UNIDAD 1

SISTEMAS DE APRENDIZAJE AUTOMÁTICO

IES SERRA PERENXISA (Torrent) Valencia

INTRODUCCIÓN

Profesor: José Rosa Rodríguez

Curso 2023-2024

OBJETIVOS:

- Diferenciar entre IA fuerte y débil.
- Conceptos generales usados en machine learning
- Los tres tipos de aprendizaje y terminología básica.
- Proceso de diseño de sistemas de machine learning
- Entender la importancia de conocer bien los tipos de algoritmos.
- Tipos de algoritmos usados en el aprendizaje automático.
- Instalar y configurar Python para análisis de datos y machine learning.
- Nociones básicas de estadística, cálculo, álgebra lineal y

BIBLIOGRAFÍA:

- 2017. **Mastering Machine Learning with Python in Six Steps.** "Manohar Swamynathan". Apress.
- 2017. **Introduction to Machine Learning with Python**. "Andreas C. Muller and Sarah Guido". O'Reilly**.**
- 2023. **Data Mining Concepts and Techniques, 4Ed.** "Jiawei Han, Jian Pei and Hanghang Tong". **Morgan Kaufmann Publishers.**
- 2017. **Statistics and Machine Learning in Python.** "Edouard Duchesnay, Tommy Löfstedt, Feki Younes". O'Reilly**.**
- 2019. **Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow. Concepts, Tools, and Techniques to Build Intelligent Systems.** "*Aurélien Géron*". ***O'Reilly***
- 2019. **Python Machine Learning, 3Ed (Machine Learning and Deep Learning with Python, scikit-learn, and TensorFlow).** "Sebastian Raschka y Vahid Mirjalili". Apress.
- https://scikit-learn.org/stable/
- Sistemas de Aprendizaje Automático. Ed Ra-Ma
- Kevin Markham: https://github.com/justmarkham
- Pandas docs: http://pandas.pydata.org/pandas-docs/stable/index.html

TABLA DE CONTENIDOS

1. INTRODUCCIÓN AL MACHINE LEARNING.

Estamos en el siglo de los datos, vivimos en la sociedad de la información, un mundo globalmente conectado en lo que Internet ha tenido mucho que ver. Nunca se ha tenido la capacidad de generar, almacenar y procesar tal cantidad de datos.

La ciencia de datos es un término general que incluye otras áreas y tecnologías como Big Data, minería de datos (Data Mining), IA (Inteligencia Artificial), Aprendizaje Automático (**ML**[1]), Aprendizaje Profundo (DL[2]), etc.

El término IA aparece formalmente en 1956 acuñado por *John McCarthy* durante la segunda conferencia de *Darmouth*. Aunque en un principio, el objetivo inicial de la IA era construir una máquina capaz de seguir las leyes de la lógica del pensamiento humano a la perfección y sin equivocarse, pronto se vio que esto era algo casi utópico. Tratar de programar todas las reglas necesarias, con sus propias correlaciones y generalizaciones, era una misión inabarcable y que siempre acababa en alguna incoherencia o fallo.

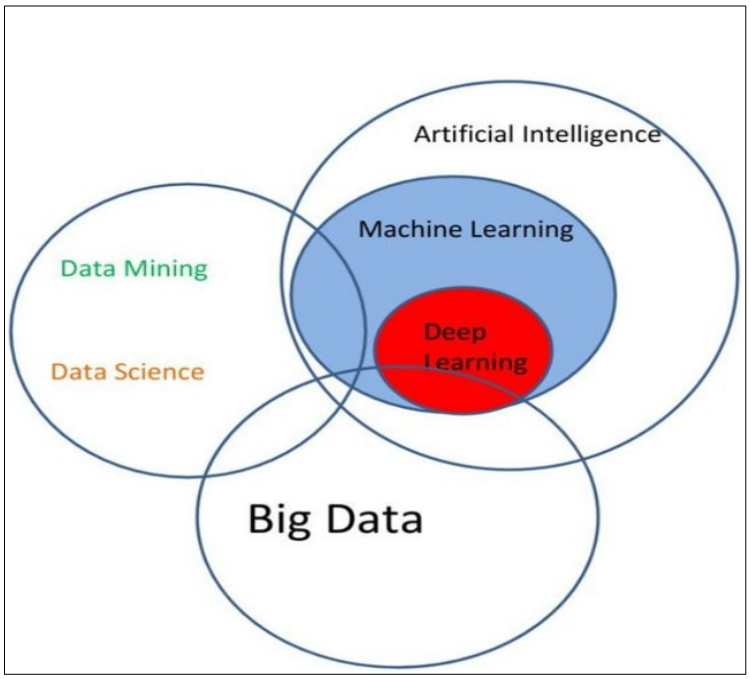


***Figura 1**. Comparación de diferentes tecnologías relacionado con estudio de datos.*

Los sistemas expertos, marcaron la clara separación entre dos tipos de inteligencia artificial. La **IA fuerte** o general, similar a la mente de los seres humanos y la **IA débil** o estrecha, capaz de realizar con gran precisión una tarea muy concreta, dentro de un campo de especialización.

Toda la evolución que ha habido en el ámbito del ML y el DL se ha desarrollado en el plano de la IA débil. Aun así, cada cierto tiempo, se van produciendo hitos y avances que vuelven a rescatar la idea de contar con una IA fuerte como la que aparece en las películas y novelas de ciencia ficción, capaz de relacionarse y servir a los humanos en modo multipropósito.

Aunque la mayoría de alusiones a las diferencias entre ambas se refieren casi siempre a la capacidad de imitación de la inteligencia humana, esto no es del todo correcto. La IA actualmente no se centra en emular las capacidades o comportamientos humanos, a no ser que nos refiramos a robots de tipo humanoide o videojuegos, y ni siquiera sería del todo exacta esta concepción en esos casos. La clave de la IA actual es la evolución y avance de la automatización.

---

1 ***ML***: *Machine Learning, aprendizaje automático o aprendizaje máquina.*
2 **DL**: Deep Learning, aprendizaje profundo. Es un tipo específico de ML.

La mayoría de desarrollos actuales considerados como IA débil superan con creces las capacidades humanas en ese ámbito de aplicación. La capacidad de abstracción y creatividad de algunos modelos también empiezan a mostrar un alcance sorprendente.

En definitiva, es recomendable no basar la distinción entre ambas IA's en la comparación con la inteligencia humana y centrarse más en la capacidad de realizar tareas en un único ámbito o contexto o ser capaz de abordar tareas muy diferentes en cuanto a entorno de aplicación y resultados.

A algunos expertos les gusta la idea de que la IA débil representa el punto en el que estamos ahora y que la IA fuerte o general es el siguiente paso evolutivo o el punto al que nos dirigimos.

1.1. INTELIGENCIA ARTIFICIAL DÉBIL Y FUERTE.

IA DÉBIL

El concepto de IA débil se refiere a **agentes y modelos inteligentes especializados en un tipo de tarea**, dentro de un campo de especialización concreto. Por ejemplo, un ordenador que juega al ajedrez o un modelo de reconocimiento de imágenes.

Diseñar una IA de este tipo no es igual en el modelo clásico que en el paradigma actual del ML. Los sistemas expertos que se programaban a base de reglas del tipo "**if-else**" del modelo clásico, eran más rígidos y no era nada sencillo actualizarlos o hacerlos evolucionar. Además, dependían mucho del equipo que lo diseñaba y construía, con el alto riesgo de introducir sesgos[3].

Ventajas de la IA Débil:

- Actualmente es relativamente sencilla de conseguir y desarrollar.
- Se controla más fácilmente y no presenta comportamientos demasiado inesperados.
- Permite arquitecturas modulares, con las que combinar distintos talentos o capacidades según las necesidades del entorno en el que se despliega.

Inconvenientes de la IA Débil:

- Es un sistema "ciego", que no cuenta con todo el contexto de la situación y eso provoca que puede mal interpretar entradas inusuales.
- Es muy limitada, solo sirve para aplicaciones muy concretas.
- Suelen ser sistemas que no ofrecen una experiencia de usuario tan cercana o personalizada como la de un ser humano.

En resumen, una IA Débil está concebida para ser aplicada en un ámbito específico de problemas, y no tanto de tener un comportamiento similar a la inteligencia humana. Desde este punto de vista, es impensable concebir que uno de estos sistemas constituyan una mente, un estado mental o una conciencia propia de sí mismos.

Ejemplos de IA débil:

- **Alexa, Siri y Google Assistant:** Estos modelos de asistentes virtuales presentes en móviles y dispositivos para el hogar, son lo que se conoce como asistentes conversacionales. Son un ejemplo de que una IA estrecha puede simular un comportamiento humano pero no es capaz de llegar a dar el mismo resultado o ejecutar la misma tarea que la inteligencia humana proporcionaría.
- **IBM Watson:** Este sistema cognitivo es capaz de dotar de la información precisa y necesaria en cada momento bajo demanda y en un contexto de conversación en lenguaje natural o coloquial. Puede analizar gran cantidad de datos y extrapolar o inferir conclusiones a partir de ellos.
- **AlphaGo:** sistema desarrollado por *DeepMind* que ha sido capaz de batir en el juego *Go* al campeón mundial, como hizo *Deep Blue* (IBM) en su día con el campeón *Kasparov*. Otras

3 **Sesgo**: errores, tendencias o interpretaciones que favorecen ciertos resultados en perjuicio del resultado correcto.

versiones del mismo sistema han sido *AlphaStar*, capaz de jugar a un conocido videojuego, el *Starcraft y AlphaZero*, una generalización de *AlphaGo* que también juega al *shogi* y al ajedrez. Es interesante analizar cómo los creadores de *AlphaGo* han logrado esta generalización del algoritmo. En concreto, la red neuronal en la que está basado, se actualiza constantemente, lo que permite en cierta manera que el modelo se adapte en los primeros movimientos al sistema de juego. ¿Podría ser ésto una forma de ir accediendo a una inteligencia artificial general?

- **Coches autónomos:** Un vehículo autónomo parece una unidad compleja que es capaz de desarrollar varias tareas muy diferentes y puede tener la apariencia de ser una IA fuerte o general, pero no. En un coche autónomo, existen diferentes sistemas que se hacen cargo de diferentes tareas, cada uno de la suya y que son coordinados o controlados por un módulo de control. Cada parte del sistema es una IA débil, controlados por otra IA débil especializada en llevar a cabo ese control de otros sistemas concretos.
- **Algoritmos de recomendación:** El acto de recomendar una película, un libro o un sitio que visitar parece tan humano que hace difícil clasificar este tipo de sistemas como IA débil. Tras una recomendación suele haber vivencias y experiencias muy variadas, entremezcladas con diversas acciones y hechos. Y lo cierto es que estos sistemas se entrenan con conjuntos de datos que configuran, con un cierto grado de simplificación, los distintos perfiles de gustos de las personas según ciertos parámetros de experiencias registradas.

IA FUERTE

El concepto de IA Fuerte o General proviene más de la imaginación o de la ciencia ficción que de una arquitectura o concepto factible real. Podemos decir que es el punto al cual queremos llegar en este campo, pero no termina de estar claro si es posible con los algoritmos actuales.

El concepto de la IA Fuerte o General parte de la posibilidad de que una máquina pueda abstraer hasta el nivel de interaccionar con seres humanos y con el resto de objetos y seres del mundo físico en todo tipo de contexto y con objetivos de todo tipo. En definitiva, se quiere emular la capacidad de interpretación y adaptación que tiene el ser humano. Pero, como ya se ha mencionado, no se trata de imitar a los humanos, sino de poder acometer las tareas necesarias para múltiples contextos y hacerlo igual o mejor que los humanos.

Aproximación filosófica al concepto de IA fuerte

La posibilidad real de la IA Fuerte se ha empezado a estudiar sobre todo a partir de que el filosofo **Searle** presentara un contra-argumento a ésta en la publicación «The Chinese Room Argument» así como a raíz del argumento del primer teorema de incompletitud de Gödell, que afirma que nunca una maquina podría adquirir algún tipo de inteligencia humana. Los contra-argumentos a la IA fuerte no han dejado de aparecer, como por ejemplo el filosofo heideggeriano Hubert Dreyfus, que dedicó gran parte de su obra argumentando como la IA llegaría a fallar porque simplemente el cerebro humano y mente humana no son análogas con el hardware y software de un ordenador. Y sin duda, nuevos estudios demuestran que la mente humana no es como un ordenador.

A partir de este punto, hay dos razones que hacen que tenga una mayor consideración la IA débil:

- Desde la vertiente lógica, se pueden estudiar los algoritmos y las funciones matemáticas con mejor claridad.
- Desde una perspectiva clásica de la filosofía, la debilidad intrínseca de la IA se podría argumentar a partir de la siguiente cuestión: nosotros los humanos fabricamos las herramientas y sistemas y por tanto, incluyen dentro nuestra propia fragilidad, la posibilidad del fallo.

Como dice *Jean Francois Dortier* en "*La cerveau et la pensée*": «Cuarenta años después de su creación, la importancia de la IA está, cuando menos, mitigada. Los especialistas se refieren cada vez más (después del análisis de *Searle*) al proyecto de una «IA débil» opuesto a la «IA fuerte» de los

primeros pasos de la IA. El proyecto de la «IA fuerte» era encontrar y reconstruir la forma en que el hombre piensa y después superarla. El proyecto de la «IA débil» es más modesto. Se trata de simular comportamientos humanos «considerados inteligentes» por los métodos de la ingeniería, sin preocuparse de saber si el hombre actúa de la misma manera. Hoy día se prefiere hablar de lógicas «de ayuda» a la creación o a la toma de decisiones más que de máquinas que remplazan al hombre».

Aproximación técnica a una arquitectura multipropósito
Más allá de que realmente sea posible construir un modelo con las capacidades de conciencia, sensibilidad, sabiduría y autoconocimiento, lo que sí se está explorando, por el momento, es la construcción de modelos basados en redes neuronales profundas, que sean capaces de ejecutar tareas en diferentes contextos.

Por ejemplo el sistema "Gato", según anunció el laboratorio de IA propiedad de *Alphabet*, puede jugar videojuegos de *Atari*, subtitular imágenes, chatear y apilar bloques con un brazo *robótico* real. En general, Gato puede realizar hasta 604 tareas diferentes. Lo malo es que Gato no realiza las tareas tan bien como aquellos modelos que solo pueden hacer una cosa. Gato es un modelo "generalista" en el sentido que puede hacer muchas cosas diferentes al mismo tiempo. Pero eso es muy distinto a una IA "general" capaz de adaptarse de manera eficaz a nuevas tareas distintas a aquellas en las que el modelo fue entrenado.

Conciencia artificial
El paso final del desarrollo de la IA es construir sistemas que puedan formar representaciones sobre sí mismos. En última instancia, los investigadores de la IA tendrán que comprender no solo la conciencia, sino también construir máquinas que la tengan.

Nada garantiza que el desarrollo de una IA fuerte o general implique el surgimiento de la conciencia artificial o que se genere una conciencia emergente, pero esta tiende a ser considerada como la posibilidad más plausible. La razón por la cual se cree que la existencia de uno de estos conceptos probablemente lleve al otro, deriva de como de intrínsecamente entrelazadas se encuentran la conciencia y la inteligencia general en los seres vivos.

Peter Voss, profesional del campo de la IA fuerte, justifica la coincidencia fundamental de las cualidades necesarias para el desarrollo tanto de un sistema de IA fuerte como de una conciencia artificial afirmando que las personas poseen "autoconciencia conceptual" (conceptos abstractos del "yo" físico y mental), característica que resultaría imprescindible en dicha IA, ya que esta tendría que "poder conceptualizar qué acciones ha tomado, de qué acciones es capaz y cuáles son sus efectos más probables".

En plena carrera por el mejor algoritmo conversacional, *Blake Lemoine*, un ingeniero de Google se refirió al sistema de IA en el que trabajaba, *LaMDA*, como una máquina pensante y que sienta. El *LaMDA* (*Language Model for Dialogue Applications*, modelo de lenguaje para aplicaciones de diálogo en español) fue diseñado por Google en 2017 y tiene como base un transformer, es decir, un entramado de redes neuronales artificiales profundas. Para los expertos en IA, transformers como LaMDA han supuesto un hito porque "permiten un procesamiento (de información, de textos) muy eficiente y han producido una auténtica revolución en el campo del ***NLP***[4] Procesamiento del Lenguaje Natural".

Pero, aunque la conversación sea fluida, de calidad y específica, no es más que una enorme fórmula que ajusta los parámetros para predecir mejor la siguiente palabra. No tiene ni idea de lo que está hablando. En definitiva: no tiene autoconciencia ni sensibilidad como tales.

4 ***NLP*: *Natural Language Processing*, *procesamiento de lenguaje natural.***

1.2. IA Y APRENDIZAJE AUTOMÁTICO (MACHINE LEARNING:ML).

La IA y el ML no son lo mismo. De hecho, el ML es una disciplina dentro de la IA. La IA alcanzó un primer apogeo en la década de 1980 y entró en crisis a finales. Obtuvo ciertos éxitos como el desarrollo de sistemas expertos. Por aquel entonces el ML estaba surgiendo muy lentamente. El desarrollo clásico de un sistema IA consistía en que un grupo de científicos (matemáticos, psicólogos, informáticos, etc.) estudiaban como realizaban su trabajo expertos en cierta área muy concreta (reparación de maquinaria, diagnósticos médicos, búsqueda de yacimientos, etc.). Estos científicos intentaban extraer y plasmar ese conocimiento en sistemas de reglas. Luego, estas reglas se programaban y se generaba un sistema experto que podía aprovecharse como asistente a la toma de decisiones, para entrenar a otros humanos en ese área, etc.

Pero incluso la IA que ya había superado perseguir sistemas de IA fuerte y se dedicaba a desarrollar con cierto éxito sistemas expertos (IA débil) pronto entró en crisis porque estos sistemas eran muy complicados de fabricar, caros y difíciles de actualizar.

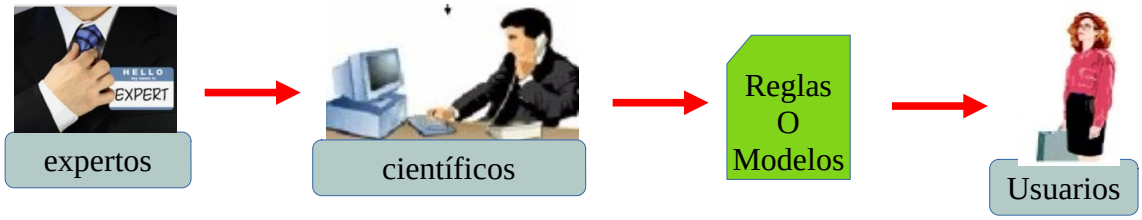


***Figura 2**. Desarrollo clásico de un sistema de IA.*

El machine learning tiene otro enfoque. En vez de ser los humanos quienes generan las reglas, que luego programarán para que lo usen los usuarios, los algoritmos generan las reglas a partir de procesar muchos datos.

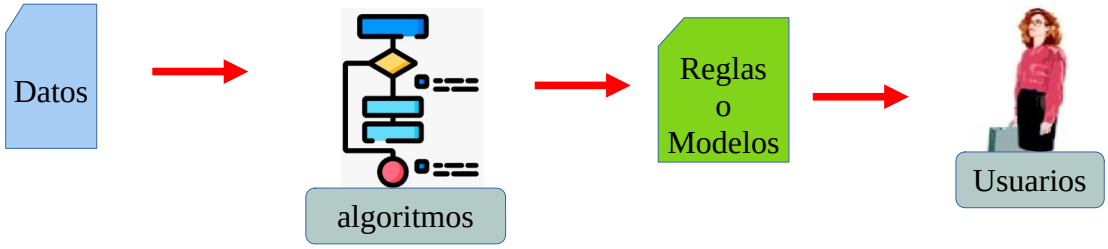


***Figura 3**. Desarrollo en Machine Learning.*

Realiza el ejercicio 1.

En resumen: la IA y el ML son dos disciplinas interrelacionadas pero distintas. La IA se enfoca en desarrollar sistemas que imiten la inteligencia humana, mientras que el ML se centra en desarrollar algoritmos y técnicas que permitan a las máquinas aprender de los datos sin ser programadas explícitamente por humanos. Un sistema IA puede no aprender a partir de datos.

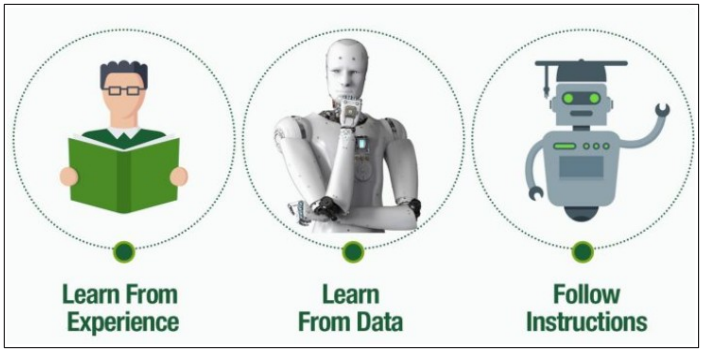


***Figura 4**. IA es más amplia que ML. ML es una parte de la IA.*

Diferencias clave entre IA y ML:

- **Ámbito de trabajo**. IA es más general, se ocupa de cualquier cosa que permita a ordenadores imitar la inteligencia humana incluyendo robótica, resolución de problemas, reconocimiento del lenguaje. ML desarrolla algoritmos que aprenden de datos.
- **Objetivo:** La IA intenta crear sistemas que realicen tareas que necesiten de la participación de inteligencia humana. ML persigue crear sistemas que aprendan de datos.
- **Aprendizaje:** Un sistema IA no necesariamente aprende de datos. Por ejemplo un sistema experto puede definirse con un conjunto de reglas prefabricadas. ML genera esas reglas a partir de los datos que se le proporcionen, a más datos y de mejor calidad, mejor aprenden.
- **Dependencia**. ML es una parte de la IA, pero hay IA más allá del ML.
- **Tipos de aprendizajes:** ML puede aprender de manera supervisada, no supervisada y reforzada. IA puede basarse o no en reglas, o combinar varios tipos de aprendizaje incluyendo pero sin limitarse a modelos o técnicas de ML.
- **Intervención humana:** En IA el papel o la intervención humana puede ser muy variada (mucha intervención o poca). En ML se intenta minimizar esta intervención para automatizar el proceso de aprendizaje.

Inconvenientes del ML:

- **Necesidad masiva de datos**.
- **Necesidad masiva de cálculos**: esto provoca también otros problemas (hardware específico, consumo de energía, huella de carbono, etc.) Mira el artículo.
- Dificultades en interpretar o explicar los resultados.

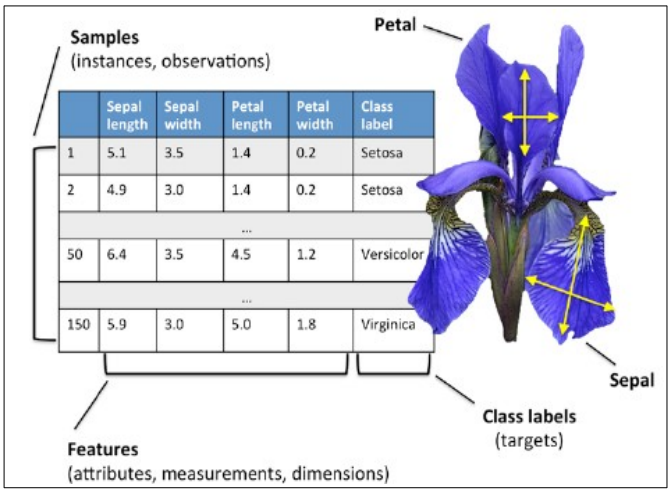
Realiza el ejercicio 2.

Realiza el ejercicio 3.

Realiza el ejercicio 4.

2. NOTACIÓN Y TERMINOLOGÍA BÁSICA.

En un sistema de ML los algoritmos aprenden a partir de datos. Los datos pueden ser estructurados (tablas SQL, hojas de cálculo, etc.) o desestructurados (audio, texto, imágenes, vídeo, etc.). Los datos usados por los algoritmos de aprendizaje se organizan en **datasets**. Nosotros por ahora usaremos datos estructurados.



	Sepal length	Sepal width	Petal length	Petal width	Class label
1	5.1	3.5	1.4	0.2	Setosa
2	4.9	3.0	1.4	0.2	Setosa
...					
50	6.4	3.5	4.5	1.2	Versicolor
...					
150	5.9	3.0	5.0	1.8	Virginica

***Figura 5**. Trozo de datos del dataset Iris.*

La tabla de la figura 5 contiene una parte del dataset Iris que es un clásico ejemplo. Este conjunto de datos contiene las mediciones de algunas características físicas de 150 flores de tipo Iris (**lírios**) de 3 especies diferentes: *Setosa*, *Versicolor* y *Virginica*. Cada fila de la tabla representa una flor y como describe algo del mundo real se le llama **ejemplo** (**sample**) o **muestra** o **fila** o **instancia**.

Cada **ejemplo (fila o instancia)** de un dataset contiene uno o más valores de ciertas características que describen uno de los elementos que describen (en este caso una flor). En este caso contiene mediciones en centímetros de algunas partes de cada flor. A cada una de estas columnas se las llama **características (features o columnas)** del dataset:

Para representar e implementar de manera eficiente y sencilla esta notación se puede usar álgebra lineal. Se utilizan vectores y matrices para representar y manipular estos datos.

Los vectores son listas de valores (**como matrices de una sola columna**) que pueden representarse como un conjunto de valores numéricos. La notación más usada escribe un vector con una letra minúscula en negrita. Por ejemplo un vector **v** de n valores es una lista de n números reales: $\mathbf{v} \in \Re^{n \times 1}$. Por ejemplo, la j-ésima (ocupa la posición j) columna del dataset Iris es un vector de 150 valores. Si prestas atención verás que los elementos del vector se organizan verticalmente unos debajo de otros.

$$\boldsymbol{x_j} = \begin{bmatrix} x_j^{(1)} \\ x_j^{(2)} \\ \dots \\ x_j^{(150)} \end{bmatrix}$$

Pero también es muy frecuente escribir los vectores de manera horizontal. En este caso se dice que hemos aplicado la operación transpuesta al vector. Transponer un vector o una matriz consiste en intercambiar filas por columnas. Para indicar que un vector o una matriz está transpuesto o transpuesta se utiliza una letra t o algún otro símbolo. Por ejemplo, la i-ésima fila (la fila que ocupa la posición i en el dataset) es un vector transpuesto de 1 fila y 4 columnas: $\mathbf{x}^{(i)} \in \Re^{1 \times 4}$

$$\boldsymbol{x}^{(i)} = \begin{bmatrix} x_1^{(i)} & x_2^{(i)} & x_3^{(i)} & x_4^{(i)} \end{bmatrix}$$

Las matrices son como una tabla devalores. Normalmente se representan los ejemplos como las filas de una matriz y cada característica como una columna de la matriz. Y en textos se utiliza una letra mayúscula para representar todos los datos de una matriz, por ejemplo **X**.

Para representar cada ejemplo indicamos su posición en la matriz usando un superíndice entre paréntesis, así que el ejemplo 30 de **X** (su fila número 30) la escribimos como $\mathbf{x}^{(30)}$. Una fila sería un vector transpuesto.

De manera similar, una columna sería un vector. Para representar todos los datos de una columna usamos un subíndice, así que todos los datos de la característica 4 la escribiríamos como $\mathbf{x}_4$.

Para referirnos a un único dato, usamos una letra minúscula que tiene tanto el número de ejemplo como el número de característica a la que da valor (el superíndice indica la fila y el subíndice indica la columna), así que el dato del ejemplo 6 en la característica 2 del dataset **X** se escibiría como $x^{(6)}_2$.

El dataset Iris que consiste en 150 ejemplos y 4 características podemos escribirlo como una matriz de 150 filas y 4 columnas a la que podemos dar el nombre de **X**, donde $\mathbf{X} \in \Re^{130 \times 4}$.[5]

$$\begin{bmatrix} x_1^{(1)} & x_2^{(1)} & x_3^{(1)} & x_4^{(1)} \\ x_1^{(2)} & x_2^{(2)} & x_3^{(2)} & x_4^{(2)} \\ \vdots & \vdots & \vdots & \vdots \\ x_1^{(150)} & x_2^{(150)} & x_3^{(150)} & x_4^{(150)} \end{bmatrix}$$

Alguna terminología usada en Machine Learning

- **Modelo**: elemento que representa el conocimiento que han aprendido los algoritmos de machine learning a partir de datos y que permite realizar predicciones, clasificaciones, etc. El modelo contiene el conocimiento que han podido extraer los algoritmos de los datos y ayuda a solucionar un determinado tipo de problema.
- **Entrenar (training)**: Entrenar tiene el significado de proporcionar datos al modelo para que aprenda a partir de esos datos y pueda definir un modelo.
- **Ejemplo para entrenamiento (train)**: es un ejemplo (fila) que se utiliza para entrenar al modelo. Es una fila de datos que se utiliza para ayudar a definir el modelo, para aprender a partir de esos datos.

---

5 $\Re$: *esta letra identifica al conjunto de los números reales. En matrices al indicar índices, primero se proporciona las filas y luego las columnas.*

- **Característica (feature)**: cada una de las columnas que describen una propiedad de los ejemplos. Otros nombres que se le dan son: variable independiente, entrada, predictora, atributo, covariable. En definitiva son los datos de una columna de un dataset.
- **Target**: una de las características que se utiliza para definir lo que el modelo debe aprender a calcular. También se le llama variable respuesta, variable dependiente, label, clase, nivel de certeza.
- **Función Loss (pérdida)**: también conocida como función de coste o función de error. Se utiliza para medir el error que comete el modelo a la hora de realizar su trabajo.[6]

3. TIPOS DE SISTEMAS DE MACHINE LEARNING.

En este apartado vemos diferentes tipos de sistemas de ML para ir conociendo sus características.

