Aprendizaje Automático y Minería de Datos

Javier Cano Salcedo

Introducción a Python

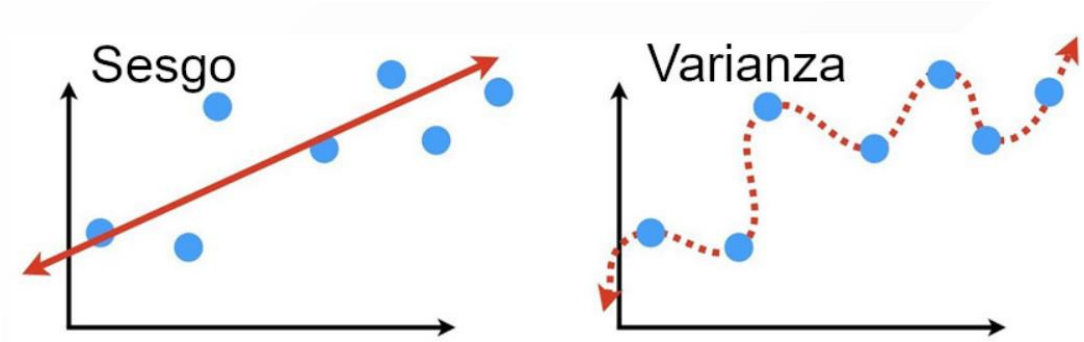
- Multiplicar un string por un entero n lo concatena n veces
- Solo se muestra el resultado de la última instrucción
- Se puede crear una tupla usando paréntesis y comas. Las tuplas son inmutables.
- Se puede crear una lista usando corchetes y comas. Las listas son mutables.
- Se puede crear un conjunto y un diccionario usando llaves y comas.
- Se puede acceder a las posiciones de un array desde el final usando números negativos
- Se pueden crear subconjuntos que van desde `i` hasta `f` de `n` en `n` con la notación `[i:f:n]`
- La entrada se recibe mediante `string input(string mensaje)`. Se debe castear si queremos `int` o `float`.
- `Else if` se abrevia a `elif`
- Se utiliza `range(i, f, n)` para iterar

Introducción al aprendizaje automático

Aprendizaje es el acto, proceso o experiencia de ganar conocimiento o habilidades. Automático es aquello que funciona por sí mismo. Algunas definiciones del aprendizaje automático (**machine learning**) son:

- **Arthur Samuel (1959):** Campo de estudio que dota a los computadores la habilidad de aprender sin ser explícitamente programados
- **Tom Mitchell (1998):** Un programa que aprende de experiencia E respecto a una tarea T y rendimiento R cuenta con aprendizaje automático si su rendimiento en T, medido con R mejora según E.

Aún no está claro cómo los seres vivos aprendemos. Se necesita memoria para almacenar conocimiento, y para resolver problemas nuevos, se requiere abstracción. El aprendizaje automático usa aprendizaje inductivo: saca conclusiones a partir de unos ejemplos, abstrayéndose de detalles que no aporten información relevante a los casos generales. Esta abstracción se conoce como **capacidad de generalización** y genera sesgos (**biases**) que son necesarios para que el aprendizaje funcione.



Tipos de aprendizaje automático:

- **Aprendizaje supervisado:** los ejemplos usados para aprender están resueltos de antemano
- **Aprendizaje no supervisado:** los ejemplos usados no incluyen su resultado esperado
- **Aprendizaje por refuerzo:** se obtiene feedback externo en vez de ejemplos

Los problemas que pueden ser resueltos mediante aprendizaje automático son:

- **Clasificación:** identifica a qué categoría pertenece una nueva observación
- **Regresión:** estimación de valores a partir de otros tan preciso como sea posible

- **Agrupamiento:** no se sabe el número ni tipo de categorías de antemano y queremos descubrirlas
- **Reducción de dimensión:** extraer las características relevantes de los datos
- **Asociación:** busca correlacionar variables

Regresión y clasificación

Terminología:

- Los datos de entrenamiento se usan para entrenar el modelo
- X: entrada
- Y: salida

Regresión lineal con una variable

La **regresión lineal con una variable**, inicialmente concebida como un método estadístico, es un modelo lineal que supone una **relación lineal entre la entrada y la salida**. Cuando solo hay una variable de entrada, se denomina **regresión lineal simple**. En cambio, cuando existen múltiples variables de entrada, se le llama **regresión lineal múltiple**.

$$y = wx + b \Rightarrow f(w, b) = wx + b$$

La determinación de los coeficientes w y b en esta ecuación puede lograrse mediante diversas técnicas, entre las que destacan:

- **Mínimos Cuadrados Ordinarios (OLS):** Este enfoque busca minimizar la suma de los cuadrados de las diferencias entre los valores predichos por el modelo y los valores reales de la variable de salida. Es un método algebraico que proporciona una solución analítica para obtener los coeficientes.
- **Descenso de Gradiente (Gradient Descent):** Esta técnica es un método iterativo que ajusta los coeficientes del modelo en la dirección opuesta al gradiente de la función de costo. Busca encontrar el mínimo local de la función de costo mediante actualizaciones sucesivas de los parámetros.
- **Regularización:** Se utiliza para prevenir el sobreajuste del modelo. La regularización agrega términos adicionales a la función de costo, penalizando los coeficientes más grandes. Las técnicas comunes de regularización son L1 (Lasso) y L2 (Ridge), que controlan la magnitud y la cantidad de coeficientes utilizados.

La **función de costo** es una fórmula matemática que permite a un algoritmo de aprendizaje automático evaluar **qué tan bien se ajusta su modelo a los datos proporcionados**. Esta función devuelve un coste que representa la diferencia entre la representación del modelo y los datos reales.

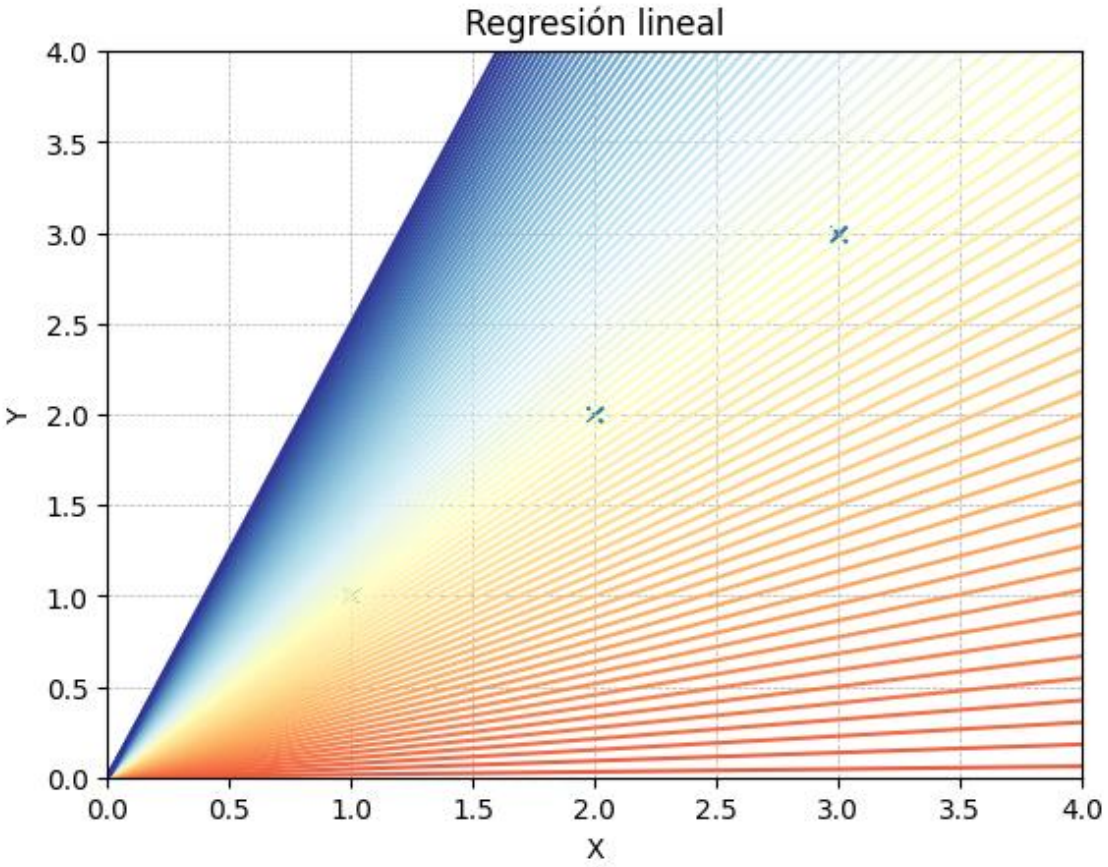
Existen diversas funciones de costo que se pueden emplear, pero una de las más ampliamente utilizadas es el **error cuadrático medio** (MSE). El MSE calcula la media de los cuadrados de las diferencias entre las predicciones del modelo y los valores reales.

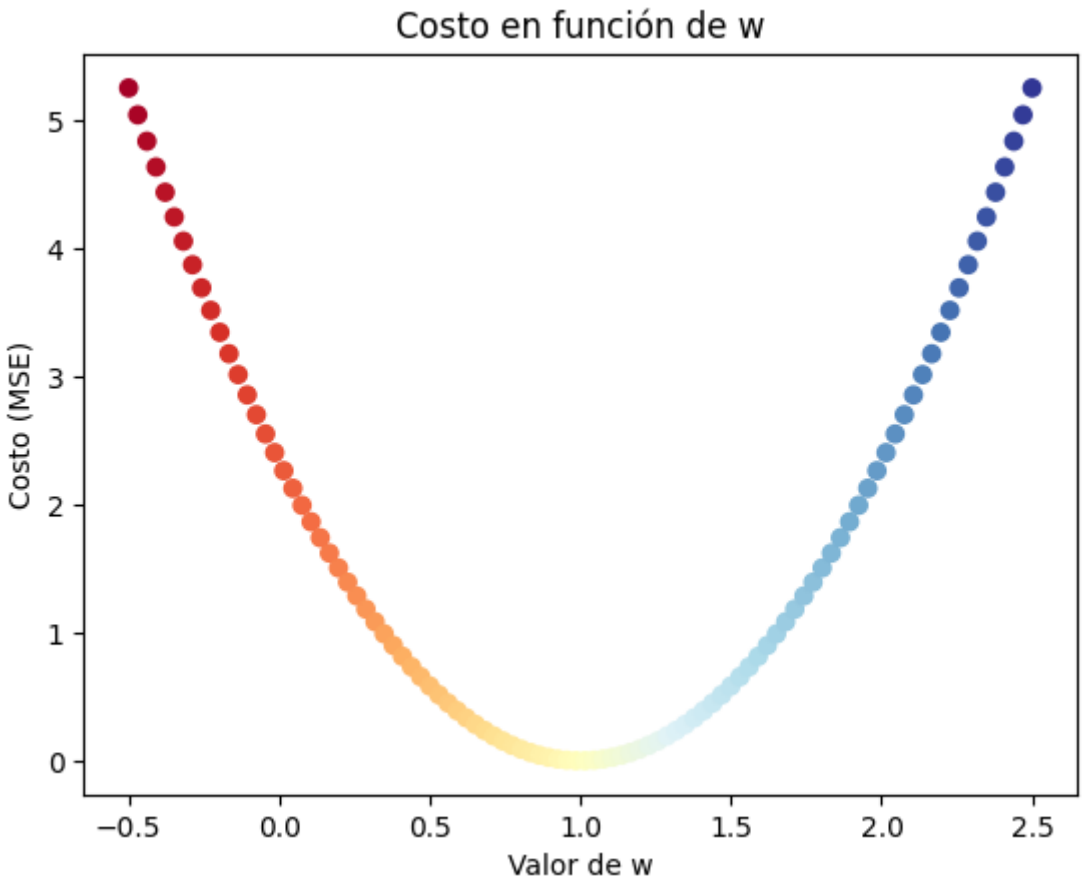
$$MSE(y, y') = \frac{1}{m}\sum_{i=1}^{m}(y_i - f(w,b)_i)^2$$

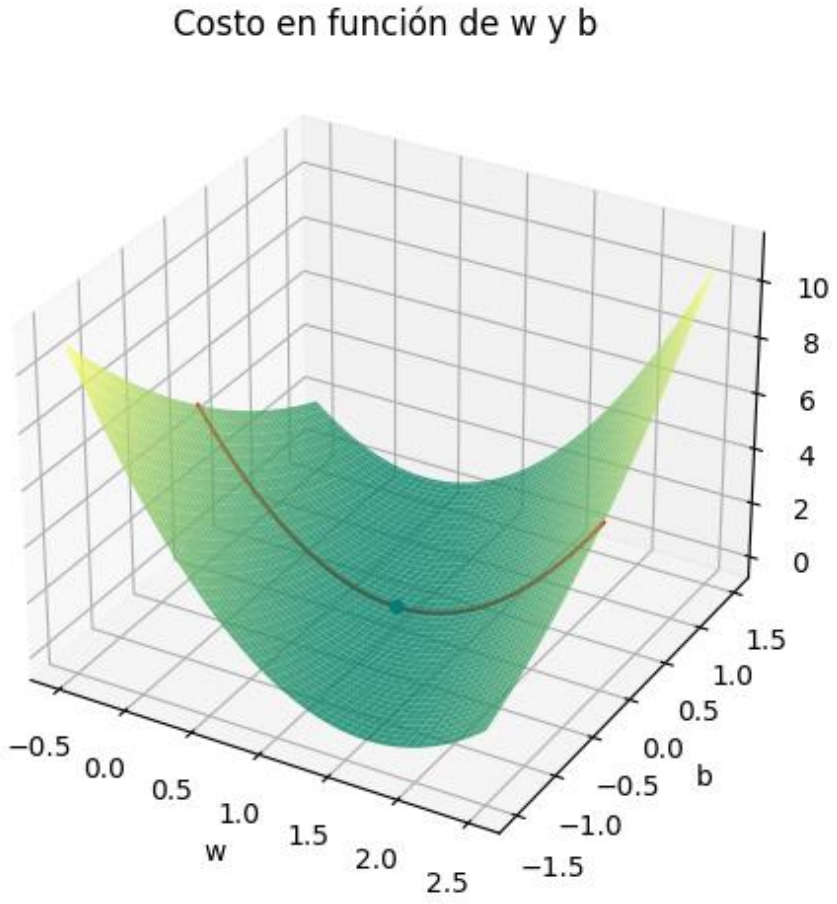
En la práctica, la división por 2 en la fórmula del MSE se realiza por conveniencia matemática al derivar parcialmente la función de costo. Esta división no afecta el propósito principal de medir la calidad del modelo en función de la discrepancia entre las predicciones y los datos reales. Utilizar el MSE como función de costo es común en problemas de regresión lineal, ya que proporciona una métrica clara y fácilmente interpretable para evaluar el rendimiento del modelo.

$$MSE(y, y') = \frac{1}{2m}\sum_{i=1}^{m}(y_i - f(w,b)_i)^2$$

El objetivo de la regresión lineal es minimizar la función de costo.




