El entorno R (II)

Una nueva generación de software estadístico gratuito

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

Manuel Febrero Bande / Beatriz Pateiro López

Funciones.

- Las funciones en R se definen con el formato:
  `nombre<-function(arg1=val,arg2=val2,…){expr}`
- No se definen los tipos de los argumentos.
- No se hace ningún chequeo sobre los parámetros hasta que se usan.
- Los argumentos de entrada pueden ponerse en cualquier orden si se especifica el nombre. Otra opción es introducirlos en el orden en el que fueron definidos sin nombre.
- Se pueden asignar valores por defecto a los parámetros. Cualquier argumento que no tenga valor debiera ser obligatorio.
- Si una variable no está definida dentro de la función, se busca en el ámbito de la función que la llamó y así sucesivamente hasta que se llega al entorno global.
- Se pueden incluir fórmulas en los argumentos de entrada. No se ejecutan hasta que se usa la variable.

Funciones.

- La función devuelve el último objeto evaluado dentro de la función o lo que indiquemos en la sentencia `return`. Habitualmente se devolverá una lista. Puede devolverse cualquier objeto de R.
- Cualquier cambio dentro de la función se perderá sino se devuelve al salir. Si queremos un cambio permanente usaremos `<<-`. Esto modifica la variable en el entorno global.
- La expresión … sirve para pasar argumentos a otras funciones internas de forma libre. Esto indica aquellos argumentos a los que no se les conoce el nombre en la definición.

```
label=function(x)
  {list(value=x,actual=substitute(x))}
label(1+1)        # ¿Qué se devuelve?
```

Funciones. Ejemplo

- ```
  mi.func<-function(a,b=3,…){
  +      z<-a^b;plot(z,…)
  +        return(z)}
  ```
- `mi.func(3,3) # 27`
- `mi.func(b=2,a=5,col=2)`
- `mi.func(seq(0,1,length=100))`
- `mi.func(b=2,3) # Error`
  ```

Loops e iteraciones

- Debe evitarse en lo posible los bucles (punto débil en R) usando cuando se pueda los cálculos vectorizados

