

UNIDAD 2. Métodos de Conjunto: Adaboost, Random Forest.

Adaboost

AdaBoost (Adaptive Boosting, refuerzo adaptativo, Freund y Schapire, 1996) es un meta-algoritmo de aprendizaje automático que construye de forma iterativa un conjunto de reglas simples de clasificación (clasificadores débiles), que después se utilizan de forma conjunta para elaborar una regla compleja (clasificador fuerte), de forma similar a la utilización de un conjunto de expertos que decide por mayoría –para resolver un problema determinado- en lugar de emplear un único experto.

Un clasificador aleatorio decide al azar la clase a la que pertenece cada elemento. Se denomina clasificador débil a aquel que produce resultados solo ligeramente mejores que un clasificador aleatorio. En cualquier conjunto de datos tendremos en general muchas variables relacionadas con el criterio de clasificación o variable de interés, y la mayoría de ellas pueden ser utilizadas como clasificador débil.

El término boosting (fortalecimiento, refuerzo) se refiere a la idea de mejorar un clasificador débil, para intentar conseguir un grado de acierto mucho más alto, un clasificador fuerte, generalmente utilizando un cierto número de clasificadores débiles o bien una modificación y mejora progresiva de uno solo.

Si buscamos una clasificación con k clases, y decidimos la clase de cada elemento de forma totalmente aleatoria acertaremos en promedio $1/k$ (50% si son dos clases, 20% si son cinco, ...), y el error de clasificación será $1-1/k$: a los clasificadores débiles se les pide simplemente que tengan un error menor que $1-1/k$, y del resto se encarga AdaBoost. Por ejemplo si queremos “adivinar” si un trabajador está en paro o no, un clasificador débil puede ser la edad, considerando que la proporción de parados es mayor entre los que tienen menos de 30 años: en lugar de decidir al azar lanzando una moneda diremos que es parado si es menor de 30 años y ocupado en otro caso. De ese modo mejoraremos ligeramente la tasa de éxito. Otros clasificadores débiles se pueden construir con variables adicionales como sexo, sector de actividad, profesión, o nivel de estudios.

El “refuerzo adaptativo” consiste en una estrategia que pondera de forma distinta los casos bien y mal clasificados en cada iteración, dando un peso mayor a los mal clasificados, para modificar de forma progresiva la regla de decisión.

Es un meta-algoritmo, porque los clasificadores débiles pueden ser generados por métodos diversos, como por ejemplo una simple recta que divide el plano en dos clases, o bien algún método más complejo como regresión logística o un árbol de clasificación.

Se considera que Adaboost es robusto al sobreajuste, problema común a la mayoría de los métodos de clasificación supervisada que consiste en que el método consigue clasificar con un alto grado de acierto la muestra disponible, es decir los casos utilizados en el entrenamiento, pero funciona mucho peor con casos nuevos, no utilizados para la construcción de la regla de clasificación.

Basicamente el algoritmo es el siguiente:

1) Se asignan pesos o ponderaciones iniciales iguales $1/n$ a todos los casos de la muestra de entrenamiento.

2) Se aplica el clasificador débil a los datos ponderados, y se asigna al clasificador un peso en función de la tasa de error "e" obtenida: $\text{peso} = 0.5 \ln[(1-e)/e]$.

Este paso se repite T veces modificando los pesos de los casos, aumentando los correspondientes a los casos mal asignados, con el fin de centrar los esfuerzos en clasificar bien los casos erróneos. En cada iteración se recalculan las ponderaciones de los casos, se aplica el clasificador, y se calcula el peso o ponderación del clasificador obtenido.

3) Finalmente se construye el clasificador mejorado como una combinación lineal de los T clasificadores encontrados, aplicando los pesos calculados en las T iteraciones.

Utilizaremos para la aplicación de este método la función boosting, del paquete de R adabag; esta función realiza el refuerzo adaptativo empleando como clasificadores débiles árboles de clasificación, que son entrenados con los datos de la muestra. Su sintaxis básica es la siguiente:

`boosting(formula, data)`

La fórmula debe expresar la variable de clasificación en función de los predictores; el argumento data, opcional, permite indicar el conjunto de datos al que se refieren los nombres de variables indicados en la fórmula.

