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

July 2015

[*] If needed, this document should be cited as:
P. Cortez, A tutorial on the rminer R package for data mining tasks, Teaching Report, Department of Information Systems, ALGORITMI Research Centre, Engineering School, University of Minho, Guimarães, Portugal, July 2015.

Abstract

This tutorial explores the rminer package of the R statistical tool. Following a learn 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 data mining tasks, including in particular three CRISP-DM stages: data preparation, modeling and evaluation. The particular case of time series forecasting is also addressed.

Keywords: 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 (due to a high number of model trainings and comparisons), the goal is to show simple demonstration code examples for executing classification and regression 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: `http://www3.dsi.uminho.pt/pcortez/rminer.html`. 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., 2008; Witten et al., 2011).

The rminer package (`http://cran.r-project.org/web/packages/rminer/index.html`) 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 for each CRISP-DM phase, with each R file receiving inputs and generating outputs (e.g., such as file `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 20th May of 2015, the package has been downloaded 10,439 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);
- predicting secondary school student performance (Cortez and Silva, 2008);

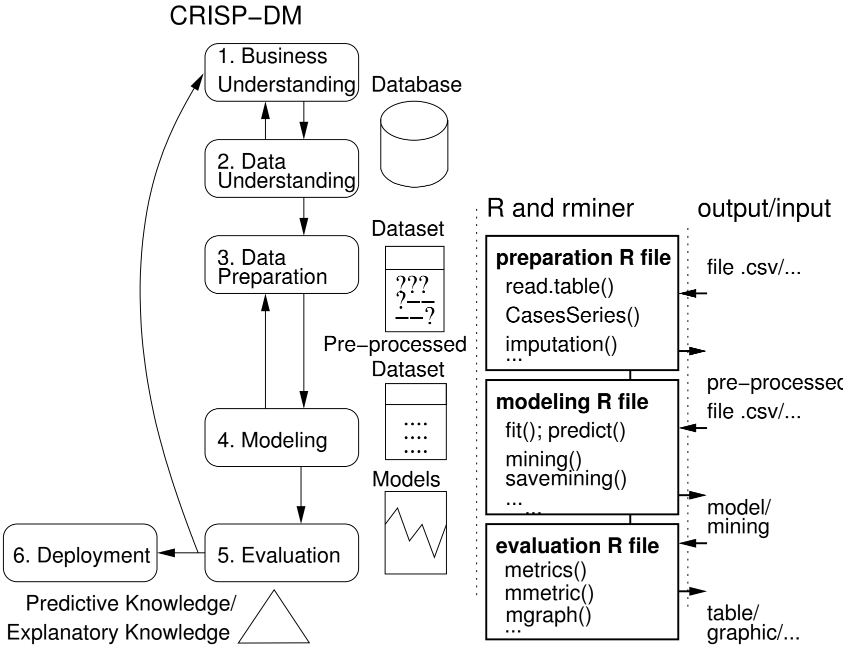


Figure 1.1: The CRISP-DM methodology and suggest rminer package use, adapted from (Cortez, 2010a).

- bank telemarketing (Moro et al., 2014);
- spam email detection (Lopes et al., 2011).

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).

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), and telemedicine (Romano et al., 2014).

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/`. 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 per 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 typical 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: `http://www3.dsi.uminho.pt/pcortez/bib/2010-rminer.txt`

Chapter 2

Data Preparation

The data preparation stage of CRISP-DM may 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). As an demonstrative example, file `prep-1.R` loads several datasets, including the famous Iris, all University California Irvine (UCI) Machine Learning (ML) repository (Asuncion and Newman, 2007) datasets donated by the author of this document, and the Internet Traffic Data (time series) (Cortez et al., 2012):

```
# 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=",")
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)
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)
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)

# internet traffic (time series):
URL="http://www3.dsi.uminho.pt/pcortez/data/internet-traffic-data-in-bits-
   fr.csv"
traffic=read.table(URL,sep=";",header=TRUE)
cat("internet traffic:",nelems(traffic),"\n")
print(class(traffic)) # show class
print(names(traffic)) # show attributes
```

The obtained output is:

```
> source("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"    "wind"
[12] "rain"  "area"
trying URL 'http://archive.ics.uci.edu/ml/machine-learning-databases/00222/
   bank-additional.zip'
Content type 'application/zip' length 444572 bytes (434 Kb)
opened URL
==================================================
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/zip' length 20029 bytes (19 Kb)
opened URL
==================================================
downloaded 19 Kb

student performance math: 395 x 33
[1] "data.frame"
[1] "school"     "sex"        "age"        "address"    "famsize"    "
   Pstatus"
[7] "Medu"       "Fedu"       "Mjob"       "Fjob"       "reason"     "
   guardian"
[13] "traveltime" "studytime"  "failures"   "schoolsup"  "famsup"     "paid
   "
[19] "activities" "nursery"    "higher"     "internet"   "romantic"   "
   famrel"
[25] "freetime"   "goout"      "Dalc"       "Walc"       "health"     "
   absences"
[31] "G1"         "G2"         "G3"
internet traffic: 1657 x 2
[1] "data.frame"
[1] "Time"
[2] "Internet.traffic.data..in.bits..from.an.ISP..Aggregated.traffic.in.the
   .United.Kingdom.academic.network.backbone..It.was.collected.between.19.
   November.2004..at.09.30.hours.and.27.January.2005..at.11.11.hours..
   Hourly.data"
```

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., 2011).

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 `prep-2.R`:

```
### 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))

# time series to data.frame using a sliding time window
# useful for fitting a data-driven model:
print("-- 6th example: time series to data.frame (internet traffic) --")
dtraffic=CasesSeries(traffic[,2],c(1,2,24),1,30)
print(dtraffic)
```

The execution result is:

```
 > source("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  19 167 209  87  17

 poor average    good
 20     376     104
 [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 (internet traffic) --"
 lag24     lag2     lag1        y
 1 64554.48 31597.00 33643.66 37522.89
 2 71138.75 33643.66 37522.89 41738.58
 3 77253.27 37522.89 41738.58 44282.89
```

```
4 77340.78 41738.58 44282.89 43803.96
5 79860.76 44282.89 43803.96 46397.15
6 81103.41 43803.96 46397.15 49180.18
```

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 Internet traffic time series (numeric vector). In the example, the sliding window is made of the $\{1,2,24\}$ time lags and only applied to the first 30 elements of the series.

