```
*******************************************
***    INTRODUCTION TO MACHINE LEARNING   ***
***           Practical sessions          ***
*******************************************
```

# EXAMPLES IN R

Barbora Hladká, Martin Holub
Institute of Formal and Applied Linguistics
Charles University in Prague
2013/14

```
*******************************************
```

## LITERATURE

Getting started with R
  -- e.g. http://data.princeton.edu/R/gettingStarted.html

An introduction available on the web:
  http://cran.r-project.org/doc/manuals/R-intro.html

R Language Definition
  -- available at http://cran.r-project.org/doc/manuals/R-lang.html

An Introduction to R
  by W. N. Venables, D. M. Smith and the R core team

R for Beginners
  by Emmanuel Paradis

... and a lot of other sources ...

```
*******************************************************************
```

## GENERAL COMMANDS/FUNCTIONS

```
# to assign a value to an object
> x <- 2
# or (the same)
> x = 2

# to get basic attributes of an object
> mode(x)
> length(x)
> str(x)

# overview of an object's contents
> summary(x)
> table(x)

# to get the list of your objects
> ls()

# to remove an object from the memory
> rm(x)
```

```
# to remove all objects
> rm(list=ls())

# to find help on any R function
> help(function)
# or simply
> ?function

# to save the history of your commands (the default file is ".Rhistory")
> savehistory()
```

**SOME OF THE ELEMENTARY ARITHMETIC AND LOGICAL OPERATORS**

```
# + - * / ^
# %% (modulo)
# %/% (integer division)

# comparison
# < > <= >= == !=

# logical
# ! & (&&) | (||) xor(x,y)

# to get the description of operators and precedence rules
> ?Syntax
> ?Logic
> ?Arith

# some of the basic mathematical functions
# log(), exp(), sin(), cos(), tan(),
# asin(), acos(), atan(),
# abs(), sqrt(), round(), choose(n,k)
```

**CREATING VECTORS**

```
# to concatenate a number of elements, e.g. to create a vector of numbers
> a <- c(0,1,2,3)

# to select some of the elements from a vector
> a[4]
> a[c(1,3)]
> a[-1]

# some basic operations with vectors
> sum(a); prod(a)
> min(a); max(a)
> length(a)
> which.min(a); which.max(a)
> mean(a)
> median(a)

# to create a regular sequence
> b <- 1:4
> prod(1:10)
> seq(1,10,0.1)
> rep(1/6,100)
```

```
# Arithmetic operations are performed on each element of vectors
> a+1
> a*b
> a == b
> prod(a == b)
> prod(a+1 == b)
```

**R IS SOMETIMES TRICKY -- SOME EXAMPLES**

```
# observe the difference between
> x <- 1:20; x[x>5 & x<15] <- 0; x
# and
> x <- 1:20; x[x>5 && x<15] <- 0; x

# -- WHY???
# -- to discover the difference between & and && -- look at
>?Logic

# try the following:
> x>5
> x>5 & x<15
> x>5 && x<15

# && gives only one value
# while & works with all vector elements
# positive numbers are considered as true, zeros are false
> c(1,2,3) &  c(2,3,3)
> c(1,2,3) && c(0,3,3)
> c(1,2,3) &  c(2,3,3)
> c(1,2,3) && c(0,3,3)
```

**COMPARE SELECTION USING VECTOR OF INDEXES VS. LOGICAL VECTOR**

```
> x[c(1,2,4)]
> x[c(T,T,F,T)]    # logical vector is repeated

# thus
> x[x>10]          # is the same as x[x>10 & TRUE]
# BUT
> x[x>10 && TRUE]  # is the same as x[FALSE]
```

**COMPARING NUMBERS**

```
# be careful when comparing numbers; try the following:
> 0.9 == 1.1 - 0.2
> all.equal(0.9, 1.1 - 0.2)
> 1:5 == 1:5
> identical(1:5, 1:5)
> seq(0.2,1,0.2) * 5
> seq(0.2,1,0.2) * 5 == 1:5
> identical(seq(0.2,1,0.2) * 5, 1:5)
> all.equal(seq(0.2,1,0.2) * 5, 1:5)
```

```
# another example
> d45 <- pi*(1/4 + 1:10)
> tan(d45)
> tan(d45) == rep(1,10)
> all.equal(tan(d45), rep(1,10))
> all.equal(tan(d45), rep(1,10), tol=0)  # to see difference
```

