysis) is built in six main steps:

Step 1: The maximum number of k principal components (PC) is initially determined according to the \mathbf{X}_{train} training matrix ($m \times n$). k is $(m - 1)$ for $(m \geq n)$ or $(n - 1)$ for $(n \geq m)$.

Step 2: \mathbf{X}_{train} is decomposed into k PCs, which are defined by the product between the training scores vector (\mathbf{t}_{train}) and the loadings vector transposed (\mathbf{I}_{train}^T):

$$\mathbf{X}_{train} = [\mathbf{t}_{train} \mathbf{I}_{train,1}^T]_1 + [\mathbf{t}_{train} \mathbf{I}_{train,2}^T]_2 + \dots + [\mathbf{t}_{train} \mathbf{I}_{train,k}^T]_k + \mathbf{E}_{train} \quad \text{Eq. 2}$$

Step 3: The score matrix of the test set (\mathbf{T}_{test}) is calculated from the loadings matrix $\mathbf{I}_{\text{train}}$ obtained in step 2.

$$\mathbf{T}_{\text{test}} = \mathbf{X}_{\text{test}} \mathbf{I}_{\text{Train}}^T \quad \text{Eq. 3}$$

Step 4: The discrimination (D_i) of the score vector \mathbf{t}_i related to each principal component is determined and ranked in descending order of discrimination:

$$D_i = \frac{S_{Bi}}{S_{Wi}} \quad \text{Eq. 4}$$

Where S_{bi} and S_{wi} correspond to the inter and intra class dispersions for the score vector \mathbf{t}_i , respectively. Intra-class dispersion S_{wi} is defined as:

$$S_{Wi} = \sum_{j=1}^C S_{ij} \quad \text{Eq. 5}$$

Where C is the number of classes in the data set and S_{ij} is the dispersion of \mathbf{t}_i in class j and expressed as:

$$S_{ij} = \sum_{k \in I_j} [t_i^k - m_{ij}]^2 \quad \text{Eq. 6}$$

Where t_i^k corresponds to the score value \mathbf{t}_i in the n th k object, and m_{ij} corresponds to the average value of \mathbf{t}_i in class j calculated as:

$$m_{ij} = \frac{1}{n_j} \sum_{k \in I_j} t_i^k \quad \text{Eq. 7}$$

The dispersion between classes (S_{Bi}) is defined as:

$$S_{Bi} = \sum_{j=1}^C n_j [m_{ij} - m_i]^2 \quad \text{Eq. 8}$$

Where m_i is the average of all training objects for the score vector \mathbf{t}_i .

Step 5: The training set scores are used as input variables for building the LDA model. The optimal number of scores is chosen based on the smallest error obtained through cross-validation.

Step 6: The LDA model built is used to predict the classes of test samples based on their scores.

Example 1: In this example we will describe a script in the R language for building and validating multivariate classification models using the PCA-LDA and Kennard-stone algorithm for two classes (healthy and dengue, ATR-FTIR data) through the *prospectr*, *mass* and *ggplot2* packages.

R Script

```
# installing packages

install.packages ("prospectr")
install.packages ("MASS")

# reading packages

library(prospectr)
library(MASS)
library(ggplot2)

# reading the samples

class1 = read.table("DATASET/healthy.csv", header=FALSE) # samples of healthy
patients
class2 = read.table("DATASET/dengue.csv", header=FALSE) # samples from dengue
patients
cm = read.table("DATASET/cm.csv", header=FALSE) # wavelength

# class1 = healthy - control samples
# class2 = dengue - case samples

dim_class1 = dim (class1) # dimension of class1 array
dim_class2 = dim (class2) # dimension of class2 array

# transposing the data

class1t = t(class1)
class2t = t(class2)
cmt = t(cm)

# viewing data - all spectra

dev.new()
matplot(cmt, class1t, type ="l", col ="blue", xlab ="Wavenumber (cm-1)", ylab
="Absorbance", main ="Raw spectra")
matplot(cmt,class2t, type ="l", col ="red", xlab ="Wavenumber (cm-1)", ylab
="Absorbance", main ="Raw spectra", add =TRUE)
```

```

# viewing data - just the averages

dev.new()
matplot(cmt, colMeans (class1), type ="l", col ="blue", xlab ="Wavenumber (cm-1)",
ylab ="Absorbance", main ="Average spectra")
matplot(cmt,colMeans (class2), type ="l", col ="red", xlab ="Wavenumber (cm-1)",
ylab ="Absorbance", main ="Mean spectra", add =TRUE )

# scaling the data

data = rbind(class1, class2) # combining matrices one under the other
data_scal = scale(data) # scaling the data
dim_data = dim(data) # dimension of the data array

# PCA Model

data.svd = svd(data_scal) # SVD
data.scores = data.svd$u %*% diag(data.svd$d) # scores
data.loadings = data.svd$v # loadings

# PCA Variances

data.vars = data.svd$d^2 / (nrow (data)-1) # variance per PC
data.totalvar = sum(data.vars) # total variance
data.relvars = data.vars/data.totalvar # cumulative variance
variances = 100*round(data.relvars, digits = 3) # cumulative variance in %
variances[1:10] # variance in % in the first 10 PCs

# Choosing the number of PCs

par(mfrow = c( 2,2))
barplot(data.vars[1:10], main="Variance", names.arg = paste("PC", 1:10))
barplot(log( data.vars[1:10]), main="Log(variance)", names.arg = paste("PC", 1:10))
barplot(data.relvars[1:10], main="Relative Variances", names.arg = paste("PC",
1:10))
barplot(cumsum (100*data.relvars[1:10]), main="Cumulative Variance(%)", names.arg
= paste("PC", 1:10), ylim = c(0,100))

npc = 2 # number of PCs chosen to run the PCA

scores = data.scores[1:dim_data[1],1:npc] # PCA scores up to the chosen PC number
loadings = data.loadings[1:dim_data[2],1:npc] # PCA loadings up to the chosen PC
number

# Creation of the category vector: classes (1 = healthy, 2 = dengue)

group1 = rep(1,dim_class1[1]) # class 1
group2 = rep(2,dim_class2[1]) # class 2

```

```

# Selection of training and testing samples based on KS

perc = 0.7 # 70% for training

ntrain1 = ceiling(perc * dim_class1[1]) # number of training samples class 1
ntrain2 = ceiling(perc * dim_class2[1]) # number of training samples class 2

scores1 = scores[1:dim_class1[1],1:npc] # scores class 1
scores2 = scores[(dim_class1[1]+ 1):dim_data[1],1:npc] # scores class 2

sel1 = kenStone(scores1, k = ntrain1) # KS class 1
sel2 = kenStone(scores2, k = ntrain2) # KS class 2

train1 = scores1[sel1$model,1:npc] # training class 1
train2 = scores2[sel2$model,1:npc] # training class 2
train = rbind(train1,train2) # joining training matrices

group1train = matrix(group1[sel1$model]) # class 1 training labels
group2train = matrix(group2[sel2$model]) # class 2 training labels
group_train = rbind(group1train,group2train) # training labels

test1 = scores1[sel1$test, 1:npc] # test class 1
test2 = scores2[sel2$test, 1:npc] # test class 2
test = rbind(test1,test2) # joining test matrices

group1test = matrix(group1[sel1$test]) # class 1 test labels
group2test = matrix(group2[sel2$test]) # class 2 test labels
group_test = rbind(group1test,group2test) # test labels

# LDA model

model_lda = lda(train,group_train ) # model without cross-validation
model_lda_cv = lda(train,group_train,CV=TRUE) # model with leave-one-out cross-
validation

# prediction of training and testing samples

pred_train = predict(model_lda, train) # training
pred_test = predict(model_lda, test) # test

# figures of merit

# training

dim_train1 = dim(train1)
dim_train2 = dim(train2)

```

```

ac_train = mean(pred_train$class == group_train) # accuracy
spec_train = mean(pred_train$class[1:dim_train1[1]]==group1train) # specificity
sens_train = mean(pred_train$class[(dim_train1[1] + 1):(dim_train1[1]+dim_
train2[1]])==group2train) # sensitivity

# cross-validation

ac_cv = mean(model_lda_cv$class == group_train) # accuracy
spec_cv = mean(model_lda_cv$class[1:dim_train1[1]]==group1train) # specificity
sens_cv = mean(model_lda_cv$class[(dim_train1[1] + 1):(dim_train1[1]+dim_
train2[1]])==group2train) # sensitivity

# test

dim_test1 = dim(test1)
dim_test2 = dim(test2)

ac_test = mean(pred_test$class == group_test) # accuracy
spec_test = mean(pred_test$class[1:dim_test1[1]]==group1test) # specificity
sens_test = mean(pred_test$class[(dim_test1[1]+ 1):(dim_test1[1]+dim_
test2[1]])==group2test) # sensitivity

print("==== Training =====")
cat("Accuracy:")
ac_train
cat("Sensitivity:")
sens_train
cat("Specificity:")
spec_train

print("==== Cross-validation =====")
cat("Accuracy:")
ac_cv
cat("Sensitivity:")
sens_cv
cat("Specificity:")
spec_cv

print("==== Test =====")
cat("Accuracy:")
ac_test
cat("Sensitivity:")
sens_test
cat("Specificity:")
spec_test

# plotting PCA scores PC1 x PC2

```

```

dev.new()
group12 = rbind(matrix(group1), matrix(group2))
matplot(data.scores[group12==1,1],data.
scores[group12==1,2],pch=19,col='blue',xlab='PC1',ylab='PC2',main= 'PCA scores')
points(data.scores[group12==2,1],data.
scores[group12==2,2],pch=19,col='red',xlab='PC1',ylab='PC2',main= 'PCA scores')

# plotting PCA loadings PC1 & PC2

dev.new()
matplot(cmt,data.loadings[,1], type = "l", col = "blue", xlab = "Wavenumber (cm-1)",
ylab = 'Loadings', main = 'PCA loadings')
par(new=TRUE)
matplot(cmt,data.loadings[,2], type = "l", col = "red", xlab = "Wavenumber (cm-1)",
ylab = 'Loadings',main = 'PCA loadings')

# viewing posterior probabilities - training

dev.new()
matplot(pred_train$posterior[1:dim_train1[1],1 ],pch=19,col="blue",xlab="Samples",
ylab = "LD1", main = "Posterior Probability - Training") # training class 1
points(pred_train$posterior[(dim_train1[ 1]+ 1):(dim_train1[1]+dim_train2[1]),1],
pch=19,col="red",xlab="Samples", ylab = "LD1 ", main = "Posterior Probability -
Training") # training class 2

# visualizing posterior probabilities - cross validation

dev.new()
matplot(model_lda_cv$posterior[1:dim_train1[1],1 ],pch
=19,col="blue",xlab="Samples", ylab = "LD1", main = "Posterior Probability - Cross-
Validation") # class 1 cross-validation
points(pred_train$posterior[(dim_train1[ 1]+ 1):(dim_train1[1]+dim_
train2[1]),1],pch=19,col="red",xlab="Samples", ylab = "LD1 ", main = "Posterior
Probability - Cross-validation") # cross-validation class 2

# viewing posterior probabilities - test

dev.new()
matplot(pred_test$posterior[1:dim_test1[1],1],pch=19,col="blue",xlab="Samples",
ylab = "LD1", main=" Posterior Probability - Test") # test class 1
points(pred_train$posterior[(dim_test1[ 1]+ 1):(dim_test1[1]+dim_
test2[1]),1],pch=19,col="red",xlab="Samples", ylab = "LD1", main="Posterior
Probability - Test") # test class 2

```

Example 2 : The example we will describe here is basically what was described in the previous example, only changing the training and prediction sample selection method. In this case we present the MLM algorithm in the construction of PCA-LDA classification models for two classes (healthy and dengue, ATR-FTIR data) through the *prospectr*, *mass* and *ggplot2* packages.

R Script

```
# installing packages

install.packages ("prospectr")
install.packages ("MASS")

# reading packages

library(prospectr)
library(MASS)
library(ggplot2)

# reading the samples

class1 = read.table("DATASET/healthy.csv", header=FALSE) # samples of healthy
patients
class2 = read.table("DATASET/dengue.csv", header=FALSE) # samples from dengue
patients
cm = read.table("DATASET/cm.csv", header=FALSE) # wavelength

# class1 = healthy - control samples
# class2 = dengue - case samples

dim_class1 = dim (class1) # dimension of class1 array
dim_class2 = dim (class2) # dimension of class2 array

# transposing the data

class1t = t(class1)
class2t = t(class2)
cmt = t(cm)

# viewing data - all spectra

dev.new()
matplot(cmt, class1t, type ="l", col ="blue", xlab ="Wavenumber (cm-1)", ylab
="Absorbance", main ="Raw spectra")
matplot(cmt,class2t, type ="l", col ="red", xlab ="Wavenumber (cm-1)", ylab
="Absorbance", main ="Raw spectra", add =TRUE)

# viewing data - just the averages
```

```

dev.new()
matplot(cmt, colMeans (class1), type ="l", col ="blue", xlab ="Wavenumber (cm-1)",
ylab ="Absorbance", main ="Average spectra")
matplot(cmt,colMeans (class2), type ="l", col ="red", xlab ="Wavenumber (cm-1)",
ylab ="Absorbance", main ="Mean spectra", add =TRUE )

# scaling the data

data = rbind(class1, class2) # combining matrices one under the other
data_scal = scale(data) # scaling the data
dim_data = dim(data) # dimension of the data array

# PCA Model

data.svd = svd(data_scal) # SVD
data.scores = data.svd$u %*% diag(data.svd$d) # scores
data.loadings = data.svd$v # loadings

# PCA Variances

data.vars = data.svd$d^2 / (nrow (data)-1) # variance per PC
data.totalvar = sum(data.vars) # total variance
data.relvars = data.vars/data.totalvar # cumulative variance
variances = 100*round(data.relvars, digits = 3) # cumulative variance in %
variances[1:10] # variance in % in the first 10 PCs

# Choosing the number of PCs

par(mfrow = c( 2,2))
barplot(data.vars[1:10], main="Variance", names.arg = paste("PC", 1:10))
barplot(log( data.vars[1:10]), main="Log(variance)", names.arg = paste("PC", 1:10))
barplot(data.relvars[1:10], main="Relative Variances", names.arg = paste("PC",
1:10))
barplot(cumsum (100*data.relvars[1:10]), main="Cumulative Variance(%)", names.arg
= paste("PC", 1:10), ylim = c(0,100))

npc = 2 # number of PCs chosen to run the PCA

scores = data.scores[1:dim_data[1],1:npc] # PCA scores up to the chosen PC number
loadings = data.loadings[1:dim_data[2],1:npc] # PCA loadings up to the chosen PC
number

# Creation of the category vector: classes (1 = healthy, 2 = dengue)

group1 = rep(1,dim_class1[1]) # class 1
group2 = rep(2,dim_class2[1]) # class 2

```

```

# Selection of training and testing samples based on MLM

perc = 0.7 # 70% for training

ntrain1 = ceiling(perc * dim_class1[1]) # number of training samples class 1
ntrain2 = ceiling(perc * dim_class2[1]) # number of training samples class 2

scores1 = scores[1:dim_class1[1],1:npc] # scores class 1
scores2 = scores[(dim_class1[1]+ 1):dim_data[1],1:npc] # scores class 2

sel1 = kenStone(scores1, k = ntrain1) # KS class 1
sel2 = kenStone(scores2, k = ntrain2) # KS class 2

train1 = scores1[sel1$model,1:npc] # training class 1
train2 = scores2[sel2$model,1:npc] # training class 2
train = rbind(train1, train2) # joining training matrices

group1train = matrix(group1[sel1$model]) # class 1 training labels
group2train = matrix(group2[sel2$model]) # class 2 training labels
group_train = rbind(group1train, group2train) # training labels

test1 = scores1[sel1$test, 1:npc] # test class 1
test2 = scores2[sel2$test, 1:npc] # test class 2
test = rbind(test1, test2) # joining test matrices

group1test = matrix(group1[sel1$test]) # class 1 test labels
group2test = matrix(group2[sel2$test]) # class 2 test labels
group_test = rbind(group1test, group2test) # test labels

dim_train1 = dim(train1) # class 1 training dimension
dim_train2 = dim(train2) # class 2 training dimension
dim_test1 = dim(test1) # test dimension of class 1
dim_test2 = dim(test2) # class 2 test dimension

prob = 0.2 # MLM mutation probability = 20%

p1 = ceiling(prob * dim_test1[1])
p2 = ceiling(prob * dim_test2[1])

t1 = 1:dim_train1[1]
t2 = 1:dim_train2[1]

v1 = 1:dim_test1[1]
v2 = 1:dim_test2[1]

train1_sub = sample(x=t1, size=p1)
train2_sub = sample(x=t2, size=p2)

```



```

test1_sub = sample(x=v1, size=p1)
test2_sub = sample(x=v2, size=p2)

train1_new = rbind(train1[-train1_sub,],test1[test1_sub,])
train2_new = rbind(train2[-train2_sub,],test2[test2_sub,])

test1_new = rbind(test1[-test1_sub,],train1[train1_sub,])
test2_new = rbind(test2[-test2_sub,],train2[train2_sub,])

train = rbind(train1_new, train2_new)
test = rbind(test1_new, test2_new )

# LDA model

model_lda = lda(train,group_train ) # model without cross-validation
model_lda_cv = lda(train,group_train,CV=TRUE) # model with leave-one-out cross-
validation

# prediction of training and testing samples

pred_train = predict(model_lda, train) # training
pred_test = predict(model_lda, test) # test

# figures of merit

# training

dim_train1 = dim(train1)
dim_train2 = dim(train2)

ac_train = mean(pred_train$class == group_train) # accuracy
spec_train = mean(pred_train$class[1:dim_train1[1]]==group1train) # specificity
sens_train = mean(pred_train$class[(dim_train1[1] + 1):(dim_train1[1]+dim_
train2[1])]==group2train) # sensitivity

# cross-validation

ac_cv = mean(model_lda_cv$class == group_train) # accuracy
spec_cv = mean(model_lda_cv$class[1:dim_train1[1]]==group1train) # specificity
sens_cv = mean(model_lda_cv$class[(dim_train1[1] + 1):(dim_train1[1]+dim_
train2[1])]==group2train) # sensitivity

# test

dim_test1 = dim(test1)
dim_test2 = dim(test2)

```

```

ac_test = mean(pred_test$class == group_test) # accuracy
spec_test = mean(pred_test$class[1:dim_test1[1]]==group1test) # specificity
sens_test      =      mean(pred_test$class[(dim_test1[1]+      1):(dim_test1[1]+dim_
test2[1]])==group2test) # sensitivity

print("==== Training =====")
cat("Accuracy:")
ac_train
cat("Sensitivity:")
sens_train
cat("Specificity:")
spec_train

print("==== Cross-validation =====")
cat("Accuracy:")
ac_cv
cat("Sensitivity:")
sens_cv
cat("Specificity:")
spec_cv

print("==== Test =====")
cat("Accuracy:")
ac_test
cat("Sensitivity:")
sens_test
cat("Specificity:")
spec_test

# plotting PCA scores PC1 x PC2

dev.new()
group12 = rbind(matrix(group1), matrix(group2))
matplot(data.scores[group12==1,1],data.
scores[group12==1,2],pch=19,col='blue',xlab='PC1',ylab='PC2',main= 'PCA scores')
points(data.scores[group12==2,1],data.
scores[group12==2,2],pch=19,col='red',xlab='PC1',ylab='PC2',main= 'PCA scores')

# plotting PCA loadings PC1 & PC2

dev.new()
matplot(cmt,data.loadings[,1], type ="l", col = "blue", xlab = "Wavenumber (cm-1)",
ylab = 'Loadings', main = 'PCA loadings ')
par(new=TRUE)
matplot(cmt,data.loadings[,2], type ="l", col = "red", xlab = "Wavenumber (cm-1)",
ylab = 'Loadings',main = 'PCA loadings')

# viewing posterior probabilities - training

```

```

dev.new()
matplot(pred_train$posterior[1:dim_train1[1],1 ],pch
=19,col="blue",xlab="Samples", ylab ="LD1", main ="Posterior Probability -
Training") # training class 1
points(pred_train$posterior[(dim_train1[ 1]+ 1):(dim_train1[1]+dim_
train2[1]),1],pch=19,col="red",xlab="Samples", ylab ="LD1 ", main ="Posterior
Probability - Training") # training class 2

# visualizing posterior probabilities - cross validation

dev.new()
matplot(model_lda_cv$posterior[1:dim_train1[1],1 ],pch
=19,col="blue",xlab="Samples", ylab ="LD1", main ="Posterior Probability - Cross-
Validation") # class 1 cross-validation
points(pred_train$posterior[(dim_train1[ 1]+ 1):(dim_train1[1]+dim_
train2[1]),1],pch=19,col="red",xlab="Samples", ylab ="LD1 ", main ="Posterior
Probability - Cross-validation") # cross-validation class 2

# viewing posterior probabilities - test

dev.new()
matplot(pred_test$posterior[1:dim_test1[1],1],pch=19,col="blue",xlab="Samples",
ylab ="LD1", main=" Posterior Probability - Test") # test class 1
points(pred_train$posterior[(dim_test1[ 1]+ 1):(dim_test1[1]+dim_
test2[1]),1],pch=19,col="red",xlab="Samples", ylab ="LD1", main="Posterior
Probability - Test") # test class 2

```

4.6 SPA-LDA

In 2001, Araújo et al. [4] proposed a variable selection technique called the Successive Projections Algorithm (SPA). This technique basically uses simple operations in a vector space to minimize collinearity problems and has good efficiency in the context of multivariate calibration, specifically when applied to Multiple Linear Regression (MLR).

In 2005, Pontes et al. [5] adapted SPA, originally proposed for selecting spectral variables in MLR models, to be used in classification problems using the LDA classifier, resulting in SPA-LDA. SPA-LDA uses a cost function that calculates the average risk G of an incorrect classification by LDA based on the validation set as per the equation below:

$$G = \frac{1}{K_V} \sum_{k=1}^{K_V} g_k \quad \text{Eq. 9}$$

Where g_k (risk of misclassifying the object \mathbf{x}_k of k th validation sample) is defined according to the equation:

$$g_k = \frac{r^2(x_k, \mu_{lk})}{\min_{l \neq k} r^2(x_k, \mu_{lj})} \quad \text{Eq. 10}$$

Where the numerator $r^2(x_k, \mu_{lk})$ consists of the square of the Mahalanobis distance between the object x_k (with class index lk) and the mean of its class (μ_{lk}). The denominator of the same equation corresponds to the square of the Mahalanobis distance between the object x_k and the center of the nearest wrong class. This distance is calculated according to the equation below:

$$r^2(x_k, \mu_{lk}) = (x_k - \mu_{lk})\Sigma^{-1}(x_k - \mu_{lk})^T \quad \text{Eq. 11}$$

where the sample mean (μ_{lk}) and covariance R are calculated on the training set. As desired, the value of g_k should be as small as possible, that is, the object x_k should be close to the center of its true class and far from the centers of other classes.

Example 3: In this example, we present the SPA-LDA algorithm together with the sample selection algorithm (KS) in building multivariate classification models into two classes (healthy and dengue, ATR-FTIR) through the *prospectr*, *mass*, *ggplot2* and *lintools* packages.

R Script

```
# installing packages

install.packages("prospectr")
install.packages("MASS")
install.packages("lintools")

# reading packages

library(prospectr)
library(MASS)
library(ggplot2)
library(lintools)

# reading the samples

class1 = read.table("DATASET/healthy.csv", header=FALSE) # samples of healthy
patients
class2 = read.table("DATASET/dengue.csv", header=FALSE) # samples from dengue
patients
cm = read.table("DATASET/cm.csv", header=FALSE) # wavelength
```

```

# class1 = healthy - control samples
# class2 = dengue - case samples

dim_class1 = dim (class1) # dimension of class1 array
dim_class2 = dim (class2) # dimension of class2 array

# transposing the data

class1t = t(class1)
class2t = t(class2)
cmt = t(cm)

# viewing data - all spectra

dev.new()
matplot(cmt, class1t, type ="l", col ="blue", xlab ="Wavenumber (cm-1)", ylab
="Absorbance", main ="Raw spectra")
matplot(cmt,class2t, type ="l", col ="red", xlab ="Wavenumber (cm-1)", ylab
="Absorbance", main ="Raw spectra", add =TRUE)

# viewing data - just the averages

dev.new()
matplot(cmt, colMeans (class1), type ="l", col ="blue", xlab ="Wavenumber (cm-1)",
ylab ="Absorbance", main ="Average spectra")
matplot(cmt,colMeans (class2), type ="l", col ="red", xlab ="Wavenumber (cm-1)",
ylab ="Absorbance", main ="Mean spectra", add =TRUE )

# scaling the data

data = rbind(class1, class2) # combining matrices one under the other
data_scal = scale(data) # scaling the data
dim_data = dim(data) # dimension of the data array

# PCA Model

data.svd = svd(data_scal) # SVD
data.scores = data.svd$u %*% diag(data.svd$d) # scores
data.loadings = data.svd$v # loadings

# Creation of the category vector: classes (1 = healthy, 2 = dengue)

group1 = rep(1,dim_class1[1]) # class 1
group2 = rep(2,dim_class2[1]) # class 2
group12 = rbind(matrix(group1), matrix(group2)) # group 1 and 2

# SPA model

```

```

nvar = 22 # number of variables to select

datam = data.matrix(data, rownames.force=NA) # converting data to matrix

m = colMeans(data) # average of the spectra

model_spa = project(x= data.loadings[,1], A=datam, b=group12, neq =0) # spa model

x = abs(model_spa$x) # leaving positive values for the SPA response vector

temp = sort.int(x, decreasing=TRUE, index.return =TRUE)
variables = temp$ix[1:nvar] # identifying selected variables

# plot of selected variables

dev.new()
matplot(cmt,m,xlab="Wavenumber (cm-1)", type="l", ylab="Absorbance", main ="Average
spectrum with selected variables")
points(cm[variables],m[variables], pch =19)

datam_spa = datam[,variables] # absorbances for the selected variables

# selection of training and testing samples based on KS

perc = 0.7 # 70% for training

ntrain1 = ceiling(perc * dim_class1[1]) # number of training samples class 1
ntrain2 = ceiling(perc * dim_class2[1]) # number of training samples class 2

datam_spa1 = datam_spa[1:dim_class1[1],] # scores class 1
datam_spa2 = datam_spa[(dim_class1[1]+ 1): dim_data[1],] # scores class 2

sel1 = kenStone(datam_spa1, k = ntrain1) # KS class 1
sel2 = kenStone(datam_spa2, k = ntrain2) # KS class 2

train1 = datam_spa1[sel1$model,] # training class 1
train2 = datam_spa2[sel2$model,] # training class 2
train = rbind(train1,train2) # joining training matrices

group1train = matrix(group1[sel1$model]) # class 1 training labels
group2train = matrix(group2[sel2$model]) # class 2 training labels
group_train = rbind(group1train,group2train) # training labels

test1 = datam_spa1[sel1$test,] # test class 1
test2 = datam_spa2[sel2$test,] # test class 2
test = rbind(test1,test2) # joining test matrices

group1test = matrix(group1[sel1$test]) # class 1 test labels
group2test = matrix(group2[sel2$test]) # class 2 test labels
group_test = rbind(group1test,group2test) # test labels

```

```

# LDA model

model_lda = lda(train,group_train ) # model without cross-validation
model_lda_cv = lda(train,group_train,CV=TRUE) # model with leave-one-out cross-
validation

# prediction of training and testing samples

pred_train = predict(model_lda, train) # training
pred_test = predict(model_lda, test) # test

# figures of merit

# training

dim_train1 = dim(train1)
dim_train2 = dim(train2)

ac_train = mean(pred_train$class == group_train) # accuracy
spec_train = mean(pred_train$class[1:dim_train1[1]]==group1train) # specificity
sens_train = mean(pred_train$class[(dim_train1[1] + 1):(dim_train1[1]+dim_
train2[1])]==group2train) # sensitivity

# cross-validation

ac_cv = mean(model_lda_cv$class == group_train) # accuracy
spec_cv = mean(model_lda_cv$class[1:dim_train1[1]]==group1train) # specificity
sens_cv = mean(model_lda_cv$class[(dim_train1[1] + 1):(dim_train1[1]+dim_
train2[1])]==group2train) # sensitivity

# test

dim_test1 = dim(test1)
dim_test2 = dim(test2)

ac_test = mean(pred_test$class == group_test) # accuracy
spec_test = mean(pred_test$class[1:dim_test1[1]]==group1test) # specificity
sens_test = mean(pred_test$class[(dim_test1[1]+ 1):(dim_test1[1]+dim_
test2[1])]==group2test) # sensitivity

print("==== Training =====")
cat("Accuracy:")
ac_train
cat("Sensitivity:")
sens_train
cat("Specificity:")
spec_train

```

```

print("=====  

cat("Accuracy:")  

ac_cv  

cat("Sensitivity:")  

sens_cv  

cat("Specificity:")  

spec_cv  

print("=====  

cat("Accuracy:")  

ac_test  

cat("Sensitivity:")  

sens_test  

cat("Specificity:")  

spec_test  

# viewing posterior probabilities - training  

dev.new()  

matplot(pred_train$posterior[1:dim_train1[1],1 ],pch  

=19,col="blue",xlab="Samples", ylab ="LD1", main ="Posterior Probability -  

Training") # training class 1  

points(pred_train$posterior[(dim_train1[ 1]+ 1):(dim_train1[1]+dim_  

train2[1]),1],pch=19,col="red",xlab="Samples", ylab ="LD1 ", main ="Posterior  

Probability - Training") # training class 2  

# visualizing posterior probabilities - cross validation  

dev.new()  

matplot(model_lda_cv$posterior[1:dim_train1[1],1 ],pch  

=19,col="blue",xlab="Samples", ylab ="LD1", main ="Posterior Probability - Cross-  

Validation") # class 1 cross-validation  

points(pred_train$posterior[(dim_train1[ 1]+ 1):(dim_train1[1]+dim_  

train2[1]),1],pch=19,col="red",xlab="Samples", ylab ="LD1 ", main ="Posterior  

Probability - Cross-validation") # cross-validation class 2  

# viewing posterior probabilities - test  

dev.new()  

matplot(pred_test$posterior[1:dim_test1[1],1],pch=19,col="blue",xlab="Samples",  

ylab ="LD1", main=" Posterior Probability - Test") # test class 1  

points(pred_train$posterior[(dim_test1[ 1]+ 1):(dim_test1[1]+dim_  

test2[1]),1],pch=19,col="red",xlab="Samples", ylab ="LD1", main="Posterior  

Probability - Test") # test class 2

```


Example 4: In this example, we present the SPA-LDA algorithm together with the sample selection algorithm (MLM) in building multivariate classification models into two classes (healthy and dengue, ATR-FTIR) through the *prospectr*, *mass*, *ggplot2* and *lintools* packages.