Costo en función de w
Costo (MSE)
Valor de w
0
1
2
3
4
5
−0.5
0.0
0.5
1.0
1.5
2.0
2.5


Costo en función de w y b
w
b
0
2
4
6
8
10
−0.5
0.0
0.5
1.0
1.5
2.0
2.5
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5

Descenso de gradiente

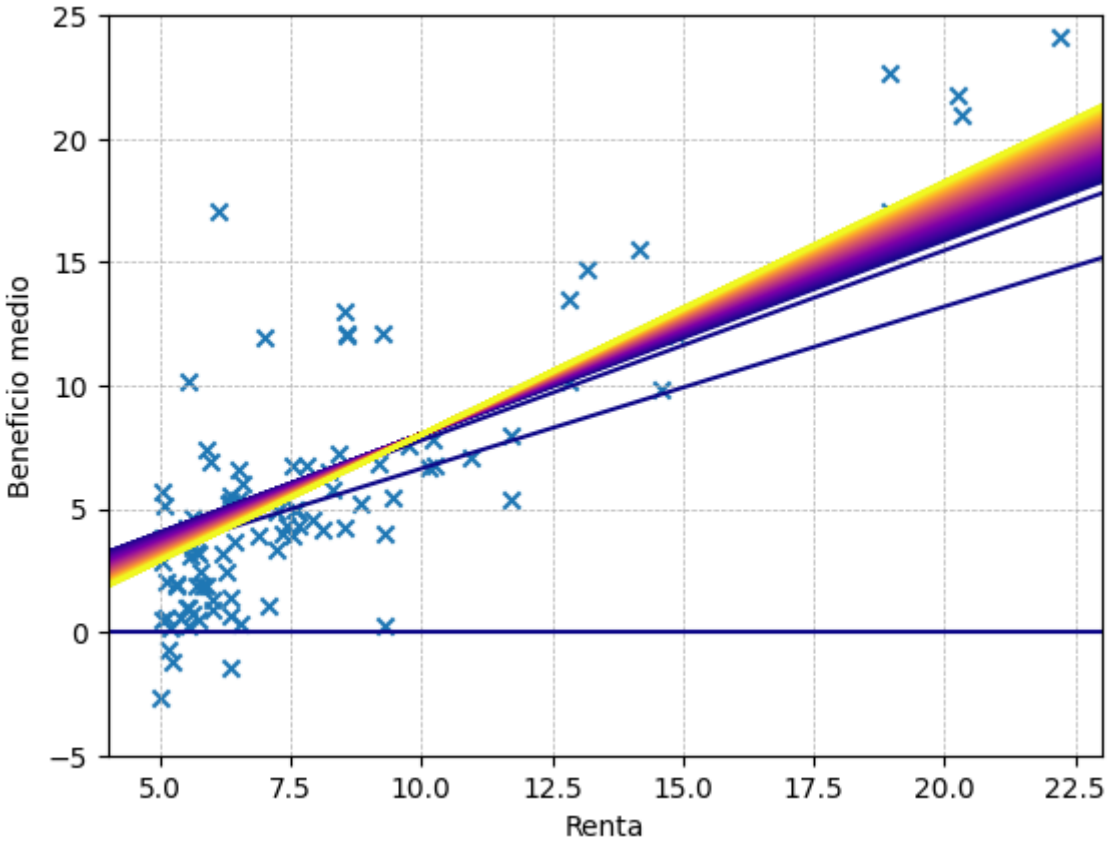
En el **descenso del gradiente**, se inicia el proceso con valores aleatorios para los pesos (w) y el sesgo (b). El proceso de optimización tiene como meta encontrar los valores de w y b que minimizan esta función de costo, iterando sobre los valores aleatorios iniciales hasta que el costo sea el mínimo o muy cercano.

La **tasa de aprendizaje** es un hiperparámetro que determina el tamaño de los pasos dados durante cada iteración del proceso de optimización. Influye en qué tan rápido o lento converge el modelo hacia el mínimo. Si la tasa de aprendizaje es demasiado alta, la optimización podría pasar por alto el mínimo; si es demasiado baja, la optimización podría tomar mucho tiempo o quedar atrapada en un mínimo local.

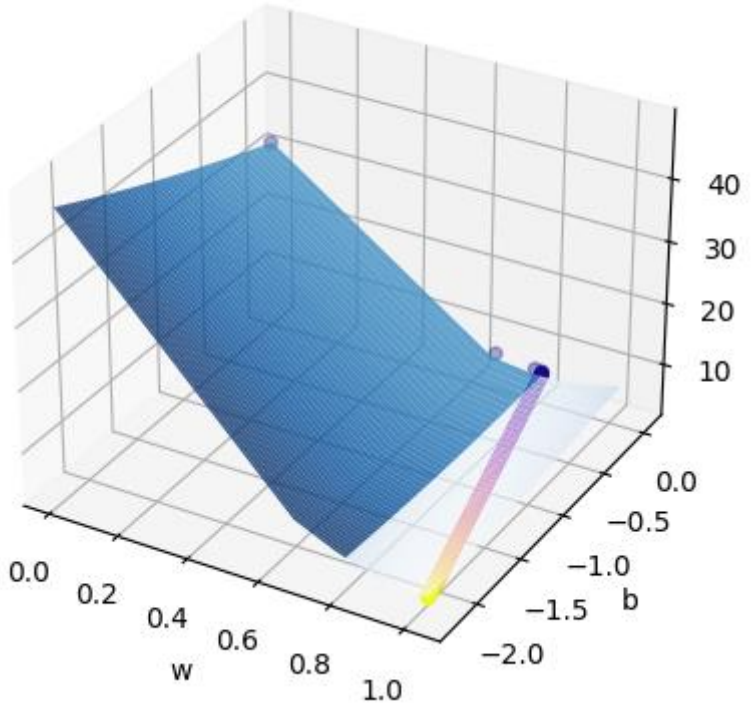
$$w_i = w_{i-1} - \alpha \frac{\partial J(w_{i-1}, b_{i-1})}{\partial w} = w_{i-1} - \alpha \frac{1}{m} \cdot \sum_{k=1}^{m} (f(w_{i-1}, b_{i-1})_k - y_k) x_k$$

$$b_i = b_{i-1} - \alpha \frac{\partial J(w_{i-1}, b_{i-1})}{\partial b} = b_{i-1} - \alpha \frac{1}{m} \cdot \sum_{k=1}^{m} (f(w_{i-1}, b_{i-1})_k - y_k)$$

Donde J es la función de costo (MSE), X es la matriz de características, y es el vector de etiquetas, w es el vector de pesos, b es el sesgo, y m es el número de ejemplos de entrenamiento.

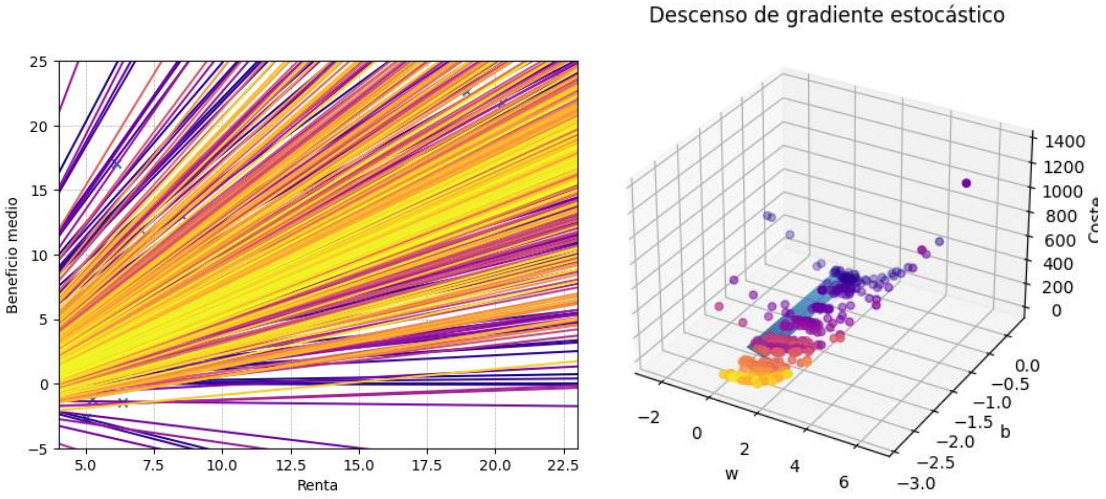


Descenso de gradiente

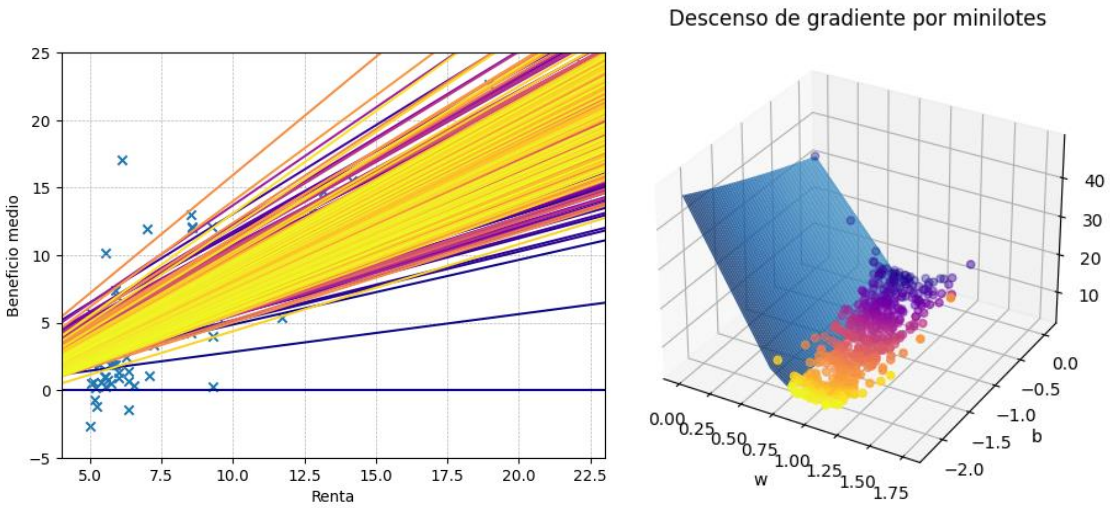


Dependiendo del número de ejemplos de entrenamiento considerados en la actualización de los parámetros del modelo, existen 3 tipos:

- **Batch Gradient Descent (Descenso de Gradiente por Lote):** Los parámetros se actualizan después de calcular el gradiente del error con respecto a todo el conjunto de entrenamiento.
- **Stochastic Gradient Descent (Descenso de Gradiente Estocástico):** Los parámetros se actualizan después de calcular el gradiente del error con respecto a un solo ejemplo de entrenamiento.



- **Mini-Batch Gradient Descent (Descenso de Gradiente por Minilotes):** Los parámetros se actualizan después de calcular el gradiente del error con respecto a un subconjunto del conjunto de entrenamiento.



