INFO-F-528 Machine learning methods for bioinformatics
Exercise 1: introduction to R - part 1

Gianluca Bontempi
Machine Learning Group, Computer Science Department,
Université Libre de Bruxelles

1 Preparations

Installation of the package ISwR containing our test data:

- Download the file ISwR_1.0-8.tar.gz from: `http://cran.r-project.org/src/contrib`.
- Unzip and untar: `tar -xzvf ISwR_1.0-8.tar.gz`.
- Update the environment variable `R_LIBS`: `export R_LIBS=''rep/ISwR''` where `rep` is the directory in which you will save the file ISwR_1.0-8.tar.gz. Include the line `export` in your file `.bashrc`, so you don't have to retype it every time you use R.
- Type R CMD INSTALL ISwR to install the package.

2 First steps

2.1 Initialisations:

- Start R: type "R" (without the quotes) in a terminal.
- Load the package ISwR: `> library(ISwR)`
- Type `q()` for ending your session.

2.2 Use of R as a calculator:

`> 2+2`

`> exp(-2)`

`> rnorm(15,mean=0,sd=2)`, generating Gaussian distributed random numbers with zero mean and std deviation equal to 2

2.3 Assignments:

```
> x<-2

> x

> x+x
```

- The variables are case-sensitive, can contain a dot (.) and numbers (but cannot start with a number or with a dot followed by a number).
- The names `c,q,t,C,D,F,I,T,diff,df,pt` in particular, are already used by the system and should therefore be avoided.

2.4 Vector arithmetics:

```
> weight <- c(60, 72, 57, 90, 95, 72)

> weight

> height <- c(1.75, 1.80, 1.65, 1.90, 1.74, 1.91)

> bmi <- weight/height^2

> bmi

> sum(weight)

> sum(weight)/length(weight)
```
(calculation of the mean)

```
> xbar <- sum(weight)/length(weight)

> weight - xbar
```
(deviation from the mean)

```
> (weight - xbar)^2

> sum((weight - xbar)^2)

> sqrt(sum((weight - xbar)^2)/(length(weight)-1))
```
(calculation of the standard deviation).

`> mean(weight)` mean

`> sd(weight)` standard deviation

- `c()` is the concatenation function, it creates a vector.
- Operation on vectors of different length: *recycling* of the shortest vector. Example:
  `> weight + 1`
  the operation is carried out elementwise. Let the vector *weight* have

six elements, *a* one element. In such cases, the shorter vector is *recycled*. A warning is issued if the longest vector is not a multiple of the shortest. The expression above is thus equivalent to:
`> weight + c(1,1,1,1,1,1)`

2.5 Graphs

`> plot(rnorm(500))` Plots 500 random numbers, following a Gaussian distribution.

```
> plot(height, weight)
```

```
> hh <- c(1.65, 1.70, 1.75, 1.80, 1.85, 1.90)
```

`> lines(hh, 22.5 * hh^2)` The function `lines` takes two vectors as parameters, one containing the abscissa points and the other their ordinates. The points are joined by lines (in a previously created graph, in a new graph generated with `plot` the parameter `"l"` has to be used to generate lines).

3 The R language

3.1 Vectors

`> c(``Huey'', ``Dewey'', ``Louie'')` character vector

`> c(T,T,F,T)` logical vector

`> bmi > 25` Vector resulting from *relational expressions*.

- Missing values in a vector: value `NA`. The value is carried thorough in computations so that operations on `NA` yield `NA` as the result.
- The following functions create vectors: `c()`, `seq()`, `rep()`.

```
> seq(4,9)
```

```
> seq(4,10,2)
```

```
> 4:9
```

```
> oops <- c(7,9,13)
```

```
> rep(oops, 3)
```

```
> rep(oops,1:3)
```

```
> rep(1:2, c(10,15))
```

3.2 Matrices and arrays

- In R, matrices (2 dimensions) and arrays (more than 2 dimensions) can contain elements of any type (characters, etc.)
- The `dim` assignment function sets or changes the *dimension attribute* of a vector `x`, causing R to treat the vector as a matrix.

  ```
  > x <- 1:12
  > dim(x) <- c(3,4)
  > x
  ```

