

Inteligencia artificial y machine learning

Introducción a la Inteligencia Artificial: Machine Learning



Qualentum Lab

MACHINE LEARNING

Inteligencia artificial y machine learning

El machine learning (ML) es un área de la inteligencia artificial que ha revolucionado muchas industrias: hemos pasado de tener que ingeniar una solución que, dados unos datos de entrada, nos pueda ofrecer un resultado, a solo tener que dar estos datos de entrada y sus soluciones a una máquina para que idee por nosotros la solución necesaria.

Partiendo de esta base, el área de machine learning engloba muchas subáreas que resuelven distintos problemas o tareas: desde problemas de clasificación y regresión hasta de generación de imágenes, entre muchos otros.

Una vez dispongamos de un conjunto de datos relacionado con el problema a resolver, antes de empezar con el entrenamiento del algoritmo/modelo de ML, es necesario cerciorarse de que estos datos pueden ser utilizados, disponen de un formato adecuado y representan la información que esperamos de ellos: este proceso recibe el nombre de preparación de datos, preprocesamiento o *data preparation, preprocessing (prep)*.

Es una de las tareas más importantes, ya que con ella podremos asegurar su calidad, precisión y consistencia: nos permitirá evitar futuros problemas e incompatibilidades tanto a la hora de la ingesta de los datos al modelo de ML como a la de poder garantizar que los mismos son veraces, representativos y válidos para el uso que queremos darles. Aunque la preparación de datos no tiene ni debe ser siempre la misma, existen ciertos pasos o tareas que suelen seguirse de forma general.

Por lo tanto, los objetivos de aprendizaje de este tema son los siguientes:

- Conocer desde el nacimiento hasta la evolución del término machine learning, así como su definición.
- Entender cuáles son los diferentes tipos de datos que puede procesar un modelo de machine learning.
- Descubrir las diferentes áreas que conforman el ML, desde su definición hasta su aplicación en casos reales con ejemplo de algoritmos usados, respectivamente.
- Aprender en qué consisten la medición de resultados y la evaluación de un modelo de ML a través del uso de diferentes métricas, según el problema o tarea a resolver.

Y una vez aprendidos estos conceptos y procesos, seremos capaces de:

- Cargar los datos disponibles desde diferentes formatos, teniendo en cuenta aspectos como el formato o memoria disponible.
- Identificar inconsistencias, datos faltantes, duplicados, etc. en el conjunto de datos y aplicar una solución.
- Transformar el conjunto de datos para mejorar su calidad e integridad teniendo en cuenta diferentes aspectos de estos, dándoles más valor de cara a su análisis y a su uso para entrenar un modelo de ML.
- Separar el conjunto de datos en diferentes partes o splits utilizando diferentes estrategias para mejorar el desempeño en el entrenamiento del modelo de ML.

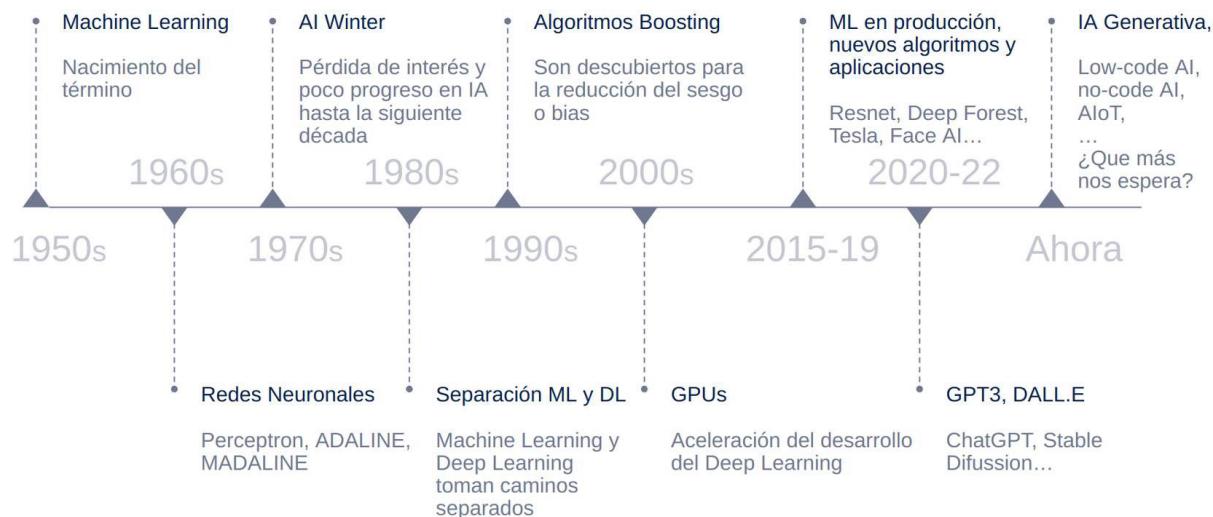
Autores: Gabriel Muñoz, Sergio Sampayo y Pablo Fernández

-  **Conceptos principales de aprendizaje basado en datos**
-  **Tipos de dato**
-  **Medición de resultados en machine learning**
-  **Estrategias para la preparación de datos y los enfoques de problemas**
-  **Tipos de valores**
-  **Transformación de datos (data transformation)**
-  **Conclusiones**
-  **Recursos útiles**

Conceptos principales de aprendizaje basado en datos



El término 'machine learning' ha crecido en popularidad a lo largo de los últimos años, así como sus implicaciones, pero su origen se sitúa más de 50 años atrás en el tiempo.



Historia resumida del machine learning. Fuente propia.

Si sientes curiosidad por conocer más detalladamente los diferentes hitos en la historia del machine learning, consulta este [enlace](#).

En la actualidad, podemos encontrar multitud de ejemplos en los que se usa **machine learning**:

- Vehículos autónomos.
- Detección de rostros en cámaras.
- Generación de imágenes a través de entradas de texto o 'prompts'.
- Recomendadores de películas.
- Chatbots que contesten nuestras preguntas.
- Inteligencias artificiales que ganan a maestros del ajedrez.

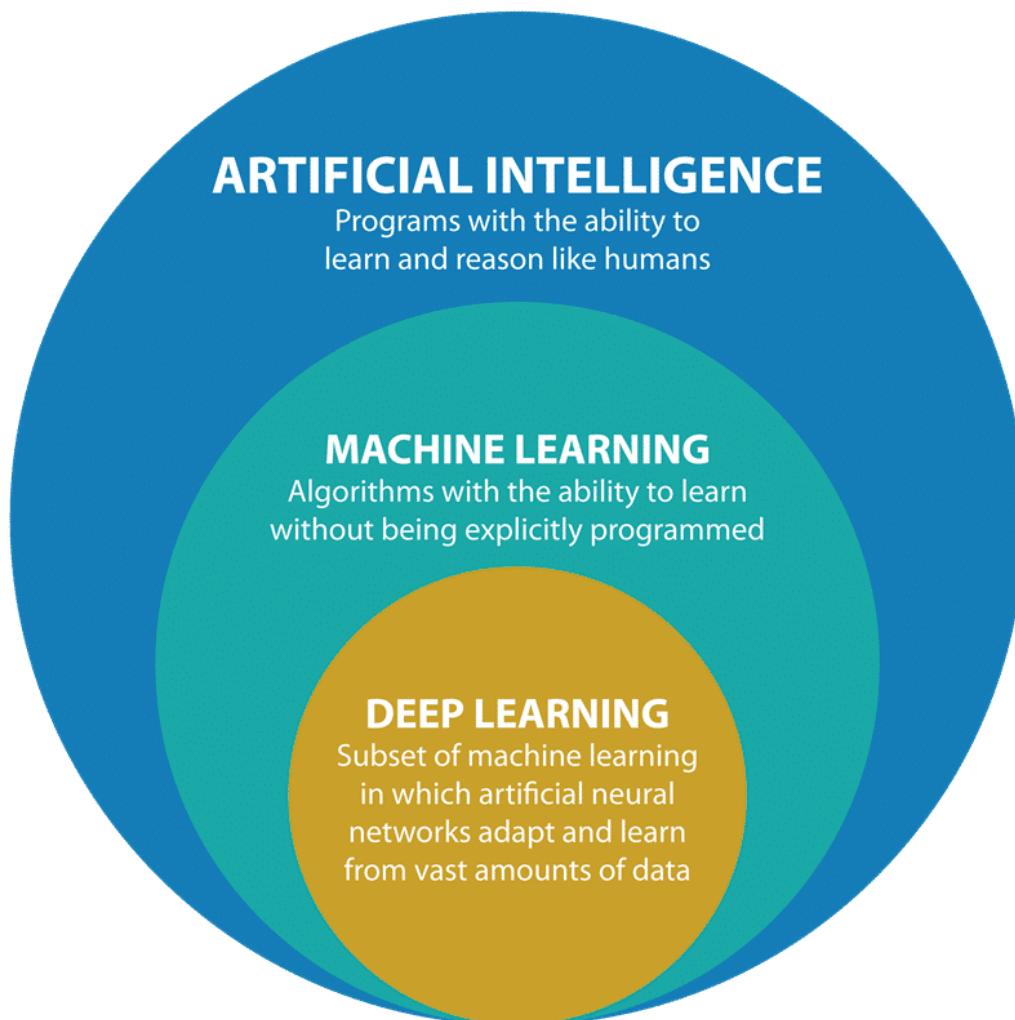
Sin embargo, es difícil discernir *a priori* **los procesos y/o elementos implicados** que permiten entender cómo estas increíbles aplicaciones llegan a hacerse realidad.

Para poder comprenderlo partiremos de la raíz del **concepto ML**, desde el propio **nacimiento del término en 1959** acuñado por **Arthur Samuel**, empleado de IBM y pionero en este campo:

"Machine learning es el campo de estudio que proporciona a las máquinas la capacidad de aprender sin haberlas programado explícitamente para ello."

- Arthur Samuel

Partiendo de esa base, podemos definir **machine learning** como **una rama de la inteligencia artificial (IA)** basada en el **uso de algoritmos y conjuntos de datos** para conseguir que **una máquina pueda realizar tareas de aprendizaje humano**, sin programación/instrucciones previas, **que mejoran con la experiencia**.



Subáreas de la inteligencia artificial. Fuente: Neurond

La **base del ML** es la **existencia y búsqueda** de una **relación matemática** entre cualquier combinación de **datos de entrada y salida**.

Parte de un conjunto de datos (dataset), pudiendo ser estos de **diferentes tipos**:

CUANTITATIVOS**CUALITATIVOS**

Datos expresados como valores numéricos que representan mediciones: cantidad, altura, peso, temperatura...

CUANTITATIVOS**CUALITATIVOS**

Datos que no pueden ser expresados con valores numéricos: texto, imágenes, sonido...

El algoritmo (conjunto ordenado y finito de operaciones que permite hallar la solución de un problema, según la RAE) buscará la relación entre los datos proporcionados, dando como resultado una predicción.

Hemos conocido el origen y evolución del término machine learning, dónde está situado dentro del ámbito de la inteligencia artificial o IA y cuál es la base de su funcionamiento. A continuación, ampliaremos el concepto de ML con la definición y usos prácticos de sus diferentes subáreas. Con ello, podremos empezar a comprender cómo se ha llegado a aplicar a los ejemplos de su uso previamente citados.

Tipos de dato



Dentro del aprendizaje máquina existen **dos amplias distinciones** entre tipos de dato:

LOS DATOS ESTRUCTURADOS

LOS DATOS NO ESTRUCTURADOS

Que a su vez pueden ser cuantitativos o cualitativos y que pueden acomodarse en una estructura, tabla o base de datos: números (medidas, conteos...), fechas, textos categóricos.

