PREDICCIÓN MEDIANTE ÁRBOLES PRÁCTICA

Funciones en R para arboles de predicción

Paquetes `rpart, rpart.plot`. Funciones para particiones recursivas (árboles de clasificación y regresión). El segundo paquete proporciona maneras más elegantes de dibujar los árboles de predicción obtenidos con `rpart`. Hay también las extensiones `rpartOrdinal`, `rpartScore`, para datos ordinales, que no usaremos en esta práctica.

Paquete `tree`. Otro paquete para particiones recursivas, de funcionalidad parecida a `rpart.` Más básico y más antiguo: su autor es también coautor de Modern Applied Statistics with S, el primer libro sobre el programa S, antecesor de R.

Paquetes `party` y `partykit`: herramientas para creación y representación de árboles.

Funciones de otros paquetes

Para visualizar interactivamente la superficie del ejemplo en 3D hay que cargar los paquetes de manejo de objetos gráficos:

```
library(tcltk)
library(tkrgl)
library(tkrplot)
```

Datos

Datos de ficheros externos (del Campus Virtual):

- Datos de supervivientes del naufragio del Titanic: `Titanic.data.txt`. Estos datos aparecen en el paquete datasets de R, bajo el nombre `Titanic`, en forma agregada, como una tabla de frecuencias de cuatro dimensiones. Aquí aparecen desplegados con un registro por cada uno de los 2201 pasajeros.
- Conjunto de datos artificiales, sobre posibilidad de practicar un deporte -tenis o golf- al aire libre, según las condiciones meteorológicas: `Golf.Dataset.csv`. Estos datos se han empleado en varios textos y también aparecen, en otro formato, en el paquete `partykit`: `data(WeatherPlay)`.
- Conjunto de datos artificiales para clasificación de animales en mamíferos, reptiles, etc., a partir de varias características somáticas o de hábitat. Hay dos versiones, una más pequeña, `Vertebrate.Dataset.txt`, y una más grande, `Zoo.Dataset.txt`.

En el paquete datasets, de la instalación básica de R:

- Datos de las flores iris: `data(iris)`.
- Datos de supervivientes del naufragio del Titanic: `data(Titanic)`.

En el paquete MASS. Instalar con: `library(MASS)`

- Datos sintéticos: `data(synth.tr), data(synth.te)`.

  Convertir la variable `yc` en un factor (viene codificada con los valores numéricos 0/1) para que el programa la pueda tratar como una etiqueta para clasificar:

  ```
  synth.tr$yc<-factor(synth.tr$yc)
  ```

  ```
  synth.te$yc<-factor(synth.te$yc)
  ```

- Características y comportamiento de CPU's de ordenadores: `data(cpus)`.

Eliminar la primera variable, que es la etiqueta de la marca. Si se desea, puede conservarse como nombre de las filas. Eliminar también la última variable, que es una predicción de la variable respuesta hecha por los autores del artículo original:

```
labels<-cpus[,1]
cpus<-cpus[,2:8]
rownames(cpus)<-as.character(labels)
```

Una transformación logarítmica de estos datos:

```
logcpus<-cpus[,1:7]
logcpus$logperf<-log10(cpus$perf)
```

- Fragmentos de vidrio recogidos en trabajos forenses: `data(fgl)`.
- Datos clásicos de color de ojos y del pelo en 5387 niños de Caithness (Escocia): `data(caith)`.

En el paquete `ElemStatLearn`. Instalar con: `library(ElemStatLearn).`

- Datos de enfermedad coronaria:

  ```
  data(SAheart)
  ```

  Convertir la variable chd en un factor (viene codificada con los valores numéricos 0/1):

  ```
  SAheart$chd<-factor(SAheart$chd)
  ```

  Convertir la variable `famhist` a numérica (viene como un factor). Esto no es estrictamente necesario para árboles de predicción, pero sí para comparar resultados con otras técnicas.

  ```
  SAheart$famhist<-as.numeric(SAheart$famhist).
  ```

- Datos de email (distinguir spam): `data(spam).`

Árboles de regresión y clasificación (CART)

La función `rpart` (recursive partitioning), del paquete de igual nombre, implementa el algoritmo de Breiman y Friedman, árboles de clasificación y regresión (CART). Sus características son: (a) capacidad de predicción de una respuesta numérica contínua, regresión, y (b) capacidad de tratar predictores de carácter numérico contínuo. Además, (c) en CART la cantidad que se optimiza en cada partición es el *índice de impureza de Gini,* en vez de la ganancia de información, como en ID3, o el estadístico Ji cuadrado, como en CHAID. La implementación `rpart` tiene además, como opción, la ganancia de información, lo que permite usar esta función para reproducir el comportamiento del algoritmo ID3, como vemos más abajo.

Aparte de esto, el algoritmo CART funciona realizando particiones sucesivas del conjunto de entrenamiento, en cada etapa según valores de una variable predictora, la que optimiza el índice de impureza. Para procesar un predictor numérico contínuo se realiza un barrido del intervalo de valores, localizando así el punto de su interior donde el índice alcanza su extremo en este predictor. La partición se hace según la variable para la que este extremo es el mejor entre todas las variables. Observad que ahora es posible encontrar, a lo largo de la sucesión de particiones en una realización del algoritmo, dos o más particiones según la misma variable.

