ntic master
revolucionamos la comunicación

UNIVERSIDAD COMPLUTENSE DE MADRID

Machine Learning
Redes Neuronales
IMPRESCINDIBLES

Inmaculada Gutiérrez García-Pardo

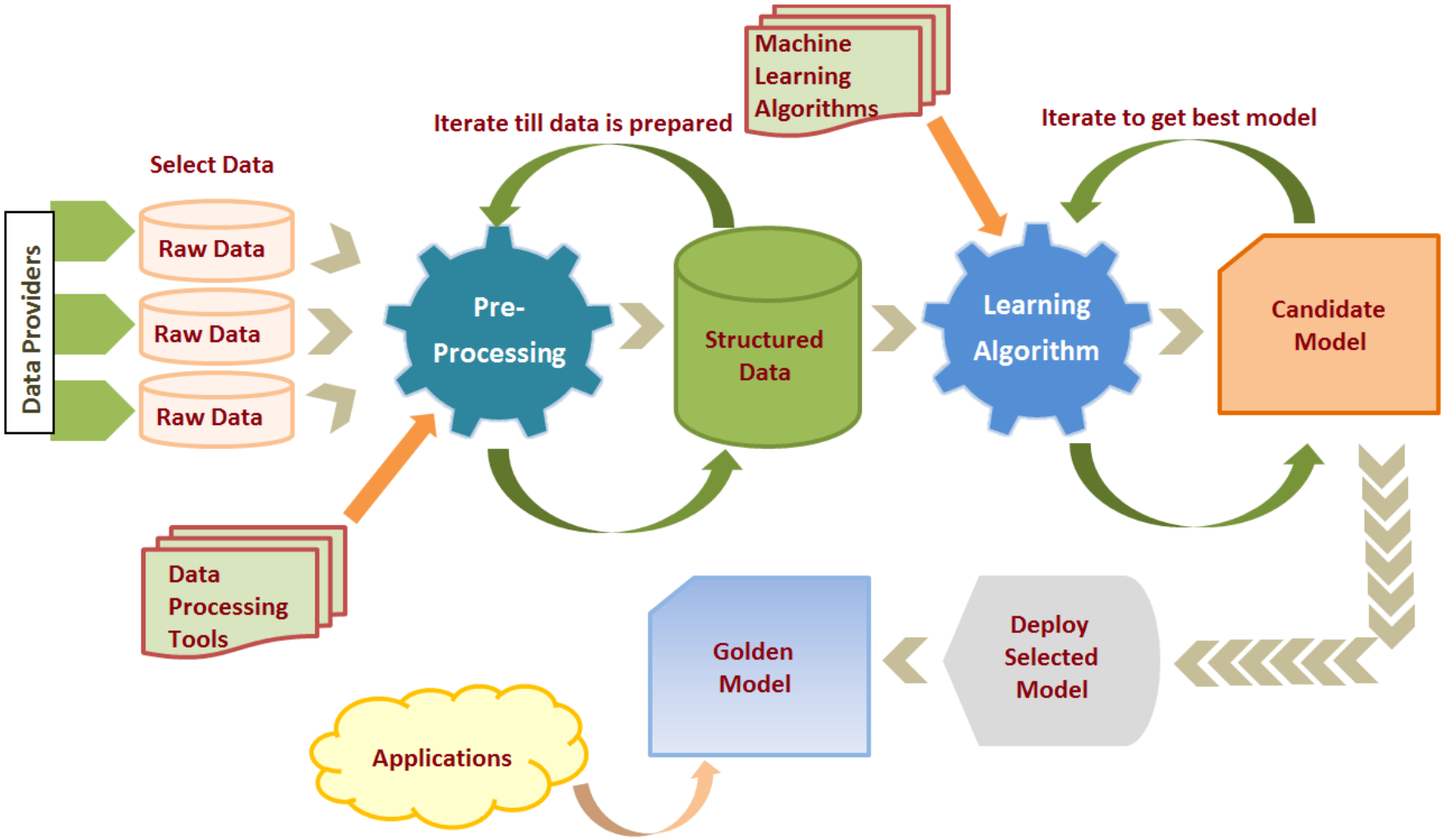
ntic master
revolucionamos la comunicación