LOS DATOS ESTRUCTURADOS

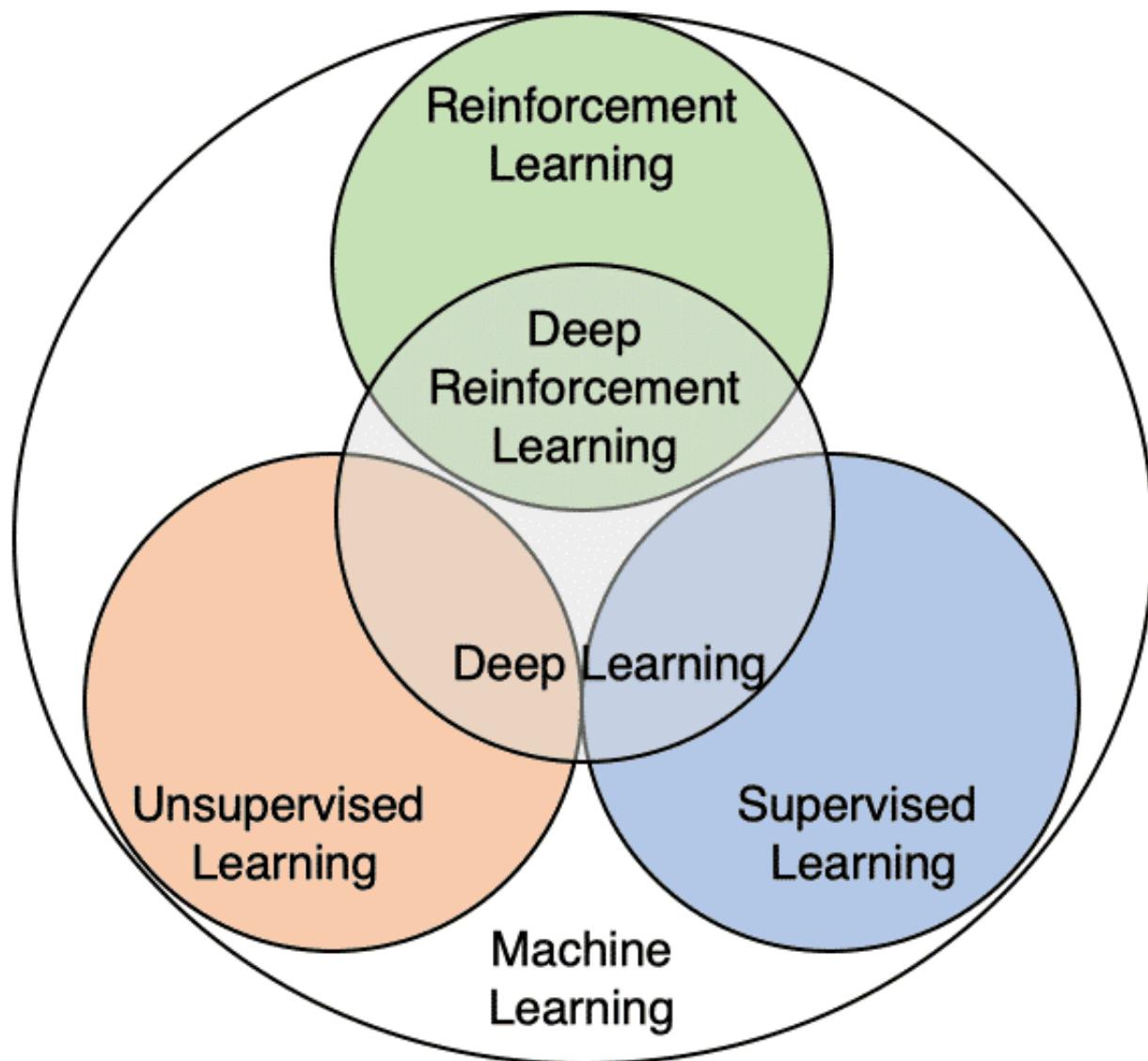
LOS DATOS NO ESTRUCTURADOS

Aquellos datos o información no organizada o sin esquema predefinido que no es posible acomodarla en una estructura, tabla o base de datos: imágenes, vídeos, lenguaje natural, audios...

Con estas definiciones podemos comprender el amplio espectro de datos que un modelo de ML puede llegar a usar para resolver distintas tareas.

Tipos de machine learning

Dentro del área de ML existen **diferentes tipos de implementaciones**, diferenciadas entre sí **según el tipo de estilo de aprendizaje del algoritmo y/o tarea a realizar**.

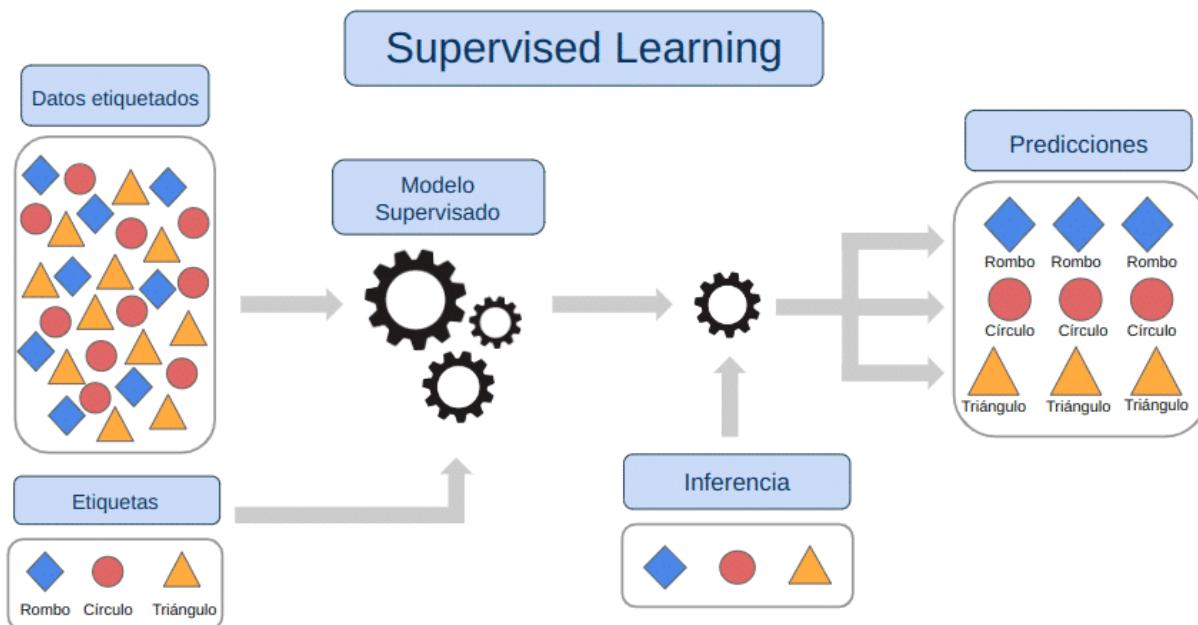


Subáreas de machine learning. Fuente: Paper Explainable AI for B5G/6G: Technical Aspects, Use Cases, and Research Challenges.

Supervised learning

El *supervised learning* o aprendizaje supervisado hace referencia al tipo de aprendizaje en ML que utiliza datos etiquetados (*labeled/with labels/labelled dataset or data*) para entrenar un modelo predictivo.

- Se provee al algoritmo de un conjunto de datos o **dataset**, el cual **contiene datos de entrada y salida** (*inputs y outputs/labels*).
- El **algoritmo** se encargará de **identificar patrones y relaciones entre los datos (inputs)**, **aprender de las observaciones (outputs/labels)** y para así **realizar predicciones**.



Esquema de un modelo de *supervised learning*. Fuente propia.

Dentro del *supervised learning* podemos distinguir entre dos tipos de tareas:

1

Clasificación (*classification*)

Algoritmos capaces de realizar una predicción de valores discretos o etiquetas/*labels*, de los datos que se pueden contar, con espacios o intervalos entre ellos. Comúnmente en el ámbito de ML representan categorías:

- Sí/no o true/false o 1/0.
- Número de categorías de 'X' elemento: verde, azul, amarilla o 0, 1, 2...

Veamos un ejemplo de caso de uso:

- Un hospital quiere facilitar el diagnóstico de diferentes tipos de diabetes al equipo médico.
- Disponemos de un *dataset* que recopila varios años de datos acerca de diferentes indicadores, mediciones, etc., de personas con un tipo de diabetes.
- Estos datos están etiquetados indicando qué tipo de diabetes posee cada persona.
- Con estos datos y un modelo supervisado de clasificación se podría predecir para una persona concreta qué tipo de diabetes sufre.

2

Regresión (*regression*)

Algoritmos capaces de realizar una predicción de valores continuos, de datos que se pueden medir, sin espacios o intervalos entre ellos:

- **Precios:** 12,112.12€, 50,000\$...
- **Altura:** 1.234m, 11km...
- **Temperatura:** 20.2°C, 270°F...

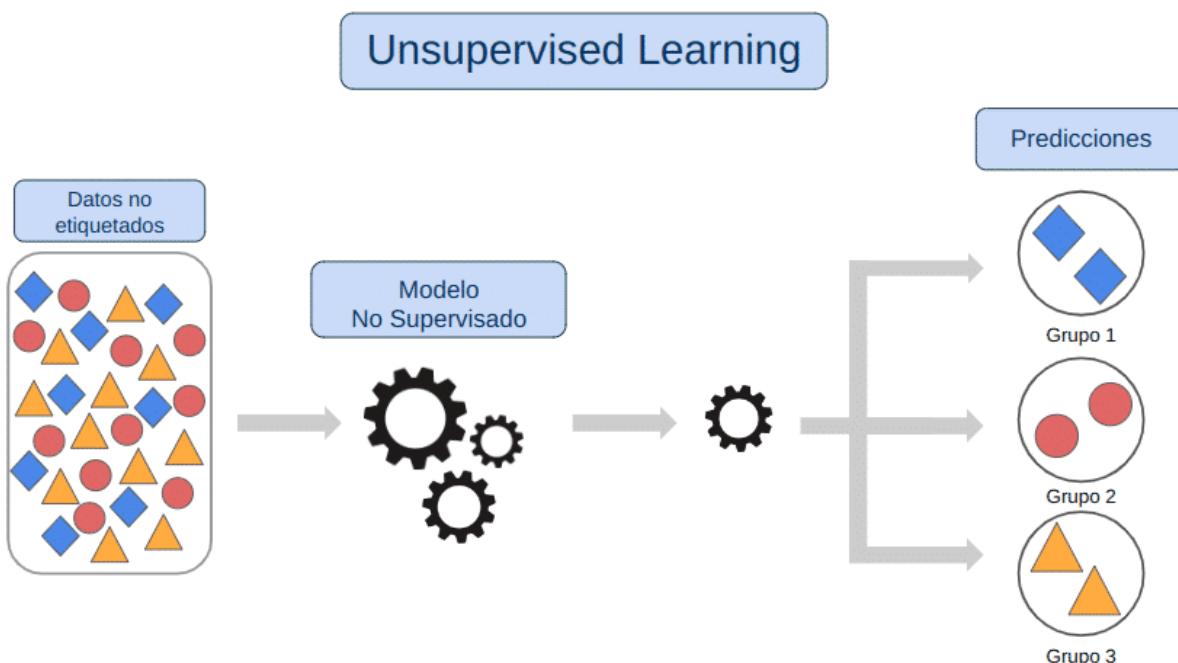
Veamos un ejemplo de caso de uso:

- Una productora quiere mejorar su toma de decisiones a la hora de producir una película de cara a sus inversores.
- Disponemos de un *dataset* con datos acerca de las características de películas (título, presupuesto, géneros, lenguaje original...), cada una de ellas etiquetada con la recaudación que consiguieron después de un año.
- Con estos datos y un modelo supervisado de regresión se podría predecir cuánto podría llegar a recaudar en un año una nueva película que va a lanzarse al mercado.

Unsupervised learning

El *unsupervised learning* o aprendizaje no supervisado hace referencia al tipo de aprendizaje en ML que utiliza **datos sin etiquetar** para entrenar un modelo predictivo.

- Se provee al algoritmo de un *dataset*, el cual **contiene solo datos de entrada (inputs)**.
- El algoritmo se encargará de **identificar patrones y relaciones entre los datos (inputs)** para **poder descubrir similitudes o grupos (clusters)**.



Esquema de un modelo de *unsupervised learning*. Fuente propia.