R Script

```
# installing packages

install.packages("prospectr")
install.packages("MASS")
install.packages("lintools")

# reading packages

library(prospectr)
library(MASS)
library(ggplot2)
library(lintools)

# reading the samples

class1 = read.table("DATASET/healthy.csv", header=FALSE) # samples of healthy
patients
class2 = read.table("DATASET/dengue.csv", header=FALSE) # samples from dengue
patients
cm = read.table("DATASET/cm.csv", header=FALSE) # wavelength

# class1 = healthy - control samples
# class2 = dengue - case samples

dim_class1 = dim(class1) # dimension of class1 array
dim_class2 = dim(class2) # dimension of class2 array

# transposing the data

class1t = t(class1)
class2t = t(class2)
cmt = t(cm)

# viewing data - all spectra

dev.new()
matplot(cmt, class1t, type = "l", col = "blue", xlab = "Wavenumber (cm-1)", ylab
="Absorbance", main = "Raw spectra")
matplot(cmt, class2t, type = "l", col = "red", xlab = "Wavenumber (cm-1)", ylab
="Absorbance", main = "Raw spectra", add = TRUE)
```

```

# viewing data - just the averages

dev.new()
matplot(cmt, colMeans (class1), type ="l", col ="blue", xlab ="Wavenumber (cm-1)",
ylab ="Absorbance", main ="Average spectra")
matplot(cmt,colMeans (class2), type ="l", col ="red", xlab ="Wavenumber (cm-1)",
ylab ="Absorbance", main ="Mean spectra", add =TRUE )

# scaling the data

data = rbind(class1, class2) # combining matrices one under the other
data_scal = scale(data) # scaling the data
dim_data = dim(data) # dimension of the data array

# PCA Model

data.svd = svd(data_scal) # SVD
data.scores = data.svd$u %*% diag(data.svd$d) # scores
data.loadings = data.svd$v # loadings

# Creation of the category vector: classes (1 = healthy, 2 = dengue)

group1 = rep(1,dim_class1[1]) # class 1
group2 = rep(2,dim_class2[1]) # class 2
group12 = rbind(matrix(group1), matrix(group2)) # group 1 and 2

# SPA model

nvar = 22 # number of variables to select

datam = data.matrix(data, rownames.force=NA) # converting data to matrix

m = colMeans(data) # average of the spectra

model_spa = project(x= data.loadings[,1], A=datam, b=group12, neq =0) # spa model

x = abs(model_spa$x) # leaving positive values for the SPA response vector

temp = sort.int(x, decreasing=TRUE, index.return =TRUE)
variables = temp$ix[1:nvar] # identifying selected variables

# plot of selected variables

dev.new()
matplot(cmt,m,xlab="Wavenumber (cm-1)", type="l", ylab="Absorbance", main ="Average
spectrum with selected variables")
points(cm[variables],m[variables], pch =19)

```

```

datam_spa = datam[,variables] # absorbances for the selected variables

# selection of training and testing samples based on MLM

perc = 0.7 # 70% for training

ntrain1 = ceiling(perc * dim_class1[1]) # number of training samples class 1
ntrain2 = ceiling(perc * dim_class2[1]) # number of training samples class 2

datam_spa1 = datam_spa[1:dim_class1[1],] # scores class 1
datam_spa2 = datam_spa[(dim_class1[ 1]+ 1): dim_data[1],] # scores class 2

sel1 = kenStone(datam_spa1, k = ntrain1) # KS class 1
sel2 = kenStone(datam_spa2, k = ntrain2) # KS class 2

train1 = datam_spa1[sel1$model,] # training class 1
train2 = datam_spa2[sel2$model,] # training class 2
train = rbind(train1,train2) # joining training matrices

group1train = matrix(group1[sel1$model]) # class 1 training labels
group2train = matrix(group2[sel2$model]) # class 2 training labels
group_train = rbind(group1train,group2train) # training labels

test1 = datam_spa1[sel1$test,] # test class 1
test2 = datam_spa2[sel2$test,] # test class 2
test = rbind(test1,test2) # joining test matrices

group1test = matrix(group1[sel1$test]) # class 1 test labels
group2test = matrix(group2[sel2$test]) # class 2 test labels
group_test = rbind(group1test,group2test) # test labels

dim_train1 = dim(train1) # class 1 training dimension
dim_train2 = dim(train2) # class 2 training dimension
dim_test1 = dim(test1) # test dimension of class 1
dim_test2 = dim(test2) # class 2 test dimension

prob = 0.2 # MLM mutation probability = 20%

p1 = ceiling(prob * dim_test1[1])
p2 = ceiling(prob * dim_test2[1])

t1 = 1:dim_train1[1]
t2 = 1:dim_train2[1]

v1 = 1:dim_test1[1]
v2 = 1:dim_test2[1]

```

```

rain1_sub = sample(x=t1, size=p1)
train2_sub = sample(x=t2, size=p2)

test1_sub = sample(x=v1, size=p1)
test2_sub = sample(x=v2, size=p2)

train1_new = rbind(train1[-train1_sub,],test1[test1_sub,])
train2_new = rbind(train2[-train2_sub,],test2[test2_sub,])

test1_new = rbind(test1[-test1_sub,],train1[train1_sub,])
test2_new = rbind(test2[-test2_sub,],train2[train2_sub,])

train = rbind(train1_new,train2_new)
test = rbind(test1_new,test2_new )

# LDA model

model_lda = lda(train,group_train ) # model without cross-validation
model_lda_cv = lda(train,group_train,CV=TRUE) # model with leave-one-out cross-
validation

# prediction of training and testing samples

pred_train = predict(model_lda, train) # training
pred_test = predict(model_lda, test) # test

# figures of merit

# training

dim_train1 = dim(train1)
dim_train2 = dim(train2)

ac_train = mean(pred_train$class == group_train) # accuracy
spec_train = mean(pred_train$class[1:dim_train1[1]]==group1train) # specificity
sens_train = mean(pred_train$class[(dim_train1[1] + 1):(dim_train1[1]+dim_
train2[1])]==group2train) # sensitivity

# cross-validation

ac_cv = mean(model_lda_cv$class == group_train) # accuracy
spec_cv = mean(model_lda_cv$class[1:dim_train1[1]]==group1train) # specificity
sens_cv = mean(model_lda_cv$class[(dim_train1[1] + 1):(dim_train1[1]+dim_
train2[1])]==group2train) # sensitivity

# test

```

```

t
dim_test1 = dim(test1)
dim_test2 = dim(test2)

ac_test = mean(pred_test$class == group_test) # accuracy
spec_test = mean(pred_test$class[1:dim_test1[1]]==group1test) # specificity
sens_test = mean(pred_test$class[(dim_test1[1]+ 1):(dim_test1[1]+dim_
test2[1]])==group2test) # sensitivity

print("==== Training =====")
cat("Accuracy:")
ac_train
cat("Sensitivity:")
sens_train
cat("Specificity:")
spec_train

print("==== Cross-validation =====")
cat("Accuracy:")
ac_cv
cat("Sensitivity:")
sens_cv
cat("Specificity:")
spec_cv

print("==== Test =====")
cat("Accuracy:")
ac_test
cat("Sensitivity:")
sens_test
cat("Specificity:")
spec_test

# viewing posterior probabilities - training

dev.new()
matplot(pred_train$posterior[1:dim_train1[1],1 ],pch
=19,col="blue",xlab="Samples", ylab ="LD1", main ="Posterior Probability -
Training") # training class 1
points(pred_train$posterior[(dim_train1[ 1]+ 1):(dim_train1[1]+dim_
train2[1]),1],pch=19,col="red",xlab="Samples", ylab ="LD1 ", main ="Posterior
Probability - Training") # training class 2

# visualizing posterior probabilities - cross validation

dev.new()
matplot(model_lda_cv$posterior[1:dim_train1[1],1 ],pch
=19,col="blue",xlab="Samples", ylab ="LD1", main ="Posterior Probability - Cross-
Validation") # class 1 cross-validation
points(pred_train$posterior[(dim_train1[ 1]+ 1):(dim_train1[1]+dim_
train2[1]),1],pch=19,col="red",xlab="Samples", ylab ="LD1 ", main ="Posterior
Probability - Cross-validation") # cross-validation class 2

```

```
# viewing posterior probabilities - test

dev.new()
matplot(pred_test$posterior[1:dim_test1[1],1],pch=19,col="blue",xlab="Samples",
ylab = "LD1", main=" Posterior Probability - Test") # test class 1
points(pred_train$posterior[(dim_test1[ 1]+ 1):(dim_test1[1]+dim_
test2[1]),1],pch=19,col="red",xlab="Samples", ylab = "LD1", main="Posterior
Probability - Test") # test class 2
```

4.7 GA-LDA

The genetic algorithm (GA) is a bioinspired optimization method developed to solve optimization and machine learning problems, created by John Henry Holland in 1975 [6]. It is also considered a method of selecting variables of stochastic nature. GA simulates natural processes of survival and reproduction of populations, especially based on the theory of species evolution proposed by Darwin. Chromosomes encoded by real numbers is a context applied to the genetic algorithm, as it is possible to construct artificial chromosomes and simulate a natural evolutionary process.

Basically, GA encodes subsets of variables in the form of a series of binary values (chromosomes) and the position on the chromosome (gene) is associated with one of the variables for selection. In this way, a population is generated from a random set of individuals and during the evolution process, the population receives an index that reflects the ability to adapt to a given fitness. Finally, the fittest individuals are selected for the selection process and the least fit are eliminated. The process is repeated until a certain satisfactory solution is reached or the maximum number of generations is reached.

In spectroscopic studies, for example, standard binary chromosomes are used with a size equal to the number of wavelengths in a spectrum. The GA-LDA algorithm in multivariate classification uses a cost function calculated as the inverse of the risk G described in equation 4.9 using the wavelengths encoded in the chromosome. Normally, crossover and mutation operators are used at a level of probability as well as the size of the population at each generation.

Example 5: In this example, we present the GA-LDA algorithm along with the sample selection algorithm (KS) to build multivariate classification models for two classes (healthy and dengue, ATR-FTIR) through the *prospectr*, *mass*, *ggplot2*, *lintools*, *caret*, *dplyr*, *lattice* and *GA* packages.

R Script

```
# installing packages

install.packages("prospectr")
install.packages("MASS")
install.packages("caret")
install.packages("GA")

# reading packages

library(prospectr)
library(MASS)
library(ggplot2)
library(lintools)
library(caret)
library(dplyr)
library(lattice)
library(GA)

# reading the samples

class1 = read.table("DATASET/healthy.csv", header=FALSE) # samples of healthy
patients
class2 = read.table("DATASET/dengue.csv", header=FALSE) # samples from dengue
patients
cm = read.table("DATASET/cm.csv", header=FALSE) # wavelength

# class1 = healthy - control samples
# class2 = dengue - case samples

dim_class1 = dim (class1) # dimension of class1 array
dim_class2 = dim (class2) # dimension of class2 array

# transposing the data

class1t = t(class1)
class2t = t(class2)
cmt = t(cm)

# viewing data - all spectra

dev.new()
matplot(cmt, class1t, type ="l", col ="blue", xlab ="Wavenumber (cm-1)", ylab
="Absorbance", main ="Raw spectra")
matplot(cmt,class2t, type ="l", col =" red ", xlab ="Wavenumber (cm-1)", ylab
="Absorbance", main ="Raw spectra", add =TRUE)

# viewing data - just the averages
```

```

dev.new()
matplot(cmt, colMeans (class1), type ="l", col ="blue", xlab ="Wavenumber (cm-1)",
ylab ="Absorbance", main ="Average spectra")
matplot(cmt,colMeans (class2), type ="l", col =" red ", xlab ="Wavenumber (cm-1)",
ylab ="Absorbance", main ="Mean spectra", add =TRUE )

# scaling the data

data = rbind(class1, class2) # combining matrices one under the other
data_scal = scale(data) # scaling the data
dim_data = dim(data) # dimension of the data array

# Creation of the category vector: classes (1 = healthy, 2 = dengue)

group1 = rep(1,dim_class1[1]) # class 1
group2 = rep(2,dim_class2[1]) # class 2
group12 = rbind(matrix(group1), matrix(group2)) # group 1 and 2

# GA algorithm

datam = data.matrix(data, rownames.force=NA) # converting data to matrix

# Function to Establish Population

myInit <- function(k){

  function(GA){
    m <- matrix(0, ncol = GA@nBits, nrow = GA@popSize)

    for(i in seq_len(GA@popSize))
      m[i, sample(GA@nBits, k)] <- 1

    m
  }
}

# Crossover Function

myCrossover <- function(GA, parents){

  parents <- GA@population[parents,] %>%
  apply(1, function(x) which(x == 1)) %>%
  t()

  parents_diff <- list("vector", 2)
  parents_diff[[1]] <- setdiff(parents[2,], parents[1,])
  parents_diff[[2]] <- setdiff(parents[1,], parents[2,])

```



```

children_ind <- list("vector", 2)
for(i in 1:2){
  k <- length(parents_diff[[i]])
  change_k <- sample(k, sample(ceiling(k/2), 1))
  children_ind[[i]] <- if(length(change_k) > 0){
    c(parents[i, -change_k], parents_diff[[i]][change_k])
  } else {
    parents[i,]
  }
}

children <- matrix(0, nrow = 2, ncol = GA@nBits)
for(i in 1:2)
  children[i, children_ind[[i]]] <- 1

list(children = children, fitness = c(NA, NA))
}

# Mutation Function

myMutation <- function(GA, parent){

  ind <- which(GA@population[parent,] == 1)
  n_change <- sample(3, 1)
  ind[sample(length(ind), n_change)] <- sample(setdiff(seq_len(GA@nBits), ind),
n_change)
  parent <- integer(GA@nBits)
  parent[ind] <- 1

  parent
}

# Adjustment Function

f <- function(x, values){

  ind <- which(x == 1)
  y <- values[ind]
  y <- ifelse(y %% 2 != 0, y, 0)
  y <- y[1:10]
  return(sum(y))
}

# GA Model

model_GA = ga(type="binary", fitness=f, values=datam , nBits=ncol(datam),
population=myInit(nrow(datam)), crossover = myCrossover, mutation=myMutation,
run=200, pmutation=0.1, maxiter=1000, popSize = nrow(datam))

```

```

# selected variables

ind = which(model_GA@solution[1,] == 1)
if (length(ind) > 22){
  indmax = 22 # maximum number of variables selected
  ind = ind[1:indmax]
}

# array with selected variables

datam_ga = datam[,ind]
m = colMeans(datam)
variables = ind

# plot of selected variables
dev.new()
matplot(cmt,m,xlab="Wavenumber (cm-1)", type ="l", ylab ="Absorbance", main
="Average spectrum with selected variables")
points(cm[variables],m[variables], pch=19)

# selection of training and testing samples based on KS

perc = 0.7 # 70% for training

ntrain1 = ceiling(perc * dim_class1[1]) # number of training samples class 1
ntrain2 = ceiling(perc * dim_class2[1]) # number of training samples class 2

datam_ga1 = datam_ga[1:dim_class1[1],] # scores class 1
datam_ga2 = datam_ga[(dim_class1[1]+ 1): dim_data[1],] # scores class 2

sel1 = kenStone(datam_ga1, k = ntrain1) # KS class 1
sel2 = kenStone(datam_ga2, k = ntrain2) # KS class 2

train1 = datam_ga1[sel1$model,] # training class 1
train2 = datam_ga2[sel2$model,] # training class 2
train = rbind(train1,train2) # joining training matrices

group1train = matrix(group1[sel1$model]) # class 1 training labels
group2train = matrix(group2[sel2$model]) # class 2 training labels
group_train = rbind(group1train,group2train) # training labels

test1 = datam_ga1[sel1$test,] # test class 1
test2 = datam_ga2[sel2$test,] # test class 2
test = rbind(test1,test2) # joining test matrices

group1test = matrix(group1[sel1$test]) # class 1 test labels
group2test = matrix(group2[sel2$test]) # class 2 test labels
group_test = rbind(group1test,group2test) # test labels

```

```

# LDA model

model_lda = lda(train,group_train ) # model without cross-validation
model_lda_cv = lda(train,group_train,CV=TRUE) # model with leave-one-out cross-
validation

# prediction of training and testing samples

pred_train = predict(model_lda, train) # training
pred_test = predict(model_lda, test) # test

# figures of merit

# training

dim_train1 = dim(train1)
dim_train2 = dim(train2)

ac_train = mean(pred_train$class == group_train) # accuracy
spec_train = mean(pred_train$class[1:dim_train1[1]]==group1train) # specificity
sens_train = mean(pred_train$class[(dim_train1[1] + 1):(dim_train1[1]+dim_
train2[1])]==group2train) # sensitivity

# cross-validation

ac_cv = mean(model_lda_cv$class == group_train) # accuracy
spec_cv = mean(model_lda_cv$class[1:dim_train1[1]]==group1train) # specificity
sens_cv = mean(model_lda_cv$class[(dim_train1[1] + 1):(dim_train1[1]+dim_
train2[1])]==group2train) # sensitivity

# test

dim_test1 = dim(test1)
dim_test2 = dim(test2)

ac_test = mean(pred_test$class == group_test) # accuracy
spec_test = mean(pred_test$class[1:dim_test1[1]]==group1test) # specificity
sens_test = mean(pred_test$class[(dim_test1[1]+ 1):(dim_test1[1]+dim_
test2[1])]==group2test) # sensitivity

print("==== Training =====")
cat("Accuracy:")
ac_train
cat("Sensitivity:")
sens_train
cat("Specificity:")
spec_train

```

```

print("=====  

cat("Accuracy:")
ac_cv
cat("Sensitivity:")
sens_cv
cat("Specificity:")
spec_cv

print("=====  

cat("Accuracy:")
ac_test
cat("Sensitivity:")
sens_test
cat("Specificity:")
spec_test

# viewing posterior probabilities - training

dev.new()
matplot(pred_train$posterior[1:dim_train1[1],1 ],pch
=19,col="blue",xlab="Samples", ylab ="LD1", main ="Posterior Probability -
Training") # training class 1
points(pred_train$posterior[(dim_train1[ 1]+ 1):(dim_train1[1]+dim_
train2[1]),1],pch=19,col="red",xlab="Samples", ylab ="LD1", main ="Posterior
Probability - Training") # training class 2

# visualizing posterior probabilities - cross validation

dev.new()
matplot(model_lda_cv$posterior[1:dim_train1[1],1 ],pch
=19,col="blue",xlab="Samples", ylab ="LD1", main ="Posterior Probability - Cross-
Validation") # class 1 cross-validation
points(pred_train$posterior[(dim_train1[ 1]+ 1):(dim_train1[1]+dim_
train2[1]),1],pch=19,col="red",xlab="Samples", ylab ="LD1", main ="Posterior
Probability - Cross-validation") # cross-validation class 2

# viewing posterior probabilities - test

dev.new()
matplot(pred_test$posterior[1:dim_test1[1],1],pch=19,col="blue",xlab="Samples",
ylab ="LD1", main=" Posterior Probability - Test") # test class 1
points(pred_train$posterior[(dim_test1[ 1]+ 1):(dim_test1[1]+dim_
test2[1]),1],pch=19,col="red",xlab="Samples", ylab ="LD1", main="Posterior
Probability - Test") # test class 2

```

Example 6: In this example, we present the GA-LDA algorithm together with the sample selection algorithm (MLM) to build multivariate classification models for two classes (healthy and dengue, ATR-FTIR) through the *prospectr*, *mass*, *ggplot2*, *lintools*, *caret*, *dplyr*, *lattice* and *GA* packages.

R Script

```
# installing packages

install.packages("prospectr")
install.packages("MASS")
install.packages("caret")
install.packages("GA")

# reading packages

library(prospectr)
library(MASS)
library(ggplot2)
library(lintools)
library(caret)
library(dplyr)
library(lattice)
library(GA)

# reading the samples

class1 = read.table("DATASET/healthy.csv", header=FALSE) # samples of healthy
patients
class2 = read.table("DATASET/dengue.csv", header=FALSE) # samples from dengue
patients
cm = read.table("DATASET/cm.csv", header=FALSE) # wavelength

# class1 = healthy - control samples
# class2 = dengue - case samples

dim_class1 = dim(class1) # dimension of class1 array
dim_class2 = dim(class2) # dimension of class2 array

# transposing the data

class1t = t(class1)
class2t = t(class2)
cmt = t(cm)

# viewing data - all spectra

dev.new()
matplot(cmt, class1t, type = "l", col = "blue", xlab = "Wavenumber (cm-1)", ylab
="Absorbance", main = "Raw spectra")
matplot(cmt, class2t, type = "l", col = "red", xlab = "Wavenumber (cm-1)", ylab
="Absorbance", main = "Raw spectra", add = TRUE)
```

```

# viewing data - just the averages

dev.new()
matplot(cmt, colMeans (class1), type ="l", col ="blue", xlab ="Wavenumber (cm-1)",
ylab ="Absorbance", main ="Average spectra")
matplot(cmt,colMeans (class2), type ="l", col ="red", xlab ="Wavenumber (cm-1)",
ylab ="Absorbance", main ="Mean spectra", add =TRUE )

# scaling the data

data = rbind(class1, class2) # combining matrices one under the other
data_scal = scale(data) # scaling the data
dim_data = dim(data) # dimension of the data array

# Creation of the category vector: classes (1 = healthy, 2 = dengue)

group1 = rep(1,dim_class1[1]) # class 1
group2 = rep(2,dim_class2[1]) # class 2
group12 = rbind(matrix(group1), matrix(group2)) # group 1 and 2

# GA algorithm

datam = data.matrix(data, rownames.force=NA) # converting data to matrix

# Function to Establish Population

myInit <- function(k){

  function(GA){
    m <- matrix(0, ncol = GA@nBits, nrow = GA@popSize)

    for(i in seq_len(GA@popSize))
      m[i, sample(GA@nBits, k)] <- 1

    m
  }
}

# Crossover Function

myCrossover <- function(GA, parents){

  parents <- GA@population[parents,] %>%
  apply(1, function(x) which(x == 1)) %>%
  t()
}

```

```

parents_diff <- list("vector", 2)
parents_diff[[1]] <- setdiff(parents[2,], parents[1,])
parents_diff[[2]] <- setdiff(parents[1,], parents[2,])

children_ind <- list("vector", 2)
for(i in 1:2){
  k <- length(parents_diff[[i]])
  change_k <- sample(k, sample(ceiling(k/2), 1))
  children_ind[[i]] <- if(length(change_k) > 0){
    c(parents[i, -change_k], parents_diff[[i]][change_k])
  } else {
    parents[i,]
  }
}

children <- matrix(0, nrow = 2, ncol = GA@nBits)
for(i in 1:2)
  children[i, children_ind[[i]]] <- 1

list(children = children, fitness = c(NA, NA))
}

# Mutation Function

myMutation <- function(GA, parent){

  ind <- which(GA@population[parent,] == 1)
  n_change <- sample(3, 1)
  ind[sample(length(ind), n_change)] <- sample(setdiff(seq_len(GA@nBits), ind),
n_change)
  parent <- integer(GA@nBits)
  parent[ind] <- 1

  parent
}

# Adjustment Function

f <- function(x, values){

  ind <- which(x == 1)
  y <- values[ind]
  y <- ifelse(y %% 2 != 0, y, 0)
  y <- y[1:10]
  return(sum(y))
}

# GA Model

```

```

model_GA = ga(type="binary", fitness=f, values=datam, nBits=ncol(datam),
population=myInit(nrow(datam)), crossover = myCrossover, mutation=myMutation,
run=200, pmutation=0.1, maxiter=1000, popSize = nrow(datam))

# selected variables

ind = which(model_GA@solution[1,] == 1)
if (length(ind) > 22){
  indmax = 22 # maximum number of variables selected
  ind = ind[1:indmax]
}

# array with selected variables

datam_ga = datam[,ind]
m = colMeans(datam)
variables = ind

# plot of selected variables
dev.new()
matplot(cmt,m,xlab="Wavenumber (cm-1)", type ="l", ylab ="Absorbance", main
="Average spectrum with selected variables")
points(cm[variables],m[variables], pch=19)

# selection of training and testing samples based on MLM

perc = 0.7 # 70% for training

ntrain1 = ceiling(perc * dim_class1[1]) # number of training samples class 1
ntrain2 = ceiling(perc * dim_class2[1]) # number of training samples class 2

datam_ga1 = datam_ga[1:dim_class1[1],] # scores class 1
datam_ga2 = datam_ga[(dim_class1[ 1]+ 1): dim_data[1],] # scores class 2

sel1 = kenStone(datam_ga1, k = ntrain1) # KS class 1
sel2 = kenStone(datam_ga2, k = ntrain2) # KS class 2

train1 = datam_ga1[sel1$model,] # training class 1
train2 = datam_ga2[sel2$model,] # training class 2
train = rbind(train1,train2) # joining training matrices

group1train = matrix(group1[sel1$model]) # class 1 training labels
group2train = matrix(group2[sel2$model]) # class 2 training labels
group_train = rbind(group1train,group2train) # training labels

test1 = datam_ga1[sel1$test,] # test class 1
test2 = datam_ga2[sel2$test,] # test class 2
test = rbind(test1,test2) # joining test matrices

```



```

group1test = matrix(group1[sel1$test]) # class 1 test labels
group2test = matrix(group2[sel2$test]) # class 2 test labels
group_test = rbind(group1test,group2test) # test labels

dim_train1 = dim(train1) # class 1 training dimension
dim_train2 = dim(train2) # class 2 training dimension
dim_test1 = dim(test1) # test dimension of class 1
dim_test2 = dim(test2) # class 2 test dimension

prob = 0.2 # MLM mutation probability = 20%

p1 = ceiling(prob * dim_test1[1])
p2 = ceiling(prob * dim_test2[1])

t1 = 1:dim_train1[1]
t2 = 1:dim_train2[1]

v1 = 1:dim_test1[1]
v2 = 1:dim_test2[1]

train1_sub = sample(x=t1, size=p1)
train2_sub = sample(x=t2, size=p2)

test1_sub = sample(x=v1, size=p1)
test2_sub = sample(x=v2, size=p2)

train1_new = rbind(train1[-train1_sub,],test1[test1_sub,])
train2_new = rbind(train2[-train2_sub,],test2[test2_sub,])

test1_new = rbind(test1[-test1_sub,],train1[train1_sub,])
test2_new = rbind(test2[-test2_sub,],train2[train2_sub,])

train = rbind(train1_new,train2_new)
test = rbind(test1_new,test2_new)

# LDA model

model_lda = lda(train,group_train ) # model without cross-validation
model_lda_cv = lda(train,group_train,CV=TRUE) # model with leave-one-out cross-
validation

# prediction of training and testing samples

pred_train = predict(model_lda, train) # training
pred_test = predict(model_lda, test) # test

# figures of merit

```

```

# training

dim_train1 = dim(train1)
dim_train2 = dim(train2)

ac_train = mean(pred_train$class == group_train) # accuracy
spec_train = mean(pred_train$class[1:dim_train1[1]]==group1train) # specificity
sens_train = mean(pred_train$class[(dim_train1[1] + 1):(dim_train1[1]+dim_
train2[1])]==group2train) # sensitivity

# cross-validation

ac_cv = mean(model_lda_cv$class == group_train) # accuracy
spec_cv = mean(model_lda_cv$class[1:dim_train1[1]]==group1train) # specificity
sens_cv = mean(model_lda_cv$class[(dim_train1[1] + 1):(dim_train1[1]+dim_
train2[1])]==group2train) # sensitivity

# test

dim_test1 = dim(test1)
dim_test2 = dim(test2)

ac_test = mean(pred_test$class == group_test) # accuracy
spec_test = mean(pred_test$class[1:dim_test1[1]]==group1test) # specificity
sens_test = mean(pred_test$class[(dim_test1[1]+ 1):(dim_test1[1]+dim_
test2[1])]==group2test) # sensitivity

print("==== Training =====")
cat("Accuracy:")
ac_train
cat("Sensitivity:")
sens_train
cat("Specificity:")
spec_train

print("==== Cross-validation =====")
cat("Accuracy:")
ac_cv
cat("Sensitivity:")
sens_cv
cat("Specificity:")
spec_cv

print("==== Test =====")
cat("Accuracy:")
ac_test
cat("Sensitivity:")
sens_test
cat("Specificity:")
spec_test

```

```

# viewing posterior probabilities - training

dev.new()
matplot(pred_train$posterior[1:dim_train1[1],1 ],pch
=19,col="blue",xlab="Samples", ylab ="LD1", main ="Posterior Probability -
Training") # training class 1
points(pred_train$posterior[(dim_train1[ 1]+ 1):(dim_train1[1]+dim_
train2[1]),1],pch=19,col="red",xlab="Samples", ylab ="LD1", main ="Posterior
Probability - Training") # training class 2

# visualizing posterior probabilities - cross validation

dev.new()
matplot(model_lda_cv$posterior[1:dim_train1[1],1 ],pch
=19,col="blue",xlab="Samples", ylab ="LD1", main ="Posterior Probability - Cross-
Validation") # class 1 cross-validation
points(pred_train$posterior[(dim_train1[ 1]+ 1):(dim_train1[1]+dim_
train2[1]),1],pch=19,col="red",xlab="Samples", ylab ="LD1", main ="Posterior
Probability - Cross-validation") # cross-validation class 2

# viewing posterior probabilities - test

dev.new()
matplot(pred_test$posterior[1:dim_test1[1],1],pch=19,col="blue",xlab="Samples",
ylab ="LD1", main=" Posterior Probability - Test") # test class 1
points(pred_train$posterior[(dim_test1[ 1]+ 1):(dim_test1[1]+dim_
test2[1]),1],pch=19,col="red",xlab="Samples", ylab ="LD1", main="Posterior
Probability - Test") # test class 2

```

4.8 TUCKER3-LDA

In 1966, the psychometrician Ledyard R. Tucker developed a multidimensional data processing method currently known as Tucker 1, Tucker 2 and Tucker 3 [7]. Tucker 1 consists of unfolding the data array of dimensions $I \times J \times K$ into a matrix of dimensions $I \times JK$, or in other words, the individual application of a PCA in the three forms of unfolding, as shown in **Figure 4.2** . However, this splitting can be carried out in the other two directions ($J \times KI$ or $K \times IJ$). As can be seen, Tucker's proposal is to ignore a trilinear structure (the instrumental response can be represented by the product of three independent vectors) of the data by decomposing them in a Bilinear method .

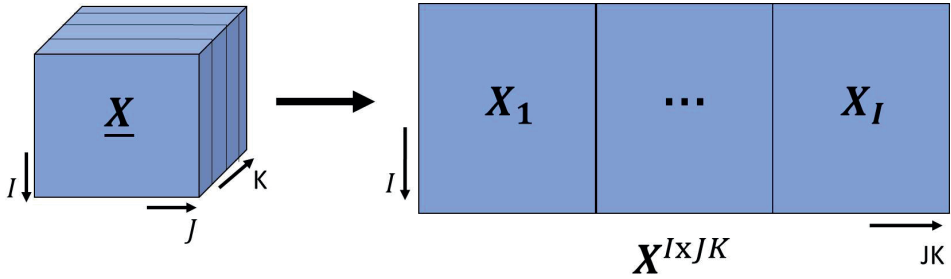


Figure 4.2: Unfolding a three-dimensional data array into a matrix.

The Tucker 3 model, for example, can be represented by the following matrix equation:

$$\underline{X} = \underline{A}\underline{G}(\underline{C} \otimes \underline{B})^t + \underline{E} \tag{Eq. 12}$$

Where the matrices \underline{A} ($I \times D$), \underline{B} ($J \times E$) and \underline{C} ($K \times F$) have dimensions containing the weights ("loadings") of the model relative to the three dimensions of the data, respectively; the matrix \underline{G} ($D \times E \times F$) corresponds to the central matrix ("core matrix") and the elements of the tensor \underline{G} indicate the importance of each interaction between the factor responses; the tensor \underline{E} ($I \times J \times K$) contains the model errors and the symbol " \otimes " represents the Kronecker product [8]. D , E and F indicate the number of factors in the three dimensions of the data, respectively. It is important to highlight that the Tucker 3 model accepts that the number of decomposed factors is different in each dimension. Figure 4.3 represents the data decomposition carried out by the Tucker 3 method:

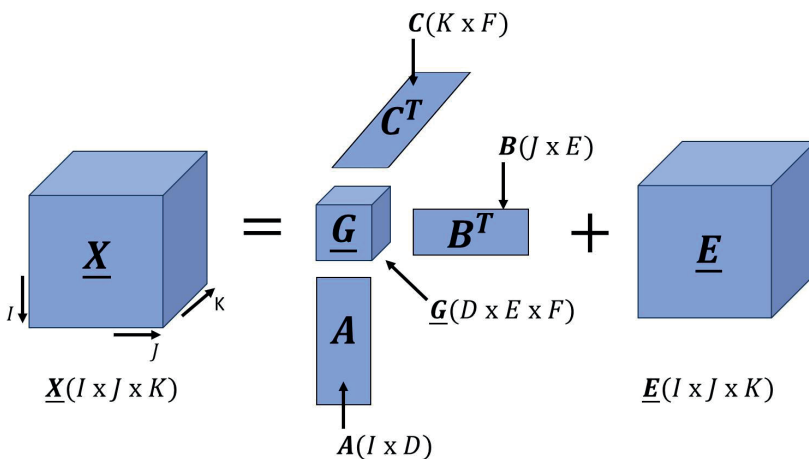


Figure 4.3: Graphical representation of the Tucker3 model.

In equation 4.13 we present the Tucker-3 equation for a single element of the data cube:

$$x_{ijk} = \left(\sum_{p=1}^P \sum_{q=1}^Q \sum_{r=1}^R a_{ip} b_{jq} c_{kr} g_{pqr} \right) + e_{ijk} \quad \text{Eq. 13}$$

Where: x_{ijk} is equivalent to an element of the data cube \mathbf{X} ; a_{ip} , b_{jq} , c_{kr} and g_{pqr} are the values corresponding to x_{ij} in matrices \mathbf{A} , \mathbf{B} and \mathbf{C} obtained for modes I , J , K ; e_{ijk} is related to the approximated error for the value of x_{ijk} .

Therefore, when we encounter 2nd order chemical data and we want to apply the LDA algorithm, there is a need for a prior data decomposition step. A viable alternative is to use the scores from the Turkey method as an input variable in the LDA algorithm, creating a new classifier, called Turkey3-LDA.

Example 8: In this example, we present the Turkey3-LDA algorithm using KS to build 2nd order multivariate classification models on normal vs. patients with colorectal cancer (CRC), obtained by molecular fluorescence spectrometry in blood plasma. The data can be obtained here: <https://ucphchemometrics.com/datasets/>. The Turkey3-LDA algorithm in this example needs the following R packages: *multiway*, *ThreeWay*, *R.matlab*, *plot3D*, *plotly*, *prospectr*, *MASS* and *caret*.

