

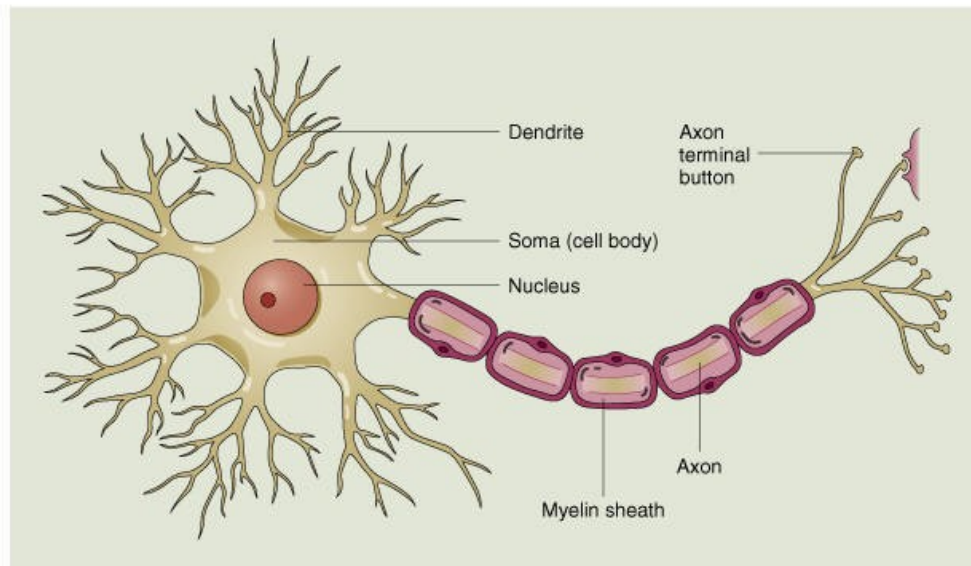
Redes Neuronales

Qué son las neuronas	3
Representación en Red	4
Red Neuronal Artificial. Planteamiento	4
Redes Neuronales para Regresión. Construcción del modelo	5
Número de parámetros.....	7
Objetivos	7
El Soporte Práctico y el Soporte Teórico	7
Teorema de aproximación: versión simplificada	8
Ejemplo básico con R, paquete nnet.....	9
1) Pre-procesado de las variables input para la construcción de una red neuronal.....	11
2) Estimación de los parámetros de la red.....	12
3) Utilización de la red creada para predecir nuevas observaciones.....	16
4) Comparación básica con regresión	17
5) Comparación con regresión vía partición training-test con el paquete caret	17
Ejemplo básico con variables input categóricas.....	21
Arquitectura de la Red	26
Número de capas ocultas.....	26
Número de nodos.....	26
Recopilación de Recomendaciones para fijar el número de nodos, dados los datos	26
Reglas simples a tener en cuenta para decidir el número de nodos y para comprender su efecto en las predicciones.....	27
Ejemplo: variando el número de nodos en el archivo compress	28
Conceptos básicos sobre optimización	30
Estrategias básicas para la Comparación de Modelos.....	33
Algunas medidas clásicas para modelos de regresión	33
Técnicas de remuestreo	34
Training, Validation, Test.....	34
Training, Test.....	34
Validación Cruzada (k grupos).....	35
El paquete caret de R.....	38
Las cuatro técnicas de remuestreo básicas con caret.....	38
Esquema general de trabajo con caret	39
Utilizando avNNet	40
El intercambio Sesgo-Varianza (The Bias-Variance Trade-off)	42

Explorando sesgo-varianza a través de validación cruzada repetida y boxplot	46
Selección de variables en redes	50
Taxonomía de métodos de selección de variables	50
Filters (ranking por importancia) en redes.....	51
Algunas ideas para la preselección de variables	53
Selección básica stepwise AIC y BIC con MASS	53
Selección repetida con submuestras	54
Checklist básico y preparación anterior a la selección de variables	57
Ejemplo Autmpg revisitado.....	57
¿Qué hacer desde el punto de vista práctico en este ejemplo?	65
Ejercicios.....	66
Redes Neuronales para Clasificación binaria	67
El problema de la clasificación supervisada	67
Funcionamiento general de los métodos de clasificación binaria	67
Matriz de Confusión de un método de clasificación y medidas asociadas	68
CURVA ROC (Receiver Operating Characteristic Curve).....	69
Planteamiento de la red neuronal con variable dependiente binaria	70
Ejemplo básico saheart	70
Algunos detalles en la utilización de R para modelización predictiva de variables binarias ..	74
Tuneado de la red, con criterio Accuracy (tasa de aciertos).....	74
Selección de variables en clasificación binaria bajo stepAIC en regresión logística	75
Validación cruzada repetida.....	76
Ejercicios (clasificación binaria).....	81
Bibliografía básica	82

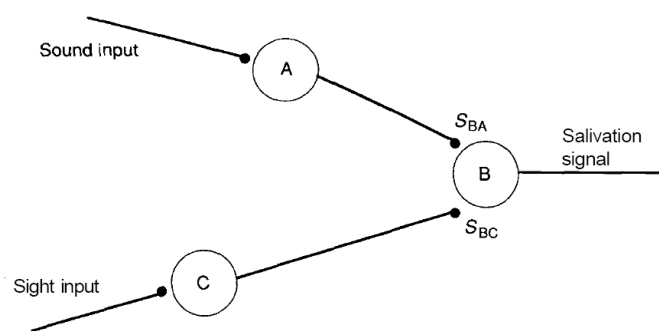
Qué son las neuronas

Las neuronas son células del sistema nervioso cuya función principal es recibir, procesar y transmitir información a través de señales químicas y eléctricas. Las neuronas se comunican con rapidez y precisión a larga distancia con otras células, mediante impulsos eléctricos. En un cierto modo puede considerarse que transmiten información. Los impulsos eléctricos se propagan a través de los axones y de las ramificaciones nerviosas llamadas dendritas.



© 2000 John Wiley & Sons, Inc.

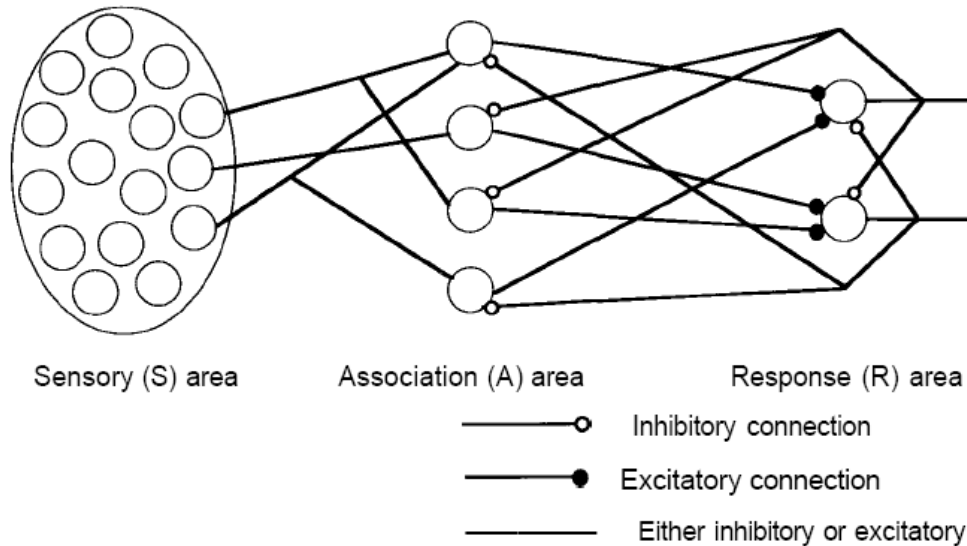
Las neuronas se distinguen por su función. Pueden ser sensoriales (reciben información del exterior), pueden ser también interneuronas, que procesan la información recibida en red, percibiendo, recordando y tomando decisiones, o pueden ser motoras, generando contracción de musculatura u otras consecuencias. En el gráfico inferior se ven dos neuronas sensoriales A y C conectadas a una motora B (o podría interpretarse la B como una interneurona que conectaría a otra motora D no incluida en el gráfico que produciría la salivación).



Two neurons, A and C, are stimulated by the sensory inputs of sound and sight, respectively. The third neuron, B, causes salivation. The two synaptic junctions are labeled S_{BA} and S_{BC} .

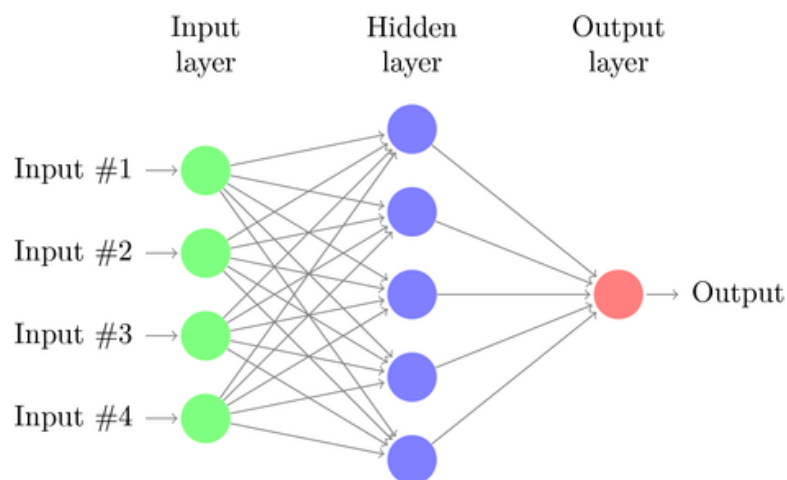
Representación en Red

Se necesitan muchas neuronas para recoger la información externa, procesarla internamente y para obtener reacciones físicas finales. Las neuronas se agrupan en redes interconectadas para procesar y transmitir correctamente la información percibida.



Red Neuronal Artificial. Planteamiento

Imitando la estructura de las redes neuronales naturales, se construye una relación entre las variables input (llamadas nodos input), y las variables de salida (nodos output), introduciendo variables artificiales (nodos en la capa oculta). N.T: Hidden=oculta, layer=capa.



Matemáticamente, una Red Neuronal es en realidad un modelo de la forma

$$y=f(x_1,x_2,x_3,...)$$

donde la función f es por lo general no lineal. En la fórmula final no aparecen los nodos intermedios de la capa oculta, dejando patente que son solamente un artificio.

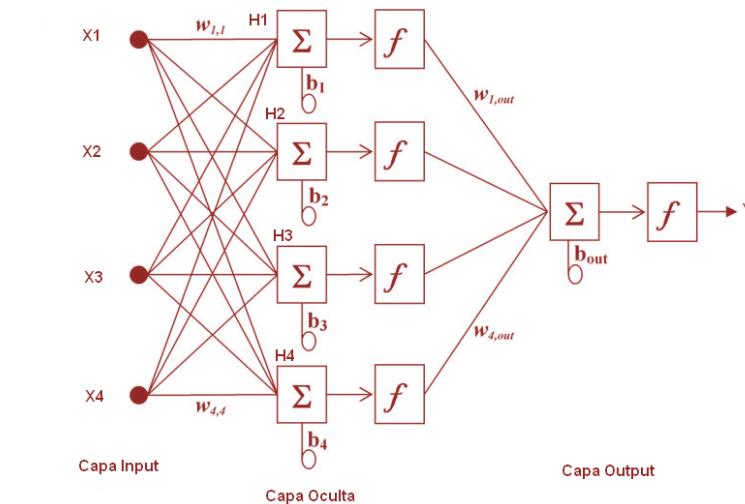
Ejemplo de red:

$$y = 29.8 + 85.2 * \tanh(-0.9 + 2.1 * X1 - 0.15 * X2) - 79 * \tanh(-3.5 + 5 * X1 + 0.01 * X2)$$

Redes Neuronales para Regresión. Construcción del modelo

Vamos a desarrollar el modelo de red neuronal de modo constructivo.

La **capa input** se conecta a la **capa oculta** mediante la **función de combinación**, representada por Σ , donde los pesos w_{ij} hacen el papel de parámetros a estimar.



Red neuronal con 4 inputs $X1, \dots, X4$, un output Y , con una capa oculta con 4 nodos ocultos $H1, H2, H3, H4$.

La función de combinación más habitual es la lineal. Previamente se han estandarizado las variables input.

Así, el valor de los nodos ocultos se obtiene a partir de la función de combinación:

$$H_1 = w_{11} x_1 + w_{21} x_2 + w_{31} x_3 + w_{41} x_4 + b_1$$

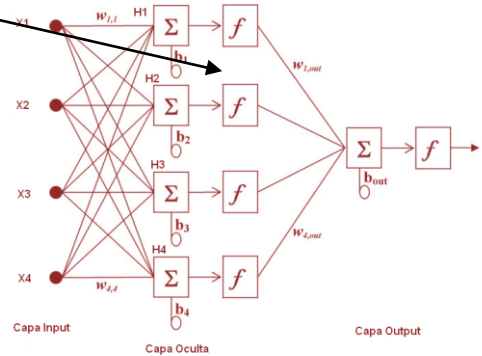
$$H_2 = w_{12} x_1 + w_{22} x_2 + w_{32} x_3 + w_{42} x_4 + b_2$$

$$H_3 = w_{13} x_1 + w_{23} x_2 + w_{33} x_3 + w_{43} x_4 + b_3$$

$$H_4 = w_{14} x_1 + w_{24} x_2 + w_{34} x_3 + w_{44} x_4 + b_4$$

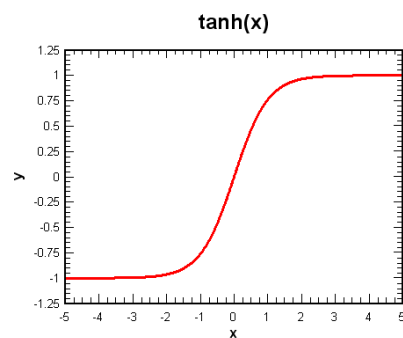
El parámetro constante b_j se denomina, en jerga neuronal, bias (sesgo)

Tras aplicar la función de combinación, aplicamos a cada nodo oculto la **función de activación**, representada por **f**.



Una función de activación muy utilizada es la tangente hiperbólica,

$$\tanh(g) = 1 - \frac{2}{1 + \exp(2g)}$$



Aplicando la función de activación a cada nodo oculto:

$$H_1 = \tanh(w_{11}x_1 + w_{21}x_2 + w_{31}x_3 + w_{41}x_4 + b_1)$$

$$H_2 = \tanh(w_{12}x_1 + w_{22}x_2 + w_{32}x_3 + w_{42}x_4 + b_2)$$

$$H_3 = \tanh(w_{13}x_1 + w_{23}x_2 + w_{33}x_3 + w_{43}x_4 + b_3)$$

$$H_4 = \tanh(w_{14}x_1 + w_{24}x_2 + w_{34}x_3 + w_{44}x_4 + b_4)$$

Finalmente aplicamos combinación Σ para dar lugar a la capa output. En el gráfico aparece otra función de activación **f**, que solo se utiliza cuando la variable output es categórica (dummy). Cuando y es continua, que es el caso que nos ocupa, no se aplica activación en la última capa.

Al aplicar combinación de los cuatro nodos ocultos quedaría:

$$y = w_{1out}\tanh(w_{11}x_1 + w_{21}x_2 + w_{31}x_3 + w_{41}x_4 + b_1) + w_{2out}\tanh(w_{12}x_1 + w_{22}x_2 + w_{32}x_3 + w_{42}x_4 + b_2) + w_{3out}\tanh(w_{13}x_1 + w_{23}x_2 + w_{33}x_3 + w_{43}x_4 + b_3) + w_{4out}\tanh(w_{14}x_1 + w_{24}x_2 + w_{34}x_3 + w_{44}x_4 + b_4) + b_{out}$$

Por lo tanto esta última fórmula sería el modelo de red neuronal para predecir y, con una red con cuatro variables input, 4 nodos en la capa oculta y función de activación tanh.

Número de parámetros

4*4 pesos capa input-capa oculta+4 bias capa oculta+4 pesos capa oculta-capa output+bias capa output=25

En general,

Número de parámetros en una Red neuronal con una capa y una variable output:

$$h(k+1)+h+1$$

donde h= número de nodos ocultos, k=número de nodos input.

Nota 1: en regresión clásica serían k+1=5 en nuestro caso.

Nota 2: la red admite varias variables output.

Objetivos

Finalizada la construcción y planteamiento, el objetivo computacional concreto es **estimar**, a partir de los datos disponibles, todos los parámetros **w** y **b** del modelo

$$y = w_{1out} \tanh(w_{11} x_1 + w_{21} x_2 + w_{31} x_3 + w_{41} x_4 + b_1) + w_{2out} \tanh(w_{12} x_1 + w_{22} x_2 + w_{32} x_3 + w_{42} x_4 + b_2) + w_{3out} \tanh(w_{13} x_1 + w_{23} x_2 + w_{33} x_3 + w_{43} x_4 + b_3) + w_{4out} \tanh(w_{14} x_1 + w_{24} x_2 + w_{34} x_3 + w_{44} x_4 + b_4) + b_{out}$$

del mismo modo que en regresión lineal habría que estimar los parámetros de una ecuación mucho más sencilla, $y = b_0 + b_1 x_1 + b_2 x_2 + b_3 x_3 + b_4 x_4$

Una vez estimados los pesos, se obtendrían las predicciones para una nueva observación con valores en x_1, x_2, x_3, x_4 (se denotará por \hat{y} la predicción obtenida con el modelo).

$$\hat{y} = \tanh(0.21(\tanh(0.15x_1 - 0.53x_2 + 2.1x_3 + 3.5x_4 + 6))) + \dots + \dots + \dots$$

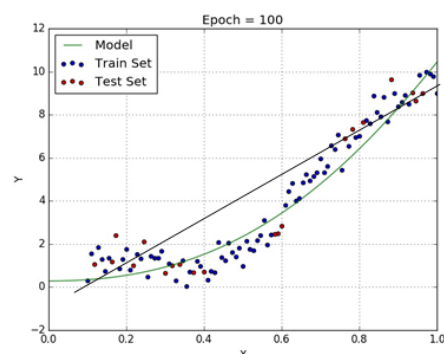
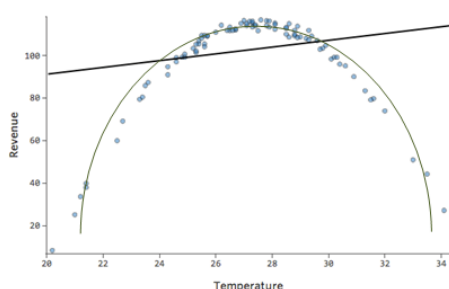
La estimación de parámetros se llama, en jerga neuronal, **“entrenar”** la red.

Los métodos utilizados son técnicas de optimización numérica, que van variando los valores de los parámetros de manera iterativa, hasta cumplir el objetivo de optimización (función de error en datos training o en datos de validación, coste, etc.).

El Soporte Práctico y el Soporte Teórico

En primer lugar surge la duda de por qué deberíamos utilizar una red neuronal que es complicada y tiene 25 parámetros en lugar de una regresión que tiene 5.

La principal razón es que las relaciones entre la variable output y , y las variables input, puede ser no lineal, y es más apropiado un modelo con no linealidad implícita (la red) que un modelo puramente lineal, que se ajustará peor a nuestros datos.



La segunda razón práctica es la flexibilidad de la red: como se verá brevemente en el soporte teórico, aunque desconozcamos cual es la naturaleza de la relación entre y y \mathbf{x} , la red es suficientemente flexible para ajustarse a esa relación. Esto es importante cuando las dimensiones del problema (número de variables input) hacen impracticable cualquier aproximación artesanal a esas relaciones.

En segundo lugar es necesario un soporte teórico para que esa función construida cumpla el objetivo de ajustarse bien a los datos.

La justificación de la red neuronal como modelo está basada en **Teoremas de aproximación universal**, en varias versiones (Cybenko-Funahashi-Hornik), que enuncia que cualquier función continua puede aproximarse al nivel requerido con una red neuronal con al menos una capa oculta y un número de nodos a determinar.

Teorema de aproximación: versión simplificada

Sea $\phi(\cdot)$ función no constante, acotada, monótona creciente y continua. Dada cualquier función $f(\mathbf{x})$ en el hipercubo $[0,1]$ y $\epsilon > 0$, existe N y constantes α_i, b_i, w_i en \mathbf{R}^m , tales que:

$$F(\mathbf{x}) = \sum_{i=1}^N \alpha_i \phi(w_i^T \mathbf{x} + b_i)$$

$$|F(\mathbf{x}) - f(\mathbf{x})| < \epsilon$$

Es decir, si existe relación entre las variables input y la variable output, representada en la fórmula por $f(\mathbf{x})$ y esta relación es no lineal, desconocida, el teorema de aproximación universal nos dice que esa función, aunque la desconozcamos, la podremos aproximar por la función $F(\mathbf{x})$ construida con la red neuronal:

$$F(\mathbf{x}) = \sum_{i=1}^N \alpha_i \phi(w_i^T \mathbf{x} + b_i)$$

La parte $w_i^T \mathbf{x} + b_i$ refiere a la combinación de nodos input, la función ϕ es la tangente hiperbólica en nuestro ejemplo, \tanh , y los α_i son parámetros de combinación de la capa intermedia a la capa output.

Esta forma simplificada dice que para aproximarse a la relación entre input y output, existen N (número de nodos en la capa oculta) y las constantes enunciadas (valores de todos los parámetros \mathbf{w} y \mathbf{b} de nuestro ejemplo), tales que la función de red $F(\mathbf{x})$ aproxima suficientemente bien a la relación real $f(\mathbf{x})$.

Por lo tanto, “solamente” hay que decidir el **número de nodos ocultos N** , la **función de activación ϕ** (en el ejemplo, $N=4$, $\phi=\tanh$) y finalmente el valor de los parámetros \mathbf{w} y \mathbf{b} para que nuestra red aproxime lo mejor posible a la relación entre variables input y output.

El número de nodos ocultos lo decidiremos con métodos empíricos y otras consideraciones, y el valor de los parámetros con métodos de optimización, siempre utilizando los datos disponibles como referencia para construir el modelo y siendo conscientes de que estamos en el campo de la modelización estadística. En este campo los modelos perfectos no existen y las relaciones matemáticas expresadas en los teoremas son solo una representación ideal, y al

ponerlas en práctica deberán tenerse en cuenta conceptos estadísticos como la varianza no explicada (aquello que las variables input no puede explicar sobre la output), sobreajuste, etc.

