

Introducción (corta) a R

Qué es?

- R es un entorno para análisis de datos.
 - Es el equivalente en la comunidad estadística del Matlab-Scilab de la ingeniería.
- R incluye un lenguaje de computación INTERPRETADO.
 - La mayoría de las funciones visibles a los usuarios están escritas en R, con llamadas a un conjunto pequeño de funciones internas compiladas.
 - Es posible (y relativamente fácil) invocar procedimientos en C, C++ o FORTRAN, por eficiencia y aprovechamiento.
 - Las llamadas a Sistema se hacen también dentro de R.
- R se usa para manejo de datos, estadística y graficación. Está compuesto de:
 - operadores (+ - <- * %*% ...) para cálculos con arreglos y matrices
 - una gran, integrada y coherente colección de funciones predefinidas
 - facilidades para ilimitados tipos de gráficos (en calidad)
 - funciones y paquetes escritos por usuarios

R vs SPSS/SAS

Pros

- Libre.
- Estado del arte: Estadísticos, Bioinformáticos, etc., publican sus métodos como paquetes R. SPSS y SAS tardan años en incorporarlos.
- Excelentes gráficos
- Comunidad activa y creciente de usuarios
- Excelente para simulaciones, programación, análisis de datos, desarrollo de métodos, etc.
- Interface con SQL y librerías optimizadas de cálculo numérico.

Contras

- Lento si se usa incorrectamente
- Uso sub-óptimo de memoria
- No es user-friendly. Difícil de aprender a usar. Mínima GUI.
- Sin soporte comercial a los usuarios.
- Es fácil equivocarse en el análisis y no notarlo.
- El pre-procesado de los datos es más difícil que con SPSS/SAS

Paquetes en R

- Funciones específicas para una tarea.
 - Tipo librerías en C, C++
- CRAN: Repositorio común de paquetes R.
- Organizado en “Task views”
 - Cada una con una descripción de los paquetes principales dedicados a esa tarea.
- Cuidado: hay paquetes muy malos!

Comenzando

- Bajarlo, instalarlo y arrancar.

R es un software libre y viene sin GARANTIA ALGUNA.
Usted puede redistribuirlo bajo ciertas circunstancias.
Escriba 'license()' o 'licence()' para detalles de distribucion.

R es un proyecto colaborativo con muchos contribuyentes.
Escriba 'contributors()' para obtener más información y
'citation()' para saber cómo citar R o paquetes de R en publicaciones.

Escriba 'demo()' para demostraciones, 'help()' para el sistema on-line de ayuda,
o 'help.start()' para abrir el sistema de ayuda HTML con su navegador.
Escriba 'q()' para salir de R.

>

Básicos

- help: `>help()`
- `help(solve)`
- `?solve`
- `Help("[")`
- R es case-sensitive
- Nombres válidos: letras numeros `_` .
 `.0a` no vale como nombre, pero `.a0` si

Básicos

- ; o <enter> para terminar un comando
- Una sentencia se evalúa y se pierde
 - pero hay “ans” como matlab, es .Last.value
- Si se asigna a una variable, no se imprime por defecto
- >1+1
- >a<-1+1
- >a;a+1

Básicos

- `{ }` agrupan comandos (como en C)
- `#` comentarios
- Si un comando está incompleto cambia el prompt

```
>2*(3+1
```

```
+)
```

```
[1] 8
```

(es interpretado!)

Básicos

- Entrada desde archivo: `>source("run.R")`
- Salida a archivo `>sink("salida.out")`
- `sink()` vuelve a la pantalla
- `objects()` o `ls()` lista los objetos en memoria
- `rm(a)` borra a de la memoria
- `save(a,file="sesion.Rdata")`
 - guarda el objeto en formato binario comprimido compatible entre versiones y sistemas operativos.
- `save.image(file="sesion.Rdata")` guarda todo

Objetos

- R usa objetos que son estructuras de datos con propiedades
- Propiedades base: `length()` y `mode()`
- Datos atómicos: todos los componentes tienen el mismo `mode()`
- Datos complejos: `list()`, `data.frame()` - mezclas
- 4 tipos (modos) básicos: `numeric`, `complex`, `logical`, `character`

Asignación

- Vectores numéricos:

```
>x <- c(10.4, 5.6, 3.1, 6.4, 21.7,3)
```

- `c()` concatena objetos. MUY USADA!
- Para R un número es un vector de long. 1

```
>y<-x
```

```
>y=x
```

```
>x->y
```

Vectores

- R es un lenguaje vectorial

```
> 1/x
```

```
[1] 0.09615385 0.17857143 0.32258065 0.15...
```

```
> y<-c(2,1)
```

```
> x+y
```

```
[1] 12.4 6.6 5.1 7.4 23.7 4
```

Warning message:

In x + y : longitud de objeto mayor no es múltiplo de la longitud de uno menor

El objeto de menor tamaño se promociona al de mayor repitiéndolo!

Vectores

- Operadores y funciones típicas:

$+$, $-$, $*$, $/$, $^$, \log , \exp , \sin , \cos , \tan , $\sqrt{}$, etc.

