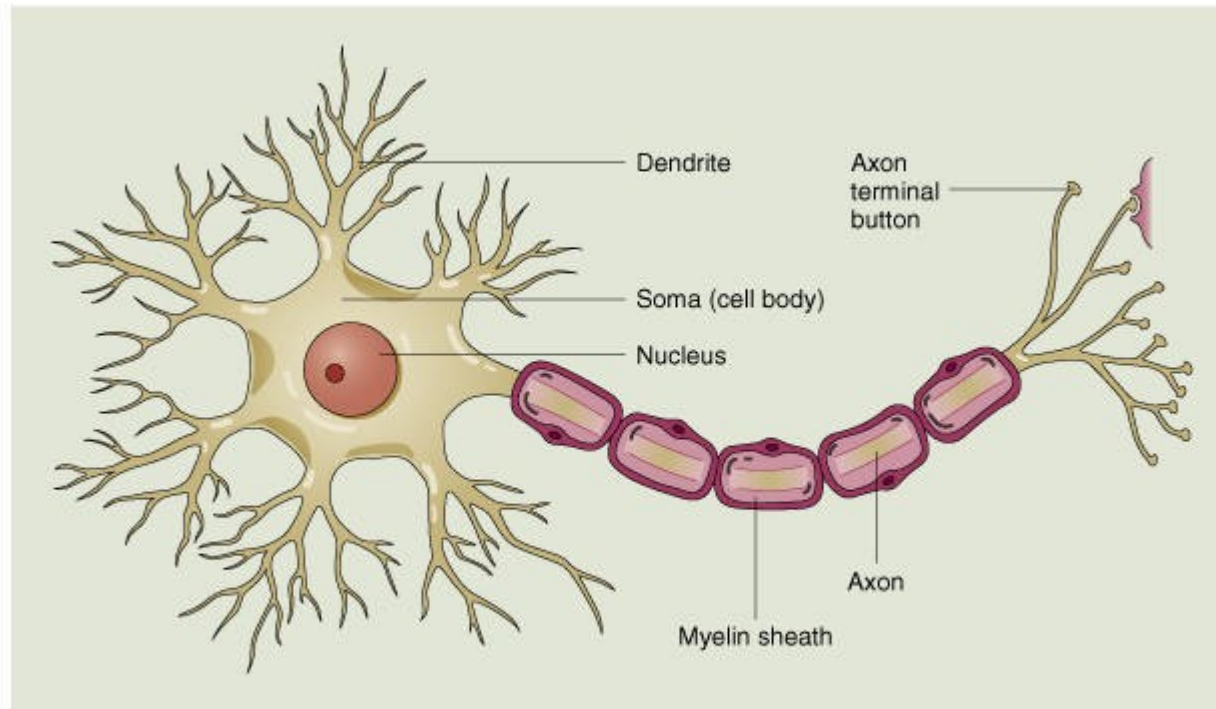


Tema 2

Redes Neuronales

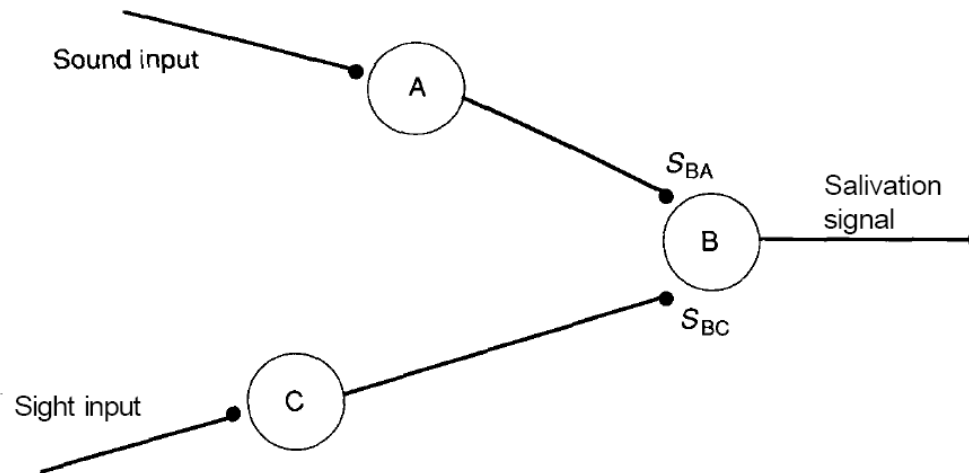
Introducción

Imagen de una Neurona



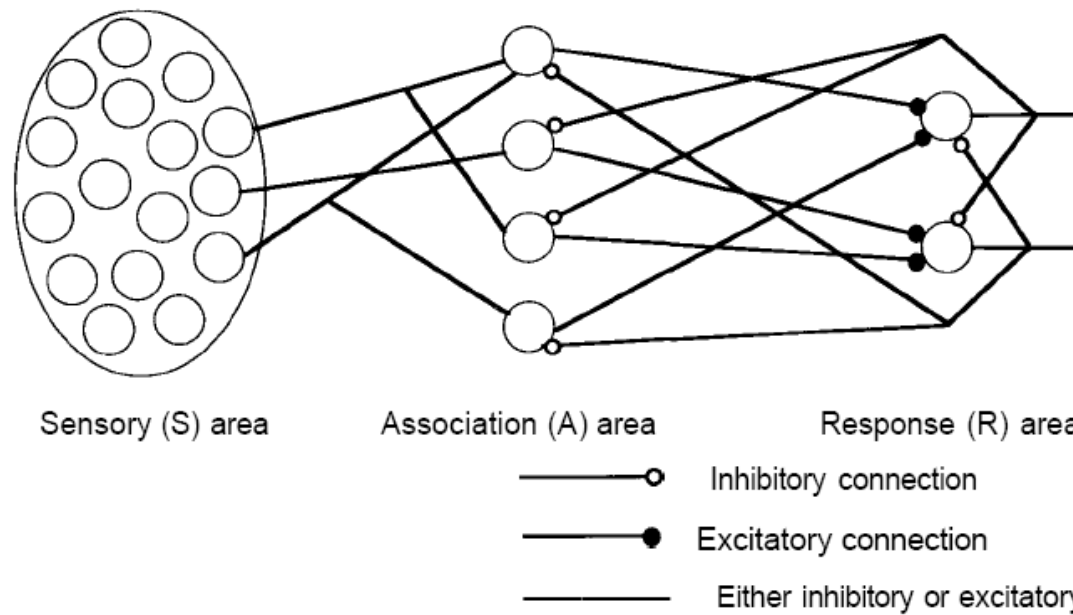
© 2000 John Wiley & Sons, Inc.

Input, Activación y Output



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



Algunos números

Animal	Neuronas en total en el sistema nervioso	Neuronas sólo en el cortex cerebral
Hombre	8×10^{10}	2×10^{10}
Mosca de la fruta	2×10^5	
Abeja	1×10^6	
Rata	2×10^8	2×10^7
Paloma	3×10^8	
Pulpo	5×10^8	
Gato	7×10^8	3×10^8
Mono capuchino	3×10^9	6×10^8
Elefante Africano	2×10^{11}	1×10^{10}

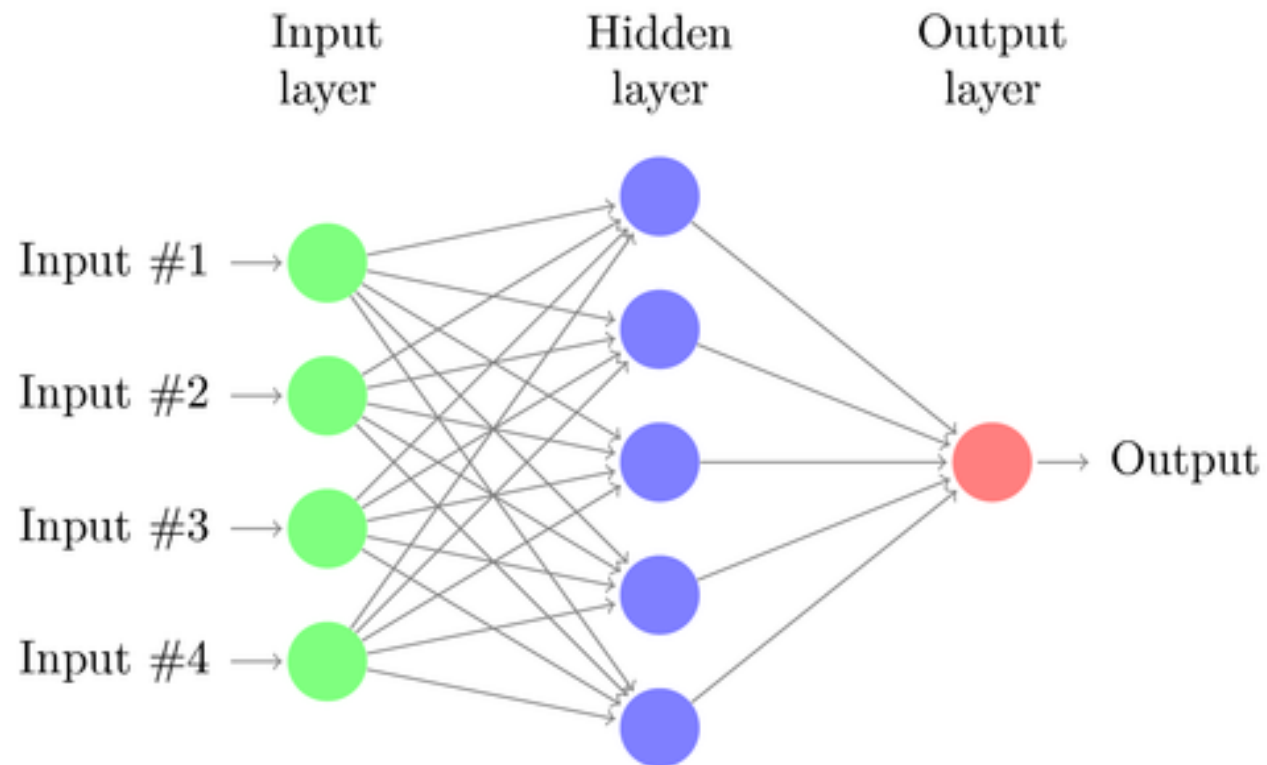
*El hombre tiene aproximadamente 1.5×10^{14} sinapsis.

*Entre los mamíferos, son considerados más inteligentes aquellos con mayor número de neuronas y sinapsis: cetáceos y primates.

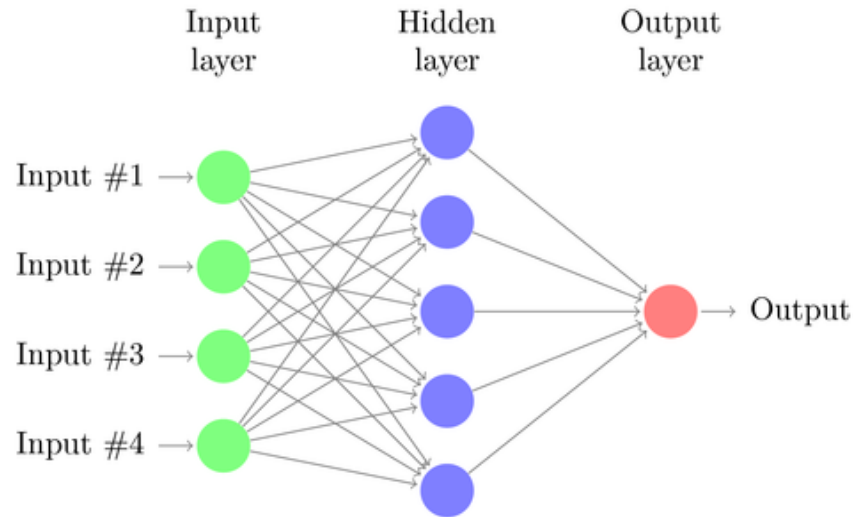
*Entre las aves, los córvidos, loros y búhos.

*El hombre no se diferencia de los primates en su proporción de neuronas relativamente al tamaño (es un primate a escala), pero tiene un cerebro más plástico e influenciable (flexible) y la anatomía del cerebro se hereda mejor que la de los primates (ref 2015 <http://www.pnas.org/content/112/48/14799.abstract>)

Red Neuronal Artificial



Planteamiento



Una Red Neuronal es en realidad un modelo de la forma

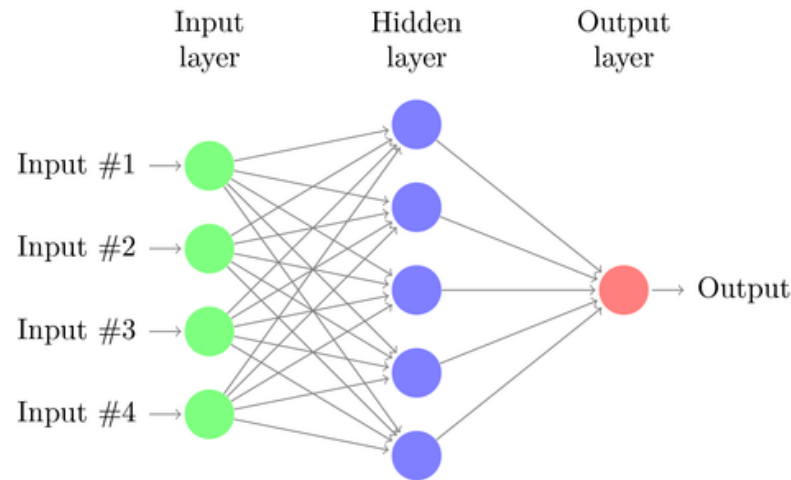
$$y=f(x_1,x_2,x_3,...)$$

donde la función f es por lo general no lineal.

Ejemplo de red:

$$y=29.8+85.2*\tanh(-0.9+2.1*X_1-0.15*X_2)-79*\tanh(-3.5+5*X_1+0.01*X_2)$$

Terminología



Nodos Input =Variables independientes del modelo

Nodos Output=Variables dependientes del modelo (puede haber más de una)

Capa oculta =Capa con nodos ocultos (variables artificiales, no existen como tal en los datos).

(layer=capa; hidden=oculto/a).

¿Qué problemas de modelización estadística se pueden abordar con las redes neuronales?

- **Regresión**
- **Regresión no lineal**
- **Regresión logística**
- **Análisis discriminante**
- **Series temporales**
- **Análisis Cluster**
- **Problemas de Optimización**
- **Etc.**

¿Qué problemas en el campo de la Inteligencia de Negocios se suelen abordar con las redes neuronales?

- Finanzas: Estudios de evolución de precios en mercados financieros
- Banca: Predicción de deserciones de clientes (Churn)
Predicción de morosidad
Credit Scoring
Detección de fraude
- Seguros: Predicción de pérdidas
Evaluación de riesgos asociados a clientes
Análisis de supervivencia
- Investigación de Mercados:
Segmentación y ranking de clientes
Predicción de Ventas
Predicción de precios en el mercado inmobiliario
- Etc.

[A]=La red neuronal supera al método clásico; [B]=Funcionan igual; [C]=Mejor clásico

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]*

[A]=La red neuronal supera al método clásico; [B]=Funcionan igual; [C]=Mejor clásico

Table 4
Applications in marketing

Reference	Statistical model	No. of variables	Sample size	Validation method	Error measure	Finding
Hruschka (1993)	Reg	5	60	Single data	MSE	[A]
Dutta et al. (1994)	Reg	7	138	Tr-Ts (64-36)	Total squared error	[C]
Dasgupta et al. (1994)	LR, DA	13 & 15	714 & 829	5-fold CV/R-15	Confusion matrix	[B]*
Fish et al. (1995)	LR, DA	7 & 5	100 & 50	Tr-Ts (60-40)	Confusion matrix	[A]
Kumar et al. (1995)	LR	15	1048	Tr-Ts (66-34)	RMSE, C-index, Confusion matrix	[A]
Agrawal and Schorling (1996)	Multinomial logit	3 in each	2301, 3927, 2493	Tr-Ts /R-3	MAE	[A]*
West et al. (1997)	LR, DA	19	800	10-fold CV	Confusion matrix	[A]*
Setiono et al. (1998)	DA	10	638	Tr-Va-Ts (60-20-20)	Confusion matrix	[A]
Ainscough and Aronson (1999)	Reg	8	575	Tr-Ts (80-20)	MSE, R^2	[A]*
Thieme et al. (2000)	k-NN, LR, Reg, DA	43	612	10-fold CV	MSE, SSE, Confusion matrix	[A]
Limsombunchai et al. (2005)	LR	19	525	Tr-Ts (80-20)	Confusion matrix	[A]
Chiang et al. (2006)	LR	18	224	5-fold CV/R-15	Confusion matrix	[A]*

¿Por qué utilizar las redes neuronales si ya existen modelos estadísticos para cada problema de predicción continua, clasificación, series o clustering?

Paréntesis Cultural

Las dos culturas de la modelización estadística

“Statistical Modeling: The Two Cultures”

**Leo Breiman,
Statistical Science (2001)**

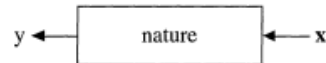
Statistical Modeling: The Two Cultures

Leo Breiman

Abstract. There are two cultures in the use of statistical modeling to reach conclusions from data. One assumes that the data are generated by a given stochastic data model. The other uses algorithmic models and treats the data mechanism as unknown. The statistical community has been committed to the almost exclusive use of data models. This commitment has led to irrelevant theory, questionable conclusions, and has kept statisticians from working on a large range of interesting current problems. Algorithmic modeling, both in theory and practice, has developed rapidly in fields outside statistics. It can be used both on large complex data sets and as a more accurate and informative alternative to data modeling on smaller data sets. If our goal as a field is to use data to solve problems, then we need to move away from exclusive dependence on data models and adopt a more diverse set of tools.