El algoritmo Mini-Batch para el descenso de gradiente sigue una serie de pasos:

1. Paso hacia adelante en los ejemplos seleccionados en el mini-lote: Se realiza el pase hacia adelante utilizando solo los ejemplos seleccionados en el mini-lote. Esto implica calcular las activaciones en cada capa de la red neuronal y obtener la predicción del modelo para esos ejemplos.
2. Realizar predicciones en el mini-lote: Con las activaciones calculadas en el paso anterior, se hacen predicciones utilizando el modelo neuronal.
3. Calcular el error en la predicción (J): Se calcula el error en la predicción utilizando una función de costo, como la función de error cuadrático medio (MSE) o la entropía cruzada.
4. Paso hacia atrás para calcular el descenso de gradiente: Se realiza un pase hacia atrás para calcular los gradientes de la función de costo con respecto a los parámetros del modelo. Esto implica calcular cómo afectaría un pequeño cambio en cada parámetro a la función de costo.
5. Actualizar parámetros: Se actualizan los parámetros del modelo utilizando los gradientes calculados y el tamaño del paso (tasa de aprendizaje). Este paso busca minimizar la función de costo y ajustar el modelo para mejorar las predicciones.

Estos pasos se repiten a lo largo de múltiples iteraciones sobre el conjunto de datos de entrenamiento. La elección de un tamaño de mini-lote específico permite un equilibrio entre la eficiencia computacional y la capacidad del modelo para generalizar patrones en los datos.

Cada enfoque tiene sus propias ventajas y desventajas. El Batch Gradient Descent puede ser computacionalmente costoso para conjuntos de datos grandes, mientras que el Stochastic Gradient Descent y el Mini-Batch Gradient Descent pueden ser más eficientes, pero también pueden mostrar una convergencia más ruidosa debido a la variabilidad introducida por el uso de subconjuntos más pequeños. La elección entre estos métodos a menudo depende de la naturaleza del conjunto de datos y los requisitos computacionales.

Regresión lineal con múltiples variables

Cuando trabajamos con múltiples variables, la ecuación de la regresión lineal se generaliza de la siguiente manera:

$$y = \sum_{i=1}^{n} w_i x_i + b \Rightarrow f(w,b) = \sum_{i=1}^{n} w_i x_i + b$$

Donde $\vec{w}$ son los parámetros del modelo y b es un número.

Las ecuaciones del descenso de gradiente generalizadas son:

$$w_{i,j} = w_{i-1,j} - \alpha \frac{\partial J(w_{i-1,j}, b_{i-1})}{\partial w} = w_{i-1,j} - \alpha \frac{1}{m} \cdot \sum_{k=1}^{m} \left(f(w_{i,j-1}, b_{i-1})_k - y_k \right) x_k$$

$$b_i = b_{i-1} - \alpha \frac{\partial J(w_{i-1,j}, b_{i-1})}{\partial b} = b_{i-1} - \alpha \frac{1}{m} \cdot \sum_{k=1}^{m} \left(f(w_{i-1,j}, b_{i-1})_k - y_k \right)$$

Una alternativa al descenso del gradiente es la **Ecuación Normal**. Esta técnica es exclusiva para la regresión lineal y proporciona una solución cerrada para los parámetros w y b sin la necesidad de realizar iteraciones.

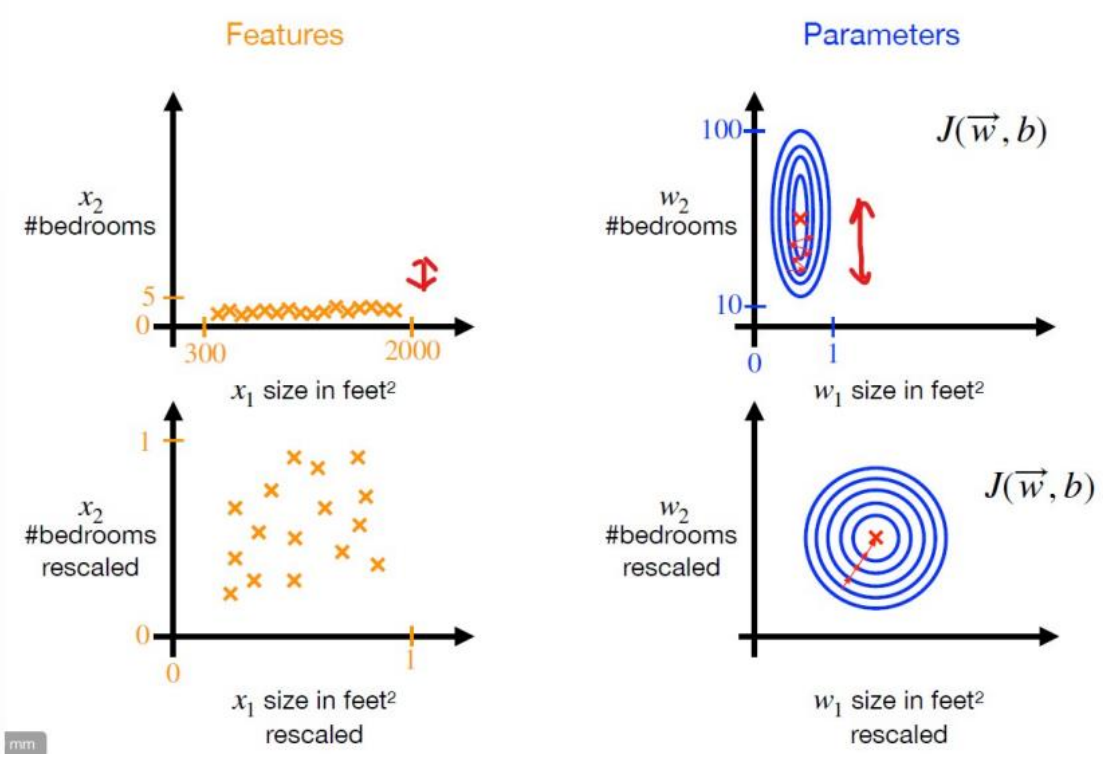
La Ecuación Normal se expresa de la siguiente manera para un problema de regresión lineal:

$$\theta = (X^T X)^{-1} X^T y$$

Donde θ es el vector de parámetros que minimiza la función de costo, X es la matriz de características del conjunto de entrenamiento y y es el vector de resultados del conjunto de entrenamiento.

Esta fórmula calcula directamente los valores óptimos de los parámetros sin necesidad de ajustar un hiperparámetro como la tasa de aprendizaje. Sin embargo, la Ecuación Normal no generaliza bien a otros algoritmos de aprendizaje y puede volverse ineficiente cuando el número de características es grande. En la práctica, el descenso del gradiente suele ser el método preferido para encontrar los parámetros, pero la Ecuación Normal puede ser útil en ciertos contextos, especialmente cuando el número de características es manejable.

Las características deben estar en la misma escala. A medida que el tamaño del error modifica los parámetros utilizando el descenso del gradiente, tener diferentes escalas hace que algunos parámetros se ajusten mejor que otros. Al reescalar todos los valores a una escala similar, los parámetros se modifican con la misma proporción. Saltos demasiado grandes o pequeños (como con la tasa de aprendizaje) dificultarán encontrar el peso correcto que minimiza la función.



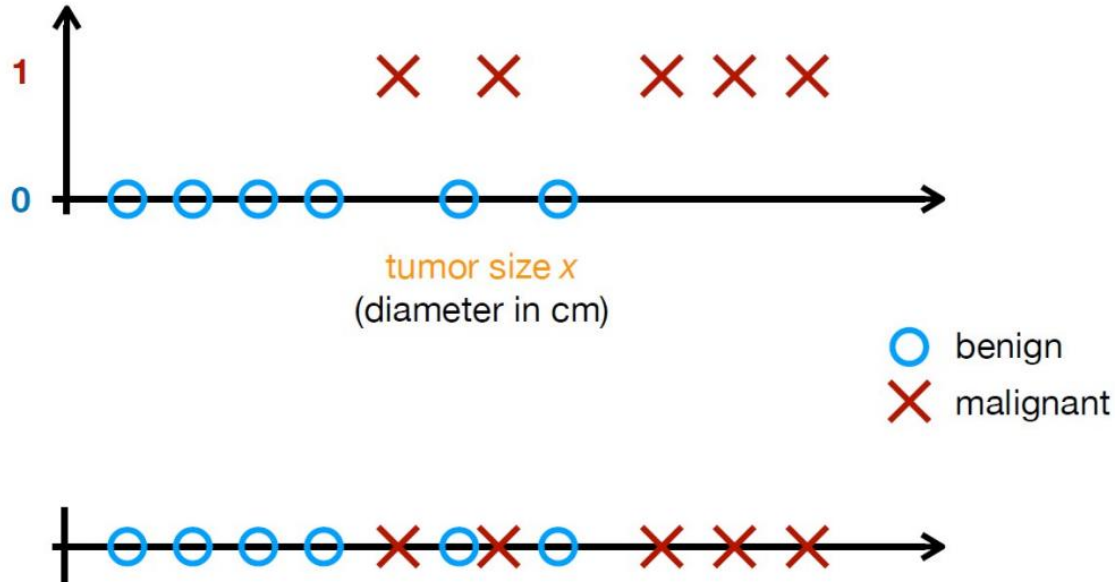
Podemos establecer diferentes criterios de convergencia, siendo los más comunes:

1. Que el error sea inferior a uno dado.
2. Falta de mejora significativa en el rendimiento del modelo durante n iteraciones del descenso del gradiente.

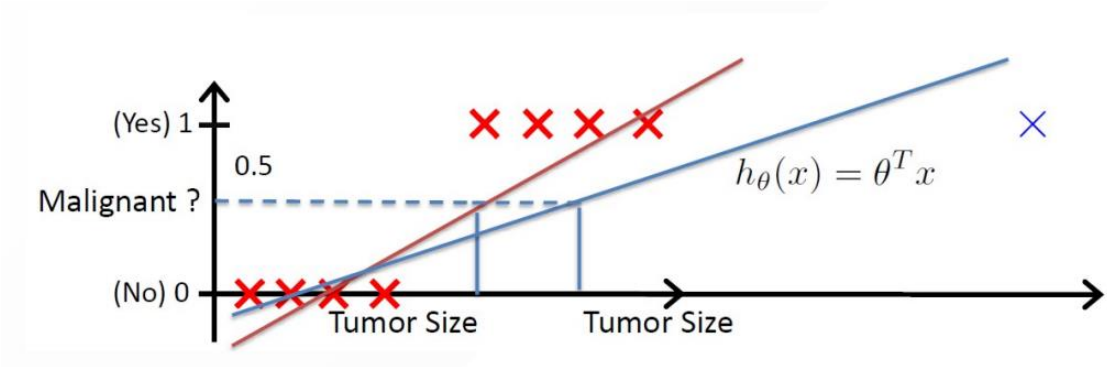
Como la regresión lineal, hay muchas más regresiones con otras formas, como la regresión polinomial, que permite modelar relaciones no lineales entre las variables. En la regresión polinomial, el grado del polinomio determina la complejidad del modelo y su capacidad para adaptarse a patrones más intrincados en los datos.

Clasificación

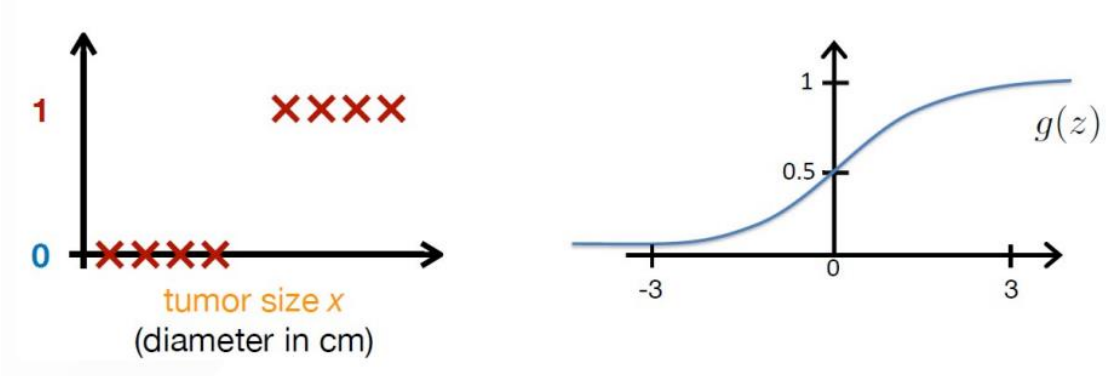
Hasta ahora, hemos estimado el valor de una variable basándonos en el valor de sus características. Sin embargo, existe otro tipo de problema en el que intentamos deducir, a partir de datos de entrada, qué tipo de ejemplo es; por ejemplo, si una imagen es de un gato o un perro, si un correo electrónico es spam o no, entre otros casos. En este contexto, no esperamos una salida continua, sino que buscamos una salida discreta, es decir, una clasificación que indique a qué categoría pertenece el ejemplo en cuestión.



