VNiVERSiDAD D SALAMANCA

CAMPUS DE EXCELENCIA INTERNACIONAL

Grado en Estadística

Trabajo de Fin de Grado

Aplicación web interactiva para el análisis de datos multivariantes mediante técnicas de aprendizaje automático

Interactive web application of multivariate machine learning statistical techniques

---

Autor

Juan Marcos Díaz

Tutoras académicas

Ana Belén Nieto Librero

Nerea González García

Facvltad D Ciencias
VNiVERSiDAD D SALAMANCA

2023

VNiVERSiDAD D SALAMANCA

CAMPUS DE EXCELENCIA INTERNACIONAL

Grado en Estadística

Trabajo de Fin de Grado

Aplicación web interactiva para el análisis de datos multivariantes mediante técnicas de aprendizaje automático

Interactive web application of multivariate machine learning statistical techniques

---

Autor

Juan Marcos Díaz

Tutores académicos

Ana Belén Nieto Librero

Nerea González García

Facvltad D Ciencias

VNiVERSiDAD D SALAMANCA

2023

Certificado de los tutores TFG Grado en Estadística

Dª. Ana Belén Nieto Librero, profesora del Departamento de Estadística de la Universidad de Salamanca, y Dª. Nerea González García, profesora del Departamento de Estadística de la Universidad de Salamanca,

HACEN CONSTAR:

Que el trabajo titulado "*Aplicación web interactiva para el análisis de datos multivariantes mediante técnicas de aprendizaje automático*", que se presenta, ha sido realizado por D. Juan Marcos Díaz, con DNI ****8971 y constituye la memoria del trabajo realizado para la superación de la asignatura Trabajo de Fin de Grado en Estadística en esta Universidad.

Salamanca, a fecha de firma electrónica.

NIETO LIBRERO ANA BELEN - 11966438K

Firmado digitalmente por NIETO LIBRERO ANA BELEN - 11966438K
Fecha: 2023.07.04 16:24:59 +02'00'

Fdo.: Ana Belén Nieto Librero

GONZALEZ GARCIA NEREA - 72828390V

Firmado digitalmente por GONZALEZ GARCIA NEREA - 72828390V
Fecha: 2023.07.04 17:27:39 +02'00'

Fdo.: Nerea González García

Índice

1) Introducción 1
2) Preprocesado de datos 4
2.1) Codificación 4
2.2) Discretización 5
2.3) Escalado 5
2.4) Tratamiento de valores perdidos 7
2.5) Selección de variables 7
3) Aprendizaje no supervisado 8
3.1) Reducción de la dimesionalidad 8
3.1.1) Análisis de componentes principales 9
3.2) Clustering 12
3.2.1) *Clustering* jerárquico 14
3.2.2) *Clustering* particional 16
3.2.3) *Clustering* basado en densidad 18
3.2.4) *Clustering* basado en modelo 20
3.2.5) *Clustering* en cuadrículas (*Grid-Based*) 20
3.2.6) Métricas de evaluación y selección del número de *clusters* 20
4) Aprendizaje Supervisado 23
4.1) Regresión 24
4.1.1) Regresión lineal – Enfoque clásico 25
4.1.2) Regresión Lineal - Enfoque del ML 27
4.1.3) Métricas de evaluación 29
4.2) Clasificación 30
4.2.1) Regresión logística 31
4.2.2) Máquinas de soporte vectorial 33
4.2.3) Métricas de evaluación 37
5) Aplicaciones Web 39
6) Resultados 42
7) Conclusiones 46
8) Bibliografía 48

Resumen

Este trabajo define y describe algunas de las principales técnicas y algoritmos de Aprendizaje Automático para después, aplicarlos en una aplicación web interactiva. En primer lugar, realizaremos una revisión bibliográfica de estas técnicas, entre las que se encuentran la reducción de la dimensionalidad, el *clustering*, la regresión y la clasificación. En esta parte teórica, para cada técnica, describiremos detalladamente el funcionamiento de uno o varios algoritmos y especificaremos las métricas o métodos necesarios para su evaluación. Posteriormente, mediante el lenguaje de programación Python y la librería Streamlit, especializada en el desarrollo de aplicaciones web, crearemos una página interactiva en la que podremos aplicar y evaluar las distintas técnicas estudiadas, ya sea con un conjunto de datos de prueba, incluido en la propia aplicación, o con conjuntos de datos propios. El objetivo principal de este trabajo es que, tras su revisión, cualquier persona sea capaz de comprender y poner en práctica los principales algoritmos de aprendizaje automático a través de una interfaz atractiva sin necesidad de programar.

Palabras clave: Aprendizaje Automático, aplicación web, reducción de la dimensionalidad, *clustering*, regresión, clasificación, Python y Streamlit.