R Script

```
## Loading Packages

install.packages("multiway")
library(multiway)

install.packages("ThreeWay")
library(ThreeWay)

install.packages("R.matlab")
library(R.matlab)

install.packages("plot3D")
library(plot3D)

install.packages("plotly")
library(plotly)

install.packages("prospectr")
install.packages("MASS")
install.packages("caret")

library(prospectr)
library(MASS)
library(caret)
```

```

## Loading Data - Establish the working directory containing the CRC.mat data
## In RStudio, go to Session > Set Working Directory > Choose Directory

data <- readMat("CRC.mat")

X <- data$Xcomb
Y <- data$Y
nmEX <- data$nmEX
nmEM <- data$nmEM

## Visualizing the data

# matrix dimensions

mydim = dim(X) # samples x emission x excitation
mydim

# average matrix of each class (1 = normal, 2 = cancer)

X1 = data$Xnormal
X2 = data$Xcancer

X1m = colMeans (X1)
X2m = colMeans (X2)

dev.new()
filled.contour(X1m,color.palette = terrain.colors,main = "Class 1 - Normal")

dev.new()
filled.contour(X2m,color.palette = terrain.colors,main = "Class 2 - Cancer")

dev.new()
matplot(t(nmEM), X1m, type="l", xlab = "Emission wavelength (nm)", ylab =
"Intensity", main = "Class 1 - Normal")

dev.new()
matplot(t(nmEM), t(X1m), type="l", xlab ="Excitation Wavelength (nm)", ylab
="Intensity", main = "Class 1 - Normal")

dev.new()
matplot(t(nmEM), X2m, type="l", xlab ="Emission wavelength (nm)", ylab
="Intensity", main ="Class 2 - Cancer")

dev.new()
matplot(t(nmEM), t(X2m), type="l", xlab ="Excitation Wavelength (nm)", ylab
="Intensity", main ="Class 2 - Cancer")

##### TUCKER3 model #####

```

```

nf = 3 # define number of factors

Xr = matrix(X, nrow = mydim[1])

model = T3func(Xr, mydim[1], mydim[2], mydim[3], nf, nf, nf, 0, 1e-6)

# R2 Adjustment

model$fp # Increase the number of factors to better adjust R2

## plot tucker3 scores 1 x 2

dev.new ()
matplot(model$A[Y==1,1], model$A[Y==1,2], pch="o", col="blue", xlab="Factor 1",
ylab="Factor 2", main="Scores (o: normal, x: cancer)")
points(model$A[Y==2,1], model$A[Y==2,2], pch="x", col="red", xlab="Factor 1",
ylab="Factor 2", main="Scores (o: normal, x: cancer)")

## plot tucker3 loadings - emission

dev.new ()
matplot(t(nmEM), model$B, type = "l", xlab = "Emission Wavelength (nm)", ylab =
"Intensity")

## plot tucker3 loadings - excitation

dev.new()
matplot(t(nmEX), model$C, type = "l", xlab = "Excitation Wavelength (nm)", ylab =
"Intensity")

##### selection of training and testing samples based on KS

perc = 0.7 # 70% for training

dim_class1 = dim(X1)
dim_class2 = dim(X2)
dim_data = dim(X)

ntrain1 = ceiling(perc * dim_class1[1]) # number of training samples class 1
ntrain2 = ceiling(perc * dim_class2[1]) # number of training samples class 2

scores_X1 = model$A[1:dim_class1[1],] # scores class 1
scores_X2 = model$A[(dim_class1[1]+ 1): dim_data[1],] # scores class 2

sel1 = kenStone(scores_X1, k = ntrain1) # KS class 1
sel2 = kenStone(scores_X2, k = ntrain2) # KS class 2

train1 = scores_X1[sel1$model,] # training class 1
train2 = scores_X2[sel2$model,] # training class 2
train = rbind(train1,train2) # joining training matrices

```

```

group1 = rep(1,dim_class1[1]) # class 1
group2 = rep(2,dim_class2[1]) # class 2

group1train = matrix(group1[sel1$model]) # class 1 training labels
group2train = matrix(group2[sel2$model]) # class 2 training labels
group_train = rbind(group1train,group2train) # training labels

test1 = scores_X1[sel1$test,] # test class 1
test2 = scores_X1[sel2$test,] # test class 2
test = rbind (test1,test2) # joining test matrices

group1test = matrix(group1[sel1$test]) # class 1 test labels
group2test = matrix(group2[sel2$test]) # class 2 test labels
group_test = rbind(group1test,group2test) # test labels

# LDA model

model_lda = lda(train,group_train ) # model without cross-validation
model_lda_cv = lda(train,group_train,CV=TRUE) # model with leave-one-out cross-
validation

# prediction of training and testing samples

pred_train = predict(model_lda, train) # training
pred_test = predict(model_lda, test) # test

# figures of merit

# training

dim_train1 = dim(train1)
dim_train2 = dim(train2)

ac_train = mean(pred_train$class == group_train) # accuracy
spec_train = mean(pred_train$class[1:dim_train1[1]]==group1train) # specificity
sens_train = mean(pred_train$class[(dim_train1[1] + 1):(dim_train1[1]+dim_
train2[1])]==group2train) # sensitivity

# cross-validation

ac_cv = mean(model_lda_cv$class == group_train) # accuracy
spec_cv = mean(model_lda_cv$class[1:dim_train1[1]]==group1train) # specificity
sens_cv = mean(model_lda_cv$class[(dim_train1[1] + 1):(dim_train1[1]+dim_
train2[1])]==group2train) # sensitivity

# test

dim_test1 = dim(test1)
dim_test2 = dim(test2)

```



```

ac_test = mean(pred_test$class == group_test) # accuracy
spec_test = mean(pred_test$class[1:dim_test1[1]]==group1test) # specificity
sens_test = mean(pred_test$class[(dim_test1[1]+ 1):(dim_test1[1]+dim_
test2[1]])==group2test) # sensitivity

print("==== Training =====")
cat("Accuracy:")
ac_train
cat("Sensitivity:")
sens_train
cat("Specificity:")
spec_train

print("==== Cross-validation =====")
cat("Accuracy:")
ac_cv
cat("Sensitivity:")
sens_cv
cat("Specificity:")
spec_cv

print("==== Test =====")
cat("Accuracy:")
ac_test
cat("Sensitivity:")
sens_test
cat("Specificity:")
spec_test

# viewing posterior probabilities - training

dev.new()
matplot(pred_train$posterior[1:dim_train1[1],1 ],pch=19,col="blue",xlab="Samples",
ylab ="LD1", main ="Posterior Probability - Training") # training class 1
points(pred_train$posterior[(dim_train1[ 1]+ 1):(dim_train1[1]+dim_
train2[1]),1],pch=19,col="red",xlab="Samples", ylab ="LD1", main ="Posterior
Probability - Training") # training class 2

# visualizing posterior probabilities - cross validation

dev.new()
matplot(model_lda_cv$posterior[1:dim_train1[1],1 ],pch
=19,col="blue",xlab="Samples", ylab ="LD1", main ="Posterior Probability - Cross-
Validation") # class 1 cross-validation
points(pred_train$posterior[(dim_train1[ 1]+ 1):(dim_train1[1]+dim_
train2[1]),1],pch=19,col="red",xlab="Samples", ylab ="LD1", main ="Posterior
Probability - Cross-validation") # cross-validation class 2

# viewing posterior probabilities - test

```

```

dev.new()
matplot(pred_test$posterior[1:dim_test1[1],1],pch=19,col="blue",xlab="Samples",
ylab = "LD1", main=" Posterior Probability - Test") # test class 1
points(pred_train$posterior[(dim_test1[ 1]+ 1):(dim_test1[1]+dim_
test2[1]),1],pch=19,col="red",xlab="Samples", ylab = "LD1", main="Posterior
Probability - Test") # test class 2

```

Example 9: Here, the Turkey3-LDA algorithm is described using the MLM algorithm for sample selection in 2nd order multivariate classification models on normal vs. patients with colorectal cancer (CRC), obtained by molecular fluorescence spectrometry in blood plasma. Data can be obtained here: <https://ucphchemometrics.com/datasets/> . The Turkey3-LDA algorithm in this example needs the following R packages: *multiway*, *ThreeWay*, *R.matlab*, *plot3D*, *plotly*, *prospectr*, *MASS* and *caret* .

R Script

```

## Loading Packages

install.packages("multiway")
library(multiway)

install.packages("ThreeWay")
library(ThreeWay)

install.packages("R.matlab")
library(R.matlab)

install.packages("plot3D")
library(plot3D)

install.packages("plotly")
library(plotly)

install.packages("prospectr")
install.packages("MASS")
install.packages("caret")

library(prospectr)
library(MASS)
library(caret)

## Loading Data - Establish the working directory containing the CRC.mat data
## In RStudio, go to Session > Set Working Directory > Choose Directory

data <- readMat("CRC.mat")

```

```

X <- data$Xcomb
Y <- data$Y
nmEX <- data$nmEX
nmEM <- data$nmEM

## Visualizing the data

# matrix dimensions

mydim = dim(X) # samples x emission x excitation
mydim

# average matrix of each class (1 = normal, 2 = cancer)

X1 = data$Xnormal
X2 = data$Xcancer

X1m = colMeans (X1)
X2m = colMeans (X2)

dev.new()
filled.contour(X1m,color.palette = terrain.colors,main = "Class 1 - Normal")

dev.new()
filled.contour(X2m,color.palette = terrain.colors,main = "Class 2 - Cancer")

dev.new()
matplot(t(nmEM), X1m, type="l", xlab = "Emission wavelength (nm)", ylab = "Intensity",
main = "Class 1 - Normal")

dev.new()
matplot(t(nmEX), t(X1m), type = "l", xlab = "Excitation Wavelength (nm)", ylab
="Intensity", main = "Class 1 - Normal")

dev.new()
matplot(t(nmEM), X2m, type="l", xlab = "Emission wavelength (nm)", ylab = "Intensity",
main = "Class 2 - Cancer")

dev.new()
matplot(t(nmEX), t(X2m), type = "l", xlab = "Excitation Wavelength (nm)", ylab
="Intensity", main = "Class 2 - Cancer")

##### TUCKER3 model #####

nf = 3 # define number of factors

Xr = matrix(X, nrow = mydim[1])

model = T3func(Xr, mydim[1], mydim[2], mydim[3], nf, nf, nf, 0, 1e-6)

```

```

# R2 Adjustment

model$fp # Increase the number of factors to better adjust R2

## plot tucker3 scores 1 x 2

dev.new ()
matplot(model$A[Y==1,1], model$A[Y==1,2], pch="o", col="blue", xlab="Factor 1",
ylab="Factor 2", main="Scores (o: normal, x: cancer)")
points(model$A[Y==2,1], model$A[Y==2,2], pch="x", col="red", xlab="Factor 1",
ylab="Factor 2", main="Scores (o: normal, x: cancer)")

## plot tucker3 loadings - emission

dev.new ()
matplot(t(nmEM), model$B, type ="l", xlab = "Emission Wavelength (nm)", ylab =
"Intensity")

## plot tucker3 loadings - excitation

dev.new()
matplot(t(nmEX), model$C, type ="l", xlab = "Excitation Wavelength (nm)", ylab =
"Intensity")

##### training and testing samples selection based on MLM

perc = 0.7 # 70% for training

dim_class1 = dim(X1)
dim_class2 = dim(X2)
dim_data = dim(X)

ntrain1 = ceiling(perc * dim_class1[1]) # number of training samples class 1
ntrain2 = ceiling(perc * dim_class2[1]) # number of training samples class 2

scores_X1 = model$A[1:dim_class1[1],] # scores class 1
scores_X2 = model$A[(dim_class1[1]+ 1): dim_data[1],] # scores class 2

sel1 = kenStone(scores_X1, k = ntrain1) # KS class 1
sel2 = kenStone(scores_X2, k = ntrain2) # KS class 2

train1 = scores_X1[sel1$model,] # training class 1
train2 = scores_X2[sel2$model,] # training class 2
train = rbind(train1,train2) # joining training matrices

group1 = rep(1,dim_class1[1]) # class 1
group2 = rep(2,dim_class2[1]) # class 2

```

```

group1train = matrix(group1[sel1$model]) # class 1 training labels
group2train = matrix(group2[sel2$model]) # class 2 training labels
group_train = rbind(group1train,group2train) # training labels

test1 = scores_X1[sel1$test,] # test class 1
test2 = scores_X1[sel2$test,] # test class 2
test = rbind (test1,test2) # joining test matrices

group1test = matrix(group1[sel1$test]) # class 1 test labels
group2test = matrix(group2[sel2$test]) # class 2 test labels
group_test = rbind(group1test,group2test) # test labels

dim_train1 = dim(train1) # class 1 training dimension
dim_train2 = dim(train2) # class 2 training dimension
dim_test1 = dim(test1) # test dimension of class 1
dim_test2 = dim(test2) # class 2 test dimension

prob = 0.2 # MLM mutation probability = 20%

p1 = ceiling(prob * dim_test1[1])
p2 = ceiling(prob * dim_test2[1])

t1 = 1:dim_train1[1]
t2 = 1:dim_train2[1]

v1 = 1:dim_test1[1]
v2 = 1:dim_test2[1]

train1_sub = sample(x=t1, size=p1)
train2_sub = sample(x=t2, size=p2)

test1_sub = sample(x=v1, size=p1)
test2_sub = sample(x=v2, size=p2)

train1_new = rbind(train1[-train1_sub,],test1[test1_sub,])
train2_new = rbind(train2[-train2_sub,],test2[test2_sub,])

test1_new = rbind(test1[-test1_sub,],train1[train1_sub,])
test2_new = rbind(test2[-test2_sub,],train2[train2_sub,])

train = rbind(train1_new,train2_new)
test = rbind(test1_new,test2_new)

# LDA model

model_lda = lda(train,group_train ) # model without cross-validation
model_lda_cv = lda(train,group_train,CV=TRUE) # model with leave-one-out cross-
validation

# prediction of training and testing samples

pred_train = predict(model_lda, train) # training
pred_test = predict(model_lda, test) # test

```

```

# figures of merit

# training

dim_train1 = dim(train1)
dim_train2 = dim(train2)

ac_train = mean(pred_train$class == group_train) # accuracy
spec_train = mean(pred_train$class[1:dim_train1[1]]==group1train) # specificity
sens_train = mean(pred_train$class[(dim_train1[1] + 1):(dim_train1[1]+dim_
train2[1])]==group2train) # sensitivity

# cross-validation

ac_cv = mean(model_lda_cv$class == group_train) # accuracy
spec_cv = mean(model_lda_cv$class[1:dim_train1[1]]==group1train) # specificity
sens_cv = mean(model_lda_cv$class[(dim_train1[1] + 1):(dim_train1[1]+dim_
train2[1])]==group2train) # sensitivity

# test

dim_test1 = dim(test1)
dim_test2 = dim(test2)

ac_test = mean(pred_test$class == group_test) # accuracy
spec_test = mean(pred_test$class[1:dim_test1[1]]==group1test) # specificity
sens_test = mean(pred_test$class[(dim_test1[1]+ 1):(dim_test1[1]+dim_
test2[1])]==group2test) # sensitivity

print("==== Training =====")
cat("Accuracy:")
ac_train
cat("Sensitivity:")
sens_train
cat("Specificity:")
spec_train

print("==== Cross-validation =====")
cat("Accuracy:")
ac_cv
cat("Sensitivity:")
sens_cv
cat("Specificity:")
spec_cv

print("==== Test =====")
cat("Accuracy:")
ac_test
cat("Sensitivity:")
sens_test
cat("Specificity:")
spec_test

```

```

# viewing posterior probabilities - training

dev.new()
matplot(pred_train$posterior[1:dim_train1[1],1 ],pch=19,col="blue",xlab="Samples",
ylab ="LD1", main ="Posterior Probability - Training") # training class 1
points(pred_train$posterior[(dim_train1[ 1]+ 1):(dim_train1[1]+dim_
train2[1]),1],pch=19,col="red",xlab="Samples", ylab ="LD1", main ="Posterior
Probability - Training") # training class 2

# visualizing posterior probabilities - cross validation

dev.new()
matplot(model_lda_cv$posterior[1:dim_train1[1],1 ],pch
=19,col="blue",xlab="Samples", ylab ="LD1", main ="Posterior Probability - Cross-
Validation") # class 1 cross-validation
points(pred_train$posterior[(dim_train1[ 1]+ 1):(dim_train1[1]+dim_
train2[1]),1],pch=19,col="red",xlab="Samples", ylab ="LD1", main ="Posterior
Probability - Cross-validation") # cross-validation class 2

# viewing posterior probabilities - test

dev.new()
matplot(pred_test$posterior[1:dim_test1[1],1],pch=19,col="blue",xlab="Samples",
ylab ="LD1", main=" Posterior Probability - Test") # test class 1
points(pred_train$posterior[(dim_test1[ 1]+ 1):(dim_test1[1]+dim_
test2[1]),1],pch=19,col="red",xlab="Samples", ylab ="LD1", main="Posterior
Probability - Test") # test class 2

```

4.9 PARAFAC–LDA

PARAFAC ("PARAllel Factor analysis") consists of a trilinear method of higher order data decomposition proposed by Professor Rasmus Bro in 1998 [9]. From a mathematical point of view, PARAFAC can be considered as a generalization of PCA, or as a restricted case of the Tucker-3 method. The PARAFAC model is formed by two weight matrices (**B** and **C**) and one of scores (**A**), in a mathematical representation very similar to the Tucker-3 method, as we can see in equation 4.14:

$$\underline{\mathbf{X}} = \mathbf{A}(\mathbf{C}|\otimes|\mathbf{B})^t + \mathbf{E}$$

Eq. 14

Where **A**, **B** and **C** have dimensions $I \times F$, $J \times F$ and $K \times F$, respectively; $|\otimes|$ is the Khatri-Rao operator and **E** is the residue tensor with the same dimensions as **X**. In the PARAFAC model, the tensor **G** appears, which is a hyperidentity whose value is 1 when $d=I=h$ and zero for all other positions. Figure 4.4 presents a graphical representation of 2nd order data with PARAFAC:

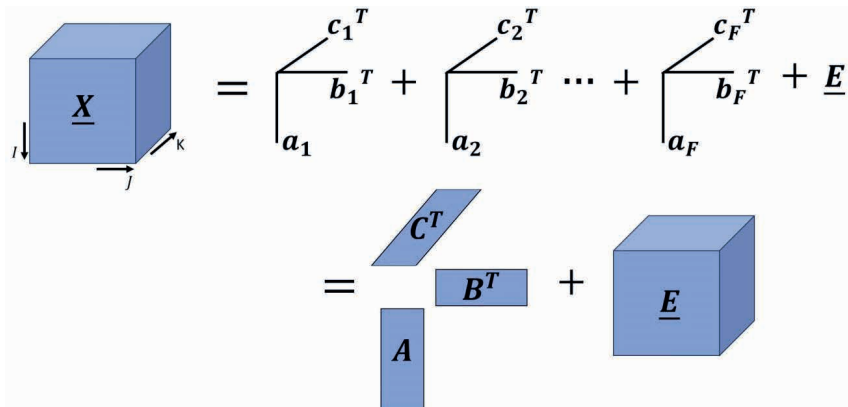


Figure 4.3: Graphical representation of the PARAFAC model through the decomposition of a three-dimensional data array into F triads of weight vectors.

Therefore, analogously to Turkey-3, we will apply the LDA algorithm after obtaining the scores from the PARAFAC method as an input variable, creating a new classifier, called PARAFAC-LDA.

Example 10 : In this example, we present the PARAFAC-LDA algorithm using KS to build 2nd order multivariate classification models on normal vs. patients with colorectal cancer (CRC), obtained by molecular fluorescence spectrometry in blood plasma. The data can be obtained here: <https://ucphchemometrics.com/datasets/>. The PARAFAC-LDA algorithm in this example needs the following R packages: *multiway*, *ThreeWay*, *R.matlab*, *plot3D*, *plotly*, *prospectr*, *MASS*, *caret*.

R Script

```
## Loading Packages

install.packages("multiway")
library(multiway)

install.packages("ThreeWay")
library(ThreeWay)

install.packages("R.matlab")
library(R.matlab)

install.packages("plot3D")
library(plot3D)

install.packages("plotly")
library(plotly)
```



```

install.packages("prospectr")
install.packages("MASS")
install.packages("caret")

library(prospectr)
library(MASS)
library(caret)

## Loading Data - Establish the working directory containing the CRC.mat data
## In RStudio, go to Session > Set Working Directory > Choose Directory

data <- readMat("CRC.mat")

X <- data$Xcomb
Y <- data$Y
nmEX <- data$nmEX
nmEM <- data$nmEM

## Visualizing the data

# matrix dimensions

mydim = dim(X) # samples x emission x excitation
mydim

# average matrix of each class (1 = normal, 2 = cancer)

X1 = data$Xnormal
X2 = data$Xcancer

X1m = colMeans (X1)
X2m = colMeans (X2)

dev.new()
filled.contour(X1m,color.palette = terrain.colors,main = "Class 1 - Normal")

dev.new()
filled.contour(X2m,color.palette = terrain.colors,main = "Class 2 - Cancer")

dev.new()
matplot(t(nmEM), X1m, type="l", xlab = "Emission wavelength (nm)", ylab = "Intensity",
main = "Class 1 - Normal")

dev.new()
matplot(t(nmEM), t(X1m), type = "l", xlab = "Excitation Wavelength (nm)", ylab
="Intensity", main = "Class 1 - Normal")

dev.new()
matplot(t(nmEM), X2m, type="l", xlab = "Emission wavelength (nm)", ylab = "Intensity",
main = "Class 2 - Cancer")

```

```

dev.new()
matplot(t(nmEX), t(X2m), type = "l", xlab = "Excitation Wavelength (nm)", ylab
="Intensity", main = "Class 2 - Cancer")

##### PARAFAC Model #####

nf = 3 # define number of factors

model = parafac(X,nfac=nf,nstart=1,maxit=500,ctol=10^-4,parallel=FALSE,cl=NULL,ou
tput=c("best","all"))

# R2 Adjustment

model$Rsq # Increase the number of factors to better adjust R2

## plot parafac scores 1 x 2

dev.new ()
matplot(model$A[Y==1,1], model$A[Y==1,2], pch="o", col="blue", xlab="Factor 1",
ylab="Factor 2", main="Scores (o: normal, x: cancer)")
points(model$A[Y==2,1], model$A[Y==2,2], pch="x", col="red", xlab="Factor 1",
ylab="Factor 2", main="Scores (o: normal, x: cancer)")

## plot parafac loadings - emission

dev.new ()
matplot(t(nmEM), model$B, type = "l", xlab = "Emission Wavelength (nm)", ylab =
"Intensity")

## plot parafac loadings - excitation

dev.new()
matplot(t(nmEX), model$C, type = "l", xlab = "Excitation Wavelength (nm)", ylab =
"Intensity")

##### selection of training and testing samples based on KS

perc = 0.7 # 70% for training

dim_class1 = dim(X1)
dim_class2 = dim(X2)
dim_data = dim(X)

ntrain1 = ceiling(perc * dim_class1[1]) # number of training samples class 1
ntrain2 = ceiling(perc * dim_class2[1]) # number of training samples class 2

scores_X1 = model$A[1:dim_class1[1],] # scores class 1
scores_X2 = model$A[(dim_class1[1]+ 1): dim_data[1],] # scores class 2

```

```

sel1 = kenStone(scores_X1, k = ntrain1) # KS class 1
sel2 = kenStone(scores_X2, k = ntrain2) # KS class 2

train1 = scores_X1[sel1$model,] # training class 1
train2 = scores_X2[sel2$model,] # training class 2
train = rbind(train1,train2) # joining training matrices

group1 = rep(1,dim_class1[1]) # class 1
group2 = rep(2,dim_class2[1]) # class 2

group1train = matrix(group1[sel1$model]) # class 1 training labels
group2train = matrix(group2[sel2$model]) # class 2 training labels
group_train = rbind(group1train,group2train) # training labels

test1 = scores_X1[sel1$test,] # test class 1
test2 = scores_X1[sel2$test,] # test class 2
test = rbind (test1,test2) # joining test matrices

group1test = matrix(group1[sel1$test]) # class 1 test labels
group2test = matrix(group2[sel2$test]) # class 2 test labels
group_test = rbind(group1test,group2test) # test labels

# LDA model

model_lda = lda(train,group_train ) # model without cross-validation
model_lda_cv = lda(train,group_train,CV=TRUE) # model with leave-one-out cross-
validation

# prediction of training and testing samples

pred_train = predict(model_lda, train) # training
pred_test = predict(model_lda, test) # test

# figures of merit

# training

dim_train1 = dim(train1)
dim_train2 = dim(train2)

ac_train = mean(pred_train$class == group_train) # accuracy
spec_train = mean(pred_train$class[1:dim_train1[1]]==group1train) # specificity
sens_train = mean(pred_train$class[(dim_train1[1] + 1):(dim_train1[1]+dim_
train2[1])]==group2train) # sensitivity

# cross-validation

ac_cv = mean(model_lda_cv$class == group_train) # accuracy
spec_cv = mean(model_lda_cv$class[1:dim_train1[1]]==group1train) # specificity
sens_cv = mean(model_lda_cv$class[(dim_train1[1] + 1):(dim_train1[1]+dim_
train2[1])]==group2train) # sensitivity

```

```

# test

dim_test1 = dim(test1)
dim_test2 = dim(test2)

ac_test = mean(pred_test$class == group_test) # accuracy
spec_test = mean(pred_test$class[1:dim_test1[1]]==group1test) # specificity
sens_test = mean(pred_test$class[(dim_test1[1]+ 1):(dim_test1[1]+dim_
test2[1]])==group2test) # sensitivity

print("==== Training =====")
cat("Accuracy:")
ac_train
cat("Sensitivity:")
sens_train
cat("Specificity:")
spec_train

print("==== Cross-validation =====")
cat("Accuracy:")
ac_cv
cat("Sensitivity:")
sens_cv
cat("Specificity:")
spec_cv

print("==== Test =====")
cat("Accuracy:")
ac_test
cat("Sensitivity:")
sens_test
cat("Specificity:")
spec_test

# viewing posterior probabilities - training

dev.new()
matplot(pred_train$posterior[1:dim_train1[1],1 ],pch=19,col="blue",xlab="Samples",
ylab ="LD1", main ="Posterior Probability - Training") # training class 1
points(pred_train$posterior[(dim_train1[ 1]+ 1):(dim_train1[1]+dim_
train2[1]),1],pch=19,col="red",xlab="Samples", ylab ="LD1", main ="Posterior
Probability - Training") # training class 2

# visualizing posterior probabilities - cross validation

dev.new()
matplot(model_lda_cv$posterior[1:dim_train1[1],1 ],pch
=19,col="blue",xlab="Samples", ylab ="LD1", main ="Posterior Probability - Cross-
Validation") # class 1 cross-validation
points(pred_train$posterior[(dim_train1[ 1]+ 1):(dim_train1[1]+dim_
train2[1]),1],pch=19,col="red",xlab="Samples", ylab ="LD1", main ="Posterior
Probability - Cross-validation") # cross-validation class 2

```

```
# viewing posterior probabilities - test

dev.new()
matplot(pred_test$posterior[1:dim_test1[1],1],pch=19,col="blue",xlab="Samples",
ylab = "LD1", main=" Posterior Probability - Test") # test class 1
points(pred_train$posterior[(dim_test1[ 1]+ 1):(dim_test1[1]+dim_
test2[1]),1],pch=19,col="red",xlab="Samples", ylab = "LD1", main="Posterior
Probability - Test") # test class 2
```

Example 11: Here, the PARAFAC-LDA algorithm is described using the MLM algorithm for sample selection in 2nd order multivariate classification models on normal vs. patients with colorectal cancer (CRC), obtained by molecular fluorescence spectrometry in blood plasma. Data can be obtained here: <https://ucphchemometrics.com/datasets/> . The PARAFAC-LDA algorithm in this example needs the following R packages: *multiway*, *ThreeWay*, *R.matlab*, *plot3D*, *plotly*, *prospectr*, *MASS* and *caret* .

R Script

```
## Loading Packages

install.packages("multiway")
library(multiway)

install.packages("ThreeWay")
library(ThreeWay)

install.packages("R.matlab")
library(R.matlab)

install.packages("plot3D")
library(plot3D)

install.packages("plotly")
library(plotly)

install.packages("prospectr")
install.packages("MASS")
install.packages("caret")

library(prospectr)
library(MASS)
library(caret)

## Loading Data - Establish the working directory containing the CRC.mat data
## In RStudio, go to Session > Set Working Directory > Choose Directory

data <- readMat("CRC.mat")
```

```

X <- data$Xcomb
Y <- data$Y
nmEX <- data$nmEX
nmEM <- data$nmEM

## Visualizing the data

# matrix dimensions

mydim = dim(X) # samples x emission x excitation
mydim

# average matrix of each class (1 = normal, 2 = cancer)

X1 = data$Xnormal
X2 = data$Xcancer

X1m = colMeans (X1)
X2m = colMeans (X2)

dev.new()
filled.contour(X1m,color.palette = terrain.colors,main = "Class 1 - Normal")

dev.new()
filled.contour(X2m,color.palette = terrain.colors,main = "Class 2 - Cancer")

dev.new()
matplot(t(nmEM), X1m, type="l", xlab = "Emission wavelength (nm)", ylab = "Intensity",
main = "Class 1 - Normal")

dev.new()
matplot(t(nmEX), t(X1m), type = "l", xlab = "Excitation Wavelength (nm)", ylab
="Intensity", main = "Class 1 - Normal")

dev.new()
matplot(t(nmEM), X2m, type="l", xlab = "Emission wavelength (nm)", ylab = "Intensity",
main = "Class 2 - Cancer")

dev.new()
matplot(t(nmEX), t(X2m), type = "l", xlab = "Excitation Wavelength (nm)", ylab
="Intensity", main = "Class 2 - Cancer")

##### PARAFAC Model #####

nf = 3 # define number of factors

model = parafac(X,nfac=nf,nstart=1,maxit=500,ctol=10^-4,parallel=FALSE,cl=NULL,ou
tput=c("best","all"))

```

```

# R2 Adjustment

model$Rsq # Increase the number of factors to better adjust R2

## plot parafac scores 1 x 2

dev.new ()
matplot(model$A[Y==1,1], model$A[Y==1,2], pch="o", col="blue", xlab="Factor 1",
ylab="Factor 2", main="Scores (o: normal, x: cancer)")
points(model$A[Y==2,1], model$A[Y==2,2], pch="x", col="red", xlab="Factor 1",
ylab="Factor 2", main="Scores (o: normal, x: cancer)")

## plot parafac loadings - emission

dev.new ()
matplot(t(nmEM), model$B, type ="l", xlab = "Emission Wavelength (nm)", ylab =
"Intensity")

## plot parafac loadings - excitation

dev.new()
matplot(t(nmEX), model$C, type ="l", xlab = "Excitation Wavelength (nm)", ylab =
"Intensity")

##### training and testing samples selection based on MLM

perc = 0.7 # 70% for training

dim_class1 = dim(X1)
dim_class2 = dim(X2)
dim_data = dim(X)

ntrain1 = ceiling(perc * dim_class1[1]) # number of training samples class 1
ntrain2 = ceiling(perc * dim_class2[1]) # number of training samples class 2

scores_X1 = model$A[1:dim_class1[1],] # scores class 1
scores_X2 = model$A[(dim_class1[1]+ 1): dim_data[1],] # scores class 2

sel1 = kenStone(scores_X1, k = ntrain1) # KS class 1
sel2 = kenStone(scores_X2, k = ntrain2) # KS class 2

train1 = scores_X1[sel1$model,] # training class 1
train2 = scores_X2[sel2$model,] # training class 2
train = rbind(train1,train2) # joining training matrices

group1 = rep(1,dim_class1[1]) # class 1
group2 = rep(2,dim_class2[1]) # class 2

group1train = matrix(group1[sel1$model]) # class 1 training labels
group2train = matrix(group2[sel2$model]) # class 2 training labels
group_train = rbind(group1train,group2train) # training labels

```

```

test1 = scores_X1[sel1$test,] # test class 1
test2 = scores_X1[sel2$test,] # test class 2
test = rbind (test1,test2) # joining test matrices

group1test = matrix(group1[sel1$test]) # class 1 test labels
group2test = matrix(group2[sel2$test]) # class 2 test labels
group_test = rbind(group1test,group2test) # test labels

dim_train1 = dim(train1) # class 1 training dimension
dim_train2 = dim(train2) # class 2 training dimension
dim_test1 = dim(test1) # test dimension of class 1
dim_test2 = dim(test2) # class 2 test dimension

prob = 0.2 # MLM mutation probability = 20%

p1 = ceiling(prob * dim_test1[1])
p2 = ceiling(prob * dim_test2[1])

t1 = 1:dim_train1[1]
t2 = 1:dim_train2[1]

v1 = 1:dim_test1[1]
v2 = 1:dim_test2[1]

train1_sub = sample(x=t1, size=p1)
train2_sub = sample(x=t2, size=p2)

test1_sub = sample(x=v1, size=p1)
test2_sub = sample(x=v2, size=p2)

train1_new = rbind(train1[-train1_sub,],test1[test1_sub,])
train2_new = rbind(train2[-train2_sub,],test2[test2_sub,])

test1_new = rbind(test1[-test1_sub,],train1[train1_sub,])
test2_new = rbind(test2[-test2_sub,],train2[train2_sub,])

train = rbind(train1_new,train2_new)
test = rbind(test1_new,test2_new)

# LDA model

model_lda = lda(train,group_train ) # model without cross-validation
model_lda_cv = lda(train,group_train,CV=TRUE) # model with leave-one-out cross-
validation

# prediction of training and testing samples

pred_train = predict(model_lda, train) # training
pred_test = predict(model_lda, test) # test

# figures of merit

```



```

# training

dim_train1 = dim(train1)
dim_train2 = dim(train2)

ac_train = mean(pred_train$class == group_train) # accuracy
spec_train = mean(pred_train$class[1:dim_train1[1]]==group1train) # specificity
sens_train = mean(pred_train$class[(dim_train1[1] + 1):(dim_train1[1]+dim_
train2[1])]==group2train) # sensitivity

# cross-validation

ac_cv = mean(model_lda_cv$class == group_train) # accuracy
spec_cv = mean(model_lda_cv$class[1:dim_train1[1]]==group1train) # specificity
sens_cv = mean(model_lda_cv$class[(dim_train1[1] + 1):(dim_train1[1]+dim_
train2[1])]==group2train) # sensitivity

# test

dim_test1 = dim(test1)
dim_test2 = dim(test2)

ac_test = mean(pred_test$class == group_test) # accuracy
spec_test = mean(pred_test$class[1:dim_test1[1]]==group1test) # specificity
sens_test = mean(pred_test$class[(dim_test1[1]+ 1):(dim_test1[1]+dim_
test2[1])]==group2test) # sensitivity

print("==== Training =====")
cat("Accuracy:")
ac_train
cat("Sensitivity:")
sens_train
cat("Specificity:")
spec_train

print("==== Cross-validation =====")
cat("Accuracy:")
ac_cv
cat("Sensitivity:")
sens_cv
cat("Specificity:")
spec_cv

print("==== Test =====")
cat("Accuracy:")
ac_test
cat("Sensitivity:")
sens_test
cat("Specificity:")
spec_test

# viewing posterior probabilities - training

```

```

dev.new()
matplot(pred_train$posterior[1:dim_train1[1],1 ],pch=19,col="blue",xlab="Samples",
ylab ="LD1", main ="Posterior Probability - Training") # training class 1
points(pred_train$posterior[(dim_train1[ 1]+ 1):(dim_train1[1]+dim_
train2[1]),1],pch=19,col="red",xlab="Samples", ylab ="LD1", main ="Posterior
Probability - Training") # training class 2

# visualizing posterior probabilities - cross validation

dev.new()
matplot(model_lda_cv$posterior[1:dim_train1[1],1 ],pch
=19,col="blue",xlab="Samples", ylab ="LD1", main ="Posterior Probability - Cross-
Validation") # class 1 cross-validation
points(pred_train$posterior[(dim_train1[ 1]+ 1):(dim_train1[1]+dim_
train2[1]),1],pch=19,col="red",xlab="Samples", ylab ="LD1", main ="Posterior
Probability - Cross-validation") # cross-validation class 2

# viewing posterior probabilities - test

dev.new()
matplot(pred_test$posterior[1:dim_test1[1],1],pch=19,col="blue",xlab="Samples",
ylab ="LD1", main=" Posterior Probability - Test") # test class 1
points(pred_train$posterior[(dim_test1[ 1]+ 1):(dim_test1[1]+dim_
test2[1]),1],pch=19,col="red",xlab="Samples", ylab ="LD1", main="Posterior
Probability - Test") # test class 2

```

PROPOSED EXERCISES

01 – Propose the application of the PCA-LDA algorithm using some sample selection method (KS or MLM) through a script in the R language on an experimental data set, presenting your hypotheses and conclusions.

