A tutorial on using the rminer R package for data mining tasks*

by **Paulo Cortez**

Teaching Report

Department of Information Systems, ALGORITMI Research Centre

Engineering School

University of Minho

Guimarães, Portugal

September 2022

*If needed, this document should be cited as:
P. Cortez, A tutorial on the rminer R package for data mining tasks, Teaching Report, Version 2, Department of Information Systems, ALGORITMI Research Centre, Engineering School, University of Minho, Guimarães, Portugal, September 2022.

Abstract

This tutorial explores the rminer package of the R statistical tool. Following a learning by example approach, several code recipes are presented and the obtained results analyzed. The goal is to demonstrate the package capabilities for executing classification and regression (including time series forecasting) data mining tasks. A special focus is provided to three Cross Industry Standard Process for Data Mining (CRISP-DM) stages: data preparation, modeling and evaluation.

Keywords: Automated Machine Learning (AutoML), Classification, Computational Intelligence, Data Mining, Regression, R tool, Time Series.

Contents

Chapter 1

Introduction

This tutorial explores the rminer package of the R tool. Rather than providing state-of-the-art predictive performances and producing code that might require an heavy computation, the goal is to show simple demonstration code examples for executing classification and regression (including time series forecasting) data mining tasks. Once these code examples are understood by users, then the code can be extended and adapted for more complex uses. All code examples presented in this tutorial are available at: `https://drive.google.com/file/d/1lpi5BQpDDQ8c9yiP4WRZ_bzFXY6p-MzT/view?usp=sharing`. This tutorial assumes previous knowledge about the R tool and data mining basic concepts. More details about these topics can be found in: R tool – (Paradis, 2002; Zuur et al., 2009; Venables et al., 2013); data mining, classification and regression – (Hastie et al., 2009; Witten et al., 2017).

The rminer package (`https://cran.r-project.org/package=rminer`) goal is to provide a reduced and coherent set of R functions to perform classification and regression. The package is particularly suited for non R expert users, as it allows to perform the full data mining process using very few lines of code. Figure 1.1 shows the suggested use of the rminer package and its relation with the Cross Industry Standard Process for Data Mining (CRISP-DM) methodology (Chapman et al., 2000). As shown by the figure, the rminer package includes functions that are useful in three CRISP-DM stages: data preparation, modeling and evaluation. Also, the advised rminer use implies writing a distinct R file or script for each CRISP-DM phase, with each R file receiving inputs and generating outputs (e.g., such as file `2b-math-1.R` used for the data preparation of the student performance analysis, as shown in Section 2.4). The next tutorial chapters are devoted to these three stages (Chapters 2, 3 and 4). Then, the particular case of time series forecasting is addressed in Chapter 5. Finally, closing conclusions are drawn (Chapter 6).

The rminer package has been adopted by users with distinct domain expertises and in a wide range of applications. From 2th May of 2013 to 15th December of 2020, the package has been downloaded 23,306 times from the RStudio CRAN servers (`cran.rstudio.com`). The rminer package has been used by both information technology (IT) and non IT users (e.g., managers, biologists or civil engineers). There is a large list of rminer applications performed by the author of this tutorial, including (among others):

Classification:

- mortality prediction (Silva et al., 2006) and rating organ failure in intensive care units (Silva et al., 2008);

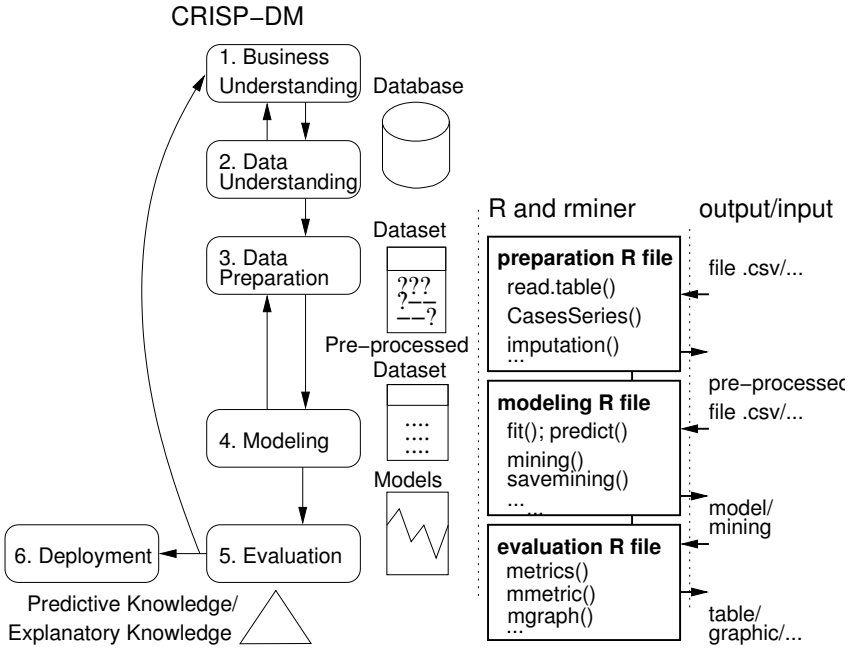

Figure 1.1: The CRISP-DM methodology and suggest rminer package use, adapted from (Cortez, 2010a).

- predicting secondary school student performance (Cortez and Silva, 2008);
- spam email detection (Lopes et al., 2011); and
- bank telemarketing (Moro et al., 2014).

Regression:

- lamb meat quality assessment (Cortez et al., 2006);
- estimating wine quality (Cortez et al., 2009);
- studying the impact of topology characteristics on wireless mesh networks (Calçada et al., 2012);
- time series forecasting (Stepnicka et al., 2013);
- predicting jet grouting columns uniaxial compressive strength (Tinoco et al., 2014);
- predicting hospital Length of Stay (Caetano et al., 2014);
- estimating earthworks and soil compaction equipment performance (Parente et al., 2015);
- stock market prediction based on microblogging data (Oliveira et al., 2017);
- estimating the number of sales of smartphone devices by eBay sellers (Silva et al., 2018); and
- multivariate time series demand forecasting (Gonçalves et al., 2021).

The package has also been used by other researchers. A few examples such applications are: bioinformatics (Fortino et al., 2014), marketing (Nachev and Hogan, 2014), oceanography (Cortese et al., 2013), software project costs (Mittas and Angelis, 2013), medicine (Sanchis-Sánchez et al., 2016) and river flow forecasting (Zhang et al., 2020).

1.1 Installation and Usage

The R is an open source and multi-platform tool that can be downloaded from the official web site: `http://www.r-project.org/` (a recent and stable version should be selected). The rminer package can be easily installed by opening the R program and using its package installation menus or by typing the command after the prompt: **`install.packages("rminer")`**. Only one package installation is needed for a particular R version.

The rminer functions and help are available once the package is loaded, by using the command: **`library(rminer)`**. Even if the package is not loaded, the package is still accessible by using the `::` (double colon) operator. Such operator can also be used to disambiguate functions from distinct packages. An example of the use of rminer is:

```
library(rminer) # load the package
help(package=rminer) # full list of rminer functions
help(mmetric) # help on rminer mmetric function
help(fit) # help on modeltools::fit and rminer::fit
help(fit,package=rminer) # direct help on fit
?rminer::fit # same direct help
# any rminer function can be called, such as mmetric:
cat("MAE:",mmetric(1:5,5:1,metric="MAE"),"\n")
```

The next example uses rminer without calling the library function:

```
# any rminer function can be called if package was installed
# and the :: operator is used:
help(package=rminer) # full list of rminer functions
help(mmetric,package=rminer) # help on rminer mmetric function
?rminer::fit # direct help
# any rminer function can be called with rminer::, such as mmetric:
cat("MAE:",rminer::mmetric(1:5,5:1,metric="MAE"),"\n")
```

1.2 Help

Once the package is loaded, the full list of the rminer functions is made available by executing: **`help(package=rminer)`**, as shown in the codes examples of Section 1.1. Examples of a direct help for some of its main functions are: **`help(fit,package=rminer)`**, **`help(predict.fit)`**, **`help(mining)`**, **`help(mmetrics)`** and **`help(mgraph)`**.

1.3 Citation

If you find the rminer package useful, please cite it in your publications. The full reference is:

Cortez, P. (2010). Data Mining with Neural Networks and Support Vector Machines using the R/rminer Tool. In Perner, P., editor, Advances in Data Mining Applications and Theoretical Aspects, 10th Industrial Conference on Data Mining, pages 572583, Berlin, Germany. LNAI 6171, Springer.

The bibtex reference can be found here: `https://dblp.uni-trier.de/rec/conf/incdm/Cortez10.html?view=bibtex`

Chapter 2

Data Preparation

The data preparation stage of CRISP-DM can include several tasks, such as data selection, cleaning and transformation.

2.1 Loading Data

The rminer package assumes that a dataset is available as a `data.frame` R object. This can easily be achieved by using the `data` or `read.table` R functions. The former function loads datasets already made available in R packages, while the latter can load tabulated data files (e.g., CSV format). One important aspect is that categorical attributes need to be converted into factors. When using the `read.table` function, this is achieved by seting the argument `stringsAsFactors=TRUE`. As an demonstrative example, file `2-prep-1.R` loads several datasets, including the famous Iris and some University California Irvine (UCI) Machine Learning (ML) repository (Asuncion and Newman, 2007) datasets donated by the author of this document, namely: Wine Quality (Cortez et al., 2009), Forest Fires (Cortez and Morais, 2007), Bank Marketing (Moro et al., 2014) and Student Performance (Cortez and Silva, 2008).

```
### 2-prep-1.R: data loading example
### saves math2.csv into current working directory, as defined by getwd()

# simple show rows x columns function
nelems=function(d) paste(nrow(d),"x",ncol(d))

# load the famous Iris dataset:
data(iris) # load the data
cat("iris:",nelems(iris),"\n")
print(class(iris)) # show class
print(names(iris)) # show attributes

### load all my UCI ML datasets ###

# White Wine Quality dataset:
URL="http://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/
   winequality-white.csv"
wine=read.table(file=URL,header=TRUE,sep=";")
cat("wine quality white:",nelems(wine),"\n")
print(class(wine)) # show class
nelems(wine) # show rows x columns
print(names(wine)) # show attributes

# forest fires dataset:
```

```
URL="http://archive.ics.uci.edu/ml/machine-learning-databases/forest-fires/
    forestfires.csv"
fires=read.table(file=URL,header=TRUE,sep=",",stringsAsFactors=TRUE)
cat("forest fires:",nelems(fires),"\n")
print(class(fires)) # show class
print(names(fires)) # show attributes

# load bank marketing dataset (in zip file):
URL="http://archive.ics.uci.edu/ml/machine-learning-databases/00222/bank-
    additional.zip"
temp=tempfile() # temporary file
download.file(URL,temp) # download file to temporary
# unzip file and load into data.frame:
bank=read.table(unz(temp,"bank-additional/bank-additional.csv"),sep=";",
    header=TRUE,stringsAsFactors=TRUE)
cat("bank marketing:",nelems(bank),"\n")
print(class(bank)) # show class
print(names(bank)) # show attributes

# student performance dataset (in zip file):
URL="http://archive.ics.uci.edu/ml/machine-learning-databases/00320/student
    .zip"
temp=tempfile() # temporary file
download.file(URL,temp) # download file to temporary
# unzip file and load into data.frame:
math=read.table(unz(temp,"student-mat.csv"),sep=";",header=TRUE,
    stringsAsFactors=TRUE)
cat("student performance math:",nelems(math),"\n")
print(class(math)) # show class
print(names(math)) # show attributes
# save data.frame to csv file:
write.table(math,file="math.csv",row.names=FALSE,col.names=TRUE)
```

The obtained output is:

```
> source("2-prep-1.R")
iris: 150 x 5
[1] "data.frame"
[1] "Sepal.Length" "Sepal.Width"  "Petal.Length" "Petal.Width"  "Species"
wine quality white: 4898 x 12
[1] "data.frame"
 [1] "fixed.acidity"        "volatile.acidity"     "citric.acid"
 [4] "residual.sugar"       "chlorides"            "free.sulfur.dioxide"
 [7] "total.sulfur.dioxide" "density"              "pH"
[10] "sulphates"            "alcohol"              "quality"
forest fires: 517 x 13
[1] "data.frame"
 [1] "X"     "Y"     "month" "day"   "FFMC"  "DMC"   "DC"    "ISI"   "temp"
      "RH"
[11] "wind"  "rain"  "area"
trying URL 'http://archive.ics.uci.edu/ml/machine-learning-databases/00222/
    bank-additional.zip'
Content type 'application/x-httpd-php' length 444572 bytes (434 KB)
==================================================
downloaded 434 KB

bank marketing: 4119 x 21
[1] "data.frame"
 [1] "age"            "job"            "marital"        "education"      "
```

```
    default"
 [6] "housing"        "loan"           "contact"        "month"          "
    day_of_week"
[11] "duration"       "campaign"       "pdays"          "previous"       "
   poutcome"
[16] "emp.var.rate"   "cons.price.idx" "cons.conf.idx"  "euribor3m"      "
   nr.employed"
[21] "y"
trying URL 'http://archive.ics.uci.edu/ml/machine-learning-databases/00320/
   student.zip'
Content type 'application/x-httpd-php' length 20478 bytes (19 KB)
==================================================
downloaded 19 KB

student performance math: 395 x 33
[1] "data.frame"
 [1] "school"     "sex"        "age"        "address"    "famsize"    "
    Pstatus"    "Medu"
 [8] "Fedu"       "Mjob"       "Fjob"       "reason"     "guardian"   "
    traveltime" "studytime"
[15] "failures"   "schoolsup"  "famsup"     "paid"       "activities" "
   nursery"    "higher"
[22] "internet"   "romantic"   "famrel"     "freetime"   "goout"      "Dalc
   "       "Walc"
[29] "health"     "absences"   "G1"         "G2"         "G3"
```

2.2 Data Selection and Transformation

Data selection and transformation is a crucial step for achieving a successful data mining project and it includes may several operations, such as outlier detection and removal, attribute and instance selection, assuring data quality, transforming attributes, etc. For further details, consult (Witten et al., 2017).

A `data.frame` can be easily manipulated as a `matrix` using standard R operations (e.g., `cut`, tt sample, `cbind`, `nrow`). The `rminer` package includes some useful preprocessing functions, such as:

`delevels` – reduce or replace factor levels;

`imputation` – missing data imputation and;

`CasesSeries` – create a data.frame from a time series (vector) using a sliding window. A sliding window contains a set of time lags that are used to define variable inputs from a series. For example, let us consider the series $6_1, 10_2, 14_3, 18_4, 23_5, 27_6$ (y_t values) with 6 elements. If the $\{1,3\}$ sliding window is used, then three training examples can be created (each example with 2 inputs and one output): $6, 14 \rightarrow 18$, $10, 18 \rightarrow 23$ and $14, 23 \rightarrow 27$. This simple example can be tested by using the command: `CasesSeries(c(6,10,14,18,23,27),c(1,3))`.