1. INTRODUCTION

Statistics starts with data. Think of the data as being generated by a black box in which a vector of input variables \mathbf{x} (independent variables) go in on one side, and on the other side the response variables \mathbf{y} come out. Inside the black box, nature functions to associate the predictor variables with the response variables, so the picture is like this:



There are two goals in analyzing the data:

Prediction. To be able to predict what the responses are going to be to future input variables;

Information. To extract some information about how nature is associating the response variables to the input variables.

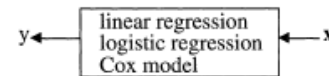
There are two different approaches toward these goals:

The Data Modeling Culture

The analysis in this culture starts with assuming a stochastic data model for the inside of the black box. For example, a common data model is that data are generated by independent draws from

response variables = $f(\text{predictor variables, random noise, parameters})$

The values of the parameters are estimated from the data and the model then used for information and/or prediction. Thus the black box is filled in like this:

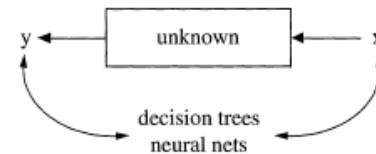


Model validation. Yes—no using goodness-of-fit tests and residual examination.

Estimated culture population. 98% of all statisticians.

The Algorithmic Modeling Culture

The analysis in this culture considers the inside of the box complex and unknown. Their approach is to find a function $f(\mathbf{x})$ —an algorithm that operates on \mathbf{x} to predict the responses \mathbf{y} . Their black box looks like this:



Model validation. Measured by predictive accuracy.

Estimated culture population. 2% of statisticians, many in other fields.

1. Limitaciones en los modelos estadísticos

Every article started with

Assume that the data are generated by the following model: ...

followed by mathematics exploring inference, hypothesis testing and asymptotics.

Regresión:

$$y = b_0 + b_1x_1 + b_2x_2 + e$$

Regresión logística: $\log(p/(1-p)) = b_0 + b_1x_1 + b_2x_2 + e$

Complejidad actual de los datos a tratar: número y tipo de variables, relaciones oscuras, lineales y no lineales, rangos de diferente comportamiento, las hipótesis de normalidad no se cumplen, etc.
-->es difícil establecer un modelo y es optimista ceñirse a él.

Los modelos clásicos pueden funcionar (son robustos), pero habrá sin duda mejores opciones.

2. La multiplicidad de modelos

En general, para los mismos datos, suelen existir muchos modelos alternativos que funcionan de manera equivalente:

- **Modelos con diferentes variables input**
- **Modelos con diferentes funciones (logística versus discriminante, regresión o regresión PLS, etc.)**

Esto es natural, pero es una realidad que se suele soslayar, pretendiendo siempre mostrar que el modelo planteado es óptimo.

La consecuencia es rigidez en los planteamientos y poca eficacia predictiva.

Las alternativas pasan por enfrentar modelos clásicos con algorítmicos a través de validación cruzada y/o datos test, y probar muchas opciones incluidos los métodos ensemble (agregado de modelos: bagging, boosting, etc.)

3. Explicar o predecir

Los modelos estadísticos tienen la ventaja de aportar información sobre la aportación de las diferentes variables input al modelo, de la significatividad, signo y magnitud de los parámetros correspondientes.

Pero a menudo eso es poco importante relativo a la capacidad predictiva del modelo.

Las medidas clásicas de ajuste, significatividad, etc. pasan a segundo plano cuando se tiene mucha información y el objetivo es predecir.

La precisión, capacidad predictiva bajo diferentes perspectivas y funciones de error son más importantes que la comprensión lógica del modelo en el contexto.

4. Cuando la modelización previa es inviable

En un gran porcentaje de los casos se desconocen las leyes que relacionan las variables input con las output:

- **No linealidad, funciones desconocidas entre variables input y output**
- **Modelos a trozos, según rangos de variables**
- **Variables latentes**
- **Información redundante**
- **Información localizada importante en algunos rangos/variables**
- **Datos missing**

-> Se necesitan métodos flexibles que puedan abordar este tipo de datos pues es casi imposible derivar artesanalmente todas las relaciones y solucionar simultáneamente todos los problemas desde una perspectiva clásica.

Métodos algorítmicos modernos como alternativa a los métodos de modelización estadística clásica, para predicción-clasificación

Universales (sólo necesitan monitorizar ciertos parámetros)

- **Gradient Boosting**
- **Redes Neuronales**
- Support Vector Machines (SVM)
- Multi Adaptive Regression Splines (MARS)
- Trees y Random Forests

No universales (necesitan un cierto tipo de modelización previa)

Splines, kernel, wavelets, naïve bayes, mixtures, etc.

(Fin del Paréntesis Cultural)

Cuándo utilizar las redes neuronales

- **No linealidad, funciones desconocidas entre variables input y output**
- **Complejidad de los datos (efecto temporal, muchas variables categóricas, datos censurados)**
- **Complejidad del output (varias variables output simultáneas, de diferente tipo)**

En general, para utilizar una red neuronal con garantías se requieren muchas observaciones (al no haber inferencia propiamente dicha se necesitan datos test para validar el modelo)

Cuándo NO utilizar las redes neuronales

- **Linealidad** en las relaciones, o bien funciones de relación conocidas entre variables input y output (modelos econométricos conocidos, datos de química, termodinámica, biometría, etc. con funciones no lineales pero bien conocidas, etc.)
- **Pocas observaciones** (aquí la inferencia clásica es fundamental para construir modelos robustos)
- El objetivo no es predecir, sino **explicar**, o bien son las dos cosas (la red es una caja negra y es difícil extraer información aparte de las buenas predicciones)

En general, siempre que se pueda se deben probar modelos clásicos y contrastarlos en términos de predicción (datos test, validación cruzada, etc.) con la red neuronal

Tabla extraída del libro The Elements of Statistical Learning (Hastie, Tibshirani, 2001)

TABLE 10.1. *Some characteristics of different learning methods. Key: ● = good, ● = fair, and ● = poor.*

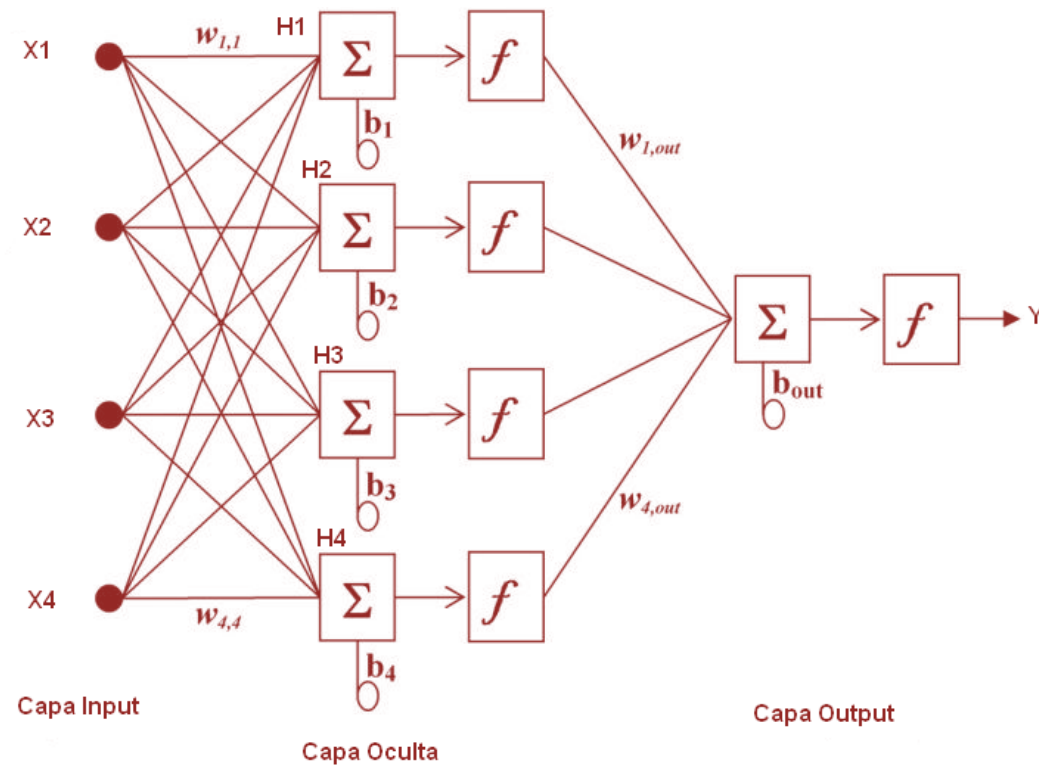
Characteristic	Neural nets	SVM	Trees	MARS	k-NN, kernels
Natural handling of data of “mixed” type	●	●	●	●	●
Handling of missing values	●	●	●	●	●
Robustness to outliers in input space	●	●	●	●	●
Insensitive to monotone transformations of inputs	●	●	●	●	●
Computational scalability (large N)	●	●	●	●	●
Ability to deal with irrelevant inputs	●	●	●	●	●
Ability to extract linear combinations of features	●	●	●	●	●
Interpretability	●	●	●	●	●
Predictive power	●	●	●	●	●

Redes Neuronales para Regresión (Variable dependiente continua)

Construcción del modelo y primeros ejemplos

Construcción del modelo

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:

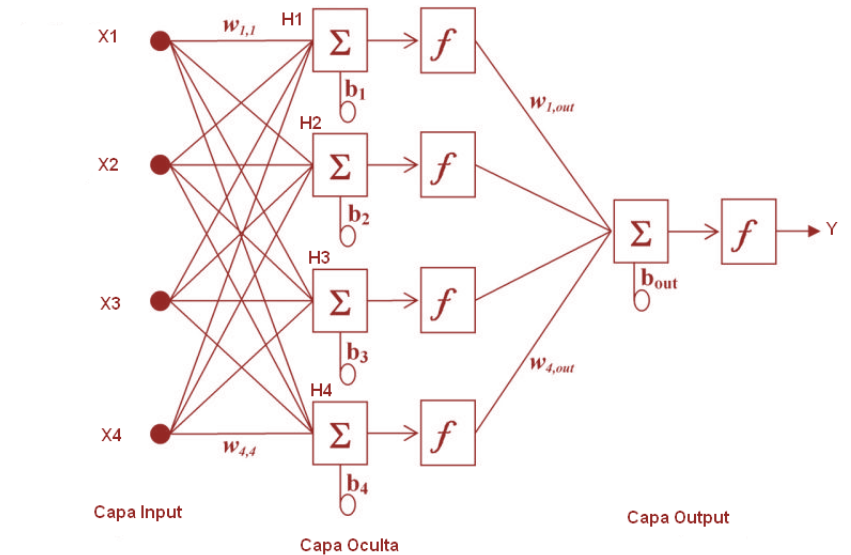
$$H1 = w_{11}X1 + w_{21}X2 + w_{31}X3 + w_{41}X4 + b_1$$

$$H2 = w_{12}X1 + w_{22}X2 + w_{32}X3 + w_{42}X4 + b_2$$

$$H3 = w_{13}X1 + w_{23}X2 + w_{33}X3 + w_{43}X4 + b_3$$

$$H4 = w_{14}X1 + w_{24}X2 + w_{34}X3 + w_{44}X4 + b_4$$

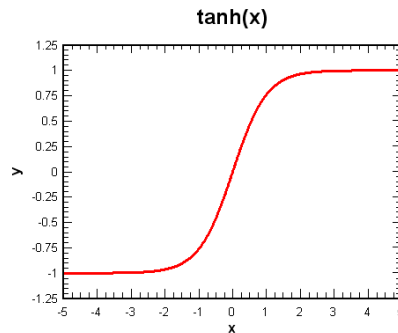
El parámetro constante b_j se denomina, en jerga neuronal, bias (sesgo)



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,

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



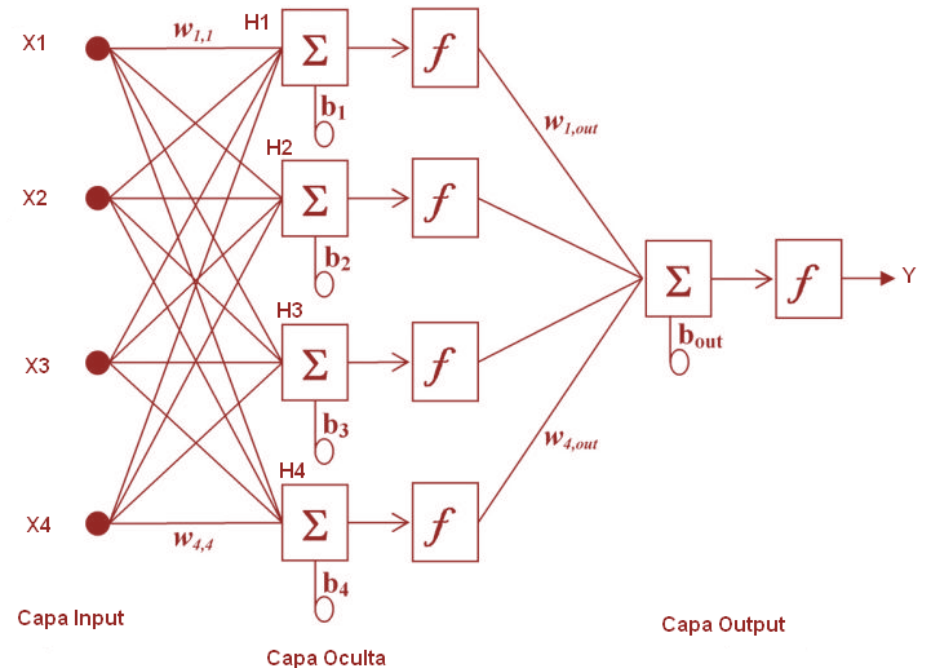
Aplicando la función de activación a cada nodo oculto:

$$\begin{aligned} H1 &= \tanh(H1) = \\ &= \tanh(w_{11}X1 + w_{21}X2 + w_{31}X3 + w_{41}X4 + b_1) \end{aligned}$$

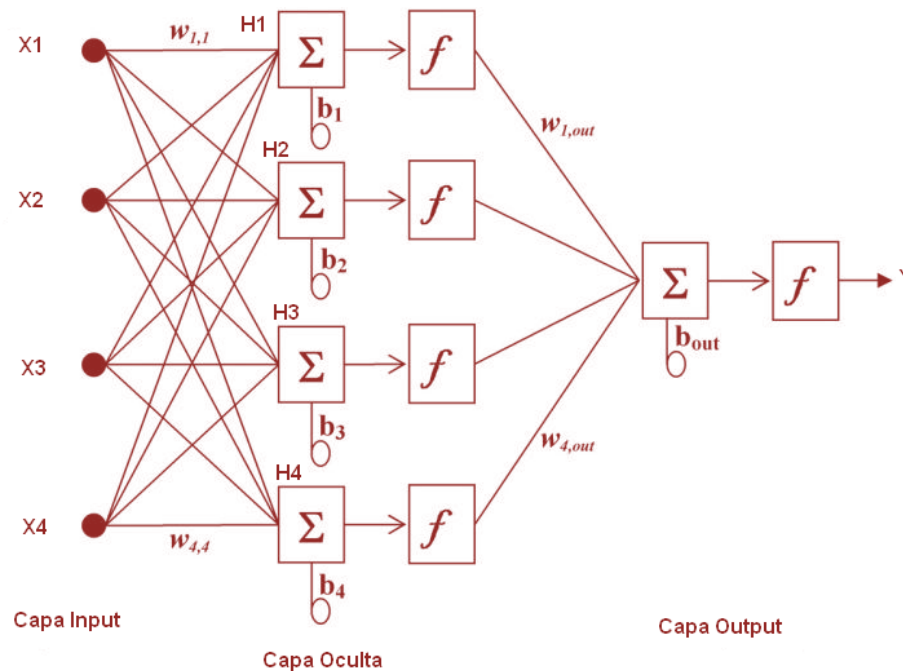
$$H2 = \tanh(w_{12}X1 + w_{22}X2 + w_{32}X3 + w_{42}X4 + b_2)$$

$$H3 = \tanh(w_{13}X1 + w_{23}X2 + w_{33}X3 + w_{43}X4 + b_3)$$

$$H4 = \tanh(w_{14}X1 + w_{24}X2 + w_{34}X3 + w_{44}X4 + b_4)$$



Finalmente aplicamos combinación y después activación de la capa oculta a la capa output



Combinación de los nodos de la capa oculta:

$$Y = w_{1,out}H1 + w_{2,out}H2 + w_{3,out}H3 + w_{4,out}H4 + b_{out}$$

Activación final (nota: cuando la variable output es **continua** no se realiza activación)

$Y = \tanh(Y) = \tanh(w_{1,out}H1 + w_{2,out}H2 + w_{3,out}H3 + w_{4,out}H4 + b_{out})$ si la variable output es binaria, codificada 0,1 o -1,1

