



Machine Learning

Redes Neuronales

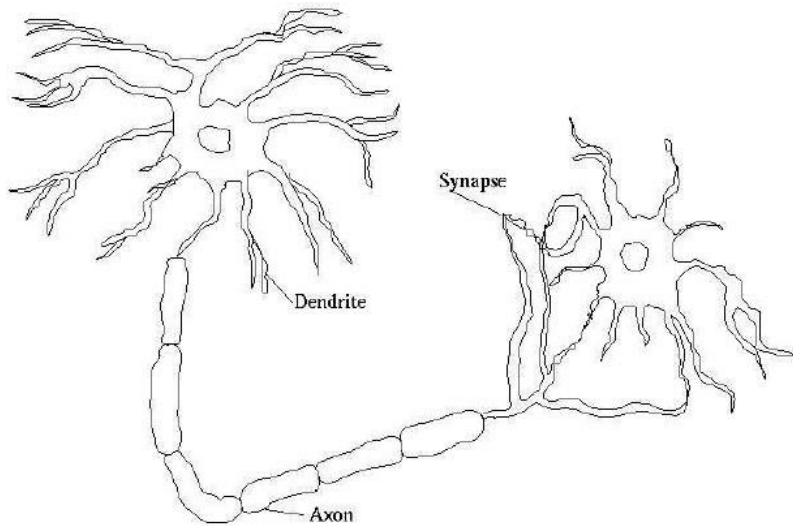
Inmaculada Gutiérrez García-Pardo

1. Introducción

En esta sección se presentarán el origen y definición de las redes neuronales artificiales (*Neural Networks NN*), su estructura más básica y aplicación en distintos sectores

Imagen de una Neurona

Las Redes Neuronales (**Neural Networks, NN**) fueron originalmente una simulación abstracta de los sistemas nerviosos biológicos. Su objetivo principal es desarrollar operaciones de síntesis y procesamiento de información.



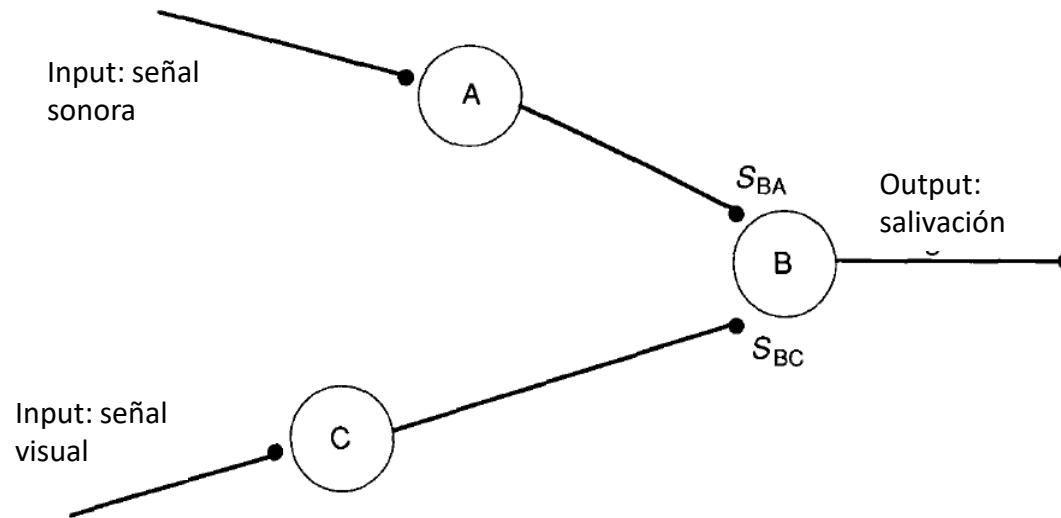
Las neuronas y las conexiones entre ellas (sinapsis) constituyen la clave para el procesamiento de la información.

La mayor parte de las neuronas poseen una estructura de árbol, llamada **dendritas**, que **reciben las señales** de entrada procedentes de otras neuronas a través de las **sinapsis**.

Una neurona consta de tres partes:

1. El cuerpo de la neurona,
2. Las dendritas, que reciben las entradas,
3. El axón, que lleva la salida de la neurona a las dendritas de otras neuronas por diferencias de potencial eléctrico.

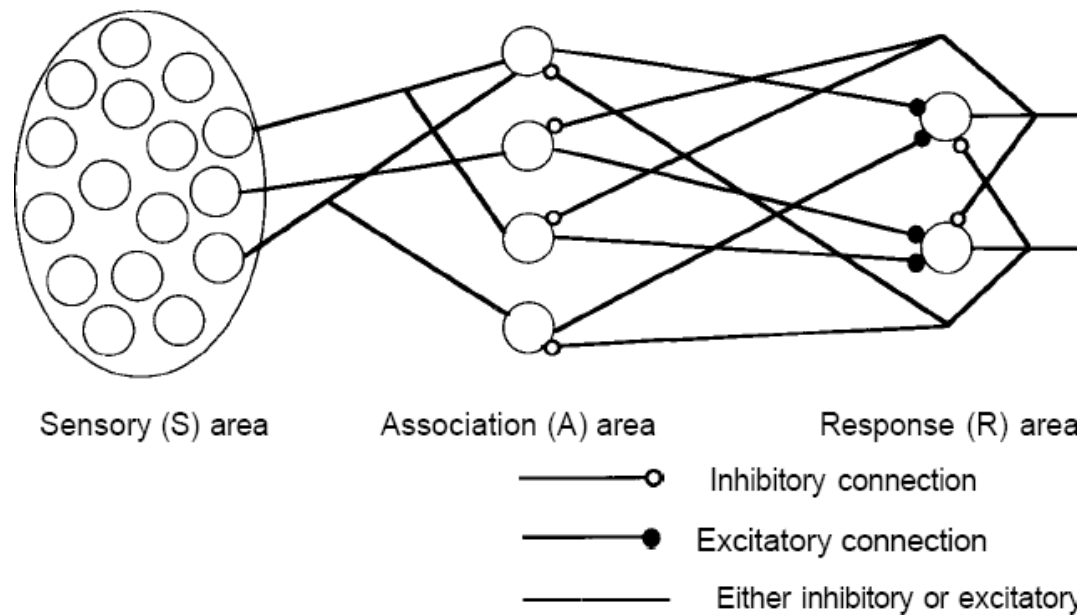
Input, Activación y Output



Las redes neuronales artificiales buscan hacer una simulación abstracta del sistema nervioso.

En este ejemplo hay dos neuronas que reciben la información o el estímulo desde fuera. La neurona A recibe un estímulo sonoro y la neurona C recibe un estímulo visual. Estas entradas a A y C se activan y envían la señal de activación a la neurona B, que efectúa la salida, en este caso salivación. Las dos uniones sinápticas se llaman S_{BA} y S_{BC} .

Representación en Red



La neurona recoge las señales por su sinapsis sumando todas las influencias **excitadoras** e **inhibidoras**. Si las influencias excitadoras positivas dominan, entonces la neurona produce una señal positiva y manda este mensaje a otras neuronas por sus sinapsis de salida.

Algunos números

¿Cuántas neuronas tiene un ser vivo?

Cuanto más neuronas y sinapsis tenga un ser vivo, más inteligente es

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.

A mayor número de neuronas ➡ sistema más inteligente

Red Neuronal Artificial. Origen

- **~1940**, W.McCulloch (neurofisiólogo) W.Pitts (matemático) desarrollaron un modelo matemático de una neurona: **neurona de McCulloch-Pitts**, capaz de recibir múltiples entradas binarias y producir una salida binaria, dependiendo de la suma de las entradas y los pesos asociados a ellas.
- **~1950**, F.Rosenblatt (psicólogo) desarrolló el **perceptrón**, red neuronal que aprende a clasificar datos linealmente separables. Primera red capaz de aprender automáticamente a partir de datos de entrenamiento.
- **~1960**, limitaciones de los patrones para aprender patrones complejos.
- **~1980**, algoritmos de retropropagación permiten el entrenamiento de redes neuronales de múltiples capas, lo que permitió que las redes neuronales aprendieran patrones más complejos.
- **HOY**, las redes neuronales artificiales se utilizan en una variedad de aplicaciones, incluyendo el reconocimiento de voz y de imagen, el procesamiento del lenguaje natural, la detección de fraude financiero, la conducción autónoma y muchas otras áreas.

Red Neuronal Artificial

¿Para qué se usan las redes neuronales? Para extraer información útil y producir inferencias a partir de los datos disponibles gracias a su capacidad de aprendizaje. Permiten realizar predicciones, clasificaciones o simplemente explicaciones de una variable respuesta o dependiente en función de otras variables independientes.

Mismo concepto general que en otros modelos predictivos más sencillos como regresión o series temporales. Dada una variable respuesta y una serie de variables independientes, poder predecir la variable respuesta (**regresión/clasificación**)

Dependiendo de la relación entre las variables y la complejidad de los datos, habrá que aplicar unas técnicas u otras.

Red Neuronal Artificial. Algunas nociones básicas

- ❑ Las neuronas artificiales se conectan entre sí para transmitirse señales.
- ❑ La información de entrada llega a una neurona. Mientras la atraviesa sufre distintas operaciones, obteniéndose un valor de salida.
- ❑ Las neuronas se conectan entre sí mediante **enlaces**.
- ❑ En los enlaces el valor de salida de la neurona anterior se multiplica por un **peso**.
- ❑ Estos pesos en los enlaces pueden **incrementar o inhibir** el estado de activación de las **neuronas adyacentes**.
- ❑ La conexión entre neuronas se lleva a cabo mediante **funciones de combinación**
- ❑ A la salida de la neurona, puede existir una **función limitadora o umbral** que modifica el valor resultado o impone un límite que no se debe sobrepasar antes de propagarse a otra neurona. Es la **función de activación**.
- ❑ En este módulo se trabajará con redes neuronales **supervisadas**: la red recibe pares de entrada y salida conocidos durante el entrenamiento y ajusta sus pesos para minimizar la diferencia entre las salidas predichas y las reales.
- ❑ Como parte del proceso de aprendizaje en redes neuronales supervisadas: **funciones de aprendizaje, pérdida o coste**, que evalúan la discrepancia entre las predicciones y las etiquetas durante el entrenamiento

Red Neuronal Artificial. Características principales

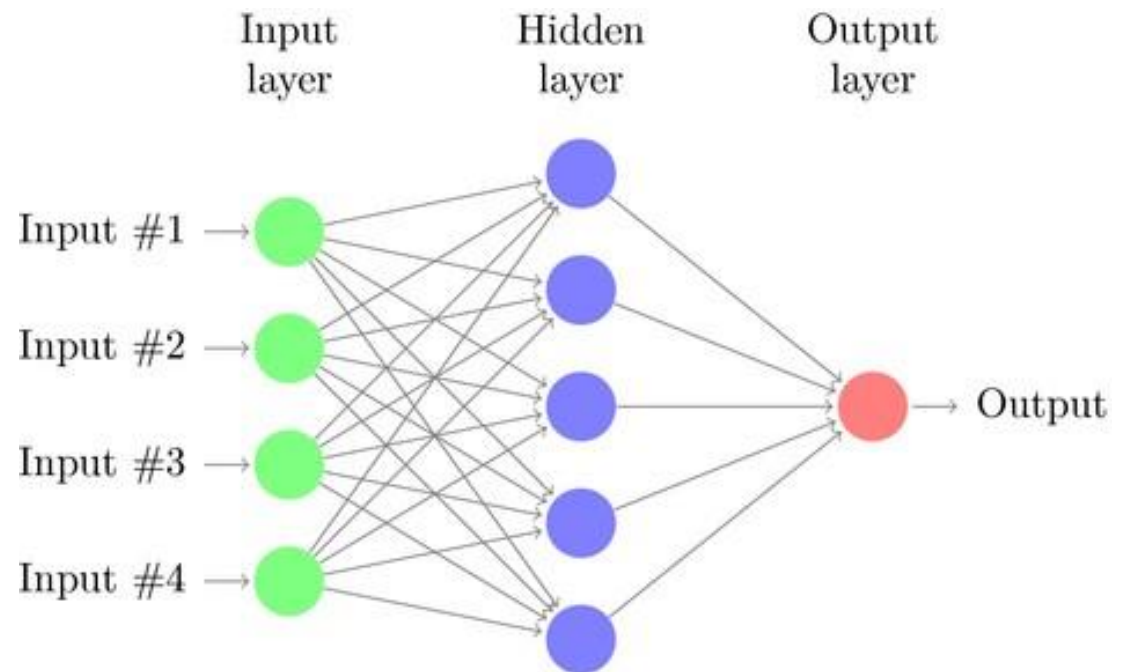
- ❑ **Auto-Organización y Adaptabilidad:** utilizan algoritmos de aprendizaje adaptativo y auto-organización, por lo que ofrecen mejores posibilidades de procesamiento robusto y adaptativo.
- ❑ **Procesado no Lineal:** aumenta la capacidad de la red para aproximar funciones, clasificar patrones y aumenta su inmunidad frente al ruido.
- ❑ **Procesado Paralelo:** normalmente se usa un gran número de nodos de procesamiento, con alto nivel de interconectividad.

Red Neuronal Artificial

A igual que en los sistemas biológicos, en las redes neuronales artificiales necesitaremos **neuronas de entrada y neuronas de salida y conexiones** entre ambas.

Siempre habrá al menos 3 capas:

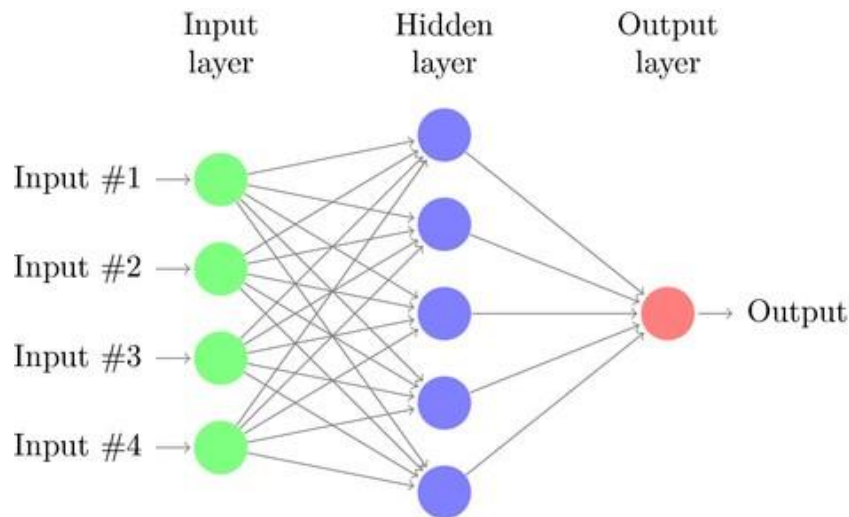
- **Capa de entrada (verde)**
- **Capa oculta (morada)**
- **Capa de salida (roja)**



Red Neuronal Artificial

- **Capa de entrada:** los nodos de esta capa son las **variables de entrada** disponibles en el conjunto de datos o **variables independientes**. Contiene los nodos **input**
- **Capa oculta:** en esta capa hay variables “**artificiales**” o **nodos ocultos** que en general no existen en los datos. Estas variables conectan la entrada con la salida, y permiten construir la función que se utiliza para determinar la variable de salida en función de las variables de entrada. En general, trabajaremos con redes neuronales con **una** sola capa oculta. Los problemas con más de una capa oculta se abordan con técnicas de *deep learning*.
- **Capa de salida:** contiene la **variable respuesta** o **dependiente** (puede haber uno o varios nodos, tantos como características se quieran predecir, clasificar o explicar). Puede ser continua o cualitativa.

Planteamiento



Al construir una red neuronal estamos definiendo un **modelo** que relaciona las entradas o los nodos de la capa de entrada (variables independientes) mediante una función que viene asociada por las **relaciones existentes** entre cada uno de los nodos de las distintas capas, con la variable respuesta de la capa de salida. En general estas funciones **no son lineales**.

Una Red Neuronal es en realidad un modelo de la forma

$$y=f(x1,x2,x3,...)$$

donde la función f es por lo general no lineal.