These functions will be applied to the datasets loaded in Section 2.1 in the next and subsequent demonstration files. Some initial selection and transformation operations are executed in file `2-prep-2.R`:

```
### 2-prep-2.R: data selection and transformation example
### saves prep2-1.pdf, wine3.csv and bank2.csv file into working directory
```

```
library(rminer) # load rminer package

### row and column selection examples:
# select only virginica data rows:
print("-- 1st example: selection of virginica rows --")
iris2=iris[iris$Species=="virginica",]
cat("iris2",nelems(iris2),"\n")
print(table(iris2$Species))

# select a random subsample of white wine dataset:
print("-- 2nd example: selection of 500 wine samples --")
set.seed(12345) # set for replicability
s=sample(1:nrow(wine),500) # random with 500 indexes
wine2=wine[s,] # wine2 has only 500 samples from wine
cat("wine2",nelems(wine2),"\n")

# column (attribute) selection example:
print("-- 3rd example: selection of 5 wine columns --")
att=c(3,6,8,11,12) # select 5 attributes
wine3=wine2[,att]
cat("wine3",nelems(wine3),"\n")
print(names(wine3))

### attribute transformation examples:
# numeric to discrete:
# three classes poor=[1 to 4], average=[5 to 6], good=[6 to 10]
print("-- 4th example: numeric to discrete transform (wine) --")
wine3$quality=cut(wine3$quality,c(1,4,6,10),c("poor","average","good"))
print(table(wine2$quality))
print(table(wine3$quality))
# save new data.frame to csv file:
write.table(wine3,file="wine3.csv",row.names=FALSE,col.names=TRUE)

# numeric to numeric:
# log transform to forest fires area (positive skew)
print("-- 5th example: numeric to numeric transform (forest fires) --")
logarea=log(fires$area+1)
pdf("prep2-1.pdf") # create pdf file
par(mfrow=c(2,1))
hist(fires$area,col="gray")
hist(logarea,col="gray")
dev.off() # end of pdf creation

# discrete to discrete (factor to factor)
# transform month into trimesters
print("-- 6th example: discrete to discrete (bank marketing) --")
print(table(bank$month))
tri=delevels(bank$month,levels=c("jan","feb","mar"),label="1st")
tri=delevels(tri,levels=c("apr","may","jun"),label="2nd")
tri=delevels(tri,levels=c("jul","aug","sep"),label="3rd")
tri=delevels(tri,levels=c("oct","nov","dec"),label="4th")
# put the levels in order:
tri=relevel(tri,"1st")
print(table(tri))
# create new data.frame with bank and new atribute:
bank2=cbind(bank[1:9],tri,bank[10:ncol(bank)])
cat("bank2",nelems(bank2),"\n")
print(names(bank2))
```

```
# save into a csv file:
write.table(bank2,file="bank2.csv",sep=";",row.names=FALSE,col.names=TRUE)

# load the airline passengers data:
data(AirPassengers)
print("-- 6th example: time series to data.frame (AirPassengers) --")
dtraffic=CasesSeries(AirPassengers,c(1,12,13),1,18) # 1st five training
   examples
print(dtraffic)
```

The execution result is:

```
> source("2-prep-2.R")
[1] "-- 1st example: selection of virginica rows --"
iris2 50 x 5

    setosa versicolor  virginica
         0          0         50
[1] "-- 2nd example: selection of 500 wine samples --"
wine2 500 x 12
[1] "-- 3rd example: selection of 5 wine columns --"
wine3 500 x 5
[1] "citric.acid"         "free.sulfur.dioxide" "density"             "
   alcohol"
[5] "quality"
[1] "-- 4th example: numeric to discrete transform (wine) --"

  3   4   5   6   7   8
  1  17 151 221  95  15

   poor average    good
     18     372     110
[1] "-- 5th example: numeric to numeric transform (forest fires) --"
[1] "-- 6th example: discrete to discrete (bank marketing) --"

 apr  aug  dec  jul  jun  mar  may  nov  oct  sep
 215  636   22  711  530   48 1378  446   69   64
tri
 1st  2nd  3rd  4th
  48 2123 1411  537
bank2 4119 x 22
 [1] "age"            "job"            "marital"        "education"      "
    default"
 [6] "housing"        "loan"           "contact"        "month"          "
    tri"
[11] "day_of_week"    "duration"       "campaign"       "pdays"          "
   previous"
[16] "poutcome"       "emp.var.rate"   "cons.price.idx" "cons.conf.idx"  "
   euribor3m"
[21] "nr.employed"    "y"
[1] "-- 6th example: time series to data.frame (AirPassengers) --"
  lag13 lag12 lag1   y
1   112   118  115 126
2   118   132  126 141
3   132   129  141 135
4   129   121  135 125
5   121   135  125 149
```

Figure 2.1 shows the result of the two created histograms. Turning to the **rminer** functions,

the `delevels` function is used to reduce the number of levels of the `bank$month` attribute, replacing the month labels by their respective trimester. Also, the `CasesSeries` function is used to build a `data.frame` from the AirPassengers time series (numeric vector). In the example, the sliding window is made of the {1,12,13} time lags and only applied to the first 18 elements of the series, generating a tabular object with five examples.

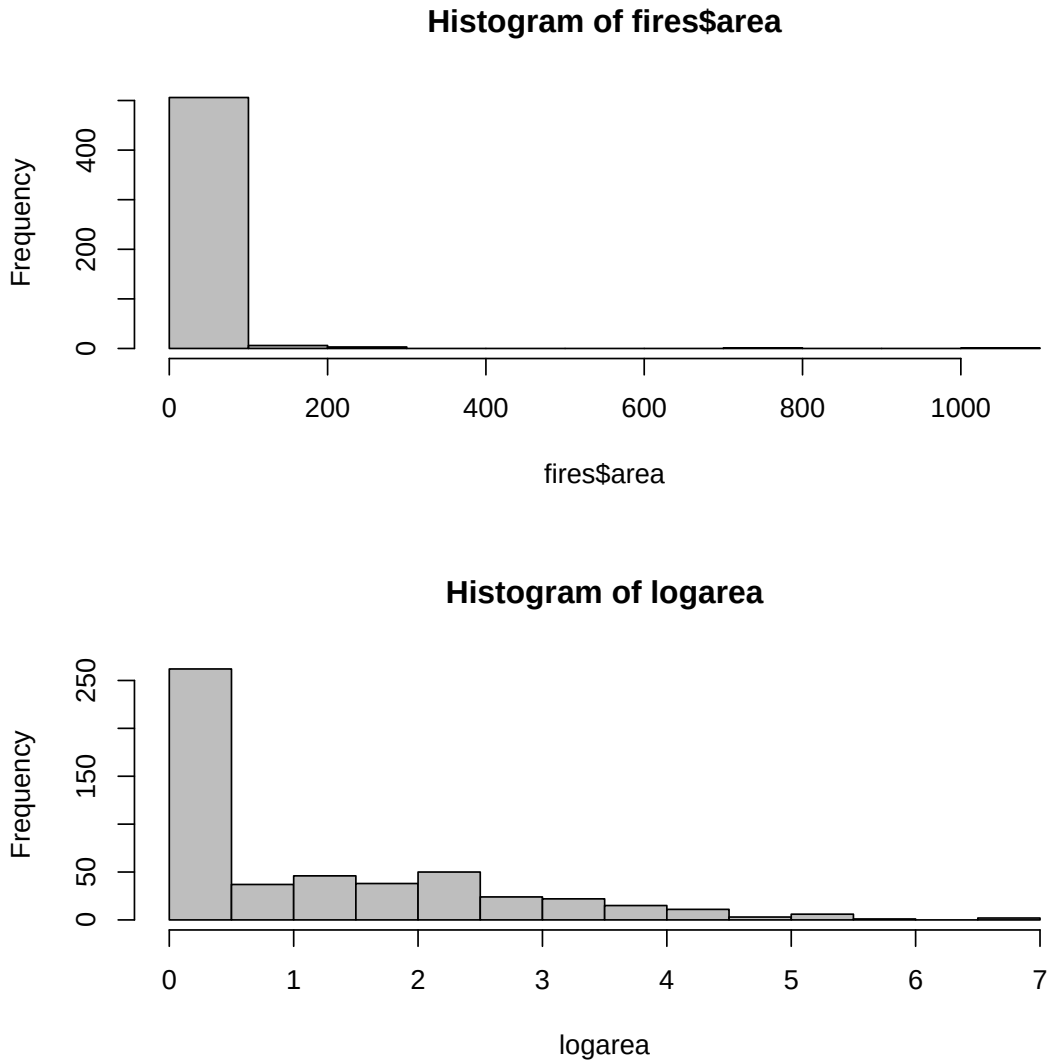

Figure 2.1: Histogram for the forest fires area (top) and its logarithm transform (bottom).

2.3 Missing Data

Missing data is quite common in some domains, such as questionnaire responses. There are several methods for handing missing data, such as: case or attribute deletion; value imputation; and hot deck (Brown and Kros, 2003).

The first method can be easily adopted in R by using the `na.omit` R function. The `imputation` function from `rminer` implements the other methods. Additional R missing data handling meth-

ods can be found at: `https://cran.r-project.org/web/views/MissingData.html`. An example for the bank data is provided in file **`2-prep-3.R`**:

```
### 2-prep-3.R: missing data example
### saves prep3-1.pdf into current working directory

require(kknn) # needed by hotdeck
# missing data example
bank3=read.table("bank2.csv",sep=";",header=TRUE,stringsAsFactors=TRUE,na.
    strings="unknown")
NRbank=nrow(bank3)
cat("rows:",NRbank,"\n")

# 1st method: case deletion
print("-- 1st method: case deletion --")
bank4=na.omit(bank3)
cat("bank4:",nelems(bank4),"\n")
cat("NA values:",sum(is.na(bank4)),"\n")
# deleted %:
cat("deleted percentage:",round(100*(1-nrow(bank4)/NRbank),1),"%\n")

# 2nd method: average imputation for age, mode imputation for job:
bank3$age[1:500]=NA # insert 500 NA values into age
# substitute NA values by the mean:
print("-- 2nd method: value imputation --")
print("original age summary:")
print(summary(bank3$age))
meanage=mean(bank3$age,na.rm=TRUE)
bank5=imputation("value",bank3,"age",Value=meanage)
cat("mean imputation age summary: mean=",meanage,"\n")
print(summary(bank5$age))
# substitute NA values by the mode (most common value of bank$job):
print("original job summary:")
print(summary(bank3$job))
jobmode=names(which.max(table(bank$job)))
bank5=imputation("value",bank5,"job",Value=jobmode)
cat("mode imputation job summary: mode=",jobmode,"\n")
print(summary(bank5$job))

# 3rd method: hot deck
# substitute NA values by the values found in most similar case (1-nearest
    neighbor):
print("-- 3rd method: hotdeck imputation --")
print("original age summary:")
print(summary(bank3$age))
bank6=imputation("hotdeck",bank3,"age")
print("hot deck imputation age summary:")
print(summary(bank6$age))
# substitute NA values by the values found in most similar case:
print("original job summary:")
print(summary(bank3$job))
bank6=imputation("hotdeck",bank6,"job")
print("hot deck imputation job summary:")
print(summary(bank6$job))
cat("bank6:",nelems(bank6),"\n")
cat("NA values:",sum(is.na(bank6)),"\n")
# full hotdeck:
bank7=imputation("hotdeck",bank3)
```

```
print("full hot deck imputation summary:")
print(summary(bank7))
cat("bank7:",nelems(bank7),"\n")
cat("NA values:",sum(is.na(bank7)),"\n")


# comparison of age densities (mean vs hotdeck):
library(ggplot2)
meth1=data.frame(length=bank4$age)
meth2=data.frame(length=bank5$age)
meth3=data.frame(length=bank6$age)
meth1$method="original"
meth2$method="average"
meth3$method="hotdeck"
all=rbind(meth1,meth2,meth3)
ggplot(all,aes(length,fill=method))+geom_density(alpha = 0.2)
ggsave(file="prep3-1.pdf")
```

The `read.table` command reads all "unknown" values as `NA` (the R constant value for missing data). The numeric age attribute does not contain missing data, however for demonstration purposes, the R file replaces the first 500 values by the **NA** constant. Then, three missing data handling methods are applied: case deletion, average and mode imputation, and hot deck imputation. In the example code, there are two instructions for using the hot deck method, one for each attribute (age and job). The same hot deck method can be applied to all data under a single execution, as exemplified by the code that generates object `bank7`. Under this use of the imputation function, the hotdeck imputation is applied to all attributes with missing data. The result of executing file `2-prep-3.R` is:

```
> source("2-prep-3.R")
rows: 4119
[1] "-- 1st method: case deletion --"
bank4: 3090 x 22
NA values: 0
deleted percentage: 25 %
[1] "-- 2nd method: value imputation --"
[1] "original age summary:"
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
  18.00   32.00   38.00   40.05   47.00   88.00     500
mean imputation age summary: mean= 40.04532
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  18.00   33.00   40.00   40.05   46.00   88.00
[1] "original job summary:"
       admin.   blue-collar  entrepreneur     housemaid    management
                retired self-employed
         1012           884           148           110           324
                      166           159
     services       student    technician    unemployed          NA's
          393            82           691           111            39
mode imputation job summary: mode= admin.
       admin.   blue-collar  entrepreneur     housemaid    management
                retired self-employed
         1051           884           148           110           324
                      166           159
     services       student    technician    unemployed
          393            82           691           111
[1] "-- 3rd method: hotdeck imputation --"
[1] "original age summary:"
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
  18.00   32.00   38.00   40.05   47.00   88.00     500
[1] "hot deck imputation age summary:"
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  18.00   32.00   38.00   40.02   47.00   88.00
[1] "original job summary:"
       admin.   blue-collar  entrepreneur     housemaid    management
                retired self-employed
         1012           884           148           110           324
                      166           159
     services       student    technician    unemployed          NA's
          393            82           691           111            39
[1] "hot deck imputation job summary:"
       admin.   blue-collar  entrepreneur     housemaid    management
                retired self-employed
         1019           890           148           112           329
                      168           161
     services       student    technician    unemployed
          399            82           697           114
bank6: 4119 x 22
NA values: 1191
[1] "full hot deck imputation summary:"
      age                 job           marital                    education
            default    housing
 Min.   :18.00   admin.     :1023   divorced: 448   basic.4y           :
    445   no :4118   no :1891
 1st Qu.:32.00   blue-collar: 890   married :2514   basic.6y           :
    240   yes:   1   yes:2228
 Median :38.00   technician : 694   single  :1157   basic.9y           :
    591
 Mean   :40.02   services   : 399                   high.school        :
    972
 3rd Qu.:47.00   management : 325                   illiterate         :
    1
 Max.   :88.00   retired    : 169                   professional.course:
    557
                 (Other)    : 619                   university.degree
                    :1313
  loan           contact         month       tri       day_of_week
     duration         campaign
 no :3436   cellular :2652   may    :1378   1st:  48   fri:768     Min.   :
       0.0   Min.   : 1.000
 yes: 683   telephone:1467   jul    : 711   2nd:2123   mon:855     1st Qu.:
     103.0   1st Qu.: 1.000
                             aug    : 636   3rd:1411   thu:860     Median :
                                 181.0   Median : 2.000
                             jun    : 530   4th: 537   tue:841     Mean   :
                                 256.8   Mean   : 2.537
                             nov    : 446              wed:795     3rd Qu.:
                                 317.0   3rd Qu.: 3.000
                             apr    : 215                          Max.
                                :3643.0   Max.   :35.000
                             (Other): 203
     pdays          previous             poutcome     emp.var.rate
        cons.price.idx
 Min.   :  0.0   Min.   :0.0000   failure    : 454   Min.   :-3.40000   Min
    .   :92.20
 1st Qu.:999.0   1st Qu.:0.0000   nonexistent:3523   1st Qu.:-1.80000   1st
```

```
     Qu.:93.08
 Median :999.0   Median :0.0000   success    : 142   Median : 1.10000
    Median :93.75
 Mean   :960.4   Mean   :0.1903                      Mean   : 0.08497
    Mean   :93.58
 3rd Qu.:999.0   3rd Qu.:0.0000                      3rd Qu.: 1.40000   3rd
     Qu.:93.99
 Max.   :999.0   Max.   :6.0000                      Max.   : 1.40000   Max
    .   :94.77

 cons.conf.idx     euribor3m      nr.employed     y
 Min.   :-50.8   Min.   :0.635   Min.   :4964   no :3668
 1st Qu.:-42.7   1st Qu.:1.334   1st Qu.:5099   yes: 451
 Median :-41.8   Median :4.857   Median :5191
 Mean   :-40.5   Mean   :3.621   Mean   :5166
 3rd Qu.:-36.4   3rd Qu.:4.961   3rd Qu.:5228
 Max.   :-26.9   Max.   :5.045   Max.   :5228

bank7: 4119 x 22
NA values: 0
Saving 7 x 7 in image
```

The density graph for attribute age and the distinct missing handling methods is plotted in Figure 2.2. Such graph confirms the disadvantage of the average substitution method, which tends to increase the density of points near the average of the attribute, as expected. In contrast, the hotdeck replacement method leads to an attribute distribution that is very similar to the original data (when analyzing the non missing values).