```
for (i in 1:n) {d[i]=a[i]*b[i]};s=0;for (i in 1:n) {s=s+d[i]}
s=sum(a*b)
```

- Las funciones que admiten vectorización incluye: Operadores, Funciones matemáticas, Generadores aleatorios, Lógicas, etc.
- La vectorización se hace usando “reciclado de vectores”(matrices).
  `a=c(1,2,3,4);b=c(1,2); c=a*b=c(1,4,3,8)`

Loops e iteraciones

- `for (name in expr1) {expr2}`
  `#expr1` puede ser cualquier tipo de vector `(1:n, c(3,7),c(sin,cos))`
- Si `expr1` tiene longitud cero no se ejecuta `expr2`.
- `break, next` # rompe el bucle o pasa a la siguiente iteración.
- `while(expr_log) {expr}`
- `repeat {expr}` #Sólo usado en demos
- El uso de los comandos de la familia `apply` es mucho más rápido. `dim(iris3);apply(iris3,c(2,3),mean)`
- Truco: Dimensionar antes.

Sentencias de control

- `if (expr_log) {expr2} else {expr3}`
  #Clásico y se puede anidar
- `ifelse(expr_log,{cierto},{falso})`
  #Version vectorizada
- `switch(expr,lista)`

Datos de ejemplo

- En el paquete `datasets` (cargado por defecto) se encuentran un montón de conjunto de datos de ejemplo.
- Se hacen accesibles mediante la instrucción `data(`*`cjto`*`)`.
- En muchos otros paquetes también aparecen conjuntos de datos interesantes (p.ej.. `DAAG, Devore, ElemStatLearn, …`)

Distribuciones y generación de números aleatorios

- En general para todas las distribuciones presentes en R tenemos cuatro funciones: p*nombre*, q*nombre*, d*nombre*, r*nombre* que devuelven respectivamente distribución, función cuantil, densidad o probabilidad y generación de números aleatorios.
- `x<-rnorm(100)`
  `curve(dnorm(x),from=-3,to=3,lwd=2,col="green")`
  `lines(density(x),lwd=2,col="red")`
- `plot(ecdf(x))`
  `curve(pnorm(x),from=-3,to=3,lwd=2,add=TRUE)`
- `qnorm(0.975)` # 1.96
- `sample()` # Devuelve una muestra con/sin reemplazamiento → Bootstrap
- Distribuciones: `beta, binom, cauchy, chisq, exp, f, gamma, geom, hyper, logis, lnorm, multinom, nbinom, norm, pois, signrank, t, tukey, unif, weibull, wilcox`
- Otros paquetes disponen de más generadores, por ej. `rnormalp` en `normalp` o `mvrnorm` en `MASS`.

Generación de números pseudo-aleatorios

- La generación de números pseudo-aleatorios es probablemente la mejor del mercado. Se usa la función `RNGking()` para establecer el tipo de generador
- "Wichmann-Hill": Ciclo: *6.9536e12*
- "Marsaglia-Multicarry": Ciclo: *2^60*
- "Super-Duper": Ciclo: *4.6*10^18*
- "Mersenne-Twister": Ciclo: *2^19937 - 1* y equidistribution en 623 dim.
- "Knuth-TAOCP": Ciclo: *2^129*.
- "Knuth-TAOCP-2002
- "user-supplied"

Generación de la normal: "Kinderman-Ramage", "Buggy Kinderman-Ramage", "Ahrens-Dieter", "Box-Muller", "Inversion" , o "user-supplied"

Regresión Lineal

- `lm(formula,data, subset, weights,…)`
- Ej.: `z<-lm(Ozone~Temp,data=airquality, subset={Month==5})`
- `summary(z).` # devuelve los contrastes habituales
- `plot(z)` # Realiza ciertos gráficos de control
- `abline(z)` # Dibuja la recta (si puede o tiene sentido)
- `coef(z)` y `residuals(z)` # Devuelve coeficientes y residuos
- `fitted(z)` y `vcov(z)` # Valores ajustados y matriz de varianzas-covarianzas de los parámetros
- `influence(z)` y `predict(z,newdata)` # Influencia y predicción
- `z` es una lista que contiene: `"coefficients", "residuals","effects","rank", "fitted.values", "assign", "qr", "df.residual","na.action", "xlevels", "call", "terms", "model"`

Regresión Lineal (Ejemplo)

```
attach(airquality);airq=na.omit(airquality)
pairs(airq[,1:4]);z=lm(Ozono~Temp,data=airq)
z2=lm(Ozone~.,data=airq)
z3=update(z2,.~.-Month-Day)
summary(z3);anova(z,z2,z3)
plot(z3,id.n=5,which=1)
z3.in=influence(z3)$coef
l=which(abs(z3.in[,4])>0.04);airq[l,]
boxplot(airq[,1:2])
points(rep(1,length(l)),airq[l,1],col=2)
points(rep(2,length(l)),airq[l,2],col=3)
segments(rep(1,length(l)),airq[l,1],rep(2,length(l))
  ,airq[l,2],col=2,lwd=2)
```

Modelos Lineales Generalizados

- `glm(formula,data, family=gaussian,…)`
- Ej.: `zlg<-glm(I(Ozone<40)~Temp,data=airq, family=binomial)`
- `summary(zlg).` # devuelve los contrastes habituales
- `plot(zlg)` # Estos gráficos pueden ser ahora inútiles
- `coef(zlg), residuals(zlg), fitted(zlg), vcov(zlg),influence(zlg),predict(zlg,newdata)`
- `family={binomial,gaussian,Gamma,inverse.gaussian,poisson,quasi,quasibinomial,quasipoisson}` con varios links posibles.

Modelos Aditivos

`library(mgcv) #incluida en la instalación básica`

- `gam(formula,data, family=gaussian,…)`
- Ej.: `zg<-gam(Ozone~s(Temp)+s(Solar.R), data=airq)`
- `summary(zg).` # devuelve los contrastes habituales
- `plot(zg,shade=TRUE)` # Estos gráficos
- `coef(zg), residuals(zg), fitted(zg), vcov(zg),influence(zg),predict(zg,newdata)`
- `family={binomial,gaussian,Gamma,inverse.gaussian,poisson,quasi,quasibinomial,quasipoisson}` con varios links posibles.

- ¡¡ Similar al glm !!. La principal diferencia está en el operador suave.
- Hay otros paquetes que también hacen modelos aditivos.

Regresión no paramétrica

- Paquete `KernSmooth`: `locpoly(x,y,bandwidth,drv=0)`
- Ej.: `est<-locpoly(Temp[!is.na(Ozone)], Ozone[!is.na(Ozone)],bandwidth=2.4)`
- `dpill(x,y).` # devuelve ventana plug-in
- `plot(est,type="l")` #Dibuja la línea de regresión
- `loess(y~x)` # Otro estilo de regresión no paramétrica.
- `smooth.spline(x,y)` # Regresión spline
- `supsmu(x,y)` y `ksmooth(x,y)` # Super Smoother de Friedman y Nadaraya-Watson
- Otros paquetes tienen más funciones para el cálculo de la ventana. Revisar documentación de `MASS, locfit` o `lokern` o usar `help.search("bandwidth")`

Regresión no paramétrica (ejemplo)