Utilizaremos el conjunto de datos "deudas", en el archivo con formato de R "deudas.RData". Se trata de una muestra de 100 clientes de un banco, algunos de los cuales han presentado impagos, de los que se dispone de información relativa a su edad, nivel educativo, antigüedad en el empleo, antigüedad en su lugar de residencia actual, ingresos, relación entre deudas e ingresos, importe de las deudas por tarjeta de crédito, e importe de otras deudas. El banco está interesado en predecir los clientes que tendrán impagos basándose en esa información disponible.

```
setwd("C:/CURSO DM")    # configura la carpeta o directorio de trabajo
load("deudas.RData")     # carga el conjunto de datos
names(deudas)            # listado de variables
summary(deudas)           # resumen descriptivo
```

```
> names(deudas)
```

```
Edad   Formacion Empleo  Residencia  Ingreso  Deud_ing  Deud_tarj  Deud_otr  Impago
```

```
> summary(deudas)
```

Edad		Formacion		Empleo		Residencia	
entre 30 y 40:	36	bachillerato	:28	entre 5 y 10 años:	24	entre 5 y 10:	21
mas de 40	:37	elemental	:56	más de 10 años	:41	más de 10	:39
menos de 30	:27	estudios universit.:	16	menos de 5 años	:35	menos de 5	:40

Ingreso		Deud_ing		Deud_tarj		Deud_otr		Impago	
Min.	: 15.00	Min.	: 0.600	Min.	: 0.0300	Min.	: 0.090	No:	68
1st Qu.:	27.00	1st Qu.:	3.550	1st Qu.:	0.3675	1st Qu.:	0.940	Si:	32
Median	: 38.50	Median	: 6.400	Median	: 0.8700	Median	: 1.780		
Mean	: 46.27	Mean	: 8.915	Mean	: 1.2622	Mean	: 2.448		
3rd Qu.:	57.00	3rd Qu.:	12.725	3rd Qu.:	1.5225	3rd Qu.:	3.027		
Max.	:176.00	Max.	:35.300	Max.	:11.3600	Max.	:16.670		

Empezaremos instalando (menú de R Gui Paquetes → instalar paquetes) el paquete adabag, que será después cargado con la orden library.

```
library(adabag) # carga el paquete adabag que contiene la función boosting
```

Dividimos los datos en dos submuestras. Una de ellas, de tamaño 80, será utilizada para construir el clasificador, y la otra, con los 20 casos restantes, la utilizaremos para validar los resultados: se predice la clase de cada elemento de la muestra de validación con las funciones generadas sin utilizar los datos de ese elemento.

```
set.seed(123) # inicializa el arranque del generador aleatorio
muestra <- sample(1:nrow(deudas), 80) # muestra de 80 números entre 1 y 100
entrenamiento <- deudas[muestra, ] # muestra de entrenamiento
prueba <- deudas[-muestra, ] # muestra de validación
```

Hemos elegido una muestra o lista de 80 números al azar entre 1 y 100 (número de filas, nrow, del conjunto deudas). Si marcamos con el cursor “muestra” y ejecutamos podremos ver la lista de números obtenida. La muestra de entrenamiento –conjunto de datos “entrenamiento”- está formada por los 80 casos o filas del conjunto deudas correspondientes a esa lista, y el conjunto “prueba” por los 20 restantes.

Ahora utilizamos la función boosting, generando el objeto de resultados al que llamaremos “modelo”, con los dos argumentos descritos anteriormente, la fórmula que relaciona la variable dependiente con las variables predictoras, y el conjunto de datos utilizado “entrenamiento”. El símbolo ~ que indica relación lineal se puede obtener en el teclado con AltGr + 4 seguido de un espacio. El método Adaboost es lento; debemos tener paciencia si el programa tarda uno o varios minutos –dependiendo de los datos y del ordenador- en completar todos los cálculos, y esperar hasta que aparezca en la ventana de salida la orden ejecutada.

```
modelo <- boosting(Impago ~ Edad + Formacion + Empleo + Residencia + Ingreso +
Deud_ing + Deud_tarj + Deud_otr, data = entrenamiento)
```

Cuando las variables explicativas son todas, como en este ejemplo, existe una forma de abreviar la fórmula, sustituyendo el conjunto de predictores por un punto, que representa a todas las variables excepto la dependiente:

```
modelo <- boosting(Impago ~ ., data = entrenamiento)
```

El objeto creado “modelo” contiene todos los resultados. El algoritmo genera 100 reglas o clasificadores débiles con forma de árbol de decisión, que se pueden listar con:

`modelo$trees`

La última de ellas, número 100, se muestra a continuación a título de ejemplo:

```
1) root 80 39 Si (0.48750000 0.51250000)
  2) Ingreso>=53 31 2 No (0.93548387 0.06451613) *
  3) Ingreso< 53 49 10 Si (0.20408163 0.79591837)
    6) Deud_tarj< 0.35 7 0 No (1.00000000 0.00000000) *
    7) Deud_tarj>=0.35 42 3 Si (0.07142857 0.92857143) *
```