2.4 Example with the Student Performance Dataset

As a case study, and for demonstrating the classification and regression capabilities of the rminer package, the student performance dataset (Cortez and Silva, 2008) is adopted. The goal is to predict one of the dataset course grades (Mathematics) taught in secondary education Portuguese schools. The data was collected using school reports and questionnaires and it includes discrete and numeric attributes related with demographic, social and school characteristics. The last attribute (`"G3"`), contains the target variable (Mathematics grade) and it ranges from 0 to 20, where a positive score means a value higher or equal to 10. Such numeric attribute will be directly used, in case of regression, and transformed into binary (`"pass"`) and five-level (`"five"`) attributes, in case of classification. The preprocessing R code, which creates the binary and five-level attributes, is presented in file `2-math.R`:

```
### 2-math.R: math data preparation example
### creates math-grades.pdf with histograms for the 3 tasks,
### binary classification (pass or fail), five grades (A to F) and
### G3 (final scores from 0 to 20, numeric target)
### saves math2.csv into current working directory (with G3, pass and five)

# data preparation:
math=read.table(file="math.csv",header=TRUE,stringsAsFactors=TRUE) # read
   previously saved file

# binary task:
pass=cut(math$G3,c(-1,9,20),c("fail","pass"))
# five-level system:
```

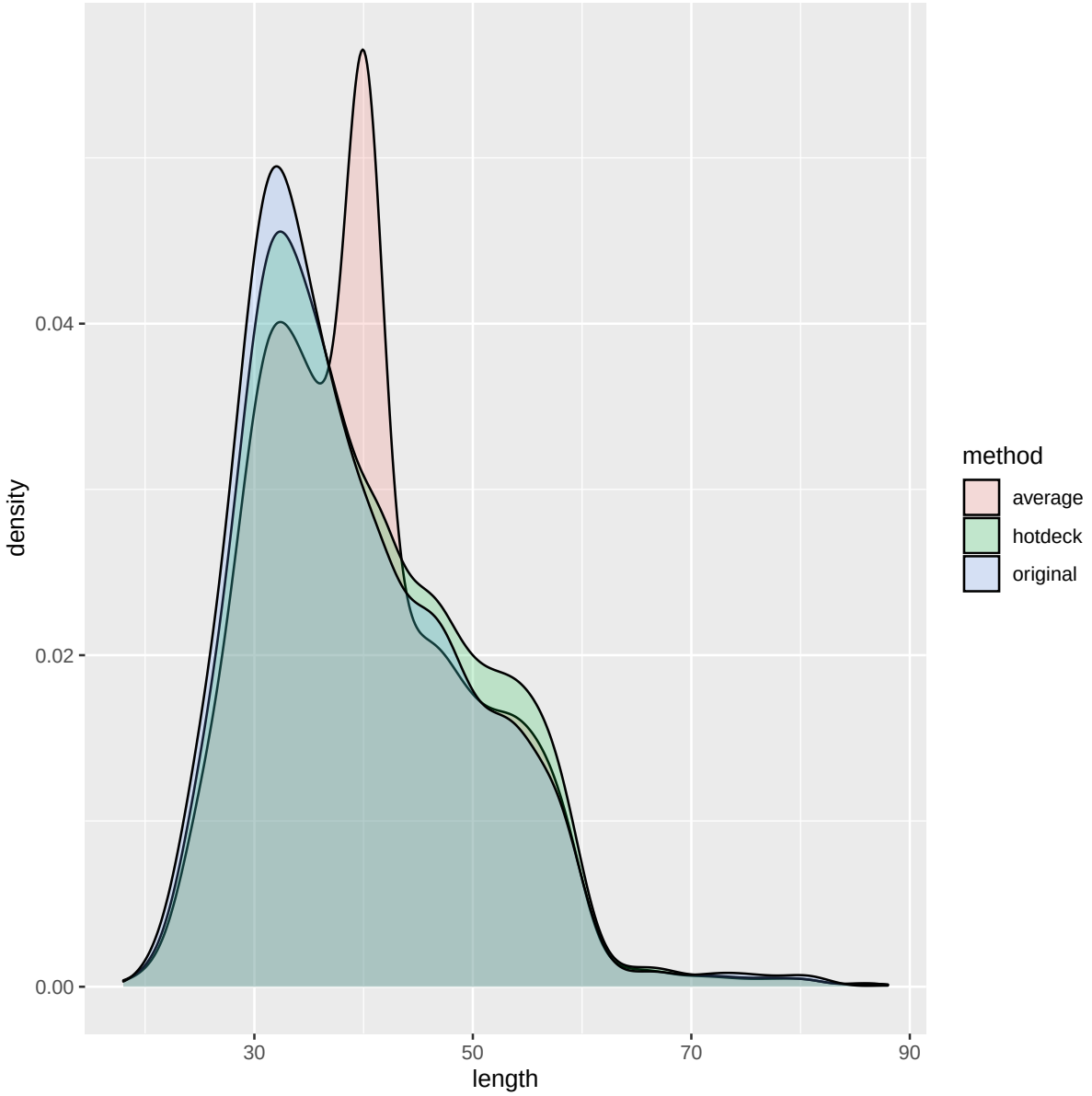

Figure 2.2: Density graphs for the two missing age imputation methods.

```
five=cut(math$G3,c(-1,9,11,13,15,20),c("F","D","C","B","A")) # Ireland
    grades
# create pdf:
pdf("math-grades.pdf")
par(mfrow=c(1,3))
plot(pass,main="pass")
plot(five,main="five")
hist(math$G3,col="gray",main="G3",xlab="")
dev.off() # end of pdf creation

# creating the full dataset:
d=cbind(math,pass,five)
write.table(d,"math2.csv",row.names=FALSE,col.names=TRUE) # save to file
```

The code produces a new data frame that is saved into file `"math2.csv"` and creates a pdf with

the distinct output histograms (Figure 2.3).

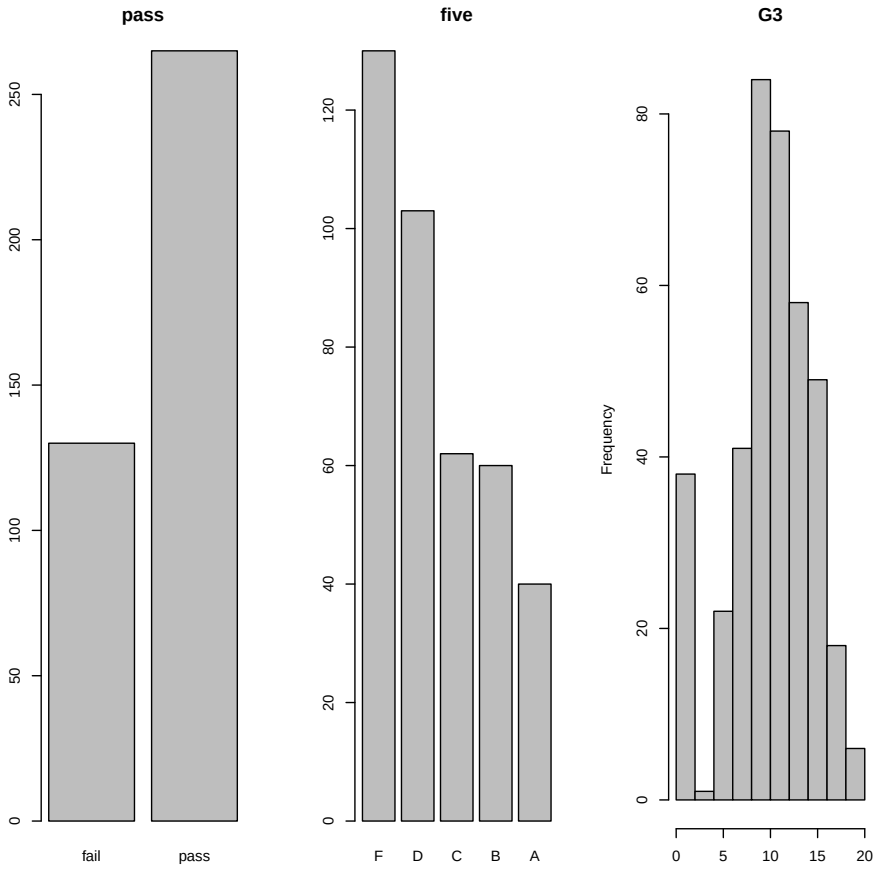

Figure 2.3: Histograms for the Mathematics student's grades (binary, five-level and numeric attributes).

Chapter 3

Modeling

The rminer includes a total of 14 classification and 15 regression methods, all directly available through its `fit`, `predict` and `mining` functions:

`fit` – adjusts a selected model to a dataset; if needed, it can automatically tune the model hyperparameters;

`predict` – given a fitted model, it computes the predictions for a (often new) dataset; and

`mining` – performs several fit and predict executions, according to a validation method and given number of runs.

By default, the type of rminer modeling (probabilistic classification or regression) is dependent of the output target type: if `factor` (categorical or discrete), then a probabilistic classification is assumed; else if numeric (e.g., `integer`, `numeric`), then a regression task is executed. Such default modeling is stored in the `task` argument/object of the `fit` and `mining` functions. Further technical details about these functions can be found in (Cortez, 2010a) and in the rminer help[1].

3.1 Classification

The rminer package includes several classification methods, which can be listed by using the command: `help(fit)`. When performing a classification task, the output variable needs to be categorical or discrete (a `factor`). By default, the package assumes a probabilistic modeling of such output (`task="prob"`), where the sum of all outputs equals 1. Using probabilities is more advantageous, as it allows to perform a receiver operating characteristic (ROC) (Fawcett, 2006) or accumulated LIFT (Witten et al., 2011) curve analysis (shown in Chapter 4). Moreover, class probabilities can easily be transformed into class labels by setting a decision threshold $D \in [0,1]$, such that the class is positive if its probability is higher than D. In case of multiclass tasks, the final label can be associated with the highest proability class.

3.1.1 Binary Classification

The first modeling code is given in file `3-math-1.R`:

```
### 3-math-1.R: modeling math binary classification example
### note: models are fit to all data, predictions are computed using all
   data
```

[1] For instance, by executing: `help(fit,package=rminer)`

```
### creates 2 pdf files: trees-1.pdf and trees-2.pdf
### saves mlpe-pass.model

library(rminer)

# read previously saved file
math=read.table(file="math2.csv",header=TRUE,stringsAsFactors=TRUE)

# select inputs:
inputs=2:29 # select from 2 ("sex") to 29 ("health")

# select outputs: binary task "pass"
bout=which(names(math)=="pass")
cat("output class:",class(math[,bout]),"\n")

# two white-box examples:
B1=fit(pass~.,math[,c(inputs,bout)],model="rpart") # fit a decision tree
print(B1@object)
pdf("trees-1.pdf")
# rpart functions:
plot(B1@object,uniform=TRUE,branch=0,compress=TRUE)
text(B1@object,xpd=TRUE,fancy=TRUE,fwidth=0.2,fheight=0.2)
dev.off()

B2=fit(pass~.,math[,c(inputs,bout)],model="ctree") # fit a conditional
   inference tree
print(B1@object)
pdf("trees-2.pdf")
# ctree function:
plot(B2@object)
dev.off()

# two black-box examples:
B3=fit(pass~.,math[,c(inputs,bout)],model="mlpe") # fit a multilayer
   perceptron ensemble
print(B3@object)

B4=fit(pass~.,math[,c(inputs,bout)],model="ksvm") # fit a support vector
   machine
print(B4@object)

# save one model to a file:
print("save B3 to file")
savemodel(B3,"mlpe-pass.model") # saves to file
print("load from file into B5")
B5=loadmodel("mlpe-pass.model") # load from file
print(class(B5@object$mlp[[1]]))
```

The code fits two white-box (`"rpart"` and `"ctree"`) and two black-box models (`"mlpe"` and `"ksvm"`). To simplify the understanding of the code, only a modeling task is executed, where each data mining or machine learning model is fit to all data examples. If the goal is to measure the predictive performance of the fitted models, then a generalization validation method should be used, such as described in Section 3.2 (e.g., holdout or k-fold). The arguments used by the `fit` function in the code example are: a formula of the model to be fit, a `data.frame` with the training data, and a character that selects the type of learning model. The formula defines the output (variable "`pass`") to be modeled ($\sim$) from the inputs (`.` means all other `data.frame` variables). The `data.frame` includes the selected inputs and output variables. This is the typical use of `fit`,

where formula is always in the format *variable* ~ . and the selection of the inputs is executed in the `data.frame` fed as training data (as shown in the code examples). The third argument defines the learning model. The rminer package includes several classification and regression models, such as decision trees (`"rpart"`), conditional inference trees (`"ctree"`), neural networks (e.g., `"mlpe"` – ensemble of multilayer perceptrons) and support vector machines (`"ksvm"`). In general, the rminer package uses the original function names for the `model` argument, as implemented in their packages. For instance, decision trees use the `rpart()` function of the rpart package, while `ksvm()` is the function of the kernlab package that implements a support vector machine. Finally, it should be noted that the `fit` function includes other optional arguments, as described in in the help (e.g., execute `?rminer::fit`) to get the full details and more examples).

The result of executing file `math-2.R` is:

```
> source("3-math-1.R")
output class: factor
n= 395

node), split, n, loss, yval, (yprob)
      * denotes terminal node

 1) root 395 130 pass (0.3291139 0.6708861)
   2) failures>=0.5 83  31 fail (0.6265060 0.3734940)
     4) failures>=1.5 33   7 fail (0.7878788 0.2121212) *
     5) failures< 1.5 50  24 fail (0.5200000 0.4800000)
      10) age>=16.5 37  14 fail (0.6216216 0.3783784)
        20) guardian=father,mother 26   6 fail (0.7692308 0.2307692) *
        21) guardian=other 11   3 pass (0.2727273 0.7272727) *
      11) age< 16.5 13   3 pass (0.2307692 0.7692308) *
   3) failures< 0.5 312  78 pass (0.2500000 0.7500000)
     6) schoolsup=yes 40  18 pass (0.4500000 0.5500000)
      12) studytime>=1.5 31  14 fail (0.5483871 0.4516129)
        24) reason=course 11   2 fail (0.8181818 0.1818182) *
        25) reason=home,other,reputation 20   8 pass (0.4000000 0.6000000)
          50) Fedu< 2.5 7   2 fail (0.7142857 0.2857143) *
          51) Fedu>=2.5 13   3 pass (0.2307692 0.7692308) *
      13) studytime< 1.5 9   1 pass (0.1111111 0.8888889) *
     7) schoolsup=no 272  60 pass (0.2205882 0.7794118)
      14) guardian=other 10   4 fail (0.6000000 0.4000000) *
      15) guardian=father,mother 262  54 pass (0.2061069 0.7938931) *
n= 395

node), split, n, loss, yval, (yprob)
      * denotes terminal node

 1) root 395 130 pass (0.3291139 0.6708861)
   2) failures>=0.5 83  31 fail (0.6265060 0.3734940)
     4) failures>=1.5 33   7 fail (0.7878788 0.2121212) *
     5) failures< 1.5 50  24 fail (0.5200000 0.4800000)
      10) age>=16.5 37  14 fail (0.6216216 0.3783784)
        20) guardian=father,mother 26   6 fail (0.7692308 0.2307692) *
        21) guardian=other 11   3 pass (0.2727273 0.7272727) *
      11) age< 16.5 13   3 pass (0.2307692 0.7692308) *
   3) failures< 0.5 312  78 pass (0.2500000 0.7500000)
     6) schoolsup=yes 40  18 pass (0.4500000 0.5500000)
      12) studytime>=1.5 31  14 fail (0.5483871 0.4516129)
        24) reason=course 11   2 fail (0.8181818 0.1818182) *
        25) reason=home,other,reputation 20   8 pass (0.4000000 0.6000000)
          50) Fedu< 2.5 7   2 fail (0.7142857 0.2857143) *
```

```
          51) Fedu>=2.5 13   3 pass (0.2307692 0.7692308) *
      13) studytime< 1.5 9   1 pass (0.1111111 0.8888889) *
     7) schoolsup=no 272  60 pass (0.2205882 0.7794118)
      14) guardian=other 10   4 fail (0.6000000 0.4000000) *
      15) guardian=father,mother 262  54 pass (0.2061069 0.7938931) *
$mlp
$mlp[[1]]
a 37-10-1 network with 391 weights
inputs: sexM age addressU famsizeLE3 PstatusT Medu Fedu Mjobhealth
   Mjobother Mjobservices Mjobteacher Fjobhealth Fjobother Fjobservices
   Fjobteacher reasonhome reasonother reasonreputation guardianmother
   guardianother traveltime studytime failures schoolsupyes famsupyes
   paidyes activitiesyes nurseryyes higheryes internetyes romanticyes
   famrel freetime goout Dalc Walc health
output(s): pass
options were - entropy fitting

$mlp[[2]]
a 37-10-1 network with 391 weights
inputs: sexM age addressU famsizeLE3 PstatusT Medu Fedu Mjobhealth
   Mjobother Mjobservices Mjobteacher Fjobhealth Fjobother Fjobservices
   Fjobteacher reasonhome reasonother reasonreputation guardianmother
   guardianother traveltime studytime failures schoolsupyes famsupyes
   paidyes activitiesyes nurseryyes higheryes internetyes romanticyes
   famrel freetime goout Dalc Walc health
output(s): pass
options were - entropy fitting

$mlp[[3]]
a 37-10-1 network with 391 weights
inputs: sexM age addressU famsizeLE3 PstatusT Medu Fedu Mjobhealth
   Mjobother Mjobservices Mjobteacher Fjobhealth Fjobother Fjobservices
   Fjobteacher reasonhome reasonother reasonreputation guardianmother
   guardianother traveltime studytime failures schoolsupyes famsupyes
   paidyes activitiesyes nurseryyes higheryes internetyes romanticyes
   famrel freetime goout Dalc Walc health
output(s): pass
options were - entropy fitting


$cx
 [1]  0.0000000 16.6962025  0.0000000  0.0000000  0.0000000  2.7493671
 [7]  2.5215190  0.0000000  0.0000000  0.0000000  0.0000000  1.4481013
[13]  2.0354430  0.3341772  0.0000000  0.0000000  0.0000000  0.0000000
[19]  0.0000000  0.0000000  0.0000000  0.0000000  3.9443038  3.2354430
[25]  3.1088608  1.4810127  2.2911392  3.5544304  0.0000000

$sx
 [1] 0.0000000 1.2760427 0.0000000 0.0000000 0.0000000 1.0947351
 [7] 1.0882005 0.0000000 0.0000000 0.0000000 0.0000000 0.6975048
[13] 0.8392403 0.7436510 0.0000000 0.0000000 0.0000000 0.0000000
[19] 0.0000000 0.0000000 0.0000000 0.0000000 0.8966586 0.9988620
[25] 1.1132782 0.8907414 1.2878966 1.3903034 0.0000000

$cy
[1] 0

$sy
```

```
[1] 0

$nr
[1] 3

Support Vector Machine object of class "ksvm"

SV type: C-svc  (classification)
 parameter : cost C = 1

Gaussian Radial Basis kernel function.
 Hyperparameter : sigma =  0.0367265122478637

Number of Support Vectors : 294

Objective Function Value : -206.4452
Training error : 0.217722
Probability model included.
[1] "save B3 to file"
[1] "load from file into B5"
[1] "nnet.formula" "nnet"
```