- Other possibilities to create matrices:

  ```
  > matrix(1:12, nr=3, byrow=T)
  > a <- matrix(1,6,7)
  > a
  ```

- Renaming of rows and transpose of a matrix

  ```
  > x <- matrix(1:12, nr=3, byrow=T)
  > rownames(x) <- LETTERS[1:3]
  > x
  > t(x)
  ```

- matrix product

  ```
  > a <- matrix(1:4,2,2,byrow=T); b<- matrix(2:5,2,2,byrow=T)
  ```

  if `a` and `b` are square matrices of the same size then

  ```
  > a*b
  ```

  is the matrix of element by element products while

  ```
  > a%*%a
  ```

  is the matrix product. Test the difference

- inversion

  ```
  > solve(a)
  ```

- solution of the linear system $Ax = b$, where A is a square matrix and b is a vector is returned by

```
> solve(A,b)
```

- "Glueing" vectors together: functions `rbind()` and `cbind()` (row- and columnwise respectively)

```
> rbind(A=1:4, B=5:8, C=9:12)
```

```
> cbind(A=1:4, B=5:8, C=9:12)
```

3.3 Factors

- *Factors* are the categorial variables (see `enum` in C++). They make it possible to assign meaningful names to the categories.
- A factor is said to have a set of *levels*.
- Ex: a factor has four levels. It consists of two items: a vector of integers between 1 and 4 and a character vector of length 4 containing strings describing what the four levels are.

```
> pain <- c(0,3,2,2,1)
```

```
> fpain <- factor(pain, levels=0:3)
```

```
> levels(fpain) <- c(``non'', ``mild'', ``medium'', ``severe'')
```

- `fpain` is a categorial variable.

```
> fpain
```

```
> as.numeric(fpain)
```

```
> levels(fpain)
```

3.4 Lists

- Useful to combine a collection of objects into a larger composite object.
- The following data concern pre- and postmenstrual energy intake in a group of women.

```
> intake.pre <- c(5260, 5470, 5640, 6180, 6390,
+ 6515, 6805, 7515, 7515, 8230, 8770)
> intake.post <- c(3910, 4220, 3885, 5160, 5645,
+ 4680, 5265, 5975, 6790, 6900, 7335)
```

- Combine the vectors into a list

```
> mylist <- list(before=intake.pre, after=intake.post)

> mylist
```

- The components of the list are named according to the arguments' names used in *list* (in the example `before` and `after`). Named components may be extracted as follows

```
> mylist$before
```

3.5 Data frames

- Data frames can be thought of as matrices with columns of mixed variable types.
- A data frame corresponds to a "data set" in other statistical packages. It is a list of vectors and/or factors of the same length. Data in the same position come from the same experimental unit.
- Creation of a data frame:

```
> d <- data.frame(intake.pre, intake.post)

> d
```

- Elements in one row correspond to the same woman
- As with lists, variables are accessible using the "$" notation

```
> d$intake.pre
```

3.6 Indexing

- Allows to select a particular element in a vector

```
> intake.pre[5]

> intake.pre[c(3,5,7)]

> v <- c(3,5,7)

> intake.pre[v]

> intake.pre[1:5]

> intake.pre[-c(3,5,7)]    delete elements
```

3.7 Conditional selection

```
> intake.post[intake.pre > 7000]
```

```
> intake.post[intake.pre > 7000 & intake.pre <=8000]
```

- Other logical operators: `!` and `|`

```
> intake.pre > 7000 & intake.post <= 8000
```

- The function `is.na(x)` is used to find out which elements of `x` are recorded as missing (`NA`).