02 – Propose the application of the SPA-LDA algorithm using some sample selection method (KS or MLM) through a script in the R language on an experimental data set, presenting your hypotheses and conclusions.

03 – Propose the application of the GA-LDA algorithm using some sample selection method (KS or MLM) through an R script on an experimental data set, presenting your hypotheses and conclusions.

04 – Build multivariate classification models for the PCA-QDA, SPA-QDA and GA-QDA algorithms using an R script from a data set and present your conclusions.

05 – Perform a comparison between the performance of PCA-LDA and PCA-QDA models using an R script for a given data set. Choose a sample selection method.

06 – Perform a comparison between the performance of SPA-LDA and SPA-QDA models using an R script for a given data set. Choose a sample selection method.

07 – Perform a comparison between the performance of GA-LDA and GA-QDA models using an R script for a given data set. Choose a sample selection method.

08 – Build 2nd order multivariate classification models based on the Turkey3-LDA algorithm with both KS and MLM for a given dataset and present your main results.

09 – Build 2nd order multivariate classification models based on the PARAFAC-LDA algorithm with both KS and MLM for a given dataset and present your main results.

10 – Implement the QDA algorithm in the Turkey3 and PARAFAC algorithms and, based on previous exercises that used LDA models, present a comparison of results using QDA.

REFERENCES

1 – Scandar , GM; Olivieri, A.C. (2014). Practical three way calibration. Elsevier.

2 – Kennard, R.; Stone, L. (1969). Computer Aided Design of Experiments. *Technometrics* , 11(1): 137–148.

3 – Morais, CLM; Lima, KMG; Martin, F. (2018). A computational protocol for sample selection in biological-derived infrared spectroscopy datasets using Morais-Lima-Martin (MLM) algorithms. *Protocol Exchange*.

4 – Araújo, MCU; Saldanha, TCB; Galvão, RKH; Yoneyama , T.; Charm, HC; Visani , V., (2001). The successive projections algorithm for variable selection in spectroscopic multicomponent analysis, *Chemometrics and Intelligent Laboratory Systems*, 57: 65 – 73.

5 – Pontes, MJC; Galvão, RKH; Araújo, MCU; Moreira, PNT; Neto, ODP; Jose, GE; Saldanha, TCBS (2005). The successive projections algorithm for spectral variable selection in classification problems. *Chemometrics and Intelligent Laboratory Systems*. 78:11-18.

6 – Holland, JH (1975). Adaptation in natural and artificial systems. The University of Michigan Press, Ann Arbor, MI.

7 – Tucker, L. R. (1966). Some mathematical notes on three-mode factor analysis. *Psychometrika*, 31, 279-311.

8 – Graham, A.; Kronecker Products and Matrix Calculus with Applications, Wiley: New York, 1981.

9 – Bro, R.; (1998). Multi-way Analysis in Food Industry: Models, Algorithms, and Applications. Doctoral Thesis, University of Amsterdam, Netherlands.

HIGHER ORDER MULTIVARIATE CALIBRATION



"Model building is the art of selecting those aspects of a process that are relevant to the question being asked. As with any art, this selection is guided by taste, elegance, and metaphor; it is a matter of induction, rather than deduction. High science depends on this art. " **John Henry Holland (1929-2015)**

CHAPTER IDEA

Calibration is, in general, an experimental procedure that builds a mathematical model between the values indicated by a given measuring instrument and the values represented by a reference standard or property of interest. Prediction is a process that uses the model built in the calibration or training stage to predict the property of interest of a sample based on instrumental information. The general process of a calibration basically consists of the modeling (training) stage, which establishes a mathematical relationship between the instrumental variables and the response in the calibration set; and, the validation stage, which seeks to optimize the relationship to find a better description of the analyte(s) of interest.

In this chapter, you will learn about the advantages of multivariate calibration (1st and 2nd order) over univariate models (0th order) and several examples guided by algorithms in the R language on real or simulated data sets. Furthermore, some multivariate calibration algorithms coupled to variable selection algorithms will be presented.

Upon completing the chapter, you should be able to:

- a) Build and evaluate figures of merit for univariate calibration models (0th order) using R scripts;
- b) Build and evaluate figures of merit for higher order calibration models (1st and 2nd order) using R scripts;
- c) Couple variable selection algorithms with multivariate calibration and evaluate their performances;
- d) Understand the sample and variable selection algorithms used in higher order multivariate calibration models;
- e) Investigate the stages for building multivariate calibration models when applied to variable selection algorithms and evaluate their performances (figures of merit);
- f) Build new scripts in the R language for decision making using 1st and 2nd order multivariate calibration;
- g) Propose new applications in chemistry or related areas of multivariate calibration techniques.

5.1 Univariate Calibration

Basically, calibration consists of a mathematical operation that determines the functional relationship between values measured by some instrument and a property of interest, such as concentration. The calibration process also includes the selection of the adjustment method between the instrumental signal and the property of interest (linear, quadratic, cubic), the estimation of model parameters and the errors associated with these estimates, as well as the validation of the model.

In most cases, calibration for quantitative analysis is determined by a linear mathematical relationship between y (instrumental response) and x (analyte concentration) as follows:

$$y = F(x) + e_y \quad \text{Eq. 1}$$

Where F is the linear calibration function.

In 0th order, or univariate calibration, the mathematical model is obtained using only a single value of the experimental measurement per sample, without the need to consider other components and interfering factors. y is a scalar quantity. However, its application has a prerequisite: the absence of interferences to the analyte which could cause deviations between its relationship with the property of interest.

5.2 Calibration by Least Squares – univariate model

The Method of Least Squares (MLS), or Ordinary Least Squares (OLS), is a mathematical optimization technique that seeks to find the best fit for a set of data by trying to minimize the sum of squares of the differences between the estimated value and the observed data. The pioneering work of the least squares method is attributed to the mathematician Carl Friedrich Gauss in 1795, but the first clear and concise explanation was published in 1805 by Adrien-Marie Legendre [1].

Minimizing the error of experimental measurements related to unknown true values is known in the literature as "the least squares problem". Here, we present a solution through a theorem to minimize this error: Let $Y \in M_{n \times 1}(\mathfrak{R})$ and $X \in M_{n \times m}(\mathfrak{R})$ be matrices whose columns form a linearly independent set, with $m \leq n$. Then, there is a singular matrix \hat{A} , such that:

$$\|Y - X\hat{A}\| \leq \|Y - XA\|, \forall A \in M_{m \times 1}(\mathfrak{R}) \quad \text{Eq. 2}$$

$$\hat{A} = (X^T X)^{-1} X^T Y \quad \text{Eq. 3}$$

Another way to understand the least squares method is to write a linear empirical model or calibration function:

$$\hat{y}_i = b_0 + b_1x_1 + \varepsilon \quad \text{Eq. 4}$$

where \hat{y} is the estimated response, b_0 and b_1 are the regression coefficients and ε is the random error of the system.

In practice, equation 4 is obtained using calibration samples or certified reference materials, standards containing one or several components, or synthetic standard materials, that is, samples with known concentrations and high precision and accuracy.

Thus, the sum of the total residuals of this model is given by:

$$\sum e_i^2 = 0 \quad \text{Eq. 5}$$

$$\text{where: } \sum e_i^2 = \sum (y_i - \hat{y}_i)^2 = 0 \quad \text{Eq. 6}$$

$$\text{so: } \sum (y_i - b_0 - b_1T_i)^2 = 0 \quad \text{Eq. 7}$$

If we calculate the partial derivative of the sum of residues as a function of b_0 and b_1 , we obtain the values of the regression coefficients, as shown in the table below:

$\frac{d \sum e_i^2}{db_0} = 0$	$\frac{d \sum e_i^2}{db_1} = 0$
$\frac{d \sum (y_i - \hat{y}_i)^2}{db_0} = 0$	$\frac{d \sum (y_i - \hat{y}_i)^2}{db_1} = 0$
$\frac{d \sum (y_i - b_0 - b_1T_i)^2}{db_0} = 0$	$\frac{d \sum (y_i - b_0 - b_1T_i)^2}{db_1} = 0$
$-2 \sum (y_i - b_0 - b_1T_i) = 0$	$-2 \sum T_i (y_i - b_0 - b_1T_i) = 0$
$\sum y_i - \sum b_0 - \sum b_1T_i = 0$	$\sum T_i y_i - \sum b_0T_i - \sum b_1T_i^2 = 0$
$\sum y_i = \sum b_0 + \sum b_1T_i$	$\sum T_i y_i = b_0 \sum T_i + \sum b_1T_i^2$
$\sum y_i = nb_0 + b_1 \sum T_i$	$\sum T_i y_i = (\bar{y} - b_1\bar{T}) \sum T_i + \sum b_1T_i^2$

$$nb_0 = \sum y_i - b_1 \sum T_i$$

$$b_0 = \frac{\sum y_i}{n} - \frac{b_1 \sum T_i}{n}$$

$$b_0 = \bar{y} - b_1 \bar{T}$$

$$\begin{aligned} \sum T_i y_i &= \bar{y} \sum T_i - b_1 \bar{T} \sum T_i \\ &\quad + b_1 \sum T_i^2 \end{aligned}$$

$$\begin{aligned} \sum T_i y_i - \bar{y} \sum T_i &= -b_1 \bar{T} \sum T_i + b_1 \sum T_i^2 \\ &= b_1 \left(\sum T_i^2 - \bar{T} \sum T_i \right) \end{aligned}$$

$$\begin{aligned} \sum T_i y_i - \bar{y} \sum T_i &= b_1 \left(\sum T_i^2 - \bar{T} \sum T_i \right) \end{aligned}$$

$$b_1 = \frac{\sum T_i y_i - \bar{y} \sum T_i}{(\sum T_i^2 - \bar{T} \sum T_i)}$$

Replacing the values in equations above for b_0 and b_1 , we have:

$$b_1 = \frac{\sum T_i y_i - \bar{y} \sum T_i}{(\sum T_i^2 - \bar{T} \sum T_i)}$$

Eq. 8

Figure 5.1 presents different least squares models that can be calculated after minimizing the sum of residuals.

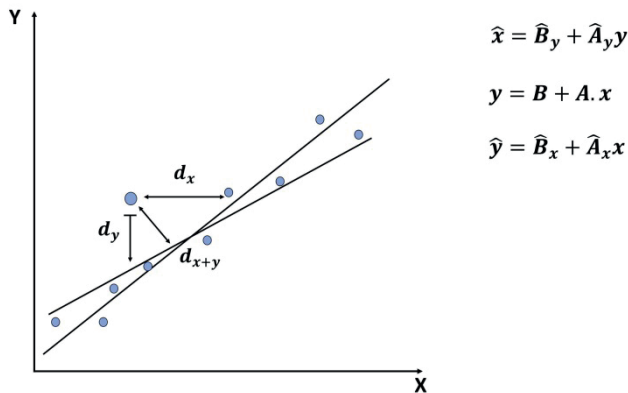


Figure 5.1: Different least squares adjustments. \hat{B}_y , \hat{B}_x , \hat{A}_y , \hat{A}_x are the estimates of B_y , B_x , A_y and A_x .

If there is homoscedasticity (constant error variance) and normal distribution, we can also estimate the parameters B_x and A_x using least squares (Gaussian algorithm):

$$\hat{A}_x = \frac{Q_{xy}}{Q_{xx}} \quad \text{Eq. 9}$$

$$\hat{B}_x = \frac{(\sum y - \hat{A}_x \sum x)}{m} \quad \text{Eq. 10}$$

Where m is the total number of calibration experiments used to construct the calibration function, using the following sums:

$$Q_{xx} = \sum (x_j - \bar{x})^2 = \frac{\sum x_j^2 - (\sum x_j)^2}{m} \quad \text{Eq. 11}$$

$$Q_{yy} = \sum (y_j - \bar{y})^2 = \frac{\sum y_j^2 - (\sum y_j)^2}{m} \quad \text{Eq. 12}$$

$$Q_{xy} = \sum (x_j - \bar{x})(y_j - \bar{y}) = \sum (x_j y_j) - (m \bar{x} \bar{y}) \quad \text{Eq. 13}$$

Another parameter in the least squares method consists of the Pearson's ρ or Pearson correlation coefficient, which measures the degree and direction (positive or negative) of the correlation between two variables, according to the equation:

$$\rho = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^n (x_i - \bar{x})^2} \cdot \sqrt{\sum_{i=1}^n (y_i - \bar{y})^2}} \quad \text{Eq. 14}$$

Typically, the value r only takes values between -1 and 1. If $\rho = 1$, we have a perfect correlation between the two variables. If $\rho = -1$, we have a perfect negative correction between the two variables. If $\rho = 0$ the two variables do not depend linearly on each other.

When properly applying calibration models, it is necessary to test whether the conditions of these models are adequate. Usually, some tests are carried out such as: i) linearity and, ii) homoscedasticity.

Linearity is obtained by observing the model residuals, which must be random, as shown in **Figure 5.2a**. If the errors present systematic deviations, **Figure 5.2b**, the indication of an inadequate linear model is evident. Linearity can also be assessed by observing the model parameters (regression coefficient, intercept and correlation coefficient). As we know

in analysis of variance (ANOVA), there is an assumption in which errors must have a constant variance, that is, they must be homoscedastic. If the variances are not homogeneous, we assume the case of heteroscedasticity. With a visual inspection of the residuals graph, as can be seen in **Figure 5.2c**, we can see that the variance is not constant.

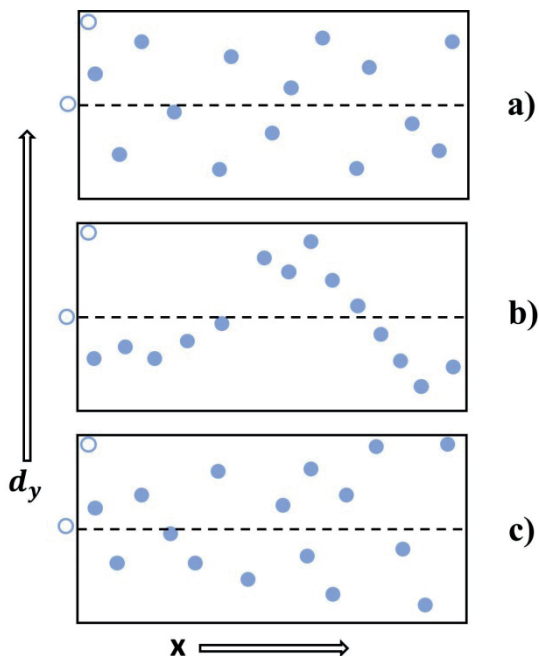


Figure 5.2: Common plots for residual deviations of calibration models.

After constructing an analytical method, the method must be validated to ensure the results obtained through it have the required efficiency under the conditions in which it will be applied. Normally, this efficiency of the model (analytical validation) is carried out through the determination of several parameters that characterize the efficiency of the method, which are called figures of merit. In addition to the parameters discussed in univariate calibration models (linearity and homoscedasticity), we can find other figures of merit in higher order calibration, such as:

i) Analytical sensitivity	$\xi_{i,a} = \frac{S_a}{S_i}$	Eq. 15
---------------------------	-------------------------------	---------------

ii) Limit of detection (LD)	$LD = 3.3 \frac{S}{S}$	Eq. 16
-----------------------------	------------------------	---------------

iii) Limit of quantification (LQ)	$LQ = 10 \frac{S}{S}$	Eq. 17
-----------------------------------	-----------------------	---------------

Where S_a is the sensitivity of the property of interest; S_i it is the sensitivity of an interferer; s is the standard deviation; and, S is the slope or angular coefficient of the analytical curve.

iv) Accuracy – basically, it consists on the difference between the value estimated by the model (here, multivariate) and the reference value. In multivariate calibration, we can write it as the square root of the mean squared error of prediction (RMSEP)

$$RMSEP = \sqrt{\frac{\sum_{i=1}^{l_v} (y_i - \hat{y}_i)^2}{l_v}} \quad \text{Eq. 18}$$

where l_v represents the number of samples in the validation set, y_i and \hat{y}_i correspond to the reference values and those predicted by the model, respectively.

v) Precision – consists of the degree of agreement between the results of a series of measurements for the same sample. In multivariate calibration, we can calculate precision from the equation below:

$$\text{precisão} = \sqrt{\frac{\sum_{i=1}^n \sum_{j=1}^m (\hat{y}_{ij} - \hat{y}_i)^2}{n(m-1)}} \quad \text{Eq. 19}$$

where, n represents the number of samples and m the number of replicates.

vi) Sensitivity – is the fraction of the signal responsible for adding a concentration unit to the property of interest. In multivariate calibration models, we can write the sensitivity as:

$$S\hat{E}N = \frac{1}{\|b_k\|} \quad \text{Eq. 20}$$

Where b_k corresponds to the vector of regression coefficients.

vii) Analytical sensitivity – it is the ratio between the sensitivity and the standard deviation of the reference signal (δx):

$$\gamma = \frac{S\hat{E}N}{\|\delta x\|} \quad \text{Eq. 21}$$

viii) Limit of detection (LD) and quantification (LQ) – these parameters express the smallest quantities of the species of interest that can be detected and determined quantitatively, respectively. In multivariate calibration, these parameters are calculated according to the following equations:

$$LD = 3\delta x \|b_k\| = 3\delta x \frac{1}{S\hat{E}N} \quad \text{Eq. 22}$$

$$LQ = 10\delta x \|b_k\| = 10\delta x \frac{1}{S\hat{E}N} \quad \text{Eq. 23}$$

where δx is the standard deviation of the reference method, \mathbf{b}_k is the vector of regression coefficients of the PLS model for species k , SEN corresponds to the analytical sensitivity.

Below, two examples of univariate calibration models will be presented, as well as some figures of merit for evaluating the constructed models.

Example 1: In the example we will describe the construction and validation of a univariate regression model using the *AER*, *ggplot2*, *caret*, *lmtest* and *olsrr* packages.

R Script

```
install.packages("AER")
library(AER)

# Data
# ensure reproducibility after training and val partition
set.seed(9)
data("USConsump1993", package = "AER")
plot(USConsump1993, plot.type = "single", col = 1:2)

# Transforming data into data.frame = spreadsheet to facilitate formatting
consumption = data.frame(USConsump1993)

#####
# Univariate Linear Regression #####

#### Training

# Number of training data (70%)
tr = round(0.70* nrow (consumption))

# defining training dataset lines
training = sample(nrow (consumption), tr , replace = F)

# separating training dataset
consumption.tr = consumption[training,]

# simple linear regression model
lm1 = lm (formula = expenditure ~ income, data = consumption.tr)
summary (lm1)
confint (lm1)

# plotting
plot (expenditure ~ income, data = consumption.tr, col = "indianred", pch = 20, cex
= 1.5)
abline (lm1, col = "palegreen", lwd = 2)
grid (lwd = 2)
```

```

library(ggplot2)
ggplot (data = consumption.tr, aes (x = income, y = expenditure)) +
  geom_point (color = 'indianred', lwd = 2)+
  geom_smooth (method = "lm", formula = y ~ x, col = "palegreen")+
  ggtitle ("Consumption vs income (training data)") +
  xlab ("income") +
  ylab ("consumption") + theme_bw ()

##### Test
# Test data
consumption.te = consumption[-training,]

# test data prediction
res.test = predict(lm1,newdata = data.frame (income = consumption.te$income))

# Chart
plot(expenditure ~ income, data = consumption.te , col = "slateblue", pch = 20, cex
= 1.5)
abline(lm1, col = "khaki4", lwd = 2)
grid(lwd = 2)

ggplot() +
  geom_point(aes(x = consumption.te$income, y = consumption.te$expenditure),
  color = 'slateblue', lwd = 2) +
  geom_smooth(method = "lm", formula = y ~ x,
  aes(x = consumption.tr$income, y = consumption.tr$expenditure), col
= "khaki4") +
  ggtitle ("consumption vs income (test data)") +
  xlab("income") +
  ylab("consumption") + theme_bw()

# Model 1 performance with test data

test1 = data.frame( obs = consumption.te [,2], pred = res.test )
library(caret)
defaultSummary(test1)

R2(consumption.te [,2], res.test )
RMSE(consumption.te [,2], res.test )
MAE(consumption.te [,2], res.test ) # average of absolute deviations

### Model suitability
# residuals graph
par(mfrow = c(2,2))
plot(lm1)

# Normality test
shapiro.test(residuals(lm1))

```

```

# Test for autocorrelation of residuals (they must be uncorrelated)
library(lmtest )
dwtest(lm1)

# Homoscedasticity test
install.packages("olsrr")
library(olsrr)
ols_test_breusch_pagan(lm1)

```

Example 2: The example we will describe here consists of the construction, validation and comparison of univariate regression models using different degrees of the polynomial through the *MASS*, *ggplot2*, *caret*, *lmtest* and *olsrr* packages.

R Script

```

library(MASS)

set.seed (9)
rehab = wtloss
? wtloss

##### 0th order model (simple) ##### ##
### training

# Number of training data (70%)
tr = round(0.70* nrow ( rehab ))

# defining training dataset lines
training = sample(nrow (rehab), tr, replace = F)

# separating training dataset
rehab.training = rehab[training,]

# simple linear regression model
lm1 = lm( Weight ~ Days , rehab.training )
summary (lm1)
confint (lm1)

# plotting the training
plot(Weight ~ Days, rehab.training, col = "navy", pch = 20, cex = 1.5)
abline(lm1, col = "seagreen", lwd = 2)
grid(lwd = 2)

library(ggplot2)
ggplot(rehab.training, aes( x = Days, y = Weight)) +
  geom_point(color = 'orange', lwd = 2)+
  geom_smooth(method = "lm", formula = y ~ x, col = "seagreen")+
  ggtitle("Weight vs Days (training data)") +
  xlab("Days") +
  ylab("Weight") + theme_bw()

```

```

##### Test
# Test data
rehab.test = rehab[-training,]

# test data prediction
res.test = predict(lm1,newdata = data.frame(Days = rehab.test[,1]))
rehab.test[,2]
res.test

# Plot

plot(Weight ~ Days, rehab.test, col = "orangered", pch = 20, cex = 1.5)
abline(lm1, col = "gold3", lwd = 2)
grid(lwd = 2)

ggplot() +
  geom_point(aes(x = rehab.test$Days, y = rehab.test$Weight),
             color = 'orangered', lwd = 2) +
  geom_smooth(method = "lm", formula = y ~ x,
             aes(x = rehab.training$Days, y = rehab.training$Weight)) +
  ggtitle ("Weight vs days (test data)") +
  xlab("days") +
  ylab("weight") + theme_bw()

# Model 1 performance with test data

test1 = data.frame(obs = rehab.test[,2], pred = res.test)
library(caret)
defaultSummary(test1)

### Model adequacy
# residuals graph
par(mfrow = c(2,2))
plot(lm1)

# Normality test
shapiro.test(residuals(lm1))

# Test for autocorrelation of residuals (they must be uncorrelated)
library(lmtest)
dwtest(lm1)

# Homoscedasticity test
install.packages("olsrr")
library(olsrr)
ols_test_breusch_pagan(lm1)

```

```

##### 0th order model (polynomial) #####
# polynomial regression model (2nd degree)
lm2 = lm(Weight ~ poly (Days, 2, raw = T), rehab.training)
summary (lm2)
confint (lm2) # confidence interval for coefficients

# plotting the training data

par(mfrow = c(1,1))
plot(Weight ~ Days, rehab.training, col = "deepskyblue3", pch = 20, cex = 1.5,
     main = "Weight vs Days (training data)")
x1 = seq(0,250, by = 0.1)
lines(x1,predict(lm2, newdata = data.frame (Days = x1)), col = "mediumvioletred",
      lwd = 2)
grid(lwd = 2)

library(ggplot2)
ggplot(rehab.training, aes( x = Days, y = Weight)) +
  geom_point(color = 'deepskyblue3', lwd = 2)+
  geom_smooth(method = "lm", formula = y ~ x + I(x^2), col = "mediumvioletred")+
  ggtitle("Weight vs Days (training data)") +
  xlab("Days") +
  ylab("Weight") + theme_bw()

#### Test

# prediction with test data

res.test2 = predict(lm2, newdata = data.frame(Days = rehab.test[,1]))
rehab.test[,2]
res.test2

# lm2 model performance with test data
test2 = data.frame(obs = rehab.test[,2], pred = res.test2)
defaultSummary(test2)

# Plot with test data

plot(Weight ~ Days, rehab.test, col = "red", pch = 20, cex = 1.5,
     main = "Weight vs Days (test data)")
x1 = seq(0,250, by = 0.1)
lines(x1,predict(lm2, newdata = data.frame (Days = x1)), col = "sandybrown", lwd
      = 2)
grid(lwd = 2)

```

```

ggplot() +
  geom_point(aes(x=rehab.test$Days, y = rehab.test$Weight),
            color = 'red', lwd = 2) +
  geom_smooth(method = "lm", formula = y ~ x + I(x^2),
            aes(x = rehab.training$Days, y = rehab.training$Weight), col =
'sandybrown') +
  ggtitle("Weight vs Days (test data)") +
  xlab("Days") +
  ylab("Weight") + theme_bw()

# Model suitability

# residuals graph
par(mfrow = c(2,2))
plot(lm2)

# Normality test
shapiro.test(residuals(lm2))

# Test for autocorrelation of residuals (they must be uncorrelated)
dwtest(lm2)

# Homoscedasticity test
ols_test_breusch_pagan(lm2)

##### 0th order model (polynomial) #####
# polynomial regression model (3rd degree)
# Training

lm3 = lm(Weight ~ poly(Days, 3, raw = T), rehab.training)
summary(lm3)
confint(lm3)

# Prediction for test data
res.test3 = predict(lm3, newdata = data.frame (Days = rehab.test[,1]))

# lm3 model performance with test data
test3 = data.frame(obs = rehab.test[,2], pred = res.test3)
defaultSummary(test3)

##### 0th order model (polynomial) #####
# polynomial regression model (5th degree)
# Training

lm5 = lm (Weight ~ poly(Days, 5, raw = T), rehab.training)
summary (lm5)
confint (lm5)

```



```

# Prediction for test data
res.test5 = predict(lm5, newdata = data.frame (Days = rehab.test[,1]))

# lm3 model performance with test data
test5 = data.frame(obs = rehab.test[,2], pred = res.test5)
defaultSummary(test5)

# Choosing the best model

# Plotting model training and testing errors

RMSE_training = c(RMSE(rehab.training [,2], fitted.values(lm1)),
                  RMSE(rehab.training [,2], fitted.values(lm2)),
                  RMSE(rehab.training [,2], fitted.values(lm3)),
                  RMSE(rehab.training [,2], fitted.values(lm5))

)

RMSE_test = c(RMSE(rehab.test[,2], res.test),
              RMSE(rehab.test[,2], res.test2),
              RMSE(rehab.test[,2], res.test3),
              RMSE(rehab.test[,2], res.test5)

)

order = rep(c(1:3,5),2)
set = c(rep("training", 4), rep("test",4))
rmse = c( RMSE_training,RMSE_test )

select = data.frame(order,set,rmse)

ggplot (select,aes(x = order , y = rmse, group = set)) +
  geom_point(aes(col = set, shape = set), size = 3) +
  geom_line(aes(col=set, linetype =set)) + theme_bw ()

# Cross-validation

vc_lm1 = train(Weight ~ Days, rehab, method = "lm",
              trControl = trainControl(method = "CV", number = 10))

vc_lm2 = train(Weight ~ Days + I(Days^2), rehab, method = "lm",
              trControl = trainControl (method = "CV", number = 10))

vc_lm3 = train(Weight ~ Days + I(Days^2) + I(Days^3), rehab, method = "lm",
              trControl = trainControl (method = "CV", number = 10))

vc_lm5 = train(Weight ~ Days + I(Days^2) + I(Days^3) + I(Days^4) + I(Days^5), rehab,
              method = "lm",
              trControl = trainControl (method = "CV", number = 10))

```

```
summary(resamples(list(
  model1 = vc_lm1,
  model2 = vc_lm2,
  model3 = vc_lm3,
  model5 = vc_lm5
)))
```

5.3 Multiple Linear Regression (MLR)

Multiple linear regression (MLR), an extension of simple linear regression, is a simpler multivariate calibration technique that aims to establish a mathematical relationship between x (instrumental response) and y (parameter of interest) through the matrix equation, for situations of $n > m$:

$$\mathbf{Y} = \mathbf{X}\mathbf{b} + \mathbf{e} \quad \text{Eq. 24}$$

where \mathbf{y} corresponds to the vector of the parameter to be determined, \mathbf{X} is the matrix of instrumental variables, \mathbf{b} is the vector of regression coefficients, and \mathbf{e} is the vector of residuals. The MLR technique assumes that concentration is a function of the instrumental responses, and we call it indirect or inverse calibration.

This equation can be represented graphically through Figure 5.3 below:

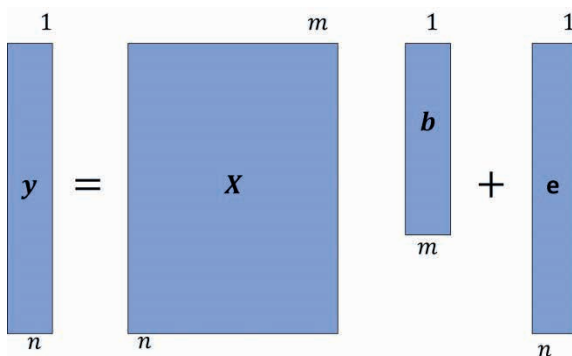


Figure 5.3: Graphical representation of the MLR technique.

In MLR, the regression coefficients \mathbf{b} can be estimated by several methods, one of the most used of which is the least squares method, as shown in the equation below:

$$\mathbf{b} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y} \quad \text{Eq. 25}$$

The analysis of equation (25) above points to the main limitations of the MLR technique: i) the inverse of $\mathbf{X}^T \mathbf{X}$ may not exist; ii) all information (significant variance)

contained in matrix X is used in building the model, regardless of whether it is relevant or not; and, iii) the number of independent variables cannot be greater than the number of calibration samples used in the analysis. When the problem of the inverse of $X^T X$ is due to matrix rank deficiency, the solution will not exist and this problem is known as collinearity.

However, the MLR technique, when $X^T X$ has a possible solution, is interesting for well-established systems. This happens when there are no collinearities in the X matrix, no interferences or interaction between the analytes, when the response is linear and with low instrumental noise. A possible solution for the MLR in systems with a great number of instrumental responses for a smaller quantity of samples is the use of variable selection algorithms, which will be discussed throughout this chapter.

Example 3: The example we will describe here consists of the construction, validation and comparison of MLR multivariate regression models using the *Stat2Data*, *ggplot2*, *caret*, *rsm*, *lmtree* and *olsrr* packages.