The `fit` function returns a model object, which contains several slots that are accessible using the `@` operator. The full list of slots can be easily accessed by using the `str` R function (e.g., `str(M1)`). In particular, the slot `@object` stores the fitted model, which is dependent of the selected `model` argument. For instance, `class(M1@object)` is `"rpart"`, `class(M2@object)` is "BinaryTree", `class(M2@object)`, while both `class(M3@object)` and `class(M4@object)` return a list. The first two models are white-box, i.e., they are often easy to be understood by humans, as shown in Figure 3.1. The last two models are black-box, i.e., they are more complex than the previous ones (Chapter 4 shows how to "open" these models using rminer). In the rminer implementation, these two models are lists because they can be made of several components or models. For instance, `M3` includes an ensemble of 3 multilayer perceptrons, where each neural network is stored in a vector list (e.g., `M3@object$mlp[[1]] contains the first multilayer perceptron, of class nnet.`). The support vector machine is accessible using `M4@object$svm` (e.g., `class(M4@object$svm` returns `"ksvm"`). The last code lines show how a `fit` model can be saved (`savemodel`) to and load from (`loadmodel`) a file.

3.1.2 Multiclass Classification

For the five-level classification, three classification models are adopted, namely bagging, boosting and random forest, as shown in file `3-math-2.R`:

```
### 3-math-2.R: modeling math multiclass classification example
### an external holdout is used: 2/3 random samples for training, 1/3 for
    testing
### bagging, boosting and randomForest machine learning algorithms
### this demonstration requires some computational effort

library(rminer)

# read previously saved file
math=read.table(file="math2.csv",header=TRUE,stringsAsFactors=TRUE)

# select inputs:
inputs=2:29 # select from 2 ("sex") to 29 ("health")
```

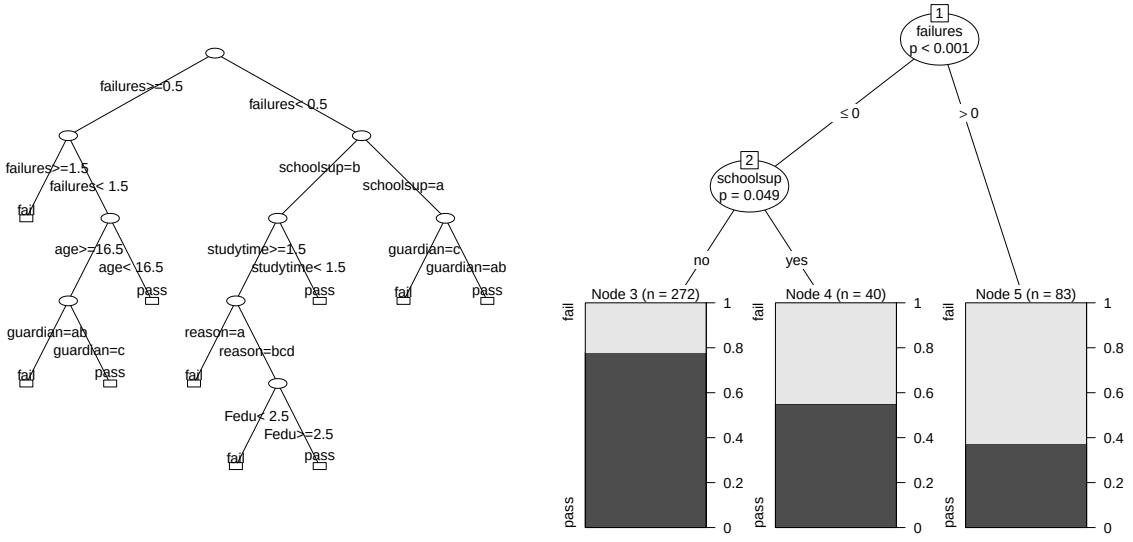

Figure 3.1: Modeled decision trees using rpart (left) and ctree (right) models.

```
# select outputs: multiclass task "five"
cout=which(names(math)=="five")
cmath=math[,c(inputs,cout)] # for easy typing, new data.frame
cat("output class:",class(cmath$five),"\n")


# auxiliary function:
showres=function(M,data,output)
{
 output=which(names(data)==output)
 Y=data[,output] # target values
 P=predict(M,data) # prediction values
 acc=round(mmetric(Y,P,metric="ACC"),2) # get accuracy
 cat(class(M@object),"> time elapsed:",M@time,", Global Accuracy:",acc,"\n"
     )
 cat("Acc. per class",round(mmetric(Y,P,metric="ACCLASS"),2),"\n")
}

# bagging example:
C1=fit(five~.,cmath,model="bagging") # bagging from adabag package
showres(C1,cmath,"five")

# boosting example:
C2=fit(five~.,cmath,model="boosting") # boosting from adabag package
showres(C2,cmath,"five")

# randomForest example:
C3=fit(five~.,cmath,model="randomForest") # from randomForest package
showres(C3,cmath,"five")
```

In this example, an auxiliary function is defined for showing the object class, time elapsed, overall classification accuracy (in %) and classification accuracy for each class ({A,B,C,D,F}). The classification metrics are achieved by using the `predict` and `mmetric` rminer functions. The

former function uses a fitted model and a `data.frame` for estimating the model predictions (see `help(predict.fit)` for more details), while the latter function uses the target and predicted values in order to compute the desired metrics (see `help(mmetric)`). The output is of executing the file is:

```
> source("3-math-2.R")
output class: factor
bagging > time elapsed: 30.22 , Global Accuracy: 74.94
Acc. per class 95.19 92.66 91.39 85.82 84.81
boosting > time elapsed: 30.635 , Global Accuracy: 74.43
Acc. per class 95.7 93.16 91.39 86.08 82.53
randomForest.formula randomForest > time elapsed: 0.995 , Global Accuracy:
   100
Acc. per class 100 100 100 100 100
```

In this example, the `randomForest` is the fastest fitting model, while also providing the best classification accuracy. However, it should be noted that accuracy on training data (as shown in this example) is not very meaningful, since complex models, such as random forests and others (e.g., neural networks, support vector machines) can easily fit to every training example, thus overfitting the data. In effect, the true predictive capability of a classifier should always be measured on unseen test data (e.g., use of an external holdout or k-fold cross-validation method), as shown in the Section 3.2.

3.2 Regression

For the regression demonstration, a random forest was selected (`model=randomForest"`), as provided in file `3-math-3.R`:

```
### 3-math-3.R: modeling math regression (G3) example
### an external holdout is used: 2/3 random samples for training, 1/3 for
   testing
### randomForest machine learning algorithm is adopted
### saves rf-1.pdf imp-1.pdf
### then an external 10-fold cross-validation is executed using "rpart" (
   decision tree = regression tree)
### saves rf-2.pdf

library(rminer)

# read previously saved file
math=read.table(file="math2.csv",header=TRUE,stringsAsFactors=TRUE)

# select inputs:
inputs=2:29 # select from 2 ("sex") to 29 ("health")

# select outputs: regression task
g3=which(names(math)=="G3")
cat("output class:",class(math[,g3]),"\n")


# fit holdout example:
H=holdout(math$G3,ratio=2/3,seed=12345)
print("holdout:")
print(summary(H))
R1=fit(G3~.,math[H$tr,c(inputs,g3)],model="randomForest")
```

```
# get predictions on test set (new data)
P1=predict(R1,math[H$ts,c(inputs,g3)])
# show scatter plot with quality of the predictions:
target1=math[H$ts,]$G3
e1=mmetric(target1,P1,metric=c("MAE","R22"))
error=paste("RF, holdout: MAE=",round(e1[1],2),", R2=",round(e1[2],2),sep="
    ")
pdf("rf-1.pdf")
mgraph(target1,P1,graph="RSC",Grid=10,main=error)
dev.off()
cat(error,"\n")

# RF example with k-fold cross-validation
print("10-fold:")
R2=crossvaldata(G3~.,math[,c(inputs,g3)],fit,predict,ngroup=10,seed=123,
    model="randomForest",task="reg")
P2=R2$cv.fit # k-fold predictions on full dataset
e2=mmetric(math$G3,P2,metric=c("MAE","R22"))
error2=paste("RF, 10-fold: MAE=",round(e2[1],2),", R2=",round(e2[2],2),sep=
    "")
pdf("rf-2.pdf")
mgraph(math$G3,P2,graph="RSC",Grid=10,main=error2)
dev.off()
cat(error2,"\n")
```

A better evaluation method is used in this example, by means of two distinct generalization validation schemes Kohavi (1995): a random holdout train/test split (using 2/3 of the data for training and 1/3 for testing); and a 10-fold cross-validation. It should be noted that while the code presented in this section explicitly performs the holdout and 10-fold validations, rminer provides a `mining` function that allows to perform several holdout or k-fold runs using a single line of code.

The `holdout` rminer function receives an output target variable and returns a list with training (with 263 examples, 2/3 of the data) and testing (132 instances, 1/3 of the data) indices. Such `H` list can then be used for fitting (`fit`) and testing (e.g., `predict`, `mmetric`) the model. By using additional optional arguments in `holdout()`, it is possible to produce other holdout variants, such as example order split, using a fixed random seed, use of stratification (for factor targets) and even a more sophisticated incremental or rolling windows validation (see `help(holdout)` and the `mode` argument description for full details and examples). Similarly, the rminer function `crossvaldata` executes a k-fold cross-validation, which is more robust than the holdout, although it requires a higher computation effort (around k times more). The function requires several arguments, including `fit` and `predict` functions, and a `task` type (e.g., it can be set to `task="reg"` for regression and `task="prob"` for classification). The function returns a list, where the element `$cv.fit` contains the k-fold predictions (use `help(crossvaldata)` for further details). This example also introduces the `mgraph` rminer function, which is capable of plotting several types of graph results. In this case, it produces a scatter plot (`graph="RSC"`). The previously explained rminer `mmetric()` is also used to compute two popular regression metrics, the mean absolute error (MAE) and the coefficient of determination (R^2).

The obtained output is:

```
output class: integer
[1] "holdout:"
    Length Class  Mode
tr  263    -none- numeric
itr   0    -none- NULL
```

```
val   0    -none- NULL
ts  132    -none- numeric
RF, holdout: MAE=3.31, R2=0.12
[1] "10-fold:"
RF, 10-fold: MAE=3.19, R2=0.13
```

This example clearly exemplifies the need for measuring predictive performance on test data. As shown in the obtained scatter plots (Figure 3.2), the quality of the predictions is not good. In effect, a large number of points are far from the diagonal line (in gray), which denotes the perfect forecast. Also, the R^2 values are close to zero and thus far away from the ideal model (R^2=1.0).

3.3 Model Parametrization

In most cases, data mining algorithms include parameters that need to be set by the user and that are often termed hyperparameters, to distinguish from the normal model parameters that are fit to the data. Yet, for the sake of simplicity, these will be called parameters in this document. This section explains some basic parameter configuration in rminer, for further details consult `help(fit,package=rminer)`.

When no additional parameters are used, rminer assumes a default parametrization. Such default corresponds to what is defined in the packages that implement the learning algorithms. For instance, by default: `ntree=500` for the `randomForest()` function of the randomForest package; and `mfinal=100` for the `bagging()` function of the adabag package. Any of these defaults can be changed by adding the new parameter values as arguments of the `fit` or `mining` functions (see examples from file `3-math-5.R`). Some R learning implementations do not have default values some some parameters, such as the number of hidden nodes (`size`) of the multilayer perceptron (`nnet` function and package). In such cases, the default is `search="heuristic"`, as explained in `help(fit,package=rminer)`.

The code file `3-math-5.R` shows examples of simple parameter adjustments:

```
### 3-math-5.R: model parameterization example
### sets hyperparameters for "rpart" and "ksvm"
### modeling example, all data is used to fit the models

library(rminer)

math=read.table(file="math2.csv",header=TRUE,stringsAsFactors=TRUE)

# select inputs and output (regression):
inputs=2:29; g3=which(names(math)=="G3")
rmath=math[,c(inputs,g3)]
# for simplicity, this code file assumes a fit to all math data:

print("examples that set some parameters to fixed values:")

print("mlp model with decay=0.1:")
R4=fit(G3~.,rmath,model="mlp",decay=0.1)
print(R4@mpar)

print("rpart with minsplit=10")
R5=fit(G3~.,rmath,model="rpart",control=rpart::rpart.control(minsplit=10))
print(R5@mpar)
print("rpart with minsplit=10 (simpler fit code)")
```

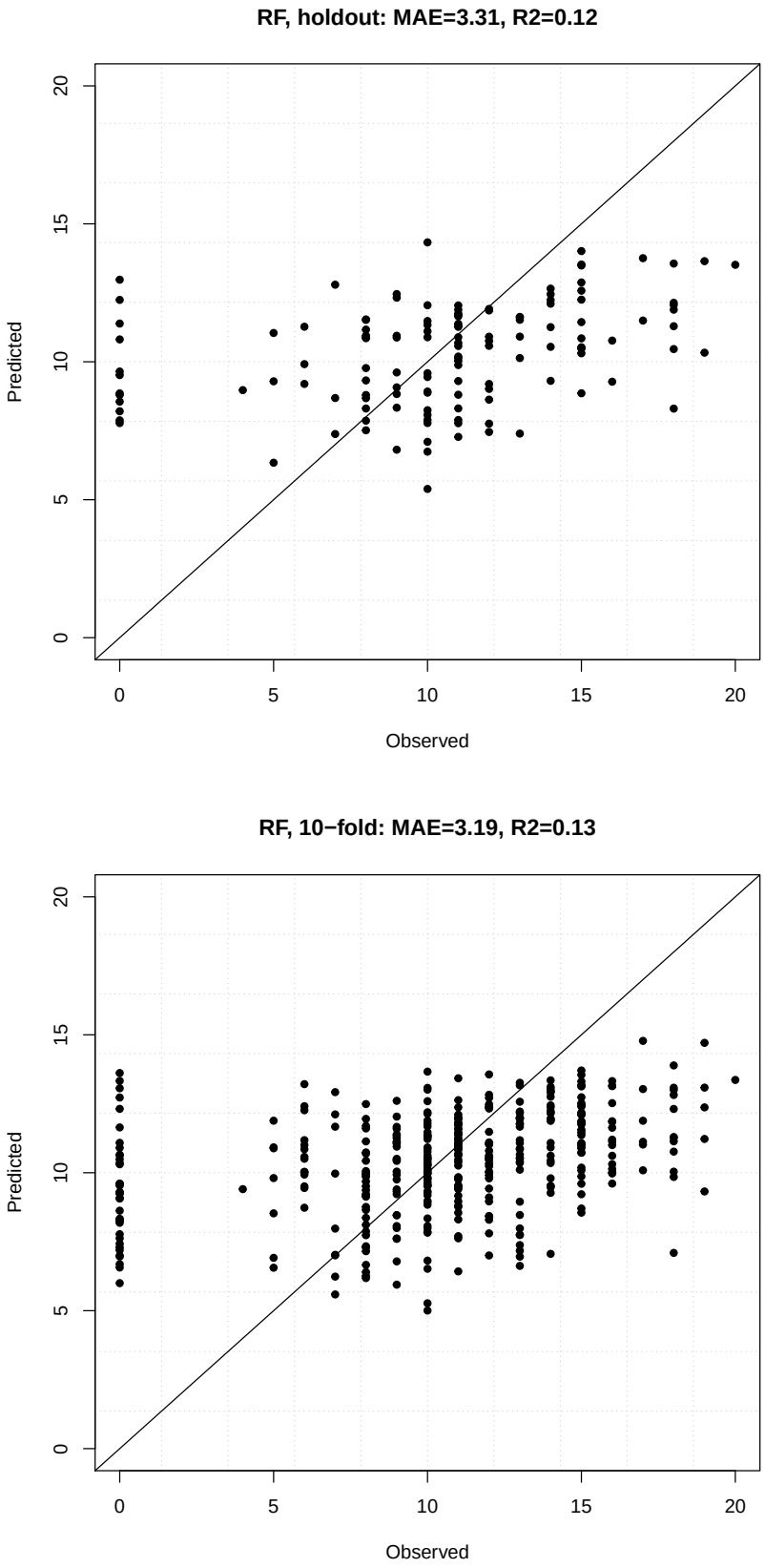

Figure 3.2: Scatter plot of randomForest (RF) predicted vs observed values using holdout (top) and 10-fold (bottom) evaluation methods.

```
R5b=fit(G3~.,rmath,model="rpart",control=list(minsplit=10))
print(R5b@mpar)

print("ksvm with kernel=vanilladot and C=10")
R6=fit(G3~.,rmath,model="ksvm",kernel="vanilladot",C=10)
print(R6@mpar)

print("ksvm with kernel=tanhdot, scale=2 and offset=2")
# fit already has a scale argument, thus the only way to fix scale of "
   tanhdot"
# is to use the special search argument with the "none" method:
s=list(smethod="none",search=list(scale=2,offset=2))
R7=fit(G3~.,rmath,model="ksvm",kernel="tanhdot",search=s)
print(R7@mpar)
```