3.1. BASADOS EN INSTANCIAS O EN MODELOS.

Este criterio de clasificación tiene que ver con la forma en la que generalizan. Es decir, los algoritmos de ML extraen conocimiento a partir de los datos de entrenamiento, y a este proceso de resumir o extraer esa información oculta que pueden aplicar a nuevos datos se le llama generalizar: a partir de datos concretos extraen datos más generales al que llamamos conocimiento. Por ejemplo: la temperatura ahora es de 10ºC, luego 9ºC, más tarde 8ºC y tanto tú como un sistema de ML puede haber aprendido que si la temperatura está bajando entonces vas a sentir frío, entonces necesitarás abrigo….). Ahora bien, hay varias formas de realizar este proceso.

La mayoría de tareas de ML están relacionadas con hacer predicciones. Esto significa que a partir de cierta cantidad de ejemplos el sistema necesita poder generalizar para hacer sus predicciones con ejemplos que nunca antes ha visto. Hacer bien este trabajo con los datos usados para entrenar es bueno, pero insuficiente, el objetivo es hacerlo bien con nuevos datos. Hay dos aproximaciones para conseguir la generalización: basada en instancias y basada en modelos.

APRENDIZAJE BASADO EN INSTANCIAS

Es la forma de aprendizaje más trivial. Si quisiéramos hacer un sistema de filtrado de spam, podríamos partir de mails previamente etiquetados por los usuarios y cuando encontremos un mail que case con uno de los etiquetados, indicar que es spam. No es mala solución, pero tampoco es la mejor. Es decir, usamos cada ejemplo o instancia como una referencia con la que comparar un nuevo mail.

En vez de marcar como spam los mails idénticos a los que el usuario ha marcado como spam previamente, también podríamos dar por spam los que sean muy parecidos aunque no idénticos. Esta mejora obliga a definir una manera de medir lo que consideramos "parecido". Una medida muy básica es contar el número de palabras que tienen en común ambos mails. Así que el sistema podría marcar como spam un mail que tenga muchas palabras comunes con alguno que sea spam.

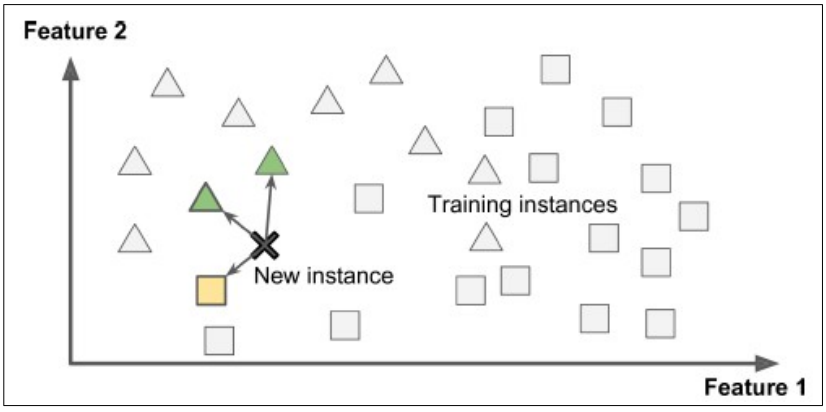


Figura 6. Aprendizaje basado en instancias.

6 ***Loss*** *se utiliza cuando nos referimos a la pérdida de exactitud cuando se considera un solo dato, mientras que* ***coste*** *se utiliza cuando se totalizan (con sumas o medias) esos errores con todos muchos datos del dataset.*

Por tanto los sistemas basados en instancias generalizan comparando los nuevos ejemplos con los ejemplos que ya tienen procesados (o un subconjunto de los que tienen), usando medidas de similitud. Por ejemplo, en la figura 6 la nueva instancia puede clasificarse como triángulo porque la mayoría de las instancias más cercanas (parecidas) a ella son triángulos.

APRENDIZAJE BASADO EN MODELOS

Consiste en fabricar un modelo de los ejemplos y usar el modelo para realizar la tarea. El modelo abstrae o generaliza el conocimiento aprendido a partir de todos los ejemplos usados.
Por ejemplo, un algoritmo basado en modelo que quisiera clasificar los datos en una de las dos clases triángulo y cuadrado podría definir un modelo que consista en una frontera de separación entre los elementos de ambas clases.

Definiría de alguna manera la línea discontinua de la figura 7 que separa las zonas que ocupan ambas clases. Cuando deba clasificar un nuevo ejemplo (la X de la figura) debe usar el modelo para saber a cuál de las dos zonas pertenece en función de sus datos feature1 y feature2, que indicarán que pertenece a la clase triángulo.

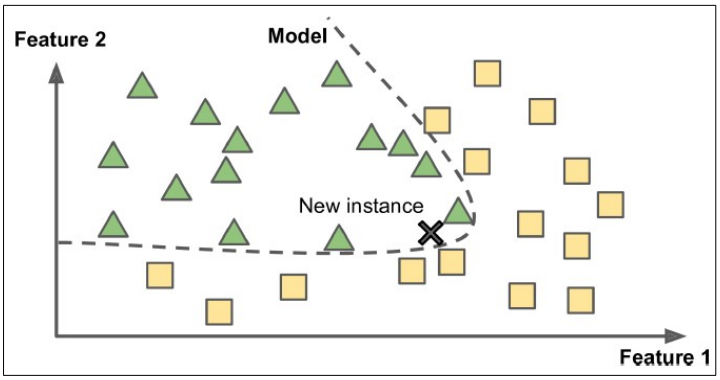


Figura 7. Aprendizaje basado en modelos.

La ventaja de los sistemas basados en modelos es que son más eficientes al realizar su tarea que sus equivalentes basados en instancias, además, son más fáciles de actualizar y no necesitan tantos recursos de computación para usarlos una vez desplegados.

3.2. POR EL TIPO DE APRENDIZAJE.

Los sistemas de ML pueden clasificarse también según la cantidad de supervisión (intervención humana) que se produce durante su entrenamiento (fase de aprendizaje). Atendiendo a este criterio podemos identificar 4 tipos principales de aprendizaje:

- Supervisado.
- No supervisado.
- Semisupervisado (SSL[7]).
- Reforzado.

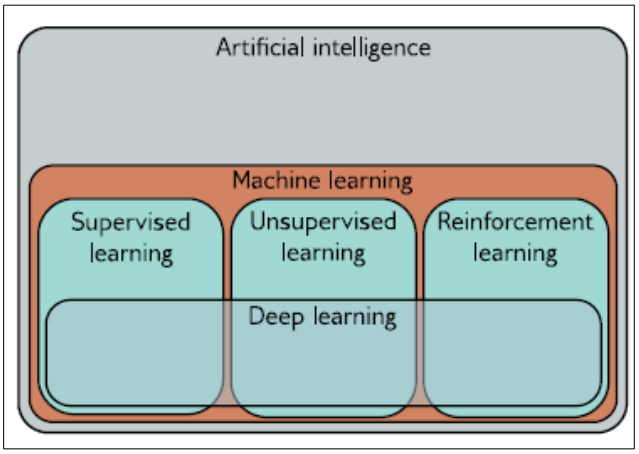


***Figura 8**. Tipos de ML por el tipo de aprendizaje.*

7 ***SSL**: Semi-Supervised Leaning.*

APRENDIZAJE SUPERVISADO

El aprendizaje supervisado define un modelo que es capaz de realizar un mapeo desde datos de entrada a un resultado denominado predicción. Los datos que utiliza para entrenar (aprender) y generar el modelo, incluyen una **etiqueta (label)** que es el valor o solución asociada a cada dato.

A la característica que se utiliza como solución de cada ejemplo se le llama característica o columna **label** del ejemplo o también **target** o **variable dependiente**.

Al resto de características de cada ejemplo que se utilizan para poder averiguar la columna target se les llama **predictoras** o **variables independientes**.

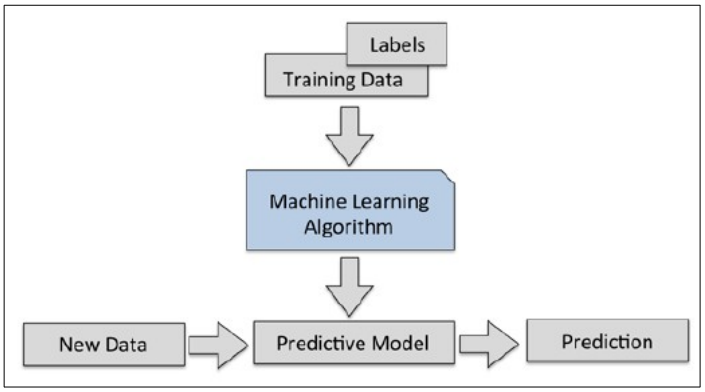


***Figura 9**. Funcionamiento de un sistema de aprendizaje supervisado.*

Una tarea típica que realizan los sistemas supervisados es **clasificar**. Un buen ejemplo serían los filtros o detectores de spam en el correo electrónico. Se entrenan con muchos mails de ejemplo que pertenecen a una de estas dos clases: spam o ham.

Así que para entrenarlo, le facilitamos muchos mails de prueba y cada uno lleva una etiqueta indicando si es spam o no lo es. El sistema aprende a identificar las características que tienen los mail de tipo spam y ese conocimiento lo plasma en un modelo.

Cuando el modelo ya está definido, le pasaremos los nuevos mail que se reciban y el sistema deberá clasificarlos como spam o no.

Es un aprendizaje supervisado porque el algoritmo de aprendizaje utiliza la etiqueta del tipo de mail de cada ejemplo para aprender.

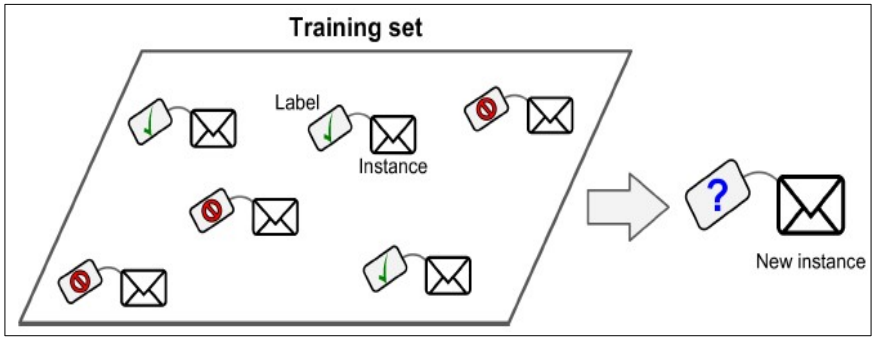


***Figura 10**. Datos de entrenamiento etiquetados con las soluciones.*

Otra tarea típica de estos sistemas es predecir un valor numérico a partir de otras características. Podría ser indicar el precio de un coche a partir de otras características.

La **característica target** sería el precio y las otras características (antigüedad, kilometraje, marca, modelo, etc.) serían las **características predictoras**. A la tarea de predecir un valor numérico se le llama **regresión**.

Para entrenar, hay que proporcionar muchos ejemplos que contengan las predictoras y el target de cada ejemplo.

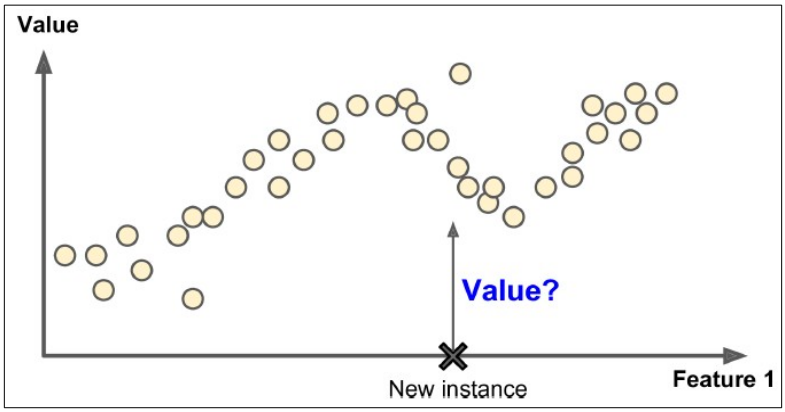


***Figura 11**. Datos de entrenamiento para la regresión.*

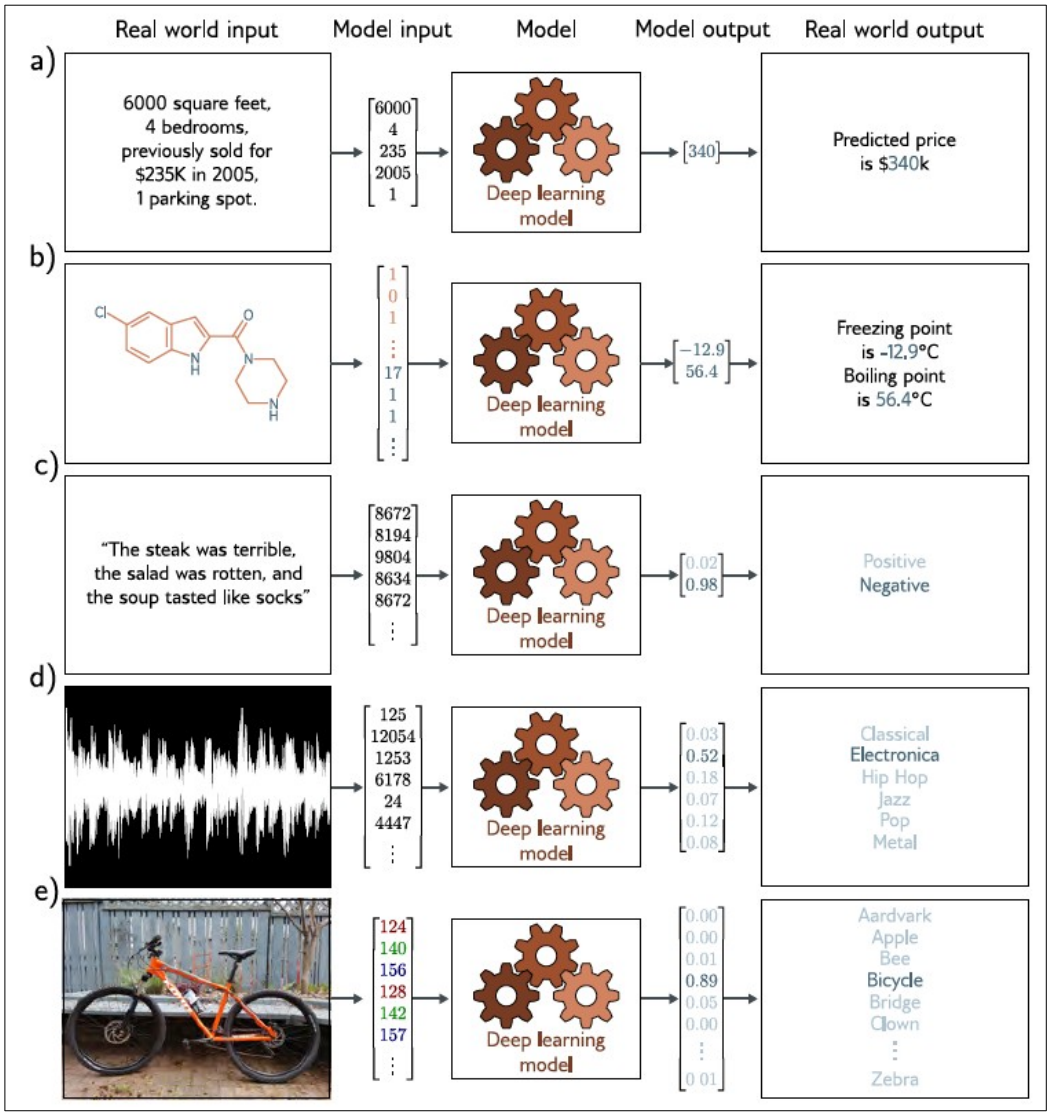


***Figura 12.** a) Predice el precio de una casa: regresión. b) A partir de la estructura química de una molécula predice la temperatura de congelación y ebullición: regresión. c) Clasificar un texto como una opinión positiva o negativa (clasificación binaria). d) Clasifica un fichero de música en una de N categorías musicales. e) Clasificar una imagen en N posibles objetos.*

Algunos algoritmos de regresión también pueden utilizarse para clasificación y viceversa. Por ejemplo, se usa frecuentemente la **Regresión Logística** para clasificar, su valor de salida es numérico y representa la probabilidad de que un ejemplo pertenezca a una determinada clase (0.2 significa que hay un 20% de que un mail sea spam).

Algunos de los algoritmos de aprendizaje supervisado que comentaremos durante el curso:

- k-Nearest Neighbors (K-vecinos cercanos)
- Regresión Lineal.
- Regresión Logística.
- Máquinas de Vectores de Soporte (SVM[8])
- Árboles de decisión.
- Random Forest.
- Redes neuronales.

La figura 12 muestra muchos problemas de regresión y clasificación usados para entrenar un modelo de deep learning (una red neuronal). En cada caso hay un entrada del mundo real (una frase de texto, un fichero de sonido, una imagen, etc.) que se codifica en un vector de números. El vector sería la entrada del modelo. El modelo genera otro vector numérico que en cada caso tendrá un significado distinto pero que es la predicción realizada por la red neuronal.

APRENDIZAJE NO SUPERVISADO

En este tipo de aprendizaje se usan datos que no están etiquetados con la solución. Estos sistemas intentan aprender sin una información que les guía.

El que no existan ejemplos de entrada etiquetados ni se generen nuevas etiquetas en datos de salida significa que no puede haber supervisión humana (comprobar si generan las etiquetas adecuadas) porque no existen tales etiquetas o soluciones al problema.

En vez de aprender a mapear una salida a partir de los datos de entrada, el objetivo de los algoritmos de aprendizaje no supervisado es estudiar la estructura de los datos. Como en el caso de los algoritmos de aprendizaje supervisado, los datos de entrada pueden tener diferentes características: pueden ser discretos o continuos, de baja o de alta dimensionalidad, con estructura o sin estructura, etc.

Los algoritmos no supervisados más importantes son:

- Hacer Clusters (agrupar):
  - K-Means
  - DBSCAN
  - Hierarchical Cluster Analysis (HCA)
- Detección de anomalías y novedades:
  - One-class SVM
  - Isolation Forest
- Visualización y reducción de dimensiones:
  - Principal Component Analysis (PCA).
  - Kernel PCA.
  - Locally-Linear Embedding (LLE)
  - T-distributed Stochastic Neighbor Embedding (t-SNE)
- Reglas de aprendizaje asociativas:
  - Apriori
  - Eclat

Por ejemplo imagina que tienes datos sobre los visitantes de un sitio web. Quieres ejecutar un algoritmo que agrupe a clientes similares. Esto te permite descubrir que el 40% son mujeres a quienes les gustan los conciertos y visitan la web por las tardes, mientras que el 20% son jóvenes

8 **SVM**: Support Vector Machine.

que la visitan el fin de semana, etc. Te ayuda a pensar en ofertas, contenidos, etc. orientado para cada grupo.

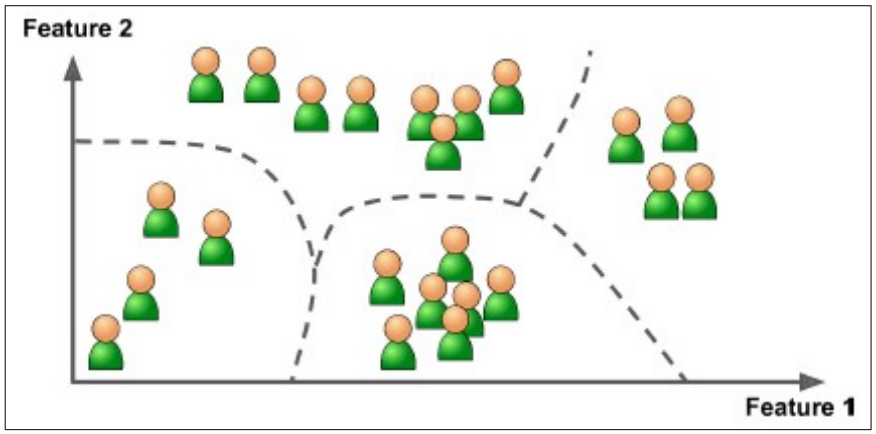


Figura 13. *Algoritmos no supervisados que agrupan los datos (clustering).*

Los algoritmos de visualización y reducción de dimensiones son también otro ejemplo de aprendizaje no supervisado. Cuando tienes datos complejos y no están etiquetados pueden representarse mediante gráficas 2D o 3D fácilmente, pero si tienen más dimensiones, es complicado hacerlo. Estos algoritmos intentan mantener la estructura de los datos (por ejemplo separar datos en diferentes clusters y representar gráficos de las cosas que más varían omitiendo las que tienen más en común).

Otro uso de la reducción de dimensiones es simplificar los datos sin perder excesiva información. Siempre es mejor trabajar con datos simples porque los algoritmos de aprendizaje trabajan mejor y de manera más eficiente. Para conseguirlo eliminan características que no aportan demasiada información o combinan varias características en solamente una para reducir la cantidad de dimensiones de los datos. Estas técnicas se llaman **extracción de características**.

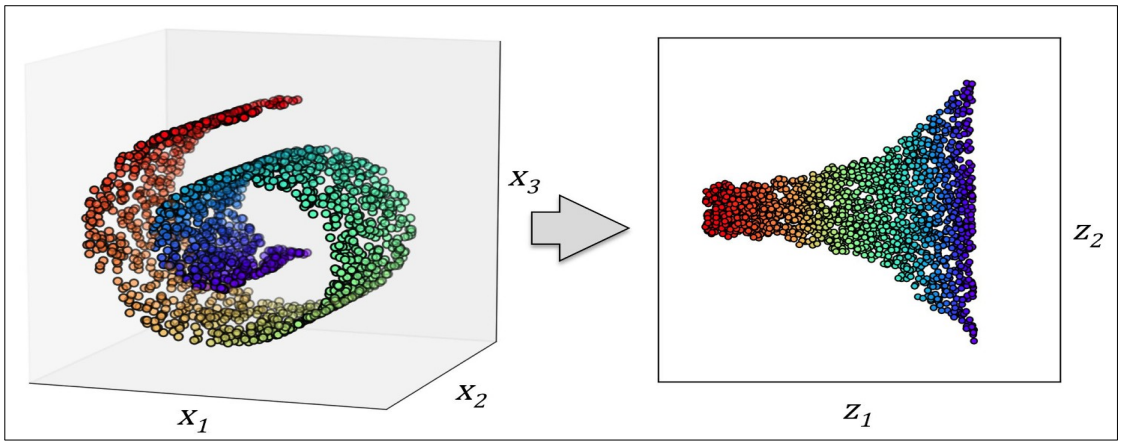


Figura 14. *Algoritmos no supervisados que reducen las dimensiones de los datos.*

Otra importante tarea de los algoritmos no supervisados es la detección de anomalías (**outliers**). Por ejemplo detectar transacciones inusuales en el uso de tarjetas de crédito para prevenir fraudes, detectar apropiaciones indebidas de productos, o eliminar los outliers de datasets que van a utilizar otros algoritmos de aprendizaje a los que les afecte mucho la presencia de estos datos anómalos.

Otra tarea muy similar a la anterior es la detección de novedades. La diferencia entre los algoritmos que detectan anomalías y los que detectan novedades es que estos últimos son más tolerantes a la hora de identificar un dato como novedoso.

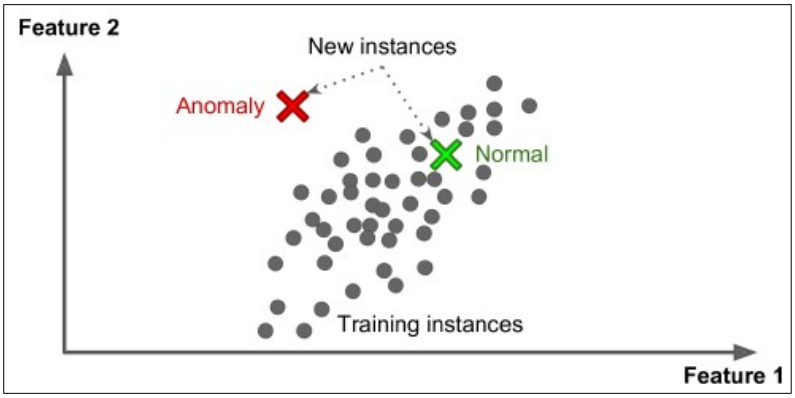


Figura 15. *Algoritmos no supervisados que detectan anomalías.*

Finalmente están los algoritmos de aprendizaje de reglas de asociación, que tienen como objetivo explorar grandes cantidades de datos para encontrar relaciones entre las diferentes características de los ejemplos. Por ejemplo si uno de estos algoritmos explora las ventas de un supermercado, podría descubrir que los clientes que compran salsa barbacoa y patatas también compran carne. Así que igual es interesante colocar estos productos cerca entre sí.

Los modelos ***GAN***[9] también podríamos considerarlos como algoritmos no supervisados al no usar etiquetas en los datos de entrada, pero los explicaremos en su propio apartado.

APRENDIZAJE SEMI-SUPERVISADO (SSL)

Algoritmos que entrenan con algunos datos etiquetados y un gran número de datos no etiquetados.

Etiquetar correctamente los datos es cada vez más laborioso para tareas complejas. Por ejemplo, para entrenar un modelo de clasificación de imágenes para que diferencie entre automóviles y motocicletas, deben etiquetarse cientos (si no miles) de imágenes de entrenamiento como "automóvil" o "motocicleta"; para una tarea más detallada, como la detección de objetos, los humanos no solo deben anotar el objeto u objetos que contiene cada imagen, sino también dónde se encuentra cada objeto (en qué zona de la imagen); para tareas aún más detalladas, como la segmentación de imágenes, las etiquetas de datos deben anotar los *límites específicos píxel por píxel* de diferentes segmentos de imagen para cada imagen.

Por lo tanto, etiquetar los datos puede resultar particularmente tedioso para ciertos casos prácticos. En casos prácticos más especializados, como el descubrimiento de fármacos, la secuenciación genética o la clasificación de proteínas, la anotación de datos no solo lleva mucho tiempo y es tediosa sino que también requiere una experiencia de dominio muy específica, es decir, cualquier persona sin importantes conocimientos de química no puede etiquetar una proteina.

El SSL ofrece una forma de extraer el máximo beneficio de una escasa cantidad de datos etiquetados a la vez que se aprovechan datos sin etiquetar relativamente abundantes.

El SSL se basa en ciertas suposiciones sobre los datos no etiquetados utilizados para entrenar el modelo y la forma en que los datos de diferentes clases se relacionan entre sí.

---

9 ***GAN***: *redes generativas antagónicas, un tipo de red neuronal profunda que se utiliza para generar imágenes o texto sintético. Su arquitectura consta de dos redes neuronales profundas, una generativa y otra discriminativa, que compiten mutuamente (de ahí lo de "antagónicas").*

Una condición necesaria del SSL es que los ejemplos no etiquetados usados en el entrenamiento del modelo deben ser relevantes para la tarea para la que se entrena el modelo. En términos más formales, SSL requiere que la distribución **p(x)** de los datos de entrada contenga información sobre la distribución posterior **p(y|x)**, es decir, la probabilidad condicional de que un dato **(x)** pertenezca a una clase determinada **(y)**. Así, por ejemplo, si se utilizan datos no etiquetados para entrenar un clasificador de imágenes para que distinga entre fotos de gatos y fotos de perros, el conjunto de datos de entrenamiento debe contener imágenes tanto de gatos como de perros y las imágenes de caballos y motocicletas no serán útiles.

Si bien un estudio de 2018 sobre algoritmos SSL descubrió que "aumentar la cantidad de datos sin etiquetar tiende a mejorar el resultado", también encontró que "agregar datos sin etiquetar de un conjunto de clases que no coinciden puede *dañar* el funcionamiento en comparación con no utilizar ningún dato sin etiquetar en absoluto".

La condición básica de que **p(x)** tenga una relación significativa con **p(x|y)** da lugar a múltiples **suposiciones** sobre la naturaleza de esa relación. Estos supuestos son la fuerza motriz de la mayoría de los métodos SSL: en términos generales, cualquier algoritmo SSL se basa en el cumplimiento explícito o implícito de uno o varios de los siguientes supuestos.

