

UNIVERSIDAD
COMPLUTENSE
DE MADRID



ntic
master
revolucionamos la comunicación

Autor: María José
Gómez Silva

Machine Learning con Python

INTRODUCCION.

La primera parte (semana 1) del módulo de Machine Learning con Python se divide en tres capítulos.

En el primer capítulo de *Introducción al Machine Learning* se estudiarán los conceptos básicos necesarios para entender e implementar métodos de Aprendizaje Automático o Machine Learning. También introduciremos la herramienta Jupyter Notebook y realizaremos un repaso general de las principales librerías usadas en el desarrollo de algoritmos de Machine Learning.

En el segundo capítulo, titulado como *Ingesta de Datos*, se explorarán distintos tipos y fuentes de datos. Además, se estudiarán casos reales de obtención de datos y como convertirlos en *DataFrames*, y se aprenderá a emplear mecanismos de organización, estructuración, análisis y visualización de los datos con ejemplos prácticos.

En el tercer capítulo de *Preprocesado* se estudiarán y aplicarán distintas técnicas de procesado de datos para eliminar el ruido en ellos, mejorar su calidad, su capacidad representativa, o aumentar su cantidad y variedad. Entre las técnicas que se tratarán se encuentra el filtrado, balanceo y una amplia variedad de métodos de aumento de datos.

Para que el modelo pueda aprender de forma más sencilla

UNIVERSIDAD
COMPLUTENSE
DE MADRID



ntic
master
revolucionamos la comunicación

Autor: María José
Gómez Silva

Machine Learning con Python. Semana 1.

CAPÍTULO 1. INTRODUCCIÓN AL MACHINE LEARNING.

El Machine Learning, o Aprendizaje Automático en castellano, es una rama de la Inteligencia Artificial que desarrolla métodos y algoritmos para hacer que las máquinas aprendan a resolver una determinada tarea de forma automática, sin que la solución tenga que ser explícitamente programada.

En este capítulo se presentan algunas de las aplicaciones, modelos y tipos de aprendizajes más empleados. Además, se estudiarán los fundamentos teóricos de los mecanismos de Machine Learning y algunos conceptos clave para su desarrollo. Y por último se describirá el entorno de programación necesario para su implementación.

Un robot industrial no es Machine Learning porque hay que programarle paso a paso

1.1 ¿Qué es el Machine Learning? Aplicaciones

¿QUÉ ES EL MACHINE LEARNING?

El Aprendizaje Automático o Aprendizaje de Maquina, más conocido por su nombre en inglés, Machine Learning, es una rama de las ciencias de la computación y de la inteligencia artificial, cuyo objetivo es desarrollar técnicas que permitan que las computadoras aprendan. Dicho aprendizaje permite que las máquinas, o programas informáticos, realicen una tarea sin estar explícitamente programados para ellas.

El término Machine Learning fue acuñado en 1959 por Arthur Samuel, un ingeniero americano, pionero en el campo de los videojuegos y la Inteligencia Artificial. Su programa para jugar al ajedrez con el computador fue una de las primeras demostraciones de los fundamentos de la Inteligencia Artificial. Arthur Samuel definió el Machine Learning como “el campo de estudio que le da a los computadores la habilidad de aprender sin ser explícitamente programados”.

En los años 80, el informático estadounidense, Tom Mitchell proporcionó una definición más precisa y moderna del Machine Learning. Según sus palabras, “se dice que un programa de ordenador aprende de una experiencia E con respecto a alguna clase de tarea T y un rendimiento medido, R , si el rendimiento en la tarea T , medido por R , mejora con la experiencia E ”.

Con cada experiencia e rendimiento aumenta

Por ejemplo, si la tarea T del computador fuese jugar al ajedrez (como en el programa de Arthur Samuel), gracias a los métodos de Machine Learning, su rendimiento R, medido como la probabilidad de ganar una partida, aumentaría con la experiencia E adquirida con cada partida jugada.

La experiencia es adquirida por las máquinas mediante el uso de datos. Los algoritmos de Machine Learning se encargan de observar datos, construir un modelo basado en esos datos y utilizar ese modelo como una hipótesis a cerca de la tarea o problema concreto a resolver.

Según muchos investigadores, el Machine Learning es la base para construir máquinas inteligentes y progresar hacia la Inteligencia Artificial de nivel humano.

APLICACIONES DEL MACHINE LEARNING

En las últimas décadas las técnicas de Machine Learning han sido aplicadas en infinidad de campos muy variopintos como la ingeniería, la bio-medicina, la informática, el transporte, el entretenimiento, el comercio, o la robótica entre otros.

Esto ha impulsado el desarrollo de tecnologías como las de los vehículos autónomos, el reconocimiento del discurso, la percepción y el control en la construcción de robots, el entendimiento de textos, que a su vez han permitido la búsqueda efectiva en la web, la visión por computador, el minado de bases de datos, o incluso un entendimiento vastamente mejorado del genoma humano.

En la actualidad, los algoritmos de Machine Learning están tan omnipresentes que probablemente la mayoría de las personas los usamos docenas de veces al día sin saberlo. Por ejemplo, el filtro de “spam” de nuestro email funciona gracias a que ha aprendido a clasificar los correos, como spam o no, con técnicas de Machine Learning. Cada vez que hacemos una búsqueda en internet, el buscador web emplea un software que usa Machine Learning para hacer el ranking de las páginas de mayor interés. Cuando las aplicaciones de redes sociales reconocen a nuestros amigos en nuestras fotos, también lo están haciendo gracias al Machine Learning.

- Robótica: Percepción, control
- Seguridad: Accesos inteligentes, Video-vigilancia
- Transporte: Sistemas de asistencia a la conducción, Vehículos autónomos
- Informática: Búsqueda web, filtro anti-spam
- Entretenimiento: Realidad aumentada/virtual, filtro fotográficos
- Comercio: Análisis de estadísticas y búsquedas de patrones, publicidad inteligente según intereses personalizados
- Bio-medicina: Reconocimiento de patrones, diagnóstico asistida

UNIVERSIDAD
COMPLUTENSE
DE MADRID



ntic
master
revolucionamos la comunicación

Autor: María José
Gómez Silva

Machine Learning con Python. Semana 1.

CAPÍTULO 1. INTRODUCCIÓN AL MACHINE LEARNING.

1.2 ¿Qué es un Modelo?

¿QUÉ ES UN MODELO?

En el ámbito de la ciencia y la tecnología, un modelo es una representación abstracta, conceptual o matemática de un componente, un sistema, un proceso o una tarea.

Los modelos son interpretaciones que intentan conceptualizar o dar un sentido a los datos observados. En términos estadísticos, con los modelos se pretende generalizar el comportamiento de la población global a partir del estudio de un conjunto de muestras. Los modelos son muy útiles para realizar predicciones.

El modelo recibe datos de entrada, y ofrece un dato o varios de salida. Esa salida, también conocida como *predicción* del modelo, es la respuesta o solución que el modelo ofrece para los datos recibidos a la entrada.

Por ejemplo, imaginemos que un empresario realiza varias campañas publicitarias, cada año varía el número de minutos al día que su producto es anunciado en la radio, y cada año registra el número de ventas, obteniendo una gráfica como la de la Figura 1. A partir de los datos registrados (muestras), intenta ajustar un modelo o función (en rojo), donde la entrada es el número de minutos, y la salida el número de productos vendidos (en millones). El modelo ajustado permitirá al empresario realizar predicciones sobre el resultado de ventas que se obtendría para un número distinto de minutos en la radio, sin necesidad de probarlo, es decir permite conocer el resultado del proceso (su salida) frente a datos de entrada nuevos (desconocidos). Cuanto mejor ajustado esté el modelo, más precisa será la predicción.

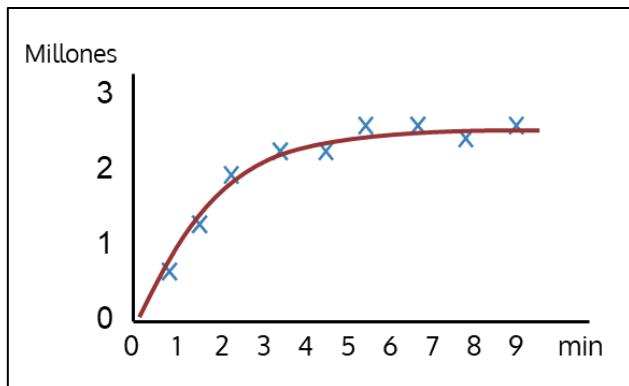


Figura 1.: Ajuste de un modelo de la cantidad de productos vendidos en función de los minutos diarios de publicidad en la radio.

El Machine Learning como su nombre indica, permite un aprendizaje automático del modelo que mejor se ajusta a los datos suministrados. Se trata de encontrar el modelo más representativo, es decir, el que mejor rendimiento presenta en la realización de la tarea.

Cuando la tarea que la maquina debe aprender es compleja, como por ejemplo jugar al ajedrez, puede ocurrir que el modelo no sea simplemente una función, sino el conjunto de funciones, ecuaciones matemáticas, y pasos que la máquina debe realizar para resolver la tarea.

Al inicio del proceso de aprendizaje, la máquina no conoce cuál es el modelo a seguir. En el caso del ajedrez, no sabe a qué piezas o movimientos debe prestar atención, ni qué movimiento realizar para obtener ventaja en la partida. Por ello, al inicio del aprendizaje, la maquina plantea una *hipótesis* (modelo inicial) de cómo resolver la tarea. Por ejemplo, en el caso del ajedrez, mover siempre la pieza que esté más a la derecha y al frente de las que tenga disponibles. Probablemente, la hipótesis inicial sea incorrecta, o tenga poco rendimiento, y haya que hacer correcciones de la misma. Si se tiene alguna idea o intuición previa de cómo resolver la tarea, la hipótesis puede tratarse de una aproximación inicial al modelo buscado, que poco a poco se irá *ajustando* con el aprendizaje.

A medida que la maquina aprende, el modelo es modificado (ajustado), para aumentar el rendimiento con que realiza la tarea en cuestión. Para medir el rendimiento del modelo, se observa su salida frente a datos conocidos como datos de validación. Cuando se alcanza un rendimiento razonable, se dice que el aprendizaje ha terminado y se da el modelo por aprendido. Ya no hablaríamos de hipótesis, sino de *modelo*. Ese modelo aprendido es el que se guardará y se empleará para resolver la tarea de forma automática en el futuro.

UNIVERSIDAD
COMPLUTENSE
DE MADRID



ntic
master
revolucionamos la comunicación

Autor: María José
Gómez Silva

Machine Learning con Python. Semana 1.

CAPÍTULO 1. INTRODUCCIÓN AL MACHINE LEARNING.

1.3 Tipos de Aprendizajes y tipos de Modelos

TIPOS DE APRENDIZAJE

El proceso de aprendizaje en una máquina es **iterativo**, es decir, consiste en la **repetición de ciclos, y depende de los datos de ejemplo** de los que se disponga. Mediante la observación iterativa de múltiples ejemplos la máquina o programa, en cada ciclo, adquiere experiencia en la realización de una tarea, y trata de aprender de sus errores para corregir su modo de realizar la tarea y así mejorar su rendimiento en el ciclo siguiente.

El **proceso de aprendizaje** a menudo es llamado **entrenamiento**, debido a la naturaleza iterativa del proceso, en el que **la mejora es progresiva, y** la experiencia es adquirida con numerosos “intentos” de realización de la tarea.

Dependiendo de la tarea a resolver y de los datos disponibles para abordarla, se puede elegir entre diferentes tipos de aprendizaje. Estos son: aprendizaje supervisado, aprendizaje no supervisado, y aprendizaje por refuerzo.

Aprendizaje Supervisado

El aprendizaje supervisado es aquel en el que el proceso de mejora del rendimiento es guiado por la **comparación de la predicción** que ofrece el modelo **con respecto a la solución o salida real** que se espera obtener.

Para realizar este tipo de entrenamiento es necesario **disponer de** ejemplos tanto de datos de **entradas** como de sus correspondientes **salidas**. En otras palabras, se necesitan ejemplos para los cuales ya conocemos el resultado o salida real. Esta salida o solución conocida, que sirve como referencia, recibe diversos nombres en la literatura. Algunos de ellos son: **etiqueta, anotación, o ground truth**, y a la labor de registrarlas, se le denomina etiquetado o anotación de los datos.

Por ejemplo, si queremos entrenar, con un aprendizaje supervisado, un modelo que determine si en una fotografía aparece el cielo o no, necesitaremos múltiples fotografías de ejemplo (unas con cielo y otras sin él) y además las etiquetas correspondientes, es decir, debemos pasarle al algoritmo de aprendizaje la solución para cada imagen, o lo que es lo mismo la información de si tiene cielo o no. En el ejemplo anterior del empresario que anunciaba su producto en la radio, los minutos de publicidad eran los datos de entrada, y para cada ejemplo de entrada, su etiqueta asociada era el volumen de ventas correspondiente.

Lo que finalmente se espera del modelo es que ofrezca predicciones similares a las etiquetas tomadas como referencia.

En conclusión, el aprendizaje supervisado necesita conjuntos de datos etiquetados, es decir, le tenemos que decir al modelo qué es lo que queremos que aprenda, como se muestra en la Figura 1.

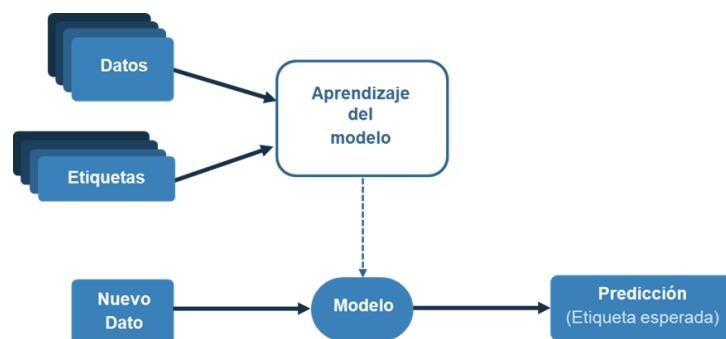


Figura 1. Esquema de aprendizaje supervisado

Dependiendo del tipo de etiqueta, dentro del aprendizaje supervisado existen dos tipos de modelos: los de **regresión**, dónde la etiqueta es una **variable continua** y los de **clasificación**, dónde la etiqueta es **discreta**. Ambos modelos serán explicados con mayor detalle en la siguiente sección.

Aprendizaje No Supervisado

A diferencia del aprendizaje supervisado, en el no supervisado **no se sigue una referencia de la que aprender**. El aprendizaje no supervisado emplea datos que no han sido etiquetados ya que **no trata de predecir un determinado valor a la salida**. Este tipo de entrenamiento se emplea para resolver tareas consistentes en el **análisis de los datos** para estructurarlos o **agruparlos por afinidad**, como se indica en la Figura 2.

Este tipo de aprendizaje no se emplea para predecir una salida a partir de los datos de entrada. No predice una solución, ni clasifica los datos de entrada con una etiqueta, sino que extrae nuevo conocimiento a cerca de los datos de entrada, los organiza, agrupa, o incluso simplifica, extrayendo la información fundamental de los mismos.

Algunos ejemplos de modelos que emplean aprendizaje no supervisado son los de agrupación (clustering), los de detección de anomalías, los asociativos y los de reducción de la dimensionalidad. Que serán definidos en la siguiente sección.

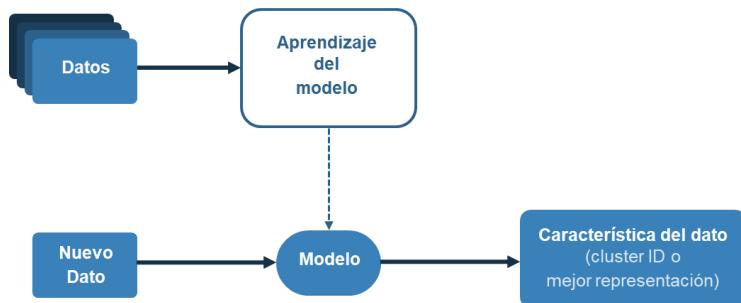


Figura 2. Esquema de Aprendizaje No Supervisado

Aprendizaje por Refuerzo

El aprendizaje por refuerzo se basa en recompensar los comportamientos deseados y penalizar los no deseados. Este aprendizaje se basa en la existencia de un *agente* (maquina o programa) que realiza acciones o toma decisiones, y un *ambiente* o entorno al que afectan las acciones y decisiones del agente, como se representa en la Figura 3. Según el efecto de las acciones del agente en el entorno, éstas serán recompensadas o penalizadas. El agente debe disponer de mecanismos para percibir e interpretar el entorno, y mediante la ejecución de acciones aprenderá a través de prueba y error.



Figura 3. Esquema de Aprendizaje por Refuerzo.

En lugar de emplear etiquetas de referencia para guiar el aprendizaje, este método fija objetivos a lograr, y asigna recompensas y penalizaciones a determinadas acciones. Concretamente se asignan valores positivos a las acciones deseadas para animar al agente a realizarlas y valores negativos a los comportamientos no deseados. Con el tiempo, el agente aprende a evitar lo negativo y buscar lo positivo.

Este tipo de aprendizaje lleva al agente a buscar recompensas globales a largo plazo para alcanzar una solución óptima. Los objetivos a largo plazo ayudan al agente a no

estancarse en objetivos menores. Si el agente únicamente valorase la recompensa a corto plazo podría quedarse estancado repitiendo la última acción con la que recibió una buena recompensa. Por ello, se debe lograr un **equilibrio entre explorar lo desconocido y explotar los recursos en el ambiente.** Eso es conocido como el dilema de exploración/explotación.

En el aprendizaje por refuerzo no se requieren etiquetas para los datos, por lo tanto, no es de tipo supervisado. Sin embargo, tampoco es de tipo no supervisado, ya que en el aprendizaje por refuerzo se persigue un objetivo a partir de los datos, y no la estructuración o agrupación de los mismos. Este método de aprendizaje puede entenderse como una forma de dirigir el aprendizaje automático no supervisado hacia un objetivo mediante recompensas y penalizaciones, y sin necesidad de etiquetar explícitamente los datos.

El aprendizaje por refuerzo se emplea en áreas como la **robótica, la optimización y el control de sistemas.** Destaca su aplicación en el ámbito del **desarrollo de videojuegos.** Por ejemplo, Pacman es un juego basado en aprendizaje por refuerzo. El agente recibe información sobre las reglas del juego y aprende a jugar por sí mismo. Al principio, se comporta de manera aleatoria, pero a medida que aprende realiza movimientos más sofisticados.

Uno de los métodos de aprendizaje por refuerzo más conocido y empleado es el ***Q-learning***, que trata de aprender un conjunto de reglas o normas de comportamiento llamada **política**. Recibe su nombre, por la inicial Q de la función de calidad, “quality” en inglés, ya que esta función devuelve **la recompensa y representa la “calidad” de la acción tomada.**

TIPOS DE MODELOS O TAREAS EN MACHINE LEARNING

Regresión

Los modelos de regresión crean un mapa entre las variables de entrada y una función continua, por lo tanto, permiten predecir una salida que toma valores reales continuos. Los modelos de regresión se entrena mediante **aprendizaje supervisado.**

Un tipo de regresión ampliamente empleado es la **regresión lineal**, donde la función que relaciona la salida con la o las entradas es lineal. En el caso concreto de tener una única variable de entrada, la ecuación que define la salida del modelo en función de la entrada sería la de una línea recta, como se observa en la Figura 4.

Un ejemplo de problema de regresión sería el de predecir el precio de venta de una vivienda en función de su tamaño, a partir de datos de ventas previas.

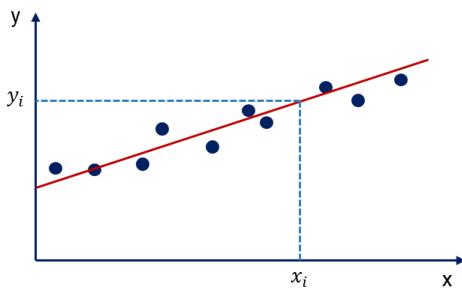


Figura 4. Modelo de regresión lineal.

Clasificación

Los modelos de clasificación mapean las variables de entrada en categorías discretas (también llamadas clases), es decir, clasifican una determinada muestra a partir de las variables de entrada tomadas. Estos modelos predicen una salida que toma valores discretos. Estos valores discretos son las etiquetas correspondientes a cada una de las clases posibles. Los modelos de clasificación se entran con aprendizaje supervisado.

Si el número de clases posibles es sólo de dos, hablamos de un modelo de clasificación binaria, como el que se muestra en la Figura 5. Si el número de clases es mayor que dos, se trata de una clasificación multiclasa.

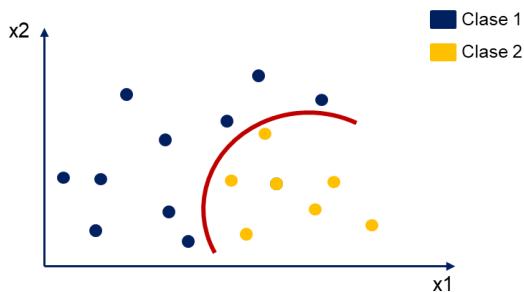


Figura 5. Modelo de clasificación binaria.

El ejemplo anterior de predicción del precio de una vivienda se puede transformar en un problema de clasificación haciendo que la salida del modelo sea si la vivienda se vende por más o menos del precio pedido. Se trata de una clasificación de las casas en dos categorías discretas basada en el precio.

Clustering

El clustering es la agrupación de los datos basada en las relaciones entre las variables de entrada que se toman de los datos, véase la Figura 6. Estos modelos no atienden a ningún tipo de etiqueta o referencia, por lo que se trata de métodos de aprendizaje no supervisado.

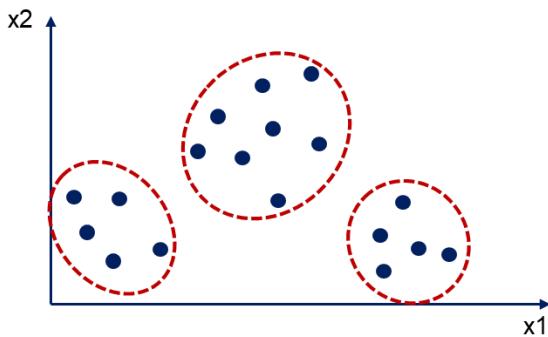


Figura 6. Modelo de clustering

Un ejemplo de problema de clustering sería el caso en el que tenemos una colección de 2000 encuestas sobre la economía de una región y se busca una manera de agrupar las encuestas automáticamente en un número pequeño de grupos de encuestas similares, relacionadas por diferentes variables como la frecuencia de algunas palabras o longitud de los párrafos.

Detección de Anomalías

Los detectores de anomalías se basan en la asunción de que la mayoría de los datos tomados tienen un comportamiento o valores normales. Los modelos de detección de anomalías no requieren de etiquetas de los datos, ya que modelan la distribución de probabilidad de todos los datos de entrada.

Si llegan datos, que, de acuerdo con el modelo, presentan poca probabilidad, estas muestras serán considerados como anómalas, como la muestra amarilla de la Figura 7.

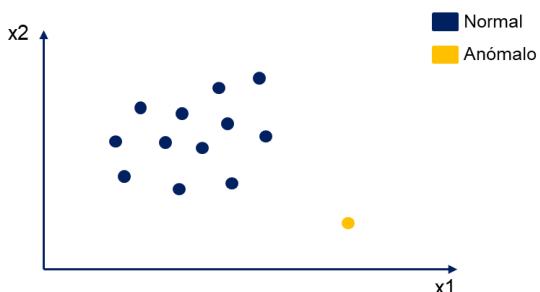


Figura 7. Modelo de detección de anomalías

Un ejemplo de problema de detección de anomalías sería el caso en el que se identifica un producto defectuoso por los valores anómalos de algunas de sus características, que difieren mucho del resto (la mayoría) de productos en buenas condiciones.

Aprendizaje Asociativo

Los modelos asociativos, como su nombre indica, asocian una nueva muestra con otras previamente estudiadas por similitud (similar al clustering), y además si las muestras previas fueron etiquetadas con anterioridad (han sido clasificadas), esa etiqueta también es asociada a la nueva muestra. Puede entenderse como una combinación de clustering y clasificación. En la Figura 8 se observa como una nueva muestra en azul claro ha sido asociada con una muestra anterior etiquetada como grupo verde.

Por ejemplo, un caso de aprendizaje asociativo sería el realizado por un doctor que a lo largo de años de experiencia forma asociaciones entre características de pacientes y diagnósticos de enfermedades confirmados. Si un nuevo paciente aparece, se basa en sus características para asociar una posible enfermedad, de acuerdo con la similitud con los pacientes previos. Difiere de un problema de clasificación, donde se buscaría una función que directamente relacionase las características del paciente con distintas enfermedades.

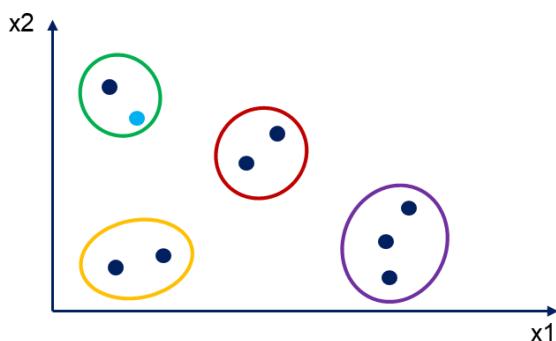


Figura 8. Modelo de aprendizaje asociativo.

Reducción de dimensionalidad

Los métodos de reducción de dimensionalidad seleccionan o combinan las variables tomadas de los datos para reducir el número de las mismas, véase la Figura 9. La reducción de dimensionalidad suele ser un paso previo al entrenamiento de otros modelos. Las razones para tratar de disminuir la dimensionalidad de los datos de entrada son múltiples: reducir el tiempo de entrenamiento de los algoritmos, mejorar el rendimiento del modelo o facilitar la representación visual de los datos.

Los algoritmos de reducción de dimensionalidad emplean métodos estadísticos y matemáticos para transformar la base de datos original en una nueva con menos dimensiones. El inconveniente es que en el proceso se pierde algo de información. Sin embargo, para minimizar la relevancia de la información perdida, estos métodos realizan un análisis de componentes, es decir, un análisis de las variables de entrada disponibles. A partir de ese análisis seleccionan solamente las variables principales o más características, como el método PCA (Principal Component Analysis).

qué variable es más representativa de los datos

Cuál tiene mayor varianza

Elimino los valores de la variable que menor varianza tiene y menos representativa es

Si las 2 variables son representativas, creo otra variable nueva que es la combinación de las 2 variables -> PCA

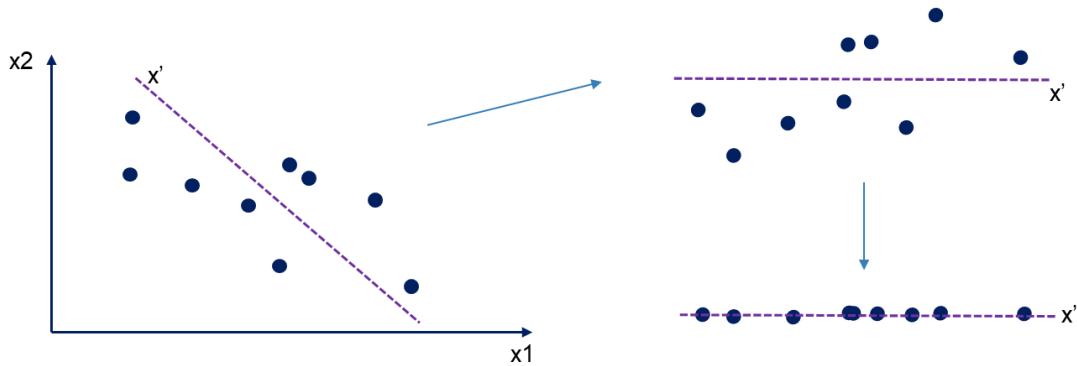


Figura 9. Reducción de la dimensionalidad de los datos

Resumen de tipos de Aprendizaje

La Tabla 1 resume los tipos de aprendizaje y de modelos de Machine Learning definidos en las secciones previas:

Tabla 1. Tipos de aprendizajes y tipos de modelos en Machine Learning

Tipos de Aprendizaje	Tipos de modelos o tareas	Ejemplos de métodos.
Supervisado	Regresión	<i>Logistic regression</i>
	Clasificación	<i>SVM, decision tree</i>
No Supervisado	Clustering	<i>k-means, hierarchical clustering</i>
	Detección de anomalías	
	Aprendizaje Asociativo	
	Reducción de dimensionalidad	<i>PCA</i>
Por Refuerzo		<i>Q-learning</i>

Dependiendo de las características del conjunto de datos del que disponemos se debe seleccionar el tipo de aprendizaje y de modelo que mejor se ajuste a las necesidades del objetivo perseguido.

1. Encuestas a clientes y cuántos perfiles distintos de clientes hay -> Clustering (No supervisado)
2. Predecir la altura de un niño en función de su edad -> Regresión (Supervisado)
3. Diagnosticar la avería de un vehículo en función de su comportamiento y el historial de otros vehículos arreglados previamente -> Asociación (No supervisado)
4. Predecir si un alumno aprobará en función de su asistencia a clase -> Clasificación binaria (Supervisado)
5. Clasificar fotografías de animales en familias -> Clasificación múltiple (Supervisado)
6. Predecir la temperatura en función de la fecha y las precipitaciones del día anterior -> Regresión (Supervisado)
7. Estimar las páginas web de interés de un usuario en función de sus búsquedas -> Aprendizaje asociativo (No supervisado) -> Modelos recomendadores
8. Ordenar una cantidad de piezas mecánicas agrupándolas por similitud -> Clustering

UNIVERSIDAD
COMPLUTENSE
DE MADRID



ntic
master
revolucionamos la comunicación

Autor: María José
Gómez Silva

Machine Learning con Python. Semana 1.

CAPÍTULO 1. INTRODUCCIÓN AL MACHINE LEARNING.

1.4 ¿Qué es aprender para una máquina? Problemas del entrenamiento

también se dice ajustar los parámetros al modelo

¿QUÉ ES APRENDER PARA UNA MÁQUINA? PROCESO DE ENTRENAMIENTO

Para una maquina o programa, aprender es ajustar los parámetros de un modelo de modo que le permita solucionar una determinada tarea (como las de regresión, clasificación, clustering, etc.), con cierto rendimiento.

El ajuste de los parámetros del modelo se realiza mediante un proceso cíclico de prueba y error, llamado *entrenamiento*, que requiere de la observación de múltiples ejemplos.

Gracias al entrenamiento, las máquinas adquieren la capacidad para generalizar una solución. A diferencia de la inteligencia humana, la artificial requiere de muchas muestras (ejemplos) de una determinada tarea antes de aprender a realizarla con precisión. Por ejemplo, si un niño es llevado al zoo y ve por primera vez un elefante, instantáneamente conceptualiza la idea de “qué es un elefante”. La próxima vez que vea un elefante, aunque esté en otra posición, a otra distancia, sea de otro tamaño, o esté parcialmente oculto por árboles, en cualquier caso, el niño será capaz de generalizar y predecir qué efectivamente esos nuevos “ejemplos” también son elefantes. Sin embargo, una máquina necesita ser entrenada con miles o incluso millones de ejemplos hasta adquirir tal capacidad de generalización.

En el entrenamiento se establece un determinado objetivo. En cada iteración (ciclo) el modelo ofrece una solución o predicción para los datos de entrada dados. Se calcula cuánto se desvía la solución dada por el modelo respecto al objetivo a cumplir y se reajustan los parámetros del modelo para intentar minimizar esa desviación.

En el caso concreto del aprendizaje supervisado, tenemos datos de entrenamiento formados no sólo por las variables de entrada, sino también por sus respectivas etiquetas (solución conocida). Existe una relación entre las entradas y sus etiquetas, pero es desconocida, y la labor del entrenamiento es encontrar tal relación, o lo que es lo mismo definir el modelo que calcula la salida (etiqueta) en función de la entrada.

Supongamos que tenemos que entrenar un modelo que permita predecir el número de préstamos de libros diarios que se realizan en una biblioteca, en función de la edad media de la población en la localidad en la que se encuentra la biblioteca. Tenemos los siguientes datos correspondientes a distintas fechas, que han sido recogidos previamente y se emplearán como datos de entrenamiento, $x=[34, 39, 43, 47, 52, 55, 60, 64, 69, 72, 78]$ e $y'=[40, 42, 50, 65, 68, 88, 87, 104, 112, 127, 128]$, donde x son los datos de entrada, la edad media de la población, e y' son sus respectivas etiquetas, las visitas diarias contabilizadas, que será la variable a predecir. Los datos están representados en la Figura 1.