The setting of parameters is highly dependent on the R function implementation and thus any change in these parameters should be performed by informed R data mining users, preferably after consulting the help of such R function or package. The obtained output of file `3-math-5.R` is:

```
> source("3-math-5.R")
[1] "examples that set some parameters to fixed values:"
[1] "mlp model with decay=0.1:"
$decay
[1] 0.1

$size
[1] 14

$task
[1] "reg"

$scale
[1] "all"

$type
[1] 1

$nr
[1] 3

[1] "rpart with minsplit=10"
$control
$control$minsplit
[1] 10

$control$minbucket
[1] 3

$control$cp
[1] 0.01

$control$maxcompete
[1] 4

$control$maxsurrogate
[1] 5
```

```
$control$usesurrogate
[1] 2

$control$surrogatestyle
[1] 0

$control$maxdepth
[1] 30

$control$xval
[1] 10


$method
[1] "anova"

[1] "rpart with minsplit=10 (simpler fit code)"
$control
$control$minsplit
[1] 10


$method
[1] "anova"

[1] "ksvm with kernel=vanilladot and C=10"
$kernel
[1] "vanilladot"

$C
[1] 10

$kpar
list()

$task
[1] "reg"

$epsilon
[1] 0.1

[1] "ksvm with kernel=tanhdot, scale=2 and offset=2"
$kernel
[1] "tanhdot"

$kpar
$kpar$scale
[1] 2

$kpar$offset
[1] 2


$task
[1] "reg"

$C
[1] 1
```

```
$epsilon
[1] 0.1
```

The parameters can have a strong impact in the model performance, as they can control for instance the model complexity or learning capability. When no a priori knowledge is available (which is often the case), the tuning of these parameters can be performed by using an internal validation method, such as holdout or k-fold, applied using only the training data. The internal test data, known as validation data, is used to select the parameter combination that provides the best generalization capability. By default and when needed, rminer assumes an internal random holdout split, with 2/3 for training and 1/3 for validation.

Non expert users can use an automatic search for these parameters by using the `search` argument of `fit` and `mining` rminer functions. After rminer version 1.4.1, the new `mparheuristic` function was introduced, which allows a simple definition of grid search values for some specific parameters and models. The currently default `search="heuristic"` is equivalent to the advised use of `search=list(search=mparheuristic(`*model*`))`, where *model* denotes a rminer model. For further details, please consult `help(mparheuristic)`.

For any search option that includes more than one search, rminer selects the parameter (or parameters) that provide the best metric value on the validation set. By default, rminer assumes the metric sum of absolute errors (`"SAE"`) for regression, global area of ROC curve for probabilistic classification ((`"AUC"`) and global accuracy for pure classification (`"ACC"`). Then, the model is refit using such parameter(s) and with all training data. The code file `3-math-6.R` exemplifies this easy `search` use:

```
### 3-math-6.R: hyperparameter search example (G3 regression task)
### randomForest machine learning algorithm
### uses all data for the fit (training data)
### last fit example explicitly uses an internal holdout for the grid
    search

library(rminer)

math=read.table(file="math2.csv",header=TRUE,stringsAsFactors=TRUE)

# select inputs and output (regression):
inputs=2:29
g3=which(names(math)=="G3")
cat("output class:",class(math[,g3]),"\n")
rmath=math[,c(inputs,g3)]
# for simplicity, this code file assumes a fit to all math data:
m=c("holdouto",2/3) # for internal validation: ordered holdout, 2/3 for
    training

# 10 searches for the mty randomForest parameter:
# after rminer 1.4.1, mparheuristic can be used:
s=list(search=mparheuristic("randomForest",n=10),method=m)
print("search values:")
print(s)
set.seed(123) # for replicability
R3=fit(G3~.,rmath,model="randomForest",search=s,fdebug=TRUE)
# show the automatically selected mtry value:
print(R3@mpar)

# same thing but with more verbose and using the full search parameter:
m=c("holdouto",2/3) # internal validation: ordered holdout, 2/3 for
```

```
   training
s=list(smethod="grid",search=list(mtry=1:10),convex=0,method=m,metric="SAE"
   )
set.seed(123) # for replicability
R3b=fit(G3~.,rmath,model="randomForest",search=s,fdebug=TRUE)
print(R3b@mpar)
```

The result of executing file `3-math-6.R` is:

```
> source("3-math-6.R")
output class: integer
[1] "search values:"
$search
$search$mtry
 [1]  1  2  3  4  5  6  7  8  9 10


$method
[1] "holdouto"          "0.666666666666667"

grid with: 10 searches (SAE values)
i: 1 eval: 441.3125 best: 441.3125
i: 2 eval: 452.7913 best: 441.3125
i: 3 eval: 464.2826 best: 441.3125
i: 4 eval: 470.2803 best: 441.3125
i: 5 eval: 470.8582 best: 441.3125
i: 6 eval: 474.8368 best: 441.3125
i: 7 eval: 472.4557 best: 441.3125
i: 8 eval: 478.4884 best: 441.3125
i: 9 eval: 478.3065 best: 441.3125
i: 10 eval: 480.4731 best: 441.3125
$mtry
[1] 1

$importance
[1] TRUE

grid with: 10 searches (SAE values)
i: 1 eval: 441.3125 best: 441.3125
i: 2 eval: 452.7913 best: 441.3125
i: 3 eval: 464.2826 best: 441.3125
i: 4 eval: 470.2803 best: 441.3125
i: 5 eval: 470.8582 best: 441.3125
i: 6 eval: 474.8368 best: 441.3125
i: 7 eval: 472.4557 best: 441.3125
i: 8 eval: 478.4884 best: 441.3125
i: 9 eval: 478.3065 best: 441.3125
i: 10 eval: 480.4731 best: 441.3125
$mtry
[1] 1

$importance
[1] TRUE
```

In this example, the `mtry` parameter of `randomForest` was automatically set to 1. The first fit performs an automatic search for the best `mtry` parameter of the `randomForest` method under an easy to use code. The second fit executes the same search but with a more explicit definition of the `search` argument. The verbose (optional argument of `fdebug=TRUE`) allows to see that the

value of `mtry=1` provides the lowest `"SAE"` metric value on the internal holdout validation set (in this case, using an ordered split). Also, the `set.seed` R command was used to warranty the same execution in both fits, since `randomForest` learning algorithm is stochastic.

For advanced users, the rminer package provides several types of internal parameter searches, such as

- `"matrix"` – assumes `search`$`search` contains several parameters (say p), each with n searches (thus it corresponds to a kind of a *matrix* of size $n \times p$));
- `"grid"` search – tests all combinations of several search parameters, each one changed according to a grid; and
- nested 2-Level grid (`"2L"`) – two levels of a grid search, where first level is set by $`search` and second level performs a fine tuning around the best first level value.

Such flexibility is obtained by setting the `search` argument as a list. More advanced parameter searches, such as use of evolutionary computation, are not currently available at rminer but can be implemented in R by installing other packages, as shown in (Cortez, 2021).

File `3-math-7.R` provides several examples of how to use the `search` argument for automatically searching for the best parameters:

```
### 3-math-7.R: provides several examples of how to use the search argument
### for automatically searching for the best hyperparameters (G3 task):
### an internal 3-fold cross-validation is used by the grid search
### fit assumes all data to fit the models
### examples with several machine learning models: "mlpe", "ksvm", "rpart"

library(rminer)
math=read.table(file="math2.csv",header=TRUE,stringsAsFactors=TRUE)

# select inputs and output (regression):
inputs=2:29; g3=which(names(math)=="G3")
rmath=math[,c(inputs,g3)]
# for simplicity, this code file assumes a fit to all math data:

mint=c("kfold",3,123) # internal 3-fold, same seed

print("more sophisticated examples for setting hyperparameters:")

cat("mlpe model, grid for hidden nodes (size):",seq(0,8,2),"\n")
s=list(smethod="grid",search=list(size=seq(0,8,2)),method=mint,convex=0)
R9=fit(G3~.,rmath,model="mlpe",decay=0.1,maxit=25,nr=5,search=s,fdebug=TRUE
    )
print(R9@mpar)

cat("mlpe model, same grid using mparheuristic function:",seq(0,8,2),"\n")
s=list(search=mparheuristic("mlpe",lower=0,upper=8,by=2),method=mint)
R9b=fit(G3~.,rmath,model="mlpe",decay=0.1,maxit=25,nr=5,search=s,fdebug=
    TRUE)
print(R9b@mpar)

cat("mlpe model, grid for hidden nodes:",1:2,"x decay:",c(0,0.1),"\n")
s=list(smethod="grid",search=list(size=1:2,decay=c(0,0.1)),method=mint,
    convex=0)
R9c=fit(G3~.,rmath,model="mlpe",maxit=25,search=s,fdebug=TRUE)
print(R9c@mpar)
```

```
cat("mlpe model, same search but with matrix method:\n")
s=list(smethod="matrix",search=list(size=rep(1:2,times=2),decay=rep(c
    (0,0.1),each=2)),method=mint,convex=0)
R9d=fit(G3~.,rmath,model="mlpe",maxit=25,search=s,fdebug=TRUE)
print(R9d@mpar)

# 2 level grid with total of 8 searches
#  note of caution: some "2L" ranges may lead to non integer (e.g. 1.3)
    values at
#  the 2nd level search. And some R functions crash if non integer values
    are used for
#  integer parameters.
cat("mlpe model, 2L search for size:\n")
s=list(smethod="2L",search=list(size=c(4,8,12,16)),method=mint,convex=0)
R9d=fit(G3~.,rmath,model="mlpe",maxit=25,search=s,fdebug=TRUE)
print(R9d@mpar)

print("ksvm with kernel=rbfdot: sigma, C and epsion (3^3=27 searches):")
s=list(smethod="grid",search=list(sigma=2^c(-8,-4,0),C=2^c(-1,2,5),epsilon
    =2^c(-9,-5,-1)),method=mint,convex=0)
R10=fit(G3~.,rmath,model="ksvm",kernel="rbfdot",search=s,fdebug=TRUE)
print(R10@mpar)

# even rpart or ctree parameters can be searched:
# example with rpart and cp:
print("rpart with control= cp in 10 values in 0.01 to 0.18 (10 searches):")
s=list(search=mparheuristic("rpart",n=10,lower=0.01,upper=0.18),method=mint
    )
R11=fit(G3~.,rmath,model="rpart",search=s,fdebug=TRUE)
print(R11@mpar)

# same thing, but with more explicit code that can be adapted for
# other rpart arguments, since mparheuristic only works for cp:
# a vector list needs to be used for the search$search parameter
print("rpart with control= cp in 10 values in 0.01 to 0.18 (10 searches):")
# a vector list needs to be used for putting 10 cp values
lcp=vector("list",10) # 10 grid values for the complexity cp
names(lcp)=rep("cp",10) # same cp name
scp=seq(0.01,0.18,length.out=10) # 10 values from 0.01 to 0.18
for(i in 1:10) lcp[[i]]=scp[i] # cycle needed due to [[]] notation
s=list(smethod="grid",search=list(control=lcp),method=mint,convex=0)
R11b=fit(G3~.,rmath,model="rpart",search=s,fdebug=TRUE)
print(R11b@mpar)

# check ?rminer::fit for further examples
```

After executing file **3-math-7.R**, one output example is (results might change since **"mlpe"** is a stochastic method):

```
> source("3-math-7.R")
[1] "more sophisticated examples for setting hyperparameters:"
mlpe model, grid for hidden nodes (size): 0 2 4 6 8
grid with: 5 searches (SAE values)
i: 1 eval: 1322.123 best: 1322.123
i: 2 eval: 1472.887 best: 1322.123
i: 3 eval: 1542.438 best: 1322.123
i: 4 eval: 1567.747 best: 1322.123
i: 5 eval: 1744.021 best: 1322.123
```

```
$decay
[1] 0.1

$maxit
[1] 25

$nr
[1] 3

$size
[1] 0

$task
[1] "reg"

$scale
[1] "all"

$type
[1] 2

mlpe model, same grid using mparheuristic function: 0 2 4 6 8
grid with: 5 searches (SAE values)
i: 1 eval: 1322.123 best: 1322.123
i: 2 eval: 1387.089 best: 1322.123
i: 3 eval: 1734.731 best: 1322.123
i: 4 eval: 1552.737 best: 1322.123
i: 5 eval: 1690.302 best: 1322.123
$decay
[1] 0.1

$maxit
[1] 25

$nr
[1] 3

$size
[1] 0

$task
[1] "reg"

$scale
[1] "all"

$type
[1] 2

mlpe model, grid for hidden nodes: 1 2 x decay: 0 0.1
grid with: 4 searches (SAE values)
i: 1 eval: 1409.903 best: 1409.903
i: 2 eval: 1377.669 best: 1377.669
i: 3 eval: 1383.751 best: 1377.669
i: 4 eval: 1391.683 best: 1377.669
$maxit
[1] 25
```

```
$size
[1] 2

$decay
[1] 0

$task
[1] "reg"

$scale
[1] "all"

$type
[1] 2

$nr
[1] 3

mlpe model, same search but with matrix method:
matrix with: 4 searches (SAE values)
i: 1 eval: 1371.93 best: 1371.93
i: 2 eval: 1413.567 best: 1371.93
i: 3 eval: 1390.079 best: 1371.93
i: 4 eval: 1397.966 best: 1371.93
$maxit
[1] 25

$size
[1] 1

$decay
[1] 0

$task
[1] "reg"

$scale
[1] "all"

$type
[1] 2

$nr
[1] 3

mlpe model, 2L search for size:
[1] " 1st level:"
2L with: 4 searches (SAE values)
i: 1 eval: 1518.898 best: 1518.898
i: 2 eval: 1799.608 best: 1518.898
i: 3 eval: 1789.155 best: 1518.898
i: 4 eval: 1753.907 best: 1518.898
[1] " 2nd level:"
2L with: 4 searches (SAE values)
i: 1 eval: 1371.975 best: 1371.975
i: 2 eval: 1508.179 best: 1371.975
i: 3 eval: 1569.905 best: 1371.975
i: 4 eval: 1629.394 best: 1371.975
```

```
end eval best: 1371.975
$maxit
[1] 25

$size
[1] 1

$task
[1] "reg"

$scale
[1] "all"

$type
[1] 2

$nr
[1] 3

[1] "ksvm with kernel=rbfdot: sigma, C and epsion (3^3=27 searches):"
grid with: 27 searches (SAE values)
i: 1 eval: 1261.186 best: 1261.186
i: 2 eval: 1246.095 best: 1246.095
i: 3 eval: 1351.187 best: 1246.095
i: 4 eval: 1240.226 best: 1240.226
i: 5 eval: 1379.571 best: 1240.226
i: 6 eval: 1362.906 best: 1240.226
i: 7 eval: 1298.342 best: 1240.226
i: 8 eval: 1448.426 best: 1240.226
i: 9 eval: 1362.906 best: 1240.226
i: 10 eval: 1261.891 best: 1240.226
i: 11 eval: 1246.106 best: 1240.226
i: 12 eval: 1352.071 best: 1240.226
i: 13 eval: 1243.662 best: 1240.226
i: 14 eval: 1379.525 best: 1240.226
i: 15 eval: 1362.913 best: 1240.226
i: 16 eval: 1302.973 best: 1240.226
i: 17 eval: 1443.571 best: 1240.226
i: 18 eval: 1362.913 best: 1240.226
i: 19 eval: 1266.911 best: 1240.226
i: 20 eval: 1260.116 best: 1240.226
i: 21 eval: 1350.944 best: 1240.226
i: 22 eval: 1254.508 best: 1240.226
i: 23 eval: 1376.628 best: 1240.226
i: 24 eval: 1385.42 best: 1240.226
i: 25 eval: 1315.634 best: 1240.226
i: 26 eval: 1392.545 best: 1240.226
i: 27 eval: 1385.42 best: 1240.226
$kernel
[1] "rbfdot"

$kpar
$kpar$sigma
[1] 0.00390625


$C
[1] 4
```

```
$epsilon
[1] 0.001953125

$task
[1] "reg"

[1] "rpart with control= cp in 10 values in 0.01 to 0.18 (10 searches):"
grid with: 10 searches (SAE values)
i: 1 eval: 1439.228 best: 1439.228
i: 2 eval: 1324.147 best: 1324.147
i: 3 eval: 1296.989 best: 1296.989
i: 4 eval: 1296.989 best: 1296.989
i: 5 eval: 1296.989 best: 1296.989
i: 6 eval: 1296.989 best: 1296.989
i: 7 eval: 1317.588 best: 1296.989
i: 8 eval: 1360.909 best: 1296.989
i: 9 eval: 1362.891 best: 1296.989
i: 10 eval: 1362.891 best: 1296.989
$control
$control$cp
[1] 0.04777778


$method
[1] "anova"

[1] "rpart with control= cp in 10 values in 0.01 to 0.18 (10 searches):"
grid with: 10 searches (SAE values)
i: 1 eval: 1439.228 best: 1439.228
i: 2 eval: 1324.147 best: 1324.147
i: 3 eval: 1296.989 best: 1296.989
i: 4 eval: 1296.989 best: 1296.989
i: 5 eval: 1296.989 best: 1296.989
i: 6 eval: 1296.989 best: 1296.989
i: 7 eval: 1317.588 best: 1296.989
i: 8 eval: 1360.909 best: 1296.989
i: 9 eval: 1362.891 best: 1296.989
i: 10 eval: 1362.891 best: 1296.989
$control
$control$cp
[1] 0.04777778


$method
[1] "anova"
```