- **Suposición de clúster:** establece que los datos que pertenecen al mismo *clúster (grupo* de datos similares entre sí) también pertenecerán a la misma clase. El cálculo de los clústeres de datos depende de la noción de similitud que se utiliza: la suposición de suavidad, la suposición de baja densidad y la suposición de las variedades utilizan una definición diferente de lo que es un dato "similar":
  - **Suposición de suavidad**: establece que si dos datos, **x** y **x'** están cerca el uno del otro en el espacio de entrada (el conjunto de todos los valores posibles para **x**, entonces sus etiquetas **y** e **y'** deberían ser las mismas. Esta suposición, también es conocida como *suposición de continuidad*, es común a la mayoría del aprendizaje supervisado: por ejemplo, los clasificadores aprenden una aproximación significativa (o "representación") de cada clase relevante durante el entrenamiento; una vez entrenados, determinan la clasificación de los nuevos puntos de datos a través de la representación a la que más se asemejen. En el contexto de SSL, la suposición de suavidad tiene la ventaja añadida de aplicarse *transitivamente* a los datos no etiquetados. Considere que tenemos 3 datos: un dato etiquetado x_1, un dato sin etiquetar x_2 cercano a x_1 y otro dato sin etiquetar x_3 que está cerca de x_2 pero no cerca de x_1. La suposición de suavidad nos dice que x_2 debe tener la misma etiqueta que x_1. También nos dice que x_3 debe tener la misma etiqueta que x_2. Por lo tanto, podemos suponer que los 3 datos tienen la misma etiqueta.
  - **Hipótesis de baja densidad**: indica que el límite de decisión entre clases no debe pasar por regiones de alta densidad. Dicho de otra manera, el límite de decisión debe estar en un área que contenga pocos puntos de datos. Por lo tanto, podría considerarse como una extensión de la suposición de clúster (en el sentido de que un clúster de alta densidad de datos representa una clase, en lugar del límite entre clases) y la suposición de suavidad (en el sentido de que si varios puntos de datos están cerca unos de otros, deben compartir una etiqueta y por lo tanto, se encuentran en el mismo lado del límite de decisión). Este diagrama ilustra cómo las suposiciones de suavidad y baja densidad pueden informar un límite de decisión mucho más intuitivo de lo que sería posible con métodos supervisados que solo pueden considerar los (escasos) puntos de datos etiquetados (los oscuros en la figura 16).

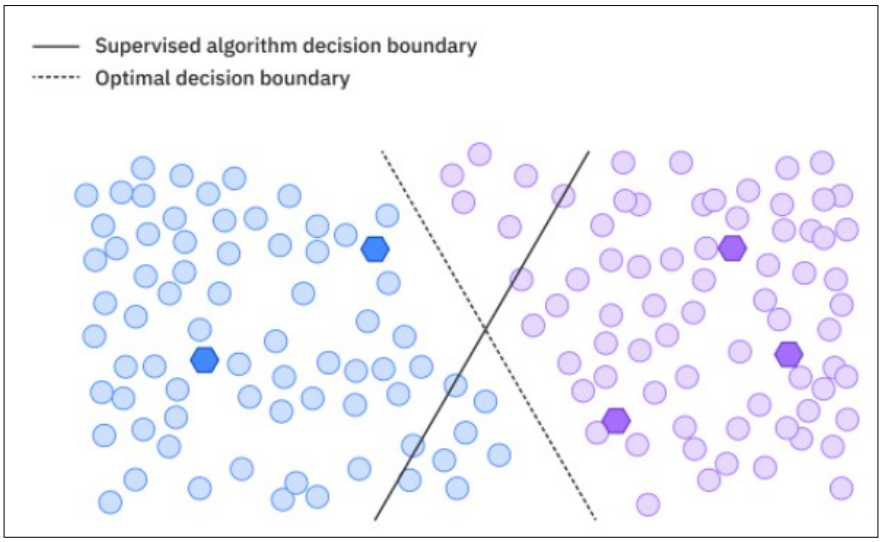


Figura 16. *Clasificación con algoritmos SSL con pocos datos etiquetados (hexágonos más oscuros) y muchos no etiquetados (los círculos de color claro).*

- **Suposición de las variedades**: indica que el espacio de entrada de mayor dimensión comprende varias variedades de dimensiones inferiores en las que se encuentran todos los datos y que los datos de la misma variedad comparten la misma etiqueta. Para un ejemplo intuitivo, considere un trozo de papel arrugado en forma de bola. La ubicación de cualquier punto en la superficie esférica solo se puede representar con coordenadas tridimensionales *x, y, z*. Pero si esa bola arrugada ahora se aplana nuevamente en una hoja de papel, esos mismos puntos ahora se pueden representar con coordenadas bidimensionales *x, y*. Esto se denomina *reducción de dimensionalidad* y se puede lograr matemáticamente utilizando métodos como autocodificadores o convoluciones. En ML las dimensiones no corresponden a las dimensiones *físicas* conocidas, sino a cada atributo de datos. Por ejemplo, en ML una pequeña imagen RGB que mida 32x32 píxeles tiene *3.072* dimensiones: 1.024 píxeles, cada uno de los cuales posee 3 valores (rojo, verde y azul). La comparación de puntos de datos con tantas dimensiones es difícil, tanto por la complejidad y los recursos computacionales necesarios como porque la mayor parte de ese espacio de altas dimensiones no contiene información significativa para la tarea en cuestión. La suposición de las variedades sostiene que cuando un modelo aprende la función de reducción de dimensionalidad adecuada para descartar la información irrelevante, los puntos de datos dispares convergen en una representación más significativa para la que las demás suposiciones SSL son más confiables.

¿Cómo se trabaja con los datos? Bueno hay varias aproximaciones y formas de hacerlo. Serían subtipos de SSL.

- **SSL TRANSDUCTIVOS:** utilizan etiquetas disponibles para predecir las de datos no etiquetados, de modo que puedan ser utilizados por un aprendiz base supervisado.
- **SSL INDUCTIVOS**: entrenan un clasificador que pueda modelar todo el espacio de entrada (etiquetado y no etiquetado), los transductivos solo pretenden obtener predicciones de etiquetas para los datos no etiquetados.
- **SSL CON PROPAGACIÓN DE ETIQUETAS:** La propagación de etiquetas es un algoritmo basado en grafos que calcula las asignaciones de etiquetas para datos sin

etiquetar en función de su proximidad relativa a los datos etiquetados, utilizando la suposición de suavidad y la suposición de clúster. La idea del algoritmo es que se puede trazar un grafo completamente conectado en el que los nodos son todos los datos disponibles, tanto etiquetados como sin etiquetar. Cuanto más cerca estén dos nodos según alguna medida de distancia, más peso tendrá el borde entre ellos en el algoritmo. Partiendo de los datos etiquetados, las etiquetas se *propagan* de forma iterativa a través de los datos vecinos no etiquetados, utilizando las suposiciones de suavidad y de clúster.

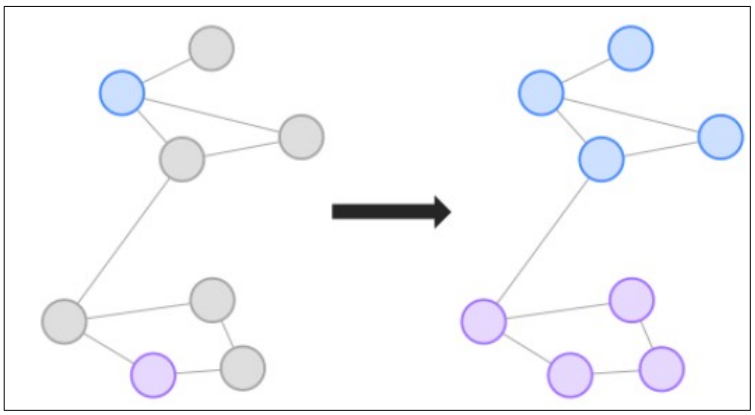


Figura 17. *A la izquierda datos originales etiquetados y sin etiquetar. A la derecha propagación de etiquetas.*

- **SSL CON APRENDIZAJE ACTIVO:** no automatizan el etiquetado de datos, en su lugar, se utilizan en SSL para determinar qué muestras sin etiquetar proporcionarían la información más útil si se etiquetaran manualmente. El uso del aprendizaje activo en SSL ha logrado resultados prometedores: por ejemplo, un estudio reciente halló que redujo a menos de la mitad la cantidad de datos etiquetados necesarios para entrenar eficazmente un modelo para segmentación semántica.
- **MÉTODOS DE ENVOLTURA:** Una forma relativamente sencilla de ampliar los algoritmos supervisados existentes a un entorno SSL consiste en entrenar primero el modelo con los datos etiquetados disponibles (o simplemente usar un clasificador preexistente adecuado) y a continuación, generar predicciones de *pseudoetiquetas* para datos no etiquetados. Posteriormente, el modelo se puede volver a entrenar utilizando los datos etiquetados originalmente y los datos pseudoetiquetados, sin diferenciar entre ambos. El principal beneficio de los métodos de envoltura, más allá de su simplicidad, es que son compatibles con casi cualquier tipo de aprendiz base supervisado. La mayoría de los métodos de envoltura introducen algunas técnicas de regularización para reducir el riesgo de reforzar predicciones potencialmente inexactas de pseudoetiquetas.
  - **AUTOENTRENAMIENTO:** un método de envoltura básico. Requiere predicciones de pseudoetiquetas *probabilísticas*, en lugar de deterministas: por ejemplo, un modelo que genere “85 por ciento perro, 15 por ciento gato” en lugar de simplemente generar “perro”. Las predicciones de pseudoetiquetas probabilísticas permiten que los algoritmos de autoentrenamiento acepten solo predicciones que superen un umbral de confianza determinado, en un proceso similar a la minimización de la entropía. Este proceso se puede realizar de forma iterativa, ya sea para optimizar el proceso de pseudoclasificación o para alcanzar un cierto número de muestras pseudoetiquetadas.
  - **COENTRENAMIENTO:** amplían el concepto de autoentrenamiento al entrenar a *varios* aprendices base supervisados para que asignen pseudoetiquetas. Con la diversificación

se pretende reducir la tendencia a reforzar las malas predicciones iniciales. Por lo tanto, es importante que las predicciones de cada aprendiz base no estén fuertemente correlacionadas entre sí. Un enfoque típico es utilizar diferentes algoritmos para cada clasificador. Otro enfoque es que cada clasificador se centre en un subconjunto diferente de datos: por ejemplo, en datos de vídeo, entrenar a un aprendiz base en datos visuales y a otro, en datos de audio.

- **SSL CON PREPROCESAMIENTO NO SUPERVISADO:** A diferencia de los métodos de envoltura que utilizan datos etiquetados y no etiquetados simultáneamente, algunos métodos SSL emplean datos etiquetados y no etiquetados en etapas separadas: una etapa de preprocesamiento no supervisada, seguida de otra supervisada. Al igual que los métodos de envoltura, estas técnicas se pueden utilizar esencialmente para cualquier aprendiz base supervisado. Pero, a diferencia de los métodos envoltura, el modelo supervisado "principal" en última instancia se entrena solo en puntos de datos etiquetados originalmente (anotados por humanos). Estas técnicas de preprocesamiento van desde extraer características útiles de datos no etiquetados hasta preagrupar datos sin etiquetar y utilizar el "entrenamiento previo" para dar parámetros iniciales de un modelo supervisado.
  - **AGRUPAR Y LUEGO ETIQUETAR:** técnica SSL directa que implica agrupar todos los datos (etiquetados y no etiquetados) mediante un algoritmo no supervisado. Aprovechando la suposición de cluster, estos clústeres se pueden utilizar para ayudar a entrenar un modelo de clasificador independiente o, si los datos etiquetados en un clúster dado son todos de la misma clase, pseudoetiquetar los datos no etiquetados y proceder de forma similar a los métodos de envoltura. Los métodos simples (como el *k-nearest neighbors*) pueden dar lugar a predicciones inadecuadas. Los algoritmos de clúster más refinados, como ***DBSCAN*** (basado en la suposición de baja densidad) han logrado una mayor fiabilidad.
  - **PREENTRENAMIENTO Y EXTRACCIÓN DE CARACTERÍSTICAS:** El preentrenamiento no supervisado (o autosupervisado) permite a los modelos aprender representaciones útiles del espacio de entrada y reduce la cantidad de datos etiquetados necesarios para afinar un modelo supervisado. Un método habitual es emplear una red neuronal, a menudo un autocodificador para aprender una representación característica de los datos de entrada y, luego, usar estas características aprendidas para entrenar a un aprendiz base supervisado. Esto, frecuentemente, implica una reducción de la dimensionalidad, lo que ayuda a hacer uso de la suposición de las variedades.
- **MÉTODOS SEMISUPERVISADOS INTRÍNSECAMENTE:** introducen directamente los datos sin etiquetar en la función objetivo del aprendiz base, en lugar de procesar los datos sin etiquetar en un paso independiente de pseudoetiquetado o preprocesamiento.
  - **SVM SEMISUPERVISADAS:** si los datos de diferentes categorías no se pueden separar linealmente (ninguna línea recta puede definir con precisión y nitidez el límite entre categorías), los algoritmos SVM asignan los datos a un espacio de características de dimensiones superiores en el que las categorías pueden separarse mediante un hiperplano. Al determinar este límite de decisión, los algoritmos SVM maximizan el *margen* entre el límite de decisión y los datos más cercanos a él. Esto, en la práctica, se aplica en la *suposición de baja densidad.* En un entorno supervisado, un término de regularización penaliza al algoritmo cuando los datos etiquetados se encuentran en el lado equivocado del límite de decisión. En las SVM semisupervisadas (S3VM), esto no

es posible para los datos no etiquetados (cuya clasificación se desconoce); por lo tanto, las S3VM también penalizan los datos que se encuentran dentro del margen prescrito.

- **DEEP LEARNING INTRÍNSECAMENTE SEMISUPERVISADO:** adaptar redes neuronales para SSL. Esto se logra añadiendo o modificando los términos de pérdida que normalmente se utilizan en estas arquitecturas, lo que permite la incorporación de datos sin etiquetar en el entrenamiento. Las arquitecturas de deep learning para SSL incluyen redes escalonadas, pseudoconjuntos, conjuntos temporales y algunas modificaciones en redes adversariales generativas (**GAN**S).

Algunos servicios de hosting de fotos como Google Photos son buenos ejemplos de este tipo de métodos. Cuando subes tus fotos familiares al servicio, automáticamente reconoce que la misma persona A ha subido las fotos 1, 5 y 11 mientras que otra persona B sube sus fotos 2, 5 y 7. Esta es la parte no supervisada del algoritmo (clustering). Ahora lo que debe hacer el sistema es reconocer a las personas de cada foto. Solamente una etiqueta por persona.

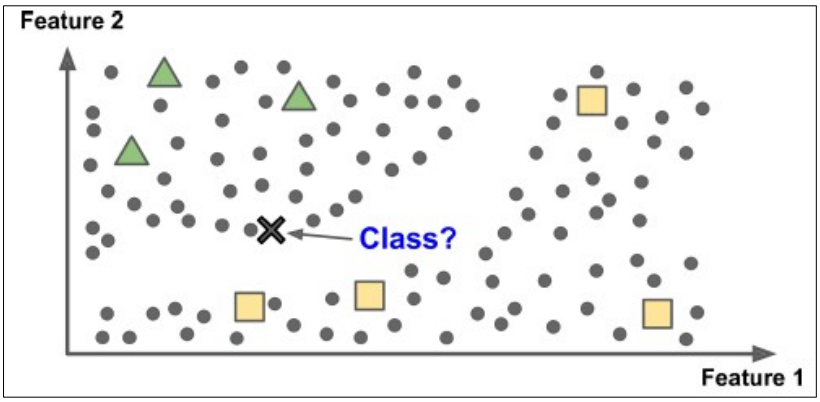


Figura 18. *Las figuras (triángulos y cuadrados) son datos etiquetados, el resto no etiquetados.*

En resumen, la mayoría de algoritmos SSL son combinaciones de no supervisados y supervisados. Por ejemplo ***Deep Belief Networks*** (DBNs) se basan en componentes no supervisados llamados ***Restricted Boltzmann Machines*** (RBMs) apiladas unas sobre otras. Las RBMs se entrenan secuencialmente de manera no supervisada y el sistema es afinado con técnicas supervisadas.

APRENDIZAJE REFORZADO

Estos algoritmos son diferentes a los anteriores. El sistema de aprendizaje se llama agente en este contexto y tiene la capacidad de observar el entorno en el que trabaja, seleccionar y realizar acciones y obtener información de vuelta del estado de su entorno (premios o penalizaciones según sea un estado positivo o negativo). Su objetivo es aprender cuál es la mejor estrategia para alcanzar el objetivo para el que se crea. A la estrategia se la llama política y será la que obtenga más premios a lo largo del tiempo. Una política define qué acción debe escoger el agente cuando se encuentra en determinada situación.

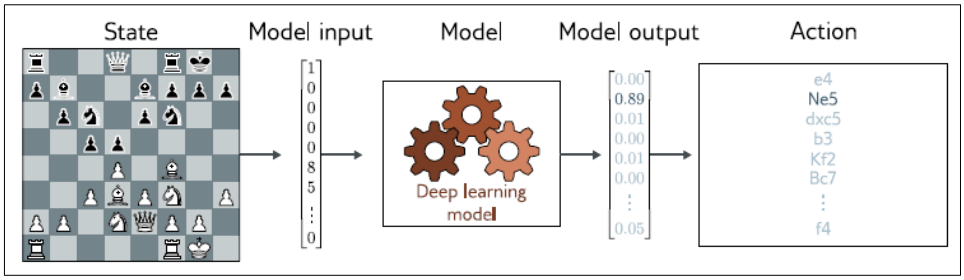


Figura 19. *Los algoritmos de aprendizaje reforzado mapean estados (posiciones del tablero en la figura) a posibles acciones (movimientos de piezas). Este mapeo se conoce como política.*

Algunos robots implementan algoritmos de aprendizaje reforzado para aprender como actuar. Los programas DeepMind y AlphaGo son también ejemplos de este tipo de sistemas. AlphaGo derrotó en mayo de 2017 a Ke Jie el campeón mundial del juego Go.

Aprendió analizando millones de partidas y jugó muchas partidas contra él mismo. El aprendizaje se produjo antes de ser usado, en el campeonato solamente aplicó la política que había aprendido con anterioridad.

El aprendizaje reforzado es un método de aprendizaje orientado a objetivos que se basan en la interacción con el entorno. El objetivo es que el agente actúa para maximizar la cantidad de premios que recibe.

Los agentes son como un cachorro de perro al que se intenta educar. No puedes decirle al perro lo que debe hacer porque no lo entiende. Sin embargo, puedes premiarlo cuando realice acciones positivas y penalizarlo cuando no. En cada acción, recordará si obtuvo un premio o una penalización. Este enfoque se denomina problemas de asignación de créditos.

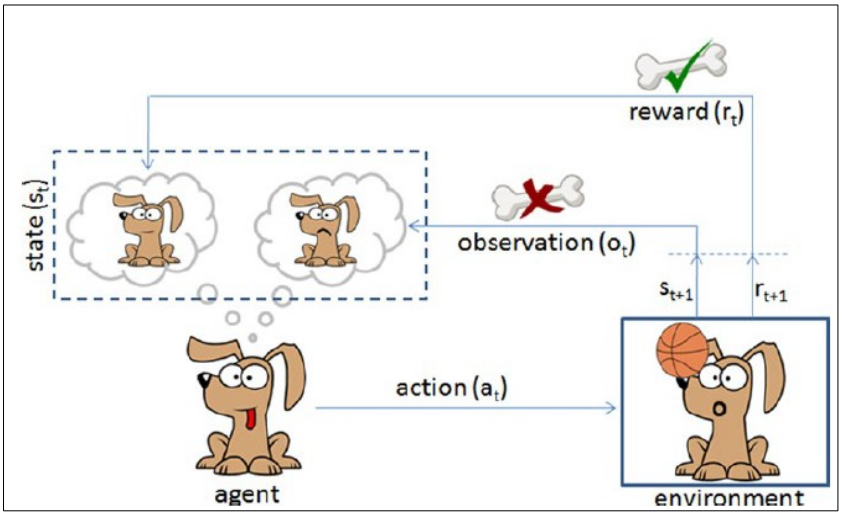


Figura 20. *Los algoritmos de aprendizaje reforzado utilizan el enfoque castigo/recompensa.*

De manera similar, podemos entrenar a los agentes de ordenador cuyo objetivo es realizar una acción para mover su entorno del estado $\mathbf{e_t}$ al estado $\mathbf{e_{t+1}}$ y encontrar una función comportamiento que maximice la suma de premios y mapee los estados a acciones.

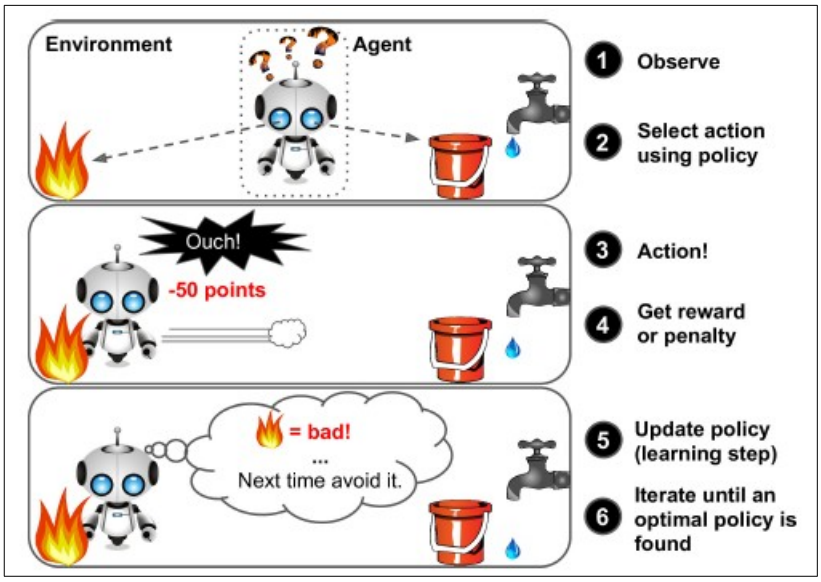


Figura 21. *El agente escoge la acción que maximiza la recompensa.*

Realiza el ejercicio 6 y el ejercicio 7.

Un ejemplo de estos algoritmos es la regla del ***Q-learning*** publicada en 2013 por ***Deepmind Technologies***. Este algoritmo actualiza el estado mediante la ecuación:

```
nuevo_Q[e,a] = previo_Q[e,a] + α * (r + y * max(e,a) – previo_Q[e,a])
```

Donde:

- `α` es el ratio de aprendizaje
- `r` es el premio obtenido en la última acción.
- `y` es el factor de descuento
- ***`max(e,a)`*** es la estimación del valor de la nueva acción

Si el valor óptimo de **`Q[e, a]`** de la secuencia `s'` en el siguiente paso era conocido por todas las acciones posibles `a'` la estrategia óptima es seleccionar la acción `a'` que maximice este valor esperado de **`r + y * max(e,a) – previo_Q[e,a]`**

EJEMPLO 1: Considera el caso donde un agente intenta salir fuera de un laberinto. Puede moverse aleatoriamente a las habitaciones a las que hemos dado nombre con letras (a, b, c, d y e) y las salidas tienen la letra f y cuando se mueva obtendrá una recompensa o no. La manera más común de formalizar un problema reforzado es presentarlo como un proceso de decisión de Markov. Asumiendo que el agente está en b (una de las habitaciones) y su objetivo es alcanzar f. Así que un paso del agente es moverse desde b hasta f, lo que le proporciona una recompensa de 100 (en otro caso 0). El grafo de la derecha muestra los estados como nodos y las acciones como las líneas:

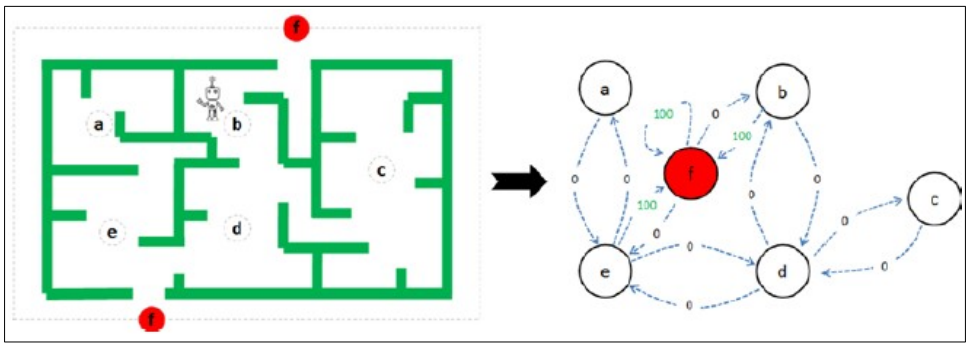


Figura 22. *Laberinto y grafo que lo representa.*

```python
# coding: utf8
# ***************************
# **  EJEMPLO DE Q-LEARNING **
# ***************************
from random import randint
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.collections import LineCollection

# define el grafo del laberinto mediante una matriz de conexión
# -1 significa que no hay enlace entre un lugar y otro por ejemplo
# el estado a no puede ir al b directamente
R = np.array([[-1, -1, -1, -1,  0,  -1],
              [-1, -1, -1,  0, -1, 100],
              [-1, -1, -1,  0, -1,  -1],
              [-1,  0,  0, -1,  0,  -1],
              [ 0, -1, -1,  0, -1, 100],
              [-1,  0, -1, -1,  0, 100]]).astype("float32")

Q = np.zeros_like(R)           # El conocimiento
gamma = 0.8                    # ratio de aprendizaje
estado_inicial = randint(0, 4) # Inicializar aleatoriamente el estado
```

```
24  def acciones_posibles(estado):   # Acciones posibles en este estado
25      fila_estado_actual = R[estado,]
26      posibilidades = np.where(fila_estado_actual >= 0)[0]
27      return posibilidades
28
```

```
29  def escoge_sig_paso(movimientos_disponibles):  # Escoge aleatoriamente una de las disponibles
30      siguiente = np.random.choice(movimientos_disponibles, 1)[0]
31      return siguiente
```

```
33  def actualiza(estado_actual, accion, gamma):    # actualiza la Q-matriz según la ruta seleccionada
34      max_idx = np.where(Q[accion,] == np.max(Q[accion,]))[0]
35      if max_idx.shape[0] > 1:
36          max_idx = np.random.choice(max_idx, size = 1)[0]
37      else:
38          max_idx = max_idx[0]
39      max_valor = Q[accion, max_idx]
40      Q[estado_actual, accion] = R[estado_actual, accion] + gamma * max_valor # fórmula del Q learning
```

```
42  pasos_disponibles = acciones_posibles(estado_inicial) # Ver acciones posibles
43  accion = escoge_sig_paso(pasos_disponibles)           # escoger una
44  for i in range(100):   # Entrenar 100 iteraciones
45      estado_actual = np.random.randint(0, int(Q.shape[0]))
46      pasos_disponibles = acciones_posibles(estado_actual)
47      accion = escoge_sig_paso(pasos_disponibles)
48      actualiza(estado_actual, accion, gamma)
49  print("Q matriz entrenada: \n", Q / np.max(Q) * 100)  # Normalizar la Q matriz entrenada
50  # Testing
51  estado_actual = 2
52  pasos = [estado_actual]
53  while estado_actual != 5:
54      idx_sig_paso = np.where(Q[estado_actual,] == np.max(Q[estado_actual,]))[0]
55      if idx_sig_paso.shape[0] > 1:
56          idx_sig_paso = np.random.choice(idx_sig_paso, size = 1)[0]
57      else:
58          idx_sig_paso = idx_sig_paso[0]
59      pasos.append(idx_sig_paso)
60      estado_actual = idx_sig_paso
61  print("Mejor secuencia de ruta: ", [int(x) for x in pasos])
```

3.3. CUANDO SE APRENDE: OFFLINE O BATCH / ON-LINE.

Este nuevo criterio de clasificar sistemas de ML se centra en si el sistema es capaz de aprender incrementalmente a partir de un stream de datos de ejemplos. Es decir, si el sistema tiene una fase de aprendizaje que cuando finaliza ya no tiene capacidad de continuar aprendiendo o si el sistema aprende continuamente.

APRENDIZAJE POR LOTES O FUERA DE LÍNEA (BATCH / OFFLINE)

En el *batch learning*, el sistema no tiene la capacidad de aprender incrementalmente: primero debe ser entrenado usando los datos disponibles. Como esta tarea consume una cantidad considerable de tiempo y recursos (RAM, CPU y HD) debe realizarse off-line. En primer lugar el sistema es entrenado y posteriormente se despliega en producción sin que aprenda más, simplemente utiliza lo que ya ha aprendido anteriormente.

Si hay que actualizar lo que sabe porque hay cambios significativos en los datos, es necesario entrenar una nueva versión del sistema desde cero con todos los datos disponibles incluyendo los nuevos. Una vez generado el nuevo modelo, sustituirá al anterior.

Por suerte, el proceso completo de entrenar, evaluar y desplegar un sistema de ML puede automatizarse fácilmente incluso hasta para sistemas batch. La solución de actualizar los datos y entrenar una nueva versión del sistema desde cero con frecuencia es necesario. Si el entrenamiento tarda horas en completarse, podría realizarse cada 24 horas o semanalmente. Si el sistema debe tener mayor capacidad de adaptación, es necesario pensar una solución más reactiva.

Además, utilizar una gran cantidad de datos implica disponer de una gran cantidad de recursos (CPU, RAM, disco, I/O de disco, etc.). Si actualizas diariamente el sistema, acaba por salir caro el tenerlo actualizado. Por eso, si la cantidad de datos crece a un ritmo alto quizás habría que pensar en dejar de utilizar algoritmos de aprendizaje de tipo batch.

Por último, si el sistema necesita poder aprender de manera autónoma y tiene recursos de cómputo limitados (un smartphone) la única solución es usar algoritmos incrementales.

APRENDIZAJE ONLINE

Estos sistemas tienen la capacidad de aprender incrementalmente recibiendo feedback de los ejemplos de datos con los que van trabajando, ya sean ejemplos individuales o pequeños grupos de ejemplos llamados **mini-batchs**. Aunque puede haber una fase de entrenamiento inicial, posteriormente cada paso de aprendizaje va usando unos pocos datos que van llegando secuencialmente y por tanto es un proceso rápido y con poca necesidad de recursos.