```
library(KernSmooth);airq=na.omit(airquality[,1:4])
attach(airq);h=dpill(Temp,Ozone)
est=locpoly(Temp,Ozone,bandwidth=h,drv=0)
est3=locpoly(Temp,Ozone,bandwidth=h, drv=0,deg=3)
plot(Temp,Ozone,main=paste("Ventana: ",format(h,4)))
lines(est,col=2,lwd=2); lines(est3,col=3,lwd=2)
lo=loess(Ozone~Temp,data=airq,span=0.3);xr=range(Temp)
lines(seq(xr[1],xr[2],len=100),predict(lo,data.frame(Temp=
  seq(xr[1],xr[2],len=100))),col=4,lwd=2)
lines(smooth.spline(Temp,Ozone),col=5,lwd=2)
lines(supsmu(Temp,Ozone),col=6,lwd=2)
lines(ksmooth(Temp,Ozone,bandwidth=h),col=7,lwd=2)
legend(("topleft"),c("Lineal", "Cubica", "loess",
  "Sspline", "SSFriedman", "NW"), col=2:7,lwd=2)
```

Análisis de supervivencia

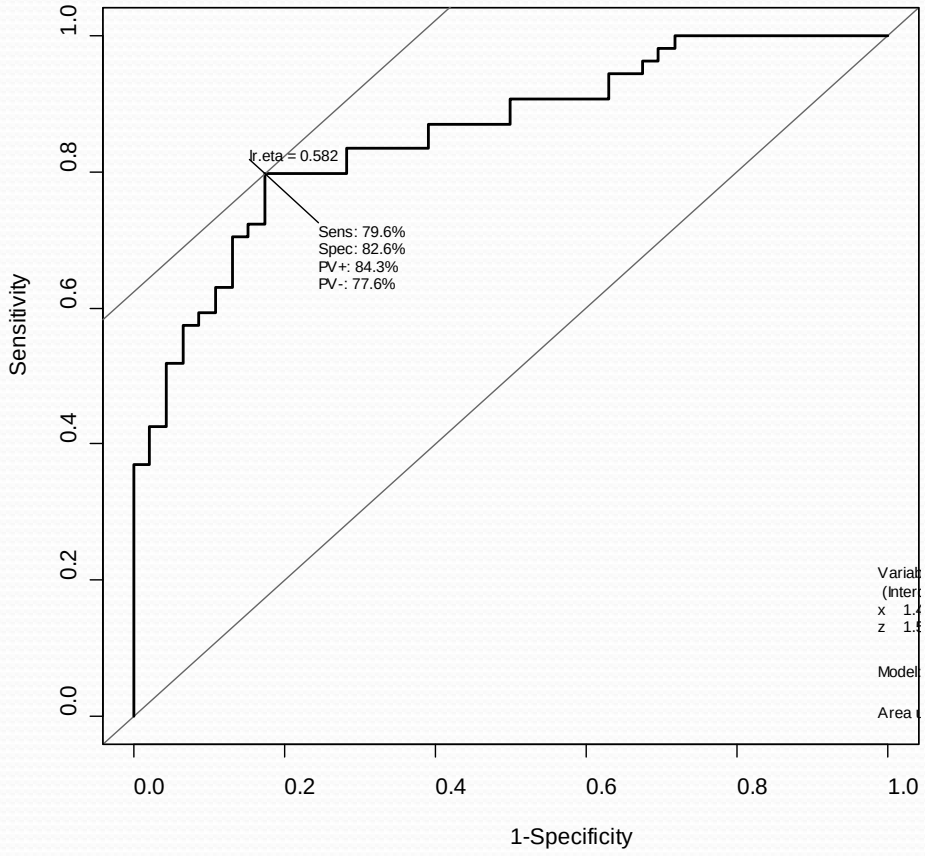
El paquete `survival` tiene funciones para tratar con datos censurados. El objeto `Surv` nos permite varios tipos de censura.