In all examples, `fdebug=TRUE` was added to the `fit` for proving more verbose. In real-world practical examples, this argument should be omitted for presenting a more clear and short console output. Please check the `help(fit,package=rminer)` for more details and examples of how to use the powerful **search** argument.

3.4 Automated Machine Learning

Since its 1.4.4 version, the rminer package can perform an automated machine learning (AutoML) (Ferreira et al., 2021), which adopts a grid search to automatically select the best machine

learning (ML) algorithm (and its hyperparameters) among several possibilities. The advantage of AutoML is that no expert ML modeling knowledge is needed. However, it should be noted that in some cases (e.g., large datasets) the computational effort is high, requiring a substantial fit time. This section briefly exemplifies the rminer AutoML execution. Further details are available by accessing the help of functions `fit` and `mparheuristic` (e.g., `help(mparheuristic)`).

By default, rminer provides 3 AutoML modes (argument `model` of function `mparheuristic`):

- `"automl"` - executes a simple search with 5 ML algorithms using default hyperparameters: generalized linear model (GLM, via `cv.glmnet`), support vector machine (SVM, via `ksvm`), multilayer perceptron (MLP, via `mlpe`), random forest (RF, via `randomForest`) and extreme gradient boosting (XG, via `xgboost`).
- `"automl2"` – similar to previous mode except that now 10 (or 13 for SVM) hyperparameter searches are performed for each ML algorithm;
- `"automl3"` – similar to `"automl2"` except that includes an extra stacking ensemble (`model="SE"`) in the search and that uses the 5 best tuned ML algorithms (one for each family type).

A demonstration of the simplest AutoML mode (`"automl"`) is provided in file `3-math-9.R`:

```
### 3-math-8.R: Automated Machine Learning (AutoML) for G3:
### for the sake of simplicity, the fit assumes all data to fit the models

library(rminer)
math=read.table(file="math2.csv",header=TRUE,stringsAsFactors=TRUE)

# select inputs and output (regression):
inputs=2:29; g3=which(names(math)=="G3")
rmath=math[,c(inputs,g3)]
task="reg" # regression
metric="MAE" # mean absolute error

# this code file assumes a fit to all math data:
mint=c("kfold",3,123) # internal 3-fold, same seed
# 5 machine learning (ML) algorithms, 1 heuristic hyperparameter per
   algorithm:
sm=mparheuristic(model="automl",task=task,inputs=inputs)
search=list(search=sm,smethod="auto",method=mint,metric=metric,convex=0)
M=fit(G3~.,rmath,model="auto",search=search,fdebug=TRUE)
P=predict(M,rmath) # in this example, predictions for training data
# show leaderboard:
cat("> leaderboard models:",M@mpar$LB$model,"\n")
cat(">  validation values:",round(M@mpar$LB$eval,4),"\n")
cat("best model is:",M@model,"\n")
cat(metric,"=",round(mmetric(rmath$G3,P,metric=metric),2),"\n")

# check ?rminer::fit and ?rminer::mparheuristic for further examples
```

An execution example of the code is:

```
> source("3-math-8.R")
auto with: 5 models (MAE values)
grid with: 1 searches (MAE values)
i: 1 eval: 3.314423 best: 3.314423
m: 1 model: cv.glmnet bmodel: cv.glmnet eval: 3.314423 best: 3.314423 time:
    0.417
grid with: 1 searches (MAE values)
```

```
i: 1 eval: 3.182032 best: 3.182032
m: 2 model: ksvm bmodel: ksvm eval: 3.182032 best: 3.182032 time: 0.569
grid with: 1 searches (MAE values)
i: 1 eval: 4.601473 best: 4.601473
m: 3 model: mlpe bmodel: ksvm eval: 4.601473 best: 3.182032 time: 3.653
grid with: 1 searches (MAE values)
i: 1 eval: 3.273599 best: 3.273599
m: 4 model: randomForest bmodel: ksvm eval: 3.273599 best: 3.182032 time:
    7.802
grid with: 1 searches (MAE values)
i: 1 eval: 5.972064 best: 5.972064
m: 5 model: xgboost bmodel: ksvm eval: 5.972064 best: 3.182032 time: 7.887
 >> best: 5 model: ksvm best: 3.182032
> leaderboard models: ksvm randomForest cv.glmnet mlpe xgboost
>  validation values: 3.182 3.2736 3.3144 4.6015 5.9721
best model is: ksvm
MAE = 2.13
```

In this execution, the best validation error, the lowest mean absolute error (MAE), was provided by the `ksvm` model. The second best model was `randomForest`, followed by `cv.glmnet`, `mlpe` and `xgboost`. Then the selected model is trained with all data, the returned MAE is 2.13.

Chapter 4

Evaluation

The rminer package includes a large range of evaluation metrics and graphs that can be used to evaluate the quality of the fitted models and extract knowledge learned from the data-driven models. The metrics and graphs can be obtained by using the `mmetric` and `mgraph()` functions, as exemplified in the next subsections. Other examples are available in the help (e.g., `help(mmetric)`; `help(mgraph)`). The `mmetric` and `mgraph` functions compute several metrics or graphs once they receive: `y` - target variable, `x` – predictions; or only `y` – the result of the `mining` function or a vector list with `mining()` results. Other useful rminer functions are: `Importance()`, which allows the extraction of knowledge from fitted models in terms of input importance and average input effect; and `mining()`, which executes several fit and predict runs according to a user defined external validation scheme. In particular, the `Importance()` function provides an explainable artificial intelligence (XAI) by means of an sensitivity analysis (Cortez and Embrechts, 2013).

4.1 Classification

4.1.1 Binary Classification

The code in `eval-1.R` shows some simple binary classification evaluations:

```
### 4-eval-1.R: math binary classification
### fit using all data
### predict using all data
### saves roc-1.pdf and lift-1.pdf

library(rminer)
# read previously saved file
math=read.table(file="math2.csv",header=TRUE,stringsAsFactors=TRUE)

# select inputs:
inputs=2:29 # select from 2 ("sex") to 29 ("health")
# select outputs: binary task "pass"
bout=which(names(math)=="pass")
cat("output class:",class(math[,bout]),"\n")
bmath=math[,c(inputs,bout)] # for easy use
y=bmath$pass # target

# fit rpart to all data, pure class modeling (no probabilities)
B1=fit(pass~.,bmath,model="rpart",task="class") # fit a decision tree
P1=predict(B1,bmath) # class predictions
```

```
print(P1[1]) # show 1st prediction
m=mmetric(y,P1,metric=c("ACC","ACCLASS"))
print(m) # accuracy, accuracy per class
m=mmetric(y,P1,metric=c("CONF")) # a)
print(m$conf) # confusion matrix
m=mmetric(y,P1,metric=c("ALL"))
print(round(m,1)) # all pure class metrics

# fit rpart to all data, default probabilistic modeling
B2=fit(pass~.,bmath,model="rpart",task="prob") # fit a decision tree
P2=predict(B2,bmath) # predicted probabilities
print(P2[1,]) # show 1st prediction
m=mmetric(y,P2,metric=c("ACC"),TC=2,D=0.5)
print(m) # accuracy, accuracy per class
m=mmetric(y,P2,metric=c("CONF"),TC=2,D=0.1) # equal to a)
print(m$conf) # confusion matrix
m=mmetric(y,P2,metric=c("AUC","AUCCLASS"))
print(m) # AUC, AUC per class
m=mmetric(y,P2,metric=c("ALL"))
print(round(m,1)) # all prob metrics

# ROC and LIFT curve:
txt=paste(levels(y)[2],"AUC:",round(mmetric(y,P2,metric="AUC",TC=2),2))
mgraph(y,P2,graph="ROC",baseline=TRUE,Grid=10,main=txt,TC=2,PDF="roc-1")
txt=paste(levels(y)[2],"ALIFT:",round(mmetric(y,P2,metric="ALIFT",TC=2),2))
mgraph(y,P2,graph="LIFT",baseline=TRUE,Grid=10,main=txt,TC=2,PDF="lift-1")
```

The obtained result is:

```
> source("4-eval-1.R")
output class: factor
[1] fail
Levels: fail pass
     ACC ACCLASS1 ACCLASS2
78.48101 78.48101 78.48101
      pred
target fail pass
  fail   66   64
  pass   21  244
       ACC          CE         BER       KAPPA     CRAMERV    ACCLASS1
          ACCLASS2   BAL_ACC1   BAL_ACC2        TPR1        TPR2
      78.5        21.5        28.6        46.8         0.5        78.5
         78.5        71.4        71.4        50.8        92.1
      TNR1        TNR2 PRECISION1 PRECISION2         F11         F12
         MCC1        MCC2
      92.1        50.8        75.9        79.2        60.8        85.2
         0.6         0.6
     fail      pass
0.8181818 0.1818182
[1] 78.48101
       pred
target  FALSE TRUE
  FALSE     0  130
  TRUE      0  265
      AUC AUCCLASS1 AUCCLASS2
0.7266183 0.7266183 0.7266183
        ACC          CE         BER       KAPPA     CRAMERV    ACCLASS1
           ACCLASS2    BAL_ACC1    BAL_ACC2        TPR1
       78.5        21.5        28.6        46.8         0.5        78.5
```

```
                78.5        71.4        71.4        50.8
       TPR2      TNR1        TNR2  PRECISION1  PRECISION2          F11
                F12        MCC1        MCC2       BRIER
       92.1      92.1        50.8        75.9        79.2        60.8
                85.2         0.6         0.6         0.2
BRIERCLASS1 BRIERCLASS2         AUC   AUCCLASS1   AUCCLASS2        NAUC
   TPRATFPR       ALIFT      NALIFT ALIFTATPERC
        0.2         0.2         0.7         0.7         0.7         0.7
                    1.0         0.6         0.6         1.0
```

As previously explained, using probabilities allows for more flexibility when deciding if a class is positive or not. This is controlled in rminer by setting the arguments: `D` - acceptance threshold and `TC` – target class. In this example, each `mgraph` function creates a PDF file, since the `PDF` argument was used. Both graphs are shown in Figure 4.1. For a better understanding of the metrics computed, readers should consult `help(mmetric)` and (Witten et al., 2011). The ROC curve suggests a predictive model that is much better than the random classifier baseline (AUC=0.7 vs AUC=0.5). The code also creates the accumulated LIFT curve, which is often used in the marketing domain (Moro et al., 2014). The LIFT graph shows that it is possible to get 60% of the `"A"` math grades by selecting 50% of the students with the highest predictive probabilities. Some caution needs to be used when analyzing these results, since as previously explained the generalization capability of a classifier should always be conducted using test and not training data (training data was used for the sake of simplicity of code in this example).

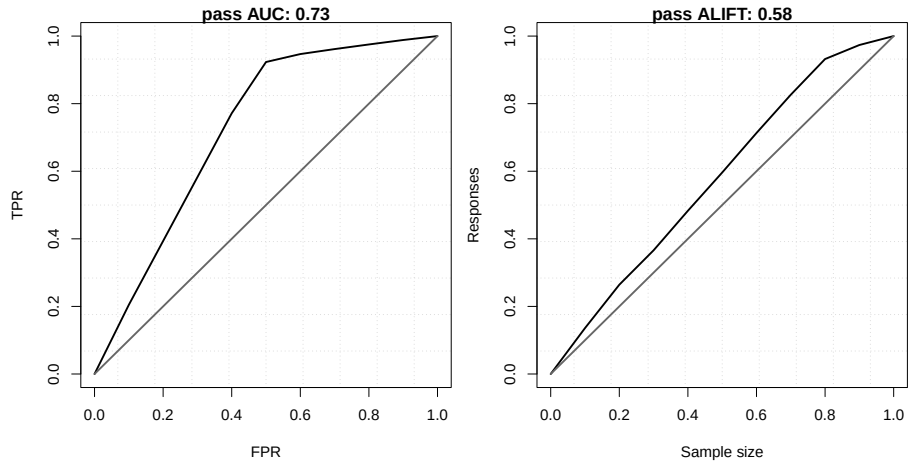

Figure 4.1: ROC (left) and acummulated LIFT (right) curves for the `"pass"` class.