**FACTORS**

```
# A factor stores both values and possible levels of a categorial variable
# levels (usually a small number) are "names" of categorial values.

> people = factor(c(1,1,1,0,1,0,0,0,1,0,1,1,1,1),
                  labels=c("male", "female")
                 )
> plot(people)

> nationality = factor(c("CZ", "CZ", "SK", "SK", "IND", "CZ", "RU", "MEX",
                         "SRB", "CZ", "CZ", "SRB")
                      )
> nationality
> table(nationality)
> plot(nationality)

# you can change the set of levels:
> levels(nationality)
[1] "CZ"  "IND" "MEX" "RU"  "SK"  "SRB"

> levels(nationality) = c(levels(nationality), "PL", "DE")
> levels(nationality)
[1] "CZ"  "IND" "MEX" "RU"  "SK"  "SRB" "PL"  "DE"

> plot(nationality)
```

**WRITING YOUR OWN FUNCTIONS IN R**

```
> f <- function(a,b) a*b
> f(17,3)

> fact <- function(n) prod(1:n)
> fact(10)
# does not work for 0

# you can also use recursion
> fact <- function(n) if(n==0) 1 else n*fact(n-1)

# to edit your function
> fact <- edit(fact)
> fact <- emacs(fact)
```

**ELAPSED TIME**

```
# to measure the elapsed time
> system.time({m = matrix(sample(1:6, 6*10^6, replace=T),nrow=6,ncol=10^6)})
   user  system elapsed
  0.280   0.044   0.322
```

**TO EXIT R**

```
> q()
```

**RANDOM SEQUENCES, RANDOM GENERATORS**

```
# a random sample from an existing vector
> sample(x, 10)
> sample(1:1000, 100)

# a random sequence
> sample(1:100, 100, replace=T)

# a random permutation
> sample(1:10, 10, replace=F)
```

**RANDOM SEQUENCES OF NUMBERS WITH GIVEN DISTRIBUTION**

```
# UNIFORM
> runif(10)
> plot(runif(1000))
> hist(runif(1000))
> hist(runif(1000), breaks=seq(0,1, by=1/20))
> floor(runif(100,0,100)) + 1          # to generate random integers

# NORMAL
> rnorm(10)
> plot(rnorm(5000))

# BINOMIAL
> rbinom(10, 1, 0.5)
> rbinom(10, 10, 0.5)
> plot(rbinom(1000, 10, 0.5))
> table(rbinom(1000, 10, 0.5))

  0   1   2   3   4   5   6   7   8   9  10
  2  13  42 120 204 238 218 109  43   9   2

> hist(rbinom(1000, 10, 0.5), breaks=0:10)
> hist(rbinom(1000, 10, 0.7), breaks=0:10)   # a fake coin
```

**DENSITY FUNCTION**

```
# UNIFORM
> dunif(-2:2)
> plot(seq(-2,2,0.01), dunif(seq(-2,2,0.01)), type="l")

# NORMAL
> dnorm(-5:5)
```

```
> plot(seq(-5,5,0.01), dnorm(seq(-5,5,0.01)), type="l")

# BINOMIAL
> plot(factor(rbinom(100000,10,0.5)))
> dbinom(0:10,10,0.5)
> plot(0:10, dbinom(0:10,10,0.5))
```

**CUMULATIVE DISTRIBUTION FUNCTION**

```
# UNIFORM
> plot(seq(-2,2,0.01), punif(seq(-2,2,0.01)), type="l")

# NORMAL
> plot(seq(-5,5,0.01), pnorm(seq(-5,5,0.01)), type="l")

# BINOMIAL
> plot(0:10, pbinom(0:10,10,0.5))
> barplot(pbinom(0:10,10,0.5))
```

**QUANTILE FUNCTION**
-- qunif, qnorm, qbinom

**OTHER DISTRIBUTIONS ALSO AVAILABLE**
-- e.g. the negative binomial (nbinom)

*******************************************************************
*****   **Exercises**
*******************************************************************