Ejemplo de red:

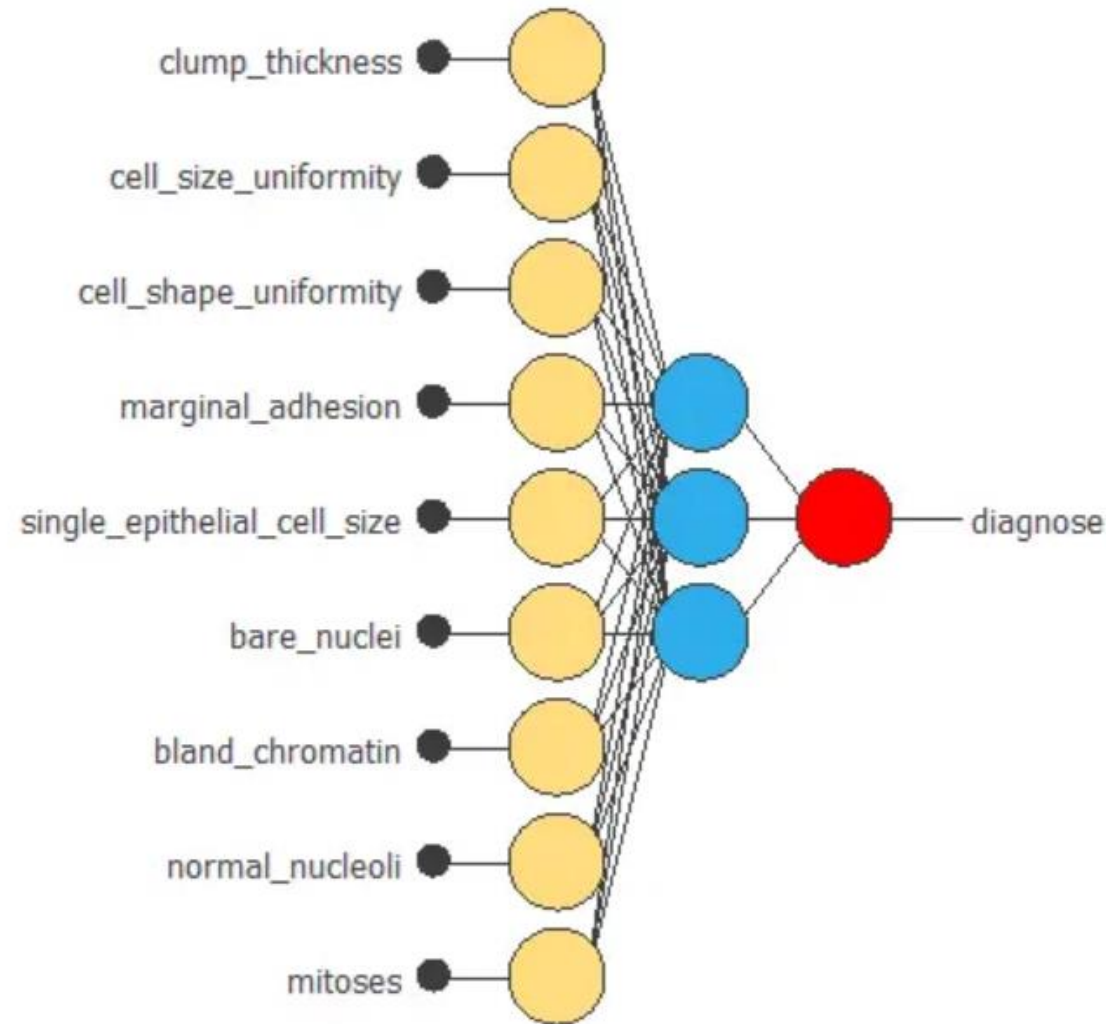
$$y=29.8+85.2*\tanh (-0.9+2.1*x1-0.15*x2)-79*\tanh (-3.5+5*x1+0.01*x2)$$

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

Cualquiera de los problemas conocidos se pueden abordar con la técnica de redes neuronales

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

Red neuronal artificial: problema de diagnóstico



¿Qué problemas en el campo de la Inteligencia de Negocios se suelen abordar con las redes neuronales? Principalmente problemas relacionados con predicción y clasificación

- **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

Otras aplicaciones

- **Negocios:** marketing, campañas de venta
- **Medicina y salud:** ayuda al diagnóstico, análisis de imágenes, desarrollo de medicamentos y distribución de recursos.
- **Alimentación:** análisis de olor y aroma, perfilamiento de clientes, desarrollo de productos, control de calidad.
- **Transporte y comunicaciones:** optimización de rutas y de distribución de recursos.
- **Tratamiento de texto y procesamiento de formas:** reconocimiento de caracteres impresos o escritos a mano, reconocimiento de gráficos...

Concretamente...

- **Reconocimiento de voz:** se utilizan en sistemas de reconocimiento de voz para transcribir audio a texto y permitir la interacción hombre-máquina mediante comandos de voz.
- **Reconocimiento de imágenes:** se utilizan para el reconocimiento de imágenes y la clasificación de objetos, lo que permite aplicaciones en visión por computadora, reconocimiento facial, seguridad, y en sistemas de cámaras de seguridad.
- **Procesamiento del lenguaje natural:** que permite aplicaciones como chatbots, asistentes virtuales, y sistemas de traducción automática.
- **Conducción autónoma:** sistemas de conducción autónoma para procesar y analizar datos de sensores de vehículos, como cámaras, radares y lidar.
- **Pronóstico del tiempo**
- **Juegos:** en juegos como ajedrez, Go y póker para aprender a tomar decisiones óptimas en función del estado actual del juego y maximizar la probabilidad de ganar.

Aplicaciones reales

- **Reconocimiento de voz**, para transcribir audio a texto. Siri
- **Pronunciación: NETtalk (87)**, aprende a pronunciar texto escrito. 29 unidades de entrada (26 letras, más espacios, puntos, comas), 80 unidades ocultas. 1024-palabras de entrenamiento. 95% de éxito en entrenamiento, y 78% en la prueba.
- **Reconocimiento de caracteres**: una de las aplicaciones más grandes de redes neuronales actuales (Le Cun et al. 89). Esta red lee códigos postales en cartas escritas a mano. El sistema tiene un preprocesador para localizar números, y la red los descifra. 3 capas ocultas (768, 192 y 30 unidades cada una). Logra un 99% de éxito, adecuado para un sistema de correo automático y se ha implementado en un chip.
- **ALVINN** (Autonomous Land Vehicle In a Neural Network) (Pomerleau 93) es una red neuronal que aprende a conducir un vehículo viendo cómo lo hace un humano. Se utiliza una cámara que alimenta una rejilla de entradas a la red. La salida (30 unidades) controla la dirección del volante.

Aplicaciones reales

- **Búsqueda de imágenes de Google:** Google utiliza una red neuronal para etiquetar automáticamente las imágenes y mejorar la precisión de la búsqueda de imágenes.
- **YouTube:** utiliza una red neuronal para recomendar videos personalizados a los usuarios, basándose en su historial de búsqueda y visualización.
- **Facebook:** para analizar y etiquetar automáticamente las fotos subidas por los usuarios, lo que permite una fácil búsqueda y organización.
- **Netflix:** para recomendar películas y programas de televisión personalizados a los usuarios, basándose en su historial de visualización y calificaciones.
- **Amazon:** para recomendar productos personalizados a los usuarios, basándose en su historial de búsqueda y compra.
- **Spotify:** para recomendar música personalizada a los usuarios, basándose en sus preferencias de escucha y hábitos.
- **Google Translate:** para mejorar la calidad de las traducciones y hacerlas más precisas y naturales.

Algunas APPs para “jugar” con redes neuronales

- **Deep Learning Studio (<https://deeplearningstudio.com/>):** plataforma en línea gratuita que permite crear, entrenar y probar modelos de redes neuronales sin necesidad de instalar software. Compatible con una variedad de marcos de aprendizaje profundo y ofrece una interfaz gráfica.
- **Neural Network Console (<https://dl.sony.com/>):** herramienta en línea gratuita que permite diseñar y entrenar redes neuronales de forma interactiva. Puedes seleccionar diferentes capas y funciones de activación, ajustar los parámetros de entrenamiento y observar el rendimiento de la red en tiempo real.
- **Keras (<https://keras.io/>):** biblioteca de aprendizaje profundo de código abierto para Python que permite crear y entrenar redes neuronales. Es fácil de usar y ofrece una amplia gama de funciones de activación y capas para construir modelos complejos de redes neuronales.
- **Caffe (<https://caffe.berkeleyvision.org/>):** marco de aprendizaje profundo de código abierto para la construcción y entrenamiento de redes neuronales. Ofrece un rendimiento rápido y eficiente para modelos de redes neuronales grandes y complejos.

Algunas APPs para “jugar” con redes neuronales

- [TensorFlow Playground](https://playground.tensorflow.org/) (<https://playground.tensorflow.org/>)
herramienta en línea gratuita y fácil de usar que te permite experimentar con redes neuronales en tiempo real. Puedes ajustar los parámetros de la red neuronal y observar cómo cambia el rendimiento de la red en diferentes tareas.
- <https://www.i-am.ai/neural-numbers.html>
- <https://www.i-am.ai/piano-genie.html>
- <https://modeldepot.github.io/tfjs-yolo-tiny-demo/>
- <http://places2.csail.mit.edu/>
- Y por supuesto, GPT:

<https://chat.openai.com> ps://chat.openai.com

2. ¿Por qué utilizar redes neuronales artificiales?

1. Comparación entre redes neuronales y métodos clásicos
2. Las dos culturas de la modelización estadística
3. Escenarios apropiados para la aplicación de redes neuronales

Comparativas entre la aplicación de redes neuronales y métodos clásicos existentes en la literatura.

Métodos de validación y datos utilizados y tamaño de las muestras consideradas.

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

Las redes neuronales han mejorado notablemente en los últimos años, ante el aumento de la cantidad y complejidad de los datos a analizar.

[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?

Las relaciones entre variables no son lineales, o no son conocidas.

OJO: también se podrían usar cuando sí se conocen las relaciones entre las variables pero el volumen de datos es demasiado grande.

Cuando las **relaciones** entre las variables son **relativamente lineales**, o el **volumen de datos no** es **excesivamente grande**, en ocasiones suelen funcionar mejor las **técnicas clásicas**, como la regresión lineal, logística o análisis discriminante, que las redes neuronales.

A día de hoy se siguen haciendo muchas comparativas entre métodos de *machine learning* y métodos clásicos, que nos permiten entender y diferenciar ambos escenarios.

Paréntesis Cultural

Las dos culturas de la modelización estadística

- ☐ Hay quien piensa que no es necesaria la aplicación de algoritmia, pudiendo considerar las técnicas estocásticas clásicas.
- ☐ Hay quien apoya lo contrario, insistiendo en la utilidad de la algoritmia.

“Statistical Modeling: The Two Cultures”

Leo Breiman, Statistical Science (2001)

En este artículo se presenta una comparativa entre ambos puntos de vista, planteando los pros/contras de cada una de ellas.

Proporciona una visión de la **modelización estocástica** y otra de la **modelización algorítmica**.

Normalmente se tiene un **conjunto de datos** y una **caja negra** para su explicación:

una serie de variables de entrada; a partir de esta caja negra se puede entender la relación entre las variables de entrada y la de salida con el objetivo de **predecir**, **clasificar** o simplemente obtener una **explicación** sobre la variable respuesta.

Desde la perspectiva de la **modelización estocástica** se considera que dentro de la **caja negra** hay **modelos estocásticos** en los que **se conocen los parámetros asociados a cada variable**, la importancia de las mismas, el error cometido, los parámetros estimados, etc. Esto permite establecer una función “sencilla” (regresión lineal, logística, clúster....) que relaciona las variables de entrada con la de salida. Para validar estos modelos se evalúa la efectividad mediante el **análisis de los residuos**.

Desde la perspectiva de la **modelización algorítmica** el **interior** de la **caja negra** es totalmente **desconocido**. Se construye una función que relaciona las variables de entrada con la independiente. No se puede determinar la influencia de cada variable sobre el objetivo. **No es posible hacer la validación mediante test ni análisis de los residuos**. Para evaluar estos métodos se analiza la diferencia entre los valores reales y los predichos, técnicas de validación cruzada, *training/test*. **NECESARIO GRAN VOLUMEN DE DATOS**.

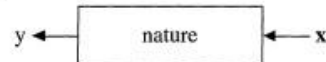
Statistical Modeling: The Two Cultures

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.

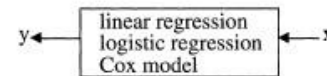
There are two different approaches toward these goals:

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:

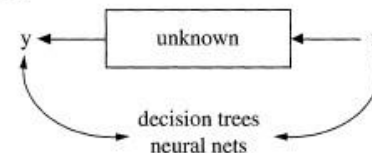


Model validation. Yes—no using goodness-of-fit tests and residual examination.

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.

**¿Métodos Estocásticos
VS
Métodos Algorítmicos?**

1. Limitaciones en los modelos estadísticos

Siempre que se define un modelo estocástico, hay que determinar algo del tipo: “*este conjunto de datos ha sido generado de acuerdo al modelo [-----].*”

Además, hay que exigir una inferencia estadística a cada uno de los parámetros y residuos resultantes del modelo. Se asocia una hipótesis previa al conjunto de datos.

- **Regresión lineal:** $y = b_0 + b_1x_1 + b_2x_2 + e$
- **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, relaciones desconocidas, lineales y no lineales, rangos de diferente comportamiento, las hipótesis de normalidad no se cumplen (sobre todo con grandes volúmenes de datos), gran cantidad de observaciones, etc.

Ante estas situaciones es difícil establecer un modelo y es optimista ceñirse a él.

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

Cuando los datos son muy numerosos o muy complejos no se pueden garantizar las hipótesis/tests.

Cuando el conjunto de datos es complejo, es conveniente tratar de comparar la aplicación de **modelos clásicos**, que ayudan a la **interpretación**, con la aplicación de **técnicas algorítmicas**, que proporcionan una **mejor predicción**.

2. La multiplicidad de modelos

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

- **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 (**lo cual no siempre es cierto**).

La rigidez que en ocasiones se exige a los planteamientos clásicos puede desembocar en una **falta de eficacia predictiva**.

IMPRESCINDIBLE: **comparar métodos clásicos con métodos algorítmicos**, y **contrastarlos** mediante **técnicas de validación cruzada y/o datos test**, probando muchas opciones, incluido métodos de ensamblado (agregado de modelos: bagging, boosting, etc.)

3. Explicar o predecir

Los modelos estadísticos tienen la ventaja de proporcionar información sobre la aportación de las diferentes variables input al modelo, de la significatividad, signo y magnitud de los parámetros correspondientes, ganando explicabilidad. Pero a menudo esto es poco importante respecto de 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.

Cuando la intención es predecir, es preferible utilizar métodos algorítmicos.

Cuando no se conocen las relaciones entre las variables, o cuando estas relaciones no son lineales, es preferible aplicar la modelización algorítmica que la modelización clásica

Cuando se tiene más interés en predecir que en entender los datos, es preferible aplicar la modelización algorítmica que la modelización clásica

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**, que no se obtienen por medición directa, sino que se pueden obtener a partir de otras variables; esto es, no se observan directamente sino que son inferidas. Por ejemplo, la calidad de vida de una persona, que se puede intuir a partir de otras variables.
- **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.

Estos problemas son habituales cuando nos enfrentamos a grandes volúmenes de datos.

Aunque en muchas ocasiones se pueden resolver, principalmente mediante la caja negra de los modelos algorítmicos, es conveniente enfrentarse en la medida de lo posible a estas casuísticas antes de proponer un modelo, mediante el análisis y depuración de datos.

Siempre se intentará tener los datos lo más limpios y claros que se pueda, antes de considerar el modelo de red neuronal. Esto nos proporcionará mejores predicciones.

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.

¿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, datos perdidos, variables latentes, etc.)
- **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 necesita que haya datos suficientes para **entrenar el modelo** y obtener la función asociada al conjunto de variables, y otra parte de los datos para usarlos como **test** en la validación del modelo).

Cuando los datos no son suficientes, las técnicas de *machine learning* tienden a sobreajustar.

¿Cuándo NO utilizar las redes neuronales?

- **Linealidad** en las relaciones, o **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 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.

Aplicaremos modelos de redes neuronales y se compararán con el modelo de regresión lineal o logística que se habría considerado en su lugar.

¿Qué características debe tener un modelo, o qué necesidades tengo, para decidir si es o no conveniente aplicar técnicas de *machine learning*, y cuáles son más convenientes?

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	●	●	●	●	●

1)

Multivariate Adaptive
Regression Splines

IMPORTANTE TENER LOS DATOS LO MAS LIMPIOS POSIBLE PARA CUBRIR LAS CARENCIAS
DEPURATIVAS DE LAS REDES NEURONALES Y APROVECHAR SU CAPACIDAD PREDICTIVA

Fases en la modelización con redes neuronales

Fase de entrenamiento: se usa un conjunto de datos o patrones de entrenamiento para determinar los pesos (parámetros) que definen el modelo de red neuronal. Se calculan iterativamente, de acuerdo con los valores de entrenamiento, con el objeto de **minimizar el error cometido entre la salida obtenida por la red neuronal y la salida deseada**.

Fase de Prueba: en la fase anterior, el modelo puede que se ajuste demasiado a las particularidades presentes en los patrones de entrenamiento, perdiendo su habilidad de generalizar su aprendizaje a casos nuevos (**sobreajuste**). Para evitar el problema del sobreajuste, es aconsejable utilizar un segundo grupo de **datos diferentes a los de entrenamiento**, el grupo de validación, que permita controlar el proceso de aprendizaje.

Las redes neuronales suelen tender al sobreajuste. Para evitarlo, se recomienda utilizar un conjunto de datos de validación (diferente al de prueba) que permita controlar el proceso de aprendizaje.