El árbol tiene 5 nodos. El nodo raíz contiene los 80 casos del conjunto de entrenamiento; la función de pérdida (loss) para este nodo vale 39 (número de casos mal clasificados ponderados), con un 51,25% de acierto. El nodo 2 separa a los 31 clientes que tienen ingresos mayores o iguales que 53, entre los cuales solo hay dos impagos, por lo que predice no impago (con probabilidad 0,935). El árbol predice impago (Si), con probabilidad 0,7959, si los ingresos del cliente son inferiores a 53, y especialmente (con probabilidad 0,92857) si además las deudas por tarjeta superan el valor 0,35.

La asignación final se realiza con las 100 reglas o clasificadores conjuntamente, ponderando cada una de ellas con el peso asignado al clasificador. Podemos ver las ponderaciones de los 100 clasificadores débiles con el elemento `modelo$weights`:

```
> modelo$weights
[1] 1.4722195 1.2296642 0.8392965 1.4622925 0.6764778 1.1545028
[7] 0.8100867 0.8108564 0.4353066 1.2916263 0.7714386 0.7964523
.....
```

La clase asignada a cada caso (impago si/no) se puede ver con el elemento `modelo$class`:

```
> modelo$class
[1] "Si" "No" "No" "Si" "No" "No" "Si" "No" "No" "No" "No" "No" "No" "Si"
[15] "No" "No" "Si" "Si" "No" "No" "No" "No" "Si" "Si" "Si" "No" "No" "No"
.....
```

La probabilidad de pertenencia a cada clase para cada caso del conjunto de datos – permite valorar la fiabilidad de la asignación- se puede ver con el elemento `modelo$prob`:

```
> modelo$prob
      [,1]      [,2]
[1,] 0.18838053 0.811619470
[2,] 0.87362158 0.126378418
[3,] 0.99129608 0.008703920
[4,] 0.07362167 0.926378333
[5,] 0.95096754 0.049032458
.....
```

La columna [,1] corresponde a “Impago NO” y la siguiente a “Impago SI”.

La importancia de cada variable en el proceso de asignación se puede ver con el elemento `modelo$importance`:

```
> modelo$importance
Deud_ing Deud_otr Deud_tarj Edad Empleo Formacion Ingreso Residencia
35.4815663 6.9497833 10.6841734 1.6271481 0.9942510 7.4120243 36.3958274 0.4552261
```

El estadístico de importancia está expresado en porcentaje (la suma es 100). Las variables Ingreso (36,39%) y Deud_ing (35,48%) son las que contribuyen en mayor medida a la clasificación.

Podemos utilizar el modelo para predecir la clase de cualquier elemento o caso con la función `predict`. Por ejemplo, aplicada a todos los elementos del conjunto “entrenamiento”:

```
resultados.entrenamiento <- predict(modelo, newdata = entrenamiento, type = "class")
```

Los resultados de la predicción, en el objeto `resultados.entrenamiento`, incluyen los votos ponderados “votes” de los predictores obtenidos en todas las iteraciones del algoritmo, la probabilidad “prob” de que cada caso pertenezca a una u otra clase, la clase “class” asignada en consecuencia (la más probable), y especialmente la matriz de “confusion” que cruza la clase real con la predicción en una tabla de doble entrada:

```
resultados.entrenamiento$confusion
```

	Observed Class	
Predicted Class	No	Si
No	57	0
Si	0	23

El 100% de los casos utilizados ha sido identificado correctamente.

Sin embargo nos queda la duda de si el método servirá para clasificar también correctamente elementos nuevos, no utilizados en el análisis. Para esto lo aplicamos a la muestra de prueba:

```
resultados.prueba <- predict(modelo, newdata = prueba, type = "class")
t <- resultados.prueba$confusion
t ; 100 * sum(diag(t)) / sum(t) # calcula el porcentaje global de acierto
```

Le hemos llamado `t` a la matriz de confusión, para calcular a continuación el porcentaje de éxito, mediante la suma de los casos que están en la diagonal `diag(t)` dividida por la suma de todos los casos o elementos de la matriz `sum(t)`.

	Observed Class	
Predicted Class	No	Si
No	11	1
Si	0	8

```
[1] 95
```

El porcentaje de acierto es ahora del 95%; solamente uno de los 20 casos ha sido clasificado erróneamente, y por lo tanto se confirma la capacidad del método para identificar con precisión el impago.