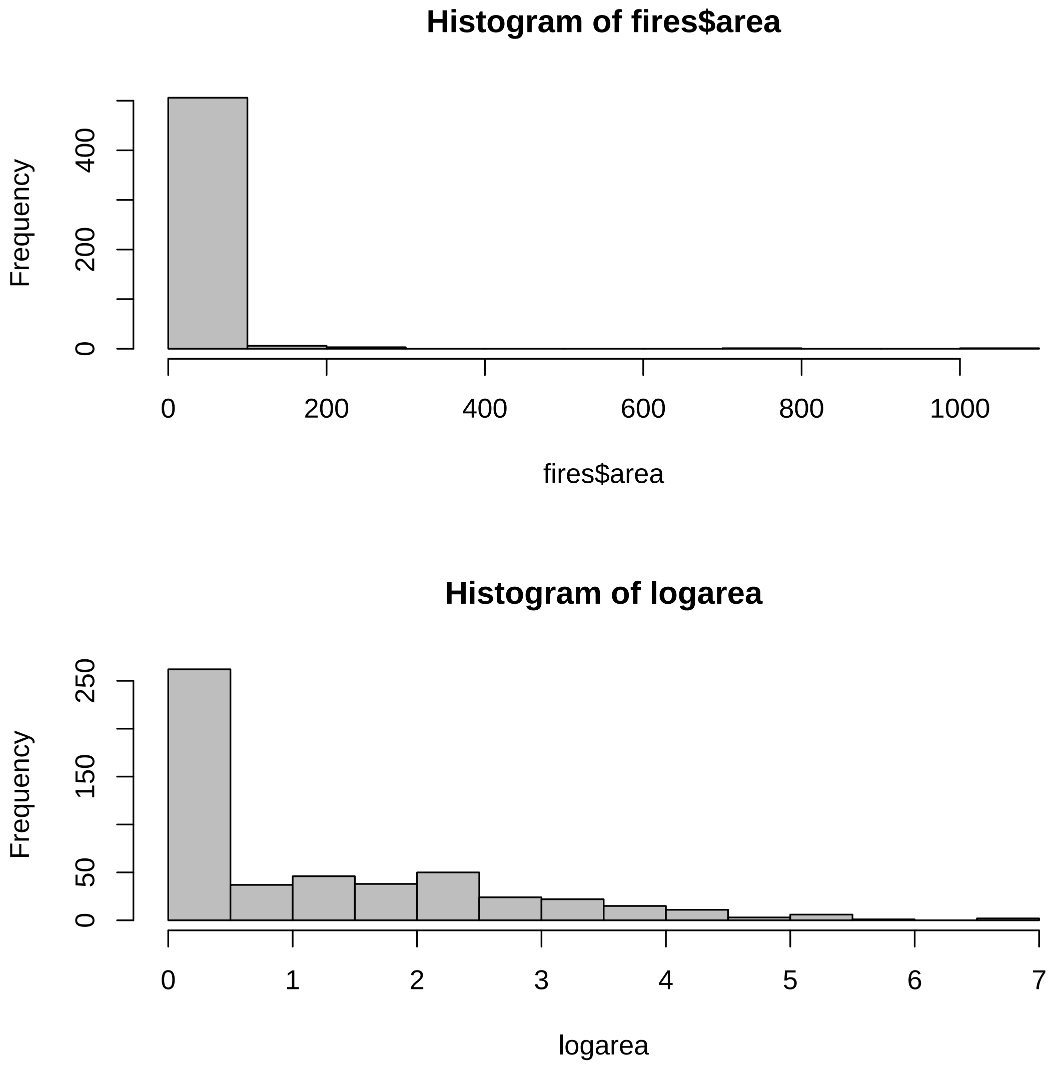


Figure 2.1: Histogram for the forest fires area (top) and its logarithm transform (bottom).

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. An example for the bank data is provided in file `prep-3.R`:

```
# missing data example
# since bank does not include missing data, lets
# synthetically create such data:
set.seed(12345) # set for replicability
bank3=bank
N=500 # randomly assign N missing values (NA) to 1st and 2nd attributes
srow1=sample(1:nrow(bank),N) # N rows
srow2=sample(1:nrow(bank),N) # N rows
bank3[srow1,1]=NA # age
bank3[srow2,2]=NA # job
print("Show summary of bank3 1st and 2nd attributes (with NA values):")
print(summary(bank3[,1:2]))
cat("bank3:",nelems(bank3),"\n")
cat("NA values:",sum(is.na(bank3)),"\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")

# 2nd method: average imputation for age, mode imputation for job:
# 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)
print("mean imputation age summary:")
print(summary(bank5$age))
# substitute NA values by the mode (most common value of bank$job):
print("original job summary:")
print(summary(bank3$job))
bank5=imputation("value",bank5,"job",Value=names(which.max(table(bank$job))
   ))
print("mode imputation job summary:")
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")

# 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")
```

This file artificially introduces 500 missing values in random positions of the first two attributes of the bank dataset. 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 could be applied under a single execution, using the command: `bank6=imputation("hotdeck",bank3)`. Under this latter use of the imputation function, the hot-deck imputation is applied to all attributes with missing data. The result of executing file `prep-3.R` is:

```
> source("prep-3.R")
[1] "Show summary of bank3 1st and 2nd attributes (with NA values):"
age                job
Min.   :18.0   admin.     :884
1st Qu.:32.0   blue-collar:770
Median :38.0   technician :589
Mean   :40.1   services   :351
3rd Qu.:47.0   management :293
Max.   :88.0   (Other)    :732
NA’s   :500    NA’s       :500
bank3: 4119 x 21
NA values: 1000
[1] "-- 1st method: case deletion --"
bank4: 3175 x 21
NA values: 0
[1] "-- 2nd method: value imputation --"
[1] "original age summary:"
Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA’s
18.0    32.0    38.0    40.1    47.0    88.0     500
[1] "mean imputation age summary:"
Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
18.0    33.0    40.0    40.1    46.0    88.0
[1] "original job summary:"
admin.   blue-collar  entrepreneur     housemaid    management
   retired
884           770           135            95           293           152
self-employed      services       student     technician    unemployed
         unknown
142           351            72           589           103            33
NA’s
500
[1] "mode imputation job summary:"
admin.   blue-collar  entrepreneur     housemaid    management
   retired
```

```
1384           770           135            95           293           152
self-employed      services       student    technician    unemployed
         unknown
142           351            72           589           103            33
[1] "-- 3rd method: hotdeck imputation --"
[1] "original age summary:"
Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
18.0    32.0    38.0    40.1    47.0    88.0     500
[1] "hot deck imputation age summary:"
Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
18.00   32.00   38.00   40.12   47.00   88.00
[1] "original job summary:"
admin.   blue-collar  entrepreneur     housemaid    management
   retired
884           770           135            95           293           152
self-employed      services       student    technician    unemployed
         unknown
142           351            72           589           103            33
NA's
500
[1] "hot deck imputation job summary:"
admin.   blue-collar  entrepreneur     housemaid    management
   retired
1015           885           154           101           325           164
self-employed      services       student    technician    unemployed
         unknown
162           396            79           680           121            37
bank6: 4119 x 21
NA values: 0
Saving 5 x 5 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 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 given in file `math-1.R`:

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

# binary task:
```