Machine Learning

Redes Neuronales

Inmaculada Gutiérrez García-Pardo

3. Arquitectura básica de las NN

1. Definición de la arquitectura de una red neuronal
2. Estructuras de conexión
3. Elementos de una red neuronal multicapa
 1. Capas y neuronas
 2. Función de combinación
 3. Función de activación
 4. Función de pérdida o costes
4. Motivación y justificación matemática del uso de NN

Arquitectura de una red neuronal: se refiere a la **estructura y organización** de las capas y conexiones dentro de la red, especificando cómo las **neuronas** y las **capas** están **dispuestas y conectadas** para realizar una tarea específica. Puede incluir detalles como el número de capas, el número de neuronas en cada capa, las funciones de activación utilizadas y la forma en que las neuronas están conectadas entre sí

- **Perceptrón:** es la unidad básica de una red neuronal. Consiste en una sola capa de neuronas conectadas directamente a la entrada y produciendo una salida. Puede utilizarse para problemas de **clasificación binaria**, pues básicamente su capacidad es “separar” los datos. Consta de una capa de entrada con neuronas que presentan las características de entrada, en la que cada conexión está asociada a un peso, y una capa de salida con una única neurona que produce la salida del perceptrón, aplicando una función de activación al resultado de la suma ponderada de las entradas y los pesos
- **Red Neuronal Multicapa (MLP):** compuesta por múltiples capas de neuronas (entrada, capas ocultas y salida). Utiliza la retropropagación del error para entrenamiento. Adecuada para una variedad de tareas, incluyendo clasificación y regresión.
 - **Este módulo se centrará en las MLP con una sola capa oculta**
- **Redes Neuronales Convolucionales (CNN):** diseñadas para procesar datos en forma de cuadrículas, como imágenes. Utilizan capas convolucionales para detectar patrones locales. Estas capas utilizan filtros o kernels para extraer características locales y comparten parámetros a través de la entrada; cada neurona está conectada solo a una región local de la entrada, no a toda la entrada. Se suelen usar en tareas de visión por ordenador.

Arquitectura de una red neuronal

- **Redes Neuronales Recurrentes (RNN):** diseñadas para trabajar con datos secuenciales, como series temporales o texto. Contienen conexiones recurrentes que permiten a la red mantener información sobre estados anteriores. Útiles para tareas de procesamiento de lenguaje natural (NLP) y predicción temporal.
- **Redes Neuronales Generativas (GAN):** compuestas por un generador y un discriminador. El generador crea datos sintéticos, mientras que el discriminador evalúa la autenticidad. Utilizadas en la generación de imágenes y otros datos.
- **Autoencoders:** formados por un codificador y un decodificador. Aprenden una representación compacta de los datos. Útiles en tareas de reducción de dimensionalidad y reconstrucción de datos.
- **Redes Residuales (ResNet):** Introducen conexiones residuales para facilitar el entrenamiento de redes profundas. Dirigidas a superar el problema de descenso del gradiente en arquitecturas profundas.
- **Redes Neuronales Siamesas:** utilizadas para comparar similitudes entre dos entradas. Comparten parámetros entre dos ramas de la red para aprender representaciones similares.
- **Redes Neuronales de Atención (Transformer):** introducen mecanismos de atención para capturar relaciones entre diferentes partes de la entrada. Ampliamente utilizadas en tareas de procesamiento de lenguaje natural. (CHATGPT)

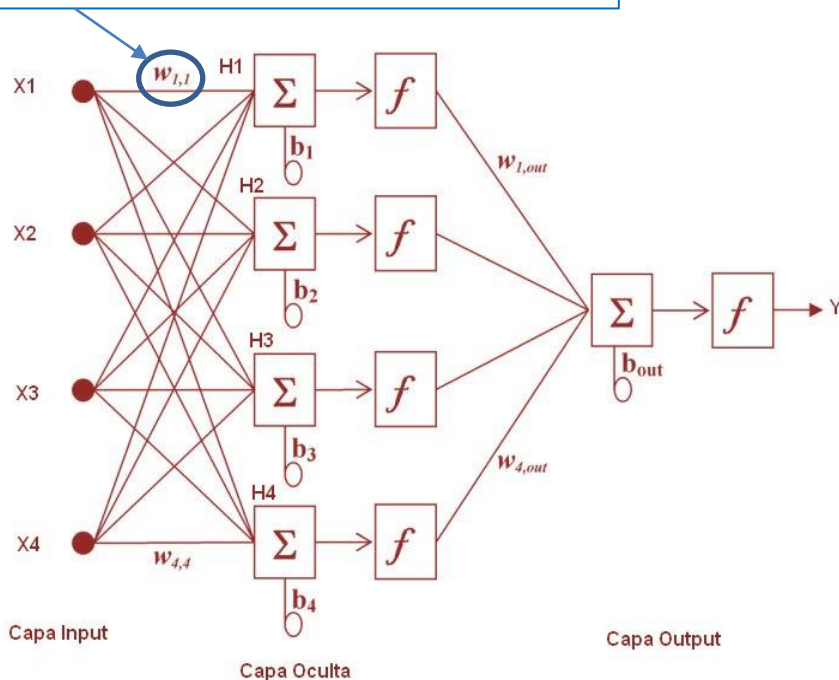
Estructuras de conexión

- **Conexiones hacia delante:** los valores de las neuronas de una capa inferior son propagados hacia las neuronas de la capa superior por medio de las redes de conexiones hacia adelante.
- **Conexiones hacia atrás:** llevan los valores de las neuronas de una capa superior a otras de la capa inferior.
- **Conexiones laterales:** un ejemplo es el *winner-takes-all*: a la neurona de salida que da el valor más alto se le asigna el valor total
- **Conexiones con retardo:** los elementos de retardo se incorporan en las conexiones para implementar modelos dinámicos y temporales

Construcción del modelo: MLP hacia adelante, backpropagation

La **capa input** o de entrada (cuyos nodos son las variables del modelo) se conecta a la **capa oculta** (cuando se construye el modelo hay que determinar cuántos nodos hay en la capa oculta) mediante la **función de combinación**, representada por Σ , donde los pesos w_{ij} hacen el papel de parámetros a estimar. En la **capa de salida** tenemos tantos nodos como **variables de salida** haya.

Peso asignado a la conexión entre el nodo X1 de la capa de entrada y el nodo H1 de la capa oculta



Modelo de caja negra: el número de nodos de la capa oculta se determina mediante **prueba/error**. No hay un número de nodos óptimo establecido.

Lo primero es conectar cada nodo de la capa de entrada con cada nodo de la capa oculta (sinapsis).

Desde cada nodo de la capa de entrada saldrán tantas conexiones como neuronas haya en la capa oculta.

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

Los nodos de la capa de entrada se conectan con los nodos de la capa oculta mediante una **función de combinación** Σ cuyos parámetros son los pesos w_{ij} .

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

Una función de combinación para cada nodo de la capa oculta:

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

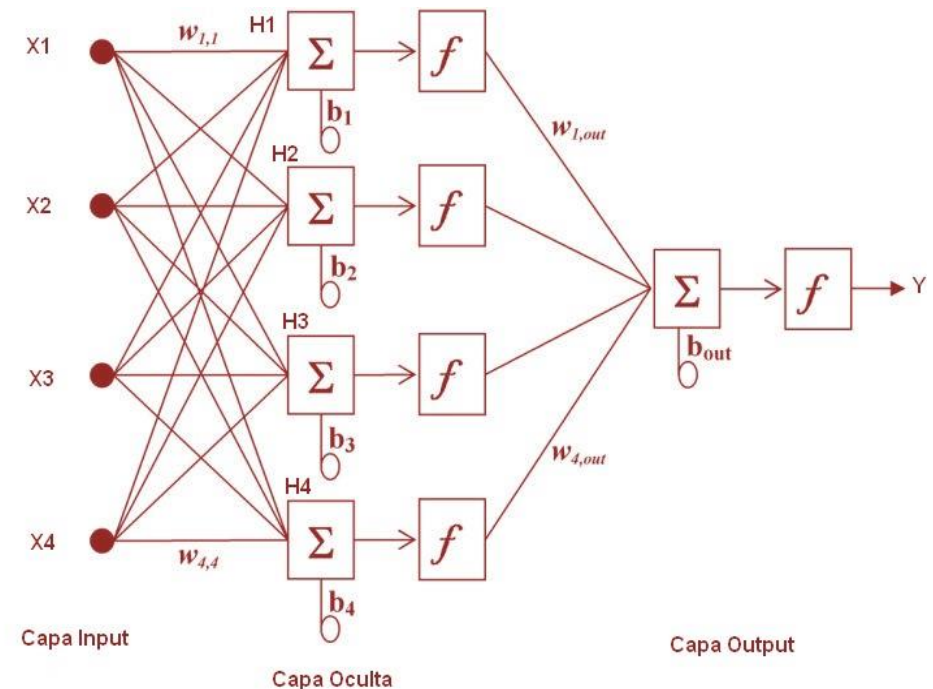
$$H2 = w_{12}X1 + w_{22}X2 + w_{32}X3 + w_{42}X4 + b_2$$

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

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

Para cada nodo de la capa oculta hay un peso asociado que lo conecta con la capa de salida.
El parámetro constante asociado a cada nodo, b_j se denomina, en jerga neuronal, **bias** (sesgo). Es el parámetro que controla “**lo que no puede explicar el modelo con los pesos**”. Hay que estimarlos.

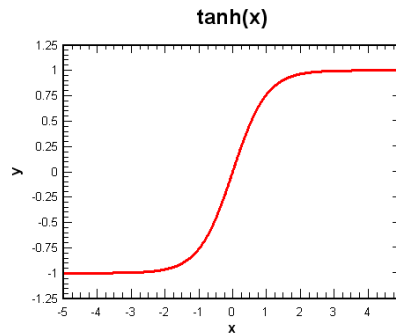
La activación externa de las neuronas de entrada junto con la función de combinación que las conecta con los nodos de la capa oculta, se **activan** y pasan a la capa de salida.



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 (que ya ha recibido la combinación con los nodos de la capa de entrada):

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

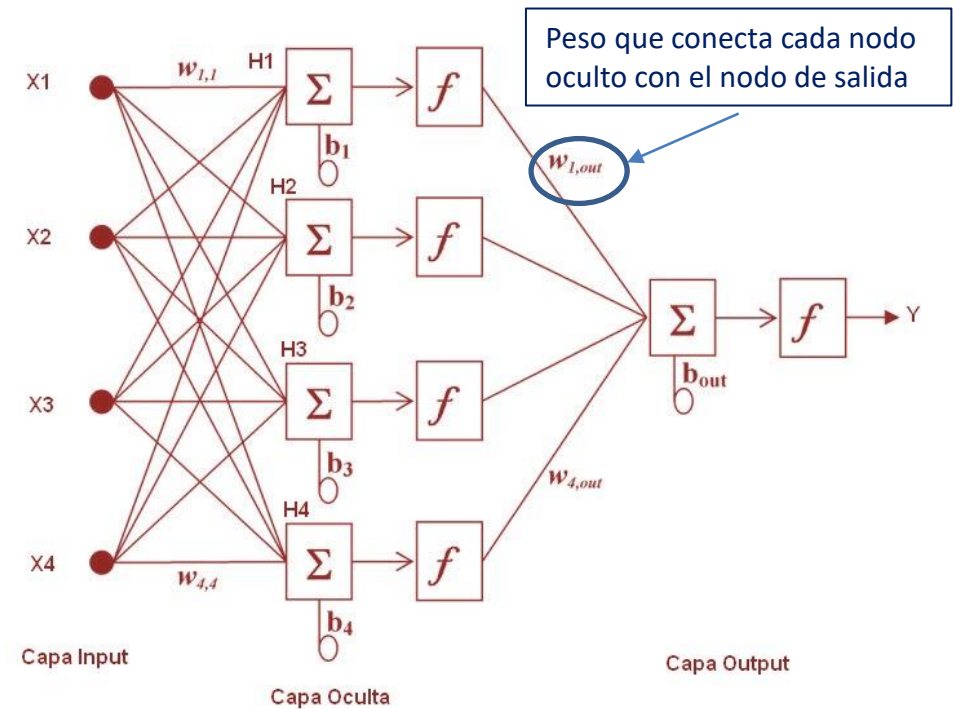
$$H2 = \tanh(w_{12}X1 + w_{22}X2 + w_{32}X3 + w_{42}X4 + b_2)$$

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

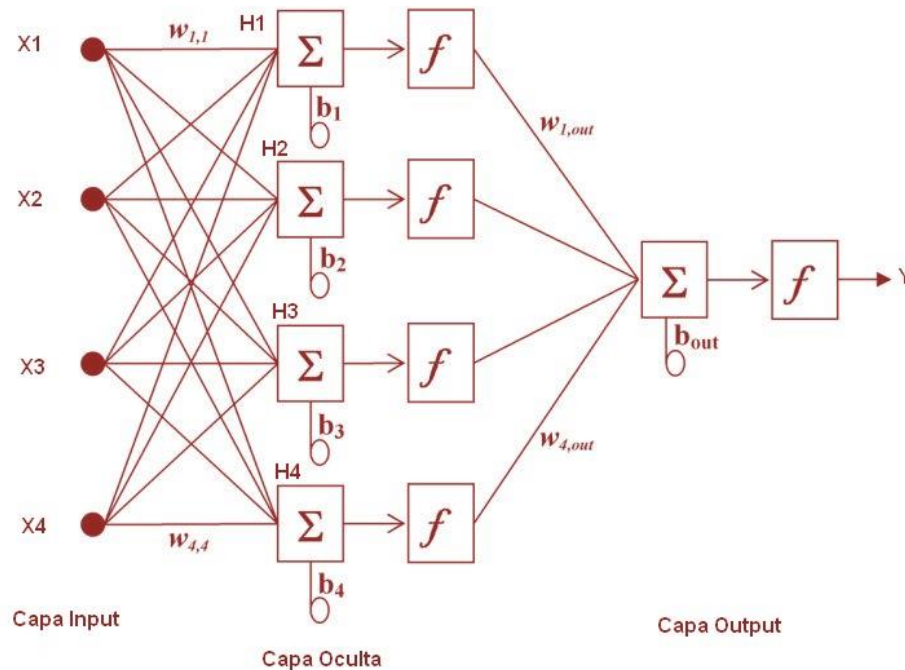
$$H4 = \tanh(w_{14}X1 + w_{24}X2 + w_{34}X3 + w_{44}X4 + b_4)$$

Las funciones de activación son la “entrada” a la siguiente capa; bien sea otra capa oculta, si es que hay más, o la capa de salida.

En general, dentro de la misma capa se suele usar la misma función de activación.



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



RECORDATORIO:

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

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

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

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

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 en el nodo de salida)

- $Y = \tanh(Y) = \tanh(w_{1,out}H1 + w_{2,out}H2 + w_{3,out}H3 + w_{4,out}H4 + b_{out})$ 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 (**si la variable dependiente fuera binaria; si es continua no hay que aplicar la primera *tanh***, correspondiente a la capa output):

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

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:

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

Para cada individuo i , el su valor en la variable Y , Y_i , dependerá de su valor en las otras variables, $X1_i, X2_i, X3_i, X4_i$, de los pesos $w_{11}, w_{12} \dots w_{44}, w_{1out}, w_{1out} \dots w_{4out}$, de los sesgos b_1, \dots, b_4 , y los resultados de las funciones de activación y combinación.

El algoritmo *backpropagation* es una etapa donde se presenta, ante la red entrenada, un patrón de entrada que se transmite a través de las capas para obtener una salida.

Algoritmo de aprendizaje:

método utilizado para ajustar los parámetros del modelo con el objetivo de minimizar la función de pérdida. El algoritmo de aprendizaje guía la optimización de la función de pérdida a lo largo del proceso de entrenamiento.

1.Backpropagation/Retropropagación del Error: utiliza el descenso del gradiente para ajustar los pesos, minimizando la diferencia entre las predicciones y las etiquetas reales. Es uno de los algoritmos más comunes para el entrenamiento de redes neuronales supervisadas.

2.Boltzmann Machines: son redes neuronales estocásticas basadas en máquinas de Boltzmann. Aplican muestreo estocástico para aprender patrones en datos y pueden utilizarse para modelar distribuciones probabilísticas.

3.LVQ (Vector Cuantizador Competitivo): algoritmo de aprendizaje no supervisado que agrupa los datos de entrada en clases o categorías. Cada neurona compite por activarse según la entrada, y la neurona ganadora actualiza sus pesos para asemejarse más a la entrada.

4.Memoria Asociativa: redes diseñadas para recordar y recuperar patrones específicos de entrada asociados con patrones de salida correspondientes.

5.ARTMap (Adaptive Resonance Theory Map): utiliza reglas de aprendizaje adaptativas y resonancia para clasificar patrones de entrada en categorías específicas.

6.Proyección de Shannon: técnicas basadas en la teoría de la información de Shannon para proyectar datos en un espacio más compacto o para reducir la dimensionalidad.

7.Kohonen SOM (Mapa Autoorganizado de Kohonen): algoritmo de aprendizaje no supervisado que organiza datos en un mapa topológico bidimensional, manteniendo las relaciones entre los datos input.

8.ART (Adaptive Resonance Theory): Es un conjunto de algoritmos diseñados para aprendizaje no supervisado y supervisado, permitiendo a la red adaptarse a nuevos patrones sin olvidar los antiguos.

9.Análisis Lineal de Discriminante (LDA): técnica de reducción de dimensionalidad que maximiza la separación entre clases en los datos.

10.Análisis de Componentes Principales (PCA): técnica de reducción de dimensionalidad que busca proyectar los datos en un nuevo espacio donde las variables están descorrelacionadas.