Como un ejemplo de este choque de paradigmas (matemático-estadístico) se puede pensar en unos datos en los que la relación entre las variables input y output no es muy clara. ¿Se podría hablar de una relación ideal $y=f(x)$ subyacente? En caso de que fuera así efectivamente la red $F(x)$ la podría aproximar suficientemente bien, pero que $f(x)$ exista o no puede estar en entredicho; es más, a esta $f(x)$ se le podría agregar, como se hace en regresión, un término de error estadístico incontrolable, $f(x)+\epsilon$.

Ejemplo básico con R, paquete nnet

Archivo de datos [compress.Rda](#)

Archivo de código [compressbis.R](#)

Se trata de predecir la fuerza de compresión del cemento.

Dos nodos input continuos: **age** y **water**

Un nodo Output continuo: **cstrength** (compressive strength)

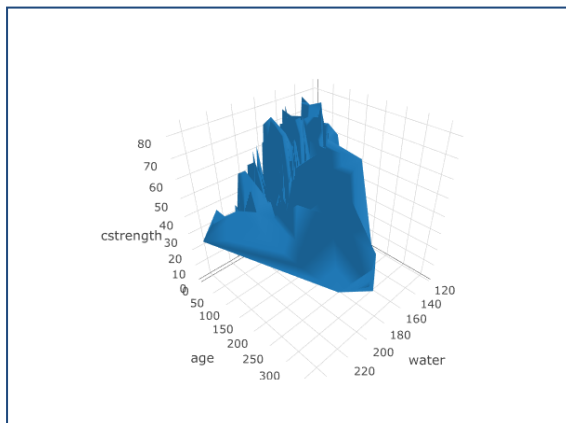
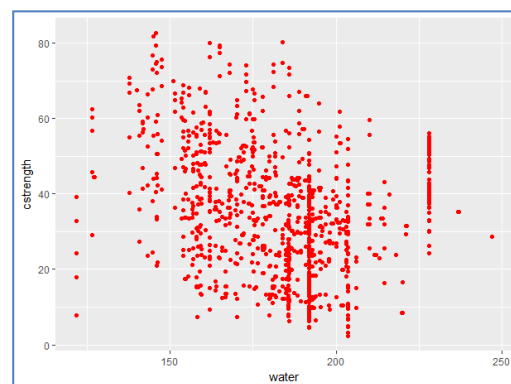
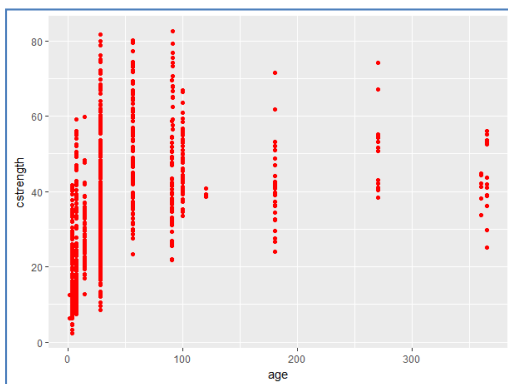
```
load("compress.Rda")
```

cstrength	cement	blast	ash	water	plasti	agggreg	fineagg	age
79.99	540.00	0.00	0.00	162.00	2.5	1040.00	676.00	28.00
61.89	540.00	0.00	0.00	162.00	2.5	1055.00	676.00	28.00
40.27	332.5	142.5	0.00	228.00	0.00	932.00	594.00	270.00
41.05	332.5	142.5	0.00	228.00	0.00	932.00	594.00	365.00
44.3	198.6	132.4	0.00	192.00	0.00	978.4	825.5	360.00
47.03	266.00	114.00	0.00	228.00	0.00	932.00	670.00	90.00
43.7	380.00	95.00	0.00	228.00	0.00	932.00	594.00	365.00
36.45	380.00	95.00	0.00	228.00	0.00	932.00	594.00	28.00
45.85	266.00	114.00	0.00	228.00	0.00	932.00	670.00	28.00
39.29	475.00	0.00	0.00	228.00	0.00	932.00	594.00	28.00
38.07	198.6	132.4	0.00	192.00	0.00	978.4	825.5	90.00
28.02	198.6	132.4	0.00	192.00	0.00	978.4	825.5	28.00
43.01	427.5	47.5	0.00	228.00	0.00	932.00	594.00	270.00
42.33	190.00	190.00	0.00	228.00	0.00	932.00	670.00	90.00
...

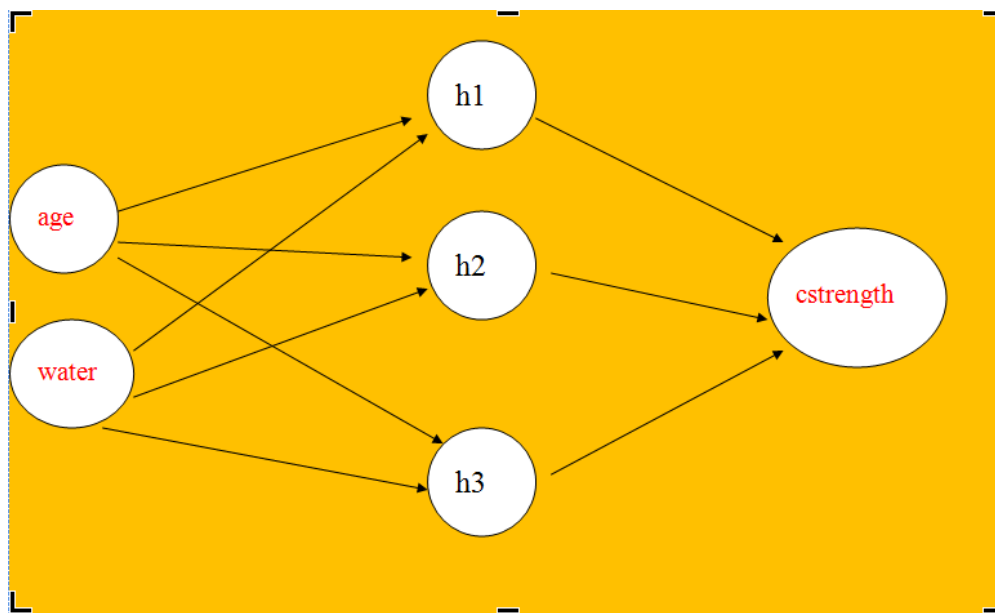
```
# Algunos gráficos básicos
```

```
library(ggplot2)
library(plotly)
ggplot(compress, aes(age, cstrength))+geom_point(color="red")
ggplot(compress, aes(water, cstrength))+geom_point(color="red")
hist(compress$age)
hist(compress$water)
plot_ly(compress, x = ~water, y = ~age, z = ~cstrength, type
='mesh3d')
```

Aparente No linealidad: buena situación para utilizar las redes



Planteamiento de una red neuronal básica con 3 nodos en la capa oculta



En este ejemplo seguiremos el siguiente esquema:

- 1) Pre-procesado de las variables input para la construcción de una red neuronal
- 2) Estimación de los parámetros de la red con el paquete nnet
- 3) Utilización de la red creada para predecir nuevas observaciones
- 4) Comparación básica con regresión
- 5) Comparación con regresión vía partición training-test con el paquete caret

1) Pre-procesado de las variables input para la construcción de una red neuronal

Esto es necesario en general, pero sobre todo en paquetes de R o Python.

1) Si hay valores missing en las variables input, deben eliminarse esas observaciones o imputar los valores missing.

2) Si hay variables input categóricas, en general deben pasarse a dummy.

3) Las variables input continuas deben estandarizarse. La razón es porque los algoritmos de optimización funcionan así mejor pues están menos expuestos a overflow y valores extremos de los parámetros.

Hay dos modos de estandarización:

(a) normalización: $(x - \text{media}) / d.\text{típica}$

La variable resultante puede tomar valores negativos, su rango es habitualmente entre -3 y 3 para variables distribuidas normalmente, aunque en variables muy asimétricas puede ser más alto en valor absoluto.

(b) Escala (0,1): $(x - \text{min}) / (\text{max} - \text{min})$

La variable resultante toma valores entre 0 y 1.

La más utilizada es la (a) , aunque muchos recomiendan la (b).

Nota: los paquetes comerciales como SAS, SPSS, etc. realizan automáticamente estas tareas de preprocesado simplificando la construcción de modelos al usuario. Desgraciadamente, en software libre tipo R y Python tendremos que realizarlas semi-manualmente.

```
# Es bueno crear listas de variables continuas, categóricas
# y la dependiente en un vector

listconti<-c("age", "water")
vardep<-c("cstrength")
cstrength<-compress[,vardep]
# ESTANDARIZACIÓN DE TODAS LAS VARIABLES CONTINUAS

means <-apply(compress[,listconti],2,mean,na.rm=TRUE)
sds<-sapply(compress[,listconti],sd,na.rm=TRUE)

# Con esta línea se sustituyen toda las variables continuas por su
# versión estandarizada
compress<-scale(compress[,listconti], center = means, scale = sds)
# Ahora se añade la columna de la variable output, que no se
# estandariza habitualmente
compress<-data.frame(cbind(compress,cstrength))
```

cstrength	age	water
79.99	-0.28	-0.92
61.89	-0.28	-0.92
40.27	3.55	2.17
41.05	5.06	2.17
44.3	4.98	0.49
...

2) Estimación de los parámetros de la red

```
# CREACIÓN DEL MODELO DE RED CON nnet
library(nnet)
# Controlar la semilla de aleatorización es importante
# pues interviene en el proceso de optimización
set.seed(22342)

# En la red se pone la fórmula del modelo, linout=TRUE para indicar
# que la variable dependiente es continua, size=3 para
# indicar 3 nodos en la capa oculta,
# y maxit=100 para indicar solo 100 iteraciones
# del proceso de estimación-optimización

red1<-nnet(data=compress,cstrength~age+water, linout=TRUE,
size=3,maxit=100)

summary(red1)
```

Información de nnet:

- a) Listado de los estimadores de los pesos (en la consola).
- b) Proceso de optimización-estimación de los pesos (en la consola).
- c) Creación del objeto nnet, con información completa del modelo y utilizable para aplicarlo para predecir nuevas observaciones

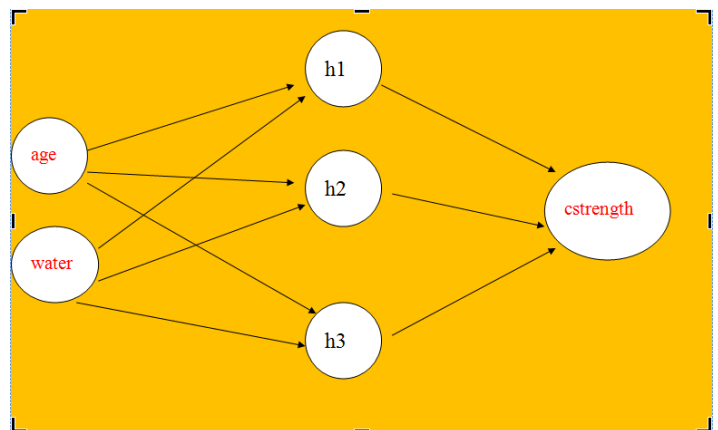
a) Listado de los estimadores de los pesos (en la consola)

i1=nodo input 1 (age)

i2=nodo input 2 (water)

o=nodo output (cstrength)

```
a 2-3-1 network with 13 weights
options were - linear output units
b->h1 i1->h1 i2->h1
 3.05  3.31  0.13
b->h2 i1->h2 i2->h2
-0.96  0.13 -1.89
b->h3 i1->h3 i2->h3
-73.01  0.02 36.16
b->o  h1->o  h2->o  h3->o
-51.42 88.52 28.69  8.51
```



Por ejemplo, para ver los parámetros relacionados con el nodo h1:

Todo lo que va al nodo 1:

```
b->h1 i1->h1 i2->h1
 3.05  3.31  0.13
```

Esto significa:

$$h1 = w11 * \text{age} + w21 * \text{water} + b1$$

$$h1 = 3.31 * \text{age} + 0.13 * \text{water} + 3.05$$

luego activación:

$$h1 = \tanh(h1)$$

El resto de nodos igual

b) Proceso de optimización-estimación de los pesos (en la consola).

Se observa como va descendiendo el valor de la función objetivo a medida que los pesos se estiman mejor según avanzan las iteraciones.

```
# weights: 13
initial value 1601216.121533
iter 10 value 166917.729534
iter 20 value 147054.970173
iter 30 value 144565.081887
iter 40 value 143638.206741
iter 50 value 143070.915174
iter 60 value 141797.145971
iter 70 value 141156.059581
iter 80 value 140535.605141
iter 90 value 140427.683543
iter 100 value 140017.488970
final value 140017.488970
stopped after 100 iterations
```

Cuando el output es una variable continua, la función de error habitual a minimizar es el Mean Squared Error

$$MSE = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

(aunque el paquete nnet presenta el valor de SSE=n*MSE)

Recordemos:

y_i son los valores **reales** de la variable output para cada observación

\hat{y}_i son los valores **predichos** de la variable output para cada observación.

Por ejemplo, con los parámetros finales,

$$\hat{y}_i = 88.52 * \tanh(h1) + 28.69 * \tanh(h2) + 8.51 * \tanh(h3) - 51.42$$

donde

$$h1 = 3.31 * \text{age} + 0.13 * \text{water} + 3.05$$

$$h2 = 0.13 * \text{age} + \dots, \text{etc.}$$

La i latina se refiere a cada observación, con sus valores de age y water.

En el ejemplo, cstrength es y , predi es \hat{y}_i , con los estimadores finales del modelo

En el proceso de optimización, en cada iteración los pesos son diferentes, la \hat{y}_i es diferente y da lugar a diferente valor de la función de error

$$SSE = n * MSE = \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

```
# weights: 13
initial value 1601216.121533
iter 10 value 166917.729534
iter 20 value 147054.970173
iter 30 value 144565.081887
iter 40 value 143638.206741
iter 50 value 143070.915174
iter 60 value 141797.145971
iter 70 value 141156.059581
iter 80 value 140535.605141
iter 90 value 140427.683543
iter 100 value 140017.488970
final value 140017.488970
stopped after 100 iterations
```

Le habíamos pedido que se detuviera en 100 iteraciones (maxit=100 en la función nnet) y se para el proceso en la última estimación, con un SSE de 140017, que dividiendo por n=1030 da MSE=135.93 y un RMSE de 11.65.

c) Creación del objeto nnet, con información completa del modelo y utilizable para aplicarlo para predecir nuevas observaciones

```
red1 | List of 18
n : num [1:3] 2 3 1
nunits : int 7
nconn : num [1:8] 0 0 0 0 3 6 9 13
conn : num [1:13] 0 1 2 0 1 2 0 1 2 0 ...
nsunits : num 6
decay : num 0
entropy : logi FALSE
softmax : logi FALSE
censored : logi FALSE
value : num 140017
wts : num [1:13] 3.054 3.308 0.126 -0.96 0.135 ...
convergence : int 1
fitted.values: num [1:1030, 1] 46.1 46.1 45.9 45.9 43.7 ...
.. attr(*, "dimnames")=List of 2
.. ..$ : chr [1:1030] "1" "2" "3" "4" ...
.. ..$ : NULL
```

3) Utilización de la red creada para predecir nuevas observaciones

En `newdata=` se pone el archivo que se quiere predecir, que debe tener los valores de las variables input para cada observación. Por simplificar en este ejemplo usamos el dataset original, pero habitualmente sería un dataset externo con solo las variables input.

```
# Se puede obtener la estimación de un vector de observaciones
# con la función predict, habitual en los modelos de R
predi<-predict(red1,newdata=compress,type="raw")
```

```
# Unimos la predicción al archivo original para verlo
compress2<-data.frame(cbind(compress,predi))
```

```
# Esto es solo para reordenar las columnas
compress2<- compress2[, c(3,4,1,2)]
```

cstrength	predi	age	water
79.99	46.0911156044459	-0.28	-0.92
61.89	46.0911156044459	-0.28	-0.92
40.27	45.8607028447145	3.55	2.17
41.05	45.9261218136113	5.06	2.17
44.3	43.6552276427606	4.98	0.49
47.03	45.4548267010264	0.7	2.17
43.7	45.9261218136113	5.06	2.17
36.45	38.3876941784839	-0.28	2.17
45.85	38.3876941784839	-0.28	2.17
39.29	38.3876941784839	-0.28	2.17
....

Se observa que las predicciones no son buenas. Como se irá viendo la precisión de las predicciones se irá refinando con una buena selección de variables y afinado ("tuneado") de los parámetros del modelo de red.

4) Comparación básica con regresión

Aplicando un modelo de regresión a los mismos datos:

```
# Para hacer una regresión sobre el mismo modelo
reg1<-lm(data=compress, cstrength~age+water)
summary(reg1)

# Para obtener el MSE y el RMSE

MSE=mean(reg1$residuals^2)
RMSE<-sqrt(MSE)
MSE
RMSE
```

Red

MSE=135.93

RMSE=11.65

Regresión

MSE=204.76

RMSE=14.30

Aparentemente la Red funciona mejor que la regresión, pues el error en su ajuste a los datos es menor. ¿Es eso cierto?

Red: 13 parámetros

Regresión: 3 parámetros

Puede que se esté **sobreaajustando**, y que para **datos nuevos** la red funcione peor que la regresión. Por ello siempre será necesario utilizar remuestreo para poder evaluar la precisión en la predicción. Lo haremos en este ejemplo con una partición sencilla training-test.

5) Comparación con regresión vía partición training-test con el paquete caret

El esquema básico de trabajo del paquete caret es el siguiente:

- 1) Crear objeto trainControl para dividir train-test
- 2) Crear rejilla (grid) de valores de parámetros
- 3) Utilizar la función train para construir la red y evaluarla sobre datos test

```
# EJEMPLO: SEPARO EN TRAIN Y TEST USANDO CARET
# Importante controlar la semilla de aleatorización

library(caret)
set.seed(12346)

# 1) Crear objeto trainControl para training test
control<-trainControl(method = "LGOCV",p=0.8,number=1,savePredictions
= "all")
# 2) Crear rejilla de valores de parámetros
nnetgrid <-expand.grid(size=c(3),decay=c(0.1))
# 3) Función train para construir la red y evaluarla sobre datos test

rednnet<-train(cstrength~age+water,data=compressbien,
method="nnet",linout = TRUE,trControl=control,tuneGrid=nnetgrid)

rednnet
```

Neural Network

1030 samples
2 predictor

No pre-processing

Resampling: Repeated Train/Test Splits Estimated (1 reps, 80%)

Summary of sample sizes: 826

Resampling results:

RMSE	Rsquared	MAE
10.18986	0.5912728	8.047731

Tuning parameter 'size' was held constant at a value of 3

Tuning parameter 'decay' was held constant at a value of 0.1

Comparo con regresión lineal, method=lm. Aquí no se usa grid de parámetros porque la regresión no tiene hiperparámetros.

```
reg1<- train(cstrength~age+water,
data=compressbien, method="lm",trControl=control)
```

reg1

Linear Regression

1030 samples
2 predictor

No pre-processing

Resampling: Repeated Train/Test Splits Estimated (1 reps, 80%)

Summary of sample sizes: 826

Resampling results:

RMSE	Rsquared	MAE
13.75035	0.2650774	11.24248

Tuning parameter 'intercept' was held constant at a value of TRUE

Finalmente, la red sí parece que funciona mejor que la regresión para datos nuevos.

Para evitar la dependencia del azar, repetimos el proceso cambiando la semilla de construcción de la división train-test.

Se trata de un programa sencillo con un bucle que varía la semilla y presenta los valores de los errores en red y regresión

```
# EJEMPLO VARIANDO LA SEMILLA, TRAINING TEST UNA SOLA VEZ
# Ponemos trace=FALSE en train para evitar listados innecesarios

for (semilla in 120:125)
{
  set.seed(semilla)
  control<-trainControl(method = "LGOCV",p=0.8,number=5,savePredictions
= "all")

  nnetgrid <-expand.grid(size=c(3),decay=c(0.1))

  rednnet<- train(cstrength~age+water,
    data=compressbien, method="nnet",linout = TRUE,
    trControl=control,tuneGrid=nnetgrid,trace=FALSE)

  cat("\n")
  print(semilla)
  cat("\n")
  print(rednnet$results$RMSE)

  # Comparo con regresión lineal, method=lm. Aquí no hay grid.

  regl<- train(cstrength~age+water,
    data=compressbien, method="lm",trControl=control)

  print(regl$results$RMSE)

}
```

Resultados (semilla, error test en red, error test en regresión)

[1] 120

[1] 11.62086

[1] 14.45374

[1] 121

[1] 11.56528

[1] 14.28581

[1] 122

[1] 11.88971

[1] 14.15259

[1] 123

[1] 14.58365

[1] 124

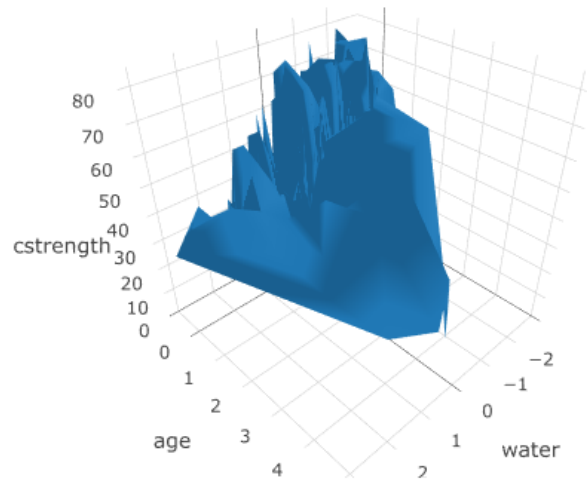
[1] 12.20924

[1] 13.93672

[1] 125

[1] 12.30061

[1] 14.60968



Se observa que la red supera a la regresión sobre datos test, y se recuerda que la relación entre cstrength y age, water era no lineal, como se ve en el gráfico. La no linealidad favorece al modelo de red frente al de regresión.

Ejemplo básico con variables input categóricas

Codificación de las variables input categóricas

Las variables input categóricas por simplicidad deben ser codificadas a dummy (0,1) antes de procesarlas en un modelo de red. Se utilizarán k-1 nodos para una variable categórica con k categorías.

Ejemplo: Predecir la capacidad pulmonar AERO a partir de pulsaciones, sexo y edad.