En este tipo de problemas no se aplica la regresión lineal, sino la regresión logística



$$g(z) = \frac{1}{1 + e^{-z}}$$
$$0 \leq g(z) \leq 1$$

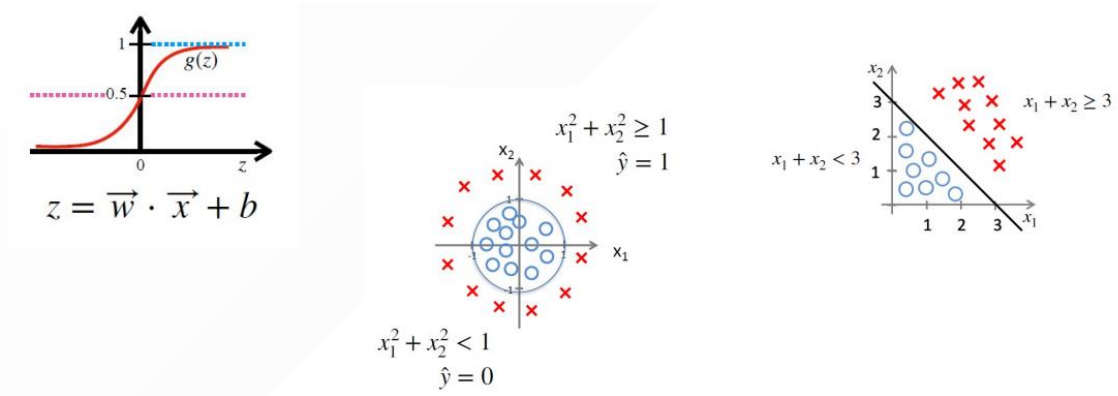


La función a optimizar en problemas de clasificación combina las funciones logísticas y de regresión. La ecuación resultante, expresada en forma vectorial, produce una salida que representa la probabilidad de que un ejemplo pertenezca a una clase en lugar de

otra. Por ejemplo, si F(X) = 0.7 en el ámbito del cáncer de mama, indicaría una probabilidad del 70% de la posibilidad de desarrollar cáncer.

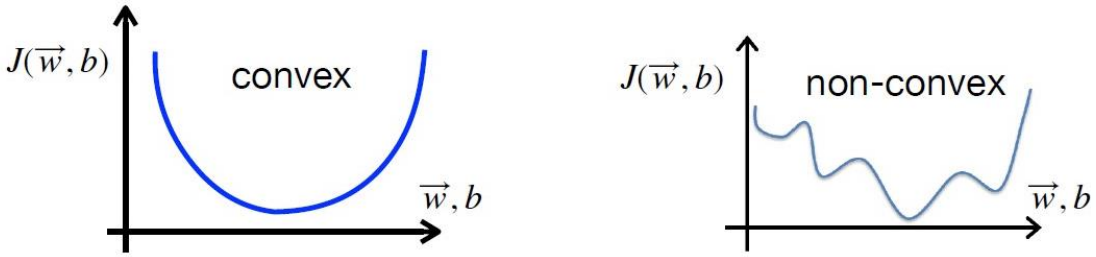
$$f_{\vec{w},b}(\vec{x}) = \frac{1}{1 + e^{-(\vec{w}\vec{x}+b)}}$$
$$0 \leq f_{\vec{w},b}(\vec{x}) \leq 1$$

La frontera de decisión en problemas de clasificación se determina a través de la expresión que utilizamos en la función logística (Z). Esta expresión define cómo delimitamos los grupos.



La función de costo en problemas de clasificación utiliza la misma medida de error, pero la función f(x) es diferente. La pérdida (loss) es un valor que representa la suma de errores en nuestro modelo y mide qué tan bien (o mal) está haciendo nuestro modelo.

Si $J(\vec{w}, b)$ es convexo, quiere decir que podemos alcanzar el mínimo global.



La pérdida (L) de la función logística está garantizada que sea convexa para todos los valores de entrada, lo que significa que tiene un solo mínimo.

$$Si\ y_i = 1 \rightarrow L\left(f_{\vec{w},b}(\vec{x})\right) = -\log\left(f_{\vec{w},b}(\vec{x})\right)$$
$$Si\ y_i = 0 \rightarrow L\left(f_{\vec{w},b}(\vec{x})\right) = -\log\left(1 - f_{\vec{w},b}(\vec{x})\right)$$

$$J(\vec{w}, b) = \frac{1}{m}\sum_{i=1}^{m} L\left(f_{\vec{w},b}(\vec{x}_i), y_i\right) = -\frac{1}{m}\sum_{i=1}^{m} y_i \log f_{\vec{w},b}(\vec{x}_i) + (1 - y_i)\log f_{\vec{w},b}(\vec{x}_i)$$

Si derivamos la función de coste (J) en la regresión logística, observamos que el resultado de la derivada es el mismo que en el algoritmo de regresión lineal normal.

Para el descenso de gradiente, los parámetros w y b se mantienen igual que en la regresión lineal.

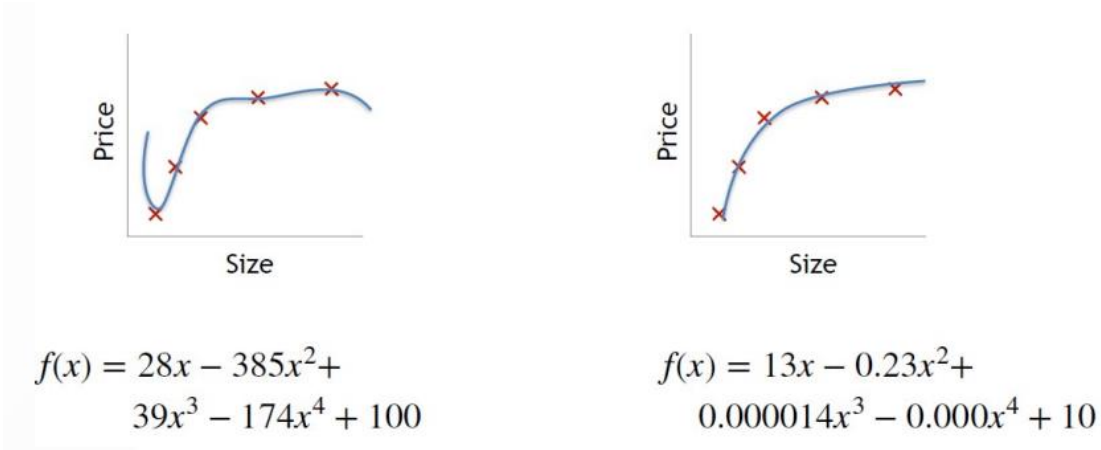
En la clasificación multiclase, aplicamos la estrategia "1 clase vs. todas las demás" para cada una de las clases, calculando sus probabilidades respectivas. Se escoge la clase que más probabilidad tenga de ser.

$$\max_i f^i_{\vec{w},b}(\vec{x})$$

Regularización

La regularización es una técnica utilizada en el aprendizaje automático para evitar el sobreajuste de modelos. El sobreajuste ocurre cuando un modelo se ajusta demasiado a los datos de entrenamiento y pierde la capacidad de generalización. La regularización de modelos nos ayuda a reducir la complejidad del modelo y evitar el sobreajuste.

El sobreajuste es un problema común en el aprendizaje automático y se aborda de varias maneras. Una opción es recolectar más ejemplos de entrenamiento para proporcionar al modelo una perspectiva más amplia de los datos. Otra opción es seleccionar manualmente características, aunque esto puede resultar en la pérdida de características útiles. Una técnica efectiva para abordar el sobreajuste es la regularización, que implica reducir el tamaño de ciertos parámetros en la matriz de pesos (W), ayudando así a controlar la complejidad del modelo.



La regularización ofrece varios beneficios al enfrentar el sobreajuste. Permite conservar todas las características del conjunto de datos mientras reduce la magnitud de los parámetros del modelo. Esta estrategia es particularmente efectiva cuando hay muchas

características, cada una de las cuales contribuye de manera marginal a las predicciones. Además, la regularización es útil para equilibrar la varianza y sesgo del modelo, mejorando así su capacidad para generalizar a datos no vistos.

La función de costo en la regularización busca reducir los pesos para disminuir la complejidad del modelo. Para reducir la función de costo, añadimos una función de penalización $p(w)$ a $J(\vec{w}, b)$. Utilizamos $\lambda > 0$ como tasa de aprendizaje. El parámetro es un hiperparámetro del modelo, similar a en el descenso de gradiente.

$$J(\vec{w}, b) = \frac{1}{2m}\sum_{i=1}^{m}(y_i - f(w,b)_i)^2 + \lambda p(w)$$

La regularización L1 (Lasso) añade una función de penalización proporcional a la suma de los valores absolutos de los coeficientes del modelo. Esto tiene el efecto de forzar que algunos coeficientes sean cero, lo que implica modelos más simples.

$$p(w) = \sum_{j=1}^{m}|w_j|$$

La regularización L2 (Ridge) añade un término de penalización proporcional a la suma de los cuadrados de los coeficientes del modelo. Esto tiene el efecto de reducir los valores de los coeficientes, lo que puede ayudar a evitar un ajuste excesivo, pero nunca llegarán a ser cero.

$$p(w) = \sum_{j=1}^{m}{w_j}^2$$

Redes de Neuronas

Redes Neuronales Artificiales

Definimos Neurona Artificial como un elemento que posee un estado interno (S) denominado nivel de activación que simula la resistencia dieléctrica de la sinapsis. Además, recibe un conjunto de señales que le permiten cambiar de estado a través de

una conexión con otras neuronas, que influyen en el nivel de activación con señales positivas y negativas (Simulando los neurotransmisores).

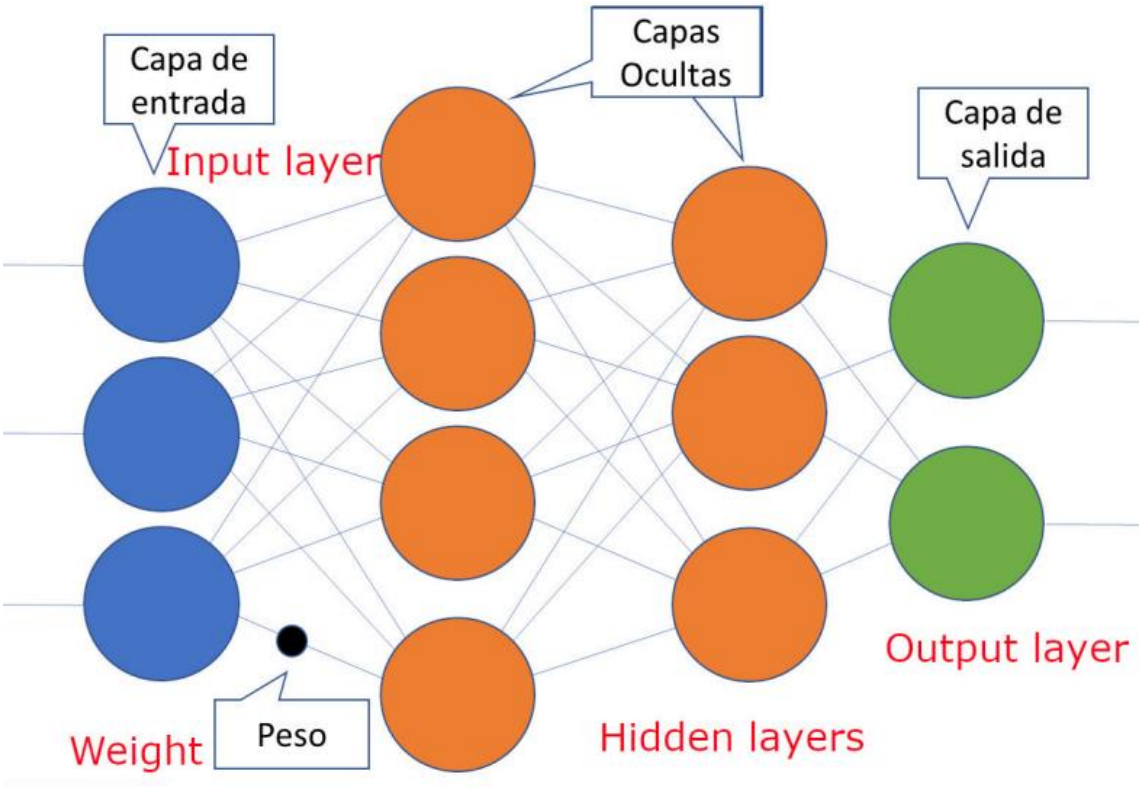
La potencia de cálculo de una red neuronal artificial reside en sus interconexiones. La neurona artificial, como analogía matemática, tiene una complejidad sináptica mucho menor. Esta neurona posee un estado interno (S) llamado nivel de activación, que simula la complejidad dieléctrica de la sinapsis. Recibe un conjunto de señales que le permiten cambiar de estado, simulando así la función de los neurotransmisores.

La función matemática que permite cambiar el estado de activación dependiendo de las entradas se denomina función de activación. Cada entrada de la neurona es multiplicada por un peso que es modificable durante el aprendizaje. Cada peso permite que llegue una cantidad mayor o menor de la señal procedente de otra neurona. Una red neuronal es una colección de neuronas cuya salida está conectada a las entradas de otras neuronas formando un grafo dirigid G(V,A) donde los vértices (V) son las neuronas y las aristas (A) son las conexiones entre ellas.

Hay 3 tipos de neuronas:

- **Neuronas de entrada:** Son las encargadas de recibir los datos del exterior de la red e inician la propagación de dichos datos por toda la estructura
- **Neuronas ocultas:** neuronas que permiten enlazar las neuronas de entrada y las neuronas de salida, aunque también pueden estar conectadas a otras neuronas ocultas
- **Neuronas de salida:** Son las que muestran al exterior los resultados obtenidos después de que la entrada haya recorrido toda la red

Una **capa** es una colección de neuronas que tienen una función común.