$Y = Y$ si la variable output es continua

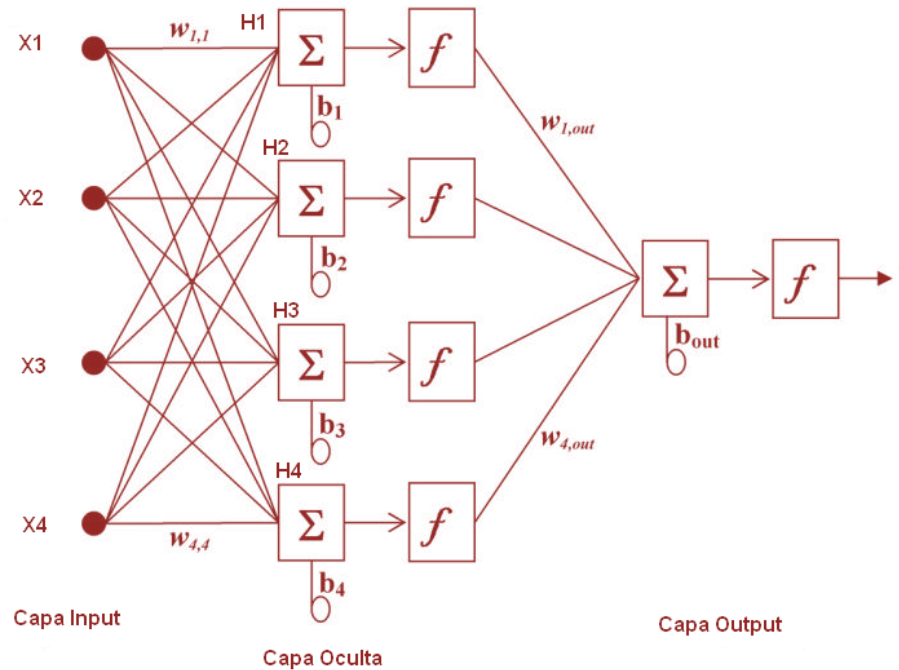
Desarrollando los valores de los nodos ocultos:

$$\begin{aligned} Y &= \tanh(w_{1,\text{out}}H_1 + w_{2,\text{out}}H_2 + w_{3,\text{out}}H_3 + w_{4,\text{out}}H_4 + b_{\text{out}}) = \\ &= \tanh(w_{1,\text{out}}(\tanh(w_{11}X_1 + w_{21}X_2 + w_{31}X_3 + w_{41}X_4 + b_1)) \\ &+ w_{2,\text{out}}(\tanh(w_{12}X_1 + w_{22}X_2 + w_{32}X_3 + w_{42}X_4 + b_2)) + \\ &+ w_{3,\text{out}}(\tanh(w_{13}X_1 + w_{23}X_2 + w_{33}X_3 + w_{43}X_4 + b_3)) + \\ &+ w_{4,\text{out}}(\tanh(w_{14}X_1 + w_{24}X_2 + w_{34}X_3 + w_{44}X_4 + b_4)) + \\ &+ b_{\text{out}}) \end{aligned}$$

En una red con 4 variables inputs , una output, una capa oculta con cuatro nodos ocultos, funciones de combinación lineal y funciones de activación tanh, el modelo planteado es:

$$\begin{aligned} Y &= \tanh(w_{1,\text{out}}(\tanh(w_{11}X_1 + w_{21}X_2 + w_{31}X_3 + w_{41}X_4 + b_1)) \\ &+ w_{2,\text{out}}(\tanh(w_{12}X_1 + w_{22}X_2 + w_{32}X_3 + w_{42}X_4 + b_2)) + \\ &+ w_{3,\text{out}}(\tanh(w_{13}X_1 + w_{23}X_2 + w_{33}X_3 + w_{43}X_4 + b_3)) + \\ &+ w_{4,\text{out}}(\tanh(w_{14}X_1 + w_{24}X_2 + w_{34}X_3 + w_{44}X_4 + b_4)) + \\ &+ b_{\text{out}}) \end{aligned}$$

Contemos los parámetros:



4×4 pesos capa input-capacapa oculta + 4 bias capa oculta + 4 pesos capa oculta-capacapa output + bias capa output = 25

(¡Más vale tener muchas observaciones!)

En general, Número de parámetros 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: en regresión clásica serían $k+1=5$ en nuestro caso).

Dejando ya a un lado la parafernalia gráfica,
el objetivo computacional concreto es **estimar** los parámetros w y b del modelo

$$\begin{aligned} Y = & \tanh(w_{1,\text{out}}(\tanh(w_{11}X_1 + w_{21}X_2 + w_{31}X_3 + w_{41}X_4 + b_1))) \\ & + w_{2,\text{out}}(\tanh(w_{12}X_1 + w_{22}X_2 + w_{32}X_3 + w_{42}X_4 + b_2))) + \\ & + w_{3,\text{out}}(\tanh(w_{13}X_1 + w_{23}X_2 + w_{33}X_3 + w_{43}X_4 + b_3))) + \\ & + w_{4,\text{out}}(\tanh(w_{14}X_1 + w_{24}X_2 + w_{34}X_3 + w_{44}X_4 + b_4))) + \\ & + b_{\text{out}}) \end{aligned}$$

Una vez estimados los pesos, se obtienen las predicciones (se denotará por \hat{y} la predicción obtenida con el modelo, mientras que y “sin gorro” son los valores reales).

$$\hat{y} = \tanh(0.21(\tanh(0.15X_1 - 0.53X_2 + 2.1X_3 + 3.5X_4 + 6)) + \dots + \dots + ..)$$

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.).

Pregunta:

¿¿¿Por qué el (exagerado, sobreparametrizado, monstruoso) modelo de red neuronal

$$\begin{aligned} Y = & \tanh(w_{1,\text{out}}(\tanh(w_{11}X_1 + w_{21}X_2 + w_{31}X_3 + w_{41}X_4 + b_1))) \\ & + w_{2,\text{out}}(\tanh(w_{12}X_1 + w_{22}X_2 + w_{32}X_3 + w_{42}X_4 + b_2)) + \\ & + w_{3,\text{out}}(\tanh(w_{13}X_1 + w_{23}X_2 + w_{33}X_3 + w_{43}X_4 + b_3)) + \\ & + w_{4,\text{out}}(\tanh(w_{14}X_1 + w_{24}X_2 + w_{34}X_3 + w_{44}X_4 + b_4)) + \\ & + b_{\text{out}} \end{aligned}$$

podría funcionar mejor que el (sencillo, bonito, clásico, conocido) modelo básico de regresión

$$Y = b_0 + b_1X_1 + b_2X_2 + b_3X_3 + b_4X_4$$

???

El Soporte Teórico

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

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

$$|F(x) - f(x)| < \epsilon$$

Es decir, **si existe relación entre las variables input y la variable output 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 construida con la red neuronal

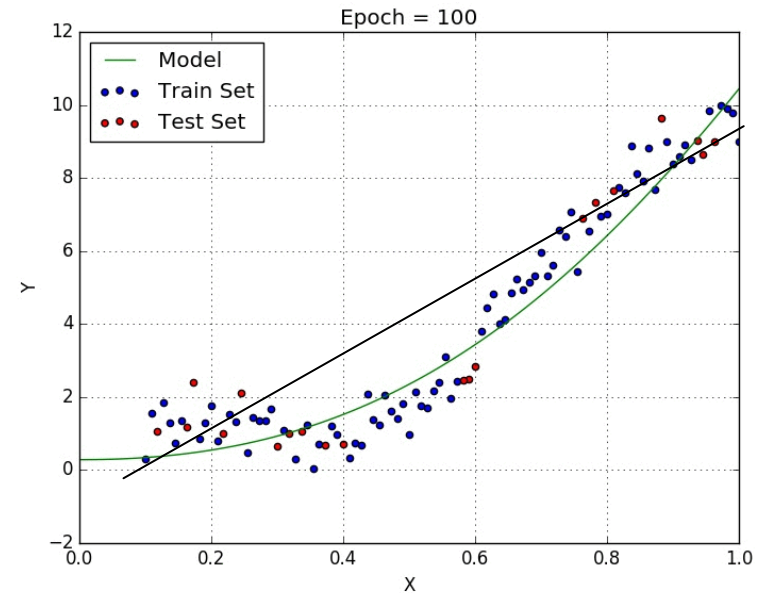
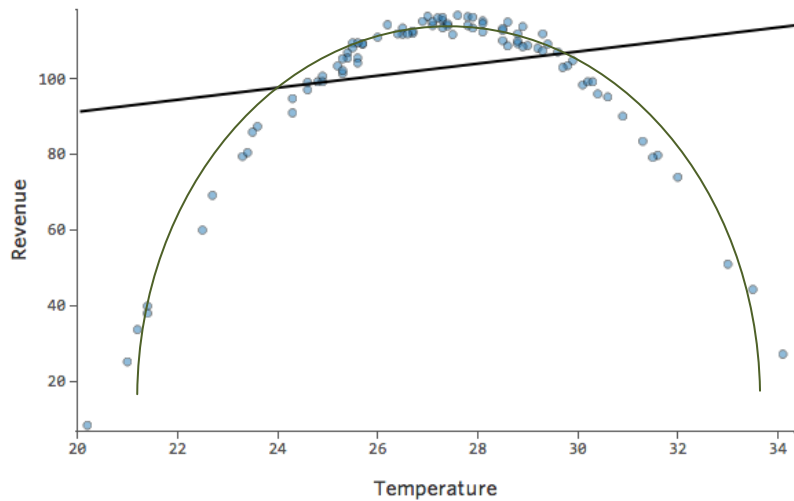
$$F(x) = \sum_{i=1}^N \alpha_i \varphi(w_i^T x + b_i)$$

En nuestro caso, $F(x)=Y$:

$$\begin{aligned} Y = & \tanh(w_{1,\text{out}}(\tanh(w_{11}X_1 + w_{21}X_2 + w_{31}X_3 + w_{41}X_4 + b_1))) \\ & + w_{2,\text{out}}(\tanh(w_{12}X_1 + w_{22}X_2 + w_{32}X_3 + w_{42}X_4 + b_2))) + \\ & + w_{3,\text{out}}(\tanh(w_{13}X_1 + w_{23}X_2 + w_{33}X_3 + w_{43}X_4 + b_3))) + \\ & + w_{4,\text{out}}(\tanh(w_{14}X_1 + w_{24}X_2 + w_{34}X_3 + w_{44}X_4 + b_4))) + \\ & + b_{\text{out}} \end{aligned}$$

“Solamente” hay que decidir el **número de nodos ocultos N**, la **función de activación φ** (en el ejemplo, $N=4$, $\varphi=\tanh$) y finalmente el valor de los parámetros. Esta elección se realiza a través de métodos iterativos, observando los valores que hacen óptimo el valor de la función objetivo sobre datos de validación.

Gráficamente, mientras la **regresión** es un modelo rígido exclusivamente lineal, la **red neuronal** podrá ajustarse a las diferentes relaciones y/x , sea cual sea la relación, aunque sea desconocida a priori.



Recapitulemos

- La red neuronal consiste en un planteamiento gráfico que resulta en un planteamiento de aproximación funcional a la relación entre las variables input y las variables output
- La red neuronal funcionará mejor que los modelos habituales si las relaciones reales subyacentes son no lineales o complejas
- Al no existir planteamientos inferenciales sobre un modelo a priori, la red tiende al sobreajuste y serán necesarios muchos datos de training y de validación para obtener un modelo robusto

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

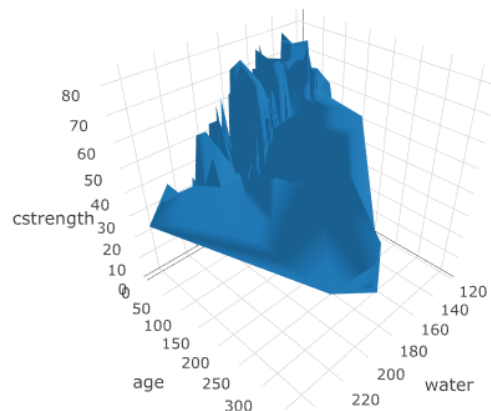
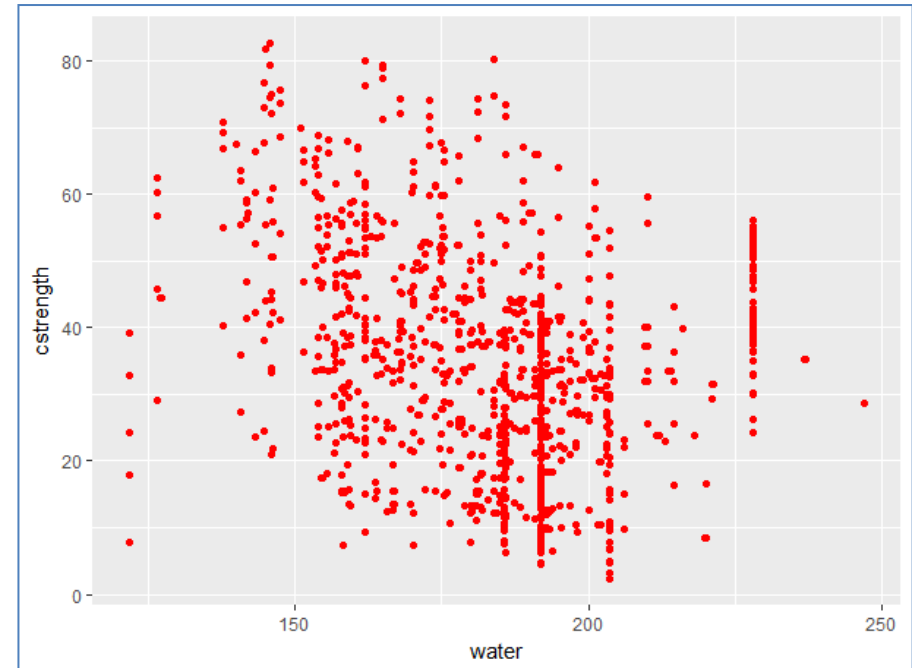
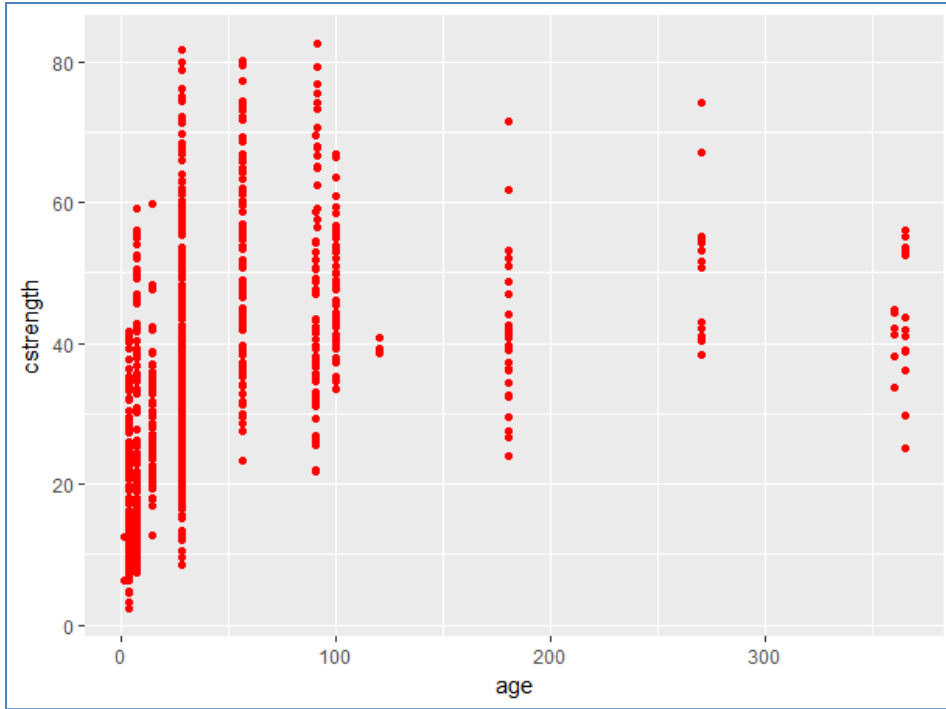
Dos nodos input continuos: age y water

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

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

[illegible]

Aparente No linealidad: buena situación para utilizar las redes



Algunos gráficos básicos

```
library(ggplot2)
```

```
library(plotly)
```

```
ggplot(compress, aes(age, cstrength))+geom_point(color="red")
```

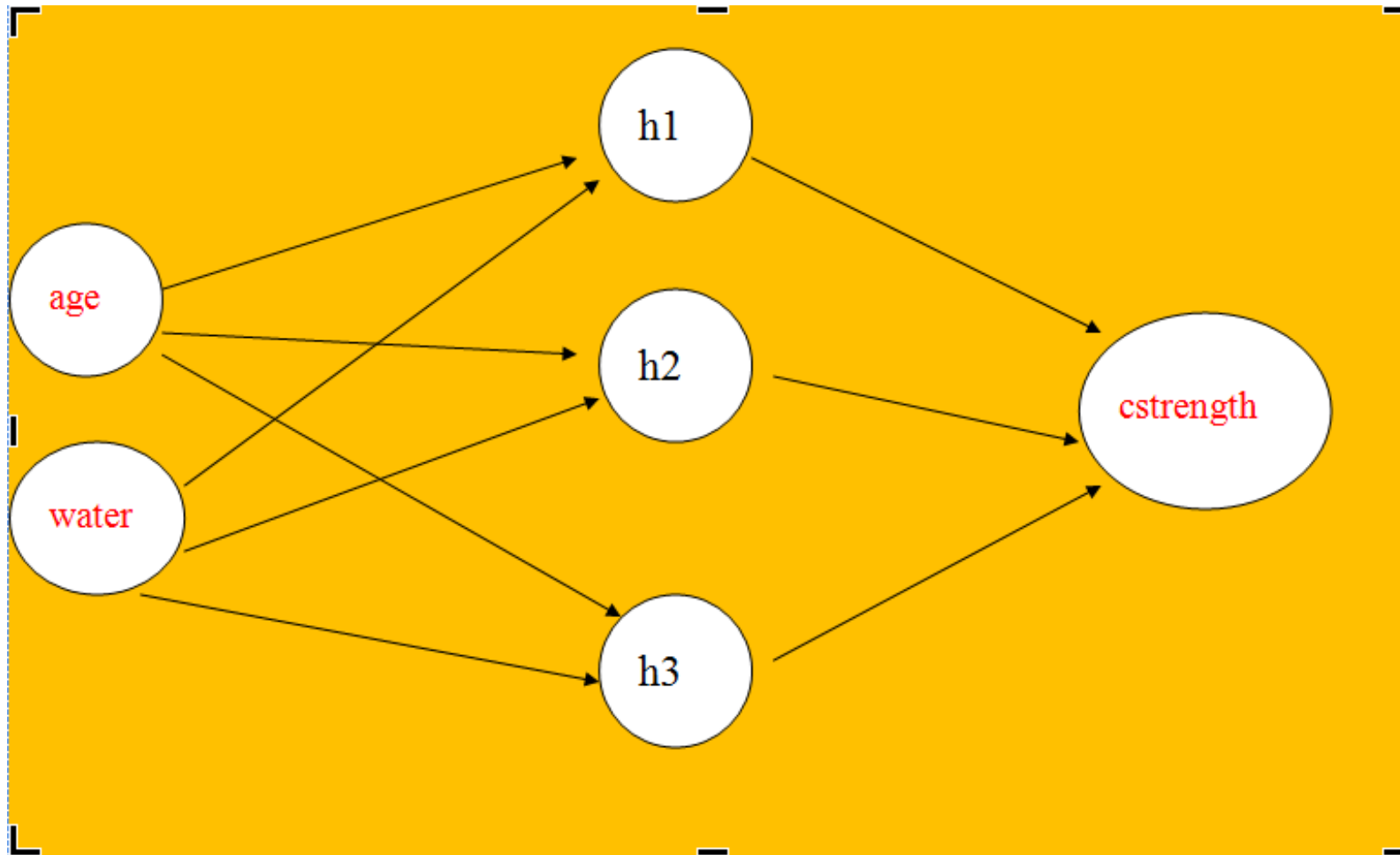
```
ggplot(compress, aes(water, cstrength))+geom_point(color="red")
```

```
hist(compress$age)
```

```
hist(compress$water)
```

```
plot_ly(compress, x = ~water, y = ~age, z = ~cstrength, type  
='mesh3d')
```

Planteamiento de una red 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
- 3) Utilización de la red creada para predecir nuevas observaciones

Pre-procesado de las variables input para la construcción de una red neuronal , 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, 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.