Función de activación: sirve para devolver una salida a partir de un valor de entrada, normalmente en un rango de salida determinado como $[0,1]$ o $[-1,1]$

Tangente hiperbólica: la función tangente hiperbólica transforma los valores introducidos a una escala $[-1,1]$, donde los valores altos tienen de manera asintótica a 1 y los valores muy bajos tienden de manera asintótica a -1. Se suele aplicar en clasificación multiclase y regresión.

$$f(x) = \frac{2}{1 + e^{-2x}} - 1$$

Sigmoide: transforma los valores introducidos a una escala $[0,1]$, donde los valores altos tienen de manera asintótica a 1 y los valores muy bajos tienden de manera asintótica a 0. Se suele aplicar en clasificación binaria. Se puede interpretar como probabilidad de pertenencia a la clase positiva.

$$f(x) = \frac{1}{1 + e^{-x}}$$

ReLU: transforma los valores introducidos anulando los valores negativos y dejando los positivos tal y como entran. Con respecto a otras funciones, tiene una ventaja significativa en términos de velocidad de entrenamiento y puede ayudar a prevenir el problema de gradiente que se encuentra en otras funciones de activación

$$f(x) = \max(0, x)$$

Función de activación:

Leaky ReLU: transforma los valores introducidos multiplicando los negativos por un coeficiente rectificativo y dejando los positivos según entran.

$$f(x) = \max \begin{cases} 0, x < 0 \\ a * x, x \geq 0 \end{cases}$$

Softmax: transforma las salidas a una representación en forma de probabilidades, de tal manera que el sumatorio de todas las probabilidades de las salidas de 1. Se suele utilizar en la capa de salida en problemas de clasificación multiclase.

$$f(Z)_j = \frac{e^{Z_j}}{\sum_{k=1}^K e^{Z_k}}$$

La elección de la función de activación es un aspecto importante en el diseño de una red neuronal y puede tener un impacto significativo en el rendimiento de la red. En la práctica real, se recomienda experimentar con diferentes funciones de activación y ajustar los parámetros de la red para encontrar la combinación que funcione mejor para tu problema.

Función de combinación/base: se refiere al cálculo realizado para obtener la entrada ponderada a una neurona antes de aplicar la función de activación.

- **Función lineal de tipo hiperplano:** el valor de la red es una combinación lineal de las entradas.
 - **Suma ponderada:** implica multiplicar cada entrada por su peso correspondiente y sumar estos productos. Supongamos que tienes n entradas x_1, x_2, \dots, x_n con pesos correspondientes w_1, w_2, \dots, w_n . La función de combinación, denotada como z , se expresa como:

$$z = w_1 \cdot x_1 + w_2 \cdot x_2 + \dots + w_n \cdot x_n$$

Para $n = 3$, la suma ponderada sería $z = w_1 \cdot x_1 + w_2 \cdot x_2 + w_3 \cdot x_3$.

- **Producto punto:** utiliza el producto punto entre el vector de entradas \mathbf{x} y el vector de pesos \mathbf{w} . La función de combinación es:

$$z = \mathbf{w} \cdot \mathbf{x} = \sum_{i=1}^n w_i \cdot x_i$$

Para $n = 2$, el producto punto se expresaría como $z = w_1 \cdot x_1 + w_2 \cdot x_2$.

- **Función radial de tipo hiperesférico:** es una función de segundo orden no lineal. El valor de la red representa la distancia a un determinado patrón de referencia.

$$z_\ell = \sqrt{\sum_{i=1}^n (x_j - w_{\ell i})^2}$$

Función de pérdida o coste: es una medida que cuantifica cuán bien el modelo hace predicciones en comparación con las etiquetas reales del conjunto de datos. Evalúa la discrepancia entre las predicciones del modelo y los valores reales, proporcionando una medida de error. El objetivo durante el entrenamiento es minimizar esta función, ya que un valor más bajo indica que el modelo está haciendo predicciones más precisas.

Regresión

1. Error Cuadrático Medio (MSE) para Regresión:

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

Utilizado en problemas de regresión para minimizar la diferencia cuadrática entre las predicciones y las etiquetas reales.

2. Error Absoluto Medio (MAE) para Regresión:

$$L(y, \hat{y}) = \frac{1}{n} \sum_{i=1}^n |y_i - \hat{y}_i|$$

Otra opción para regresión que minimiza la diferencia absoluta entre las predicciones y las etiquetas reales.

3. Huber Loss para Regresión Robusta:

$$L(y, \hat{y}) = \frac{1}{n} \sum_{i=1}^n \begin{cases} \frac{1}{2} (y_i - \hat{y}_i)^2, & \text{si } |y_i - \hat{y}_i| \leq \delta \\ \delta |y_i - \hat{y}_i| - \frac{1}{2} \delta^2, & \text{en otro caso} \end{cases}$$

Útil en regresión para minimizar la pérdida cuadrática en presencia de valores atípicos.

Función de pérdida o coste. Clasificación

4. Entropía Cruzada Binaria para Clasificación Binaria:

$$L(y, \hat{y}) = -\frac{1}{n} \sum_{i=1}^n [y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)]$$

Empleada en problemas de clasificación binaria para evaluar la discrepancia entre las predicciones de probabilidad y las etiquetas.

5. Hinge Loss (SVM) para Clasificación Binaria:

$$L(y, \hat{y}) = \frac{1}{n} \sum_{i=1}^n \max(0, 1 - y_i \cdot \hat{y}_i)$$

Común en máquinas de soporte vectorial (SVM) para clasificación binaria.

6. Focal Loss para Clasificación Binaria o Multiclase:

$$L(y, \hat{y}) = -\frac{1}{n} \sum_{i=1}^n \alpha (1 - \hat{y}_i)^\gamma \log(\hat{y}_i)$$

Introduce ponderaciones para abordar desequilibrios en datos, se puede utilizar en problemas de clasificación binaria o multiclase.

7. Entropía Cruzada Categórica para Clasificación Multiclase:

$$L(y, \hat{y}) = -\frac{1}{n} \sum_{i=1}^n \sum_{j=1}^m y_{ij} \log(\hat{y}_{ij})$$

Utilizada en problemas de clasificación multiclase para medir la discrepancia entre las predicciones de probabilidad y las etiquetas categóricas.

Función de pérdida o coste. Clasificación

8. Sparse Categorical Cross-Entropy para Clasificación Multiclase:

$$L(y, \hat{y}) = -\frac{1}{n} \sum_{i=1}^n \log(\hat{y}_i)$$

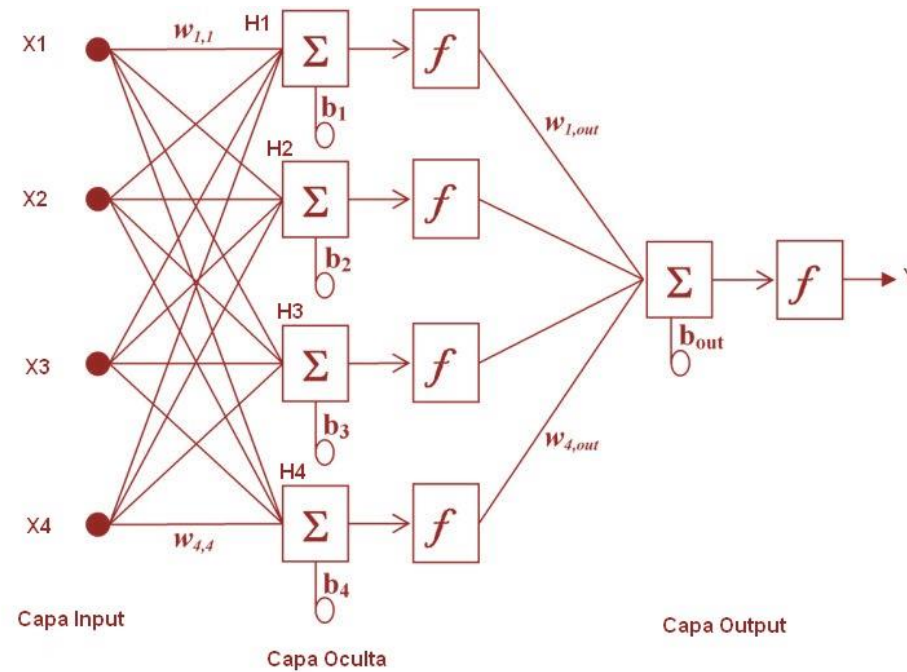
Otra opción para problemas de clasificación multiclase, especialmente cuando las etiquetas son escasas.

9. Kullback-Leibler Divergence para Clasificación Multiclase:

$$L(y, \hat{y}) = \sum_{i=1}^n y_i \log \left(\frac{y_i}{\hat{y}_i} \right)$$

Mide la divergencia entre las distribuciones de probabilidad y se utiliza en problemas de clasificación multiclase.

Parámetros de una red neuronal



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

(¡Más vale tener muchas observaciones!)

En general, Número de parámetros Red neuronal con una capa y una variable output:

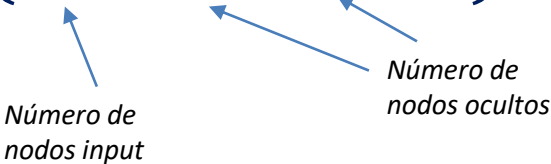
$$h(k+1)+h+1$$

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

En regresión clásica serían $k+1=5$ en nuestro caso con 4 variables de entrada.

OJO: no tiene por qué haber el mismo número de nodos en la capa oculta que en la capa de entrada

NOTA: si tengo una red neuronal con o nodos en la capa de salida, la fórmula del número de parámetros vendría dada por:

$$(k * h + h * o) + h + o$$


Número de nodos input

Número de nodos ocultos

El objetivo computacional concreto es **estimar** los parámetros w y b del modelo.

$$\begin{aligned} Y = & \tanh(w_{1,out}(\tanh(w_{11}X1 + w_{21}X2 + w_{31}X3 + w_{41}X4 + b_1))) \\ & + w_{2,out}(\tanh(w_{12}X1 + w_{22}X2 + w_{32}X3 + w_{42}X4 + b_2)) + \\ & + w_{3,out}(\tanh(w_{13}X1 + w_{23}X2 + w_{33}X3 + w_{43}X4 + b_3)) + \\ & + w_{4,out}(\tanh(w_{14}X1 + w_{24}X2 + w_{34}X3 + w_{44}X4 + b_4)) + b_{out} \end{aligned}$$

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).

$$\hat{Y} = \tanh(0.21(\tanh(0.15X1 - 0.53X2 + 2.1X3 + 3.5X4 + 6)) + \dots + \dots + \dots)$$

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

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

La idea es que la diferencia entre \hat{Y} e Y sea lo más pequeña posible.



Machine Learning

Redes Neuronales

Inmaculada Gutiérrez García-Pardo

4. ¿Por qué funcionan bien las redes neuronales?

En esta sección se presentan los conceptos matemáticos que sustentan y garantizan el buen funcionamiento de las redes neuronales

Resumen conceptual

- **Neuronas:** unidades básicas de procesamiento de información en las redes neuronales. Cada neurona tiene **una entrada**, que recibe señales de otras neuronas o del entorno, y **una salida**, que produce una respuesta. Pueden estar conectadas entre sí para formar una red compleja.
- **Sinapsis:** son las conexiones entre las neuronas en una red neuronal. Cada sinapsis tiene un peso asociado que determina la importancia relativa de la entrada de la neurona presináptica en la salida de la neurona postsináptica.
- **Funciones de activación:** funciones no lineales que se aplican a la salida de cada neurona para determinar su activación. Las funciones de activación más comunes incluyen la función sigmoidea, la función ReLU y la función tanh.
- **Capas:** las redes neuronales están organizadas en capas, que pueden ser de entrada, ocultas o de salida. La capa de entrada recibe los datos de entrada, la capa de salida produce la salida final y las capas ocultas realizan el procesamiento intermedio.
- **Aprendizaje:** se produce mediante el ajuste de los pesos de las sinapsis en función del error en la salida de la red. El objetivo es minimizar el error en la salida de la red mediante el ajuste de los pesos de las sinapsis a través del proceso de retropropagación del error.
- **Algoritmos de aprendizaje:** se utilizan para entrenar las redes. Los algoritmos más comunes incluyen el descenso de gradiente estocástico, la retropropagación del error y el algoritmo de propagación hacia atrás resiliente (RPROP).

En resumen, las redes neuronales son un **modelo computacional** poderoso que se basa en la **estructura** y el **funcionamiento** del sistema nervioso humano. El **aprendizaje** en una red neuronal se produce mediante el **ajuste de los pesos** de las sinapsis, lo que permite a la red aprender a realizar tareas complejas de manera autónoma.

Pregunta:

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

$$\begin{aligned} Y = & \tanh(w_{1,out}(\tanh(w_{11}X1 + w_{21}X2 + w_{31}X3 + w_{41}X4 + b_1))) \\ & + w_{2,out}(\tanh(w_{12}X1 + w_{22}X2 + w_{32}X3 + w_{42}X4 + b_2)) + \\ & + w_{3,out}(\tanh(w_{13}X1 + w_{23}X2 + w_{33}X3 + w_{43}X4 + b_3)) + \\ & + w_{4,out}(\tanh(w_{14}X1 + w_{24}X2 + w_{34}X3 + w_{44}X4 + b_4)) + b_{out} \end{aligned}$$

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

$$Y = b_0 + b_1X1 + b_2X2 + b_3X3 + b_4X4$$

TEOREMA DE APROXIMACIÓN UNIVERSAL

¿Por qué funcionan bien las redes neuronales?

- Alta capacidad para **aprender** y **adaptarse** a partir de los datos de entrada que se les proporcionan. Su **estructura**, basada en la organización en capas de neuronas interconectadas y la capacidad de **ajustar los pesos** de las conexiones entre ellas.
- Cuando se **entrena** una red neuronal, se proporcionan **datos de entrada** y se comparan con los **resultados** de salida esperados.
- La red **ajusta** continuamente los **pesos** de sus conexiones y el **sesgo** de cada neurona para minimizar el error en la salida y mejorar su capacidad para realizar tareas específicas.
- A medida que se entrena la red con más datos y se ajustan los pesos de sus conexiones, la red puede aprender a reconocer patrones en los datos de entrada y realizar tareas complejas con mayor precisión.
- **Alta capacidad de generalización** a partir de los datos de entrenamiento para realizar predicciones o clasificar nuevos datos de entrada: puede aplicar lo que ha aprendido a nuevos conjuntos de datos que nunca antes ha visto y producir resultados precisos.

Explicación matemática

- Las redes neuronales se basan en el **cálculo** y el **álgebra lineal**.
- Se puede representar una red neuronal como una función matemática que toma una serie de entradas y produce una salida. Esta función se construye a partir de múltiples capas de neuronas, cada una de las cuales realiza una operación matemática simple.
- Cada neurona tiene un **vector de pesos** y un **sesgo**.
- Los pesos se representan mediante una matriz y se utilizan para **ponderar las entradas** de la neurona.
- El sesgo se representa mediante un valor escalar y se utiliza para ajustar el **umbral de activación** de la neurona.
- La **salida** de una neurona se calcula mediante una **función de activación no lineal** que toma como entrada la **suma ponderada de las entradas y los pesos de la neurona más el sesgo**. Las funciones de activación más comunes incluyen la función sigmoidea, la función ReLU, la función tanh y la función softmax.
- La salida de una capa de neuronas se utiliza como entrada para la siguiente capa. La red neuronal se **entrena ajustando los pesos** de las conexiones entre las neuronas para **minimizar** una **función de pérdida** que mide la discrepancia entre la salida de la red y la salida esperada.
- El **ajuste de los pesos** se realiza mediante el cálculo del **gradiente de la función** de pérdida con respecto a los pesos de las conexiones y la actualización de los pesos en la dirección del gradiente descendente.
- Esto les permite aprender a partir de los datos de entrenamiento y producir resultados precisos en nuevos conjuntos de datos.

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:

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

Script para entender de manera interactiva el Teorema de Aproximación Universal:

<https://ichi.pro/es/comprender-el-teorema-de-aproximacion-universal-con-codigo-131176068372273>

Si existe relación entre las variables input y la variable output y esta relación es no lineal o desconocida, el Teorema de Aproximación Universal dice que esa función, aunque sea desconocida, se puede aproximar por la función construida con la red neuronal

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

Función de activación,
nosotros la denotamos por f

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

$$\begin{aligned} Y = & \tanh(w_{1,out}(\tanh(w_{11}X1 + w_{21}X2 + \\ & w_{31}X3 \\ & + w_{41}X4 + b_1) + \\ & w_{2,out}(\tanh(w_{12}X1 + w_{22}X2 + w_{32}X3 + w_{42}X4 + b_2) + \\ & w_{3,out}(\tanh(w_{13}X1 + w_{23}X2 + w_{33}X3 + w_{43}X4 + b_3) + \\ & w_{4,out}(\tanh(w_{14}X1 + w_{24}X2 + w_{34}X3 + w_{44}X4 + b_4) + \\ & b_{out})) \end{aligned}$$

“Sólamente” hay que decidir el **número de nodos ocultos**, 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.

En otras palabras

El teorema de aproximación universal dice que una red neuronal con una capa oculta y un número suficientemente grande de neuronas puede aproximar cualquier función continua en un intervalo finito de forma arbitrariamente precisa.

Por tanto, una red neuronal con **una sola capa oculta** puede modelar **cualquier función matemática continua** si tiene suficientes neuronas en esa capa oculta.

En otras palabras...si se puede representar una función matemática en un gráfico, entonces una red neuronal de una capa oculta con suficientes neuronas puede aproximar esa función con una precisión arbitraria.

Este teorema es importante porque muestra la capacidad de las redes neuronales para modelar cualquier tipo de función matemática.

Además, el teorema también proporciona una guía para determinar el número adecuado de neuronas en la capa oculta de una red neuronal para aproximar una función dada con una precisión determinada.