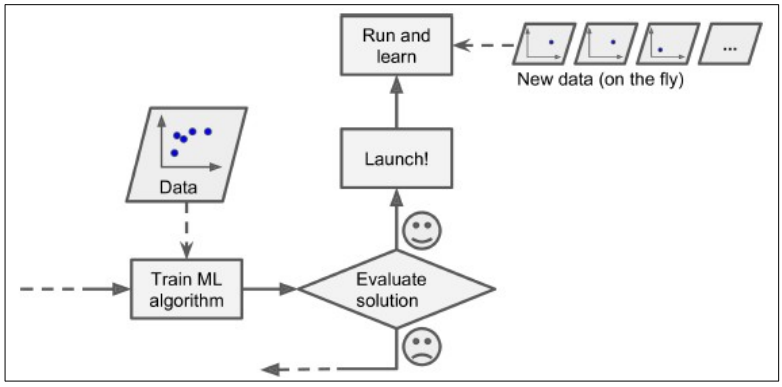


Figura 22. *Un sistema online aprende en el entrenamiento y/o posteriormente.*

Los sistemas de aprendizaje online son ideales cuando reciben datos en forma de flujos continuos (por ejemplo precios de artículos) y necesitan adaptarse rápidamente o de manera autónoma a esos cambios. También son una buena opción si tienes recursos de cómputo limitados: como utilizan datos al vuelo, no tienen necesidad de manejar una gran cantidad de datos, salvo que necesites los datos para otras cosas no tienes necesidad de almacenarlos ni gestionarlos.

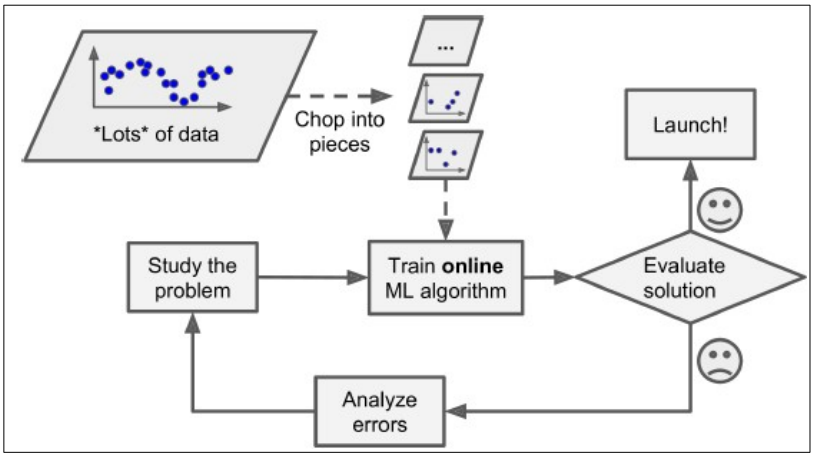


Figura 23. *Un sistema online facilita el aprendizaje con enormes cantidades de datos.*

Estos algoritmos también se utilizan para entrenar sistemas que deben procesar enormes datasets de datos que no pueden contenerse en la RAM de un mismo sistema (aprendizaje ***out-of-core***). Los algoritmos particionan los datos y repiten el proceso hasta que todas las partes se han procesado.

Aunque el aprendizaje out-of-core se realiza normalmente offline, entraría dentro de la categoría de aprendizaje *online,* así que piensa en aprendizaje incremental más que en online/offline porque estos últimos términos son más confusos.

Un gran inconveniente de los sistemas online es que los datos malos o erróneos van engañando al sistema que va degradándose. Por ejemplo si el sensor de un robot deja de funcionar correctamente y envía datos erróneos, etc. Para reducir este riesgo el sistema debe monitorizarse y cambiar a offline si se detecta un mal funcionamiento, o bien en vez de monitorizar el sistema, monitorizar la entrada de datos para descartar o ajustar posibles errores.

3.4 MODELOS GENERATIVOS: GAN.

Intentan sintetizar nuevos ejemplos de datos que sean estadísticamente indistinguibles de los datos de entrenamiento. Algunos describen explícitamente la distribución de probabilidad de los datos de entrada y se generan los nuevos ejemplos a partir del muestreo de esta distribución. Otros aprenden un mecanismo para generar nuevos ejemplos sin describir de manera explícita su distribución.

Obviamente no son sistemas de aprendizaje supervisado porque se inventan un nuevo ejemplo, no predicen algo sobre uno existente que luego puedas comprobar. Se ha tenido éxito en generar imágenes o textos creíbles pero distintos a los usados como entradas.

También pueden aplicarse ciertas restricciones (generación condicional). Los modelos generativos de texto actuales son tan potentes que pueden parecer inteligentes. Dado un cuerpo de texto seguido por una pregunta, el modelo puede completar la respuesta que falta para completar el documento. Sin embargo, el modelo en realidad sabe la estadística del lenguaje pero no comprende nada del significado de las respuestas.

Figura 24. *Imágenes generadas (derecha) a partir de ejemplos (izquierda).*

Variables Latentes

Algunos (pero no todos) los modelos generativos se aprovechan de que los datos de salida tienen menos dimensionalidad de la que sugieren las variables observadas. Por ejemplo, el número de frases correctas del inglés es menor que las posibles frases que se pueden crear uniendo palabras al azar. De manera similar, las imágenes que tienen sentido son un subconjunto menor que todas las combinaciones aleatorias posibles de valores RGB de sus píxels.

Esto nos permite la idea de que podemos describir cada ejemplo de datos usando un número pequeño de variable latentes ocultas. Aquí el papel del deep learning es descubrir un mapeo entre estas variables latentes y los datos. Las variables latentes normalmente tienen una sencilla distribución de probabilidad. Muestreando esta distribución y pasando el resultado a un modelo de deep learning, podemos crear nuevos ejemplos.

Estos modelos aportan nuevos métodos para manipular datos reales. Por ejemplo, considera encontrar las variables latentes que unen dos ejemplos reales. Podemos interpolar estos ejemplos

interpolando las representaciones de variables latentes y esto nos da la posibilidad de obtener cualquier ejemplo intermedio usando datos de baja dimensionalidad para generar datos de alta dimensionalidad.

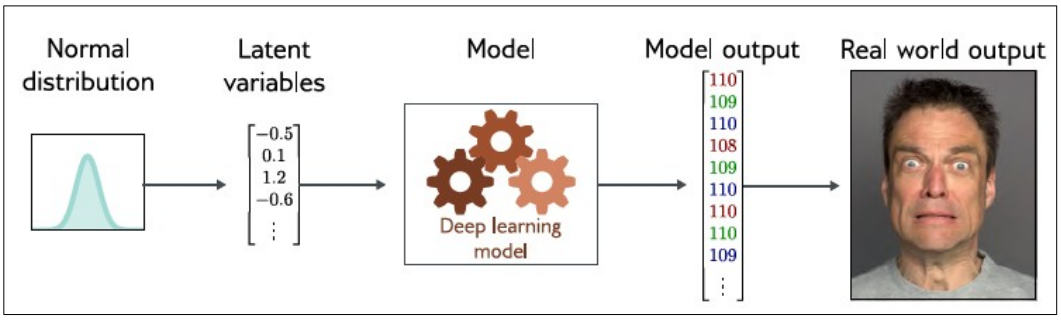


Figura 25. *Variables latentes. Muchos modelos generativos utilizan modelos deep learning.*

Figura 26. *Interpolación de imágenes. En cada fila, la figura de la izquierda y derecha son originales y las intermedias son representaciones más cerca a una u otra que crea el modelo generativo interpolando sus variables latentes.*

Conectando el aprendizaje supervisado y no supervisado

Los modelos generativos con variables latentes también se benefician de los modelos de aprendizaje supervisado cuando las salidas que producen tienen estructura. Por ejemplo, considera aprender a generar imágenes que coincidan con una descripción. En vez de mapear directamente el texto de la entrada a una imagen, aprendemos la relación entre las variables latentes del texto y las variables latentes de las imágenes.

Esto tiene 3 ventajas:

- Necesitamos menos parejas (texto, imagen) para aprender este mapeo porque las entradas y salidas ahora son menos dimensionales.
- Hay más facilidad para generar una imagen posible porque cualquier valor de las variables latentes deberían producir algo que parezca un ejemplo posible.
- Si introducimos aleatoriedad al mapeo entre dos conjuntos de variables latentes la imagen cambiará, y podemos generar muchas imágenes distintas todas descritas con el mismo texto.

4. EL PROCESO DE

Este diagrama ilustra el proceso de trabajo en Machine Learning de un sistema supervisado:

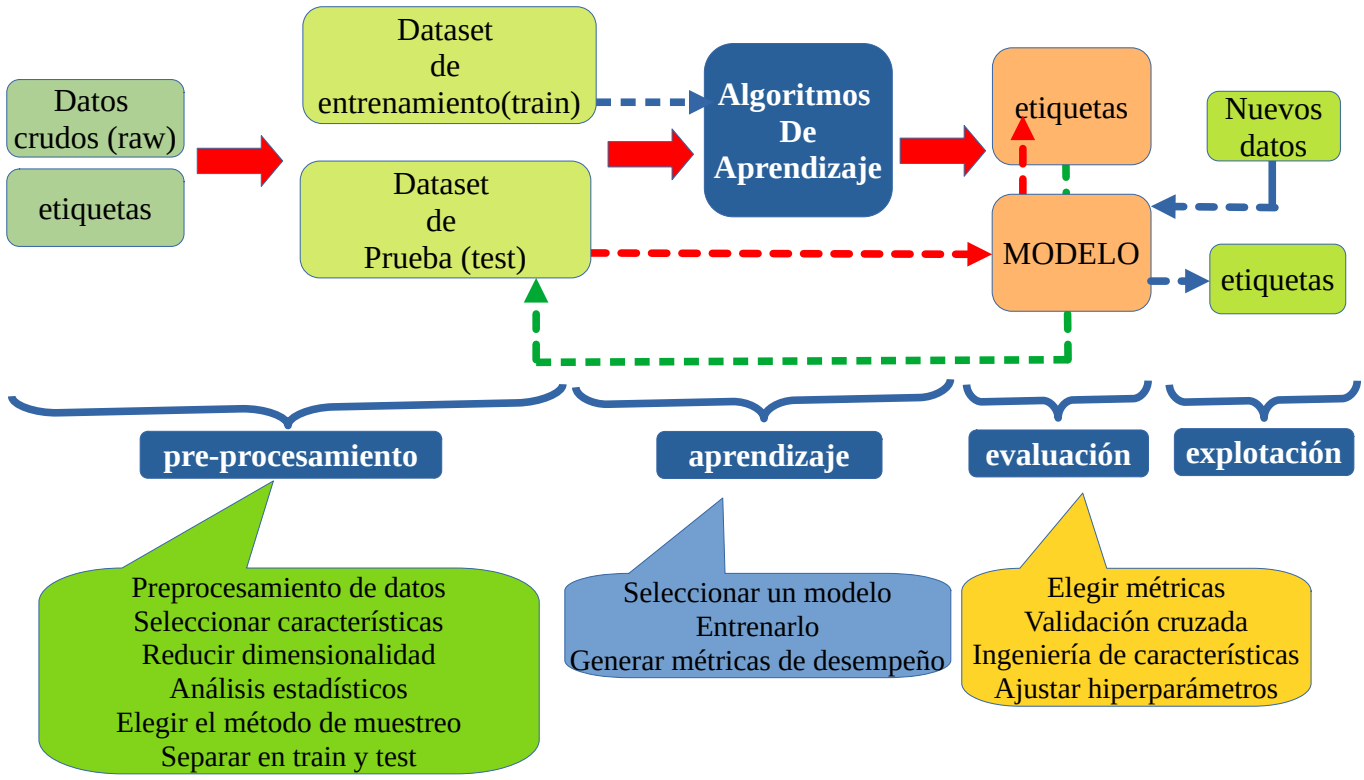


Figura 27. *Fases de la implementación de un sistema ML.*

Cuando desarrollemos un sistema ML, tras evaluarlo, difícilmente podremos estar satisfechos con el resultado en su primera versión. Y aquí es donde entra en juego la necesidad de conocer lo máximo posible los elementos que influyen en el resultado.

Es decir, hay tantos factores que podemos ajustar durante el proceso y que afectan al resultado, que modificarlos por fuerza bruta o por prueba-error es inviable. Hay que aplicar una especie de sentido común (que es más conveniente modificar para obtener la mayor mejora posible). Es decir, como una especie de método de ingeniería: máximo beneficio con mínimo esfuerzo.

4.1. PREPROCESAMIENTO DE DATOS.

El modelo que queremos obtener debe cumplir ciertos requisitos. Las cosas que pueden influir en obtener un sistema que no funcione tan bien como necesitamos son muchas, a continuación comentamos algunas de las más importantes asociadas a problemas en los datos.

Ingeniería de características: no se suele utilizar en deep learning, pero es común en el resto de algoritmos de ML. Consiste en una serie de técnicas para investigar, detectar y corregir o al menos minimizar algunos problemas que tienen los datos antes de dárselos a los algoritmos de aprendizaje para que configuren el modelo. Durante esta fase aplicaremos mejoras a los datos para hacerlos más adecuados a nuestros algoritmos de aprendizaje. Entre estas tareas:

- **Limpieza de datos**: Debemos conseguir que los datos no tengan valores ausentes (**missing**), que no haya datos incorrectos (no haya datos anómalos llamados **outliers**), que sus valores estén dentro de un rango similar (escalar datos) ni ruidos.
  - **Normalización**: la diferencia de valores que toman diferentes características puede tener un impacto muy negativo en muchos algoritmos de aprendizaje. Si tenemos una variable con valores entre 0 y 10 y otra variables con valores entre 0 y 20000, algunos algoritmos darán mucha influencia a la segunda y poca a la primera cuando en la realidad podría no suceder esto. Otro inconveniente es que ralentizan el aprendizaje de otros, o incluso generan errores de cálculo. La normalización consiste en cambiar la

escala de los valores que toman las columnas sin perder sus propiedades estadísticas y se puede implementar escalando o normalizando los datos.
    - **Detectar/Corregir outliers**: los modelos lineales están muy influenciados por los datos outliers y dan a lugar a modelos inexactos. Si lo que buscamos son patrones outliers podemos usar algoritmos de clusterización.
    - **Datos ausentes**: Aunque hay modelos probabilísticos que se comportan bien antes la ausencia de valores, en general son muy perjudiciales para la mayoría. La solución en valores de una columna depende del porcentaje de datos ausentes que tenga. Se puede cambiar el valor con la media o mediana (si es continuo) o la moda (si es categórico), o quitar el ejemplo del dataset, o construir un modelo predictivo para darle valor a partir del resto de variables, …
- **Selección de características**: eliminar columnas del dataset porque se combinan los valores en otra o directamente porque no aportan información que ayude a los algoritmos o porque al tener demasiadas generamos modelos demasiado complejos o con poca capacidad de generalizar (con overfitting).
- **Extracción de características**: mezclar características existentes en nuevas (operaciones).

PROBLEMA 1: CANTIDAD DE DATOS DE ENTRENAMIENTO INSUFICIENTE.
Para que un bebé aprenda que es una manzana, debes mostrarle una y decirle "manzana" señalándola. Y tendrás que repetir esto unas cuantas veces. Con el paso del tiempo el niño aprenderá a reconocer todo tipo de manzanas, de varios tamaños, colores y formas. Un genio el niño. Los sistemas de ML no son muy diferentes, incluso para problemas sencillos necesitas cientos de ejemplos y para problemas más complejos como reconocimiento de imágenes o de habla necesitas millones (salvo que reutilizas partes existentes de un modelo previo).

La No Razonable Pérdida de Efectividad de los Datos: en un documento publicado en 2001 por los investigadores de Microsoft Michele Banko y Eric Brill donde se mostró como muy diferentes algoritmos de ML incluyendo algunos muy simples, realizaban igual de bien tareas complejas como la desambiguación de lenguaje natural cuando se les aportaba suficiente cantidad de datos (figura 28). Como los autores indicaban, hay que pensar si es más ventajoso invertir esfuerzo y dinero en desarrollar un mejor algoritmo o esforzarse en conseguir una gran cantidad de datos.

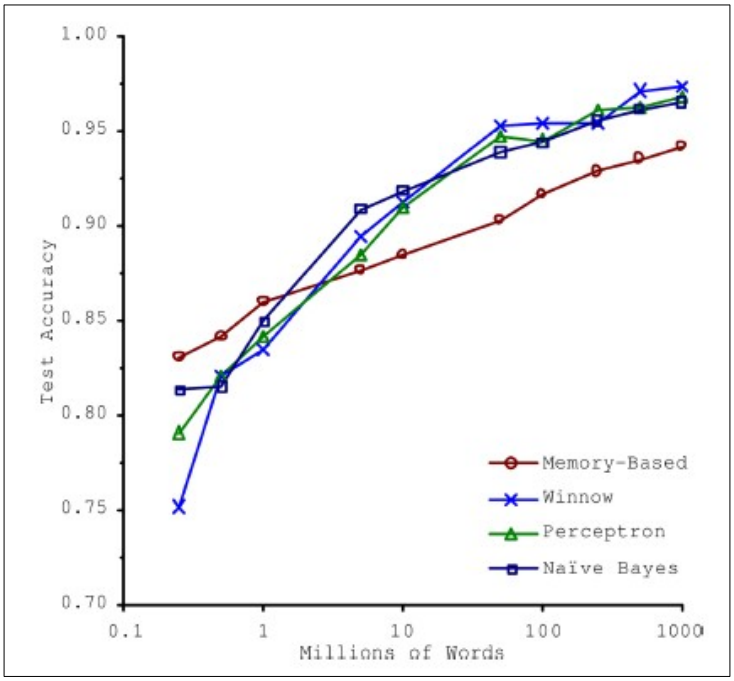


Figura 28. *La importancia de los datos contra la de los algoritmos.*

La idea de que para resolver problemas complejos los datos influyen más en los resultados que los propios algoritmos usados la popularizó en 2009 Peter Norvig et al. En un documento titulado ***"The Unreasonable Effectiveness of Data"***. Sin embargo, debemos ser conscientes de que algunas veces usamos datasets de tamaños pequeños o medianos porque obtener más datos es complicado, caro o imposible. Por tanto: **mejorar los algoritmos sigue siendo una tarea fundamental en ML.**

PROBLEMA 2: DATOS DE ENTRENAMIENTO NO REPRESENTATIVOS.
Para generalizar bien es importante que los datos de entrenamiento sean representativos de los nuevos casos que quieres utilizar. Esto es cierto tanto para sistemas de aprendizaje basados en instancias o aprendizaje basado en modelos.

Por ejemplo, el conjunto de países que podemos usar para entrenar un modelo lineal que prediga el nivel de satisfacción de vida de un país no es representativo porque no aparecen unos cuantos países clave (los 7 cuadrados rojos de la figura 29). La figura 29 muestra como es el modelo sin estos datos (línea punteada azul) y como es cuando añades estos 7 países ausentes (línea continua negra).

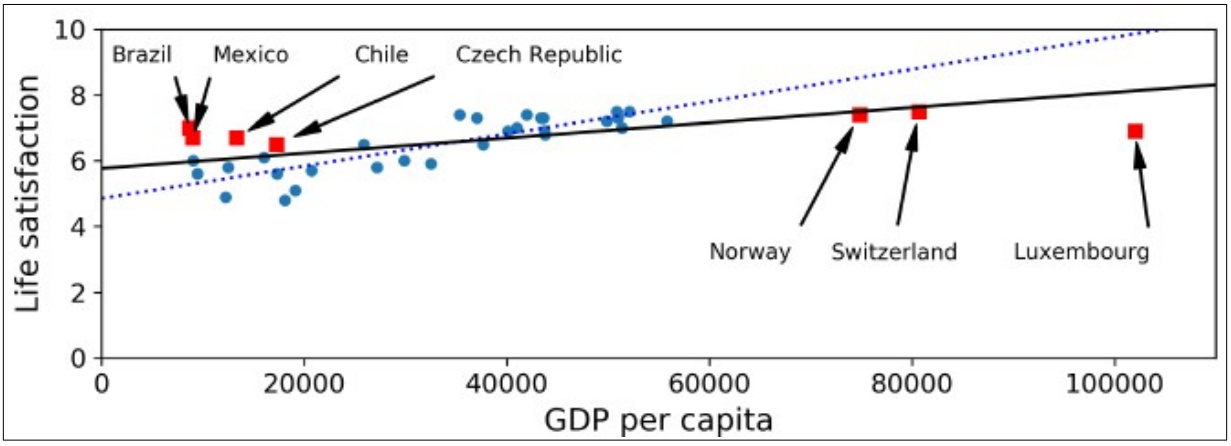


Figura 29. *Una muestra de ejemplos más representativa.*

Como ves, no solo cambias el modelo al añadir unos cuantos países sino que parece claro que el modelo orignal probablemente nunca funcionará muy bien. Los datos originales dan como países muy ricos a países de riqueza moderada y dará como felices a muchos países pobres, más que muchos países ricos. Usando datos no representativos para entrenar, el modelo no puede hacer predicciones precisas, sobre todo para países muy pobres o países muy ricos.

Es importante **usar datos de entrenamiento que representen los casos que quieres generalizar.** Solucionar esto es más difícil que decirlo: si tienes poca cantidad de datos tendrás **ruido muestral** (*sampling noise*[10]), es decir, datos no representativos. Pero aunque tengas muchos datos, puede que no sean representativos. Esto se llama **muestra sesgada** (*sampling bias*).

Quizás el ejemplo más conocido de este problema ocurrió en las elecciones de 1936 a la presidencia de USA en la que competían Landon contra Roosevelt: se realizó una gran encuesta enviando una carta a unos 10 millones de personas. Obtuvieron 2.4 millones de respuestas y predijeron que ganaría Landon con un 57% de los votos. *Pero la realidad fue que ganó* Roosevelt con el 62% de votos. Los fallos del método de muestreo fueron:

- Para obtener la dirección a quien enviar la encuesta, la empresa utilizó el listín telefónico, listas de suscripciones a revistas y listas de miembros de clubs. Todas estas listas tienden a tener personas bien posicionadas socialmente que son más propensos a votar al partido republicano (Landon).

10 ***Sampling noise****: ruido muestral que aparece cuando hay pocos datos o muchos pero no muy representativos del total en cuyo caso se denomina sampling bias (muestra sesgada).*

- Menos del 25% de quienes recibieron la encuesta respondieron. De nuevo esto introduce más sampling bias, descartando a quienes no simpatizaban con este político y otros grupos clave. Este es un tipo especial de sampling bias llamado *nonresponse bias*.

Otro ejemplo: imagina que quieres implementar un sistema que reconozca vídeos de música funk. Una forma de preparar el dataset de entrenamiento es buscar en YouTube y usar los vídeos resultantes. Pero este método está asumiendo que el motor de búsqueda de YouTube devuelve un conjunto de vídeos que son representativos de todos los vídeos de música funk. En realidad los resultados están sesgados (tienen bias) favoreciendo a los artistas más populares (si vives en Brasil obtienes una gran cantidad de vídeos de "funk carioca" que no se parecen en nada a la música de James Brown). Por otro lado, ¿Cómo obtener una cantidad grande de datos si no es así?

PROBLEMA 3: DATOS DE POBRE CALIDAD.
Obviamente, si tus datos de entrenamiento están repletos de errores, outliers y ruidos (es decir, poca calidad), será difícil de conseguir un buen sistema que detecte los patrones ocultos de los datos. Este es el objetivo de la limpieza de datos de entrenamiento. Los científicos de datos pasan mucho tiempo realizando tareas como:

- Si algunas instancias son claramente outliers, mejorarían los datos descartando simplemente el ejemplo o modificando los valores manualmente.
- Si algunas instancias tienen características perdidas (es decir, el 5% de tus clientes no indican su edad por ejemplo), debes decidir si quieres ignorar esta característica, ignorar el 5% de los ejemplos de clientes, rellenar los valores (con la edad mediana) o entrenar un modelo sin esta característica y otro con ella.

4.2. ANÁLISIS EXPLORATORIO DE DATOS.

Durante esta fase recopilamos información de nuestros datos. Calcularemos estadísticos descriptivos de cada columna continua y de frecuencias en las categóricas. Investigamos las posibles relaciones de cada columna con el resto para asegurarnos de que son estadísticamente independientes, podemos hacer contrastes de hipótesis estadísticos sobre su normalidad, o mediante cálculos o mediante visualizaciones.

También podemos aplicar algoritmos de clusterización para estudiar posibles zonas con alta densidad de datos y saber si es preferible tener varios modelos uno para cada cluster, o un único modelo para todos los datos.

PROBLEMA 4: CARACTERÍSTICAS IRRELEVANTES.
Tu sistema solamente será capaz de aprender si los datos de entrenamiento contienen suficientes características relevantes y no demasiadas irrelevantes. Otra parte crítica del proceso de implementar un modelo es escoger un buen conjunto de características para entrenarlo. Este proceso se llama **ingeniería de características** (*feature engineering*) y consiste en:

- ***Seleccionar características***: seleccionar las características más usadas o existentes.
- ***Extracción de características:*** combinando características existentes y sustituirlas por otra más usable (los algoritmos de reducción de dimensiones pueden ayudar).
- ***Crear nuevas características***: recolectar nuevos datos que no se tenían antes.

4.3. MODELADO.

Ahora que hemos repasado algunos problemas que pueden estar originados en los datos, vamos a ver otros que pueden provocar los algoritmos.

PROBLEMA 5: SOBREAJUSTAR LOS DATOS DE ENTRENAMIENTO (Overfitting).
Si al visitar una ciudad diferente a la que vives, das con un taxista que te da un mal servicio (te pega un sablazo, etc.) tendrás la tentación de generalizar diciendo que los taxistas de ese lugar son poco profesionales. El defecto de generalizar en exceso es algo que los humanos hacemos con mucha frecuencia y un defecto que los sistemas de ML también pueden cometer. En ML a este defecto se le llama ***overfitting***: significa que el modelo funciona bien con los datos de

entrenamiento pero no generaliza bien, es decir, no funciona tan bien con los datos de test ni con los datos de trabajo.

La figura 30 muestra un modelo de regresión polinómica de alto grado que sobreajusta a los datos de entrenamiento (overfitting). Con los datos de entrenamiento funciona muy bien. Si un modelo simple funciona bien con los datos de entrenamiento ¿Puedes confiar en sus predicciones?

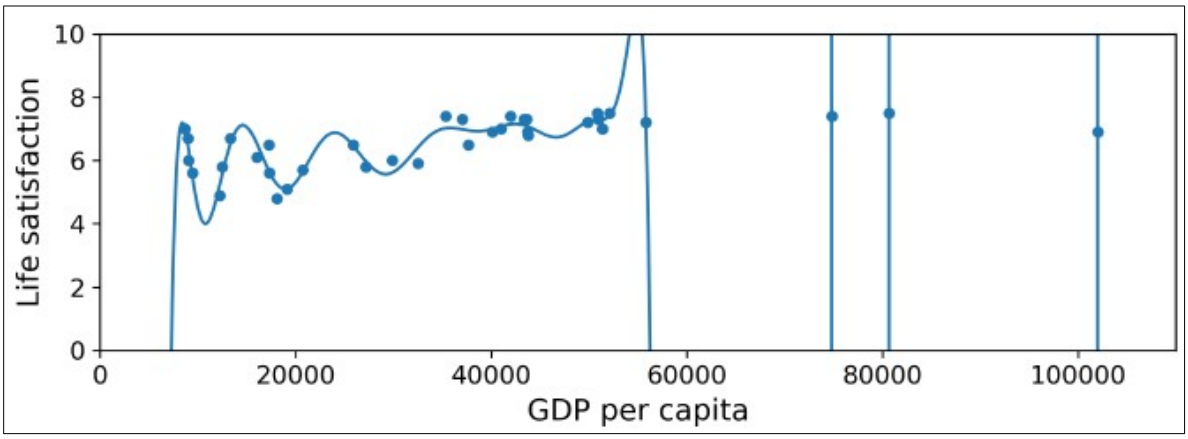


***Figura 30**. Modelo con Overfitting de los datos de entrenamiento.*

Los modelos complejos como las redes neuronales profundas son capaces de detectar patrones sutiles en los datos, pero si los datos de entrenamiento son ruidosos o hay escasa cantidad de datos (sampling noise), el modelo detecta y aprende los patrones del propio ruido. Lógicamente esto impide generalizar a nuevas instancias.

Si añadimos otra característica al modelo que no aporta datos significativos como el nombre del país, un modelo deep learning podría detectar que el nivel de vida de los países cuyo nombre tiene una letra 'w' tienen como mínimo un nivel de satisfacción de 7 porque le hemos dado datos como New Zealand (7.3), Norway (7.4), Sweden (7.2) y Switzerland (7.5).

Pero meterá la pata cuando use este patrón para generalizar a países como Rwanda o Zimbabwe. El problema es que el nivel de satisfacción no depende en absoluto del nombre del país, este patrón aparece por puro azar pero que el nombre del país tenga una letra 'w' o no la tenga no es algo que influya en el nivel de satisfacción de los ciudadanos de ese país (al menos, no en general), pero el modelo no tiene manera de diferenciar cuando estos patrones son producto de la realidad o de datos ruidosos.