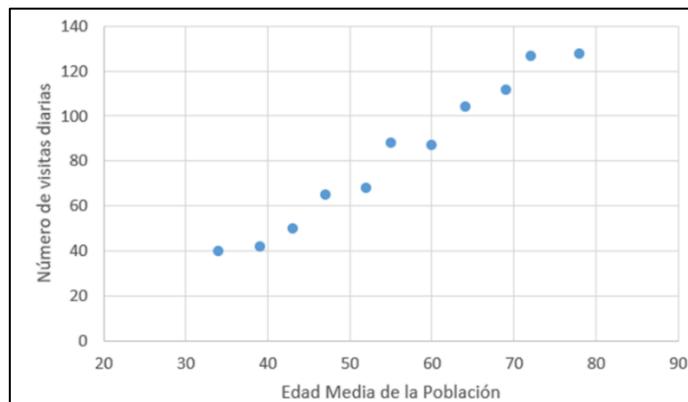


Figura 1. Representación de los datos de entrenamiento

Los datos parecen presentar una relación (tendencia) lineal entre los valores de entrada y sus respectivas etiquetas. Por lo tanto, en este caso, se empleará un modelo lineal de regresión para realizar predicciones. Es decir, partimos de la idea de que la función que relaciona una determinada entrada, x_i , y su correspondiente salida, y_i , sigue la ecuación de una recta: $y_i = a + b * x_i$. Esa ecuación de la recta será el modelo a entrenar, donde a y b son los parámetros del modelo a ajustar.

En el proceso de entrenamiento se parte de una hipótesis o modelo inicial, es decir, de unos valores iniciales para los parámetros, a y b , del modelo. Suponiendo que $a = -20$ y $b = 1$ son los valores iniciales, nuestra hipótesis sería una línea recta como la representada en la Figura 2, que se aleja bastante de modelar correctamente la tarea.

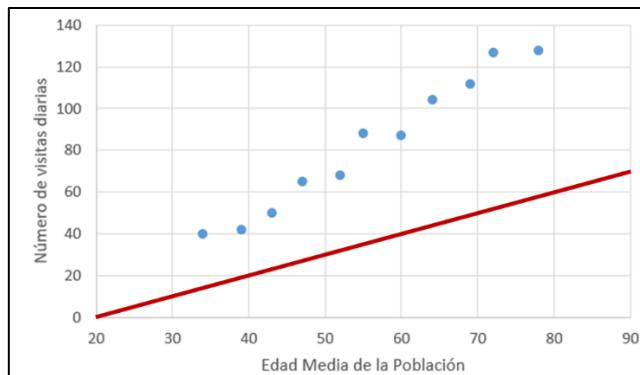


Figura 2. Hipótesis inicial del modelo $y_i = a + b * x_i$, con $a = -20$ y $b = 1$.

Si empleamos la hipótesis anterior para realizar predicciones, y , en función de los datos x de entrada, o lo que es lo mismo, si calculamos el valor de y ofrecido por la ecuación $y_i = -20 + 1 * x_i$, para cada valor x de los datos de entrenamiento, obtenemos los siguientes valores $y = [14, 19, 23, 27, 32, 35, 40, 44, 49, 52, 58]$, que distan mucho de las etiquetas reales y' de referencia. Concretamente la diferencia entre las etiquetas de referencia y las predicciones del modelo son $y' - y = [26, 23, 27, 38, 36, 53, 47, 60, 63, 75, 70]$. La figura 3 muestra tales diferencias (en verde) entre la altura real de los puntos (etiqueta y' de referencia) y la altura que tendrían según la predicción, y , del modelo.

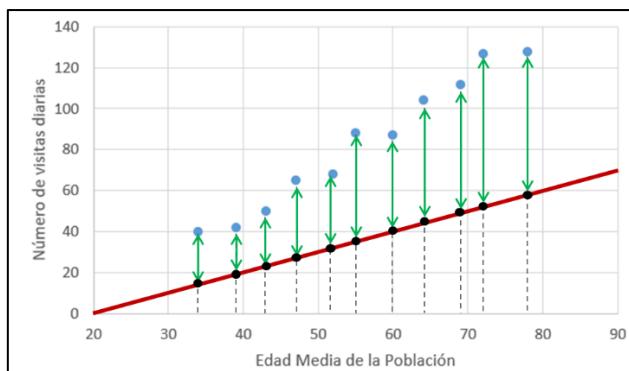


Figura 3. Desviación de la predicción del modelo, y , respecto a los valores de referencia, y' , de los datos de entrenamiento.

Gracias a la observación del error cometido en cada iteración, los algoritmos de Machine Learning son capaces de estimar automáticamente si deben aumentar o disminuir el valor actual de los parámetros del modelo. El incremento o decremento del valor de los *parámetros* es lo que llamamos ajuste de los parámetros del modelo, o simplemente *ajuste del modelo*.

Por ejemplo, siguiendo el caso anterior y suponiendo que los nuevos valores de los parámetros son $a = -18$ y $b = 1.4$, el modelo tendría el aspecto de la recta representada en la Figura 4.

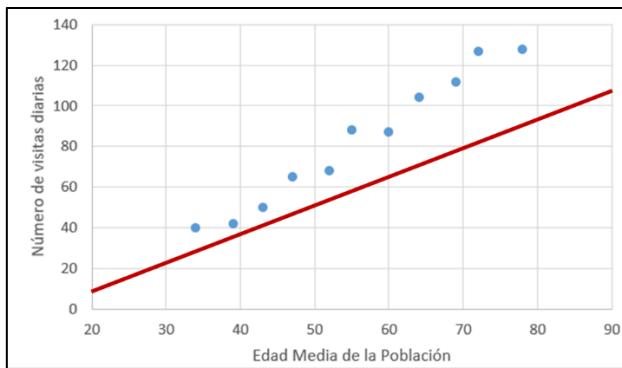


Figura 4. Modelo $y_i = a + b * x_i$, con $a = -18$ y $b = 1.4$.

Tras el ajuste de los parámetros, la ecuación del modelo cambia y el ciclo vuelve a repetirse. En cada iteración, se realizan predicciones con el modelo, se calcula la desviación de las predicciones con respecto a las etiquetas de referencia y se vuelven a ajustar los parámetros del modelo. Este proceso cíclico se repetirá automáticamente hasta que el modelo se ajuste correctamente a los datos, es decir, hasta alcanzar un valor pequeño del error cometido por las predicciones.

Una posible solución al problema planteado en el ejemplo sería la mostrada en la Figura 5, donde podemos observar que el modelo aprendido ($a = -46.7$, $b = 2.33$) sigue la ecuación de una recta (roja) que se ajusta bastante bien a los datos.

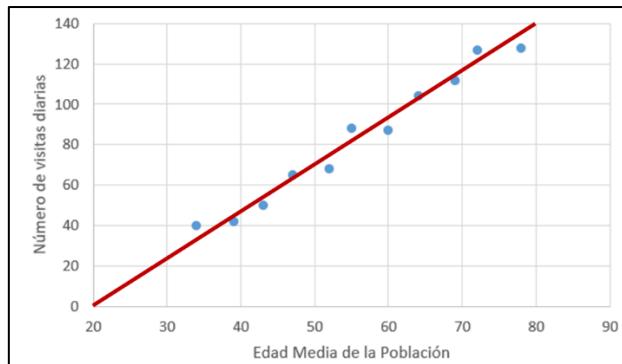


Figura 5. Modelo final, ajustado a los datos de entrenamiento

El entrenamiento no tiene por qué anular por completo el error existente entre las predicciones ofrecidas por el modelo para los datos de entrada y las etiquetas de referencia de los mismos. El objetivo es tratar de minimizar la suma de los errores, para obtener una estimación de la tendencia que siguen los datos. De este modo cuando llegue un nuevo dato de entrada, por ejemplo, la edad media de la población según el último censo, el modelo podrá ofrecer una predicción de los préstamos de libros que se espera que sean solicitados en la biblioteca. Puede que finalmente el valor real no sea exactamente el predicho por el modelo, pero el modelo nos permite tener una estimación.

PROBLEMAS DEL ENTRENAMIENTO

Las ecuaciones que definen un modelo pueden ser muy dispares y con distintos grados de complejidad. En la sección anterior se empleó como ejemplo un problema de regresión lineal, donde se parte de una hipótesis que sigue la ecuación de una línea recta. Sin embargo, dependiendo de la complejidad de los datos, podría ocurrir que la línea recta no fuese adecuada para modelar dichos datos.

En la Figura 6, se puede observar como el modelo lineal no se ajusta bien a los datos disponibles. En este caso, el problema no reside en los valores dados a los parámetros a y b del modelo ($y_i = a + bx_i$), sino al hecho de que el modelo es demasiado sencillo para la tarea encomendada.

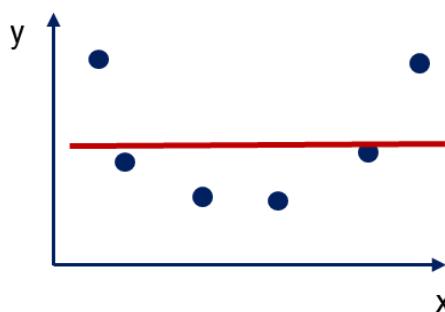


Figura 6. Modelo lineal que no se ajusta bien a los datos de entrenamiento.

Una posible solución es aumentar la complejidad del modelo, lo cual implica un aumento del número de parámetros. Por ejemplo, en este caso, podríamos pasar de la ecuación de una línea recta a la ecuación de una parábola ($y = a + bx + cx^2$), como muestra la Figura 7. Esto supone pasar de tener que ajustar dos parámetros (a y b) a tener que ajustar tres parámetros (a , b y c).

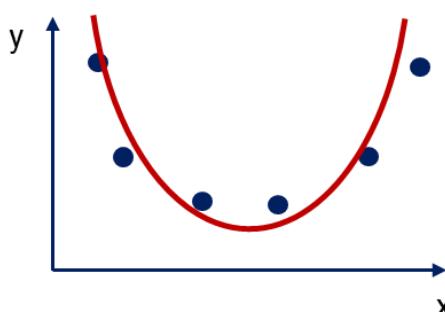


Figura 7. Modelo parabólico que se ajusta bien a los datos de entrenamiento.

En conclusión, la complejidad del modelo a entrenar debe ser proporcional a la complejidad de la tarea a desempeñar o a la complejidad de los datos disponibles. De igual modo que un modelo demasiado sencillo no es adecuado para modelar datos con

tendencias complejas, un modelo demasiado complejo tampoco dará buenos resultados sobre datos sencillos. Esto se debe a que cuantos más parámetros tiene un modelo, más flexible es su forma y por ello presenta más facilidad para adaptarse a los datos, como se observa en la Figura 8.

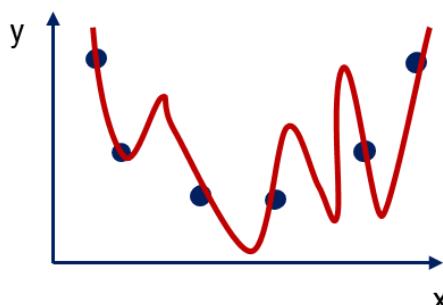


Figura 8. Modelo complejo perfectamente ajustado a los datos de entrenamiento.

Sin embargo, no siempre es conveniente que el modelo final aprendido se ajuste con total exactitud a los datos de entrenamiento. El objetivo del entrenamiento es ajustarse a la tendencia de los datos lo máximo que sea posible sin que la capacidad de generalización se vea comprometida. La capacidad de generalización del modelo se refiere a la precisión con la que ofrece una predicción a partir de nuevos datos de entrada, diferentes de los empleados en el entrenamiento. A estos nuevos datos les llamamos datos de prueba o de test. La figura 9 muestra como el modelo anterior no se ajusta bien a un nuevo dato desconocido (azul claro).

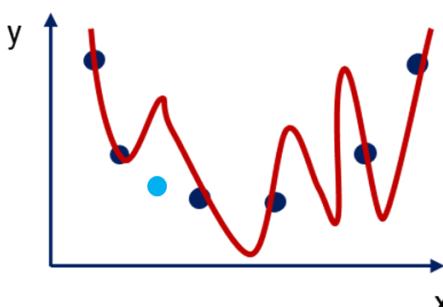


Figura 9. Modelo complejo perfectamente ajustado a los datos de entrenamiento.

Un buen modelo es aquel que ofrece predicciones precisas para los datos de test. Si esto no sucede, puede deberse a dos causas, que el modelo está **desajustado (under-fitting)** o que el modelo ha sido **sobre-ajustado (over-fitting)**. En la Figura 10 puede observarse el caso de un modelo desajustado, otro ajustado apropiadamente y otro sobre-ajustado, para las tareas de regresión y de clasificación.

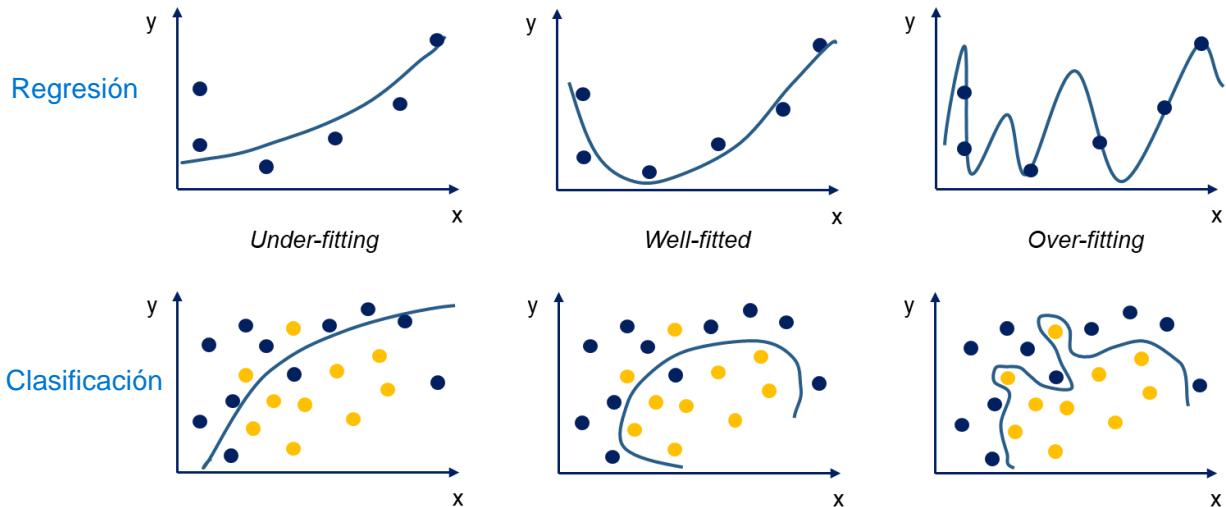


Figura 10. Modelos de regresión y clasificación desajustados, ajustados apropiadamente y sobre-ajustados.

Cuando el modelo está desajustado (under-fitting), sus predicciones no son precisas ni con los datos de entrenamiento, ni con los de test. El desajuste puede deberse a que el modelo es demasiado sencillo, o a que no se ha entrenado el tiempo suficiente para ajustar bien los parámetros. Puede corregirse, aumentando el tiempo de entrenamiento o la complejidad del modelo, dependiendo del caso.

Cuando el modelo está sobre-ajustado, sus predicciones son muy precisas con los datos de entrenamiento pero no con los de test. El sobre-ajuste puede deberse a que el modelo es demasiado complejo o a que ha sido entrenado durante demasiado tiempo. Puede corregirse, finalizando el entrenamiento de forma temprana, reduciendo la complejidad del modelo o aumentando el número de datos de entrenamiento.

Un modelo bien entrenado alcanza el equilibrio adecuado entre las dos situaciones anteriores.

UNIVERSIDAD
COMPLUTENSE
DE MADRID



ntic
master
revolucionamos la comunicación

Autor: María José
Gómez Silva

Machine Learning con Python. Semana 1.

CAPÍTULO 1. INTRODUCCIÓN AL MACHINE LEARNING.

1.5 Función de Pérdidas y Gradient Descent

FUNCIÓN DE PÉRDIDAS

Como ya se ha comentado en secciones anteriores, el ajuste de los parámetros del modelo se realiza a partir del error cometido por las predicciones. En otras palabras, la decisión de cuánto hay que aumentar o disminuir los valores de los parámetros del modelo, se calcula en función de cuánto se ha desviado el modelo con respecto a algún tipo de referencia. En el caso del aprendizaje supervisado la referencia a seguir viene dada por las etiquetas de los datos.

La función encargada de medir la desviación del modelo con respecto a su objetivo se denomina *función de pérdidas o de costes*. En algunos textos también se le denomina *función objetivo*, dado que implícitamente define el objetivo final del modelo.

Como regla general, la función de pérdidas calcula la media del error cometido por el modelo para un conjunto de datos de entrada. En los ejemplos anteriores hemos medido el error, para cada dato de entrada, como la diferencia entre la predicción del modelo y la etiqueta correspondiente. Sin embargo, existen multitud de métricas disponibles para calcular tal error, dependiendo de las necesidades del problema. En consecuencia, la literatura presenta una gran variedad de funciones de pérdida con distintas formulaciones, cada una de ellas específica para un tipo de entrenamiento, de modelo, o adaptada a las características de los datos. Por ejemplo, la función de pérdidas denominada *Focal Loss Function* presenta una formulación que la hace apropiada para resolver problemas de clasificación cuando el número de ejemplos de datos disponibles de cada clase está muy desbalanceado.

En cada iteración del entrenamiento, la función de pérdidas mide la precisión del modelo con los valores actuales de los parámetros. La Figura 1 muestra los valores de la función de pérdidas calculados para distintos valores de los parámetros a y b . En este caso, se ha calculado el error como el valor absoluto de la diferencia entre las predicciones y , y las etiquetas de los datos, y' .

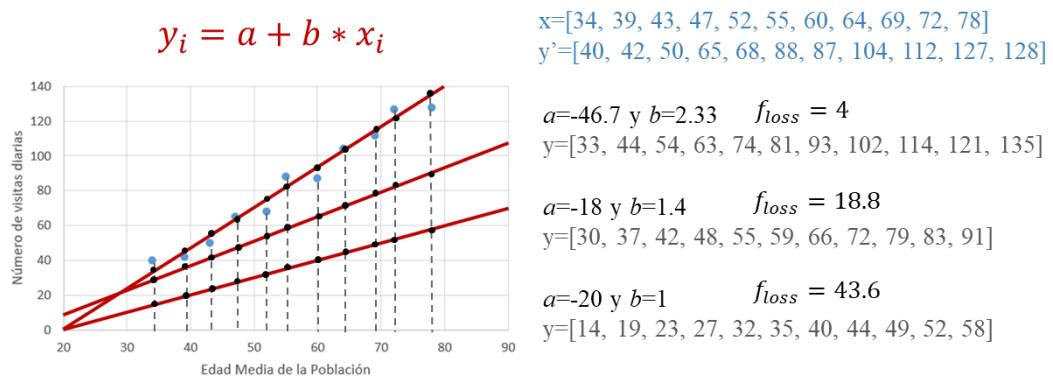


Figura 1. Cálculo de la función de pérdidas para distintos valores de los parámetros del modelo.

Por lo tanto, el valor de la función de pérdidas depende de los valores de los parámetros del modelo. El objetivo del entrenamiento es encontrar los valores de los parámetros del modelo que minimizan el valor de la función de pérdidas.

GRADIENT DESCENT

En la sección anterior, se concluyó que el objetivo del entrenamiento es encontrar los valores de los parámetros del modelo que minimizan la función de pérdidas. La siguiente pregunta a plantearse sería **¿cómo se realiza esa minimización?**

Si se conociese a priori como varía el valor de las pérdidas en función de los valores de los parámetros del modelo, podríamos buscar el valor mínimo mediante métodos aritméticos. En la Figura 2 se muestra un ejemplo de cómo varía el valor de las pérdidas para distintos valores de los parámetros a y b , dando lugar a una superficie. Sin embargo, la forma de tal superficie puede ser desconocida a priori o muy difícil de obtener. A medida que aumenta el número de parámetros, aumentan las dimensiones del problema y su complejidad. Además, el valor de la función de pérdidas también depende de la ecuación que define al modelo.

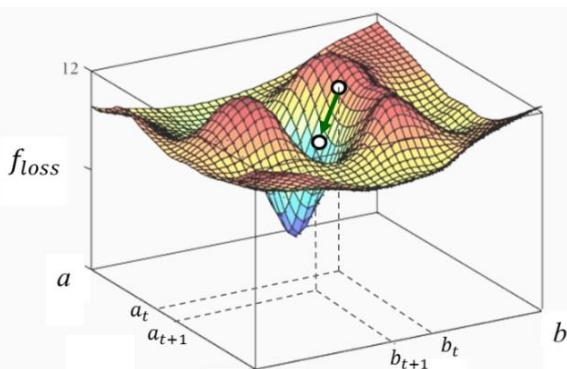


Figura 2. Valor de las pérdidas en función de los valores de los parámetros a y b .

Debido a la gran cantidad de parámetros que afectan al valor de las pérdidas, no es viable aplicar métodos de optimización aritméticos y se opta por seguir un proceso de exploración progresiva, paso a paso. De ahí el carácter iterativo o cíclico del entrenamiento.

En cada iteración del entrenamiento el valor de los parámetros debe ser actualizado. La idea es que esa actualización se realice de forma que, para los nuevos valores de los parámetros, la pérdida sea menor, o lo que es lo mismo, que en cada iteración el modelo sea más preciso y se desvíe menos de su objetivo.

De acuerdo con la figura anterior, se trataría de ir avanzado en la dirección en la que la superficie desciende hasta alcanzar el valor mínimo, o un valor cercano al mínimo. Para realizar ese proceso se emplea el método del *Descenso del Gradiente (Gradient Descent)*.

Un gradiente en una operación matemática que describe una variación. Siguiendo con el ejemplo de la figura, si en un determinado momento del entrenamiento los parámetros del modelo toman los valores, a_t y b_t , podemos calcular el gradiente de la superficie en ese punto, y eso nos daría información de como varía la superficie en cualquier dirección desde ese punto. El gradiente nos da una idea de la pendiente en un punto. Lo que debemos hacer entonces es tomar la dirección que haga que el gradiente (la pendiente) sea negativo, es decir que la superficie descienda. Buscamos un descenso en el gradiente, de ahí el nombre del método.

El gradiente nos indica en qué dirección avanzar en cuanto a la actualización de los parámetros para minimizar el valor de las pérdidas, pero cuánto avanzamos viene determinado por otro parámetro, α , denominado *tasa de aprendizaje (learning rate)*.

Finalmente, el valor de los parámetros, $p=[a,b,\dots]$, es actualizado al final de cada ciclo de entrenamiento de acuerdo con la siguiente fórmula:

$$p_{t+1} = p_t - \alpha \nabla f_{loss}(p_t)$$

Donde p_t y p_{t+1} son los parámetros del modelo en un ciclo de entrenamiento y en el siguiente, y $\nabla f_{loss}(p_t)$ es el gradiente de la función de pérdidas en el punto dado por los valores de los parámetros p_t .

PROCESO ITERATIVO

Como se ha comentado en las secciones anteriores, el proceso de aprendizaje o entrenamiento de un modelo es iterativo. A continuación, se describe la secuencia de

pasos realizada en cada iteración, t , del algoritmo de aprendizaje del descenso del gradiente:

1. Se toman los datos de entrada X y sus etiquetas Y' .
2. Se calculan las predicciones del modelo Y .

Las predicciones del modelo en cada iteración, t , depende de los valores de los parámetros en esa iteración, p_t .

3. Se calcula el valor de las pérdidas f_{loss} .

La función de pérdidas compara Y' con Y . Dado que, Y es la predicción del modelo con los valores actuales de los parámetros, p_t , implícitamente la función de pérdidas depende de p_t .

4. Se calcula el gradiente de la función de pérdidas con respecto a los parámetros $\nabla f_{loss}(p_t)$.

5. Se actualizan los valores de los parámetros $p_{t+1} = p_t - \alpha \nabla f_{loss}(p_t)$.

p_{t+1} serán los valores de los parámetros en la iteración siguiente.

Por lo tanto, implícitamente se actualiza el modelo.

Dependiendo de la cantidad de datos tomados como datos de entrada en cada iteración, aparecen distintas versiones del método del descenso del gradiente:

Gradient Descent: En cada iteración, los datos de entrada son el conjunto completo de datos de entrenamiento. Por lo tanto, en cada iteración todos los datos disponibles para el entrenamiento son estudiados. La actualización final de los parámetros del modelo está basada en la desviación del modelo con respecto al objetivo medida para todos los datos de entrenamiento. *No adecuado si tengo muchos datos, no se pueden almacenar todos a la vez en el ram*

Stochastic Gradient Descent: En cada iteración, se toma como dato de entrada una sola muestra de los datos de entrenamiento. Por lo tanto, la actualización final de los parámetros del modelo está basada en la desviación del modelo con respecto al objetivo medida para una única muestra. Por ese motivo, los valores de los pesos pueden oscilar bastante de una iteración a la siguiente. En este caso, son necesarias tantas iteraciones como datos de entrenamiento haya, para llegar a estudiarlos todos. *provoco muchas oscilaciones dependiendo de la muestra*

Mini-batch Gradient Descent: En cada iteración, se toma como datos de entrada un subconjunto de los datos de entrenamiento. Se trata de una solución a medio camino entre las anteriores y es la más empleada actualmente. Cuando se dispone de grandes bases de datos para entrenar, es inviable tomar todos los datos de entrenamiento en cada iteración, debido a las limitaciones de memoria de los procesadores. Tomando subconjuntos de datos, se evitan las oscilaciones propias del Stochastic Gradient Descent y la lentitud en el proceso de aprendizaje, propia del Gradient Descent.

UNIVERSIDAD
COMPLUTENSE
DE MADRID



ntic
master
revolucionamos la comunicación

Autor: María José
Gómez Silva

Machine Learning con Python. Semana 1.

CAPÍTULO 1. INTRODUCCIÓN AL MACHINE LEARNING.

1.6 Convergencia y Learning Rate

CONVERGENCIA

La secuencia de pasos descrita en la sección anterior se repite cíclicamente hasta cumplir una determinada condición que define el final del proceso de aprendizaje. Esta condición podría consistir simplemente en haber realizado un número predeterminado de iteraciones, o puede estar basada en la convergencia del modelo.

Para estudiar la convergencia del modelo, es necesario introducir el concepto de curva de aprendizaje. La *curva de aprendizaje* representa el valor de la función de pérdidas en las sucesivas iteraciones del entrenamiento. El comportamiento deseable es que el valor de la función de pérdidas disminuya a medida que avanza el proceso, como en la Figura 1. Eso indicaría que las predicciones del modelo cada vez se parecen más al objetivo marcado, y por lo tanto, se puede decir que el modelo está aprendiendo.

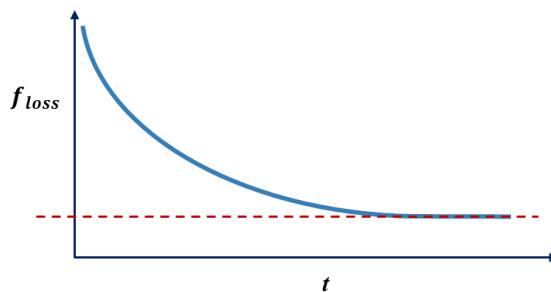


Figura 1. Ejemplo de curva de aprendizaje en la que el modelo converge.

Se dice que el proceso de aprendizaje converge, cuando el valor de las pérdidas varía poco (menos de un cierto umbral), entre sucesivas iteraciones. En la curva de aprendizaje puede identificarse la convergencia como la estabilización del valor de pérdidas en torno a un determinado valor. Si el valor de la función de pérdidas se estabiliza y deja de variar, o varía poco, también lo harán los valores de los parámetros del modelo. Por lo tanto, puede decirse que el aprendizaje ha convergido hacia unos valores concretos de los parámetros o lo que es lo mismo se ha aprendido un determinado modelo.

LEARNING RATE Cuánto avanzamos

La tasa de aprendizaje (learning rate), α , regula la magnitud de las variaciones (incrementos o decrementos) de los valores de los parámetros del modelo en el momento de su actualización, al final de cada iteración del entrenamiento, tal y como define la siguiente ecuación.

$$p_{t+1} = p_t - \alpha \nabla f_{loss}(p_t)$$

La tasa de aprendizaje es un hiperparámetro. Llamamos **hiperparámetros** a todos aquellos parámetros, que no son los parámetros propios del modelo, pero que regulan su proceso de aprendizaje.

La tasa de aprendizaje determina la magnitud de cada paso dado en la dirección del gradiente descendente, es decir determina en qué medida o con qué velocidad se avanza en el proceso de aprendizaje.

Si el valor de α es grande, el avance en cada iteración, hacia un valor mínimo de las pérdidas será grande, y consecuentemente el aprendizaje será rápido (se necesitan pocas iteraciones). Sin embargo, si escogemos un valor de α demasiado grande, el avance puede adquirir un comportamiento divergente y no llegar a alcanzar el mínimo buscado, como se puede observar en la Figura 2. Si el valor de α es pequeño, el avance, en cada iteración, hacia un valor mínimo de la función de pérdidas será pequeño, y consecuentemente el aprendizaje será lento (se necesitan muchas iteraciones). Por eso, este hiperparámetro se denomina **tasa o ratio de aprendizaje**, ya que determina la velocidad con que el modelo aprende.

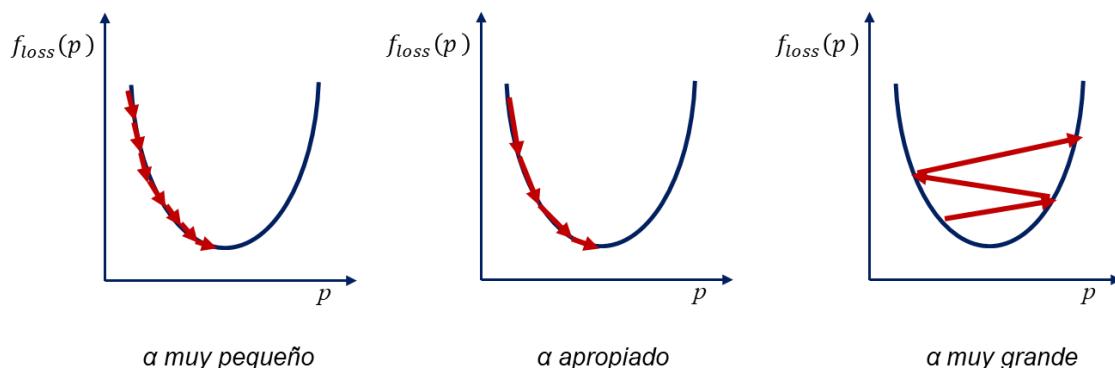


Figura 2. Avance del aprendizaje para distintas tasas de aprendizaje.

UNIVERSIDAD
COMPLUTENSE
DE MADRID



ntic
master
revolucionamos la comunicación

Autor: María José
Gómez Silva

Machine Learning con Python. Semana 1.

CAPÍTULO 1. INTRODUCCIÓN AL MACHINE LEARNING.

1.7 Etapas de un proyecto de Machine Learning

La primera etapa cuando se trabaja con aprendizaje automático es la de **preparación de los datos**, también llamada **extracción de características**, como puede observarse en el diagrama de bloques de la Figura 1. En esta etapa los datos pueden ser filtrados, normalizados, escalados, seleccionados y en general, procesados. Esta etapa trata de **mejorar los datos o seleccionar aquellos más representativos** para la realización de la tarea en cuestión. En esta etapa se realiza el procesamiento necesario para adecuar los datos de modo que el proceso de entrenamiento sea favorecido, facilitado o mejorado. Algunos de los métodos más usuales en esa primera etapa serán abordados en el siguiente capítulo de *Ingesta de Datos*.

Cuando se realiza un aprendizaje supervisado, además, es necesaria una etapa de **etiquetado o anotación**, para determinar manualmente (o con algún mecanismo semiautomático) la solución o etiqueta de cada muestra empleada como dato de entrada en el entrenamiento. Es decir, para los datos de entrenamiento, es necesario anotar la predicción que esperamos que el modelo dé cuando haya aprendido. No sólo los datos de entrenamiento deben ser anotados, también los de validación y test, como se verá a continuación.

Una vez preparados los datos, el proceso de **entrenamiento o aprendizaje** automático es un proceso cíclico en el que **el modelo es actualizado en función de la desviación de sus predicciones con respecto a la solución deseada**, como se puede apreciar en la figura.

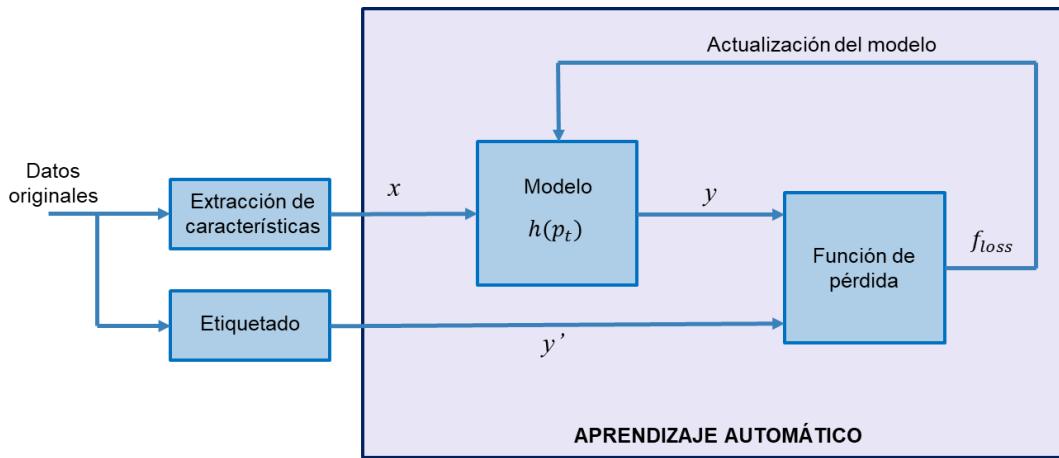


Figura 1. Esquema del proceso de entrenamiento

Por lo tanto, aprender es actualizar los parámetros del modelo de modo que en cada ciclo se desvíe menos de su objetivo.