Variables input estandarizadas

```
# 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)

compress<-scale(compress[,listconti], center = means, scale = sds)
compress<-data.frame(cbind(compress,cstrength))
```

$age = (age - 45.7) / 63.2$

$water = (water - 181.6) / 21.4$

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
47.03	0.7	2.17
43.7	5.06	2.17
36.45	-0.28	2.17
45.85	-0.28	2.17
39.29	-0.28	2.17
38.07	0.7	0.49
28.02	-0.28	0.49
43.01	3.55	2.17
42.33	0.7	2.17
47.81	-0.28	2.17
52.91	0.7	2.17

Estimación de los parámetros de la red (“entrenar” la red en la jerga habitual)

```
# 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, linou=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)
```

Informacion 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)

Salida del nnet:

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
```

Todo lo que va al nodo 1:

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

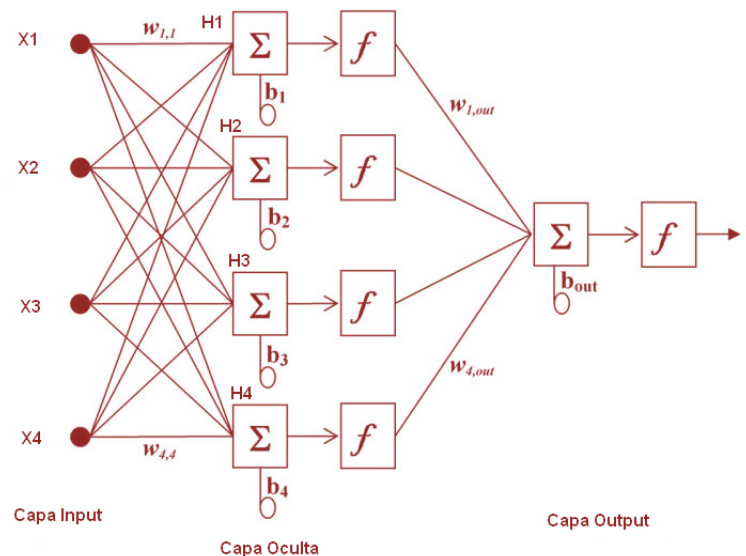
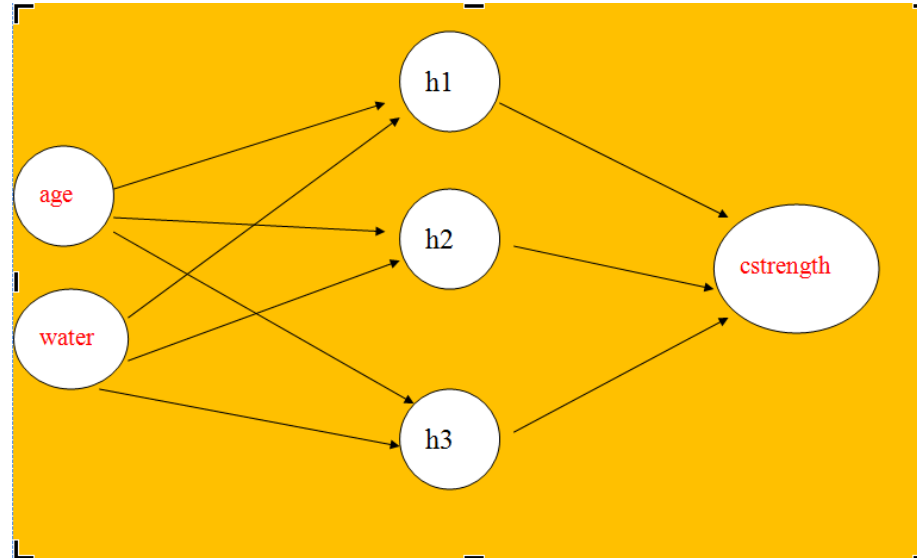
Esto significa:

$h1 = w_{11} * \text{age} + w_{21} * \text{water} + b_1$
 $h1 = 3.31 * \text{age} + 0.13 * \text{water} + 3.05$

luego activación:

$h1 = \tanh(h1)$

El resto de nodos igual



b) Proceso de optimización-estimación de los pesos.

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

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

(aunque el paquete nnet presenta el valor de $\text{SSE} = n * \text{MSE}$)

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 $h1 = 3.31 * \text{age} + 0.13 * \text{water} + 3.05$, $h2 = 0.13 * \text{age} + \dots$, 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

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

```
# 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 $\text{MSE}=135.93$.

c) nnet presenta también en la consola el listado de los estimadores de los pesos (ya visto) y crea el objeto con el nombre del modelo, que contiene información relevante.

```
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
```

```
# 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
....

Una 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 **sobreajustando**, y que para **datos nuevos** la red funcione peor que la regresión

Una prueba básica con training-test utilizando el paquete caret

Esquema básico paquete caret

- 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)
```

```
# 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(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 hay grid.

```
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

```
# 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.

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

  print(reg1$results$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] 11.88971

[1] 14.58365

[1] 124

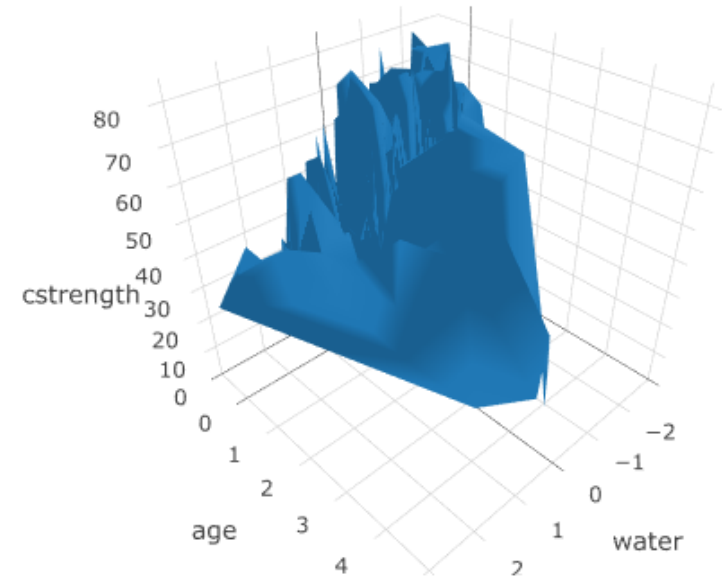
[1] 12.20924

[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

Codificación de las variables input categóricas

Las variables input categóricas 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.

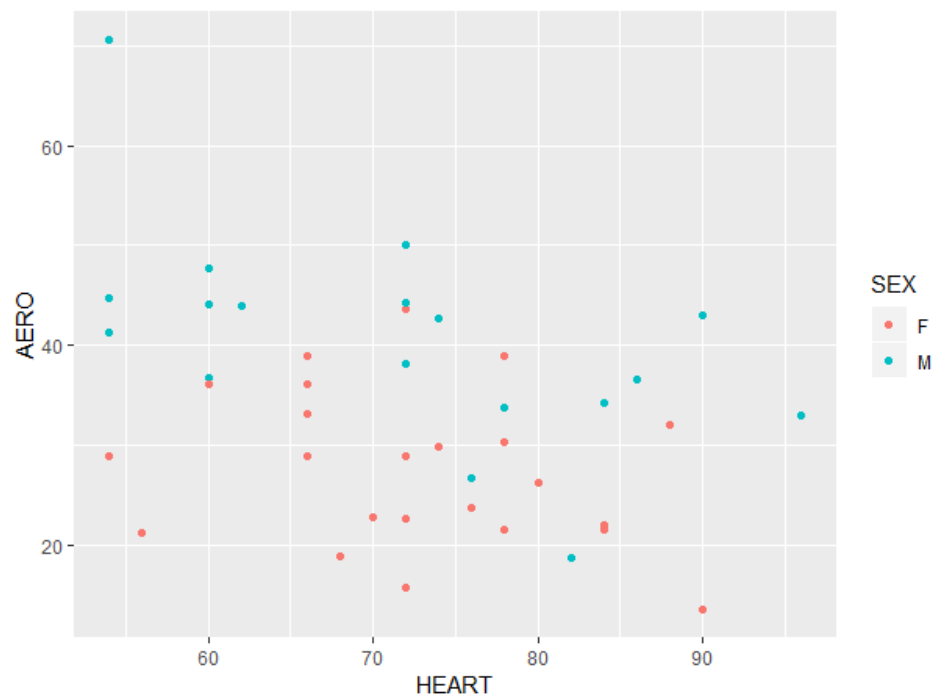
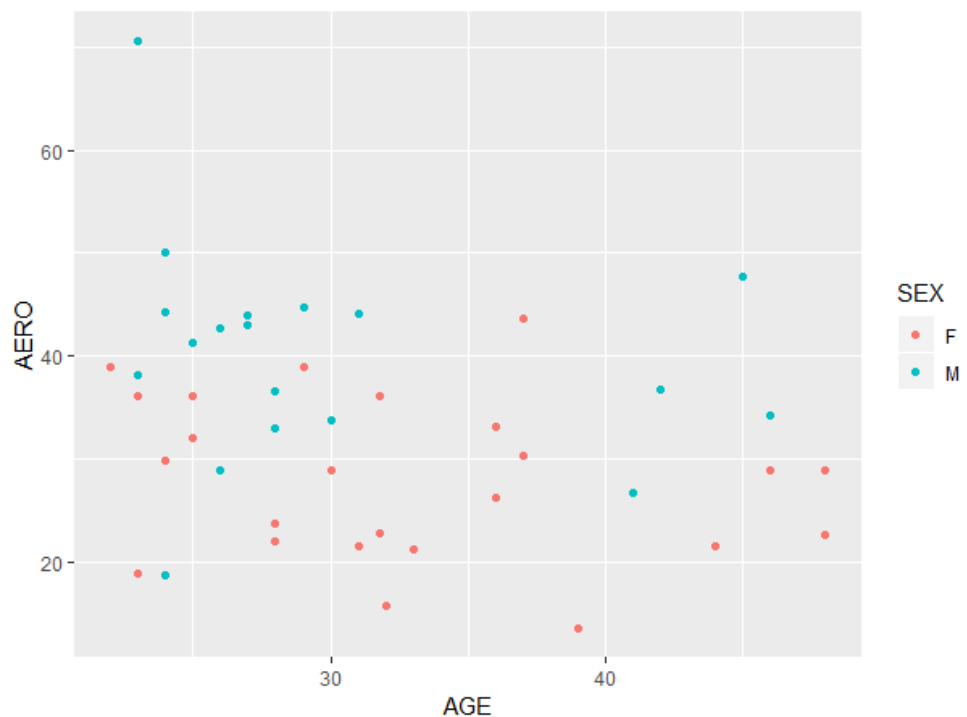
Archivo de datos `fitness.Rda`

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
33.00	Yang	28.00	M	96.00	1.00
42.7	Yang	26.00	M	74.00	2.00
36.1	Yang	NA	F	66.00	4.00
22.6	Yang	48.00	F	72.00	2.00
44.1	Yang	31.00	M	60.00	3.00
22.1	Reed	28.00	F	84.00	2.00
21.3	Reed	33.00	F	56.00	4.00
30.3	Reed	37.00	F	78.00	2.00
34.2	Reed	46.00	M	84.00	1.00
38.1	Reed	23.00	M	72.00	2.00
32.00	Reed	25.00	F	88.00	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
33.00	Yang	28.00	M	96.00	1.00
42.7	Yang	26.00	M	74.00	2.00
36.1	Yang	NA	F	66.00	4.00
22.6	Yang	48.00	F	72.00	2.00
44.1	Yang	31.00	M	60.00	3.00
22.1	Reed	28.00	F	84.00	2.00
21.3	Reed	33.00	F	56.00	4.00
30.3	Reed	37.00	F	78.00	2.00
34.2	Reed	46.00	M	84.00	1.00
38.1	Reed	23.00	M	72.00	2.00
32.00	Reed	25.00	F	88.00	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 sería 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 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 igual que en el 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 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)

Con el archivo **ozono.Rda**, construir una red neuronal con 5 nodos y comparar con regresión 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

Realizar la comparación bajo el esquema

- a) Training-test
- b) Training-test repetido

Ejercicio (2)

Con el archivo **autompg.Rda**,

1) Construir una red neuronal con 5 nodos y comparar con regresión 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 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 más capas ocultas que hace por ejemplo 5 años, pero sobre todo en los problemas a los que se suele aplicar Deep Learning (proceso de imágenes, texto, sonido, etc.), 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

- *"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.

- Número de nodos +

Pocas
observaciones

Muchas
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** sobre datos test, utilizando validación cruzada, etc. 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

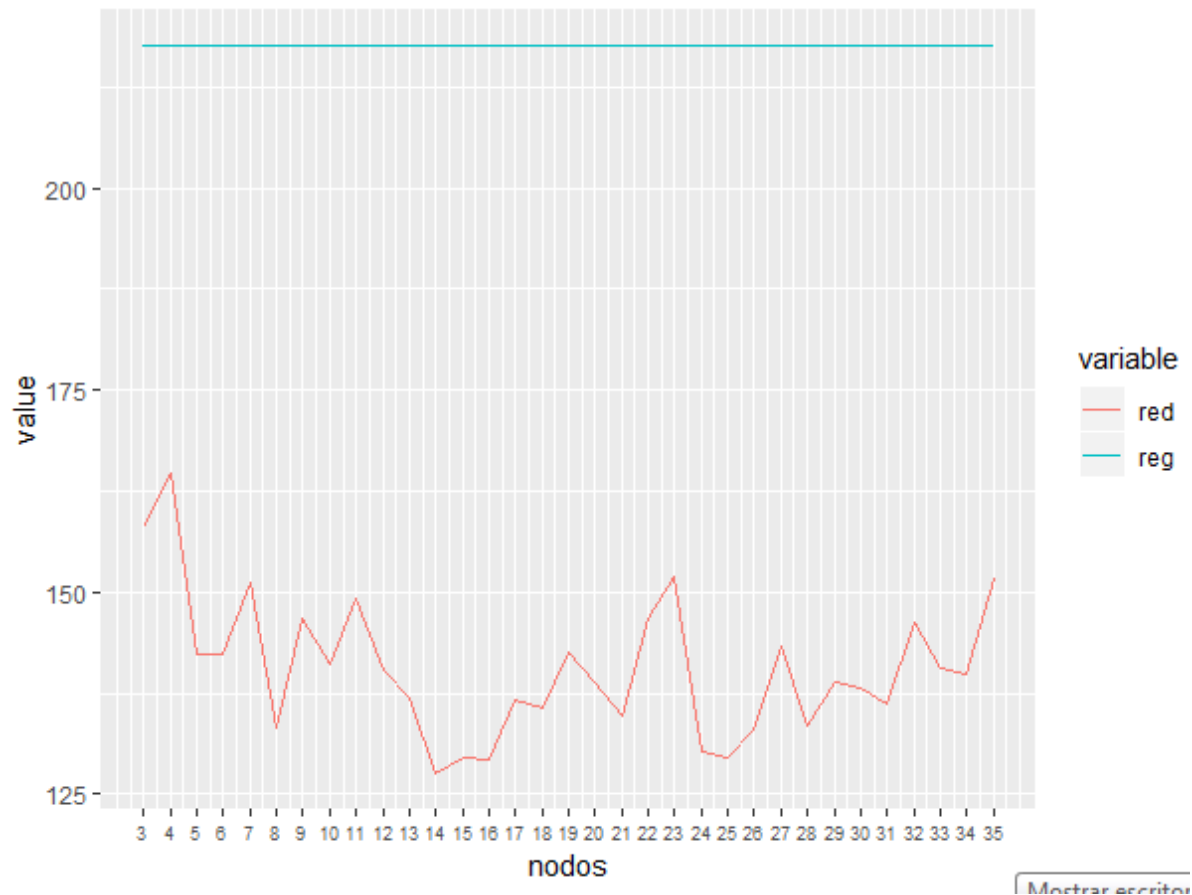
Es bueno hacer una reflexión a priori para desarrollar nuestra intuición:

- 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).

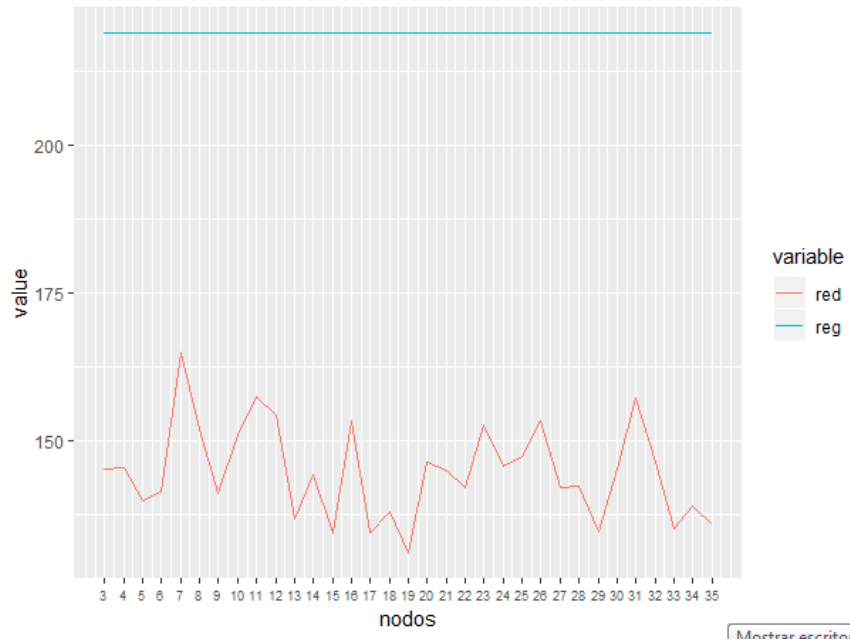
Se puede ir cambiando manualmente en el programa básico el número de nodos, pero técnicamente lo mejor es programar un bucle (bucle nodos 1.R).

Tomamos como referencia el bucle sobre la semilla train-test (lo vemos en Rstudio). 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.

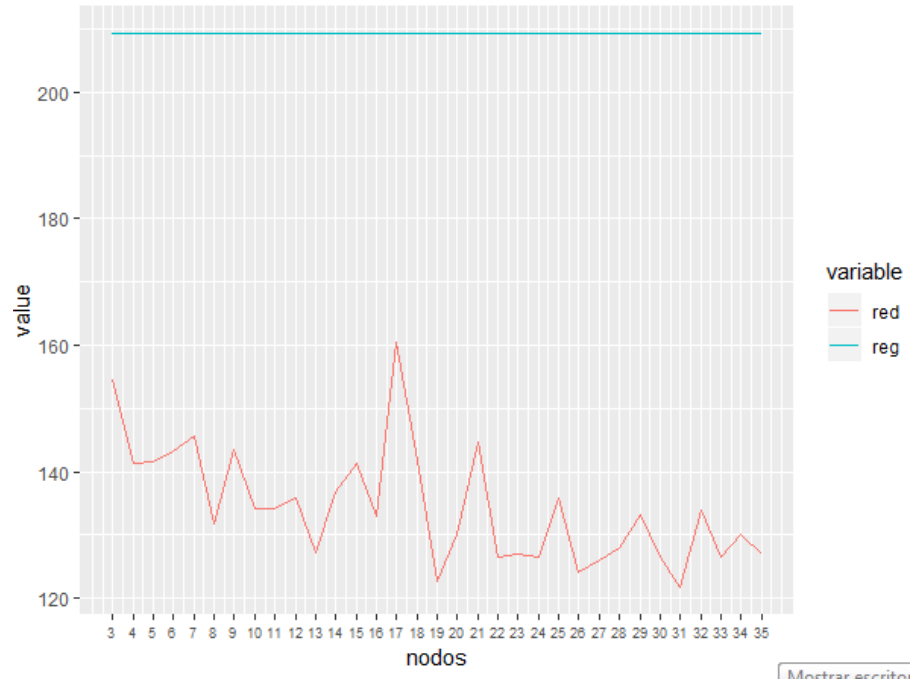
Semilla train-test 12345: 14 nodos parece bien



Semilla train-test 12346: 19 nodos



Semilla train-test 12347: 31 nodos



Obviamente hay variabilidad dependiendo del reparto train-test

Parece que necesita un cierto número de nodos para ajustar la curva.

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:

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).

Conceptos básicos sobre optimización

Objetivo del algoritmo de optimización

Minimizar la función de error o función objetivo:

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

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

$$\begin{aligned} \hat{y}_i = & \tanh(W_{1,\text{out}}(\tanh(W_{11}X_1 + W_{21}X_2 + W_{31}X_3 + W_{41}X_4 + b_1))) \\ & + W_{2,\text{out}}(\tanh(W_{12}X_1 + W_{22}X_2 + W_{32}X_3 + W_{42}X_4 + b_2))) + \\ & + W_{3,\text{out}}(\tanh(W_{13}X_1 + W_{23}X_2 + W_{33}X_3 + W_{43}X_4 + b_3))) + \\ & + W_{4,\text{out}}(\tanh(W_{14}X_1 + W_{24}X_2 + W_{34}X_3 + W_{44}X_4 + b_4))) + \\ & + b_{\text{out}}) \end{aligned}$$

Y habrá que hallar los parámetros (que llamaremos pesos o weights) $W_{1,\text{out}}, W_{11}, W_{21}, \dots$, etc. que minimicen el MSE. Es decir, se trata de minimizar MSE como función de los parámetros $W_{1,\text{out}}, W_{11}, W_{21}, \dots$, etc.

Es decir

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

- 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, 5000 términos).
- La complejidad y dificultad del proceso de optimización viene dada por el número de parámetros a estimar.

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

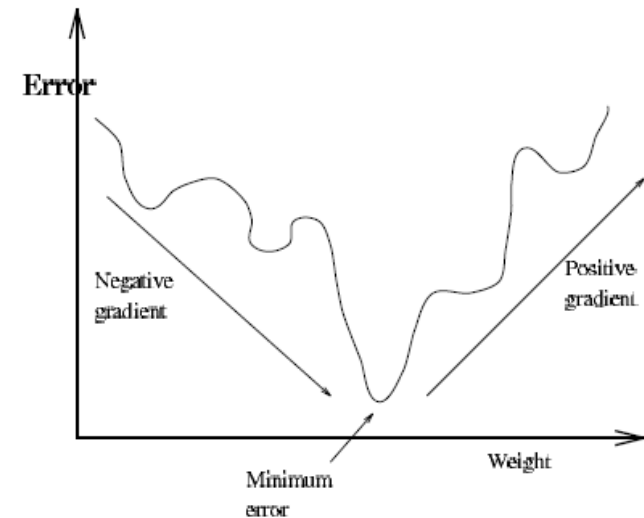
2) Se calcula la derivada de la función del error en ese punto (esos pesos concretos w_{ij}): $\frac{\partial E(n)}{\partial w_{ij}}$

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:

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

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).

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.

En un ejemplo anterior, la salida del nnet ofrece los valores finales de los pesos y el proceso de optimización:

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

Todo lo que va al nodo 1:

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

Esto significa:

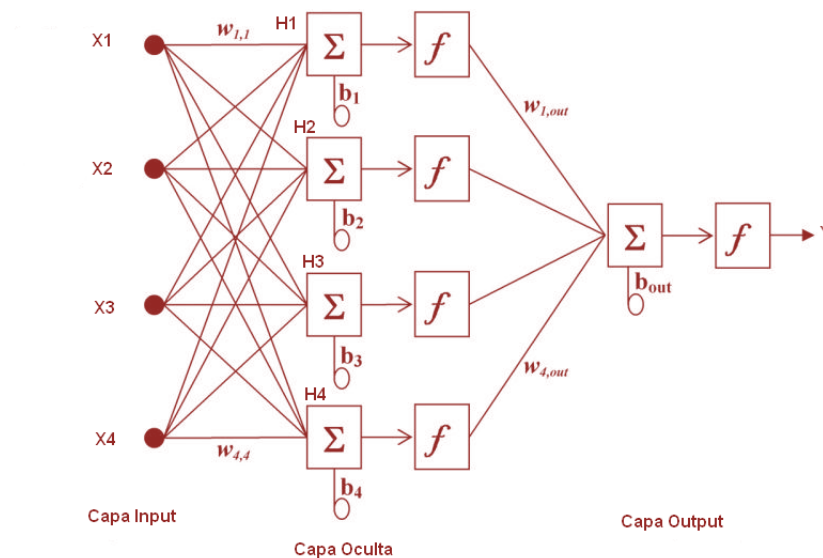
$h1 = w_{11} * age + w_{21} * water + b_1$
 $h1 = 3.31 * age + 0.13 * water + 3.05$

luego activación:

$h1 = \tanh(h1)$

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
```



