ANÁLISIS DE DATOS

Ricardo Aler Mur

Ricardo Aler Mur

TUTORIAL PRIMERA PRÁCTICA: CONSTRUCCIÓN SIMPLIFICADA DE ÁRBOLES DE MODELOS

Se recuerda que un árbol de modelos, como el que construye M5' (que hemos visto en clase de teoría y que está en el paquete RWeka), puede tener nodos internos con atributos tanto continuos como discretos. Podéis consultar la construcción de árboles M5' en las transparencias "árboles para predicción numérica" en Aula Global. En la primera práctica, nosotros construiremos árboles de modelos de una manera simplificada, de manera que **los nodos internos del árbol sólo tengan atributos discretos**. A continuación sigue un tutorial donde se detallan los pasos que hay que seguir para construir nuestros árboles de modelos simplificados de manera manual. Lo que tendréis que hacer en la primera práctica es poner esos pasos en forma de función R. El tutorial os ayudará a entender lo que hay que hacer.

1. DATOS DE PARTIDA

Aplicaremos las operaciones sobre los datos de ejemplo de las algas que ya hemos venido manejando y cuyos dataframes se pueden encontrar aquí: http://www.dcc.fc.up.pt/~ltorgo/DataMiningWithR/. Recordar usar setwd('…') para poner el directorio de trabajo. Recordad que en Windows se usa doble barra \\ para separar los directorios. Recordar que para preparar los datos tuvimos que poner en R:

```
load('algae.RData')
load('testAlgae.RData')
load('algaeSols.RData')
algae=algae[complete.cases(algae),1:12]
algaeTest=cbind(test.algae,algae.sols)
algaeTest=algaeTest[complete.cases(algaeTest),1:12]
```

Lo anterior quita los datos con missing values (*NAs*) y se queda sólo con las 12 primeras columnas: 11 variables de entrada y 1 de salida (la concentración del alga *a1*).

2. Descomposición de las columnas de los datos en entradas discretas, entradas continuas y salidas

Vamos a generar un árbol de modelos simplificado, de un único nivel (un único nodo) de atributos discretos y modelos de regresión lineal en los nodos hoja. Lo mejor será separar desde el principio las columnas con variables discretas, las continuas y la salida, de esta manera:

```
entradas = algae[,-ncol(algae)]
esNumero = sapply(entradas[1,],is.numeric)
entradasDiscretas = entradas[,!esNumero]
entradasContinuas = entradas[,esNumero]
salida = algae[,ncol(algae)]
```

La primera instrucción quita la salida *a1*. La segunda determina que columnas son numéricas y cuales discretas. La función *is.numeric()* determina si un valor es numérico o no. La función *sapply()* aplica la función que va como segundo argumento a todos los elementos del vector que va como primer argumento. Los resultados se pueden ver aquí:

```
> head(esNumero)
season   size  speed   mxPH   mnO2     Cl
 FALSE  FALSE  FALSE   TRUE   TRUE   TRUE

> head(entradasDiscretas)
  season  size  speed
1 winter small medium
2 spring small medium
3 autumn small medium
4 spring small medium
5 autumn small medium
6 winter small   high

> head(entradasContinuas)
  mxPH mnO2     Cl     NO3     NH4    oPO4     PO4 Chla
1 8.00  9.8 60.800  6.238 578.000 105.000 170.000 50.0
2 8.35  8.0 57.750  1.288 370.000 428.750 558.750  1.3
3 8.10 11.4 40.020  5.330 346.667 125.667 187.057 15.6
4 8.07  4.8 77.364  2.302  98.182  61.182 138.700  1.4
5 8.06  9.0 55.350 10.416 233.700  58.222  97.580 10.5
6 8.25 13.1 65.750  9.248 430.000  18.250  56.667 28.4

> head(salida)
[1]  0.0  1.4  3.3  3.1  9.2 15.1
```

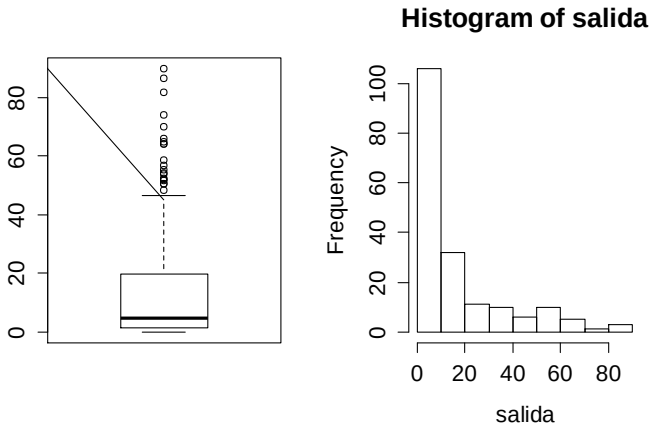
3. ¿QUÉ ATRIBUTO PONGO EN LA RAÍZ?