Tras el proceso de aprendizaje, el modelo debe ser evaluado. La **evaluación** o **test** se realiza calculando la precisión, o cualquier otra métrica que permita medir la bondad del modelo. En este proceso, el modelo no es actualizado, como se observa en la Figura 2.

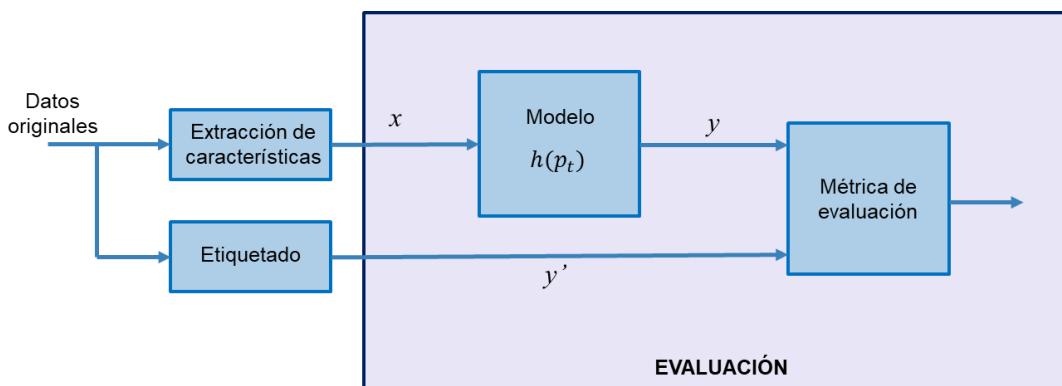


Figura 2. Esquema del proceso de evaluación

Una práctica muy común es realizar algunas evaluaciones del modelo durante su entrenamiento, cada un cierto número de iteraciones de aprendizaje. Estas evaluaciones, puntuales y periódicas durante el entrenamiento, se denominan **validaciones**. La validación permite observar cómo progresó la precisión del modelo para datos desconocidos para el modelo, distintos a los de entrenamiento.

En resumen, un proyecto de Machine Learning requiere tres conjuntos de datos diferentes: el de **entrenamiento**, el de **validación** y el de **test**. En caso de realizar un aprendizaje supervisado, todos los datos deben estar etiquetados, no sólo los datos de entrenamiento, ya que, para estimar la métrica o métricas de evaluación, es necesario comparar la predicción del modelo con respecto al valor esperado de referencia.

El rendimiento del modelo sobre los datos de entrenamiento determina el ajuste de los parámetros del modelo. El resultado sobre los datos de validación puede emplearse para ajustar algunos hiper-parámetros como el *learning rate*, o para decidir el final del entrenamiento. Finalmente, el rendimiento del modelo entrenado es evaluado sobre el conjunto de datos de test.

Una vez finalizado el aprendizaje del modelo, y habiendo obtenido una evaluación positiva del mismo, el modelo puede ser empleado para realizar la tarea para la que se le ha entrenado, pero ahora sobre nuevos datos, de los cuales no se conoce la solución a priori, sino que es el modelo quien debe estimarla, como se observa en la Figura 3.

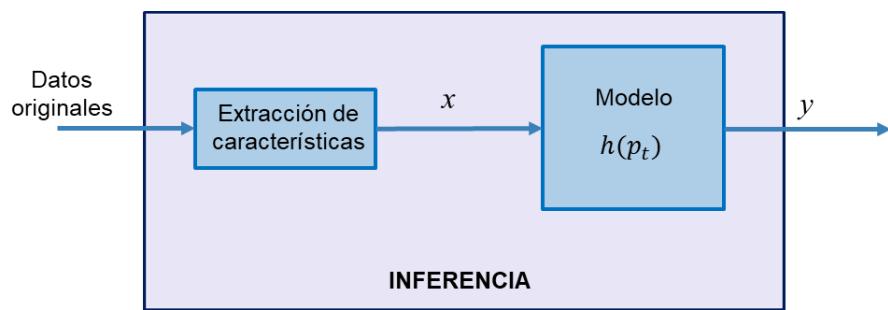


Figura 3. Esquema de inferencia con un modelo

En un entorno de experimentación real, las etapas de preparación de los datos, entrenamiento y evaluación son a su vez iterativas. Por ejemplo, de la evaluación de distintos modelos entrenados con distintas características de los datos se pueden obtener conclusiones sobre qué características mejoran o no la tarea. Por ello, el aprendizaje automático es un proceso interdisciplinar, cuyo éxito depende de la colaboración de expertos en distintas áreas: expertos en la tarea que se esté intentando modelar, en adquisición y análisis de datos, en algoritmos de “Machine Learning”, y en implementación de los modelos finales, entre otras.

UNIVERSIDAD
COMPLUTENSE
DE MADRID



ntic
master
revolucionamos la comunicación

Autor: María José
Gómez Silva

Machine Learning con Python. Semana 1.

CAPÍTULO 1. INTRODUCCIÓN AL MACHINE LEARNING.

1.8 Programación de un Proyecto de Machine Learning con Python

PROGRAMACIÓN DE UN PROYECTO DE MACHINE LEARNING.

Para implementar un proyecto de Machine Learning, será necesario el desarrollo de software (código) que ejecute cada una de sus etapas y módulos. Para desarrollar dicho software, o lo que es lo mismo para programar el código del proyecto, es necesario emplear un *entorno de desarrollo* en el que escribir nuestro código. Además, dicho código será escrito en un determinado *lenguaje de programación*, en el que ya existirán algunas tareas programadas y almacenadas en *librerías*. A continuación, se definen los conceptos enfatizados.

Lenguaje de programación.

Un lenguaje de programación es un lenguaje artificial (con reglas bien definidas) que le proporciona a una persona, en este caso el programador, la capacidad de escribir (o programar) una serie de instrucciones o secuencias de órdenes, para la realización de una determinada tarea. A todo este conjunto de órdenes escritas en un determinado lenguaje de programación se le denomina *programa* o *código*. El lenguaje de programación que se empleará en este módulo es *Python*, del que se darán más detalles en la siguiente sección.

Librerías.

En informática, una librería o biblioteca es un conjunto de archivos con el código necesario para implementar determinadas funcionalidades. Su objetivo es facilitar la programación, al proporcionar funcionalidades comunes, que ya han sido resueltas previamente por otros programadores. En este módulo, emplearemos librerías de Python, como *Matplotlib*, *Numpy*, *Pandas*, o *Scikit-learn*, que serán descritas con más detalles, más adelante.

Entorno de desarrollo.

Un entorno de desarrollo de software es una herramienta utilizada para escribir, editar, generar, probar y depurar el código de un programa. También proporcionan a los desarrolladores una interfaz de usuario para desarrollar y depurar el programa. La plataforma que emplearemos en este módulo para desarrollar nuestro código será *Jupyter Notebook*.

PYTHON. LENGUAJE DE PROGRAMACIÓN PARA MACHINE LEARNING.

Python es un lenguaje de programación cuyo uso se ha extendido ampliamente en campos como el de desarrollo de software y aplicaciones web, y recientemente en el área de la Inteligencia Artificial, para implementar programas de Data Science y Machine Learning (ML). Su extenso uso en el contexto científico es debido a que dispone de un gran número de librerías que incluyen herramientas para la preparación de los datos (carga, manipulación, limpieza y procesamiento), su representación gráfica y su posterior análisis, ya sea descriptivo o predictivo.

Python es un lenguaje accesible para un amplio público, no necesariamente con un perfil estrictamente tecnológico. Python es eficiente y fácil de aprender, y se puede ejecutar en muchas plataformas diferentes. La documentación de Python está disponible en la siguiente dirección: <https://docs.python.org/3/tutorial/index.html>.

Python es un lenguaje interpretado. Esto significa que el código fuente se ejecuta directamente, gracias a un programa llamado interprete que lee cada instrucción en tiempo real y la ejecuta.

Para poder implementar programas con Python, es necesario tener accesible un intérprete de Python y las librerías necesarias. Algunas de las librerías más empleadas en el desarrollo de proyectos de Machine Learning son Numpy, Pandas, Matplotlib y Scikit-learn, que se describen en la siguiente sección.

Anaconda proporciona una distribución libre (gratuita) de todos estos paquetes (intérprete, librerías, editor de código) y puede descargarse de su sitio web <https://www.anaconda.com/products/individual>. Para instalarla, sólo hay que descargar desde su página web la versión de Anaconda adecuada para el sistema operativo que se tenga y seguir los pasos indicados. Una vez completada la instalación, verifíquela abriendo Anaconda Navigator, un programa que se incluye con Anaconda y que deberá aparecer entre las aplicaciones instaladas en su PC.

LIBRERÍAS DE PYTHON PARA MACHINE LEARNING

Existen numerosas librerías de Python que son ampliamente empleadas para desarrollar proyectos de Machine Learning, y que cubren distintas funcionalidades. Hay librerías para **cálculo numérico y análisis de datos** (Numpy, SciPy, Pandas, Numba), librerías de **visualización** (Matplotlib, Seaborn, Bokeh), y librerías para entrenar y ejecutar algoritmos de **Machine Learning** (scikit-learn) y de **Deep Learning** (TensorFlow, Keras, PyTorch), entre otras. A continuación, se describen algunas de ellas.

Numpy

NumPy es una librería de cálculo numérico, que proporciona una estructura de datos universal que posibilita el análisis y el intercambio de datos entre distintos algoritmos. Las estructuras de datos que implementa, llamadas **arrays**, son vectores multidimensionales y matrices con gran capacidad (véase la Figura 1). Además, esta librería permite realizar operaciones matemáticas de alto nivel, sobre las estructuras de datos mencionadas. La documentación de Numpy está disponible en la siguiente dirección: <https://numpy.org/doc/stable/>.

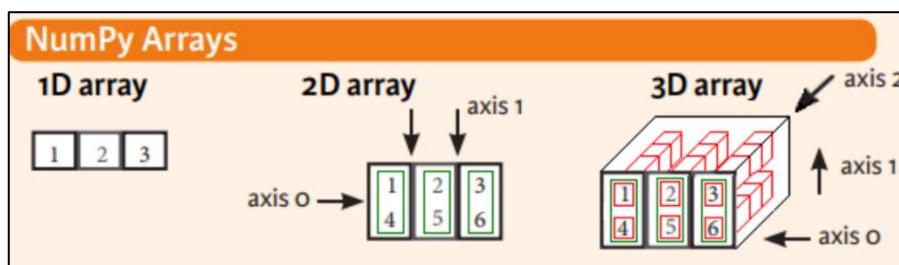


Figura 1. Estructuras de arrays de Numpy.

Pandas

Pandas es una de las librerías de python más útiles en Data Science, por el tratamiento rápido, sencillo y flexible que hace los datos para su análisis. Pandas maneja dos estructuras de datos principales, las **Series** para datos en una dimensión y los **DataFrames** para datos en dos dimensiones. Estas estructuras son muy usadas en multitud de áreas, tales como estadística, finanzas, ciencias sociales y muchas ramas de la ingeniería. Pandas está construido sobre los fundamentos de Numpy, adaptando las operaciones sobre arrays a las nuevas estructuras. Resulta especialmente útil para trabajar con datos heterogéneos representados de forma tabular, como los de la Figura 2. La documentación de Pandas está disponible en la siguiente dirección: <https://pandas.pydata.org/docs/>.

	name	city	phone-number	date
0	James Bass	Fife Lake	340-848-7354	2010-12-11
1	Cody Werner	Topanga	326-520-2048	1996-12-22
2	Joshua West	Richwoods	265-159-8349	2012-03-12
3	Kenneth Hanson	Northfield Woods	184-496-6411	1979-07-12
4	Michelle Brown	Lake Beulah	554-703-6417	1986-06-04

Figura 2. Ejemplo de DataFrame de Pandas

Matplotlib

Matplotlib es una librería gráfica de Python, que permite generar gráficos de una gran variedad de tipos: series temporales, histogramas, espectros de potencia, diagramas de barras, diagramas de errores, etc. La Figura 3 muestra algunos ejemplos. Los gráficos generados por esta librería ofrecen la calidad necesaria para publicarlos tanto en papel como digitalmente. La documentación de Matplotlib está disponible en la siguiente dirección: <https://matplotlib.org/stable/index.html>.

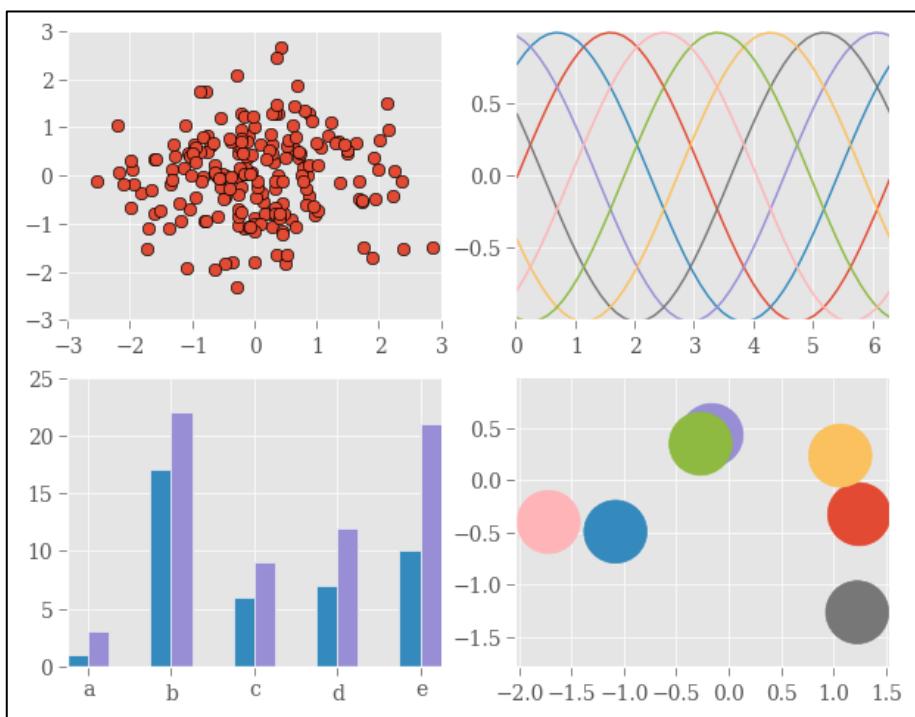


Figura 3. Ejemplos de gráficos obtenidos con Matplotlib

Scikit-learn

Scikit-learn es una librería para desarrollar métodos de Machine Learning y Análisis de Datos. Está basada en otras librerías como NumPy, SciPy y Matplotlib. Una de las mayores cualidades de scikit-learn es su facilidad de uso, gracias a una interfaz simple y muy consistente. Además, permite implementar multitud de técnicas de Machine Learning, tanto con aprendizaje supervisado, como no supervisado. Algunas de las tareas que permite implementar son: regresión (lineal, polinómica y logística), clasificación (máquinas de vectores de soporte, árboles de decisión, bosques aleatorios, clasificadores bayesianos), agrupamiento (clustering), reducción de dimensionalidad, y detección de anomalías, como muestra la Figura 4. La documentación de scikit-learn está disponible en la siguiente dirección: <https://scikit-learn.org/stable/>.

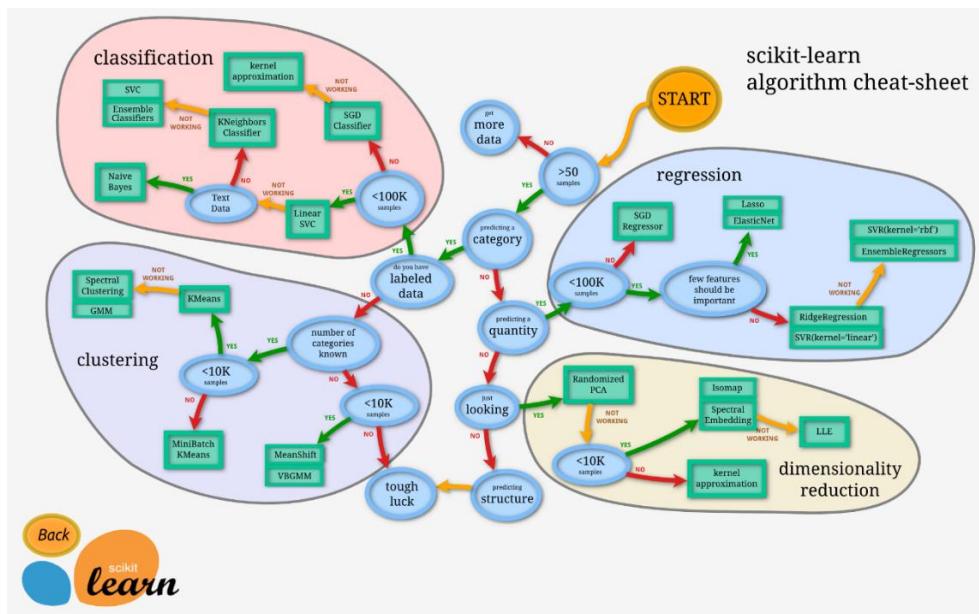


Figura 4. Esquema de ruta de los algoritmos disponibles en scikit-learn

Todas las librerías mencionadas se emplearán en las actividades de este módulo.

JUPYTER NOTEBOOK

Jupyter Notebook es una aplicación web que permite crear documentos (llamados notebooks de Jupyter) que ofrecen una herramienta de ejecución interactiva. Los notebooks contienen código vivo, texto, fórmulas, figuras y medios audiovisuales. Estos documentos se visualizan con un navegador (Explorer, Firefox, Chrome, ...) y permiten la ejecución de código escrito en el lenguaje de programación Python, ya que permiten enviar órdenes directamente al intérprete de Python y obtener una respuesta inmediata de cada una de ellas. La documentación de Jupyter está disponible en la siguiente dirección: <https://docs.jupyter.org/en/latest/>.

Puesta en marcha de Jupyter Notebook.

Una vez realizada la instalación de Anaconda, la ejecución de Jupyter Notebook se puede realizar a través de la aplicación Anaconda Navigator, o directamente, buscando Jupyter Notebook entre las aplicaciones instaladas en nuestro sistema operativo.

La ejecución de Jupyter Notebook abrirá una ventana nueva en el navegador de internet, como se observa en la Figura 5. Esta interfaz web actúa como un explorador de archivos, que inicialmente muestra el contenido de la carpeta en que se abre Jupyter. Jupyter se abre en la carpeta *home* del usuario del equipo. Es recomendable crear una carpeta nueva donde guardar las actividades desarrolladas en este módulo.

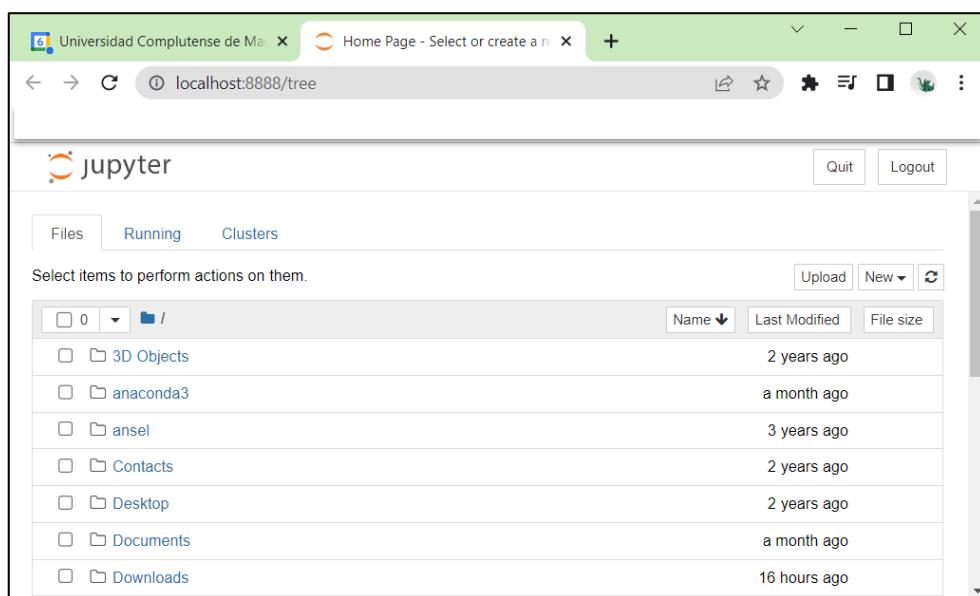


Figura 5. Manejo de los archivos locales a través del entorno de Jupyter

UNIVERSIDAD
COMPLUTENSE
DE MADRID



ntic
master
revolucionamos la comunicación

Autor: María José
Gómez Silva

Machine Learning con Python. Semana 1.

CAPÍTULO 2. INGESTA DE DATOS.

La ingesta de datos (**data ingestion**, en inglés) es el **proceso de recolección de datos de distintas fuentes en un repositorio común de almacenaje**. El propósito de este proceso es que los datos necesarios sean fácilmente accesibles para su empleo en el entrenamiento y la evaluación de los algoritmos de Machine Learning.

En este capítulo se estudiarán distintos tipos y posibles fuentes de procedencia de los datos, y como leerlos y almacenarlos en estructuras de tipo *Series* o *DataFrames*, que son estructuras propias de la librería de *Pandas* en Python.

Además, se introducirán algunas de las múltiples funciones y operaciones necesarias para la gestión de los datos, y la combinación de estos cuando proceden de distintas fuentes o archivos.

Por último, y como enlace con el siguiente capítulo de “Preprocesado”, se presentarán herramientas para el análisis y visualización de los datos, de modo que faciliten las tareas posteriores de preparación de los datos.

2. 1 Tipos y Fuentes de Datos

Los algoritmos de Machine Learning trabajan con una amplia variedad de tipos de datos procedentes de múltiples fuentes, como bases de datos, vídeos, archivos de audio, etc.

Independientemente de su origen, los datos deben ser almacenados y organizados de forma apropiada para su empleo en los algoritmos de Machine Learning. Para ello, es importante conocer las características de los datos, tales como sus dimensiones, número de variables, si son heterogéneos, si tienen dependencia temporal, o si van acompañados por anotaciones. Además, es necesario tener en cuenta cómo están almacenados o el tipo de archivo en que están contenidos, para establecer el modo adecuado de acceder a ellos. A continuación, se detallan cada una de las consideraciones mencionadas.

Dimensiones de los datos

El número de dimensiones de los datos dependerá de su naturaleza y complejidad. Por ejemplo, una grabación de una señal de audio, como puede ser cualquier canción en formato mp3 o wav, es una señal de una única dimensión, es decir, una simple sucesión de valores. En cambio, en una imagen en escala de grises aparecen datos en 2 dimensiones, ancho por alto, ya que una imagen es una matriz de valores con tantas celdas como píxeles tenga la imagen, como puede observarse en la Figura 1.

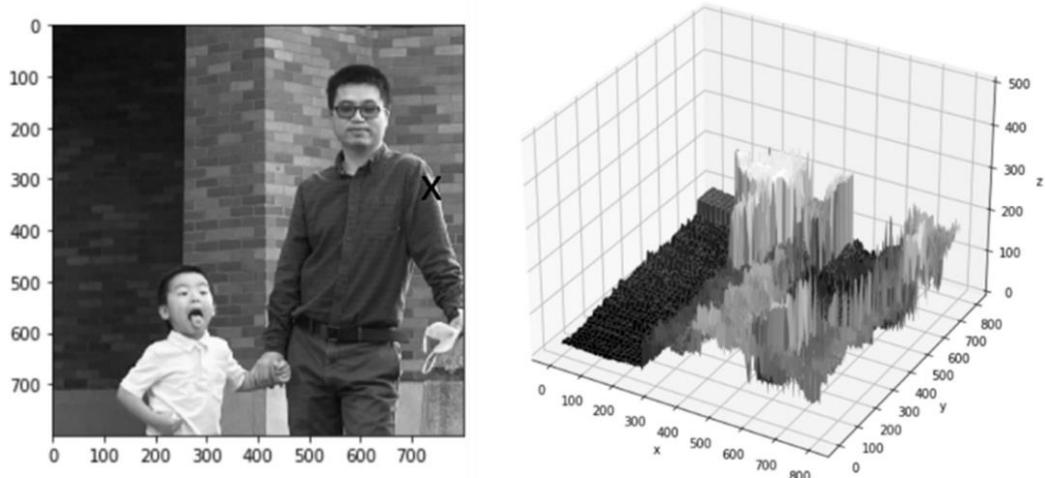


Figura 1. Imagen en escala de grises.

En cambio, si tomamos puntos del entorno con un escáner laser, podemos adquirir una nube de puntos, como la de la Figura 2. Cada punto constituye un dato con 3 dimensiones, sus coordenadas x, y, z.

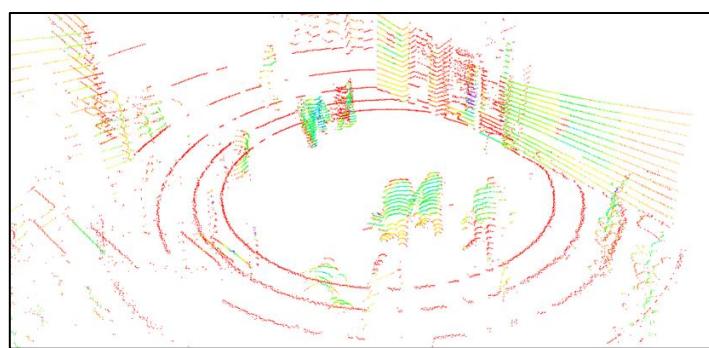


Figura 2. Nube de puntos tridimensionales

Sin embargo, el número de dimensiones de los datos no tiene por qué estar necesariamente ligado al número de dimensiones espaciales que representan. Por ejemplo, si tenemos una imagen en color, en lugar de en escala de grises, la matriz de

píxeles adquiere profundidad. Esto significa que la matriz tiene una dimensión más para poder representar el color como combinación de tres valores (rojo, verde y azul), en lugar de uno sólo (nivel de gris), como se observa en la Figura 3. Además, si los datos son de video, es decir, una sucesión de imágenes en el tiempo, la nueva coordenada de tiempo estaría introduciendo una dimensión más.

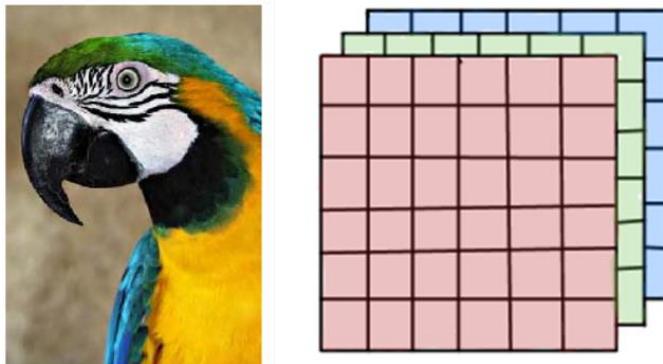


Figura 3. Imagen en color y esquema de la matriz imagen con 3 canales.

Numero de variables

De cada muestra o entrada de nuestra lista de datos, podemos tener almacenada una o más variables. Por ejemplo, si tenemos un listado de viviendas en venta, en el que cada muestra es un inmueble, y lo único que almacenamos de cada inmueble es su precio de venta, en ese caso, tendremos *datos con una única variable*. Los datos con una variable se pueden representar simplemente como una serie de valores. Por el contrario, si aumentamos el número de variables a tener en cuenta (como el número de habitaciones, el número de baños, la superficie total en metros cuadrados, la dirección, los gastos de comunidad, el año de construcción o la fecha en que se puso a la venta cada inmueble) estaremos generando *datos multivariable*. La forma más usual de organizar y almacenar datos multivariable es mediante tablas, en las que cada muestra se almacena en una fila de la tabla, y donde cada columna se corresponde con el valor de una determinada variable.

Heterogeneidad

Cuando tenemos datos con múltiples variables, puede que todas ellas tengan el mismo tipo de dato, por ejemplo, numérico. En ese caso los datos son *homogéneos* en cuanto a formato o tipo de variable. En cambio, es bastante habitual que las bases de datos contengan variables de distintos tipos (numéricos, cadenas de caracteres, binarios, etc.). En ese caso hablamos de datos *heterogéneos*, también llamados a veces, *multimodales*.

Consistencia temporal

Existen algunos tipos de datos, como los de video, que necesariamente deben presentar consistencia temporal, es decir, el orden cronológico de los datos les da sentido. En otros casos, la información de tiempo es opcional, como en el ejemplo anterior del listado de inmuebles, en el que una variable más podría ser la fecha de puesta en venta. Normalmente, las bases de datos, en numerosos y diferentes ámbitos, añaden información temporal, ya que suele ser una variable muy útil para organizar los datos.

Anotaciones

Como ya se explicó en el capítulo anterior, los algoritmos de aprendizaje supervisado requieren de datos etiquetados. Es muy habitual que las anotaciones o etiquetas de los datos, se almacenen como una variable más, en una columna extra de la tabla de datos.

Si las anotaciones son complejas, y no se pueden almacenar como una simple variable, se pueden usar archivos independientes para almacenarlas, siempre que se tenga de algún modo la trazabilidad entre los datos y sus anotaciones. Por ejemplo, en visión por computador, donde los datos de entrada son imágenes, es muy habitual, tener archivos conocidos como *archivos de ground truth*, donde se guardan las etiquetas correspondientes a cada imagen, pudiendo ser esas etiquetas, por ejemplo, la localización de cada objeto en la imagen y su clase.

Fuentes y tipos de archivos

El origen o fuente de los datos (por ejemplo, de audio, visuales, inventarios, encuestas, hojas de características, etc.) determina en gran medida la forma de almacenarlos y manejarlos. Sin embargo, para un caso genérico, en el que necesitemos guardar grandes cantidades de datos multivariable, disponemos de multitud de formatos o tipos de archivos. A continuación, se describen algunos de ellos.

En primer lugar, una *base de datos* es un conjunto de datos que pertenecen a un mismo contexto. Las bases de datos se encargan no solo de almacenar datos, sino también de conectarlos entre sí. Se emplean para administrar de forma electrónica grandes cantidades de información. Las bases de datos pueden ser *relacionales*, como las de tipo *SQL* (Structured Query Language). Esto significa que almacenan datos *estructurados*, en forma de tabla, organizándose en registros y campos. Cada fila es interpretada como un registro y cada columna contiene los campos de cada registro. Sin embargo, también existen *bases de datos no relacionales*, como las de tipo *JSON* (acrónimo de JavaScript Object Notation), para gestionar datos *no-estructurados* o *semi-estructurados*. El formato JSON permite representar los datos de forma jerárquica (en forma de árbol).

Por otro lado, los lenguajes de marcado, como **XML** (Extensible Markup Language) y **HTML** (HyperText Markup Language), son lenguajes empleados en el desarrollo web, para transportar datos entre aplicaciones y servidores. Normalmente **XML** se utiliza para almacenar datos estructurados como documentos, facturas, catálogos, libros, etc., y datos en aplicaciones web, como formularios. **HTML**, además, se ocupa de mostrar datos a los visitantes de los sitios web. Cuando se necesita alojar datos en un sitio web, estos pueden ser integrados en la propia página a través de su código **HTML**.

Por su parte, el **HDF** (Hierarchical Data Format), es un conjunto de formatos, como el **HDF5**, diseñados para almacenar y organizar grandes cantidades de datos. Emplea una estructura jerárquica con dos tipos de objetos, bases de datos y grupos. Estos últimos son contenedores de bases de datos u otros grupos.

Además, los datos provenientes de ficheros de texto y hojas de cálculo, con formato tabular, se pueden almacenar en archivos de tipo **CSV** (Comma Separated Values) y **XLSX** (estándar de Excel desde la versión de Microsoft Office 2003). Se considera que un archivo CSV es cualquier archivo de texto en el que los caracteres están separados por comas, formando una tabla con filas y columnas. Los archivos XLSX, además de tablas, tienen soporte para almacenar texto formateado, imágenes y gráficos.

Almacen de los datos

Dependiendo de la cantidad de datos que necesitemos, de las características de los mismos, y de la capacidad de almacenaje de nuestros equipos, habrá opciones de almacenaje que se adapten mejor que otras a nuestro caso. Cuando la cantidad de datos es relativamente pequeña o disponemos de mucho espacio en la memoria de nuestro equipo, podemos emplear un almacenaje *local* de los datos. Si ese no es el caso, podemos optar por un almacenaje en la **nube** (almacenaje en cloud) gracias a plataformas que ofrecen tales servicios (como **AWS** o **AZURE**). Una solución que se está extendiendo entre las empresas con departamentos de I+D+i y en los grupos y centros de investigación, es el uso de sus propios clusters. Un **cluster** es un conjunto de equipos (CPUs, GPUs, etc.) que funcionan como un servidor para un cierto número de usuarios.

UNIVERSIDAD
COMPLUTENSE
DE MADRID



:ntic
master
revolucionamos la comunicación

Autor: María José
Gómez Silva

Machine Learning con Python. Semana 1.