Paréntesis técnico

Comparación de modelos

Medidas básicas de comparación de modelos

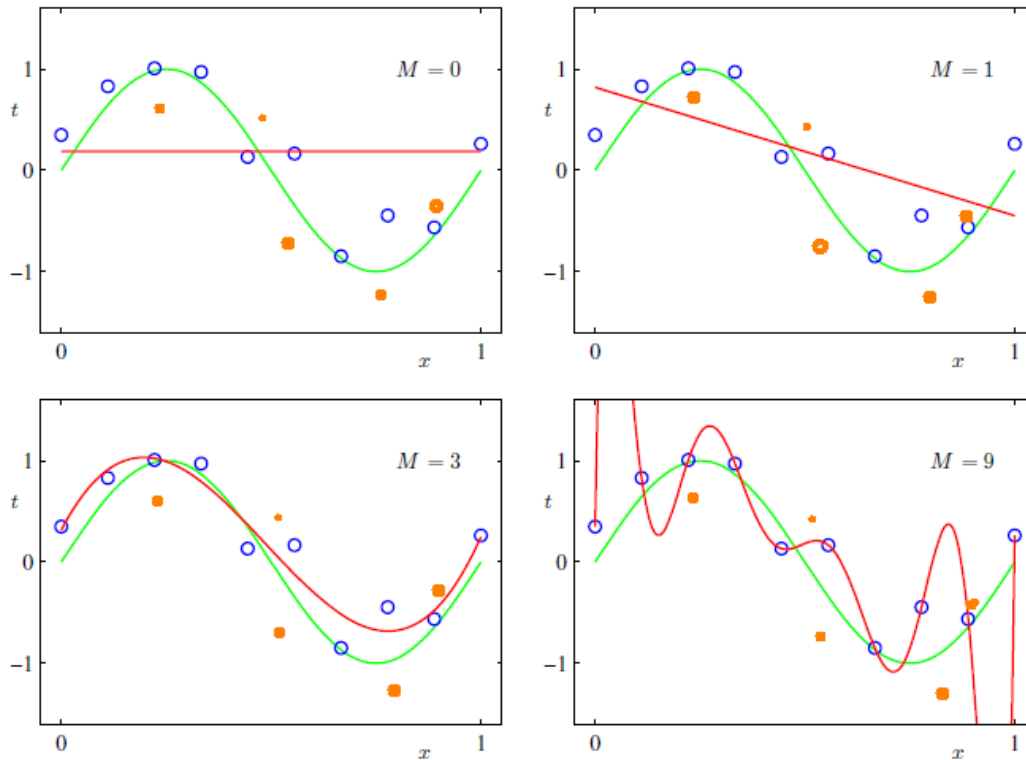
MODEL Option or Statistic	Definition or Formula
n	the number of observations
p	the number of parameters including the intercept
$\hat{\sigma}^2$	the estimate of pure error variance from fitting the full model
SST	the total sum of squares corrected for the mean for the dependent variable
SSE	the error sum of squares
ASE	$\frac{SSE}{n}$
MSE	$\frac{SSE}{n - p}$
R^2	$1 - \frac{SSE}{SST}$
ADJRSQ	$1 - \frac{(n - 1)(1 - R^2)}{n - p}$
AIC	$n \ln \left(\frac{SSE}{n} \right) + 2p$
AICC	$1 + \ln \left(\frac{SSE}{n} \right) + \frac{2(p + 1)}{n - p - 2}$
BIC	$n \ln \left(\frac{SSE}{n} \right) + 2(p + 2)q - 2q^n$ where $q = \frac{n \hat{\sigma}^2}{SSE}$
CP (C_p)	$\frac{SSE}{\hat{\sigma}^2} + 2p - n$
PRESS	$\sum_{i=1}^n \frac{r_i^2}{(1 - h_i)^2}$ where r_i = the residual at observation i and h_i = the leverage of observation $i = \mathbf{x}_i(\mathbf{X}'\mathbf{X})^{-1}\mathbf{x}_i'$
RMSE	\sqrt{MSE}
BIC	SBC $n \ln \left(\frac{SSE}{n} \right) + p \ln(n)$

A menudo se utiliza la misma fórmula que ASE

¿Son suficientes las medidas de penalización (AIC, AICC, BIC, Cp, etc. ?

En general, estas medidas no funcionan mal para **selección de variables dentro de un mismo tipo de modelo**, pero hay opciones más apropiadas para determinar qué modelo de entre varios modelos de diferente tipo (redes, regresión, árboles, etc.) funcionará mejor para **datos nuevos**.

Definición 1: Sobreajuste (Overfitting)=cuando un modelo está demasiado ajustado a los datos utilizados para su construcción, y funciona relativamente mal para **nuevos datos**. Se da este fenómeno cuando el modelo es **excesivamente complejo** o con un **número de parámetros excesivo**.



Nuevos datos en naranja

El modelo verde es bueno

$M=0$, $M=1$: El modelo **rojo** es insuficientemente complejo. **La predicción es muy pobre para los datos utilizados y para nuevos datos.**

$M=3$: El modelo **rojo** es adecuado (recordar la multiplicidad de modelos: no existe un único modelo correcto). **La predicción para nuevos datos es correcta.**

$M=9$: El modelo **rojo** es excesivamente complejo: **está sobreajustado. La predicción para nuevos datos será muy pobre.**

Definición 2: Capacidad de generalización, eficacia predictiva=Facultad de un modelo de predecir bien nuevos datos tomados de la misma población que los datos training. Estos conceptos deben tenerse en cuenta paralelamente o por encima, de los clásicos conceptos de ajuste.

Métodos más habituales para medir o comparar la capacidad predictiva de uno o varios modelos

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
- 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).

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]*

(Fin del paréntesis técnico)

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)

Archivo de código ejemplos caret compress.R

```
# *****  
# 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,savePredictions = "all")
```

(aquí number es el número de grupos de validación cruzada y repeats el número de repeticiones)

Esquema 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.

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.

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.