Partimos del árbol vacío. Tenemos que determinar qué atributo poner en la ráiz. Hay tres posibilidades: *season*, *size*, y *speed*. Siguiendo la idea del algoritmo M5', pondremos en la raíz aquel atributo que disminuya la desviación típica lo máximo posible (es decir que concentre los datos lo máximo posible). Por curiosidad, la desviación típica de la variable de salida *a1* en los datos originales es:

```
> sd(salida)
[1] 20.31698
```

Puede ser interesante hacer un histograma y un boxplot de esta variable para ver su distribución. Se puede ver que inicialmente los valores de *a1* están concentrados en valores bajos 0-20, aunque hay unos cuantos outliers por encima de 50, según el boxplot:

```
> par(mfrow=c(1,2))
> boxplot(salida)
> hist(salida)
```



Vamos a probar cada uno de los tres atributos discretos para ver hasta que punto disminuyen la desviación típica inicial (standard deviation). El objetivo es seleccionar aquel que tenga la menor desviación típica.

El atributo *season* tiene cuatro valores: *autumn, spring, summer, winter*. Eso quiere decir que si ponemos *season* en la raíz, me dividirá el conjunto de datos en cuatro particiones:

```
> head(with(entradasDiscretas, entradasDiscretas[season=="autumn",]))
   season  size  speed
3  autumn small medium
5  autumn small medium
8  autumn small   high
16 autumn small   high
23 autumn small   high
31 autumn small   high
> head(with(entradasDiscretas, entradasDiscretas[season=="spring",]))
   season  size  speed
2  spring small medium
4  spring small medium
11 spring small   high
18 spring small   high
20 spring small medium
22 spring small   high
> head(with(entradasDiscretas, entradasDiscretas[season=="summer",]))
   season  size speed
7  summer small  high
12 summer small  high
14 summer small  high
19 summer small  high
26 summer small  high
33 summer small  high
> head(with(entradasDiscretas, entradasDiscretas[season=="winter",]))
   season  size  speed
1  winter small medium
6  winter small   high
9  winter small medium
10 winter small   high
13 winter small   high
15 winter small   high
```

Nótese el uso de *with* para evitar tener que poner *entradasDiscretas$season* cada vez. Puede ser interesante calcular cuántos datos hay para cada valor del atributo con *nrow()*, el cual nos devuelve el número de líneas (datos). *ncol()* nos hubiera dado el número de columnas:

```
> nrow(with(entradasDiscretas, entradasDiscretas[season=="autumn",]))
[1] 36
> nrow(with(entradasDiscretas, entradasDiscretas[season=="spring",]))
[1] 48
> nrow(with(entradasDiscretas, entradasDiscretas[season=="summer",]))
[1] 43
> nrow(with(entradasDiscretas, entradasDiscretas[season=="winter",]))
[1] 57
```

Lo anterior se puede condensar con la función *table*, la cual nos devuelve justamente el número de datos para cada valor del atributo *season*. Esto nos será útil después.

```
> table(entradasDiscretas$season)

autumn spring summer winter
    36     48     43     57
```

Pero nos estamos apartando del objetivo. Lo que queremos saber es la desviación típica que resultará si ponemos el atributo *season* en la raíz del árbol de modelos (y de *size* y de *speed*, también). Como hemos visto, *season* tiene cuatro valores y generará las cuatro particiones de datos, con 36, 48, 43 y 57 datos respectivamente, como ya hemos visto. Podríamos calcular la desviación típica de cada una de esas particiones y después calcular la media de las cuatro desviaciones resultantes, para calcular la desviación media del atributo *season*. Podemos hacerlo así:

```
> sd(salida[entradasDiscretas$season=="autumn"])
[1] 22.13206
> sd(salida[entradasDiscretas$season=="spring"])
[1] 18.13439
> sd(salida[entradasDiscretas$season=="summer"])
[1] 17.48134
> sd(salida[entradasDiscretas$season=="winter"])
[1] 22.98939
```