Dentro del *unsupervised learning* podemos distinguir entre **tres tipos de tareas**:

1

Agrupamiento (*clustering*)

Algoritmos capaces de **realizar una división de poblaciones o datos en un número concreto de grupos o clusters**. Unos datos serán más similares entre sí, si pertenecen a un mismo grupo, y serán más diferentes en comparación con los datos que pertenecen a otros grupos.

Veamos un ejemplo de caso de uso:

- Una universidad quiere acelerar la clasificación de nuevas especies de plantas en una región con ausencia de este tipo de estudios.
- Disponemos de un *dataset* que recoge datos como las medidas de las hojas, número de pétalos, etc. de una serie de especies de plantas distintas, sin estar estas etiquetadas.
- Con un modelo no supervisado orientado a una tarea de *clustering*, se podría conseguir separar los datos en diferentes grupos o especies y así poder categorizarlas rápidamente.

2

Reducción de dimensiones (*dimensionality reduction*)

Algoritmos que transforman los datos desde un espacio **dimensional alto** (*high-dimensional*) a **uno bajo** (*low-dimensional*) intentando **mantener las propiedades significativas en los datos originales**.

Veamos un ejemplo de caso de uso:

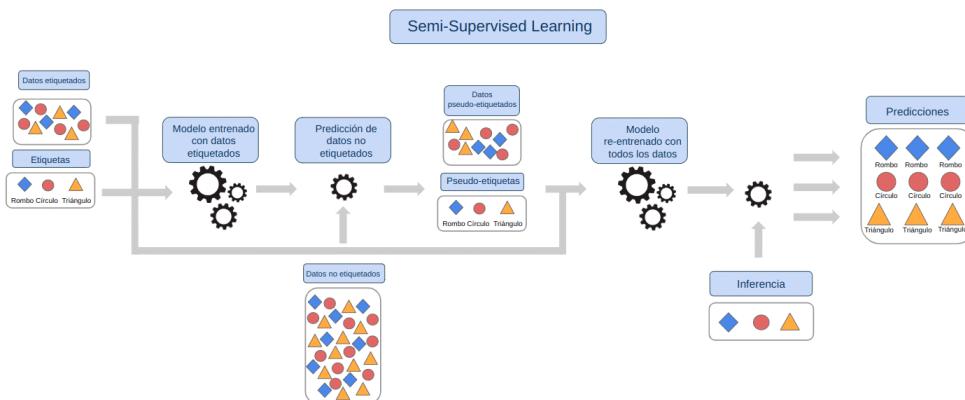
- Una cadena de supermercados quiere estudiar el comportamiento de sus clientes para mejorar su estrategia de venta a largo plazo.
- Disponemos de un *dataset* acerca de productos comprados por clientes en cada cesta de la compra de una cadena de supermercados.
- Con estos datos y un modelo no supervisado de reglas de asociación se podrían descubrir patrones de compra de los clientes en función de las relaciones entre varios productos (por ejemplo: la compra de un jabón lleva al cliente a la compra de un perfume en un 30% de las ocasiones con un 75% de confianza).



Revisa [aquí](#) la explicación detallada de un modelo de reglas de asociación.

Semisupervised learning

Semisupervised learning o aprendizaje semisupervisado se encuentra situado entre el *supervised* y el *unsupervised learning*, ya que hace referencia al tipo de aprendizaje en ML que utiliza una pequeña proporción de ejemplos etiquetados para entrenar el modelo inicial y **después usa un gran número de ejemplos no etiquetados iterativamente para obtener el modelo final**.



Esquema de un modelo de *unsupervised learning*. Fuente propia.

Al combinar ambas tareas, *supervised* y *unsupervised*, **puede realizar cualquier tipo de las tareas descritas previamente en ambos aprendizajes**: clasificación, regresión, *clustering*, etc., y, por tanto, los algoritmos que utiliza el aprendizaje semisupervisado son los respectivos a estas tareas, comentados anteriormente.

Veamos un ejemplo de caso de uso:

- Un investigador quiere **clasificar el diagnóstico temprano** de enfermedades mentales en base a unos indicadores.
- Los datos disponibles son **escasos**, con pocas observaciones etiquetadas con su respectiva enfermedad mental y muchas otras con solo la información de los indicadores.
- Con un modelo semisupervisado se podría **entrenar primero un modelo de clasificación** que utilice los datos etiquetados disponibles. Con este modelo entrenado, podemos intentar clasificar el resto de los datos no etiquetados disponibles (pseudoetiquetados).

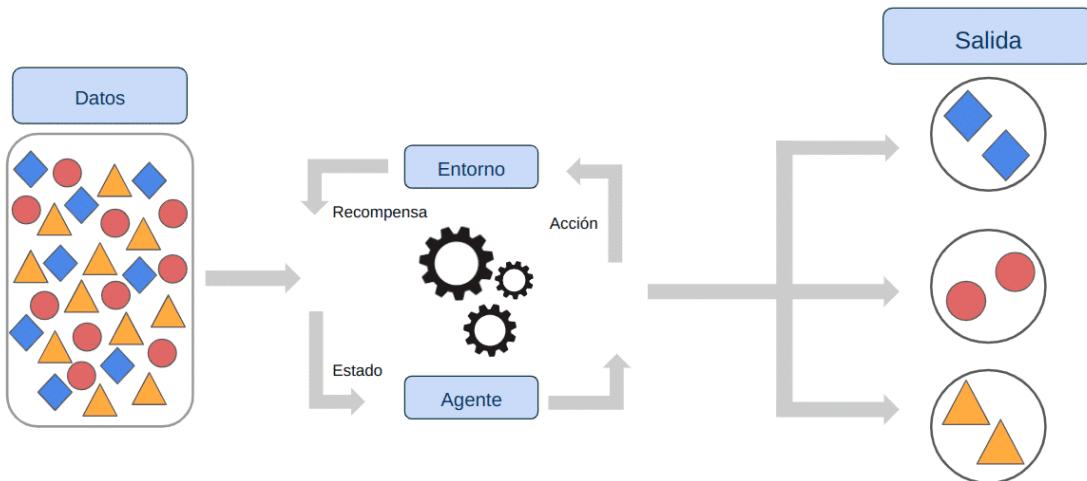
Al disponer ahora de todos los datos con etiquetas (etiquetados inicialmente + pseudoetiquetados) podemos reentrenar el modelo inicial con todos los datos, supliendo la falta inicial de observaciones sin etiquetar.

Reinforcement learning

El **aprendizaje por refuerzo o reinforcement learning** hace referencia al tipo de aprendizaje en ML basado en **interactuar con el entorno realizando acciones y descubriendo errores** en un proceso de ensayo-error.

Este tipo de aprendizaje permite **determinar automáticamente el comportamiento ideal en un contexto concreto** con el objetivo de maximizar su eficiencia o desempeño. Puede ser usado en aplicaciones que requieran toma de decisiones en escenarios inciertos.

Reinforcement Learning



Esquema de un modelo de *reinforcement learning*. Fuente propia.

Veamos un ejemplo de caso de uso:

- Una ciudad posee una elevada densidad de tráfico y quiere reducirla mejorando el sistema de señalización por semáforos utilizando machine learning.
- Disponemos de acceso a datos del control de semáforos de la ciudad.
- Con esos datos y un modelo de *reinforcement learning* se podrían optimizar los tiempos adecuándose a la densidad de tráfico existente.

Deep learning

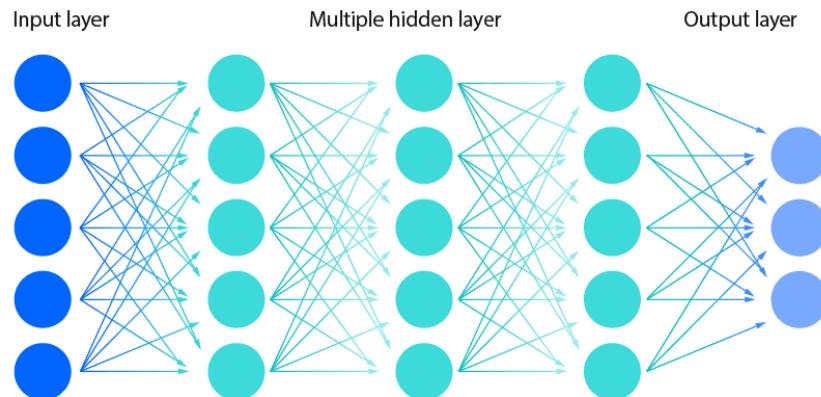
El **aprendizaje profundo o deep learning** hace referencia al tipo de aprendizaje en ML **basado en el uso de redes neuronales** (*neural networks* o NNs) de tres o más capas.

Las redes neuronales basan su nombre en tratar de simular el comportamiento de las **neuronas cerebrales humanas**.

Las capas o *layers* son las **estructuras o redes dentro de la arquitectura del modelo**. Estas capas reciben información, datos o *inputs* de una capa previa y pasa sus resultados u *outputs* a la siguiente capa.

El término 'profundo' o *deep* hace referencia a la presencia de **capas ocultas o hidden layers en la arquitectura del modelo al situarse entre las capas de entrada y salida** (*input layer, output layer*), las cuales toman unos datos de entrada ponderados (*weighted inputs*) y producen un *output* a través de una función de activación. En resumen, estas capas realizan transformaciones no lineales sobre los inputs de entrada.

Deep neural network



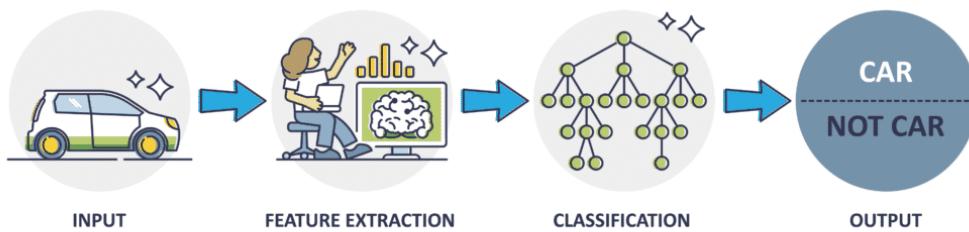
Estructura básica de una red neuronal. Fuente:
<https://www.ibm.com/topics/neural-networks>.

Veamos un ejemplo caso de uso:

- Un nuevo parque natural quiere catalogar las diferentes especies de pájaros presentes.
 - Disponemos de bases de datos de todas las especies aviares en España, tanto de datos tabulares como de imágenes y sonidos grabados de su canto.
 - Con estos datos y con la utilización de *deep learning* se podría desarrollar un modelo de reconocimiento de especies por imagen que permitan rápidamente realizar esta catalogación. También se podría desarrollar un modelo de *deep learning* de reconocimiento de cantos de pájaro, para la identificación de aquellas especies que son más difíciles de visualizar.
-

El uso de redes neuronales cambió el modo en el que se aplica ML, por lo que es necesario entender las principales diferencias entre el conjunto de tareas de ML y el deep learning.

MACHINE LEARNING



DEEP LEARNING



Comparación simplificada de machine learning y *deep learning*. Fuente:
<https://www.quantace.in>.

Machine learning	Deep learning
Subárea de la IA.	Subárea de ML.
Puede utilizar desde pequeños a grandes conjuntos de datos.	Requiere grandes conjuntos de datos.
Utiliza datos estructurados/tabulares.	Utiliza datos estructurados y/o datos no estructurados/no tabulares.
Utiliza diferentes algoritmos automáticos.	Utiliza redes neuronales.
Requiere mayor intervención humana para corregir errores y mejorar resultados.	Requiere menos intervención humana para mejorar resultados una vez está en funcionamiento.
Tiempo de entrenamientos más cortos.	Tiempo de entrenamientos más largos.
Puede entrenarse en una CPU.	Necesita, por lo general, entrenarse en una GPU.
Es necesario probar variables o <i>features</i> para tratar de obtener mejores resultados.	Aprende de las <i>features</i> y detecta las más relevantes automáticamente.