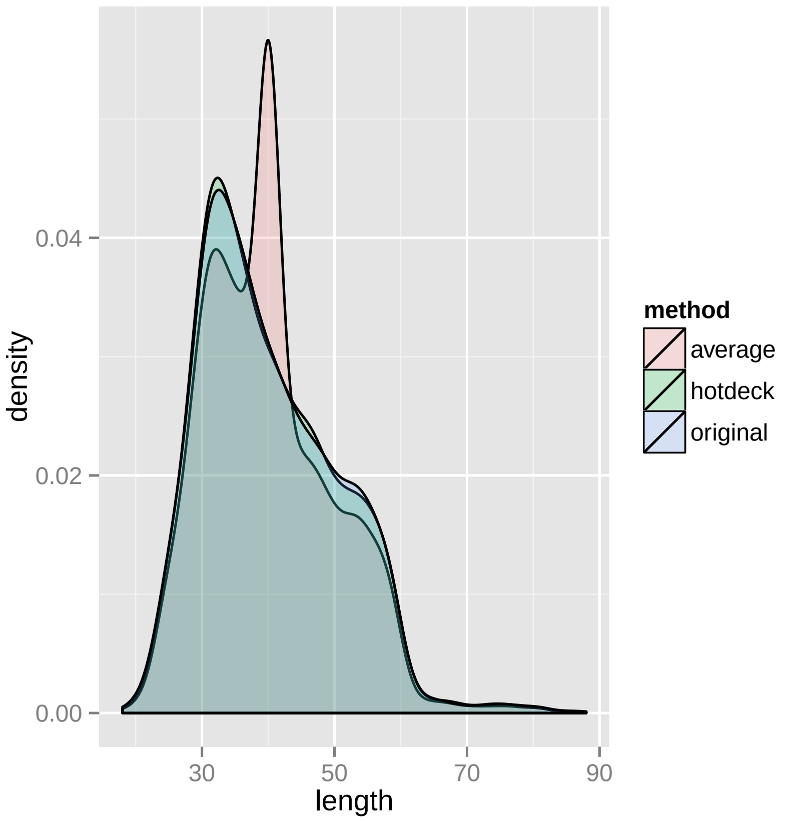


Figure 2.2: Density graphs for the two missing age imputation methods.

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

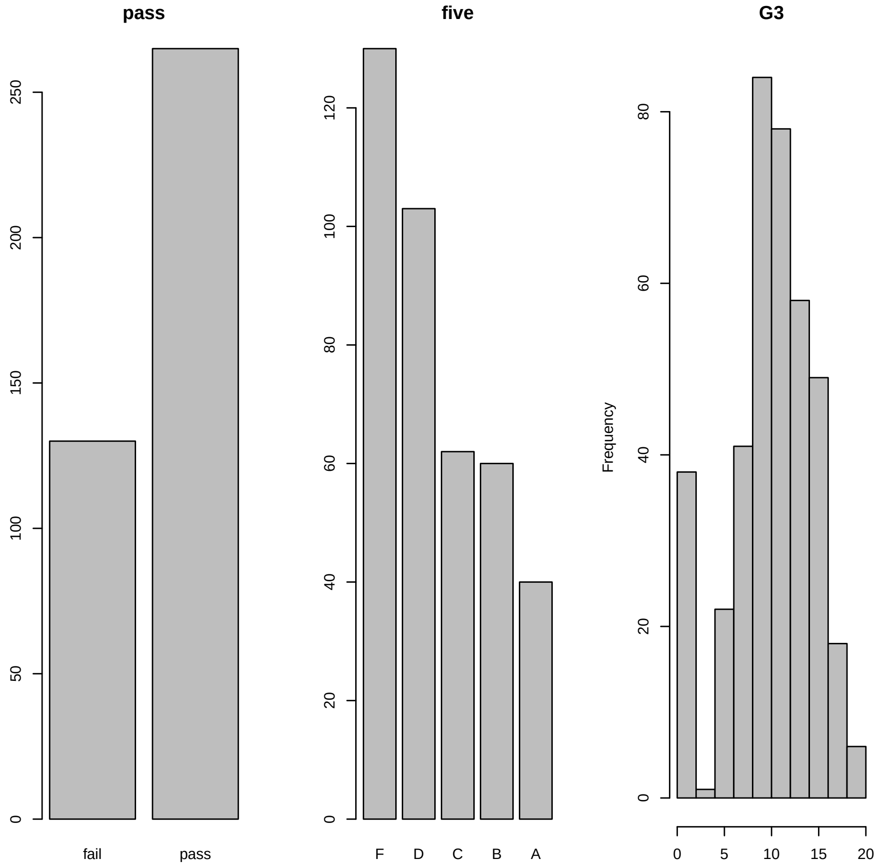


Figure 2.3: Histograms for the Mathematics student's grades (binary, five-level and numeric attributes).

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` (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 (e.g., `help(fit,package=rminer)`).

3.1 Classification

The rminer package includes a large number of classification methods, which can be listed by using the command: `help(fit)`. When performing a classification task, the output variable needs to be 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 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.