UNIVERSIDAD COMPLUTENSE DE MADRID

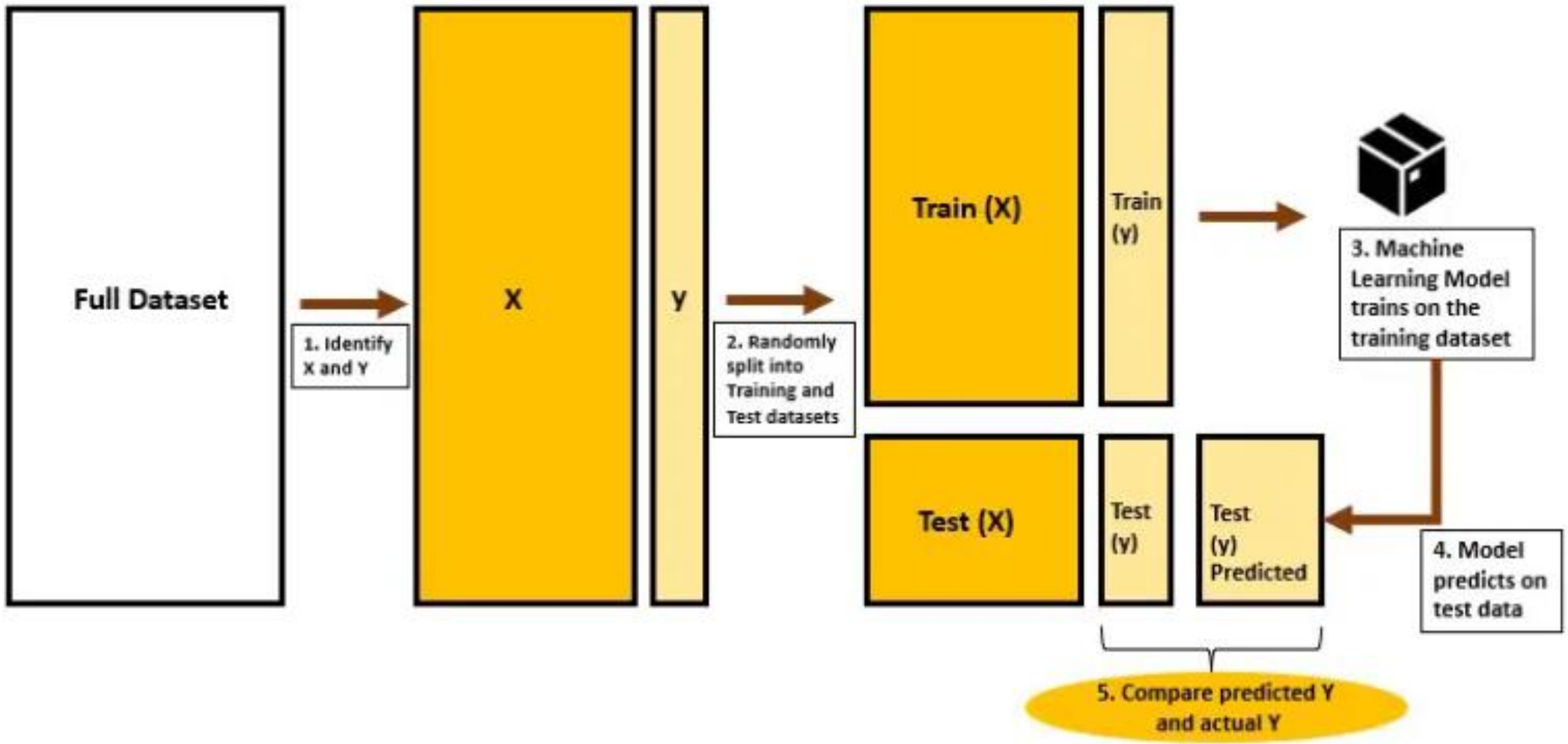
Machine Learning
Redes Neuronales
IMPRESCINDIBLES

Inmaculada Gutiérrez García-Pardo

Sobre el proceso de ML



Sobre la estructura de validación



Sobre la estructura de validación

VALIDACIÓN CRUZADA (k grupos)