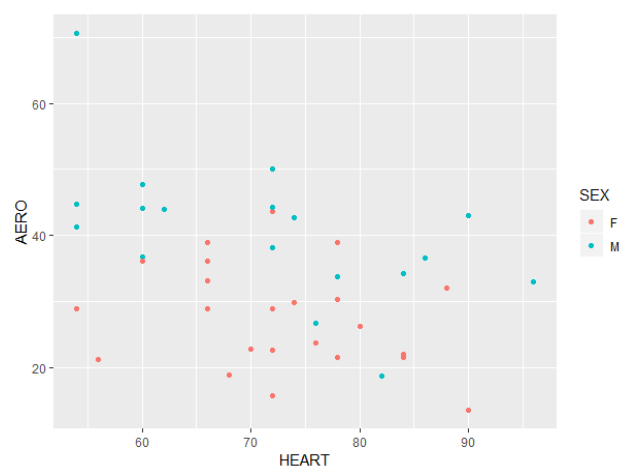
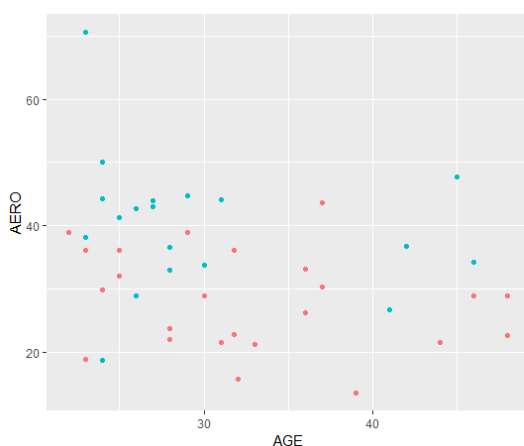
Archivo de datos `fitness.Rda`

Archivo de código `ejemplo fitness.R`

AERO	TEACHER	AGE	SEX	HEART	EXER
36.6	Yang	28.00	M	86.00	2.00
26.7	Yang	41.00	M	76.00	3.00
33.8	Yang	30.00	M	78.00	2.00
13.6	Yang	39.00	F	90.00	1.00

```
load("fitness.Rda")
```

La relación parece más bien **lineal**: terreno más apropiado para la regresión que para la red. Veremos.



```
# Gráficos básicos con puntos coloreados por sexo
ggplot(data, aes(AGE, AERO, color=SEX))+geom_point()
ggplot(data, aes(HEART, AERO, color=SEX))+geom_point()
```

Recordemos el pre-procesado:

- 1) Si hay valores missing en las variables input, deben eliminarse esas observaciones o imputar los valores missing.
- 2) Si hay variables input categóricas, deben pasarse a dummy.
- 3) Las variables input continuas deben estandarizarse. La razón es porque los algoritmos de optimización funcionan así mejor pues están menos expuestos a overflow y valores extremos de los parámetros.

AERO	TEACHER	AGE	SEX	HEART	EXER
36.6	Yang	28.00	M	86.00	2.00
26.7	Yang	41.00	M	76.00	3.00
33.8	Yang	30.00	M	78.00	2.00
13.6	Yang	39.00	F	90.00	1.00

Para el modelo con AGE (continua y tiene missing), SEX (categórica), HEART (continua) como input, hay que realizar las operaciones 1),2),3)

```
# Preparación del archivo

# na.omit se utilizaría para eliminar las observaciones con algún
# missing en alguna variable, no lo hacemos en este archivo

# data<-na.omit(data)

# IMPUTACIÓN POR LA MEDIA EN CONTINUAS

for (vari in listconti)
{
  data[,vari]<-ifelse(is.na(data[,vari]),
    mean(data[,vari],na.rm=TRUE),data[,vari])
}

# a)pasar las categóricas a dummies en todo el archivo
library(dummies)

if (listclass!=c(""))
{
  databis<-data[,c(vardep,listconti,listclass)]
  databis<- dummy.data.frame(databis, listclass, sep = ".")
} else {
  databis<-data[,c(vardep,listconti)]
}
# databis es el nuevo archivo con dummies, sin missing
```

Estandarizo las continuas y uno con el resto de archivo

```
# ESTANDARIZACIÓN

means <-apply(databis[,listconti],2,mean,na.rm=TRUE)
sds<-sapply(databis[,listconti],sd,na.rm=TRUE)

databis2<-scale(databis[,listconti], center = means, scale = sds)

numerocont<-which(colnames(databis)%in%listconti)
databis<-cbind(databis2,databis[, -numerocont,drop=FALSE ])
```

Comparo modelos sobre datos test

```
# En el modelo se ponen todas las dummies menos una referentes
# a las categorías de las variables categóricas
library(caret)
set.seed(12346)
# Training test una sola vez

control<-trainControl(method = "LGOCV",p=0.8,number=1,savePredictions
= "all")
nnetgrid <-expand.grid(size=c(3),decay=c(0.1))

rednnet<- train(AERO~HEART+AGE+SEX.F,
  data=databis, method="nnet",linout = TRUE,maxit=100,
  trControl=control,tuneGrid=nnetgrid)

# EL ERROR PRESENTADO ES EL ERROR SOBRE DATOS TEST
rednnet

RMSE      Rsquared    MAE
13.81197  0.2070891  10.19762

# Comparo con regresión lineal, method=lm. Aquí no hay grid.

reg1<- train(AERO~HEART+AGE+SEX.F,
  data=databis, method="lm",trControl=control)

reg1

RMSE      Rsquared    MAE
10.4509  0.334525  10.25568
```

Parece claramente mejor la regresión. Si repetimos el proceso con diferentes semillas, con un programa similar al del ejemplo anterior gana la regresión:

```
[1] 120
[1] 8.281393
[1] 8.006199
[1] 121
[1] 11.71235
[1] 10.4142
[1] 122
[1] 6.925477
[1] 9.133142
[1] 123
[1] 9.645709
[1] 7.784926
[1] 124
[1] 11.21359
[1] 8.548712
[1] 125
[1] 10.64062
[1] 7.339037
```

Por qué es mejor la regresión en este caso?

- 1) Relación lineal clara, óptima para regresión
- 2) Insuficientes observaciones para estimar bien los parámetros de red

Ejercicio 1.1

Archivo `ozono.Rda`

1) Construir una red neuronal con 5 nodos los siguientes modelos:

Variable dependiente continua=ozone

Modelo 1:Variables input continuas= temp invHt vis hum

Modelo 2:Variables input continuas= temp invHt vis hum milPress invTemp

Nota: realizar previamente el pre-procesado, estandarizando las variables continuas

2) Realizar la comparación con regresión bajo el esquema

a) Training-test

b) Training-test repetido

Ejercicio 1.2

Archivo `autompg.Rda`

1) Construir una red neuronal con 5 nodos el siguiente modelo:

Variable dependiente continua=mpg

Modelo 1:Variables input continuas= displacement horsepower weight origin modelyear

Nota: realizar previamente el pre-procesado, estandarizando las variables continuas del modelo , e imputando los missing si es necesario

Realizar la comparación con regresión bajo el esquema

a) Training-test

b) Training-test repetido

2) Considerar modelyear como categórica: Construir dummies y realizar el modelo 1 con las mismas variables pero modelyear como categórica, en dummies.

Arquitectura de la Red

Número de capas ocultas

Aunque con una capa oculta en la mayor parte de los casos suele ser suficiente (los teoremas de aproximación así lo aseguran), en ciertos casos complejos el modelo puede mejorar añadiendo alguna capa oculta más.

Recientemente, se ha impuesto el uso de muchas capas ocultas, pero sobre todo en los problemas a los que se suele aplicar Deep Learning (proceso de imágenes, texto, sonido, etc.), problemas que por su complejidad necesitan varias capas.

En muchas aplicaciones de modelos predictivos en inteligencia de negocios suele ser suficiente una sola capa.

Número de nodos

El mínimo número de nodos **necesario** para un buen modelo predictivo aumenta teniendo en cuenta los siguientes aspectos:

- Número de variables input
- Número de variables output
- Complejidad del problema (variables de diferente tipo, fuerte no linealidad, etc.)
- Varianza no explicada (no explicable) ["ruido"] del output
- Tipo de técnica de optimización utilizado en la red: algunas técnicas se pueden (y deben) utilizar con muchos nodos, otras manifiestamente no.

Pero...ese **mínimo** número de nodos **necesario** puede no alcanzarse si no tenemos suficientes observaciones.

Recopilación de Recomendaciones para fijar el número de nodos, dados los datos

Estas frases están tomadas de libros y manuales sobre redes neuronales:

- "10-20 observaciones por parámetro"
- "Para regresión, entre 5 y 25 observaciones por parámetro; para clasificación, entre 5 y 25 observaciones en la categoría más pequeña, por parámetro" (SAS-manual)
- "Tantos como dimensiones en componentes principales suficientes para capturar un 70-90 % de la varianza de los inputs"
- "menos que 1/30 de los casos de entrenamiento"
- "Para ajustar 20 nodos es necesario habitualmente tener entre 150 y 2500 observaciones" (Bishop).

Etc.

Estas recomendaciones tomadas así no suelen tener sentido, pues cada una de ellas deja de tener en cuenta alguna faceta (variables input, observaciones, complejidad, algoritmo, ruido, etc.)

Reglas simples a tener en cuenta para decidir el número de nodos y para comprender su efecto en las predicciones

a) Respecto a ajuste-sobreajuste: si queremos menos error, aumentamos el número de nodos, pero corremos el riesgo de que el modelo es demasiado complejo y funcione mal para nuevas observaciones test.

Número de nodos

-

+

Ajuste insuficiente

Sobreajuste

b) Respecto a complejidad de los datos (número de variables, relaciones raras, muchas categóricas, etc.). **Más complejidad requiere más nodos**, porque con pocos nodos la red puede no ajustarse bien. Recíprocamente, **si los datos son simples, demasiados nodos pueden provocar sobreajuste**.

Número de nodos

-

+

Datos menos complejos

Datos más complejos

c) Respecto al número de observaciones. Más observaciones nos permiten más nodos, pocas observaciones son insuficientes. La regla de 20 observaciones por parámetro también es algo razonable a respetar.

sobreajuste.

Número de nodos

-

+

Menos observaciones

Más observaciones

Notas:

- 1) Muchos paquetes parten por defecto de una red con 3 nodos, mínimo número a partir del cual la red puede empezar a estimar relaciones no lineales en la práctica.
- 2) Nosotros utilizaremos el método **prueba-error** utilizando técnicas de remuestreo, variando el número de nodos y observando el resultado sobre el error de predicción en datos test, siempre intentando comprender y verbalizar las razones por las cuales funciona mejor un número específico de nodos que otro, teniendo en cuenta las reglas simples anteriores.

Ejemplo: variando el número de nodos en el archivo compress

Es bueno hacer una reflexión a priori para desarrollar nuestra intuición:

- 1) Las relaciones parecen no lineales: la red puede funcionar bien
- 2) El modelo solo tiene dos variables, es relativamente sencillo; puede que con pocos nodos se consiga ajustar bien pero necesitará un cierto número de nodos para ajustar la curva.
- 3) Hay 1030 observaciones. Según la fórmula $h(k+1)+h+1$ es el número de parámetros, h =nodos ocultos, k =nodos input. Para tener 20 observaciones al menos por parámetro, podemos tener como máximo $1030/20=51$ parámetros. $h(k+1)+h+1=1030/20$ implica, al ser $k=2$ nodos input en nuestro modelo,

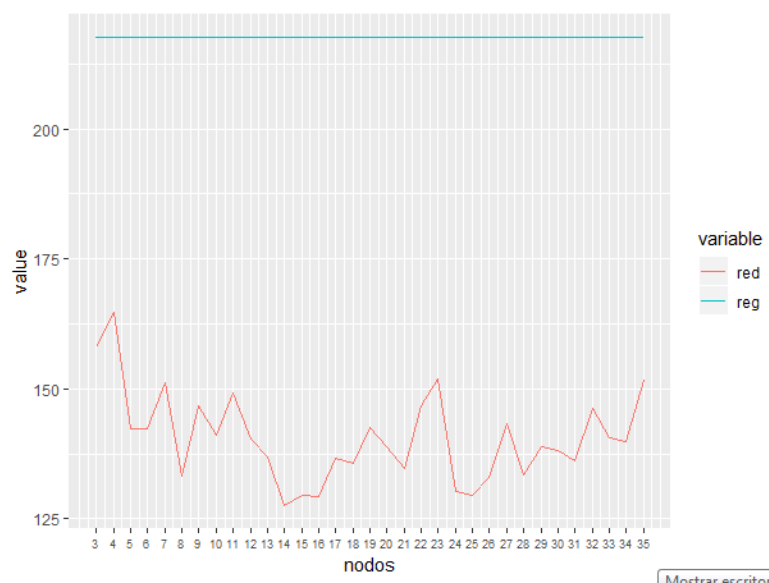
$4h+1=51.5$ o sea como máximo $h=12$ nodos.

(no es una regla exacta; sirve como referencia inicial, si fijamos 25 obs por parámetro salen $h=10$ nodos).

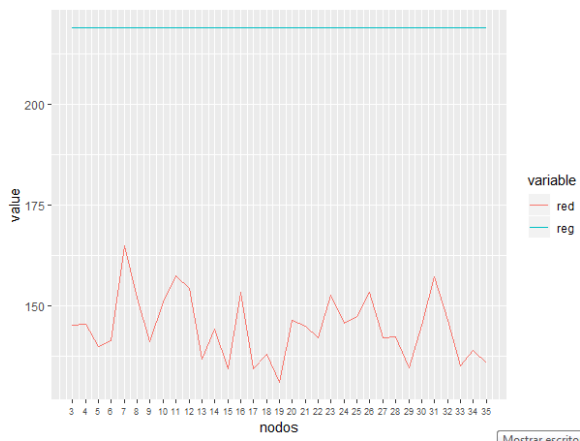
Se puede ir cambiando manualmente en el programa básico el número de nodos, pero técnicamente lo mejor es programar un bucle (**bucle nodos 1.R**).

Tomamos como referencia el bucle sobre la semilla train-test. Se hace el bucle sobre el número de nodos, pero hay que tener en cuenta que es sobre una prueba train-test, y puede depender de esta división.

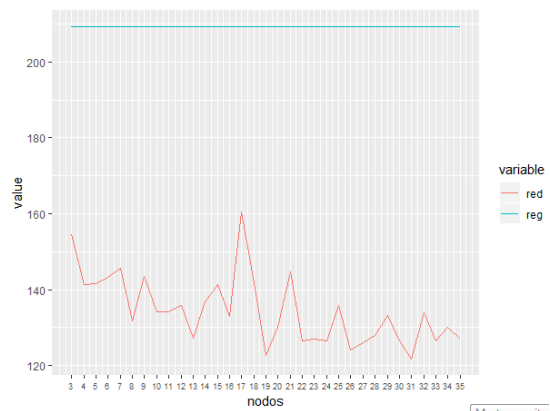
Semilla train-test 12345: 14 nodos parece bien



Semilla train-test 12346: 19 nodos



Semilla train-test 12347: 31 nodos



Obviamente hay variabilidad dependiendo del reparto train-test: una sola prueba train-test no es fiable y hay que tomar los resultados de modo global e intuitivo. Parece que se necesitan un número de nodos mínimo para ajustar bien.

Teniendo en cuenta la restricción del número de observaciones y el hecho de que hasta 12-13 nodos el error parece descender, normalmente nos quedaríamos con 12-13 nodos como mucho.

Con estos nodos se mejorará mucho a la regresión, sin riesgos de modelos exageradamente complicados.

Ejercicio 1.3

Explorar el número de nodos óptimo en el modelo *temp invHt vis hum* del archivo ozono. Razonar sobre el número de nodos máximo.

Simplemente utilizar el esquema (bucle nodos 1.R) cambiando lo necesario (archivo, listconti, variable dependiente).

Conceptos básicos sobre optimización

El objetivo del algoritmo de optimización es minimizar la función de error o función objetivo:

$$MSE = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

La predicción \hat{y}_i depende de la forma funcional de la red. En el primer ejemplo visto,

$$\begin{aligned} \hat{y}_i = & \tanh(w_{1,out}(\tanh(w_{11}x_1 + w_{21}x_2 + w_{31}x_3 + w_{41}x_4 + b_1)) \\ & + w_{2,out}(\tanh(w_{12}x_1 + w_{22}x_2 + w_{32}x_3 + w_{42}x_4 + b_2)) + \\ & + w_{3,out}(\tanh(w_{13}x_1 + w_{23}x_2 + w_{33}x_3 + w_{43}x_4 + b_3)) + \\ & + w_{4,out}(\tanh(w_{14}x_1 + w_{24}x_2 + w_{34}x_3 + w_{44}x_4 + b_4)) + \\ & + b_{out}) \end{aligned}$$

Habría que hallar los parámetros (que llamaremos pesos o weights) $w_{1,out}$, w_{11} , w_{21} , ..., etc. que minimicen el MSE. Es decir, se trata de minimizar MSE como función de los parámetros $w_{1,out}$, w_{11} , w_{21} , ..., etc.

Es decir

$$\begin{aligned} \min_{w_{ij}} MSE &= \min_{w_{ij}} \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2 = \\ \min_{w_{ij}} \sum_{i=1}^n (y_i - &\tanh(w_{1,out}(\tanh(w_{11}x_{1i} + \dots) + w_{2,out}(\tanh(w_{21}x_{1i} + \dots) + \dots))^2 \end{aligned}$$

Se trata de una función de los parámetros w_{ij} . Hay que hallar los valores de los w_{ij} que hagan mínima esa función.

Todos los valores x_{1i} , x_{2i} , etc. son valores reales de nuestros datos. El sumatorio tiene tantos términos como observaciones (por ejemplo, $n=5000$ términos).

La complejidad y dificultad del proceso de optimización viene dada por el número de parámetros a estimar. A menudo son 200 o 300 parámetros, y en deep learning llegan a ser un número muy alto (p.ej. 5000).

Como se trata de una función de los parámetros no lineal y compleja, es necesario utilizar **algoritmos numéricos de aproximación**. En general, el funcionamiento es similar en todos ellos:

- 1) Tomar unos **valores iniciales** para los pesos (se suelen aleatorizar)
- 2) Calcular la **función de error** con esos valores
- 3) Calcular una **dirección de decrecimiento** de la función de error
- 4) **Retocar** los valores de los pesos en esa dirección de decrecimiento
- 5) Volver al paso 2)

El algoritmo se detiene al llegar a cualquier **criterio de parada** preestablecido.

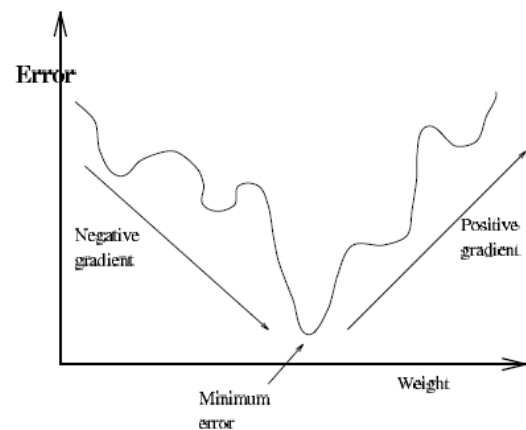
En general los algoritmos que utilizaremos se basarán en el **Algoritmo básico Gradient Descent**. El objetivo es hallar los pesos w_{ij} que minimicen el error (MSE).

Proceso básico:

- 1) Se inicializan los pesos w_{ij} aleatoriamente
- 2) Se calcula la derivada de la función del error en ese punto (esos pesos concretos w_{ij}): $\frac{\partial E(n)}{\partial w_{ij}}$
- 3) Se hacen variar los pesos en la dirección de descenso del error (recordemos que w_{ij} es un vector) (los pesos se varían según un learning rate ϵ):

$$\Delta w_{ij}(n) = -\epsilon \frac{\partial E(n)}{\partial w_{ij}}$$

$$w_{ij}(n+1) = w_{ij}(n) + \Delta w_{ij}(n)$$



- 4) Se repiten los pasos 2 y 3 hasta llegar a un criterio de parada.

En ciertas versiones se introduce una constante α llamada weight decay o momentum, que provoca que a medida que aumentan las iteraciones el efecto del cambio es menor:

$$\Delta w_{ij}(n) = -\epsilon \frac{\partial E(n)}{\partial w_{ij}} + \alpha \Delta w_{ij}(n-1)$$

Notas:

1) El learning rate ϵ refleja en qué medida vamos a cambiar los pesos w_{ij} en cada iteración:

a) learning rate muy bajo (p.ej. 0.001) hace que el proceso de optimización sea lento, y con el peligro de quedarse enganchado en óptimos locales (el entrenamiento preliminar reduce este riesgo)

b) Learning rate muy alto (p. ej. 0.5) hace diverger demasiado los valores provocando inconstancia y mala optimización

2) El momentum (<1 siempre) hace que en cada iteración del algoritmo los pesos se vayan cambiando algo menos, acercándose “con cuidado” al mínimo, más despacio cada vez .

Nota 1

La **inicialización aleatoria de los pesos** puede tener gran influencia en los resultados, cambiando mucho los parámetros finales de la red, si ésta no está bien calibrada o el programa de optimización no es demasiado fino. (**probar a variar la semilla de la red en el primer ejemplo cstrength –age, water y observar el MSE sobre datos test**).

Una manera de corregir esto es construir la red de manera repetida con diferentes semillas y promediar la predicción. R permite esto con el paquete `avnnnet` del paquete `caret` (se verá más adelante). Otro modo para un tipo de datos específico es tener unos pesos preentrenados con unos valores básicos de inicio. Otra manera es hacer “entrenamiento preeliminar”, es decir, probar la red con pocas iteraciones con diferentes sets de nodos con valores aleatorios y finalmente ejecutar la red con aquel set de nodos que funcionó mejor en esas pruebas con pocas iteraciones.

Nota 2

Aumentar mucho el número de iteraciones en algunos casos puede afectar negativamente a la red, ajustando “demasiado bien” los pesos y provocando mala generalización (la red funciona mal para datos test a partir de un número de iteraciones). Reducir el número de iteraciones para evitar esto es una técnica que se denomina “early stopping”. No siempre es necesario, depende de los casos.

Estrategias básicas para la Comparación de Modelos

De cara a poder comparar diferentes modelos de red (con diferentes variables, número de nodos, iteraciones, etc.) es necesario disponer de medidas de diagnóstico que nos digan si un modelo es mejor que otro. Aunque nunca se deben de tomar estrictamente, nos orientan y son la principal base para decidirnos por un modelo.

Algunas medidas clásicas para modelos de regresión

Como siempre en este capítulo suponemos que la variable objetivo y es continua.