CAPÍTULO 2. INGESTA DE DATOS.

2. 2 Estructuras e Ingesta de Datos en Pandas

ESTRUCTURAS DE DATOS EN PANDAS. SERIES Y DATAFRAMES

En pandas disponemos de varias estructuras de datos y operaciones (métodos) para manipular dichas estructuras de forma sencilla y eficiente. Las dos estructuras de datos principales son las *Series* (datos en una dimensión) y los *DataFrames* (datos en dos dimensiones), capaces de representar secuencias de datos y datos en forma tabular, respectivamente.

Series

Una variable del tipo *Series* es una variable de una dimensión que contiene datos de un cierto tipo y tiene asociado un índice o etiqueta para cada dato.

La función *pandas.Series* es la constructora de series (la documentación está disponible en: <https://pandas.pydata.org/docs/reference/api/pandas.Series.html>).

Como se puede observar en la Figura 1, la serie *s* contiene elementos de tipo int64. Los elementos de la serie están indexados mediante un array de enteros comenzando desde el cero (columna de la izquierda).

```
In [1]: import pandas as pd
         s = pd.Series([0,1,4,9,16,25])
         s
Out[1]: 0    0
        1    1
        2    4
        3    9
        4   16
        5   25
       dtype: int64
```

Figura 1. Creación de una estructura de tipo Series

Además, las Series permiten indexar cada uno de sus elementos con un valor descriptivo o etiqueta. Por ejemplo, si nuestros datos son el número de días que tiene cada mes, será de gran utilidad usar el nombre de los meses como etiquetas del índice, en lugar de etiquetas de tipo entero. Con el argumento `name` podemos asignar un nombre a la serie, que es posible consultar a posteriori. También, podemos acceder a los valores y los índices de la serie por separado, con los métodos `values` e `index`, como se observa en el ejemplo de la Figura 2.

```
In [2]: dias = pd.Series([31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31],  
                      index=['Ene', 'Feb', 'Mar', 'Abr', 'May', 'Jun', 'Jul', 'Ago', 'Sep', 'Oct', 'Nov', 'Dic'],  
                      name='Días de cada mes')  
dias  
  
Out[2]: Ene    31  
Feb    28  
Mar    31  
Abr    30  
May    31  
Jun    30  
Jul    31  
Ago    31  
Sep    30  
Oct    31  
Nov    30  
Dic    31  
Name: Días de cada mes, dtype: int64  
  
In [3]: dias.name  
  
Out[3]: 'Días de cada mes'  
  
In [7]: dias.values  
  
Out[7]: array([31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31], dtype=int64)  
  
In [8]: dias.index  
  
Out[8]: Index(['Ene', 'Feb', 'Mar', 'Abr', 'May', 'Jun', 'Jul', 'Ago', 'Sep', 'Oct',  
              'Nov', 'Dic'],  
              dtype='object')
```

Figura 2. Creación de una serie con etiquetas para los índices y nombre, y uso de los métodos `name`, `values` e `index`.

DataFrame

Las estructuras de tipo `DataFrame` están diseñadas para manejar datos representados en forma de tabla, donde tanto las filas como las columnas están indexadas. Cada columna puede tener asociado un tipo de dato diferente. Un `DataFrame` puede entenderse como una colección de `Series`, ya que cada columna es una estructura de tipo `Series`.

Con la función `pandas.DataFrame` es posible crear un `DataFrame` a partir de los datos contenidos en una lista, un diccionario, una `serie` o en otro `DataFrame`. En la Figura 3 se crea un `DataFrame` a partir de un diccionario de python. De forma similar a como se hacía en las `Series`, es posible acceder a los valores del `DataFrame`. Sin embargo, ahora no se tiene un único índice asociado a cada fila (`index`), sino que también se tiene un índice para las columnas (`columns`).

```
In [9]: tabla = pd.DataFrame([('Enero', 31, 'inv', 2, 4),('Febrero', 28, 'inv', 3, 2),('Marzo', 31, 'inv', 7, 5),
                           ('Abril', 30, 'pri', 7, 9),('Mayo', 31, 'pri', 9, 10),('Junio', 30, 'pri', 15, 14),
                           ('Julio', 31, 'ver', 20, 24),('Agosto', 31, 'ver', 27, 26),('Septiembre', 30, 'ver', 25, 18),
                           ('Octubre', 31, 'oto', 20, 14),('Noviembre', 30, 'oto', 11, 10),('Diciembre', 31, 'oto', 6, 7)],
                           columns=['Mes', 'Días', 'Estación', 'Temp.2021', 'Temp.2022'],
                           index=['Ene', 'Feb', 'Mar', 'Abr', 'May', 'Jun', 'Jul', 'Ago', 'Sep', 'Oct', 'Nov', 'Dic'])
tabla
```

```
Out[9]:
      Mes   Días  Estación  Temp.2021  Temp.2022
    Ene   Enero     31      inv        2          4
    Feb  Febrero     28      inv        3          2
    Mar   Marzo     31      inv        7          5
    Abr   Abril     30      pri        7          9
    May   Mayo     31      pri        9         10
    Jun   Junio     30      pri       15         14
    Jul   Julio     31      ver       20         24
    Ago   Agosto     31      ver       27         26
    Sep  Septiembre  30      ver       25         18
    Oct  Octubre     31      oto       20         14
    Nov  Noviembre     30      oto       11         10
    Dic  Diciembre     31      oto        6          7
```

```
In [10]: tabla.values
```

```
Out[10]: array([['Enero', 31, 'inv', 2, 4],
                ['Febrero', 28, 'inv', 3, 2],
                ['Marzo', 31, 'inv', 7, 5],
                ['Abril', 30, 'pri', 7, 9],
                ['Mayo', 31, 'pri', 9, 10],
                ['Junio', 30, 'pri', 15, 14],
                ['Julio', 31, 'ver', 20, 24],
                ['Agosto', 31, 'ver', 27, 26],
                ['Septiembre', 30, 'ver', 25, 18],
                ['Octubre', 31, 'oto', 20, 14],
                ['Noviembre', 30, 'oto', 11, 10],
                ['Diciembre', 31, 'oto', 6, 7]], dtype=object)
```

```
In [11]: tabla.index
```

```
Out[11]: Index(['Ene', 'Feb', 'Mar', 'Abr', 'May', 'Jun', 'Jul', 'Ago', 'Sep', 'Oct',
                 'Nov', 'Dic'],
                 dtype='object')
```

```
In [12]: tabla.columns
```

```
Out[12]: Index(['Mes', 'Días', 'Estación', 'Temp.2021', 'Temp.2022'],
                 dtype='object')
```

Figura 3. Creación de un DataFrame a partir de un diccionario y uso de los métodos values, index y columns

INGESTA DE DATOS CON PANDAS

Pandas proporciona funciones para realizar la lectura de una gran variedad de fuentes de datos y almacenar su contenido en estructuras *DataFrame*.

Lectura y escritura de ficheros HTML

leer guardar

Los métodos `pd.read_html` y `pd.to_html` de Pandas se emplean, respectivamente, para leer y escribir ficheros en formato HTML.

El método `pd.read_html` recorre un fichero HTML en busca de tablas, y devuelve una lista de *DataFrames*, uno por cada tabla encontrada. Previamente, para poder acceder al código HTML de una página web hay que realizar una petición con protocolo HTTP Request/Response, como se muestra en la Figura 4.

```
In [58]: import requests
import pandas as pd
url = "https://es.wikipedia.org/wiki/Europa"
respuesta = requests.get(url)
if respuesta.status_code == 200:
    print('Ok')
else:
    print( 'No ok')

Ok

In [59]: codigoHTML = respuesta.text
lista_df = pd.read_html(codigoHTML, header=0)
len(lista_df)

Out[59]: 10

In [61]: lista_df[1].head(4)    llamo a la 2a tabla de la lista
Out[61]:
```

	Bandera	Nombre/Nombre oficial	Establecido	Superficie(km ²)	Población	Habitantes por km ²	Capital
0	NaN	Albania Repùblica de Albania	1912	28 748	3 038 594	1056	Tirana
1	NaN	Alemania Repùblica Federal de Alemania	1871	357 022	80 722 792	2261	Berlín
2	NaN	Andorra Co-Principado de Andorra	1278	468	85 660	183	Andorra la Vieja
3	NaN	Armenia Repùblica de Armenia	1991	29 749	3 229 900	1085	Ereván

Figura 4. Lectura de datos en formato HTML

Lectura y escritura de archivos JSON

Los métodos `pd.read_json` y `pd.to_json` de Pandas se emplean, respectivamente, para leer y escribir ficheros en formato JSON. En la Figura 5 se muestra como una *DataFrame*, previamente creado, es guardado en un archivo JSON, cuyo contenido se muestra en la Figura 6. Seguidamente, la Figura 7 muestra la operación inversa, de lectura de un archivo JSON.

```
In [88]: tabla = pd.DataFrame([('Enero', 31, 'inv', 2, 4),('Febrero', 28, 'inv', 3, 2),('Marzo', 31, 'inv', 7, 5),
                           ('Abril', 30, 'pri', 7, 9),('Mayo', 31, 'pri', 9, 10),('Junio', 30, 'pri', 15, 14),
                           ('Julio', 31, 'ver', 20, 24),('Agosto', 31, 'ver', 27, 26),('Septiembre', 30, 'ver', 25, 18),
                           ('Octubre', 31, 'oto', 20, 14),('Noviembre', 30, 'oto', 11, 10),('Diciembre', 31, 'oto', 6, 7)],columns=['Mes', 'Días', 'Estación', 'Temp.2021', 'Temp.2022'],
                           index=['Ene', 'Feb', 'Mar', 'Abr', 'May', 'Jun', 'Jul', 'Ago', 'Sep', 'Oct', 'Nov', 'Dic'])
tabla
```

	Mes	Días	Estación	Temp.2021	Temp.2022
Ene	Enero	31	inv	2	4
Feb	Febrero	28	inv	3	2
Mar	Marzo	31	inv	7	5
Abr	Abri	30	pri	7	9
May	Mayo	31	pri	9	10
Jun	Junio	30	pri	15	14
Jul	Julio	31	ver	20	24
Ago	Agosto	31	ver	27	26
Sep	Septiembre	30	ver	25	18
Oct	Octubre	31	oto	20	14
Nov	Noviembre	30	oto	11	10
Dic	Diciembre	31	oto	6	7

```
Out[88]: si no pongo el directorio, se guarda en la misma carpeta que el notebook
In [89]: import json
tabla.to_json('meses.json')
```

Figura 5. Escritura de un archivo JSON a partir de un DataFrame

```
{"Mes":{ "Ene":"Enero", "Feb":"Febrero", "Mar":"Marzo", "Abr":"Abril",
      "May":"Mayo", "Jun":"Junio", "Jul":"Julio", "Ago":"Agosto",
      "Sep":"Septiembre", "Oct":"Octubre", "Nov":"Noviembre", "Dic":"Diciembre" },
  "Días":{ "Ene":31, "Feb":28, "Mar":31, "Abr":30, "May":31, "Jun":30,
           "Jul":31, "Ago":31, "Sep":30, "Oct":31, "Nov":30, "Dic":31 },
  "Estación":{ "Ene":"inv", "Feb":"inv", "Mar":"inv", "Abr":"pri",
               "May":"pri", "Jun":"pri", "Jul":"ver", "Ago":"ver",
               "Sep":"ver", "Oct":"oto", "Nov":"oto", "Dic":"oto" },
  "Temp.2021":{ "Ene":2, "Feb":3, "Mar":7, "Abr":7, "May":9, "Jun":15,
                 "Jul":20, "Ago":27, "Sep":25, "Oct":20, "Nov":11, "Dic":6 },
  "Temp.2022":{ "Ene":4, "Feb":2, "Mar":5, "Abr":9, "May":10, "Jun":14,
                 "Jul":24, "Ago":26, "Sep":18, "Oct":14, "Nov":10, "Dic":7 }
}
```

Si leo este JSON en un dataframes, se visualiza correctamente

Figura 6. Archivo JSON generado.

In [90]:	tabla = pd.read_json('./meses.json')																																																																	
Out[90]:																																																																		
	<table border="1"> <thead> <tr> <th>Mes</th><th>Días</th><th>Estación</th><th>Temp.2021</th><th>Temp.2022</th></tr> </thead> <tbody> <tr><td>Ene</td><td>Enero</td><td>31</td><td>inv</td><td>2</td></tr> <tr><td>Feb</td><td>Febrero</td><td>28</td><td>inv</td><td>3</td></tr> <tr><td>Mar</td><td>Marzo</td><td>31</td><td>inv</td><td>7</td></tr> <tr><td>Abr</td><td>Abril</td><td>30</td><td>pri</td><td>7</td></tr> <tr><td>May</td><td>Mayo</td><td>31</td><td>pri</td><td>9</td></tr> <tr><td>Jun</td><td>Junio</td><td>30</td><td>pri</td><td>15</td></tr> <tr><td>Jul</td><td>Julio</td><td>31</td><td>ver</td><td>20</td></tr> <tr><td>Ago</td><td>Agosto</td><td>31</td><td>ver</td><td>27</td></tr> <tr><td>Sep</td><td>Septiembre</td><td>30</td><td>ver</td><td>25</td></tr> <tr><td>Oct</td><td>Octubre</td><td>31</td><td>oto</td><td>20</td></tr> <tr><td>Nov</td><td>Noviembre</td><td>30</td><td>oto</td><td>11</td></tr> <tr><td>Dic</td><td>Diciembre</td><td>31</td><td>oto</td><td>6</td></tr> </tbody> </table>	Mes	Días	Estación	Temp.2021	Temp.2022	Ene	Enero	31	inv	2	Feb	Febrero	28	inv	3	Mar	Marzo	31	inv	7	Abr	Abril	30	pri	7	May	Mayo	31	pri	9	Jun	Junio	30	pri	15	Jul	Julio	31	ver	20	Ago	Agosto	31	ver	27	Sep	Septiembre	30	ver	25	Oct	Octubre	31	oto	20	Nov	Noviembre	30	oto	11	Dic	Diciembre	31	oto	6
Mes	Días	Estación	Temp.2021	Temp.2022																																																														
Ene	Enero	31	inv	2																																																														
Feb	Febrero	28	inv	3																																																														
Mar	Marzo	31	inv	7																																																														
Abr	Abril	30	pri	7																																																														
May	Mayo	31	pri	9																																																														
Jun	Junio	30	pri	15																																																														
Jul	Julio	31	ver	20																																																														
Ago	Agosto	31	ver	27																																																														
Sep	Septiembre	30	ver	25																																																														
Oct	Octubre	31	oto	20																																																														
Nov	Noviembre	30	oto	11																																																														
Dic	Diciembre	31	oto	6																																																														

Figura 7. Lectura de un archivo JSON como DataFrame

Lectura y escritura de ficheros CSV y XLSX

Con las funciones `pd.read_csv` y `pd.read_excel` es posible generar *DataFrames* a partir de los datos contenidos en archivos de tipo CSV y XLSX (y xls), respectivamente. A su vez, un *DataFrame* puede ser guardado en formato CSV y XLSX con los métodos `pd.to_csv` y `pd.to_excel`, respectivamente.

Las funciones `read_csv` y `read_excel` son flexibles y permiten saltar cabeceras, saltar filas, leer un número determinado de filas o columnas, dar nuevos nombres a las columnas, etc. Su funcionamiento depende de los valores dados a los argumentos (`header`, `names`, `skiprows`, `index_col`, `nrows`) de las funciones.

En la Figura 8 se muestra la generación de un archivo CSV a partir de un *DataFrame* y posteriormente se vuelve a leer el CSV como *DataFrame*, empleando diferentes opciones como índices. En la última ejecución del ejemplo se usan varias columnas como índices jerárquicos.

```
In [126]: tabla = pd.DataFrame([('Enero', 31, 'inv', 2, 4),('Febrero', 28, 'inv', 3, 2),('Marzo', 31, 'inv', 7, 5),
                           ('Abril', 30, 'pri', 7, 9),('Mayo', 31, 'pri', 9, 10),('Junio', 30, 'pri', 15, 14),
                           ('Julio', 31, 'ver', 20, 24),('Agosto', 31, 'ver', 27, 26),('Septiembre', 30, 'ver', 25, 18),
                           ('Octubre', 31, 'oto', 20, 14),('Noviembre', 30, 'oto', 11, 10),('Diciembre', 31, 'oto', 6, 7),],
                           columns=['Mes', 'Días', 'Estación', 'Temp.2021', 'Temp.2022'],
                           index=['Ene', 'Feb', 'Mar', 'Abr', 'May', 'Jun', 'Jul', 'Ago', 'Sep', 'Oct', 'Nov', 'Dic'])
tabla.to_csv('meses.csv', header=True, index=True)

In [127]: tabla=pd.read_csv('meses.csv')
tabla
```

Out[127]:

	Unnamed: 0	Mes	Días	Estación	Temp.2021	Temp.2022
0	Ene	Enero	31	inv	2	4
1	Feb	Febrero	28	inv	3	2
2	Mar	Marzo	31	inv	7	5
3	Abr	Abril	30	pri	7	9
4	May	Mayo	31	pri	9	10
5	Jun	Junio	30	pri	15	14
6	Jul	Julio	31	ver	20	24
7	Ago	Agosto	31	ver	27	26
8	Sep	Septiembre	30	ver	25	18
9	Oct	Octubre	31	oto	20	14
10	Nov	Noviembre	30	oto	11	10
11	Dic	Diciembre	31	oto	6	7

```
In [130]: tabla=pd.read_csv('meses.csv', index_col = ['Estación', 0])
tabla
```

Out[130]:

	Estación	Mes	Días	Temp.2021	Temp.2022
inv	Ene	Enero	31	2	4
	Feb	Febrero	28	3	2
	Mar	Marzo	31	7	5
pri	Abr	Abril	30	7	9
	May	Mayo	31	9	10
	Jun	Junio	30	15	14
ver	Jul	Julio	31	20	24
	Ago	Agosto	31	27	26
	Sep	Septiembre	30	25	18
oto	Oct	Octubre	31	20	14
	Nov	Noviembre	30	11	10
	Dic	Diciembre	31	6	7

Al guardar el archivo en un CSV, La etiqueta o el index se guarda como una columna más

Genero un índice jerárquico, le indico qué columnas quiero que tome como índice

Figura 8. Escritura y lectura de un archivo CSV.

En la Figura 9 se muestra un ejemplo en el que se crea un *DataFrame* a partir del archivo *Superstore_Dataset.xlsx*, disponible en el campus virtual. Este archivo contiene datos de pedidos y devoluciones de unos grandes almacenes en dos hojas de cálculo, *Orders* y *Returns*, respectivamente. En el ejemplo de la Figura 9 se cargan únicamente los datos de la hoja *Orders*. Por defecto, si no se indica nada, la función *read_excel* lee los datos de la primera hoja dentro del fichero.

Las bases de datos contenidas en archivos suelen tener una gran extensión. El método *head* visualiza únicamente las cinco primeras filas de la tabla.

le indico la hoja que quiero visualizar, por defecto lee la 1a hoja

In [29]:	orders=pd.read_excel('Superstore_Dataset.xlsx', sheet_name='Orders') orders.head()
Out[29]:	Row ID+O6G3A1:R6 Order ID Order Date Ship Date Ship Mode Customer ID Customer Name Segment Country City State Region
0	1 CA-2019-152156 2019-11-08 2019-11-11 Second Class CG-12520 Claire Gute Consumer United States Henderson Kentucky South
1	2 CA-2019-152156 2019-11-08 2019-11-11 Second Class CG-12520 Claire Gute Consumer United States Henderson Kentucky South
2	3 CA-2019-138688 2019-06-12 2019-06-16 Second Class DV-13045 Darrin Van Huff Corporate United States Los Angeles California West
3	13 CA-2020-114412 2020-04-15 2020-04-20 Standard Class AA-10480 Andrew Allen Consumer United States Concord North Carolina South
4	14 CA-2019-161389 2019-12-05 2019-12-10 Standard Class IM-15070 Irene Maddox Consumer United States Seattle Washington West

Figura 9. Cinco primeras filas del DataFrame generado a partir de un archivo xlsx.

Si se necesita trabajar con varias hojas de cálculo de Excel, la clase *ExcelFile* de Pandas es más eficiente, ya que carga el fichero completo en memoria una única vez y permite crear posteriormente *DataFrames* a partir de cada hoja del libro, como se muestra en la Figura 10.

```
In [54]: libro = pd.ExcelFile('Superstore_Dataset.xlsx')
orders = pd.read_excel(libro, sheet_name = 'Orders')
returns = pd.read_excel(libro, sheet_name = 'Returns')
```

Figura 10. Creación de dos DataFrames a partir de distintas hojas de cálculo de un libro Excel.

UNIVERSIDAD
COMPLUTENSE
DE MADRID



ntic
master
revolucionamos la comunicación

Autor: María José
Gómez Silva

Machine Learning con Python. Semana 1.

CAPÍTULO 2. INGESTA DE DATOS.

2. 3 Manejo de los Datos en Pandas

A continuación, se presentan algunos mecanismos para acceder y actualizar los datos contenidos en estructuras de tipo Series y de tipo DataFrame, y se introducen algunas de las operaciones que es posible realizar sobre esos datos, como las de ordenación y las aritméticas.

ACCESO

El uso de etiquetas para indexar los elementos, tanto de una Serie como de un DataFrame, permite acceder a los datos con dos procedimientos: usando notación de corchetes, o usando la etiqueta como si fuera una propiedad de la serie. Además, en este caso, sería posible consultar si una determinada etiqueta está entre los índices, como en el ejemplo de la Figura 1. La Figura 1 muestra el acceso a elementos de una Serie, y la Figura 2 muestra el acceso a columnas de un DataFrame.

```
In [3]: dias = pd.Series([31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31],  
                      index=['Ene', 'Feb', 'Mar', 'Abr', 'May', 'Jun', 'Jul', 'Ago', 'Sep', 'Oct', 'Nov', 'Dic'],  
                      name='Días de cada mes')  
dias['Abr']  
Out[3]: 30  
  
In [4]: dias.Abr  
Out[4]: 30  
  
In [5]: 'Nov' in dias.index  
Out[5]: True
```

Figura 1. Acceso a un elemento de la serie y comprobación de la presencia de una determinada etiqueta.

```
In [13]: tabla = pd.DataFrame([('Enero', 31, 'inv', 2, 4),('Febrero', 28, 'inv', 3, 2),('Marzo', 31, 'inv', 7, 5),
 ('Abril', 30, 'pri', 7, 9),('Mayo', 31, 'pri', 9, 10),('Junio', 30, 'pri', 15, 14),
 ('Julio', 31, 'ven', 20, 24),('Agosto', 31, 'ven', 27, 26),('Septiembre', 30, 'ven', 25, 18),
 ('Octubre', 31, 'oto', 20, 14),('Noviembre', 30, 'oto', 11, 10),('Diciembre', 31, 'oto', 6, 7),],
 columns=['Mes', 'Días', 'Estación', 'Temp.2021', 'Temp.2022'],
 index=['Ene', 'Feb', 'Mar', 'Abr', 'May', 'Jun', 'Jul', 'Ago', 'Sep', 'Oct', 'Nov', 'Dic'])

tabla.Mes
```

```
Out[13]: Ene      Enero
Feb      Febrero
Mar      Marzo
Abr      Abril
May      Mayo
Jun      Junio
Jul      Julio
Ago      Agosto
Sep      Septiembre
Oct      Octubre
Nov      Noviembre
Dic      Diciembre
Name: Mes, dtype: object
```

```
In [14]: tabla['Mes']
```

```
Out[14]: Ene      Enero
Feb      Febrero
Mar      Marzo
Abr      Abril
May      Mayo
Jun      Junio
Jul      Julio
Ago      Agosto
Sep      Septiembre
Oct      Octubre
Nov      Noviembre
Dic      Diciembre
Name: Mes, dtype: object
```

Figura 2. Acceso a una columna de un DataFrame.

Para acceder a partes de un DataFrame se emplean los indexadores *loc* e *iloc*. El método *loc* permite una selección por etiquetas, es decir, permite acceder a partes de un DataFrame indicando el nombre de las etiquetas correspondientes a las filas y columnas a las que queremos acceder. El método *iloc* permite la selección por posición, es decir, permite acceder a partes de un DataFrame indicando el índice de las filas y columnas que queremos acceder. Hay que tener en cuenta que los índices son valores enteros que comienzan con el cero, si empezamos a contar desde el primer elemento. Si empezamos a contar desde el último elemento, en sentido inverso, los índices comienzan en -1 y son negativos. La Figura 3 muestra ejemplos del uso de *loc* e *iloc*.

```
In [15]: tabla.loc[['Abr', 'Jul'], ['Temp.2021', 'Temp.2022']]
```

```
Out[15]:
Temp.2021  Temp.2022
Abr        7        9
Jul       20       24
```

```
In [16]: tabla.iloc[[3,6],[-2, -1]]
```

```
Out[16]:
Temp.2021  Temp.2022
Abr        7        9
Jul       20       24
```

Figura 3. Empleo de los indexadores *loc* e *iloc*.

Pandas también permite la selección de elementos usando *Series* de tipo bool (True o False) que actúan como filtro. Posteriormente, el filtro se aplica para indexar los elementos de una *Serie* o *DataFrame*. La Figura 4, muestra la creación de un filtro para seleccionar los meses con más de 30 días y su aplicación a una Serie. La Figura 5 y 6 muestran las generación y aplicación, respectivamente, de más ejemplos de filtros para acceder a datos de un *DataFrame*.

```
In [17]: dias = pd.Series([31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31],  
                      index=['Ene', 'Feb', 'Mar', 'Abr', 'May', 'Jun', 'Jul', 'Ago', 'Sep', 'Oct', 'Nov', 'Dic'],  
                      name='Días de cada mes')  
filtro=dias>30  
filtro  
  
Out[17]: Ene    True  
Feb   False  
Mar    True  
Abr   False  
May    True  
Jun   False  
Jul    True  
Ago   True  
Sep   False  
Oct    True  
Nov   False  
Dic    True  
Name: Días de cada mes, dtype: bool  
  
In [18]: meses_largos=dias[filtro]  
meses_largos  
  
Out[18]: Ene    31  
Mar    31  
May    31  
Jul    31  
Ago    31  
Oct    31  
Dic    31  
Name: Días de cada mes, dtype: int64
```

Figura 4. Acceso a datos de una Serie mediante un filtro

```
In [24]: tabla = pd.DataFrame([('Enero', 31, 'inv', 2, 4),('Febrero', 28, 'inv', 3, 2),('Marzo', 31, 'inv', 7, 5),  
                           ('Abril', 30, 'pri', 7, 9),('Mayo', 31, 'pri', 9, 10),('Junio', 30, 'pri', 15, 14),  
                           ('Julio', 31, 'ver', 20, 24),('Agosto', 31, 'ver', 27, 26),('Septiembre', 30, 'ver', 25, 18),  
                           ('Octubre', 31, 'oto', 20, 14),('Noviembre', 30, 'oto', 11, 18),('Diciembre', 31, 'oto', 6, 7),],  
                           columns=['Mes', 'Días', 'Estación', 'Temp.2021', 'Temp.2022'],  
                           index=['Ene', 'Feb', 'Mar', 'Abr', 'May', 'Jun', 'Jul', 'Ago', 'Sep', 'Oct', 'Nov', 'Dic'])  
filtro1=tabla['Estación']=='oto'  
filtro1  
  
Out[24]: Ene   False  
Feb   False  
Mar   False  
Abr   False  
May   False  
Jun   False  
Jul   False  
Ago   False  
Sep   False  
Oct    True  
Nov   True  
Dic    True  
Name: Estación, dtype: bool  
  
In [25]: filtro2=tabla['Temp.2021']>10  
filtro2  
  
Out[25]: Ene   False  
Feb   False  
Mar   False  
Abr   False  
May   False  
Jun    True  
Jul    True  
Ago    True  
Sep    True  
Oct    True  
Nov    True  
Dic   False  
Name: Temp.2021, dtype: bool
```

Figura 5. Generación de filtros a partir de un DataFrame

In [26]:	tabla[filtro1]																																								
Out[26]:	<table> <thead> <tr> <th>Mes</th><th>Días</th><th>Estación</th><th>Temp.2021</th><th>Temp.2022</th></tr> </thead> <tbody> <tr> <td>Oct</td><td>Octubre</td><td>31</td><td>oto</td><td>20 14</td></tr> <tr> <td>Nov</td><td>Noviembre</td><td>30</td><td>oto</td><td>11 10</td></tr> <tr> <td>Dic</td><td>Diciembre</td><td>31</td><td>oto</td><td>6 7</td></tr> </tbody> </table>	Mes	Días	Estación	Temp.2021	Temp.2022	Oct	Octubre	31	oto	20 14	Nov	Noviembre	30	oto	11 10	Dic	Diciembre	31	oto	6 7																				
Mes	Días	Estación	Temp.2021	Temp.2022																																					
Oct	Octubre	31	oto	20 14																																					
Nov	Noviembre	30	oto	11 10																																					
Dic	Diciembre	31	oto	6 7																																					
In [27]:	tabla[filtro2]																																								
Out[27]:	<table> <thead> <tr> <th>Mes</th><th>Días</th><th>Estación</th><th>Temp.2021</th><th>Temp.2022</th></tr> </thead> <tbody> <tr> <td>Jun</td><td>Junio</td><td>30</td><td>pri</td><td>15 14</td></tr> <tr> <td>Jul</td><td>Julio</td><td>31</td><td>ver</td><td>20 24</td></tr> <tr> <td>Ago</td><td>Agosto</td><td>31</td><td>ver</td><td>27 26</td></tr> <tr> <td>Sep</td><td>Septiembre</td><td>30</td><td>ver</td><td>25 18</td></tr> <tr> <td>Oct</td><td>Octubre</td><td>31</td><td>oto</td><td>20 14</td></tr> <tr> <td>Nov</td><td>Noviembre</td><td>30</td><td>oto</td><td>11 10</td></tr> </tbody> </table>	Mes	Días	Estación	Temp.2021	Temp.2022	Jun	Junio	30	pri	15 14	Jul	Julio	31	ver	20 24	Ago	Agosto	31	ver	27 26	Sep	Septiembre	30	ver	25 18	Oct	Octubre	31	oto	20 14	Nov	Noviembre	30	oto	11 10					
Mes	Días	Estación	Temp.2021	Temp.2022																																					
Jun	Junio	30	pri	15 14																																					
Jul	Julio	31	ver	20 24																																					
Ago	Agosto	31	ver	27 26																																					
Sep	Septiembre	30	ver	25 18																																					
Oct	Octubre	31	oto	20 14																																					
Nov	Noviembre	30	oto	11 10																																					
In [28]:	tabla[filtro1 & filtro2]																																								
Out[28]:	<table> <thead> <tr> <th>Mes</th><th>Días</th><th>Estación</th><th>Temp.2021</th><th>Temp.2022</th></tr> </thead> <tbody> <tr> <td>Oct</td><td>Octubre</td><td>31</td><td>oto</td><td>20 14</td></tr> <tr> <td>Nov</td><td>Noviembre</td><td>30</td><td>oto</td><td>11 10</td></tr> </tbody> </table>	Mes	Días	Estación	Temp.2021	Temp.2022	Oct	Octubre	31	oto	20 14	Nov	Noviembre	30	oto	11 10																									
Mes	Días	Estación	Temp.2021	Temp.2022																																					
Oct	Octubre	31	oto	20 14																																					
Nov	Noviembre	30	oto	11 10																																					
In [29]:	tabla[filtro1 filtro2]																																								
Out[29]:	<table> <thead> <tr> <th>Mes</th><th>Días</th><th>Estación</th><th>Temp.2021</th><th>Temp.2022</th></tr> </thead> <tbody> <tr> <td>Jun</td><td>Junio</td><td>30</td><td>pri</td><td>15 14</td></tr> <tr> <td>Jul</td><td>Julio</td><td>31</td><td>ver</td><td>20 24</td></tr> <tr> <td>Ago</td><td>Agosto</td><td>31</td><td>ver</td><td>27 26</td></tr> <tr> <td>Sep</td><td>Septiembre</td><td>30</td><td>ver</td><td>25 18</td></tr> <tr> <td>Oct</td><td>Octubre</td><td>31</td><td>oto</td><td>20 14</td></tr> <tr> <td>Nov</td><td>Noviembre</td><td>30</td><td>oto</td><td>11 10</td></tr> <tr> <td>Dic</td><td>Diciembre</td><td>31</td><td>oto</td><td>6 7</td></tr> </tbody> </table>	Mes	Días	Estación	Temp.2021	Temp.2022	Jun	Junio	30	pri	15 14	Jul	Julio	31	ver	20 24	Ago	Agosto	31	ver	27 26	Sep	Septiembre	30	ver	25 18	Oct	Octubre	31	oto	20 14	Nov	Noviembre	30	oto	11 10	Dic	Diciembre	31	oto	6 7
Mes	Días	Estación	Temp.2021	Temp.2022																																					
Jun	Junio	30	pri	15 14																																					
Jul	Julio	31	ver	20 24																																					
Ago	Agosto	31	ver	27 26																																					
Sep	Septiembre	30	ver	25 18																																					
Oct	Octubre	31	oto	20 14																																					
Nov	Noviembre	30	oto	11 10																																					
Dic	Diciembre	31	oto	6 7																																					

Figura 6. Acceso a los datos de un Dataframe mediante filtros

ACTUALIZACIÓN

Pandas permite eliminar filas y columnas, crear otras nuevas y modificar los valores de los datos contenidas en las ya existentes.