3.1.1 Binary Classification

The first modeling code is given in file `math-2.R`:

```
library(rminer)

# read previously saved file
math=read.table(file="math2.csv",header=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 model is fit to all data examples. If the goal is to measure the predictive performance of the fitted models, then a 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 (∼) 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 a large range of classification and regression models, such as decision trees (`"rpart"`), conditional inference trees (`"ctree"`), neural networks (e.g., ensemble of multilayer perceptrons, `"mlpe"`) and support vector machines (`"ksvm"`). 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("math-2.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
   2.5215190  0.0000000
[9]  0.0000000  0.0000000  0.0000000  1.4481013  2.0354430  0.3341772
   0.0000000  0.0000000
[17]  0.0000000  0.0000000  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 1.0882005
   0.0000000 0.0000000
[10] 0.0000000 0.0000000 0.6975048 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 1.1132782
   0.8907414 1.2878966
[28] 1.3903034 0.0000000

$cy
[1] 0

$sy
[1] 0

$nr
[1] 3

$svm
```

```
Support Vector Machine object of class "ksvm"

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

Gaussian Radial Basis kernel function.
Hyperparameter : sigma =  0.0349072151826539

Number of Support Vectors : 294

Objective Function Value : -207.878
Training error : 0.21519
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$mlp[[1]] contains the first multilayer perceptron, of class nnet.`). The support vector machine is acessible using `M4@object$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.

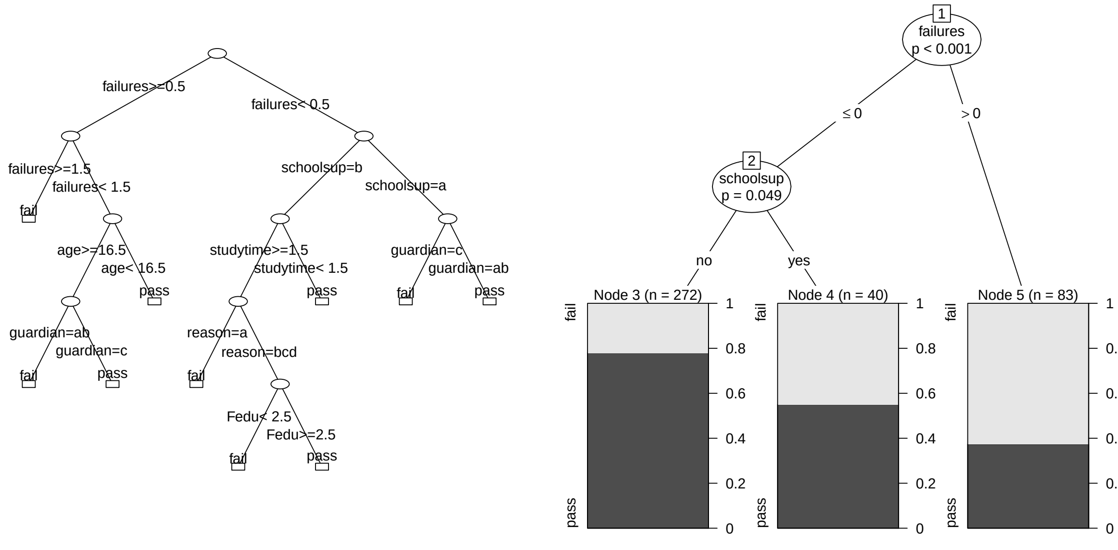


Figure 3.1: Modeled decision trees using rpart (left) and ctree (right) models.

3.1.2 Multiclass Classification

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

```
library(rminer)

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

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

# 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 `math-4.R` is:

```
 > source("math-3.R")
 output class: factor
 bagging > time elapsed: 16.138 , Global Accuracy: 74.68
 Acc. per class 94.94 91.65 91.39 86.08 85.32
 boosting > time elapsed: 16.506 , Global Accuracy: 69.37
 Acc. per class 92.91 91.39 89.37 82.78 82.28
```

```
randomForest.formula randomForest > time elapsed: 1.018 , 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 `math-4.R`:

```
library(rminer)

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

# rpart example with k-fold cross-validation
print("10-fold:")
R2=crossvaldata(G3∼.,math[,c(inputs,g3)],fit,predict,ngroup=10,seed=123,
    model="rpart",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 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 in 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 and the coefficient of determination (R^2).

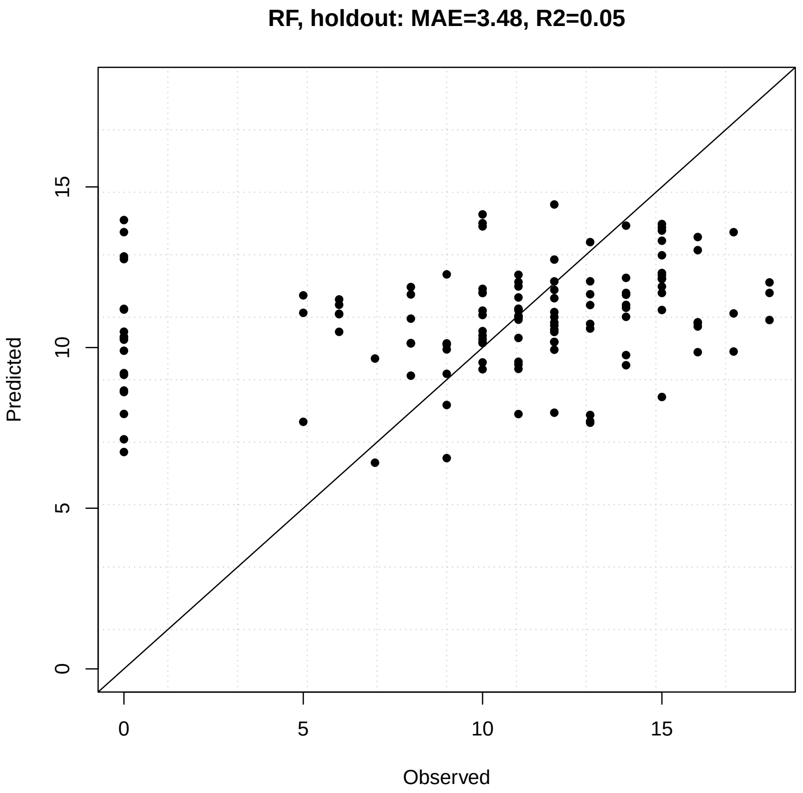
The obtained file `math-4.R` output is:

```
> source("math-4.R")
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=3.48, R2=0.05
[1] "10-fold:"
RF, 10-fold: MAE=3.78, R2=-0.14
```

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.