3.8 Indexing of data frames

```
> d <- data.frame(intake.pre, intake.post)
```

```
> d[5,1]
```

```
> d[5,]
```

```
> d[d$intake.pre > 7000,]
```

- This is equivalent to

```
> sel <- d$intake.pre > 7000
```

```
> sel
```

```
> d[sel,]
```

3.9 subset and transform

```
> data(thuesen)
```

```
> thue2 <- subset(thuesen, blood.glucose < 7)
```

```
> thue2
```

```
> thue3 <- transform(thuesen, log.gluc=log(blood.glucose))
```

3.10 Grouped data and data frames

```
> data(energy)
```

```
> energy
```

- In order to have data in a separate vector for each group:

```
> exp.lean <- energy$expend[energy$stature==''lean'']
```

```
> exp.obese <- energy$expend[energy$stature==''obese'']
```

- Or, alternatively:

```
> l <- split(energy$expend, energy$stature)
```

```
> l
```

3.11 Sorting

```
> intake.post
```

```
> sort(intake.post)
```

- Sort a series of variables according to the values of some other variable

```
> order(intake.post)
```

```
> o <- order(intake.post)
```

```
> intake.post[o]
```

```
> intake.pre[o]
```

- The function `order()` can also be used with multiple parameters: ex: `order(sex, age)` creates a main division into men and women and within each sex an ordering by age.

3.12 Implicit loops

- Apply a function to each element of a set of values or vectors and collect the results in a single structure.
- In R this can be done by using one of the two functions `lapply` and `sapply`. The former always returns a list whereas the latter tries to simplify the result into a vector or matrix if possible.
- Ex: to compute the mean of each variable in a data frame of numeric vectors:

```
> lapply(thuesen, mean, na.rm=T)
```

```
> sapply(thuesen, mean, na.rm=T)
```

- The function `apply` allows to apply a function to the rows or columns of a matrix.

```
> m <- matrix(rnorm(12),4)
```

```
> m
```

```
> apply(m, 2, sum)
```

- Compare with `sum(m)`
- The second argument is the index (or the index vector) that defines what the function is applied to; in this case the columnwise minima.
- The function `tapply` allows to create tables of the function's value on subgroups defined by its second argument, which can be a factor or a list of factors. (The grouping can also be defined by ordinary vectors. They will be converted to factors internally.)

  ```
  > tapply(energy$expend, energy$stature, median)
  ```

4 The graphics subsystem

4.1 Titles, labels and miscellaneous

```
> x <- runif(50,0,2)

> y <- runif(50,0,2)

> plot(x,y,main="Main title", sub="subtitle",

+ xlab="x-label", ylab="y-label")

> text(0.6, 0.6, "text at (0.6, 0.6)")

> abline(h=.6, v=.6)
```

- Using a second graphical device `> x11()`
- List of graphical devices: `> dev.list()`
- Consultation of the current graphical device: `> dev.cur()`
- Change of current graphical device: `dev.set(id)`
- Closing a graphical device: `dev.off(id)`
- Other types of graphical devices: `pdf()`, `postscript()`.

4.2 Building a plot from pieces

`> plot(x, y, type=''n'', xlab='''', ylab='''', axes=F)` empty plot.

```
> points(x,y)

> axis(1)

> axis(2, at=seq(0.2,1.8,0.2))
```

```
> box()

> title(main="Main title", sub="subtitle",

+ xlab="x-label", ylab="y-label")
```

4.3 The function "par"

- The function `par()` allows to specify many graphical parameters. Many of these parameters exist, consult the documentation for a more detailed overview.
- Example: change the default margin sizes:

  `> par(mar=c(4,4,2,2)+0.1)`

  `> par(ask=TRUE)` allows to ask for a confirmation (by hitting "Enter") before changing the plot in the graphics window.

5 R programming

5.1 Grouped commands

- Use of { ... ... }.
- Form an expression whose value is the result of the last group expression.

5.2 If

- `if(exprLogique1) expr2 else expr3`
- Variants: see documentation.

5.3 Loops

- `for(name in expr1) expr2` where `expr1` is a vector expression (ex: `1:20`).
- `repeat expr`
- `while(condition) expr`
- Existence of the `break` instruction.

5.4 Writing functions

- `name <- function(arg1, arg2, ...) expression`
- The value of `expression` is the value returned for the function.