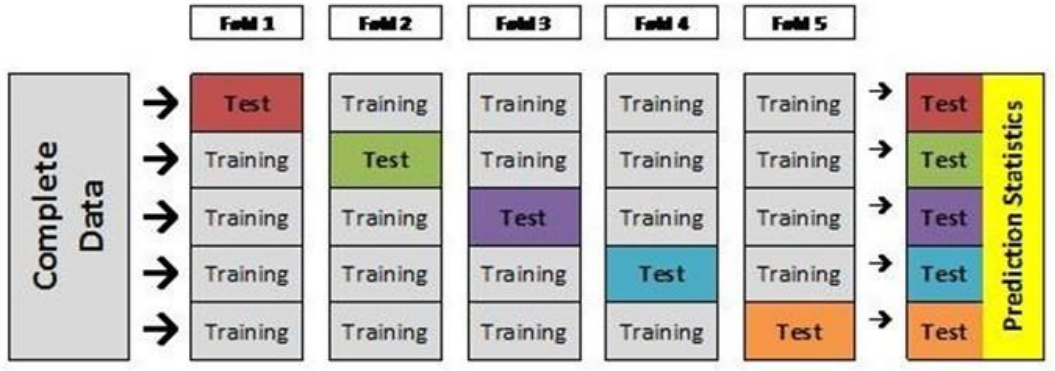
Si no se tienen muchas observaciones, la división en Training-Validación-Test arrojará resultados muy variables según la selección aleatoria de esos grupos.

Algoritmo de validación cruzada

1) Se dividen los datos aleatoriamente en *k* grupos. La idea es que **todos los datos pasen** por alguna fase de **train** y por alguna de **test**.
2) Se realiza iterativamente:

Desde i=1 hasta k
Dejar aparte el grupo i;
Construir el modelo con los grupos restantes;
Estimar el error (por ejemplo ASE) al predecir el grupo i: $Error_i$
Fin

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



Con distintos conjuntos de train/test, usamos la validación cruzada para evaluar cómo se comporta el modelo

Medidas de bondad de ajuste: Clasificación

Indican cómo de bien el modelo predice las **clases** en comparación con los **datos reales**. Se mide usando métricas específicas, dependiendo de lo que quieras evaluar

- **MATRIZ DE CONFUSIÓN:** resumen de las predicciones correctas e incorrectas de cada clase

	PREDICCIÓN (Clase 0)	PREDICCIÓN (Clase 1)
REAL (Clase 0)	Verdadero Negativo (VN/TN)	Falso Positivo (FP/FP)
REAL (Clase 1)	Falso Negativo (FN/FN))	Verdadero Positivo (VP/TP)

- **ACCURAY (Precisión Global):** porcentaje de predicciones correctas sobre el total

$$Accuracy = \frac{VP + VN}{Total\ predicciones = VP + VN + FP + FN}$$

- **PRECISIÓN (Precisión por Clase):** ¿cuántas predicciones han sido correctas? Especialmente importante cuando los datos son desbalanceados.

$$Precision = \frac{VP}{VP + FP}$$

- **RECALL/SENSIBILIDAD:** qué porcentaje de los casos positivos reales son correctamente clasificados

$$Precision = \frac{VP}{VP + FN}$$

- **F1-SCORE:** promedio armónico entre precisión y sensibilidad. Especialmente importante cuando los datos son desbalanceados.

$$F1 = 2\ x\ \frac{Precision\ x\ Recall}{Precision + Recall}$$

Medidas de bondad de ajuste: Clasificación

- **ESPECIFIDAD (Tasa Verdaderos Negativos):** mide qué proporción de los negativos reales son correctamente identificados como negativos. Se enfoca en los negativos reales

$$Especifidad = \frac{VN}{VN + FP}$$

- **La Curva ROC:** representa el porcentaje de verdaderos positivos (**Eje Y: True Positive Rate, *TPR***), también conocido como **Recall**, contra el ratio de falsos positivos (**Eje X: False Positive Rate, *FPR=1-Especifidad***). muestra gráficamente cómo de bien un modelo de clasificación puede separar dos clases al cambiar el umbral de decisión

$$Recall = TPR = \frac{VP}{VP + FN}; \qquad 1 - Especifidad = FRP = \frac{FP}{VN + FP}$$

- **AUC** (*Área Bajo la Curva/Area Under Curve*): métrica sólida y muy útil para problemas de clasificación binaria. Toma valores entre 0 y 1, donde 0.5 indica el resultado de una clasificación aleatoria, 0 una clasificación aún peor que el azar, y 1 indica el resultado de la clasificación perfecta.