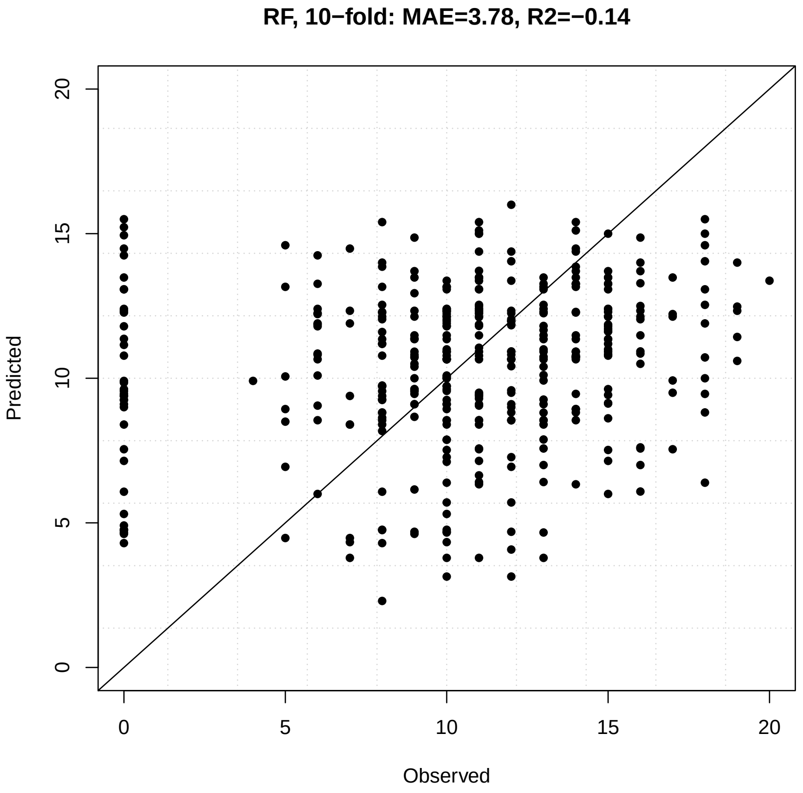


Figure 3.2: Scatter plot of randomForest (RF) predicted vs observed values using holdout (top) and 10-fold (bottom) evaluation methods.

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()` of randomForest package; and `mfinal=100` for the `bagging()` from 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 `math-5`). 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 `math-5.R` shows examples of simple parameter changes:

```
library(rminer)

math=read.table(file="math2.csv",header=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)")
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. The obtained output of file `math-5.R` is:

```
> source("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

$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


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


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

$C
[1] 10

$kpar
list()

$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


$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), tuning these parameters is mostly performed 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 set the parameter 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 `math-6.R` exemplifies this easy `search` use:

```
library(rminer)

math=read.table(file="math2.csv",header=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 now 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 `math-6.R` is:

```
> source("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: 440.6558 best: 440.6558
i: 2 eval: 455.8344 best: 440.6558
i: 3 eval: 460.8564 best: 440.6558
i: 4 eval: 470.7868 best: 440.6558
i: 5 eval: 471.8136 best: 440.6558
i: 6 eval: 474.4212 best: 440.6558
i: 7 eval: 479.0687 best: 440.6558
i: 8 eval: 481.6272 best: 440.6558
i: 9 eval: 482.8256 best: 440.6558
i: 10 eval: 484.2908 best: 440.6558
$mtry
[1] 1

grid with: 10 searches (SAE values)
i: 1 eval: 440.6558 best: 440.6558
i: 2 eval: 455.8344 best: 440.6558
i: 3 eval: 460.8564 best: 440.6558
i: 4 eval: 470.7868 best: 440.6558
i: 5 eval: 471.8136 best: 440.6558
i: 6 eval: 474.4212 best: 440.6558
i: 7 eval: 479.0687 best: 440.6558
i: 8 eval: 481.6272 best: 440.6558
i: 9 eval: 482.8256 best: 440.6558
i: 10 eval: 484.2908 best: 440.6558
$mtry
[1] 1
```

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, 2014).

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

```
#library(rminer)
#math=read.table(file="math2.csv",header=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 `math-7.R`, one output example is (results might change since `"mlpe"` is a stochastic method):

```
> source("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: 1380.104 best: 1380.104
i: 2 eval: 1485.448 best: 1380.104
i: 3 eval: 1526.856 best: 1380.104
i: 4 eval: 1625.727 best: 1380.104
i: 5 eval: 1639.794 best: 1380.104
$decay
[1] 0.1

$maxit
[1] 25
```

```
$nr
[1] 5

$size
[1] 0

mlpe model, same grid using mparheuristic function: 0 2 4 6 8
$decay
[1] 0.1

$maxit
[1] 25

$nr
[1] 5

$size
[1] 14

mlpe model, grid for hidden nodes: 1 2 x decay: 0 0.1
grid with: 4 searches (SAE values)
i: 1 eval: 1381.814 best: 1381.814
i: 2 eval: 1430.534 best: 1381.814
i: 3 eval: 1380.219 best: 1380.219
i: 4 eval: 1539.093 best: 1380.219
$maxit
[1] 25

$size
[1] 1

$decay
[1] 0.1

$nr
[1] 3

mlpe model, same search but with matrix method:
matrix with: 4 searches (SAE values)
i: 1 eval: 1363.471 best: 1363.471
i: 2 eval: 1505.472 best: 1363.471
i: 3 eval: 1418.222 best: 1363.471
i: 4 eval: 1486.6 best: 1363.471
$maxit
[1] 25

$size
[1] 1

$decay
[1] 0

$nr
[1] 3

mlpe model, 2L search for size:
[1] " 1st level:"
2L with: 4 searches (SAE values)
```

```
i: 1 eval: 1513.924 best: 1513.924
i: 2 eval: 1709.59 best: 1513.924
i: 3 eval: 1692.115 best: 1513.924
i: 4 eval: 1720.349 best: 1513.924
[1] " 2nd level:"
2L with: 4 searches (SAE values)
i: 1 eval: 1538.486 best: 1538.486
i: 2 eval: 1559.193 best: 1538.486
i: 3 eval: 1581.442 best: 1538.486
i: 4 eval: 1582.232 best: 1538.486
$maxit
[1] 25

$size
[1] 4

$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: 1272.057 best: 1272.057
i: 2 eval: 1242.453 best: 1242.453
i: 3 eval: 1352.331 best: 1242.453
i: 4 eval: 1260.587 best: 1242.453
i: 5 eval: 1359.491 best: 1242.453
i: 6 eval: 1362.235 best: 1242.453
i: 7 eval: 1295.967 best: 1242.453
i: 8 eval: 1411.589 best: 1242.453
i: 9 eval: 1362.235 best: 1242.453
i: 10 eval: 1273.31 best: 1242.453
i: 11 eval: 1242.038 best: 1242.038
i: 12 eval: 1353.583 best: 1242.038
i: 13 eval: 1259.483 best: 1242.038
i: 14 eval: 1358.536 best: 1242.038
i: 15 eval: 1362.04 best: 1242.038
i: 16 eval: 1291.036 best: 1242.038
i: 17 eval: 1408.434 best: 1242.038
i: 18 eval: 1362.04 best: 1242.038
i: 19 eval: 1279.397 best: 1242.038
i: 20 eval: 1269.723 best: 1242.038
i: 21 eval: 1351.666 best: 1242.038
i: 22 eval: 1279.775 best: 1242.038
i: 23 eval: 1371.068 best: 1242.038
i: 24 eval: 1379.658 best: 1242.038
i: 25 eval: 1331.566 best: 1242.038
i: 26 eval: 1389.553 best: 1242.038
i: 27 eval: 1379.658 best: 1242.038
$kernel
[1] "rbfdot"

$kpar
$kpar$sigma
[1] 0.0625


$C
[1] 0.5
```

```
$epsilon
[1] 0.03125

[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: 1480.594 best: 1480.594
i: 2 eval: 1397.211 best: 1397.211
i: 3 eval: 1317.436 best: 1317.436
i: 4 eval: 1297.775 best: 1297.775
i: 5 eval: 1297.775 best: 1297.775
i: 6 eval: 1297.775 best: 1297.775
i: 7 eval: 1297.775 best: 1297.775
i: 8 eval: 1361.541 best: 1297.775
i: 9 eval: 1361.541 best: 1297.775
i: 10 eval: 1361.541 best: 1297.775
$control
$control$cp
[1] 0.06666667


[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: 1480.594 best: 1480.594
i: 2 eval: 1397.211 best: 1397.211
i: 3 eval: 1317.436 best: 1317.436
i: 4 eval: 1297.775 best: 1297.775
i: 5 eval: 1297.775 best: 1297.775
i: 6 eval: 1297.775 best: 1297.775
i: 7 eval: 1297.775 best: 1297.775
i: 8 eval: 1361.541 best: 1297.775
i: 9 eval: 1361.541 best: 1297.775
i: 10 eval: 1361.541 best: 1297.775
$control
$control$cp
[1] 0.06666667
```