- Otras funciones útiles:

\min , \max , sort , mean , sd , var

- Complejos:

```
> sqrt(-1)
```

```
[1] NaN (+ un warning)
```

```
> sqrt(-1+0i)
```

```
[1] 0+1i
```

Valores especiales

- NaN: not a number
- NA: Valor faltante. (NaN es un NA)

```
> is.na(NaN)
```

```
[1] TRUE
```

```
> is.nan(NA)
```

```
[1] FALSE
```

- Inf y -Inf:

```
> -1/0
```

```
[1] -Inf
```

- Lógicos: TRUE, T, FALSE, F, NA

Secuencias

- 1:4 es c(1,2,3,4)
- 4:1
- 2*1:4 (distinto a matlab, creo)
- seq(in,fin,paso) para más modos, ver help

```
> y
```

```
[1] 2 1
```

```
> rep(y,3)
```

```
[1] 2 1 2 1 2 1
```

```
> rep(y,each=3)
```

```
[1] 2 2 2 1 1 1
```

Indices!!!

- Una de las capacidades más interesantes
- `X[]` permite acceder a ciertos elementos de `x` solamente
- Dos usos:
 - Filtrado: `a<-x[lista]`
 - Asignación `a[lista]<-x`

Indices

- 4 tipos:
 - Lógicos: `x[x>10]`
 - Enteros positivos `x[1:2]` (elige los listados)
 - Enteros negativos `x[-(1:2)]` (todos menos esos)
(sin el paréntesis?)
 - Nombres de columnas:

```
> y<-c(y1=-1,y2=3)
```

```
> y
```

```
y1 y2
```

```
-1  3
```

```
> y["y2"]
```

```
y2
```

```
3
```

Más...

- Los vectores crecen dinámicamente, si asignamos fuera de lo declarado.

```
> y[4]<-2
```

```
> y
```

```
y1 y2
```

```
-1 3 NA 2
```

- Lo no declarado queda vacío (NA)

Práctica

- Genere las secuencias:
 - 1 a 30, solo los múltiplos de 3
 - 1 a 30, menos los múltiplos de 3
- Cree un vector x con 10 valores en $[0,1]$
 - A los $x > 0.5$ restarles 1
 - Agregue 10 valores en $[1,2]$
 - Calcule la media, el máximo y el mínimo de x
 - Ordene los $x < 0$

Práctica

- Calcule:
 - `sum(x>0.5)`
 - `sum(x>2)`
 - `sum(x[x>0.5])`
 - La media del `log()` de x
 - Igual pero bien :)

Arreglos

- Son vectores con dimensión (atributo dim())

```
> a<-1:8
```

```
> a
```

```
[1] 1 2 3 4 5 6 7 8
```

```
> dim(a)<-c(2,4)
```

```
> a
```

```
 [,1] [,2] [,3] [,4]
```

```
[1,]  1  3  5  7
```

```
[2,]  2  4  6  8
```

```
> dim(a)<-c(2,2,2)
```

```
> a
```

Arreglos

- Se aplican las mismas reglas para subindices (con comas)

>a[2,-1,] (indice en blanco es todo)

- Siguen siendo vectores:

>a[3:5]

- `matrix()` para crear matrices (2 dim)
- `array()` para más dimensiones

Perdida de dimensiones!

```
>dim(a)
```

```
>dim(a[2,,])
```

```
>dim(a[2,2,])
```

```
>dim(a[2,2,,drop=F])
```

La perdida de dimensiones es una de las mayores fuentes de bugs en R!

Algebra de matrices

- Productos (A y B matrices del mismo orden)

>A*B

>A%*%B

>t(A)

>solve(A)

>diag(A)

>B<-diag(2) #ver diag(0,2)

>eigen(A)

- Generar dos matrices random y aplicar todo

Listas

- Objetos “contenedores” - muy útiles

```
>a<-list(x=c(1:2),y=FALSE,tt="text")
```

```
>a
```

```
>a$y
```

```
>a[[2]]
```

```
>a[[c(2,1)]] #extraccion recursiva
```

```
>a[2] ( apliquen mode() )
```

```
>a[-1]
```

- Los “objetos” de una clase en R son listas con atributo clase.

Data Frame

- Los data.frame son listas con restricciones
 - Vectores de igual longitud
- Son prácticamente matrices, pero con columnas potencialmente de distintas clases
- Son el lugar tradicional para guardar nuestros datos.