Es posible cambiar los valores de toda una columna, dando el mismo valor a todos sus elementos, o especificando el valor de cada uno de ellos. De igual modo es posible modificar los valores almacenados en filas y columnas concretas indexándolas apropiadamente, como se muestra en la Figura 7.

In [30]:	tabla = pd.DataFrame([('Enero', 31, 'inv', 2, 4), ('Febrero', 28, 'inv', 3, 2), ('Marzo', 31, 'inv', 7, 5), ('Abril', 30, 'pri', 7, 9), ('Mayo', 31, 'pri', 9, 10), ('Junio', 30, 'pri', 15, 14), ('Julio', 31, 'ver', 20, 24), ('Agosto', 31, 'ver', 27, 26), ('Septiembre', 30, 'ver', 25, 18), ('Octubre', 31, 'oto', 20, 14), ('Noviembre', 30, 'oto', 11, 10), ('Diciembre', 31, 'oto', 6, 7)], columns=['Mes', 'Días', 'Estación', 'Temp.2021', 'Temp.2022'], index=['Ene', 'Feb', 'Mar', 'Abr', 'May', 'Jun', 'Jul', 'Ago', 'Sep', 'Oct', 'Nov', 'Dic'])	tabla['Estación']='oto'																																																																														
Out[30]:	<table border="1"> <thead> <tr><th></th><th>Mes</th><th>Días</th><th>Estación</th><th>Temp.2021</th><th>Temp.2022</th></tr> </thead> <tbody> <tr><td>Ene</td><td>Enero</td><td>31</td><td>oto</td><td>2</td><td>4</td></tr> <tr><td>Feb</td><td>Febrero</td><td>28</td><td>oto</td><td>3</td><td>2</td></tr> <tr><td>Mar</td><td>Marzo</td><td>31</td><td>oto</td><td>7</td><td>5</td></tr> <tr><td>Abr</td><td>Abri</td><td>30</td><td>oto</td><td>7</td><td>9</td></tr> <tr><td>May</td><td>Mayo</td><td>31</td><td>oto</td><td>9</td><td>10</td></tr> <tr><td>Jun</td><td>Junio</td><td>30</td><td>oto</td><td>15</td><td>14</td></tr> <tr><td>Jul</td><td>Julio</td><td>31</td><td>oto</td><td>20</td><td>24</td></tr> <tr><td>Ago</td><td>Agosto</td><td>31</td><td>oto</td><td>27</td><td>26</td></tr> <tr><td>Sep</td><td>Septiembre</td><td>30</td><td>oto</td><td>25</td><td>18</td></tr> <tr><td>Oct</td><td>Octubre</td><td>31</td><td>oto</td><td>20</td><td>14</td></tr> <tr><td>Nov</td><td>Noviembre</td><td>30</td><td>oto</td><td>11</td><td>10</td></tr> <tr><td>Dic</td><td>Diciembre</td><td>31</td><td>oto</td><td>6</td><td>7</td></tr> </tbody> </table>			Mes	Días	Estación	Temp.2021	Temp.2022	Ene	Enero	31	oto	2	4	Feb	Febrero	28	oto	3	2	Mar	Marzo	31	oto	7	5	Abr	Abri	30	oto	7	9	May	Mayo	31	oto	9	10	Jun	Junio	30	oto	15	14	Jul	Julio	31	oto	20	24	Ago	Agosto	31	oto	27	26	Sep	Septiembre	30	oto	25	18	Oct	Octubre	31	oto	20	14	Nov	Noviembre	30	oto	11	10	Dic	Diciembre	31	oto	6	7
	Mes	Días	Estación	Temp.2021	Temp.2022																																																																											
Ene	Enero	31	oto	2	4																																																																											
Feb	Febrero	28	oto	3	2																																																																											
Mar	Marzo	31	oto	7	5																																																																											
Abr	Abri	30	oto	7	9																																																																											
May	Mayo	31	oto	9	10																																																																											
Jun	Junio	30	oto	15	14																																																																											
Jul	Julio	31	oto	20	24																																																																											
Ago	Agosto	31	oto	27	26																																																																											
Sep	Septiembre	30	oto	25	18																																																																											
Oct	Octubre	31	oto	20	14																																																																											
Nov	Noviembre	30	oto	11	10																																																																											
Dic	Diciembre	31	oto	6	7																																																																											
In [35]: tabla[['Estación']] = ['inv', 'inv', 'inv', 'pri', 'pri', 'pri', 'ver', 'ver', 'ver', 'oto', 'oto', 'oto']																																																																																
tabla.loc[['Jun'], ['Temp.2022']] = 16																																																																																
Out[35]:	<table border="1"> <thead> <tr><th></th><th>Mes</th><th>Días</th><th>Estación</th><th>Temp.2021</th><th>Temp.2022</th></tr> </thead> <tbody> <tr><td>Ene</td><td>Enero</td><td>31</td><td>inv</td><td>2</td><td>4</td></tr> <tr><td>Feb</td><td>Febrero</td><td>28</td><td>inv</td><td>3</td><td>2</td></tr> <tr><td>Mar</td><td>Marzo</td><td>31</td><td>inv</td><td>7</td><td>5</td></tr> <tr><td>Abr</td><td>Abri</td><td>30</td><td>pri</td><td>7</td><td>9</td></tr> <tr><td>May</td><td>Mayo</td><td>31</td><td>pri</td><td>9</td><td>10</td></tr> <tr><td>Jun</td><td>Junio</td><td>30</td><td>pri</td><td>15</td><td>16</td></tr> <tr><td>Jul</td><td>Julio</td><td>31</td><td>ver</td><td>20</td><td>24</td></tr> <tr><td>Ago</td><td>Agosto</td><td>31</td><td>ver</td><td>27</td><td>26</td></tr> <tr><td>Sep</td><td>Septiembre</td><td>30</td><td>ver</td><td>25</td><td>18</td></tr> <tr><td>Oct</td><td>Octubre</td><td>31</td><td>oto</td><td>20</td><td>14</td></tr> <tr><td>Nov</td><td>Noviembre</td><td>30</td><td>oto</td><td>11</td><td>10</td></tr> <tr><td>Dic</td><td>Diciembre</td><td>31</td><td>oto</td><td>6</td><td>7</td></tr> </tbody> </table>			Mes	Días	Estación	Temp.2021	Temp.2022	Ene	Enero	31	inv	2	4	Feb	Febrero	28	inv	3	2	Mar	Marzo	31	inv	7	5	Abr	Abri	30	pri	7	9	May	Mayo	31	pri	9	10	Jun	Junio	30	pri	15	16	Jul	Julio	31	ver	20	24	Ago	Agosto	31	ver	27	26	Sep	Septiembre	30	ver	25	18	Oct	Octubre	31	oto	20	14	Nov	Noviembre	30	oto	11	10	Dic	Diciembre	31	oto	6	7
	Mes	Días	Estación	Temp.2021	Temp.2022																																																																											
Ene	Enero	31	inv	2	4																																																																											
Feb	Febrero	28	inv	3	2																																																																											
Mar	Marzo	31	inv	7	5																																																																											
Abr	Abri	30	pri	7	9																																																																											
May	Mayo	31	pri	9	10																																																																											
Jun	Junio	30	pri	15	16																																																																											
Jul	Julio	31	ver	20	24																																																																											
Ago	Agosto	31	ver	27	26																																																																											
Sep	Septiembre	30	ver	25	18																																																																											
Oct	Octubre	31	oto	20	14																																																																											
Nov	Noviembre	30	oto	11	10																																																																											
Dic	Diciembre	31	oto	6	7																																																																											

Figura 7. Actualización de datos de un DataFrame

Para eliminar datos de un DataFrame se pueden emplear las operaciones *pop* y *drop*, como muestran las Figuras 8 y 9. La primera permite eliminar una columna, modificando el *DataFrame*, y devuelve la columna eliminada en una estructura de tipo *Series*.

En cambio, *drop* permite eliminar tanto filas como columnas, pero no modifica el *DataFrame* original, sino que devuelve una copia, y no devuelve los datos eliminados. Si queremos que *drop* modifique el *DataFrame*, debemos usar el argumento *inplace* con valor True.

```
In [36]: tabla.pop('Temp.2022')
out[36]: Ene    4
          Feb    2
          Mar    5
          Abr    9
          May   10
          Jun   16
          Jul   24
          Ago   26
          Sep   18
          Oct   14
          Nov   10
          Dic    7
Name: Temp.2022, dtype: int64
```

```
In [37]: tabla
out[37]:
```

	Mes	Días	Estación	Temp.2021
Ene	Enero	31	inv	2
Feb	Febrero	28	inv	3
Mar	Marzo	31	inv	7
Abr	Abril	30	pri	7
May	Mayo	31	pri	9
Jun	Junio	30	pri	15
Jul	Julio	31	ver	20
Ago	Agosto	31	ver	27
Sep	Septiembre	30	ver	25
Oct	Octubre	31	oto	20
Nov	Noviembre	30	oto	11
Dic	Diciembre	31	oto	6

Figura 8. Eliminación de datos con la operación pop

```
In [81]: nueva_tabla=tabla.drop(['Jul', 'Ago', 'Sep'])
nueva_tabla
```

```
out[81]:
```

	Mes	Días	Estación	Temp.2021
Ene	Enero	31	inv	2
Feb	Febrero	28	inv	3
Mar	Marzo	31	inv	7
Abr	Abril	30	pri	7
May	Mayo	31	pri	9
Jun	Junio	30	pri	15
Oct	Octubre	31	oto	20
Nov	Noviembre	30	oto	11
Dic	Diciembre	31	oto	6


```
In [82]: tabla
```

```
out[82]:
```

	Mes	Días	Estación	Temp.2021
Ene	Enero	31	inv	2
Feb	Febrero	28	inv	3
Mar	Marzo	31	inv	7
Abr	Abril	30	pri	7
May	Mayo	31	pri	9
Jun	Junio	30	pri	15
Jul	Julio	31	ver	20
Ago	Agosto	31	ver	27
Sep	Septiembre	30	ver	25
Oct	Octubre	31	oto	20
Nov	Noviembre	30	oto	11
Dic	Diciembre	31	oto	6

la tabla original no sufre cambios, se genera una copia en la que se elimina la columna

Figura 9. Eliminación de datos con la operación drop

Pandas también permite añadir filas (con el método *append*) y columnas a una tabla, como se muestra en la Figura 10.

In [83]:	nueva_tabla=nueva_tabla.append(pd.Series({'Mes':'Julio', 'Dias': 31, 'Estación': 'ver', 'Temp.2021': 20}, name='Jul')) nueva_tabla=nueva_tabla.append(pd.Series({'Mes':'Agosto', 'Dias': 31, 'Estación': 'ver', 'Temp.2021': 27}, name='Ago')) nueva_tabla=nueva_tabla.append(pd.Series({'Mes':'Septiembre', 'Dias': 30, 'Estación': 'ver', 'Temp.2021': 25}, name='Sep')) nueva_tabla																																																																														
Out[83]:	<table><thead><tr><th></th><th>Mes</th><th>Días</th><th>Estación</th><th>Temp.2021</th></tr></thead><tbody><tr><td>Ene</td><td>Enero</td><td>31</td><td>inv</td><td>2</td></tr><tr><td>Feb</td><td>Febrero</td><td>28</td><td>inv</td><td>3</td></tr><tr><td>Mar</td><td>Marzo</td><td>31</td><td>inv</td><td>7</td></tr><tr><td>Abr</td><td>Abrial</td><td>30</td><td>pri</td><td>7</td></tr><tr><td>May</td><td>Mayo</td><td>31</td><td>pri</td><td>9</td></tr><tr><td>Jun</td><td>Junio</td><td>30</td><td>pri</td><td>15</td></tr><tr><td>Oct</td><td>Octubre</td><td>31</td><td>oto</td><td>20</td></tr><tr><td>Nov</td><td>Noviembre</td><td>30</td><td>oto</td><td>11</td></tr><tr><td>Dic</td><td>Diciembre</td><td>31</td><td>oto</td><td>6</td></tr><tr><td>Jul</td><td>Julio</td><td>31</td><td>ver</td><td>20</td></tr><tr><td>Ago</td><td>Agosto</td><td>31</td><td>ver</td><td>27</td></tr><tr><td>Sep</td><td>Septiembre</td><td>30</td><td>ver</td><td>25</td></tr></tbody></table>		Mes	Días	Estación	Temp.2021	Ene	Enero	31	inv	2	Feb	Febrero	28	inv	3	Mar	Marzo	31	inv	7	Abr	Abrial	30	pri	7	May	Mayo	31	pri	9	Jun	Junio	30	pri	15	Oct	Octubre	31	oto	20	Nov	Noviembre	30	oto	11	Dic	Diciembre	31	oto	6	Jul	Julio	31	ver	20	Ago	Agosto	31	ver	27	Sep	Septiembre	30	ver	25													
	Mes	Días	Estación	Temp.2021																																																																											
Ene	Enero	31	inv	2																																																																											
Feb	Febrero	28	inv	3																																																																											
Mar	Marzo	31	inv	7																																																																											
Abr	Abrial	30	pri	7																																																																											
May	Mayo	31	pri	9																																																																											
Jun	Junio	30	pri	15																																																																											
Oct	Octubre	31	oto	20																																																																											
Nov	Noviembre	30	oto	11																																																																											
Dic	Diciembre	31	oto	6																																																																											
Jul	Julio	31	ver	20																																																																											
Ago	Agosto	31	ver	27																																																																											
Sep	Septiembre	30	ver	25																																																																											
In [84]:	nueva_tabla=nueva_tabla.reindex(['Ene', 'Feb', 'Mar', 'Abr', 'May', 'Jun', 'Jul', 'Ago', 'Sep', 'Oct', 'Nov', 'Dic']) nueva_tabla																																																																														
Out[84]:	<table><thead><tr><th></th><th>Mes</th><th>Días</th><th>Estación</th><th>Temp.2021</th></tr></thead><tbody><tr><td>Ene</td><td>Enero</td><td>31</td><td>inv</td><td>2</td></tr><tr><td>Feb</td><td>Febrero</td><td>28</td><td>inv</td><td>3</td></tr><tr><td>Mar</td><td>Marzo</td><td>31</td><td>inv</td><td>7</td></tr><tr><td>Abr</td><td>Abrial</td><td>30</td><td>pri</td><td>7</td></tr><tr><td>May</td><td>Mayo</td><td>31</td><td>pri</td><td>9</td></tr><tr><td>Jun</td><td>Junio</td><td>30</td><td>pri</td><td>15</td></tr><tr><td>Jul</td><td>Julio</td><td>31</td><td>ver</td><td>20</td></tr><tr><td>Ago</td><td>Agosto</td><td>31</td><td>ver</td><td>27</td></tr><tr><td>Sep</td><td>Septiembre</td><td>30</td><td>ver</td><td>25</td></tr><tr><td>Oct</td><td>Octubre</td><td>31</td><td>oto</td><td>20</td></tr><tr><td>Nov</td><td>Noviembre</td><td>30</td><td>oto</td><td>11</td></tr><tr><td>Dic</td><td>Diciembre</td><td>31</td><td>oto</td><td>6</td></tr></tbody></table>		Mes	Días	Estación	Temp.2021	Ene	Enero	31	inv	2	Feb	Febrero	28	inv	3	Mar	Marzo	31	inv	7	Abr	Abrial	30	pri	7	May	Mayo	31	pri	9	Jun	Junio	30	pri	15	Jul	Julio	31	ver	20	Ago	Agosto	31	ver	27	Sep	Septiembre	30	ver	25	Oct	Octubre	31	oto	20	Nov	Noviembre	30	oto	11	Dic	Diciembre	31	oto	6													
	Mes	Días	Estación	Temp.2021																																																																											
Ene	Enero	31	inv	2																																																																											
Feb	Febrero	28	inv	3																																																																											
Mar	Marzo	31	inv	7																																																																											
Abr	Abrial	30	pri	7																																																																											
May	Mayo	31	pri	9																																																																											
Jun	Junio	30	pri	15																																																																											
Jul	Julio	31	ver	20																																																																											
Ago	Agosto	31	ver	27																																																																											
Sep	Septiembre	30	ver	25																																																																											
Oct	Octubre	31	oto	20																																																																											
Nov	Noviembre	30	oto	11																																																																											
Dic	Diciembre	31	oto	6																																																																											
In [85]:	nueva_tabla['Temp.2022']=[4,2,5,9,10,16,24,26,18,14,10,7] nueva_tabla																																																																														
Out[85]:	<table><thead><tr><th></th><th>Mes</th><th>Días</th><th>Estación</th><th>Temp.2021</th><th>Temp.2022</th></tr></thead><tbody><tr><td>Ene</td><td>Enero</td><td>31</td><td>inv</td><td>2</td><td>4</td></tr><tr><td>Feb</td><td>Febrero</td><td>28</td><td>inv</td><td>3</td><td>2</td></tr><tr><td>Mar</td><td>Marzo</td><td>31</td><td>inv</td><td>7</td><td>5</td></tr><tr><td>Abr</td><td>Abrial</td><td>30</td><td>pri</td><td>7</td><td>9</td></tr><tr><td>May</td><td>Mayo</td><td>31</td><td>pri</td><td>9</td><td>10</td></tr><tr><td>Jun</td><td>Junio</td><td>30</td><td>pri</td><td>15</td><td>16</td></tr><tr><td>Jul</td><td>Julio</td><td>31</td><td>ver</td><td>20</td><td>24</td></tr><tr><td>Ago</td><td>Agosto</td><td>31</td><td>ver</td><td>27</td><td>26</td></tr><tr><td>Sep</td><td>Septiembre</td><td>30</td><td>ver</td><td>25</td><td>18</td></tr><tr><td>Oct</td><td>Octubre</td><td>31</td><td>oto</td><td>20</td><td>14</td></tr><tr><td>Nov</td><td>Noviembre</td><td>30</td><td>oto</td><td>11</td><td>10</td></tr><tr><td>Dic</td><td>Diciembre</td><td>31</td><td>oto</td><td>6</td><td>7</td></tr></tbody></table>		Mes	Días	Estación	Temp.2021	Temp.2022	Ene	Enero	31	inv	2	4	Feb	Febrero	28	inv	3	2	Mar	Marzo	31	inv	7	5	Abr	Abrial	30	pri	7	9	May	Mayo	31	pri	9	10	Jun	Junio	30	pri	15	16	Jul	Julio	31	ver	20	24	Ago	Agosto	31	ver	27	26	Sep	Septiembre	30	ver	25	18	Oct	Octubre	31	oto	20	14	Nov	Noviembre	30	oto	11	10	Dic	Diciembre	31	oto	6	7
	Mes	Días	Estación	Temp.2021	Temp.2022																																																																										
Ene	Enero	31	inv	2	4																																																																										
Feb	Febrero	28	inv	3	2																																																																										
Mar	Marzo	31	inv	7	5																																																																										
Abr	Abrial	30	pri	7	9																																																																										
May	Mayo	31	pri	9	10																																																																										
Jun	Junio	30	pri	15	16																																																																										
Jul	Julio	31	ver	20	24																																																																										
Ago	Agosto	31	ver	27	26																																																																										
Sep	Septiembre	30	ver	25	18																																																																										
Oct	Octubre	31	oto	20	14																																																																										
Nov	Noviembre	30	oto	11	10																																																																										
Dic	Diciembre	31	oto	6	7																																																																										

Figura 10. Añadido de columnas y filas a un DataFrame

OPERACIONES ARITMÉTICAS

Las *Series* y *DataFrames* proporcionan claridad en cuanto a los datos que contienen y su acceso, y sobre ellas pueden realizarse las funciones definidas en la librería *Numpy*, que permiten aplicar operaciones aritméticas (+, -, *, /) entre otras.

Una característica muy importante en la realización de operaciones con Pandas es su capacidad para alinear dos estructuras (*Series* o *DataFrames*) de acuerdo con las etiquetas de sus índices. Esta capacidad es muy útil cuando se realizan operaciones entre estructuras que no tienen exactamente las mismas etiquetas en sus índices, o estos están desordenados, como en los ejemplos de la Figura 11.

```
In [88]: cosecha_finca1= pd.DataFrame([[300, 400], [100, 500], [400, 200], [600, 100]],
                                         columns=['trigo', 'maiz'), index=[2018, 2017, 2021, 2022])
cosecha_finca1
Out[88]:
      trigo  maiz
2018    300   400
2017    100   500
2021    400   200
2022    600   100

In [89]: cosecha_finca2= pd.DataFrame([[500, 300], [700, 400], [200, 800], [500, 600]],
                                         columns=['trigo', 'maiz'), index=[2017, 2018, 2022, 2020])
cosecha_finca2
Out[89]:
      trigo  maiz
2017    500   300
2018    700   400
2022    200   800
2020    500   600

In [92]: cosecha_total = cosecha_finca1 + cosecha_finca2
cosecha_total
Out[92]:
      trigo  maiz
2017  600.0  800.0
2018 1000.0  800.0
2020   NaN   NaN
2021   NaN   NaN
2022  800.0  900.0

In [93]: cosecha_finca1.add(cosecha_finca2, fill_value = 0)
Out[93]:
      trigo  maiz
2017  600.0  800.0
2018 1000.0  800.0
2020  500.0  600.0
2021  400.0  200.0
2022  800.0  900.0
```

Figura 11. Ejemplos de suma de dos *DataFrames*

ORDENACIÓN

Pandas ofrece los métodos `sort_index` y `sort_values` para ordenar, respectivamente, los índices y los valores de una Serie o DataFrame, como muestra la Figura 12. Estos métodos `sort_index` no modifican el DataFrame original, sino que devuelven una copia. Si queremos que modifiquen el DataFrame, hay que indicarlo dando el valor True al argumento `inplace`.

```
In [94]: cosecha_finca2.sort_index(ascending=False)
Out[94]:
      trigo  maiz
2022    200   800
2020    500   600
2018    700   400
2017    500   300

In [95]: cosecha_finca2.sort_values(by=['maiz'])
Out[95]:
      trigo  maiz
2017    500   300
2018    700   400
2020    500   600
2022    200   800
```

Figura 12. Ordenación de índices y valores de un DataFrame

UNIVERSIDAD
COMPLUTENSE
DE MADRID



ntic
master
revolucionamos la comunicación

Autor: María José
Gómez Silva

Machine Learning con Python. Semana 1.

CAPÍTULO 2. INGESTA DE DATOS.

2. 4 Relaciones entre tablas

En ocasiones, los datos necesarios para alimentar un algoritmo de Machine Learning están distribuidos en varios *DataFrames*. En esos casos es necesario combinar los *DataFrames* en uno nuevo que contenga toda la información necesaria.

Pandas ofrece la función `pd.merge` que permite combinar las filas de dos *DataFrames* de acuerdo a la comparación del valor de una o varias columnas. Esta operación es similar al operador *join* del lenguaje SQL empleado sobre bases de datos relacionales. A la función `pd.merge` hay que indicarle los dos *DataFrames* que tiene que relacionar y el resultado será un nuevo *DataFrame*. Para definir la relación a aplicar se dispone de dos argumentos, `on` y `how`. Las columnas de combinación se indican mediante una lista en el argumento `on`. Y el tipo de combinación se indica en el argumento `how`. Los tipos de combinaciones posibles son `left`, `right`, `outer` e `inner`.

Vamos a estudiar el funcionamiento de las combinaciones mencionadas, empleando dos tablas, una con datos sobre las calificaciones obtenidas por algunos alumnos en algunas materias, y otra con el listado de los profesores que imparten algunas de las materias. En este caso, la columna en común y que relaciona ambas tablas es ‘*materia*’, como puede observarse en la Figura 1.

```
In [70]: tabla1 = pd.DataFrame({'Alumno':['Pedro', 'Pedro', 'Irene', 'Samuel'],
                           'Materia':['Matemáticas', 'Literatura', 'Matemáticas', 'Tecnología'],
                           'Nota':[8.2, 3.4, 5.8, 6.7]})

Out[70]:
      Alumno    Materia  Nota
0     Pedro  Matemáticas   8.2
1     Pedro    Literatura   3.4
2     Irene  Matemáticas   5.8
3   Samuel    Tecnología   6.7

In [72]: tabla2 = pd.DataFrame({'Materia':['Matemáticas', 'Literatura', 'Biología'],
                           'Profesor':['Rodríguez', 'Díaz', 'García']})

Out[72]:
      Materia  Profesor
0  Matemáticas  Rodríguez
1   Literatura       Díaz
2    Biología     García
```

Figura 1. DataFrames relacionados por la columna ‘materia’.

Emplearemos la columna ‘Materia’ como columna de comparación.

La operación de combinación de tipo *inner* devuelve únicamente aquellas filas que tienen valores idénticos en la columna que se compara. Si no se indica nada, la función *pd.merge* realiza por defecto una combinación de tipo *inner*.

La combinación de tipo *left* devuelve un *DataFrame* con las filas que tienen valores idénticos en la columna comparada y todas las filas del *DataFrame* de la izquierda (con o sin correspondencia con el *DataFrame* de la derecha). Las celdas sin correspondencia se rellenan con NaN.

De forma análoga, la combinación de tipo *right* devuelve un *DataFrame* con las filas que tienen valores idénticos en la columna comparada y todas las filas del *DataFrame* de la derecha (con o sin correspondencia con el *DataFrame* de la izquierda).

Por último, la combinación de tipo *outer*, devuelve la unión de las filas devueltas por las opciones *left* y *right*. La figura 2 muestra el resultado de los cuatro tipos de combinaciones empleando las tablas de la Figura 1.

```
In [74]: resultado=pd.merge(tabla1, tabla2, on=['Materia'], how='inner')
resultado
```

```
Out[74]:
```

	Alumno	Materia	Nota	Profesor
0	Pedro	Matemáticas	8.2	Rodríguez
1	Irene	Matemáticas	5.8	Rodríguez
2	Pedro	Literatura	3.4	Díaz

```
In [75]: resultado=pd.merge(tabla1, tabla2, on=['Materia'], how='left')
resultado
```

```
Out[75]:
```

	Alumno	Materia	Nota	Profesor
0	Pedro	Matemáticas	8.2	Rodríguez
1	Pedro	Literatura	3.4	Díaz
2	Irene	Matemáticas	5.8	Rodríguez
3	Samuel	Tecnología	6.7	NaN

```
In [76]: resultado=pd.merge(tabla1, tabla2, on=['Materia'], how='right')
resultado
```

```
Out[76]:
```

	Alumno	Materia	Nota	Profesor
0	Pedro	Matemáticas	8.2	Rodríguez
1	Irene	Matemáticas	5.8	Rodríguez
2	Pedro	Literatura	3.4	Díaz
3	NaN	Biología	NaN	García

```
In [77]: resultado=pd.merge(tabla1, tabla2, on=['Materia'], how='outer')
resultado
```

```
Out[77]:
```

	Alumno	Materia	Nota	Profesor
0	Pedro	Matemáticas	8.2	Rodríguez
1	Irene	Matemáticas	5.8	Rodríguez
2	Pedro	Literatura	3.4	Díaz
3	Samuel	Tecnología	6.7	NaN
4	NaN	Biología	NaN	García

Figura 2. Resultado de aplicar los cuatro tipos de combinaciones de *DataFrames* disponibles con la función *pd.merge*.

En el ejemplo anterior, ambos *DataFrames* contenían una columna con el mismo nombre, *Materia*, que ha sido usada para realizar las comparaciones. Si esta columna recibiese nombres distintos en cada *DataFrame*, por ejemplo, *Materia* en la tabla 1 y *Asignatura* en la tabla 2, habría que indicar los nombres de estas dos columnas a comparar en la función *pd.merge*, tal y como se muestra en la Figura 3, con los comandos *left_on* y *right_on*.

```
In [79]: tabla1 = pd.DataFrame({'Alumno':['Pedro', 'Pedro', 'Irene', 'Samuel'],
                           'Materia':['Matemáticas', 'Literatura', 'Matemáticas', 'Tecnología'],
                           'Nota':[8.2, 3.4, 5.8, 6.7]})
tabla1
```

```
Out[79]:
      Alumno    Materia  Nota
0     Pedro  Matemáticas   8.2
1     Pedro    Literatura   3.4
2     Irene  Matemáticas   5.8
3   Samuel  Tecnología   6.7
```



```
In [78]: tabla2 = pd.DataFrame({'Asignatura':['Matemáticas', 'Literatura', 'Biología'],
                           'Profesor':['Rodríguez', 'Díaz', 'García']})
tabla2
```

```
Out[78]:
      Asignatura  Profesor
0  Matemáticas  Rodríguez
1  Literatura      Díaz
2    Biología     García
```



```
In [80]: resultado=pd.merge(tabla1, tabla2, left_on=['Materia'], right_on=['Asignatura'], how='inner')
resultado
```

```
Out[80]:
      Alumno    Materia  Nota  Asignatura  Profesor
0     Pedro  Matemáticas   8.2  Matemáticas  Rodríguez
1     Irene  Matemáticas   5.8  Matemáticas  Rodríguez
2     Pedro    Literatura   3.4    Literatura      Díaz
```

Figura 3. Comparación de columnas con distintos nombres en cada DataFrame.

En algunos casos, los valores a comparar no están en las columnas en común, si no en los índices de los *DataFrames*. En ese caso, habrá que emplear los argumentos *left_index* y *right_index* de la función *pd.merge* (en lugar de *left_on* y *right_on*). Los argumentos *left_index* y *right_index* solo admiten los valores True o False, dependiendo, respectivamente, de si se deben usar o no los índices como valores de comparación. También es posible realizar la comparación entre las columnas de un *DataFrame* y los índices del otro, combinando correctamente los argumentos anteriores.

Cuando se trata de recopilar información distribuida en varios *DataFrames*, además de la función *pd.merge* para combinarlos, Pandas ofrece la función *pd.concat* para concatenarlos. Esta función forma un nuevo *DataFrame* uniendo la información contenida en otros dos, apilándola horizontal (axis=1) o verticalmente (axis=0), o lo que es lo mismo, uniendo sus columnas o sus filas, como muestra la Figura 4.

```
In [83]: tabla1 = pd.DataFrame({'Materia':['Matemáticas', 'Matemáticas', 'Tecnología'],
                               'Nota':[8.2, 3.4, 6.7]},
                               index = ['Pedro', 'Irene', 'Samuel'])
tabla1
```

	Materia	Nota
Pedro	Matemáticas	8.2
Irene	Matemáticas	3.4
Samuel	Tecnología	6.7


```
In [84]: tabla2 = pd.DataFrame({'Curso':[2, 1],
                               'Tutor':['Rodríguez', 'Díaz']},
                               index = ['Pedro', 'Irene'])
tabla2
```

	Curso	Tutor
Pedro	2	Rodríguez
Irene	1	Díaz


```
In [86]: resultado = pd.concat([tabla1, tabla2], axis=1)    columnas
```

	Materia	Nota	Curso	Tutor
Pedro	Matemáticas	8.2	2.0	Rodríguez
Irene	Matemáticas	3.4	1.0	Díaz
Samuel	Tecnología	6.7	NaN	NaN

Concat viene mejor para añadir nuevos registros (filas)


```
In [87]: resultado = pd.concat([tabla1, tabla2], axis=0)    filas
```

	Materia	Nota	Curso	Tutor
Pedro	Matemáticas	8.2	NaN	NaN
Irene	Matemáticas	3.4	NaN	NaN
Samuel	Tecnología	6.7	NaN	NaN
Pedro	NaN	NaN	2.0	Rodríguez
Irene	NaN	NaN	1.0	Díaz

Figura 4. Ejemplo de concatenación de dos DataFrames en horizontal y en vertical

UNIVERSIDAD
COMPLUTENSE
DE MADRID



ntic
master
revolucionamos la comunicación

Autor: María José
Gómez Silva

Machine Learning con Python. Semana 1.

CAPÍTULO 2. INGESTA DE DATOS.

2. 5 Análisis y Visualización de los Datos

ANÁLISIS DE VARIABLES

El análisis o exploración de los datos permite entender mejor la información contenida en cada variable y detectar posibles errores antes de emplear los datos para realizar cálculos o entrenar un modelo.

Cuando tenemos datos contenidos en una estructura *DataFrame*, podemos considerar que cada columna es una variable de un tipo determinado. La estructura *DataFrame* ofrece varios métodos (comandos) para realizar el análisis de cada una de las variables que contiene.

El método *info* muestra información relativa a cada variable, o lo que es lo mismo, a cada columna, como el nombre de la columna, el tipo de dato, y el número de valores distintos de NaN que contiene. NaN son las siglas de *Not a Number*, y es la forma de tipificar los valores vacíos (celdas vacías de una tabla) o desconocidos y que no son computables. Por lo tanto, el método *info* nos permite comprobar que los datos de cada columna se hayan almacenado con el tipo correcto y así evitar situaciones como por ejemplo en la que los datos numéricos son reconocidos como texto o viceversa.

El método *info*, además, proporciona información general sobre la tabla como el número de filas (*entries*) y columnas y la cantidad de memoria empleada. El método *shape* también nos da información de las dimensiones del *DataFrame*, devuelve el número de filas y de columnas.