Medidas de bondad de ajuste. ¿Cómo afectan los ajustes?

Ajuste de parámetros: afecta a la capacidad general del modelo para adaptarse a los datos.

- **Sensibilidad (TPR o Recall):** si el modelo es demasiado restrictivo o simple (bajo ajuste), podría no detectar correctamente los positivos reales, disminuyendo la sensibilidad. *Ejemplo*: En k-NN, un valor muy grande de k podría suavizar demasiado las predicciones, afectando la detección de positivos.

- **Especificidad:** un modelo muy complejo (sobreajustado) podría clasificar erróneamente más negativos reales, disminuyendo la especificidad. *Ejemplo*: Un árbol de decisión muy profundo podría clasificar demasiados ejemplos negativos como positivos.

- **Precisión:** los parámetros que reducen los falsos positivos aumentan la precisión, ya que esta mide qué porcentaje de las predicciones positivas son correctas. *Ejemplo*: En SVM, aumentar el valor de C puede reducir los falsos positivos, aumentando la precisión.

- **F1-Score:** como combina precisión y sensibilidad, un cambio que mejore una métrica pero afecte negativamente la otra podría reducir el F1-Score.

Medidas de bondad de ajuste. ¿Cómo afectan los ajustes?

Threshold o punto de corte

El *threshold* define a partir de qué probabilidad un ejemplo se clasifica como positivo. Cambiar este umbral tiene los siguientes efectos:

- **Sensibilidad (TPR)**
  - **Umbral bajo:** Detecta más positivos, aumentando la sensibilidad, pero podría incrementar los falsos positivos.
  - **Umbral alto:** Detecta menos positivos, reduciendo la sensibilidad, pero mejora la especificidad.
- **Especificidad**
  - **Umbral bajo:** Clasifica menos negativos correctamente, reduciendo la especificidad.
  - **Umbral alto:** Clasifica más negativos correctamente, aumentando la especificidad.
- **Precisión**
  - **Umbral bajo:** Aumenta los falsos positivos, reduciendo la precisión.
  - **Umbral alto:** Reduce los falsos positivos, aumentando la precisión.
- **F1-Score:** cualquier cambio extremo puede desbalancear sensibilidad y precisión, reduciendo el F1-Score. Se maximiza cuando el balance entre ambas métricas es óptimo.

Medidas de bondad de ajuste. ¿Cuál escoger?