Normalmente en cada neurona se hace la suma ponderada de los pesos para calcular su activación (aunque tambien se usa max y min) Podemos definir por lo tanto de forma matemática una red de neuronas como una ecuación vectorial, de la siguiente forma:

$$S = f(\dots f(f(\vec{x} \cdot w_1) \cdot w_2) \dots \cdot w_n)$$

Si la función de activación es lineal, añadir más capas en la red es irrelevante y se podría crear una red de una sola capa oculta equivalente. Pero si la función de activación no es lineal (lo habitual), entonces añadir más capas si tiene sentido.

Las funciones de activación más usadas son:

- La función escalón donde α es el valor umbral de activación

$$g(x) = \begin{cases} 1 \; si \; x \geq \alpha \\ -1 \vee 0 \; si \; x \leq \alpha \end{cases}$$

- La función sigmoidal

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

- La tangente hiperbólica

$$g(x) = \tanh x = \frac{\sinh x}{\cosh x} = \frac{1 - e^{-2x}}{1 + e^{-2x}}$$

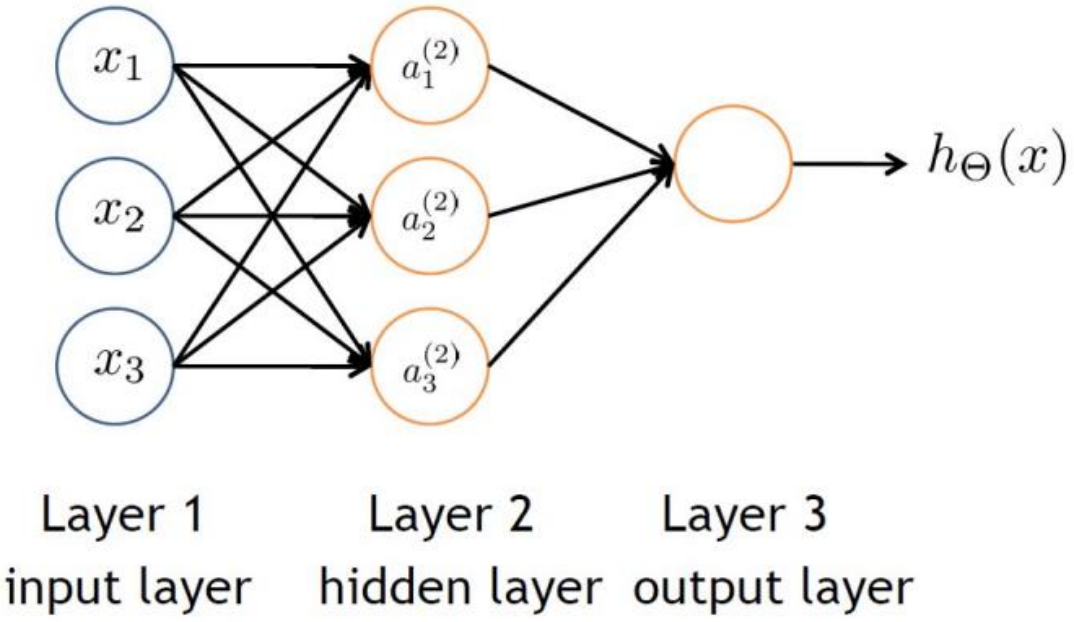
- La función rectificadora (ReLU)

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

Perceptrón multicapa

Es una generalización multicapa del perceptrón definido por Rosenblattd (1962). Usa como función de entrada la suma ponderada y como función de activación la que el usuario establezca (inicialmente el perceptrón de Rosenblattd usaba la función escalón) Rumelhartt, Hilton y Willians en 1989 presentaron un mecanismo que permitía dicha propagación del error por todos los pesos de las conexiones, permitiendo crear modelos de redes de más de una capa. El perceptrón multicapa está demostrado que es un aproximador universal de cualquier función.

La primera fase de una red neuronal es el feedforward.



Sea a_i^j la activación de la unidad i en la capa j y Θ^j la matriz de pesos que controla la activación desde la capa j a la capa j+1. La capa de entrada tendrá tantas neuronas como datos de entrada. La capa oculta podría tener cualquier número de neuronas, pero para este ejemplo supondremos que tiene el mismo número que la entrada. Para cada conexión desde la entrada a la capa oculta hay un peso w que conecta cada neurona de la capa de entrada con otra neurona de la capa oculta. La activación es la suma de la entrada multiplicada por los pesos y añadiendo un parámetro más llamado sesgo/bias. Es como si hubiera otra entrada siempre a 1.

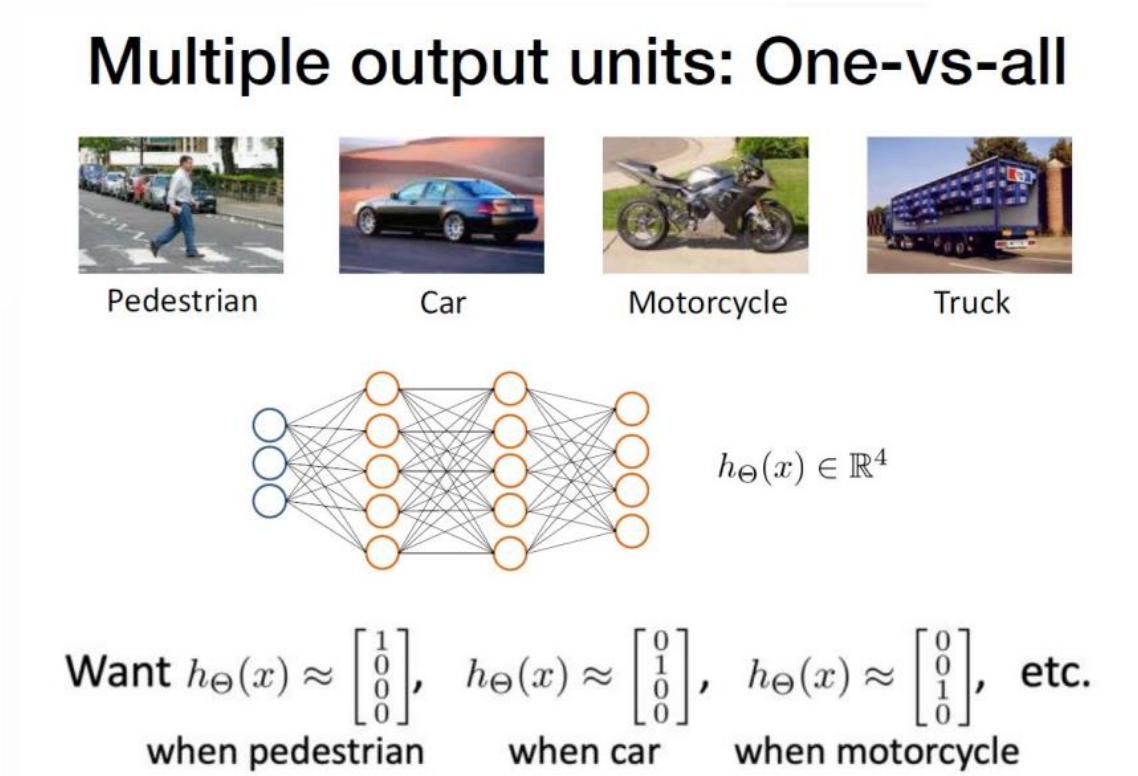
$$a_i^j = g\left(\Theta_{i,0}^{j-1}a_0^{j-1} + \sum_{k=1}^{|a^{j-1}|}\Theta_{i,k}^{j-1}a_k^{j-1} + b_i^{j-1}\right)\forall i \in [1..|a^j|], j \in [1..|C|]$$

$$\Theta_{i,0}^{j-1}a_0^{j-1} = bias$$

$$a_0^{j-1} = 1$$

Donde C son las capas del modelo, $|C|$ es el número de capas, $|a^j|$ es el número de neuronas en la capa j, $|a^{j-1}|$ número de neuronas en la capa anterior y g la función de activación (ReLU, sigmoidal, etc).

Cualquier codificación de la salida sería válida en teoría. Lo más práctico es que cada neurona represente una clase. La codificación binaria, por ejemplo, produce peores resultados y no expresa la salida en forma de probabilidad, lo cual es bastante útil para razonar sobre la salida de la red a posteriori.



La red neuronal funciona mejor con datos normalizados, ya sea en el rango 0..1 o en el rango -1..1. Por lo tanto, es un requisito previo que los valores de entrada de la red estén normalizados.

Algoritmos de optimización

En el ámbito del aprendizaje automático, la elección del algoritmo de optimización es crucial para el rendimiento del modelo. Uno de los métodos más comunes es el Descenso

de Gradiente, que ajusta iterativamente los parámetros del modelo para minimizar la función de costo. Otros enfoques populares incluyen BFGS / L-BFGS, que es eficiente para la estimación de parámetros; ADAM, un algoritmo de optimización basado en gradientes que destaca en problemas grandes y ruidosos; y Descenso de Gradiente Estocástico (SGD), que estima el gradiente utilizando subconjuntos de datos y es especialmente útil en problemas de alta dimensionalidad. La elección del algoritmo adecuado depende de diversos factores, como el tamaño del conjunto de datos, la dimensionalidad y la naturaleza del problema.

Diseño de sistemas de Aprendizaje Automático

Proceso de desarrollo del Aprendizaje Automático

En proyectos reales, gran parte del tiempo de desarrollo se dedica a obtener, pulir y limpiar datos. Los algoritmos de ML suelen tener que volver a entrenarse para mantener su relevancia, ya que el entorno cambia a menudo. Y durante el entrenamiento hay que hacer muchos ajustes finos. Así, tenemos un proceso iterativo de desarrollo de proyectos basados en aprendizaje automático:

1. Elegir la arquitectura: incluyendo datos, modelos, etc.
2. Entrenar el modelo: Entrenamiento del propio modelo, selección de hiperparámetros, etc.
3. Diagnóstico: sesgo, varianza, error, etc.

Limpiar y codificar los datos

Para abordar errores y valores nulos en el conjunto de datos, se pueden emplear diversas estrategias. **Eliminar observaciones** con datos faltantes es una opción, pero no siempre es recomendable, especialmente si hay pocos datos. Otra estrategia común es la **imputación**, que implica asignar valores significativos a las características afectadas. Esto puede lograrse asignando el valor medio o moda de la característica, o utilizando métodos más avanzados como la imputación basada en vecinos más cercanos (**KNNImputer**), donde se imputa el valor medio de entradas similares al dato en cuestión.

En cuanto a la codificación de variables categóricas, una técnica esencial es el One-Hot Encoding. Esta transformación convierte una característica categórica en tantas variables como valores únicos tenga la categoría. Cada variable binaria resultante toma el valor 0 o 1 según la presencia o ausencia de la categoría en la observación correspondiente. El One-Hot Encoding es crucial al trabajar con algoritmos de aprendizaje automático que requieren entradas numéricas, permitiendo así representar de manera efectiva la información categórica en un formato numérico comprensible para los modelos.

Evaluando el modelo

La evaluación de un modelo es un paso crítico en el proceso de aprendizaje automático para medir su desempeño y su capacidad para generalizar a datos no vistos. Se emplean diversas métricas para evaluar la calidad del modelo:

- **Error Cuadrático Medio (MSE):** Calcula la diferencia cuadrática promedio entre las predicciones del modelo y los valores reales. Es particularmente útil en problemas de regresión.

- **Matriz de Confusión:** Proporciona una visión detallada del rendimiento del modelo, mostrando el número de verdaderos positivos, verdaderos negativos, falsos positivos y falsos negativos.

	Observed True	Observed False
Predicted True	True Positive	False Positive
Predicted False	False Negative	True Negative

- **Exactitud (Accuracy):** Mide la proporción de predicciones correctas sobre el total de observaciones. Sin embargo, puede ser problemática en casos de clases desequilibradas.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

- **Tasa de Verdaderos Positivos (Recall):** Calcula la proporción de verdaderos positivos entre todos los positivos observados, destacando la capacidad del modelo para identificar correctamente las instancias positivas.

$$Recall = \frac{TP}{TP + FN}$$

- **Valor Predictivo Positivo (Precisión):** Representa la proporción de verdaderos positivos entre todas las instancias predichas como positivas, indicando la precisión del modelo en sus afirmaciones positivas.

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

- **F1 Score:** Es una métrica que combina precisión y recall en un solo valor, útil cuando hay un equilibrio deseado entre ambas métricas.

$$F1\ Score = 2\frac{Precision \cdot Recall}{Precision + Recall} = \frac{2TP}{2TP + FP + FN}$$

Estas métricas proporcionan una visión holística del rendimiento del modelo en diferentes aspectos, permitiendo una evaluación comprehensiva en función de los objetivos específicos del problema.