*** **Exercise 1** ***

Question:
How many times would you need to roll a die to get number 1?

```
# Probability p(i) = p("number 1 is rolled only after i trials")
#             p(i) = (5/6)^(i-1)*(1/6)
# Expected number of trials to get number 1 is
#             SUM( p(i)*i ) for i = 1,..., infinity
# Because the value of p(i)*i goes to zero quickly, it is
# sufficient to compute the sum of first 100 members of the sequence.


> n = 100
> p = numeric(n)
> for(i in 1:n){ p[i] <- (5/6)^(i-1)*(1/6) }
> plot(p)              # p(i)
> plot(p * 1:n)        # p(i)*i
> barplot(p*(1:n), names.arg = 1:n)

> sum(p * 1:n)         # SUM( p(i)*i )


# More R-like solution:
> sum( (5/6)^(1:n - 1)*(1/6) * 1:n )
```

```
# To compare the elapsed time:
> system.time( for(j in 1:1000){
                 for(i in 1:n){ p[i] <- (5/6)^(i-1)*(1/6) }; sum(p * 1:n)
            } )
   user  system elapsed
  0.916   0.016   0.964

> system.time( for(j in 1:1000){
                 sum( (5/6)^(1:n - 1)*(1/6) * 1:n )
              } )
   user  system elapsed
  0.028   0.000   0.028



# Another solution is based on using the enormous computational
# power of our computers. Instead of thinking about exact and
# explicite formulas we can "measure" the probability as the
# average of a large number of experimental results.


# To simulate 1,000,000 trials we can use random generator:
> n = 10^6
> x = sample(1:6, n, replace=T)

# Now we will transform the vector x to a 0/1 vector
> x = ifelse(x==1, 1, 0)

# However, the same random vector could be obtained using rbinom:
> x = rbinom(n, 1, 1/6)          # which is much faster

# And now the average number of trials needed to get 1 is the same
as the average distance between ones in x, thus the result is
> lenght(x)/sum(x)

# or simply
> n/sum(rbinom(n, 1, 1/6))
```

*** **Exercise 2** ***

Question:
If you generate 100 random integers from {1,...,100},
what is the probability that n will be generated?

```
# to generate n numbers from {1,..., n}
> n = 100
> x = sample(1:n, n, replace=T)

# to count how many times n was generated
> sum(ifelse(x==n, 1, 0))

# ... or simply
> sum(x==n)

# or
> sum( sample(1:n, n, replace=T) == n )
```

```
# to test whether n was generated or not
> any(x==n)

# now we can do e.g. 1,000,000 trials and count how
# many times n occurs in the generated random sample
> n = 100; k = 0; N = 10^6
> for(i in 1:N){ if( any(sample(1:n, n, replace=T) == n) ) k <- k+1 }

> k/N   # to estimate the probability that n is generated in the sample

[1] 0.633979

# Again, much faster solution can be obtained using rbinom
> n <- 100
> sum(rbinom(N, n, 1/n) > 0)/N

[1] 0.633823    # the same estimate obtained different way

# Of course, the obvious exact answer is simply
> n <- 100
> 1 - ((n-1)/n)^n

[1] 0.6339677   # the exact probability
```

*** **Exercise 3** ***

Question:
If you generate 10 random integers from {1,...,10},
how many different numbers will you get?

```
# To empirically estimate the probabilities we can repeat the experiment
# 1,000,000 times:
#
> n <- 10
> N <- 10^6
> x <- numeric(N)
> for(i in 1:N){ x[i] <- length(unique(sample(1:n, n, replace=T))) }

> table(x)         # the summary of results
> plot(factor(x))


# A clever solution using recursion:
# p(m,k,n) = p("there are just k different numbers among m numbers
#               randomly selected from {1,..., n}")
#             -- assuming n>=k, m>=k.

> p <- function(m,k,n){
    if(k==0 || m<k) 0 else
    if(m==1) 1 else
    p(m-1,k,n)*k/n + p(m-1,k-1,n)*(n-k+1)/n
  }
> n=10; x <- numeric(n); for(i in 1:n) x[i] <- p(n,i,n); x

 [1] 0.000000001 0.000004599 0.000671760 0.017188920 0.128595600 0.345144240
 [7] 0.355622400 0.136080000 0.016329600 0.000362880
```

```
> sum(x)   # just to check
[1] 1

> barplot(x, names.arg=1:10)
```

*** **Homework exercises** ***

1. How many times do you need to roll a die to get all different numbers?
   You need to determine the probability distribution.
   Display the histogram!

   ... and particularly:

2. What is the probability that you will get all 6 numbers after 6 rolls?

3. How many times do you need to roll a die to have (at least) 95% chance
   that you will get all different numbers?

*** **Exercise 4  (also homework)** ***

In 2001 it was found that the average male height in the Czech Republic is
180.3 cm, while the average female height is 167.2 cm...
We assume that both distributions are normal with mean=180.3 and
variance=100, and with mean=167.2 and variance=50, respectively.

Question:
When you meet a man and a woman -- independently(!), then
  What is more likely?
    * that the man will be bigger than 200 or smaller than 165?, or
    * that the woman will be between 166 and 168?

First, compute the probabilities, and then draw a picture with the density
functions and display the probabilities as areas under the density curves.

Hint: to fill the area under curve use polygon().