Por otro lado, el método *describe* muestra los primeros estadísticos de cada variable/columna, permitiendo así su análisis. El análisis estadístico sólo es realizado para las columnas que contengan datos de tipo numérico, y las métricas mostradas son el número de valores empleados en el análisis (*count*), la media (*mean*), la desviación típica (*std*), el mínimo (*min*) y máximo (*max*) y los cuartiles (el cuartil 50% es la

mediana). La Figura 1 muestra el resultado de emplear los métodos *info*, *shape* y *describe*.

```
In [5]: cosecha_finca1= pd.DataFrame([[300, 400], [100, 500], [400, 200], [600, 100]],
                                     columns=['trigo', 'maiz'), index=[2018, 2017, 2021, 2022])
cosecha_finca1

Out[5]:
      trigo  maiz
2018    300   400
2017    100   500
2021    400   200
2022    600   100

In [6]: cosecha_finca1.info()

<class 'pandas.core.frame.DataFrame'>
Int64Index: 4 entries, 2018 to 2022
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype  
---  -- 
 0   trigo    4 non-null      int64  
 1   maiz     4 non-null      int64  
dtypes: int64(2)
memory usage: 96.0 bytes

In [8]: cosecha_finca1.shape

Out[8]: (4, 2)

In [9]: cosecha_finca1.describe()

Out[9]:
      trigo        maiz
count    4.0000    4.000000
mean    350.0000  300.000000
std     208.1666 182.574186
min     100.0000 100.000000
25%    250.0000 175.000000
50%    350.0000 300.000000
75%    450.0000 425.000000
max    600.0000 500.000000
```

Figura 1. Métodos de Pandas para el análisis de cada variable de un DataFrame.

VISUALIZACIÓN

La librería Pandas junto con la librería Matplotlib proporcionan los métodos necesarios para generar los gráficos más comunes en estadística descriptiva, como los **diagramas de barras (bar)** y de **tartas (pie)**.

En el ejemplo de la Figura 2 se ha empleado la base de datos de pedidos del archivo “Superstore_Dataset.xlsx”. Se ha generado un nuevo *DataFrame* llamado *ventas* que contiene el número de artículos (columna *Quantity*) de cada producto vendido en los cinco primeros pedidos. Se ha añadido el tipo de producto (columna *Sub-Category*) como índice de la nueva tabla. Finalmente, se ha representado una única figura con dos gráficos (*ax1* y *ax2*), mediante el método *subplot*.

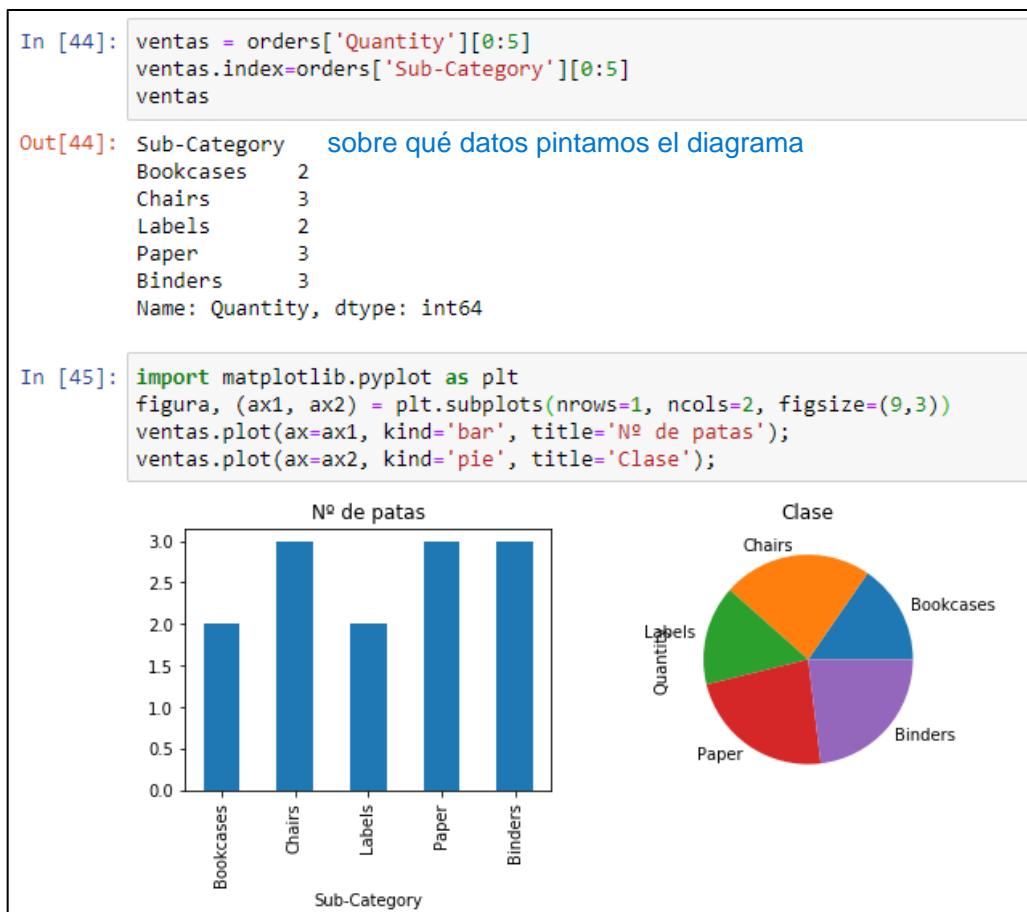


Figura 2. Diagrama de barras y de tartas a partir de los datos de un DataFrame

El método *plot* de los *DataFrames* permite representar múltiples tipos de diagramas, dando distintos valores al argumento *kind*. Entre los diagramas más empleados encontramos los de **barras (bar)** y **tarta (pie)**, mostrados en el ejemplo anterior y el de **líneas (line)**, que es el representado por defecto, el **histograma (hist)**, el **diagrama de caja (box)**, o el de **dispersión (scatter)**, entre otros. Puede encontrarse más documentación disponible en <https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.plot.html>.

ANÁLISIS MULTI-VARIABLE Y VISUALIZACIÓN

Además del análisis de cada una de las variables contenidas en un *DataFrame*, Pandas permite estudiar las relaciones entre ellas, o lo que es lo mismo, permite realizar un análisis multi-variable. Este tipo de análisis ayuda a obtener una visión inicial de qué variables son más o menos descriptivas y por lo tanto más o menos adecuadas para ser empleadas como datos de entrada de un modelo predictivo de Machine Learning.

Dos de los estadísticos más empleados para analizar la relación lineal entre dos variables son la covarianza y la correlación.

- La **covarianza** indica como los cambios en una variable provocan cambios en la otra. Su signo indica si ambas variables varían en el mismo sentido (covarianza positiva) o en sentido opuesto (covarianza negativa). Si la covarianza es cercana a cero o cero, no se puede definir si los cambios en una variable se relacionan con algún tipo de cambio en la otra.
- La **correlación** nos indica la magnitud de la relación o similitud entre las variables. Cuanto mayor sea su valor absoluto, mayor es la relación entre las variables.

```
In [104]: tabla = pd.DataFrame([('Enero', 31, 'inv', 2, 4),('Febrero', 28, 'inv', 3, 2),('Marzo', 31, 'inv', 7, 5),
                           ('Abril', 30, 'pri', 7, 9),('Mayo', 31, 'pri', 9, 10),('Junio', 30, 'pri', 15, 14),
                           ('Julio', 31, 'ver', 20, 24),('Agosto', 31, 'ver', 27, 26),('Septiembre', 30, 'ver', 25, 18),
                           ('Octubre', 31, 'oto', 20, 14),('Noviembre', 30, 'oto', 11, 10),('Diciembre', 31, 'oto', 6, 7),],
                           columns=['Mes', 'Días', 'Estación', 'Temp.2021', 'Temp.2022'],
                           index=['Ene', 'Feb', 'Mar', 'Abr', 'May', 'Jun', 'Jul', 'Ago', 'Sep', 'Oct', 'Nov', 'Dic'])

Out[104]:
      Mes   Días  Estación Temp.2021 Temp.2022
    Ene   Enero    31      inv        2        4
    Feb   Febrero   28      inv        3        2
    Mar   Marzo    31      inv        7        5
    Abr   Abril    30      pri        7        9
    May   Mayo     31      pri        9       10
    Jun   Junio    30      pri       15       14
    Jul   Julio     31      ver       20       24
    Ago   Agosto    31      ver       27       26
    Sep   Septiembre 30      ver       25       18
    Oct   Octubre   31      oto       20       14
    Nov   Noviembre 30      oto       11       10
    Dic   Diciembre 31      oto        6        7

In [108]: tabla.corr()

Out[108]:
          Días  Temp.2021  Temp.2022
    Días  1.000000  0.256108  0.349989
    Temp.2021  0.256108  1.000000  0.928141
    Temp.2022  0.349989  0.928141  1.000000
```

Figura 3. Cálculo de la matriz de correlación de las variables de un DataFrame

El cálculo de la covarianza y la correlación de todas las variables con respecto a cada una de ellas permite obtener la matriz de covarianza y la matriz de correlación,

respectivamente. La matriz de covarianza para todas las variables numéricas de un *DataFrame*, se obtiene con el método *cov*, y la matriz de correlación, con el método *corr* (véase un ejemplo en la Figura 3).

La librería de Python *Seaborn* permite representar la matriz de correlación mediante un mapa de color, con el método *heatmap*, tal y como se muestra en la Figura 4. Además, una vez identificados los pares de variables correlacionadas, es posible representar su gráfico de dispersión (*scatter*) gracias a la librería *Matplotlib*. El gráfico de dispersión de dos variables es la representación de una nube de puntos cuyas coordenadas son dichas variables.

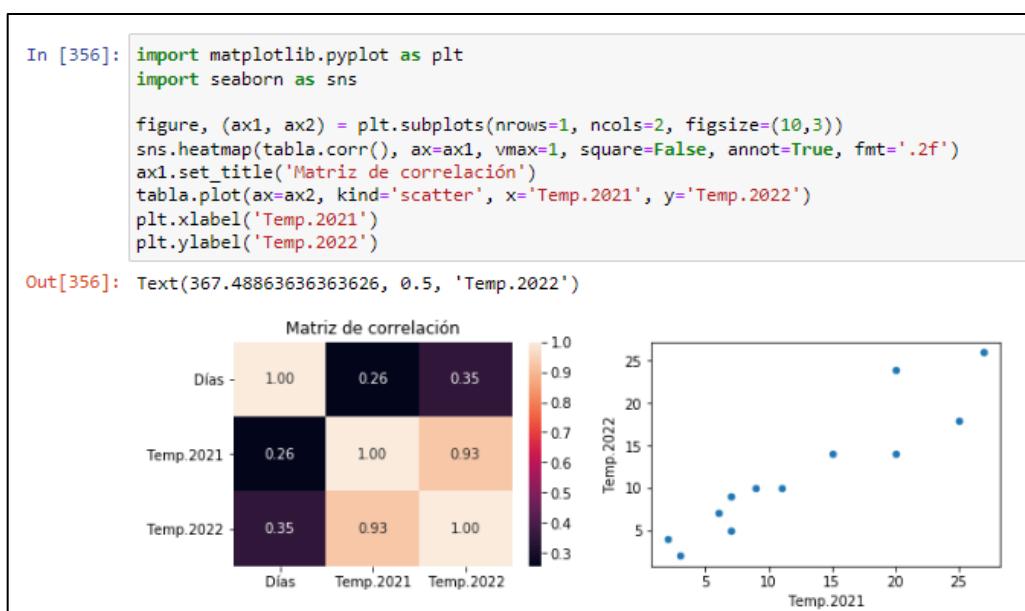


Figura 4. Representación de la matriz de correlación con un mapa de color y el gráfico de dispersión de dos variables correlacionadas.

ANÁLISIS DE LA VARIABLE A PREDECIR

El objetivo de entrenar un modelo de Machine Learning es que dicho modelo finalmente sea capaz de predecir el valor de una determinada variable. A esta variable se le llama *salida, respuesta o predicción* del modelo o también, variable *target*, y en las ecuaciones es simbolizada con la letra *y*.

Antes de entrenar el modelo es conveniente analizar la distribución de la variable de salida y su relación con el resto de las variables de entrada. Este análisis arrojará información útil que permitirá tomar decisiones en cuanto a la elección del modelo y los métodos empleados para preparar su entrenamiento.

En la figura 5 se muestra la cabecera de un DataFrame que contiene información sobre características de un listado de viviendas y además se incluye su precio de venta. Estos datos se han cargado desde un archivo csv, ‘HouseDataset.csv’.

```
In [55]: viviendas = pd.read_csv('HouseDataset.csv', sep=',')
# Renombramos las columnas
viviendas.columns = ["precio", "metros_totales", "antiguedad", "precio_terreno", "metros_habitable", "universitarios", "dormitorios", "chimenea", "banyos", "habitaciones", "calefaccion", "consumo_calefacion", "desague", "vistas_lago", "nueva_construccion", "aire_acondicionado"]
datos.head(3)

Out[55]:
   precio  metros_totales  antiguedad  precio_terreno  metros_habitable  universitarios  dormitorios  chimenea  banyos  habitaciones
0    132500           0.09          42        50000            906             35            2           1         1.0            5
1    181115           0.92          0         22300            1953            51            3           0         2.5            6
2    109000           0.19         133         7300            1944            51            4           1         1.0            8
```

Figura 5. Creación de un DataFrame con características de viviendas a partir de un archivo csv.

displot: busca la distribución de probabilidad
le decimos que no represente el histograma
rug es la cosa que hay abajo

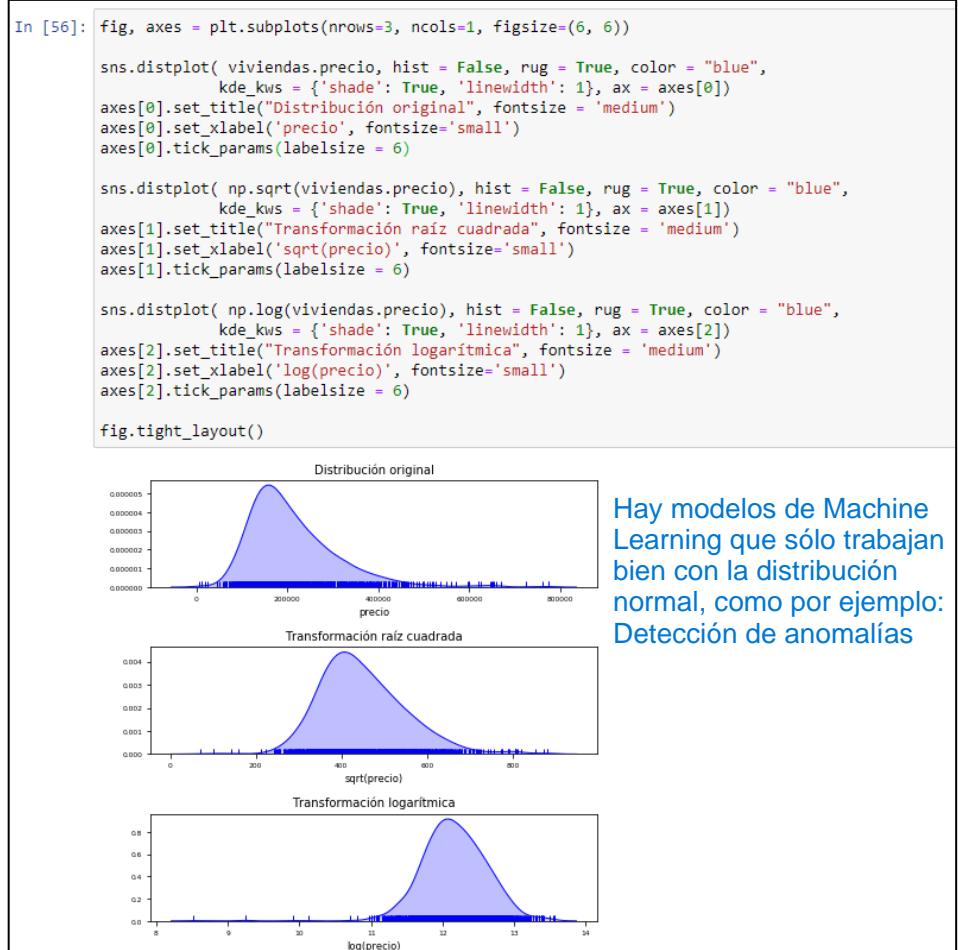


Figura 6. Representación de la distribución de los datos de precio de las viviendas.

Supongamos que queremos entrenar un modelo que sea capaz de predecir el precio de venta de una vivienda en función de sus características y vamos a usar los datos del csv anterior para entrenar el modelo. La variable de salida será el precio de las viviendas y como primer paso analizaremos la distribución de sus valores. La figura 6 muestra un ejemplo en el que se representa la distribución de los valores de la columna “precio”, y además se muestran dos distribuciones más, las resultantes de calcular la raíz cuadrada de los datos, y el logaritmo. Este análisis es interesante porque existen algunos métodos de Machine Learning que sólo funcionan, o funcionan mejor, para variables de salida con distribución Gaussiana o normal. En ocasiones puede ocurrir que la distribución de los datos no sea de tipo Gaussiana, pero sí la de la raíz o la del logaritmo de los datos.

Además del análisis visual que ofrecen las gráficas anteriores, la librería `fitter` de Python permite identificar la distribución que mejor se ajusta a nuestros datos. La Figura 7 muestra los resultados obtenidos para el ajuste de los datos de la columna ‘precio’ con nueve tipos de distribuciones estadísticas distintas. Los resultados se muestran ordenados en función del error total de cada ajuste. Por lo tanto, la primera distribución es la que mejor se ajusta a los datos.

fitter: ajusta los datos a las distintas distribuciones

```
In [58]: from fitter import Fitter, get_common_distributions
distribuciones = ['cauchy', 'chi2', 'expon', 'gamma', 'norm', 'beta', 'logistic']

fitter = Fitter(viviendas.precio, distributions=distribuciones)
fitter.fit()
fitter.summary(Nbest=10, plot=False)

Fitting 7 distributions: 100%|██████████| 7/7 [00:00<00:00, 12.81it/s]
```

	sumsquare_error	aic	bic	kl_div	ks_statistic	ks_pvalue
beta	2.497420e-11	3068.852573	-55037.908642	inf	0.051211	2.229230e-04
logistic	4.913880e-11	3147.967019	-53883.297831	inf	0.071310	4.368612e-08
cauchy	5.221450e-11	2956.669693	-53778.388707	inf	0.121728	9.001774e-23
chi2	5.776892e-11	3321.818880	-53596.249282	inf	0.093520	1.326966e-13
norm	6.947514e-11	3324.534158	-53284.856663	inf	0.104149	8.972690e-17
expon	2.915346e-10	2824.103160	-50806.577128	inf	0.316530	2.028906e-154
gamma	4.841645e-10	inf	-49922.566370	3.958212	0.947917	0.000000e+00

Figura 7. Ajuste de los datos de precio de las viviendas con varios tipos de distribuciones.

Además, como se ha mencionado anteriormente, una práctica muy útil es la exploración de la relación entre la variable de salida y el resto de las variables del *Dataframe*. Esto permite extraer la información necesaria para realizar la selección de las características que serán empleadas como datos de entrada del modelo que queremos entrenar.

La librería `Seaborn` con la función `regplot` ofrece la representación del diagrama de dispersión entre dos variables, acompañado por la visualización del ajuste de una regresión lineal entre esas dos variables. La Figura 8 muestra el diagrama de dispersión y la regresión lineal entre cada variable numérica y la variable de salida *precio*.

```
In [63]: import matplotlib.ticker as ticker
representa 3 columnas y 3 filas
fig, axes = plt.subplots(nrows=3, ncols=3, figsize=(9, 5))
axes = axes.flat
columnas_numeric = viviendas.select_dtypes(include=['float64', 'int64']).columns
columnas_numeric = columnas_numeric.drop('precio')

for i, colum in enumerate(columnas_numeric):
    sns.replot(x = viviendas[colum], y = viviendas['precio'], color = "green", marker = '.', 
                scatter_kws = {"alpha":0.4}, line_kws = {"color":"r", "alpha":0.7}, ax = axes[i])
    axes[i].set_title(f"precio vs {colum}", fontsize = 7, fontweight = "bold")
    axes[i].yaxis.set_major_formatter(ticker.EngFormatter())
    axes[i].xaxis.set_major_formatter(ticker.EngFormatter())
    axes[i].tick_params(labelsize = 6)
    axes[i].set_xlabel("")
    axes[i].set_ylabel("")

fig.tight_layout()
plt.subplots_adjust(top=0.9)
fig.suptitle('Correlación con precio', fontsize = 10, fontweight = "bold");
```

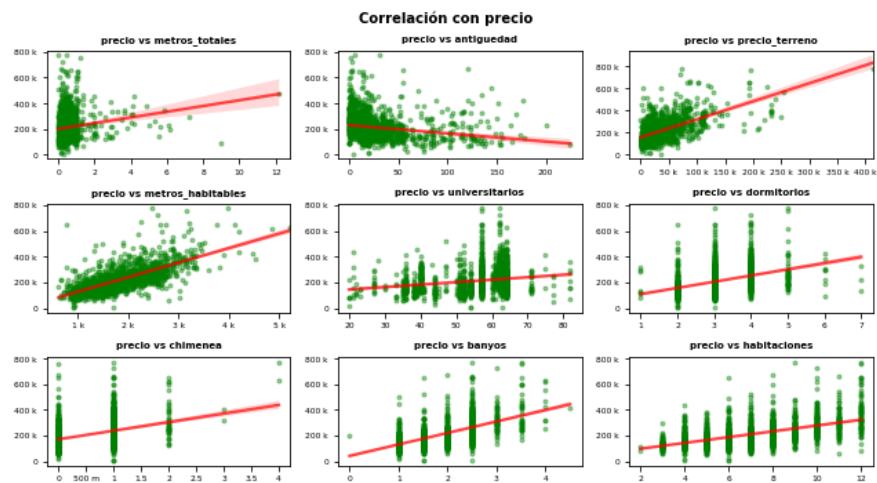


Figura 8. Relación entre la variable de salida y el resto de las variables numéricas.

Además de la exploración visual realizada sobre las gráficas anteriores, es posible realizar un análisis cuantitativo de la relación entre la variable de salida y el resto de las variables numéricas gracias al cálculo y la representación de la matriz de correlación, como muestra la Figura 9.

Las variables poco correlacionadas con la variable de salida pueden no funcionar bien como descriptores de la situación y por lo tanto colaborar poco en la tarea de predicción del modelo. Sin embargo, las variables altamente correlacionadas con la de salida en ocasiones también pueden perjudicar el entrenamiento ya que hacen que el algoritmo de aprendizaje aparte el foco de la información que ofrecen el resto de las variables, perdiendo así capacidad descriptiva.



Figura 9. Representación de la matriz de correlación entre las columnas de un DataFrame

ANÁLISIS DE VARIABLES CATEGÓRICAS

Consideramos como variables categóricas aquellas que, en lugar de tomar un valor numérico, toman como valor una etiqueta o clase entre varias opciones disponibles. Las columnas de este tipo normalmente son designadas en *Pandas* como de tipo *objeto*.

Es posible emplear el método `describe` sobre las columnas de este tipo, como muestra la Figura 10. La información que se obtiene es el número de registros en cada columna (`count`), el número de valores distintos que toma (`unique`), el valor que más se repite (`top`) y el número de veces que se repite (`freq`).

	calefaccion	consumo_calefacion	desague	vistas_lago	nueva_construcion	aire_acondicionado
count	1728	1728	1728	1728	1728	1728
unique	3	3	3	2	2	2
top	hot air	gas	public/commercial	No	No	No
freq	1121	1197	1213	1713	1647	1093

Figura 10. Método `describe` aplicado sobre columnas de tipo 'object'.

Además, también es posible visualizar la distribución de este tipo de variables mediante la representación del número de apariciones de cada valor posible (realizado con el método `counts()`), utilizando diagramas de barras (con el método `plot.barh`), como se muestra en el ejemplo de la Figura 11.

```
In [71]: fig, axes = plt.subplots(nrows=2, ncols=3, figsize=(9, 5))
axes = axes.flat
columnas_object = viviendas.select_dtypes(include=['object']).columns
for i, colum in enumerate(columnas_object):
    datos[colum].value_counts().plot.barh(ax = axes[i])
    axes[i].set_title(colum, fontsize = 8, fontweight = "bold")
    axes[i].tick_params(labelsize = 8)
    axes[i].set_xlabel("")

fig.tight_layout()
plt.subplots_adjust(top=0.9)
```

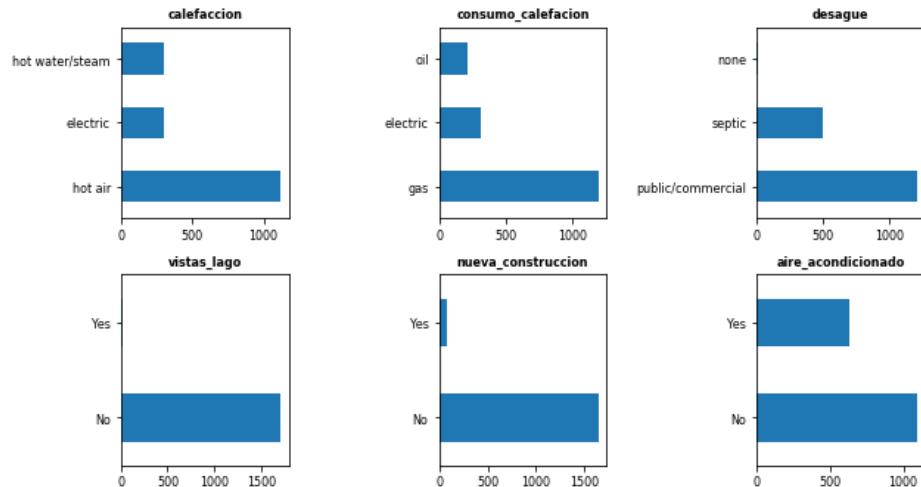


Figura 11. Representación de la distribución de valores en variables categóricas

También es posible analizar la relación de la variable de salida con las variables categóricas. Para ello, se emplea un gráfico de tipo violín que relacione cada variable categórica con la de salida, como se muestra en el ejemplo de la Figura 12, donde la variable de salida es `precio`. Este tipo de gráfico muestra la distribución de una variable numérica para cada una de las clases de una variable categórica.

```
In [84]: fig, axes = plt.subplots(nrows=2, ncols=3, figsize=(9, 5))
axes = axes.flat
columnas_object = viviendas.select_dtypes(include=['object']).columns

for i, colum in enumerate(columnas_object):
    sns.violinplot( x = colum, y = 'precio', data = datos, color = "white", ax = axes[i])
    axes[i].set_title(f"precio vs {colum}", fontsize = 8, fontweight = "bold")
    axes[i].yaxis.set_major_formatter(ticker.EngFormatter())
    axes[i].tick_params(labelsize = 8)
    axes[i].set_xlabel("")
    axes[i].set_ylabel("")

fig.tight_layout()
```

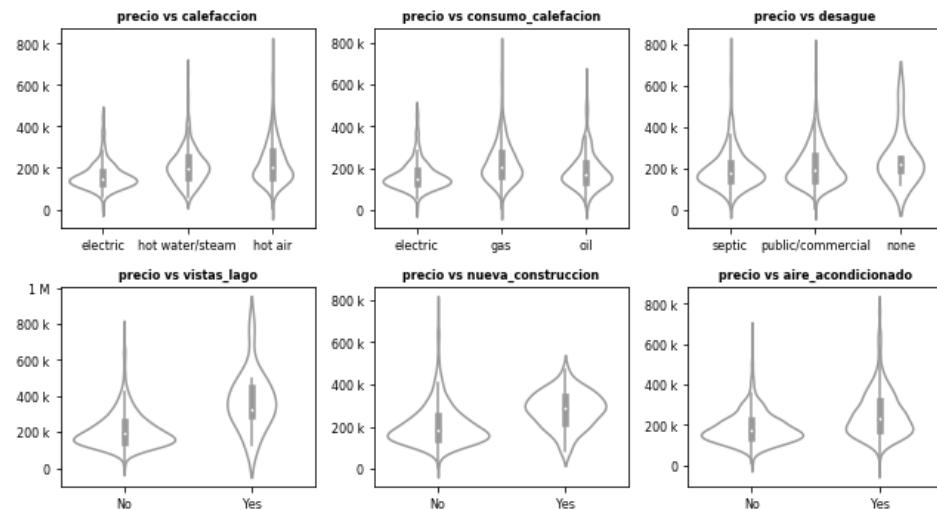


Figura 12. Relación entre cada variable categórica de un DataFrame y la variable numérica de salida.

UNIVERSIDAD
COMPLUTENSE
DE MADRID



ntic
master
revolucionamos la comunicación

Autor: María José
Gómez Silva

Machine Learning con Python. Semana 1.

CAPÍTULO 3. PREPROCESADO.

Antes de usar los datos para el entrenamiento y la evaluación de un modelo de Machine Learning, es necesario *preparar y preprocesar* esos datos. Entre las tareas más comunes de la etapa de preprocesado se encuentran las de localizar y corregir errores o ruido, eliminar valores atípicos y aplicar las transformaciones necesarias para la normalización de los valores presentados por los datos.

En este capítulo se describirán todas las técnicas mencionadas para el preprocesamiento de los datos, además de otras para el aumento de su cantidad y variedad, y la corrección del sesgo cuando sea necesario.

Tras la preparación de los datos, es necesaria la selección de aquellos más influyentes en la predicción del modelo o incluso la generación de nuevos datos o características a partir de los existentes, proceso conocido como “*feature engineering*”.

3. 1 Ruido

Ruido es el término general que se emplea para referirse a las modificaciones indeseadas y, en general, desconocidas que sufre una señal o colección de datos durante su captura, su almacenaje o su tratamiento. En ocasiones, también nos referimos con el término ruido a la información no útil que acompaña a nuestros datos. Los mecanismos empleados para la eliminación o reducción del ruido en los datos se suelen denominar, de forma genérica, *filtros*.

La presencia de ruido en nuestros datos puede deberse a múltiples causas y factores, como errores en las mediciones (por error humano o saturación de los sensores), errores en el almacenaje, limitaciones en la representación numérica de los computadores, o interferencias en la captura de los datos, entre otros.

Un ejemplo muy visual, es el del ruido presente en las imágenes. La Figura 66 muestra a la izquierda una imagen con ruido *gaussiano*. El ruido *gaussiano* suele deberse a la acumulación de errores e interferencias en el proceso de captura, trasmisión y almacenamiento de los datos y se presenta como pequeñas desviaciones de carácter

aleatorio sobre el valor ideal de los datos. La Figura 1 muestra una segunda imagen, a la derecha, con ruido de *sal y pimienta*. Este ruido suele estar ocasionado por la saturación de los sensores y aparece en las imágenes con la presencia de píxeles totalmente blancos y otros totalmente negros.

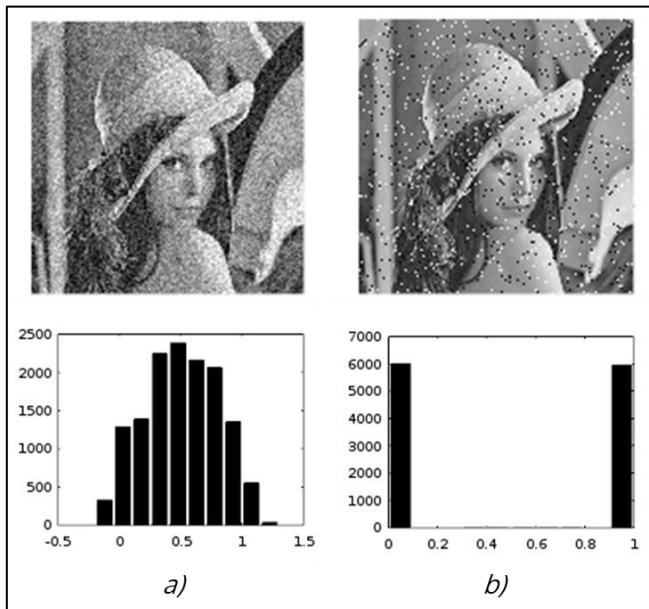


Figura 1. Imágenes con ruido a) gaussiano y b) de sal y pimienta e histogramas de los valores del ruido añadido a los valores de los píxeles de la imagen ideal.

Antes de emplear los datos recolectados en un algoritmo de Machine Learning es necesario su filtrado para que la presencia de ruido no dañe el rendimiento de los procesos posteriores.

En la práctica, tendremos que explorar los datos, e identificar el ruido o presencia de valores molestos a eliminar. Además, dependiendo del tipo de ruido, deberemos seleccionar el método de filtrado apropiado. Por ejemplo, el ruido *gaussiano* de la imagen anterior se puede suavizar mediante un filtro de media, que consiste en sustituir el valor de cada píxel por la media de los valores de los píxeles vecinos. Sin embargo, ese filtro tiene un resultado desastroso si se aplica a una imagen con ruido de sal y pimienta. En ese caso, en lugar del filtro de la media, hay que aplicar el filtro de la mediana.