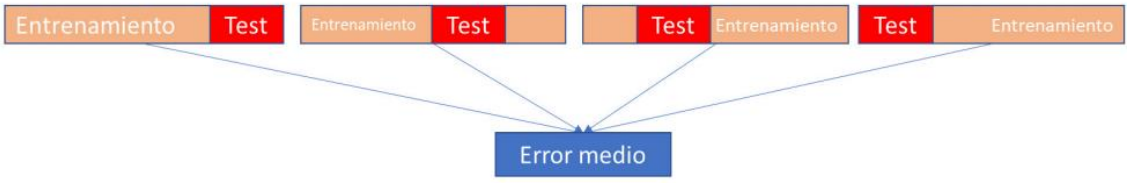
Al evaluar un modelo, es fundamental dividir el conjunto de datos en conjuntos de entrenamiento y prueba. La distribución debe ser equitativa y proporcional al número de clases, evitando sesgos. Durante el entrenamiento, se monitorea tanto el error de entrenamiento como el de validación. El error de validación, similar al de prueba, se calcula durante el entrenamiento para evaluar la capacidad de generalización del modelo.

Es crucial distinguir entre el conjunto de validación y el de prueba. Aunque parecen similares, se diferencian en que el conjunto de prueba se utiliza después del entrenamiento, mientras que el de validación contribuye a mantener la capacidad de generalización durante el mismo. La información del conjunto de prueba no debe filtrarse en el modelo para evitar sobreajustes.

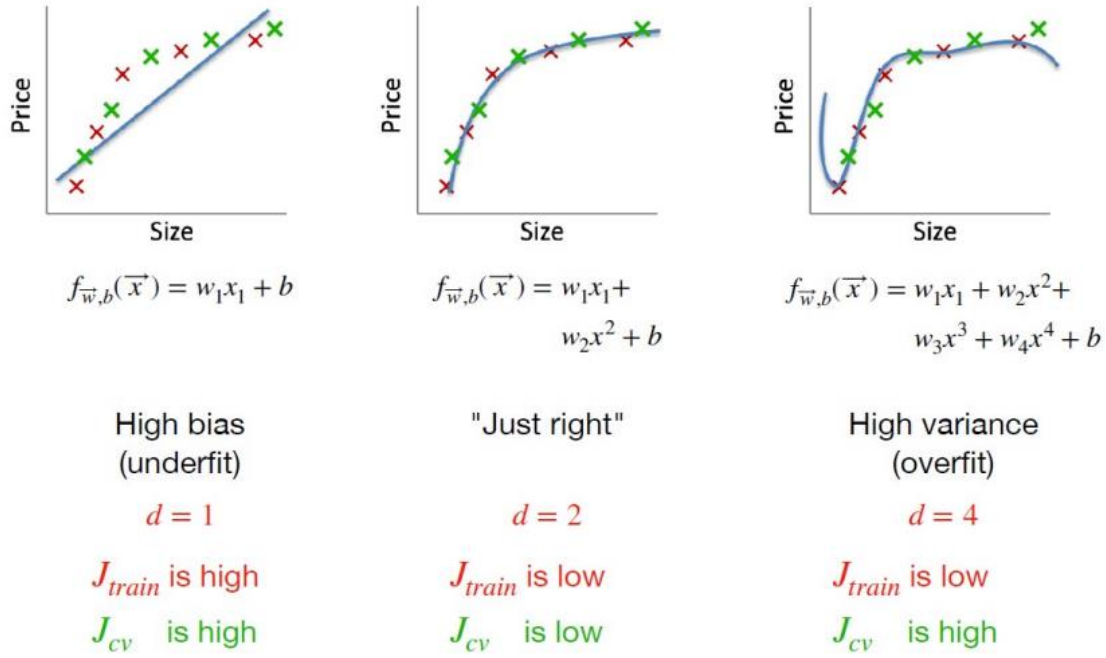
En este proceso, se tienen tres conjuntos distintos:

- **Error de Entrenamiento:** Evaluación del rendimiento en el conjunto de datos de entrenamiento.
- **Error de Validación:** Evaluación continua durante el entrenamiento para mantener la capacidad de generalización.
- **Error de Pruebas:** Evaluación con datos excluidos del entrenamiento, realizada después del mismo.

Sin embargo, al dividir los datos en tres conjuntos, se reduce la cantidad de muestras disponibles para el aprendizaje del modelo, y los resultados pueden depender de la elección aleatoria de los conjuntos de entrenamiento y validación. La validación cruzada, como K-Fold Cross Validation, supera este desafío dividiendo los datos en k subgrupos utilizando k-1 para el entrenamiento y 1 para la validación en cada iteración. Esto permite aprovechar más datos y evitar sesgos en la elección del conjunto de validación.



Diagnóstico de sesgo y varianza



Las grandes redes neuronales son modelos de bajo sesgo. Es importante seleccionar el parámetro de regularización correcto.

Añadir más datos

Añadir más datos de entrenamiento ayuda a reducir la varianza y, por tanto, el error de prueba. Si no se dispone de más datos, podemos realizar algunas técnicas que nos ayudan a generar más datos de forma artificial: **Data aumentation**.

Para distintos tipos de datos, como sonidos o imágenes, se pueden introducir diversos tipos de ruido o distorsiones. Por ejemplo, en el caso de sonidos, se puede añadir ruido de fondo, como el de maquinaria o multitudes. En imágenes, se pueden aplicar diferentes tipos de iluminación o transformaciones de perspectiva para mejorar el reconocimiento de objetos.

Es importante destacar que no suele ser útil añadir ruido aleatorio o sin sentido a los datos. En cambio, la incorporación de distorsiones específicas y relevantes puede mejorar la capacidad de generalización del modelo.

Antes de dedicar esfuerzos a buscar más datos, es crucial asegurarse de que el clasificador tiene un sesgo bajo. Esto se puede lograr trazando curvas de aprendizaje y ajustando parámetros clave, como el número de características o unidades ocultas en una red neuronal.

Una vez confirmada la necesidad de más datos, se pueden abordar de diversas maneras, ya sea mediante la síntesis artificial de datos, la recopilación manual o el crowdsourcing a través de plataformas como Amazon Mechanical Turk.

Transfer learning

El Transfer Learning se puede definir como "conjunto de métodos que permiten transferir conocimientos adquiridos gracias a la resolución de problemas para resolver otros problemas."

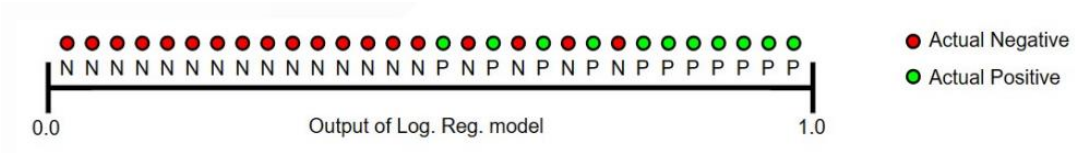
Las redes de neuronas pueden encadenarse mediante capas. Esta es la base del Deep Learning. Podemos tener un sistema que sea capaz de realizar una tarea, preentrenarlo y usar ese sistema ya entrenado con otros datos, añadiendo nuevas capas.

Esas nuevas capas son las que vamos a entrenar, dejando el resto de los parámetros con los valores establecidos en el preentrenamiento.

Curva de Característica Operativa al Receptor

Una curva ROC (curva de característica operativa del receptor) es un gráfico que muestra el rendimiento de un modelo de clasificación. Esta curva representa dos parámetros: la tasa de verdaderos positivos y la tasa de falsos positivos.

Para cada sesgo que decidamos, podemos construir una matriz de confusión, y calcular el punto en el espacio ROC correspondiente. Dibujar la curva ROC consiste en poner juntos todos los puntos correspondientes a todos los sesgos o puntos de corte.



Cuestiones éticas

Todos los métodos de aprendizaje automático utilizan sesgos para aprender. Hay que tener mucho cuidado con los sesgos que introduzcamos artificialmente, ya que pueden producirse sesgos machistas, racistas... El aprendizaje automático se puede usar también para el mal como para hacer fake news, fake nudes, cometer fraudes...

Otras técnicas de Aprendizaje Automático Supervisado

Aunque la redes de neuronas han alcanzado un gran popularidad y son el estándar del Machine Learning actual, es recomendable conocer otras técnicas ya que cuentan con algunas ventajas sobre las redes de neuronas y para ciertos problemas pueden ser suficientemente buenas.

Redes de neuronas	
Ventajas	Desventajas
Tolerancia a fallos	Coste computacional elevado
Modelos escalables	Algoritmo de caja negra
Pueden resolver problemas no lineales	

K-Nearest Neighbour

El algoritmo no intenta extraer una característica de los datos haciendo un proceso de entrenamiento exhaustivo con ellos. Simplemente los almacena para recuperarlos en el futuro. Cuando se presenta un nuevo caso, éste es comparado con la base de casos y se extrae aquel más similar (o los K más similares) para comprobar cuál era la clase a la que pertenecía. Se asume que, si el caso es similar, la solución será similar.

Se necesita definir una función de similitud entre dos ejemplos. La calidad de esta función de similitud es crítica para este algoritmo. Si la función de similitud no es capaz de medir correctamente la similitud que existe entre dos casos dados, la asunción de que a similares casos se espera que la instancia sea de la misma clase pierde vigencia y KNN deja de clasificar correctamente.

La solución que se elige depende del valor de K seleccionado. Si K = 1, la instancia recuperada más similar se convierte en la solución. En este caso, la clase de la instancia más cercana se asigna como la clase predicha. Sin embargo, cuando K es mayor que 1, es necesario tomar una decisión sobre qué clase elegir, ya que ahora hay múltiples instancias cercanas.

Una estrategia común es considerar la clase mayoritaria entre las K instancias más cercanas. Si la salida del modelo es un número real, se podría optar por calcular la media de los valores de salida de las K instancias más cercanas.

Hay multitud de medidas de distancia que podemos usar:

- $D_{Euclídea}(x, y) = \sqrt{\sum_{i=1}^{N}(x_i - y_i)^2}$

- $D_{Manhattan}(x,y) = \sum_{i=1}^{N}|x_i - y_i|$
- $D_{Minkowski}(x,y) = \sqrt[p]{\sum_{i=1}^{N}(x_i - y_i)^2}$
- Distancia de Levenshtein: número de cambios que hay que hacer para convertir una instancia en otra (casa → cala → calla → calle)
- $D_{Mahalanobis}(x,y) = \sqrt{(x-\mu)^T \cdot C^{-1} \cdot (x-\mu)}$ donde μ es la media de los datos y C^{-1} la inversa de la matriz de covarianzas
- $D_{Euclídea\ ponderada}(x,y,p) = \sqrt{\sum_{i=1}^{N}(p_i(x_i - y_i)^2)}$

K-Nearest Neighbour	
Ventajas	Desventajas
No paramétrico. No hace suposiciones explícitas sobre la forma de los datos	Sensible a atributos irrelevantes
Algoritmo simple	Sensible al ruido
Alta precisión relativa. Alta, aunque no superior a otros modelos.	Ejecución lenta
Entrenamiento inmediato	Ocupa mucha memoria

Case-Based Reasoning

Almacenamiento de sabiduría en base a ejemplos. CBR solo necesita:

- Una base de casos ya resueltos
- Una medida de similitud de casos
- Un conocimiento de como adaptar los casos si no sean iguales

Se diferencia de KNN porque la representación puede ser más compleja, los casos se indexan para mejorar su recuperación, las soluciones se adaptan a los casos nuevos, la base de datos se va actualizando y el usuario puede añadir conocimiento extra.

Se computa la similitud de un caso respecto a otro, igual que KNN. Se obtiene un caso muy similar, pero adaptado al caso actual, como si fuera un correo automático. Si el nuevo caso es correcto, podemos almacenar el caso como ejemplo. Esta técnica requiere por tanto de un sistema de feedback. Sin embargo, cuanto más aumente la base de datos, más tiempo tarda la recuperación de un caso y más espacio ocupa.