5.5 Examples

- Some standard functions include `log10` and `weighted.mean`.
- Example to test: compute the square root using Newton's method

```
> y <- 12345
> x<- y/2
> while(abs(x*x-y) > 1e-10) x <- (x+y/x)/2
> x
> x^2
```

- Using `repeat`

```
> x <- y/2
> repeat{
+      x <- (x+y/x)/2
+ if(abs(x*x-y) < 1e-10) break
+ }
> x
```

- Example of a `for` loop :

```
> x <- seq(0,1,.05)
> plot(x,x, ylab=''y'')
> for( j in 2:8) lines(x, x^j)
```

5.6 Executing an R script

Function `source(filename)`.

6 Session management

6.1 The workspace

- All variables created in R are stored in a common workspace. The function `ls()` can be used to see which variables are defined in the workspace.

```
> ls()
```

- To delete some of the objects use the function `rm()`

  ```
  > rm(height, weight, bmi)
  ```

- `rm(list=ls())` clears the entire workspace.

- The workspace can be saved to a file at any time using the function `save.image()`. It will be saved to a file called `.RData` in the working directory.

- Saved files can be loaded into the current workspace using `load`.

6.2 Help

- `help.start()` starts an html help, and `help(function_name)` provides documentation on the specified function.

6.3 The packages

- An R installation contains a library of packages. A package can contain function, libraries of compiled code (C, fortran,..) and data sets.

- A package can be loaded by `> library(package_name)`.

- Enter `> library(help=package_name)` to obtain further details about the package.

6.4 Built-in data

- The function `data` is used to load a built-in data set (one that comes with R or one of the packages) into memory, ex: `data(thuesen)`.

- The function `data` goes through the data directories associated with each package and looks for files whose basename matches the given name. Depending on the file extension different functions are called: `.data` files are loaded using the function `read.table`, those with an extension `.R` are executed as sources files.

6.5 attach() and detach()

- The notation for accessing variables in data frames can get very long,ex: `plot(thuesen$blood, thuesen$short.velocity)`. The function `attach()` makes R look for objects among the variables in a given data frame:

  ```
  > attach(thuesen)
  ```

- The variables of the data frame "thuesen" are now directly available.

  ```
  > blood.glucose
  ```

- What happens is that the data frame is placed in the system's *search path*. The search path can be viewed using the function `search()` :

  ```
  > search()
  ```

  Note that the data frames are stored in the search path one with respect to the other. This implies a priority in the order of search.

- The function `detach()` removes a data frame from the search path. If used without argument, the data frame in position 2 is detached.

6.6 Errors

The default R action in case of error is to print an error message and exit from the top-level expression being evaluated. If you are programming you may want to specify a debugging action to take when an error occurs. The action is specified by the value of the global `error` option, specified by a call to the `options()`. The recommended option during program development is:

```
options(error = recover)
```

With this option in place, an error during an interactive session will call `recover()` from the lowest relevant function call, usually the call that produced the error. You can browse in this or any of the currently active calls, and recover arbitrary information about the state of computation at the time of the error.

Another useful function for debugging is the `browser()` function. The evaluation of the expression: `browser()` invokes a parse-and-evaluate interaction at the time and in the context where the call to `browser()` took place.

7 Data entry

7.1 Reading from a text file

- Using the function `read.table()`, data in "ASCII format" can be read and a stored in a data frame. The function assumes that the file has the following format: each line contains all data with respect to a single "subject" (person, animal, etc.) separated by spaces. The first line may contain a header providing the variable names.

- Suppose a file `thuesen.txt` with the following content:

  ```
  blood.glucose   short.velocity
  15.3            1.76
  10.8            1.34
  8.1             1.27
  ```

, etc... The corresponding data frame can be created by

```
> thuesen <- read.table(‘‘thuesen.txt’’, header=T)
```

```
> thuesen
```

7.2 The data editor

- To edit a data frame interactively the `edit()` can be used

  ```
  > data(airquality)
  ```

  ```
  > aq <- edit(airquality)
  ```

- An old version of the old data frame can be deleted using `fix(airquality)`.

- Enter data into an empty data frame:

  ```
  > dd <- data.frame()
  ```

  ```
  > fix(dd)
  ```

8 References

- Peter Dalgaard, “Introductory statistics with R”, Springer.

- The following tutorials are available from http://www.R-project.org/ and others:

  – “An introduction to R”,

  – “Practical regression and anova using R”,

  – “R for beginners”.