Pero es más cómodo usar la función *tapply(vector, índice, funcion)*. Esta función descompone los valores del vector en términos de los valores de la variable índice y a cada subconjunto de valores la función. Es más fácil entenderlo con la siguiente instrucción. Comparad con las desviaciones típicas calculadas anteriormente.

```
> tapply(salida, entradasDiscretas$season, sd)
  autumn   spring   summer   winter
22.13206 18.13439 17.48134 22.98939
```

Ahora podríamos calcular la desviación típica media de *season*, calculando la media de los cuatro valores anteriores. Pero cuidado, no podemos calcular la media normal, sino ponderada, porque el número de datos para *autumn*, *spring*, *summer* y *winter* no es el mismo. Recordemos que el número de datos para cada valor era justamente este:

```
> table(entradasDiscretas$season)

autumn spring summer winter
    36     48     43     57
```

Por tanto, la desviación típica media de *season* debería ser calculada así (siendo el número total de datos 36+48+43+57=184): (36*22.13 + 48*18.13+43*17.48+57*22.99)/184 = 20.27. Pero resulta que la función weighted.mean() ya hace esto mismo:

```
sds = tapply(salida, entradasDiscretas$season, sd)
pesos = table(entradasDiscretas$season)
weighted.mean(sds,pesos)
[1] 20.26792
```

Podemos hacer lo mismo para los otros dos atributos:

```
sds = tapply(salida, entradasDiscretas$size, sd)
pesos = table(entradasDiscretas$size)
weighted.mean(sds,pesos)
[1] 19.21751

sds = tapply(salida, entradasDiscretas$speed, sd)
pesos = table(entradasDiscretas$speed)
weighted.mean(sds,pesos)
[1] 19.05286
```

Está claro que el mejor atributo (mínima desviación típica media) es *speed*, con desviación 19.05. Podemos automatizar el proceso anterior, para que se calcule la desviación típica media de todos los atributos discretos, independientemente de cuantos haya, y sin usar bucles. Usaremos para ello la función *apply(vector, filasColumnas, función)*, la cual aplica la función a cada columna del vector si filasColumnas vale 2 (si vale 1, se hace para cada fila). Lo haremos así:

Primero, definimos una función que aplica *tapply* sobre una columna cualquiera *x*:

```
desviacionesAtributo = function(x){tapply(salida,x,sd)}
```

Después, aplicamos esa función sobre cada una de las columnas de entradasDiscretas:

```
> desviacionesAtributos = apply(entradasDiscretas,2,
desviacionesAtributo)
> desviacionesAtributos
$season
  autumn   spring   summer   winter
22.13206 18.13439 17.48134 22.98939

$size
   large   medium    small
16.86448 16.94859 24.08440

$speed
    high      low   medium
22.11319 12.63193 18.61731
```

Vemos que se ha generado una lista con tres campos ($season, $size, $speed). Cada campo contiene un vector con las desviaciones típicas para cada valor de cada atributo. Recordemos que la desviación típica inicial de *a1* era:

```
> sd(salida)
[1] 20.31698
```

Podemos ver que *speed* consigue disminuir mucho la desviación típica para el valor *low*, e incluso con *medium* es menor que la original, aunque es a costa de incrementar algo la de *high*.

Para poder calcular las medias ponderadas de las desviaciones típicas, necesitamos saber también cuántos datos hay para cada valor de cada atributo. Podemos hacerlo igualmente con *apply*, pero ahora usaremos la función *table* en lugar de *sd*.

```
> vecesValoresAtributos = apply(entradasDiscretas,2, table)
> vecesValoresAtributos
$season

autumn spring summer winter
    36     48     43     57

$size

 large medium  small
    42     83     59

$speed

  high    low medium
    76     31     77
```

Ahora tenemos que calcular la desviación típica media ponderada. Para ello, usaremos la función *mapply*:

```
mediasPorAtributo =
mapply(weighted.mean,desviacionesAtributos,vecesValoresAtributos)
mediasPorAtributo
  season     size    speed
20.26792 19.21751 19.05286
```

Hemos llegado a la misma situación que antes, pero automatizando todo el proceso. Nuevamente, la variable *speed* es la que menor desviación típica tiene, y por tanto, es el atributo que debería ser puesto en la raíz del árbol. De hecho, dado que la desviación típica original era:

```
> sd(salida)
[1] 20.31698
```

El poner el atributo *speed* consigue reducir ligeramente la desviación típica (de 20.3 a 19.0).