Para medir la similitud de 2 casos, se utiliza el producto de dos medidas: la frecuencia de término y la frecuencia inversa de documento.

$$S(t,d) = tf(t,d) \cdot idf(t,D)$$
$$tf(t,d) = \frac{frecuencia(t,d)}{\max\big(f(i,d)\big)\,\forall i \in d}$$
$$idf(t,D) = \log\frac{|D|}{1 + count(t \in D)}$$

También se puede utilizar la similitud del coseno, soft cosine o la correlación de Pearson.

$$cosSim(a,b) = \frac{\sum_{i=1}^{n} a_i b_i}{\sqrt{\sum_{i=1}^{n} {a_i}^2} \cdot \sqrt{\sum_{i=1}^{n} {b_i}^2}}$$

$$softCos(a,b) = \frac{\sum_{i=1,j=1}^{n} s_{i,j} a_i b_j}{\sqrt{\sum_{i=1,j=1}^{n} s_{i,j} a_i a_j} \cdot \sqrt{\sum_{i=1,j=1}^{n} s_{i,j} b_i b_j}}$$

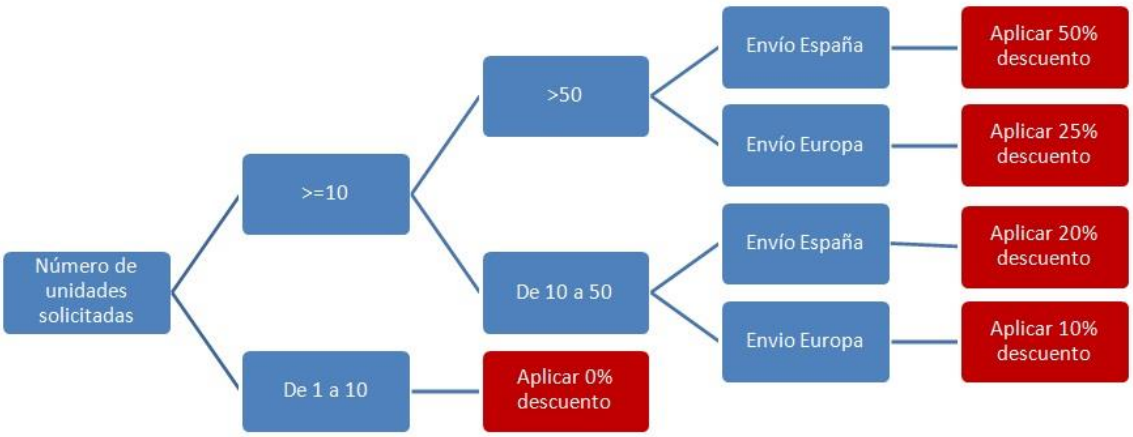
$$\rho(a,b) = \frac{cov(a,v)}{\sqrt{var(a)var(b)}}$$

Case-Based Reasoning	
Ventajas	Desventajas
No paramétrico. No hace suposiciones explícitas sobre la forma de los datos	Sensible a atributos irrelevantes
Puede adaptarse a cambios en el entorno	Sensible al ruido
Alta precisión relativa. Alta, aunque no superior a otros modelos.	Ejecución lenta
Entrenamiento inmediato	Ocupa mucha memoria

Árboles de decisión

Son muy útiles para visualizar las diversas opciones de las que se dispone para resolver un problema o modelar un comportamiento. Se pueden convertir en reglas. Se pueden aplicar métodos de inducción hacia atrás para descubrir el razonamiento que encierra la lógica del sistema.

Un árbol de decisión está compuesto de dos tipos de nodo, un nodo condicional y una clase. Los nodos condicionales son los nodos internos del árbol y la clase, los nodos hoja o terminales. Cada nodo condicional es una pregunta para el sistema acerca de una variable y tiene dos posibilidades, que sea cierta o falsa. En función de eso seguirá un camino y otro. Los nodos finales son nodos hoja y nos dice la clase.



Este algoritmo utiliza la entropía para ayudar a decidir qué atributo debe ser el siguiente en ser evaluado. Es decir, el siguiente atributo seleccionado es aquel que deja la información más ordenada o mejor clasificada.

$$Entropia(s) = \sum_{i=1}^{c} (-p_i \cdot \log_2 p_i)$$

Donde c son los posibles valores de clasificación, S es el conjunto de todos los ejemplos y p_i es la proporción de ejemplos de S que están en la clase i.

$$Ganancia(S, A) = Entropia(s) - \sum_{v \in V(A)}^{|A|} \left(\frac{|sv|}{|s|} Entropia(s_v) \right)$$

Donde V(A) es el conjunto de todos los valores posibles para el atributo A y s_v es un subconjunto de S para el cual el atributo A tiene el valor v.

Como selecciona el atributo más prometedor de entre todos, podemos concluir que realiza una búsqueda voraz entre los mejores atributos. Después aplica Divide y vencerás recursivamente con el problema.

Los árboles de decisión dividen el espacio de soluciones mediante hiperplanos fijando una de las variables a un valor frontera.

Árboles de decisión	
Ventajas	Desventajas
Entrenamiento muy rápido	Tienden al sobreajuste

Algoritmo de caja blanca	Sensible al ruido
Buena precisión para algunos problemas	No se garantiza que el árbol generado sea óptimo
Se pueden convertir fácilmente en reglas	Hay que balancear el conjunto de datos antes de entrenar
Puede trabajar con variables cualitativas y cuantitativas	Puede que sean poco efectivos para modelos sin una aproximación lineal

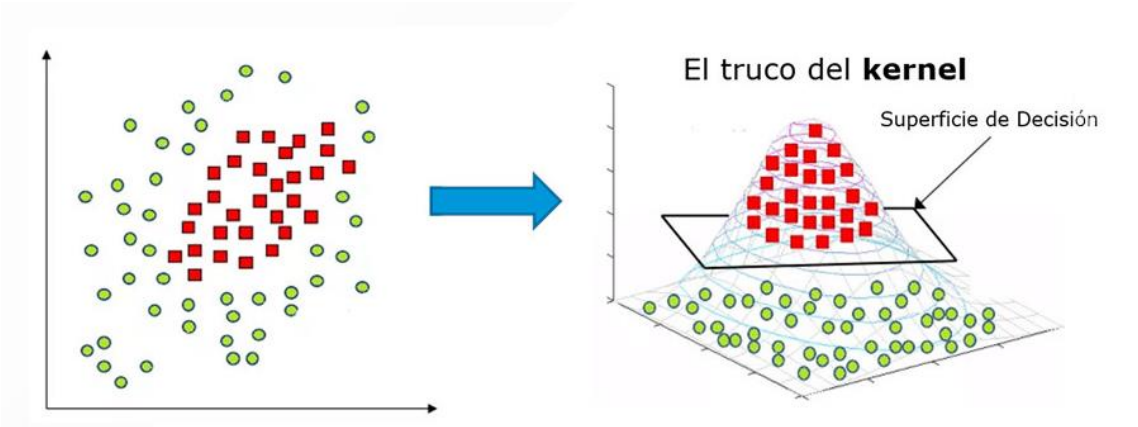
La versión Random Forest ejecuta diferentes árboles de decisión y realiza un proceso de votación para elegir cuál es la predicción final. Se trata entonces de un método de conjunto. Este método obtiene mejores resultados en general que los árboles de decisión convencionales, aunque dificultan la comprensión del modelo generado.

Random Forest	
Ventajas	Desventajas
Resultados muy buenos	Tienden al sobreajuste
Fácil de calcular	Sensible al ruido
Estima qué variables son importantes para clasificar	Más difícil de interpretar que los árboles de decisión convencionales

Support Vector Machine

Los vectores de soporte son los puntos que definen el margen máximo de separación del hiperplano que separa las clases. Se llaman vectores porque estos puntos tienen tantos elementos como dimensiones tenga nuestro espacio de entrada.

Hay veces en las que no hay forma de encontrar un hiperplano que permita separar dos clases. En estos casos, decimos que las clases no son linealmente separables. Para resolver este problema podemos usar un kernel. El truco del kernel consiste en inventar una dimensión nueva en la que podamos encontrar un hiperplano para separar las clases.



Los puntos más cercanos que al hiperplano se conocen como vectores de soporte.

Support Vector Machine	
Ventajas	Desventajas
Eficaz en espacios de grandes dimensiones	Si el número de dimensiones es mucho mayor que el número de muestras es crucial elegir correctamente la función del kernel y el término de regularización
Eficaz en casos donde el número de dimensiones es mayor que el número de muestras	Los vectores de soporte no proporcionan directamente estimaciones de probabilidad, estas de calculan utilizando validación cruzada
Utiliza un subconjunto de puntos de entrenamiento en la función de decisión, por lo que es eficiente en memoria	
Se pueden especificar kernels personalizados	

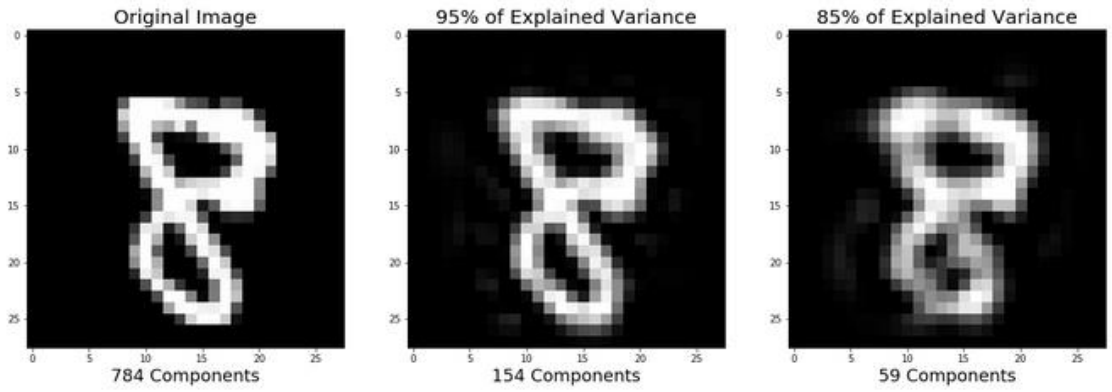
Aprendizaje no supervisado

No siempre se conocen las clases o valores esperados de los datos. Si no se conocen, no podemos hacer aprendizaje supervisado y tenemos que recurrir al aprendizaje no supervisado.

También se aplica aprendizaje no supervisado cuando queremos extraer las características más relevantes de un conjunto de datos.

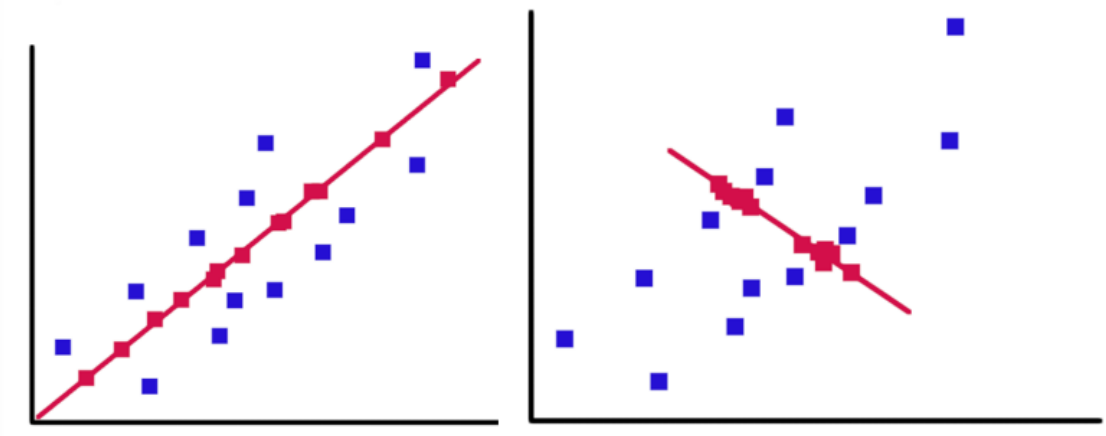
Reducción de dimensionalidad

Los datos con gran número de atributos son difíciles de interpretar y procesar.



El Principal Component Analysis (**PCA**) nos permite reducir la dimensionalidad de los datos de entrada sin perder toda la información.

PCA transforma linealmente las variables correlacionadas en un número menor de variables no correlacionadas. Esto se hace proyectando los datos originales en el espacio reducido utilizando la descomposición en valores y vectores propios de la matriz de covarianza/correlación. Por tanto, PCA es efectivo cuando la correlación es alta.



PCA debe obtener la mayor varianza de los datos proyectados, manteniendo las variables implicadas como independientes entre sí. Existen 2 formas de calcularlo: el método basado en la matriz de correlaciones y el método basado en la matriz de covarianzas.

La matriz de correlaciones se construye con $r_{i,j} = \frac{cov(f_i, f_j)}{\sqrt{var(f_i)var(f_j)}}$.

La matriz de covarianzas se construye con $cov(X,Y) = \frac{\sum_{i=1}^{n}((X_i - \hat{X})(Y_i - \hat{Y}))}{n}$ y después se calculan los eigenvectors y eigenvalues con multiplicadores de Lagrange para obtener los valores máximos de los componentes seleccionados y se derivan.

Los datos deben estar en la misma escala, lo normal es normalizar los datos restándoles la media y dividiendo por su desviación típica.

Reducción de dimensionalidad	
Ventajas	Desventajas
Mejora el desempeño de los algoritmos de aprendizaje automático	Presupone que los datos pueden ser proyectados usando una combinación lineal
Reduce el ruido	Asume correlación entre las características
Mejora la visualización gráfica	Se pierde información
	Se pierde interpretabilidad de los datos
	No es fiable si hay muchos casos atípicos

No usar PCA para prevenir el sobreajuste. Mejor usar regularización. Tampoco usar PCA si se consiguen buenos resultados sin usarlo.