Hemos podido **comprender y asimilar las diferentes subáreas** que existen dentro del machine learning tanto teóricamente como de forma práctica a través de casos de uso que se dan en la realidad. Este conocimiento nos permitirá comprender y dar solución de forma más precisa a las diferentes situaciones y problemáticas que se presenten y que requieran de una solución de ML.

Modelos generativos

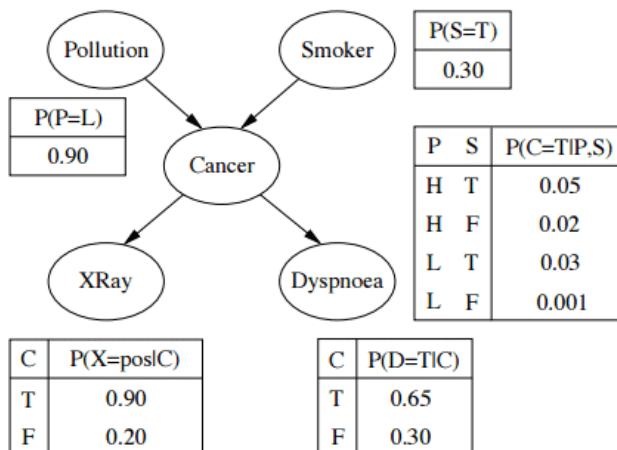
Los modelos generativos o *generative models* hacen referencia al área del ML que trata de **identificar patrones y/o distribuciones en los datos con intención de generar datos nuevos**, similares a los originales.

Dentro de los modelos generativos existen **diferentes tipos**:

1

Redes bayesianas o bayesian networks

Modelos gráficos que representan un conjunto de variables y sus dependencias condicionales/probabilidad de las relaciones entre ellas a través de **un grafo acíclico dirigido** o *directed acyclic graph* (DAG). Se suelen usar en escenarios donde es importante entender las relaciones casuales: diagnósticos médicos, series temporales, detección de anomalías, etc.

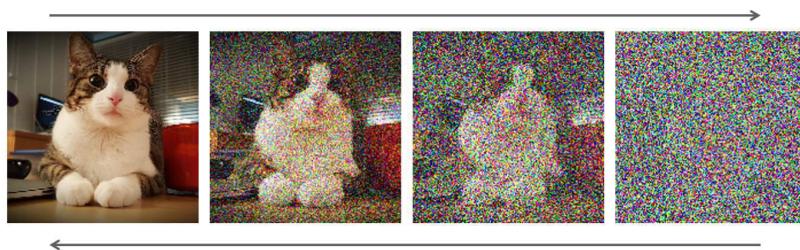


Ejemplo de red bayesiana. Fuente: <https://math.stackexchange.com>.

2

Modelos de difusión o diffusion models

Modelos que simulan el proceso de generación de datos transformando una distribución simple en la distribución objetivo, más compleja, a través de una serie de operaciones: más detalladamente, se aplica **ruido gaussiano** a los datos de entrenamiento sucesivamente y el **modelo trata de recuperar los datos originales**, revirtiendo este proceso. Se suelen usar para limpiar ruido de imágenes o *denoising*, aumentar la resolución de imágenes, generar imágenes, etc.

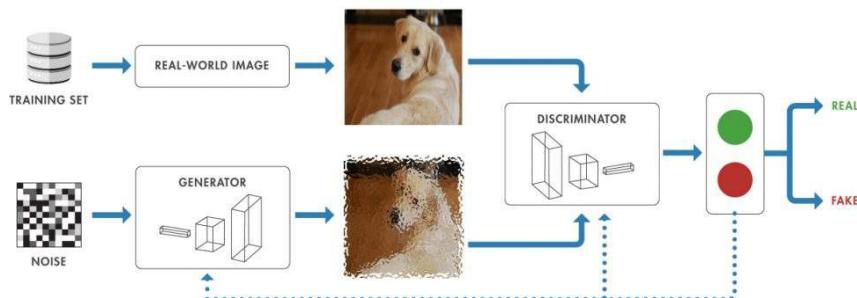


Ejemplo de cómo funciona un modelo de difusión. Fuente: Nvidia.
<https://developer.nvidia.com>.

3

Redes generativas antagónicas o Generative Adversarial Networks (GANs)

Modelos que consisten en dos redes neuronales, **la generativa y la discriminatoria**, que se entrenan conjuntamente: la generativa produce datos y la discriminatorio intenta distinguir si son reales/originales o generados. Su uso también va dirigido a la generación de imágenes.

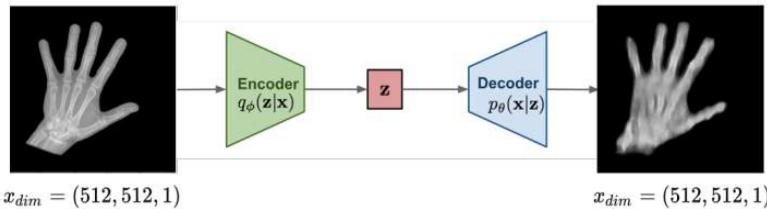


Esquema de cómo funcionan las GANs. Fuente: MathWorks.
<https://blogs.mathworks.com>.

4

Autocodificadores varacionales o *variational autoencoders* (VAEs)

Modelos que consisten en un tipo de **autocodificador que produce una representación comprimida de los datos de entrada** y la decodifica generando nuevos datos. Suelen usarse para eliminación de ruido en imágenes, generación de imágenes...



Un ejemplo de cómo funciona un autocodificador varacional. Fuente:
ResearchGate.

5

Transformers

Modelos que **usan mecanismos de atención** para modelar las relaciones entre los diferentes elementos de una secuencia, siendo excelentes en aprender el contexto y el significado de estas. Su uso se enfoca sobre todo en las imágenes y el texto, como el famoso modelo preentrenado GPT (GPT-3, aplicación derivada ChatGPT...) que es uno de los modelos perteneciente a esta categoría, capaz de ejecutar tareas de generación de texto, respuestas a preguntas, summarización de texto, extracción de texto, etc.

6

Makov chains

Modelos matemáticos usados para **representar procesos estocásticos** (conjunto de variables aleatorias que representan el estado del sistema), donde el estado futuro del sistema está basado solamente en el estado actual, sin tener en cuenta estados anteriores. Suelen usarse en la generación de texto, música, imágenes, generando palabras, notas o píxeles de forma sucesiva.

 Puedes conocer más acerca de los modelos generativos en estos enlaces:
[YouTube](#) y [Datacamp](#).

Como has podido comprobar, las distintas áreas del machine learning engloban diferentes tareas que pueden resolver, por lo que es importante saber en qué área debemos centrarnos cuando queremos dar solución a un problema con machine learning.

Medición de resultados en machine learning



La medición de resultados y/o del desempeño de un modelo en ML es una parte fundamental: todo modelo de ML necesita de una métrica que nos permita evaluarlo.

Existen múltiples métricas que pueden ser usadas en cada tipo de problema o tarea, resultando unas más apropiadas que otras en función de nuestro objetivo, de las características del conjunto de datos...

Estas son **algunas de las métricas más usadas**, ¡anota!

Regresión

- **Error medio absoluto** (*mean absolute error* o MAE): mide la diferencia entre los valores objetivo y los predichos en términos absolutos.
- **Error cuadrático medio** (*mean square error* o MSE): mide la diferencia elevada al cuadrado entre los valores objetivo y los predichos.
- **Raíz del error cuadrático medio** (*root mean square error* o RMSE): mide la raíz cuadrada de la diferencia elevada al cuadrado entre los valores objetivo y los predichos.

Clasificación

- **Accuracy:** número de predicciones correctas dividida entre el número total de predicciones, multiplicado por 100.
- **Precisión:** ratio de valores verdaderos positivos predichos entre todos los verdaderos positivos y falsos positivos (todos los positivos predichos).
- **Sensitividad o recall:** ratio de valores verdaderos positivos predichos entre todos los verdaderos positivos y falsos negativos.
- **F1:** media armónica entre la precisión y la sensitividad. Representa un buen balance entre ambas.
- **AUROC** (área bajo la curva-curvas ROC): combina la curva ROC (*receiver operating characteristic*) referida a la sensibilidad y la especificidad, y el área bajo la curva referida a la separabilidad de las diferentes clases/*labels*.
- **Matriz de confusión:** recoge el conteo de las predicciones del modelo, diferenciando entre verdaderos positivos y los verdaderos negativos (aciertos), y en falsos positivos y falsos negativos (desaciertos).

Deberemos analizar nuestro caso de uso y, considerando el tipo de tarea de machine learning, las métricas más adecuadas que midan el desempeño de nuestro modelo.



Puedes encontrar más información en [este enlace](#).

Estrategias para la preparación de datos y los enfoques de problemas



Carga de datos

Para poder comenzar la preparación de datos necesitamos cargarlos y, para ello, debemos considerar en qué formato se encuentran.

En el caso de **datos tabulares**, los más comunes suelen ser:

- .csv**: comma separated values.
- .json**: JavaScript object notation (dictionary like).
- .parquet**: formato tabular de Apache.

En el caso de datos no tabulares, dependerá del tipo de dato:

- .jpg/.png**: imágenes.
- .wav/.mp3**: archivos de audio.
- .txt/.json/.html**: texto.

Cada formato tiene sus pros y sus contras, y deberemos evaluar cuál nos es más conveniente, ya sea por decisión de accesibilidad, espacio, u otras necesidades.

Según qué tipo de librería utilicemos, suelen existir diferentes métodos o funciones que nos facilitarán la carga.

Es importante inspeccionar los datos previamente (si es posible), ya que podemos atisbar si es necesario **dar instrucciones extra al método de carga** y prevenir futuros errores o fallos, por ejemplo:

- Si la separación de los datos se realiza utilizando delimitadores específicos (comas, guiones, tabulaciones...), es necesario indicarlo.

- La existencia de un índice y/o encabezado.

- Si es necesario cargar los datos a partir de 'n' observación/fila, etc.



Consulta [este enlace](#) para ver un caso práctico sobre las diferentes formas de cargar un mismo archivo que contiene un conjunto de datos. Asimismo, haz clic en [este enlace](#) para ver un ejemplo práctico sobre la carga de imágenes.

Archivos grandes y problemas de memoria

Puede darse la situación de que el/los archivo/s que contienen los datos a tratar sean demasiado grandes para cargarlos en memoria temporal/RAM. Ante esta circunstancia existen diferentes estrategias que se pueden aplicar:

Cargar solo de una muestra de las observaciones/filas

Nos facilitará el análisis y/o construcción del procesamiento de los datos, y quizás nos permitirá descubrir qué observaciones y/o variables/columnas nos son útiles para, consecuentemente, solo cargar las necesarias, ahorrando así memoria, o para directamente procesar todos los datos cuando ya tenemos nuestros procesos/pipelines establecidos a través del método que describimos a continuación.

Utilizar un tipo de dato adecuado y/o las variables/columnas necesarias

En relación con el punto anterior, normalmente las funciones o módulos de carga de datos nos permiten indicar qué columnas exactamente necesitamos, así como el tipo de dato o *data type/dtype* y, por tanto, ahorramos memoria.

Algunos tipos de dato ocupan más memoria que otro que lo puede sustituir, por ejemplo: una columna tipo int64 (números enteros de hasta 64 dígitos) ocupa más que otra tipo int8, cuando quizás en nuestro caso este último tipo nos es suficiente.

Carga progresiva de las observaciones y/o archivos

No es necesario cargar todas las observaciones/archivos en memoria al mismo tiempo; también es posible hacerlo en tandas o *batches*.

Las diferentes librerías de procesamiento de datos dispondrán de métodos que nos lo permitan. Con ello, solo un número de observaciones/filas serán cargados cada vez, impidiendo la saturación de nuestra memoria temporal disponible. También existen acercamientos de este tipo para datos no estructurados como imágenes.

Cambio del formato de archivos

Como ya se ha comentado anteriormente, algunos formatos de archivo son más favorables a la hora de optimizar mejor el tamaño en memoria usado para el mismo conjunto de datos.