Se comentarán tres medidas clásicas:

- R^2
- AIC
- BIC

Partiendo de la fórmula

$$ASE = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

1) La referencia más conocida en modelos de regresión es el R^2

$$R^2 = \frac{\sum_{i=1}^n (y_i - \hat{y}_i)^2}{\sum_{i=1}^n (y_i - \bar{y}_i)^2}$$

Su principal ventaja es que $0 \leq R^2 \leq 1$, con lo cual $R^2=1$ implica ajuste o relación perfecta entre la variable objetivo y las variables input. Como una primera aproximación está bien, pero dos problemas la hacen poco útil:

- a) Es una medida de ajuste, solo mide cuánto se ajustan nuestros datos al modelo. Pero no nos dice qué tal se ajustarían datos nuevos a nuestro modelo (predicción).
- b) A medida que introducen variables o complejidad en el modelo, R^2 aumenta automáticamente, con lo cual usar solo esta medida nos llevaría a modelos sobreajustados. Por ejemplo introducir variables x en el modelo que no tienen relación con la y aumentaría el R^2 a pesar de empeorar el modelo.

2) La segunda referencia más importante es el **AIC** (Akaike Information Criterion)

$$AIC = n \log(ASE) + 2p$$

donde p es el número de parámetros del modelo.

El AIC se evalúa de la siguiente manera: **cuánto mayor, peor**.

Ventajas:

- a) Penaliza el ajuste (ASE) con el término $2p$, de modo que modelos menos complejos serán preferidos a modelos más complejos con el mismo ajuste. En eso es una mejora al R^2

Desventajas:

- a) No tiene escala, en cada problema toma valores diferentes, pudiendo tomar valores negativos.
- b) No mide qué tal funcionaría el modelo sobre datos nuevos (predicción).
- c) En general sirve para comparar modelos dentro de un mismo algoritmo (regresión frente a regresión), pero no debería ser usado para comparar entre diferentes algoritmos (redes frente a regresión)

3) El **BIC** (Bayesian Information Criterion) es una alternativa al AIC

$$BIC = n \log(ASE) + p \log(n)$$

donde p es el número de parámetros del modelo.

Tiene las mismas ventajas y desventajas que el AIC, aunque el término de penalización depende de n , con lo cual a mayor n , más se penaliza la complejidad.

Diferencias entre AIC y BIC :

- a) En principio BIC escoge asintóticamente (es decir, a medida que aumenta n), con probabilidad 1, el modelo óptimo si está entre los candidatos. AIC por otro lado, tiende a escoger, de los modelos candidatos, el que tiene menor error asintóticamente.
- b) Desde el punto de vista práctico, AIC tiende a recomendar modelos más complejos (por ejemplo, en regresión con más variables), y BIC tiende a recomendar modelos menos complejos. AIC puede tender al sobreajuste y BIC al no ajustar lo suficiente.

Técnicas de remuestreo

Las medidas anteriores tienen el defecto de ser ante todo, medidas de ajuste. Sin embargo, la complejidad de los métodos más modernos exige poder evaluar la eficacia predictiva, y evitar en la medida de lo posible, el sobreajuste.

Training, Validation, Test

Se dividen aleatoriamente (o por procedimientos de muestreo) los datos en tres grupos. Se necesitan muchas observaciones. Habitual es (70, 20,10), (60, 20,20), etc. dependiendo del número de observaciones.

Datos Training--> Se utilizan para estimar los posibles modelos

Datos Validation--> Se aplican los modelos sobre esos datos y se observa su performance. A veces intervienen un poco en el proceso de construcción de modelos (Redes Neuronales, Gradient Boosting).

Datos Test--->Se utilizan para calibrar el funcionamiento predictivo real de los modelos. Sirven para evaluar el error de predicción del modelo "óptimo".

Training, Test

Se dividen aleatoriamente los datos en dos grupos.

Datos Training--> Se utilizan para estimar los posibles modelos

Datos Test--> Se aplican los modelos sobre esos datos y se observa su performance.

El error de predicción puede estar subestimado (en realidad puede ser mayor, pues se ha elegido el modelo que mejor funciona en los datos test).

Validación Cruzada (k grupos)

Si no se tienen muchas observaciones, la división en Training-Validación-Test arrojará resultados muy variables según la selección aleatoria de esos grupos.

Algoritmo de validación cruzada

1) Se dividen los datos aleatoriamente en k grupos

2) Se realiza iterativamente la siguiente operación:

Desde $i=1$ hasta k

Dejar aparte el grupo i

Construir el modelo con los grupos restantes

Estimar el error (por ejemplo ASE) al predecir el grupo i: $Error_i$

Fin

3) Una medida de error de predicción del modelo será la suma o media de los $Error_i$



Ventajas de la validación cruzada:

- Es el mejor esquema, pues utiliza todos los datos como si fueran datos test
- La validación cruzada repetida permite evaluar la variabilidad del modelo

Desventajas de la validación cruzada:

- A veces el resultado (qué modelo funciona mejor) depende del número de grupos k
- El método también depende de la asignación aleatoria a los grupos (habría que probar con varias semillas)
- El error de predicción también suele estar subestimado
- Algunos investigadores proponen incluir la selección de variables para cada iteración i

En la práctica habitual de comparación de modelos para variable dependiente continua:

1) Se suele utilizar ASE-MSE como medida básica del error de predicción

2) Para evaluar la capacidad predictiva del modelo, **de peor a mejor:**

a) Training-test una sola vez

Recomendado solo para datos muy grandes, y siempre dependiendo de la complejidad del modelo (número de variables, número de parámetros) $\gg 10.000$ observaciones (por decir algo).

b) Training-test repetido, variando la semilla de selección

Mejor que el anterior esquema, pero solo para datos muy grandes, y siempre dependiendo de la complejidad del modelo (número de variables, número de parámetros) $\gg 5.000$ observaciones (por decir algo).

c) Validación cruzada una sola vez

Mejor que el anterior esquema, para datos moderados o grandes, según la potencia del ordenador, y siempre dependiendo de la complejidad del modelo (número de variables, número de parámetros) $\gg 5.000$ observaciones (por decir algo).

d) Validación cruzada repetida, variando la semilla de ordenación

El mejor esquema, para datos pequeños, moderados o grandes, según la potencia del ordenador, y siempre dependiendo de la complejidad del modelo (número de variables, número de parámetros) $\gg 500$ observaciones (por decir algo).

La tabla siguiente está extraída de un artículo de comparación del uso de redes neuronales en artículos de business analytics . El artículo es ya antiguo, de 2009, y se puede observar cómo muchos de los métodos utilizados no son suficientes para comparar: Se utiliza Tr-Ts (training-Test) sin repetición a menudo, y muy poca validación cruzada (CV). El tamaño muestral es a veces demasiado pequeño para poder comparar bien, etc. El estándar ha mejorado con los años debido a los recursos de software y hardware y no ahora se admiten generalmente en publicaciones científicas comparaciones poco fiables.

Table 1
Applications in accounting and finance

Reference	Statistical model	No. of variables	Sample size	Validation Method	Error measure	Finding
Odom and Sharda (1990)	DA	5	129	Tr-Ts /R-3 times	Confusion matrix	[A]
Duliba (1991)	Reg	5-10	600	Tr-Ts	R^2 Value	[A]-Random effect [C]-Fixed effect
Salchenberger et al. (1992)	Logit	29	3479	Tr-Va-Ts	Confusion matrix	[A]*
Tam and Kiang (1992)	k-NN, DA, ID3	19	118	Jackknifing	Confusion matrix	[A]
Fletcher and Goss (1993)	LR	3	36	18-fold CV	Confusion matrix, MSE	[A]
Yoon et al. (1993)	DA	4	151	Tr-Ts (50-50)	Confusion matrix	[A]
Altman et al. (1994)	DA	10- DA 15- NN	1108	Tr-Ts (70-30)	Confusion matrix	[C]
Dutta et al. (1994)	Reg, LR	6, 10	47	Tr-Ts (70-30)	Confusion matrix	[A]
Wilson and Sharda (1994)	DA	5	129	Tr-Ts/R-3 times	Confusion matrix	[A]*
Boritz and Kennedy (1995)	Logit, Probit, DA	5, 9	342	Tr-Ts (70-30) / R-5 times	Confusion matrix	[B]
Lenard et al. (1995)	LR	4 & 8	80	Tr-Ts (50-50)	Confusion matrix	[A]*
Desai et al. (1996)	LR, DA	18	2733	Tr-Ts (70-30) / R-10 times	Confusion matrix	[B]*
Leshno and Spector (1996)	DA	41	88	Tr-Ts	Confusion matrix	[A]*
Jo et al. (1997)	DA, CBR	20	564	Tr-Ts	Confusion matrix	[A]*
Spear and Leis (1997)	DA, LR, Reg	4	328	Tr-Va-Ts (76-12-12)	Confusion matrix	[B]
Zhang et al. (1999)	LR	6	220	5-fold CV	Confusion matrix	[A]*
Lee and Jung (2000)	LR	11	21678	Tr-Ts	C-index, Some measure for degree of separation	[A]-Rural customer [C]-Urban customer
Limsombunchai et al. (2005)	LR	11	16560	Tr-Ts	Confusion matrix	[B]
Lee et al. (2005)	DA, LR	5	168	4-fold CV	Confusion matrix	[A]*
Pendharkar (2005)	C4.5, DA	3	100- sim 200-real	Bootstrapping	Confusion matrix	[A]*
Landajo et al. (2007)	Robust reg, Loglinear reg	9	Multiple models	Tr-Ts	MAE	[C]*

*Neural networks and statistical techniques: A review of applications. Mukta Paliwal, Usha A. Kumar *. Expert Systems with Applications 36 (2009) 2-17*

El paquete caret de R

Paquete para comparación y tuneado de modelos predictivos. Incorpora modelos predictivos de muchos paquetes de R y contiene un esquema propio para tuneado de parámetros y comparación vía remuestreo

Ventajas

Permite training test repetido, validación cruzada repetida, etc.

Permite rápido y sencillo tuneado de parámetros con salida sencilla

Admite muchos modelos predictivos de paquetes de R:

<https://rdrr.io/cran/caret/man/models.html>

Desventajas

No permite el tuneado de todos los parámetros para cada técnica o paquete; algunos faltan.

No incorpora todos los paquetes (por ejemplo h2o no está, de momento)

Como se observa en el siguiente ejemplo, caret permite utilizar todas las técnicas de remuestreo vistas a través de un sencillo esquema.

Archivo de código [ejemplos caret compress.R](#)

Las cuatro técnicas de remuestreo básicas con caret

```
# *****
# TUNING CON CARET
# *****

set.seed(12346)

# Training test una sola vez
control<-trainControl(method = "LGOCV",p=0.8,number=1,savePredictions
= "all")

# Training test repetido
control<-trainControl(method = "LGOCV",p=0.8,number=5,savePredictions
= "all")
# (aquí number es el número de repeticiones)

# Validación cruzada una sola vez
control<-trainControl(method= "cv",number=4,savePredictions = "all")
# (aquí number es el número de grupos de validación cruzada)

# Validación cruzada repetida
control<-trainControl(method="repeatedcv",number=4,repats=5,savePredic
tions = "all")

# (aquí number es el número de grupos de validación cruzada y repats
# el número de repeticiones)
```

Esquema general de trabajo con caret

1) Se elige el **esquema de evaluación del modelo** (train-test, cv, repetido o no...) con la función **trainControl**.

```
control<-trainControl(method = "LGOCV",p=0.8,number=1,savePredictions
= "all")
```

2) Se genera una **rejilla (grid)** con parámetros a probar-tunear con la función **expand.grid** (puede ser un solo valor). En el ejemplo size es el número de nodos y decay el weight decay del proceso de optimización:

```
nnetgrid <- expand.grid(size=c(5,10,15,20),decay=c(0.01,0.1,0.001))
```

3) Se crea el **modelo** con la función **train**, poniendo el paquete función que se va a utilizar (**method="caret"**) y si es necesario algún parámetro más de la función (aquí por ejemplo linout=TRUE, maxit=100). Se nombra el método de control con **trControl** y la rejilla con **tuneGrid** (eventualmente se puede poner tuneGrid=NULL).

```
set.seed(123)
```

```
rednnet<- train(cstrength~age+water, data=compressbien,
method="nnet",linout= TRUE,maxit=100,
trControl=control,tuneGrid=nnetgrid)
```

```
rednnet
```

El objeto creado es un modelo-lista al estilo habitual de R, se puede utilizar con predict, etc. Al final del proceso caret elige la mejor combinación de parámetros, pero esto requiere más pruebas y afinamiento por parte del usuario para estar seguro.

No pre-processing

Resampling: Repeated Train/Test Splits Estimated (1 reps, 80%)

Summary of sample sizes: 826

Resampling results across tuning parameters:

size	decay	RMSE	Rsquared	MAE
5	0.001	12.64716	0.4491247	9.849803
5	0.010	12.51108	0.4618682	9.551499
5	0.100	12.61107	0.4528269	9.778720
10	0.001	13.01478	0.4193348	10.017053
10	0.010	12.89038	0.4309158	9.954152
10	0.100	12.44754	0.4686292	9.704466
15	0.001	12.75922	0.4410283	10.145570
15	0.010	12.12655	0.4944920	9.053970
15	0.100	12.68526	0.4494209	9.854102
20	0.001	12.32071	0.4844616	9.393490
20	0.010	12.27308	0.4847887	9.446235
20	0.100	12.33129	0.4774554	9.570546

RMSE was used to select the optimal model using the smallest value. The final values used for the model were size = 15 and decay = 0.01.

Utilizando avNNet

El paquete avNNet es como nnet pero promedia la predicción sobre varias redes con diferente aleatorización inicial de los parámetros.

Es claro que es más fiable utilizar avNNet que nnet si se puede. El paquete caret lo permite.

avNNet exige poner un parámetro más en el tuneado, bag (ponemos bag=FALSE) por si queremos hacer subsampling en cada construcción de red.

El número de repeticiones de las redes es arbitrario. Por defecto pondremos 5. Como cada ejecución implica 5 redes (repeats=5) y hemos puesto validación cruzada repetida, el proceso es largo. Lo probamos fijando size=15 y decay=0.01, para comparar el error cometido (RMSE) con nnet básico.

```
# UTILIZANDO avNNet
# Validación cruzada repetida
control<-trainControl(method = "repeatedcv",
  number=4, repeats=5, savePredictions = "all")

set.seed(123)
nnetgrid <- expand.grid(size=c(15), decay=c(0.01), bag=F)

rednnet<- train(cstrength~age+water,
  data=compressbien,
  method="avNNet", linout = TRUE, maxit=100,
  trControl=control, repeats=5, tuneGrid=nnetgrid)

rednnet
```

El valor del error promedio es mejor que con nnet (11.68), como se ve:

RMSE	Rsquared	MAE
11.29333	0.5447947	8.701083

Para comparar con regresión lineal se utiliza el mismo trainControl con la misma semilla

```
# Mismo ejemplo con regresión lineal
# Se utiliza el paquete lm (linear model). No hay parámetros.

control<-trainControl(method = "repeatedcv",
  number=4, repeats=5, savePredictions = "all")

set.seed(123)

regl<- train(cstrength~age+water,
  data=compressbien,
  method="lm", trControl=control)

regl
```

Se observa como la red (RMSE=11.29) es superior a la regresión (RMSE=14.37) en estos datos

No pre-processing

Resampling: Cross-Validated (4 fold, repeated 5 times)

Summary of sample sizes: 772, 772, 772, 774, 773, 771, ...

Resampling results:

RMSE	Rsquared	MAE
14.37004	0.2661054	11.54276

Si en el ejemplo se quiere utilizar el método más potente, validación cruzada repetida:

```
# Validación cruzada repetida
control<-trainControl(method = "repeatedcv",
  number=4, repeats=5, savePredictions = "all")

set.seed(123)
nnetgrid <- expand.grid(size=c(5,10,15,20), decay=c(0.01,0.1,0.001))

rednnet<- train(cstrength~age+water,
  data=compressbien,
  method="nnet", linout = TRUE, maxit=100,
  trControl=control, tuneGrid=nnetgrid)

rednnet
```

Los parámetros recomendados son diferentes, pero se ve que la diferencia en RMSE es mínima en este ejemplo y no merece la pena usar 20 nodos en lugar de 15 (por estas cosas el usuario debe estar atento y no automatizar demasiado los procesos):

size	decay	RMSE	Rsquared	MAE
5	0.001	11.94754	0.4910029	9.172898
5	0.010	11.81204	0.5016910	9.049938
5	0.100	11.77441	0.5046287	9.020851
10	0.001	11.98678	0.4865241	9.230794
10	0.010	11.87870	0.4958783	9.160337
10	0.100	11.74937	0.5076695	9.043405
15	0.001	11.69364	0.5130754	8.992823
15	0.010	11.68751	0.5157879	9.004645
15	0.100	11.55252	0.5237625	8.901551
20	0.001	11.54378	0.5272004	8.906013
20	0.010	11.59587	0.5227075	8.949583
20	0.100	11.57629	0.5241745	8.925807

RMSE was used to select the optimal model using the smallest value.
The final values used for the model were size = 20 and decay = 0.001.

El intercambio Sesgo-Varianza (The Bias-Variance Trade-off)

Un modelo $\hat{f}(x)$ es estimado por $f(x)$ a través de regresión, redes o cualquier algoritmo. En este caso, el error cuadrático esperado en la predicción para la observación x es:

$$Err(x) = E[(Y - \hat{f}(x))^2]$$

Este error puede ser descompuesto en las componentes sesgo, varianza, y error irreducible:

$$Err(x) = (E[\hat{f}(x)] - f(x))^2 + E[\hat{f}(x) - E[\hat{f}(x)]]^2 + \sigma_\epsilon^2$$

Que se puede nombrar así:

$$Err(x) = \text{sesgo}^2 + \text{Varianza} + \text{Error irreducible}$$

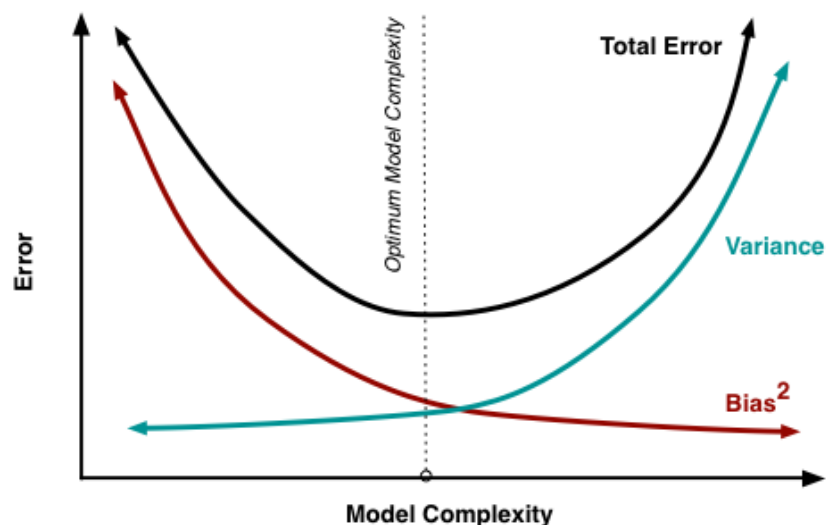
Como consecuencia, al utilizar un modelo predictivo, el **error global** cometido por éste se debe a **tres factores que se suman**:

- El sesgo², que es el error promedio cometido por desfase de los verdaderos valores.
- La varianza, que es la variabilidad de las predicciones del modelo al variar los datos utilizados para construir el modelo.
- El error “irreducible” o no explicable, llamado también “varianza no explicada” en muchos textos.

Normalmente deseamos modelos con bajo sesgo (bajo error promedio) y baja varianza (el modelo no varía mucho según los datos usados al construirlo). Pero en general no se pueden minimizar ambas cosas a la vez.

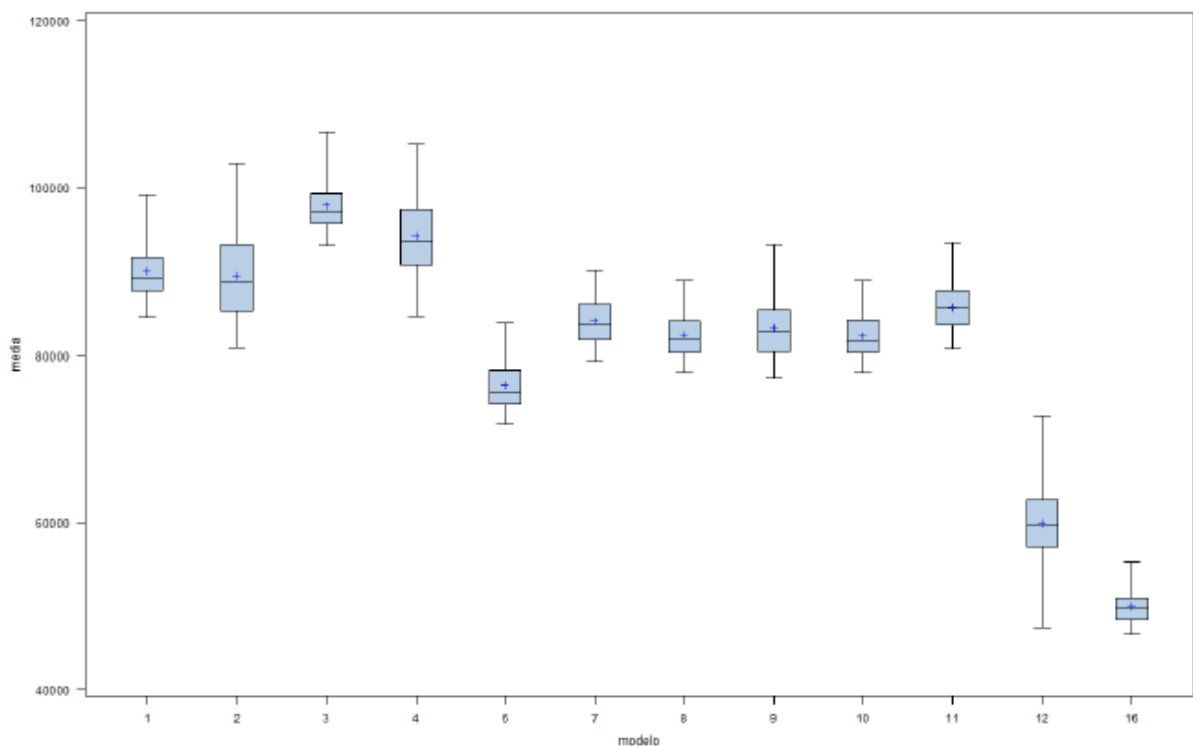
Si aumentamos la complejidad del modelo, bajamos el sesgo, pero aumentamos a la vez la varianza. Modelos con alta varianza pueden originar errores muy grandes en ocasiones puntuales, lo cual no es deseable a nivel práctico.