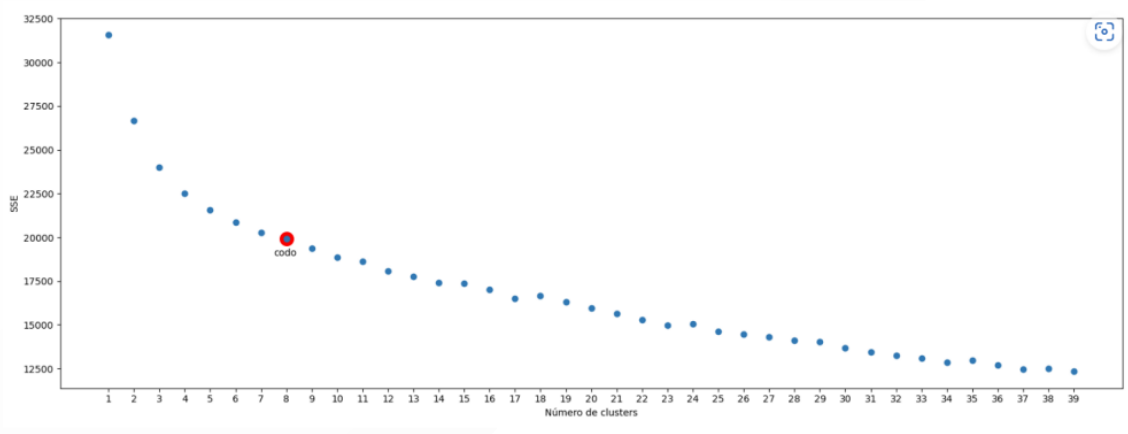
Clustering

El objetivo es agrupar los n individuos de nuestro conjunto de forma que los individuos del mismo grupo sean lo más similares entre sí y los de grupos diferentes sean lo más diferentes entre sí. Estos grupos revelan la estructura de los datos, por lo que suele ser útil hacer clustering aunque sólo sea para analizarlos. 2 estrategias para hacer clustering son la estrategia aglomerativa y K-means.

La estrategia aglomerativa inicialmente crea una clase por cada ejemplo de entrada. Después busca los 2 conceptos más cercanos entre sí y los agrupa. Este proceso se repite iterativamente hasta que no quedan más nodos que agrupar.

K-means elije un conjunto de k semillas, siendo k un parámetro dado por el usuario. Después iterativamente va agrupando los ejemplos en las k clases y recalculando el valor de dicha semilla como su centroide.

Como no sabemos cuántos grupos hay, no podemos saber el valor óptimo de k. Debemos encontrar el valor de k que produce un punto de inflexión en la curva. A esto se le denomina técnica del codo.



El coeficiente de Silhouette mide qué tan cerca está una muestra de otras muestras de su cluster y qué tan lejos está con respecto a las muestras del clúster más cercano. Toma valores de [-1, 1], donde -1 indica que los clusters están superpuestos, 0 que hay overlap y 1 que no se tocan.

El puntaje de Silhouette es el promedio de los coeficientes de Silhouette de todas las muestras. Valores bajos cercanos a 0 nos indica que la calidad de los clusters no es muy buena.

Deep Learning

El algoritmo de aprendizaje basado en gradiente tiene un problema fundamental: cada una de las capas aprende a diferentes velocidades. Se le llama **desvanecimiento o explosión del gradiente**. Las capas más cercanas a la salida tienden a recibir actualizaciones de peso más grandes durante el backpropagation, mientras que las capas más cercanas a la salida pueden aprender muy lentamente o incluso estancarse.

Se ha demostrado que una red con una sola capa puede modelar funciones muy complejas, siempre que tenga suficientes neuronas, pero no alcanzan la eficiencia de las redes profundas, es decir, pueden modelar funciones complejas utilizando **exponencialmente menos** neuronas que las redes poco profundas.

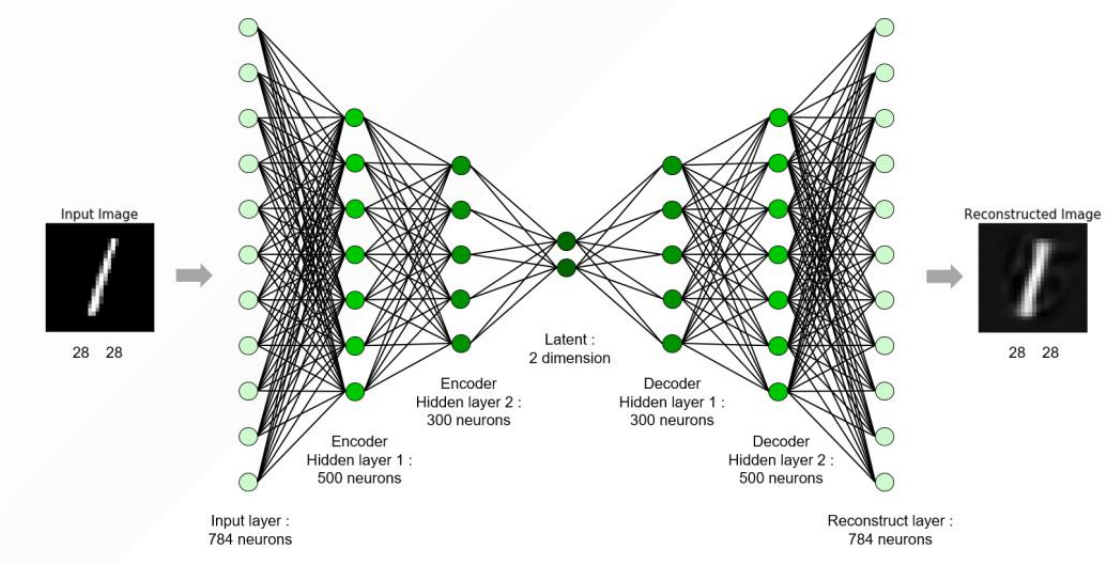
Una práctica común es dimensionar las capas ocultas para formar un embudo, es decir, cada capa tendrá bastantes más neuronas que la siguiente. Esta aproximación se basa en que muchas características de bajo nivel pueden unirse en muchas menos características de alto nivel. Se suele usar la función de activación ReLU en las capas ocultas, y softmax en la capa de salida para tareas de clasificación cuando las clases son mutualmente excluyentes. Si no lo son, generalmente se prefiere la función logística. Para las tareas de regresión, se suelen emplear funciones lineales de activación.

Softmax

Función de activación que normalmente se coloca en la capa de salida que recoge las entradas de la red aplicando la función $y_j = \frac{e^{z_j^C}}{\sum_k e^{z_j^C}}$, donde u_j^C es el umbral $Z_j^C = \sum_k W_{i,j} \cdot y_k^{C-1} + u_j^C$ de cada neurona de salida. Esta función permite interpretar la salida como una probabilidad.

Autoencoders

Los autoencoders son un tipo especial de redes neuronales que funcionan intentando reproducir los datos de entrada en la salida.



Si configuramos la red en forma de doble embudo, podemos dividir la red y obtener una red que comprime la información de entrada en unos pocos valores y otra que los descomprime. Su mayor utilidad es extraer las características más relevantes de una red usando aprendizaje no supervisado.

Suelen utilizar funciones de activación sigmoidal o ReLU en la capa de salida.

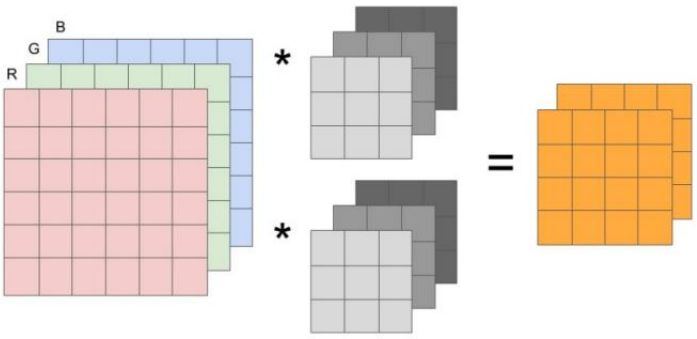
Redes convolucionales

Las redes convolucionales son un tipo especial de redes neuronales para procesar datos con tipología cuadriculada, como las imágenes. Las imágenes tienen demasiados datos. Una imagen de 1000x1000 píxeles a 3 canales por píxel tiene un vector de entrada de 10^6 dimensiones. Necesitamos reducir la complejidad de la imagen conservando su información esencial. La convolución es una operación lineal sobre dos funciones f y g que producen una tercera función s que se expresa cómo una es modificada por la otra y se suele denotar por $\circ$.

$$s(t) = (f \circ g)(t) = \int f(t)g(t-T)\,dT \cong \sum_{T=-\infty}^{\infty} f(t)g(t-T)$$

Donde f es la entrada, s el feature map y g el kernel.

El kernel reduce la dimensionalidad de la imagen.



Otras capas de las redes convolucionales son:

- Capa de pooling
- Capa fully connected
- Capa ReLU
- Capa de dropout

Sus aplicaciones más destacables son la visión artificial en coches autónomos, el reconocimiento facial, la clasificación de imágenes y la transferencia de estilos.

Aprendizaje Automático por Refuerzo

El aprendizaje por refuerzo es un tipo especial de aprendizaje automático que se realiza de forma online, es decir, que no se entrena previamente, sino que aprende en base a prueba y error. Un evaluador, bien humano o una simulación, le indica si su comportamiento es correcto, pero no el resultado correcto.

Estos algoritmos son muy útiles para aprender comportamientos en entornos poco conocidos y cambiantes. Los agentes que utilizan aprendizaje supervisado o no supervisado necesitan entrenarse previamente. Muchas veces el reentrenamiento es costoso y no son ágiles si el entorno cambia. El aprendizaje por refuerzo sí permite adaptarse mejor a cambios.

Proceso de decisión de Markov

Este tipo de problemas juegan con decisiones probabilísticas y no deterministas, principalmente asociadas a los **problemas de decisión de Markov** (MDP) donde:

- El entorno se encuentra en un conjunto de posibles estados S
- Existe una función de transición $T(s, a)$ que es desconocida (si fuera conocida tendríamos una máquina de estados) de forma que dado un estado s y una acción a ejecutada en ese estado, el entorno cambia a un estado s'.
- Una función de refuerzo $R(s, a)$ (normalmente también desconocida) que recompensa al agente por tomar la secuencia de decisiones hasta la decisión a en el estado s a lo largo del tiempo.

El objetivo es encontrar una política, preferiblemente la óptima, que permita seleccionar en cada estado s aquellas acciones que **maximicen en el futuro el refuerzo**.

Q-Learning

Q-Learning es el algoritmo de aprendizaje por refuerzo más conocido y que se basa en el proceso de Markov. La salida del algoritmo es una tabla de políticas $Q(s, a)$ que contiene para cada situación s y acción a, el refuerzo esperado en el tiempo. La condición de parada del algoritmo es el número de ciclos o comprobar que la matriz Q no varía demasiado entre 2 iteraciones. La matriz Q se va actualizando con cada acción del agente siguiendo la siguiente ecuación:

$$Q_t(s, a) = \alpha \left(R(s, a) + \gamma \cdot \max_{b \in A} Q_{t-1}(s', b) \right) + (1 - \alpha) Q_{t-1}(s, b)$$

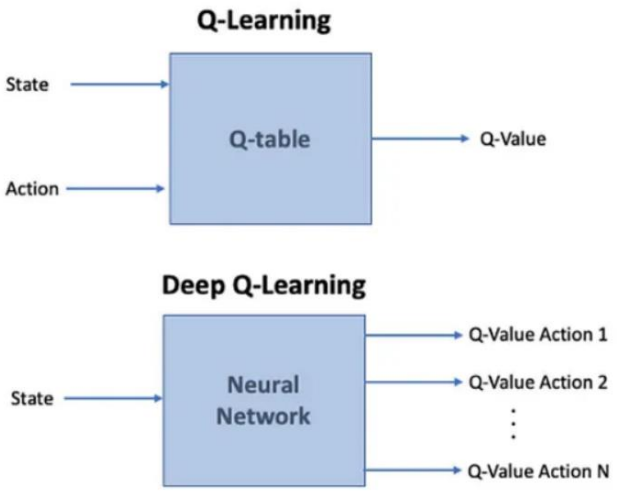
Donde $Q_{t-1}(s,b)$ es el valor que ya tenía la tabla, ponderado por α; $R(s,a)$ es la recompensa directa del estado, si la tiene; $\max_{b \in A} Q_{t-1}(s',b)$ es la recompensa máxima del estado destino de la transición; $\alpha \in [0,1]$ es la tasa de aprendizaje, que regula la importancia que se le quiere dar a la experiencia pasada; y $\gamma \in [0,1]$ es el descuento que tiene la importancia del refuerzo en el tiempo.

La tabla se calcula haciendo programación dinámica. Hay que explorar el espacio de estados para encontrar la política óptima, por tanto, necesitamos repetir el proceso de aprendizaje no siempre tomando las decisiones aprendidas, para adquirir más conocimiento del entorno. La idea es usar una probabilidad e (e-greedy) de elegir la política o de elegir una acción aleatoriamente. El valor de e puede ser dinámico.

Las limitaciones de Q-Learning son que el espacio de estados debe ser finito, ya que es muy complicado que la búsqueda encuentre una política óptima si no lo es. Aun así, es posible que los resultados no sean muy buenos. Se puede hacer una aproximación mixta aprendiendo la tabla Q con aprendizaje supervisado para encontrar más fácilmente la solución.

Aprendizaje por refuerzo profundo

Otra aproximación para aprender la tabla Q es sin un modelo. Deep Q-Learning (**DQN**) utiliza esta aproximación y es la base de AlphaZero y AlphaStar de DeepMind. DQN sustituye la tabla Q por una red neuronal profunda, que predice el valor de Q para todos los posibles estados.



Estas versiones son más inestables, los valores de Q oscilan mucho y se requiere una gran cantidad de datos para que converjan. Se usa la ecuación de Bellman para calcular los valores estimados y luego entrenar la red con ellos.

ML-Agents

ML-Agents es un entorno de aprendizaje basado en DQL que está disponible en Unity.


MACHINE
LEARNING