4.1.2 Multiclass Classification

In this example, an external holdout is used, by using function `holdout()`, in order to compute some multiclass metrics and graphs (file `4-eval-2.R`):

```
### 4-eval-2.R: math multiclass classification
### external holdout (2/3 for training, 1/3 for testing)
### randomForest machine learning
### saves roc-2.pdf, imp-1.pdf and vec-1.pdf

library(rminer)
# read previously saved file
math=read.table(file="math2.csv",header=TRUE,stringsAsFactors=TRUE)
```

```
inputs=1:32 # all except G3 and pass
fout=which(names(math)=="five")
cat("output class:",class(math[,fout]),"\n")
cmath=math[,c(inputs,fout)] # for easy use
H=holdout(cmath$five,2/3,123)
y=cmath[H$ts,]$five # target

# simple fit of randomForest
C1=fit(five~.,cmath[H$tr,],model="randomForest")
P1=predict(C1,cmath[H$ts,]) # class predictions
print(P1[1,]) # show 1st prediction
m=mmetric(y,P1,metric=c("AUC","AUCCLASS"))
print(m) # global AUC, AUC per class
m=mmetric(y,P1,metric=c("CONF")) # a)
print(m$conf) # confusion matrix
m=mmetric(y,P1,metric=c("ALL"))
print(round(m,1)) # all prob. metrics

# ROC curve for class "A"
TC=1
txt=paste("class",levels(y)[TC],"AUC:",round(mmetric(y,P1,metric="AUC",TC=
    TC),2))
mgraph(y,P1,graph="ROC",baseline=TRUE,Grid=10,main=txt,TC=2,PDF="roc-2")

I=Importance(C1,cmath[H$tr,])
print(round(I$imp,digits=2))
imax=which.max(I$imp)

L=list(runs=1,sen=t(I$imp),sresponses=I$sresponses) # create a simple
    mining list
par(mar=c(2.0,2.0,2.0,2.0)) # enlarge PDF margin
mgraph(L,graph="IMP",leg=names(cmath),col="gray",Grid=10,PDF="imp-1")
txt=paste("VEC curve for",names(cmath)[imax],"influence on class",levels(y)
    [TC])
mgraph(L,graph="VEC",xval=imax,Grid=10,data=cmath[H$tr,],TC=1,main=txt,PDF=
    "vec-1")
```

The output of executing file `4-eval-2.R` is:

```
> source("4-eval-2.R")
output class: factor
    A     B     C     D     F
0.016 0.026 0.022 0.134 0.802
      AUC AUCCLASS1 AUCCLASS2 AUCCLASS3 AUCCLASS4 AUCCLASS5
0.9372570 0.9882660 0.9299550 0.9199134 0.8908429 0.9704017
      pred
target  A  B  C  D  F
     A  8  5  0  0  0
     B  0 16  4  0  0
     C  0  3 10  8  0
     D  0  1  2 23  8
     F  0  0  0  4 39
              ACC                CE               BER              KAPPA
                             CRAMERV          ACCLASS1
             73.3                26.7              30.5              64.8
                               0.7              96.2
         ACCLASS2          ACCLASS3          ACCLASS4          ACCLASS5
                      BAL_ACC1          BAL_ACC2
```

```
            90.1              87.0              82.4              90.8
                             80.8              85.9
       BAL_ACC3          BAL_ACC4          BAL_ACC5              TPR1
                        TPR2              TPR3
            71.1              77.6              90.8              61.5
                             80.0              47.6
            TPR4              TPR5              TNR1              TNR2
                             TNR3              TNR4
            67.6              90.7             100.0              91.9
                             94.5              87.6
            TNR5        PRECISION1        PRECISION2        PRECISION3
                       PRECISION4        PRECISION5
            90.9             100.0              64.0              62.5
                             65.7              83.0
             F11               F12               F13               F14
                              F15              MCC1
            76.2              71.1              54.1              66.7
                             86.7               0.8
            MCC2              MCC3              MCC4              MCC5
                            BRIER       BRIERCLASS1
             0.7               0.5               0.6               0.8
                              0.1               0.0
     BRIERCLASS2       BRIERCLASS3       BRIERCLASS4       BRIERCLASS5
                      AUC          AUCCLASS1
             0.1               0.1               0.1               0.1
                              0.9               1.0
       AUCCLASS2         AUCCLASS3         AUCCLASS4         AUCCLASS5
                        NAUC          TPRATFPR
             0.9               0.9               0.9               1.0
                              1.0               1.0
          ALIFT            NALIFT       ALIFTATPERC          macroTNR
                       microTNR       weightedTNR
             0.8               0.8               1.0              93.0
                             93.3              91.7
  macroPRECISION weightedPRECISION          macroTPR       weightedTPR
                macroF1        weightedF1
            75.0              74.0              69.5              73.3
                             70.9              72.8
        macroACC       weightedACC
            89.3              88.5
 [1] 0.00 0.01 0.01 0.01 0.03 0.02 0.02 0.02 0.03 0.03 0.01 0.01 0.01 0.02
    0.05 0.02 0.01 0.01 0.00 0.00 0.06 0.02 0.05
[24] 0.02 0.04 0.03 0.03 0.01 0.02 0.01 0.14 0.23 0.00
```

In the example, all `math2.csv` inputs are fed into the random forest, including the previous trimester grades (`G2` and `G1`) that are correlated with the final grade **G3**. Thus, high quality prediction results are achieved. For instance, the global AUC value is 0.95 and the AUC for class `"A"` is 0.96, which corresponds to a high quality discrimination, as shown by the ROC curve of Figure 4.2. The rminer `Importance` function uses a sensitivity analysis method for extracting input relevance and variable effect characteristic (VEC) curves. The `Importance` function can be applied to virtually any supervised learning method, thus being useful for opening black-box models. This usage is detailed in (Cortez and Embrechts, 2013), where several XAI sensitivity analysis methods and visualization techniques are also discussed, such as input pair relevance and overall effect on the output (e.g., via VEC surface and contour plots). For some additional code examples, please check `help(Importance)`. The result of `Importance` is put in a simple mining list, such that it can be used by `mgraph()`. The left of Figure 4.3 shows a barplot with the

input relevance values, confirming the importance of `G2` and `G1` grades. In the right of Figure 4.3, the VEC curve for `G2` corresponds to the solid line, while the gray bars denote the `G2` histogram. This plot allows to verify that high `G2` values are less frequent. Also, a high `G2` increases the probability for the `"A"` by more than 0.15 points.

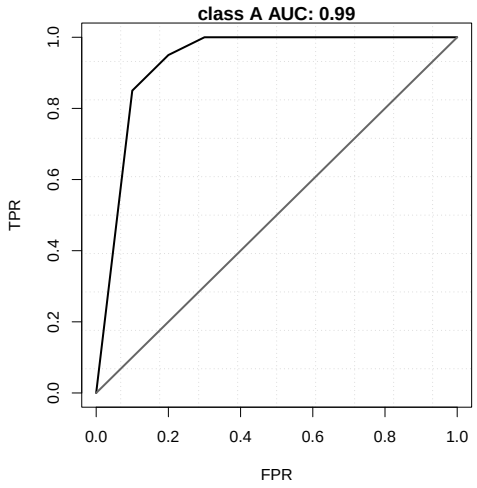

Figure 4.2: ROC curve for class `"A"`.

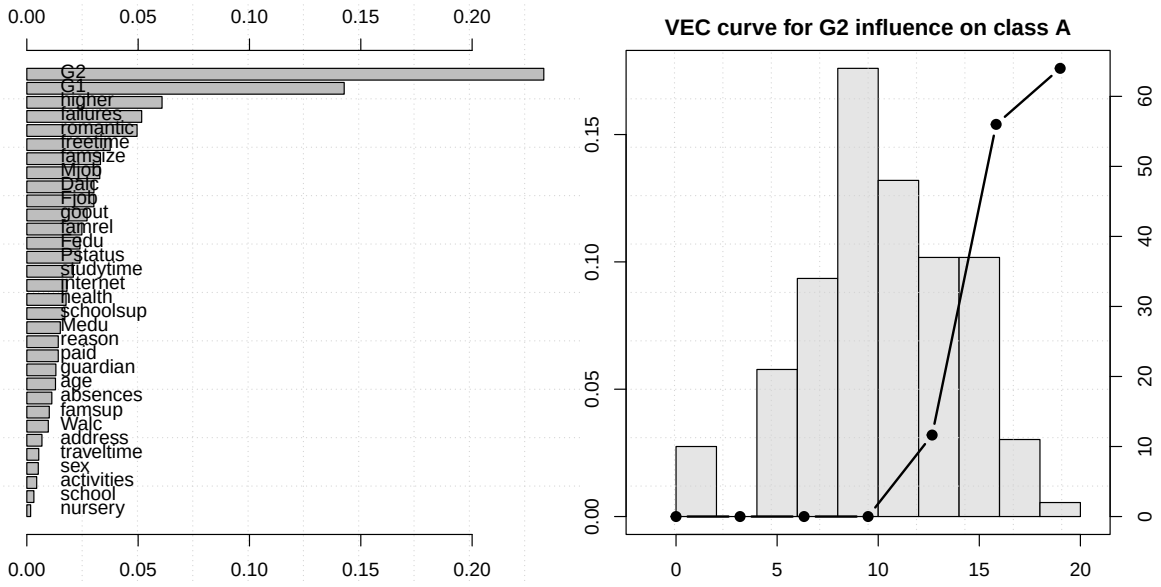

Figure 4.3: Input importance (left, in %) and Variable Effect Curve for the `G2` input average effect on class `"A"`.

4.2 Regression

The powerful `mining` function will be adopted in the regression evaluation example. This function executes several fit and predict runs under a user defined external validation method, returning a mining list with useful execution indicators. In order to reduce the usage of memory,

the function does not store individual fit models, since for some external validation methods and selected number of runs, the total number of fitted models can be high. More details are given in: **help(mining)**. The regression example is provided in file **4-eval-3.R**:

```
### 4-eval3.R: demonstration of mining function for regression (G3)
### 20 runs x external 3-fold: requires some computation
### 2 ML algorithms: randomForest vs mr
### saves rsc-1.pdf and rec-1.pdf

library(rminer)
math=read.table(file="math2.csv",header=TRUE,stringsAsFactors=TRUE)

inputs=1:32 # all except pass and five
g3=which(names(math)=="G3")
cat("output class:",class(math[,g3]),"\n")
rmath=math[,c(inputs,g3)] # for easy use
y=rmath$g3 # target

# mining for randomForest, external 3-fold, 20 Runs (=60 fitted models)
M1=mining(G3~.,rmath,model="randomForest",method=c("kfold",3,123),Runs=20)
m=mmetric(M1,metric=c("MAE","RMSE")) # 2 metrics:
print(m) # show metrics for each run
mi=meanint(m[,1])
cat("RF MAE values:",round(mi$mean,2),"+-",round(mi$int,2),"\n")

# regression scatter plot:
txt=paste("G3 MAE:",round(mi$mean,2))
mgraph(M1,graph="RSC",Grid=10,main=txt,PDF="rsc-1")

# REC curve, comparison with multiple regression: "mr":
M2=mining(G3~.,rmath,model="mr",method=c("kfold",3,123),Runs=20)
L=vector("list",2) # list of minings
L[[1]]=M1
L[[2]]=M2
mgraph(L,graph="REC",leg=c("randomForest","mr"),main="REC curve",xval=10,
   PDF="rec-1")
```

The obtained result is:

```
> source("4-eval-3.R")
output class: integer
        MAE     RMSE
1  1.191093 1.815548
2  1.203746 1.823500
3  1.238816 1.875031
4  1.278862 1.980725
5  1.205809 1.852967
6  1.192323 1.806255
7  1.205364 1.829047
8  1.203903 1.828953
9  1.243389 1.882187
10 1.208198 1.813607
11 1.174015 1.769573
12 1.230345 1.860536
13 1.193067 1.790573
14 1.206698 1.827090
15 1.179501 1.805789
16 1.211689 1.837586
17 1.199138 1.822167
```

```
18 1.229448 1.865620
19 1.194857 1.785262
20 1.250204 1.860416
RF MAE values: 1.21 +- 0.01
```

Using a single line of code, the `mining` function executes 20 runs of an external 3-fold cross-validation procedure. If needed, the `mining` function can be replaced by a cycle for executing the several runs and that uses the `fit` and `predict` functions within an external validation scheme (e.g., use of `holdout` or `crossvaldata` functions, as shown in example `3-math-4.R` from Section 3.2).

The result of `mining()` is a list that contains, among others, the target values and predictions for each run. Such list can be directly used by the `mmetric` and `mgraph` functions, as shown in the example code. The graphs created by `mgraph()` are shown in Figure 4.4. The left of Figure 4.4) shows an interesting (but not perfect) observed (x-axis) versus predicted ($y-$axis) scatter plot, while the right of Figure 4.4 compares the regression error characteristic (REC) performance (Bi and Bennett, 2003) of two methods (`randomForest` vs `mr`). The latter graph confirms a slightly better performance of the `randomForest` method, which presents a higher REC area when compared with the multiple regression method.

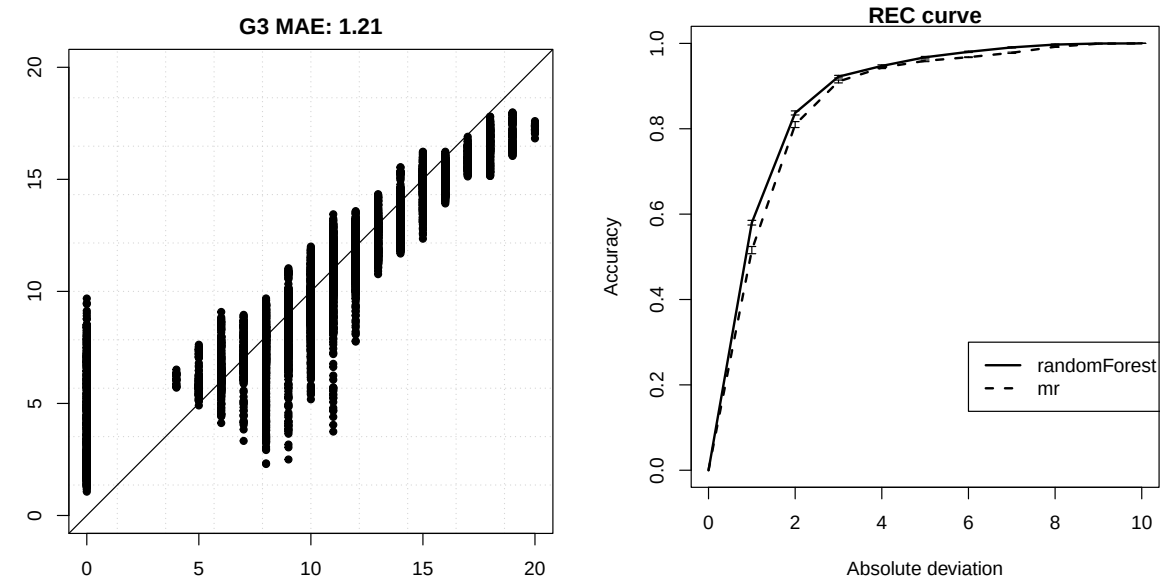

Figure 4.4: Scatter plot for `randomForest` (left) and REC curve (right).

Chapter 5

Time Series Forecasting

This chapter is devoted to time series forecasting (TSF), which is a special case of regression and involves the analysis of a time ordered phenomenon. In particular, it covers univariate TSF, where there is access to only one times series variable. There are several classical pure univariate TSF methods, such as Holt-Winters and ARIMA, which are available in the forecast package. As explained in Section 2.2, the rminer package includes the useful `CasesSeries` function, which transforms a time series into a data.frame (with inputs and a target output variable), thus facilitating the modeling of time series by machine learning methods (e.g., regression).

The example file `5-passengers.R` assumes that the forecast and rminer packages are installed and contains the code:

```
### 5-passengers-1.R: time series forecasting demonstration
### uses AirPassengers data
### saves air.pdf, hw.pdf, ar.pdf and nn.pdf

library(forecast)
library(rminer)

# example with R data AirPassengers
# other time series could be read from a CSV file via read.table
data(AirPassengers)
yrange=diff(range(AirPassengers)) # range for all values, used by NMAE

H=12 # number of ahead predictions
L=length(AirPassengers)
LTR=L-H # length of the training set
# time series monthly object:
# start means: year of 1949, 1st month (since frequency=K=12)
TR=ts(AirPassengers[1:LTR],frequency=12,start=c(1949,1))

pdf("air.pdf")
tsdisplay(TR)
dev.off()

# holt winters method
HW=HoltWinters(TR)
F=forecast(HW,h=H) # 1 to H ahead forecasts
Pred=F$mean[1:H] # HW predictions
Target=AirPassengers[(L-H+1):L]
txt=paste("HW NMAE:",round(mmetric(Target,Pred,metric="NMAE",val=yrange),2)
    ,"\n")
mgraph(Target,Pred,graph="REG",Grid=10,col=c("black","blue"),
       leg=list(pos="topleft",leg=c("target","predictions")),main=txt,PDF="
```

```
          hw")

# arima method:
AR=auto.arima(TR)
F1=forecast(AR,h=H) # 1 to H ahead forecasts
Pred1=F1$mean[1:H] # AR predictions
txt=paste("AR NMAE:",round(mmetric(Target,Pred1,metric="NMAE",val=yrange)
   ,2),"\n")
mgraph(Target,Pred1,graph="REG",Grid=10,col=c("black","blue"),
       leg=list(pos="topleft",leg=c("target","predictions")),main=txt,PDF="
          ar")

# neural network modeling:
d=CasesSeries(AirPassengers,c(1,12,13)) # data.frame from time series
LD=nrow(d)
dtr=1:(LD-H) # train indices
NN=fit(y~.,d[dtr,],model="mlpe")
# from 1 to H ahead forecasts:
Pred2=lforecast(NN,d,start=(LD-H+1),horizon=H)
txt=paste("NN NMAE:",round(mmetric(Target,Pred2,metric="NMAE",val=yrange)
   ,2),"\n")
mgraph(Target,Pred2,graph="REG",Grid=10,col=c("black","blue"),
       leg=list(pos="topleft",leg=c("target","predictions")),main=txt,PDF="
          nn")
```

First, the code loads the `AirPassengers` time series. This is a famous series that corresponds to the number of passengers (in thousands) of an airline company from 1949 to 1960. Then, it sets the training data (`TR`) with the first 132 elements from the series. Next, it plots the training series data and its autocorrelation (ACF) and partial autocorrelation (PACF) values (Figure 5.1). The top graph reveals a series with a monthly seasonal behavior. This is also confirmed by the autocorrelation values, which have local peaks for the 12th and 24th time lags. The dashed horizontal blue lines in the ACF and PACF graphs denote the boundaries of a purely random series. Given that several ACF and PACF values are above and below these blue lines, this series is predictable.