El formato '.parquet' suele ser usado para realizar una buena optimización de la memoria ocupada por los datos en comparación con el conocido formato '.csv', aunque existen otras formas como los formatos binarios HDF, GRIB, etc.

Uso de bases de datos relacionales

Ya sea desde un acceso local u online, es posible cargar los datos a través de una *query* usando el *standard query language* (SQL). Esto nos puede ahorrar memoria según la base de datos usada y las especificaciones de la *query* usada.

-  Consulta [este enlace](#) con ejemplos prácticos sobre el ahorro de memoria en la carga de datos.

La carga de datos puede parecer trivial, pero es importante considerar los aspectos anteriormente descritos para que el comienzo de la preparación de los mismos sea lo más rápida y sencilla posible.

Perfilado y limpieza de datos (*data profiling and data cleaning*)

El perfilado de datos o *data profiling* hace referencia a la búsqueda de errores, inconsistencias, etc., dentro del conjunto de datos, mientras que la limpieza de datos o *data cleaning* hace referencia a su corrección.

Son dos tareas muy importantes ya que, por un lado, la **incoherencia dentro de los datos** puede llegar a sesgar y/o **perjudicar el desempeño del modelo de machine learning** y, por otro, pueden llegar a **crear problemas en la parte computacional** y/o programática, impidiéndonos incluso entrenar un modelo al no aceptar este los datos que le proporcionamos por cuestiones de formato, tipo, etc.

Datos faltantes (*missing data*)

Los valores faltantes o *missing data* son aquellos valores que se encuentran ausentes dentro de su respectiva fila o entrada de datos y columna, variable o *feature* en un *dataset*.

	First Score	Second Score	Third Score	Fourth Score
0	100.0	30.0	52.0	60
1	NaN	NaN	NaN	67
2	NaN	45.0	80.0	68
3	95.0	56.0	98.0	65

Ejemplo de *missing values* en un *dataset*. Fuente: <https://www.geeksforgeeks.org>.

Normalmente, **la ausencia de datos suele ser representada de diferentes formas**, aparte del común espacio vacío o en blanco. Diferentes librerías representan los datos faltantes de diferentes formas. Sin embargo, existen **ciertas nomenclaturas y/o acrónimos** que suelen usarse para su representación:

None

Representa la ausencia de un valor no numérico.

NaN (not a number)

Representa la ausencia de un valor numérico, normalmente, del tipo *float* (la presencia de NaN en una columna/*feature* tipo *integer* la convierte en tipo *float*). Otras formas comunes de NaN son NAN, <NaN>, <NAN>...

NaT (not a time)

Representa la ausencia de un valor tipo fecha. Suele ser específico de variables de tipo fecha-tiempo o *datetime*.

'','etc.

Representan la ausencia de texto o *strings* o de cualquier otro dato, ya que dependerá del contexto del resto de datos presentes.

Debemos tener presente que, aunque estas sean las formas convencionales de representación de datos faltantes, **puede llegar a darse el caso de que existan otros valores que los representan**, ya sea por convenciones existentes en el momento de creación del *dataset* o por otros motivos (por ejemplo: los valores atípicos, como '99999999' o '00000000', en una variable 'peso' con un rango de datos entre 0-120.00 KG).

Para la detección y limpieza de datos faltantes, disponemos de diferentes librerías que contienen diversos métodos programáticos que nos facilitarán esta tarea de detección y limpieza de datos faltantes. **Una vez confirmada la presencia de datos faltantes**, debemos pensar en qué tipo de estrategia queremos aplicar para eliminarlos o limpiarlos, según nuestro caso de uso.

1

Eliminar o desechar (*drop/dropping*)

Se desechan las observaciones/entradas de datos/*rows* o columnas/*features* que contengan datos faltantes. La eliminación de observaciones puede realizarse en todo el *dataset* o solo en un subconjunto de *features* del mismo. Es recomendable comprobar previamente qué porcentaje de datos vamos a perder en el proceso y si nos es asumible la pérdida o no.

PROS

CONTRAS

- Es la estrategia más simple de implementar.
- Adecuada si los datos faltantes no poseen importancia.

PROS

CONTRAS

- Pérdida de información.
- Si la proporción de datos faltantes es grande puede crear un sesgo o bias.

2

Imputación (*imputation*)

Consiste en el reemplazo de **datos faltantes por otros**. Para que este reemplazo de datos tenga sentido y se aadecue a la variable/*feature* en cuestión, se suelen utilizar diferentes estrategias: imputación con la media/mediana (para valores numéricos) de cada respectiva *feature* o con valores aleatorios respectivos (para valores numéricos y/o categóricos), etc.

También existen métodos como la imputación múltiple/iterativa, que tiene en cuenta la información de otras columnas/*features*.

PROS

CONTRAS

- Puede ser simple y fácil de implementar, según el tipo de imputación.
- La imputación se realiza en base a datos existentes.

PROS

CONTRAS

- Solo es posible de implementar en columnas/*features* numéricas.
- La media como valor de imputación es sensitiva a valores atípicos.
- Estos valores de imputación asumen que la causa de los datos faltantes es aleatoria, algo que no tiene por qué ser siempre cierto.

Los datos faltantes pueden tener un **impacto negativo en nuestro modelo** por lo que es importante saber identificarlos y lidiar con ellos.

- i Para **ampliar la información** acerca de las estrategias de detección y tratamiento de datos faltantes junto con ejemplos prácticos, consulta los siguientes enlaces sobre: [técnicas para manejar valores perdidos](#), [imputación estadística](#) e [imputación iterativa](#).

Tipos de valores



Valores atípicos (*outliers*)

Los valores atípicos o *outliers* son valores extremos que difieren significativamente del resto del conjunto de datos en cuanto a situarse a una distancia anormal.

Pueden ser valores extremadamente bajos o altos relativos al siguiente valor más cercano, cuyo origen puede ser natural o artificial, por ejemplo:

Errores de anotación

Durante la recolección de datos.

Errores de medición

Por fallo del instrumento de medición, por fallo humano...

Errores de preprocessado

Durante la manipulación de datos...

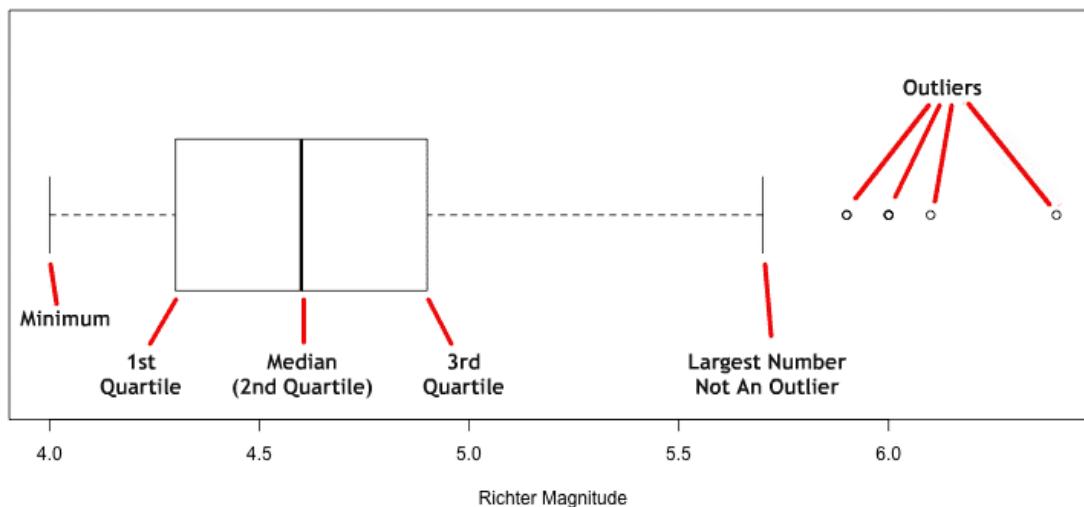
Errores de sampleado

En la extracción de datos.

Outlier natural

No es un valor erróneo, sino que es un valor verdadero y de causa natural.

Boxplot of Richter Magnitude for Earthquakes off Fiji



Ejemplo de presencia de *outliers*. Fuente: <https://www.stats4stem.org/boxplots>.

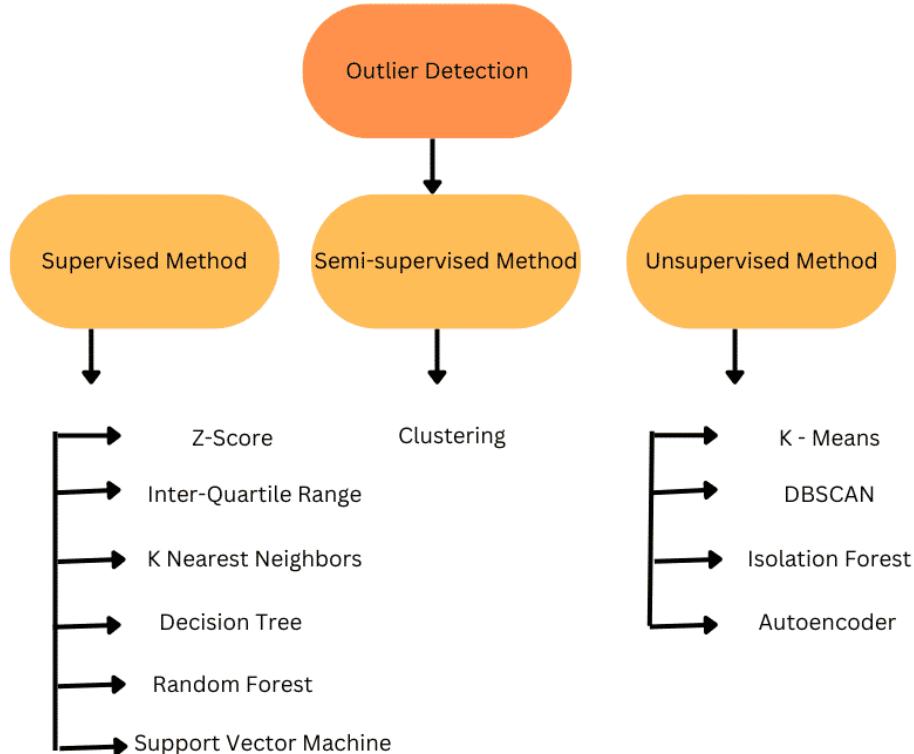
Estos valores **pueden llegar a afectar a los análisis estadísticos del conjunto de datos** y/o confundir al modelo a entrenar, afectando a su rendimiento, por lo que es importante que sean tratados.

Existen múltiples métodos de detección de *outliers*, algunos incluso basados en el uso de algoritmos de ML. Se pueden **agrupar en varias categorías**:

- Métodos supervisados o *supervised*** que utilizan datos etiquetados o *labeled data*.
- Métodos no supervisados o *unsupervised*** que utilizan datos no etiquetados.
- Métodos semisupervisados** que utilizan datos etiquetados y no etiquetados.
- También existen otras categorías, como el **ensamble de métodos** o *ensemble methods*, que combinan varios métodos, o los basados en *deep learning*.

La elección y uso de estos depende del caso de uso y los datos a tratar.

A continuación, vamos a describir los más comunes y simples dentro de los métodos supervisados. El resto de los métodos resultarán más fáciles de entender una vez que conozcamos en profundidad los diferentes algoritmos de ML.



Métodos de detección de outliers más comunes. Fuente:
<https://www.almabetter.com>.