R Script

```
install.packages("Stat2Data")
library(Stat2Data)

data("ThreeCars2017")

pairs(ThreeCars2017) #scatter diagram

pairs(ThreeCars2017,
      col = viridis :: viridis(3)[ThreeCars2017$CarType],
      pch = c(15:17)[ThreeCars2017$CarType])

pairs(ThreeCars2017[,1:4],
      col = viridis :: viridis (3)[ThreeCars2017$CarType],
      pch = c(15:17)[ThreeCars2017$CarType])

##### Model 1 #####
#### Training
set.seed (11)

# Number of training samples (80%)
tr = round(0.8* nrow (ThreeCars2017))

# defining training dataset rows
training = sample( nrow (ThreeCars2017), tr, replace = F)

# separating the training dataset
cars.training = ThreeCars2017[training,]
```

```

# Model 1
model1 = lm (Price ~ Mileage, cars.training )
summary(model1)

# Preview
plot (Price ~ Mileage , cars.training , col = "palegreen2", pch = 20, cex = 1.5,
      main = "Price vs Mileage (training data)")
abline (model1, col = "steelblue", lwd = 2)
grid (lwd = 2)

library(ggplot2)
ggplot ( cars.training , aes (x = Mileage, y = Price)) +
  geom_point ( color = "palegreen", lwd = 2)+
  geom_smooth (method = "lm", formula = y ~ x)+
  ggtitle ("Price vs Mileage (training data)") +
  xlab ("Mileage") +
  ylab ("Price") + theme_bw ()

### Test
# Test data
cars.test = ThreeCars2017[-training,]

# prediction with test data
res.test1 = predict(model1, newdata = data.frame (Mileage = cars.test$Mileage))

# Model 1 performance with test data
test1 = data.frame (obs = cars.test$Price , pred = res.test1)
library(caret)
defaultSummary (test1)

# Preview
plot(Price ~ Mileage, cars.test, col = "mediumorchid", pch = 20, cex = 1.5,
     main = "Price vs Mileage (test data)")
abline (model1, col = "olivedrab3", lwd = 2)
grid( lwd = 2)

library(ggplot2)
ggplot () +
  geom_point ( aes(x = cars.test$Mileage , y = cars.test$Price ),
             color = "mediumorchid", lwd = 2) +
  geom_smooth (method = "lm", formula = y ~ x,
             aes(x = cars.training$Mileage , y = cars.training$Price ), col =
"olivedrab3") +
  ggtitle ("Price vs Mileage (test data)") +
  xlab ("Mileage") +
  ylab ("Price") + theme_bw ()

```

```

##### Model 2 (considering 2 regressor variables )
model2 = lm (Price ~ Mileage + Age, cars.training )
summary(model2)

## Preview
install.packages("rsm")
library(rsm)
persp(model2, Age ~ Mileage, col = rainbow(50), contours = "colors", theta = 50,
phi = 30)

contour(model2, Age ~ Mileage, image = TRUE)

#####
xs = seq (min( cars.training$Mileage ), max( cars.training$Mileage ), length = 20)
ys = seq (min( cars.training$Age ), max( cars.training$Age ), length = 20)
xys = expand.grid ( xs,ys )
colnames ( xys ) = c("Mileage", "Age")
zs = matrix(predict(model2, xys ), nrow = length( xs ))

n.cols = 100
palette = colorRampPalette (c("lightseagreen", "mediumvioletred"))( n.cols )
zfacet = zs[-1,-1] + zs[-1, -20] + zs[-20, -1] + zs[-20, -20]
facetcol = cut( zfacet , n.cols )

pl = persp (x = xs , y = ys , z = zs, theta = 50, phi = 30, ticktype = 'detailed',
          xlab = "Mileage", ylab = "Age", zlab = "Price", col = palette [ facetcol ])

with ( cars.training , points(trans3d(Mileage, Age, Price, pl), pch = 20, col =
"orangered"))

### Test
# Prediction with test data
res.test2 = predict(model2, newdata = data.frame ( cars.test [c(2,4)]))

## Model 2 performance with test data
test2 = data.frame ( obs = cars.test$Price , pred = res.test2)
defaultSummary (test2)

#### Model 3
model3 = lm (Price ~ Mileage*Age, cars.training )
summary(model3)

# Preview
persp (model3, Age ~ Mileage, col = rainbow(50), contours = "colors")
contour(model3, Age ~ Mileage, image = TRUE)

```

```

# Test
# Prediction with test data

res.test3 = predict(model3, newdata = data.frame(cars.test [c(2,4)]))

## Model 3 performance with test data
test3 = data.frame ( obs = cars.test$Price , pred = res.test3)
defaultSummary (test3)

##### Model 4 #####
# Model 4
model4 = lm (Price ~ Mileage*Age + I(Mileage^2) + I(Age^2), cars.training )
summary(model4)

# Preview
persp(model4, Age ~ Mileage, col = rainbow(50), contours = "colors", theta = 60,
phi = 30,)
contour(model4, Age ~ Mileage, image = TRUE)

# Prediction with test data

res.test4 = predict (model4, newdata = data.frame ( cars.test [c(2,4)]))

## Model 3 performance with test data
test4 = data.frame ( obs = cars.test$Price , pred = res.test4)
defaultSummary (test4)

##### Model 5 #####
# Model 5
model5 = lm (Price ~ Mileage*Age + I(Mileage^2) + I(Age^2) + CarType , cars.training
)
summary(model5)

# Preview
persp(model5, Age ~ Mileage, col = rainbow(50), contours = "colors", theta = 60,
phi = 30)
contour(model5, Age ~ Mileage, image = TRUE)

# Prediction with test data
res.test5 = predict (model5, newdata = data.frame ( cars.test [c(1,2,4)]))

## Model 3 performance with test data
test5 = data.frame ( obs = cars.test$Price , pred = res.test5)
defaultSummary (test5)

##### Model 6 #####
# Model 6
model6 = step(model5, trace = 1, direction = "backward")
# tests whether it is worth removing a coefficient or not to improve the model
summary (model6)

```

```

# Prediction with test data
res.test6 = predict(model6, newdata = data.frame ( cars.test [c(1,2,4)]))

## Model 3 performance with test data
test6 = data.frame (obs = cars.test$Price , pred = res.test6)
defaultSummary (test6)

##### Choosing the best model #####
#### Plotting training and testing error of the best model

rmse_training = c(RMSE( cars.training$Price , fitted.values (model1)),
                  RMSE( cars.training$Price , fitted.values (model2)),
                  RMSE( cars.training$Price , fitted.values (model3)),
                  RMSE( cars.training$Price , fitted.values (model4)),
                  RMSE( cars.training$Price , fitted.values (model5)),
                  RMSE( cars.training$Price , fitted.values (model6))
)

rmse_test = c(RMSE( cars.test$Price , res.test1),
              RMSE( cars.test$Price , res.test2),
              RMSE( cars.test$Price , res.test3),
              RMSE( cars.test$Price , res.test4),
              RMSE( cars.test$Price , res.test5),
              RMSE( cars.test$Price , res.test6)
)

model = rep(1:6,2)
set = c(rep("training",6), rep("test",6))
rmse = c( rmse_training,rmse_test )

select = data.frame ( model, set, rmse )

ggplot (select, aes (x = model , y = rmse , group = set)) +
  geom_point ( aes (col = set, shape = set), size = 3) +
  geom_line ( aes (col=set, linetype =set)) + theme_bw ()

#### Cross-validation

vc_lm6 = train(Price ~ Mileage*Age + I(Mileage^2) + I(Age^2) + CarType , ThreeCars2017,
method = "lm",
          trControl = trainControl (method = "cv", number = 10))
vc_lm6

# Suitability model 6
par(mfrow = c(2,2))
plot(model6)

```

```
shapiro.test(residuals(model6))
install.packages ("lmtest")
library(lmtest)
dwtest(model6)
install.packages("olsrr")
library(olsrr)
ols_test_breusch_pagan(model6)
```

5.4 MLR – SPA

As discussed previously, the MLR algorithm is one of the simplest multivariate calibration techniques. In MLR, the analysis can be described by a linear relationship between the independent variables \mathbf{X} and a dependent variable \mathbf{Y} , as described in Eq. 25. It was also mentioned that this algorithm does not need to know the concentration for all species spectroscopy active in the samples belonging to the calibration set, i.e., unknown chemical species, interferences and baseline effects; since these when present, can be modeled. However, an important problem in MLR calibration is that the matrix $(\mathbf{X}^T\mathbf{X})$ may not be invertible or promote the propagation of errors when there is strong correlation or multicollinearity between the variables. This happens when the number of variables is greater than the number of samples.

As we normally obtain a large number of instrumental responses and a smaller number of samples, this problem of inverting the $\mathbf{X}^T\mathbf{X}$ matrix tends to continue, except when we use variable selection algorithms to get around these restrictions. The SPA variable selection algorithm selects subsets of variables with minimally redundant content, from a succession of projections comprising the instrumental variables column, to correct collinearity problems, which is interesting for the MLR algorithm.

Example 4 : The following example consists of an application of the MLR-SPA multivariate regression algorithm to mid-infrared spectra in plasma samples with different synthetic (spiked) concentrations of dengue virus. The packages used in the script are *R.matlab*, *prospectr*, *MASS*, *lintools*, *ggplot2*, *Stat2Data* and *Metrics*.

R Script

```
## Loading packages

install.packages ("R.matlab")
library(R.matlab)

install.packages("prospectr")
install.packages("MASS")
install.packages("lintools")
```



```

library(prospectr)
library(MASS)
library(ggplot2)
library(lintools)

install.packages("Stat2Data")
library(Stat2Data)

install.packages("Metrics")
library(Metrics)

## Loading data

# Navigate in RStudio to the directory with the dataset to work with
# Session > Set Working Directory > Choose Directory
data <- readMat("data_reg.mat")

x = data$data # spectra
y = data$concentration # concentration
cm = data$cm # wave number
cmt = t(cm)

## data plot

dev.new()
matplot(t(cm),t(x), type = 'l', xlab = "Wavenumber (cm-1)", ylab = "Absorbance")

## SPA for variable selection

# PCA Model

x_scal = scale(x) # mean centering
dim_x = dim(x) # dimension of the data array

x.svd = svd ( x_scal ) # SVD
x.scores = x.svd$u %*% diag ( x.svd$d ) # scores
x.loadings = x.svd$v # loadings

# SPA model

nvar = 22 # number of variables to select

xm = data.matrix (x, rownames.force =NA) # converting data to matrix

m = colMeans (x) # average of spectra

model_spa = project (x= x.loadings [,1], A= xm , b=y, neq =0) # spa model

```

```

xabs = abs ( model_spa$x ) # leaving positive values for the SPA response vector

temp = sort.int(xabs , decreasing=TRUE, index.return =TRUE)
variables = temp$ix [1:nvar] # identifying selected variables

dev.new () # plot of selected variables
matplot (cmt,m,xlab ="Wave number (cm-1)", type ="l", ylab ="Absorbance", main
="Average spectrum with selected variables")
points(cm[ variables ],m[ variables ], pch =19)

xm_spa = xm [, variables ] # absorbances for the selected variables
xm_spa_df = data.frame ( xm_spa )

## Division into Calibration and Prediction

perc = 0.7 # 70% for calibration and 30% for testing

size = 1:dim_x[1]

ntrain = ceiling (perc * dim_x [1]) # number of calibration samples

sel = sample(size, ntrain ) # sample selection

xcal = xm_spa [ sel , ] # calibration
ycal = matrix (y[ sel ]) # concentration calibration

xpred = xm_spa [- sel , ] # prediction
ypred = matrix (y[- sel ]) # concentration prediction

## MLR Model

xcal_df = data.frame ( xcal )
xpred_df = data.frame ( xpred )

model_mlr = lm ( ycal ~ xcal , xcal_df ) # MLR model

coef = model_mlr$coefficients # regression coefficients

ycal_calc = cbind ( rowMeans ( xcal ), xcal )%% matrix ( coef ) # predicted
concentration calibration
ypred_calc = cbind ( rowMeans ( xpred ), xpred )%% matrix ( coef ) # predicted
concentration prediction

## Plot measured vs.predicted concentration

dev.new ()
plot (ycal,ycal_calc,pch ='o', col ="blue", xlab ="Measured Concentration (mg/L)",
ylab ="Predicted Concentration (mg/L)", main ="Calibration")
lines(ycal,ycal,col ='red')

```

```

dev.new ( )
plot (ypred,ypred_calc,pch =15,col="red", xlab = "Measured Concentration (mg/L)",
ylab = "Predicted Concentration (mg/L)", main = "Predicted")
lines(ypred,ypred,col ='red')

dev.new ( )
plot (ycal,ycal_calc,pch ='o', col ="blue", xlab ="Measured Concentration (mg/L)",
ylab ="Predicted Concentration (mg/L)", main ="Calibration (blue) and Prediction
(red)")
points(ypred,ypred_calc,pch =15,col="red")
lines(y,y,col ="red")

## Figures of merit

# Calibration

MAPEC = mean(abs (( ycal-ycal_calc )/ ycal ))*100
R2cal = cor(ycal,ycal_calc)^2
RMSEC = rmse(ycal,ycal_calc )

# Prediction

MAPEP = mean(abs(( ypred-ypred_calc )/ ypred ))*100
R2pred = cor(ypred,ypred_calc )^2
RMSEP = rmse(ypred,ypred_calc )

print("==== Calibration =====")
print("Mean absolute error percentage (MAPE) (%):")
MAPEC
print("R2cal:")
R2cal
print("RMSEC (mg/L):")
RMSEC

print("==== Test =====")
print("Mean absolute error percentage (MAPE) (%):")
MAPEP
print("R2pred:")
R2pred
print("RMSEP (mg/L):")
RMSEP

```

5.5 Principal Component Regression (PCR)

Principal component regression (PCR) is a multivariate regression technique used when there are many independent variables or multicollinearities (when two or more independent variables in a regression model are highly correlated), experimental noise or lack of linearity.

We can divide the PCR technique into two phases: description of the block of independent variables (matrix X), being orthogonal to each other, therefore there is no correlation between them.

This first step of the PCR technique can be represented graphically in **Figure 5.4** below:

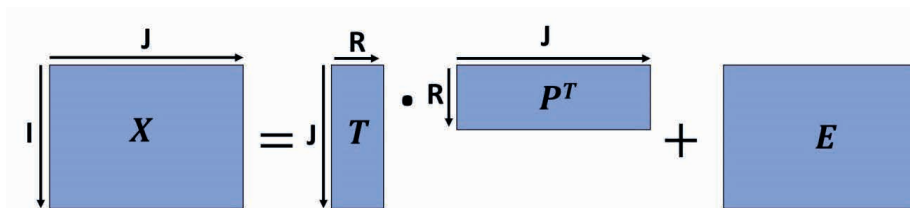


Figure 5.4: Graphical representation of the PCR technique

The second stage of the PCR technique consists of using the MLR technique to establish a mathematical relationship between the matrix of T scores (the new block of independent variables) and the block of dependent variables (matrix Y). Thus the MLR equation can be written:

$$Y = TB + E \quad \text{Eq. 26}$$

and the solution for the regression coefficients is:

$$B = (T^T T)^{-1} T^T Y \quad \text{Eq. 27}$$

Note that inverting $T^T T$ will not cause problems due to the mutual orthogonality of the scores, correcting the collinearity problem. However, it is important to highlight that a crucial step in the PCR technique is choosing the number of principal components due to the risk of loss of information. Another issue is that PCR ignores all the information contained in the Y matrix in the initial step. The data from the Y matrix are only used in the second step, when the number of components has already been determined.

Example 5: The following example consists of a practical script for multivariate regression algorithms (MLR and PCR) using the *Ecdat*, *corrplot*, *car*, *caret* and *pls* packages.

R Script

```
install.packages("hdcrcde")
library(hdcrcde)

install.packages("DEoptimR")
library(DEoptimR)

install.packages("Ecfun")
library(Ecfun)

install.packages("Ecdat")
library(Ecdat)

#data
data = ManufCost

? ManufCost

data = data.frame ( ManufCost )

data = na.omit (data)

# Correlation
library ( corrplot )

r = cor(data)
round(r,2)

# preview
corrplot :: corrplot ( r,method = "color",
                      type = "upper",
                      order= "hclust",
                      addCoef.col = "black", tl.srt = 45,
                      diag = FALSE)

pairs (data, col = "mediumseagreen")

##### #####
#### Training and Testing Data

set.seed (33)

# Separating training and testing data
tr = round(0.8* nrow (data))
training = sample( nrow (data), tr , replace = F)

data.training = data[training,]
data.test = data[-training,]
```

```

#####

### Visualizing correlation and principal components between two variables

par(mfrow = c(1,1))
# plotting two correlated variables
plot(scale(s1)~scale(p1), asp = 1, data, pch = 20, cex = 1.5, col = "mediumseagreen")

# separating such variables
d = data[,c(3,7)]

# correlation matrix of such variables
cm = cor(d)
cm

#eigenvalues of cm
e = eigen(cm)

# Slopes of the principal components
s1 = e$vectors [1,1]/ e$vectors [2,1] # PC1
s2 = e$vectors [1,2]/ e$vectors [2,2] # PC2

# Principal axes
abline (a=0, b=s1, col = "blue", lwd = 2)
abline (a=0, b=s2, col = "lightblue", lwd = 2)

##### Multiple Linear Regression (MLR)
# Complete model - Training
lm1 = lm (cost ~., data.training )
library("car")
vif (lm1) # variance inflation factor due to multicollinearity
confint (lm1)

# Test
pred.lm1 = predict(lm1, newdata = data.test)

# metrics
library("caret")
test.lm = data.frame ( data.test$cost , pred.lm1)
colnames ( test.lm ) = c("obs", "pred")
defaultSummary ( test.lm )

#####

### PCR
install.packages("pls")
library(pls)

```

```

# PCR model training
pcr1 = pcr( cost ~., ncomp = 8, data = data.training , validation = "LOO", scale = T)
summary(pcr1)

#cross validation
ncomp.onesigma = selectNcomp (pcr1, method = "onesigma", plot = TRUE )
ncomp.permut = selectNcomp (pcr1, method = "randomization", plot = TRUE)

# preview
plot(RMSEP(pcr1), legendpos = "topright")

plot(pcr1,ncomp = 6, line = TRUE)

plot(pcr1,plottype = "scores", comps = 1:2)

plot(pcr1, "loadings", comps = 1:2, legendpos = "bottomright")
abline (h = 0, col = " lightblue ")

#model test
pred.pcr1 = predict(pcr1, ncomp = 2, newdata = data.test )

#metrics
test.pcr = data.frame ( data.test$cost , pred.pcr1)
colnames ( test.pcr ) = c("obs", "pred")
defaultSummary ( test.pcr )

```

5.6 Partial least squares (PLS) regression

The PLS technique was proposed by H. Wold in 1982 [2]. Unlike the PCR algorithm, PLS estimates scores both in the matrix of independent variables (matrix **X**) and in the dependent variables (matrix **Y**). However, the PLS algorithm has two main stages in its development. The first consists of simultaneously decomposing the matrices **X** and **Y** into a sum of "h" latent variables, as we can see in the following equations:

$$\mathbf{X} = \mathbf{TP}^T + \mathbf{E} = \sum \mathbf{t}_h \mathbf{p}_h^T + \mathbf{E} \quad \text{Eq. 28}$$

$$\mathbf{Y} = \mathbf{UQ}^T + \mathbf{F} = \sum \mathbf{u}_h \mathbf{q}_h^T + \mathbf{F} \quad \text{Eq. 29}$$

where **T** and **U** are the score matrices of matrices **X** and **Y**, respectively. **P** and **Q** are the loading matrices of matrices **X** and **Y**, respectively; and **E** and **F** are the residuals.

A representation of this first step of the PLS algorithm is shown in **Figure 5.5** below:

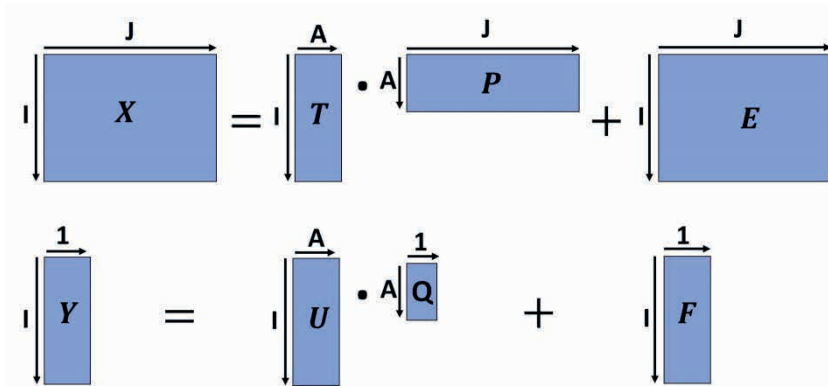


Figure 5.5: Graphical representation of the 1st stage of the PLS algorithm.

The second step of the PLS algorithm consists of calculating the linear correlation between the scores in matrix **Y** and the scores in matrix **X**, as described in the following equation:

$$\mathbf{u}_h = b_h \mathbf{t}_h \tag{Eq. 30}$$

for "h" latent variables, in which the values of b_h are grouped in the diagonal matrix **B** that contains the regression coefficients between the scores matrix **U** of **Y** and the scores matrix **T** of **X**.

A geometric illustration of the PLS algorithm in this second stage is shown in **Figure 5.6** below:

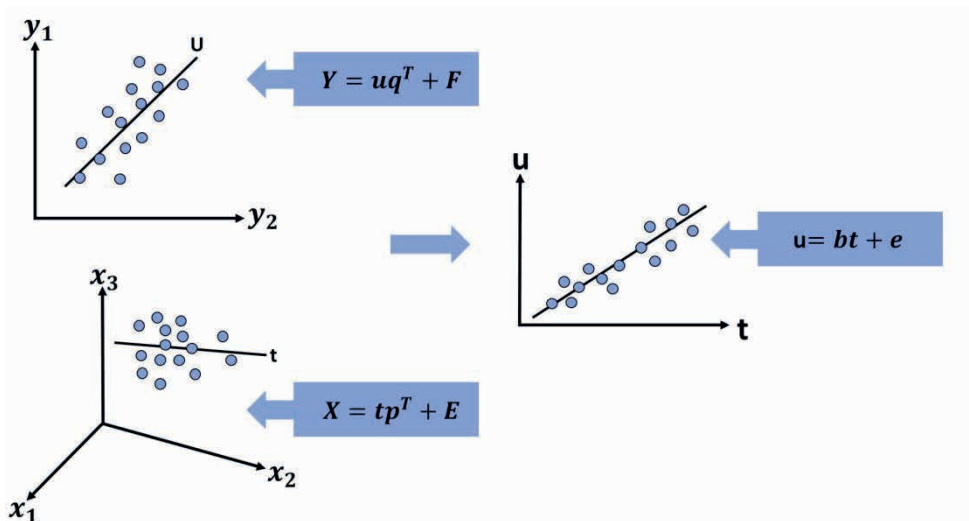


Figure 5.6 : Geometric representation of the PLS model using one latent variable modeling each block (**X** and **Y**).

An example of algorithm for calculating the PLS regression is the NIPALS algorithm [3], which presents two main steps: i) obtaining the orthogonal projector of the columns of matrix \mathbf{X} (instrumental variables), in the subspace generated by the columns of matrix \mathbf{Y} (parameter of interest), and find the inverse projector; ii) calculate the corresponding directions in the spaces generated by the \mathbf{Y} and \mathbf{X} columns to reduce the information in the reducer matrix.

We can see here that the PLS algorithm's main objective is to maintain a compromise between the ability of the principal components to describe the samples in individual spaces (obtaining the scores of the matrices \mathbf{X} and \mathbf{Y}) and maximizing the correlation between \mathbf{t} and \mathbf{u} .

It is also worth mentioning that in the process of searching for the best number of latent variables in the PLS algorithm, the cross-validation process is necessary. In cross-validation, the RMSEP of the prediction samples is calculated. Another important information is the differentiation between PLS1 and PLS2. In PLS1, the regression is performed for one dependent variable at a time (the \mathbf{Y} matrix is a column vector), while in PLS2 all dependent variables are calculated simultaneously.

Example 6: The following example consists of an application of the PLS multivariate regression algorithm through the *pls*, *Ecdat* and *caret* packages.

R Script

```
install.packages("Ecdat")
library(Ecdat)
library(caret)
install.packages("pls")
library(pls)

#data
data = ManufCost
? ManufCost
data = data.frame ( ManufCost )
data = na.omit (data)

#### Training and Testing Data
set.seed (33)
# Separating training and testing data
tr = round(0.8* nrow (data))
training = sample( nrow (data), tr , replace = F)
data.training = data[training,]
data.test = data[-training,]

##### #####
### Visualizing correlation and principal components between two variables
```

```

par(mfrow = c(1,1))
# plotting two correlated variables
plot(scale(s1)~scale(pl), asp = 1, data, pch = 20, cex = 1.5, col = "mediumseagreen")

# separating such variables
d = data[,c(3,7)]

# correlation matrix of such variables
cm = cor(d)
cm

#eigenvalues of cm
e = eigen (cm)

# Slopes of the principal components
s1 = e$vectors [1,1]/ e$vectors [2,1] # PC1
s2 = e$vectors [1,2]/ e$vectors [2,2] # PC2

# Principal axes
abline (a=0, b=s1, col = "blue", lwd = 2)
abline (a=0, b=s2, col = "lightblue", lwd = 2)

### PLS
# PLS training model
pls1 = plsr ( cost ~., ncomp = 8, data = data.training , validation = "LOO", scale
= T)
summary(pls1)

#cross validation
ncomp.onesigma = selectNcomp (pls1, method = "onesigma", plot = TRUE )
ncomp.permut = selectNcomp (pls1, method = "randomization", plot = TRUE)

# preview
plot(RMSEP(pls1), legendpos = "topright")

plot(pls1, ncomp = 6, line = TRUE)

plot(pls1,plottype = "scores", comps = 1:5)

plot(pls1, "loadings", comps = 1:2, legendpos = "bottomright")
abline (h = 0, col = "orange")

#model test
pred.pls1 = predict (pls1,ncomp = 2, newdata = data.test )

#metrics
test.pls = data.frame ( data.test$cost , pred.pls1)
colnames ( test.pls ) = c("obs", "pred")
defaultSummary ( test.pls )

```

5.6 PARAFAC

The PARAFAC algorithm, already described in the previous chapter, is also used as a 2nd order calibration algorithm and suitable for processing data with a trilinear structure, expressed through the following equation:

$$\mathbf{X} = \sum_{p=1}^F \mathbf{a} \otimes \mathbf{b} \otimes \mathbf{c} \quad \text{Eq. 31}$$

where F is the number of factors or components of the model; \mathbf{a} , \mathbf{b} and \mathbf{c} are vectors; and, \otimes is the external product.

For its validity, the equation 31 above assumes: i) the analytical signals of each source of variation contribute additively to the analytical signal of the sample; ii) the magnitudes of the signals are proportional to the concentration of the analyte to be calibrated; iii) the analytical signals of each analyte are common in all samples.

In addition to these assumptions, some aspects must be taken into account when applying the PARAFAC algorithm for multivariate calibration purposes: i) the agreement of the data structure; ii) the algorithm initialization method, the restrictions imposed on the model and the convergence criterion; iii) the number of factors or components; iv) identify the profiles estimated by the model in each factor with the species of interest and the interference present; v) construction and validation of the regression model to estimate the concentration of the species of interest.

Thus, the PARAFAC algorithm exemplifies a decomposition of a three-way tensor into three weight matrices \mathbf{A} , \mathbf{B} and \mathbf{C} , called modes A, B and C. Normally, equation 31 is solved by alternating least squares (ALS), in which the ALS algorithm iteratively estimates two modes to estimate the third until some convergence criterion is reached or the previously defined number of iterations is reached. However, several methods can be used to obtain these profiles, such as singular value decomposition (SVD) or direct trilinear decomposition (DTLD).

Once an initial estimate of, for example, \mathbf{B} and \mathbf{C} has been obtained, an initial estimate of \mathbf{A} can be obtained by least squares as follows:

$$\mathbf{A} = \mathbf{X}[(\mathbf{C} \mid \otimes \mathbf{B}^T)]^+ \quad \text{Eq. 32}$$

where "+" indicates the Moore-Penrose pseudo-inverse of the matrix \mathbf{X} .

For example, in excitation-emission matrices obtained by fluorescence spectroscopy, mode A, B and C would be, respectively, the analytical concentration of the pure components, excitation spectra of the pure components and emission spectra of the pure components. Thus, mode A, which corresponds to the concentrations of pure components, is used in the construction of calibration models by linear regression.

As we have discussed in the previous chapter, one of the challenges of the PARAFAC algorithm is determining the number of components. Depending on the complexity of the system, the choice of the number of components can be made based on prior knowledge of the number of species responsible

for the measured instrumental signal. Generally, core consistency testing (CORCONDIA) is typically employed when choosing the number of components. CORCONDIA values close to 100% indicate trilinear consistency and values below 100% indicate a deficiency in trilinearity or trilinear inconsistency (values close to 0). However, in systems with species in equilibrium, the use of core consistency does not lead to the correct number of components.

Finally, the regression model for PARAFAC, after the considerations previously described, can be calculated through a least squares regression between the columns of \mathbf{A} related to the concentration of the species of interest (\mathbf{a}) and the vector with the reference concentrations (\mathbf{y}) of the calibration samples, according to the equation below:

$$\omega = \mathbf{y}^+ \mathbf{a} \quad \text{Eq. 33}$$

Where, ω are the regression coefficients between the weights \mathbf{a} and the concentrations \mathbf{y} .

The interesting fact about the equation 33 above is that to determine a sample of unknown composition we have a data cube formed by I_c calibration samples (known concentration of the species of interest) and a sample of unknown composition that may contain interferences present or not in the calibration samples. This consequence of 2nd order calibration models is known as the second order advantage. Thus, the concentration of the composition of an unknown sample is obtained according to equation 34, if the signal of interest is only in one column of \mathbf{A} :

$$\hat{y}_{un} = \frac{a}{\omega} \quad \text{Eq. 34}$$

Example 7: The following example consists of an application of the PARAFAC multivariate regression algorithm in the analysis of fluorescence data (excitation-emission matrix) obtained in plasma samples with different synthetic concentrations (spiked) of a fluorescent standard. This script in R language uses the *multiway*, *threeway*, *R.matlab*, *plot3D*, *plotly*, *prospectr*, *MASS*, *Stat2Data*, *Metrics* and *lintools* packages.

R Script

```
## Loading Packages

install.packages ("multiway")
library(multiway)

install.packages ("ThreeWay")
library(ThreeWay)

install.packages ("R.matlab")
library(R.matlab)

install.packages ("plot3D")
library(plot3D)
```

```

install.packages ("plotly")
library(plotly)

install.packages ("prospectr")
install.packages ("MASS")
install.packages ("caret")

library(prospectr)
library(MASS)
library(caret)

install.packages ("Stat2Data")
library(Stat2Data)

install.packages ("Metrics")
library(Metrics)

install.packages ("lintools")
library(lintools)

## Loading Data

# Navigate in RStudio to the directory with the dataset to work with
# Session > Set Working Directory > Choose Directory

data <- readMat ("data_reg_parafac.mat")

x <- data$data
y <- data$concentration
nmEX <- data$nmEX
nmEM <- data$nmEM

## Visualizing the data

# matrix dimensions

mydim = dim (x) # samples x emission x excitation
mydim

# average matrix

xm = colMeans (x)

dev.new ()
filled.contour ( xm, color.palette = terrain.colors )

dev.new ()
matplot (t( nmEM ), xm,type ="l", xlab ="Emission Wavelength (nm)", ylab ="Intensity")

```

```

dev.new ()
matplot (t( nmEX ),t( xm ), type ="1", xlab ="Excitation Wavelength (nm)", ylab
="Intensity")

##### PARAFAC model #####

nf = 3 # define number of factors

model = parafac ( x,nfac = nf,nstart =1,maxit=500,ctol=10^-4,parallel=FALSE, cl=
NULL,output =c("best","all"))

# R2 Adjustment

model$Rsq # Increase the number of factors to better adjust R2

## plot Parafac scores 1 x 2

dev.new ()
matplot ( model$A [,1], model$A [,2], pch ="o", col ="blue", xlab ="Factor
1",ylab="Factor 2",main="PARAFAC Scores" )

## plot parafac loadings - emission

dev.new ()
matplot (t( nmEM ), model$B,type ="1", xlab ="Emission Wavelength (nm)", ylab
="Intensity")

## plot parafac loadings - excitation

dev.new ()
matplot (t( nmEX ), model$C,type ="1", xlab ="Excitation Wavelength (nm)", ylab
="Intensity")

## Division into Calibration and Prediction

perc = 0.7 # 70% for calibration and 30% for testing

A = model$A # PARAFAC Scores

dim_x = dim (x)
size = 1:dim_x[1]

ntrain = ceiling ( perc * dim_x [1]) # number of calibration samples

sel = sample(size, ntrain ) # sample selection

xcal = A[ sel ,] # calibration
ycal = matrix (y[ sel ]) # concentration calibration

```

```

xpred = A[- sel ,] # prediction
ypred = matrix (y[- sel ]) # concentration prediction

## Regression model

xcal_df = data.frame ( xcal )
xpred_df = data.frame ( xpred )

model_mlr = lm ( ycal ~ xcal , xcal_df ) # MLR model

coef = model_mlr$coefficients # regression coefficients

ycal_calc = xcal [,1] * coef [2] + xcal [,2] * coef [3] + coef [1] # predicted
concentration calibration
ypred_calc = xpred [,1] * coef [2] + xpred [,2] * coef [3] + coef [1] # predicted
concentration prediction

## Plot measured concentration vs. prediction

dev.new ()
plot (ycal,ycal_calc,pch ='o', col ="blue", xlab ="Measured Concentration (mg/L)",
ylab ="Predicted Concentration (mg/L)", main ="Calibration")
lines(ycal,ycal,col ='red')

dev.new ()
plot (ypred,ypred_calc,pch =15,col="red", xlab = "Measured Concentration (mg/L)",
ylab = "Predicted Concentration (mg/L)", main = "Predicted")
lines(ypred,ypred,col ='red')

dev.new ()
plot (ycal,ycal_calc,pch ='o', col ="blue", xlab ="Measured Concentration (mg/L)",
ylab ="Predicted Concentration (mg/L)", main ="Calibration (blue) and Prediction
(red)")
points(ypred,ypred_calc,pch =15,col="red")
lines(y,y,col ="red")

## Figures of merit

# Calibration

MAPEC = mean( abs (( ycal-ycal_calc )/ ycal ))*100
R2cal = cor( ycal,ycal_calc )^2
RMSEC = rmse( ycal,ycal_calc )

# Prediction

MAPEP = mean(abs(( ypred-ypred_calc )/ ypred ))*100
R2pred = cor ( ypred,ypred_calc )^2
RMSEP = rmse ( ypred,ypred_calc )

```

```

print("==== Calibration =====")
print("Mean absolute error percentage (MAPE) (%):")
MAPEC
print("R2cal:")
R2cal
print("RMSEC (mg/L):")
RMSEC

print("==== Test =====")
print("Mean absolute error percentage (MAPE) (%):")
MAPEP
print("R2pred:")
R2pred
print("RMSEP (mg/L):")
RMSEP

```

5.7 MCR-ALS

As discussed in the previous chapter, the MCR-ALS algorithm is the alternating least squares multivariate curve resolution algorithm whose main purpose is to deconvolve curves in bilinear data. In other words, it reassembles the original data matrix through individual contributions from a given number of recovered profiles plus a random noise. However, MCR can be viewed as a signal resolution method that can also be used for quantitative purposes.