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)
```

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

```
reg1
```

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

Paréntesis técnico:

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

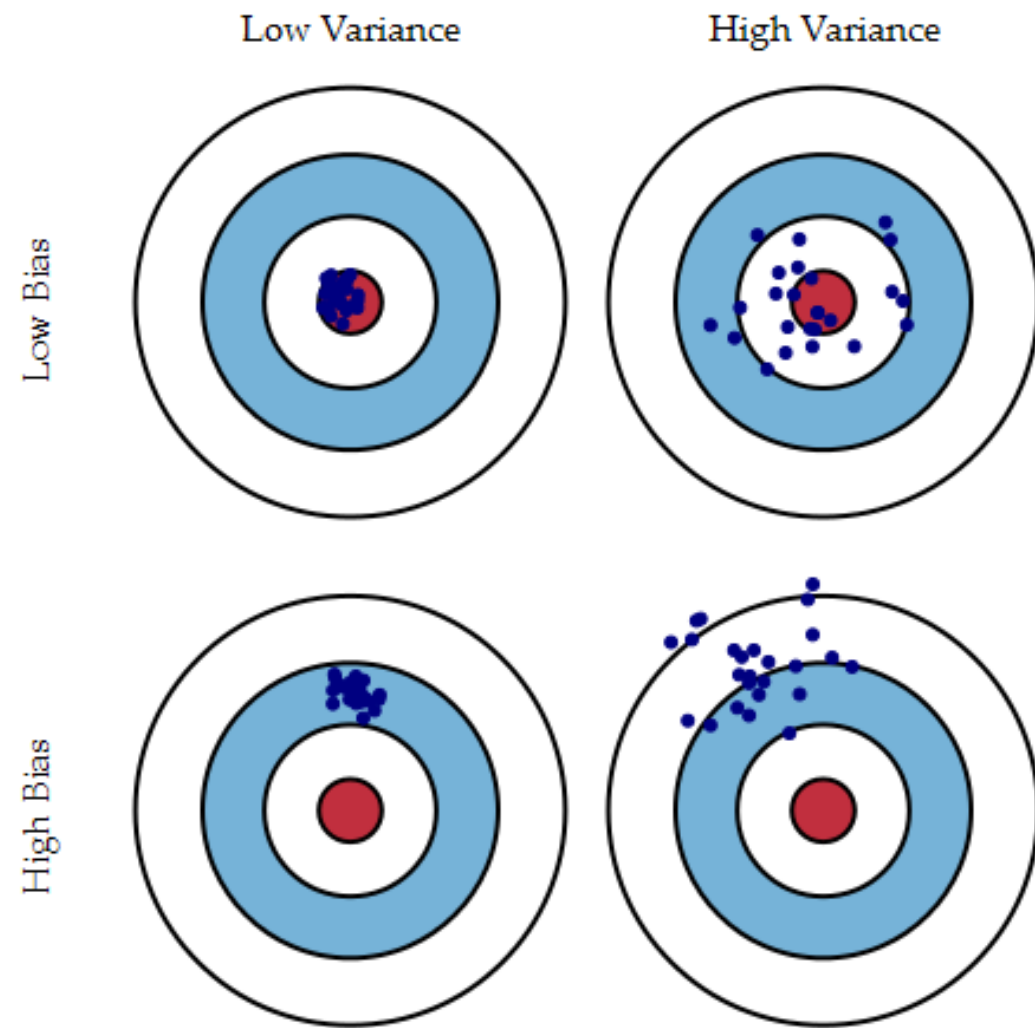


Fig. 1 Graphical illustration of bias and variance.

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

- El error “irreducible” o no explicable
- 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 construirlo.

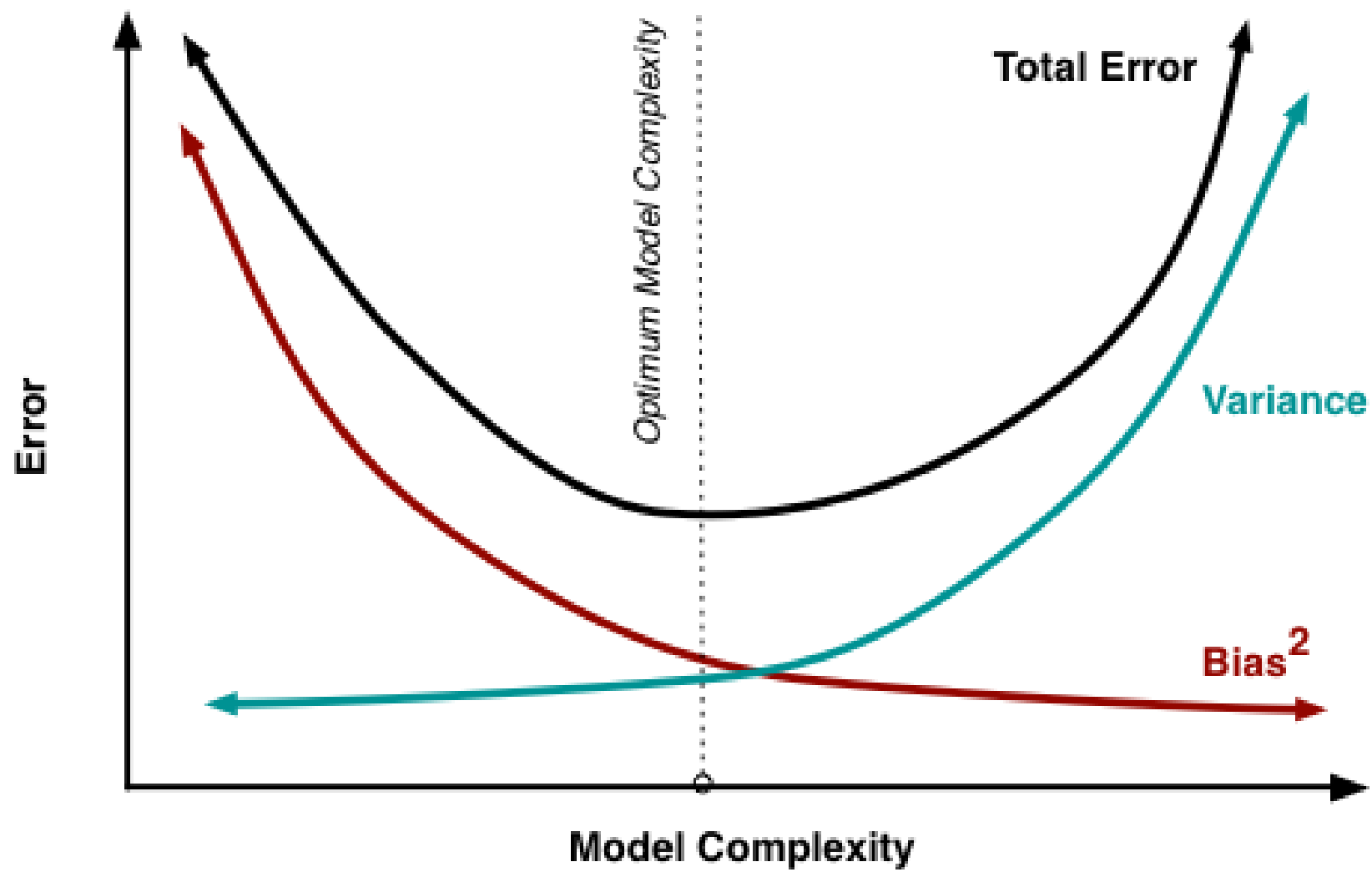
We may estimate a model $\hat{f}(X)$ of $f(X)$ using linear regressions or another modeling technique. In this case, the expected squared prediction error at a point x is:

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

This error may then be decomposed into bias and variance components:

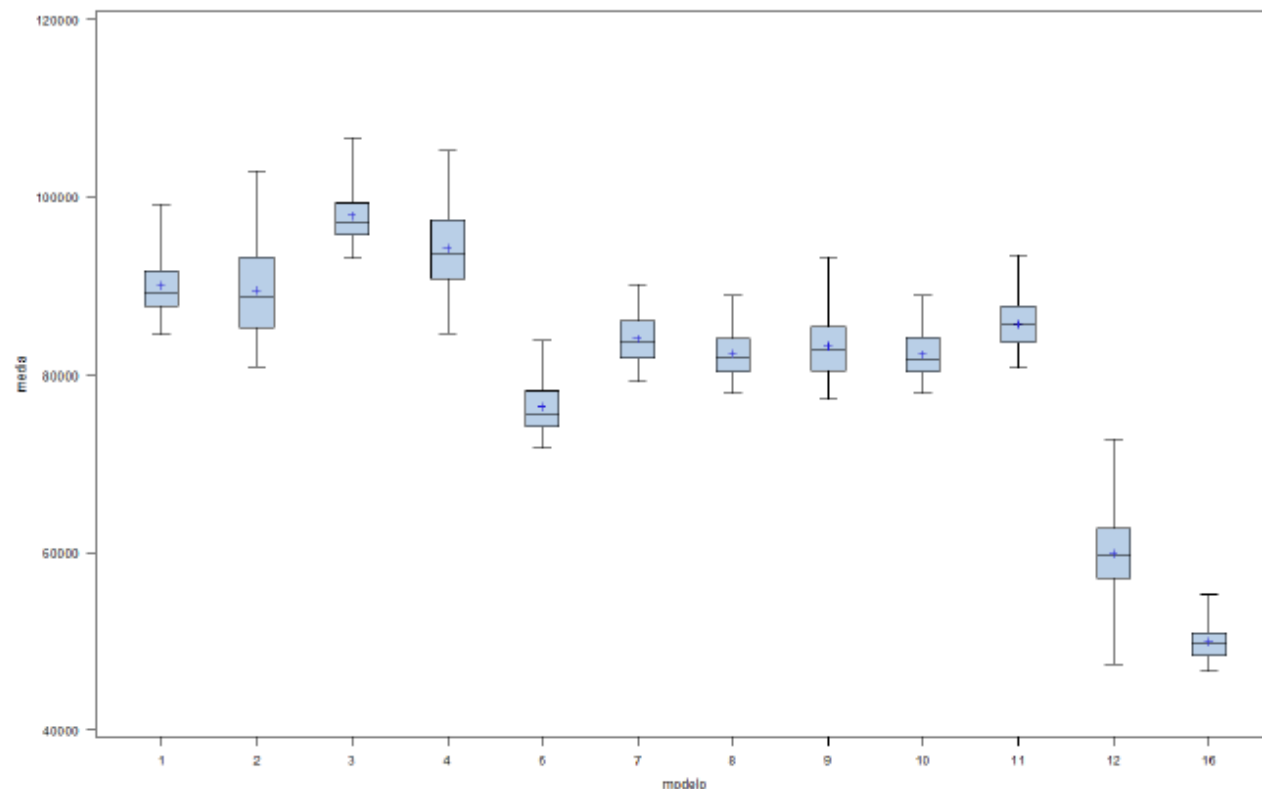
$$Err(x) = \left(E[\hat{f}(x)] - f(x)\right)^2 + E\left[\hat{f}(x) - E[\hat{f}(x)]\right]^2 + \sigma_e^2$$

$$Err(x) = \text{Bias}^2 + \text{Variance} + \text{Irreducible Error}$$



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 (altura de la caja). Yo utilizo 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. El modelo 16 tiene bajo sesgo y baja varianza: parece óptimo.



		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	-	+

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, 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

He creado dos funciones en R:

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

Archivos de código [cruzada avnnet](#) y [lin.R](#), [cruzada avnnet 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  
data<-compressbien
```

```
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="avnnet"
```

```
## Ejemplo de utilización cruzada regresión lineal
```

```
data<-compressbien
```

```
medias2<-cruzadalin(data=data,  
vardep="cstrength",listconti=c("age","water"),  
listclass=c(""),grupos=4,sinicio=1234,repe=5)
```

```
medias2$modelo="lineal"
```

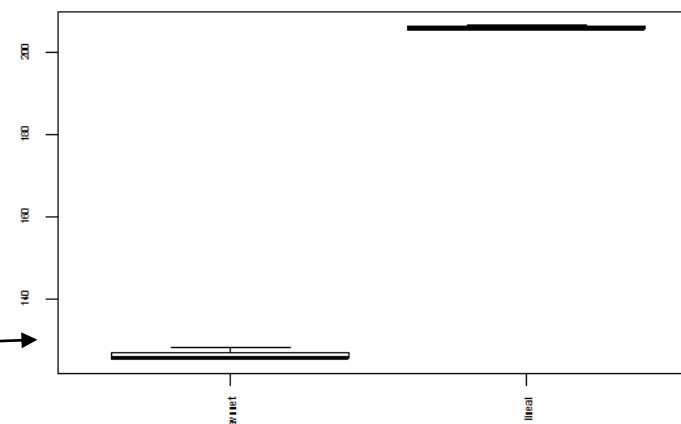
```
## Unión de las medias y presentación del boxplot
```

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

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

```
boxplot(data=union1,error~modelo)
```

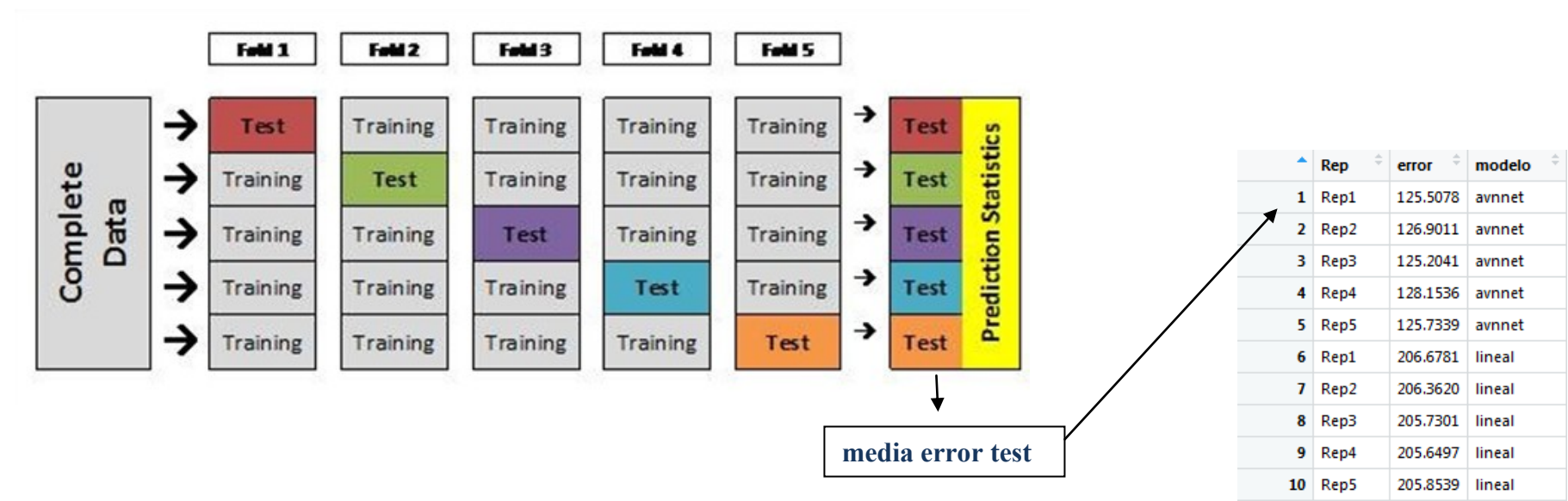
	Rep	error	modelo
1	Rep1	125.5078	avnnet
2	Rep2	126.9011	avnnet
3	Rep3	125.2041	avnnet
4	Rep4	128.1536	avnnet
5	Rep5	125.7339	avnnet
6	Rep1	206.6781	lineal
7	Rep2	206.3620	lineal
8	Rep3	205.7301	lineal
9	Rep4	205.6497	lineal
10	Rep5	205.8539	lineal



Recordatorio

Cada repetición de validación cruzada se realiza reordenando aleatoriamente el archivo con una semilla diferente.

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)
```

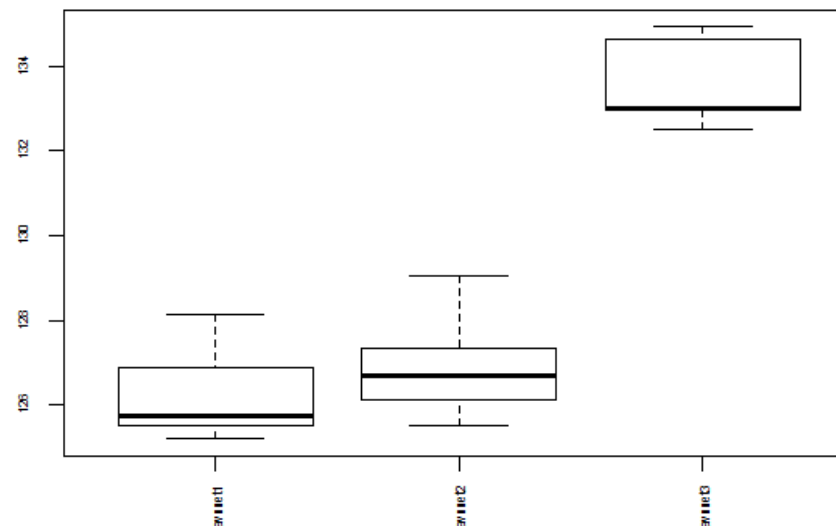
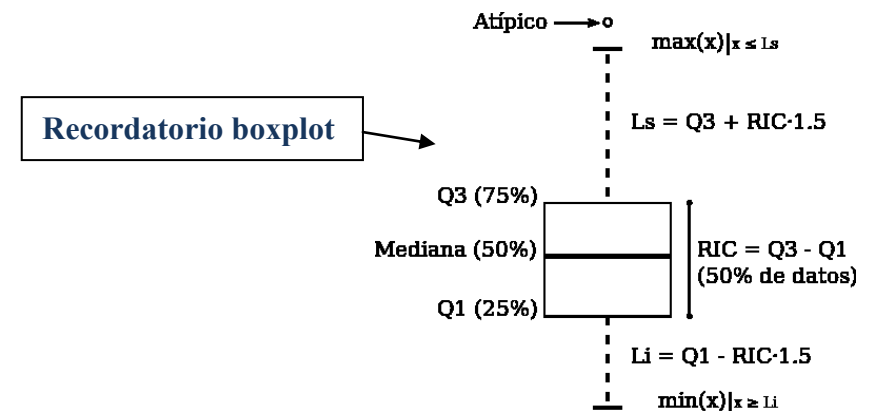
```
medias2$modelo="avnnnet2"
```

```
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)
```



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

Ambas funciones 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)
```