Si reducimos la complejidad del modelo, bajamos la varianza pero aumentamos el sesgo. Un sesgo-error promedio más alto sería el precio a pagar para evitar errores ocasionales demasiado altos (varianza alta).



Una manera de explorar el sesgo y varianza de los modelos es haciendo **validación cruzada repetida (k-fold repeated crossvalidation)** y observando el error promedio cometido (sesgo) y su variabilidad (longitud-altura de la caja). En este curso se utiliza este método a través de boxplot .

El modelo 12 tiene un bajo sesgo pero alta varianza; si es un modelo complicado puede ser peligroso/inestable a pesar de su bajo sesgo. La variabilidad observada en la caja es solamente indicativa y se calcula sobre el error promedio sobre muchas observaciones. En predicciones puntuales de una sola observación la variabilidad del error es mucho mayor. El modelo 16 tiene bajo sesgo y baja varianza: parece óptimo dados el resto de modelos.



Se presenta una tabla orientativa sobre el efecto que pueden tener ciertas acciones sobre el sesgo y varianza de nuestros modelos. Las técnicas avanzadas que se verán en Machine Learning tienen como objetivo mejorar el sesgo de los algoritmos clásicos (regresión), con el peligro implícito de aumentar la varianza y perder explicabilidad. Queda a juicio del investigador en cada caso particular si merece la pena asumir estas desventajas.

La tabla debe interpretarse como medidas a *probar* con el objetivo deseado (bajar sesgo o varianza). Como ejemplo de interpretación de la tabla, para bajar el sesgo se puede probar a aumentar el número de variables del modelo, y si se quiere bajar la varianza, se debería probar a reducir ese número de variables.

		Quiero bajar el sesgo	Quiero bajar la Varianza
General (1)	número de variables	+	-
	introducir interacciones entre variables, creación de dummies	+	-
	Complejidad del modelo	+	-
	tamaño muestral	+	+
Logística, regresión			
	p-valor (stepwise, backward, forward) (2)	+	-
	categorías poco representadas en variables input	+=	-
Redes			
	número de nodos	+	-
	linealidad (3)	-	-
	número de iteraciones en algoritmo de optimización (usar early stopping puede reducir la varianza)	+	-
	en algoritmo de optimización: learn rate, momentum (en general si subimos momentum bajamos learn rate). learn rate bajos requieren nº de iteraciones más altas	-	+

Notas

1) En general, hacer el modelo más complejo suele reducir el sesgo y aumentar la varianza. Esto incluye aumentar el número de variables, utilizar variables categóricas con muchas categorías, incluir interacciones, utilizar modelos no lineales (por ejemplo regresión polinómica), etc.

Aumentar el tamaño muestral tiene incidencia positiva sobre el sesgo y la varianza. Ambas cantidades disminuyen aumentando n .

2) El p-valor de corte en procesos de selección de variables mide la exigencia en la inclusión de variables o efectos en el modelo. A menor p-valor de corte, más exigentes somos en la inclusión de variables; reducir el p-valor implica entonces menos variables y por lo tanto más sesgo pero menos varianza. Aumentar el p-valor permite la inclusión de más variables y por lo tanto modelos más complejos que pueden tener menos sesgo pero más varianza.

3) La linealidad en modelos de regresión o separabilidad lineal en modelos de clasificación mejora en general ambos sesgo y varianza. Si se puede conseguir esta linealidad con transformaciones sencillas y utilizar regresión o regresión logística (en clasificación) puede ser mejor que probar algoritmos más complicados como redes.

4) Monitorizar modelos con excesivo número de aspectos a probar (activación, algoritmo, etc.) y centrándonos en el resultado que se tiene en datos test puede llevar a un sobreajuste implícito (mareamos demasiado nuestros datos). Esto se suele denominar Sesgo de Selección de Modelos.

Explorando sesgo-varianza a través de validación cruzada repetida y boxplot

Se utilizarán dos funciones en R:

- `cruzadaavnnnet` (para redes)
- `cruzadalin` (para regresión lineal)

Archivos de código [cruzada avnnnet y lin.R](#), [cruzada avnnnet continua ejemplos.R](#)

Ambas construyen la media de error por cada grupo en validación cruzada repetida para poder comparar modelos a través del boxplot, en el sentido de explorar sesgo y varianza.

```
## Ejemplo de utilización cruzada AVNNET
source("cruzadas avnnnet y lin.R")
data<-compressbien
medias1<-cruzadaavnnnet(data=data,
vardep="cstrength",listconti=c("age","water"),
listclass=c(""),grupos=4,sinicio=1234,repe=5,
size=c(15),decay=c(0.01),repeticiones=5,itera=100)

medias1$modelo="avnnnet"

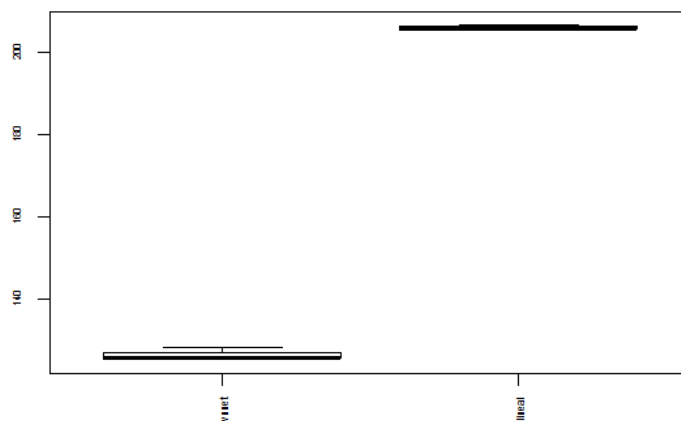
## Ejemplo de utilización cruzada regresión lineal
medias2<-cruzadalin(data=data,
vardep="cstrength",listconti=c("age","water"),
listclass=c(""),grupos=4,sinicio=1234,repe=5)

medias2$modelo="lineal"

## Unión de las medias y presentación del boxplot
union1<-rbind(medias1,medias2)

par(cex.axis=0.5)
boxplot(data=union1,error~modelo)
```

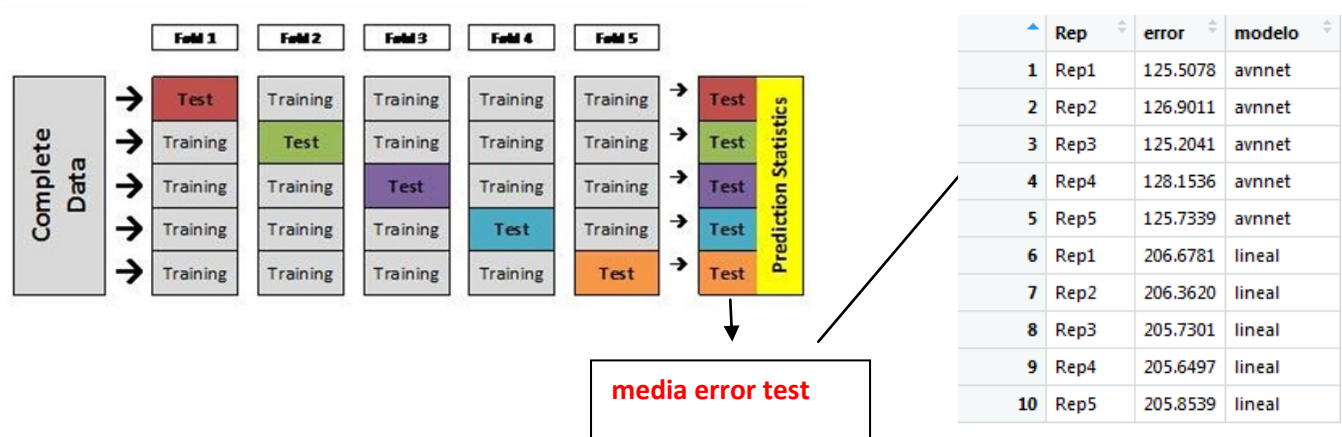
	Rep	error	modelo
1	Rep1	125.5078	avnnnet
2	Rep2	126.9011	avnnnet
3	Rep3	125.2041	avnnnet
4	Rep4	128.1536	avnnnet
5	Rep5	125.7339	avnnnet
6	Rep1	206.6781	lineal
7	Rep2	206.3620	lineal
8	Rep3	205.7301	lineal
9	Rep4	205.6497	lineal
10	Rep5	205.8539	lineal



Recordatorio

Cada repetición de validación cruzada se realiza reordenando aleatoriamente el archivo con una semilla diferente (aparecen como Rep1, Rep2, etc. en la tabla).

El valor del error que aparece en cada repetición de validación cruzada es el promedio del error test (columna de colores de la derecha).



Se pueden comparar varios modelos de redes, previamente han de haberse explorado los parámetros más interesantes con caret.

```
# Ejemplo comparación tres redes recomendadas por nuestro
# trabajo previo con tuning con caret
```

```
data<-compressbien
```

```
medias1<-cruzadaavnnnet(data=data,
vardep="cstrength",listconti=c("age","water"),
listclass=c(""),grupos=4,sinicio=1234,repe=5,
size=c(15),decay=c(0.01),repeticiones=5,itera=100)
```

```
medias1$modelo="avnnnet1"
```

```
medias2<-cruzadaavnnnet(data=data,
vardep="cstrength",listconti=c("age","water"),
listclass=c(""),grupos=4,sinicio=1234,repe=5,
size=c(20),decay=c(0.001),repeticiones=5,itera=100)
```

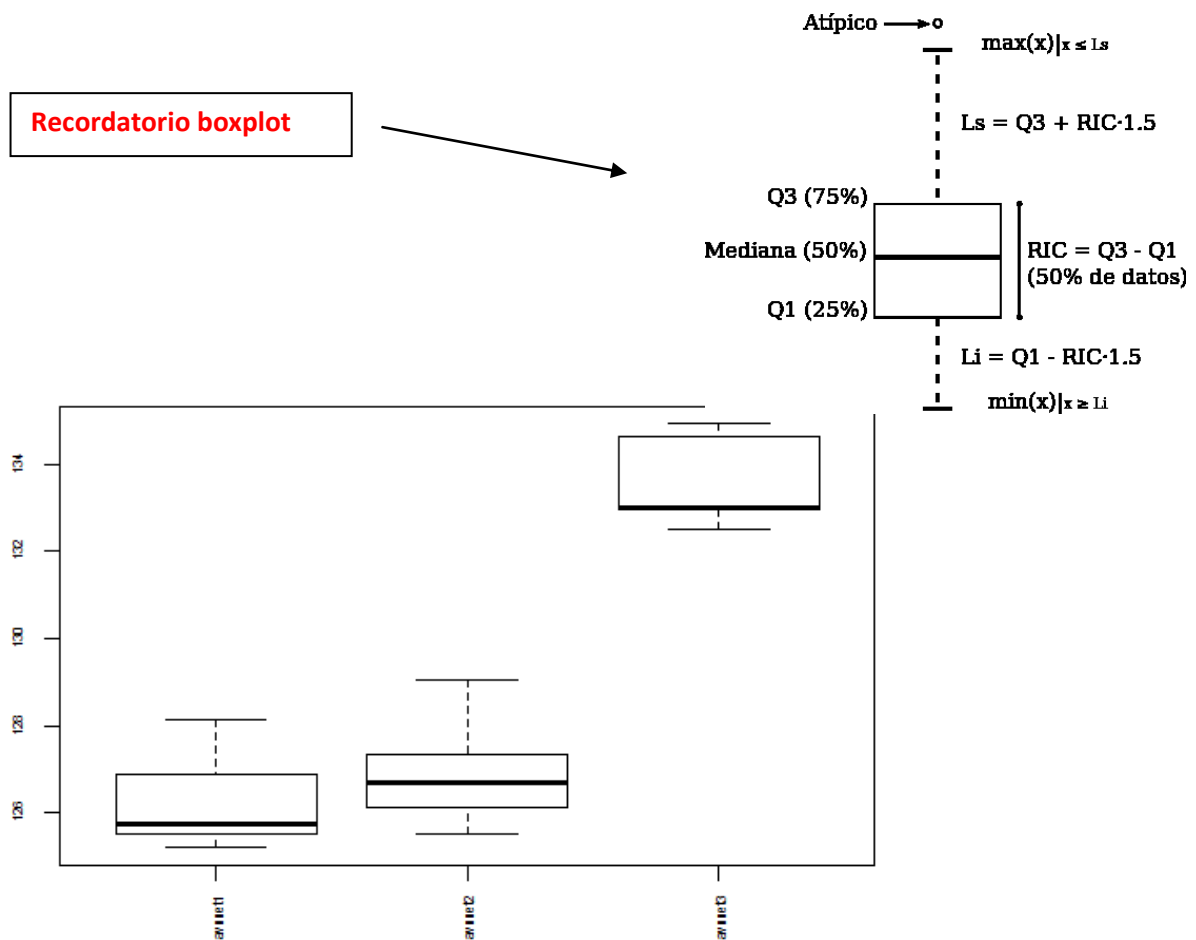
```
medias2$modelo="avnnnet2"
```

```
medias3<-cruzadaavnnnet(data=data,
vardep="cstrength",listconti=c("age","water"),
listclass=c(""),grupos=4,sinicio=1234,repe=5,
size=c(10),decay=c(0.01),repeticiones=5,itera=100)
medias3$modelo="avnnnet3"
```

```
union1<-rbind(medias1,medias2,medias3)
```

```
par(cex.axis=0.5)
boxplot(data=union1,error~modelo)
```

Recordatorio boxplot



El modelo 1 parece el mejor, menor sesgo y varianza que el 2. El 3 tiene un sesgo demasiado alto.

Ambas funciones `cruzadaavnnet` y `cruzadalin` realizan automáticamente el proceso de construcción de **dummies** y **estandarización de variables input continuas**. Podemos comparar modelos con diferentes variables.

```
# Ejemplo comparación diferentes modelos
# Las variables input no están estandarizadas
# pero esta operación se realiza internamente
# dentro de la función cruzada

load("compress.Rda")
data<-compress

medias1<-cruzadaavnnet(data=data,
vardep="cstrength",listconti=c("age","water"),
listclass=c(""),grupos=4,sinicio=1234,repe=5,
size=c(15),decay=c(0.01),repeticiones=5,itera=100)

medias1$modelo="avnnet1"

medias2<-cruzadaavnnet(data=data,
vardep="cstrength",listconti=c("age","water","cement"),
listclass=c(""),grupos=4,sinicio=1234,repe=5,
size=c(15),decay=c(0.01),repeticiones=5,itera=100)

medias2$modelo="avnnet2"

medias3<-cruzadaavnnet(data=data,
vardep="cstrength",listconti=c("age","water","cement","blast"),
listclass=c(""),grupos=4,sinicio=1234,repe=5,
size=c(15),decay=c(0.01),repeticiones=5,itera=100)

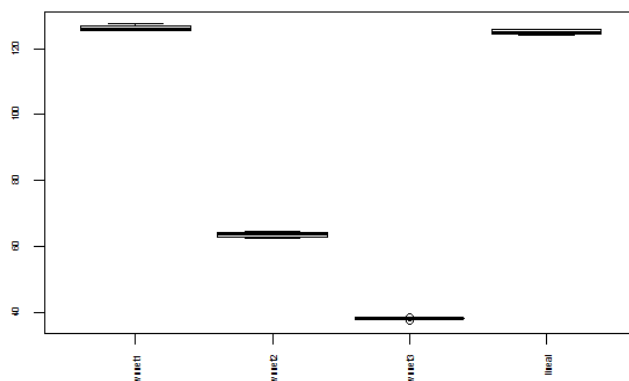
medias3$modelo="avnnet3"

medias4<-cruzadalin(data=data,
vardep="cstrength",listconti=c("age","water","cement","blast"),
listclass=c(""),grupos=4,sinicio=1234,repe=5)

medias4$modelo="lineal"

union1<-rbind(medias1,medias2,medias3,medias4)

par(cex.axis=0.5)
boxplot(data=union1,error~modelo)
```



Está claro que incluir las variables `blast` y `cement` mejora mucho el modelo.

Ejercicio 1.3

Con el modelo con las variables input

```
c("age", "water", "cement", "blast")
```

- 1) Tinear con caret bajo validación cruzada repetida, para determinar nodos y weight decay.
- 2) Con los diferentes modelos que parezcan interesantes, utilizar la función `cross_validate` para comparar gráficamente.

Comparar siempre con regresión.

Ejercicio 1.4

Utilizar el mismo esquema para elegir los mejores modelos con el archivo `autompg.Rda` (variables input: displacement horsepower weight origin modelyear)

Selección de variables en redes

Este problema es un problema no resuelto en modelización estadística y predictiva, y es un **punto flaco** de las redes neuronales.

Trataremos el tema de selección a nivel básico, sin entrar por falta de tiempo en lo que sería feature engineering, creación de nuevas variables como combinación o transformación de existentes, tratamiento de categóricas con alta cardinalidad, etc.

Taxonomía de métodos de selección de variables

Filters. Ordenan las variables por importancia (en términos de relación con la variable dependiente). En algunos de estos métodos su principal defecto es que no tienen en cuenta la información relativa que aporta cada variable cuando otras están en el modelo. En general no hay información sobre “donde cortar”, es decir, a partir de qué valor de importancia u orden de variable debemos considerar desechar variables. Ejemplos: lista ordenada por correlaciones, lista de importancia en árboles-gradient boosting, SVM, etc.

Wrappers. Métodos de búsqueda secuencial. Buscan en el espacio de todas las posibles combinaciones de variables. Al no ser exhaustivos (no evalúan todos los 2k modelos), pueden pasar por alto buenos modelos intermedios, o construir modelos que tienen al sobreajuste pero que en el proceso de búsqueda dan buenos resultados por existir relaciones matemáticas entre los datos que en realidad son derivadas del azar. Ejemplos: métodos stepwise (llamado también SFS “sequential forward selection”), forward, backward, etc. Son el método más potente y fiable, pero suele ser impracticable cuando hay demasiadas observaciones y variables (siempre dependiendo de la potencia del ordenador). Es frecuente encontrarse con que los modelos construidos en la práctica en concursos de predicción tipo Kaggle, donde el volumen y complejidad de datos es alta, se prefieren los filtros basados en árboles, por ejemplo, y no se usan los métodos wrappers. Pero a nivel de modelos de empresa con volumen moderado de datos, deberían ser preferibles los wrappers.

Embedded. Algunos algoritmos predictivos incorporan funciones de regularización o de deshecho automático de variables, reduciendo el parámetro asociado a las variables “malas”. En principio no necesitan de selección previa y tienen la ventaja de ser robustos frente a colinealidad. Su problema es que siguen siendo erráticos y sensibles al cambio (sensibles a la eliminación de observaciones o variables). Ejemplos: regresión lasso, modelos basados en

árboles, etc. En redes algo parecido a las técnicas de regularización es el early stopping y dropout.

Filters (ranking por importancia) en redes

Se dice que los pesos de las redes neuronales son “ininterrumpibles”, expresión que quiere explicar la interdependencia entre ellos, de modo que, contrariamente a modelos como el de regresión, donde cada parámetro va asociado a una variable o categoría, en las redes los pesos asociados a las variables toman rutas complejas, de donde no se puede fácilmente deducir la influencia o importancia de cada variable en el modelo de red neuronal.

El método más fiable aunque insuficiente, es el Método de Olden – Jackson (o Connection Weights). Mide la importancia de cada variable a partir de los productos de los pesos. En principio parece que es el que funciona mejor, sobre todo porque tiene en cuenta el signo y la posibilidad de que se cancelen los términos positivos y negativos.

$$RI_x = \sum_{y=1}^m w_{xy} w_{yz}$$

Se trata de un orden de importancia construido a posteriori, después de construir la red. Nos puede dar una idea general de la utilización o peso que tiene cada variable en las predicciones que da la red, pero la utilidad de este método es escasa. Puede servir como herramienta para análisis de sensibilidad, es decir, una vez trabajada y explorada mucho la red, presentar la importancia de variables como una medida descriptiva del funcionamiento de la red. Pero no debería usarse para seleccionar variables para la red.

El paquete NeuralNetTools de R calcula la medida de importancia de Olden-Jackson.

Ejemplo

Archivo [ejemplo importancia variables olden garson.R](#)

```
library(NeuralNetTools)
library(caret)
load("compressbien.Rda")

control<-trainControl(method = "none")

set.seed(12345)
nnetgrid <- expand.grid(size=c(15),decay=c(0.01))

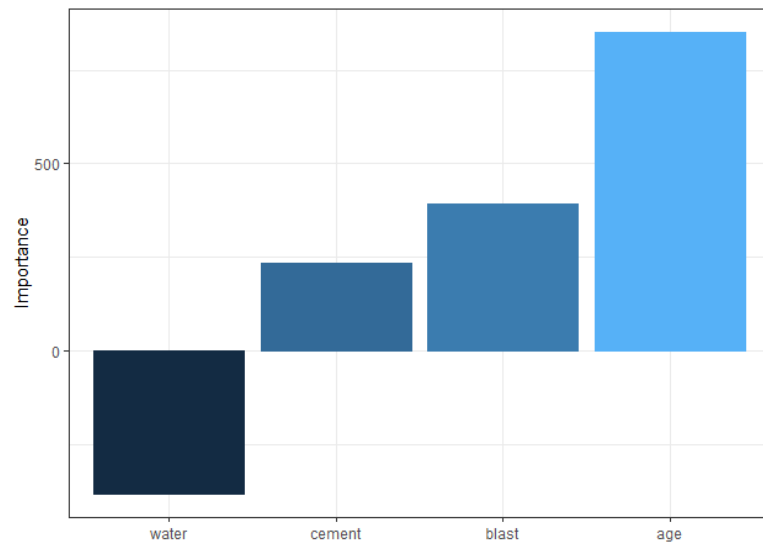
red1 <- train(cstrength~ age + water+cement+blast, method = 'nnet',
data = compressbien,tuneGrid=nnetgrid,linout = TRUE,trControl=control)

olden(red1)

importancia<-olden(red1,bar_plot=FALSE)
```

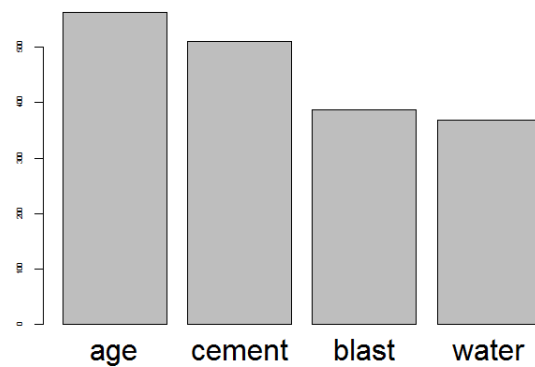
```
impor<-importancia[order(-importancia$importance),,drop = FALSE]
```

	importance
age	852.2432
blast	391.6422
cement	233.9316
water	-384.3720



```
# El signo aporta cierta información pero el orden de importancia
# hay que tomarlo en valor absoluto
```

```
impor$importance2<-abs(impor$importance)
barplot(impor$importance2,names=rownames(impor))
```



El orden de importancia planteado por esos métodos puede ayudar, pero a menudo el problema es complejo con muchas posibilidades y necesario utilizar varios métodos de manera alternativa/paralela.