```
library(survival);data(ovarian)
rkm=survfit(Surv(futime,fustat)~rx,data=ovarian)
plot(rkm)
rcox=coxph(Surv(futime,fustat)~rx+age,data=ovarian);
  plot(survfit(rcox)) #Modelo Cox proporcional
```

- La función `survreg` incluye otros modelos paramétricos
- El objeto de la clase `survfit` tiene las siguientes componentes:
  `"n","time","n.risk","n.event","surv","type", "ntimes.strata","strata","strata.all","std.err","up per","lower","conf.type","conf.int“`

Curvas ROC y Epidemiología

Paquetes: `Epi, epicalc, PresenceAbsence`



Componentes principales

- Función princomp que admite varios formatos y devuelve un objeto de la clase princomp.

```
pc.cr=princomp(na.omit(airquality[,1:4]),cor=T)
```

- `summary(pc.cr).` # devuelve tabla con resumen de PC
- `plot(pc.cr)` #Gráfico de barras de varianza por componente
- `loadings(pc.cr)` # Matriz de autovalores
- `biplot(pc.cr)` # Biplot de las componentes principales
- El objeto de la clase princomp tiene las siguientes componentes: `"sdev", "loadings", "center", "scale", "n.obs", "scores", "call"`
- Análisis factorial: `factanal()` permite rotaciones `varimax()` y `promax().` Se pueden definir funciones personalizadas.

Análisis discriminante

- En el paquete MASS tenemos las funciones `lda()` y `qda()` para análisis discriminante lineal y cuadrático.
- `data(iris);dl<-lda(Species~.,data=iris)`
- `ql<-qda(Species~., data=iris)`
- Admiten los procedimientos `predict()` útiles para determinar clasificar nuevas observaciones.
- `plot(dl)` # Gráfico de pares de las componentes discriminantes
- El procedimiento cuadrático no admite plot predeterminado y habría que hacerlo a mano.
- Otra posibilidad es usar redes neuronales MLP (`library(nnet)`)
- ```
  samp <- c(sample(1:50,25), sample(51:100,25),
  sample(101:150,25));
  ir.nn2 <- nnet(Species ~ ., data = iris, subset = samp,
  size = 2, rang = 0.1, decay = 5e-4);
  table(iris$Species[-samp], predict(ir.nn2, iris[-samp,],
  type = "class"))
  ```
- Otros paquetes disponen de mas rutinas, por ej. `mda, ade4 o fpc.`
  ```

Análisis cluster

- El análisis cluster jerárquico se realiza mediante
- `data(USArrests);`
  `hc<-hclust(dist(USArrests));plot(hc)`
- `cutree(hc,k=4) # Permite seleccionar k grupos`
- `rect.hclust(hc,k=4)` # Selecciona en el gráfico tantos clusters como indique k.
- `identify(hc)` # Permite seleccionar con el ratón un cluster
- métodos de aglomeración: `"ward", "single", "complete", "average", "mcquitty", "median"` o `"centroid"`.
- También está disponible el comando `kmeans()`.
- `cl<-kmeans(iris[,1:4],3,20);`
  `plot(iris[,1:2], col=cl$cluster);`
  `points(cl$centers[,1:2], col = 1:3, pch = 8)`
- `help.search("cluster")` proporciona información de muchos otros paquetes con rutinas más complejas.