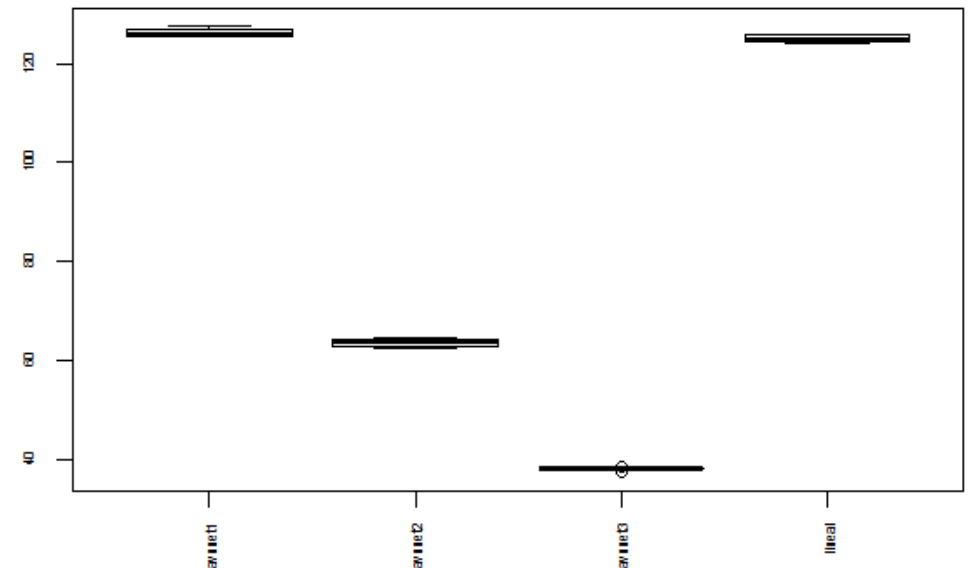
```
medias3$modelo="avnnet3"
```

```
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):

Con el modelo con las variables input `c("age", "water", "cement", "blast")`

1) Tunear con caret bajo validación cruzada repetida, para determinar nodos y weight decay.

2) Con los diferentes modelos que parezcan interesantes, utilizar la función `cruzadaavnnnet` para comparar gráficamente.

Comparar siempre con regresión.

Ejercicio (2):

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

Ejercicio (3) avanzado opcional: crear una función que haga un bucle sobre los nodos en la función `cruzadaavnnnet`, vaya acumulando el archivo de errores y presente finalmente las cajas para cada modelo de red con diferentes nodos.

Selección de variables en redes

El problema de la selección de variables en redes neuronales

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

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 tiene 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 2^k 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.

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 (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.

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

El paquete NeuralNetTools de R calcula la medida de importancia.

Ejemplo

Archivo `ejemplo importancia variables olden garson.R`

```
library(NeuralNetTools)
library(caret)

control<-trainControl(method = "none")

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

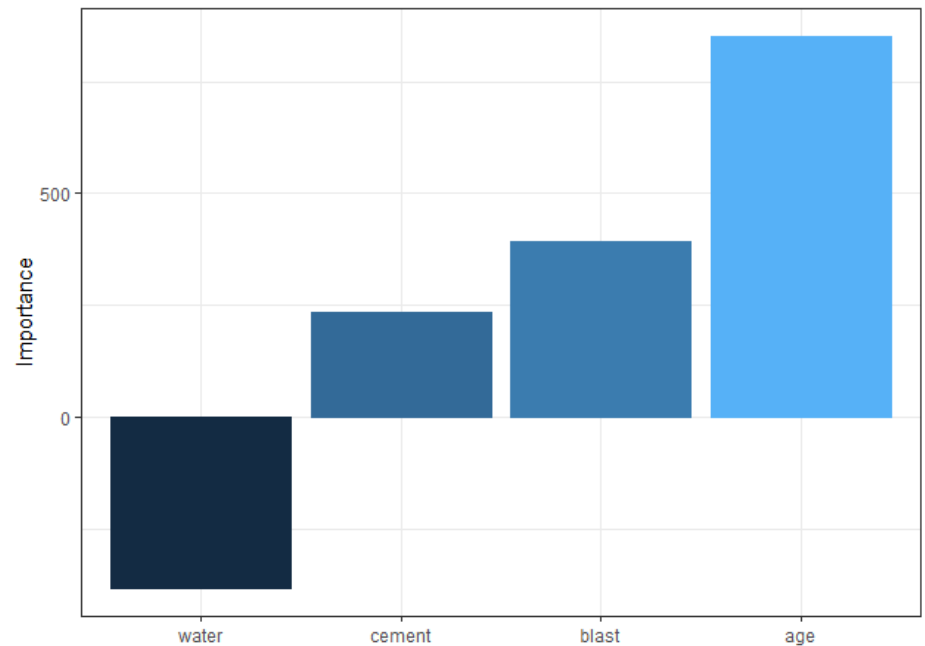
red1 <- train(cstrength~ age + water+cement+blast, method = 'nnet',
data = compressbien,tuneGrid=nnetgrid,linout = TRUE,trControl=control)

olden(red1)

importancia<-olden(red1,bar_plot=FALSE)

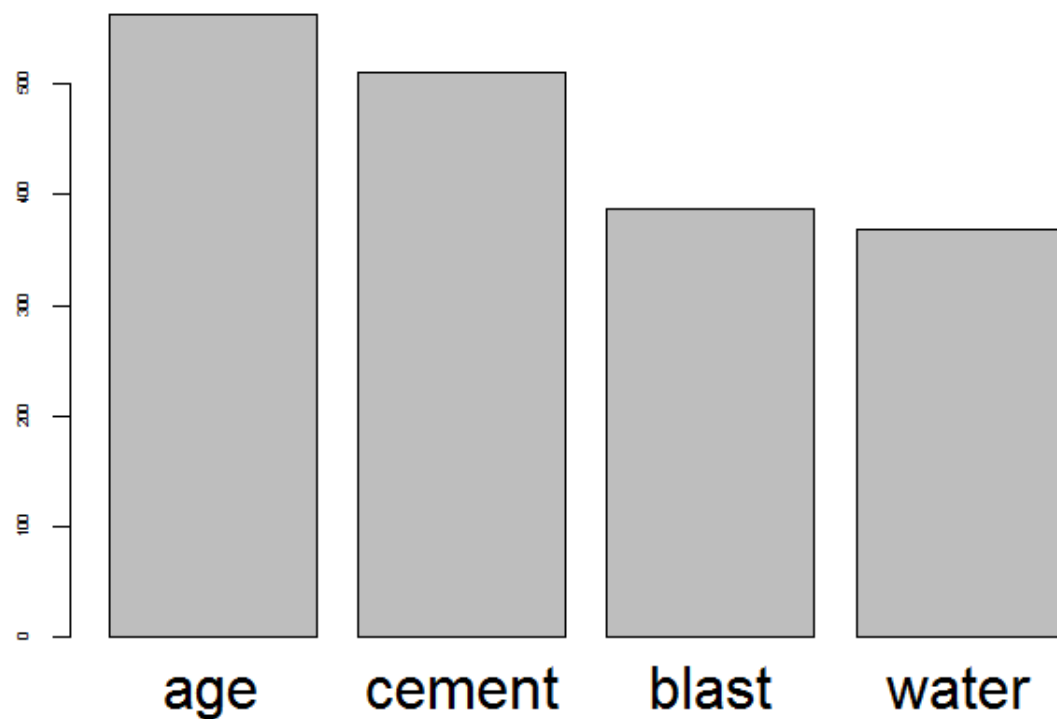
impor<-importancia[order(-importancia$importance),,drop = FALSE]
```

	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(select$coefficients))
```

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

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 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]])
```

	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")

```

```
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

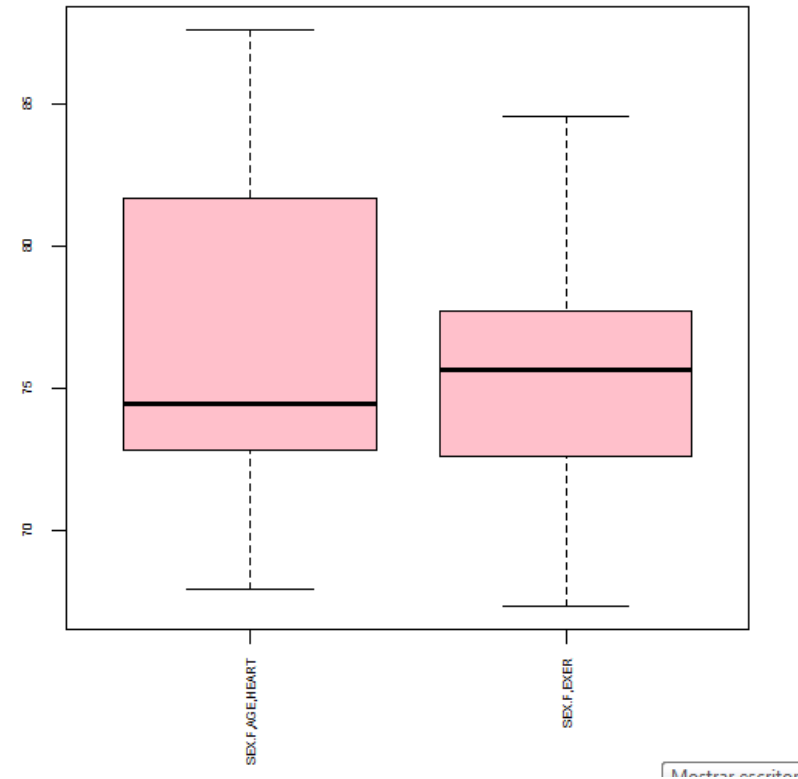
```
lista<-steprepetido(data=fitnessbien,vardep=vardep,
listconti=listconti,sinicio=12345,sfinal=12385,porcen=0.8,criterio="BIC")
tabla<-lista[[1]]
```

	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

Tras explorar 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)
```



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.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<-autompg[,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
```

```

# 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"),
inicio=12345,sfinal=12385,porcen=0.8,criterio="AIC")

lista<-steprepetido(data=data,vardep=c("mpg"),
listconti=
c("cylinders", "displacement", "horsepower", "weight", "acceleration",
"modelyear"),
inicio=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
```

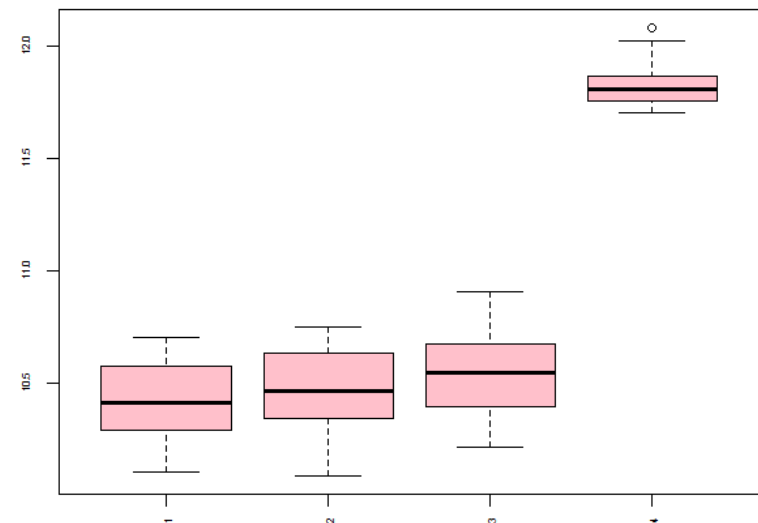
```
medias3<-cruzadalin(data=auto3,
vardep="mpg",listconti=c("weight",
"modelyear.73","modelyear.77","modelyear.78",
"modelyear.79", "modelyear.80",
"modelyear.81", "modelyear.82"),
listclass=c(""),grupos=4,sinicio=1234,repe=30)
```

```
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,sinicio=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,sinicio=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)
```

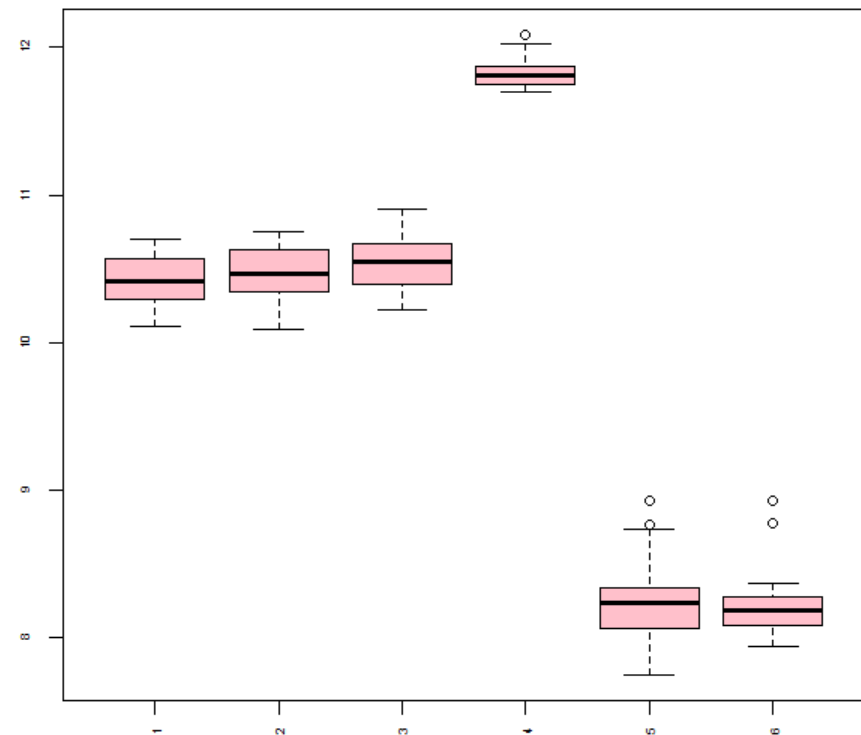
```
par(cex.axis=0.5)  
boxplot(data=union1,col="pink",error~modelo)
```

```
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).



¿Qué hacer desde el punto de vista práctico?

Los gráficos son una manera rápida de aproximarse al problema, pero nos puede engañar la escala: 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, 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 (según el tiempo disponible):

1) Un ejemplo más complicado: Ameshousing
(Ameshousing 5.0.R)

2) Ozono. Variable dependiente ozone. Realizar todo el proceso.
(Ozone.Rda)

- 1) Observar el archivo y verificar missing y categóricas
- 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

3) Ejemplo con California Housing:
(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

Conceptos básicos

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 $0, 1, 2, \dots$

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.

Inicialmente se estudiará 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

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 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, 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

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
 $=VP/(VP+FN)=0.65$

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

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)

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

CURVA ROC

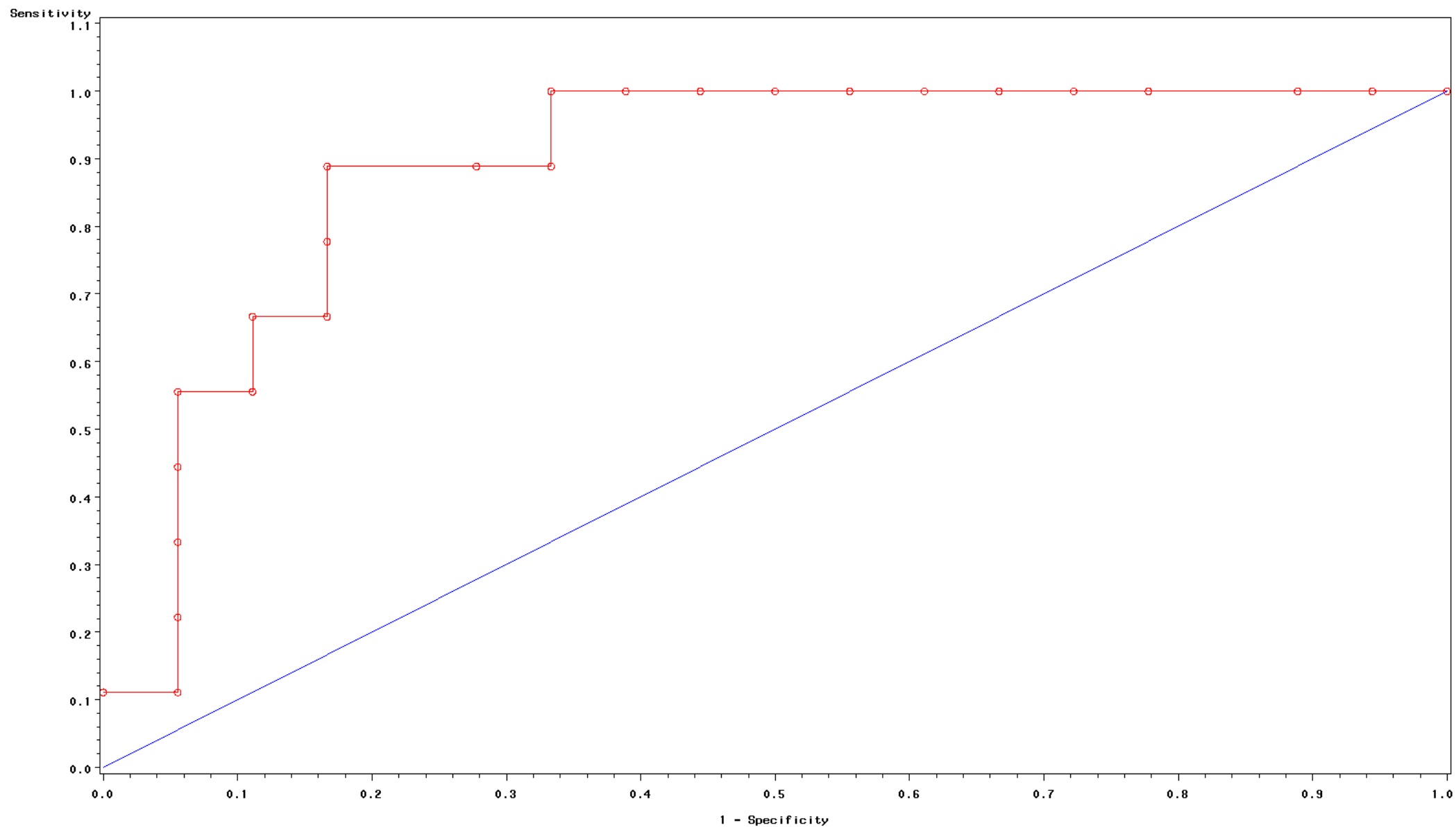
(Receiver Operating Characteristic Curve)

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.

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 $(0,1)$.