La asignación de cada cliente del banco a Impago SI/NO se realiza a la clase más probable. En la muestra ya sabemos si cada cliente ha tenido o no impagos; sin embargo el banco puede aplicar el modelo construido a los casos desconocidos (clientes nuevos, de los que no se sabe si han tenido impagos o no); para ello solo es necesario añadir los casos nuevos al conjunto (con la variable Impago = NA) y observar la probabilidad de impago con la variable prob del objeto de resultado, resultados.prueba\$prob:

```
> resultados.prueba$prob
      [,1]      [,2]
[1,] 0.09421557 0.905784431
[2,] 0.99037138 0.009628624
[3,] 0.88963508 0.110364918
[4,] 0.72835935 0.271640653
[5,] 0.11239068 0.887609315
[6,] 0.88714299 0.112857010
.....
```

La probabilidad de impago es la que aparece en la última columna.

Como ahora sabemos que no hay problema de sobreajuste podemos mejorar el modelo utilizando los 100 casos, toda la información disponible, es decir el conjunto completo deudas:

```
modelo <- boosting(Impago ~ ., data = deudas)
resultados <- predict(modelo, newdata = deudas, type = "class")
t <- resultados$confusion
t ; 100 * sum(diag(t)) / sum(t)
```

```

              Observed Class
Predicted Class No Si
No 68 0
Si 0 32
[1] 100
```

El resultado es del 100% de acierto.

Random Forest

Random Forest o Bosque Aleatorio (Leo Breiman y Adele Cutler, 2001) consiste en una combinación de árboles de predicción. Se trata de una técnica de clasificación supervisada no paramétrica que busca segmentar el espacio de los predictores en regiones simples, definidas por intervalos de valores de los predictores o variables explicativas, dentro de las cuales la predicción es mucho más eficaz que en el espacio completo.

Un árbol de clasificación es fácil de interpretar, admite la representación gráfica, permite utilizar predictores cuantitativos –que deben ser categorizados- o cualitativos (o ambos combinados simultáneamente), y no requiere hipótesis restrictivas.

El árbol, representación gráfica de las reglas que permiten establecer la clasificación, está formado por nodos –internos y externos- y arcos que los unen. Los nodos externos –definido cada uno de ellos por una combinación concreta de valores de las variables explicativas- son las hojas del árbol, que agrupan elementos de un solo tipo (preferiblemente), o bien de varios tipos estableciendo una distribución de probabilidad sobre ellos, buscando siempre la mayor homogeneidad posible dentro de cada hoja, en la que debe haber en general un tipo claramente predominante.

Se utiliza el método de partición recursiva, que comienza buscando todos los valores distintos de cada predictor o variable explicativa, para encontrar la variable y el valor de corte que dividan los datos en dos partes de modo que se minimice la suma de cuadrados de los residuos, es decir la partición que permite explicar una mayor proporción de la variabilidad total de los datos. A continuación se hace lo mismo con cada uno de los dos grupos, buscando dentro de ellos la variable y punto de corte que permiten dividirlos a su vez en dos partes, y así sucesivamente hasta que ya no es posible la partición porque las clases obtenidas tienen un número inferior a un valor umbral prefijado.

Random Forest ajusta un número elevado de árboles, en lugar de uno solo, y toma como resultado la clase más frecuente, la decisión de la mayoría del bosque, en el conjunto de predicciones de todos los árboles individuales.

La función que utilizaremos con R para este método es `randomForest`, del paquete del mismo nombre, que es necesario instalar (la primera vez) y cargar en la sesión de R en la que va a ser utilizada. La sintaxis básica es:

```
randomForest(formula, data)
```

La fórmula expresa la variable de clasificación en función de los predictores, y el argumento opcional `data` sirve para indicar a que conjunto de datos pertenecen las variables que aparecen en ella. Se puede sustituir la fórmula por un conjunto de datos, en cuyo caso funciona como método de clasificación no supervisada, buscando en los datos una clasificación natural (como *k-medias* o EM). El número de árboles que forman el bosque es 500 por defecto, número considerado adecuado para la mayoría de las aplicaciones, aunque puede ser cambiado con el argumento opcional `ntree`.

En cada división del árbol elige las variables más adecuadas –buscando el mejor resultado- y su punto de corte, utilizando para ello un número de variables candidatas igual a la raíz cuadrada de m, siendo m el número de predictores.