Debugging

- Cuando las cosas no van bien debemos saber por qué. –traceback()–
- Ejemplo cabrón:
  `outer(2:5,100*(1:5),function(k,d)pbirthday(n=50,classes=d,coincident=k))` # ¿Dondé está el problema?
- `traceback()`

```
5: stop("f() values at end points not of opposite
   sign")
4: uniroot(f1, lower = 0, upper = upper, tol = eps)
3: pbirthday(n = 50, classes = d, coincident = k)
2: FUN(X, Y, ...)
1: outer(2:5, 100 * (1:5), function(k, d)
   {pbirthday(n = 50, classes = d, coincident = k)})
```

Debugging

- `browser(),debug(fn),undebug()`

```
> debug(uniroot)
> outer(2:5,100*(1:5),function(k,d)
  {pbirthday(n=50,classes=d,coincident=k)})
debugging in: uniroot(f1, lower = 0, upper = upper,
  tol = eps)
  debug: …}
Browse[1]> f(lower)
[1] -49
Browse[1]> f(upper)
 [1] -49 -43 -31 -14 -49 -39 -19  13 -48 -36  -8  37
  -48 -33   3  59 -48 -30  12  80
Browse[1]> Q
```

- Problema: Ambos vectores no tienen la misma dimensión

Midiendo velocidades

- `system.time(fn)` # user time, CPU time, elapsed time, ?,?

```
Rprof("birthprof");for(i in 100:10000) a[i]<-
   qbirthday(0.5,classes=i)
Rprof(NULL);summaryRprof("birthprof")
```

```
$by.self
          self.time self.pct total.time total.pct
qbirthday      0.06     27.3       0.22     100.0
log            0.06     27.3       0.06      27.3
+              0.04     18.2       0.04      18.2
*              0.02      9.1       0.02       9.1
gamma          0.02      9.1       0.02       9.1
lgamma         0.02      9.1       0.02       9.1
$by.total
          total.time total.pct self.time self.pct
qbirthday       0.22     100.0      0.06     27.3
log             0.06      27.3      0.06     27.3
+               0.04      18.2      0.04     18.2
*               0.02       9.1      0.02      9.1
gamma           0.02       9.1      0.02      9.1
lgamma          0.02       9.1      0.02      9.1
$sampling.time
[1] 0.22
```

Mezclando C y Fortran con R

- La opción más simple es crear una DLL y llamarla desde R.
  `dyn.load("fichero.dll")`
- Precauciones:
  Exportación nombre rutinas:

```
!DEC$ ATTRIBUTES DLLEXPORT:: DENSIDAD
!DEC$ ATTRIBUTES C, REFERENCE, ALIAS:'densidad_' :: DENSIDAD        #CVF 6.1
```

  No usar escritura en pantalla desde rutina DLL

  Conversión entre tipos de datos

R storage mode	C type	FORTRAN type
logical	int *	INTEGER
integer	int *	INTEGER
double	double *	DOUBLE PRECISION
complex	Rcomplex *	DOUBLE COMPLEX
character	char **	CHARACTER*255

Ejemplo (Mezclando R y Fortran)

```
dyn.load("Densidad.dll") #
   http://eio.usc.es/pub/febrero/Densidad
is.loaded("dens2d")
n=100
x=rnorm(n)
y=rnorm(n)
 h=c(.5,.5)
 npt=51
 bx=seq(-3,3,len=npt)
 by=bx
 zxy=matrix(0.0,ncol=npt,nrow=npt)
res=.Fortran("dens2d",n=as.integer(length(x)),x=as.double(x),
   y=as.double(y),h=as.double(h), npt=as.integer(npt),
   bx=as.double(bx),by=as.double(by),zxy=matrix(as.double(zxy)
   ,npt,npt))
filled.contour(res$bx,res$by,res$zxy)
```

Compilación cruzada

- Sólo se pueden llamar subrutinas de Fortran o Funciones void de C. (jamás funciones)
- El paso contrario también se puede hacer. Si queremos incluir objetos de R en C en el directorio `include` están los ficheros adecuados.
- El proceso de creación de DLL cambia según el compilador y es dependiente del sistema que se use. (Distinto Windows de Linux)
- Para crear extensiones de R universales se puede usar `R CMD SHLIB` que admite código en distinto formato (`C, C++, FORTRAN 77, Fortran 9x`) que será compilado al instalar el paquete. (Mingw32)
- El fichero de definiciones para compiladores está es `Makeconf` en el directorio `etc`.