El overfitting ocurre cuando el modelo es demasiado complejo en relación a la cantidad de datos de entrenamiento y a la cantidad de ruido que tienen o se ha entrenado en exceso. Las posibles soluciones son:

- **Simplificar el modelo seleccionando con menos parámetros:**
  - Cambiar de modelo: un modelo lineal en vez de un modelo polinomial de alto grado.
  - Reduciendo el número de características de los datos de entrenamiento
  - Restringiendo el modelo: **aplicando una regularización**.
- **Recopilar más datos de entrenamiento**.
- **Reducir el ruido en los datos de entrenamiento**
  - Corregir datos erróneos
  - Eliminar outliers.

A la técnica de restringir el modelo para simplificarlo y reducir el riesgo de tener overfitting se le llama ***regularización***. Por ejemplo, si un modelo lineal está definido con dos parámetros θ_0 y θ_1, esto da al modelo dos grados de libertad para adaptar el modelo a los datos de entrenamiento: puede modificar tanto la altura (θ_0) como la pendiente (θ_1) de la línea. Si forzamos a que $\theta_1 = 0$, el algoritmo solo tiene libertad para modificar un parámetro y tendrá más dificultades para sobre

ajustar bien todos los datos porque todo lo que puede hacer es aproximarse a los ejemplos y debe quedarse como mucho en la media. Pierde ajuste y gana capacidad de generalización.

Si permitimos al algoritmo modificar θ_1 pero forzamos a mantenerlo pequeño, entonces el algoritmo de aprendizaje tendrá una libertad efectiva entre 1 y 2 grados de libertad. Esto genera un modelo más simple que uno de dos grados de libertad.

Debes conseguir un balance entre aproximar los datos de entrenamiento y tener la capacidad de generalizar bien los nuevos datos con los que deba trabajar del modelo. La figura 31 muestra los tres modelos del nivel de satisfacción de diferentes países: la línea punteada de color azul tiene el modelo de datos con los datos que faltan, la línea discontinua tiene todos los países y la línea continua usa los mismos datos que el primer modelo pero regularizado con una restricción que lo obliga a tener una pendiente pequeña. Esta restricción hace que no fije tan bien los datos de entrenamiento pero consigue generalizar mejor nuevos datos reduciendo el overfitting y pareciéndose más a la situación ideal (la línea discontinua con todos los datos).

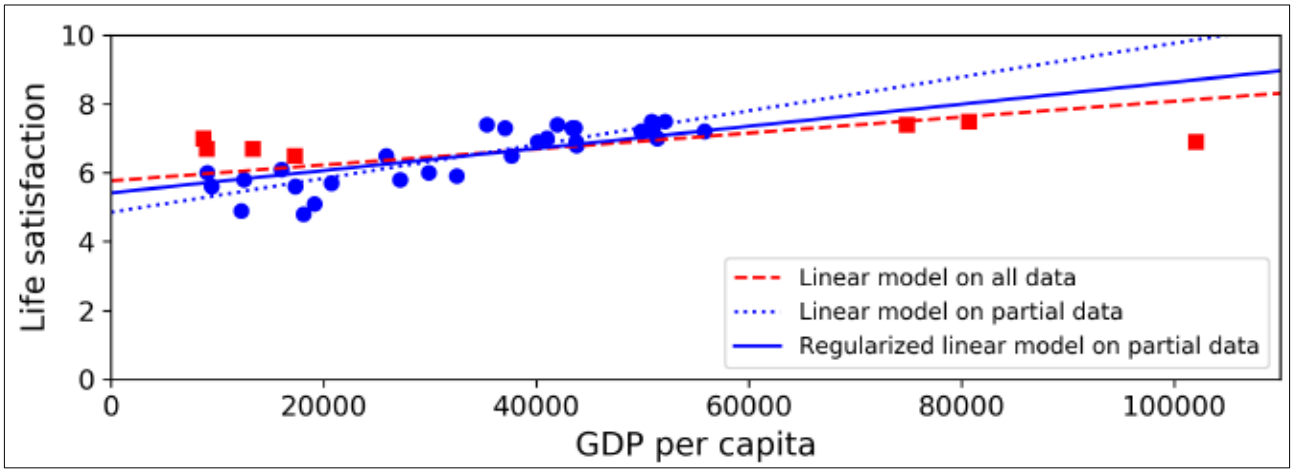


***Figura 31**. La regularización reduce el riesgo de overfitting.*

La cantidad de regularización a aplicar durante el aprendizaje se puede controlar con un hiperparámetro.

Un **hiperparámetro** es un parámetro que afecta al algoritmo de aprendizaje (no al modelo). Si afecta al propio algoritmo, debe establecerse antes de entrenar y permanecer constante durante el entrenamiento. Si fijas un valor del hiperparámetro muy alto obtendrás un modelo casi plano (la pendiente caso cero). Aunque no tenga overfitting, no generaliza bien. Escoger bien los hiperparámetros es otro aspecto clave de la implementación de sistemas de aprendizaje.

PROBLEMA 6: UNDERFITTING DE LOS DATOS DE ENTRENAMIENTO.
El underfitting es lo opuesto al overfitting: ocurre cuando el modelo es demasiado simple para aprender las estructuras de los datos. Por ejemplo un modelo lineal es propenso a tenerlo si una línea recta no es un buen elemento para solucionar el problema. Para solucionarlo:

- Seleccionar un modelo más potente, con más parámetros.
- Usar mejores características (feature engineering).
- Reducir las restricciones del modelo (reducir el hiperparámetro de regularización).

4.4. TESTEO, VALIDACIÓN Y AJUSTE DEL MODELO.

La única forma de comprobar cuando un modelo generaliza bien los nuevos casos es trabajar con ellos. Una manera de hacerlo sería poner el modelo en producción y ver como de bien funciona. Pero esta estrategia es muy peligrosa, porque si el modelo funciona horriblemente mal, puede tener consecuencias importantes para la organización que lo ha desplegado.

DIVIDIR DATOS EN TRAIN Y TEST

Una mejor opción, que es la normalmente se utiliza es dividir los datos disponibles en dos conjuntos distintos: el ***training set*** (conjunto de datos de entrenamiento) y el ***test set*** (conjunto de datos de prueba). Como sus nombres ya proponen, usas los datos de entrenamiento para que el modelo aprenda (entrenar el modelo) y usas los datos de prueba para medir lo bien o mal que generaliza el modelo cuando debe trabajar con datos nuevos. El ratio de error que cometa con los nuevos datos se llama error de generalización (o también *out-ofsample error*) y al evaluar el modelo con los datos de test puedes hacer una estimación de este error. Este valor te dará una indicación de lo bien que realiza su trabajo el modelo con datos que nunca ha visto antes.

Si el error de entrenamiento es bajo (el modelo comete pocos fallos con los datos de entrenamiento) pero el error de generalización es alto, tu modelo tiene overfitting (tiene demasiado entrenamiento, o es muy complejo, o los datos son ruidosos).

Lo normal es utilizar un 70 a 80% de los datos para entrenar y el 30 a 20% restante para testear. Sin embargo estos porcentajes dependen de la cantidad de datos disponibles: si hay 10 millones de instancias, probar con un 1% significa que que probarás cien mil ejemplos que probablemente sea suficiente cantidad para estimar el error.

***Figura 32**. Dividir datos en train y test para medir como generaliza el modelo.*

DIVIDIR DATOS EN TRAIN, VALIDACIÓN Y TEST

Evaluar un modelo es una tarea bastante simple: usar los datos prueba. Pero imagina que estás dudando entre dos modelos (por ejemplo uno lineal y otro polinomial). ¿Por cuál te decides? Una opción es entrenar ambos modelos y comparar como de bien generalizan con los datos de prueba.

Ahora imagina que el modelo lineal generaliza mejor, pero quieres aplicarle alguna regularización para evitar el overfitting. Ahora la pregunta es: ¿Cómo escoges los valores de los hiperparámetros de regularización? Una forma es entrenar 100 modelos diferentes usando 100 valores diferentes de este hiperparámetro. Supongamos que encuentras el mejor hiperparámetro que da lugar al modelo con menor error de generalización, digamos un error de 5%.

Así que pones ese modelo en producción pero con tan mala suerte que su error alcanza el 15%. ¿Qué acaba de ocurrir? El problema es que has medido el error de generalización varias veces con los datos de prueba, y has adaptado el modelo y sus hiperparámetros a ese particular conjunto de datos. Es decir, el modelo no funciona tan bien con nuevos datos.

Una solución muy usada para este problema es llamada la **validación de reserva** (*holdout validation*): simplemente sacas algunos de los datos de entrenamiento para evaluar modelos candidatos y seleccionas el mejor. Los nuevos datos se llaman datos de validación (*validation set* o *development set*, o *dev set*). Es decir, entrenas muchos modelos con varios hiperparámetros con una reducida cantidad de datos de entrenamiento (todos los datos de entrenamiento menos los datos de validación) y seleccionas el modelo que mejor trabaja sobre los datos de validación. Es decir, paras de entrenar cuando el modelo presenta overfitting (cuando el error con los datos de validación comienzan a crecer pero con train no).

***Figura 33**. Dividir datos en train, test y validación para medir como generaliza el modelo.*

Esta solución también nos permite averiguar cuando un modelo se comporta mal debido a que tiene sobreajuste o si lo hace porque hemos usado datos incongruentes.

DATOS INCONGRUENTES

Algunas veces es sencillo obtener una gran cantidad de datos para entrenar, pero no son perfectamente representativos de los datos usados en producción. Por ejemplo, imagina que creamos una app para el móvil que tome fotos de flores y automáticamente determine su especie. Puedes descargar millones de flores, de la web, pero no serán representativas de las fotografías realizadas con un móvil. Quizás solamente 10000 sean aprovechables. En estos casos la regla más importante a recordar es que el conjunto de validación y el conjunto de test deben ser lo más representativos posibles de los datos que pueda usar el modelo en el entorno de producción. Así que en el ejemplo, solamente estas 10000 fotografías deben ser datos de test o datos de validación. Por ejemplo puedes repartirlas entre ambos conjuntos, pero asegurando que no hay repeticiones, o están en un lugar o en otro. Si mezclas y el modelo no se comporta bien, no sabrás si es por overfitting o por que tienes datos no representativos.

Una solución es separar parte de las fotografías (de la web) en otro conjunto de validación (Andrew Ng llama *train-dev set)*. Después entrenas el modelo (con el training set, no con el de validación), puedes evaluarlo con el de validación:

- Si se comporta bien, no tiene overfitting, así que si se comporta mal con los datos de test, el problema debe venir de datos incompatibles. Puedes intentar preprocesar las imágnes web para que se parezcan más a las fotografías capturadas con el móvil y entonces reentrenar el modelo.
- Si el modelo se comporta mal con el de validación, entonces el modelo debe tener overfitting, así que debes intentar simplificarlo o regularizarlo, obtener más datos de entrenamiento, preprocesar los datos, etc.

TEOREMA DEL DESAYUNO NO GRATIS

Un modelo es una versión simplificada de las observaciones. Las simplificaciones consisten en descartar detalles superfluos que no ayudan a generalizar con nuevas instancias. Sin embargo, para decidir qué datos descartar y cuales mantener,no deberías realizar suposiciones.

Por ejemplo, un modelo lineal hace la suposición de que los datos tienen relaciones lineales y que las distancias entre la línea recta del modelo y los datos reales son ruido y puede ignorarse de manera segura.

En un famoso documento de 1996, David Wolpert demostraba que si no haces ninguna suposición sobre los datos, no hay ninguna razón por la que debas preferir un modelo sobre otro. Esto es a lo que se llama el teorema de no hay desayuno gratis. Para algunos datos el mejor modelo es el lineal, mientras que para otros es una red neuronal. No hay modelo que a priori garantice que sea mejor que otro (de ahí el nombre del teorema). La única manera de saberlo es evaluarlos todos.

Esto no es posible, en la práctica debes realizar suposiciones razonables sobre los datos y evaluar solamente unos cuantos de los posibles. Para problemas sencillos, modelos sencillos y para problemas más complejos otras opciones.

En cualquier caso, tras pasar la prueba de validación, este proceso, entrenas el mejor modelo con todos los datos de entrenamiento (incluidos los datos de validación), y ese será tu modelo final. Por último evalúas este modelo final con los datos de test para obtener la estimación del error de generalización.

Esta solución normalmente funciona muy bien, sin embargo, si hay pocos datos de validación, las evaluaciones de los modelos serán imprecisas: puedes acabar seleccionando un modelo no óptimo. Por otro lado, si la cantidad de datos usados para la validación es demasiado grande, la cantidad de datos usados para entrenar será mucho más pequeña que el total de los datos. ¿Porqué esto no es nada bueno? Porque como el modelo al final se entrena con todos los datos de entrenamiento, no es buena idea comparar diferentes modelos entrenados con muy pocos datos, porque sería como escoger a un velocista para correr una maratón.

AJUSTAR EL MODELO CON VALIDACIÓN CRUZADA

Una manera de solucionar este problema es realizar validaciones repetidas llamada validaciones cruzadas (*cross-validation*), usando muchos diferentes conjuntos de datos de validación. Proporciona estimaciones de error más robustas. Cada modelo se evalúa una vez con sus datos de evaluación, después es entrenado con el resto de datos. De media, todas las evaluaciones de un modelo obtienen medidas mucho más precisas de su funcionamiento.

- En primer lugar se dividen los datos en train y test.
- En segundo lugar se dividen los datos de train en K carpetas (K-folds).
- Cada posible modelo se entrena y valida k veces usando un fold de validación distinto en cada ocasión. Se usa la media de los k errores de validación para obtener una estimación del error promedio que es mejor medida del error mejor.
- Se escoge el mejor modelo y se entrena con todos los datos de train.
- Luego se testea contra los datos de test.

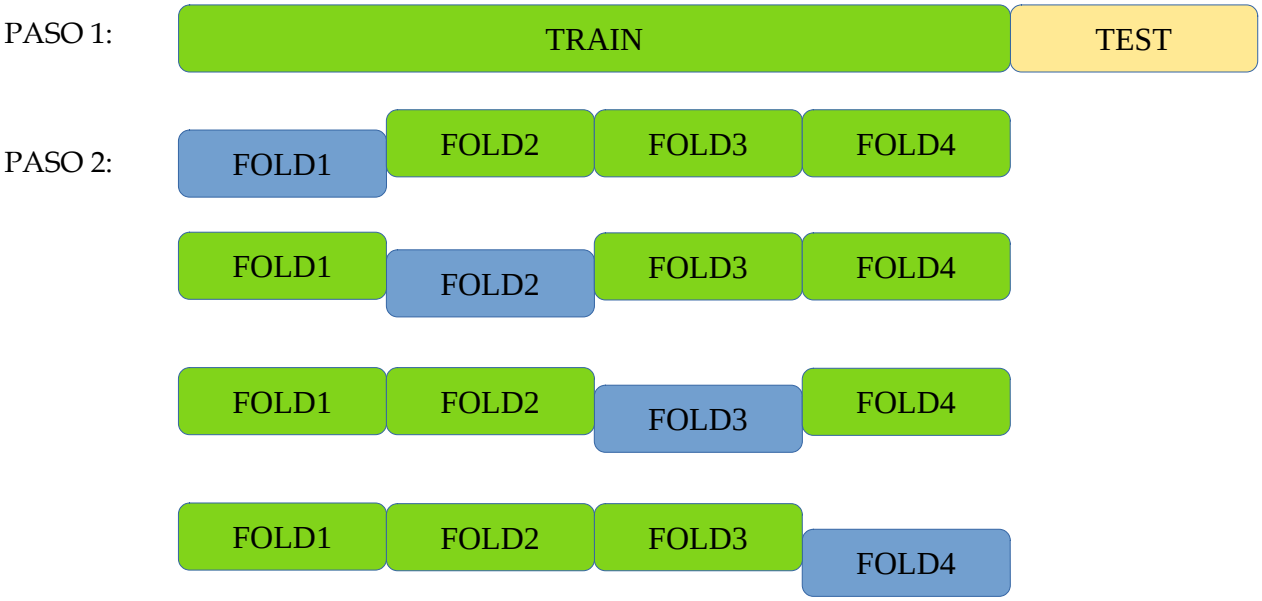


***Figura 34**. Dividir datos de train en 4-folds.*

Solo hay un inconveniente: el tiempo de entrenamiento se multiplica por k, por el número de conjuntos de validación.

5. HERRAMIENTAS Y CONOCIMIENTOS PARA ML.

La mayoría de grandes empresas ponen en sus nubes entornos para desarrollar sistemas ML. Si hiciésemos sistemas reales lo ideal sería utilizar estos recursos, a no ser que tuviésemos la posibilidad de hacer una gran inversión en hardware especializado sobre todo. Por mencionar algunos entornos:

- Amazon SageMaker.
- Google Cloud AI.
- Gooble collab (si tienes cuenta de google puedes darte de alta y tienes en drive una carpeta)
- Watson de IBM
- Azure ML

También tenemos sitios que promueven el aprendizaje, concentran muchos recursos o incluso organizan desafíos y concursos como por ejemplo: Kaggle, DriveData, Devpots, Innoncentive, CrowdAnalytix.

Además tenemos acceso a algunos modelos exitosos ya entrenados y listos para usar que universidades y empresas hacen disponibles de manera abierta como Gemma (de Google) o Llama 2 (de meta, que es una gan para texto que tiene 70 billones de parámetros), etc. Además te comento algunas herramientas como:

- Herramientas matemáticas para construir prototipos de modelos:
  - **MatLab**: herramienta de pago para estudio matemático. Tiene su propio lenguaje de programación, muy simbólico, además de poder utilizarlo de manera muy intuitiva y potente.
  - **Octave**: es la versión open de matlab. Comparte incluso su lenguaje.
  - **SPS**: paquete estadístico de pago de IBM, implementa también algoritmos de ML.
  - **GNU PSPP**: la versión open de GNU.
  - **R** y R Studio.
- Paquetes y librerías de ciencia de datos / minería de datos:
  - **rapidminer studio**: ya es de pago
  - **Knime**: open source. Se realizan proyectos pegando operaciones.
  - **Orange**
  - **Anaconda**. Basada en Python.
- Frameworks / Lenguajes de programación:
  - python:
    - Anaconda
    - Librerías: scikitlearn, numpy, matplotlib, pandas, keras, tnsorFlow, Pytorch, seaborn, nlp, ...
  - java:
    - Weka
    - 4j

Nosotros vamos a utilizar sobre todo librerías basadas en el lenguaje de programación Python, principalmente:

- **scypy:** librería científica con muchos algoritmos matemáticos.
- **scikitlearn:** librería que implementa muchos algoritmos de ML listos para utilizar.
- **NumPy**: librería de Python para operar con matrices y vectores.
- **Pandas:** librería de Python para manejar datasets.
- **Matplotlib:** librería de Python para realizar gráficos.
- **TensorFlow y Pytorch**: para deep learning.

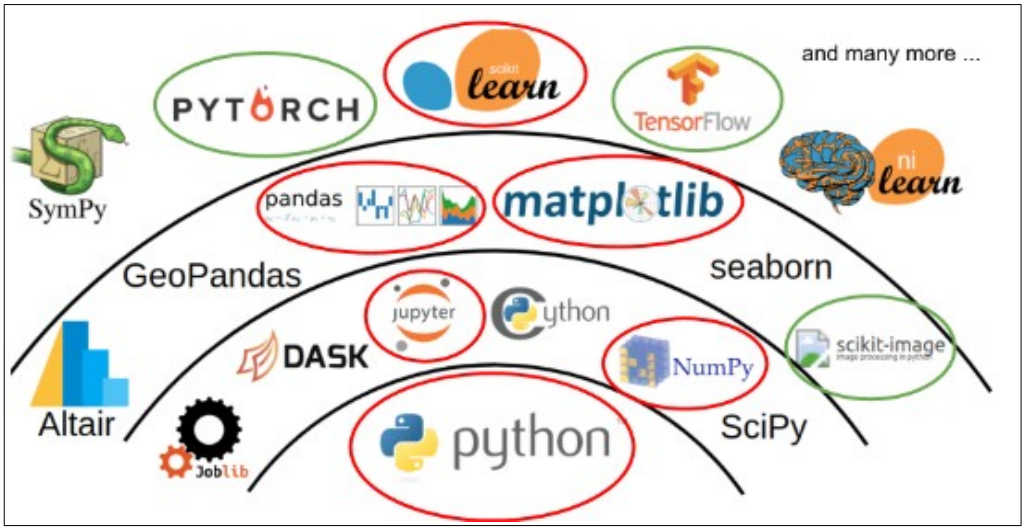


***Figura 35**. Librerías de Python para ML.*

5.1. EL LENGUAJE DE PROGRAMACIÓN Python.

Este apartado lo he suprimido porque tendréis a vuestra disposición un documento introductorio de este lenguaje. Es muy importante que lo dominéis con cierta soltura porque utilizaremos de manera frecuente muchas de sus librerías. Muy importante que le dediques el tiempo necesario para dominarlo sobre todo si nunca has usado este lenguaje de programación con anterioridad.

5.2. INTRODUCCIÓN A NUMPY.

NumPy es una extensión del lenguaje Python que añade soporte para arrays numéricos grandes, multidimensionales, matrices que usan otras muchas librerías adicionales de ML. Para utilizarla debes importarla:

```
import numpy as np
```

CREAR ARRAYS

Crea arrays a partir de listas. Todos los elementos de la lista deben tener el mismo tipo (o intenta convertirlos si puede):

```
datos1 = [1, 2, 3, 4, 5]              # Una lista
a1 = np.array(datos1)                 # un array 1d (una dimensión)
datos2 = [range(1, 5), range(5, 9)]   # Una lista de dos listas
a2 = np.array(datos2)                 # 2d array
a2.tolist()                           # Convierte un array a una lista
```

Crear arrays especiales:

```
np.zeros(10)                # Array de 10 ceros
np.zeros((3, 6))            # matriz de 3 filas y 6 columnas de ceros
np.ones(10)                 # Array de 10 unos
np.full((3,6), 4)           # matrix 3x6 con todos los valores a 4
np.empty((3, 6))            # matriz de 3 filas y 6 columnas sin valores
np.random.random((3, 6))    # matriz de 3 filas y 6 columnas con valores aleatorios
np.identity(6)              # matriz cuadrada de 6x6 con diagonal a 1.0 y resto a 0.
np.linspace(0, 1, 5)        # Array con 5 valores equidistantes entre 0 y 1 inclusive
np.logspace(0, 3, 4)        # Array con 4 valores desde 10^0 hasta 10^3
```

El método `arange()` es como `range()`, pero devuelve los valores en un array, no en una lista:

```
int_array = np.arange(5)
float_array = int_array.astype(float)
```

EXAMINAR ARRAYS

A un array podemos consultarle los siguientes datos:

- **`a.ndim`**: número de dimensiones.
- **`a.shape`**: estructura de cada dimensión.
- **`a.size`**: número de elementos que almacena.
- **`a.dtype`**: tipo de datos de los elementos que almacena.

```
print("a1:", a1, "tipo:", a1.dtype)        # int32
print("a2:", a2, "tipo:", a2.dtype)        # float64
print("Dimensiones de a2:", a2.ndim)       # 2
print("Estructura de a2:", a2.shape)       # (2, 4) - eje 0 son las filas,eje1 columnas
print("Longitud de a2:", a2.size)          # 8 cantidad de elementos en total
print("Longitud de 1ªdimensión:", len(a2)) # Tamaño de primera dimensión (aka eje)
```

CAMBIAR LA ESTRUCTURA

La estructura de un array indica la manera en que organiza los datos que almacena. Hablamos de dimensiones o ejes. Tenemos operaciones para consultarla y cambiarla sin cambiar los datos:

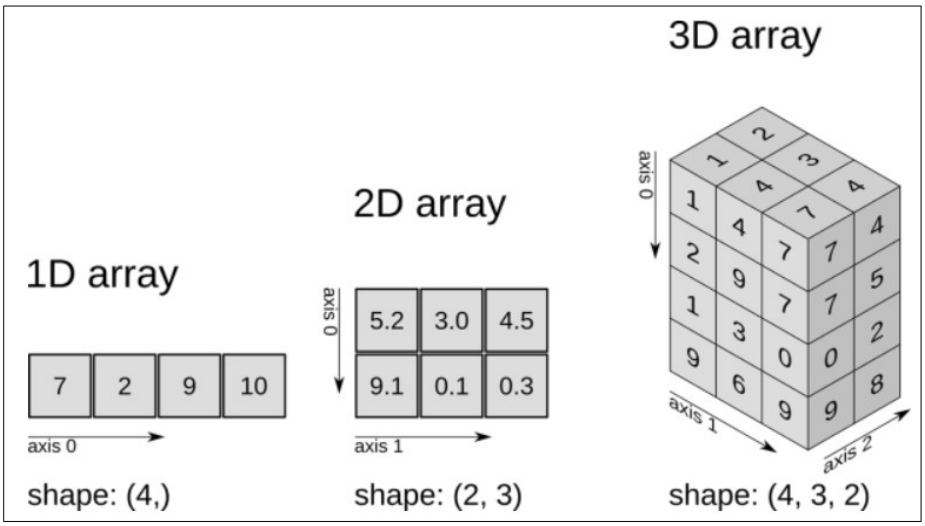


***Figura 36**. Estructuras de arrays de Numpy más usadas.*

```
a = np.arange(10, dtype=float)
print("Estructura:", a.shape, "valores:", a)
a = a.reshape(2,5)                          # método reshape(filas, columnas)
print("Estructura:", a.shape, "valores:", a)
```

Cambiar ejes (añadir o borrar):

```
a = np.array([0, 1])
a_col = a[:, np.newaxis]
print(a_col)
a_col = a[:, None]                          # o también podemos hacerlo así
```

Transpuesta:

```
print(a_col.T)
```

El método **`flatten()`** siempre devuelve una copia plana del array original.

```
a_flt = a.flatten()
a_flt[0] = 33
print("Original a:", a, "a.flatten():", a_flt)
print(arr)
```

El método **`ravel()`** devuelve una vista del array original.

```
a_flt = a.ravel()
a_flt[0] = 33
print("Original a:", a, "a.ravel():", a_flt)
```

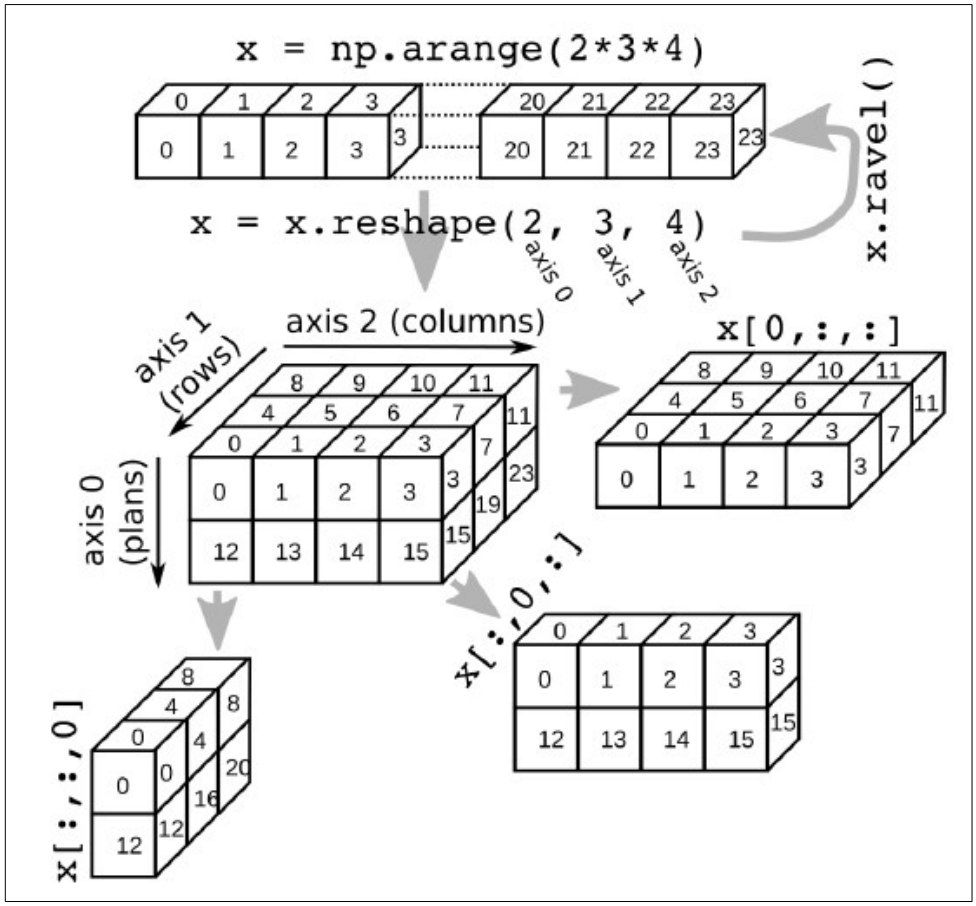


Figura 37. *Algunas operaciones de cambio de estructura.*

RECORRER LOS DATOS DEL ARRAY

Por defecto Numpy usa convenciones del lenguaje C: la matriz se almacena por filas. El último índice cambia más rápidamente cuando los programas procesan los datos de una matriz almacenada en memoria.