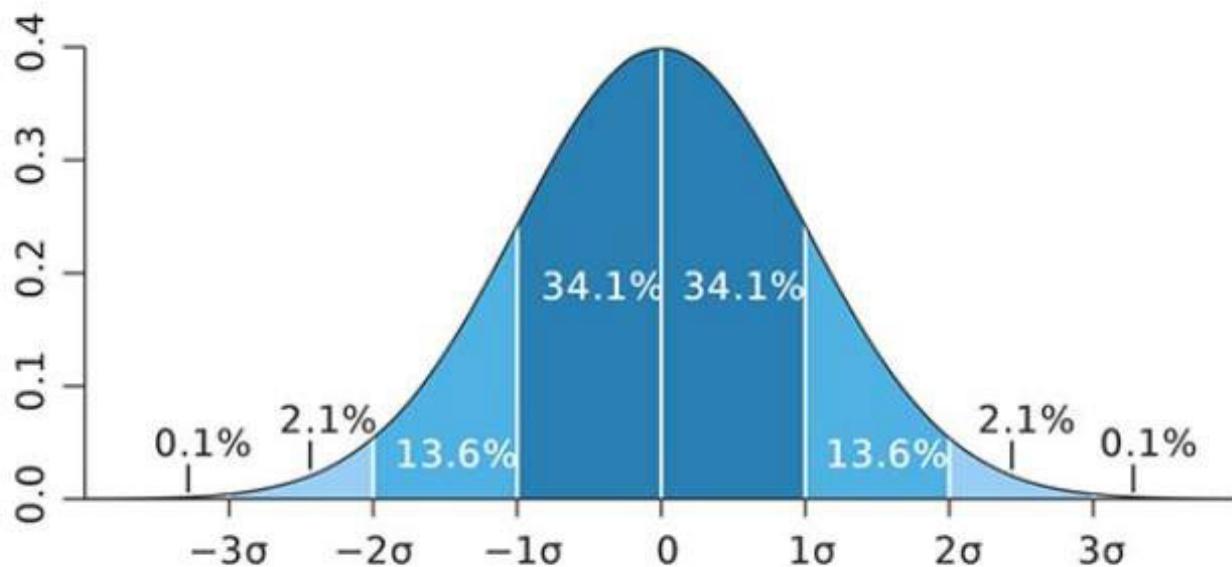
1

Z-score

Método estadístico que describe **la relación que tienen los valores con la media del conjunto**, midiéndose con el número de desviaciones estándar a esa media.

$$z = \frac{(x - \mu)}{\sigma}$$

z es el Z-score, x es el valor, μ es la media y σ es la desviación estándar.



Distribución del Z-score. Fuente: <https://sugermint.com>.

El Z-score es igual a cero cuando $x = \mu$ (idéntico a la media); es $\pm 1, \pm 2$, o ± 3 , dependiendo de si x es $\pm 1, \pm 2$, o ± 3 , respectivamente (es decir, el valor se encuentra a $\pm 1, \pm 2$, o ± 3 desviaciones estándar de la media).

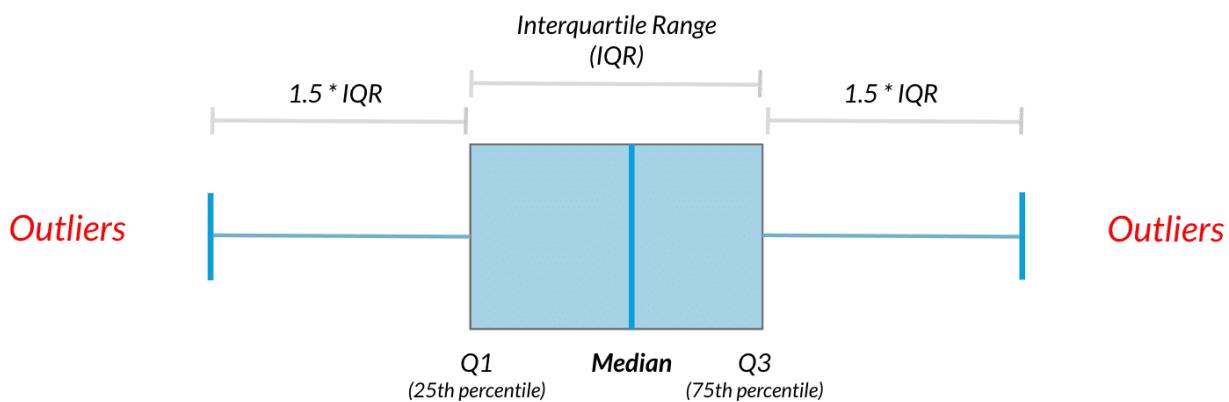
Debemos tener en cuenta que este método asume una distribución normal, por lo que puede no ser apropiado si no se cumple esta condición.

2

Rango intercuartil (inter quartile range o IQR)

Método **basado en el cálculo del primer y tercer cuartil** (Q_1 , Q_3) del conjunto de datos para poder identificar los rangos $Q_1 - 1.5 * \text{IQR}$ y $Q_3 + 1.5 * \text{IQR}$ y, así, determinamos como *outliers* aquellos valores que se encuentran fuera de estos rangos.

- Q1:** representa el primer cuartil o el 25 percentil de los datos.
- Q2:** representa el segundo cuartil, la mediana o el 50 percentil de los datos.
- Q3:** representa el tercer cuartil o 75 percentil de los datos.
- IQR:** representa $Q_3 - Q_1$.
- ($Q_1 - 1.5 * \text{IQR}$):** representa el valor más pequeño dentro del conjunto de datos.
- ($Q_3 + 1.5 * \text{IQR}$):** representa el valor más grande dentro del conjunto de datos.



Detección de *outliers* usando el método del rango intercuartil. Fuente:
<https://www.almabetter.com>.

3

Percentiles o porcentajes

Método basado en el uso de un umbral porcentual para identificar como *outliers* aquellos valores que se encuentran fuera de ellos.

Primero, se determina el umbral a partir del cual se considera como *outlier* un valor: normalmente se usa el percentil 95 o 99 (el 5%/1% de los valores son considerados como *outliers*, respectivamente). La elección dependerá del conjunto de datos y del objetivo de identificación de *outliers*. Después, se calcula el **umbral inferior y superior**, fuera de los cuales los valores son identificados como *outliers*.

 Puedes consultar la aplicación práctica de estos métodos de detección de outliers y otros más avanzados en los siguientes enlaces:
www.freecodecamp.org y www.machinelearningmastery.com.

Al igual que con los datos faltantes, existen diferentes **estrategias que se pueden aplicar para tratarlos**:

Eliminar o desechar (drop/dropping)

Se desechan las observaciones/entradas de datos/rows que contengan *outliers*. Para ello, es necesario previamente la detección de estos con los análisis estadísticos.

Imputación (imputation)

Al igual que con los datos faltantes (*missing values*), los *outliers* se pueden imputar con valores como la media, mediana, moda, etc. Es necesario cerciorarse previamente si su origen es natural o artificial/aleatorio, como ya se explicó anteriormente.

Transformación

La transformación de columnas/variables/features puede llegar a eliminar la presencia de *outliers*. Existen diferentes métodos que se pueden aplicar, pero aquí destacamos tres:

- **Escalado de datos o scaling:** transforma/escala los datos a un rango concreto. Se explicará con más detalle en el siguiente apartado dedicado a la data preparation.
- **Transformación logarítmica:** uso del logaritmo para cambiar la escala/ rango de los datos.
- **Transformación Box-Cox:** técnica estadística que transforma los datos a una distribución normal.

Estas transformaciones consiguen que los outliers sean transformados a una escala menor, disimulando o eliminando su efecto anterior como valor extremo.

Como en cualquier otro proceso, **debemos considerar cuáles el acercamiento más adecuado para nuestros propósitos.**

Los *outliers*, al igual que los datos faltantes, pueden causar un impacto negativo en nuestro modelo y resultados, por lo que debemos identificarlos y transformarlos (o eliminarlos) para que se adecuen a los rangos del resto de datos a los que pertenecen.

 Puedes ampliar esta información con ejemplos prácticos en estos enlaces:
www.machinelearningmastery.com y www.builtin.com.

Valores duplicados

Los valores duplicados o *duplicates*, como el nombre indica, hacen referencia a aquellos valores o entradas de datos repetidas dentro del *dataset*.

Por lo general, es importante que un *dataset* posea heterogeneidad en las observaciones o entradas de datos para que el modelo disponga de más recursos de aprendizaje.

Las entradas de datos duplicadas no son más que ejemplos repetidos para el modelo, de los cuales ya ha aprendido anteriormente y, por lo general, no van a mejorar su desempeño. Por ello, suelen desecharse las entradas de datos duplicadas.

Como ya se ha comentado, debemos considerar nuestro caso de uso para decidir la mejor solución.

La **eliminación de valores duplicados** puede realizarse por observación/fila o por columna/*feature*. También puede ser más específica, por ejemplo:

- Considerar solo ciertas *features* a la hora de evaluar las observaciones duplicadas.
- Conservar la primera ocurrencia/observación duplicada, la última o ninguna.
- Reemplazar los valores duplicados con valores alternativos (valores de imputación, etc.).

No es deseable que tengamos información repetida en nuestro conjunto de datos por lo que es importante eliminar o reemplazar los valores duplicados.

Datos incorrectos

Puede llegar a darse el caso de que existen **valores incorrectos**, ya sea por error humano o no, intencional o no.

La **única manera de detectarlos** es a través de la observación y el conocimiento de los datos y su contexto. También dependerá de la propia naturaleza del dato: los errores en variables numéricas pueden llegar a ser muy diferentes de errores en variables categóricas.

Es recomendable, por tanto, realizar un análisis de distribución, de frecuencias, etc., para poder llegar a detectarlos.

Los **datos incorrectos** deben ser arreglados o descartados para impedir que afecten negativamente al desempeño de nuestro modelo de machine learning.

Hemos podido comprobar que no debemos asumir que la calidad y consistencia de los datos va a ser buena y/o correcta inicialmente, por tanto, es necesario efectuar análisis orientados a la detección de valores faltantes, extremos, duplicados o erróneos.

Transformación de datos (data transformation)



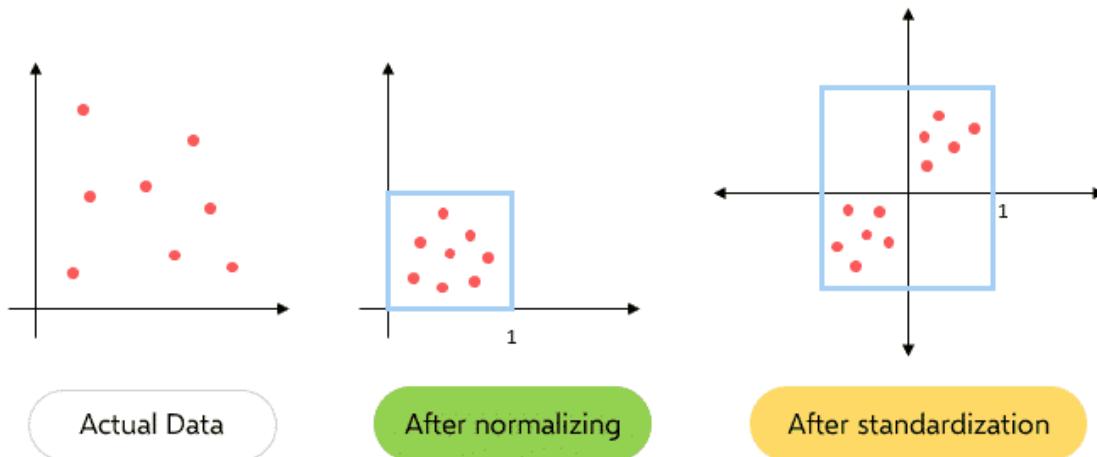
Escalado de datos (*scaling*)

El escalado de datos o *scaling* hace referencia a la transformación de valores a un rango o escala concreta.

La escala de los datos es importante ya que **diferentes campos/columnas/features pueden poseer diferentes escalas** (por ejemplo, el precio de una vivienda: valores entre 50,000 a 1,000,000\$, y los metros cuadrados de una vivienda: valores entre 20 y 2,000 m²), lo que puede llegar a confundir al algoritmo en cuestión, dándole más importancia a unos valores y no a otros.

El escalado de datos ayuda a comparar diferentes campos al convertirlos a un mismo o similar rango de valores.

A continuación, vamos a estudiar las diferentes estrategias de escalado de datos. La elección de cada una debe ajustarse a nuestros criterios o caso de uso.



Ejemplo visual del cambio de la distribución de los datos después de aplicar diferentes estrategias de escalado o *scaling*. Fuente: <https://www.someka.net>.