The following equation describes the decomposition of a data matrix into a bilinear model that makes it possible to estimate concentration and spectral profiles individually, maintaining the best data variance:

$$D = CS^T + E \quad \text{Eq. 35}$$

Where **D** corresponds to the instrumental matrix (**j** × **k**), **C** corresponds to the relative concentration matrix (**j** × **i**), **S** corresponds to the pure spectra matrix (**i** × **k**), and **E** corresponds to the residuals matrix (**j** × **k**).

In the iterative resolution of equation 35 described above, it is necessary to estimate the number of components present in the mixture that produces the analytical signal; in other words, we must estimate the rank of the data matrix. The number of components is normally estimated based on knowledge of the investigated system or from the decomposition results, using, for example, the singular value decomposition (SVD) algorithm. In the SVD algorithm, the number of species is approximated by the number of singular values, and above the singular value we have the estimate of the instrumental noise level of the data.

Unlike PARAFAC, which uses a trilinear decomposition and requires the presence of at least two different samples to form a data cube, MCR is based on bilinear decomposition

of data and can be applied to more than one data matrix simultaneously, as well as a single sample (resolution or deconvolution of signals). Here we have one of the major limitations of the MCR-ALS algorithm: the problem of freedom of rotation or ambiguity, that is, the existence of more than one set of profiles that present the same fit to the data. In PARAFAC, as it is based on a trilinear decomposition, this ambiguity problem is absent.

However, ambiguity minimization in MCR-ALS models is achieved through some restrictions, such as:

- i) Non-negativity - the calculated profiles cannot be negative. This restriction can be applied to both concentrations and analytical signal;
- ii) Unimodality – requires that the calculated profiles have only one maximum. This restriction can also be imposed on matrices \mathbf{C} or \mathbf{S}^T .
- iii) Closure – the sum of relative concentrations remains constant during the optimization process. This restriction is only applied to the concentration profile.
- iv) Trilinearity – consists of subjecting the spectral and concentration profiles to non-variation between one sample and another, establishing a unique response.

Here we list some mathematical operations used in the MCR-ALS calibration models:

- i) Determination of the number of components (n).
- ii) Construction of the initial \mathbf{S} or \mathbf{C}_k concentration spectrum profile matrix.
- iii) Selection of restrictions to be applied.
- iv) Optimization of initial estimates (ALS) in each iteration.
- v) Reproduction of the initial matrix \mathbf{D}_k in each iteration from \mathbf{S} and \mathbf{C}_k .
- vi) Repeat steps iv and v until the convergence criterion is satisfied.
- vii) Determination of the matrix of concentration profiles and spectra.
- viii) Construction of the least squares regression model between the concentration profile of the species of interest from step vii and the vector with the reference concentrations of the calibration samples. A pseudo-univariate regression is therefore performed based on the recovered concentration profiles, their area or norm, and the analytical concentration of the sample of interest.

The MCR-ALS algorithm is successfully used on 2nd order data through data matrices augmented by rows and/or columns, as can be seen in **Figure 5.7**:

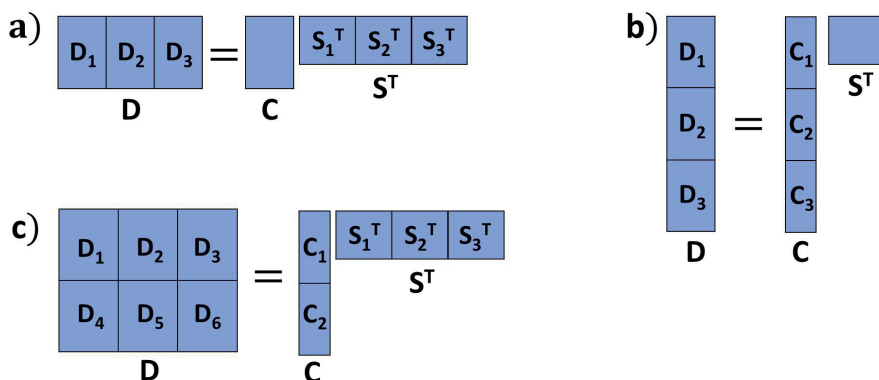


Figure 5.7: Bilinear model of the augmented MCR: (a) by rows, (b) by columns, and (c) by rows and columns simultaneously.

In higher order calibration models, quantification using MCR-ALS presents several advantages over conventional calibration. Here, we will list the advantages in order of importance:

- I. Neither knowledge nor the inclusion of interferers in the calibration model is necessary, thus achieving the so-called 2nd order advantage;
- II. Possibility of using a smaller number of calibration samples as the regression is performed based on the relative concentration profile recovered by MCR-ALS as a function of known concentration values;
- III. Figures of merit in the validation stage can be calculated in a similar way to univariate calibration as the model takes a very simple mathematical form (pseudo-univariate).

Example 8: The following example consists of an application of the MCR-ALS multivariate regression algorithm applied to mid-infrared spectra (2 replicates per experiment) in plasma samples with different synthetic (spiked) concentrations of dengue virus. The packages used in the script are *R.matlab*, *prospectr*, *MASS*, *lintools*, *ggplot2*, *lintools*, *Stat2Data*, *Metrics* and *ALS*.

R Script

```
## Loading packages

install.packages ("R.matlab")
library(R.matlab)

install.packages ("prospectr")
install.packages ("MASS")
install.packages ("lintools")

library(prospectr)
library(MASS)
library(ggplot2)
library(lintools)

install.packages ("Stat2Data")
library(Stat2Data)

install.packages ("Metrics")
library(Metrics)

install.packages ("ALS")
library (ALS)

## Loading data

# Navigate in RStudio to the directory with the dataset to work with
# Session > Set Working Directory > Choose Directory

data <- readMat ("data_reg_mcr.mat")

x1 = data$data1 # spectra replicate 1
x2 = data$data2 # spectra replicate 2
y1 = data$concentration1 # replicate concentration 1
y2 = data$concentration2 # concentration replicate 2
cm = data$cm # wave number
cmt = t(cm)
xm = (x1+x2)/2 # average spectra
ym = (y1+y2)/2 # average concentration

## data plot

dev.new ()
matplot (t(cm),t(xm), type ='l', xlab ="Wavenumber (cm-1)", ylab ="Absorbance")

## MCR-ALS

dimS = dim (x1) # matrix dimensions
```

```

ncomp = 2 # number of components (component 1=sample, component 2=waste)

mcr <- als (CList=list(y1,y2), S=matrix(1,nrow= dimS [2], ncol=ncomp),
PsiList=list(x1,x2), normS=0)

# MCR-ALS with non-negativity in concentration (nonnegC=TRUE) and normalization
(normS =0)

mcr <- als (CList=list(y1,y2), S=matrix(1,nrow=dimS [2], ncol=ncomp),
PsiList=list(x1,x2), nonnegC=TRUE, normS=0)

## Plot of recovered components (component 1= sample, component 2 = waste)

plots (mcr$S,cm)

## comparing the spectrum of component 1 with the original spectrum

matchFactor (colMeans (xm), mcr$S [,1])

# Copt (concentration) values

matplot (ym, mcr$CList [[1]], pch="o", col="blue", xlab="Measured concentration
(mg/L)", ylab="Copt1") #component 1
matplot (ym,ym,type="l",col="red",add=TRUE)

Copt1=matrix (mcr$CList [[1]][,1]) # Copt of component 1

print("R2:")
print(cor(ym [,1],Copt1)^2) # R2 value between Copt and real concentration

## Division into Calibration and prediction

perc=0.7 # 70% for calibration and 30% for prediction

dim_xm=dim (xm)

size=1:dim_xm[1]

ntrain=ceiling (perc*dim_xm [1]) # number of calibration samples

sel=sample(size, ntrain) # Sample selection

xcal=Copt1[sel,] # calibration
ycal = matrix (ym [sel,1]) # concentration calibration

xpred = Copt1[-sel,] # prediction
ypred = matrix (ym [-sel,1]) # concentration prediction

```

```

## Regression model

xcal_df = data.frame (xcal)
xpred_df = data.frame (xpred)

model_mlr = lm (yca1~xcal , xcal_df) # MLR model

coef = model_mlr$coefficients # regression coefficients

yca1_calc = xcal * coef [2] + coef [1] # predicted concentration calibration
ypred_calc = xpred * coef [2] + coef [1] # predicted concentration prediction

## Plot measured concentration vs. Prediction

y = ym [.1] # real concentration

dev.new ()
plot (yca1,yca1_calc,pch='o', col="blue", xlab="Measured Concentration (mg/L)",
ylab="Predicted Concentration (mg/L)", main="Calibration")
lines(yca1,yca1,col="red")

dev.new ()
plot (ypred,ypred_calc,pch=15,col="red", xlab="Measured Concentration (mg/L)",
ylab="Predicted Concentration (mg/L)", main="Predicted")
lines(ypred,ypred,col='red')

dev.new ()
plot (yca1,yca1_calc,pch='o', col="blue", xlab="Measured Concentration (mg/L)",
ylab="Predicted Concentration (mg/L)", main="Calibration (blue) and prediction
(red)")
points(ypred,ypred_calc,pch=15,col="red")
lines(y,y,col="red")

## Figures of merit

# Calibration

MAPEC = mean (abs((yca1-yca1_calc)/yca1))*100
R2cal = cor(yca1,yca1_calc)^2
RMSEC = rmse (yca1,yca1_calc)

# Prediction

MAPEP=mean(abs((ypred-ypred_calc)/ypred))*100
R2pred=cor(ypred,ypred_calc)^2
RMSEP=rmse(ypred,ypred_calc)

```

```

print("==== Calibration =====")
print("Mean absolute error percentage (MAPE) (%):")
MAPEC
print("R2cal:")
R2cal
print("RMSEC (mg/L):")
RMSEC

print("==== Test =====")
print("Mean absolute error percentage (MAPE) (%):")
MAPEP
print("R2pred:")
R2pred
print("RMSEP (mg/L):")
RMSEP

```

5.8 UPLS/RBL

The PLS algorithm must be the first-order calibration model most used and described in the literature. The Unfolded Partial Least Squares (UPLS) algorithm is an extended version of partial least squares (PLS) regression, proposed by Wold and Bro [4,5], used for processing multilinear data by unfolding the data matrix into vectors. However, unlike PARAFAC and MCR-ALS, this algorithm cannot obtain the second-order advantage when interferences are present in the test sample but not present in the calibration set. To overcome this limitation and achieve second-order data analysis, a mathematical procedure called residual bilinearization (RBL) [4] was developed, which can model the residuals of the test sample as a sum of the bilinear contributions of unexpected components. Therefore, the function of RBL is to model the residuals and present results from test samples free of interferences, adjusting the values of the score matrix with the information of the interferent modeled separately by the RBL step.

In the first calibration step of the UPLS-RBL algorithm, a data tensor with dimensions ($i \times j \times k$) is unfolded into a two-dimensional matrix ($i \times jk$), where i corresponds to the number of samples, j is the dimension corresponding to the excitation spectra, and k is the dimension of the emission spectra, in the case of molecular fluorescence data. With all the calibration data unfolded, a new matrix is constructed by arranging them adjacent to each other for the application of UPLS regression, as represented in **Figure 5.8** below:

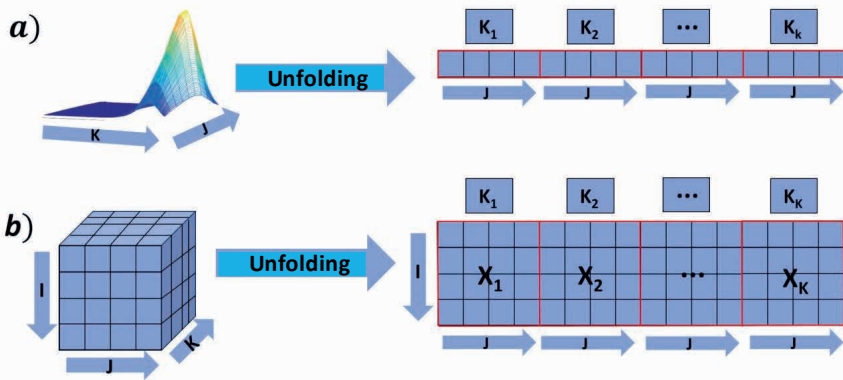


Figure 5.8: Representation to show the breakdown of a sample into a vector (a) and a set of samples into vectors (b) for building a UPLS model.

Initially, the UPLS-RBL algorithm unfolds the calibration data matrix (\mathbf{X}_{cal}), and a PLS model is built in the traditional way together with a cross-validation step to define the number of latent variables. The weight (\mathbf{P}) and scores (\mathbf{t}) matrices, originating from the unfolded matrix, are estimated iteratively by maximizing the variance of \mathbf{X}_{cal} and its covariance with the response vector \mathbf{y}_{cal} . Then, with the calculated \mathbf{t} values, a regression model is calculated between the nominal concentration vector \mathbf{y}_{cal} and \mathbf{t} to estimate the regression vectors \mathbf{v} that minimizes the error \mathbf{e}_y , as shown in the following equation:

$$\mathbf{y}_{cal} = \mathbf{t}\mathbf{v} + \mathbf{e}_y \quad \text{Eq. 36}$$

If there is a UPLS residue in the test sample greater than the calibration one, we have the presence of some uncalibrated constituent in the residue or the presence of interferents in the sample. If this occurs, an RBL post-calibration step is necessary to remove the contribution of this interference from the test sample's score value.

In the RBL step, the UPLS residue vector is reassembled into a matrix with the original dimensions, and an SVD is performed on the reassembled test sample residuals matrix (\mathbf{E}_p). When the test sample noise approaches the calibration value, we have the correct value of the amount of RBL needed to be used in the calibration model. Equation 37 shows the equation of the test sample of a classic PLS plus the contribution of the interferent found in the RBL step plus the unfolded RBL residue (\mathbf{e}_u).

$$\mathbf{x}_u = (\mathbf{P}\mathbf{t}_u^T) + \mathbf{BGC}^T + \mathbf{e}_u \quad \text{Eq. 37}$$

Where \mathbf{x}_u represents the test sample; $(\mathbf{P}\mathbf{t}_u^T)$ represents the analyte signal; \mathbf{BGC}^T represents the interference signal; and, \mathbf{e}_u represents the random error after the RBL step.

After convergence, the values obtained for t_u are used to estimate the sample concentration by the relationship expressed in Eq. 36. Therefore, the RBL step calculates and removes the interference contribution, represented as \mathbf{BGC}^T of the noise after UPLS modeling. Finally, after separating the interferent from the noise, the scores used to estimate the analyte concentration are corrected and can be used for quantification.

Example 9: The following example consists of an application of the UPLS multivariate regression algorithm in the analysis of fluorescence data (excitation-emission matrix) obtained in plasma samples with different synthetic concentrations (spiked) of a fluorescent standard. This script in R language uses the *R.matlab*, *pls*, *Stat2Data* and *Metrics* packages.

R Script

```
# Loading packages

install.packages ("R.matlab")
library(R.matlab)

install.packages ("pls")
library(pls)

install.packages ("Stat2Data")
library(Stat2Data)

install.packages ("Metrics")
library (Metrics)

## Loading Data

# Navigate in RStudio to the directory with the dataset to work with
# Session > Set Working Directory > Choose Directory

data <- readMat ("data_reg_parafac.mat")

x <- data$data
y <- data$concentration
nmEX <- data$nmEX
nmEM <- data$nmEM

## Visualizing the data

# matrix dimensions

mydim = dim (x) # samples x emission x excitation
mydim
```



```

# average matrix

xm = colMeans (x)

dev.new ()
filled.contour (xm,color.palette = terrain.colors)

dev.new ()
matplot (t(nmEM), xm,type="l", xlab="Emission Wavelength (nm)", ylab="Intensity")

dev.new ()
matplot (t(nmEX),t(xm), type="l", xlab="Excitation Wavelength (nm)",
ylab="Intensity")

## Division into Calibration and Prediction

Xr=matrix (x,nrow = mydim [1])
perc=0.7 # 70% for calibration and 30% for testing

dim_x = dim (x)
size = 1:dim_x[1]

ntrain = ceiling (perc * dim_x [1]) # number of calibration samples

sel = sample(size, ntrain) # KS

xcal = Xr [sel ,] # calibration
ycal = matrix (y[sel]) # concentration calibration

xpred = Xr [-sel,] # prediction
ypred = matrix (y[-sel]) # concentration prediction

## UPLS Model

xcal_df = data.frame (xcal)
xpred_df = data.frame (xpred)

model_pls = pls (ycal~xcal, data=xcal_df, validation="CV")

## Determine number of components

validationplot (model_pls)

validationplot (model_pls,val.type="MSEP")

validationplot (model_pls,val.type="R2")

ncomp = 2 # number of selected components

```

```

## Prediction

ycal_calc <- predict(model_pls, xcal, ncomp=ncomp)
ypred_calc <- predict(model_pls, xpred, ncomp=ncomp)

ycal_calc = ycal_calc [,1,1]
ypred_calc = ypred_calc [,1,1]

## Plot measured concentration vs. Prediction

dev.new ()
plot (ycal,ycal_calc,pch='o', col="blue", xlab="Measured Concentration (mg/L)",
ylab="Predicted Concentration (mg/L)", main="Calibration")
lines(ycal,ycal,col='red')

dev.new ()
plot (ypred,ypred_calc,pch=15,col="red", xlab="Measured Concentration (mg/L)",
ylab="Predicted Concentration (mg/L)", main="Predicted")
lines(ypred,ypred,col='red')

dev.new ()
plot (ycal,ycal_calc,pch='o', col="blue", xlab="Measured Concentration (mg/L)",
ylab="Predicted Concentration (mg/L)", main="Calibration (blue) and prediction
(red)")
points(ypred,ypred_calc,pch=15,col="red")
lines(y,y,col="red")

## Figures of merit

# Calibration

MAPEC = mean (abs((ycal-ycal_calc)/ycal))*100
R2cal = cor(ycal,ycal_calc)^2
RMSEC = rmse (ycal,ycal_calc)

# Prediction

MAPEP = mean(abs((ypred-ypred_calc)/ypred))*100
R2pred = cor (ypred,ypred_calc)^2
RMSEP = rmse (ypred,ypred_calc)

print("==== Calibration =====")
print("Mean absolute error percentage (MAPE) (%):")
MAPEC
print("R2cal:")
R2cal
print("RMSEC (mg/L):")
RMSEC

```

```

print("===== Test =====")
print("Mean absolute error percentage (MAPE) (%):")
MAPEP
print("R2pred:")
R2pred
print("RMSEP (mg/L):")
RMSEP

```

5.9 N-PLS

The N-way Partial Least Squares (N-PLS) algorithm, proposed by Bro [5], is also an extension of the PLS model. Basically, the N-PLS algorithm decomposes three-dimensional arrays $\underline{\mathbf{X}}$ ($i \times j \times k$) (independent data cubic matrix) and the vector of reference concentrations \mathbf{Y} ($i \times 1$) (or physicochemical property) in a set of triads, to find the maximum covariance between the scores of $\underline{\mathbf{X}}$ and \mathbf{Y} .

The construction of N-PLS models, analogous to traditional PLS, is carried out in two stages: calibration and prediction. Each triad is equal to a latent variable as in the PLS model, and in the calibration stage the arrangement $\underline{\mathbf{X}}$ is decomposed into scores (\mathbf{t}_n) and weights (\mathbf{w}^j and \mathbf{w}^k), as exemplified in Figure 5.9 below:

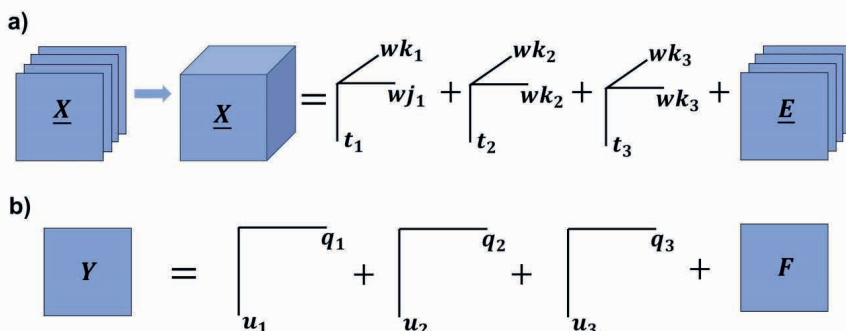


Figure 5.9: Schematic representation of the N-PLS algorithm.

In Figure 5.9, $\underline{\mathbf{X}}$ is the cubic matrix of independent data, $\mathbf{T}_{(i,F)}$ is the score matrix and the matrices $\mathbf{w}_{(j,F)}^j$ and $\mathbf{w}_{(k,F)}^k$ are weight matrices containing information about the variables. The arrangement $\underline{\mathbf{E}}$ ($i \times j \times k$) represents the part not explained by the model (residuals) and \mathbf{F} represents the number of latent variables. The matrix \mathbf{Y} is also decomposed into scores and weights, represented by Figure 5.9, where \mathbf{Y} is the matrix containing the property of interest, \mathbf{U} is the matrix containing the scores of \mathbf{Y} , \mathbf{Q} is the matrix containing the loadings of \mathbf{Y} , and \mathbf{F} is the matrix of residues of \mathbf{Y} that, as in the arrangement $\underline{\mathbf{X}}$, cannot be explained by the model.

The decomposition of the three-dimensional array $\underline{\mathbf{X}}$ can be represented by the following equation:

$$\underline{\mathbf{X}}_{i,j,k} = \sum t_{ij} \mathbf{w}_{j,F}^j \mathbf{w}_{k,F}^k + \underline{\mathbf{E}}_{i,j,k} = \underline{\mathbf{X}} = \mathbf{T}(\mathbf{W}^K \otimes |\mathbf{W}^J|) + \underline{\mathbf{E}} \quad \text{Eq. 38}$$

Where the symbol " \otimes " represents the Khatri-Rao product, an operator used in higher order matrices.

For the concentration vector or physicochemical parameter of interest (\mathbf{y}), the decomposition is written by the equation below:

$$\mathbf{y} = \mathbf{T}\mathbf{b}^T \text{ where } \mathbf{b} = (\mathbf{T}^T\mathbf{T})^{-1}\mathbf{T}^T\mathbf{y} \quad \text{Eq. 39}$$

where \mathbf{T} is a scores matrix, whose columns consist of the individual score vectors of each component and \mathbf{b} are the regression coefficients.

Finally, the predicted concentration of samples with unknown concentrations (\mathbf{y}^*), can be estimated from new scores (\mathbf{T}^*) according to the following equation:

$$\mathbf{y}^* = \mathbf{T}^*\mathbf{b}^T \quad \text{Eq. 40}$$

Some considerations deserve to be highlighted in the N-PLS models. Despite the simplicity of the calibration models and the simplicity in interpreting the results, in addition to the lower sensitivity to noise, the 2nd order advantage will only be achieved with the application of the RBL algorithm. In the original work, Bro [5] compares and suggests that N-PLS is superior to UPLS because it uses data in its original form (without performing unfolding).

Example 10: The following example consists of an application of the N-PLS multivariate regression algorithm in the analysis of fluorescence data (excitation-emission matrix) obtained in plasma samples with different synthetic (spiked) concentrations of a fluorescent standard. This script in R language uses the *R.matlab*, *pls*, *Stat2Data*, *Metrics* and *sNPLS* packages.

R Script

```
## Loading packages

install.packages ("R.matlab")
library(R.matlab)

install.packages ("pls")
library(pls)

install.packages ("Stat2Data")
library(Stat2Data)

install.packages ("Metrics")
library(Metrics)

install.packages ("sNPLS")
library (sNPLS)

## Loading Data

# Navigate in RStudio to the directory with the dataset to work with
# Session > Set Working Directory > Choose Directory

data <- readMat ("data_reg_parafac.mat")

x <- data$data
y <- data$concentration
nmEX <- data$nmEX
nmEM <- data$nmEM

## Visualizing the data

# matrix dimensions

mydim = dim (x) # samples x emission x excitation
mydim

# average matrix

xm = colMeans (x)

dev.new ()
filled.contour (xm,color.palette=terrain.colors)

dev.new ()
matplot (t(nmEM), xm,type="l", xlab="Emission Wavelength (nm)", ylab="Intensity")

dev.new ()
matplot (t(nmEX),t(xm), type="l", xlab="Excitation Wavelength (nm)",
ylab="Intensity")
```

```

## Division into Calibration and Forecast

perc = 0.7 # 70% for calibration and 30% for testing

dim_x = dim (x)
size = 1:dim_x[1]

ntrain = ceiling (perc*dim_x [1]) # number of calibration samples

sel = sample(size, ntrain) # sample selection

xcal = x[sel ,,] # calibration
ycal = matrix (y[sel]) # concentration calibration

xpred = x[-sel,,] # prediction
ypred = matrix (y[-sel]) # concentration prediction

## nPLS Model

cv = cv_snpls(xcal,ycal,ncomp=1:3,keepJ=1:2,keepK=1:2,sample=10,parallel=FALSE) #
cross-validation - takes a long time

ncomp = 2 # set number of components

model_npls = snPLS (xcal,ycal,ncomp=ncomp, keepJ=rep(2,ncomp), keepK=rep(1,ncomp))
# nPLS model

# Prediction

ycal_calc=predict(model_npls, xcal)
ypred_calc = predict(model_npls, xpred)

## Plot measured concentration vs. Prediction

dev.new ()
plot (ycal,ycal_calc,pch='o', col="blue", xlab="Measured Concentration (mg/L)",
ylab="Predicted Concentration (mg/L)", main="Calibration")
lines(ycal,ycal,col='red')

dev.new ()
plot (ypred,ypred_calc,pch=15,col="red", xlab="Measured Concentration (mg/L)",
ylab="Predicted Concentration (mg/L)", main="Predicted")
lines(ypred,ypred,col='red')

dev.new ()
plot (ycal,ycal_calc,pch='o', col="blue", xlab="Measured Concentration (mg/L)",
ylab="Predicted Concentration (mg/L)", main="Calibration (blue) and Prediction
(red)")
points(ypred,ypred_calc,pch =15,col="red")
lines(y,y,col="red")

```

```

## Figures of merit

# Calibration

MAPEC = mean (abs((ycal-ycal_calc)/ycal))*100
R2cal = cor(ycal,ycal_calc)^2
RMSEC = rmse (ycal,ycal_calc)

# Prediction

MAPEP = mean(abs((ypred-ypred_calc)/ypred))*100
R2pred = cor(ypred,ypred_calc)^2
RMSEP = rmse (ypred,ypred_calc)

print("=====  

print("Mean absolute error percentage (MAPE) (%):")
MAPEC
print("R2cal:")
R2cal
print("RMSEC (mg/L):")
RMSEC

print("=====  

print("Mean absolute error percentage (MAPE) (%):")
MAPEP
print("R2pred:")
R2pred
print("RMSEP (mg/L):")
RMSEP

```

PROPOSED EXERCISES

01 – Compare the performance of a univariate calibration model with several polynomial degrees in a data set through a script in the R language, presenting the models' performance, error and conclusions.

02 – Propose an application of the MLR algorithm on a data set with more than one independent variable and, using a script in the R language, present your results, graphs and conclusions.

03 – From the previous exercise, use the MLR-SPA algorithm through a script in the R language and compare the results of the calibration models with and without the variable selection algorithm.

04 – Build multivariate calibration models for the PCR and PLS algorithms on a given 1st order data set using an R script and present your results, figures of merit for both models and your main conclusions.

05 – Apply the PARAFAC algorithm to a 2nd order data set (molecular fluorescence in excitation-emission mode or chromatography, for example) using R language and present your results and conclusions.

06 – Apply the MCR-ALS algorithm to a 2nd order data set (molecular fluorescence in excitation-emission mode or chromatography, for example) using R language and present your results and conclusions.

07 – Apply the UPLS algorithm to a 2nd order data set (molecular fluorescence in excitation-emission mode or chromatography, for example) using R language and present your results and conclusions.

08 – Apply the N-PLS algorithm to a 2nd order data set (molecular fluorescence in excitation-emission mode or chromatography, for example) using R language and present your results and conclusions.

09 – Using a 2nd order data set, perform a comparison between the calibration models (PARAFAC, MCR-ALS, UPLS and N-PLS), presenting their results, figures of merit and main conclusions.

REFERENCES

1 – Legendre, AM (1805). *Nouvelles Méthodes pour la Détermination des Orbites des Comètes*, Firmin Didot, Paris; second edition Courcier, Paris.

2 – Hold, H. (1982). *Systems under Indirect Observation*. North-Holland, Amsterdam, 1982.

3 – Vandeginste, BGM; Sielhorst, C.; Gerritsen, M.; (1988). The NIPALS algorithm for the calculation of the principal components of a matrix. *Trends Anal. Chem.* 7, 286-287.

4 – Wold, S.; Geladi, P.; Esbensen, K.; Öhman, J. (1987). Multi-way main components and PLS-analysis. *J. Chemom.* 1, 41.

5 – Bro, R. (1996). Multiway calibration. *Multilinear PLS. J. Chemom.* 10, 47

4 – Olivieri, AC, and Escandar, GM (2000). *Practical Three-Way Calibration*. Elsevier.

DIGITAL IMAGES



"A picture is worth a thousand words." **Confucius (552 BC - 489 BC)**

CHAPTER IDEA

A digital image can be interpreted as a representation of a scene through a set of discrete elements of finite size, known as pixels, organized in a two-dimensional arrangement. Commonly, the acquisition of digital images occurs through electronic devices (photo cameras, webcams, drones, for example) in a process known as optoelectronic transduction, which involves a reduction in the dimensionality of the scene through a sensor (Charge Coupled Device, for example).

In this chapter you will find a theoretical foundation of digital images, color models and some chemometric studies using digital images in classification and multivariate calibration models. Examples guided by multivariate algorithms in the R language using digital images will be found throughout the chapter, as well as details of the models developed.

Upon completing the chapter, you should be able to:

- a) Understand the stages of acquiring and importing digital images, their pre-processing and construction of calibration and multivariate classification models using scripts in the R language;
- b) Employ algorithms of unsupervised analysis on digital images using R scripts;
- c) Build and validate multivariate classification models using digital images with R scripts;
- d) Build and validate multivariate calibration models using digital images with R scripts;
- e) Compare models based on variable selection for classification and calibration of digital images;
- f) Propose new applications in chemistry or related areas using digital images in one of the areas of Chemometrics.

6.1 Digital Imaging: an overview

An image to be processed on a computer must be in digital format, and this is represented by a two-dimensional matrix of $M \times N$ pixels. The word pixel is an abbreviation of "Picture element", which means "image element". Thus, the pixel is the smallest element of a digital image and each pixel location in a monochrome image (normally 8 bits) corresponds to the gray level ranging from black (0) to white (255), thus being able to contain 256 levels of gray colors. Figure 6.1 presents a monochrome image and its representation in a digital image in the form of a data matrix.

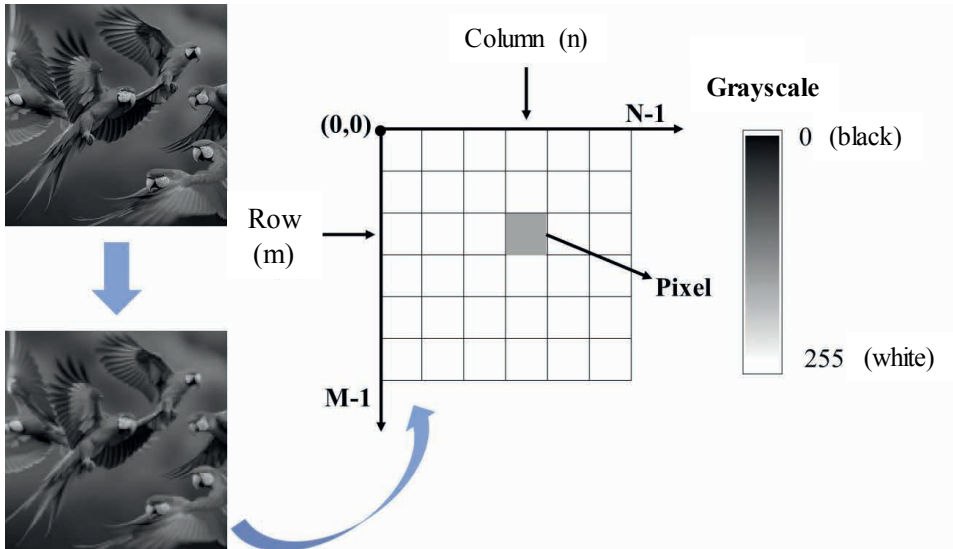


Figure 6.1: Representation of a digital image in the form of an $M \times N$ pixel matrix.