- Para arrays 2D, el recorrido secuencial en memoria equivale a recorrer los datos:
  - Iterando sobre las filas (eje 0)
  - Para cada fila, iterar por columnas (eje 1)
- Para arrays 3D el movimiento secuencial acaba por:
  - iterar sobre cada plano (eje 0)
  - Iterar sobre las filas (eje 1)
  - Iterar por las columnas (eje 2)

```
x = np.arange(2 * 3 * 4)                    # Crear array 1d de 24 elementos del 0 al 23
print("x:", x)
x = x.reshape(2, 3, 4)                      # Reshape a 3D (eje 0 (2), eje 1 (, eje 2)
print("x en 3D:", x)
print("Primer plano:", x[0, :, :])          # Seleccionar el primer plano
print("Primera fila:", x[:, 0, :])          # Seleccionar la primera fila
print("Primera columna:", x[:, :, 0])       # Seleccionar la primera columna
print("Ravel:", x.ravel())                  # aplicar ravel()
```

APILAR ARRAYS

Podemos crear un array uniendo otros arrays planos en columnas o en filas:

```
a = np.array([0, 1])                        # crear un array a
b = np.array([2, 3])                        # crear un array b
ab = np.stack((a, b))                       # Apilarlos en columnas en otro array
print("a:", a, "b:", b, "ab:", ab, "ab.T:", ab.T)
```

```
abh = np.hstack((a[:, None], b[:, None]))         # Apilarlos en filas
print("a:", a, "b:", b, "abh:", abh, "abh.T:", abh.T)
```

SELECCIONAR DATOS CON ÍNDICES Y OBJETOS SLICING

Un único elemento se selecciona como en las listas:

```
a = np.arange(10, dtype=float).reshape((2, 5))
print("Elemento 0:", a[0])                # 0-ésimo elemento (como si fuese una lista)
print("Con 2 índices:", a[0, 3])          # fila 0, columna 3: devuelve 4
print("Con 2 índices:", a[0][3])          # sintaxis alternativa
```

```
>>> a[0, 3:5]
array([3, 4])

>>> a[4:, 4:]
array([[44, 55],
       [54, 55]])

>>> a[:, 2]
a([2, 12, 22, 32, 42, 52])

>>> a[2::2, ::2]
array([[20, 22, 24],
       [40, 42, 44]])
```

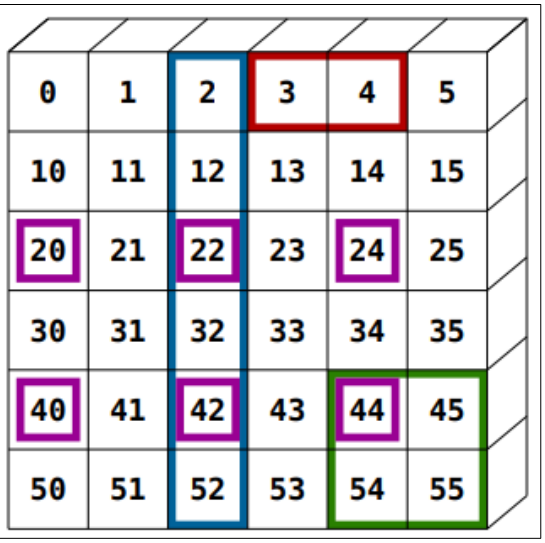


Figura 38. *Algunas operaciones de slicing en un array 2D.*

Un objeto slicing tiene esta sintaxis: desde:hasta:cambio donde el inicio por defecto es 0, el hasta por defecto es el último y el cambio por defecto es 1. Selecciona los elementos: `desde, desde+cambio, desde+2cambio, … hasta-1`. *La operación slicing devuelve una vista, por tanto si cambias en el resultado de slicing, estás cambiando el array original.*

```
print(a[0, :])              # Fila 0 todas las columnas: devuelve array 1d ([1, 2, 3, 4])
print(a[:, 0])              # Todas las filas de columna 0: devuelve array 1d ([1, 5])
print(a[:, :2])             # Todas las filas de las columnas menores a 2
print(a[:, 2:])             # Todas las filas de las columnas mayores o iguales a 2
print(a[:, 1:4])            # Todas las filas de las columnas 1, 2 y 3
a2 = a[1,:]
a2[0] = 33
print("Debes ver los cambios:", a)
print(a[0, ::-1])           # Desde el final al principio
```

INDEXAR CON OTROS ARRAYS (FANCY INDEXING)

Consiste en usar otro array para indicar las posiciones de otro que nos interesan utilizar). Indexación elegante en inglés, es como una indexación indirecta. Sigue devolviendo una vista, no una copia, por tanto si haces cambios al resultado, los estás haciendo al array original.

```
>>> a[(0,1,2,3,4), (1,2,3,4,5)]
array([1, 12, 23, 34, 45])

>>> a[3:, [0,2,5]]
array([[30, 32, 35],
       [40, 42, 45],
       [50, 52, 55]])

>>> mask = np.array([1,0,1,0,0,1], dtype=bool)
>>> a[mask, 2]
array([2, 22, 52])
```

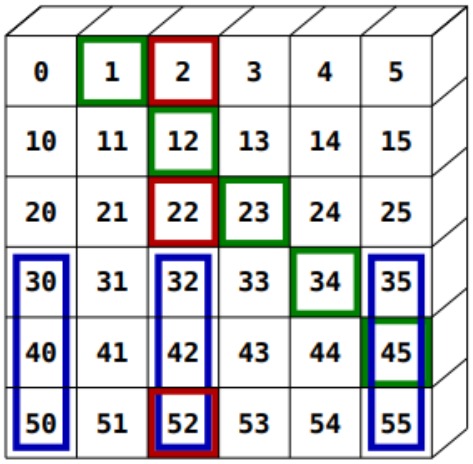


Figura 39. *Algunas operaciones de fancy indexing.*

Con arrays de enteros:

```
a = np.array([3, 13, 12, 10, 10])
a2 = a[:, [1,2,3]]          # Devuelve una copia
print("a2:", a2)
a2[0, 0] = 44
print("a2 cambiado:", a2, "a original:", a)
```

Con arrays de booleanos:

```
a = np.array([3, 13, 12, 10, 10])
mascara = (a % 3 == 0)              # Devuelve [True, False, True, False, False]
multiplos = a[mascara]              # También a[a % 3 == 0]
a[a % 3 == 0] = -1                  # Deja a como [-1, 13, -1, 10, 10]
a2 = a[a > 5]                       # Devuelve una copia [13, 10, 10]
print("a2:", a2)
a2[0] = 44
print("a2 cambiado:", a2, "a original:", a)
```

OPERACIONES VECTORIZADAS

Operaciones básicas con otro valor se aplican a todos los elementos:

```
a = np.array([1, 2, 3, 4])
print("a + 1:", a + 1)       # Imprime  [2, 3, 4, 5]
b = np.ones(4) + 1           # Devuelve [2, 2, 2, 2]
print("a:", a)               # Imprime: a: [1, 2, 3, 4]
print("b:", b)               # Imprime: b: [2, 2, 2, 2]
print("a - b:", a-b)         # Imprime: a-b: [-1., 0., 1., 2.]
a = a.reshape((2,2))         # Devuelve [[1, 2], [3, 4]]
b = np.array([1, 2], [0, 1]) # Devuelve [[1, 2], [0, 1]]
print("a * b:", a * b)       # Imprime [[1, 4], [0, 4]]  (producto elemento a elemento)
print("a @ b:", a @ b)       # Imprime [[1, 4], [3, 10]]  (producto matricial)

# Nota: Numpy tiene np.linalg que implementa operaciones de álgebra lineal como:
# descomposición en valores singulares, descomposición en eigenvalores y eigen vectores
# solución de sistemas de ecuaciones, inversas y pseudoinversas de matrices, etc.
# Sin embarga es más aconsejable usar scipy.linalg que usará algoritmos más edicientes
```

Operaciones de tipo resumen: resumen todos los valores en uno solo. Por ejemplo, máximos, mínimos, medias, sumas, etc. Es más rápido que usar operaciones equivalentes con código Python.

```
a = np.array([1, 2, 3, 4])                # array 1d
print("suma:",a.sum(),"media:", a.mean(), "Desviación:", a.std(), "mediana:", a.median())
print("Valor mínimo:", a.min(), "Posición del mínimo:", a.argmin())
a = a.reshape((2,2))                      # Estructura 2d de 2x2: [[1,2],[3,4]]
print("Suma por columnas:", a.sum(axis=0))       # [4, 6]
print("Suma por filas:   ", a.sum(axis=1))       # [3, 7]
```

```
a = np.array([1, 2, 3, 4])
b = a + 1
distancia = np.sqrt( np.sum((a - b) ** 2))
print("vector a:", a, "vector b:", b, "distancia:", distancia)
print("Contar mayores de 2:", (a > 2).sum() )
print("Alguno mayor de 2:  ", (a > 2).any() )
print("Todos mayores de 2:", (a > 2).all() )
```

BROADCASTING

Es una operación que permite transformar el tamaño de los arrays para poder realizar operaciones elemento a elemento que de otra forma fallarían. Las reglas son (aparecen representadas en la figura) que comenzando con los ejes de cada array, Numpy compara las dimensiones de los arrays y:

- Si las dimensiones son iguales continúa.
- Si una de las dimensiones tiene menos valores que la otra, los hace coincidir con el de mayor cantidad.
- Si no coinciden en la cantidad de dimensiones, irá haciendo coincidir los valores de cada dimensión copiando los valores originales.

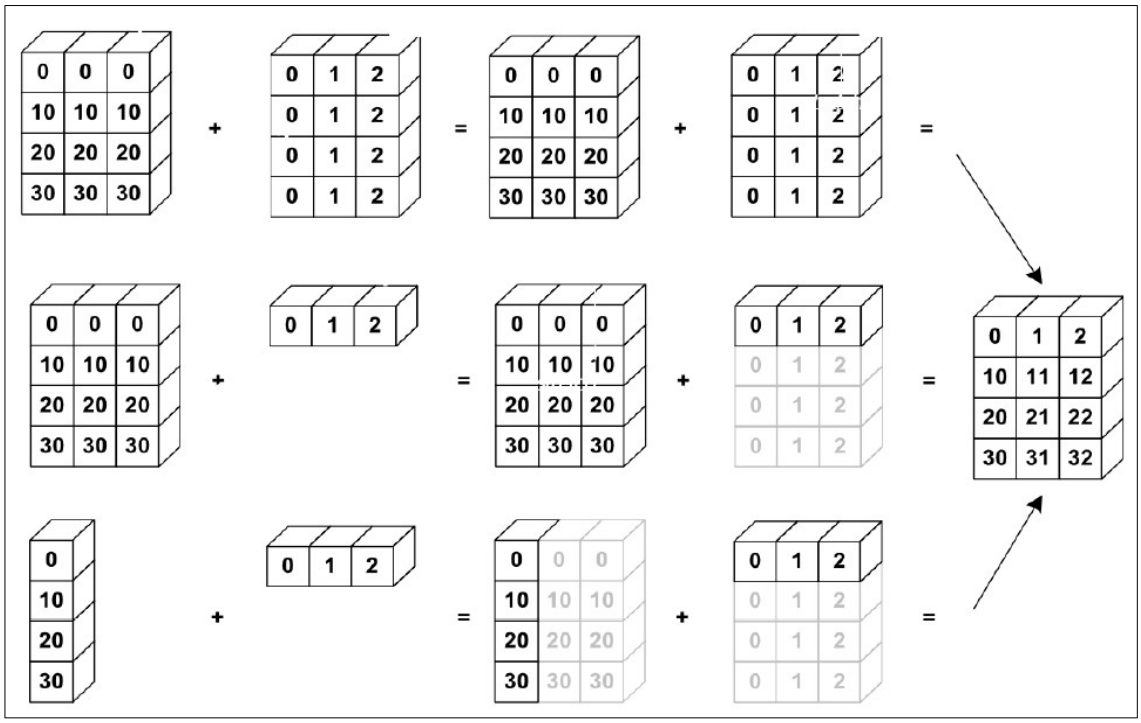


Figura 40: Broadcasting al realizar operaciones.

```
a = np.array([0, 1])
b = np.array([[0, 1], [2,3]])
print("a:", a, "b: [", b[0], ", ", b[1], "] a + b:", a + b)
```

Para profundizar en el manejo de numpy repasa el documento:

https://lectures.scientific-python.org/

5.3. INTRODUCCIÓN A MATPLOTLIB.

Sources - Nicolas P. Rougier: http://www.labri.fr/perso/nrougier/teaching/matplotlib - https://www.kaggle.com/benhamner/d/uciml/iris/python-data-visualizations

DIBUJOS BÁSICOS

```
import numpy as np
```

```
import matplotlib.pyplot as plt
%matplotlib inline          # inline plot (for jupyter)

x = np.linspace(0, 10, 50)
senos = np.sin(x)
plt.plot(x, senos)
plt.show()
```

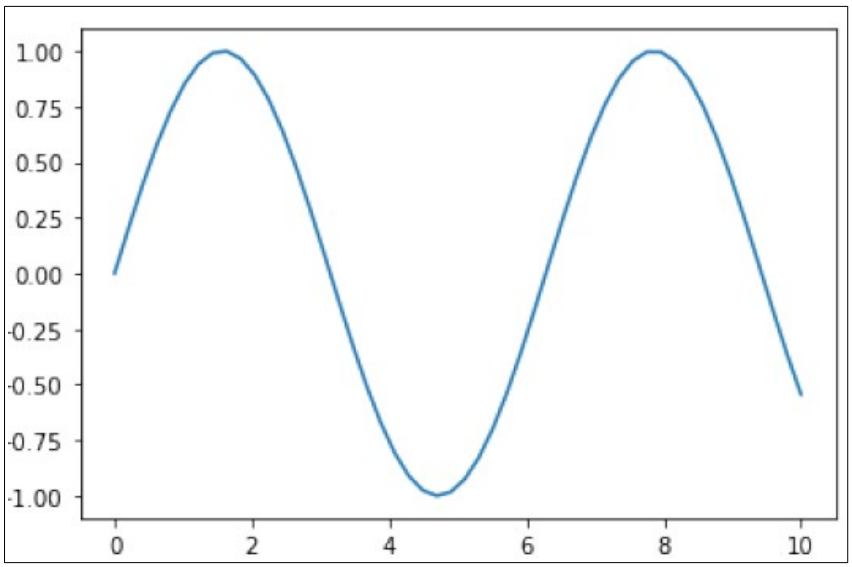


Figura 41: Gráfico de una función `plt.plot(x, f(x))`.

```
# Indicando una figura en vez de una línea recta
x = np.linspace(0, 10, 50)
senos = np.sin(x)
plt.plot(x, senos, "o")      # Si no se indica por defecto es "-" una línea
plt.show()
```

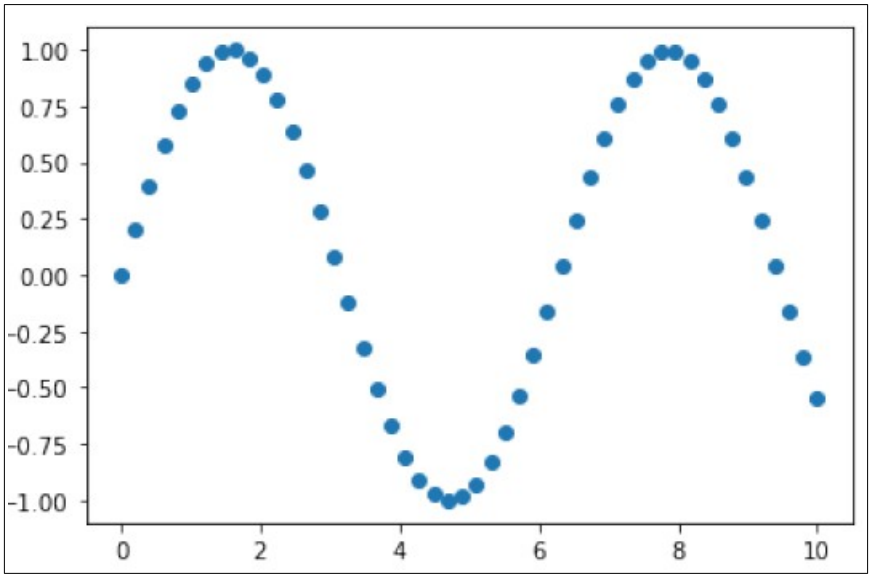


Figura 42: Gráfico de una función `plt.plot(x, f(x), "dibujo")`.

```
# Indicando línea y figura y dibujando 4 gráficas indicando "dibujo y color"
x = np.linspace(0, 10, 50)
senos = np.sin(x)
cosenos = np.cos(x)
plt.plot(x, senos, "-b", x, senos, "ob", x, cosenos, "-r", x, cosenos, "or")
plt.xlabel("Este es el eje x!")     # Etiqueta del eje X
plt.ylabel("Este es el eje y!")     # Etiqueta del eje Y
plt.title("Mis primeras gráficas")  # Título
```

```
plt.show()
```

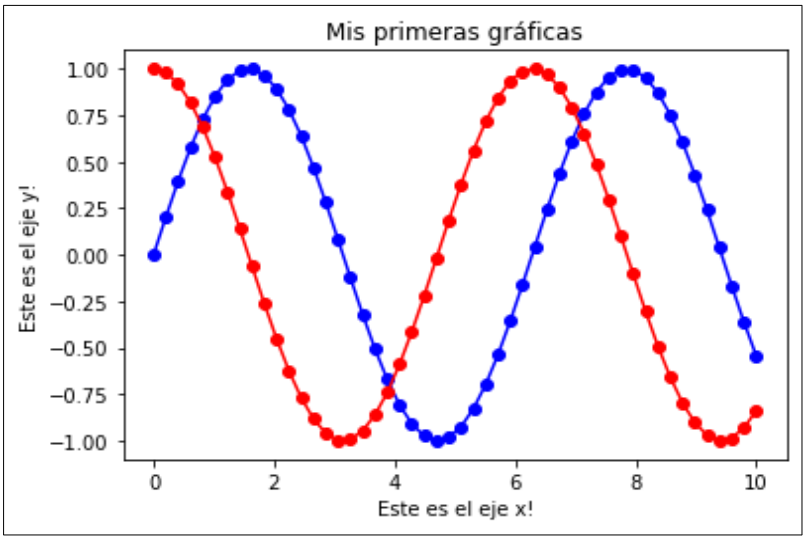


Figura 43: Gráfico de varias funciones `plt.plot(x, f(x), "dibujo color", ...)`.

```
# Parecido al anterior pero indicando el nombre de cada cosa de manera individual
x = np.linspace(0, 10, 50)
senos = np.sin(x)
cosenos = np.cos(x)
plt.plot(x, senos, label = "seno", color="blue", linestyle = "--", linewidth = "2")
plt.plot(x, cosenos, label = "coseno", color="red", linestyle = "-", linewidth = "1")
plt.legend()                          # leyenda
plt.show()
```

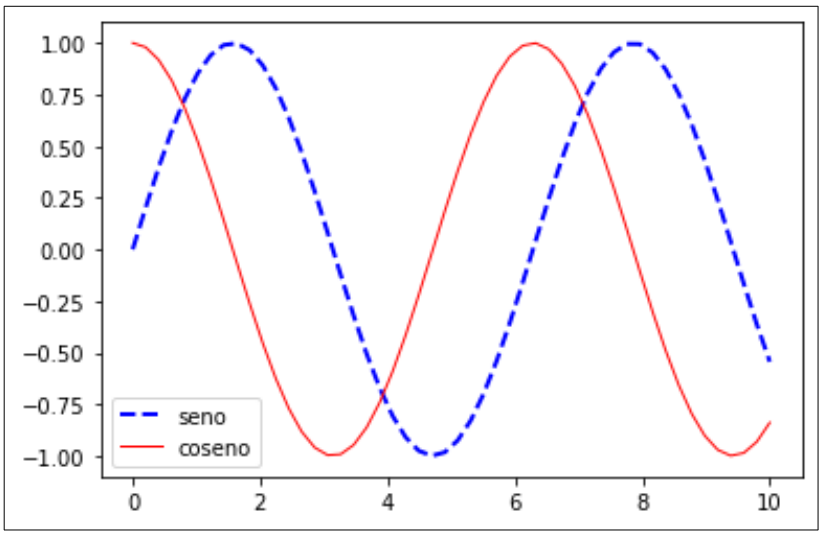


Figura 44: Gráfico de varias funciones con varias llamadas a `plt.plot()` y `nombres de parámetros`.

GRÁFICOS DE NUBES DE PUNTOS (SCATTER PLOTS) EN 2D

```
# importamos un dataset con pandas (si no tenemos el dataset lo descarga con una URL)
import pandas as pd
try:
    salarios = pd.read_csv("salary_table.csv")
except:
    url = "https://media.githubusercontent.com/media/" +
          "neurospin/pystatsml/master/datasets/salary_table.csv"
    salarios = pd.read_csv(url)
df = salarios
```

```
# Gráfico nube puntos con colores asociados a valores del dataset con un diccionario
colores = colores_edu = {'Bachelor':'r', 'Master':'g', 'Ph.D':'blue'}
plt.scatter(df['experience'], df['salary'],
            c=df['education'].apply(lambda x: colors[x]), s=100)
```

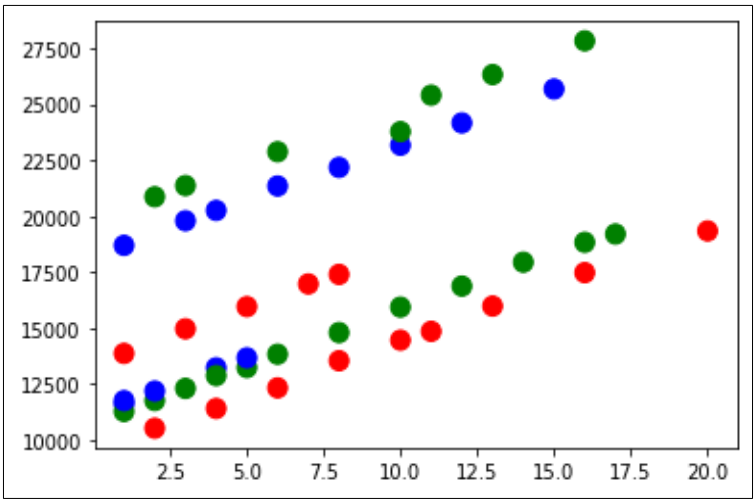


Figura 45: Nube de puntos de experiencia contra salario dando colores según educación.

```
# Gráfico nube puntos con colores y símbolos haciendo grupos por educación + manegement
plt.figure( figsize = (6,5) )                    ## Tamaño de la Figura
simbolos = dict(Y='*', N='.')      ## Define colores / símboloss manualmente
color_edu = {'Bachelor':'r', 'Master':'g', 'Ph.D':'b'}
## Agrupar por: education + management => 6 grupos
for valores, d in salary.groupby(['education','management']):
  edu, manager = valores
  plt.scatter(d['experience'], d['salary'],
       marker = simbolos[manager], color=colores_edu[edu], s=150, label=manager+"/"+edu)
## Fijar etiquetas
plt.xlabel('Experiencia')
plt.ylabel('Salario')
plt.legend(loc=4)      # Posición Inferior derecha
plt.show()
```

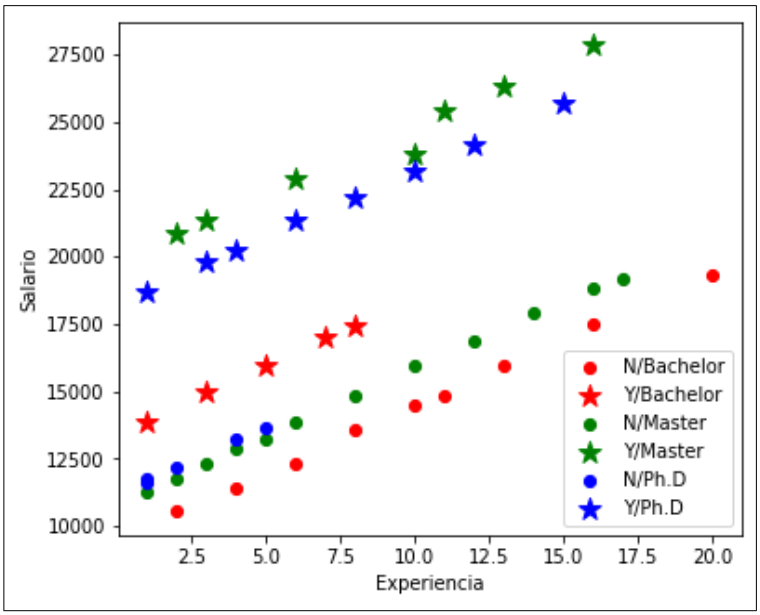


Figura 46: Nube de puntos de 6 grupos agrupando por educación: experiencia x salario.

GUARDAR FIGURAS

```
plt.plot(x, senos)
plt.savefig("senos.png")     # Formato mapa de bits
plt.savefig("senos.svg")     # Vectorial editable con Inkscape, Adobe Illustrator, etc.
plt.plot(x, sinus)           # Formato pdf
plt.savefig("senos.pdf")
plt.close()
```

DIAGRAMAS BOXPLOT (O DE BIGOTES).

Los boxplot son gráficos no paramétricos: muestran la variación de los ejemplos de una población estadística sin hacer ninguna suposición sobre su distribución.

Consisten en una caja de anchura la del ***IQR***. El ***IQR*** es ***el índice intercuartílico***, es decir, el tercer cuartil (Q3) menos el primero (Q1). La caja tiene marcada donde se encuentra la mediana en su interior. Además, tiene dos líneas donde indican donde está la mayoría de datos. Si hay datos más allá de las líneas, pueden considerarse *outliers* (datos anómalos).

Puedes utilizar la función **`boxplot()`**. En los ejemplos generamos una variable X que se basa en la distribución normal.

```
import numpy as np
import matplotlib.pyplot as plt
np.random.seed(5)                    # Fijar semilla para generar mismos valores aleatorios
x = np.random.normal(0, 1, 200)      # Simular los datos
fig, ax = plt.subplots()
ax.boxplot(x)                        # ax.boxplot(x, vert=False) para horizontal
plt.show()
```

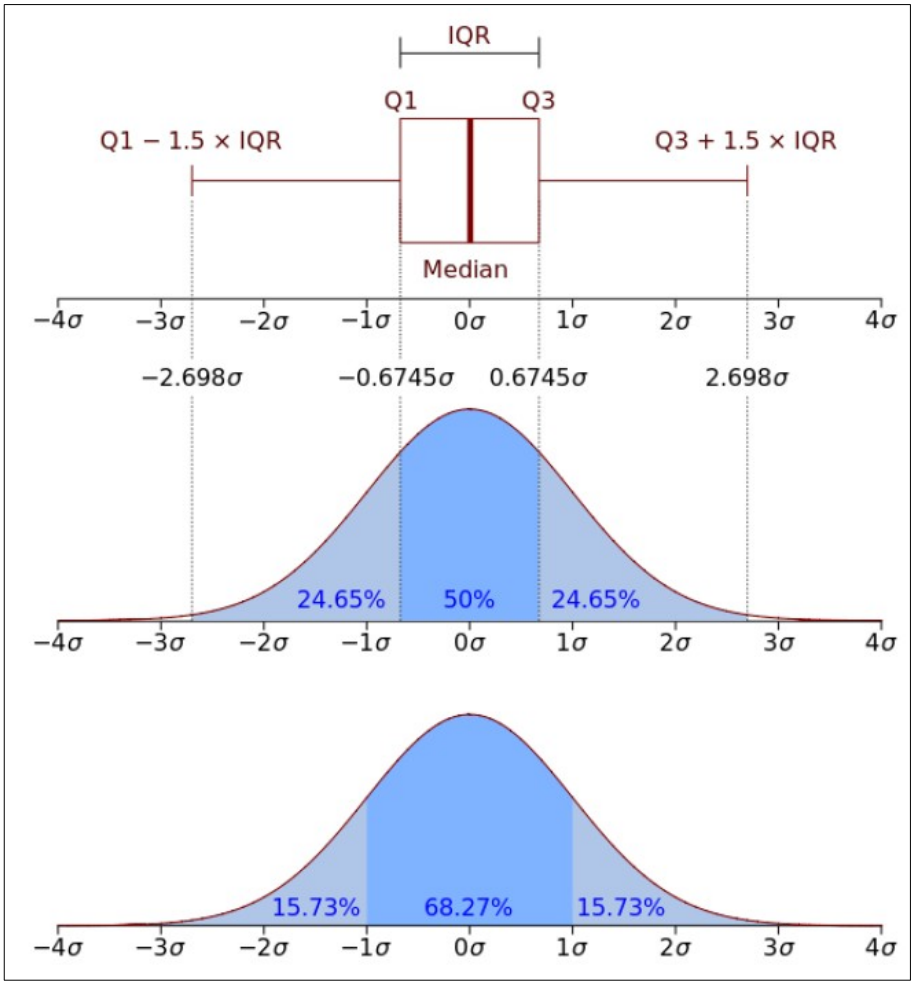


Figura 47: Representar datos con gráfico boxplot y su equivalencias con distribución Normal.

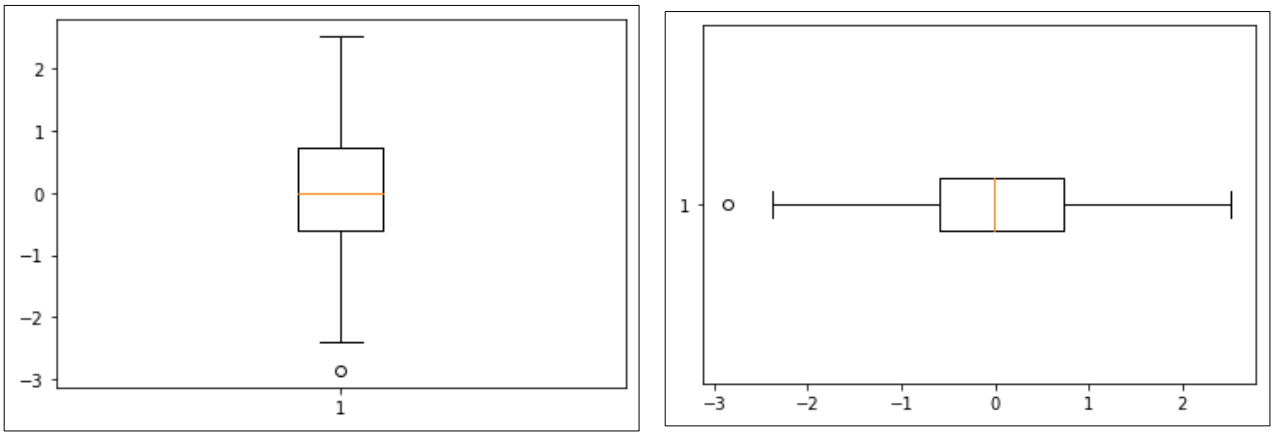


Figura 48: Un diagrama boxplot vertical y su equivalente horizontal.