En términos generales la selección de variables debe realizarse:

- previamente a la construcción de la red
- debe fijarse en este estudio un rango de conjuntos de variables que vaya desde el conjunto más conservador y robusto pero con menos capacidad predictiva, al conjunto que esté en el límite del sobreajuste o un poco por encima
- sobre esa lista de conjuntos se trabajará con la red, siempre teniendo en cuenta el estudio de la capacidad predictiva y sobreajuste, con datos de validación-test.

Algunas ideas para la preselección de variables

Selección sesgada por el planteamiento lineal

Métodos Stepwise, Backward y Forward en regresión

Selección no sesgada por el planteamiento lineal

Variable importance en árboles y gradient boosting

Agrupaciones de categorías

Árboles, otros métodos de agrupación

A estos métodos habría que añadir consideraciones inferenciales y descriptivas realizadas tras un estudio artesanal e imaginativo por parte del técnico.

Selección básica stepwise AIC y BIC con MASS

ejemplo selección variables.R

```
library(MASS)
load("compressbien.Rda")
dput(names(compressbien))

listconti<-c("cement", "blast", "ash", "water",
             "plasti", "agggreg", "fineagg", "age")
vardep<-c("cstrength")

# Del archivo, solo me quedo con las variables que me interesan,
# así es más facil poner ~. en el modelo

# Si alguna vez tenemos que pegar la formula del modelo a partir de
una lista de
# variables listconti , para no tener que eliminar comillas y escribir
se usa este truco:
#
# cosa<-paste(listconti,sep="",collapse='+')
# cat(cosa)

# Este ejemplo es para AIC, si se quiere BIC se calcula log(n) y se
pone k=log(n)
```

```
# en el stepAIC. Como truco para reducir las variables del modelo se
puede poner
# k grande
```

```
data<-compressbien[,c(listconti,vardep)]
full<-lm(cstrength~.,data=data)
null<-lm(cstrength~1,data=data)
selec1<-
stepAIC(null,scope=list(upper=full),direction="both",trace=FALSE)
summary(selec1)
```

Coefficients:

	Estimate	Std. Error	t value	Pr(> t)	
(Intercept)	35.8180	0.3243	110.434	< 2e-16	***
cement	11.0164	0.4438	24.825	< 2e-16	***
plasti	1.4356	0.5052	2.842	0.00458	**
age	7.1693	0.3416	20.988	< 2e-16	***
blast	7.4607	0.4291	17.385	< 2e-16	***
water	-4.6571	0.4512	-10.322	< 2e-16	***
ash	4.3941	0.4950	8.877	< 2e-16	***

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 10.41 on 1023 degrees of freedom
Multiple R-squared: 0.614, Adjusted R-squared: 0.6118
F-statistic: 271.2 on 6 and 1023 DF, p-value: < 2.2e-16

```
# Esto es para obtener la lista de variables que entran en el modelo y
pegarla en la función de validación cruzada. HAY QUE BORRAR LA
CONSTANTE (Intercept) DEL MODELO
```

```
dput(names(selec1$coefficients))
```

```
# Esto si se quiere en versión formula
formula(selec1)
```

Selección repetida con submuestras

Es interesante realizar el proceso repetidamente con submuestras de los datos (por ejemplo, un 80% de las observaciones elegido aleatoriamente), para :

- 1) Observar la estabilidad del modelo escogido con todos los datos
- 2) Buscar modelos alternativos

La función **steprepetido** realiza este proceso, se puede utilizar stepwise AIC y stepwise BIC por separado (AIC tiende a seleccionar más variables que BIC).

La tabla de salida muestra los modelos más frecuentes obtenidos tras realizar particiones con diferentes semillas y el proceso de selección de variables por separado en cada partición.

```
# La función steprepetido permite realizar el proceso training test
# varias veces obteniendo el modelo por stepwise sobre datos train
# y la tabla de frecuencias de los modelos escogidos.
#
# Previamente hay que pre procesar y meter en dummies las variables
# categóricas y meterlas en la lista listconti.
```

```
source("funcion steprepetido.R")
```

```
lista<-steprepetido(data=compressbien,vardep=c("cstrength"),
```

```
listconti=c("cement", "blast", "ash", "water",
"plasti", "aggreg", "fineagg", "age"),
sinicio=12345,sfinal=12355,porcen=0.8,criterio="AIC")
```

```
tabla<-lista[[1]]
```

```
dput(lista[[2]][[1]])
dput(lista[[2]][[2]])
```

En la siguiente tabla se observa que el modelo más frecuentemente elegido por stepwise AIC al realizar diferentes muestreos de 80% de los datos es el que tiene las variables cement,plasti,age,blast water y ash.

	resultados	Freq
1	cement+plasti+age+blast+water+ash	10
2	cement+plasti+age+blast+water+ash+aggreg+fineagg	1

```
c("cement", "plasti", "age", "blast", "water", "ash")
c("cement", "plasti", "age", "blast", "water", "ash", "aggreg",
"fineagg")
```

```
lista<-steprepetido(data=compressbien,vardep=c("cstrength"),
listconti=c("cement", "blast", "ash", "water",
"plasti", "aggreg", "fineagg", "age"),
sinicio=12345,sfinal=12355,porcen=0.8,criterio="BIC")
```

```
tabla<-lista[[1]]
```

```
dput(lista[[2]][[1]])
dput(lista[[2]][[2]])
```

	resultados	Freq
2	cement+plasti+age+blast+water+ash	6
1	cement+age+blast+water+ash	5

```
c("cement", "plasti", "age", "blast", "water", "ash")
c("cement", "plasti", "age", "blast", "water", "ash", "aggreg",
"fineagg")
```

En este segundo ejemplo se hará lo mismo con el archivo fitness.

```
load("fitnessbien.Rda")
```

```
# La función dput es muy importante para copiar y pegar
# modelos y listas de variables
```

```
dput(names(fitnessbien))
```

```
# En la consola quedan los nombres de las variables para copiar y
pegar
```

```
# c("AERO", "AGE", "HEART", "EXER", "TEACHER.Amund", "TEACHER.Czika",
# "TEACHER.Reed", "TEACHER.Yang", "SEX.F", "SEX.M")
```

```
listconti<-c("AGE", "HEART", "EXER", "SEX.F", "SEX.M")
vardep<-c("AERO")
```

```
lista<-steprepetido(data=fitnessbien,vardep=vardep,
listconti=listconti,sinicio=12345,sfinal=12385,porcen=0.8,criterio="AIC")
```

```
tabla<-lista[[1]]
```

```

      resultados Freq
6      SEX.F+EXER    11
3    SEX.F+HEART+AGE  10
4    SEX.F+EXER+AGE   6
1      SEX.F+HEART    4
5      EXER+SEX.F     3
7    SEX.M+HEART     2
2    EXER+SEX.F+AGE   1
8    SEX.F+AGE+HEART   1
9  SEX.F+EXER+AGE+HEART  1
10     SEX.M+EXER     1
11  SEX.F+HEART+AGE+EXER  1

```

```

lista<-steprepetido(data=fitnessbien,vardep=vardep,
listconti=listconti,sinicio=12345,sfinal=12385,porcen=0.8,criterio="BI
C")
tabla<-lista[[1]]

```

```

      resultados Freq
4      SEX.F+EXER   16
1      SEX.F+HEART   8
3  SEX.F+HEART+AGE   7
2      EXER+SEX.F    4
7  SEX.F+EXER+AGE    3
5    SEX.M+HEART     2
6    SEX.M+EXER      1

```

El objetivo de estas pruebas no es escoger un modelo, sino obtener modelos alternativos para probar. A partir de una lista de modelos tentativos, se pueden comparar vía validación cruzada repetida y boxplot.

```

# En este último ejemplo, hay dos modelos plausibles,
# uno muy simple. Podemos probar validación cruzada repetida para
# comparar vía sesgo-varianza

```

```

source("cruzadas avnnet y lin.R")

medias1<-cruzadalin(data=fitnessbien,
vardep=vardep,listconti=c("SEX.F","EXER"),
listclass=c(""),grupos=4,sinicio=1234,repe=30)

medias1$modelo="SEX.F,EXER"

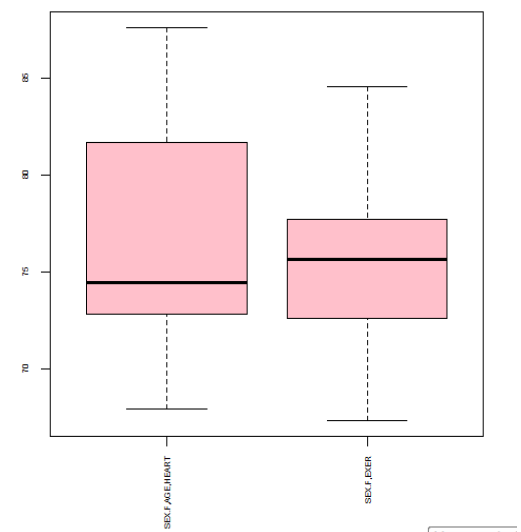
medias2<-cruzadalin(data=fitnessbien,
vardep=vardep,listconti=c("SEX.F","AGE","HEART"),
listclass=c(""),grupos=4,sinicio=1234,repe=30)

medias2$modelo="SEX.F,AGE,HEART"

union1<-rbind(medias1,medias2)

par(cex.axis=0.5)
boxplot(data=union1,col="pink",error~modelo)

```



Claramente es mejor el segundo modelo, con SEX.F, AGE, HEART.

Posteriormente se trabajarían los modelos más prometedores con redes. En este último ejemplo no merece la pena al ser claramente mejor la regresión.

Checklist básico y preparación anterior a la selección de variables

- 1) Borrar todas las variables sospechosas (no relación con la variable dependiente) o que seguro que no se van a utilizar
 - 2) Dar solución al problema de missing, si existen:
 - a) Borrar observaciones con demasiados missing
 - b) Borrar variables con demasiados missing
 - b) Imputar
 - 3) Tratamiento de variables categóricas:
 - a) Unir categorías previamente si es coherente
 - b) Pasar a dummy, borrando las dummy que correspondan a categorías poco representadas (<20 observaciones)
- Nota: las variables dummy 0-1 tienen la ventaja de poder ser nombradas como continuas en todos los modelos de regresión
- 4) Para redes, estandarizar las variables continuas.

Ejemplo Autompg revisitado

Archivo [Autompg revisitado.R](#)

- 1) Observar el archivo y verificar missing y categóricas (pasar a dummy modelyear)
- 2) Probar métodos de selección de variables
- 3) Comparar vía validación cruzada repetida –boxplot bajo regresión
- 4) Con el mejor o mejores modelos, estudiar los mejores parámetros (nodos, decay) en redes neuronales tuneando con caret
- 5) Comparación vía validación cruzada repetida –boxplot de los mejores modelos de redes y regresión

1) Observar el archivo y verificar missing y categóricas (pasar a dummy modelyear)

```
load("autompg.Rda")

dput(names(autompg))

# c("mpg", "cylinders", "displacement", "horsepower", "weight",
#   "acceleration", "modelyear", "origin", "carname")

listconti<-c("cylinders", "displacement", "horsepower", "weight",
"acceleration")
listclass<-c("modelyear")

vardep<-c("mpg")
```

```

# Primero me quedo con solo las variables de interés
# (listconti, listclass y vardep)

auto<-automp[g[,c(listconti,listclass,vardep)]]

# Borro observaciones con algún missing
# (son pocas, también se puede imputar)

auto<-na.omit(auto)

# Copio y pego de otros ejemplos estandarización continuas

means <-apply(auto[,listconti],2,mean,na.rm=TRUE)
sds<-sapply(auto[,listconti],sd,na.rm=TRUE)

auto2<-scale(auto[,listconti], center = means, scale = sds)
auto<-data.frame(cbind(auto2,auto[,c(listclass,vardep)]))

# Hago dummies

# Antes Observo si hay categorías poco representadas en modelyear
# para eliminarlas tras hacer dummies

table(auto$modelyear)

# Todo ok

library(dummies)
auto3<- dummy.data.frame(auto, listclass, sep = ".")

```

2) Probar métodos de selección de variables

```

# SELECCIÓN DE VARIABLES

# Utilizamos
# a) stepwise sobre todos los datos
# b) Con remuestreo (función steprepetido)

data<-auto3

full<-lm(mpg~.,data=data)
null<-lm(mpg~1,data=data)

selec1<-
stepAIC(null,scope=list(upper=full),direction="both",trace=FALSE)

summary(selec1)

formula(selec1)

# La función steprepetido permite realizar el proceso training test
# varias veces obteniendo el modelo por stepwise sobre datos train
# y la tabla de frecuencias de los modelos escogidos.
dput(names(auto3))

# c("cylinders", "displacement", "horsepower", "weight",
# "acceleration",
# "modelyear.70", "modelyear.71", "modelyear.72", "modelyear.73",

```

```
# "modelyear.74", "modelyear.75", "modelyear.76", "modelyear.77",
# "modelyear.78", "modelyear.79", "modelyear.80", "modelyear.81",
# "modelyear.82", "mpg")

source("funcion steprepetido.R")

lista<-steprepetido(data=data,vardep=c("mpg"),
listconti=c("cylinders", "displacement", "horsepower", "weight",
"acceleration",
"modelyear.70", "modelyear.71", "modelyear.72", "modelyear.73",
"modelyear.74", "modelyear.75", "modelyear.76", "modelyear.77",
"modelyear.78", "modelyear.79", "modelyear.80", "modelyear.81",
"modelyear.82"),
sinicio=12345,sfinal=12385,porcen=0.8,criterio="AIC")

tabla<-lista[[1]]
```

	modelo	Freq	contador
1	weight+modelyear.80+modelyear.82+modelyear.81+mo...	7	9
8	weight+modelyear.80+modelyear.82+modelyear.81+mo...	6	10
14	weight+modelyear.80+modelyear.82+modelyear.81+mo...	3	10
17	weight+modelyear.80+modelyear.82+modelyear.81+mo...	3	10
20	weight+modelyear.80+modelyear.82+modelyear.81+mo...	2	10
22	weight+modelyear.80+modelyear.82+modelyear.81+mo...	2	11
24	weight+modelyear.80+modelyear.82+modelyear.81+mo...	2	10
26	weight+modelyear.80+modelyear.82+modelyear.81+mo...	2	9
28	weight+modelyear.80+modelyear.82+modelyear.81+mo...	2	9
30	weight+modelyear.80+modelyear.82+modelyear.81+mo...	1	12
31	weight+modelyear.80+modelyear.82+modelyear.81+mo...	1	9

```

lista<-steprepetido(data=data,vardep=c("mpg"),
listconti=c("cylinders", "displacement", "horsepower", "weight",
"acceleration",
"modelyear.70", "modelyear.71", "modelyear.72", "modelyear.73",
"modelyear.74", "modelyear.75", "modelyear.76", "modelyear.77",
"modelyear.78", "modelyear.79", "modelyear.80", "modelyear.81",
"modelyear.82"),
sinicio=12345,sfinal=12385,porcen=0.8,criterio="BIC")

tabla<-lista[[1]]

```

	modelo	Freq	contador
1	weight+modelyear.80+modelyear.82+modelyear.81+mo...	9	7
10	weight+modelyear.80+modelyear.82+modelyear.81+mo...	8	8
18	weight+modelyear.80+modelyear.82+modelyear.81+mo...	7	8
25	weight+modelyear.80+modelyear.82+modelyear.81+mo...	4	8
29	weight+modelyear.80+modelyear.82+modelyear.81+mo...	3	7
32	weight+modelyear.80+modelyear.82+modelyear.81+mo...	2	7
34	weight+modelyear.80+modelyear.82+modelyear.81+mo...	2	8
36	weight+modelyear.80+modelyear.82+modelyear.81+mo...	1	6
37	weight+modelyear.80+modelyear.82+modelyear.81+mo...	1	6
38	weight+modelyear.80+modelyear.82+modelyear.81+mo...	1	7
39	weight+modelyear.80+modelyear.82+modelyear.81+mo...	1	7
40	weight+modelyear.80+modelyear.82+modelyear.81+mo...	1	9
41	weight+modelyear.80+modelyear.82+modelyear.81+mo...	1	9

```

# Obtenemos 3 modelos:

# El de stepAIC todos los datos

# c("(Intercept)", "weight", "modelyear.80", "modelyear.82",
"modelyear.81",
# "modelyear.79", "modelyear.73", "modelyear.78", "modelyear.77",
# "horsepower", "modelyear.76", "modelyear.74")

# El que sale mas veces de stepAIC repetido con AIC

# c("weight", "modelyear.80", "modelyear.82", "modelyear.81",
"modelyear.79",
# "horsepower", "modelyear.77", "modelyear.78", "modelyear.73")

# El que sale mas veces de stepAIC repetido con BIC

# c("weight", "modelyear.80", "modelyear.82", "modelyear.81",
"modelyear.79",
# "modelyear.77", "modelyear.78")

# Probamos también con modelyear continua
# para obtener más modelos .

# El archivo auto es antes de pasar modelyear a dummy

data<-auto

full<-lm(mpg~.,data=data)
null<-lm(mpg~1,data=data)

selec1<-
stepAIC(null,scope=list(upper=full),direction="both",trace=FALSE)

summary(selec1)

formula(selec1)
dput(names(selec1$coefficients))

# mpg ~ weight + modelyear

dput(names(auto))

c("cylinders", "displacement", "horsepower", "weight", "acceleration",
"modelyear", "mpg")

source("c:/funcion steprepetido bien.R")
lista<-steprepetido(data=data,vardep=c("mpg"),
listconti=
c("cylinders", "displacement", "horsepower", "weight",
"acceleration","modelyear"),
sinicio=12345,sfinal=12385,porcen=0.8,criterio="AIC")

lista<-steprepetido(data=data,vardep=c("mpg"),
listconti=
c("cylinders", "displacement", "horsepower", "weight", "acceleration",
"modelyear"),
sinicio=12345,sfinal=12385,porcen=0.8,criterio="BIC")

# mpg ~ weight + modelyear

```

3) Comparar vía validación cruzada repetida –boxplot bajo regresión

```
# COMPARAMOS TODOS LOS MODELOS CON VALIDACIÓN CRUZADA REPETIDA
# PARA LOS DE MODELYEAR CATEGORICA USAMOS
# EL ARCHIVO auto3
# PARA LOS DE MODELYEAR CATEGORICA USAMOS
# EL ARCHIVO auto
# COMO SON LAS MISMAS OBSERVACIONES NO HAY PROBLEMA
source("cruzadas avnnet y lin.R")
medias1<-cruzadalin(data=auto3,
vardep="mpg",listconti=c("horsepower", "weight",
"modelyear.74","modelyear.76", "modelyear.77",
"modelyear.78", "modelyear.79", "modelyear.80", "modelyear.81",
"modelyear.82"),
listclass=c(""),grupos=4,sinicio=1234,repe=30)
medias1$modelo=1

medias2<-cruzadalin(data=auto3,
vardep="mpg",listconti=c("horsepower", "weight",
"modelyear.73","modelyear.77","modelyear.78",
"modelyear.79", "modelyear.80",
"modelyear.81", "modelyear.82"),
listclass=c(""),grupos=4,sinicio=1234,repe=30)

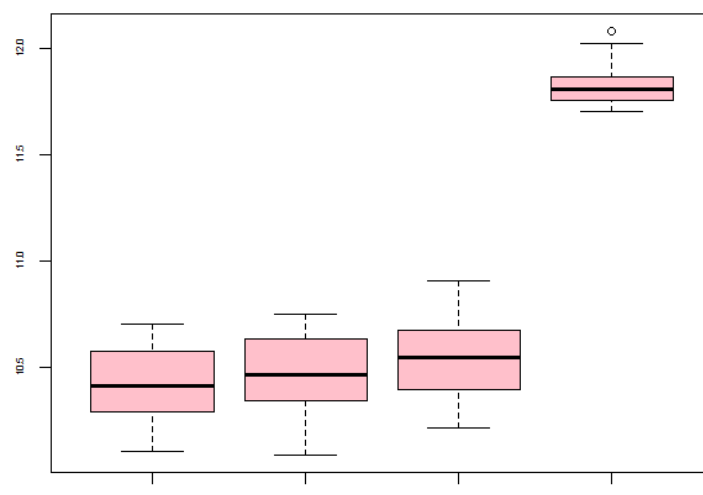
medias2$modelo=2

medias3<-cruzadalin(data=auto3,
vardep="mpg",listconti=c("weight",
"modelyear.73","modelyear.77","modelyear.78",
"modelyear.79", "modelyear.80",
"modelyear.81", "modelyear.82"),
listclass=c(""),grupos=4,sinicio=1234,repe=30)
medias3$modelo=3

medias4<-cruzadalin(data=auto,
vardep="mpg",listconti=c("weight", "modelyear"),
listclass=c(""),grupos=4,sinicio=1234,repe=30)

medias4$modelo=4

union1<-rbind(medias1,medias2,medias3,medias4)
par(cex.axis=0.5)
boxplot(data=union1,col="pink",error~modelo)
```