Teorema de aproximación universal. Explicación fácil

Se aborda un problema complicado: predecir el precio de una casa en base a características como el tamaño, el número de habitaciones, etc. difícil de resolver con reglas simples o fórmulas exactas.

Ahí es donde entra en juego la "**aproximación universal**".

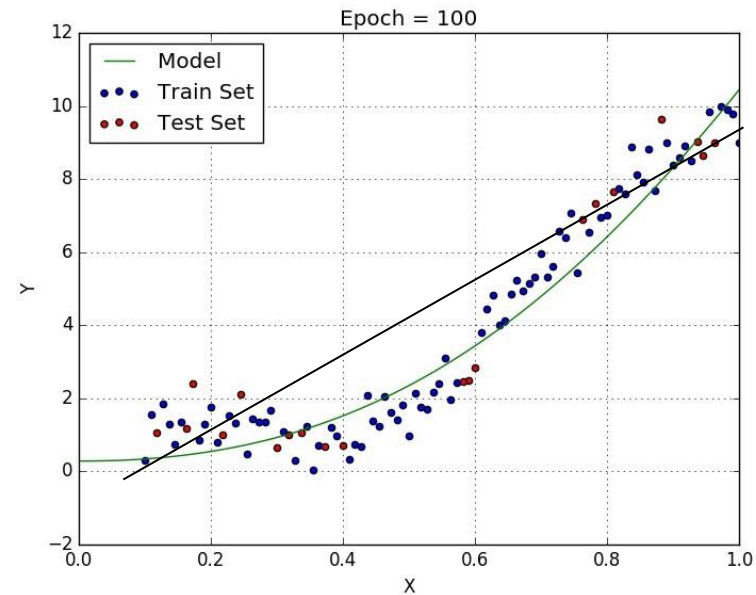
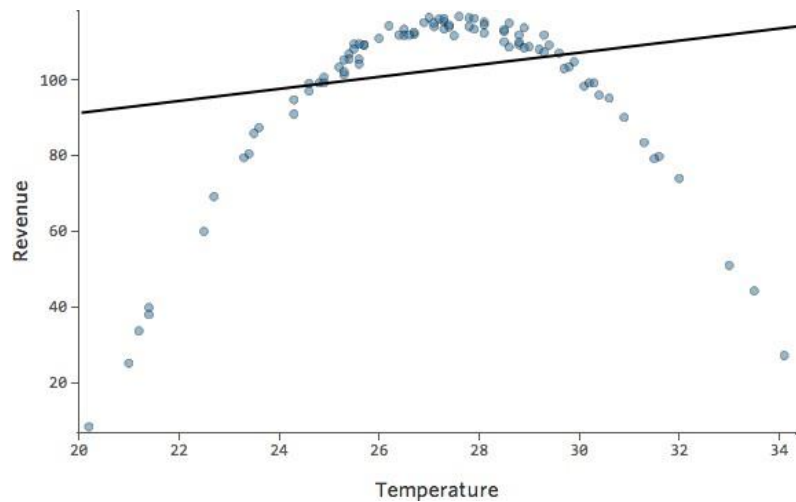
Se refiere a la capacidad de una red neuronal para aprender y representar cualquier función matemática. Entonces, se podría usar una red neuronal para aproximarse a la función que relaciona las características de una casa con su precio.

Piensa en la red neuronal como un "aprendiz inteligente". Durante el entrenamiento, le mostramos ejemplos de casas con sus características y precios correspondientes. La red neuronal ajusta sus "conexiones" (pesos y sesgos) de manera que, al final del entrenamiento, sea capaz de hacer predicciones precisas sobre el precio de una casa incluso si no ha visto exactamente la misma antes.

Cuanto más nodos haya en la capa oculta, mejor se ajustará el modelo a los datos de entrenamiento. Sin embargo, esto puede provocar el problema de **sobreajuste**, de manera que cuando se introduzcan datos nuevos el modelo no sepa ajustarse. Por eso, es importante llegar a un **equilibrio**.

El número de nodos de la capa oculta se calcula mediante **prueba-error**, atendiendo al número de observaciones, su complejidad, cantidad de variables, relaciones entre ellas, etc. Lo que está claro es que si no hay muchas observaciones, hay que evitar cantidades grandes de nodos en la capa oculta.

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.



En este ejemplo podemos ver cómo una regresión lineal no se ajusta nada bien a unos datos cuya relación es claramente no lineal.

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** (o desconocidas).
- 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.



Machine Learning

Redes Neuronales

Inmaculada Gutiérrez García-Pardo

5. Tipos de redes neuronales

En esta sección se presentarán algunos de los tipos más comunes de redes neuronales artificiales, explicando sus reglas de aprendizaje, arquitectura, algoritmos de aprendizaje y las tareas para las que se desarrollan.

El resto del módulo se centrará en las redes neuronales supervisadas.

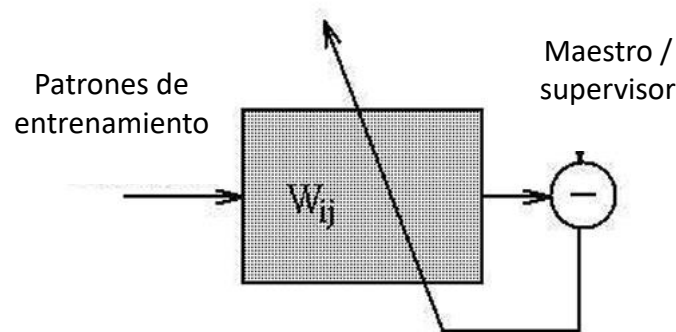
Redes neuronales supervisadas

El modelo de red neuronal supervisada se entrena utilizando un **conjunto de datos etiquetado**, donde cada entrada-instancia-caso del conjunto de datos está asociada a una **etiqueta** o resultado deseado (valor conocido de la variable respuesta). Durante el **entrenamiento**, la red **ajusta** sus **parámetros** internos mediante el proceso de **retropropagación**, minimizando la diferencia entre las predicciones del modelo y las etiquetas reales. Una vez entrenada, la red puede generalizar para hacer **predicciones** sobre **datos no vistos**.

Entrenamiento supervisado:

Las redes neuronales de entrenamiento supervisado son las más populares. Los datos de entrenamiento los forman varios pares de **patrones de entrenamiento de entrada y de salida**. Al conocer la salida (o valor/etiqueta de la variable respuesta) el entrenamiento se puede beneficiar de la **supervisión de un maestro** (analista que va controlando los resultados). Dado un nuevo patrón de entrenamiento, en la etapa $(m+1)$ -ésima, los pesos se adaptan de la siguiente forma:

$$w_{ij}^{m+1} = w_{ij}^m + \Delta w_{ij}^m$$



Redes neuronales supervisadas

Regla de aprendizaje	Arquitectura	Algoritmo de aprendizaje	Tareas
Corrección de error	Perceptrón / Perceptrón multicapa	Algoritmos de aprendizaje perceptrón, retropropagación del error, ADALINE, MADALINE	Clasificación de patrones, aproximación de funciones, predicción, control
Corrección de error	Elman y Jordan recurrentes	Retropropagación del error	Síntesis de series temporales
Boltzmann	Recurrente	Boltzmann	Clasificación de patrones
Competitivo	Competitivo	LVQ	Categorización intra-clases, compresión de datos
Competitivo	Red Art	ARTMap	Categorización intra-clases, clasificación de patrones

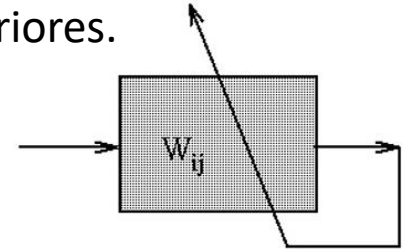
Redes neuronales no supervisadas

En general, para cualquier modelo de aprendizaje no supervisado, el conjunto de datos de entrenamiento consiste sólo en los patrones de entrada, de manera que el entrenamiento se lleva a cabo sin la supervisión y ajuste de un maestro analista. Una red no supervisada aprende a adaptarse en base a las experiencias recogidas de los patrones de entrenamiento anteriores.

Entrenamiento NO supervisado:

Es un enfoque en el aprendizaje automático donde el algoritmo se enfrenta a datos **sin etiquetas** y tiene la tarea de descubrir patrones, estructuras o relaciones inherentes en esos datos por sí mismo. A diferencia del aprendizaje supervisado, donde el modelo se entrena con ejemplos etiquetados, en el entrenamiento no supervisado, el modelo debe encontrar patrones sin la guía explícita de las etiquetas. Reglas generales:

- **Ausencia de etiquetas**
- **Descubrimiento de patrones:** el objetivo principal es identificar patrones, relaciones o estructuras intrínsecas en los datos. Esto podría incluir clustering o la reducción de dimensionalidad.
- **Exploración de la distribución de datos**
- **Autoencoders y redes generativas:** para aprender representaciones comprimidas de datos o el uso de redes generativas para crear nuevas muestras parecidas a las del conjunto de entrenamiento original.
- **Validación indirecta:** en lugar de utilizar métricas de rendimiento específicas (como precisión o error), la eficacia de los modelos no supervisados a menudo se valida indirectamente mediante la utilidad de los patrones descubiertos en aplicaciones específicas.



Redes neuronales no supervisadas

Algunos ejemplos de reglas de aprendizaje son la regla de **Hebb**, consistente en reforzar el peso que conecta dos nodos que se excitan simultáneamente, o la regla de aprendizaje **competitivo**. En el aprendizaje competitivo, si un patrón nuevo pertenece a una clase conocida previamente, entonces la inclusión del nuevo patrón en esta clase matizará la representación de la misma. Si el nuevo patrón no pertenece a ninguna de las clases reconocidas, entonces la estructura y los pesos de la red se ajustan para reconocer la nueva clase.

Redes neuronales no supervisadas

Regla de aprendizaje	Arquitectura	Algoritmo de aprendizaje	Tareas
Corrección de error	Red de Hopfield	Aprendizaje de memoria asociativa	Memoria asociativa
Corrección de error	Multicapa sin realimentación	Proyección de Sannon	Análisis de datos
Competitivo	Competitiva	VQ	Categorización, compresión de datos
Competitivo	SOM	Kohonen SOM	Categorización, análisis de datos
Competitivo	Redes ART	ART1, ART2	Categorización

Redes neuronales por refuerzo

Regla de aprendizaje	Arquitectura	Algoritmo de aprendizaje	Tareas
Hebbian	Multicapa sin realimentación	Análisis lineal discriminante	Análisis de datos, clasificación de patrones
Hebbian	Sin realimentación o competitiva	Análisis de componentes principales	Análisis de datos, compresión de datos



Machine Learning

Redes Neuronales

Inmaculada Gutiérrez García-Pardo

6. Redes neuronales multicapa. Uso general

1. Presentación de la aplicación de redes neuronales multicapa con una sola capa oculta para problemas de predicción. Se utilizará **Python**.
2. Fases en el entrenamiento de una red:
 1. Preparación de los datos. Especificaciones para NN: las variables deben seguir una distribución normal o uniforme, y el rango de posibles valores debe ser aproximadamente el mismo y acotado dentro del intervalo de trabajo de la función de activación empleada en las capas ocultas y de salida de la red neuronal. Así, las variables de entrada y salida suelen acotarse en valores comprendidos entre 0 y 1 ó entre -1 y 1 .
 2. Selección de variables
 3. Elección del conjunto inicial de pesos
 4. Evaluación del rendimiento

Preparación de los datos. Especificaciones para NN

Las variables deben seguir una distribución normal o uniforme, y el rango de posibles valores debe ser aproximadamente el mismo y acotado dentro del intervalo de trabajo de la función de activación empleada en las capas ocultas y de salida de la red neuronal. Así, las variables de entrada y salida suelen acotarse en valores comprendidos entre 0 y 1 ó entre -1 y 1 .

- Las variables categóricas hay que convertirlas a **dummies** antes de introducirlas en la red.
- Para cada variable categoría con **k categorías: $(k-1)$ variables dummies**. El valor de la “última” variable dummy se puede obtener a partir de las $(k-1)$ restantes.
- Los **datos** deben estar **depurados y sin datos missing**: las redes no son una buena herramienta para la depuración de datos.

Selección de variables en NN ¿Por qué es importante?

- 1.Eficiencia Computacional:** reducir la cantidad de variables puede hacer que el proceso de entrenamiento sea más rápido y menos intensivo computacionalmente. Menos variables implican menos cálculos durante la retropropagación del error y, por lo tanto, menos tiempo de entrenamiento.
- 2.Evitar dimensionalidad:** a medida que aumenta el número de variables, el espacio de búsqueda se vuelve más grande, y el modelo puede tener dificultades para encontrar patrones significativos en los datos. La dimensionalidad puede llevar a overfitting y a una disminución del rendimiento del modelo.
- 3.Mejora de la generalización:** un modelo entrenado con un conjunto de características más pequeño puede generalizar mejor a datos no vistos.
- 4.Interpretabilidad del Modelo:** una red neuronal más simple con un conjunto de variables más pequeño es más fácil de entender e interpretar.
- 5.Reducción del riesgo de overfitting:** un conjunto de datos con un gran número de variables puede llevar a un modelo que memoriza el conjunto de entrenamiento en lugar de aprender patrones generales. Esto aumenta el riesgo de overfitting, donde el modelo se ajusta demasiado a los datos de entrenamiento y no generaliza bien a nuevos datos.
- 6.Manejo de variables redundantes o colineales:** la selección de variables ayuda a identificar y eliminar características redundantes o altamente correlacionadas. El uso de variables redundantes puede afectar negativamente al rendimiento y dificultar la interpretación del modelo.
- 7.Reducción del Ruido:** para mejorar la calidad de los datos de entrada, facilitando que el modelo capture patrones significativos

Selección de variables en NN

Las redes neuronales son **cajas negras** con nula **capacidad explicativa** y escasa capacidad para lidiar con datos de entrada irrelevantes. **No incluyen procesos inherentes** de selección de variables (como por ejemplo los árboles de decisión).

La forma de lidiar (no del todo hábilmente) con los datos no relevantes (o variables con poca capacidad predictiva) es mediante la asignación de **pesos más altos** a las conexiones asociadas con **características más informativas** y reducir la influencia de aquellas menos relevantes. Durante la retropropagación del error, el algoritmo de aprendizaje ajusta estos pesos para minimizar la función de pérdida en el conjunto de entrenamiento. Las conexiones que contribuyen menos a la predicción del modelo pueden tener pesos más pequeños y, en última instancia, tener un impacto menor en las salidas finales.

Selección de variables en NN ¿Cómo?

1.Importancia de las Características: análisis de **sensibilidad** (cómo las predicciones cambian cuando se modifican los valores de una característica específica), selección preliminar con modelos que miden la importancia inherentemente (árboles, gradient boosting, etc.), RFE (eliminación recursiva de características),

2.Selección Manual: hecha por expertos en el dominio que pueden seleccionar manualmente un subconjunto de características basándose en su conocimiento y comprensión del problema.

3.Técnicas Automáticas: eliminación recursiva de variables (RFE) o técnicas basadas en información mutua.

4.Regularización: L1 (LASSO) o L2 (Ridge), pueden ayudar a reducir la importancia de ciertas características al penalizar sus coeficientes.

5.Análisis de Correlación: si dos variables están altamente correlacionadas, una de ellas podría ser redundante y eliminarse.

6.Reducción de Dimensionalidad: como Análisis de Componentes Principales (PCA) o t-SNE

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). Su principal defecto es que no tiene en cuenta la información relativa que aporta cada variable cuando otras están en el modelo. Ejemplos: lista ordenada por correlaciones o 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, pueden pasar por alto buenos modelos intermedios, o construir modelos que tienden 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.

Filters (ranking por importancia) en redes

En redes neuronales, contrariamente a modelos como el de regresión en los que cada parámetro va asociado a una variable o categoría, 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.

Un método clásico aunque insuficiente, es el **Método de Olden – Garson** (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.

En Python no hay librerías específicamente desarrolladas para esto, en su lugar se ofrecen alternativas de interpretabilidad como **SHAP, LIME, y Grad-CAM**.

Selección de variables en NN ¿Es posible interpretar los pesos de una red neuronal?

Bajo ningún concepto se pueden interpretar de una forma tan trivial, evidente y directa como se haría en un modelo de regresión. Sin embargo, en ciertos casos y bajo ciertas condiciones, es posible obtener algunas interpretaciones aproximadas de los pesos. Algunos enfoques son:

- **Visualización de Pesos:** en las capas de entrada, los pesos están asociados con las conexiones entre las entradas y las neuronas. Visualizar estos pesos puede dar una idea de qué características de entrada son más influyentes para la red.
- **Mapas de Activación** en capas intermedias de la red, para observar qué partes del espacio de características están siendo activadas más fuertemente. Esto puede proporcionar cierta intuición sobre las representaciones aprendidas.
- **Análisis de Importancia de Características:** se pueden utilizar técnicas de análisis de sensibilidad para evaluar cómo cambiar una característica afecta a la salida del modelo.
- **Capas de Atención (en el caso de modelos basados en atención):** en modelos como los Transformers, las capas de atención pueden indicar qué partes de la entrada reciben más atención durante el proceso de atención, proporcionando cierta interpretabilidad.
- **Estudio de Conexiones Ponderadas:** al analizar las conexiones más fuertes o más débiles en las capas intermedias y de salida, se puede obtener información sobre qué conexiones tienen un mayor impacto en la salida del modelo.

Selección de variables en NN ¿Es posible medir la importancia?