- **Prioridad: minimizar FALSOS NEGATIVOS (ALTA SENSIBILIDAD):** no identificar un positivo es más costoso/grave que clasificar incorrectamente un negativo. *Ejemplos*:
  - Diagnóstico médico: es más grave no identificar un paciente con cáncer (falso negativo) que generar una alerta innecesaria (falso positivo).
  - Seguridad: identificar correos con malware. No marcar un malware como peligroso puede hundir un sistema.
  - **Métrica recomendada: recall/sensibilidad**, si fuera necesario equilibrar, **F1**.
- **Prioridad: minimizar FALSOS POSITIVOS (ALTA PRECISIÓN):** clasificar un negativo como positivo tiene un alto coste/gravedad *Ejemplos*:
  - Detección de fraude: clasificar una transacción legítima como fraudulenta puede provocar la huida de clientes.
  - Clasificación de spam: identificar un correo importante como spam (falso positivo), puede provocar que el usuario pierda información importante.
  - **Métrica recomendada: precisión**, si fuera necesario equilibrar, **F1**.
- **Prioridad: BALANCEAR FALSOS POSITIVOS Y FALSOS NEGATIVOS:** es importante evitar tanto los falsos positivos como los falsos negativos. *Ejemplos*:
  - Reconocimiento facial: en accesos restringidos, un error podría negar la entrada a un usuario legítimo (falso negativo) o permitir el acceso a un intruso (falso positivo)
  - Sistemas de recomendación: clasificar incorrectamente un producto como relevante o no relevante afecta la experiencia del usuario.
  - **Métrica recomendada: F1**, para considerar el equilibrio entre precisión y sensibilidad, **Accuracy** (si las clases están más o menos balanceadas, **ROC-AUC**.
- **Problemas con clases DESBALANCEADAS: cuando** una clase es mucho más frecuente que la otra, las métricas estándar como la exactitud (accuracy) pueden ser engañosas. *Ejemplos*:
  - Detección de fallos en maquinaria: si los fallos son eventos raros (clase minoritaria). Un modelo que prediga siempre "no falla" podría tener alta exactitud pero sería inútil.
  - Sistemas de recomendación: clasificar incorrectamente un producto como relevante o no relevante afecta la experiencia del usuario.
  - **Métrica recomendada: ROC-AUC** para medir la capacidad del modelo de diferenciar entre las clases. También se puede usar **Precision-Recall AUC** si la clase positiva es extremadamente minoritaria.
- **Problema de clasificación MULTI-CLASE:** hay más de una clase y todas tienen una importancia relativa. *Ejemplos*:
  - **Clasificación de noticias:** Etiquetar artículos en categorías como deportes, política, tecnología.
  - **Reconocimiento de imágenes:** Identificar objetos en imágenes (perro, gato, coche, etc.).
  - **Métrica recomendada: Acuracy**, si las clases están balanceadas; **Macro F1-Score si**i las clases están desbalanceadas y se quiere evaluar el desempeño promedio en todas las clases.

Medidas de bondad de ajuste: Regresión

1. **Error Cuadrático Medio (MSE)**:

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

donde: - y_i son los valores reales. - $\hat{y}_i$ son los valores predichos. - n es el número total de observaciones.

2. **Raíz del Error Cuadrático Medio (RMSE)**:

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

La diferencia principal con el MSE es que el RMSE toma la raíz cuadrada del MSE, lo que lo hace más interpretable al estar en las mismas unidades que la variable objetivo.

3. **Error Absoluto Medio (MAE)**:

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

Aquí, la diferencia es simplemente la media de las diferencias absolutas entre los valores reales y los predichos.

4. **R^2 (Coeficiente de Determinación)**:

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

donde: - $\bar{y}$ es la media de los valores reales. - El numerador es la suma de los cuadrados de los errores (SSE), y el denominador es la suma de los cuadrados totales (SST).

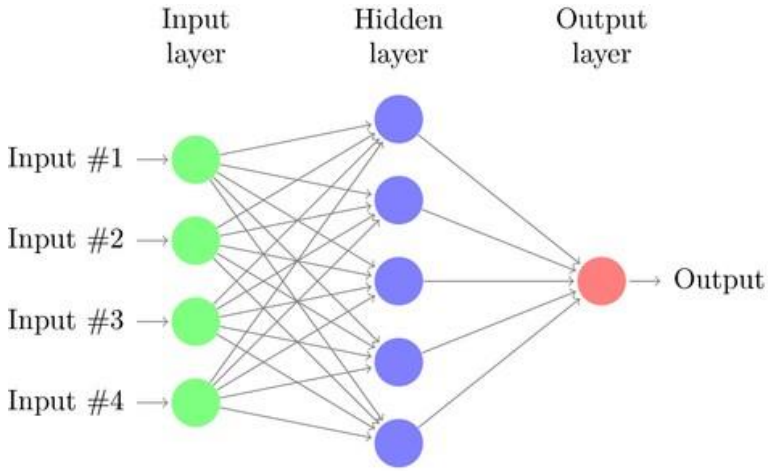
Red Neuronal Artificial. Algunas nociones básicas

- 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

- **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

$$y=f(x1,x2,x3,\ldots)$$

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)

¿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.

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**.