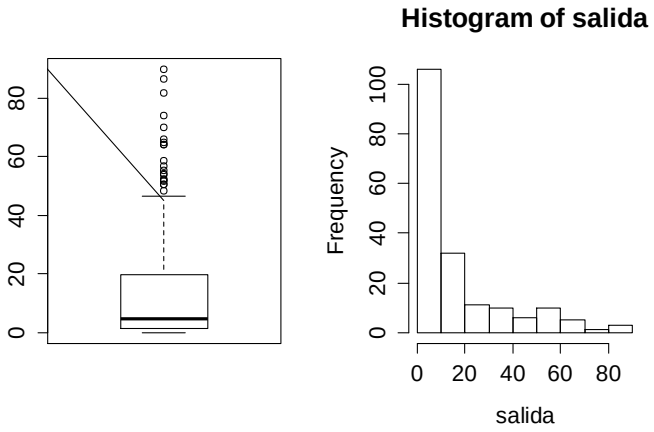
Es interesante recordar que el proceso de selección del mejor atributo lo hemos reducido a los siguientes pasos:

```
desviacionesAtributo = function(x){tapply(salida,x,sd)}

desviacionesAtributos = apply(entradasDiscretas,2,
desviacionesAtributo)
```

```
vecesValoresAtributos = apply(entradasDiscretas,2, table)

mediasPorAtributo =
mapply(weighted.mean,desviacionesAtributos,vecesValoresAtributos)

seleccionado = which.min(mediasPorAtributo)

seleccionado

speed
    3
```

En la última secuencia hemos utilizado la función *which.min()* que dice cuál es el número de atributo que alcanza el valor mínimo de la desviación típica media (el tercero, o sea, *speed*).

Puede ser interesante ver como se distribuían los datos antes de poner el atributo speed, y una vez que lo ponemos en la raíz del árbol y creamos tres ramas (tantas como valores de *speed*). Recordemos que incialmente, los valores de la variable de salida *a1* se distribuían así:



Después de poner el atributo *speed*, se crean tres subconjuntos de datos:

```
salidaConSpeedLow = salida[entradasDiscretas[,seleccionado]=="low"]
salidaConSpeedMedium =
salida[entradasDiscretas[,seleccionado]=="medium"]
salidaConSpeedHigh = salida[entradasDiscretas[,seleccionado]=="high"]
```

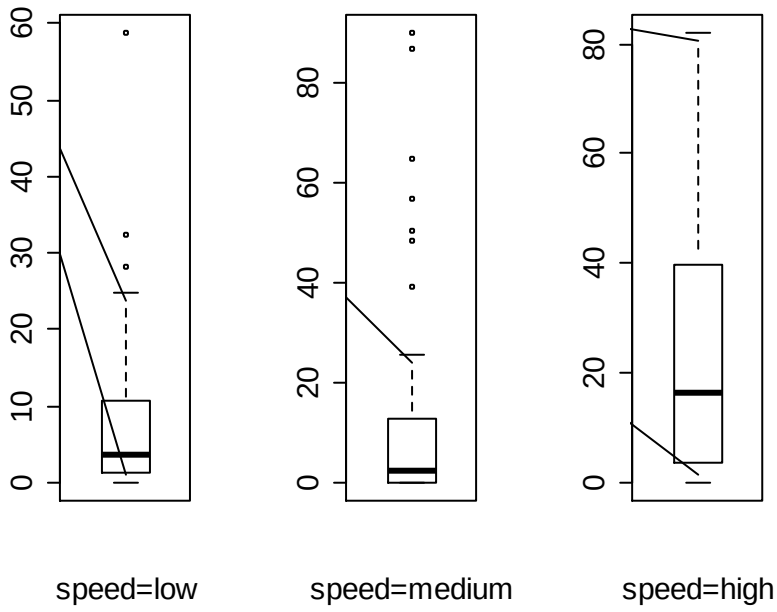
```
> salidaConSpeedLow
 [1]  5.7  3.6  1.2 58.7  8.7 17.0 12.3  8.8  1.9  1.2  9.2 28.1  2.5
4.4  6.5 24.8
[17] 32.5  0.0  0.0  0.0 12.3  7.2  3.4  0.0  1.4 12.5  0.0  1.1  0.0
0.0  1.7
> salidaConSpeedMedium
 [1]  0.0  1.4  3.3  3.1  9.2 25.4  0.0  0.0 13.6  5.3 18.3  2.0  2.2
0.0  0.0 89.8
[17] 24.8  0.0 39.1  0.0  1.9 25.5 11.3  4.4  1.9  1.6 64.9 15.1 14.4
3.3  0.0  2.8
[33]  0.0  0.0  0.0  0.0  0.0 16.5  7.0  3.7 18.1 86.6  1.4  1.6  3.3
0.0  0.0  1.5
[49]  0.0  2.4  7.8 10.3  1.5  2.1  1.4  3.4  2.8  2.5  1.7  1.4  0.0
0.0  0.0 48.3
[65] 50.4 56.8 17.3  0.0  7.6  2.9  2.2  3.8 18.9 12.7 18.0  0.0  2.4
> salidaConSpeedHigh
 [1] 15.1  2.4 18.2 17.0 16.6 32.1 43.5 31.1 52.2 69.9 46.2 31.8 50.6
15.5 23.2 74.2
```

```
[17] 13.0  4.1 29.7 17.1 33.9  3.4  6.9  0.0 66.0  0.0  4.1 51.8 29.5
54.4 81.9 54.0
[33] 20.3 15.8 55.5 10.3 64.2  2.2  6.7 10.8  1.2 12.6 14.7  1.2  0.0
0.0  4.0  5.9
[49] 23.7  0.0  3.6 64.3 46.6 24.0 43.7 14.6  1.7  3.3  4.2  4.1  1.2
19.0 42.5 39.7
[65] 32.8 12.2  1.9  0.0  0.0  2.2  0.0  1.1 39.7 37.3 52.4 18.1
```