Se puede aplicar el método de **Olden-Garson** que mide la importancia relativa de las variables en modelos de regresión, incluidos los modelos basados en redes neuronales. Este método se centra en la contribución de cada variable en la predicción de la variable de respuesta. Fases:

Entrenamiento del Modelo:

Cálculo de la Contribución Relativa: para cada variable independiente input, se evalúa la contribución relativa al error cuadrático medio (ECM) o algún otro criterio de rendimiento del modelo. Se evalúa cómo cambiaría el rendimiento del modelo si se eliminara cada variable independiente.

Normalización: se normalizan las contribuciones para que sumen 100% o 1. Esto proporciona una medida relativa de la importancia de cada variable.

Interpretación: las contribuciones normalizadas indican la importancia relativa de cada variable en términos de su contribución al rendimiento del modelo.

OJO: la interpretación de la importancia de variables con el método de Olden-Garson puede depender de la **métrica** de rendimiento utilizada y de la naturaleza específica del problema.

Aunque este método proporciona una perspectiva sobre la importancia relativa de las variables, **no siempre revela la naturaleza de las relaciones no lineales o interacciones en el modelo**. La interpretación de la importancia puede ser **limitada en modelos de redes neuronales profundas** debido a la complejidad inherente de estas arquitecturas.

El orden de importancia planteado por los métodos de selección 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 (salvo quizás unas pocas) 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 (**modelo sencillo**), al conjunto que esté en el límite del sobreajuste o un poco por encima (**modelo complejo**)
- 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:** variables importantes 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.

INCISO: recordatorio sobre modelos de selección de variables

- **Método Backward** (eliminación hacia atrás): parte de un modelo muy complejo que incorpora todos los efectos que pueden influir en la respuesta, y en **cada etapa se elimina la variable menos influyente**, hasta que no procede suprimir ningún término más.
- **Método Forward** (selección hacia adelante): parte de un modelo mínimo sin variables y va agregando términos con algún criterio, hasta que no procede añadir ninguno más. En **cada etapa se introduce la variable más significativa** hasta alcanzar una cierta **regla de parada**.
- **Método Stepwise**: es una combinación de los dos anteriores. Comienza como el de introducción progresiva, pero en cada etapa se plantea si todas las variables introducidas deben permanecer en el modelo. Es probablemente la metodología más robusta.

¿Criterio para introducir/eliminar una variable?

- **Criterios de significación:** en un método backward se suprime el término que resulte menos significativo, y en un método forward se añade el término que al añadirlo al modelo sea más significativo. Un criterio puede ser la significación de cada coeficiente.
- **Criterios globales:** podemos basarnos en un criterio global, una medida global de cada modelo, de modo que tenga en cuenta el ajuste y el exceso de parámetros. Escogeremos el modelo cuya medida global sea mejor. Como criterios destacamos el Criterio de Información de Akaike (AIC) y el Criterio de Información de Bayes (BIC). Se trata de buscar un modelo cuyo **AIC** o **BIC** sea **pequeño**, ya que en ese caso habría una **verosimilitud muy grande y pocos parámetros**.
- **Otros métodos:**
 - **best subset:** dadas p variables explicativas, este método consiste en formar todos los posibles subconjuntos de variables explicativas y efectuar todas las posibles regresiones, reteniendo aquella que, de acuerdo con el criterio de bondad de ajuste que hayamos elegido, parezca mejor. El inconveniente de este método es el gran volumen de cálculo que es preciso realizar: método de Lasso, métodos Bridge, método SCAD.

Elección de los pesos iniciales

Es un paso crucial durante el proceso de entrenamiento y puede afectar significativamente el rendimiento y la convergencia del modelo.

Evitar Simetría Indeseada: una elección inicial incorrecta puede llevar a que todas las neuronas de una capa sean idénticas en las primeras etapas del entrenamiento. Esto puede hacer que las neuronas aprendan lo mismo y, por lo tanto, no contribuyan efectivamente al modelo.

Facilitar la Convergencia: pesos demasiado pequeños o grandes pueden ralentizar la convergencia o hacer que la red neuronal quede atrapada en mínimos locales.

Prevenir problemas con el gradiente

Estabilidad Numérica: valores demasiado grandes pueden llevar a la inestabilidad numérica, mientras que valores demasiado pequeños pueden no ser efectivos para transmitir información.

Métodos Comunes para la Elección Inicial de Pesos:

- **Inicialización Aleatoria:**
- **Zeros o Ones** (todos los pesos inicializados como 1 o 0, puede provocar problemas de simetría); .
 - Inicializar los pesos con valores aleatorios. Esto puede ayudar a romper la simetría y facilitar la convergencia.
- **Inicialización Xavier/Glorot:** ajusta la inicialización aleatoria para que la varianza de las salidas de una capa sea igual a la varianza de las entradas, ayudando a mitigar problemas con el gradiente.
- **Inicialización He:** similar a la inicialización Xavier, pero ajusta la varianza de las salidas de una capa para ser la mitad de la varianza de las entradas. Se suele utilizar con funciones de activación ReLU.
- **Inicialización de LeCun:** similar a la inicialización He, pero ajusta la varianza de las salidas de una capa para ser la misma que la varianza de las entradas. Se suele utilizar con funciones de activación ReLU.

Normalmente se hace una asignación de **pesos pequeños** generados de forma **aleatoria**, en un rango de valores entre -0.5 y 0.5 o similar.

Evaluación del rendimiento

Una vez seleccionado el modelo de red cuya configuración de parámetros ha obtenido la ejecución escogida como candidata en el conjunto de prueba, hay que evaluar la capacidad de generalización de la red de una forma completamente objetiva. Para ello, sería conveniente dividir los datos en tres partes, y reservar los datos de test para este propósito.

Cuando se trata de estimar un valor continuo se utiliza la medida cuadrática del error, mientras que en problemas de clasificación es mejor tener en cuenta la frecuencia de clasificaciones correctas e incorrectas (matriz de confusión)

Fases del entrenamiento de una red neuronal

1. Elección de los pesos iniciales
2. Elección de la arquitectura de la red
3. Evaluación del rendimiento
4. Interpretación de los pesos obtenidos



Machine Learning

Redes Neuronales

Inmaculada Gutiérrez García-Pardo

7. Ideas rápidas sobre la arquitectura de MLP

En esta sección se presentan algunos conceptos e ideas que pueden servir de guía o punto de partida iniciar la definición de la arquitectura de redes neuronales multicapa conectadas hacia adelante y con una sola capa oculta.

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** (normalización o cambio de escala). La razón es porque los algoritmos de optimización (usados para estimar los parámetros de la red) funcionan así mejor pues están menos expuestos a overflow (desbordamiento computacional) 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.

Arquitectura de la Red

Número de capas ocultas

Con **una capa oculta** en la mayor parte de los problemas de clasificación/regresión suele ser suficiente (los teoremas de aproximación así lo aseguran).

Ciertos casos complejo, sobre todo relacionados con análisis de imagen o texto, pueden precisar de modelos más complejos con una sola capa oculta.

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
- Función de activación
- 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.

Algunas 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 nodos como dimensiones en componentes principales suficientes para capturar un 70-90 % de la varianza de los inputs
- número de nodos inferior a $1/30$ de los datos de entrenamiento
- Para ajustar 20 nodos es necesario habitualmente tener entre 150 y 2500 observaciones” (Bishop).

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.)

Otros conceptos importantes a tener en cuenta

1. Experimentación: se recomienda comenzar con un número pequeño de nodos y aumentarlo gradualmente, evaluando el rendimiento de la red en un **conjunto de validación**. Se suele observar una mejora en el rendimiento a medida que aumenta el número de nodos, pero **OJO: peligro de sobreajustar**.

2. Técnicas de regularización: en lugar de fijar el número de nodos de antemano, se pueden aplicar técnicas de regularización para **penalizar los pesos** de la red y evitar el sobreajuste, lo que proporciona mayor flexibilidad en la elección del número de nodos, ya que ayuda a controlar la complejidad del modelo.

3. Arquitecturas predefinidas: algunas bibliotecas o marcos de trabajo de *Deep Learning* incluyen arquitecturas predefinidas o mejores prácticas para diferentes tipos de problemas.

Recuerda: no existe una solución única para determinar el número de nodos en una capa oculta, y es probable que debas realizar varias pruebas y ajustes para encontrar el equilibrio adecuado entre el rendimiento y la complejidad del modelo en tu problema específico.

Inciso: algunos conceptos teóricos sobre técnicas de regularización

- **Regularización L1 y L2:** se basan en agregar un término de penalización a la función de pérdida durante el entrenamiento de la red neuronal. Estos términos penalizan los pesos grandes y promueven pesos más pequeños. La regularización L1 añade la suma de los valores absolutos de los pesos como término de penalización, mientras que la regularización L2 añade la suma de los cuadrados de los pesos. Esto ayuda a evitar que los pesos se vuelvan demasiado grandes y contribuyan al sobreajuste.
- **Dropout:** es una técnica de regularización que consiste en desactivar aleatoriamente un porcentaje de las neuronas (aplicando una probabilidad de "descartar" cada neurona) durante el entrenamiento, lo que fuerza a la red a aprender características más robustas y reduce la dependencia de neuronas individuales. Durante la fase de evaluación se utilizan todas las neuronas pero se escala la salida para compensar la desactivación del entrenamiento.

Inciso: algunos conceptos teóricos sobre técnicas de regularización

- **Data augmentation:** implica aumentar artificialmente la cantidad de datos de entrenamiento mediante transformaciones como rotaciones, traslaciones, zoom, espejado, entre otros, lo que crea una variedad de muestras sintéticas que ayudan a que la red neuronal generalice mejor y evite el sobreajuste.
- **Early stopping:** consiste en detener el entrenamiento de la red antes de que se alcance el número total de épocas previsto, en función de un criterio de parada. Por lo general, se detiene el entrenamiento cuando el rendimiento en un conjunto de validación comienza a empeorar. Esto evita que la red continúe ajustándose demasiado a los datos de entrenamiento y mejora su capacidad de generalización.

Cada técnica tiene su propia aplicación y efecto en la red neuronal, y es posible combinar varias técnicas para obtener un mejor rendimiento y evitar el sobreajuste en el modelo. La elección de la técnica de regularización dependerá del problema específico y de las características de los datos.

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 sea 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) Algunos paquetes parten por defecto de una red con una cantidad mínima de nodos.
- 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 argumentar las razones por las cuales funciona mejor un número específico de nodos que otro, teniendo en cuenta las reglas simples anteriores.

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

- 1) Las relaciones parecen no lineales: la red puede funcionar bien
- 2) Modelo sencillo con dos variables, ¿cuántos nodos ocultos son necesarios para ajustar la curva?
- 3) Hay 1030 observaciones. Según la fórmula sobre el número de parámetros $h(k+1)+h+1$ (h =nodos ocultos, k =nodos input), para tener al menos 20 observaciones por parámetro, podemos tener como máximo

$$1030/20=51 \quad \text{parámetros.}$$

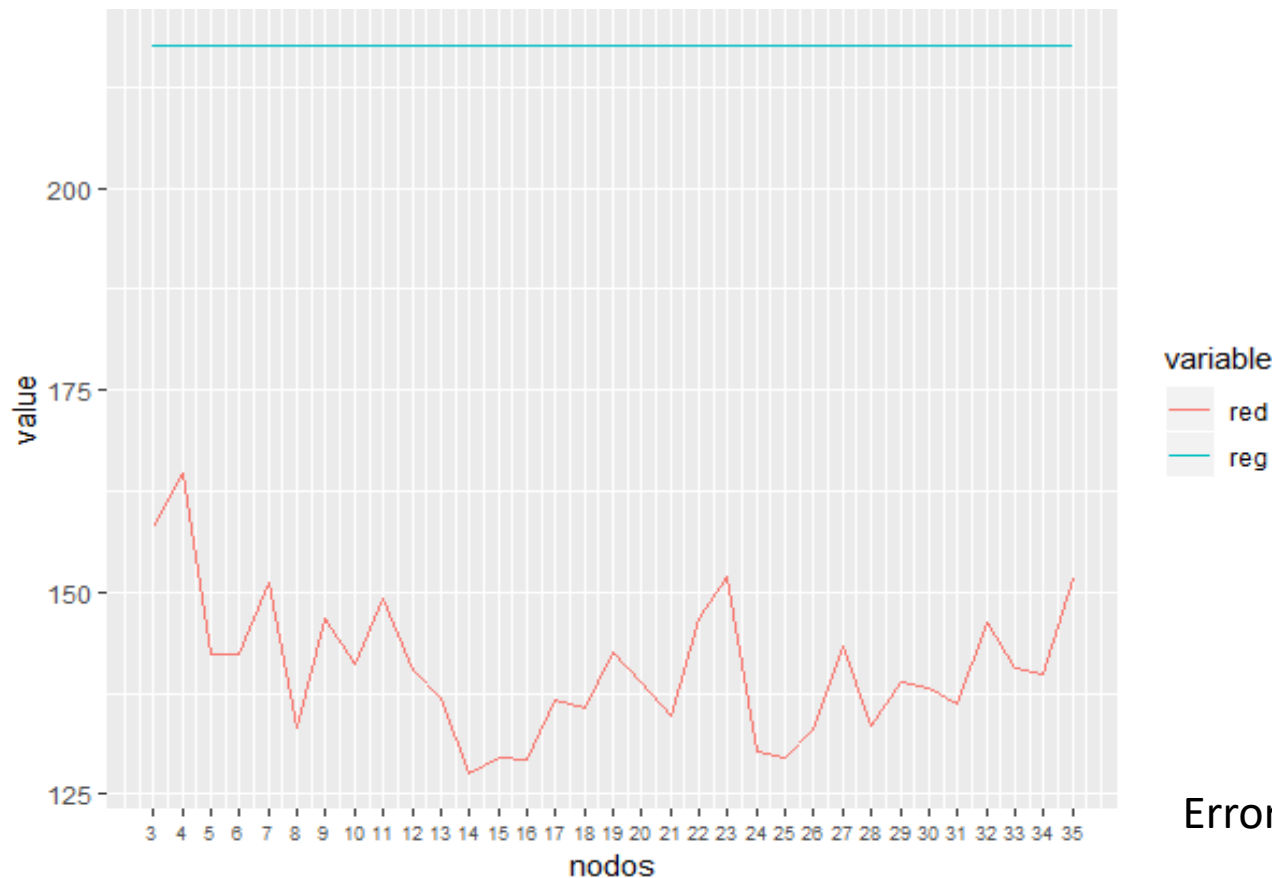
Como $k=2$, despejando $h(k+1)+h+1=1030/20$ obtenemos $4h+1=51.5$, por lo que, como máximo **$h=12$ nodos ocultos**.

Esto no es una regla exacta, pero sirve como referencia inicial
En otro ejemplo, fijando 25 observaciones por parámetro con $k=2$, $h=10$

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

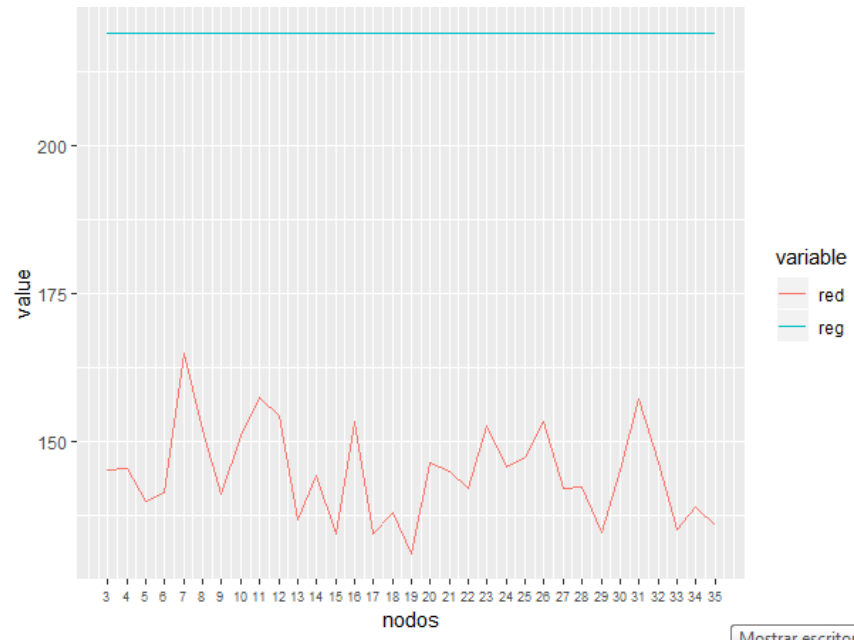
Tomamos como referencia el bucle sobre la semilla *train-test*. Se hace el bucle sobre el número de nodos, pero hay que tener en cuenta que es sobre una prueba *train-test*, y puede depender de esta división. **Semilla train-test 12345: 14 nodos parece bien.**

OJO: el error en datos test siempre está subestimado; al aplicar en datos nuevos será mayor

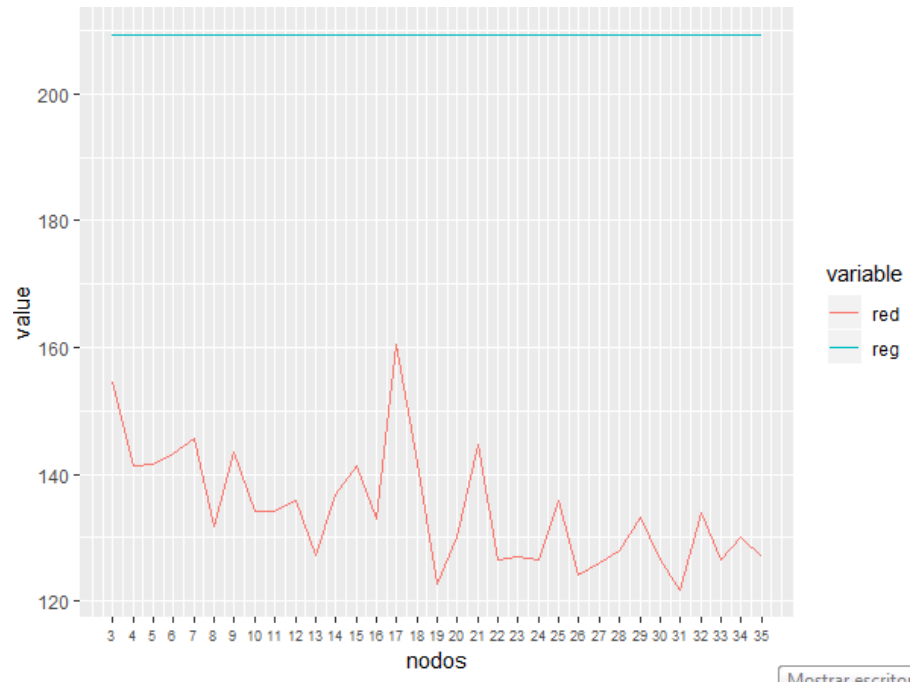


Error de cada modelo

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.