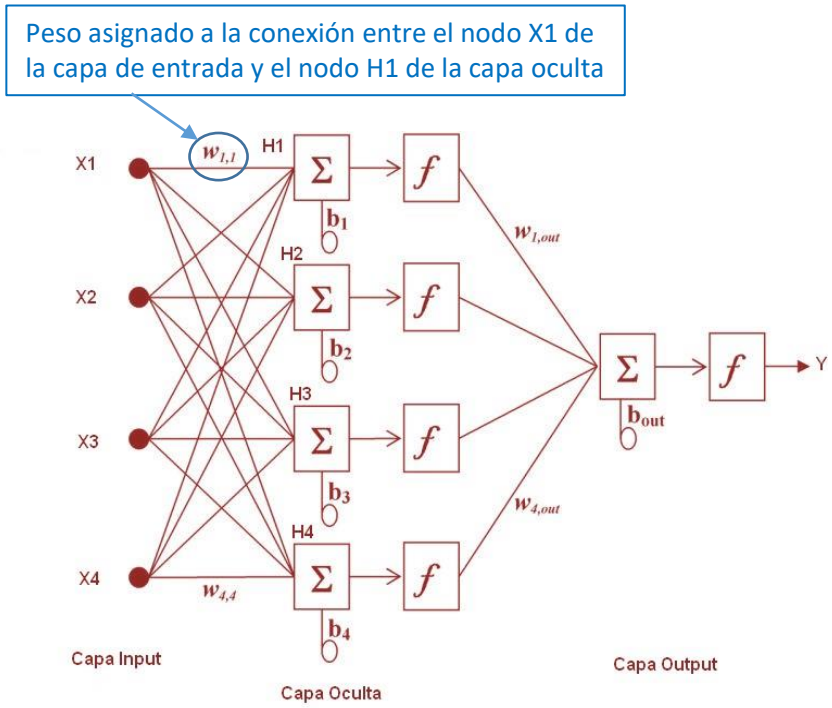
Fase de Prueba: en la fase anterior, el modelo puede que se ajuste demasiado a las particularidades presentes en los patrones de entrenamiento, perdiendo su habilidad de generalizar su aprendizaje a casos nuevos (**sobreajuste**). Para evitar el problema del sobreajuste, es aconsejable utilizar un segundo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

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.



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.

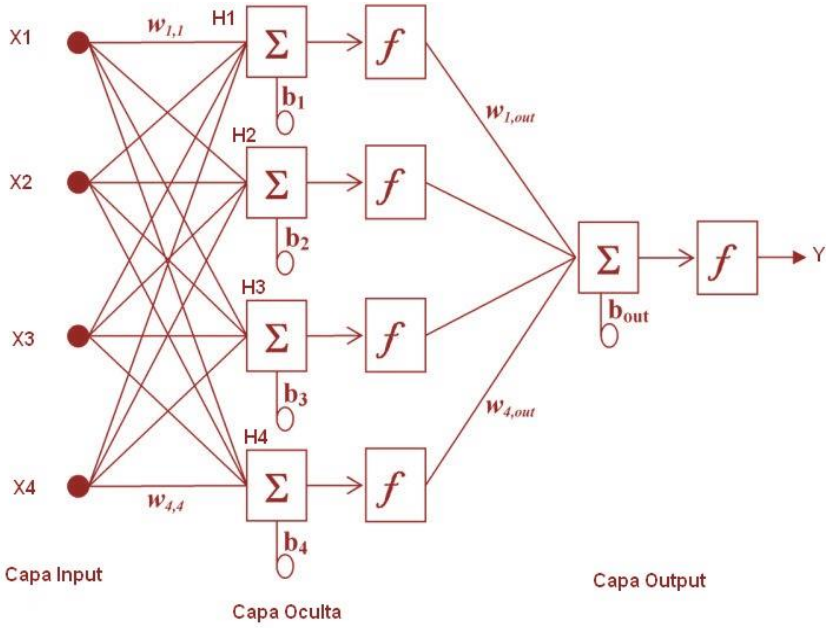
Red neuronal con 4 inputs X1,…,X4, un output Y, con una capa oculta con 4 nodos ocultos H1, H2, H3, H4.