In all examples, `fdebug=TRUE` was added to the `fit` for proving more verbose. In real-world practical examples, such argument should typically 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.

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.

4.1 Classification

4.1.1 Binary Classification

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

```
library(rminer)
# read previously saved file
math=read.table(file="math2.csv",header=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("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
         TPR1       TPR2
78.5       21.5       28.6       46.8        0.5       78.5       78.5
         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
           TPR1        TPR2
78.5        21.5        28.6        46.8         0.5        78.5
   78.5        50.8        92.1
TNR1        TNR2  PRECISION1  PRECISION2         F11          F12
   MCC1        MCC2       BRIER
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
```

```
0.2         0.2         0.7         0.7         0.7         0.7         1.0
            0.6         0.6
ALIFTATPERC
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, 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).

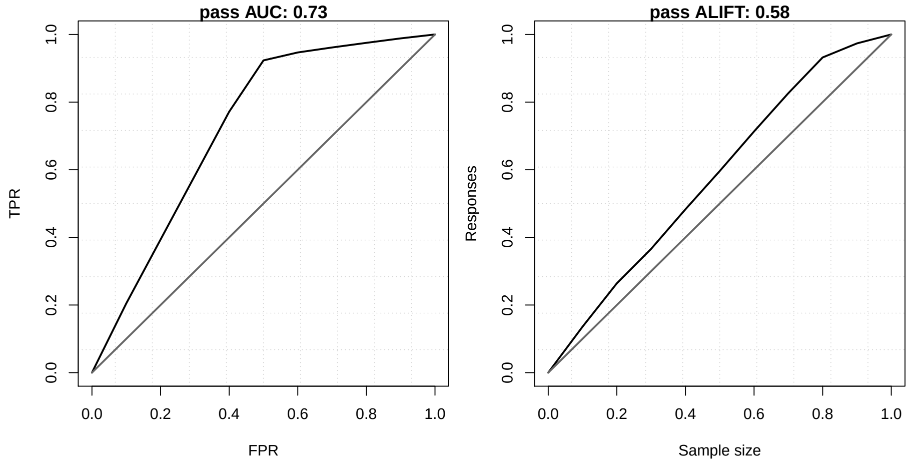


Figure 4.1: ROC (left) and acummulated LIFT (right) curves for the `"pass"` class.

4.1.2 Multiclass Classification

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