Para hacerlo horizontal, hay que pasar el parámetro **`vert`** a `False`. Para profundizar en el uso de estos diagramas visita la página siguiente: enlace.

5.4. SEABORN

Seaborn es otra librería de gráficos adicional a matplotlib. Mientras que matplotlib se orienta a permitir personalizar los gráficos que genera y por tanto es de bajo nivel, seaborn se orienta a ofrecer gráficos predefinidos de más alto nivel, sobre todo orientados a estadística. **Enlaces:**

- `http://stanford.edu/~mwaskom/software/seaborn`
- `https://elitedatascience.com/python-seaborn-tutorial`

Si necesitas instalarlo: `pip install -U --user seaborn`
Si estás usando *spyder*, quizás debas instalar antes *pip*. Esto lo puedes conseguir descargando el fichero **`get-pip.py`**. Ir a la carpeta donde está el python que está usando el IDE y ejecutarlo (`python.exe get-pip.py`). Mira: `https://stackoverflow.com/questions/72804579/how-to-get-pip-for-spyder`

```
# Gráfico que posicione en el eje Y el salario y dibuje un plotbox de cada tipo de
# educación, comparando la experiencia
import seaborn as sns
sns.boxplot(x="education", y="salary", hue="management", data=salario)
```

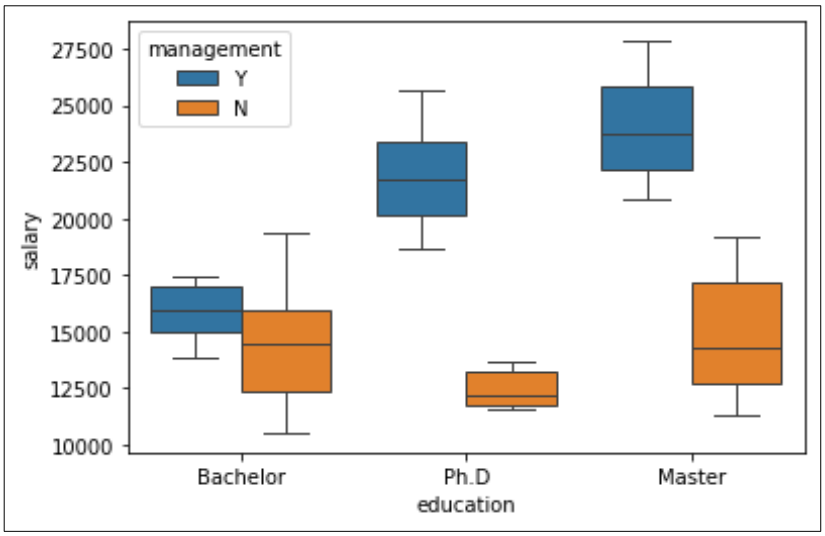


Figura 49: Representar boxplots del salario de cada tipo de educación.

```
# Gráfico que posicione en el eje Y el salario y dibuje un plotbox de cada tipo de
# educación, comparando la experiencia
import seaborn as sns
sns.boxplot(x="management", y="salary", hue="education", data=salary)
```

```
sns.stripplot(x="management", y="salary", hue="education", data=salary,
              jitter=True, dodge=True, linewidth=1) # Muestra los puntos de datos
```

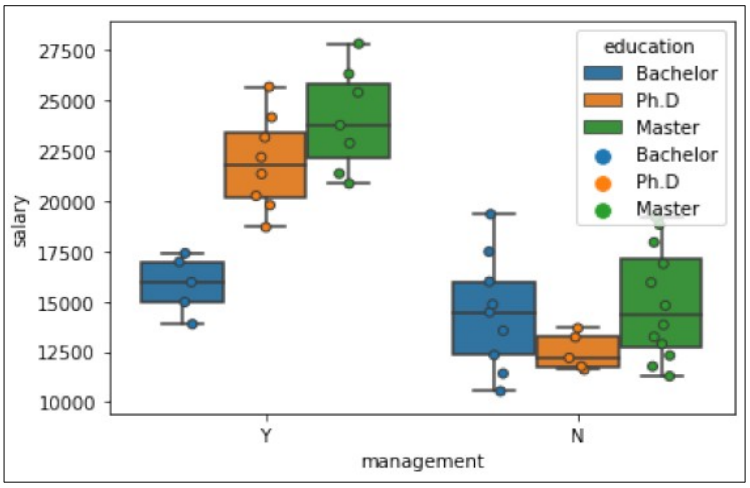


Figura 50: Añadir puntos de datos al gráfico anterior.

Diagramas de Densidad o Histogramas

Una figura que muestra como se distribuyen los datos. Pueden tener varios ejes, grupos, etc.

```
# Gráfico que posiciona en el eje Y el salario y dibuja un histograma de cada tipo de
# educación, comparando la experiencia en cada grupo. Cada grupo en un eje vertical.
f, ejes = plt.subplots(3, 1, figsize=(9, 9), sharex=True)       # Configurar 3 x 1 ejes
i = 0
for edu, d in salarios.groupby(['education']):
    sns.distplot(d.salary[d.management=="Y"], color="b", bins=10, label="Directivo", ax = ejes[i])
    sns.distplot(d.salary[d.management == "N"], color="r", bins=10, label="Empleado",ax = ejes[i])
    ejes[i].set_title(edu)
    ejes[i].set_ylabel('Densidad')
    i += 1
ax = plt.legend()
```

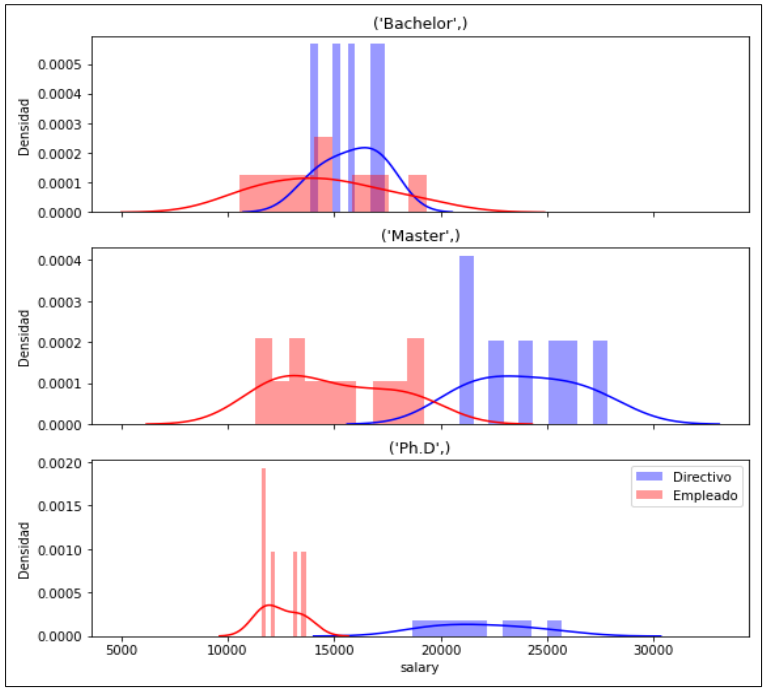


Figura 51: Gráfico con varios histogramas que permiten comparar varios grupos de datos.

Diagramas de violín

Combina un diagrama *boxplot* con un diagrama de densidad girado a ambos lados de una línea central. La parte destacada de esta línea representa el ***IQR***. El punto blanco es la mediana y los bigotes (la parte delgada de la línea) es el 95% de los intervalos de confianza de los datos.

```
ax = sns.violinplot(x="salary", data=salarios)
```

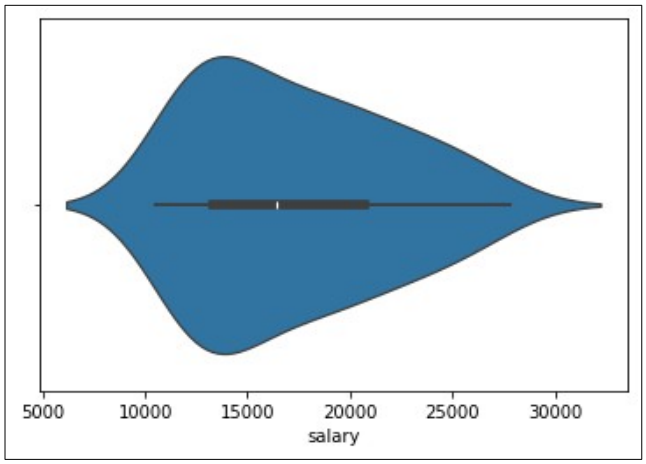


Figura 52: Diagrama de Violín.

```
# Ajustando el ancho de banda
ax = sns.violinplot(x="salary", data=salarios, bw=.15)
```

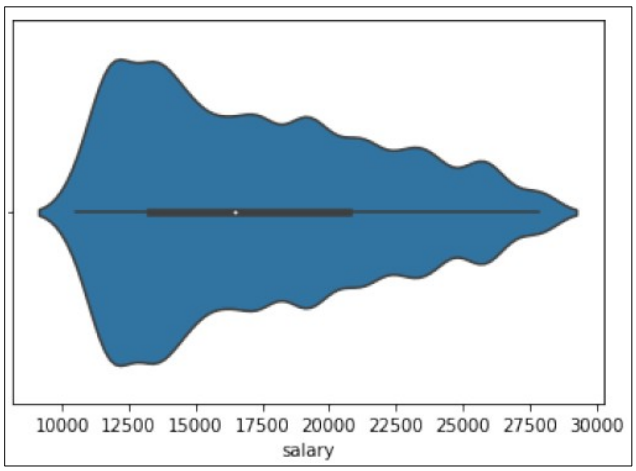


Figura 53: Diagrama de Violín con ancho de banda ajustado.

```
# Algo equivalente a los anteriores con boxplot y distribuciones
ax = sns.violinplot(x="management", y="salary", hue="education", data=salarios)
```

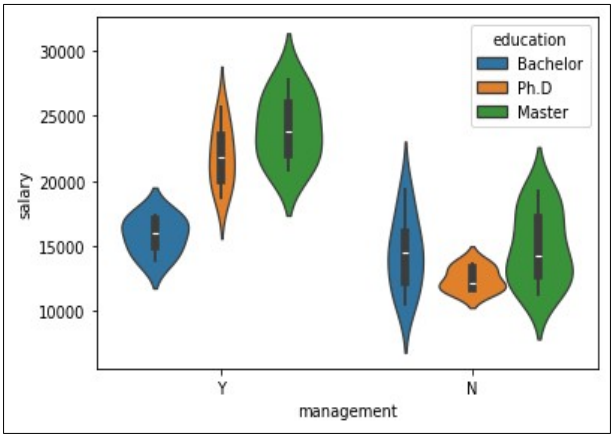


Figura 54: Diagramas de Violín por directivos o no.

El dataset tips recoge información sobre los pedidos recibidos durante unos cuantos meses en un restaurante. Tenemos diferentes variables de las que vemos algunos ejemplos con el siguiente código:

```
import seaborn as sns
# Cargarlo usandolo como uno de los datasets de ejemplo de seaborn
tips = sns.load_dataset("tips")
# Cargarlo usando pandas desde un archivo externo
# import pandas as pd
# tips = pd.read_csv("tips.csv")
print(tips.head())
ax = sns.violinplot(x=tips["total_bill"])
```

Los datos mostrados son los siguientes:

	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4

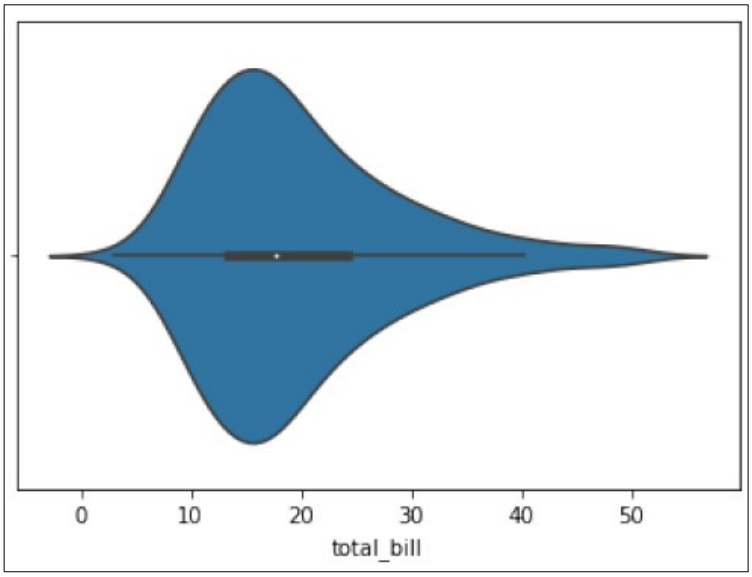


Figura 55: Diagramas de Violín agrupados por días.

```
# Haciendo grupos por días
ax = sns.violinplot(x="day", y="total_bill", data=tips, palette="muted")
```

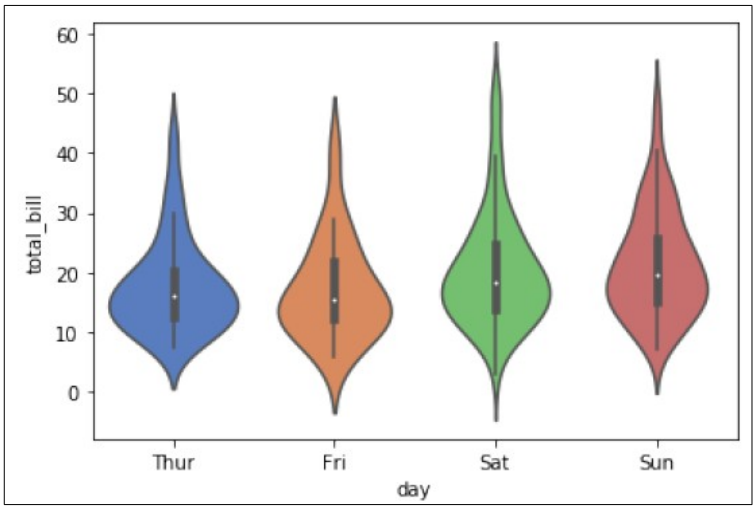


Figura 56: Diagramas de Violín agrupados por días por directivos o no.

```
# Haciendo grupos por días y separando Lunch y Dinner
ax = sns.violinplot(x="day", y="total_bill", hue="time", data=tips, palette="muted",
split=True)
```

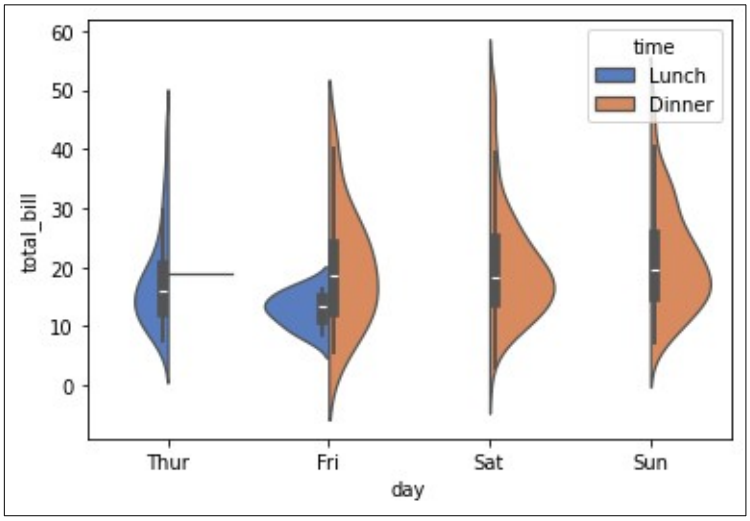


Figura 57: Diagramas de Violín agrupados por días por directivos o no.

DIAGRAMAS DE NUBES DE PUNTOS POR PAREJAS

```
# Haciendo grupos por días y separando Lunch y Dinner
g = sns.PairGrid(salary, hue="management")
g.map_diag(plt.hist)
g.map_offdiag(plt.scatter)
ax = g.add_legend()
```

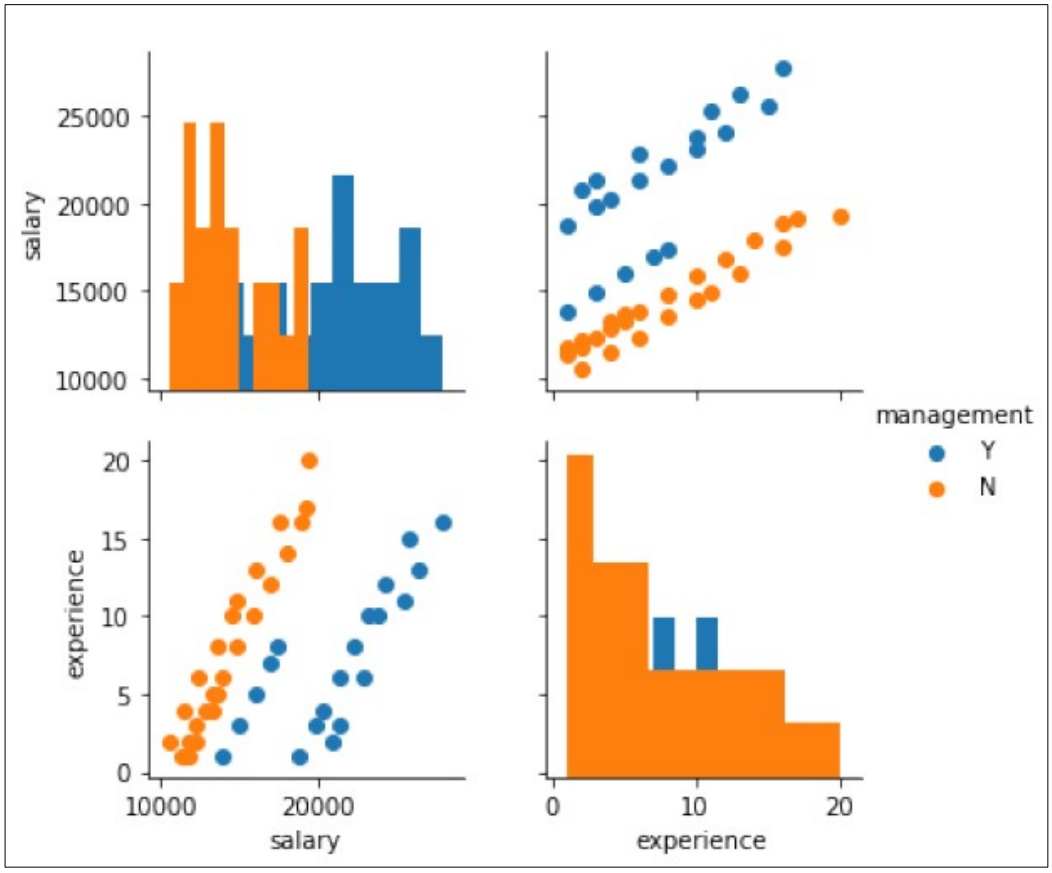


Figura 58: Diagramas por parejas.

SERIES TEMPORALES

```
sns.set(style="darkgrid")
fmri = sns.load_dataset("fmri")
ax = sns.pointplot(x="timepoint", y="signal", hue="region", data=fmri)
```

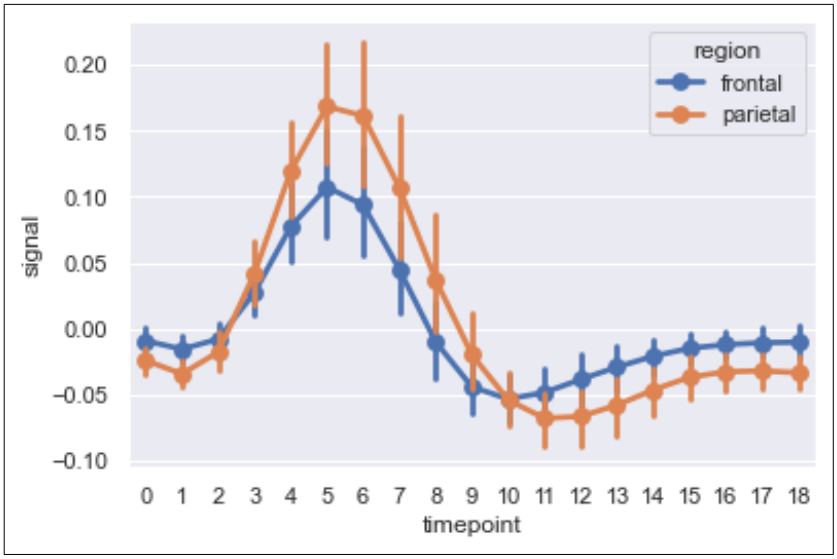


Figura 59: Diagramas de una serie.

5.6. INTRODUCCIÓN A PANDAS.

Es un paquete open source que aporta estructuras de datos flexibles, potentes y eficientes de trabajar con datos relacionales o etiquetados de manera sencilla e intuitiva. Fue desarrollado por

Wes McKinney en 2008 para ***AQR Capital Management*** y sus necesidades de una herramienta flexible y de alto rendimiento para el análisis de datos financieros. Antes de dejar *AQR* pudo convencer a los responsables para que la dejaran como una librería open source. Pandas trata datos como si estuviesen organizados de manera tabular como una tabla ***SQL*** o una hoja de cálculo.

ESTRUCTURAS DE DATOS
Pandas introduce dos nuevas estructuras de datos a *Python Series* y *DataFrame*, ambas basadas en ***NumPy***.

Series
Similar a una columna de una hoja de cálculo o una tabla SQL. Por defecto a cada elemento de la serie se le asigna un índice entero desde 0 a N.

```
# crear una serie pasando una lista de valores e indicando un índice personalizado
# Los índices etiquetados que referencian cada fila pueden tener duplicados
s = pd.Series([1,2,3,np.nan,5,6], index=['A','B','C','D','E','F'])
print(s)
```

DataFrame
Es un objeto bidimensional similar a una tabla *SQL* o a una hoja de cálculo. Es el objeto más usado de Pandas.

```
datos = {'Genero': ['F', 'M', 'M'],'Emp_ID': ['E01', 'E02','E03'], 'Edad': [25, 27, 25]}
# Creamos el DataFrame a parir del diccionario especificando las columnas como parámetros
df = pd.DataFrame(datos, columns=['Emp_ID','Genero', 'Edad'])
print(df)
```

LEYENDO Y ESCRIBIENDO DATOS
Hay 3 formatos de ficheros que se usan frecuentemente para almacenar en ficheros los datasets: csv, ficheros de texto *.txt* y ficheros de *Excel*. La escritura por defecto siempre sobrescribe el contenido anterior.

```
# Leyendo datos de ficheros csv
import tempfile, os.path
df= pd.read_csv('Datos/mtcars.csv')                # desde csv
df= pd.read_csv('Datos/mtcars.txt', sep='\t')      # desde fichero de texto
url = 'https://raw.github.com/neurospin/pystatsml/master/datasets/salary_table.csv'
df = pd.read_csv(url)
df= pd.read_excel('Datos/mtcars.xlsx','Hoja2')     # desde Excel
# Leyendo varias hojas del mismo fichero Excel en diferentes dataframes
xls = pd.ExcelFile('fichero.xls')
hoja1 = pd.read_excel(xls, 'Sheet1')
hoja2 = pd.read_excel(xls, 'Sheet2')
# Escribiendo
# index = False indica que no se escriban los valores de índices
df.to_csv('Datos/mtcars_new.csv', index=False)
df.to_csv('Datos/mtcars_new.txt', sep='\t', index=False)
df.to_excel('Datos/mtcars_new.xlsx',sheet_name='Sheet1', index = False)
```

RESUMEN ESTADÍSTICO DE LOS DATOS
Pandas tiene funciones predefinidas para realizar estadística básica como contar los valores, el mínimo valor, la media, la desviación, primer, segundo y tercer cuartil, etc. Puedes usar el método **`describe()`** para consultar todos estos parámetros de cada columna.

```
df = pd.read_csv('datos/iris.csv')
df.describe()
```

El método **`cov()`** devuelve la covarianza entre cada dos variables numéricas del dataset. Una covarianza positiva indica que las dos variables están correlacionas de manera directa (si una crece la otra también). Mientras que una covarianza negativa indica una relación inversa (cuando una crece, la otra decrece). Más allá de esto, no nos indica la cantidad de relación entre las dos.

```
df = pd.read_csv('datos/iris.csv')
print(df.cov())
```

El método **corr()** es la forma de saber cuánta correlación hay entre dos variables. Además de saber si la correlación es directa o inversa , también te indica el nivel de correlación. Si una variable está correlacionada con otra significa que cuando la primera cambia de valor, este cambio influye en el valor de la segunda. La correlación siempre está entre -1 y +1. Se puede interpretar como un porcentaje siendo -1 una correlación total inversa, el +1 una correlación directa del 100% y 0 que no hay correlación alguna.

```
df = pd.read_csv('datos/iris.csv')
print(df.corr())
```

VISUALIZANDO LOS DATOS

La siguiente tabla resume los métodos de visualización.

DESCRIPCIÓN	SINTAXIS
Mostrar los n primeros datos (por defecto 5)	df.head(n=2)
Mostrar los n últimos datos (por defecto 5)	df.tail()
Obtener los nombres de las columnas	df.columns
Obtener los tipos de datos de las columnas	df.dtypes
Obtener los índices de las filas	df.index
Obtener los valores únicos de cada columna	df[nombre_columna].unique()
Obtener los valores	df.values
Ordenar los valores por columnas	df.sort_values(by=['c1','c2'],ascending=[True, True])
Seleccionar/ver por nombre de columna	df[columna]
Seleccionar/ver filas por índices	df[0:3]
Seleccionar/ver filas por índices	df.loc[0:3]
Filas por índices pero solo algunas columnas	df.loc[0:3,['c1', 'c2']]
Filas y/o columnas por índices o rangos	df.iloc[2,3,6] df.iloc[0:2,0:2]
Seleccionar sin indicar el primer índice	df.ix[1,1] df.ix[:,2]
Obtener los valores escalares más eficiente	df.iat[1,1]
Transponer el dataframe	df.T
Filtrar por valores de una columna	df[df['c2'] > 7.5]
Filtrar por ser uno de los valores	df[df['c1'].isin(['condición_v1','condición_v2'])]
Filtrar por varias condiciones con AND	df[(df['c1']>7.5) & (df['c2']>3)]
Filtrar por varias condiciones con OR	df[(df['c1']>7.5) & (df['c2']>3)]

OPERACIONES BÁSICAS DE MANIPULACIÓN

Algunas operaciones básicas sobre los datos aparecen en la siguiente tabla.