Mathematically, as shown in equation 1, we can consider a pixel as a vector formed by three monochromatic images in which the components represent the intensities of the RGB model [1] (red (R), green (G) and blue (B)), which corresponds to the primary colors.

$$f(x,y) = f_r(x,y) + f_g(x,y) + f_b(x,y) \quad \text{Eq. 1}$$

The main purpose of the RGB color model is for the detection, representation and exhibition of images in electronic systems such as televisions and computers. The frequency distribution of the values which a pixel contains in the image is called a histogram. It shows how many times a varying color value (0-255) can appear in the image. To illustrate this concept, **Figure 6.2** presents a color digital image and the histograms referring to the frequency distribution of all possible values of a pixel in the red, green, blue and gray levels.

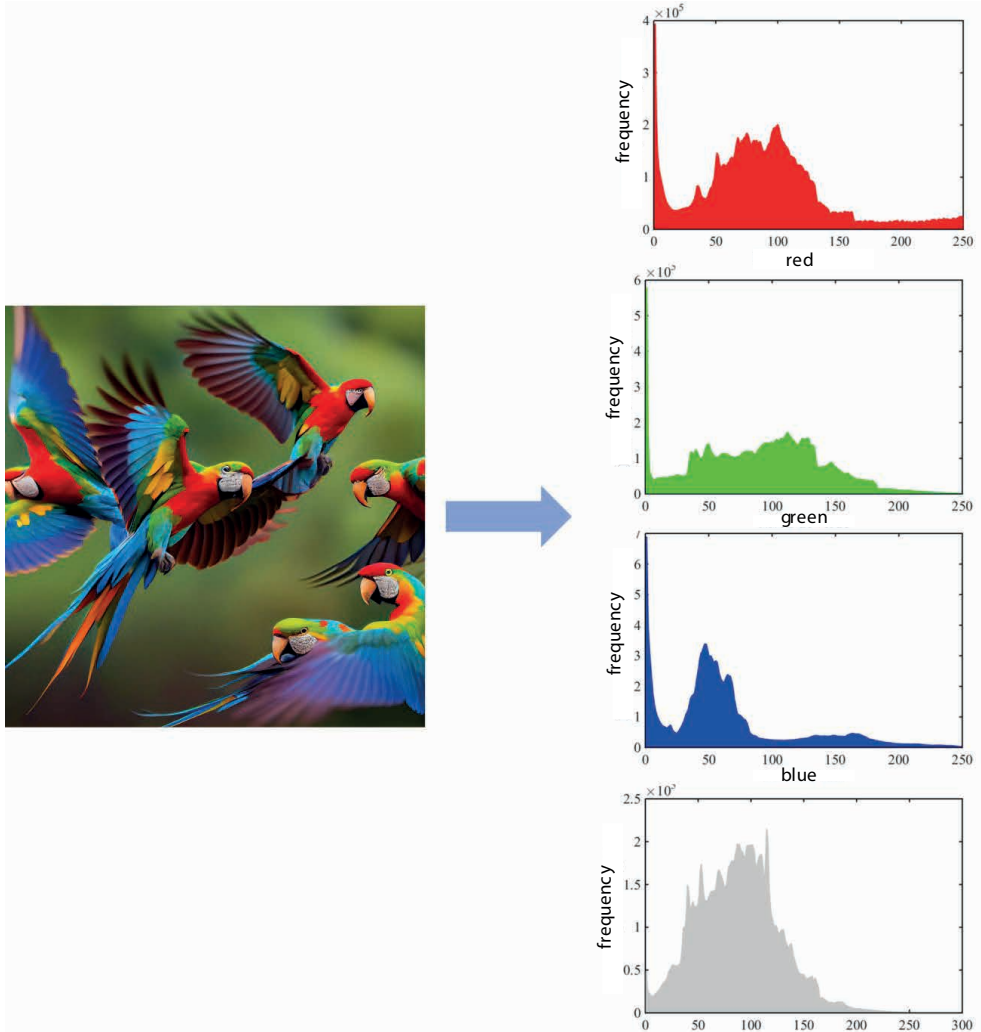


Figure 6.2: Histograms in the red, green, blue and gray channels resulting from a color digital image.

We can represent the RGB model by a cube on the R, G and B axes, which takes on 256 color levels or values from 0-255. Each color channel is made up of a set of 8 bits resulting in an image with 16.7 million different colors. As shown in **Figure 6.3**, the edges of the cube have the primary colors of the RGB model and the faces in the planes GB, BR, RG have the secondary colors (cyan, magenta and yellow), formed from the combination of two primary colors. Black corresponds to the origin of the cube, white corresponds to the vertex furthest from the origin, and gray corresponds to the diagonal between these two points.

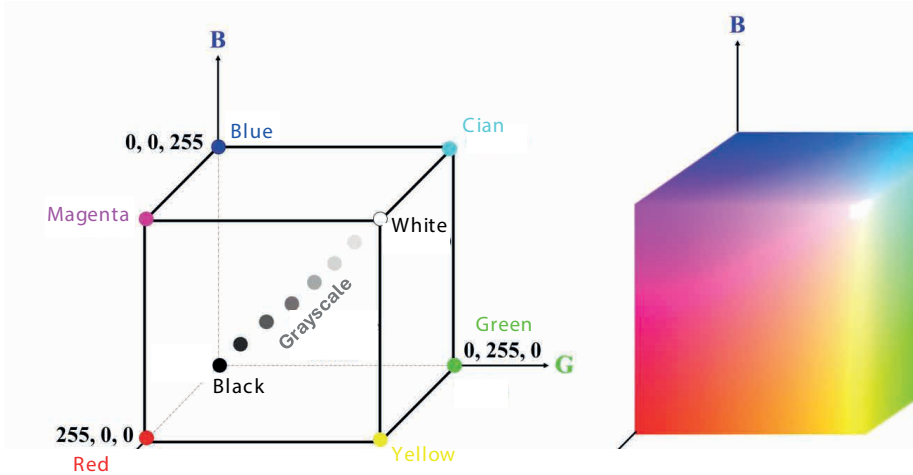


Figure 6.3: RGB color model

In other words, zero intensity for each component gives the darkest color (without light, considered black) and the total intensity of each one results in white. When the intensities of all components are the same, the result is a shade of gray, darker or lighter, depending on the intensity. When the intensities are different, the result is a colorful hue, more or less saturated, depending on the difference between the strongest and weakest intensities of the primary colors used [2].

Another color model, based on polar coordinates, represented by an inverted six-sided pyramid, frequently used in computer graphics is called HSV, as shown in **Figure 6.4**.

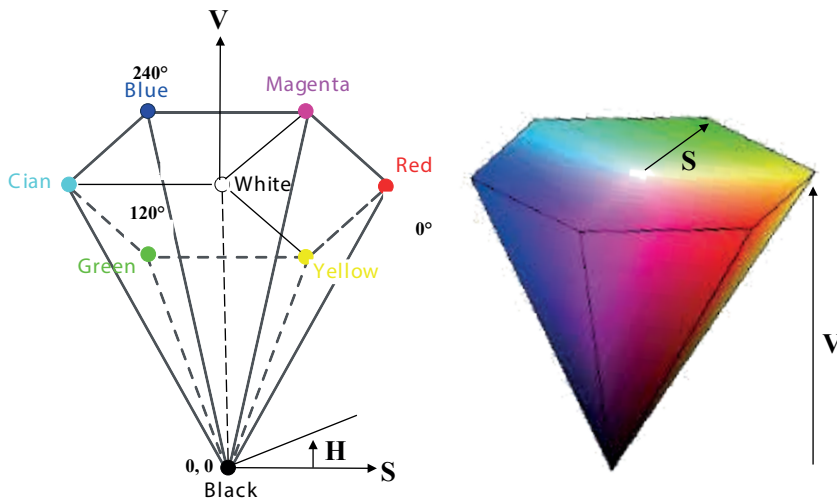


Figure 6.4: HSV color model

This model describes three fundamental attributes: hue (H), saturation (S) and value (V)

(V). In this model, hue (H) is the dominant wavelength of the color (defines the tone of an area) and is measured in angles, arranged around the central axis, ranging from 0 ° to 360 ° (red corresponds to 0 °, green corresponds to 120° and blue to 240°). The secondary colors are in opposite positions (180°) on the graduated circle. Saturation (S) is the purity of color, in the sense of the amount of white light mixed with hue. In other words, saturation is given by the distance from the central axis to the edges, and can vary from 0 (completely white color) to 1 (pure color). Finally, the value (V), also known as luminance, is the brightness of the color and corresponds to the height of the pyramid, and can vary from 0 (black) to 1 (white) along the V axis of the pyramid, where the gray scale is located [3].

Next, several examples based on R scripts will be presented with the use of digital images in unsupervised and supervised analysis as well as in calibration and multivariate classification. Steps such as selecting the area on the figure, obtaining the histograms and arranging the data matrices for chemometric analysis (exploratory analysis, classification and multivariate regression) will be detailed throughout these examples.

6.2 RGB to HSV and Grayscale Conversion

Example 1: The example we will describe here consists of converting an RGB image to HSV and grayscale using the *imager* package in R.

R Script

```
## Loading packages

install.packages("imager")
library(imager)
library(purrr)

## Loading image

# Navigate to the directory containing the image for testing
# In RStudio, go in Session > Set Working Directory > Choose Directory...

im_rgb = load.image("bird.jpg")

## viewing image

plot (im_rgb)

## converting to HSV
RGBtoHSV ( im_rgb ) %>% imsplit ("c") %>%
  modify_at ( 2, ~ . / 2 ) %>% imappend ("c") %>%
  HSVtoRGB %>% plot(rescale=FALSE)

## converting to grayscale
grayscale ( im_rgb ) %>% plot ( colourscale = gray, rescale = FALSE)
```

6.3 Exploratory Analysis

Example 2: The example we will describe here consists of the exploratory analysis (PCA, HCA and K-means) of RGB images obtained from ELISA plates containing albumin and creatinine concentrations using the *imager*, *ggplot2*, *dplyr*, *prospectr*, *MASS*, *plot3D*, *plotly* and *factoextra* packages.

R Script

```
## Loading packages

install.packages("imager")
library(imager)
library(ggplot2)
library(dplyr)
library(prospectr)
library(MASS)

install.packages("plot3D")
library(plot3D)

install.packages("plotly")
library(plotly)

install.packages("factoextra")
library(factoextra)

# establishing the working directory

# Navigate to the directory containing the images for testing
# In RStudio, go in Session > Set Working Directory > Choose Directory...

## loading images

im1 = load.image ("creatinine/im1.png")
im2 = load.image ("creatinine/im2.png")
im3 = load.image ("creatinine/im3.png")
im4 = load.image ("creatinine/im4.png")
im5 = load.image ("creatinine/im5.png")
im6 = load.image ("creatinine/im6.png")
im7 = load.image ("creatinine/im7.png")
im8 = load.image ("creatinine/im8.png")

## generating histograms

im1df <- as.data.frame (im1)
im2df <- as.data.frame (im2)
im3df <- as.data.frame (im3)
im4df <- as.data.frame (im4)
im5df <- as.data.frame (im5)
im6df <- as.data.frame (im6)
im7df <- as.data.frame (im7)
im8df <- as.data.frame (im8)
```

```

## plotting histograms for each image - image 1

bdf <- mutate(im1df,channel=factor( cc,labels =c('R','G','B')))
ggplot( bdf,aes (value,col=channel))+geom_histogram(bins=30)+facet_wrap(~ channel)

## image 2

bdf <- mutate(im2df,channel=factor( cc,labels =c('R','G','B')))
ggplot( bdf,aes (value,col=channel))+geom_histogram(bins=30)+facet_wrap(~ channel)

## image 3

bdf <- mutate(im3df,channel=factor( cc,labels =c('R','G','B')))
ggplot( bdf,aes (value,col=channel))+geom_histogram(bins=30)+facet_wrap(~ channel)

## image 4

bdf <- mutate(im4df,channel=factor( cc,labels =c('R','G','B')))
ggplot( bdf,aes (value,col=channel))+geom_histogram(bins=30)+facet_wrap(~ channel)

## image 5

bdf <- mutate(im5df,channel=factor( cc,labels =c('R','G','B')))
ggplot( bdf,aes (value,col=channel))+geom_histogram(bins=30)+facet_wrap(~ channel)

## image 6

bdf <- mutate(im6df,channel=factor( cc,labels =c('R','G','B')))
ggplot( bdf,aes (value,col=channel))+geom_histogram(bins=30)+facet_wrap(~ channel)

## image 7

bdf <- mutate(im7df,channel=factor( cc,labels =c('R','G','B')))
ggplot( bdf,aes (value,col=channel))+geom_histogram(bins=30)+facet_wrap(~ channel)

## figure 8

bdf <- mutate(im8df,channel=factor( cc,labels =c('R','G','B')))
ggplot( bdf,aes (value,col=channel))+geom_histogram(bins=30)+facet_wrap(~ channel)

## extracting all image pixels to a vector

im1v <- matrix (im1)
im2v <- matrix (im2)
im3v <- matrix (im3)
im4v <- matrix (im4)
im5v <- matrix (im5)
im6v <- matrix (im6)
im7v <- matrix (im7)
im8v <- matrix (im8)

```

```

## joining all images

data = rbind(t(im1v),t(im2v),t(im3v),t(im4v),t(im5v),t(im6v),t(im7v),t(im8v))

dim_data = dim (data)

## performing PCA

# scaling data

data_scal = scale ( data, center=TRUE, scale =FALSE)

# PCA Model

data.svd = svd ( data_scal ) # SVD
data.scores = data.svd$u %*% diag ( data.svd$d ) # scores
data.loadings = data.svd$v # loadings

# PCA Variances

data.vars = data.svd$d^2 / ( nrow (data)-1) # variance per PC
data.totalvar = sum( data.vars ) # total variance
data.relvars = data.vars / data.totalvar # cumulative variance
variances = 100 * round( data.relvars , digits = 3) # cumulative variance in %
variances [ 1:10] # variance in % in the first 10 PCs

# Choose the number of PCs

par(mfrow = c( 2,2))
barplot (data.vars [1:10], main="Variance", names.arg = paste("PC", 1:10))
barplot (log( data.vars [1:10]), main="Log( variance )", names.arg = paste("PC",
1:10))
barplot (data.relvars [1:10], main="Relative Variances", names.arg = paste("PC",
1:10))
barplot (cumsum (100* data.relvars [1:10]), main="Cumulative variances (%)", names.
arg = paste("PC", 1:10), ylim = c(0,100))

npc = 2 # number of PCs chosen to run the PCA

scores = data.scores [1:dim_data[1],1:npc] # PCA scores up to the chosen number of
PC
loadings = data.loadings [1:dim_data[2], 1:npc] # PCA loadings up to the chosen PC
number

# concentration vector in mg/ dL

y <- c( 0.03125,0.0625,0.125,0.25,0.5,1,2,4)
ys <- c( "1","2","3","4","5","6","7","8")

```



```

# plotting PCA scores PC1 x PC2

dev.new ( )
plot(data.scores      [,1],data.scores[,2],pch=ys,xlab='PC1',ylab='PC2',main='PCA
scores - numbers for each image')

# plotting PCA loadings PC1 & PC2

dev.new ( )
matplot(data.loadings
[,1],type="l",col="blue",xlab="Pixels",ylab='Loadings',main='PCA loadings')
lines(data.loadings
[,2],type="l",col="red",xlab="Pixels",ylab='Loadings',main='PCA loadings')

## repeating PCA to compare creatinine images vs. albumin

creatinine_data = data # saving creatinine data

## Loading albumin images

im1 = load.image ("albumin/im1.png")
im2 = load.image ("albumin/im2.png")
im3 = load.image ("albumin/im3.png")
im4 = load.image ("albumin/im4.png")
im5 = load.image ("albumin/im5.png")
im6 = load.image ("albumin/im6.png")
im7 = load.image ("albumin/im7.png")
im8 = load.image ("albumin/im8.png")

## extracting all image pixels to a vector

im1v <- matrix (im1)
im2v <- matrix (im2)
im3v <- matrix (im3)
im4v <- matrix (im4)
im5v <- matrix (im5)
im6v <- matrix (im6)
im7v <- matrix (im7)
im8v <- matrix (im8)

## joining all albumin images

albumin_data =
rbind(t(im1v),t(im2v),t(im3v),t(im4v),t(im5v),t(im6v),t(im7v),t(im8v))

## joining the images of creatinine and albumin

data = rbind ( creatinine_data , albumin_data )

```

```

group1 = rep(1,8) # class 1 - creatinine
group2 = rep(2,8) # class 2 - albumin
group12 = rbind ( matrix (group1), matrix (group2))

dim_data = dim (data)

## performing PCA

# scaling data

data_scal = scale ( data, center=TRUE, scale =FALSE)

# PCA Model

data.svd = svd ( data_scal ) # SVD
data.scores = data.svd$u %>% diag ( data.svd$d ) # scores
data.loadings = data.svd$v # loadings

# PCA Variances

data.vars = data.svd$d^2 / ( nrow (data)-1) # variance per PC
data.totalvar = sum( data.vars ) # total variance
data.relvars = data.vars / data.totalvar # cumulative variance
variances = 100 * round( data.relvars , digits = 3) # cumulative variance in %
variances [ 1:10] # variance in % in the first 10 PCs

# Choose the number of PCs

par(mfrow = c( 2,2))
barplot (data.vars [1:10], main=" Variance ", names.arg = paste("PC", 1:10))
barplot (log( data.vars [1:10]), main="Log( variance )", names.arg = paste("PC",
1:10))
barplot (data.relvars [1:10], main="Relative Variance", names.arg = paste("PC",
1:10))
barplot (cumsum (100* data.relvars [1:10]), main="Cumulative variance (%)", names.
arg = paste("PC", 1:10), ylim = c(0,100))

npc = 2 # number of PCs chosen to run the PCA

scores = data.scores [1:dim_data[1],1:npc] # PCA scores up to the chosen number
of PC
loadings = data.loadings [1:dim_data[2], 1:npc] # PCA loadings up to the chosen
PC number

# plotting PCA scores PC1 x PC2

col =
c("blue", "blue", "blue", "blue", "blue", "blue", "blue", "blue", "red", "red", "red", "red
", "red", "red", "red", "red", "red")
dev.new ( )
plot(data.scores[,1],data.
scores[,2],pch=19,col=col,xlab='PC1',ylab='PC2',main='PCA scores - blue =
creatinine , red = albumin ')

```

```

# plotting PCA loadings PC1 & PC2

dev.new ( )
matplot(data.loadings
[,1],type="l",col="blue",xlab="Pixels",ylab='Loadings',main='PCA loadings')
lines(data.loadings
[,2],type="l",col="red",xlab="Pixels",ylab='Loadings',main='PCA loadings')

##### HCA model #####
#####

clusters <- hclust(dist(data_scal),method ="average")

# viewing dendrogram

plot(clusters,xlab="Samples (1-8: creatinine, 9-16: albumin)", ylab ="Distance",
main =" HCA dendrogram, numbers = sample index ")

# samples are grouped into 2 clusters - high concentration (on the left) and low
concentration (on the right)

##### K-Means model #####
#####

set.seed (123)
km.res <- kmeans(data_scal,2,nstart =1)

# viewing results

kmeans_out = km.res$cluster

plot(kmeans_out,pch =19,xlab="Samples (1-8: creatinine , 9-16: albumin)", ylab ="
Response K- Means",main ="K-Means")
points(c( 1,1,1,1,1,1,1,1,2,2,2,2,2,2,2))

# Classification rate to differentiate albumin x creatinine

ac = sum(kmeans_out==group12,na.rm=T)/ nrow(group12) * 100
ac

# Classification rate to differentiate low and high concentrations

groupConc = c( 2,2,2,1,1,1,1,1,2,2,2,1,1,1,1)
ac = sum(kmeans_out==groupConc,na.rm=T)/ nrow (group12) * 100
ac

# K-Means is distinguishing lows vs. high concentrations

```

6.4 Multivariate classification

Example 3: The example we will describe here consists of multivariate classification using three algorithms (PCA-LDA, SPA-LDA and GA-LDA) on RGB images obtained from ELISA plates containing albumin and creatinine concentrations using the *imager*, *ggplot2*, *dplyr*, *prospectr*, *MASS*, *plot3D*, *plotly* and *factoextra* packages.

Script

```
## Loading packages

install.packages("imager")
library(imager)
library(ggplot2)
library(dplyr)
library(prospectr)
library(MASS)

install.packages("plot3D")
library(plot3D)

install.packages("plotly")
library(plotly)

install.packages("factoextra")
library(factoextra)

install.packages("lintools")
library(lintools)

install.packages("caret")
install.packages("GA")

library(caret)
library(GA)

# establishing the working directory

# Navigate to the directory containing the images for testing
# In RStudio, go in Session > Set Working Directory > Choose Directory...

## loading images - creatinine

im1 = load.image ("creatinine/im1.png")
im2 = load.image ("creatinine/im2.png")
im3 = load.image ("creatinine/im3.png")
im4 = load.image ("creatinine/im4.png")
im5 = load.image ("creatinine/im5.png")
im6 = load.image ("creatinine/im6.png")
im7 = load.image ("creatinine/im7.png")
im8 = load.image ("creatinine/im8.png")
```

```

## extracting all image pixels to a vector

im1v <- matrix (im1)
im2v <- matrix (im2)
im3v <- matrix (im3)
im4v <- matrix (im4)
im5v <- matrix (im5)
im6v <- matrix (im6)
im7v <- matrix (im7)
im8v <- matrix (im8)

## joining all images

creatinine_data =
rbind(t(im1v),t(im2v),t(im3v),t(im4v),t(im5v),t(im6v),t(im7v),t(im8v))

## Loading albumin images

im1 = load.image ("albumin/im1.png")
im2 = load.image ("albumin/im2.png")
im3 = load.image ("albumin/im3.png")
im4 = load.image ("albumin/im4.png")
im5 = load.image ("albumin/im5.png")
im6 = load.image ("albumin/im6.png")
im7 = load.image ("albumin/im7.png")
im8 = load.image ("albumin/im8.png")

## extracting all image pixels to a vector

im1v <- matrix (im1)
im2v <- matrix (im2)
im3v <- matrix (im3)
im4v <- matrix (im4)
im5v <- matrix (im5)
im6v <- matrix (im6)
im7v <- matrix (im7)
im8v <- matrix (im8)

## joining all albumin images

albumin_data =
rbind(t(im1v),t(im2v),t(im3v),t(im4v),t(im5v),t(im6v),t(im7v),t(im8v))

## joining the images of creatinine and albumin

data = rbind ( creatinine_data , albumin_data )

```

```

group1 = rep(1,8) # class 1 - creatinine
group2 = rep(2,8) # class 2 - albumin
group12 = rbind ( matrix (group1), matrix (group2))

dim_data = dim (data)
dim_class1 = 8
dim_class2 = 8

##### PCA-LDA #####

# scaling the data

data_scal = scale (data, center=TRUE, scale =FALSE)
dim_data = dim(data) # dimension of the data array

# PCA Model

data.svd = svd ( data_scal ) # SVD
data.scores = data.svd$u %*% diag ( data.svd$d ) # scores
data.loadings = data.svd$v # loadings

# PCA Variances

data.vars = data.svd$d^2 / ( nrow (data)-1) # variance per PC
data.totalvar = sum( data.vars ) # total variance
data.relvars = data.vars / data.totalvar # cumulative variance
variances = 100 * round( data.relvars , digits = 3) # cumulative variance in %
variances [ 1:10] # variance in % in the first 10 PCs

# Choose the number of PCs

par(mfrow = c( 2,2))
barplot (data.vars [1:10], main="Variance", names.arg = paste("PC", 1:10))
barplot (log( data.vars [1:10]), main="Log( variance )", names.arg = paste("PC",
1:10))
barplot (data.relvars [1:10], main="Relative variance", names.arg = paste("PC",
1:10))
barplot (cumsum (100* data.relvars [1:10]), main="Cumulative variance (%)", names.
arg = paste("PC", 1:10), ylim = c(0,100))

npc = 2 # number of PCs chosen to run the PCA

scores = data.scores [1:dim_data[1],1:npc] # PCA scores up to the chosen number
of PC
loadings = data.loadings [1:dim_data[2], 1:npc] # PCA loadings up to the chosen
PC number

```

```

# selection of training and testing samples based on KS

perc = 0.7 # 70% for training and 30% for testing

ntrain1 = ceiling (perc * dim_class1[1]) # number of training samples class 1
ntrain2 = ceiling (perc * dim_class2[1]) # number of training samples class 2

scores1 = scores[1:8,1:npc] # scores class 1
scores2 = scores[(8+1): dim_data[1],1:npc] # scores class 2

sel1 = kenStone (scores1, k = ntrain1) # KS class 1
sel2 = kenStone (scores2, k = ntrain2) # KS class 2

train1 = scores1[sel1$model,1:npc] # training class 1
train2 = scores2[sel2$model,1:npc] # training class 2
train = rbind (train1,train2) # joining training matrices

group1train = matrix (group1[sel1$model]) # class 1 training labels
group2train = matrix (group2[sel2$model]) # class 2 training labels
group_train = rbind (group1train,group2train ) # training labels

test1 = scores1[sel1$test, 1:npc] # test class 1
test2 = scores2[sel2$test, 1:npc] # test class 2
test = rbind (test1,test2) # joining test matrices

group1test = matrix (group1[sel1$test]) # class 1 test labels
group2test = matrix (group2[sel2$test]) # class 2 test labels
group_test = rbind (group1test,group2test ) # test labels

# LDA Model

model_lda = lda ( train,group_train ) # model without cross-validation
model_lda_cv = lda ( train,group_train , CV=TRUE) # model with cross-validation
leave-one-out

# prediction of training and testing samples

pred_train = predict( model_lda, train ) # training
pred_test = predict( model_lda, test ) # test

# figures of merit

# training

dim_train1 = dim(train1)
dim_train2 = dim(train2)

ac_train = mean(pred_train$class == group_train ) # accuracy
spec_train = mean(pred_train$class[1:dim_train1[1]]==group1train) # specificity
sens_train = mean(pred_train$class[(dim_train1[ 1]+ 1):(dim_train1[1]+dim_
train2[1]]==group2train) # sensitivity

```

```

# cross-validation

ac_cv = mean (model_lda_cv$class == group_train ) # accuracy
spec_cv = mean (model_lda_cv$class[1:dim_train1[1]]==group1train) # specificity
sens_cv = mean(model_lda_cv$class[(dim_train1[1]+ 1):(dim_train1[1]+dim_
train2[1]])==group2train) # sensitivity

# test

dim_test1 = dim(test1)
dim_test2 = dim(test2)

ac_test = mean(pred_test$class == group_test ) # accuracy
spec_test = mean(pred_test$class[1:dim_test1[1]]==group1test) # specificity
sens_test = mean(pred_test$class[(dim_test1[1]+ 1):(dim_test1[1]+dim_
test2[1]])==group2test) # sensitivity

print( "===== Training =====")
cat ("Accuracy:")
ac_train
cat ("Sensitivity:")
sens_train
cat ("Specificity:")
spec_train

print( "===== Cross-validation =====")
cat ("Accuracy:")
ac_cv
cat ("Sensitivity:")
sens_cv
cat ("Specificity:")
spec_cv

print( "===== Test =====")
cat("Accuracy:")
ac_test
cat ("Sensitivity:")
sens_test
cat ("Specificity:")
spec_test

# plotting PCA scores PC1 x PC2

col =
c("blue","blue","blue","blue","blue","blue","blue","blue","red","red","red","red",
"red","red","red","red","red")
dev.new ( )
plot(data.scores[,1],data.
scores[,2],pch=19,col=col,xlab='PC1',ylab='PC2',main='PCA scores - blue =
creatinine, red = albumin ')

```



```

# plotting PCA loadings PC1 & PC2

dev.new ( )
matplot(data.
loadings[,1],type="l",col="blue",xlab="Pixels",ylab='Loadings',main='PCA
loadings')
lines(data.loadings[,2],type="l",col="red",xlab="Pixels",ylab='Loadings',main='PCA
loadings')

# viewing posterior probabilities - training

col =
c("blue","blue","blue","blue","blue","blue","red","red","red","red","red","red",
"red")
dev.new()
plot(pred_train$posterior,pch =19,col= col,xlab ="LD1", ylab ="LD2", main="
Training - blue = creatinine , red = albumin ")

# visualizing posterior probabilities - cross-validation

col =
c("blue","blue","blue","blue","blue","blue","red","red","red","red","red","red",
"red")
dev.new ( )
plot(model_lda_cv$posterior,pch =19,col= col,xlab ="LD1", ylab ="LD2", main=
"Cross Validation - blue = creatinine, red = albumin")

# viewing posterior probabilities - test

col = c("blue","blue","red","red")
dev.new ( )
plot(pred_test$posterior,pch=19,col= col,xlab ="LD1", ylab ="LD2", main="Test -
blue = creatinine , red = albumin ")

##### SPA-LDA #####

# SPA model

nvar = 8 # number of variables to select

datam = data.matrix(data, rownames.force=NA) # converting data to matrix

m = colMeans (data) # average of the spectra

spa_model = project (x= data.loadings [,1], A= datam , b=group12, neq =0) # spa
model

```

```

x = abs ( model_spa$x ) # leaving positive values for the SPA response vector

temp = sort.int( x, decreasing=TRUE, index.return =TRUE)
variables = temp$ix [1:nvar] # identifying selected variables

dev.new ( ) # plot of selected variables
matplot( m,xlab="Pixels",type="l",ylab="Intensity",main="Average spectrum with
selected variables")
points( variables,m [ variables ], pch =19,col=" red ")

datam_spa = datam [, variables ] # absorbances for the selected variables

# selection of training and testing samples based on KS

perc = 0.7 # 70% for training and 30% for testing

ntrain1 = ceiling ( perc * dim_class1[1]) # number of training samples class 1
ntrain2 = ceiling ( perc * dim_class2[1]) # number of training samples class 2

datam_spa1 = datam_spa [1:dim_class1[1],] # scores class 1
datam_spa2 = datam_spa [(dim_class1[1]+ 1): dim_data[1],] # scores class 2

sel1 = kenStone ( datam_spa1, k = ntrain1) # KS class 1
sel2 = kenStone ( datam_spa2, k = ntrain2) # KS class 2

train1 = datam_spa1[sel1$model,] # training class 1
train2 = datam_spa2[sel2$model,] # training class 2
train = rbind (train1,train2) # joining training matrices

group1train = matrix (group1[sel1$model]) # class 1 training labels
group2train = matrix (group2[sel2$model]) # class 2 training labels
group_train = rbind (group1train,group2train ) # training labels

test1 = datam_spa1[sel1$test,] # test class 1
test2 = datam_spa2[sel2$test,] # test class 2
test = rbind (test1,test2) # joining test matrices

group1test = matrix (group1[sel1$test]) # class 1 test labels
group2test = matrix (group2[sel2$test]) # class 2 test labels
group_test = rbind (group1test,group2test ) # test labels

# LDA Model

model_lda = lda ( train,group_train ) # model without cross-validation
model_lda_cv = lda ( train,group_train , CV=TRUE) # model with cross-validation
leave-one-out

# prediction of training and testing samples

```

```

pred_train = predict( model_lda, train ) # training
pred_test = predict( model_lda, test ) # test

# figures of merit

# training

dim_train1 = dim(train1)
dim_train2 = dim(train2)

ac_train = mean(pred_train$class == group_train ) # accuracy
spec_train = mean(pred_train$class[1:dim_train1[1]]==group1train) # specificity
sens_train = mean(pred_train$class[(dim_train1[ 1]+ 1):(dim_train1[1]+dim_
train2[1]])==group2train) # sensitivity

# cross-validation

ac_cv = mean (model_lda_cv$class == group_train ) # accuracy
spec_cv = mean (model_lda_cv$class[1:dim_train1[1]]==group1train) # specificity
sens_cv = mean(model_lda_cv$class[(dim_train1[1]+ 1):(dim_train1[1]+dim_
train2[1]])==group2train) # sensitivity

# test

dim_test1 = dim(test1)
dim_test2 = dim(test2)

ac_test = mean(pred_test$class == group_test ) # accuracy
spec_test = mean(pred_test$class[1:dim_test1[1]]==group1test) # specificity
sens_test = mean(pred_test$class[(dim_test1[1]+ 1):(dim_test1[1]+dim_
test2[1]])==group2test) # sensitivity

print( "==== Training =====")
cat ("Accuracy:")
ac_train
cat ("Sensitivity:")
sens_train
cat ("Specificity:")
spec_train

print( "==== Cross-validation =====")
cat ("Accuracy:")
ac_cv
cat ("Sensitivity:")
sens_cv
cat ("Specificity:")
spec_cv