```
library(rminer)
# read previously saved file
math=read.table(file="math2.csv",header=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 `eval-2.R` is:

```
> source("eval-2.R")
output class: factor
A     B     C     D     F
0.002 0.006 0.022 0.168 0.802
AUC AUCCLASS1 AUCCLASS2 AUCCLASS3 AUCCLASS4 AUCCLASS5
0.9472834 0.9618644 0.9265766 0.9419913 0.9237417 0.9737051
pred
target  A  B  C  D  F
A  8  5  0  0  0
B  3 16  1  0  0
C  0  8 11  2  0
D  0  0  1 24  9
F  0  0  0  4 39
ACC          CE          BER       KAPPA     CRAMERV    ACCLASS1    ACCLASS2
       ACCLASS3
74.8        25.2        29.0        67.0         0.7        93.9
    87.0        90.8
ACCLASS4    ACCLASS5        TPR1        TPR2        TPR3        TPR4
          TPR5        TNR1
87.8        90.1        61.5        80.0        52.4        70.6
    90.7        97.5
TNR2        TNR3        TNR4        TNR5  PRECISION1  PRECISION2
    PRECISION3  PRECISION4
88.3        98.2        93.8        89.8        72.7        55.2
    84.6        80.0
PRECISION5         F11         F12         F13         F14         F15
          MCC1        MCC2
81.2        66.7        65.3        64.7        75.0        85.7
    0.7         0.6
```

```
MCC3         MCC4         MCC5        BRIER BRIERCLASS1 BRIERCLASS2
   BRIERCLASS3 BRIERCLASS4
0.6          0.7          0.8          0.1          0.0          0.1          0.1
             0.1
BRIERCLASS5          AUC   AUCCLASS1   AUCCLASS2   AUCCLASS3   AUCCLASS4
   AUCCLASS5        NAUC
0.1          0.9          1.0          0.9          0.9          0.9          1.0
             1.0
TPRATFPR       ALIFT       NALIFT ALIFTATPERC
1.0          0.8          0.8          1.0
[1] 0.01 0.01 0.02 0.01 0.03 0.01 0.02 0.02 0.02 0.02 0.02 0.01 0.02 0.03
   0.06 0.04 0.01 0.02
[19] 0.02 0.02 0.02 0.01 0.02 0.02 0.01 0.02 0.04 0.01 0.01 0.04 0.15 0.24
   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 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 0.20 points.

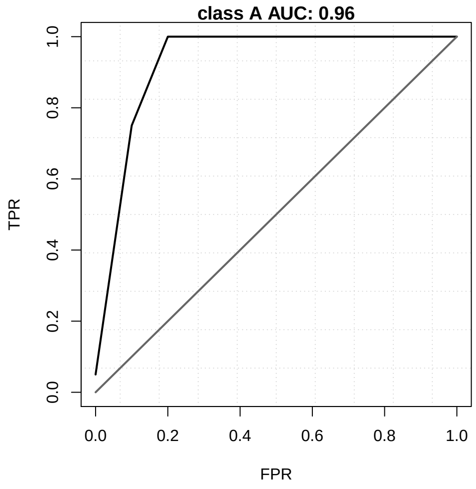


Figure 4.2: ROC curve for class `"A"`.

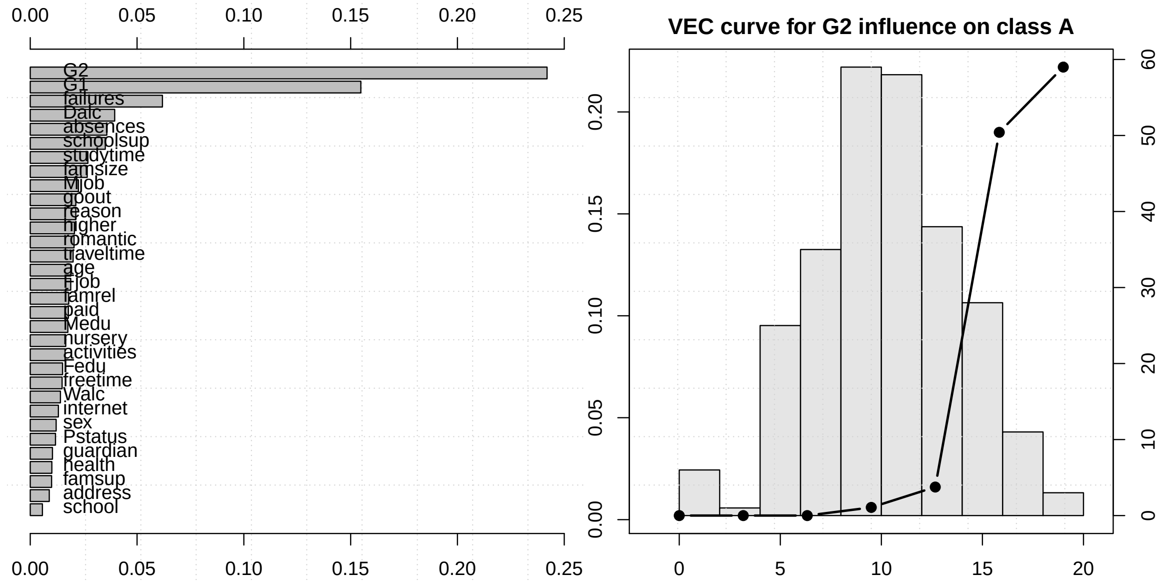


Figure 4.3: Input importance (left, in %) and Variable Effect Curve for the `G2` input average effect on class `"A"`.

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 `eval-3.R`:

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

# input importance
mgraph(M1,graph="imp",leg=names(rmath),xval=10,PDF="rec-1")
```

The obtained result is:

```
> source("eval-3.R")
output class: integer
MAE      RMSE
1  1.260034 1.909381
2  1.200359 1.813237
3  1.221447 1.855083
4  1.202814 1.866552
5  1.213576 1.831553
6  1.214270 1.866656
7  1.183217 1.763938
8  1.180784 1.796339
9  1.209531 1.825594
10 1.218593 1.860474
11 1.203119 1.776782
12 1.221206 1.831095
13 1.217995 1.872522
14 1.250911 1.889030
15 1.203933 1.832018
16 1.220694 1.868813
17 1.245375 1.918572
18 1.226128 1.813113
19 1.213104 1.838675
20 1.241061 1.896131
RF MAE values: 1.22 +- 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 `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.

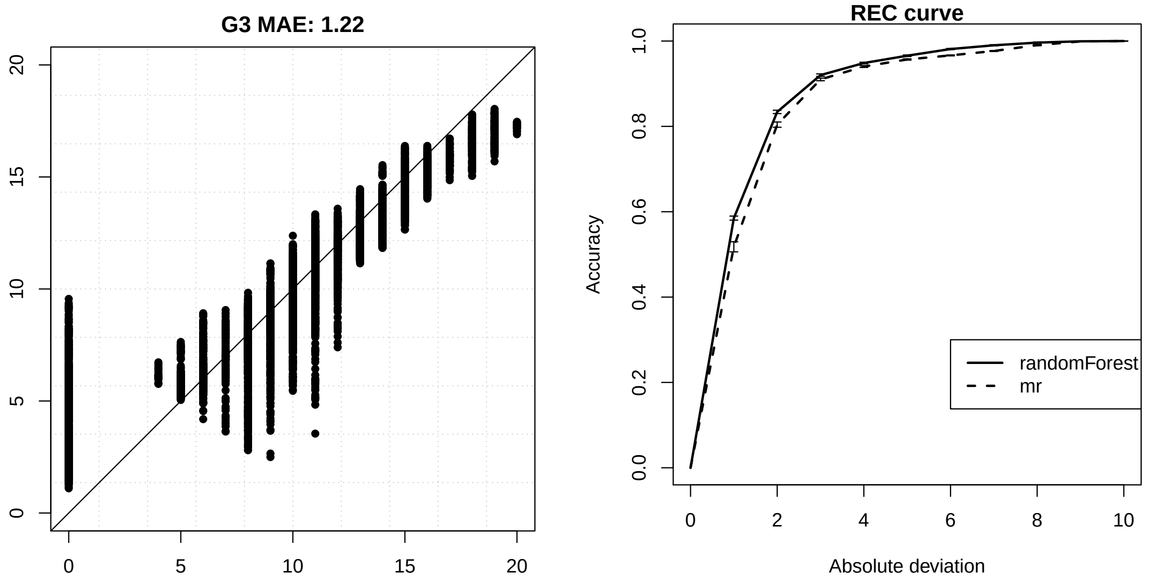


Figure 4.4: Scatter plot for `randomForest` (left) and REC curve (right).

Chapter 5

Time Series Forecasting

This chapter shows how to use the rminer package for time series forecasting (TSF), which is a special case of regression and involves the analysis of a time ordered phenomenon. There are several classical pure 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, thus facilitating the modeling of time series by data mining methods (e.g., regression).

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

```
library(forecast)
library(rminer)

# example with R data AirPassengers
# other time series could be read from a CSV file via read.table
data(AirPassengers)

H=12 # number of ahead predictions
L=length(AirPassengers)
# time series monthly object:
TR=ts(AirPassengers,frequency=12,start=1,end=L-H)

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 SMAPE:",round(mmetric(Target,Pred,metric="SMAPE"),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 SMAPE:",round(mmetric(Target,Pred1,metric="SMAPE"),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 SMAPE:",round(mmetric(Target,Pred2,metric="SMAPE"),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 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 (`"mlpe"`). 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 SMAPE 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 SMAPE metric is often used in the forecasting domain (Stepnicka et al., 2013), where lower values correspond to better forecasts. The plots and also SMAPE metric values from Figure 5.2 clearly confirm the neural network ensemble as the best forecasting model in this particular experiment. It should be noted that both Holt-Winters and ARIMA methods were fit with their default values and better predictions might be achieved under other Holt-Winters/ARIMA model configurations.