Normalización (normalization)

Estrategia de escalado que transforma los datos a una escala entre el menor y el mayor valor dados.

Es la estrategia usada cuando los valores de los diferentes campos o *features* tienen diferentes rangos o escalas y queremos que posean una distribución normal.

$$X_{nuevo} = \frac{X - X_{media}}{X_{max} - X_{min}}$$

Escalado por mínimo-máximo (min-max scaling)

Estrategia de escalado que substraе a cada valor el valor mіnimo del conjunto de datos y lo divide por el rango de dicho conjunto.

Es la estrategia usada cuando los valores de los diferentes campos o features tienen diferentes rangos o escalas y queremos modificar su rango, sin cambiar la forma de su distribuciоn.

$$X_{nuevo} = \frac{X - X_{min}}{X_{max} - X_{min}}$$

Estandarizaciоn (standardization o z-score normalization)

Estrategia de escalado que transforma los datos extrayendo la media y dividiendo por la desviaciоn est谩ndar.

Es un tipo de estrategia suele usarse cuando queremos asegurarnos de que la media es cero y la desviaciоn est谩ndar es uno, lo cual ayuda a transformar las features a una misma escala y a una desviaciоn est谩ndar, ayudando as{ a el modelo en el entrenamiento.

$$X_{nuevo} = \frac{X - X_{media}}{\sigma}$$

Siempre debemos tener en cuenta las posibles diferencias entre las escalas de los diferentes campos de nuestro conjunto de datos para decidir si es necesario transformarlas a escalas parecidas y facilitar así el entrenamiento de nuestro modelo de machine learning.

Codificación (*encoding*)

La codificación de datos o *encoding* hace referencia a la transformación de valores categóricos en valores numéricos.

Esta operación suele ser necesaria ya que los **diferentes algoritmos de machine learning**, por norma general, solo aceptan la ingesta de datos numéricos.

Existen múltiples estrategias de codificación; a continuación, se describen las más comunes y sus pros y contras.

1

Label/ordinal encoding

Codificación de categorías con valores enteros siguiendo un orden: 0, 1, 2... Se suele utilizar **cuando los distintos valores categóricos son ordinales** (poseen un orden concreto), aunque no es imprescindible.

PROS**CONTRAS**

Fácil codificación y bajo coste computacional. No modifica el tamaño del dataset.

PROS**CONTRAS**

Si los valores no guardan entre sí un orden natural, se introduce un sesgo de orden que puede ser malinterpretado por el algoritmo de ML.

Ordinal Encoding

Grades	Encoded
A	4
B	3
C	2
D	1
Fail	0

Label Encoding

Places	Map	Encoded
New York	1	New York
Boston	2	Boston
Chicago	3	New York
California	4	California
New Jersey	5	Boston

Ejemplos de ordinal y *label encoding*. Fuente: <https://medium.com>.

2

One-hot/dummy encoding

Codificación de cada valor único con la creación de una nueva variable. Cada nueva variable es mapeada con valores binarios de 0 o 1 indicando la ausencia (0) o presencia (1) de dicho valor.

Se suele utilizar cuando las variables o features son nominales (no tienen un orden concreto).

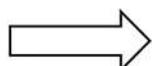
El *dummy encoding* es similar al *one-hot encoding*: para el caso de one-hot, N categorías/valores únicos crean N variables binarias nuevas; en el caso del *dummy encoding*, crea N-1 variables nuevas, es decir, elimina una *feature*, cuya ausencia puede ser inferida por la presencia del resto.

PROS	CONTRAS
La codificación no depende del orden.	

PROS	CONTRAS
	Incrementa rápidamente la dimensionalidad/número de <i>features</i> del dataset, haciéndolo más pesado y complejo. Estas nuevas <i>features</i> creadas con pocos unos y muchos ceros son esparcidas o <i>sparse</i> , las cuales incrementan el coste computacional.

One-Hot Encoding

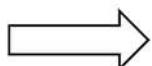
Places
New York
Boston
Chicago
California
New Jersey



	New York	Boston	Chicago	California	New Jersey
New York	1	0	0	0	0
Boston	0	1	0	0	0
Chicago	0	0	1	0	0
California	0	0	0	1	0
New Jersey	0	0	0	0	1

Dummy Encoding

Places
New York
Boston
Chicago
California
New Jersey



	New York	Boston	Chicago	California	New Jersey
New York	0	0	0	0	0
Boston	1	0	0	0	0
Chicago	0	1	0	0	0
California	0	0	0	1	0
New Jersey	0	0	0	0	1

Ejemplo de *one-hot* y *dummy encoding*. Fuente: <https://medium.com>.

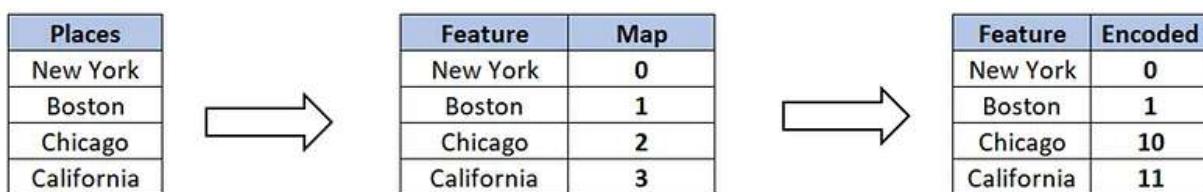
3

Binary encoding

Codificación similar al *one-hot encoding* solo que, en este caso, no se crean nuevas variables para cada categoría/valores únicos, sino que **se representan en una misma columna con valores binarios únicos.**

PROS	CONTRAS
El dataset mantiene el mismo tamaño inicial, con las ventajas que ello supone.	

PROS	CONTRAS
Se induce involuntariamente un sesgo de orden que puede llegar a ser malinterpretado por el algoritmo de ML.	

Binary Encoding

Ejemplo de *binary encoding*. Fuente: <https://medium.com>.

4

Frequency/count encoding

Codificación basada en el **conteo o frecuencia** de cada categoría.

PROS**CONTRAS**

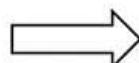
El *dataset* mantiene el mismo tamaño inicial y la codificación usa la frecuencia como un valor con información representativa y auténtica.

PROS**CONTRAS**

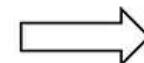
Se induce involuntariamente un sesgo de orden que puede llegar a ser malinterpretado por el algoritmo de ML. Al mismo tiempo, las categorías con frecuencias parecidas o iguales pueden llegar a ser relacionadas erróneamente por el algoritmo.

Count Encoding

Feature
Apple
Banana
Apple
Banana
Banana



Feature	Count
Apple	2
Banana	3



Feature	Encoded
Apple	2
Banana	3
Apple	2
Banana	3
Banana	3

Ejemplo de *count encoding*. Fuente: <https://medium.com>.

5

Target encoding

Codificación basada en el **cálculo del valor promedio de la frecuencia** del target/label/clase para cada categoría.

PROS**CONTRAS**

El dataset mantiene el mismo tamaño inicial y, en los valores de codificación, se incluye la información acerca de la relación que posee la categoría con su respectiva clase.

PROS**CONTRAS**

Puede llegar a producir un sobreajuste (*overfitting*), si no se utiliza con precaución, por lo tanto, se debe prestar atención al conjunto de datos sobre el que se aplica la codificación, ya que puede introducir sesgos en el conjunto de entrenamiento que no se presenten en los conjuntos de evaluación o test (de los cuales hablaremos más adelante).

Target Encoding

Feature	Target
Apple	0
Banana	1
Apple	0
Banana	0
Banana	1



Feature	Average(Target)
Apple	0
Banana	2/ 3 = 0.66



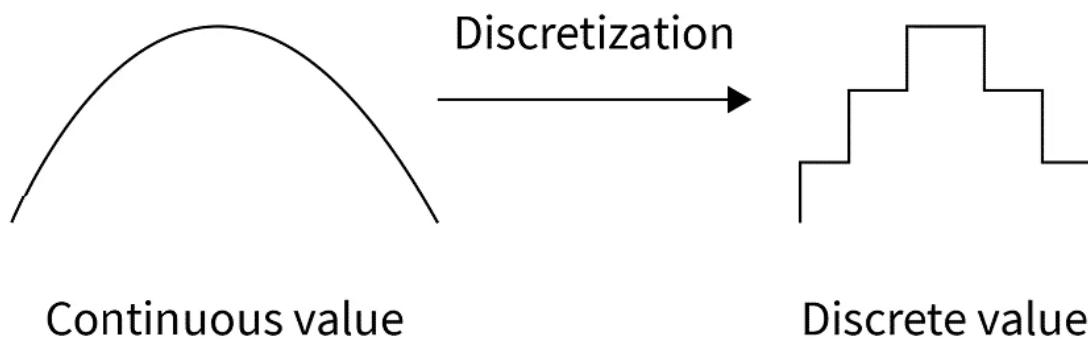
Feature	Encoded
Apple	0
Banana	0.66
Apple	0
Banana	0.66
Banana	0.66

Ejemplo de *target encoding*. Fuente: <https://medium.com>.

Recuerda: nuestro modelo de machine learning necesita información en modo de dígitos en la mayoría de los casos. Los métodos de codificación son variados y no existe uno mejor que el resto, por lo que debemos experimentar y hallar cuál es el más adecuado para nuestro uso.

Discretización (*discretization*)

La discretización o *discretization* consiste en la **conversión o transformación de variables/features continuas** (temperatura, tiempo, peso...) en discretas.



Ejemplo visual de la discretización de los valores de una variable. Fuente:
<https://www.scaler.com>.

Las **ventajas** de la discretización son varias:

- Algunos algoritmos de ML, como Naive Bayes, los árboles de decisión, etc., mejoran su rendimiento y/o decisiones cuando tratan con valores discretos, optimizando así su desempeño y su tiempo de computación.
- Los valores discretos son más fáciles de entender y es más sencillo también detectar la presencia de *outliers*.

Como **desventajas** siempre está presente la posible pérdida de información al transformar valores: las variaciones en la relación previa con otros datos o con clases/*labels*, etc.

La ventaja y desventaja al mismo tiempo de este tipo de discretización es que no afecta significativamente a la distribución de la variable.

1

Discretización por **misma frecuencia o por quantiles** (*equal-frequency* o *quantile discretization*)

Que consiste en **ordenar los valores continuos de la variable en intervalos** con el mismo número de observaciones o frecuencia. La diferencia con el método de discretización por misma anchura o uniforme es que, aunque el tamaño de los bins, no sea el mismo para todos los bins, cada bin posee el mismo número de valores.

La **ventaja** de este tipo de discretización es que es útil para la detección de datos extremos o outliers, ya que estos valores terminarán agrupándose en los mismos grupos o bins dada su frecuencia.

2

Discretización avanzada con uso de algoritmos de ML

Es que existen otros tipos de discretización más avanzados que requieren el uso de algoritmos de ML. Aquí presentamos los más populares.

Clustering con K-means

El algoritmo es capaz de crear k grupos o *clusters*. Su funcionamiento se explicará más adelante en la sección de aprendizaje no supervisado o *unsupervised learning*.

Los árboles de decisión

Un algoritmo de este tipo es capaz de evaluar todos los posibles valores dentro de una variable y seleccionar el ‘punto de corte’ que crea intervalos óptimos y coherentes. El funcionamiento de los algoritmos basados en árboles de decisión se explicará cuando hablemos del aprendizaje supervisado (*supervised learning*).

La discretización es importante si queremos facilitar el entrenamiento de nuestro modelo de machine learning, por lo que siempre debemos considerar aplicarla.



Dispones de más información y de ejemplos prácticos sobre estos y otros métodos de discretización en el siguiente [enlace](#).

Reducción dimensional (*dimensionality reduction*)

La reducción de dimensiones consiste en limitar el número de variables o *features* en un *dataset* para preservar la información relevante, aquella que soluciona un problema a través de una solución de ML.