Los nodos de la capa de entrada se conectan con los nodos de la capa oculta mediante una **función de combinación** $\sum$ 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:

$$H1=w_{11}X1+w_{21}X2+w_{31}X3+w_{41}X4+b_1$$
$$H2=w_{12}X1+w_{22}X2+w_{32}X3+w_{42}X4+b_2$$
$$H3=w_{13}X1+w_{23}X2+w_{33}X3+w_{43}X4+b_3$$
$$H4=w_{14}X1+w_{24}X2+w_{34}X3+w_{44}X4+b_4$$



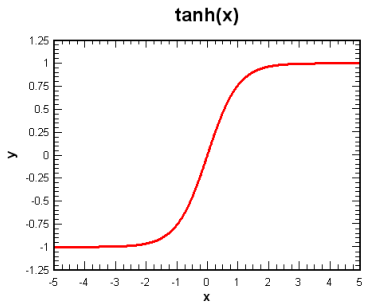
Para cada nodo de la capa oculta hay un peso asociado que lo conecta con la capa de salida.
El parámetro constante asociado a cada nodo, b_j se denomina, en jerga neuronal, **bias** (sesgo). Es el parámetro que controla "**lo que no puede explicar el modelo con los pesos**". Hay que estimarlos.

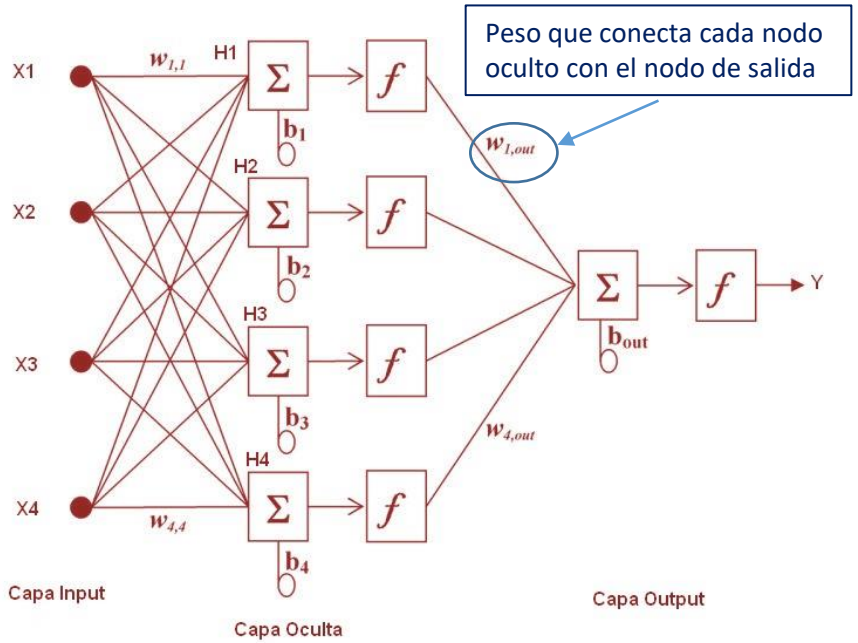
La activación externa de las neuronas de entrada junto con la función de combinación que las conecta con los nodos de la capa oculta, se **activan** y pasan a la capa de salida.

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):

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

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

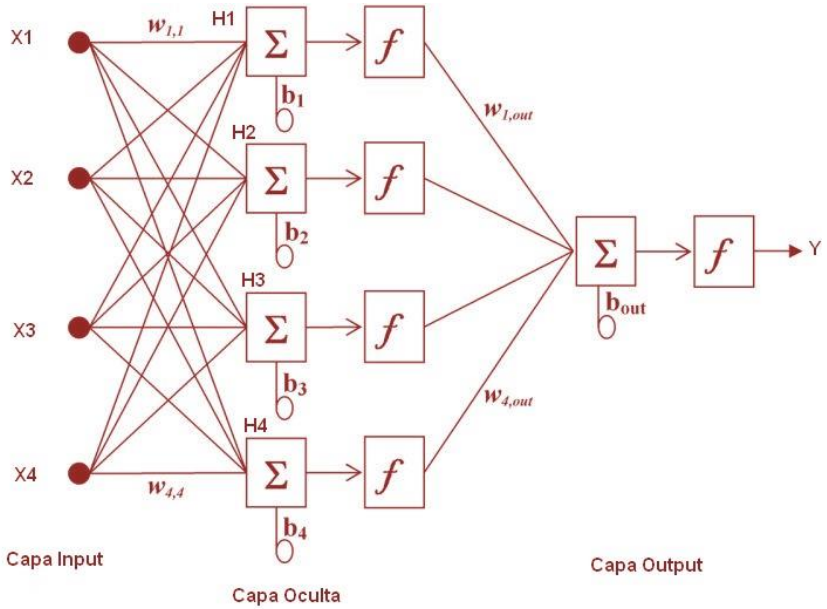
$H3=\tanh(w_{13}X1+w_{23}X2+w_{33}X3+w_{43}X4+b_3)$