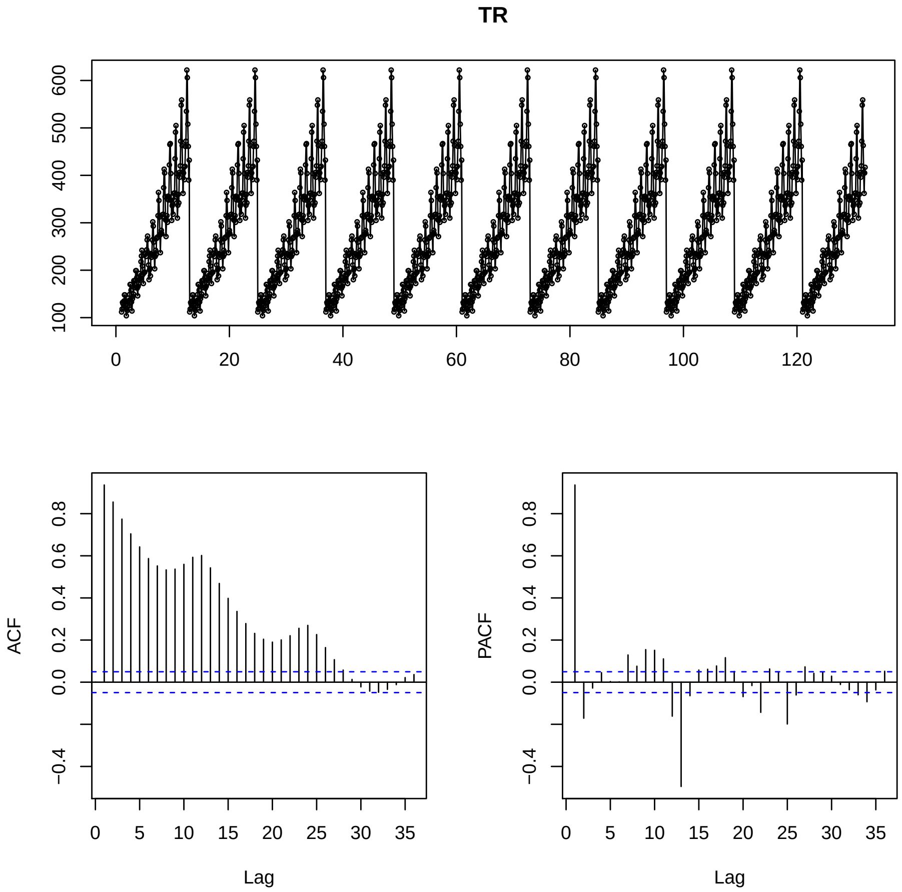


Figure 5.1: Plot of the airline passenger training data (top) and its ACF (bottom left) and PACF (bottom right) statistics.

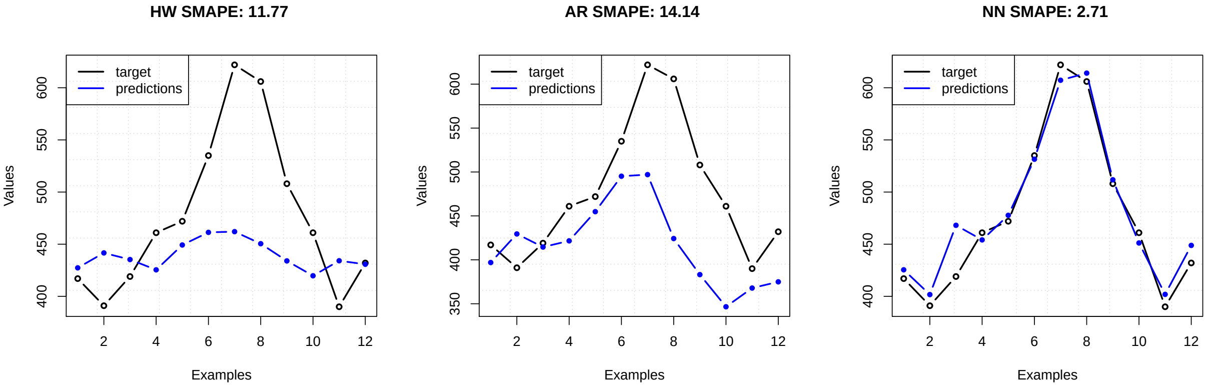


Figure 5.2: Regression plot (REG) for holt-winters (left), ARIMA (middle) and mlpe (right).

Chapter 6

Conclusions

The goal of this tutorial was to present some simple rminer code recipes, in order to 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 (`help(package=rminer)`), or explaining data mining concepts (available in several textbooks, such as (Witten et al., 2011)), this tutorial followed a learn 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: `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: `http://www.springer.com/mathematics/book/978-3-319-08262-2` (Cortez, 2014).

Acknowledgments

The author wishes to thank Sérgio Moro, which kindly reviewed 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. *CRISP-DM consortium*.

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 on Data Mining*, pages 572–583, Berlin, Germany. LNAI 6171, 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. (2014). *Modern Optimization with R*. Springer.

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., 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., Rio, M., Rocha, M., and Sousa, P. (2012). Multi-scale internet traffic forecasting using neural networks and time series methods. *Expert Systems*, 29(2):143–155.

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.

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.

Hastie, T., Tibshirani, R., and Friedman, J. (2008). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer-Verlag, NY, USA, 2nd 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).

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.

Romano, M. F., Sardella, M. V., Alboni, F., L'Abbate, A., Mariotti, R., and Di Bello, V. (2014). The informative contribution of the virtual medical visit in a new heart failure telemedicine integrated system. *TELEMEDICINE and e-HEALTH*, 20(6):508–521.

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.

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.

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

Index