Abstract

This project defines and describes some of the main Machine Learning techniques and algorithms to subsequently engage them in an interactive web application. In the first place, we will carry out a bibliographical revision of the aforementioned techniques, among which we may find the reduction of dimensionality, clustering, regression and classification. In this theoretical part, we will describe in detail for each technique the functioning of one or more algorithms and we will specify the metrics or methods needed for their evaluation. Afterwards, through Python programming language and the Streamlit library, specialised in the development of web applications, we will create an interactive webpage where we will be able to apply and evaluate the different techniques we have studied, be it with an ensemble of test data included in the application, or with a collection of our own data. The main objective of this project is, after its revision, for any person to be able to understand and put into practice the main machine learning algorithms through an active interface without needing to program.

Keywords: Machine Learning, web application, reduction of dimensionality, clustering, regression, classification, Python and Streamlit.

1) Introducción

Durante la década de los 50 (McCarthy et al., 2006) se comenzó a utilizar el término **Inteligencia Artificial (IA)** para describir el comportamiento de máquinas inteligentes, es decir, que tienen capacidad para imitar la inteligencia humana y realizar tareas que normalmente requerirían la intervención humana, como el aprendizaje, la resolución de problemas o la toma de decisiones.

También en esta década (Samuel, 1959) se propuso la que hasta ahora es la definición más aceptada del término **Aprendizaje Automático** o ***Machine Learning*** **(ML)**:

"Es el área de estudio que confiere a los ordenadores la capacidad de aprender una tarea específica sin ser explícitamente programados para ella".

Así pues, consideramos el ML como un subcampo de la IA, o más concretamente, una técnica que se utiliza para desarrollar sistemas de IA, que permite a las máquinas aprender a partir de los datos (Helm et al., 2020). El ML surge para resolver las limitaciones del enfoque simbólico en la IA, una corriente que dominó el campo de la IA durante la década de los 50 hasta bien entrados los 80 y que se basa en representaciones de alto nivel "simbólico" (comprensibles para los humanos) de los problemas. Los algoritmos de enfoque simbólico, entre los que destacan los sistemas expertos (Jackson, 1986), resultaban útiles en problemas que podían ser definidos mediante reglas, pero carecen de la capacidad de aprender a partir de los datos con los que trabajan y por tanto su uso es limitado. Es ahí cuando el ML empieza a ganar relevancia.

En este trabajo, realizaremos una descripción detallada de algunos de los algoritmos de *ML* más populares, pero para ello, primero debemos describir brevemente la clasificación y los tipos de algoritmos. En función del tipo de aprendizaje podemos clasificar los algoritmos de ML en los siguientes grupos:

- **Aprendizaje supervisado (*supervised learning*):** estos algoritmos funcionan y se entrenan con conjuntos de datos etiquetados, es decir, conjuntos de observaciones en las que se conoce la salida esperada y cuyo objetivo es predecir esta variable salida para nuevos conjuntos de datos (Singh et al., 2016). Dependiendo de si la variable etiqueta es categórica o numérica trabajaremos con algoritmos de clasificación o regresión respectivamente. Un ejemplo de algoritmos de aprendizaje supervisado serían las máquinas de vector soporte (SVM) (Boser et al., 1992; Cortes & Vapnik, 1995) utilizadas en clasificación.

- **Aprendizaje no supervisado (*unsupervised learning*):** son algoritmos que trabajan con conjuntos de datos no etiquetados (no existe una variable salida) y cuya misión es encontrar estructuras y patrones ocultos en dichos datos (Gentleman & Carey, 2008). Un ejemplo sería el análisis de componentes principales o *Principal Component Analysis* (Hotelling, 1933), para la reducción de la dimensionalidad.

- **Aprendizaje semisupervisado (*semi-supervised learning*)**: utilizaremos estos algoritmos cuando trabajemos con conjuntos de datos formados por una minoría de datos etiquetados (con variable salida) y una inmensa mayoría de datos sin etiquetar. El funcionamiento genérico de estos modelos suele ser utilizar los datos etiquetados para

aprender patrones y aplicarlos en los datos sin etiquetar (Van Engelen & Hoos, 2020). Uno de los algoritmos más populares de este tipo es la propagación de etiquetas o *Label Encoder*, en el que se propagan las etiquetas de los datos etiquetados a los datos sin etiquetar en función a un grafo de similitud de las observaciones (Lee, 2013).