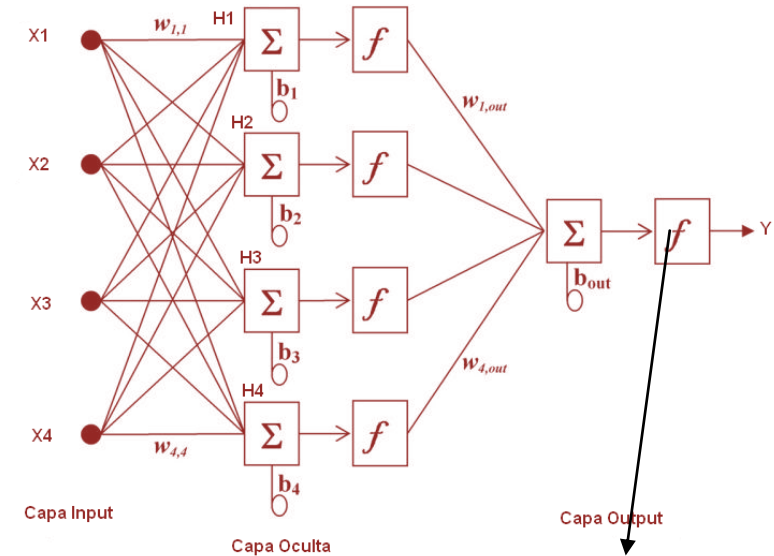
Ejemplo básico

Ejemplo saheart.R

chd variable dependiente

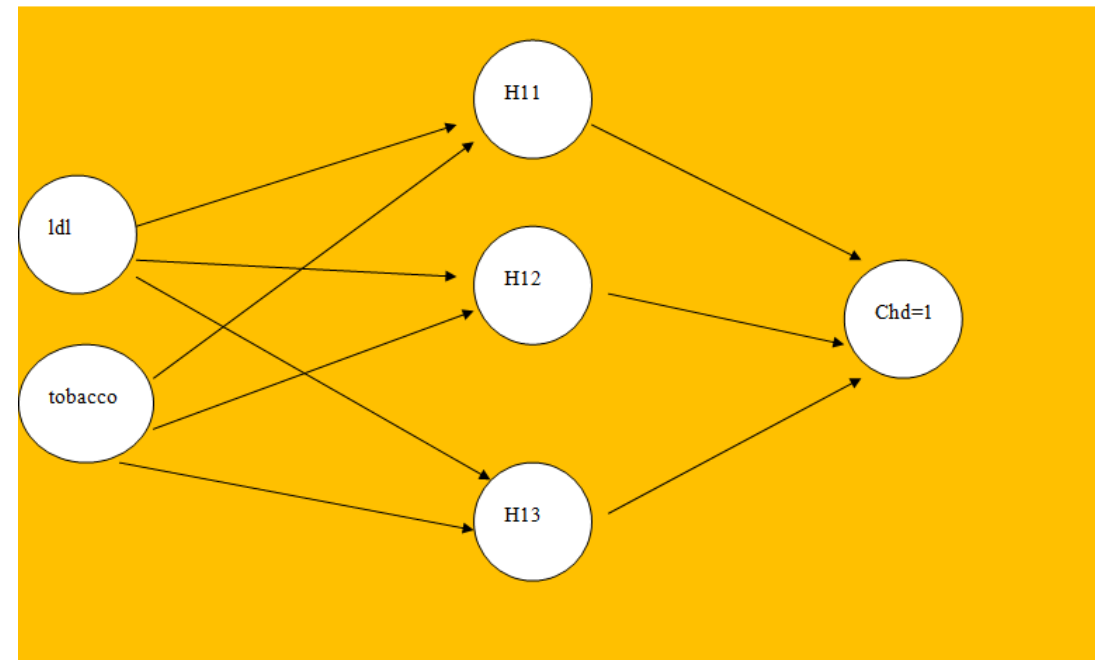


	chd	sbp	tobacco	ldl	adiposity	famhist	typea	obesity	alcohol	age
1	1	160	12.00	5.73	23.11	Present	49	25.30	97.20	52
2	1	144	0.01	4.41	28.61	Absent	55	28.87	2.06	63
3	0	118	0.08	3.48	32.28	Present	52	29.14	3.81	46
4	1	170	7.50	6.41	38.03	Present	51	31.99	24.26	58
5	1	134	13.60	3.50	27.78	Present	60	25.99	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



En variables output categóricas SÍ se aplica función de activación al nodo output para que estén en rango 0-1; en variables output continuas NO

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 dtipica 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,repeats=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
```

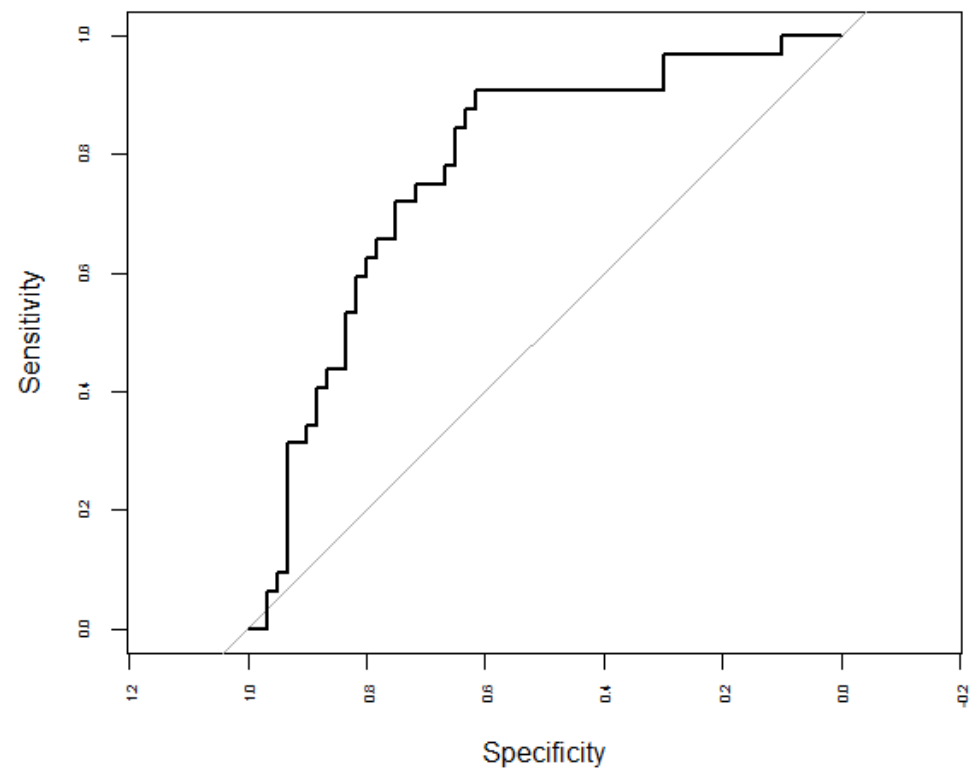
```

      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` 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)
# *****
avnnnetgrid <- 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=avnnnetgrid,
  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")
```

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

	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

	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

```
# *****
# APLICANDO cruzadalogistica a los modelos candidatos
# *****

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"

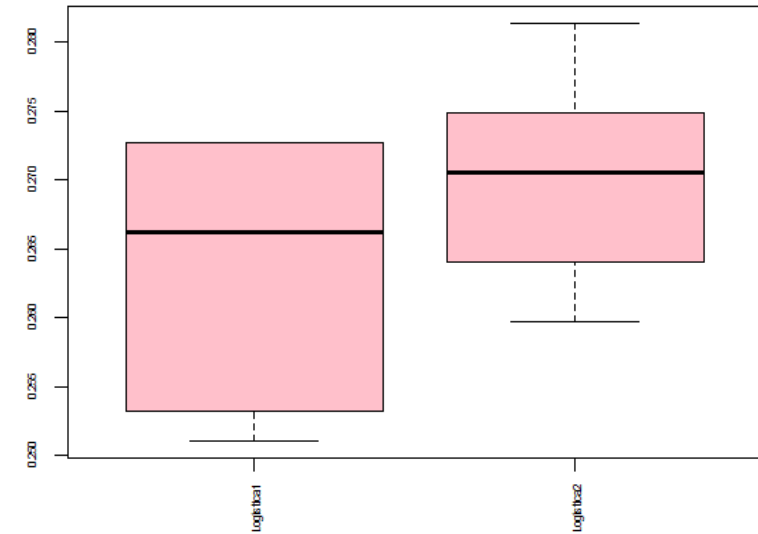
medias2<-cruzadalogistica(data=saheartbis,
  vardep="chd",listconti=c("age", "ldl", "famhist.Absent"),
  listclass=c(""), grupos=4,sinicio=1234, repe=5)

medias2$modelo="Logística2"

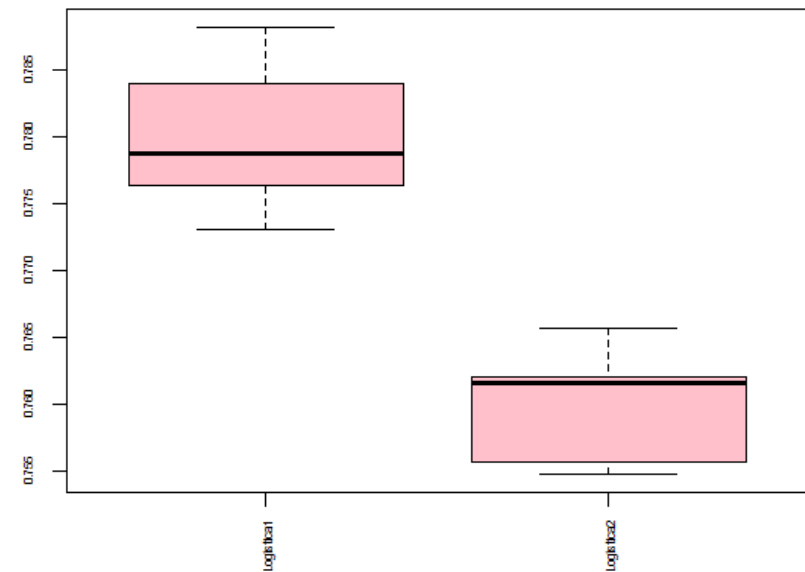
union1<-rbind(medias1,medias2)

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

TASA FALLOS



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,sinicio=1234,repe=5,
  size=c(5),decay=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,sinicio=1234,repe=5,
  size=c(10),decay=c(0.1),repeticiones=5,itera=200)
```

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

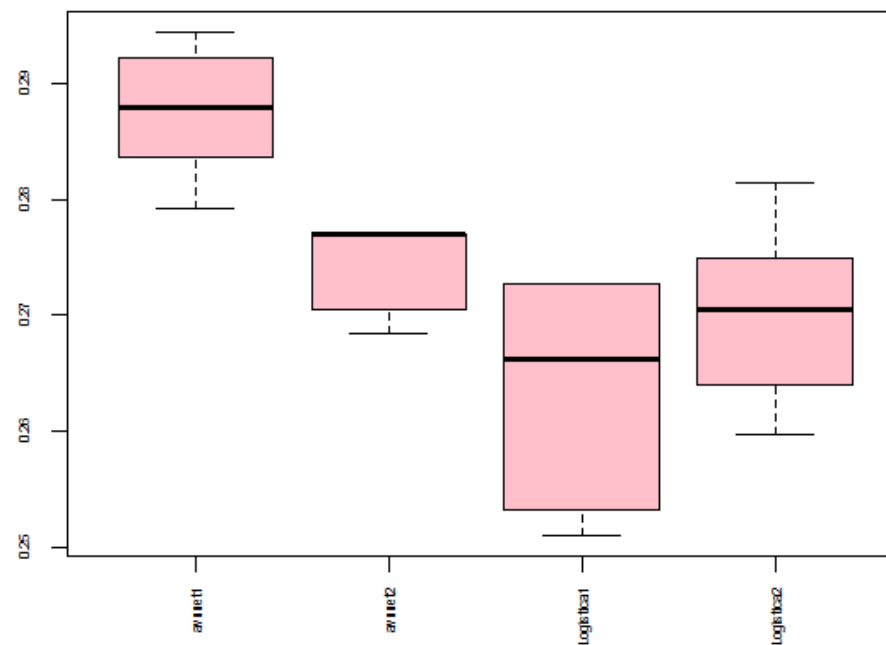
```
union1<-rbind(medias1,medias2,medias3,medias4)
```

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

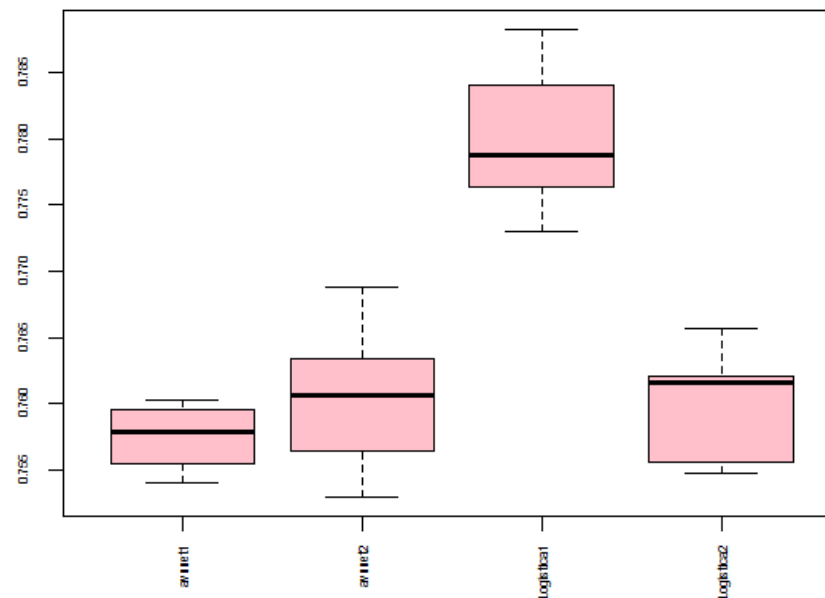
```
boxplot(data=union1,tasa~modelo,col="pink",main="TASA FALLOS")
```

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

TASA FALLOS



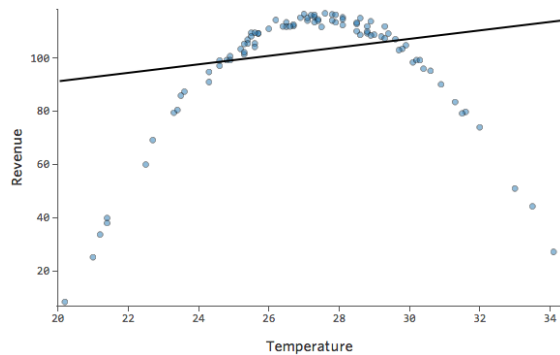
AUC



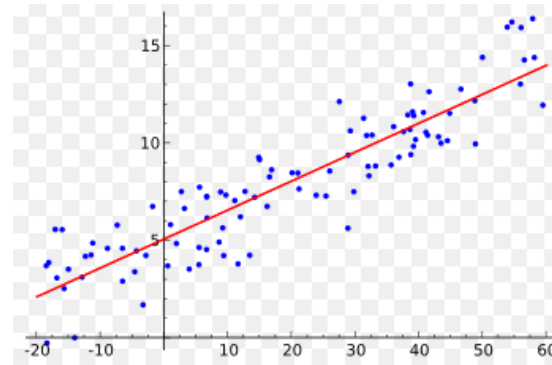
Nota: En clasificación binaria, si la separación entre clases es aproximadamente lineal, es mejor la regresión logística que la red.

Variable dependiente continua

Mejor la red

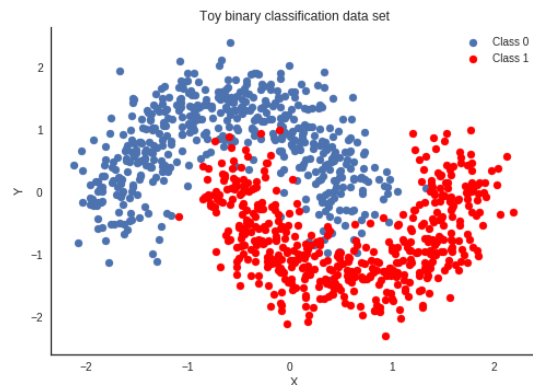


Mejor la regresión lineal

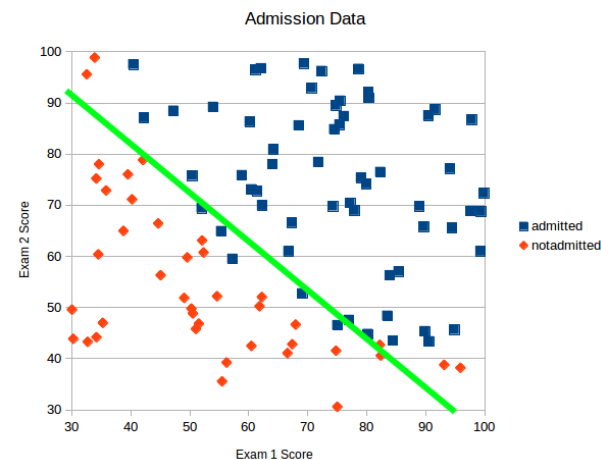


Variable dependiente binaria

Mejor la red



Mejor la regresión logística



Un Ejemplo más complicado: bank

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

Revisar el código `ejemplo bank.R`

Otros archivos de prueba

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))

Continuas

A2 A3 A7 A10 A13 A14

Cualitativas

A1 A4 A5 A6 A8 A9 A11 A12

Dependiente

A15