```

```

print( "===== Test =====")
cat("Accuracy:")
ac_test
cat ("Sensitivity:")
sens_test
cat ("Specificity:")
spec_test

# viewing posterior probabilities - training

col =
c("blue","blue","blue","blue","blue","blue","red","red","red","red","red","red",
"red")
dev.new()
plot(pred_train$posterior,pch =19,col= col,xlab ="LD1", ylab ="LD2", main="
Training - blue = creatinine , red = albumin ")

# visualizing posterior probabilities - cross-validation

col =
c("blue","blue","blue","blue","blue","blue","red","red","red","red","red","red",
"red")
dev.new ( )
plot(model_lda_cv$posterior,pch =19,col= col,xlab ="LD1", ylab ="LD2", main=
"Cross Validation - blue = creatinine, red = albumin")

# viewing posterior probabilities - test

col = c("blue","blue","red","red")
dev.new ( )
plot(pred_test$posterior,pch=19,col= col,xlab ="LD1", ylab ="LD2", main="Test -
blue = creatinine , red = albumin ")

##### GA-LDA #####

# GA algorithm

datam = data.matrix (data, rownames.force =NA) # converting data to matrix
datam_m = colMeans ( datam )
ind = datam_m != 1
datam_ga = datam[, ind ==TRUE]
datam = datam_ga

# Function to Establish Population

myInit <- function(k){

```

```

function(GA){
  m <- matrix(0, ncol = GA@nBits, nrow = GA@popSize)

  for(i in seq_len(GA@popSize))
    m[i, sample(GA@nBits, k)] <- 1

  m
}

# Function for Crossover

myCrossover <- function(GA, parents){

  parents <- GA@population[parents,] %>%
    apply(1, function(x) which(x == 1)) %>%
    t()

  parents_diff <- list("vector", 2)
  parents_diff[[1]] <- setdiff(parents[2,], parents[1,])
  parents_diff[[2]] <- setdiff(parents[1,], parents[2,])

  children_ind <- list("vector", 2)
  for(i in 1:2){
    k <- length(parents_diff[[i]])
    change_k <- sample(k, sample(ceiling(k/2), 1))
    children_ind[[i]] <- if(length(change_k) > 0){
      c(parents[i, -change_k], parents_diff[[i]][change_k])
    } else {
      parents[i,]
    }
  }
}

children <- matrix(0, nrow = 2, ncol = GA@nBits)
for(i in 1:2)
  children[i, children_ind[[i]]] <- 1

list(children = children, fitness = c(NA, NA))
}

# Mutation Function

myMutation <- function(GA, parent){

  ind <- which(GA@population[parent,] == 1)
  n_change <- sample(3, 1)
  ind[sample(length(ind), n_change)] <- sample(setdiff(seq_len(GA@nBits), ind),
n_change)
  parent <- integer(GA@nBits)
  parent[ind] <- 1
}

```

```

parent
}

# Adjustment Function

f <- function(x, values){

  ind <- which(x == 1)
  y <- values[ind]
  y <- ifelse(y %% 2 != 0, y, 0)
  y <- y[1:10]
  return(sum(y))
}

# GA Model

GA_model = ga(type="binary", fitness=f, values= datam, nBits = ncol (datam),
population= myInit ( nrow ( datam )), crossover = myCrossover , mutation= myMutation
, run=200, pmutation =0.1 , maxiter =1000, popSize = nrow ( datam ))

# selected variables

ind = which ( GA_model@solution[1,] == 1)
if ( length ( ind ) > 8){
  indmax = 8 # maximum number of variables selected
  ind = ind [1:indmax]
}

# array with selected variables

datam_ga = datam [, ind ]
m = colMeans ( datam )
variables = ind

dev.new ( ) # plot of selected variables
matplot( m,xlab="Pixels",type="l",ylab="Intensity",main="Average spectrum with
selected variables")
points( variables,m [ variables ], pch =19,col="red")

# selection of training and testing samples based on KS

perc = 0.7 # 70% for training and 30% for testing

ntrain1 = ceiling ( perc * dim_class1[1]) # number of training samples class 1
ntrain2 = ceiling ( perc * dim_class2[1]) # number of training samples class 2

datam_ga1 = datam_ga [1:dim_class1[1],] # scores class 1
datam_ga2 = datam_ga [(dim_class1[1]+ 1): dim_data[1],] # scores class 2

sel1 = kenStone ( datam_ga1, k = ntrain1) # KS class 1
sel2 = kenStone ( datam_ga2, k = ntrain2) # KS class 2

```

```

train1 = datam_ga1[sel1$model,] # training class 1
train2 = datam_ga2[sel2$model,] # training class 2
train = rbind (train1,train2) # joining training matrices

group1train = matrix (group1[sel1$model]) # class 1 training labels
group2train = matrix (group2[sel2$model]) # class 2 training labels
group_train = rbind (group1train,group2train ) # training labels

test1 = datam_ga1[sel1$test,] # test class 1
test2 = datam_ga2[sel2$test,] # test class 2
test = rbind (test1,test2) # joining test matrices

group1test = matrix (group1[sel1$test]) # class 1 test labels
group2test = matrix (group2[sel2$test]) # class 2 test labels
group_test = rbind (group1test,group2test ) # test labels

# LDA Model

model_lda = lda ( train,group_train ) # model without cross-validation
model_lda_cv = lda ( train,group_train , CV=TRUE) # model with cross-validation
leave-one-out

# prediction of training and testing samples

pred_train = predict( model_lda, train ) # training
pred_test = predict( model_lda, test ) # test

# figures of merit

# training

dim_train1 = dim(train1)
dim_train2 = dim(train2)

ac_train = mean(pred_train$class == group_train ) # accuracy
spec_train = mean(pred_train$class[1:dim_train1[1]]==group1train) # specificity
sens_train = mean(pred_train$class[(dim_train1[ 1]+ 1):(dim_train1[1]+dim_
train2[1])]==group2train) # sensitivity

# cross-validation

ac_cv = mean (model_lda_cv$class == group_train ) # accuracy
spec_cv = mean (model_lda_cv$class[1:dim_train1[1]]==group1train) # specificity
sens_cv = mean(model_lda_cv$class[(dim_train1[1]+ 1):(dim_train1[1]+dim_
train2[1])]==group2train) # sensitivity

# test

```

```

dim_test1 = dim(test1)
dim_test2 = dim(test2)

ac_test = mean(pred_test$class == group_test ) # accuracy
spec_test = mean(pred_test$class[1:dim_test1[1]]==group1test) # specificity
sens_test = mean(pred_test$class[(dim_test1[1]+ 1):(dim_test1[1]+dim_
test2[1]])==group2test) # sensitivity

print( "===== Training =====")
cat ("Accuracy:")
ac_train
cat ("Sensitivity:")
sens_train
cat ("Specificity:")
spec_train

print( "===== Cross-validation =====")
cat ("Accuracy:")
ac_cv
cat ("Sensitivity:")
sens_cv
cat ("Specificity:")
spec_cv

print( "===== Test =====")
cat("Accuracy:")
ac_test
cat ("Sensitivity:")
sens_test
cat ("Specificity:")
spec_test

# viewing posterior probabilities - training

col =
c("blue", "blue", "blue", "blue", "blue", "blue", "red", "red", "red", "red", "red", "red",
"red")
dev.new()
plot(pred_train$posterior,pch =19,col= col,xlab ="LD1", ylab ="LD2", main="
Training - blue = creatinine , red = albumin ")

# visualizing posterior probabilities - cross-validation

col =
c("blue", "blue", "blue", "blue", "blue", "blue", "red", "red", "red", "red", "red", "red",
"red")
dev.new ( )
plot(model_lda_cv$posterior,pch =19,col= col,xlab ="LD1", ylab ="LD2", main
="Cross Validation - blue = creatinine, red = albumin")

```



```
# viewing posterior probabilities - test

col = c("blue","blue","red","red")
dev.new ( )
plot(pred_test$posterior,pch=19,col= col,xlab = "LD1", ylab = "LD2", main="Test -
blue = creatinine , red = albumin ")
```

6.5 Multivariate Regression

Example 4: The example we will describe here consists of multivariate regression (MLR, PCR and PLS) of RGB images obtained from ELISA plates containing albumin and creatinine concentrations using the *imager*, *ggplot2*, *dplyr*, *prospectr*, *MASS*, *plot3D*, *plotly*, *factoextra*, *Stat2Data*, *Metrics* and *PLS* packages.

Script

```
# Loading Packages

install.packages("imager")
library(imager)
library(ggplot2)
library(dplyr)
library(prospectr)
library(MASS)

install.packages("plot3D")
library(plot3D)

install.packages("plotly")
library(plotly)

install.packages("factoextra")
library(factoextra)

install.packages("Stat2Data")
library(Stat2Data)

install.packages("Metrics")
library(Metrics)

install.packages("pls")
library(pls)

# establishing the working directory

# Navigate to the directory containing the images for testing
# In RStudio, go in Session > Set Working Directory > Choose Directory...
```

```

## loading images

im1 = load.image ("creatinine/im1.png")
im2 = load.image ("creatinine/im2.png")
im3 = load.image ("creatinine/im3.png")
im4 = load.image ("creatinine/im4.png")
im5 = load.image ("creatinine/im5.png")
im6 = load.image ("creatinine/im6.png")
im7 = load.image ("creatinine/im7.png")
im8 = load.image ("creatinine/im8.png")

## extracting all image pixels to a vector

im1v <- matrix (im1)
im2v <- matrix (im2)
im3v <- matrix (im3)
im4v <- matrix (im4)
im5v <- matrix (im5)
im6v <- matrix (im6)
im7v <- matrix (im7)
im8v <- matrix (im8)

## joining all images

data = rbind(t(im1v),t(im2v),t(im3v),t(im4v),t(im5v),t(im6v),t(im7v),t(im8v))
dim_data = dim(data)

# concentration vector in mg/ dL

y <- c(0.03125,0.0625,0.125,0.25,0.5,1,2,4)

##### MLR Model #####

# RGB intensities

im1_RGB = cbind (mean(mean(R(im1))),mean(mean(G(im1))),mean(mean(B(im1))))
im2_RGB = cbind (mean(mean(R(im2))),mean(mean(G(im2))),mean(mean(B(im2))))
im3_RGB = cbind (mean(mean(R(im3))),mean(mean(G(im3))),mean(mean(B(im3))))
im4_RGB = cbind (mean(mean(R(im4))),mean(mean(G(im4))),mean(mean(B(im4))))
im5_RGB = cbind (mean(mean(R(im5))),mean(mean(G(im5))),mean(mean(B(im5))))
im6_RGB = cbind (mean(mean(R(im6))),mean(mean(G(im6))),mean(mean(B(im6))))
im7_RGB = cbind (mean(mean(R(im7))),mean(mean(G(im7))),mean(mean(B(im7))))
im8_RGB = cbind (mean(mean(R(im8))),mean(mean(G(im8))),mean(mean(B(im8))))

# RGB absorbances

```

```

im1abs = -log10(im1_RGB/1)
im2abs = -log10(im2_RGB/1)
im3abs = -log10(im3_RGB/1)
im4abs = -log10(im4_RGB/1)
im5abs = -log10(im5_RGB/1)
im6abs = -log10(im6_RGB/1)
im7abs = -log10(im7_RGB/1)
im8abs = -log10(im8_RGB/1)

x = rbind (im1abs,im2abs,im3abs,im4abs,im5abs,im6abs,im7abs,im8abs)
dim_x = dim (x)

## Division into Calibration and Prediction

perc = 0.7 # 70% for calibration and 30% for testing

size = 1:dim_x [ 1]

ntrain = ceiling ( perc * dim_x [1]) # number of calibration samples

sel = sample( size, ntrain ) # sample selection

xcal = x[ sel ,] # calibration
ycal = matrix (y[ sel ]) # concentration calibration

xpred = x[- sel ,] # prediction
ypred = matrix (y[- sel ]) # concentration prediction

## MLR Model

xcal_df = data.frame ( xcal )
xpred_df = data.frame ( xpred )

model_mlr = lm ( ycal ~ xcal , xcal_df ) # MLR model

coef = model_mlr$coefficients # regression coefficients

ycal_calc = xcal %%% coef [2:4] + coef [1] # predicted concentration calibration
ypred_calc = xpred %%% coef [2:4] + coef [1] # predicted concentration prediction

## Plot measured concentration vs. predicted

dev.new ( )
plot (ycal,ycal_calc,pch='o',col ="blue", xlab ="Measured concentration (mg/dL)",
ylab ="Predicted concentration (mg/dL)", main ="Calibration")
lines(ycal,ycal,col="red")

```

```

dev.new ( )
plot (ypred,ypred_calc,pch=15,col="red", xlab = "Measured concentration (mg/dL)",
ylab = "Predicted concentration (mg/dL)", main = "Prediction")
lines(ypred,ypred ,col ="red")

dev.new ( )
plot (ycal,ycal_calc,pch='o', col="blue", xlab ="Measured concentration (mg/dL)",
ylab ="Predicted concentration (mg/dL)", main ="Calibration (blue) and Prediction
(red)")
points(ypred,ypred_calc,pch =15,col="red")
lines(y,y ,col="red")

## Figures of merit

# Calibration

MAPEC = mean (abs((ycal-ycal_calc)/ycal))*100
R2cal = cor (ycal,ycal_calc)^2
RMSEC = rmse (ycal,ycal_calc)

# Prediction

MAPEP = mean(abs((ypred-ypred_calc )/ypred))* 100
R2pred = cor(ypred,ypred_calc)^2
RMSEP = rmse(ypred,ypred_calc)

print("===== Calibration =====")
print("Mean Absolute Error Percentage (MAPE) (%):")
MAPEC
print("R2cal:")
R2cal
print("RMSEC (mg/dL):")
RMSEC

print("===== Test =====")
print("Mean Absolute Error Percentage (MAPE) (%):")
MAPEP
print("R2pred:")
R2pred
print("RMSEP (mg/dL):")
RMSEP

##### PCR Model #####

# scaling the data

data_scal = scale(data, center=TRUE, scale =FALSE)
dim_data = dim (data) # dimension of the data array

```

```

# PCA Model

data.svd = svd ( data_scal ) # SVD
data.scores = data.svd$u %*% diag ( data.svd$d ) # scores
data.loadings = data.svd$v # loadings

# PCA Variances

data.vars = data.svd$d^2 / (nrow (data)-1) # variance per PC
data.totalvar = sum( data.vars ) # total variance
data.relvars = data.vars / data.totalvar # cumulative variance
variances = 100 * round( data.relvars , digits = 3) # cumulative variance in %
variances [1:10] # variance in % in the first 10 PCs

# Choose the number of PCs

par(mfrow = c( 2,2))
barplot (data.vars [1:10], main="Variance", names.arg = paste("PC", 1:10))
barplot (log( data.vars [1:10]), main="Log( variance )", names.arg = paste("PC",
1:10))
barplot (data.relvars [1:10], main="Relative variance", names.arg = paste("PC",
1:10))
barplot (cumsum (100* data.relvars [1:10]), main="Cumulative variance (%)", names.
arg = paste("PC", 1:10), ylim = c(0,100))

npc = 3 # number of PCs chosen to run the PCA

scores = data.scores[1:dim_data[1],1:npc] # PCA scores up to the chosen number of PC
loadings = data.loadings[1:dim_data[2], 1:npc] # PCA loadings up to the chosen PC
number

# Linear Regression on PCA Scores

x = scores
dim_x = dim (x)

## Division into Calibration and Prediction

perc = 0.7 # 70% for calibration and 30% for testing

size = 1:dim_x[1]

ntrain = ceiling( perc * dim_x [1]) # number of calibration samples

sel = sample( size, ntrain ) # sample selection

xcal = x[ sel ,] # calibration
ycal = matrix (y[ sel ]) # concentration calibration

```

```

xpred = x[- sel ,] # prediction
ypred = matrix (y[- sel ]) # concentration prediction

## PCR model

xcal_df = data.frame(xcal)
xpred_df = data.frame(xpred)

modelo_mlr = lm(ycal ~ xcal, xcal_df) # MLR model

coef = modelo_mlr$coefficients # regression coefficients

ycal_calc = xcal %% coef[2:4] + coef[1] # predicted concentration - calibration
ypred_calc = xpred %% coef[2:4] + coef[1] # predicted concentration - prediction

## Plot measured concentration vs. Prediction

dev.new ( )
plot (ycal,ycal_calc,pch ='o', col ="blue", xlab ="Measured concentration (mg/dL)",
ylab ="Predicted concentration (mg/dL)", main ="Calibration")
lines(ycal,ycal,col="red")

dev.new ( )
plot (ypred,ypred_calc,pch =15,col="red", xlab = "Measured concentration (mg/dL)",
ylab = "Predicted concentration (mg/dL)", main = "Prediction")
lines(ypred,ypred,col ="red")

dev.new ( )
plot (ycal,ycal_calc,pch ='o', col ="blue", xlab ="Measured concentration (mg/dL)",
ylab ="Predicted concentration (mg/dL)", main ="Calibration (blue) and Prediction
(red)")
points(ypred,ypred_calc,pch =15,col="red")
lines(y,y,col ="red")

## Figures of merit

# Calibration

MAPEC = mean (abs((ycal-ycal_calc)/ycal))*100
R2cal = cor (ycal,ycal_calc)^2
RMSEC = rmse (ycal,ycal_calc)

# Prediction

MAPEP = mean(abs((ypred-ypred_calc )/ypred))* 100
R2pred = cor(ypred,ypred_calc)^2
RMSEP = rmse(ypred,ypred_calc)

```

```

print("==== Calibration =====")
print("Mean Absolute Error Percentage (MAPE) (%):")
MAPEC
print("R2cal:")
R2cal
print("RMSEC (mg/dL):")
RMSEC

print("==== Test =====")
print("Mean Absolute Error Percentage (MAPE) (%):")
MAPEP
print("R2pred:")
R2pred
print("RMSEP (mg/dL):")
RMSEP

##### PLS Model #####

## Division into Calibration and Prediction

mydim = dim (data)
Xr = data
perc = 0.7 # 70% for calibration and 30% for testing

dim_x = dim (x)
size = 1:dim_x[1]

ntrain = ceiling ( perc * dim_x [1]) # number of calibration samples

sel = sample(size, ntrain) # KS

xcal = Xr [ sel , ] # calibration
ycal = matrix (y[ sel ]) # concentration-calibration

xpred = Xr [- sel , ] # prediction
ypred = matrix (y[- sel ]) # concentration-prediction

## PLS Model

xcal_df = data.frame ( xcal )
xpred_df = data.frame ( xpred )

model_pls = pls ( ycal ~ xcal , data = xcal_df )

## Determine number of components

validationplot ( model_pls )
validationplot ( model_pls, val.type = "MSEP")
validationplot ( model_pls, val.type = "R2")

```

```

ncomp = 2 # number of selected components

## prediction

ycal_calc <- predict( model_pls, xcal, ncomp = ncomp )
ypred_calc <- predict( model_pls, xpred, ncomp = ncomp )

ycal_calc = ycal_calc [,1,1]
ypred_calc = ypred_calc [,1,1]

## Plot measured concentration vs. prediction

dev.new ( )
plot (ycal,ycal_calc,pch='o', col="blue", xlab="Measured concentration (mg/dL)",
ylab="Predicted concentration (mg/dL)", main="Calibration")
lines(ycal,ycal,col="red")

dev.new ( )
plot (ypred,ypred_calc,pch=15,col="red", xlab="Measured concentration (mg/dL)",
ylab="Predicted concentration (mg/dL)", main="Prediction")
lines(ypred,ypred,col="red")

dev.new ( )
plot (ycal,ycal_calc,pch='o', col="blue", xlab="Measured concentration (mg/dL)",
ylab="Predicted concentration (mg/dL)", main="Calibration (blue) and Prediction
(red)")
points(ypred,ypred_calc,pch=15,col="red")
lines(y,y,col="red")

## Figures of merit

# Calibration

MAPEC = mean (abs((ycal-ycal_calc)/ycal))*100
R2cal = cor (ycal,ycal_calc)^2
RMSEC = rmse (ycal,ycal_calc)

# Prediction

MAPEP = mean(abs((ypred-ypred_calc )/ypred))* 100
R2pred = cor(ypred,ypred_calc)^2
RMSEP = rmse(ypred,ypred_calc)

print("=====  

print("Mean Absolute Error Percentage (MAPE) (%):")
MAPEC
print("R2cal:")
R2cal
print("RMSEC (mg/dL):")
RMSEC

```



```
print("=====  
Test =====")  
print("Mean Absolute Error Percentage (MAPE) (%):")  
MAPEP  
print("R2pred:")  
R2pred  
print("RMSEP (mg/dL):")  
RMSEP
```

PROPOSED EXERCISES

01 – Using a digital image dataset available in repositories or through images collected with some equipment (cell phone, scanner), perform an exploratory analysis using the main algorithms in the R language (PCA, HCA and K- means). Present your results and main conclusions.

02 – Using a digital image dataset available in repositories or through images collected with some equipment (cell phone, scanner), perform a multivariate classification using algorithms in the R language (PCA-LDA, SPA-LDA and GA-LDA). Present your results and main conclusions.

03 – From the previous exercise, write multivariate classification scripts in R (PCA-QDA, SPA-QDA and GA-QDA) for a given dataset (simulated or experimental). Finally, present your results and conclusions when compared to the multivariate classification models used in the LDA function.

04 – Using a digital image dataset available in repositories or through images collected with some equipment (cell phone, scanner), perform a multivariate regression using algorithms in the R language (MLR, PCR and PLS). Present your results and main conclusions.

REFERENCES

- 1 – Solomon , C.; Breckon , T. (2011). Fundamentals of Digital Image Processing – A Practical Approach with Examples in Matlab . 1st Edition. USA, John Wiley & Sons Ltd.
- 2 – Plataniotis , KN; Venetsanopoulos, A. N. (2000). Color Image Processing and Applications. Berlin; Heidelberg; New York; Barcelona; Hong Kong; London; Milano; Paris; Singapore; Tokyo: Springer.
- 3 – Gonzalez, RC; Woods, RE Digital image processing. 1st Ed. São Paulo: Editora Blucher , 2000.

APPENDIX A



A1 – LOADING SPECTRAL DATA USING R

Example A1: Loading raw data (.csv format) of near-infrared (NIR) spectroscopy in R software using the *data.table* package. Data extracted from: <https://doi.org/10.1016/j.dib.2020.106647>

R Script

```
install.packages("data.table")
library (data.table)

## loading data .csv

data = fread ("dataset/spectra_standard_cells.csv")

## visualize data

View (data)

## extracting the spectra

d = dim (data) # data dimensions

nm = data[1,4:d[2]] # wavelength = 1st row of table from column 4

x = data[2:16,4:d[2]] # spectrum = rows 2-16 and columns 4 to the end of the table

## plotting spectra

matplot (t(nm),t(x), type ="l", xlab ="Wavelength (nm)", ylab ="Absorbance")
```

A2 – PREPROCESSING SPECTRAL DATA IN R

Example A2: Applying different preprocessing to near-infrared (NIR) spectroscopy data using *data.table*, *pracma*, *pls* and *hyperSpec* packages. Data extracted from: <https://doi.org/10.1016/j.dib.2020.106647>

R Script

```
## packages

library(data.table)

# Smoothing Savitzkt-Golay

install.packages("pracma")
library(pracma)

#MSC

install.packages("pls")
library(pls)

# Baseline correction

install.packages("hyperSpec")
library(hyperSpec)

## loading data. csv

data = fread("dataset/spectra_standard_cells.csv")

## visualize data

View (data)

## extracting the spectra

d = dim (data) # data dimensions

nm = data[1,4:d[2]] # wavelength = 1st row of table from column 4

x = data[2:16,4:d[2]] # spectrum = rows 2-16 and columns 4 to the end of the table

x_dm = data.matrix (x) # converting spectra to numeric format

## plotting spectra

matplot (t(nm), t(x), type ="l", xlab ="Wavelength (nm)", ylab ="Absorbance", main
="Raw data")
```

```

## ===== Pre-processing =====

##### Savitzky-Golay smoothing #####

w = 7 # window size (must be an odd number )
ord = 2 # filter order (example: 2 = 2nd order polynomial)
der = 0 # order of the derivative (0 = no derivative, 1 = 1st derivative, 2 = 2nd
derivative)

# smoothing across all spectra

x_sg = matrix (, nrow = dim (x)[1], ncol = dim (x)[2])
for ( i in 1:dim(x)[1]){
  x_sg [ i ,] = savgol ( x_dm [ i ,], w, ord , der)
}

x_sg [,1:w] = x_dm [,1:w]
x_sg [, (dim(x)[2]-w):dim(x)[2]] = x_dm [, (dim(x)[2]-w):dim(x)[2]]

# plotting pre -processed data

matplot (t( nm ),t( x_sg ), type ="l", xlab ="Wavelength (nm)", ylab ="Absorbance",
main ="Smoothed data - Savitzky-Golay ")

##### smoothing by moving window #####

w = 7 # window size (must be an odd number )

# smoothing function

SmoothFast <-function( Spectra,windowsize ){
  Mat<-matrix(0,length((windowsize+1):(ncol(Spectra)-windowsize)),2*windowsize+1)
  for(j in 1:nrow(Mat)){Mat[j,]<- seq (j,j+2*windowsize,1)}
  newspectra <-matrix(0,nrow(Spectra),
    length((windowsize+1):( ncol (Spectra)- windowsize )))
  for( i in 1:nrow(Mat)){ newspectra [, i ]<-apply(Spectra[,Mat[ i ,]],1,mean)}
  fronttail <- newspectra [,1]
  endtail <- newspectra [, ncol ( newspectra )]
  for(k in 1:(windowsize-1)){fronttail<-data.frame(fronttail,newspectra[,1])
  endtail <- data.frame ( endtail, newspectra [, ncol ( newspectra )]}}
  newspectra <- data.frame ( fronttail,newspectra,endtail )
  return ( newspectra )}

# applying the smoothing function

x_sw = SmoothFast ( x_dm,windowsize =w)

```

```

# plotting preprocessed data

matplot (t( nm ),t( x_sw ), type ="l", xlab ="Wavelength (nm)", ylab ="Absorbance",
main ="Smoothed data - Moving Window")

##### SNV (Standard Normal Variate ) #####

# SNV function

SNV<-function(spectra){
  spectra<- as.matrix (spectra)
  spectrat <-t(spectra)
  spectrat_snv <-scale( spectrat,center = TRUE,scale =TRUE)
  spectra_snv <-t( spectrat_snv )
  return ( spectra_snv )}

# applying SNV

x_snv = SNV( x_dm )

# plotting preprocessed data

matplot (t( nm ),t( x_snv ), type ="l", xlab ="Wavelength (nm)", ylab ="Absorbance",
main ="SNV")

##### MSC (Multiplicative Scatter Correction) #####

# applying MSC

x_msc = msc ( x_dm )

# plotting preprocessed data

matplot (t( nm ),t( x_msc ), type ="l", xlab ="Wavelength (nm)", ylab ="Absorbance",
main ="MSC")

##### Baseline correction #####

# pseudo-image object

x_im = new("hyperSpec", spc = x_dm , wavelength = as.numeric ( nm ))

# applying baseline correction

pord = 2 # order of the baseline adjustment polynomial function

baseline = spc.fit.poly.below (fit.to = x_im , poly.order = pord )

x_base = x_im@data$spc - baseline@data$spc

# plotting preprocessed data

```

```

matplot (t( nm ),t( x_base ), type ="l", xlab ="Wavelength (nm)", ylab ="Absorbance",
main ="Baseline Correction")

##### 1st derivative #####

w = 7 # window size (must be an odd number )
ord = 2 # filter order (example: 2 = 2nd order polynomial)
der = 1 # order of derivative (0 = no derivative, 1 = 1st derivative, 2 = 2nd
derivative)

# SG filter across all spectra

x_1d = matrix (, nrow = dim (x)[1], ncol = dim (x)[2])
for ( i in 1:dim(x)[1]){
  x_1d[ i ,] = savgol ( x_dm [ i ,], w, ord , der)
}

x_1d = x_1d[,w:(dim(x)[2]-w)]
nm_dm = data.matrix (nm)
nm_1d = nm_dm [,w:(dim(x)[2]-w)]

# plotting preprocessed data

matplot (nm_1d,t(x_1d), type ="l", xlab ="Wavelength (nm)", ylab ="Absorbance",
main ="1st derivative Savitzky-Golay ")

##### 2nd derivative #####

w = 7 # window size (must be an odd number )
ord = 2 # filter order (example: 2 = 2nd order polynomial)
der = 2 # order of the derivative (0 = no derivative, 1 = 1st derivative, 2 = 2nd
derivative)

# SG filter across all spectrums

x_2d = matrix (, nrow = dim (x)[1], ncol = dim (x)[2])
for ( i in 1:dim(x)[1]){
  x_2d[ i ,] = savgol ( x_dm [ i ,], w, ord , der)
}

x_2d = x_2d[,w:(dim(x)[2]-w)]
nm_dm = data.matrix (nm)
nm_2d = nm_dm [,w:(dim(x)[2]-w)]

# plotting preprocessed data

matplot (nm_2d,t(x_2d), type ="l", xlab ="Wavelength (nm)", ylab ="Absorbance",
main ="2nd derivative Savitzky-Golay ")

```

A3 – LOADING MOLECULAR FLUORESCENCE DATA: EXCITATION-EMISSION MATRIX (EEM)

Example A3: Loading raw molecular fluorescence data (excitation-emission matrix – EEM). Data extracted from: <https://doi.org/10.3390/data8050081>

R Script

```
## loading data

# sample 1

data <- read.csv( "dataset\\Raw_data\\Aging Step 0\\Fluorescence\\20210512_0752_
A50_Q1K2V1U0.csv",header=FALSE, sep =",")

## visualize data

View (data)

# comments:
# columns V1 to V50 are the excitation wavelengths
# for each excitation wavelength there is an emission wavelength and the associated
intensity
# the emission wavelength values are constant, only the intensity varies

## extracting the excitation-emission matrix

dim_data = dim (data) # data dimension

# excitation wavelength

nm_ex = data[seq (1, dim_data[2], 2)] # extracting each 2nd column of data from
column 1
nm_ex = nm_ex[1,] # extracting only the 1st row

# emission wavelength

nm_em = data[3:253,1] # extracting rows 3 to 253 from the 1st column

# intensities

eem = data[seq(2, dim_data[2], 2)] # extracting each 2nd column of data from column
2
eem = eem [3:253,1:length(eem)] # extracting rows between 3 and 253 to match with
emission
eem = data.matrix(eem) # converting to numeric values

# comments:
# in the matrix, each row corresponds to an emission wavelength and each column to
an excitation wavelength
```

```

## viewing the eem matrix

filled.contour(eem,color.palette = terrain.colors,main ="Sample 1")

## comments:

# each sample must be loaded individually or through a loop
# the eem matrices of each sample can be organized into a 3D tensor using the
command:
# Example

# sample 2

data2 <- read.csv("dataset\\Raw_data\\Aging Step 0\\Fluorescence\\20210512_0915_
AS0_X0S0V2W2.csv",header=FALSE, sep =",")

dim_data2 = dim (data2)

# eem2 - intensities

eem2 = data2[seq(2, dim_data2[2], 2)]# extracting each 2nd column of data from
column 2
eem2 = eem2[3:253,1:length(eem2)] # extracting rows between 3 and 253 to match with
emission
eem2 = data.matrix(eem2) # converting to numeric values

filled.contour(eem2,color.palette = terrain.colors,main = "Sample 2")

# 3D tensor

tensor = array (c(eem,eem2), c(dim(eem)[1], dim(eem2)[2], 2)) # for 2 samples

dim(tensor) # dimensions of the 3D tensor

##### EEM and eemR PACKAGES #####

# EEM data in other formats (other devices) can be loaded using the EEM or eemR
packages

# Examples:

# EEM PACKAGE: https://cran.r-project.org/web/packages/EEM/vignettes/vignette.html

## loading packages

# install.packages ("EEM")
# library (EEM)

```



```
## loading sample

# data = readEEM('filename') # data in .csv or .txt

## viewing the sample

# drawEEM(data, n=1)

# eemR PACKAGE: https://cran.r-project.org/web/packages/eemR/eemR.pdf

## loading packages

# install.packages ("eemR")
# library (eemR)


## loading sample

# data = eem_read ('filename', recursive = FALSE, import_function ="cary") # Cary
equipment data
# data = eem_read ('filename', recursive = FALSE, import_function ="aqualog") #
Aqualog equipment data
# data = eem_read ('filename', recursive = FALSE, import_function ="shimadzu") #
Shimadzu equipment data
# data = eem_read ('filename', recursive = FALSE, import_function = "fluoromax4") #
Fluoromax equipment data

# eemR packages perform pre-processing on loaded data through specific routines - see
tutorials in the links
# EEM: https://cran.r-project.org/web/packages/EEM/vignettes/vignette.html
# eemR: https://cran.r-project.org/web/packages/eemR/eemR.pdf
```

CHEMOMETRICS

A UNIVERSITY COURSE USING
— THE R LANGUAGE —

 www.atenaeditora.com.br

 contato@atenaeditora.com.br

 [@atenaeditora](https://www.instagram.com/atenaeditora)

 www.facebook.com/atenaeditora.com.br

CHEMOMETRICS

A UNIVERSITY COURSE USING
THE **R** LANGUAGE

 www.atenaeditora.com.br

 contato@atenaeditora.com.br

 [@atenaeditora](https://www.instagram.com/atenaeditora)

 www.facebook.com/atenaeditora.com.br