DESCRIPCIÓN	SINTAXIS
Convertir strings en una serie de fechas	pd.to_datetime(pd.Series(['2017-04-01', '2017-04-02']))
Renombrar una columna	df.rename(columns={'anterior':'nuevo'}, inplace=True)
Renombrar todas las columnas del DataFrame	df.columns = ['nuevo_c1', 'nuevo_c2' ...]
Marcas de duplicados	df.duplicated()
Borrar duplicados	df = df.drop_duplicates()
Borrar duplicados de ciertas columnas	df.drop_duplicates(['c1'])

DESCRIPCIÓN	SINTAXIS
Borrar duplicados de columnas obteniendo en un set las primeras o últimas repeticiones	`df.drop_duplicates(['c1'], keep='first') # tb. last`
Crear nueva columna a partir de otra	`df['nueva'] = df['existente'] + 5`
Crear nueva columna desde otras existentes	`df['nueva'] = df['c1'] + '_' + df['c2']`
Añadir una lista como nueva columna	`df['nueva'] = pd.Series(lista)`
Borrar filas con valores ausentes	`df.dropna()`
Remplazar valores ausentes con 0 (o int/str)	`df.fillna(value=0)`
Rmplazar con la última observación. Hay dos formas: 1) pad / ffill – hacia adelante 2) bfill / backfill – hacia atrás	`df.fillna(method='ffill', inplace=True, limit = 1)` inplace indica si los cambios se realizan sobre el propio objeto.
Comprobar valores ausentes	`pd.isnull(df)`
Remplazar ausentes en columna por su media	`media=df['c1'].mean()` `df['c1'].fillna(media)`
Devolver media de cada columna	`df.mean()`
Devolver máximo de cada columna	`df.max()`
Devolver mínimo de cada columna	`df.min()`
Devolver suma de cada columna	`df.sum()`
Devolver cuenta de valores de cada columna	`df.count()`
Devolver suma acumulada de cada columna	`df.cumsum()`
Aplicar una función sobre un eje	`df.apply(np.cumsum)`
Operar iterando sobre elementos	`df['c1'].map(lambda x: 1+x)`
Operar con función lambda predefinida	`funcion = lambda x: x + 1` `df.applymap(funcion)`

OPERACIONES DE MEZCLA DE DATAFRAMES

Pandas aporta utilidades para combinar varios objetos *Serie, DataFrame* y *Panel* usando diferentes tipos de operadores como los del álgebra relacional. En los ejemplos usamos estos dos *dataframes*:

```
datos = {
'emp_id': ['1', '2', '3', '4', '5'],
'nombre': ['Juan', 'Andrés', 'Alejandro', 'Alicia', 'Ana'],
'apellido': ['Loreto', 'García', 'González', 'Rosa', 'Santos']}
df_1 = pd.DataFrame(datos, columns = ['emp_id', 'nombre', 'apellido'])
datos = {
'emp_id': ['4', '5', '6', '7'],
'nombre': ['Benito', 'Sara', 'Kim', 'Jose'],
'apellido': ['Alexandre', 'Cenador', 'Mancha', 'Garrido']}
df_2 = pd.DataFrame(datos, columns = ['emp_id', 'nombre', 'apellido'])
```

Ampliar un *DataFrame* con otro usando `df3 = pd.concat(df1, df2)` o `df.append(df)`

```
df = pd.concat([df_1, df_2])
print(df)
# Unir los dos dataframes por columnas
print( pd.concat([df_1, df_2], axis=1) )
```

Mezclar dos *DataFrames* usando `df3 = pd.merge(df1, df2, columna_on )`

```
# Mezclar df_1 y df_2 por la columna emp_id
# Es como hacer un natural join por esa columna también conocido como inner join
print( pd.merge(df_1, df_2, on='emp_id') )
```

También podemos realizar operaciones *left* y *right join* o un *outer join*:

```
# left join
print( pd.merge(df_1, df_2, on='emp_id', how='left') )
# ahora además añadimos sufijos a las columnas duplicadas de ambas tablas
print( pd.merge(df_1, df_2, on='emp_id', how='left', suffixes=('_left', '_right')) )
# right join
print( pd.merge(df_1, df_2, on='emp_id', how='right') )
# outer join
print( pd.merge(df_1, df_2, on='emp_id', how='outer') )
```

OPERACIÓN DE AGRUPAMIENTO

Hacer grupos significa realizar las siguientes operaciones:

- Dividir los datos en grupos usando algún criterio
- Aplicar una función a cada grupo de manera independiente si queremos información de cada grupo.
- Combinar los resultados en una estructura de datos.

```
import numpy as np
import pandas as pd
df= pd.DataFrame({'Nombre': ['Jose', 'Ana', 'Jose', 'Ana', 'Jose', 'Ana', 'Jose'],
'Ciudad': ['Sevilla', 'Sevilla', 'Madrid', 'Cáceres', 'Madrid', 'Madrid', 'Sevilla'],
'Estudios':['A','A','B','A','C','B','C'],
'Edad': np.random.uniform(24, 50, size=7),
'Salario' : np.random.uniform(3000, 5000, size=7),})
# Las columnas están ordenadas automáticamente por orden alfabético
# Para cambiarlo: df= pd.DataFrame(datos, columns=['Nombre','Ciudad','Edad','Salario'])
print( df )
# Encuentra la edad máxima y el salario medio de cada nombre y Ciudad teniendo en cuenta
# que con groupby(), puedes usar funciones agregadas como min, max, mean, count, cumsum
print( df.groupby(['Nombre','Ciudad']).max() )
```

OPERACIONES DE PIVOTEO DE TABLAS

El método **`pivot_table()`** crea tablas de tipo pivote. Pueden pasarse los siguientes parámetros:

- `data`: objeto DataFrame.
- **`values`**: columnas a añadir
- **`index`**: etiquetas de las filas
- **`columns`**: etiquetas de las columnas
- **`aggfunc`**: función agregada a utilizar con los valores. Por defecto la media: `NumPy.mean`

OPERACIONES DE MEJORA DE DATOS

Por ejemplo podemos eliminar datos repetidos:

```
df = pd.concat([df, df[df.Edad > 30]])
print('Con repetidos:\n', df)
print("Fila duplicada?\n",df.duplicated()) # Serie booleanos: True si es = a una anterior
print('Cantidad: ', df.duplicated().sum()) # cuenta de duplicados
print('Filas:', df[df.duplicated()])       # Muestra solamente las filas duplicadas
print('Repite edad:', df.Edad.duplicated())# comprueba duplicados de una única columna
df = df.drop_duplicates()                  # Borra las filas duplicadas
```

Trabajar con valores inexistentes:

```
df = pd.DataFrame({'altura': [1,3,None,2], 'peso': [9,23,0,2]})
print("Original:\n", df)
print("describe():\n",df.describe(include='all')) # include para tenerlos en cuenta
# Buscar valores ausentes en una columna
print("Altura ausente?\n",df.altura.isnull())     # True si NaN, False en otro caso
print("Altura presente?\n",df.altura.notnull())   # Al contrario
print("Solo Filas con altura:\n",df[df.altura.notnull()])
print("Cuantas alturas faltan:",df.altura.isnull().sum())
# Buscar valores ausentes en todo el DataFrame
print("Valores ausentes:\n",df.isnull())          # DataFrame de nooleanos
print("Ausentes por columna:\n",df.isnull().sum())
```

```
# Estrategia 1: borrar filas con valores ausentes
df.dropna()                                     # borra fila si hay algún ausente
df.dropna(how='all')                            # borra fila si todos están ausentes
# Estrategia 2: rellenar ausentes con la media
print("Media de altura:", df.altura.mean())
df.loc[df.altura.isnull(), "altura"] = df["altura"].mean()
print(df)
```

Trabajar con valores anómalos (outliers) de manera básica. Primero generamos datos:

```
serie = pd.Series(np.random.normal(loc=175, size=20, scale=10))
serie[:3] += 500                    # Corrompemos 3 primeros datos
print(serie)
```

Si usamos un método estadístico paramétrico (la media): Asumiendo que nuestros datos siguen una distribución normal consideramos que un dato es anómalo si está más alejado de la media que 3 veces la desviación estándar. El motivo es que la teoría de la probabilidad nos indica que dentro de esta distancia estarán el 99.73% (68.27 + 2 * 15.73) de los datos

```
s = serie.copy()
print("Media:", serie.mean(),"desviación:", serie.std())
print("outliers:\n", s[((serie - serie.mean()).abs() > 3 * serie.std())])
s[((serie - serie.mean()).abs() > 3 * serie.std())] = serie.mean()
print("Media sin outliers:",s.mean())
```

Si usamos un método estadístico no paramétrico (la mediana): La mediana de la desviación absoluta (mad) es otro método robusto para detectar outliers:

```
s = serie.copy()
mad = 1.4826 * np.median(np.abs(s - s.median()))
print("mad:",mad, "Outliers:\n", s[(serie - serie.median()).abs() > 3 * mad])
s[((serie - serie.median()).abs() > 3 * mad)] = serie.median()
print("Media:",s.mean(), "mediana:",s.median())
```

OPERACIONES CON BASES DE DATOS

Por ejemplo podemos eliminar datos repetidos:

```
import pandas as pd
import sqlite3
db_fichero = os.path.join(tmpdir, "users.db")
conn = sqlite3.connect(db_fichero)
s = pd.read_csv(salarios)
s.to_sql("salario", conn, if_exists="replace")
cur = conn.cursor()
values = (100, 14000, 5, 'Bachelor', 'N')
cur.execute("insert into salary values (?, ?, ?, ?, ?)", values)
conn.commit()
s_sql = pd.read_sql_query("select * from salarios;", conn)
print(s_sql.head())
pd.read_sql_query("select * from salarios;", conn).tail()
pd.read_sql_query('select * from salarios where salario > 25000;', conn)
pd.read_sql_query('select * from salarios where experience = 16;', conn)
pd.read_sql_query('select * from salarios where education="Master";', conn)
```

5.7. REPASO DE ESTADÍSTICA

Solamente si fuese necesario...

5.8. REPASO DE CÁLCULO DIFERENCIAL.

Solamente si fuese necesario…

5.9. REPASO DE ÁLGEBRA LINEAL.

Solamente si fuese necesario…

6. RIESGOS DE LA IA.

Las tecnologías potentes han cambiado el mundo: la electricidad, el motor de combustión, el transistor o Internet. Y ahora llega la IA.Pero los científicos e ingenieros no siempre son optimistas sobre su funcionamiento porque contienen una serie de defectos como:

- **Bias o prejuicios:** si entrenamos un sistema para predecir salarios, probablemente prediga que el de una mujer es inferior al de un hombre al haber usado datos históricos para aprender. O si pedimos a un sistema que genera la imagen de un jefe, casi siembre devolverá un hombre blanco. Estos resultados se deben a un bias en los datos usados para aprender. Mirar Binns (2018) para ampliar.
- **Explicabilidad:** En el caso de sistemas complejos de deep learning que toman decisiones, es complicado indicar cómo la han tomado o en qué información se han basado. Pueden utilizar billones de parámetros y no podemos saber como los han combinado. Aunque pueden generar explicaciones locales, no es posible explicar el proceso global, ni siquiera sus creadores tienen acceso a esta información. Mirar Grennan et al. (2022) para profundizar.
- **Militarización de la IA:** Todas las tecnologías importantes han comenzado o han acabado incorporándose directa o indirectamente a la guerra. Desgraciadamente la violencia y los conflictos bélicos parecen inherentes al comportamiento humano. Por tanto es inevitable que la IA se aplique y se despliegue de manera intensiva en sistemas militares como ya está ocurriendo (Heikkilä, 2022).
- **Concentración de poder:** No es el altruismo lo que hay detrás del hecho de que las más importantes empresas del mundo están invirtiendo grandes cantidades de dinero en investigación en la IA. La realidad es que esperan obtener una gran cantidad de beneficios. La concentración de poder ocurrirá si unas cuantas organizaciones la controlan. La IA tendrá la capacidad de automatizar trabajos que actualmente solamente realizan los humanos. Esto cambiará el entorno económico, dará ventajas competitivas y su control añade un riesgo para la sociedad (ver David, 2015).
- **Riesgo existencial:** Los mayores riesgos existenciales provienen de la tecnología como armas de destrucción masiva, el cambio climático influenciado por la industrialización, nuevas pandemias favorecidas por la mejora de los sistemas de transporte y agricultura que genera una mayor concentración de población en las ciudades. Quienes controlen la IA tendrán una poderosa herramienta que pueden usar para su propio beneficio si la usan sin control (mirar Tegmark, 2018).

Esta lista de riesgos está lejos de ser exhaustiva. No hemos citado la supervivencia, la desinformación, violaciones de privacidad, fraude, manipulación de mercados financieros, el consumo de energía necesario para entrenar los modelos de manera que contribuye a la huella de carbono y al cambio climático. Hay ya recopilaciones de ejemplos de aplicaciones que usan IA y son éticamente cuestionables (consultar Dao, 2021). Además, la combinación con las redes sociales e Internet favorece la aparición y divulgación de fake news, spam, acoso online, fraude, ciberbulling, manipulación cultural, política, pornográfica, doxxing, radicalización, etc.

Quienes crean e investigan en IA deben pensar también en qué grado de usabilidad dejan herramientas a los usuarios, y para qué objetivos. Aunque las empresas priorizan los beneficios, las leyes y el personal técnico deberían controlar la aplicación y la evolución para maximizar los beneficios que aporte y minimizar los riesgos. ¿Cómo conseguirlo? Las empresas suelen constituir comités éticos para reducir los riesgos reputacionales o frenar proyectos que sean cuestionables éticamente.

El curso online en https://ethics-of-ai.mooc.fi/ puede ser un recurso introductorio a estos temas.

7. EJERCICIOS

EJERCICIO 1. Mira el vídeo de este enlace sobre la diferencia entre algoritmos de aprendizaje y modelos en Machine Learning y responde a estas preguntas:

a) ¿Qué tarea quieren que haga el sistema de Machine Learning?
b) ¿Qué datos se usan de cada cliente?
c) ¿Qué modelo se usa en el vídeo? El modelo estará formado por unos parámetros y un criterio de decisión ¿Cuales son sus parámetros? ¿Y el criterio?
d) ¿Cuál es el primer paso del algoritmo?

EJERCICIO 2. Consulta el artículo sobre un inconveniente del ML y:

a) Resume en un par de líneas el problema que presenta.
b) ¿En qué consisten las 4M que se proponen para solucionarlo?

EJERCICIO 3. Mira el vídeo de este enlace sobre cómo saber si es apropiado usar Machine Learning para resolver un problema o no es una buena idea y sería mejor usar otras alternativas:

a) Escribe la definición de ML que indican en el vídeo.

b) Escribe un árbol de decisión que esquematice cuando sería buena idea y cuando no.

> ***Nota:*** *Un árbol de decisión es un árbol binario invertido (la raíz arriba) donde cada nodo equivale a una pregunta con dos posibles respuestas de tipo SI/NO a la pregunta del nodo. y su se sigue la respuesta te lleva a una nueva pregunta o a un nodo final que indica una solución. Ejemplo:*

EJERCICIO 4. Responde a estas cuestiones:

1. ¿Cómo podrías definir con tus palabras lo que es el Machine Learning?

2. ¿Puedes nombrar 4 tipos de problemas donde haya tenido éxito?

3. ¿Qué es un *training set etiquetado*?

4. ¿Cuales son las dos tareas más comunes del aprendizaje supervisado?

5. ¿Puedes dar nombre a 4 algoritmos de aprendizaje supervisado?

6. ¿Qué tipo de algoritmo de aprendizaje usarías para permitir a un robot andar por terrenos desconocidos?

7. ¿Qué tipo de algoritmo podrías usar para segmentar a tus clientes en varios grupos?

8. Si necesitas implementar un sistema de filtrado de correos electrónicos que detecte si un email es o no spam, este problema sería de aprendizaje supervisado o no supervisado?

9. ¿Qué es un sistema de aprendizaje online? ¿Cómo se llaman los que no son de este tipo? ¿Qué ventajas suele tener con respecto a los otros?

10. ¿Qué significa el término ***out-of-core learning***?

11. ¿Qué tipo de algoritmos de aprendizaje usan una medida de similitud para realizar predicciones?

12. ¿Qué diferencia hay entre los parámetros de un modelo y los hiperparámetros de los algoritmos de aprendizaje? Indica un ejemplo de cada uno.

13. ¿Qué busca un algoritmo de aprendizaje basado en modelo? ¿Cuál es el enfoque más común que utilizan para realizar este trabajo? ¿Cómo hacen las predicciones?

14. Indica 4 desafíos que tenga el machine learning.

15. Si tu modelo funciona bien con los datos de entrenamiento pero no generaliza bien con los nuevos datos con los que debe trabajar ¿Qué está ocurriendo? Puedes nombrar 3 posibles causas. Puedes nombrar 3 posibles soluciones.

16. ¿Qué es un test set y porqué es necesario usarlo?

17. ¿Cuál es el objetivo de utilizar un *validation set*?

18. ¿Qué puede ir mal si tuneas los hiperparámetros de un modelo usando el test set?

EJERCICIO 5. Tenemos estos datos sobre diferentes coches. Responde a estas cuestiones:

	A	B	C	D	E	F	G	H	I
1	**mpg**	**cilindros**	**desplazamiento**	**potenciaCV**	**peso**	**aceleracion**	**modelo_year**	**origen**	**nombre**
2	18	8	307	130	3504	12	70	1	chevrolet chevelle malibu
3	15	8	350	165	3693	11.5	70	1	buick skylark 320
4	18	8	318	150	3436	11	70	1	plymouth satellite
5	16	8	304	150	3433	12	70	1	amc rebel sst
6	17	8	302	140	3449	10.5	70	1	ford torino
7	15	8	429	198	4341	10	70	1	ford galaxie 500
8	14	8	454	220	4354	9	70	1	chevrolet impala
9	14	8	440	215	4312	8.5	70	1	plymouth fury iii
10	14	8	455	225	4425	10	70	1	pontiac catalina
11	15	8	390	190	3850	8.5	70	1	amc ambassador dpl
12	15	8	383	170	3563	10	70	1	dodge challenger se
13	14	8	340	160	3609	8	70	1	plymouth 'cuda 340
14	15	8	400	150	3761	9.5	70	1	chevrolet monte carlo
15	14	8	455	225	3086	10	70	1	buick estate wagon (sw)
16	24	4	113	95	2372	15	70	3	toyota corona mark ii
17	22	6	198	95	2833	15.5	70	1	plymouth duster
18	18	6	199	97	2774	15.5	70	1	amc hornet
19	21	6	200	85	2587	16	70	1	ford maverick
20	27	4	97	88	2130	14.5	70	3	datsun pl510
21	26	4	97	46	1835	20.5	70	2	volkswagen 1131 deluxe sedan

a) ¿Cuántos ejemplos hay?

b) ¿Cuántas característica tiene cada uno?

c) Si eliminamos la característica nombre, y queremos predecir el valor de la aceleración a partir del resto de características ¿Cuántas predictoras hay? ¿Cuál es la columna label? ¿Cuál es la columna *target*? ¿Qué tipo de algoritmo de aprendizaje debemos usar supervisado o no supervisado? ¿Qué tarea estamos haciendo una regresión o una clasificación?

d) Si creamos una división en *train* y test y queremos dejar el 30% de los ejemplos para el algoritmo de entrenamiento del modelo ¿Cuántos ejemplos usará este algoritmo para aprender? ¿Y cuantos para testear el modelo calculado?

EJERCICIO 6: **Mira el vídeo donde se explica en qué consiste el aprendizaje reforzado y responde a estas cuestiones**:

a) ¿El aprendizaje reforzado siempre se basa en modelos o nunca lo hace?
b) ¿En resolver qué juegos han tenido éxito los algoritmos de aprendizaje reforzado?
c) ¿A qué datos tienen acceso los algoritmos de aprendizaje reforzado libres de modelo?
d) ¿Cuáles son los dos algoritmos más importantes del aprendizaje reforzado?
e) Qué problema ayuda a solucionar el Machine Learning a estos algoritmos?

EJERCICIO 7: Copia el código del ejemplo 1 en el fichero ***q_learn.py***, realiza dos ejecuciones y responde a las preguntas:

a) Entrega capturas del resultado de las ejecuciones.
b) Si la primera fila de la matriz R representa estar en la habitación (a) ¿A qué lugares puedes moverte desde ella? ¿Qué recompensa obtiene el agente si lo hace?
c) Lo que aprende el algoritmo ¿Lo almacena en?
d) Si está en la habitación d, ¿Qué dos acciones tienen la mayor recompensa?

EJERCICIO 8: Mira el vídeo sobre ***SAA U01 EDA Análisis exploratorio de datos.mp4*** y responde a estas preguntas:

a) Enumera los pasos que definen el proceso.
b) En el paso 3 ¿Qué tipos de características nos podemos encontrar en un dataset?
c) En el paso 4 ¿Qué métricas se utilizan para describir los datos de una característica?
d) ¿Qué desventaja tiene la media y la desviación estándar para describir una característica numérica? ¿Hay alguna otra métrica que no tenga esta desventaja?
e) ¿Qué es el IQR?
f) ¿Qué es la mediana, el percentil 50 y el cuartil 2?
g) ¿Qué porcentaje de los datos están entre el cuartil 1 y el cuartil 3? ¿Y entre el percentil 25 y el percentil 75? ¿Y qué porcentaje de datos son mayores del cuartil 3?
h) Para graficar (representar gráficamente) como se distribuyen los datos de una característica ¿Qué dos tipos de gráficos se suelen utilizar y para qué tipo de características?
i) La matriz de correlaciones todas las variables numéricas de un dataset ¿Qué valores tiene en su diagonal?

EJERCICIO 9: Mira el vídeo ***SAA U01 Ingeniería de características.mp4*** y responde a estas preguntas:

a) Enumera los problemas que según el vídeo pueden tener los datos.
b) Enumera como se llaman las técnicas usadas en ingeniería de características para solucionar los problemas de los datos.
c) Indica los tipos de características categóricas podemos encontrarnos y el motivo de de querer transformarlas en información numérica. ¿Cómo se llaman las transformaciones?

EJERCICIO 10: Mira el vídeo ***SAA U01 Selección de características.mp4*** y responde a estas preguntas:

a) ¿Qué métodos utiliza la selección de características?
b) ¿Porqué puede resultar beneficioso descartar características y no usarlas para aprender?
c) El método de filtrado, ¿Necesita construir el modelo?
d) En el método de filtrado ¿Cambian las características a descartar si cambias el modelo a construir?

e) Si tenemos 10 características y queremos quedarnos con las 6 más importantes y para ello usamos el *método wrapper hacia adelante*, ¿Cuántos modelos debemos entrenar y validar para descartar las 4 peores?

EJERCICIO 11: Mira el vídeo ***SAA U01 set de datos train validación test.mp4*** y responde a estas preguntas:

a) ¿Porqué no es buena idea utilizar todos los datos para entrenar al modelo? Si lo hacemos y su comportamiento con estos datos es bueno, por ejemplo acierta el 96% de las veces ¿No es suficiente para decir que es un buen modelo? ¿O hay algo que no sabemos aún para darlo por bueno?

b) ¿Porqué nos puede interesar hacer 3 divisiones de los datos (train+validación+test) en ves de dos (train+test)?

c) ¿Qué porcentaje se suelen usar para cada partición de datos?

d) Si no tenemos muchos datos, en vez de hacer la partición en 3 datasets (train+validación+prueba) ¿Qué otros métodos se pueden utilizar?

EJERCICIO 12: Vamos a trabajar con los datos ausentes.

a) Define este *dataframe* que construiremos en código

```
# -*- coding: utf-8 -*-
import pandas as pd
from io import StringIO
import sys
csv_data = \
'''A,B,C,D
1.0,2.0,3.0,4.0
5.0,6.0,,8.0
10.0,11.0,12.0,'''
# Si estás usando 2.7, necesitas convertirlo a string unicode
if (sys.version_info < (3, 0)):
    csv_data = csv_data.unicode(csv_data)
df = pd.read_csv(StringIO(csv_data))
print(df)
```

b) En la siguiente imagen aparecen unas sentencias. Imprime el resultado de ejecutarlas y asocia la etiqueta que tiene cada una como un comentario a la derecha, con la tarea que realizan:

```
df.dropna(subset=['C']) # 1
df.dropna(how='all')    # 2
df.values               # 3
df.dropna(axis=0)       # 4
df.isnull().sum()       # 5
df.dropna(axis=1)       # 6
df.dropna(thresh=4)     # 7
```

- Cuenta el número de valores ausentes que hay en cada columna / característica.
- Devuelve los arrays Numpy que contienen los datos del *dataframe*.
- Borra las filas que contienen algún valor ausente.
- Borra las columnas que contienen algún valor ausente.
- Borra todas las filas donde todos los valores estén ausentes.
- Borra las filas que tengan 3 o menos valores porque el resto están ausentes.
- Borra una o varias columnas cuyo nombre se indica, si tienen valores ausentes.

c) Ahora en el mismo dataset, vamos a imputar la media de las columnas a los valores ausentes. Esa sentencia que usa un método de *Pandas* lo hace.

```
print("Datos transformados\n", df.fillna(df.mean()))
```

Busca y ordena las siguientes sentencias para que realicen algo equivalente.

```
print("Datos transformados\n", x)
from sklearn.impute import SimpleImputer
import numpy as np
si = SimpleImputer(missing_values=np.nan, strategy='mean')
x = si.transform(df.values)
si = si.fit(df.values)
print("Datos originales\n", df.values)
```

EJERCICIO 13: Transformar datos nominales en numéricos.

a) Define este dataset

```
import pandas as pd
df = pd.DataFrame([['verde', 'M',  10.1, 'clase2'],
                   ['rojo',  'L',  13.5, 'clase1'],
                   ['azul',  'XL', 15.3, 'clase2']])
df.columns = ['color', 'talla', 'precio', 'class']
```

b) Codifica la columna ***"class"*** para que tenga valores numéricos 0,1 para sustituir los valores **'clase1'** y **'clase2'** utilizando un diccionario que defina la transformación y otro que describa la transformación inversa. Imprime los diccionarios:

```
mapeo = {label: idx for idx, label in enumerate(np.unique(df['class']))}
mapeo_inverso = {v: k for k, v in mapeo.items()}
```

```
df['class'] = df['class'].map(mapeo)
```

La última línea es la que aplica la transformación. Aplica la transformación, imprime el *dataframe* y deshaz la transformación y vuelve a imprimir el *dataframe* para comprobar que ha ido todo bien.

c) Ahora vamos a realizar lo mismo pero usando un objeto ***LabelEncoder*** de la clase ***Scikit-Learn***. Define una transformación, la aplicas y muestra la columna modificada. Imprime el *dataframe* y verás que no ha cambiado, te falta un paso, debes sustituir la columna por sus nuevos valores. Hazlo y vuelve a imprimir el *dataframe* transformado. Ahora vuelve a deshacer la transformación y dejas al *dataframe* como al principio.

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
y = le.fit_transform(df['class'].values)
```

Debes generar una salida similar a esta:

```
Datos transformados
    color talla  precio  class
0  verde     M    10.1      1
1   rojo     L    13.5      0
2   azul    XL    15.3      1
Datos originales
    color talla  precio   class
0  verde     M    10.1  clase2
1   rojo     L    13.5  clase1
2   azul    XL    15.3  clase2
```

EJERCICIO 14: Transformar columnas nominales con más de 2 valores en numéricos usando *one-hot-Encoded*.

a) Completa las partes subrayadas de las sentencias de este código que define nuevos datos ***X*** seleccionando las columnas color, talla y precio y codifica el color con un ***LabelEncoder*** como en el ejercicio anterior y los imprime.

```
X = df[['color', 'talla', 'precio']].values
```

```
le = ____________________              # Instancia el objeto LabelEncoder
_____________________                  # Imprime los datos originales
X[:, 0] = ______________________       # Transforma la columna
print("X transformado:\n", X)          # Imprime los datos transformados
```

a) Haz lo mismo con un ***OneHotEncoder*** e imprime los datos transformados.

```
_____________________________________________
_____________________________________  # objeto OneHotEncoder
_____________________________________  # Imprime los datos originales
_____________________________________  # Transforma la columna
print("X transformado:\n", X)          # Imprime los datos transformados
```

c) Utiliza **`get_dummies( ['c1', …])`** Para obtener un nuevo *dataframe* añadiendo nuevas columnas para aplicar one-hot-encoder a las que tengan valores categóricos. Aplica a las columnas color, talla y precio.

d) Repite el apartado c), pero ahora indica que se elimine la primera columna dummy porque realmente se deduce que si no hay valores ausentes en el resto aparecerá valores todos a 0. Usa el parámetro **`drop_first = True`**.

EJERCICIO 15: Haz un programa python que pregunte por los valores a y b y dibuje la función $f(x)=ax^2+b$ en el intervalo [-5, 5] con 100 puntos. Rellena un array de 100 valores x entre **`[-5,5]`** y otro **`fx`** y dibuja.

EJERCICIO 16: Pregunta por el punto de corte (w_0) y la pendiente (w_1) de una línea recta y haz que el programa la dibuje en el intervalo [-5,5] sabiendo que su ecuación es $y = w_0 + w_1 x$.

EJERCICIO 17: Imagina que tenemos estos datos con el peso y la altura de 4 personas: **`datos=[[3, 0.4], [8, 0.6], [18, 1.2], [27, 1.4]]`**.

a) Crea un *dataframe* con los datos.

b) Dibuja un **`sccaterplot()`** con círculos rojos dejando el peso en el eje **X** y la altura en el eje Y.

c) Pregunta por el punto de corte (w_0) y la pendiente de una recta (w_1) que intente acercarse a estos 4 puntos lo máximo posible.

d) Dibuja la recta en el gráfico con color azul.

e) Calcula e imprime el cuadrado de la distancia de cada punto de datos (cuando x=peso) con el valor de la recta en ese punto.

f) Calcula la media de la suma de los cuadrados de esas diferencias.

g) Si usamos ese valor como una medida del error que cometemos al intentar que la recta represente a los puntos, es como una función de coste. Intenta crear varias rectas hasta que bajes ese error de 0.5.

h) Utiliza esta fórmula tratando a los datos como vectores y matrices para calcular los parámetros w_0 y w_1 de la mejor recta. Debes crear una columna a unos en los datos X y añadirle la columna peso. Prueba tu programa con esa recta:

$$X = \begin{vmatrix} 1 & 3 \\ 1 & 8 \\ 1 & 18 \\ 1 & 27 \end{vmatrix} \quad y = \begin{vmatrix} 0.4 \\ 0.6 \\ 1.2 \\ 1.4 \end{vmatrix} \qquad W = (X^T X)^{-1} X^T y$$

EJERCICIO 18: Utiliza los datos del fichero **`mpg.csv`** para realizar las siguientes operaciones:

a) Crear un dataframe a partir del fichero.

b) Eliminar la columna nombre.

c) Hacer una versión escalada en [0,1] de los datos con código propio.

d) Haz lo mismo con el objeto ***`MinMaxScaler`*** del paquete ***`sklearn.preprocessing.`***

EJERCICIO 19: Utiliza los datos del fichero **`mpg.csv`** para realizar las siguientes operaciones :

a) Crear un ***`DataFrame`*** eliminando la columna nombre.

b) Hacer una versión escalada normalizando los datos de las columnas con tu propio código.

c) Haz lo mismo con el objeto ***`StandardScaler`*** del paquete ***`sklearn.preprocessing.`***

d) Encontrar outliers en las columnas con el método de la mediana con tu propio código.

e) Encontrar outliers en las columnas con los ***z-scores*** con tu propio código.

f) Detectar outliers dibujando ***boxplots*** de las columnas.