NUNCA HAY UN MODELO ÓPTIMO. Prueba-error, validación cruzada y elegir un modelo bueno con poca variabilidad.



Machine Learning Redes Neuronales

Inmaculada Gutiérrez García-Pardo

8. Redes neuronales para clasificación binaria y regresión

En esta sección se presentan las especificaciones propias de los problemas de predicción más sencillos: clasificación de variable binaria o regresión de variable continua, explicando las peculiaridades de cada uno y las medidas de bondad consideradas.

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**; **logit generalizada** (variable dependiente con más de 2 categorías); **logit acumulativa** (variable dependiente ordinal), o bien **probit** (para estimar la probabilidad de que una observación con características particulares caerá en una categoría específica).

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 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) **OJO:** este punto de corte se puede modificar.

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	Observados	
	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%

- **Sensibilidad** = 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, qué proporción acierto?)
- **Tasa de verdaderos negativos** = $VN/(VN+FN)=0.757$ (cuando digo negativo, qué proporción acierto?)

CURVA ROC

(Receiver Operating Characteristic Curve)

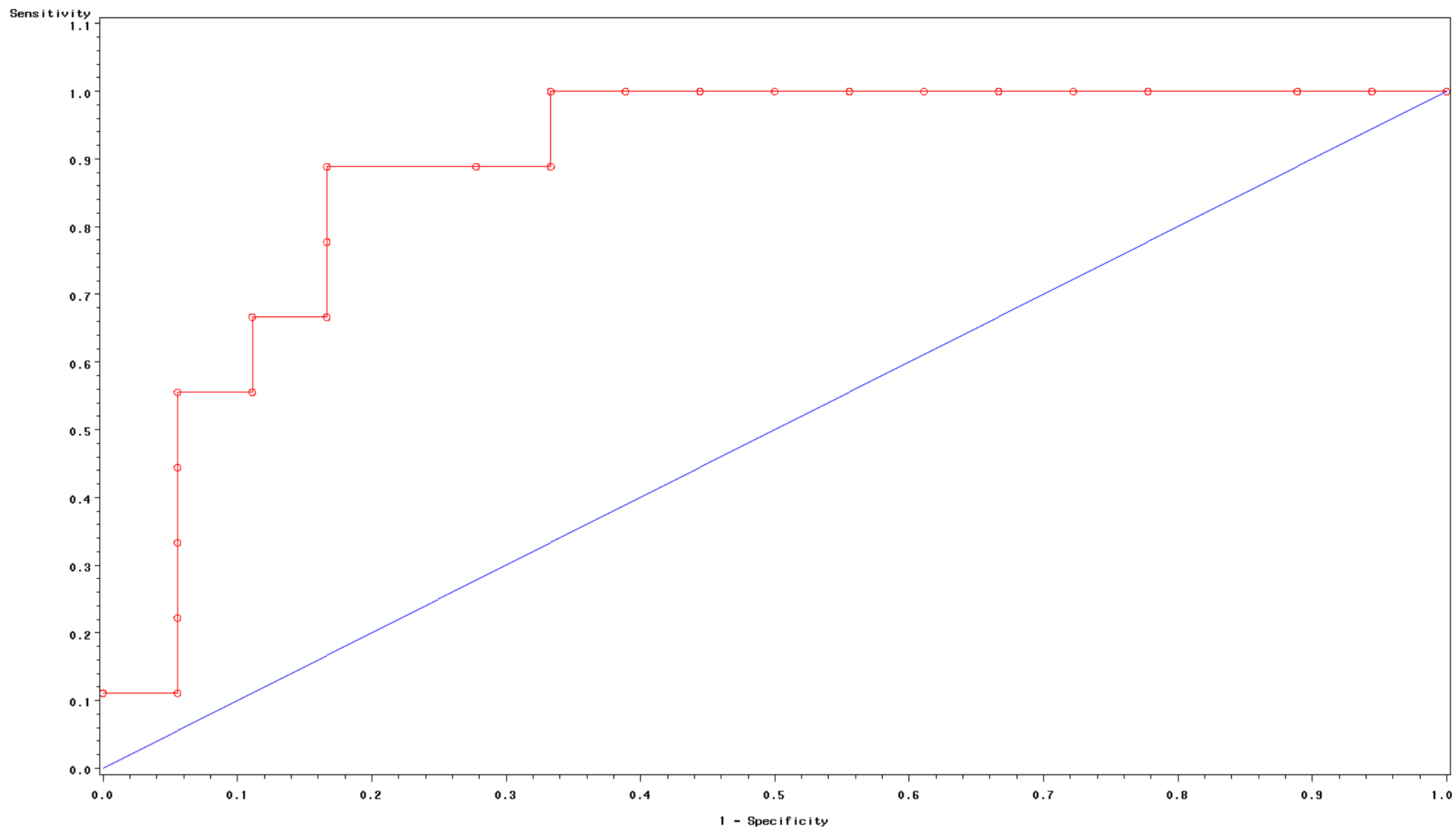
En general, cada punto corresponde a un punto de corte de probabilidad. En el eje X está (*1-Especificidad*), en el eje Y la *Sensibilidad*. Esta representación se hace lo para todos los posibles puntos de corte.

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. Lo ideal es que la curva está lo más próxima posible a los "bordes" (valor de X próximo a 0, para que la sensibilidad sea grande, y valor de Y próximo a 1, para garantizar que cierto

Cada punto de la curva ROC 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**.

Cuando **mayor** sea el **área** de la curva **ROC**, **mejor** será el modelo.



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 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 $y=1$.

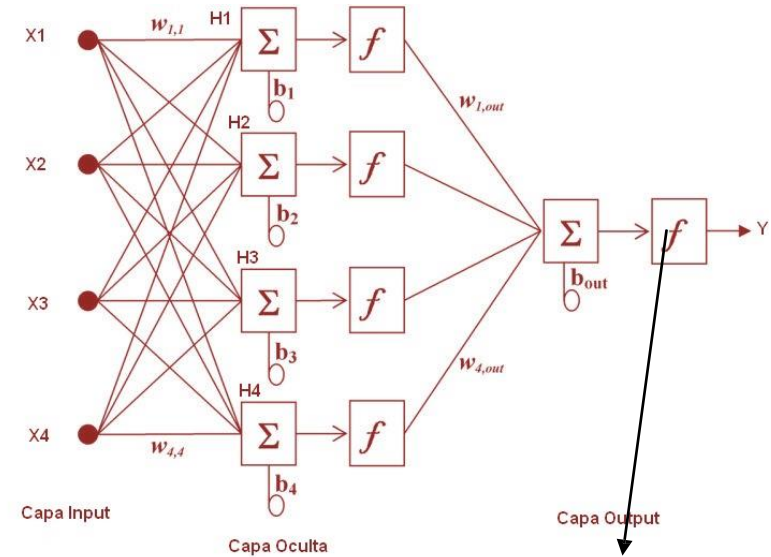
2) La red se plantea técnicamente añadiendo una función de activación de la capa oculta al nodo output, para que el valor resultante (probabilidad estimada de 1) tome valores en el intervalo (0,1). Necesitamos esta función de activación (*tanh* en general), para convertir el resultado de la red neuronal en un valor 0-1. Este paso no es necesario cuando la variable de salida es continua.

Ejemplo básico

chd variable dependiente



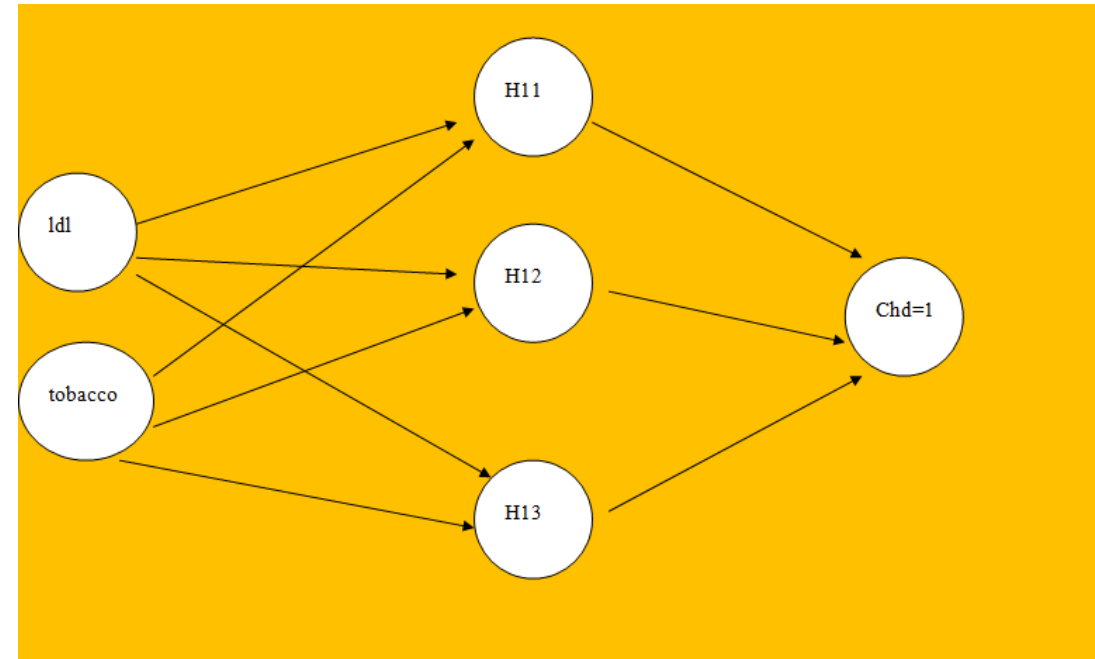
	chd	sbp	tobacco	ldl	adiposity	famhist	typea	obesity	alcohol	age
1	1	160	12.00	5.73	23.11	Present	49	25.30	97.20	52
2	1	144	0.01	4.41	28.61	Absent	55	28.87	2.06	63
3	0	118	0.08	3.48	32.28	Present	52	29.14	3.81	46
4	1	170	7.50	6.41	38.03	Present	51	31.99	24.26	58
5	1	134	13.60	3.50	27.78	Present	60	25.99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

Estas son las únicas diferencias con respecto al tratamiento necesario cuando la variable output es continua.



Problema de regresión

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

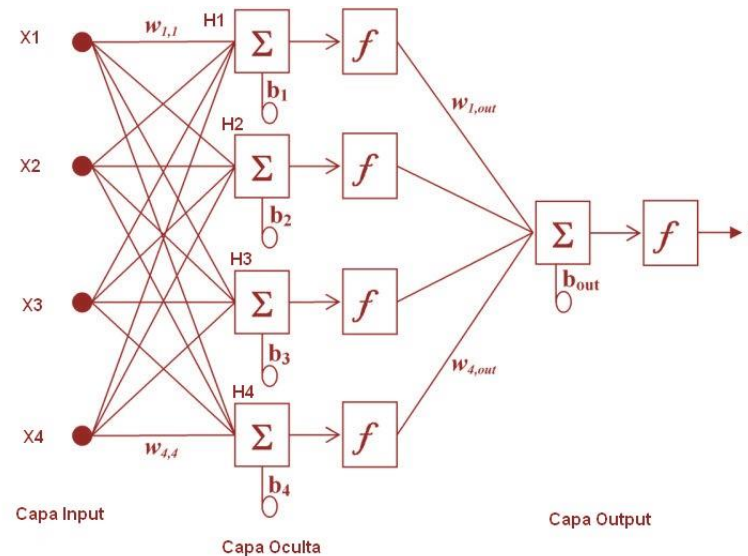
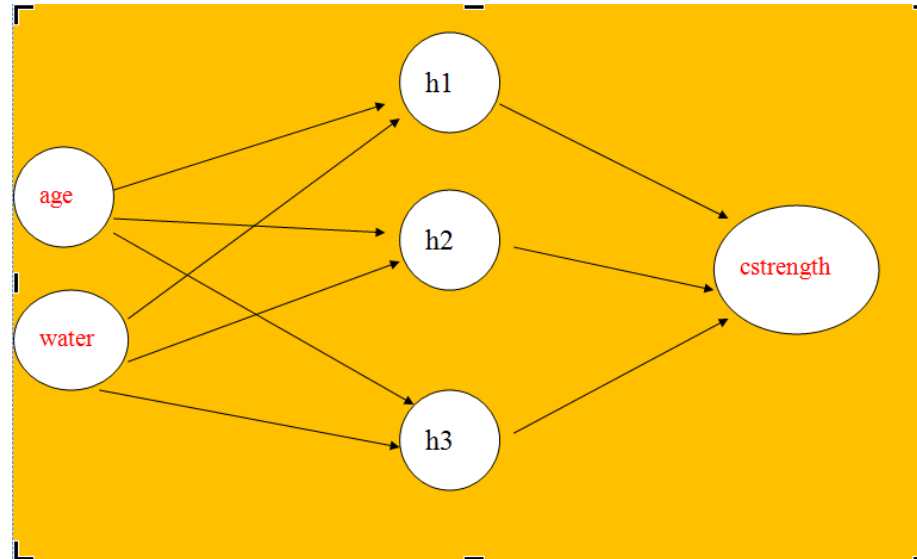
Esto significa:

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

luego activación:

$h1 = \tanh(h1)$

El resto de nodos igual



Red neuronal para regresión

(1) Proceso de optimización-estimación de los pesos. Se observa cómo va descendiendo el valor de la función objetivo a medida que los pesos se estiman mejor según avanzan las iteraciones.

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

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

(algunos paquetes presentan el valor de SSE (sum of squared errors)=n*MSE)

Recordemos:

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

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

Por ejemplo, con los parámetros finales,

$$\hat{Y}_i = 88.52 * \tanh(H1) + 28.69 * \tanh(H2) + 8.51 * \tanh(H3) - 51.42$$

Donde $H1 = 3.31 * age + 0.13 * water + 3.05$; $H2 = 0.13 * age - 1.89 * water - 0.96 \dots etc.$

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

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

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

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

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

Le pedimos que se detuviera en 100 iteraciones y se para el proceso en la última estimación, con un SSE de 140017, que dividiendo por $n=1030$ da $MSE=135.93$.

```
# El objeto red1 contiene el SSE para los datos
originales en el elemento "value" de la lista
# A partir de SSE se pueden calcular el MSE y su raíz
cuadrada
```

```
SSE<-red1[["value"]]
MSE<-SSE/nrow(compress)
RMSE<-sqrt(MSE)
```

```
SSE
MSE
RMSE
```

Redes neuronales para regresión

Nota sobre R^2

Cuando el R^2 (coeficiente de determinación) es **negativo** en un problema de predicción de variable continua, indica que el modelo evaluado es **peor** que un simple modelo que solo predice la **media** de la **variable dependiente**. Esto significa que el modelo no se ajusta bien a los datos y que su **capacidad predictiva** es prácticamente **inexistente** o incluso **peor** que una predicción **aleatoria**.

$$R^2 = 1 - SST/SSE$$

SST y SSE son la Suma Total de Cuadrados y la Suma de los Cuadrado de los Residuos,

$$SST = \sum_{i=1}^n (y_i - \bar{y}_i)^2 \qquad SSE = \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

En general, R^2 puede variar de 0 a 1, donde un valor más cercano a **1** indica un **ajuste** muy **bueno** del modelo a los datos, mientras que un valor cercano a **0** indica que el modelo **no explica** la **variabilidad** de los datos.

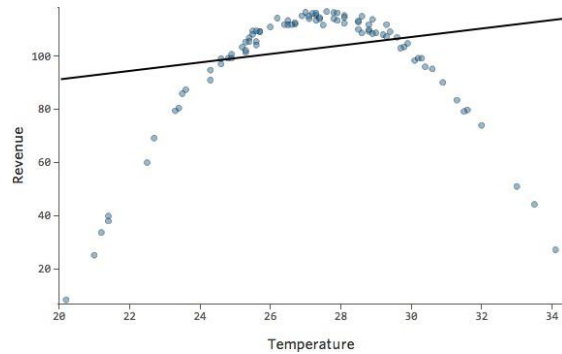
Un valor **negativo** indica que el **modelo** es sustancialmente **peor** que una simple **media** y sugiere que algo está seriamente mal con el ajuste del modelo a los datos.