En general, cuando tenemos una colección de datos en un *DataFrame*, deberemos aplicar diferentes procesos para eliminar cada tipo de ruido presente. Por ejemplo, pueden aparecer valores no medidos o desconocidos, que identificaremos como valores *NaN*. También, puede aparecer ruido como pequeñas fluctuaciones con respecto a la tendencia esperada de los datos, que suavizaremos con filtros paso bajo, como el de la media. O incluso, pueden aparecer valores atípicos, que identificaremos y descartaremos con técnicas de análisis estadístico. Todos estos procesos son descritos en las siguientes secciones.

UNIVERSIDAD
COMPLUTENSE
DE MADRID



ntic
master
revolucionamos la comunicación

Autor: María José
Gómez Silva

Machine Learning con Python. Semana 1.

CAPÍTULO 3. PREPROCESADO.

3. 2 Proceso de Filtrado

FILTRADO DE LOS DATOS EN PANDAS

Filtrado de valores NaN

Los valores *NaN* pueden ocasionar problemas a la hora de realizar ciertos análisis, pero en muchas situaciones es inevitable o incluso necesario su aparición en las tablas.

Los valores *NaN* pueden ayudar a estandarizar la nomenclatura para los valores no definidos en una base de datos. En ocasiones, podemos encontrar celdas simplemente vacías, o en las que se indica que el valor no está definido mediante un guion, “-”, o de forma explícita con palabras, “*no calculado*”. Cuando se leen los datos de una hoja de cálculo, el argumento *na_values* permite sustituir algunos valores por *NaN*. Las celdas vacías se transforman automáticamente a *NaN*, y el resto de los valores que queramos convertir deben ser indicados. La Figura 1 muestra la estandarización a *NaN* de todos los datos no válidos de una hoja de cálculo que recoge los kilogramos cosechados de trigo y de maíz en distintos años.

```
In [47]: cosecha=pd.read_excel('./cosecha.xlsx')
cosecha
out[47]:
      trigo  maiz
0       300   NaN
1       100   -
2       400  200
3  no calculado  100

In [48]: cosecha=pd.read_excel('./cosecha.xlsx', na_values={'trigo': ['no calculado'], 'maiz':['-']})
cosecha
out[48]:
      trigo  maiz
0    300.0   NaN
1    100.0   NaN
2    400.0  200.0
3     NaN  100.0
```

Figura 1. Estandarización de distintos valores a *NaN*.

Para tener en cuenta la presencia de valores NaN (`np.nan` en numpy o `None` en Python en general) entre los datos, existen métodos como `isna`, `isnull` y `notnull` que permiten identificarlos. El resultado de estos métodos es una nueva tabla de tipo bool. Tanto `isna` como `isnull` indican como True la posición de los valores de tipo NaN de la tabla analizada. Ambos métodos ofrecen el mismo resultado. Por su parte, `notnull` indica como True los datos que no son de tipo NaN, como muestra la Figura 2. En esta Figura, se muestra además un ejemplo en el que se han contabilizado todas las apariciones de valores NaN de cada columna (combinando `isna` y `sum`) y se han ordenado las columnas en función del número de elementos de tipo NaN que contienen.

El método `dropna` permite eliminar las filas y columnas que contienen datos de tipo NaN en un *DataFrame*, como muestra la Figura 3.

```
In [12]: cosecha.isnull()
Out[12]:
      trigo  maiz
0   False  True
1   False  True
2   False False
3   True False

In [13]: cosecha.notnull()
Out[13]:
      trigo  maiz
0   True False
1   True False
2   True  True
3  False  True

In [15]: cosecha.isna().sum().sort_values()
Out[15]: trigo    1
          maiz    2
          dtype: int64
```

Figura 2. Identificación de datos NaN en un DataFrame

```
In [184]: cosecha.dropna()
Out[184]:
      trigo  maiz
2  400.0 200.0
```

Figura 3. Eliminación de valores NaN en un DataFrame

Gestión de valores duplicados

Es habitual que ciertos valores se repitan en las bases de datos. Con el método `unique` podemos identificar todos los valores diferentes que toma una determinada variable de una *Serie* o un *DataFrame*. Además, con el método `value_counts`, obtenemos las repeticiones de cada valor. En la Figura 4, se usa como ejemplo una serie que contiene los días de cada mes y se obtiene los posibles valores (28, 30 o 31 días) y cuántos meses tienen cada uno de los posibles valores.

```
In [49]: dias = pd.Series([31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31],  
                      index=['Ene', 'Feb', 'Mar', 'Abr', 'May', 'Jun', 'Jul', 'Ago', 'Sep', 'Oct', 'Nov', 'Dic'],  
                      name='Días de cada mes')  
dias.unique()  
  
Out[49]: array([31, 28, 30], dtype=int64)  
  
In [51]: dias.value_counts()  
  
Out[51]: 31    7  
30    4  
28    1  
Name: Días de cada mes, dtype: int64
```

Figura 4. Empleo de los métodos `unique` y `value_counts` sobre los datos de una Serie

```
In [3]: tabla = pd.DataFrame([('Enero', 31, 'inv', 2, 4),('Febrero', 28, 'inv', 3, 2),('Marzo', 31, 'inv', 7, 5),  
                           ('Abril', 30, 'pri', 7, 9),('Mayo', 31, 'pri', 9, 10),('Junio', 30, 'pri', 15, 14),  
                           ('Julio', 31, 'ver', 20, 24),('Agosto', 31, 'ver', 27, 26),('Septiembre', 30, 'ver', 25, 18),  
                           ('Octubre', 31, 'oto', 20, 14),('Noviembre', 30, 'oto', 11, 10),('Diciembre', 31, 'oto', 6, 7)],  
                           columns=['Mes', 'Días', 'Estación', 'Temp.2021', 'Temp.2022'],  
                           index=['Ene', 'Feb', 'Mar', 'Abr', 'May', 'Jun', 'Jul', 'Ago', 'Sep', 'Oct', 'Nov', 'Dic'])  
tabla  
  
out[3]:  
      Mes   Días Estación Temp.2021 Temp.2022  
    Ene   Enero    31     inv        2        4  
    Feb   Febrero   28     inv        3        2  
    Mar   Marzo    31     inv        7        5  
    Abr   Abril    30     pri        7        9  
    May   Mayo     31     pri        9       10  
    Jun   Junio    30     pri       15       14  
    Jul   Julio     31     ver       20       24  
    Ago   Agosto    31     ver       27       26  
    Sep   Septiembre 30     ver       25       18  
    Oct   Octubre   31     oto       20       14  
    Nov   Noviembre 30     oto       11       10  
    Dic   Diciembre 31     oto        6        7  
  
In [5]: tabla.duplicated(subset=['Días', 'Temp.2021'])  
out[5]: Ene  False  Sólo me aparece la segunda duplicación  
Feb  False  
Mar  False  
Abr  False  
May  False  
Jun  False  
Jul  False  
Ago  False  
Sep  False  
Oct  True  
Nov  False  
Dic  False  
dtype: bool
```

Figura 5. Detección de filas con valores duplicados en un DataFrame

Con el método `duplicated` es posible identificar filas repetidas en un *DataFrame*. Este método devuelve una Serie de tipo *bool* con tantos elementos como filas tiene el *DataFrame* en el que se ha aplicado. El valor *True* indica que en el *DataFrame* ya existe una fila anterior con los mismos valores. Por lo tanto, la primera aparición de una fila repetida no se considera como duplicada. Además, los *DataFrames* también ofrecen el método `drop_duplicates` para eliminar las filas repetidas.

Estos métodos, por defecto, indican y eliminan, respectivamente, las filas en las que los valores de todas las columnas son idénticos, pero también se les puede indicar que sólo consideren la coincidencia en un subconjunto de columnas, como muestra la Figura 5 y 6. La fila correspondiente al mes de octubre es identificada como repetida, ya que su número de días y su temperatura media en 2021 coindicen con los de julio, que aparece antes.

```
In [7]: tabla2 = tabla.drop_duplicates(subset=['Días', 'Temp.2021'])  
tabla2
```

```
Out[7]:
```

	Mes	Días	Estación	Temp.2021	Temp.2022
Ene	Enero	31	inv	2	4
Feb	Febrero	28	inv	3	2
Mar	Marzo	31	inv	7	5
Abr	Abril	30	pri	7	9
May	Mayo	31	pri	9	10
Jun	Junio	30	pri	15	14
Jul	Julio	31	ver	20	24
Ago	Agosto	31	ver	27	26
Sep	Septiembre	30	ver	25	18
Nov	Noviembre	30	oto	11	10
Dic	Diciembre	31	oto	6	7

Figura 6. Eliminación de filas con valores duplicados en un *DataFrame*.

Filtrado selectivo

La librería Pandas permite la selección de elementos usando *Series* de tipo *bool* (*True* o *False*) que actúan como filtros. Estos filtros se pueden aplicar sobre otra *Serie* o sobre un *DataFrame*, como se explicó en el capítulo anterior, en la sección de “Manejo de los Datos en Pandas”.

MEDIA VS. DIFERENCIA

En general, cuando tenemos una secuencia de datos correspondiente a los valores de una determinada variable o característica medida, y especialmente cuando se trata de una secuencia temporal, esos valores suelen ser el resultado de sumar una tendencia o comportamiento general con pequeñas oscilaciones o desviaciones.

Por ejemplo, si representamos la temperatura media de cada mes a lo largo de dos años, se apreciará la tendencia general de crecimiento de la temperatura en verano y decrecimiento en invierno. Sin embargo, esa curva no siempre es suave, ya que pueden darse pequeñas fluctuaciones en la temperatura debidas a múltiples causas, que además suelen ser muy variables o difíciles de predecir, como cambios en las presiones, tormentas puntuales, etc.

En una señal o colección de datos, podemos diferenciar componentes de baja y alta frecuencia, cuya suma dan lugar a los valores medidos. Cuando hablamos de las componentes de baja frecuencia, nos estamos refiriendo a la tendencia general subyacente de los valores. Por el contrario, cuando hablamos de las componentes de alta frecuencia, nos estamos refiriendo a las pequeñas oscilaciones o fluctuaciones que sufren los valores de la señal, que se tratan de cambios rápidos (frecuentes, de ahí el apelativo de alta frecuencia).

Consecuentemente, los filtros paso bajo son los que dejan pasar las componentes de baja frecuencia, y los de paso alto, los que dejan pasar las componentes de alta frecuencia.

A continuación, se presenta un ejemplo de aplicación de este tipo de filtros. En primer lugar, generamos una *Serie* con datos de las temperaturas medias de cada mes durante dos años a partir de un *DataFrame* previamente generado. Los valores de la serie son convertidos de tipo *int* a *float*, como muestra la Figura 7.

Seguidamente, se representa la serie generada, con un diagrama de líneas, como muestra la Figura 8.

A continuación, programamos un filtro de media, que consiste en sustituir cada valor por el promedio con sus vecinos. El filtro de media es un ejemplo de filtro pasa baja, ya que el resultado es el suavizado de las oscilaciones puntuales y la extracción de la tendencia general de los datos. También se programa un filtro de diferencias, que consiste en sustituir cada valor por la diferencia de él mismo con el anterior. El filtro de diferencias es un ejemplo de filtro paso alto, ya que calcula las diferencias o cambios entre valores consecutivos, es decir, extrae las oscilaciones de los datos. Posteriormente, ambos filtros son aplicados sobre la *Serie* original y el resultado de ambos es representado, como muestra la Figura 9.

```
In [149]: tabla = pd.DataFrame([('Enero', 31, 'inv', 2, 4), ('Febrero', 28, 'inv', 3, 2), ('Marzo', 31, 'inv', 7, 5),
                           ('Abril', 30, 'pri', 7, 9), ('Mayo', 31, 'pri', 9, 10), ('Junio', 30, 'pri', 15, 14),
                           ('Julio', 31, 'ver', 20, 24), ('Agosto', 31, 'ver', 27, 26), ('Septiembre', 30, 'ver', 25, 18),
                           ('Octubre', 31, 'oto', 20, 14), ('Noviembre', 30, 'oto', 11, 10), ('Diciembre', 31, 'oto', 6, 7)], 
                           columns=['Mes', 'Días', 'Estación', 'Temp.2021', 'Temp.2022'],
                           index=['Ene', 'Feb', 'Mar', 'Abr', 'May', 'Jun', 'Jul', 'Ago', 'Sep', 'Oct', 'Nov', 'Dic'])
tabla
```

```
Out[149]:
      Mes   Días Estación Temp.2021 Temp.2022
    Ene   Enero   31      inv        2        4
    Feb   Febrero  28      inv        3        2
    Mar   Marzo   31      inv        7        5
    Abr   Abril   30      pri        7        9
    May   Mayo   31      pri        9       10
    Jun   Junio   30      pri       15       14
    Jul   Julio   31      ver       20       24
    Ago   Agosto  31      ver       27       26
    Sep   Septiembre  30      ver       25       18
    Oct   Octubre  31      oto       20       14
    Nov   Noviembre 30      oto       11       10
    Dic   Diciembre 31      oto        6        7
```

```
In [155]: temp = pd.concat([tabla['Temp.2021'], tabla['Temp.2022']], axis = 0)
temp = temp.apply(lambda val: float(val))
temp
```

```
Out[155]:
Ene    2.0
Feb    3.0
Mar    7.0
Abr    7.0
May    9.0
Jun   15.0
Jul   20.0
Ago   27.0
Sep   25.0
Oct   20.0
Nov   11.0
Dic   6.0
Ene   4.0
Feb   2.0
Mar   5.0
Abr   9.0
May   10.0
Jun   14.0
Jul   24.0
Ago   26.0
Sep   18.0
Oct   14.0
Nov   10.0
Dic   7.0
dtype: float64
```

Figura 7. Generación de una Serie con datos de las temperaturas medias de cada mes durante dos años a partir de un DataFrame previamente generado

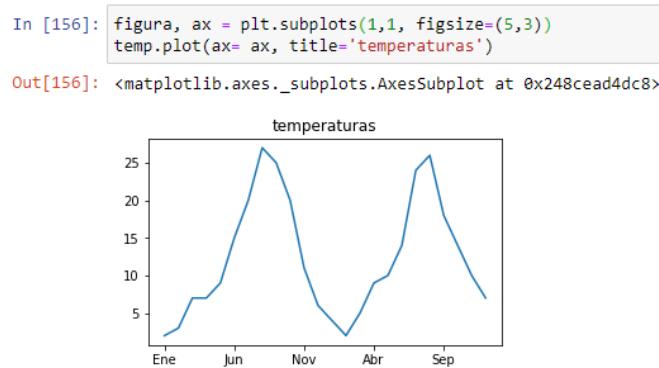


Figura 8. Representación de los datos de las temperaturas medias de cada mes en dos años

```

In [164]: def filtro_media(serie):
    valores=serie.values
    new_valores = valores.copy()
    for i in range(len(valores)):
        if i<3 or i>len(valores)-4:
            new_valores[i]=np.nan
        else:
            new_valores[i]=(valores[i-3]+valores[i-2]+valores[i-1]+valores[i]+valores[i+1]+valores[i+2]+valores[i+3])/7
    new_serie=pd.Series(new_valores, index=serie.index)
    return new_serie

def filtro_diferencia(serie):
    valores=serie.values
    new_valores = valores.copy()
    for i in range(len(valores)):
        if i<1:
            new_valores[i]=np.nan
        else:
            new_valores[i]=valores[i]-valores[i-1]
    new_serie=pd.Series(new_valores, index=serie.index)
    return new_serie

In [165]: tendencia = filtro_media(temp)
cambios = filtro_diferencia(temp)

In [166]: figura, (ax1, ax2) = plt.subplots(nrows=1, ncols=2, figsize=(8,3))
tendencia.plot(ax = ax1, title='tendencia de temperaturas')
cambios.plot(ax = ax2, title='cambios de temperaturas')

Out[166]: <matplotlib.axes._subplots.AxesSubplot at 0x248ceda5688>

```

Figura 9. Definición y aplicación de un filtro de media y uno de diferencias y representación de los resultados.

VALORES ATÍPICOS

Los valores atípicos (outliers, en inglés) son datos muy diferentes numéricamente al resto de valores de una colección.

Las causas de la presencia de estos valores son múltiples y variadas. Pueden ser **ocasionados por errores en la medición o captura de los datos**. En esos casos, **esos valores atípicos pueden causar problemas en el aprendizaje automático de una tarea**, ya que no son valores representativos del comportamiento del proceso a modelar, y por ello es importante eliminarlos en la etapa de preprocesado.

En otros casos, los valores atípicos no se deben a errores en la captura de los datos, sino que se tratan de **datos válidos pero que reflejan un comportamiento inusual de alguna de las variables que estamos midiendo**. En esos casos, en los que por extraños que parezcan, **los valores atípicos son parte del proceso, deben analizarse y emplearse como pista para la detección de anomalías**.

En cualquier caso, sea cual sea la causa de los valores atípicos, es muy importante su identificación. Para tal fin, se recurre a métricas estadísticas.

Dada la subjetividad en cuanto a qué se considera como valor atípico, existen una gran variedad de métodos para identificarlos. El método más impartido académicamente por su sencillez y efectividad es el **test de Tukey**. Este test toma como referencia la diferencia entre el primer y el tercer cuartil, también conocida como **rango intercuartílico**, $IQR=(Q3-Q1)$.

- Se considera un **valor atípico leve** el que se encuentra a más de 1,5 veces el **rango intercuartílico** del primer o del tercer cuartil.

$$q < Q1 - 1.5 \cdot IQR \text{ o } q > Q3 + 1.5 \cdot IQR$$

- Se considera un **valor atípico extremo** el que se encuentra a más de 3 veces el **rango intercuartílico** del primer o del tercer cuartil.

$$q < Q1 - 3 \cdot IQR \text{ o } q > Q3 + 3 \cdot IQR$$

Los diagramas de cajas (método `plot.box` para *Series* y `boxplot` para *DataFrames*), son muy útiles para analizar la dispersión de un conjunto de datos. La figura 10 representa el diagrama de caja de una *Serie* formada por los valores de la columna de “*Quantity*” (cantidad de productos pedidos) de la base de datos “*Superstore_Dataset.xlsx*”. La mediana es representada con una línea verde y el rango intercuartílico con una caja. Las líneas que salen de la caja, habitualmente llamadas *bigotes*, representan la distancia de 1.5 veces el rango intercuartílico desde el primer y el tercer cuartil. Los valores atípicos, aquellos que salen fuera del rango marcado por los bigotes, son representados con círculos.

```
In [11]: orders = pd.read_excel('Superstore_Dataset.xlsx', sheet_name = 'Orders')
cantidad = orders.Quantity
figura, ax = plt.subplots(1,1,figsize=(5,5))
ax=cantidad.plot.box()
```

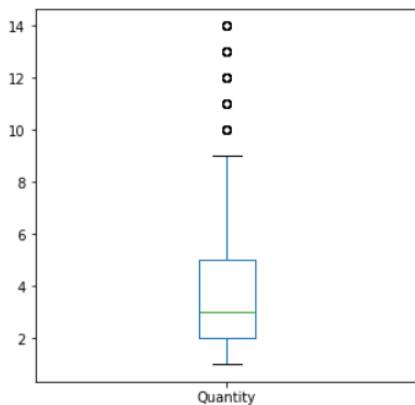


Figura 10. Diagrama de caja de los datos de una Serie

IMPUTACIÓN DE VALORES AUSENTES CON SCIKIT-LEARN

En la sección de *Filtrado de los datos en Pandas* se explica como eliminar las filas o columnas en las que hay datos ausentes (NaN). Este paso en muchas ocasiones es necesario, ya que la mayoría de los algoritmos de Machine Learning no aceptan muestras incompletas (registros de datos con valores NaN).

Sin embargo la eliminación de cada muestra en la que alguno de los campos esté incompleto supone la pérdida de mucha información y sólo es aceptable cuando se dispone de muchos datos o el porcentaje de valores NaN es muy bajo.

La alternativa a la eliminación es la imputación, es decir, la estimación de los valores ausentes empleando el resto de los datos disponibles. Con esta técnica se corre el riesgo de introducir valores que no sean completamente fieles a la realidad del proceso medido. Es importante considerar el riesgo de introducir estos valores dependiendo de la variable en la que se introducen y su influencia en la predicción del modelo.

Con el método `SimpleImputer` del módulo `impute` de la librería `Scikit-learn` es posible imputar los valores ausentes empleando una métrica estadística (media, mediana, moda, etc.) o un valor constante. La figura 11 muestra un ejemplo en el que se ha empleado la media de los valores disponibles en la misma columna en la que se impulta el valor.

```
In [148]: cosecha_finca1= pd.DataFrame([[300, 400], [100, np.nan], [400, 200], [np.nan, 100]], columns=['trigo', 'maiz'), index=[2018, 2017, 2021, 2022])  
cosecha_finca1  
  
Out[148]:  
      trigo   maiz  
2018  300.0  400.0  
2017  100.0   NaN  
2021  400.0  200.0  
2022    NaN  100.0  
  
In [155]: from sklearn.impute import SimpleImputer  
imputer_mean = SimpleImputer(missing_values=np.Nan, strategy='mean')  
cosecha_finca1[['trigo','maiz']] = imputer_mean.fit_transform(cosecha_finca1[['trigo','maiz']])  
cosecha_finca1  
  
Out[155]:  
      trigo       maiz  
2018  300.000000  400.000000  
2017  100.000000  233.333333  
2021  400.000000  200.000000  
2022  266.666667  100.000000
```

Figura 11. Imputación de valores ausentes.

El módulo `impute` de la librería `Scikit-learn` también contiene un método iterativo (`IterativeImputer`) que permite emplear algoritmos de imputación iterativos como el KNN (k-Nearest Neighbors). Estos tipos de métodos son computacionalmente más costosos y pueden ofrecer valores poco realistas en el caso de las variables categóricas.

UNIVERSIDAD
COMPLUTENSE
DE MADRID



ntic
master
revolucionamos la comunicación

Autor: María José
Gómez Silva

Machine Learning con Python. Semana 1.

CAPÍTULO 3. PREPROCESADO

3. 3. Cambios de Formato, Normalización y Agrupación de los Datos

CAMBIOS DE FORMATO

En ocasiones es necesario cambiar el formato de los datos, para que sean mejor entendidos tanto por los analistas como por la máquina.

In [8]:	def mayusculas(cadena): return cadena.upper() orders=pd.read_excel('Superstore_Dataset.xlsx', converters={'Customer Name':mayusculas}) orders.head(3)																																																								
Out[8]:	<table><thead><tr><th>Row ID+O6G3A1:R6</th><th>Order ID</th><th>Order Date</th><th>Ship Date</th><th>Ship Mode</th><th>Customer ID</th><th>Customer Name</th><th>Segment</th><th>Country</th><th>City</th><th>State</th><th>Region</th><th>Product ID</th><th>Category</th></tr></thead><tbody><tr><td>0</td><td>1 CA-152156</td><td>2019-11-08</td><td>2019-11-11</td><td>Second Class</td><td>CG-12520</td><td>CLAIRE GUTE</td><td>Consumer</td><td>United States</td><td>Henderson</td><td>Kentucky</td><td>South</td><td>FUR-BO-10001798</td><td>Furniture</td></tr><tr><td>1</td><td>2 CA-152156</td><td>2019-11-08</td><td>2019-11-11</td><td>Second Class</td><td>CG-12520</td><td>CLAIRE GUTE</td><td>Consumer</td><td>United States</td><td>Henderson</td><td>Kentucky</td><td>South</td><td>FUR-CH-10000454</td><td>Furniture</td></tr><tr><td>2</td><td>3 CA-138688</td><td>2019-06-12</td><td>2019-06-16</td><td>Second Class</td><td>DV-13045</td><td>DARRIN VAN HUFF</td><td>Corporate</td><td>United States</td><td>Los Angeles</td><td>California</td><td>West</td><td>OFF-LA-10000240</td><td>Office Supplies</td></tr></tbody></table>	Row ID+O6G3A1:R6	Order ID	Order Date	Ship Date	Ship Mode	Customer ID	Customer Name	Segment	Country	City	State	Region	Product ID	Category	0	1 CA-152156	2019-11-08	2019-11-11	Second Class	CG-12520	CLAIRE GUTE	Consumer	United States	Henderson	Kentucky	South	FUR-BO-10001798	Furniture	1	2 CA-152156	2019-11-08	2019-11-11	Second Class	CG-12520	CLAIRE GUTE	Consumer	United States	Henderson	Kentucky	South	FUR-CH-10000454	Furniture	2	3 CA-138688	2019-06-12	2019-06-16	Second Class	DV-13045	DARRIN VAN HUFF	Corporate	United States	Los Angeles	California	West	OFF-LA-10000240	Office Supplies
Row ID+O6G3A1:R6	Order ID	Order Date	Ship Date	Ship Mode	Customer ID	Customer Name	Segment	Country	City	State	Region	Product ID	Category																																												
0	1 CA-152156	2019-11-08	2019-11-11	Second Class	CG-12520	CLAIRE GUTE	Consumer	United States	Henderson	Kentucky	South	FUR-BO-10001798	Furniture																																												
1	2 CA-152156	2019-11-08	2019-11-11	Second Class	CG-12520	CLAIRE GUTE	Consumer	United States	Henderson	Kentucky	South	FUR-CH-10000454	Furniture																																												
2	3 CA-138688	2019-06-12	2019-06-16	Second Class	DV-13045	DARRIN VAN HUFF	Corporate	United States	Los Angeles	California	West	OFF-LA-10000240	Office Supplies																																												
In [7]:	def titulo(cadena): return cadena.title() orders['Customer Name']=orders['Customer Name'].apply(titulo) orders.head(3)																																																								
Out[7]:	<table><thead><tr><th>Row ID+O6G3A1:R6</th><th>Order ID</th><th>Order Date</th><th>Ship Date</th><th>Ship Mode</th><th>Customer ID</th><th>Customer Name</th><th>Segment</th><th>Country</th><th>City</th><th>State</th><th>Region</th><th>Product ID</th><th>Category</th></tr></thead><tbody><tr><td>0</td><td>1 CA-152156</td><td>2019-11-08</td><td>2019-11-11</td><td>Second Class</td><td>CG-12520</td><td>Claire Gute</td><td>Consumer</td><td>United States</td><td>Henderson</td><td>Kentucky</td><td>South</td><td>FUR-BO-10001798</td><td>Furniture</td></tr><tr><td>1</td><td>2 CA-152156</td><td>2019-11-08</td><td>2019-11-11</td><td>Second Class</td><td>CG-12520</td><td>Claire Gute</td><td>Consumer</td><td>United States</td><td>Henderson</td><td>Kentucky</td><td>South</td><td>FUR-CH-10000454</td><td>Furniture</td></tr><tr><td>2</td><td>3 CA-138688</td><td>2019-06-12</td><td>2019-06-16</td><td>Second Class</td><td>DV-13045</td><td>Darrin Van Huff</td><td>Corporate</td><td>United States</td><td>Los Angeles</td><td>California</td><td>West</td><td>OFF-LA-10000240</td><td>Office Supplies</td></tr></tbody></table>	Row ID+O6G3A1:R6	Order ID	Order Date	Ship Date	Ship Mode	Customer ID	Customer Name	Segment	Country	City	State	Region	Product ID	Category	0	1 CA-152156	2019-11-08	2019-11-11	Second Class	CG-12520	Claire Gute	Consumer	United States	Henderson	Kentucky	South	FUR-BO-10001798	Furniture	1	2 CA-152156	2019-11-08	2019-11-11	Second Class	CG-12520	Claire Gute	Consumer	United States	Henderson	Kentucky	South	FUR-CH-10000454	Furniture	2	3 CA-138688	2019-06-12	2019-06-16	Second Class	DV-13045	Darrin Van Huff	Corporate	United States	Los Angeles	California	West	OFF-LA-10000240	Office Supplies
Row ID+O6G3A1:R6	Order ID	Order Date	Ship Date	Ship Mode	Customer ID	Customer Name	Segment	Country	City	State	Region	Product ID	Category																																												
0	1 CA-152156	2019-11-08	2019-11-11	Second Class	CG-12520	Claire Gute	Consumer	United States	Henderson	Kentucky	South	FUR-BO-10001798	Furniture																																												
1	2 CA-152156	2019-11-08	2019-11-11	Second Class	CG-12520	Claire Gute	Consumer	United States	Henderson	Kentucky	South	FUR-CH-10000454	Furniture																																												
2	3 CA-138688	2019-06-12	2019-06-16	Second Class	DV-13045	Darrin Van Huff	Corporate	United States	Los Angeles	California	West	OFF-LA-10000240	Office Supplies																																												

Figura 1. Conversión de formato de una columna de tipo cadena de caracteres

En la Figura 1 se muestra un ejemplo en el que se aplican dos conversiones de formato a los datos de la columna “*Customer Name*” de la hoja de cálculo “*Superstore_Dataset.xlsx*”. Se emplea una función para pasar a mayúsculas todos los nombres guardados en la columna y se aplica esa función en el momento de su lectura como *DataFrame*. Posteriormente se vuelve a aplicar otra función de conversión para guardar los nombres en formato *Título*.

```
In [56]: orders['Order Date'] = orders['Order Date'].astype(str)
orders.dtypes
```

```
Out[56]: Row ID+O6G3A1:R6      int64
Order ID          object
Order Date        object
Ship Date       datetime64[ns]
Ship Mode         object
Customer ID      object
Customer Name    object
Segment          object
Country          object
City              object
State             object
Region            object
Product ID        object
Category          object
Sub-Category      object
Product Name      object
Sales             float64
Quantity          int64
Profit            float64
dtype: object
```



```
In [57]: orders['Order Date'] = pd.to_datetime(orders['Order Date'], format='%Y-%m-%d')
orders.head(2)
```

```
Out[57]:
```

	Row ID+O6G3A1:R6	Order ID	Order Date	Ship Date	Ship Mode	Customer ID	Customer Name	Segment	Country	City	State	Region	Product ID	Category
0	1	CA-152158	2019-11-08	2019-11-11	Second Class	CG-12520	Claire Gute	Consumer	United States	Henderson	Kentucky	South	FUR-BO-10001798	Furniture
1	2	CA-152158	2019-11-08	2019-11-11	Second Class	CG-12520	Claire Gute	Consumer	United States	Henderson	Kentucky	South	FUR-CH-10000454	Furniture


```
In [58]: orders.dtypes
```

```
Out[58]: Row ID+O6G3A1:R6      int64
Order ID          object
Order Date       datetime64[ns]
Ship Date       datetime64[ns]
Ship Mode         object
Customer ID      object
Customer Name    object
Segment          object
Country          object
City              object
State             object
Region            object
Product ID        object
Category          object
Sub-Category      object
Product Name      object
Sales             float64
Quantity          int64
Profit            float64
dtype: object
```

Figura 2. Conversión a tipo datetime de una columna de un DataFrame

Cuando los datos incorporan columnas con información de fechas, resulta muy útil que tales columnas sean de tipo *objeto datetime*, ya que esto permite realizar una ordenación

cronológica. El *DataFrame* del ejemplo anterior contenía dos columnas con información de fechas “*Orders Date*” y “*Ship Date*”. Supongamos que por algún motivo la información en “*Orders Date*” se ha almacenado como cadenas de caracteres, en lugar de como objetos *datetime*. Para simular tal situación, en la Figura 2 se fuerza a que esa columna esté formada por cadenas de caracteres (str). Para recuperar su formato *datetime*, se puede emplear la función de Pandas *pd.to_datetime*, indicando el formato de fecha o fecha y hora apropiado. Una vez que la columna es de tipo *datetime*, ésta puede emplearse para ordenar cronológicamente el *DataFrame*, como en la Figura 3.

In [59]: orders.sort_values(by='Order Date')												
out[59]:												
	Row ID+06G3A1:R6	Order ID	Order Date	Ship Date	Ship Mode	Customer ID	Customer Name	Segment	Country	City	State	Region
2861		4919	CA-2019-160304	2019-01-02	2019-01-07	Standard Class	BM-11575	Brendan Murry	Corporate	United States	Gaithersburg	Maryland
2862		4920	CA-2019-160304	2019-01-02	2019-01-07	Standard Class	BM-11575	Brendan Murry	Corporate	United States	Gaithersburg	Maryland
5608		9495	CA-2019-106207	2019-01-03	2019-01-08	Standard Class	BO-11360	Bill Overfelt	Corporate	United States	Broken Arrow	Oklahoma

Figura 3. Ordenación cronológica de un DataFrame

NORMALIZACIÓN DE LOS DATOS

Especialmente en el caso de variables numéricas, la aplicación de algoritmos de Machine Learning, no sólo requiere que los datos aparezcan en un determinado formato, sino también el escalado de sus valores. Si los datos tienen campos numéricos con distintas escalas, como por ejemplo datos en una columna con valores de 0 a 1, y en otra de 100 a 1000, entonces es necesario convertirlos a una misma escala común. Ese proceso es denominado *normalización*.

La representación de las distribuciones estadísticas de los datos de cada columna de tipo numérico de un DataFrame nos permitirá observar los rangos de valores que presenta cada variable, como muestra el ejemplo de la Figura 4.