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

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



Combinación de los nodos de la capa oculta:

$Y=w_{1,out}H1+w_{2,out}H2+w_{3,out}H3+w4_{,out}H4+bout$

RECORDATORIO:

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

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

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

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

Activación final (nota: cuando la variable output es continua no se realiza activación en el nodo de salida)

- $Y=\tanh(Y)=\tanh(w_{1,out}H1+w_{2,out}H2+w_{3,out}H3+w_{4,out}H4+bout)$ 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):

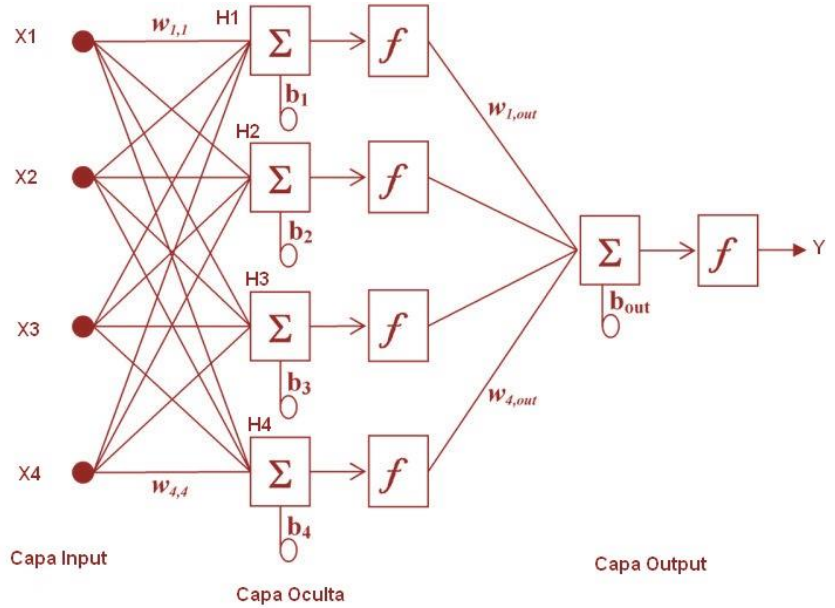
$$Y=\tanh(W_{1,out}H1+W_{2,out}H2+W_{3,out}H3+W_{4,out}H4$$

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

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:

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

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.

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

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

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).

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

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.

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:

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

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

Script para entender de manera interactiva el Teorema de Aproximación Universal:
https://ichi.pro/es/comprender-el-teorema-de-aproximacion-universal-con-codigo-131176068372273

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.

- Cuantos 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.

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.

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 categória 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 ¿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 orden de importancia planteado por los métodos de selección puede ayudar, pero a menudo el problema es complejo con muchas posibilidades y necesario utilizarvarios 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.

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.

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.)

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

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.

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

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

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.

Todas las especificaciones sobre el código a utilizar, explicación de parámetros, análisis de resultado y comentarios se ofrece en los scripts de Jupyter proporcionados en clase.

Aunque hay librerías más concretas y especializadas en la modelización de redes neuronales, las posibilidades que ofrece *sklearn* son suficientes para una primera aproximación al entrenamiento y análisis de redes neuronales para problemas de clasificación binaria y regresión, en los que basta con una sola capa oculta.

ntic master
revolucionamos la comunicación

UNIVERSIDAD COMPLUTENSE DE MADRID

Machine Learning
Redes Neuronales

Inmaculada Gutiérrez García-Pardo