IMPORTANTE: revisar cuidadosamente el modelo y considerar posibles problemas como multicolinealidad, sobreajuste, errores en los datos o problemas en el diseño del modelo.

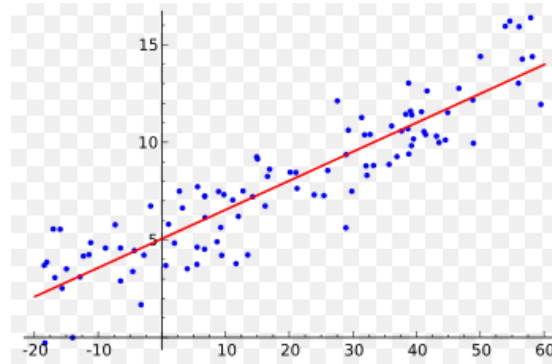
Nota: en clasificación binaria, si la separación entre clases es aproximadamente lineal, es mejor la regresión logística que la red.

Variable dependiente continua

Mejor la red

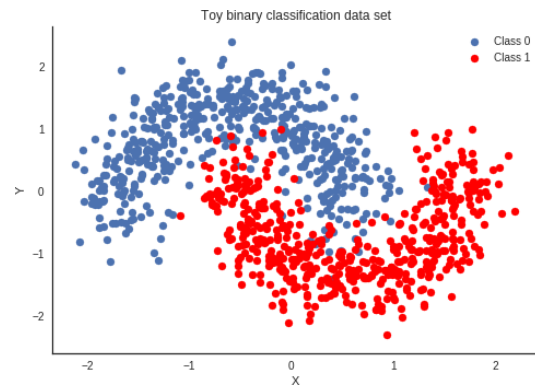


Mejor la regresión lineal

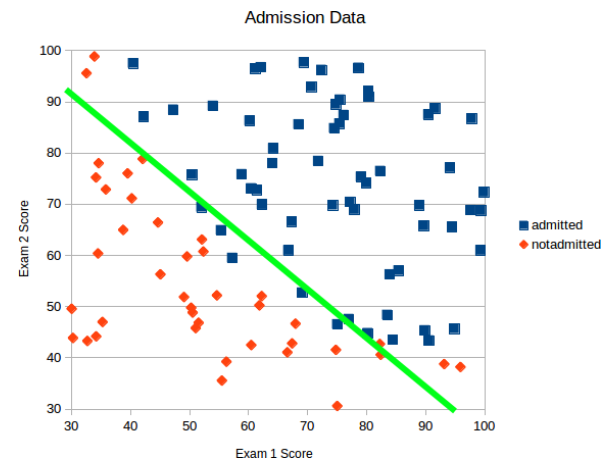


Variable dependiente binaria

Mejor la red



Mejor la regresión logística





Machine Learning

Redes Neuronales

Inmaculada Gutiérrez García-Pardo

9. NN en Python: scikit-learn

En esta sección se presentan aplicaciones sencillas de las redes neuronales en Python, utilizando principalmente la librería scikit-learn

Iniciación a Scikit-Learn / Sklearn

- **¿Qué es?** Una de las bibliotecas de software libre más populares para llevar a cabo tareas relacionadas con el machine learning.
- **¿Qué proporciona?** Extenso soporte para diversos algoritmos de **clasificación, regresión** y agrupación de datos, permitiendo un entrenamiento y evaluación sencillos con una sintaxis amigable y desvinculada de los detalles de implementación.
- **¿Valor añadido?** Incluye métodos para el **preprocesamiento y transformación** de datos antes del entrenamiento, lo que, junto con la diversidad de modelos disponibles, facilita una **experimentación rápida y efectiva**

<https://scikit-learn.org/stable/index.html>

¿Qué puedo hacer con scikit-learn?

- **Depuración de datos:** imputación de valores perdidos, codificación de etiquetas, codificación one-hot, normalización y reducción de la dimensionalidad.
- **Selección** de características
- Aplicar de forma **fácil y directa**: árboles de decisión, máquinas de soporte vectorial (SVM), redes neuronales, K-nearest neighbor, naive bayes, regresión lineal, regresión logística, random forest, gradient boosting, y algunas técnicas de aprendizaje no supervisado como K-means, DBSCAN, BIRCH y clustering aglomerativo, entre otros.
- **Pipelines** para aplicar procesos de manera coherente y consistente a lo largo del desarrollo.
- **Evaluación de modelos:** métricas comunes integradas de manera simple para facilitar la selección de modelos mediante búsqueda de hiperparámetros.

Básicos de scikit-learn

1. Modelos de Aprendizaje Supervisado:

- **Redes neuronales para clasificación/regresión (MLPClassifier/MLPRegressor):** herramientas básicas útiles para tareas sencillas, pero con menos flexibilidad y opciones que otras librerías como TensorFlow o PyTorch. Scikit-learn permite trabajar con MLP *feedforward**
- **Regresión Lineal (LinearRegression):** Para modelar la relación lineal entre variables.
- **Regresión Logística (LogisticRegression):** Para problemas de clasificación binaria.
- **Support Vector Machines (SVM):** Para clasificación y regresión, especialmente efectivo en espacios de alta dimensión.
- **Random Forest (RandomForestClassifier, RandomForestRegressor):** Para clasificación y regresión mediante conjuntos de árboles de decisión.
- **K-Nearest Neighbors (KNeighborsClassifier, KNeighborsRegressor):** Para clasificación y regresión basada en la proximidad a puntos vecinos.

2. Modelos de Aprendizaje No Supervisado:

- **K-Means (KMeans):** Para agrupar datos en clústeres.
- **DBSCAN (DBSCAN):** Para detectar clústeres de forma no paramétrica.
- **Principal Component Analysis (PCA):** Para reducción de la dimensionalidad.
- **Factorización de Matrices No Negativas (NMF):** Para la descomposición de matrices no negativas

**feedwordard*: redes conectadas de atrás hacia la salida sin bucles ni comunicación entre redes de la misma capa

Básicos de scikit-learn

3. Herramientas de Preprocesamiento y Transformación de Datos:

- **StandardScaler, MinMaxScaler:** Para normalización y escalado de características.
- **LabelEncoder, OneHotEncoder:** Para codificación de variables categóricas.
- **Imputer:** Para imputar valores faltantes en los datos.

4. Evaluación y Métricas:

- **cross_val_score, train_test_split:** Para evaluar modelos mediante validación cruzada y dividir conjuntos de entrenamiento/prueba.
- **accuracy_score, precision_score, recall_score, f1_score:** Métricas comunes para evaluar el rendimiento de clasificadores.
- **mean_squared_error, r2_score:** Métricas para evaluar el rendimiento de modelos de regresión.

5. Pipeline: para encadenar múltiples pasos de procesamiento y modelado de datos en un flujo único.

6. Herramientas para Selección de Modelos:

GridSearchCV, RandomizedSearchCV: Para la búsqueda sistemática de hiperparámetros óptimos

Métodos scikit-learn https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html

<u>fit</u>(X, y)	Ajusta el modelo a la matriz de datos (input) X, para el (los) objetivo(s) y
<u>get_metadata_routing</u> ()	Obtén la ruta de metadatos de este objeto.
<u>get_params</u>([deep])	Obtén los parámetros de este estimador
<u>partial_fit</u> (X, y[, classes])	Actualiza el modelo con una sola iteración sobre los datos daos
<u>predict</u>(X)	Predice utilizando el clasificador MLP entrenado con X.
<u>predict_log_proba</u> (X)	Logaritmo de las estimaciones de probabilidad
<u>predict_proba</u>(X)	Probabilidades estimadas
<u>score</u> (X, y[, sample_weight])	Precisión media en los datos de prueba y las etiquetas prorpocionadas
<u>set_params</u>(**params)	Establecer los parámetros de un estimador
<u>set_partial_fit_request</u> (*[, classes])	Obtener los parámetros proporcionados al método <code>partial_fit</code>
<u>set_score_request</u> (*[, sample_weight])	Obtener metadatos pasados al método <code>score</code>

MLPClassifier

MLPClassifier se entrena **iterativamente**, ya que en cada paso de tiempo se calculan las derivadas parciales de la función de pérdida con respecto a los parámetros del modelo para actualizarlos.

Puede incluir un **término de regularización** agregado a la función de pérdida que contrae los parámetros del modelo para **evitar el sobreajuste**.

Esta implementación funciona con datos representados como matrices densas de numpy o matrices dispersas de scipy de valores de punto flotante.

MLPClassifier: algunos parámetros importantes

- **hidden_layer_sizes**: array de tamaño *número de capas ocultas*, que en la posición i contiene un entero que se corresponde con el número de neuronas de la capa i . *Por defecto: (100,)* una capa oculta con 100 neuronas; (10, 15,) dos capas ocultas con 10 y 15 neuronas, respectivamente
- **activation**: función de activación, puede tomar los valores *'identity', 'logistic', 'tanh', 'relu'*. *Por defecto: 'relu'*
- **random_state**: número aleatorio considerado (se recomienda fijarlo al inicio), afecta a la división *train/test*, a la asignación inicial de pesos aleatorios y sesgos, etc.
- **solver**: motor del algoritmo de optimización de pesos (*'lbfgs', 'sgd', 'adam'*). *Por defecto: 'adam'*. *'adam'* funciona bastante bien en conjuntos de datos relativamente grandes (con miles de muestras de entrenamiento o más) en términos tanto del tiempo de entrenamiento como de la puntuación de validación. Sin embargo, para conjuntos de datos pequeños, *'lbfgs'* puede converger más rápido y tener un mejor rendimiento.
- **max_iter**: número de iteraciones máximas (valor entero) permitidas al algoritmo de optimización, que itera hasta la convergencia (determinada por *tol*) o hasta alcanzar este número. Para algoritmos estocásticos como *sgd* o *adam*, este parámetro determina el número de *epochs*
- **alpha**: parámetro de regularización L2, valor decimal, *por defecto: 0.0001*
- **learning_rate_init**: estrategia de para la actualización de pesos, valores muy altos actualizaciones muy bruscas (se puede saltar algún óptimo); valores muy bajos, actualizaciones muy lentas (puede no converger nunca). *Por defecto: 0.001*. Usar sólo con *'sgd'* o *'adam'*

MLPClassifier: algunos parámetros importantes

- **early_stopping:** uso o no de detención temprana para finalizar el entrenamiento cuando la puntuación de validación no mejora. Si *early_stopping=True*, reservará automáticamente el 10% de los datos de entrenamiento como validación y detendrá el entrenamiento cuando la puntuación de validación no mejore al menos 'tol' durante '*n_iter_no_change*' épocas consecutivas. La división es estratificada, excepto en un entorno multietiqueta. Si *early_stopping=False* (por defecto), el entrenamiento se detiene cuando la pérdida de entrenamiento no mejora en más de 'tol' durante '*n_iter_no_change*' pasadas consecutivas sobre el conjunto de entrenamiento. Usar con 'sgd' o 'adam'.

MLPClassifier: otros parámetros

learning_rate (para modificar la forma de actualizar los pesos); *power_t* (para modificar de forma efectiva el learning rate; *batch_size* (tamaño de los minibatches en optimizadores estocásticos, 200 por defecto, se analiza con profundidad en Deep Learning); *shuffle* (aleatorizar o no las muestras en cada iteración); *tol* (tolerancia para la optimización, ganancia mínima exigida para que el algoritmo siga iterando); *beta_1* y *beta_2* (tasa de decay para estimar los vectores de primer y segundo momento bajo el optimizador *adam*); *validation_fraction* (si se quiere modificar la cantidad de datos usada en *early_stopping*); ETC.

MLPRegressor

Misma estructura y parámetros que MLPClassifier, pero destinado a resolver problemas de regresión con redes neuronales
Únicamente hay que tener en cuenta la medida de bondad de ajuste utilizada: MAE, MSE, R2

Más información y ejemplos en los archivos *.ipynb*:

Redes_clasificación_1.ipynb

Redes_clasificación_2.ipynb

Redes_regresión_1.ipynb

Redes_regresión_2.ipynb

Redes neuronales en python

Medidas de evaluación para el parámetro scoring

Métricas de clasificación:

Precisión: 'accuracy'

Sensibilidad: 'recall' o 'recall_macro'

F1-score: 'f1' o 'f1_macro'

Área bajo la curva ROC (AUC-ROC): 'roc_auc'

NO existe una métrica concreta para la especificidad

Métricas de regresión:

Error cuadrático medio (MSE): 'neg_mean_squared_error'

R-cuadrado (R^2): 'r2'

Error absoluto medio (MAE): 'neg_mean_absolute_error'

Otras métricas:

Log-verosimilitud (para modelos de clasificación probabilística): 'neg_log_loss'

Coeficiente de determinación ajustado (Adjusted R-squared): 'adjusted_rand_score' (para clustering)

Cosas a tener en cuenta:

Para calcular la curva ROC, las predicciones se deben calcular en forma de probabilidad, para ello, predict_proba

En problemas de clasificación binaria, si la variable objetivo tiene una codificación distinta a 1-0, hay que especificar al calcular las medidas de bondad cuál es la etiqueta positiva

Datasets de prueba

- Archivos populares de Kaggle (adjuntos en el material de este módulo): SAheart, compress, Life Expectancy Data, Ozone
- Otros archivos de Kaggle: ameshousing, autompg, bank, california housing, magic (<https://www.kaggle.com/>)
- Archivos scikit-learn: puedes acceder a ellos usando el siguiente código

```
from sklearn import datasets
# Ejemplos de conjuntos de datos
iris = datasets.load_iris()
digits = datasets.load_digits()
wine = datasets.load_wine()
```

- UCI Machine Learning Repository: puedes acceder a los datos mediante descargas o cargarlos directamente desde la web. Puedes encontrar más información en UCI ML Repository (<https://archive.ics.uci.edu/>)
- OpenML: plataforma en línea que permite a los usuarios compartir conjuntos de datos para el aprendizaje automático (<https://www.openml.org/>).
- TensorFlow Datasets: puedes acceder a ellos usando el siguiente código:

```
import tensorflow as tf
import tensorflow_datasets as tfds
# Lista de conjuntos de datos disponibles
datasets_list = tfds.list_builders()
# Imprimir la lista de conjuntos de datos
print("Conjuntos de datos disponibles:")
for dataset in datasets_list:
    print(dataset)
# Cargar un conjunto de datos, por ejemplo, 'mnist'
dataset_name = 'mnist' (train_data, test_data), info = tfds.load(name=dataset_name, split=['train', 'test'], with_info=True)
# Imprimir información sobre el conjunto de datos
print(info)
```

Redes neuronales en Python (Keras: (1) 'tensorflow' + (2) 'keras')

Cuidado con el cálculo de las medidas de bondad

Algunas funciones están preparadas para funcionar con 1-0, otras con si-no, y depende de las necesidades propias de cada clasificador.

El modelo Sequential() (Keras) necesita que la variable objetivo esté codificada 1-0, sin embargo las funciones, ConfusionMatrixDisplay, y las distintas de los score necesitan un evidente clasificador. Por tanto, hay que buscar otras opciones para calcular las medidas de bondad SIEMPRE SOBRE DATO TEST

Tuneo hiperparámetros numéricos (número de nodos capa oculta y decay): *hp.Int('hiperparámetro', min_value=, max_value=, step=)*, a la que se le dice qué hiperparámetro se está buscando, qué valor mínimo y máximo del mismo se considerarán y el "salto" de un valor a otro

kt.Hyperband (biblioteca keras-tuner): se utiliza para configurar y ejecutar una búsqueda de hiperparámetros utilizando el algoritmo *Hyperband*, que busca de manera eficiente el conjunto óptimo de hiperparámetros:

hypermodel: función que define el modelo de red neuronal. Debes pasar una función que tome un objeto hp como argumento y devuelva un modelo de Keras construido según los hiperparámetros especificados.

objective: métrica que se intenta optimizar en la búsqueda paramétrica, "val_accuracy" para precisión en el conjunto de validación o "val_loss" para pérdida en el conjunto de validación.

Seed: semilla de aleatorización

Otros parámetros: *max_epochs; Hyperband; factor; directory; project_name; tune_new_entries; allow_new_entries*

Ejemplo desarrollado en los archivos Redes_clasificación_4_keras.ipynb

Bibliografía

- Jason Brownlee. *Machine Learning Mastery with R. Get started, build accurate models and work through projects step-by-step.*
- G.Ciaburro and B.Venkateswaran. *Neural Networks with R: Smart models using CNN, RNN, deep learning, and artificial intelligence principles.* ISBN-13: 978-1788397872.
- Bishop, C.M. (1995), *Neural Networks for Pattern Recognition*, Oxford: Oxford University Press.
- Hastie, T., Tibshirani, R., Friedman, J. (2009): *The Elements of Statistical Learning: data mining, inference and prediction.*
- Davis, L. (1991): *Handbook of Genetic Algorithms*. Van Nostrand Reinhold
- Jason Brownlee. *Statistical Methods for Machine Learning. Discover how to transform data into knowledge with Python.*
- Michie, D., Spiegelhalter, D.J., Taylor, C.C. (1994) *Machine Learning, Neural and Statistical Classification.*