4) Con el mejor o mejores modelos, estudiar los mejores parámetros (nodos, decay) en redes neuronales tuneando con caret

```
# TOMAMOS EL MODELO 1 COMO REFERENCIA Y HACEMOS REDES
# EL PROBLEMA ES EL NÚMERO DE VARIABLES (10)

# PRIMERO TUNEAMOS CON CARET PARA VER EL NÚMERO DE NODOS
# LO MEJOR: UTILIZAR AVNNET Y CONTROL CON CV REPETIDA

library(caret)

control<-trainControl(method = "repeatedcv",
  number=4, repeats=5, savePredictions = "all")

set.seed(123)
nnetgrid <- expand.grid(size=c(5,8,10,12,15,20),
  decay=c(0.01,0.1), bag=F)

rednnet<- train(mpg ~ weight + modelyear.80 + modelyear.82 +
  modelyear.81 + modelyear.79 +
    modelyear.73 + modelyear.78 + modelyear.77 + horsepower +
    modelyear.76 + modelyear.74,
  data=auto3,
  method="avNNet", linout = TRUE, maxit=100,
  trControl=control, repeats=5, tuneGrid=nnetgrid)

rednnet
```

size	decay	RMSE	Rsquared	MAE
5	0.01	2.875499	0.8654667	2.112741
5	0.10	2.815127	0.8709468	2.068317
8	0.01	2.990146	0.8549291	2.180233
8	0.10	2.952384	0.8585912	2.158686
10	0.01	3.119445	0.8423496	2.269757
10	0.10	3.022877	0.8518039	2.204694
12	0.01	3.202938	0.8343574	2.324962
12	0.10	3.043459	0.8499974	2.205230
15	0.01	3.305497	0.8249391	2.378247
15	0.10	3.114548	0.8430909	2.248867
20	0.01	3.335485	0.8216830	2.435328
20	0.10	3.140918	0.8402888	2.284172

```
# PROBAMOS TAMBIÉN LA RED CON EL MODELO 4 MAS SENCILLO
# A LAS REDES LES GUSTAN LAS VARIABLES CONTINUAS
```

```
library(caret)
```

```
control<-trainControl(method = "repeatedcv",
  number=4, repeats=5, savePredictions = "all")
```

```
set.seed(123)
```

```
nnetgrid <- expand.grid(size=c(5,8,10,12,15,20),
  decay=c(0.01,0.1), bag=F)
```

```
rednnet<- train(mpg ~ weight+modelyear,
  data=auto,
  method="avNNet", linout = TRUE, maxit=100,
  trControl=control, repeats=5, tuneGrid=nnetgrid)
```

```
rednnet
```

size	decay	RMSE	Rsquared	MAE
5	0.01	3.027767	0.8583901	2.184601
5	0.10	2.956458	0.8588198	2.110556
8	0.01	3.056349	0.8582153	2.223310
8	0.10	2.954725	0.8588667	2.115822
10	0.01	2.953040	0.8589259	2.111096
10	0.10	2.952354	0.8592508	2.110439
12	0.01	2.978157	0.8604041	2.133787
12	0.10	2.955410	0.8588843	2.117016
15	0.01	2.982156	0.8593353	2.140748
15	0.10	2.956010	0.8587224	2.111372
20	0.01	2.959957	0.8602834	2.140502
20	0.10	2.951049	0.8593312	2.112285

Aquí cogería 5 nodos y decay=0.1, no parece justificado 20 nodos por la escasa diferencia en RMSE

5) Comparación vía validación cruzada repetida –boxplot de los mejores modelos de redes y regresión

```
# FINALMENTE COMPARAMOS LOS DOS MODELOS DE REDES
# CON LOS DE REGRESIÓN ANTERIORES CON CV REPETIDA
```

```
medias5<-cruzadaavnnnet(data=auto3,
  vardep="mpg", listconti=c("horsepower", "weight",
  "modelyear.74", "modelyear.76", "modelyear.77",
  "modelyear.78", "modelyear.79", "modelyear.80", "modelyear.81",
  "modelyear.82"),
  listclass=c(""), grupos=4, inicio=1234, repe=30,
  size=c(5), decay=c(0.1), repeticiones=5, itera=100)
```

```
medias5$modelo=5
```

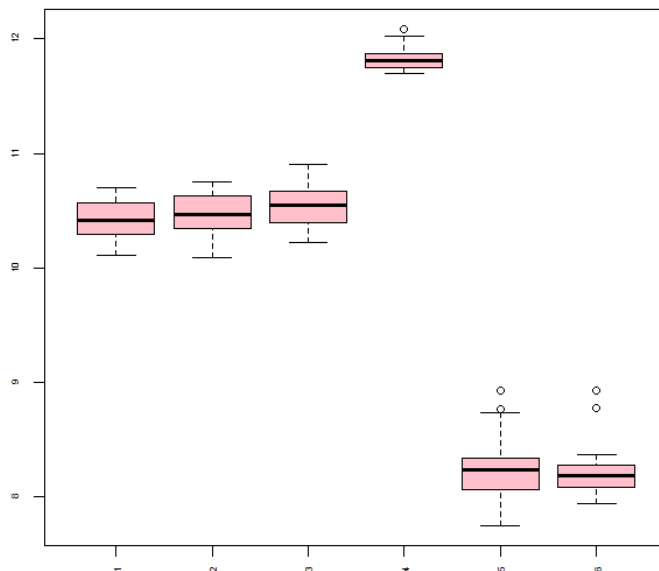
```
medias6<-cruzadaavnnnet(data=auto,
  vardep="mpg", listconti=c("weight", "modelyear"),
  listclass=c(""), grupos=4, inicio=1234, repe=30,
  size=c(5), decay=c(0.1), repeticiones=5, itera=100)
```

```
medias6$modelo=6
```

```
union1<-rbind(medias1,medias2,medias3,medias4,medias5,medias6)
```

```
par(cex.axis=0.5)
```

```
boxplot(data=union1, col="pink", error~modelo)
```

Modelo 1: 11 parámetros
 Modelo 2: 9 parámetros
 Modelo 3: 9 parámetros
 Modelo 4: 3 parámetros
 Modelo 5: 61 parámetros
 Modelo 6: 21 parámetros

Diagnóstico:

Las redes parecen funcionar mejor.

De los dos modelos de redes es preferible el modelo 6, es mucho más sencillo que el 5 (2 variables continuas input, 21 parámetros frente a 61). Curiosamente aunque a priori utilizar modelyear como discreta parecía funcionar mejor bajo regresión, en el caso de la red es un modelo claramente preferible.

¿Qué hacer desde el punto de vista práctico en este ejemplo?

Los gráficos son una manera rápida de aproximarse al problema, pero la escala nos puede engañar: una escala más pequeña o grande en el eje y alteraría nuestra percepción del problema.

Hay que tener en cuenta los valores numéricos y otras consideraciones:

1) El archivo autmpg tiene 398 observaciones. 21 parámetros en la mejor red equivale a 19 observaciones por parámetro.

2) ¿Se gana mucho con la bajada de error respecto al modelo 4, el modelo más sencillo de regresión con solo 3 parámetros?

a) El error cuadrático medio en la regresión es aproximadamente de 11.8 en este caso, por 8.2 en la red. Tomando la raíz cuadrada del error, para ponerlo en la escala de la variable dependiente (millas por galón), sería 3.4 frente a 2.8. La media de la variable mpg (millas por galón) es de 23.51.

b) El investigador-decisor tiene que decidir si le interesa un modelo más sencillo y explicativo como el de regresión, que comete un error promedio de ± 3.4 millas por galón en la predicción, frente a un modelo más complicado pero en principio más preciso, con un error de ± 2.8 .

Ejercicios

1) Un ejemplo más complicado: Ameshousing. Revisar el código.

(Ameshousing 5.0.R)

2) Ozono. Variable dependiente ozone. Realizar todo el proceso. Ejercicio no corregido.

(Ozone.Rda)

- a) Observar el archivo y verificar missing y categóricas
- b) Probar métodos de selección de variables
- c) Comparar vía validación cruzada repetida –boxplot bajo regresión
- d) Con el mejor o mejores modelos, estudiar los mejores parámetros (nodos, decay) en redes neuronales tuneando con caret
- e) Comparación vía validación cruzada repetida –boxplot de los mejores modelos de redes y regresión

3) Ejemplo con California Housing. Ejercicio no corregido.

(Variable dependiente Median_House_Value)

(californiahousing mapas.R)

No es necesario en este caso hacer selección de variables, usarlas todas.

- a) Regresión básica, histograma de todas las variables y alguna nube de puntos.
- b) Tunear la red con caret bajo diferentes esquemas página 91 y 94
- c) Observar tiempo de proceso y diferencia en los resultados en apartado b)
- d) CV repetida: 1 única repetición, y varias. Ver tiempos de proceso.

Redes Neuronales para Clasificación binaria

El problema de la clasificación supervisada

Dado un conjunto de variables independientes X_1, \dots, X_k , plantear un modelo predictivo para la variable dependiente Y , categórica, con valores codificados 0,1,2,...

Es uno de los problemas más habituales en Business Intelligence.

Hay muchos métodos. Algunos son:

- Regresión logística
- Redes Neuronales
- Árboles de regresión, gradient boosting, etc.
- Análisis discriminante
- K-NN

Etc.

Por cuestiones de tiempo se estudiará solamente el caso de clasificación binaria, en el que Y tiene dos categorías, 0 y 1. El valor 1 suele representar el evento de interés (personas que consumen un producto, que cometen fraude, que aceptan un seguro o un crédito, etc.).

Funcionamiento general de los métodos de clasificación binaria

1) Se define una función objetivo para seleccionar los parámetros que la optimicen

Regresión logística: máxima verosimilitud del modelo suponiendo la función logit, o bien logit generalizada (variable dependiente con más de 2 categorías), o bien logit acumulativa (variable dependiente ordinal), o bien probit.

Red Neuronal: máxima verosimilitud suponiendo la función Binomial por defecto, o bien entropía, o función de coste .

2) El algoritmo optimiza los parámetros y se obtienen predicciones

Éstas suelen ser **en forma de probabilidad** de pertenecer a cada clase.

3) Si las predicciones son en forma de probabilidad, a menudo es necesario determinar cual es el criterio o punto de corte (threshold) para asignar cada observación a una clase.

Por defecto, el punto de corte es $p=0.5$ (a partir de una probabilidad predicha de pertenecer a la clase A de 0.5 se asigna la observación a la clase A).

4) Una vez asignadas las observaciones a las clases, se obtienen medidas de eficacia en la clasificación como la matriz de confusión, área bajo la curva ROC (AUC), etc.

En general, para evitar el sobreajuste, las medidas de eficacia se calcularán siempre sobre datos de **validación** o **test**, al igual que se observaba en regresión el MSE para validación o test.

Matriz de Confusión de un método de clasificación y medidas asociadas

La matriz de confusión se obtiene tras haber aplicado el punto de corte a las probabilidades predichas para asignar un valor concreto de clase a cada observación. A partir de la matriz de confusión se pueden construir medidas de diagnóstico.

		Observados	
Predichos		0	1
	0	25	8
	1	6	15

VP=Verdaderos positivos: 15: 27.7% (sobre el total)

VN=Verdaderos negativos: 25: 46.3%

FP=Falsos positivos: 6: 11.1%

FN=Falsos negativos: 8: 14.8%

Sensitividad=Probabilidad de que la predicción sea 1, dado que la observación es realmente 1
 $=VP/(VP+FN)=0.65$ (es una medida de *detección* de unos). Se llama también "Recall".

Especificidad=Probabilidad de que la predicción sea 0, dado que la observación es realmente 0
 $=VN/(VN+FP)=0.80$ (es una medida de *detección* de ceros)

Tasa de aciertos= $(VP+VN)/N=0.747$

Tasa de fallos= $1-(VP+VN)/N=0.253$

Tasa de verdaderos positivos= $VP/(VP+FP)=0.71$ (cuando digo positivo, que proporción acierto).
 Se llama también "Precision".

Tasa de verdaderos negativos= $VN/(VN+FN)=0.71$ (cuando digo negativo, que proporción acierto)=0.757

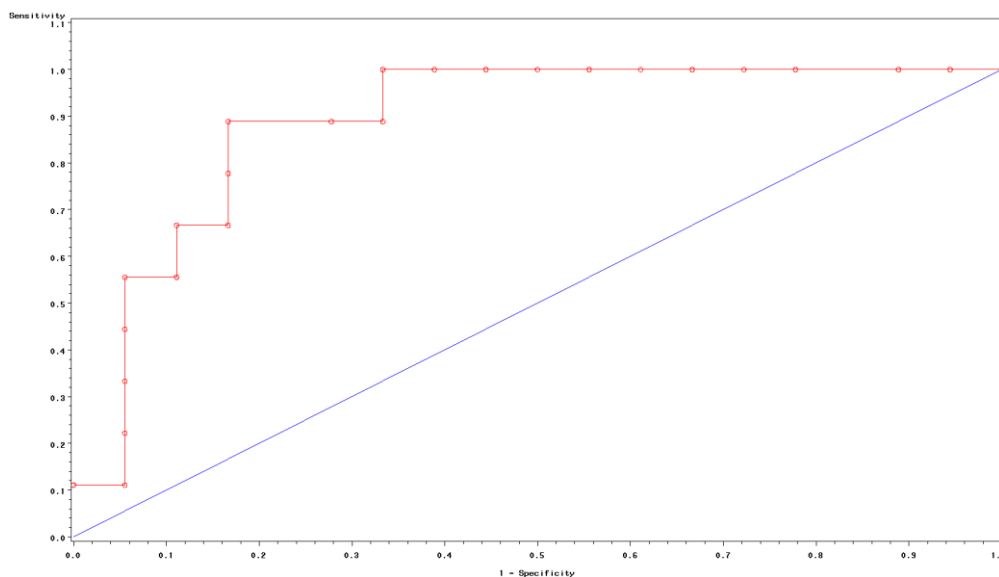
CURVA ROC (Receiver Operating Characteristic Curve)

En general, cada punto corresponde a un punto de corte de probabilidad. En el eje x está 1-Especificidad, en el eje y la sensibilidad.

La clasificación trivial es la recta diagonal. Si un método clasifica bien, la curva tocará cerca de la esquina superior izquierda. Los puntos de corte más cercanos a esa esquina son los mejores.

En realidad, cada punto de la curva ROC corresponde a una observación, que a su vez corresponde a un punto de corte de probabilidad.

En general, para evitar el sobreajuste, la tabla de clasificación y curva ROC se calcularán siempre sobre datos de **validación** o **test**, al igual que se observaba en regresión el ASE para validación o test.



El Área bajo la curva ROC es uno de los estadísticos más utilizados para comparar modelos de clasificación binaria. En el ejemplo anterior, es $AUC=0.889$. Cuanto más cerca del valor 1 (área total del cuadrado) esté el AUC, mejor.

No es el objeto de este módulo incidir en las sutilezas de las medidas de diagnóstico en clasificación binaria pero son necesarias algunas consideraciones.

1) Por simplicidad, se trabajará con dos medidas de diagnóstico simples:

a) La tasa de fallos, o 1-accuracy, que necesita un punto de corte en las probabilidades predichas que tomaremos por defecto como 0.5 (en R existe el paquete `OptimalCutpoints` si se desea jugar con el punto de corte). La ventaja de la tasa de fallos es que es intuitiva y está entre 0 y 1.

b) El área bajo la curva ROC, que llamaremos AUC. Su ventaja es que no depende del punto de corte y permite entre modelos con un criterio global. Su desventaja es que no es intuitivo.

2) Desde el punto de vista práctico, si la tasa de fallos no parece una medida adecuada por muestras excesivamente desequilibradas o que el algoritmo no parece obtener una tasa de fallos menor al porcentaje de unos, o que preferiríamos otro punto de corte, en esos casos sería mejor usar el AUC como medida para elegir entre modelos.

3) Si hay discrepancia entre el criterio tasa de fallos y AUC (el modelo A es mejor que el B en tasa de fallos pero peor que el B en AUC), se elegirá AUC como medida.

4) Una cuestión importante no tratada aquí es que cualquier modelo es bueno, siempre que las variables independientes aporten información sobre la y . El resultado básico que a nivel de práctico es suficiente en la mayor parte de los casos, es una lista ordenada de las observaciones predichas ordenadas de mayor a menor probabilidad predicha de uno. A partir de ahí el investigador puede tomar decisiones sin necesidad de nada más que esa lista.

Planteamiento de la red neuronal con variable dependiente binaria

1) Cuando la variable dependiente es categórica con k categorías, la red se plantea simplemente poniendo $k-1$ nodos output (cada uno correspondiente a una categoría). Esto lo hace el paquete-programa automáticamente, no es necesario crear dummies en la variable output.

Si la variable output y es binaria (dos categorías), normalmente daremos valor 1 a la categoría de interés (suele ser la menos numerosa) y automáticamente la red modeliza la probabilidad de $y=1$.

2) La red se plantea técnicamente añadiendo **una función de activación de la capa oculta al nodo output**, para que el valor resultante (probabilidad estimada de 1) tome valores en el intervalo $(0,1)$.

Ejemplo básico saheart

Ejemplo saheart.R

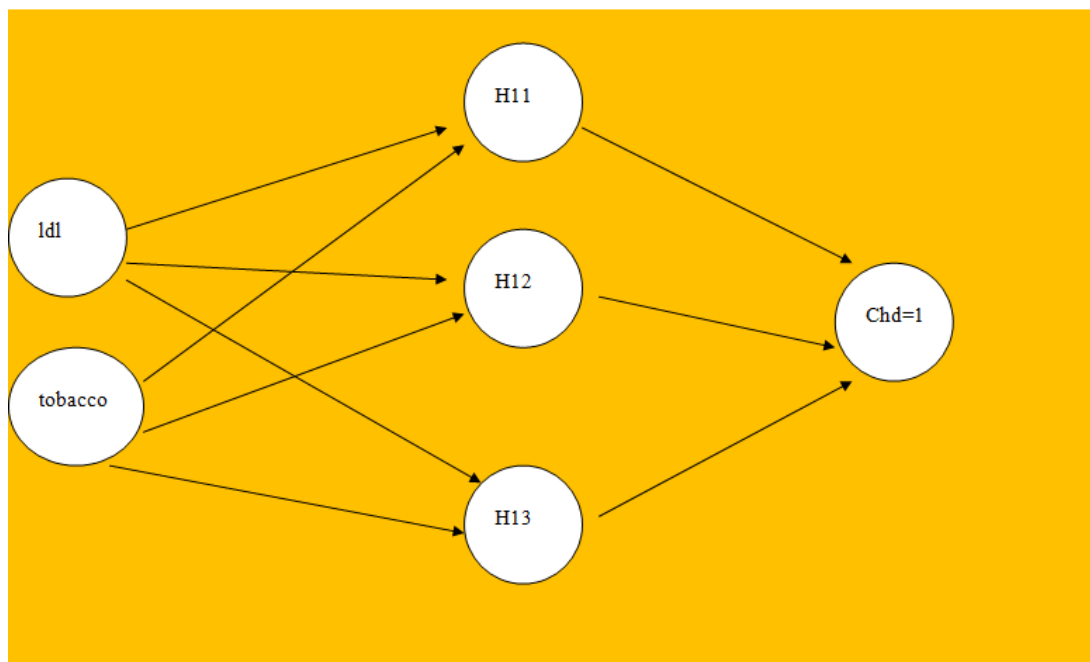
chd variable dependiente



	chd	sbp	tobacco	ldl	adiposity	famhist	typea	obesity	alcohol	age
1	1	160	12.00	5.73	23.11	Present	49	25.30	97.20	52
2	1	144	0.01	4.41	28.61	Absent	55	28.87	2.06	63
3	0	118	0.08	3.48	32.28	Present	52	29.14	3.81	46
4	1	170	7.50	6.41	38.03	Present	51	31.99	24.26	58
5	1	134	13.60	3.50	27.78	Present	60	25.99	57.34	49
6	0	132	6.20	6.47	36.21	Present	62	30.77	14.14	45
7	0	142	4.05	3.38	16.20	Absent	59	20.81	2.62	38

La variable dependiente chd es categórica, con $k=2$ categorías ($chd=1, chd=0$).