Ejemplo (Mezclando R y Fortran)

```
dyn.load("Densidad.dll") #
   http://eio.usc.es/pub/febrero/Densidad
is.loaded("dens2d")
n=100
x=rnorm(n)
y=rnorm(n)
 h=c(.5,.5)
 npt=51
 bx=seq(-3,3,len=npt)
 by=bx
 zxy=matrix(0.0,ncol=npt,nrow=npt)
res=.Fortran("dens2d",n=as.integer(length(x)),x=as.double(x),
   y=as.double(y),h=as.double(h), npt=as.integer(npt),
   bx=as.double(bx),by=as.double(by),zxy=matrix(as.double(zxy)
   ,npt,npt))
filled.contour(res$bx,res$by,res$zxy)
```

Más mezclas con R.

- Uno de los proyectos más interesantes es `R (D)COM Server` que permite usar R desde proyectos Visual Basic de Excel (y guardar los resultados en Excel).
- `rJava` permite mezclar R y Java.
- `CGIwithR, Rweb` permite usar una página web como servidor de R. Los resultados se pueden escribir con `RHTML`.
- `Rcmdr:` Si quereis algo más parecido a SPSS
- Proyectos relacionados:
  www.bioconductor.org, www.omegahat.org
  Basados en R, son desarrollos especiales para Genómica y Biomatemática.

Configuración de R

- En el directorio `etc` están varios ficheros de configuración
- `Rprofile.site`: Contiene los comandos que se ejecutan al iniciar R.
- `Rdevga`: Sobre fuentes a usar en gráficos
- `Rconsole`: Sobre aspecto gráfico
- `rgb.txt`: Definiciones de colores

Si se quieren versiones personalizadas para los usuarios deben incluirse los ficheros apropiados en el directorio %R_USER%

Paquetes por defecto

KernSmoothFunctions for kernel smoothing for Wand & Jones (1995)
MASSMain Package of Venables and Ripley's MASS
baseThe R Base Package
bootBootstrap R (S-Plus) Functions (Canty)
classFunctions for Classification
clusterFunctions for clustering (by Rousseeuw et al.)
datasetsThe R Datasets Package
foreignRead data stored by Minitab, S, SAS, SPSS, Stata, ...
grDevicesThe R Graphics Devices and Support for Colours and Fonts
graphicsThe R Graphics Package
gridThe Grid Graphics Package latticeLattice Graphics
mgcvGAMs with GCV smoothness estimation and GAMMs by REML/PQL
nlmeLinear and nonlinear mixed effects models
nnetFeed-forward Neural Networks and Multinomial Log-Linear Models
rpartRecursive Partitioning
spatialFunctions for Kriging and Point Pattern Analysis
splinesRegression Spline Functions and Classes
statsThe R Stats Package stats4Statistical functions using S4 classes
survivalSurvival analysis, including penalised likelihood.
tcltkTcl/Tk Interface
toolsTools for Package Development
utilsThe R Utils Package

Más información

- La ayuda de R viene en ficheros pdf en la distribución
- http://www.r-project.org
  Sección Documentation/Manuals
  Sección Documentation/Contributed
- "Statistical Models in S". Chambers, J.M. & Hastie, T.J. 1992
- "Elements of Computational Statistics". Gentle, J.E. 2002
- "Modern Applied Statistics with S-Plus". Venables, W.N. & Ripley, B.D. 1994
- "S Programing". Venables, W.N. & Ripley, B.D. 1994
- "Estadística Aplicada con S-Plus". Ugarte, M.D. & Militino, A.F. 2001
- "Introductory Statistics with R". Dalgaard, P.
- y por supuesto, Gooooooogle.....