Los siguientes ejemplos ilustran este funcionamiento. Observad que las funciones de predicción son localmente constantes (funciones escalonadas).

Regresión con un predictor continuo:

```
library(rpart)
x<-seq(-3,3,by=0.1)
y<-x^2
plot(x,y, "l",lwd=2,col="blue")
```

Mostrar el árbol:

```
r<-rpart(y~x)
plot(r,branch=0.5)
```

```
text(r,use.n=TRUE,xpd=2)
```

Visualizar la función de predicción:

```
yp<-predict(r,newdata=data.frame(x))
plot(x,y, "l",lwd=2,col="blue")
lines(x,yp,lwd=2,col="red")
```

Regresión con dos predictores continuos:

```
require(tcltk)
require(tkrgl)
require(tkrplot)
xy<-expand.grid(x,x)
fxy.v<-(1/2*pi)*exp(-0.5*(xy[,1]^2+xy[,2]^2))

fxy.m<-matrix(fxy.v,nrow=length(x))
rotate.persp(x,x,fxy.m)
```

Mostrar el árbol:

```
xy.rpart<-rpart(z~xy[,1]+xy[,2])
plot(xy.rpart)
text(xy.rpart,use.n=TRUE,xpd=2,cex=0.6)
```

Visualizar la función de predicción:

```
xy.pred<-predict(xy.rpart,newdata=data.frame(xy))
xy.pred.m<-matrix(xy.pred,nrow=length(x))
rotate.persp(x,y,xy.pred.m)
```

Clasificación con `rpart`

```
srp<-rpart(yc~xs+ys,data=synth.tr)
```

Mostrar el árbol:

```
plot(srp)
text(srp,use.n=TRUE,xpd=2)
```

Matriz de confusión:

```
ycp <- predict(srp,synth.te, type = "class")
table(True = synth.te$yc, Pred = ycp)
```

Realizad la clasificación de los datos SAheart, spam y zip, calculando en cada caso las matrices de confusión.

La siguiente clasificación, con los datos Titanic, reproduce el comportamiento del algoritmo ID3. Además, se usa la función prp, para gráficos de árboles, del paquete rpart.plot. Sólo se aprecia una de las muchísimas opciones de presentación del gráfico. Ved la documentación del paquete para conocer estas opciones.

```
require(rpart)
require(rpart.plot)
Titanic1<-read.table("Titanic.data.txt",header=TRUE)
Titanic.CART<-rpart(Survived~.,data=Titanic1,minsplit=10,
    parms=list(split="information"))
prp(Titanic.CART,type=4,extra=1,col=1:4,tweak=1.2,
    box.col="Moccasin")
```

Detalles sobre rpart

CON LOS DATOS SYNTH

```
require(rpart)
source("prob.err.r")

synth.rpart.1<-rpart(yc~xs+ys,data=synth.tr)
```

Mostrar el árbol:

```
plot(synth.rpart.1)

text(synth.rpart.1,use.n=TRUE,xpd=2)
```

Matriz de confusión:

```
synth.rpart.1.pred<-
predict(synth.rpart.1,synth.te,type="class")
synth.rpart.1.conf<-table(True = synth.te$yc,
                          Pred=synth.rpart.1.pred)
synth.rpart.1.conf
```

Estimación de la probabilidad de error:

```
prob.err(synth.rpart.1.conf)
```

El árbol que se obtiene se puede modular modificando los parámetros de control de la función `rpart`. Esto se consigue mediante la función `rpart.control()`. Mirad el help para ver los posibles parámetros a modificar y los valores por defecto. Por ejemplo,

```
control.parms<-rpart.control(minsplit = 10,cp=0.005)
synth.rpart.2<-
rpart(yc~xs+ys,data=synth.tr,control=control.parms)
plot(synth.rpart.2)
text(synth.rpart.2,use.n=TRUE,xpd=2,cex=0.7)
```

Matriz de confusión:

```
synth.rpart.2.pred<-
predict(synth.rpart.2,synth.te,type="class")
synth.rpart.2.conf<-table(True = synth.te$yc,
                          Pred=synth.rpart.2.pred)
synth.rpart.2.conf
```

Estimación de la probabilidad de error:

```
prob.err(synth.rpart.2.conf)
```

Los parámetros con mayor repercusión son `minsplit` y `cp`. Probad con varios valores de estos y de los otros parámetros, viendo el efecto sobre la probabilidad de error.

CON LOS DATOS STAGEC

En `stagec.r` hay el principio del análisis `rpart`. Haced como en el caso de los datos `synth`, variando los parámetros del árbol.

Luego, con las funciones `printcp()` y `plotcp()`, ver como varía el error de validación cruzada en función del parámetro `cp`. A continuación, con `prune()`, recortar el árbol para diversos valores de `cp`.