Esta reducción puede ser lineal o no lineal, dependiendo del método usado. Existen dos componentes en la reducción de dimensiones:

Selección de features/variables (*feature selection*)

Trata de encontrar un *subset* de variables más pequeño que el original que pueda utilizarse en un modelo de ML y que resuelva el problema en cuestión. Se disponen de diferentes métodos para lograrlo, por ejemplo:

- El método de filtrado (*filter methods*): selecciona un *subset* de *features* basado en su relación estadística con el *target/etiqueta/label*, usando test o técnicas estadísticos (chi-cuadrado, ANOVA...) y medidas estadísticas como la correlación entre variables (Pearson, Spearman's Fho, Kendall Tau...).
- Método de envoltura (*wrapper methods*): selecciona un *subset* de *features* en base a su mejor desempeño al compararlo con otros *subsets*, y cada *subset* se usa para entrenar un modelo y evaluar sus resultados. Algunos de estos métodos son *backward selection* (partiendo del conjunto completo de *features*, se comprueba el desempeño con la eliminación de una *feature* cada vez sistemáticamente), *forward selection* (igual que el anterior, pero partiendo de una *feature* hasta el set completo de *features*), *recursive feature elimination* (similar a *backward selection* solo que se toma la decisión de desempeño en base a la importancia de las *features* y sin utilizar métricas).
- Métodos embebidos (*embedded methods*): consiste en aplicar una regresión lineal regularizada, la cual dará un peso u otro a cada *feature* según su importancia en el modelo. Se suele utilizar regresión tipo LASSO (Least Absolute Shrinkage and Selection operator).

Extracción de features/variables (feature extraction)

Trata de reducir las dimensiones presentes en el conjunto de datos o *dataset* desde un espacio dimensional alto a uno con menor número de dimensiones. Estos son algunos métodos utilizados:

- Análisis principal de componentes (*Principal Component Analysis* o PCA): método que busca un espacio dimensional más bajo preservando la varianza del espacio dimensional original. Este método es no supervisado y aplica transformaciones lineales, por lo que no es efectivo si las distribuciones de las *features* de nuestro *dataset* son no-lineales.
- Análisis discriminatorio lineal (*Linear Discriminant Analysis* o LDA): método que se usa para encontrar una combinación lineal de *features* en un *dataset*. Al igual que el PCA, aplica transformaciones lineales, pero es un método supervisado, por lo que tiene en cuenta las etiquetas/*labels*/clases del *dataset*. Además, asume que las matrices de covarianza son iguales.
- Análisis discriminatorio generalizado/gaussiano (*Generalized/Gaussian Discriminant Analysis* o GDA): método basado en el LDA que asume que las diferentes clases siguen una distribución gaussiana o normal, además de no asumir que las matrices de covarianza son iguales.

No siempre todas las *features* dentro del conjunto de datos tienen que ser usadas para el entrenamiento de un modelo de machine learning: a veces **menos es mejor**. La reducción de dimensiones es una opción para tener en cuenta en estos casos.

 Para una explicación más amplia y detallada y con demostraciones prácticas, recomendamos visitar estos enlaces: [métodos de selección de funciones](#), [reducción de dimensionalidad](#) y [método de selección de funciones de aprendizaje automático](#).

Ingeniería de variables (*feature engineering*)

La ingeniería de variables o *feature engineering* consiste en **la creación de nuevas variables o features a partir de las features originales**, con el objetivo de obtener features más valiosas o significativas que ayuden a mejorar el desempeño del modelo de ML.

Requiere de **análisis e imaginación** en cuanto a deducir información adicional que puede extraerse de las features originales y que pueda proporcionar mejores resultados, teniendo en cuenta el modelo de ML a usar y la tarea a resolver, aparte del contexto completo del problema y los datos.

La ingeniería de variables/*feature* suele usarse como el proceso completo que engloba todas las secciones anteriores o como sinónimo del *data preparation*, lo que puede llevar a confusión, aunque el objetivo sea el mismo. En este caso, hemos optado por utilizar el término como 'creación de variables'.

Hemos podido comprobar que puede llegar a ser necesario transformar los datos para que estos aporten más valor al modelo de ML y, por tanto, a la resolución de la tarea en cuestión, ya sea utilizando el escalado de datos, la codificación, **la discretización o la reducción dimensional**, aplicándose cada método según el caso de uso concreto y de las características del conjunto de datos disponible.

Del mismo modo, **la ingeniería de variables trata de transformar los datos** en algo nuevo y con igual o más valor.

Separación de datos (*data splitting*)

1

Subsets de entrenamiento, validación y test (*train, validation, test*)

Una vez poseemos los datos limpios/preprocesados para entrenar un modelo de machine learning, es necesario realizar un último paso que consiste en su división en diferentes grupos o subsets: **la separación o *splitting***. Normalmente esta división se realiza de estas maneras:

Entrenamiento (*train*) —

Es el *subset* del conjunto de datos el que se usará para entrenar al modelo de ML en el reconocimiento de patrones y relaciones entre los datos de entrada o inputs. Normalmente es el subset más grande, conteniendo un 80–90% del total del conjunto de datos original, según el caso de uso.

Validación (*validation* o *val*) —

El *subset* del conjunto de datos se usará para evaluar el desempeño o performance del modelo de ML durante el entrenamiento: le ayudará a ajustar los hiperparámetros del algoritmo en cuestión. Al mismo tiempo, el *subset* de validación ayuda a prevenir el sobreajuste o *overfitting*: cuando el modelo de ML no es capaz de generalizar más allá de los datos de entrenamiento, invalida los resultados obtenidos con nuevos datos no vistos previamente.

Test

El *subset* de los datos se usa para evaluar el desempeño del modelo entrenado. Este *subset* solo se utiliza una vez.

Con esta división de los datos en *subsets* podemos evaluar el nivel de desempeño de un modelo de machine learning con datos nuevos que no ha podido ver durante el entrenamiento. Si no realizamos esta separación o *splitting*, se incrementan las probabilidades de que el desempeño del modelo sea pobre: solo habrá aprendido acerca de los datos concretos del entrenamiento, en vez de aprender patrones y relaciones entre los mismos que permitan al modelo generalizarlos a nuevos datos.

Estrategias de separación (*splitting*)

Existen diferentes **estrategias de separación o *splitting* de los datos** y su elección depende, como siempre, en el caso de uso concreto y en el conjunto de los datos. Veamos algunas de ellas.

Sampleo aleatorio (random sampling)

En el sampleo aleatorio o *random sampling* los datos son separados en *subsets* de forma aleatoria. Se suele aplicar en conjuntos de datos grandes representativos de la población que se quiere modelar, así como cuando no existen relaciones aparentes en los datos que sugieran la búsqueda de un método más especializado.

Sampleo estratificado (stratified sampling)

En el sampleo estratificado o *stratified sampling*, los datos son separados en *subsets* basados en las etiquetas/clases/*labels* u otras características seguidos de una división aleatoria. Esta estrategia se aplica en el caso de *datasets* desbalanceados, cuando las observaciones pertenecientes a una clase exceden significativamente a las de otra: el sampleo estratificado ayudará a que los diferentes *subsets* tengan una distribución similar de observaciones para cada una de las clases.

Sampleo basado en el tiempo (time-based sampling)

En el sampleo basado en el tiempo o *time-based sampling* los datos son separados en función de diferentes rangos de fechas. Esta estrategia se aplica cuando el tiempo/fecha de las observaciones es importante, evitando que datos del 'pasado' (datos usados para entrenar el modelo) no aparezcan en el 'futuro' (datos usados para evaluar o testear el modelo) o viceversa.

Validación cruzada (cross-validation)

En la validación cruzada o *cross-validation* los datos son separados en múltiples *subsets* o *folds*. Algunos de estos *folds* se usan para entrenar el modelo mientras otros se usan para evaluar su desempeño.

La ventaja principal de la validación cruzada es que podemos usar todos los datos disponibles.

A su vez existen **diferentes técnicas de validación cruzada**, nosotros vamos a ver las más comunes.

HOLD-OUT CROSS VALIDATION

K-FOLD CROSS-VALIDATION

LEAVE-ONE-OUT CROSS-VALIDATION

Este método sería el mismo que el anteriormente comentado sampleo aleatorio, dividiendo el *dataset* en entrenamiento, validación y test o en entrenamiento y test.

**HOLD-OUT CROSS
VALIDATION****K-FOLD CROSS-VALIDATION****LEAVE-ONE-OUT CROSS-
VALIDATION**

El conjunto de datos se separa en k partes/*folds* del mismo tamaño. Se realizan n iteraciones donde n es el número de *folds*: en cada iteración se utiliza un *fold* como test y el resto como entrenamiento. Con este acercamiento utilizamos todos los datos como test eventualmente, disminuyendo la varianza, generalizando mejor el modelo y, en consecuencia, consiguiendo un modelo más robusto/fiable.

**HOLD-OUT CROSS
VALIDATION****K-FOLD CROSS-VALIDATION****LEAVE-ONE-OUT CROSS-
VALIDATION**

El modelo de ML es entrenado n veces donde n es el tamaño de observaciones en el *dataset*. En cada iteración, solo una muestra/observación es usada como test y el resto se utiliza como entrenamiento. La ventaja de este acercamiento es la de utilizar el máximo número de observaciones posibles para el entrenamiento, siendo más apropiado en caso de *datasets* pequeños al tener que realizar muchos más entrenamientos que con el método anterior.



Puedes encontrar más información sobre los métodos de validación cruzada en los siguientes enlaces: <https://neptune.ai> y <https://www.turing.com>.

Recuerda: la separación del conjunto de datos en diferentes partes, *splits* o *folds* es importante para que nuestro modelo obtenga el mejor desempeño posible, además de maximizar así el valor y el uso de los datos que disponemos.

Conclusiones



El machine learning es el área de la inteligencia artificial que aprende por sí mismo las relaciones entre los datos que se le proporcionan y el resultado que esperamos de él. Si le proporcionamos datos (numéricos, texto, imágenes, audios...) nos permite resolver problemas de diversos tipos (clasificación, regresión, generación, análisis...) que acaban evolucionando en soluciones prácticas y útiles (detección de enfermedades, conducción autónoma, recomendador de viviendas...), cuyo impacto siempre debemos evaluar desde varios puntos de vista como el ético, legal, social, etc.

Por otro lado, los datos son la primera pieza y uno de los pilares en el desarrollo de una solución de machine learning: en ellos está contenida la información que permitirá a un algoritmo de ML aprender y, por tanto, poder solucionar el problema o tarea en cuestión.

- Por ello, es necesario **analizarlos con atención**: desde considerar su formato, tamaño y composición hasta la información contenida en ellos (tipo de valores, distribución, escalas...) pudiendo transformarlos o incluso crear nuevos.
- Este preprocessado de los datos **nos permitirá asegurarnos de que su calidad sea la mayor posible**, evitando futuros problemas técnicos o computacionales en el entrenamiento del modelo de ML, así como en su evaluación y análisis de resultados finales con el paso final de su partición en diferentes sets o *folds*.
- El tiempo empleado en el análisis y la preparación de datos nunca es malgastado si se utiliza con **criterio y sentido**: solo así estaremos preparados para empezar a entrenar propiamente un modelo de machine learning.

Recursos útiles



-
- Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow by Geron Aurelien (Oreilly).
[https://www.oreilly.com/library/view/hands-on-machine-learning/9781492032632/.](https://www.oreilly.com/library/view/hands-on-machine-learning/9781492032632/)
 - The Hundred-Page Machine Learning Book by Andriy Burkov.
[https://themlbook.com/.](https://themlbook.com/)
 - Machine Learning Engineering, de Andriy Burkov.
[https://www.mlebook.com/wiki/doku.php.](https://www.mlebook.com/wiki/doku.php)
 - Python for Data Analysis, 3E. [https://wesmckinney.com/book/.](https://wesmckinney.com/book/)
 - Amplio repositorio de librerías, frameworks y software de machine learning:
[https://github.com/josephmisiti/awesome-machine-learning.](https://github.com/josephmisiti/awesome-machine-learning)

¡Enhорabuena! Fastbook superado



Qualentum.com