Vamos a plotear estos tres subconjuntos de datos, desglosados por el valor del atributo *speed*.

```
> par(mfrow=c(1,3))
> boxplot(salidaConSpeedLow,xlab="speed=low")
> boxplot(salidaConSpeedMedium,xlab="speed=medium")
> boxplot(salidaConSpeedHigh,xlab="speed=high")
```

En comparación con el boxplot original, podemos ver que el valor *médium* y sobre todo el valor *low* está más concentrado que el original, aunque es a costa de dispersar los datos con valor *high*



Quedaría por construir modelos lineales con los datos de cada rama, cosa que podemos hacer fácilmente con:

```
> entradaSalida=cbind(entradasContinuas,salida)
> entradaSalidaLow=entradaSalida[entradasDiscretas[,3]=="low",]
> entradaSalidaMedium=entradaSalida[entradasDiscretas[,3]=="medium",]
> entradaSalidaHigh=entradaSalida[entradasDiscretas[,3]=="high",]
> lmlow = lm(salida~.,data=entradaSalidaLow)
> lmmedium = lm(salida~.,data=entradaSalidaMedium)
> lmhigh = lm(salida~.,data=entradaSalidaHigh)
> lmlow

Call:
lm(formula = salida ~ ., data = entradaSalidaLow)

Coefficients:
```

```
(Intercept)          mxPH          mnO2            Cl           NO3
NH4          oPO4
  204.00431    -21.27049     -1.27166     -0.06165     -2.61536
0.00172     -0.01404
         PO4          Chla
    0.01801      0.04214

> lmmedium

Call:
lm(formula = salida ~ ., data = entradaSalidaMedium)

Coefficients:
(Intercept)          mxPH          mnO2            Cl           NO3
NH4          oPO4
  40.770202    -3.429763     2.103906    -0.045775    -3.243339    -
0.000243    -0.020582
         PO4          Chla
  -0.006883    -0.169126

> lmhigh

Call:
lm(formula = salida ~ ., data = entradaSalidaHigh)

Coefficients:
(Intercept)          mxPH          mnO2            Cl           NO3
NH4          oPO4
 -13.007634     4.455454     1.609640     0.015303    -1.638664
0.001856     0.044978
         PO4          Chla
  -0.154126    -0.080797
```

En resumen, nuestro árbol de modelos simplificado queda:

Si speed==low, entonces usar regresión lineal lmlow

Si speed==médium, entonces usar regresión lineal lmmedium

Si speed==high, entonces usar regresión lineal lmhigh

Cuidado, es un modelo muy simplificado porque sólo permite un único atributo discreto en la raíz del árbol. M5' devuelve mejores árboles de modelos, pero su programación es mucho mas dificultosa.