Utilizaremos para el ejemplo los datos del archivo vinos.RData, con una muestra de 54 vinos monovarietales en los que se ha determinado la concentración de 6 ácidos orgánicos, con dos clases definidas por la variedad de uva utilizada en su elaboración.

```
setwd("C:/CURSO DM")
```

```
load("vinos.RData") # lee el conjunto "vinos"  
library(randomForest) # carga el paquete randomForest
```

Dividimos el conjunto de datos en dos partes: una muestra de entrenamiento, elegida al azar, con la que se construirá el modelo, y los casos restantes que utilizaremos como muestra de validación o prueba:

```
set.seed(12345)  
muestra <- sample(1:nrow(vinos), 40) # 40 números de fila o caso elegidos al azar  
entrenamiento <- vinos[muestra, ]  
prueba <- vinos[-muestra, ]
```

A continuación construiremos el modelo con la función randomForest:

```
modelo <- randomForest(var~ gal+tar+mal+shi+cit+suc, data=entrenamiento)  
modelo
```

```
randomForest(formula = var ~ gal + tar + mal + shi + cit + suc,  
              data = entrenamiento)  
  
Type of random forest: classification  
Number of trees: 500  
No. of variables tried at each split: 2  
  
OOB estimate of error rate: 0%  
Confusion matrix:  
  A  G class.error  
A 27  0          0  
G  0 13          0
```

Como se puede observar en la matriz de confusión –con los 40 casos de entrenamiento- predice correctamente el 100% de los casos.

Podemos ver la clase (variedad de uva) asignada a cada caso de la muestra de entrenamiento con el elemento de resultado modelo\$predicted:

```
> modelo$predicted  
39 47 40 46 23  9 16 24 34 45  2  7 31  1 48 19 15 38 43 51  
 A  G  G  G  A  A  A  A  G  G  A  G  A  G  A  A  A  A  A  A  
52 11 42 22 20 12 30 37  6 13 28 41  5 27  8 36 35 18 10  3  
 A  A  A  A  A  A  A  A  G  A  A  G  G  A  G  A  G  A  A  G  
Levels: A G
```


La decisión de asignación se toma por mayoría, considerando la asignación –el voto- de los 500 árboles. La proporción de “votos” obtenida por las dos clases en cada uno de los casos del conjunto de datos, que es un indicador de la fiabilidad de la asignación, se puede ver con el elemento de resultado `modelo$votes`:

```
> round(modelo$votes, 3)
```

	A	G
39	0.977	0.023
47	0.279	0.721
40	0.011	0.989
46	0.159	0.841
23	0.994	0.006

El método calcula una medida de la importancia de cada variable explicativa en el proceso de clasificación, basada en el índice de Gini, que mide el grado de pureza o de homogeneidad de los nodos del árbol: un nodo es puro cuando solo tiene elementos de la misma clase; el nodo es más puro cuanto más pequeño es el valor del índice. La medida de importancia consiste en la reducción del índice de Gini debida a esa variable, por lo que las variables más importantes son las que consiguen reducción mayor. Se puede ver con el elemento de resultado `modelo$importance`:

```
> modelo$importance
      MeanDecreaseGini
gal          2.9373551
tar          0.9418765
mal          1.9600440
shi          9.4045883
cit          0.7578310
suc          1.1044052
```

Las variables más importantes en la identificación de la variedad de uva son los ácidos shiquímico (9,4) y galacturónico (2,9).

Veamos ahora la identificación de la variedad de uva para los casos que hemos mantenido al margen del modelo, con el fin de ser utilizados como validación, en el conjunto “prueba”:

```
predicciones <- predict(modelo, prueba)
t <- with(prueba, table(predicciones, var)) # Matriz de confusión
t ; 100 * sum(diag(t)) / sum(t)
```

	var	
predicciones	A	G
A	8	1
G	0	5

[1] 92.85714

Solamente uno de los 14 casos de la muestra de validación es clasificado erróneamente, con un porcentaje de éxito elevado, el 92,857%, y por lo tanto podemos esperar que el

modelo clasifique de forma razonablemente correcta las muestras nuevas, sin problema de sobreajuste. Esto nos permite mejorarlo utilizando toda la muestra para el entrenamiento:

```
# con toda la muestra
modelo <- randomForest(var~ ., data=vinos)
predicciones <- predict(modelo, vinos) # con toda la muestra
t <- with(vinos, table(predicciones, var)) # Matriz de confusión
t ; 100 * sum(diag(t)) / sum(t)
```

```
var
predicciones  A  G
              A 35  0
              G  0 19
[1] 100%
```

El porcentaje de éxito ahora es del 100%, todos los vinos son identificados correctamente.

Si queremos identificar la variedad de uva –desconocida- de casos nuevos solo tenemos que añadir esos casos al conjunto de datos, con `var = NA`, y aplicar la función `predict` como anteriormente: el objeto `predicciones` indicará la clase asignada a todos los casos, incluidos los nuevos.