El nodo output se construye con $k-1=1$ nodo con valor 1 o 0



```

library(nnet)
library(dummies)
library(MASS)
library(reshape)
library(caret)
library(pROC)

# Lectura y esquema de variables

load("saheart.Rda")
dput(names(saheart))

# c("sbp", "tobacco", "ldl", "adiposity", "famhist", "typea",
# "obesity",
# "alcohol", "age", "chd")

continuas<-c("sbp", "tobacco", "ldl", "adiposity", "obesity",
"alcohol", "age", "typea")
categoricas<-c("famhist")
vardep<-c("chd")

# a) Eliminar las observaciones con missing en alguna variable
saheart2<-na.omit(saheart, (!is.na(saheart)))

# b) pasar las categóricas a dummies
saheart3<- dummy.data.frame(saheart2, categoricas, sep = ".")

# c) estandarizar las variables continuas

# Calculo medias y dtípica de datos y estandarizo (solo las continuas)

```

```

means <-apply(saheart3[,continuas],2,mean)
sds<-sapply(saheart3[,continuas],sd)

# Estandarizo solo las continuas y uno con las categoricas

saheartbis<-scale(saheart3[,continuas], center = means, scale = sds)
numerocont<-which(colnames(saheart3)%in%continuas)
saheartbis<-cbind(saheartbis,saheart3[,-numerocont])

# El archivo saheartbis ya está preparado:no hay missing, las
continuas salvo la dependiente
# están estandarizadas y las categoricas pasadas a dummy

dput(names(saheartbis))

# NOTA: En los modelos pondremos solo k-1 dummies por cada categórica

c("sbp", "tobacco", "ldl", "adiposity", "obesity", "alcohol",
"age", "typea", "famhist.Absent", "famhist.Present", "chd")

# PARA EVITAR PROBLEMAS, MEJOR DEFINIR LA VARIABLE OUTPUT
# con valores alfanuméricos Yes, No

saheartbis$chd<-ifelse(saheartbis$chd==1, "Yes", "No")

# EJEMPLO BÁSICO CON CARET TRAIN TEST


# CON LOGÍSTICA

# Training test una sola vez
control<-trainControl(method = "LGOCV",p=0.8,number=1,
  classProbs=TRUE,savePredictions = "all")

logi<- train(chd~ldl+tobacco+famhist.Absent,data=saheartbis,
method="glm",trControl=control)

summary(logi)
logi
sal<-logi$pred

```



	pred	obs	No	Yes	rowIndex	parameter	Resample
1	No	Yes	0.5241112	0.47588877	8	none	Resample1
2	No	No	0.6659234	0.33407662	9	none	Resample1
3	Yes	Yes	0.2792245	0.72077553	12	none	Resample1
4	Yes	Yes	0.2060469	0.79395309	18	none	Resample1
5	Yes	Yes	0.4179224	0.58207758	19	none	Resample1
6	Yes	Yes	0.1830983	0.81690174	26	none	Resample1
7	Yes	No	0.4915159	0.50848406	27	none	Resample1
8	Yes	No	0.4748223	0.52517769	29	none	Resample1


```
# CON RED

nnetgrid <- expand.grid(size=c(5),decay=c(0.1))

red1<- train(chd~ldl+tobacco+famhist.Absent,data=saheartbis,
method="avNNet",linout =
FALSE,maxit=100,repats=5,trControl=control,tuneGrid=nnetgrid)

summary(red1)
red1
sal<-red1$pred
```

Calculo de la matriz de confusión

La función confusionMatrix de caret calcula la matriz de confusión

```
salconfu<-confusionMatrix(sal$pred,sal$obs)
salconfu
```

Confusion Matrix and Statistics

```

              Reference
Prediction No Yes
No      52  19
Yes      8  13

      Accuracy : 0.7065
      95% CI   : (0.6024, 0.7969)
No Information Rate : 0.6522
P-Value [Acc > NIR] : 0.16236

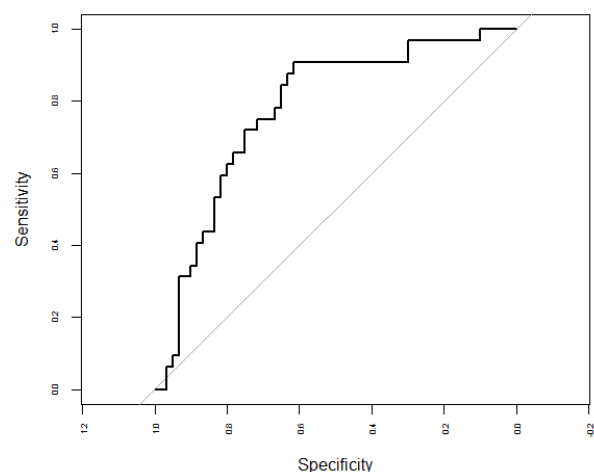
      Kappa : 0.2967
McNemar's Test P-Value : 0.05429

      Sensitivity : 0.8667
      Specificity : 0.4062
      Pos Pred Value : 0.7324
      Neg Pred Value : 0.6190
      Prevalence : 0.6522
      Detection Rate : 0.5652
      Detection Prevalence : 0.7717
      Balanced Accuracy : 0.6365
```

Para dibujar la curva roc y calcular el auc se usa el paquete pROC

```
curvaroc<-roc(response=sal$obs,predictor=sal$Yes)
auc<-curvaroc$auc
plot(roc(response=sal$obs,predictor=sal$Yes))
```

Area under the curve: 0.775



Algunos detalles en la utilización de R para modelización predictiva de variables binarias

- a) poner `linout=FALSE` en los modelos de red u otros
- b) usar `factor` en la variable dependiente por si acaso, mejor cambiando a `character`, "Yes", "No"
- c) En general en `trainControl` del paquete `caret` hay que poner `classProbs=TRUE`, y `savePredictions="all"`
- d) Si se usa `predict` para predecir nuevas observaciones la predicción puede venir en dos formas:
 - Si se pone `type="prob"` da como resultado dos variables 0 y 1 (o bien "Yes", "No"), con las probabilidades predichas respectivas.
 - Si se pone `type="class"` da como resultado 0 y 1 utilizando 0.5 como punto de corte
- e) Otras veces en lugar de `type="prob"` se pone `type="response"` o `type="raw"`, depende del paquete/algoritmo.
- f) Para obtener medidas se puede cruzar el vector de predicciones con el real con la función `confusionMatrix` de `caret` y con la función `roc` del paquete `pROC`

Tuneado de la red, con criterio Accuracy (tasa de aciertos)

```
# Validación cruzada repetida
control<-trainControl(method = "repeatedcv", number=4, repeats=5,
  savePredictions = "all", classProbs=TRUE)

# *****
# avNNet: parámetros
#   # Number of Hidden Units (size, numeric)
#   # Weight Decay (decay, numeric)
#   # Bagging (bag, logical)
# *****
avnnetgrid <- expand.grid(size=c(5,10,15,20),
  decay=c(0.01,0.1,0.001), bag=FALSE)

redavnnnet<- train(chd~ldl+tobacco+famhist.Absent, data=saheartbis,
  method="avNNet", linout = FALSE, maxit=100,
  trControl=control, tuneGrid=avnnetgrid,
  repeats=5)

redavnnnet
```

No pre-processing

Resampling: Cross-validated (4 fold, repeated 5 times)
 Summary of sample sizes: 347, 346, 347, 346, 347, 346, ...
 Resampling results across tuning parameters:

size	decay	Accuracy	Kappa
5	0.001	0.6987069	0.2889402
5	0.010	0.7047826	0.3027912
5	0.100	0.7182271	0.3293770
10	0.001	0.6757984	0.2389904
10	0.010	0.6930585	0.2833069
10	0.100	0.7156372	0.3225488
15	0.001	0.6675825	0.2253850
15	0.010	0.6865705	0.2678660

```

15    0.100  0.7147639  0.3200235
20    0.001  0.6567391  0.2045194
20    0.010  0.6870015  0.2723317
20    0.100  0.7143291  0.3192151

```

Tuning parameter 'bag' was held constant at a value of FALSE
 Accuracy was used to select the optimal model using the largest value.
 The final values used for the model were size = 5, decay = 0.1 and bag
 = FALSE.

Selección de variables en clasificación binaria bajo stepAIC en regresión logística

```
# SELECCIÓN DE VARIABLES EN CLASIFICACIÓN BINARIA LOGÍSTICA
```

```
full<-glm(factor(chd)~.,data=saheartbis,family=binomial(link="logit"))
null<-glm(factor(chd)~1,data=saheartbis,family=binomial(link="logit"))
```

```
library(MASS)
```

```
seleccion<-stepAIC(null,scope=list(upper=full),direction="both")
```

```
# Para ver los efectos escogidos
dput(names(seleccion$coefficients))
```

```
# Esto si se quiere en versión formula
formula(seleccion)
```

```
dput(names(saheartbis))
```


```
# *****
# APLICANDO steprepetidobinaria
# *****
source("funcion steprepetido binaria.R")
```

```
listconti<-c("sbp", "tobacco", "ldl", "adiposity",
"obesity", "alcohol", "age", "typea",
"famhist.Absent", "famhist.Present")
vardep<-c("chd")
```

```
data<-saheartbis
```

```
lista<-steprepetidobinaria(data=data,
vardep=vardep,listconti=listconti,sinicio=12345,
sfinal=12355,porcen=0.8,criterio="AIC")
```

```
tabla<-lista[[1]]
```




	modelo	Freq	contador
1	age+famhist.Absent+typea+ldl+tobacco	6	5
7	age+famhist.Absent+typea+ldl+tobacco+obesity+sbp	2	7
9	age+famhist.Absent+typea+ldl+tobacco+obesity	1	6
10	age+famhist.Absent+typea+ldl+tobacco+sbp	1	6
11	age+typea+ldl+tobacco+famhist.Present	1	5

```
dput(lista[[2]][[1]])
dput(lista[[2]][[2]])
```

```
lista<-steprepetidobinaria(data=data,
  vardep=vardep,listconti=listconti,sinicio=12345,
  sfinal=12355,porcen=0.8,criterio="BIC")
```

```
tabla<-lista[[1]]
```



	modelo	Freq	contador
1	age+famhist.Absent+typea+ldl+tobacco	6	5
7	age+famhist.Absent+ldl	1	3
8	age+famhist.Absent+ldl+tobacco	1	4
9	age+famhist.Absent+tobacco	1	3
10	age+famhist.Absent+typea+tobacco	1	4
11	age+ldl+tobacco+famhist.Present	1	4

```
dput(lista[[2]][[1]])
dput(lista[[2]][[2]])
```

Validación cruzada repetida

Se aplicará el método gráfico usando resultados de validación cruzada repetida con la función

`cruzadas avnnet y log binaria.R`

```
# *****
# APLICANDO cruzadalogistica a los modelos candidatos
# *****
source("cruzadas avnnet y log binaria.R")

medias1<-cruzadalogistica(data=saheartbis,
  vardep="chd",listconti=c("age", "famhist.Absent",
    "typea", "ldl", "tobacco"),
  listclass=c(""), grupos=4,sinicio=1234,repe=5)

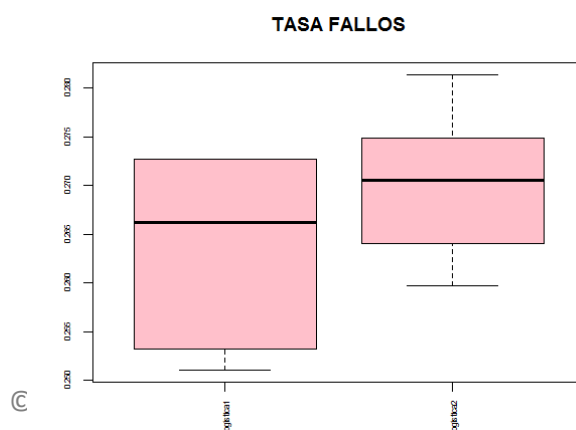
medias1$modelo="Logística1"

medias2<-cruzadalogistica(data=saheartbis,
  vardep="chd",listconti=c("age", "ldl", "famhist.Absent"),
  listclass=c(""), grupos=4,sinicio=1234,repe=5)

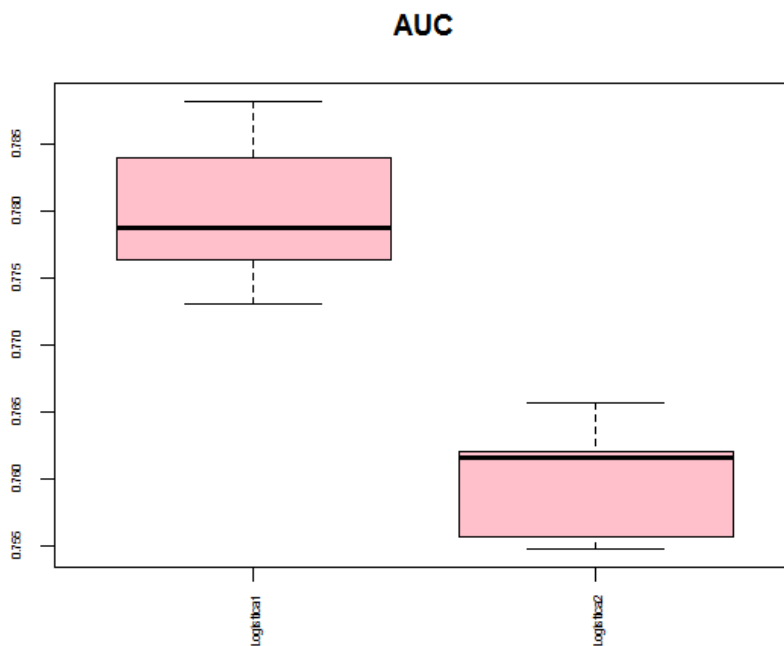
medias2$modelo="Logística2"

union1<-rbind(medias1,medias2)

par(cex.axis=0.5)
boxplot(data=union1,tasa~modelo,main="TASA FALLOS")
```



```
boxplot(data=union1, auc~modelo, main="AUC")
```



```
# *****
# TUNEANDO LA RED CON LOS DOS MODELOS CANDIDATOS
# *****
```

```
avnnnetgrid <- expand.grid(size=c(5,10,15,20),
  decay=c(0.01,0.1,0.001), bag=FALSE)
```

```
redavnnnet<- train(chd~age+tobacco+famhist.Absent+typea+ldl,
  data=saheartbis, method="avNNet", linout = FALSE, maxit=100,
  trControl=control, tuneGrid=avnnnetgrid, repeats=5)
```

```
redavnnnet
```

size	decay	Accuracy	Kappa
5	0.001	0.7038081	0.3100197
5	0.010	0.7081559	0.3207328
5	0.100	0.7137519	0.3342440
10	0.001	0.6861357	0.2754626
10	0.010	0.6873613	0.2813921
10	0.100	0.7137594	0.3341563
15	0.001	0.6717954	0.2471592
15	0.010	0.6874138	0.2818923
15	0.100	0.7155060	0.3379428
20	0.001	0.6796252	0.2702120
20	0.010	0.6727286	0.2543777
20	0.100	0.7180960	0.3425956

Tuning parameter 'bag' was held constant at a value of FALSE
 Accuracy was used to select the optimal model using the largest value.
 The final values used for the model were size = 20, decay = 0.1 and
 bag
 = FALSE.

```
avnnnetgrid <- expand.grid(size=c(5,10,15,20),
  decay=c(0.01,0.1,0.001), bag=FALSE)
```

```
redavnnnet<- train(chd~age+famhist.Absent+ldl,
  data=saheartbis,
  method="avNNet", linout = FALSE, maxit=100,
  trControl=control, tuneGrid=avnnnetgrid,
  repeats=5)
```

```
redavnnnet
```

size	decay	Accuracy	Kappa
5	0.001	0.6904423	0.2807609
5	0.010	0.7034595	0.3070282
5	0.100	0.7255210	0.3494964
10	0.001	0.6735795	0.2555621
10	0.010	0.6835232	0.2668565
10	0.100	0.7255247	0.3503699
15	0.001	0.6661544	0.2422829
15	0.010	0.6817879	0.2641992
15	0.100	0.7246627	0.3475806
20	0.001	0.6510607	0.2078073
20	0.010	0.6830922	0.2673769
20	0.100	0.7242316	0.3475541

Tuning parameter 'bag' was held constant at a value of FALSE
Accuracy was used to select the optimal model using the largest value.
The final values used for the model were size = 10, decay = 0.1 and
bag = FALSE.

```
# *****
# COMPARANDO LOS MODELOS FINALES
# *****
```

```
medias3<-
cruzadaavnnnetbin(data=saheartbis, vardep="chd", listconti=c("age",
  "famhist.Absent", "typea", "ldl",
  "tobacco"), listclass=c(""), grupos=4, inicio=1234, repe=5, size=c(5), deca
y=c(0.1), repeticiones=5, itera=200)
```

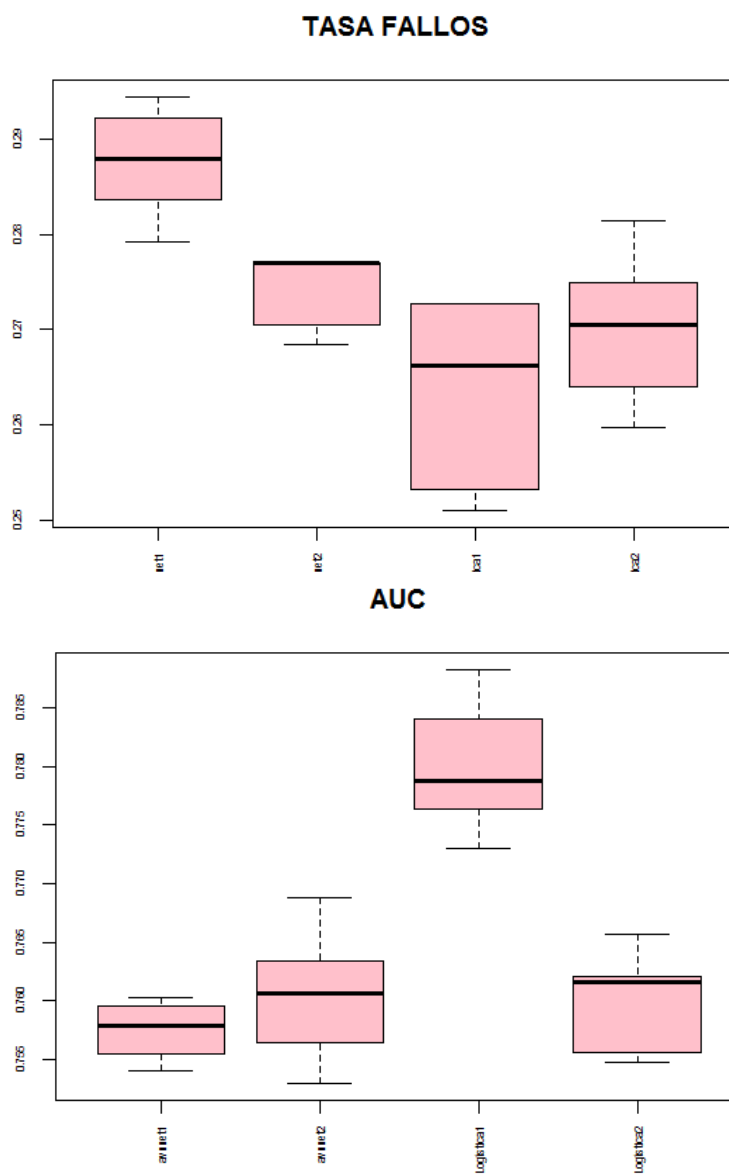
```
medias3$modelo="avnnnet1"
```

```
medias4<-
cruzadaavnnnetbin(data=saheartbis, vardep="chd", listconti=c("age",
  "ldl", "famhist.Absent"),
  listclass=c(""), grupos=4, inicio=1234, repe=5,
  size=c(10), decay=c(0.1), repeticiones=5, itera=200)
```

```
medias4$modelo="avnnnet2"
```

```
union1<-rbind(medias1,medias2,medias3,medias4)
```

```
par(cex.axis=0.5)
boxplot(data=union1, tasa~modelo, col="pink", main="TASA FALLOS")
boxplot(data=union1, auc~modelo, col="pink", main="AUC")
```

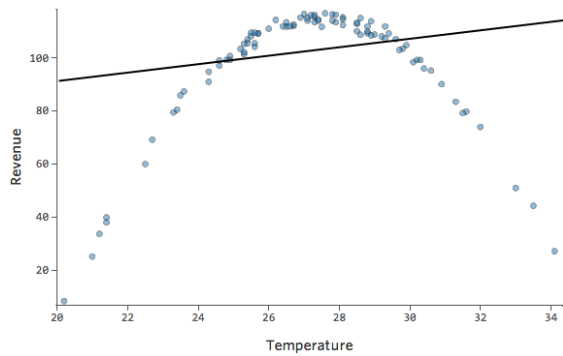


Está claro que en este caso funciona mejor la regresión logística: datos sencillos, con pocas variables input y posiblemente separación lineal entre clases.

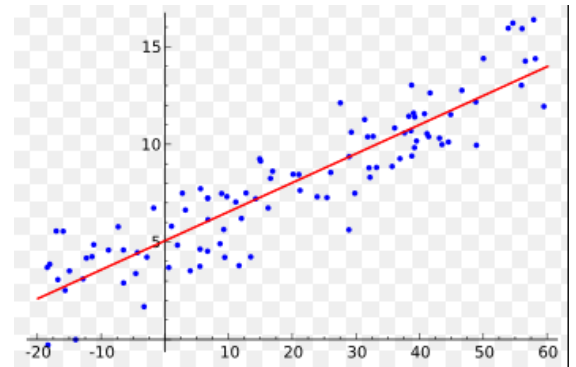
Nota: En clasificación binaria, si la separación entre clases es aproximadamente lineal, es mejor la regresión logística que la red.

Variable dependiente continua

Mejor la red

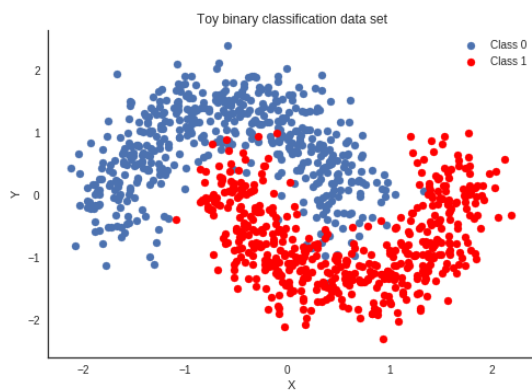


Mejor la regresión lineal

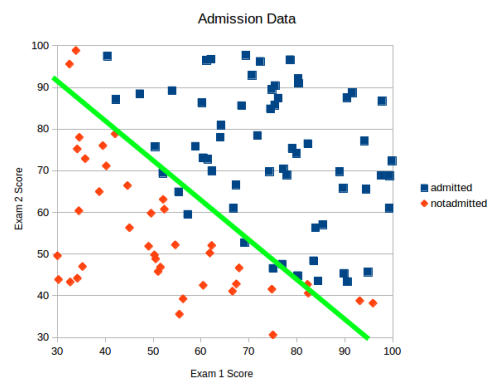


Variable dependiente binaria

Mejor la red



Mejor la regresión logística



Ejercicios (clasificación binaria)

En todos los datasets se seguirá el esquema:

- a) Depuración de datos
- b) Selección de variables
- c) Validación cruzada repetida para comparar modelos con logística
- d) Tuneado de la red y comparación de los mejores modelos con logística

1) Un Ejemplo más complicado: bank

<https://archive.ics.uci.edu/ml/datasets/bank+marketing>

Revisar el código `ejemplo bank.R`

Otros archivos de prueba sin corrección, a discreción del estudiante

`magic.Rda` (v dependiente class)

<https://archive.ics.uci.edu/ml/datasets/magic+gamma+telescope>

`german2.Rda` (v dependiente bad)

[https://archive.ics.uci.edu/ml/datasets/statlog+\(german+credit+data\)](https://archive.ics.uci.edu/ml/datasets/statlog+(german+credit+data))

`australian.Rda`

[http://archive.ics.uci.edu/ml/datasets/statlog+\(australian+credit+approval\)](http://archive.ics.uci.edu/ml/datasets/statlog+(australian+credit+approval))

En este archivo australian credit las variables son:

Continuas

A2 A3 A7 A10 A13 A14

Cualitativas

A1 A4 A5 A6 A8 A9 A11 A12

Dependiente

A15

Bibliografía básica

FAQ que cubre todos los aspectos importantes de las redes neuronales (son 7 FAQ html):

<ftp://ftp.sas.com/pub/neural/FAQ.html>

Referencia clásica

Bishop, C.M. (1995), Neural Networks for Pattern Recognition, Oxford: Oxford University Press.

Libros disponibles en PDF

(muy buenos y apropiados para este curso, aunque no tratan solo de redes neuronales):

Hastie, Tibshirani: The Elements of Statistical Learning (PDF)

(En la web hay más información)

<http://statweb.stanford.edu/~tibs/ElemStatLearn/>

Hastie, Tibshirani: An Introduction to Statistical Learning with Applications in R (PDF)

(básicamente el mismo que el anterior, pero para R)

<http://www-bcf.usc.edu/~gareth/ISL/data.html/>

Paquetes R

<http://topepo.github.io/caret/index.html>

<https://cran.r-project.org/web/packages/caret/index.html>

<https://cran.r-project.org/web/packages/nnet/index.html>