Importando datos

- Función `read.table()`
 - Da como respuesta un `data.frame`
 - Lee “casos” en filas, “variables” en columnas
 - Ver valores por defecto para separador y decimal
 - Pasa los caracteres a factores
- Otras:
 - `read.csv()` #para excel
 - `read.spss()` #library(foreign)

Factores

- Son tipos especiales de variables que toman solo un número discreto de valores distintos: “clases” para discriminar algo

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

```
>data(iris)
```

```
>iris[,5]
```

(data()) accede a los datasets disponibles)

Funciones

`name <- function(arg_1, arg_2, ...) expression`

- Valores por defecto: `arg_1=1` en la declaración
- `return(a)` o `a` como última línea de la f.
- Argumentos: se llenan por orden o usando parte del nombre:
 - `seq(1,3)` es `seq(from=1,to=3)`
 - `seq(-3,by=0.5)` es `seq(from=-3,to=1,by=0.5)`

Funciones – Scope de variables

3 tipos de variables: parametros de la función, variables locales y variables globales.

```
>cuad<-function(x) x^2
```

```
>cuad(2)
```

```
>cuad<-function() z^2
```

```
>cuad(3)
```

```
>z=3;cuad()
```

```
>cuad<-function(x) {z=x;z^2}
```

```
>cuad(2)
```

Funciones útiles

`sample(x, size, replace = F, prob = NULL)`

- `sample(1:10)`
- `sample(1:10,rep=T)`
- `sample(1:10,5,rep=T,prob=c(9,rep(0.1,9)))`

`rnorm(n, mean = 0, sd = 1)`

`runif(n, min=0, max=1)`

Funciones útiles

`table()`

- `a<-c(1,1,1,2,2,2,2,3,3,3,3,3)`
- `table(a)`
- `table(a,sample(a))`

`rbind()` y `cbind()`

- `cbind(diag(2),diag(2))`
- `rbind(diag(2),diag(2))`

Funciones útiles

- `apply(X,margin,FUN)`
 - Ej: Máximo de cada columna de una matriz

```
>apply(a,2,max)
```

```
>a<-list(x=c(1:2),y=FALSE,tt='text')
```

```
>is.na(a)
```

```
>lapply(a,is.na) #apply para listas
```

- `rep()` y `seq()`

```
>rep(1:5, 4:0)
```

Funciones útiles

- Conversión: `as.factor()`, `as.matrix()`, `as.list()`, etc
- `length()`
 - `length(1:4)`
 - `length(as.list(1:4))`
 - `length(diag(2))`
- `dim()`
 - `dim(1:4)`
 - `dim(diag(3))`
 - `dim(as.data.frame(diag(3)))`

Control de ejecución

- For(i in lista) {...}
 - Lista<-1:10 o c(1,5,10:5) o cualquier cosa
- If (logic) { } else { }
 - & y | , && y ||

```
> c(T,T)&c(F,T) #pairwise
```

```
[1] FALSE TRUE
```

```
> c(T,T)&&c(F,T) #para decidir
```

```
[1] FALSE
```

```
> if( (a<-T)|| (b<-F)) {print(a);print(b)} #error
```

Plots

```
>data(iris)
```

```
>plot(iris)
```

```
>plot(x=iris[,2],type="b")
```

```
>plot(iris[, -5], col=iris[, 5])
```

```
>hist(x=iris[,3])
```

```
>boxplot(iris[, -5])
```

— Más por venir...

Estilo R

- R es interpretado, no compilado
- Los for son muy lentos. Siempre se prefiere usar los índices. Casi todas las operaciones sobre una matriz se pueden hacer con los índices.

Para poner en 0 todos los elementos negativos de la matriz a:

```
>for(i in 1:dim(a)[1]) for(j ...) if(a[i,j]<0) a[i,j]=0  
>a[a<0]<-0
```

Modelos en R

- El lenguaje apunta a crear “modelos”
 - Ejemplos: modelos lineales, ANN, k-nn, trees
- Se crean clases para cada modelo. Hay funciones que tienen que estar siempre:
 - Constructor – lo crea y lo ajusta
 - `predict()` - usa el objeto para predecir casos
 - `print()` - lo muestra
 - `plot()` - `summary()` - `table()`

Modelos en R

```
>library(class) #clasificadores - knn  
>library(rpart) #arboles de decisión  
>data(iris)  
>summary(iris)  
>train<-sample(1:dim(iris)[1],100) #creo un indice para train  
>help(knn)  
>mod.knn<-knn(iris[train,-5],test=iris[-train,-5],cl=iris[train,5],k=3)  
>mod.knn  
>sum(mod.knn==iris[-train,5])
```

Modelos en R

Interfaz de fórmula: Lenguaje estadístico.

Para hacer un modelo en el que y es una función lineal de x_1, x_2 , se escribe: $y \sim x_1 + x_2$

Pero es mucho más complicado.

Casi no se usa actualmente, pero algunas funciones base lo usan.

Yo creo un dataset con todas las X y al final la clase (en una columna con nombre), y uso la forma simple:

```
>mod.tre<-rpart("Species~.",data=iris,sub=train)
```

```
>predict(mod.tre,iris[-train,-5])
```


Práctica

- Crear una función que retorne:
 - n datos en p dimensiones (n y p entradas)
 - Muestreos de dos normales – `rnorm()`
 - Incluir la clase como factor
- Hacer plots
- Aplicarles algún clasificador

Práctico 1

- Crear una función que haga el datos-diagonal
 - `rnorm()`
- Crear una función para las dos elipses
 - `runif()`
- Hacer plots
- Aplicarles knn y rpart