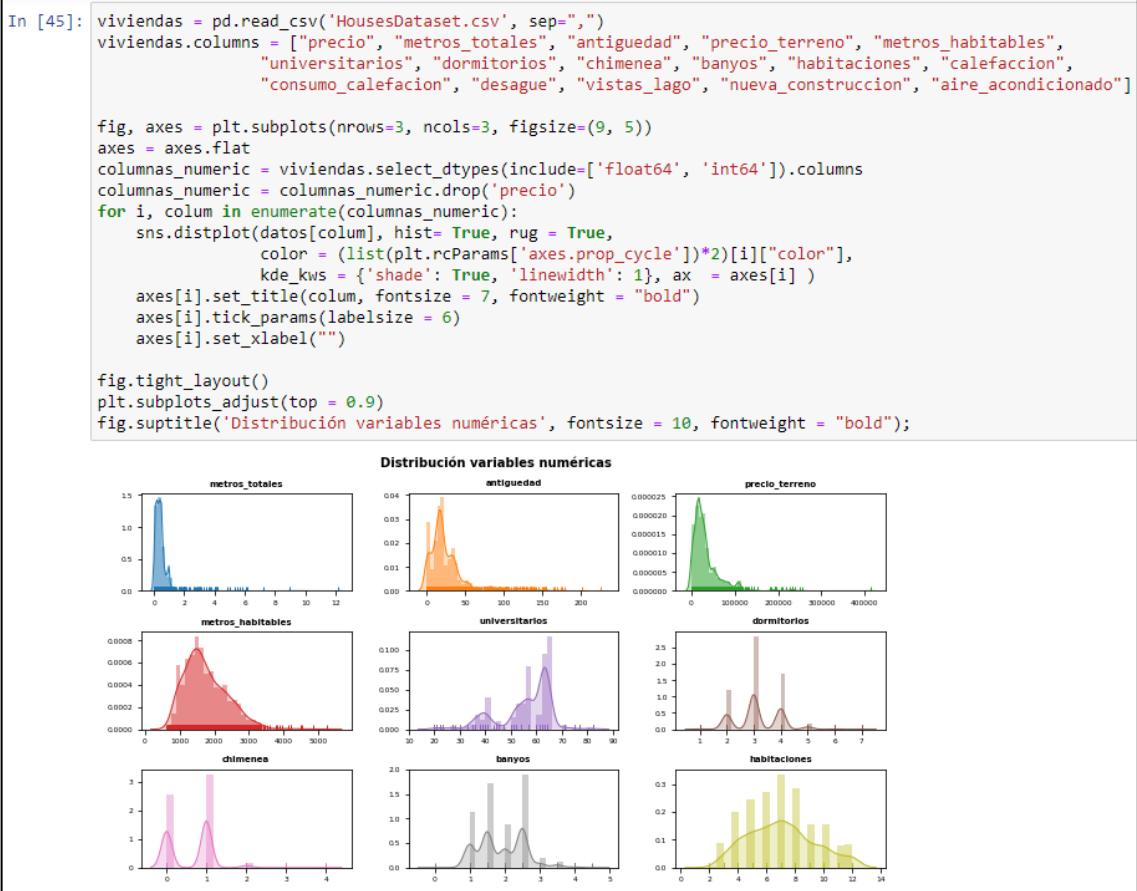


Figura 4. Representación de las distribuciones estadísticas de varias variables numéricas.

```
In [53]: viviendas.chimenea.value_counts()

Out[53]: 1      942
          0      740
          2       42
          4       2
          3       2
          Name: chimenea, dtype: int64

In [54]: viviendas.chimenea = viviendas.chimenea.astype("str")
viviendas.chimenea

Out[54]: 0      1
          1      0
          2      1
          3      1
          4      0
          ..
         1723     1
         1724     1
         1725     0
         1726     1
         1727     0
          Name: chimenea, Length: 1728, dtype: object
```

Figura 5. Cambio de formato de la columna 'chimenea'.

Cuando existen variables que presentan un rango de posibles valores discreto y muy reducido, como es el caso de la columna *chimenea* (véase la figura 4), suele ser conveniente transformar la variable en cuestión de numérica a categórica. Esto se consigue modificando el formato o tipo de los datos de numérico a cadena de caracteres (*str*), como muestra la Figura 5. En Pandas, el tipo *object* se refiere a datos de tipo *string* (cadenas de caracteres).

La normalización o escalado de las variables se consigue aplicando transformaciones aritméticas para comprender sus valores entre unos valores mínimos y máximos comunes. De este modo se homogenizan los datos, todas las variables se mueven en unos mismos rangos de valores y eso favorece el aprendizaje automático, ya que facilita que el peso de cada variable dependa de su relevancia para generar la predicción del modelo, y no de la escala de sus valores.

Existen dos formas principales de normalizar los datos:

- La **estandarización**. Para cada valor de una variable le resta la media y lo divide entre la desviación típica de la variable. De esta forma, la variable pasa a tener una distribución normal. Se puede realizar con la función *StandardScaler* del módulo *preprocessing* de *Scikit-learn*.

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

- El **escalado**. Para cada valor de una variable le aplica la siguiente fórmula. De esta forma, la variable pasa a tener un valor mínimo de cero y un valor máximo de uno. Se puede realizar con la función *MinMaxScaler* del módulo *preprocessing* de *Scikit-learn*.

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

Por su parte, el centrado consiste en únicamente restar a cada valor el valor de la media de la variable en cuestión. Se puede realizar con la función *StandardScaler* del módulo *preprocessing* de *Scikit-learn*, poniendo el argumento *with_std* a *False*.

BINARIZACIÓN DE LAS VARIABLES CATEGÓRICAS

La mayoría de los algoritmos de Machine Learning requieren que las variables categóricas sean binarizadas. Para ello, se emplea el formato conocido como one-hot-encoding. Se trata de indicar con valores binarios (0 o 1) si la variable toma un determinado valor o no. Por ejemplo, si tenemos una columna de un dataframe que toma valores de colores de un determinado producto y los únicos valores que puede tomar son ‘rojo’, ‘verde’ y ‘azul’, su binarización implica la creación de tres nuevas columnas que sustituirán a la columna original. Será necesaria una columna por cada valor de la variable, en este caso una columna para cada color. En cada fila se indicará el color que tomaba la variable original, colocando un 1 en la columna que corresponda y 0 en las demás.

```
In [163]: from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.compose import make_column_selector
numeric_cols = viviendas.select_dtypes(include=['float64', 'int64']).columns.to_list()
cat_cols = viviendas.select_dtypes(include=['object', 'category']).columns.to_list()
preprocessor = ColumnTransformer( [( 'scale', StandardScaler(), numeric_cols),
                                    ('onehot', OneHotEncoder(handle_unknown='ignore'), cat_cols)],
                                    remainder='passthrough')
viviendas_prep = preprocessor.fit_transform(viviendas)
encoded_cat = preprocessor.named_transformers_['onehot'].get_feature_names(cat_cols)
labels = np.concatenate([numeric_cols, encoded_cat])
datos_viviendas_prep = preprocessor.transform(viviendas)
datos_viviendas_prep = pd.DataFrame(datos_viviendas_prep, columns=labels)
datos_viviendas_prep.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1728 entries, 0 to 1727
Data columns (total 27 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   precio          1728 non-null    float64
 1   metros_totales  1728 non-null    float64
 2   antiguedad      1728 non-null    float64
 3   precio_terreno  1728 non-null    float64
 4   metros_habitable 1728 non-null    float64
 5   universitarios  1728 non-null    float64
 6   dormitorios     1728 non-null    float64
 7   banyos          1728 non-null    float64
 8   habitaciones    1728 non-null    float64
 9   chimenea_0      1728 non-null    float64
 10  chimenea_1      1728 non-null    float64
 11  chimenea_2_mas  1728 non-null    float64
 12  calefaccion_electric 1728 non-null    float64
 13  calefaccion_hot_air 1728 non-null    float64
 14  calefaccion_hot_water/steam 1728 non-null    float64
 15  consumo_calefaccion_electric 1728 non-null    float64
 16  consumo_calefaccion_gas 1728 non-null    float64
 17  consumo_calefaccion_oil 1728 non-null    float64
 18  desague_none    1728 non-null    float64
 19  desague_public/commercial 1728 non-null    float64
 20  desague_septic   1728 non-null    float64
 21  vistas_lago_No  1728 non-null    float64
 22  vistas_lago_Yes 1728 non-null    float64
 23  nueva_construccion_No 1728 non-null    float64
 24  nueva_construccion_Yes 1728 non-null    float64
 25  aire_acondicionado_No 1728 non-null    float64
 26  aire_acondicionado_Yes 1728 non-null    float64
dtypes: float64(27)
memory usage: 364.6 KB
```

Figura 6. Binarización de variables categóricas y estandarización de variables numéricas.

En el ejemplo de la Figura 6 se muestra como la librería *Scikit-learn* permite binarizar todas las variables categóricas de un *DataFrame*, empleando el método *ColumnTransformer* y la clase *OneHotEncoder*. En este ejemplo también se emplea *ColumnTransformer* para estandarizar las variables numéricas.

Por defecto, la clase *OneHotEncoder* binariza todas las variables, por lo que hay que aplicarlo únicamente a las variables categóricas. Con el argumento *drop='first'* se elimina una de las nuevas columnas generadas para evitar redundancias. En el ejemplo anterior de las columnas para representar tres colores, no son necesarias las tres columnas, ya que con dos de ellas tendremos toda la información necesaria. Si las dos están a cero, por descarte el color es el de la columna eliminada.

En ciertos escenarios puede ocurrir que, en los datos de test, aparezca un nuevo nivel que no estaba en los datos de entrenamiento. Si no se conoce de antemano cuáles son todos los posibles niveles, se puede evitar errores en las predicciones indicando *OneHotEncoder(handle_unknown='ignore')*.

TRANSFORMACIONES CON SCIKIT-LEARN

La librería *Scikit-learn* permite realizar operaciones de preprocesado sobre las columnas de un *DataFrame* gracias a las clases *ColumnTransformer* y *make_column_transformer* del módulo *compose*. Es posible realizar varias transformaciones diferentes sobre los datos especificando a qué columnas se aplica cada una. Estas clases son *transformers*, eso significa que tienen un método de entrenamiento (*fit*) y otro de transformación (*transform*). Esto permite que el aprendizaje de las transformaciones se haga únicamente con datos de entrenamiento, y se puedan aplicar después a cualquier conjunto de datos, al de entrenamiento, validación y test.

En el ejemplo de la Figura 6 se realiza una selección de columnas por tipo, y luego se define la transformación a aplicar sobre cada grupo seleccionado. El resultado devuelto por *ColumnTransformer* es un array de *numpy*, por lo que se pierden los nombres de las columnas. Por ese motivo hay que volver a generar un *DataFrame* con los nombres de las nuevas columnas y los datos generados por *ColumnTransformer*.

ColumnTransformer aplica las transformaciones en paralelo, no de forma secuencial. Por lo tanto, solo puede aplicar una transformación a una misma columna. Si se necesita aplicar más de una transformación sobre una columna, se puede emplear el método *pipeline*, como muestra la Figura 7. Este método también agrupa transformaciones, pero las ejecutan de forma secuencial, de forma que la salida de una operación es la entrada de la siguiente.

```
In [165]: from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.compose import make_column_selector
numeric_cols = viviendas.select_dtypes(include=['float64', 'int64']).columns.to_list()
cat_cols = viviendas.select_dtypes(include=['object', 'category']).columns.to_list()
numeric_transformer = Pipeline( steps=[ ('imputer', SimpleImputer(strategy='median')),
                                       ('scaler', StandardScaler()) ])
categorical_transformer = Pipeline(steps=[ ('imputer', SimpleImputer(strategy='most_frequent')),
                                           ('onehot', OneHotEncoder(handle_unknown='ignore')) ])
preprocessor = ColumnTransformer( transformers=[ ('numeric', numeric_transformer, numeric_cols),
                                                 ('cat', categorical_transformer, cat_cols) ], remainder='passthrough' )
viviendas_prep = preprocessor.fit_transform(viviendas)
encoded_cat = preprocessor.named_transformers_['cat']['onehot'].get_feature_names(cat_cols)
labels = np.concatenate([numeric_cols, encoded_cat])
datos_viviendas_prep = preprocessor.transform(viviendas)
datos_viviendas_prep = pd.DataFrame(datos_viviendas_prep, columns=labels)
datos_viviendas_prep.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1728 entries, 0 to 1727
Data columns (total 27 columns):
 #   Column            Non-Null Count  Dtype  
--- 
 0   precio             1728 non-null   float64
 1   metros_totales     1728 non-null   float64
 2   antiguedad         1728 non-null   float64
 3   precio_terreno     1728 non-null   float64
 4   metros_habitable   1728 non-null   float64
 5   universitarios     1728 non-null   float64
 6   dormitorios        1728 non-null   float64
 7   banyos              1728 non-null   float64
 8   habitaciones       1728 non-null   float64
 9   chimenea_0          1728 non-null   float64
 10  chimenea_1          1728 non-null   float64
 11  chimenea_2_mas      1728 non-null   float64
 12  calefaccion_electric 1728 non-null   float64
 13  calefaccion_hot_air 1728 non-null   float64
 14  calefaccion_hot_water/steam 1728 non-null   float64
 15  consumo_calefacion_electric 1728 non-null   float64
 16  consumo_calefacion_gas    1728 non-null   float64
 17  consumo_calefacion_oil   1728 non-null   float64
 18  desague_none         1728 non-null   float64
 19  desague_public/commercial 1728 non-null   float64
 20  desague_septic        1728 non-null   float64
 21  vistas_lago_No        1728 non-null   float64
 22  vistas_lago_Yes       1728 non-null   float64
 23  nueva_construcción_No 1728 non-null   float64
 24  nueva_construcción_Yes 1728 non-null   float64
 25  aire_acondicionado_No 1728 non-null   float64
 26  aire_acondicionado_Yes 1728 non-null   float64
```

Figura 7. Agrupación de transformaciones en pipelines

AGRUPACIÓN DE DATOS EN PANDAS

En algunos casos es muy útil agrupar los datos en función de alguna característica, especialmente si se trata de la característica que queremos que nuestro modelo aprenda a predecir. De este modo, podremos conocer, por ejemplo, el número de clases diferentes en nuestro conjunto de datos, y el número de muestras de cada una de ellas.

Los *DataFrames* ofrecen el método *groupby* que permite dividir los datos en función de los valores de una o más variables, y permite hacer grupos de filas (*axis=0*) o de columnas (*axis=1*). El resultado es un objeto de tipo *Groupby* que tiene propiedades

como la de `ngrups` o `size` que permite conocer el número de grupos y el número de elementos de cada grupo. En la Figura 8 se muestra un ejemplo de agrupación por estación sobre un *DataFrame* con datos de meses.

```
In [8]: tabla = pd.DataFrame([('Enero', 31, 'inv', 2, 4),('Febrero', 28, 'inv', 3, 2),('Marzo', 31, 'inv', 7, 5),
                           ('Abril', 30, 'pri', 7, 9),('Mayo', 31, 'pri', 9, 10),('Junio', 30, 'pri', 15, 14),
                           ('Julio', 31, 'ver', 20, 24),('Agosto', 31, 'ver', 27, 26),('Septiembre', 30, 'ver', 25, 18),
                           ('Octubre', 31, 'oto', 20, 14),('Noviembre', 30, 'oto', 11, 10),('Diciembre', 31, 'oto', 6, 7)],columns=['Mes', 'Días', 'Estación', 'Temp.2021', 'Temp.2022'],
                           index=['Ene', 'Feb', 'Mar', 'Abr', 'May', 'Jun', 'Jul', 'Ago', 'Sep', 'Oct', 'Nov', 'Dic'])
tabla

Out[8]:
      Mes   Días Estación Temp.2021 Temp.2022
Ene     Enero    31      inv        2         4
Feb    Febrero    28      inv        3         2
Mar     Marzo    31      inv        7         5
Abr     Abril    30      pri        7         9
May     Mayo    31      pri        9        10
Jun     Junio    30      pri       15        14
Jul     Julio    31      ver       20        24
Ago    Agosto    31      ver       27        26
Sep   Septiembre 30      ver       25        18
Oct   Octubre    31      oto       20        14
Nov  Noviembre   30      oto       11        10
Dic  Diciembre   31      oto        6         7

In [11]: g = tabla.groupby(['Estación'])
g.groups
Out[11]: {'inv': Index(['Ene', 'Feb', 'Mar'], dtype='object'),
          'oto': Index(['Oct', 'Nov', 'Dic'], dtype='object'),
          'pri': Index(['Abr', 'May', 'Jun'], dtype='object'),
          'ver': Index(['Jul', 'Ago', 'Sep'], dtype='object')}

In [12]: g.size()
Out[12]:
Estación
inv    3
oto    3
pri    3
ver    3
dtype: int64
```

Figura 8. Ejemplo de división en grupos de los datos de un *DataFrame*.

La división en grupos permite la aplicación de determinadas operaciones sobre los elementos de cada grupo, por ejemplo, en la Figura 9 se muestra como calcular la temperatura media de cada estación en 2022.

```
In [15]: tabla.groupby(['Estación'])['Temp.2022'].mean()
Out[15]:
Estación
inv    3.666667
oto    10.333333
pri   11.000000
ver   22.666667
Name: Temp.2022, dtype: float64
```

Figure 9. Resultado de aplicar una operación estadística por grupos.

UNIVERSIDAD
COMPLUTENSE
DE MADRID



ntic
master
revolucionamos la comunicación

Autor: María José
Gómez Silva

Machine Learning con Python. Semana 1.

CAPÍTULO 3. PREPROCESADO.

3. 4 Sesgo en Machine Learning y Aumento de los Datos

SESGO EN MACHINE LEARNING

Es usual que las bases de datos que empleamos para entrenar los algoritmos de Machine Learning estén sesgadas, es decir, que exista un cierto predominio de algún o algunos tipos de datos más que de otros.

En ocasiones, tal sesgo se produce durante la captura de los datos. Por ejemplo, si queremos desarrollar un sistema de visión artificial para vehículos autónomos y tomamos secuencias de vídeo durante distintos trayectos por Madrid, el número de ejemplos de vehículos que aparecerán entre nuestros datos será mucho más alto que el de bicicletas. Sin embargo, esa proporción, inevitablemente cambiaría si los videos fuesen tomados en Ámsterdam.

Cuando trabajamos con algoritmos de clasificación, ese sesgo puede traducirse en un desbalance entre clases. En el caso anterior, si clasificamos los datos en vehículos con o sin motor, existirá un gran desbalanceo entre las dos clases. El hecho de que las clases de nuestra base de datos estén o no desbalanceadas, dependerá del tipo de clasificación que realicemos y vendrá determinado por el número de ejemplos de cada clase. Sin embargo, una misma clase puede seguir presentando cierto sesgo o predominio de ejemplos con ciertas características (por ejemplo, pueden existir ciertas marcas dentro de la clase de vehículos a motor que sean más frecuentes que otras). El sesgo dentro de los datos tipificados como de una misma clase es en muchas ocasiones difícil de identificar y evitar, y suele estar ocasionado por circunstancias sociales.

En otros casos, la propia naturaleza de la tarea a resolver provoca tal desequilibrio en los datos. Por ejemplo, en la tarea de re-identificación hay que comparar la imagen de una persona, con cientos de imágenes de personas diferentes entre las que vuelve a encontrarse la persona buscada. En ese caso, sólo un emparejamiento es correcto (clase positiva) y el resto de los emparejamientos son incorrectos (clase negativa). Por ello, si se aborda la re-identificación como una calificación binaria (parejas correctas o incorrectas), existe un gran desequilibrio entre ambas clases.

El entrenamiento de un modelo de clasificación con una base de datos con clases desbalanceadas puede provocar su colapso. Esto significa que el modelo acaba

ofreciendo únicamente predicciones de una clase. Por ejemplo, si tenemos un 10% de ejemplos de una clase y un 90% de otra, es fácil que el modelo tienda a aprender que siempre tiene que predecir que los ejemplos son de la segunda clase, ya que tiene menos probabilidad de equivocarse.

Si las clases no están desbalanceadas, pero existe sesgo dentro de una misma clase, o si estamos entrenando otro tipo de modelo (distinto al de clasificación) pero igualmente los datos están sesgados, entonces se produce lo que se conoce como **sesgo o bias en el modelo**, o también, **bias en Machine Learning**, o **bias en Inteligencia Artificial** en general. Este término se refiere a la tendencia de los modelos a ofrecer predicciones sesgadas, que en general reflejan sesgos sociales presentes en las bases de datos. La Figura 1 refleja el sesgo en cuanto a la procedencia de las imágenes en un par de bases de datos para clasificación de imágenes.

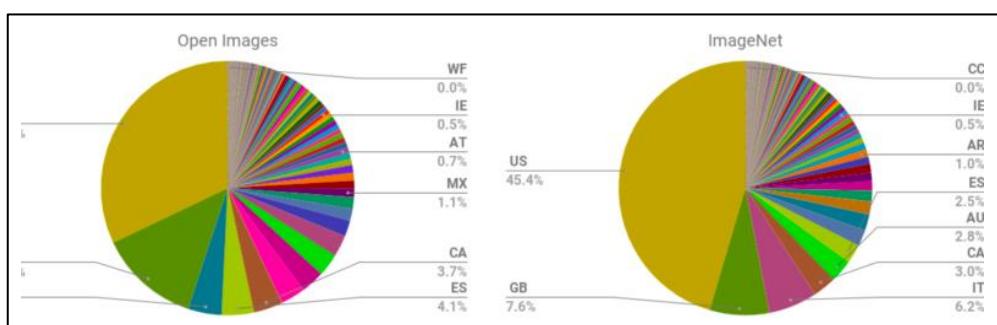


Figura 1. Fracción de cada país, representado con su código ISO, en las bases de datos de Open Images e ImageNet. Esquema procede del artículo: *A Survey on Bias and Fairness in Machine Learning. ACM Computing Survey (CSUR), 54, 1 - 35.* De los autores Mehrabi, N., Morstatter, F., Saxena, N.A., Lerman, K., & Galstyan, A.G. (2019).

Entre los métodos para corregir el sesgo y desbalanceo de las clases se encuentran los siguientes:

- **Adaptación de las funciones de pérdida.** Se puede cambiar el peso que la función de pérdidas da a cada clase para evitar el colapso de los modelos. Se trata de que la función de pérdidas penalice en la misma medida los errores cometidos al predecir cualquiera de las clases. Existen algunas funciones de pérdidas, como la **Focal Loss Function**, diseñada para resolver problemas de clasificación cuando las clases no están balanceadas.
- **Submuestreo o sobremuestreo.** Puede ser muy beneficioso corregir el balance o el sesgo desde la raíz, en la propia base de datos. La solución óptima sería aumentar el número de ejemplos (sobre-muestrear, over-sampling en inglés) el tipo de datos más desfavorecido inicialmente, como muestra la Figura 2. Si esta opción no es posible, una alternativa sería reducir el número de muestras (submuestreo, down-sampling, en inglés) de los datos con más predominio.

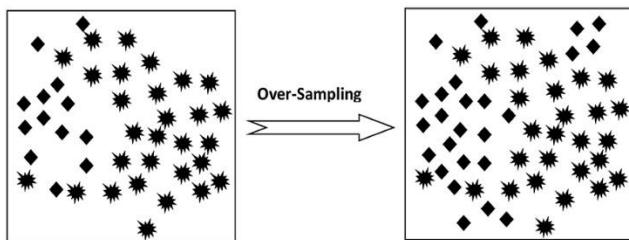


Figura 2. Sobre-muestreo para compensar el desbalance de clases

- **Combinaciones y permutaciones de los datos.** En los casos en los que los datos estén formados por pares (o incluso ternas), como en el ejemplo de la tarea de re-identificación, se pueden buscar métodos apropiados de combinación y permutación para balancear los ejemplos de cada clase.

AUMENTO DE LOS DATOS

A medida que aumenta la complejidad de un modelo de Machine Learning, y consecuentemente, el número de parámetros que contiene es necesario una mayor cantidad de datos para su entrenamiento.

El **incremento de la cantidad de datos ayuda a evitar el sobreajuste de los modelos**, como se explicó en la sección de *Problemas del entrenamiento* del capítulo 1. Por otro lado, la variedad de datos ayuda a entrenar un modelo robusto, capaz de ofrecer buenas predicciones frente a diversos datos de entrada.

La estrategia de aumento de datos engloba a todas aquellas técnicas empleadas para aumentar la cantidad de muestras disponibles mediante la modificación o transformación leve de copias de los datos ya existentes o la creación de datos sintéticos nuevos. Algunas de las técnicas más empleadas se enumeran a continuación, y se representan en la Figura 3.

- El **añadido de ruido a los datos, o pequeñas variaciones de sus valores**
- El **desplazamiento de la zona de captura, cuando tenemos datos secuenciales o en imágenes.**
- Las **transformaciones geométricas**, que son bastante populares para entrenar algoritmos de visión por computador, como por ejemplo la rotación y el volteo (flip) de las imágenes.
- La **generación de nuevas muestras sintéticas**, gracias a modelos neuronales como las Redes Adversarias Generativas (Generative Adversarial Networks, GANs).

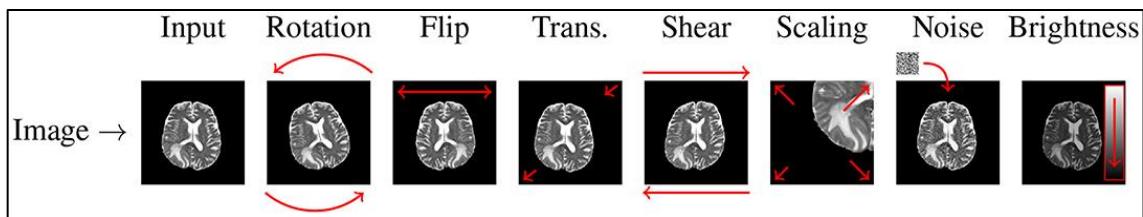


Figura 3. Ejemplos de técnicas de aumento de datos sobre imágenes.

Cuando se realiza aumento de datos, es importante seguir la premisa de que, el empleo de nuevos datos, sintéticos o modificaciones de los antiguos, será beneficioso siempre y cuando los nuevos datos tengan un aspecto similar a los datos reales a los que el modelo final tendrá que enfrentarse. Por ejemplo, si estamos diseñando un detector de peatones, podemos hacer un volteo vertical de las imágenes que ya tengamos de peatones, pero no conviene realizar un volteo horizontal, ya que el modelo nunca va a tener que lidiar con personas andando con los pies hacia arriba y la cabeza hacia abajo.

UNIVERSIDAD
COMPLUTENSE
DE MADRID



ntic
master
revolucionamos la comunicación

Autor: María José
Gómez Silva

Machine Learning con Python. Semana 1.

CAPÍTULO 3. PREPROCESADO.

3.5 Selección e Ingeniería de Características y División en Subconjuntos

SELECCIÓN DE CARACTERÍSTICAS E INGENIERÍA DE CARACTERÍSTICAS

Cada una de las variables almacenadas en una tabla o base de datos (cada columna de un DataFrame) es una característica de una muestra, que se ha medido y por lo tanto toma un determinado valor. Volviendo al antiguo ejemplo en el que teníamos un listado de viviendas en venta, en ese caso, cada vivienda es una muestra de la que se han medido distintas características (variables), como el número de habitaciones, el número de baños, la superficie total en metros cuadrados, etc.

Antes de entrenar un modelo es necesario revisar los datos y seleccionar las características que más influencia tendrán en la predicción del modelo y descartar las demás. Para ello, se suelen emplear representaciones como la de la matriz de correlación (estudiada en la sección *Análisis multi-variable y visualización* del Capítulo 2). Se trata de encontrar las características que presenten más correlación con la variable a predecir. En el ejemplo anterior, si queremos predecir el precio de venta de las viviendas, y tenemos ejemplos de ventas anteriores, usaremos el precio por el que se vendieron como etiqueta o referencia, y buscaremos características que guarden relación con ese valor. Por ejemplo, características como la superficie o el número de habitaciones posiblemente estén más correlacionadas con el precio de venta, que otras como, por ejemplo, el número de cuadros que haya en las paredes.

Otra consideración bastante habitual cuando se seleccionan características para el entrenamiento de un modelo de Machine Learning es la de no incluir variables que tomen un único valor, ya que no aportan información. Esas variables son fáciles de identificar ya que su varianza es igual a cero.

La clase `VarianceThreshold` del módulo `sklearn.feature_selection` identifica y excluye todos aquellos predictores cuya varianza no supera un determinado umbral (`threshold`). Cuando una variable presenta una varianza cercana a cero, significa que toma solo unos pocos valores, de los cuales algunos aparecen con muy poca frecuencia. En ese caso se corre el riesgo de que, al dividir los datos en subconjuntos para el entrenamiento, validación y test, no haya muestras de algunos de los valores minoritarios en alguno de los subconjuntos. Si se pretende realizar una selección de características basada en un cierto umbral de varianza, esta selección debe realizarse antes de la estandarización de los datos, ya que, tras la estandarización, todos tienen varianza igual a 1.

Por su parte, la *ingeniería de características* no consiste simplemente en la selección de las características, sino en su combinación para generar otras nuevas basadas en las actuales.

DIVISIÓN EN SUBCONJUNTOS

Una vez que los datos están preparados y preprocesados, hay que proceder a su división en tres conjuntos, el de entrenamiento, el de validación y el de test, como se explicó en la sección *Etapas de un Proyecto de Machine Learning* del capítulo 1.

Una división común en el ámbito del Machine Learning, es la de 60%, 20% y 20% para los conjuntos de entrenamiento, validación y test, respectivamente. Aunque si el modelo es muy complejo, se suele dar más peso al conjunto de entrenamiento para evitar el sobreajuste del modelo.

Una práctica extendida, es la de desordenar los datos antes de realizar la división en los tres grupos, de forma que el reparto sea lo más aleatorio posible y haya variedad de distintos tipos de muestras en cada subconjunto.

La *validación cruzada* (cross-validation en inglés) es una técnica utilizada para evaluar las predicciones de un modelo y garantizar que son independientes de la forma en que se realizó la partición de los datos. Consiste en realizar diferentes particiones de los datos, y para cada una de ellas repetir el entrenamiento y la validación. El rendimiento final del modelo se medirá como la media de los valores obtenidos con cada partición. En este caso no solo el valor medio es importante, sino también la dispersión de los resultados obtenidos, ya que nos indicará la dependencia del comportamiento del modelo en función de los datos de entrada.

Al realizar la división en distintos subconjuntos, es deseable que todos los subconjuntos contengan muestras de todos los posibles valores de cada variable, especialmente cuando estas son numéricas discretas y cuando son categóricas. Si el conjunto de datos global tiene algunas variables (columnas) en las cuales hay valores/clases que se repiten muy poco, hay muchas probabilidades de que esos valores no estén presentes en alguno de los subconjuntos tras la división de los datos. Es decir, algunos subconjuntos no contendrán observaciones de dicha clase.

Para evitar tal situación, existen distintas estrategias de preprocesado o filtrado, como eliminar las observaciones del grupo minoritario si se trata de una variable multiclasa, o eliminar la propia variable si esta es binaria. También se pueden agrupar las clases minoritarias en una sola clase, como se muestra en el ejemplo de la Figura 1, donde la variable *chimenea* pasa de tener 5 posibles clases a 3. El objetivo es asegurar que todas las clases estén representadas en cada subconjunto de datos.

```
In [78]: viviendas.chimenea.value_counts().sort_index()
Out[78]: 0    740
1    942
2     42
3      2
4      2
Name: chimenea, dtype: int64

In [79]: dic_replace = {'2': "2_mas", '3': "2_mas", '4': "2_mas"}
viviendas['chimenea'] = viviendas['chimenea'].map(dic_replace).fillna(viviendas['chimenea'])
viviendas.chimenea.value_counts().sort_index()

Out[79]: 0      740
1      942
2_mas   46
Name: chimenea, dtype: int64
```

Figura 1. Agrupación de las clases minoritarias de una variable categórica.

La librería **Scikit-learn** permite realizar la división de los datos en distintos subconjuntos de forma muy sencilla con la función `train_test_split`, como muestra la Figura 2. No solo se dividen los datos a usar como entrada sino también los datos a usar como referencia o etiqueta (variable de salida).

```
In [85]: from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split( viviendas.drop('precio', axis = 'columns'),
                                                    viviendas['precio'], train_size = 0.8, random_state = 1234,
                                                    shuffle = True )

In [86]: print(y_train.describe())
          Esto desordena los datos
count    1382.000000
mean    211436.516643
std     96846.639129
min     10300.000000
25%    145625.000000
50%    190000.000000
75%    255000.000000
max    775000.000000
Name: precio, dtype: float64

In [87]: print(y_test.describe())
count    346.000000
mean    214084.395954
std     104689.155889
min     5000.000000
25%    139000.000000
50%    180750.000000
75%    271750.000000
max    670000.000000
Name: precio, dtype: float64
```

Figura 2. División de los datos en subconjuntos de entrenamiento y de test

Con el argumento `stratify` es posible indicar en función de que variable queremos que se realice el reparto de los datos. En ese caso se está realizando un reparto estratificado. Su objetivo es asegurar que el conjunto de entrenamiento y el de test sean similares en cuanto a la variable indicada con el argumento `stratify`. La práctica más común es emplear la variable a predecir como variable en función de la cual realizar el reparto.

- Accuracy: % de aciertos, pero si las clases están desbalanceadas esto no me muestra la verdadera calidad del modelo.
- Precisión: Penaliza los FP
- Recall: Penaliza los FN