The example code fits then three TSF models: Holt-Winters, ARIMA and a neural network ensemble (rminer `"mlpe"` model). In particular, the fitting of ARIMA requires some computational effort (function `auto.arima` from package forecast). For using data mining methods, the `CaseSeries` function needs to be used with a selected sliding time window (`W`), in order to create a training dataset (`data.frame`) (Cortez, 2010b; Stepnicka et al., 2013). For this example, a priori knowledge was used to set `W=c(1,12,13)`, namely by using the time lags commonly adopted by ARIMA method for monthly series. The resulting `data.frame` has 131 examples, from which the first 119 samples are used as the training set (`d[dtr,]`). While the length of `dtr` is lower than `TR` (it corresponds to `nrow(d)-H`), the last training data value is the same, i.e., `d[dtr[119],]$y==TR[132]` (value of 405). To create the one to `H` ahead forecasts, the long term forecast rminer `lforecast()` was used . The `lforecast` function only uses past training data (called in-samples) to compute the forecasts (also termed out-of-samples). Multi-step ahead forecasts are built by iteratively using 1-ahead predictions as inputs of the data mining model (Cortez, 2010b). The `lforecast` is set to predict up to `H` ahead forecasts, starting on the example `LD-H+1`=120 from object `d`. If the `predict` rminer function was used instead of the `lforecast` function, then only 1-ahead forecasts (i.e., predict $t+1$ knowing all data elements up to time t) could be estimated.

The results are presented in Figure 5.1, in terms of regression plots and the normalized mean absolute error (NMAE) metric. The quality of the forecasts can be visually inspected in

the regression plots, where the predictions should be close to the target values. The NMAE is a scale independent metric that can be interpreted as a percentage error, where lower values correspond to better forecasts (Oliveira et al., 2017). The plots and also NMAE metric values from Figure 5.2, confirming that for this experiment the best forecasts were provided by the Holt-Winters method, followed by the neural network ensemble.

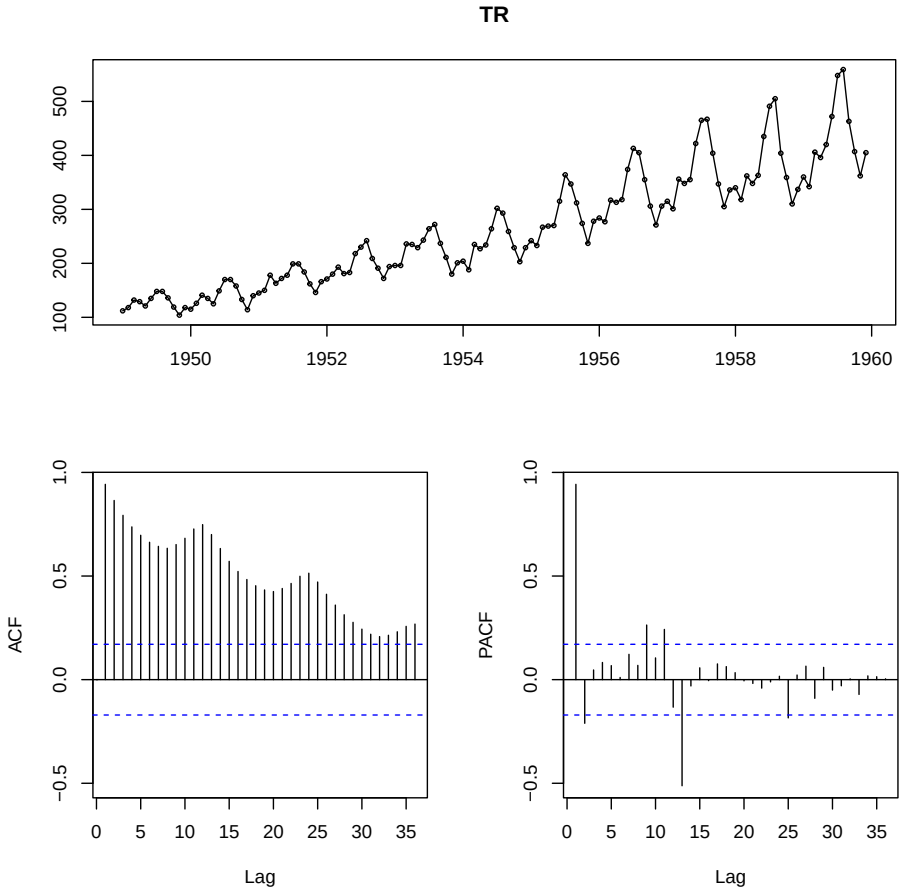

Figure 5.1: Plot of the airline passenger training data (top) and its ACF (bottom left) and PACF (bottom right) statistics.

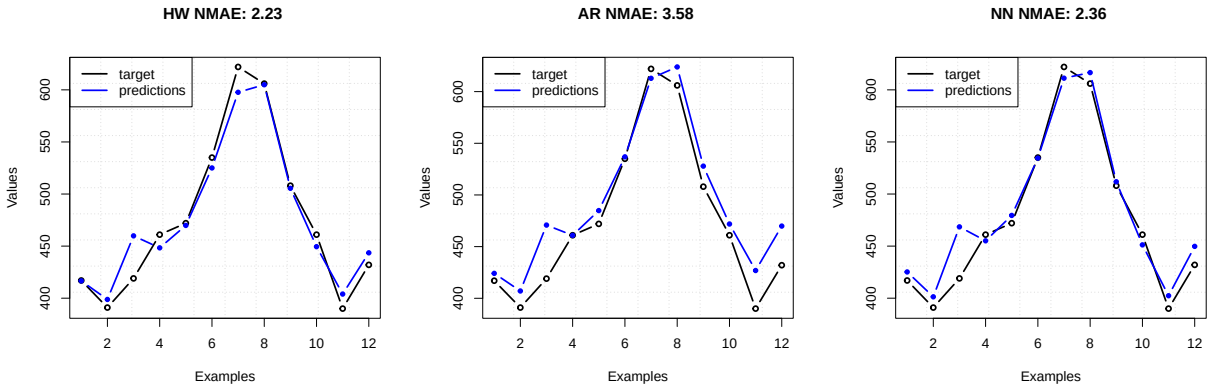

Figure 5.2: Regression plot (`REG`) for holt-winters (left), ARIMA (middle) and `mlpe` (right).

Chapter 6

Conclusions

The goal of this tutorial was to present some simple rminer code recipes that demonstrate the package capabilities for executing classification and regression (including time series forecasting) tasks. The tutorial started with a brief introduction. Then, three CRISP-DM stages were approached: data preparation, modeling (including model parametrization) and evaluation. Finally, a time series forecasting example was shown for the airline passengers series.

Rather than detailing every rminer function, which is available by calling its documentation, such as by executing `help(package=rminer)`, this tutorial followed a learning by example approach. Once the rminer recipes are understood and executed, it is expected that a better knowledge of the rminer package is gained, allowing a more easy writing of code that adopts the package to fulfill the user's needs. Any feedback about the package can be given by sending a message to the package owner: `pcortez@dsi.uminho.pt`.

Readers that found this document useful might also be interest in this Springer book, which is written from a practical point of view and explains how to approach modern optimization or metaheuristics (e.g., simulated annealing; tabu search; genetic algorithms; differential evolution; and particle swarm optimization) with R: `https://link.springer.com/book/10.1007/978-3-030-72819-9` (Cortez, 2021).

Acknowledgments

The author wishes to thank Sérgio Moro, which kindly reviewed the first version of this document.

Bibliography

Asuncion, A. and Newman, D. (2007). UCI Machine Learning Repository, Univ. of California Irvine, `http://www.ics.uci.edu/~mlearn/`.

Bi, J. and Bennett, K. (2003). Regression Error Characteristic curves. In Fawcett, T. and Mishra, N., editors, *Proceedings of 20th Int. Conf. on Machine Learning (ICML)*, Washington DC, USA, AAAI Press.

Brown, M. and Kros, J. (2003). Data mining and the impact of missing data. *Industrial Management & Data Systems*, 103(8):611–621.

Caetano, N., Laureano, R. M. S., and Cortez, P. (2014). A data-driven approach to predict hospital length of stay - A portuguese case study. In Hammoudi, S., Maciaszek, L. A., and Cordeiro, J., editors, *ICEIS 2014 - Proceedings of the 16th International Conference on Enterprise Information Systems, Volume 1, Lisbon, Portugal, 27-30 April, 2014*, pages 407–414. SciTePress.

Calçada, T., Cortez, P., and Ricardo, M. (2012). Using data mining to study the impact of topology characteristics on the performance of wireless mesh networks. In *WCNC*, pages 1725–1730. IEEE.

Chapman, P., Clinton, J., Kerber, R., Khabaza, T., Reinartz, T., Shearer, C., and Wirth, R. (2000). CRISP-DM 1.0: Step-by-step data mining guide. SPSS inc, 1-74.

Cortese, G., Dunbar, G., Carter, L., Scott, G., Bostock, H., Bowen, M., Crundwell, M., Hayward, B., Howard, W., Martínez, J., et al. (2013). Southwest pacific ocean response to a warmer world: insights from marine isotope stage 5e. *Paleoceanography*, 28(3):585–598.

Cortez, P. (2010a). Data mining with neural networks and support vector machines using the r/rminer tool. In Perner, P., editor, *Advances in Data Mining. Applications and Theoretical Aspects, 10th Industrial Conference, ICDM 2010, Berlin, Germany, July 12-14, 2010. Proceedings*, volume 6171 of *Lecture Notes in Computer Science*, pages 572–583. Springer.

Cortez, P. (2010b). Sensitivity Analysis for Time Lag Selection to Forecast Seasonal Time Series using Neural Networks and Support Vector Machines. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN 2010)*, pages 3694–3701, Barcelona, Spain. IEEE.

Cortez, P. (2021). *Modern Optimization with R*. Springer, second edition.

Cortez, P., Cerdeira, A., Almeida, F., Matos, T., and Reis, J. (2009). Modeling wine preferences by data mining from physicochemical properties. *Decision Support Systems*, 47(4):547–553.

Cortez, P. and Embrechts, M. J. (2013). Using sensitivity analysis and visualization techniques to open black box data mining models. *Information Sciences*, 225:1–17.

Cortez, P. and Morais, A. (2007). A Data Mining Approach to Predict Forest Fires using Meteorological Data. In et al., J. N., editor, *New Trends in Artificial Intelligence, 13th EPIA 2007 - Portuguese Conference on Artificial Intelligence*, pages 512–523, Guimarães, Portugal. APPIA.

Cortez, P., Portelinha, M., Rodrigues, S., Cadavez, V., and Teixeira, A. (2006). Lamb Meat Quality Assessment by Support Vector Machines. *Neural Processing Letters*, 24(1):41–51.

Cortez, P. and Silva, A. (2008). Using Data Mining to Predict Secondary School Student Performance. In Brito, A. and Teixeira, J., editors, *Proceedings of the 5th FUture BUsiness TEChnology Conference (FUBUTEC 2008)*, pages 5–12, Porto, Portugal. EUROSIS.

Fawcett, T. (2006). An introduction to ROC analysis. *Pattern Recognition Letters*, 27:861–874.

Ferreira, L., Pilastri, A. L., Martins, C. M., Pires, P. M., and Cortez, P. (2021). A comparison of automl tools for machine learning, deep learning and xgboost. In *International Joint Conference on Neural Networks, IJCNN 2021, Shenzhen, China, July 18-22, 2021*, pages 1–8. IEEE.

Fortino, V., Smolander, O.-P., Auvinen, P., Tagliaferri, R., and Greco, D. (2014). Transcriptome dynamics-based operon prediction in prokaryotes. *BMC bioinformatics*, 15(1):145.

Gonçalves, J. N. C., Cortez, P., Carvalho, M. S., and Frazão, N. M. (2021). A multivariate approach for multi-step demand forecasting in assembly industries: Empirical evidence from an automotive supply chain. *Decis. Support Syst.*, 142:113452.

Hastie, T., Tibshirani, R., and Friedman, J. (2009). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer-Verlag, NY, USA, second edition.

Kohavi, R. (1995). A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, Volume 2, Montreal, Quebec, Canada, Morgan Kaufmann.

Lopes, C., Cortez, P., Sousa, P., Rocha, M., and Rio, M. (2011). Symbiotic filtering for spam email detection. *Expert Systems with Applications*, 38(8):9365–9372.

Mittas, N. and Angelis, L. (2013). Ranking and clustering software cost estimation models through a multiple comparisons algorithm. *Software Engineering, IEEE Transactions on*, 39(4):537–551.

Moro, S., Cortez, P., and Rita, P. (2014). A data-driven approach to predict the success of bank telemarketing. *Decision Support Systems*, 62:22–31.

Nachev, A. and Hogan, M. (2014). Application of multilayer perceptrons for response modeling. In *Proceedings on the International Conference on Artificial Intelligence (ICAI)*, page 1. The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp).

Oliveira, N., Cortez, P., and Areal, N. (2017). The impact of microblogging data for stock market prediction: Using twitter to predict returns, volatility, trading volume and survey sentiment indices. *Expert Systems with Applications*, 73:125–144.

Paradis, E. (2002). R for beginners. Montpellier (F): University of Montpellier, `http://cran.r-project.org/doc/contrib/rdebuts_en.pdf`.

Parente, M., Cortez, P., and Correia, A. G. (2015). Combining data mining and evolutionary computation for multi-criteria optimization of earthworks. In Gaspar-Cunha, A., Antunes, C. H., and Coello, C. A. C., editors, *Evolutionary Multi-Criterion Optimization - 8th International Conference, EMO 2015, Guimarães, Portugal, March 29 -April 1, 2015. Proceedings, Part II*, volume 9019 of *Lecture Notes in Computer Science*, pages 514–528. Springer.

Sanchis-Sánchez, E., Sanchis-Sánchez, C., Sánchez-Lorente, M., and Blasco, J. (2016). Use of thermography in the diagnosis of pressure ulcers category i: A protocol proposal. In *2016 Global Medical Engineering Physics Exchanges/Pan American Health Care Exchanges (GMEPE/PAHCE)*, pages 1–4. IEEE.

Silva, A., Cortez, P., Santos, M. F., Gomes, L., and Neves, J. (2006). Mortality assessment in intensive care units via adverse events using artificial neural networks. *Artificial Intelligence in Medicine*, 36(3):223–234.

Silva, A., Cortez, P., Santos, M. F., Gomes, L., and Neves, J. (2008). Rating organ failure via adverse events using data mining in the intensive care unit. *Artificial Intelligence in Medicine*, 43(3):179–193.

Silva, A. T., Moro, S., Rita, P., and Cortez, P. (2018). Unveiling the features of successful ebay smartphone sellers. *Journal of Retailing and Consumer Services*, 43:311–324.

Stepnicka, M., Cortez, P., Donate, J. P., and Stepnicková, L. (2013). Forecasting seasonal time series with computational intelligence: On recent methods and the potential of their combinations. *Expert Systems with Applications*, 40(6):1981–1992.

Tinoco, J., Gomes Correia, A., and Cortez, P. (2014). Support vector machines applied to uniaxial compressive strength prediction of jet grouting columns. *Computers and Geotechnics*, 55:132–140.

Venables, W., Smith, D., and R Core Team (2013). An introduction to R. `http://cran.r-project.org/doc/manuals/R-intro.pdf`.

Witten, I., Frank, E., and Hall, M. (2011). *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, San Franscico, USA, San Francisco, CA, 3rd edition.

Witten, I., Frank, E., Hall, M., and Pal, C. (2017). *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, San Francisco, USA, San Francisco, CA, fourth edition.

Zhang, X., Ma, Z., and Lv, G. (2020). Applying and assessing multi-output support vector regression with rainfall as additional output for monthly river flow forecasting. *Arabian Journal of Geosciences*, 13(24):1–14.

Zuur, A., Ieno, E., and Meesters, E. (2009). *A Beginner's Guide to R*. Springer.