- **Aprendizaje reforzado (*reinforcement learning*):** son algoritmos cuyo objetivo es entrenar un modelo para tomar decisiones en entornos complejos y dinámicos, en los que no podemos obtener todos los datos y por tanto sería imposible aplicar el aprendizaje supervisado. La idea principal sería enseñar al modelo a maximizar una recompensa y minimizar posibles castigos (Wiering & Van Otterlo, 2012). El algoritmo más popular de este tipo es el *Q-Learning* (Watkins & Dayan, 1992), que utiliza una tabla de valores o estados Q para guiar la toma de decisiones. Podemos imaginar por ejemplo un dron aprendiendo a moverse de forma autónoma (esquivar obstáculos y calibrar su posición) hasta una determinada posición que sería la recompensa.

Otra posible categoría en la que agrupar algunos algoritmos de ML sería el popular ***deep learning*** o **aprendizaje profundo** (LeCun et al., 2015) que se caracteriza por el uso de redes neuronales artificiales profundas, es decir, con muchas capas, como las redes convolucionales CNN (O'Shea & Nash, 2015) para el reconocimiento de imágenes. Aunque podríamos considerar este conjunto de algoritmos como un subgrupo dada su importancia y complejidad, a su vez consideramos que podemos agrupar cada algoritmo de *deep learning* de forma individual en alguna de las categorías mencionadas anteriormente. En este trabajo nos centraremos en describir algunas de las técnicas más populares de aprendizaje supervisado y del aprendizaje no supervisado, pero consideramos importante recalcar la existencia del resto.

Así pues, en esta memoria y después de esta breve introducción en la que definimos algunos conceptos clave como el ML y sus subtipos, estudiaremos más detalladamente algunas de las técnicas y algoritmos más populares de este campo para posteriormente aplicarlos en la aplicación web.

En el siguiente capítulo, el capítulo 2, introduciremos las principales técnicas de preprocesado como pueden ser la codificación o el escalado, con las que transformaremos los conjuntos de datos para poder posteriormente utilizarlos en los modelos de ML.

En los capítulos 3 y 4 desarrollaremos más detalladamente los algoritmos de aprendizaje no supervisado y aprendizaje supervisado respectivamente. En el aprendizaje no supervisado describiremos la reducción de la dimensionalidad, centrándonos en el análisis de componentes principales y el *clustering* con todos sus subtipos. Por otra parte, para el aprendizaje supervisado describiremos la regresión lineal, así como algunos de los algoritmos más populares de clasificación. Además, para cada técnica detallaremos las principales métricas o técnicas de evaluación.

A continuación, en el capítulo 5, describimos brevemente las principales técnicas o lenguajes de programación necesarios para desarrollar una aplicación web, puesto que ese es el objetivo final del trabajo.

Posteriormente, en el capítulo 6, incluiremos una pequeña sección de resultados describiendo brevemente los datos obtenidos en la aplicación web, aunque toda esa información la encontraremos mucho más detallada en la propia aplicación.

Por último, encontraremos un apartado de conclusiones en el capítulo 7 en el que recogemos todas las impresiones obtenidas tanto de esta memoria cómo de la propia aplicación web.

2) Preprocesado de datos

Como hemos indicado anteriormente, los algoritmos de ML aprenden y trabajan con datos y, aunque existen algunos que no lo necesitan, para poder aplicar correctamente la mayoría de ellos primero debemos transformar los datos en lugar de trabajar con los datos en crudo directamente. Esta parte del tratamiento de los datos se conoce como preprocesado y es de vital importancia (Alasadi & Bhaya, 2017). En este capítulo describiremos algunas de las técnicas más utilizadas en esta etapa.

2.1) Codificación

Estrictamente consiste en transformar datos no numéricos (texto, imágenes…) en datos numéricos, aunque en nuestro caso vamos a considerar sólo las técnicas que transforman variables categóricas a numéricas. La codificación es un proceso fundamental para muchos algoritmos de ML que sólo trabajan con variables numéricas (Seger, 2018). Algunas de estas técnicas son:

- **Codificación ordinal:** útil para variables categóricas en las que exista un orden natural. Por ejemplo, tres categorías: Menor de 30 años, 30-60 años y Más de 60 años, les podemos asignar las modalidades 0, 1 y 2 respectivamente sin perder información.

- **Codificación *One-Hot*:** consiste en crear variables *dummies* para cada una (suponemos *n* categorías) o para *n-1* categorías de la variable. Por ejemplo, si contamos con una variable sexo con tres categorías (hombre, mujer y otro) podríamos construir dos (*n-1*) variables auxiliares (mujer y otro) de tal forma que el valor en la variable Mujer fuera 1 para todas las mujeres y 0 para el resto, de manera análoga para la variable Otro de tal forma que los individuos con valor 0 en ambas variables sean hombres. Podemos ver otro ejemplo en la siguiente imagen (*Ilustración 1*).

Ilustración 1. Codificación One-Hot. Fuente: elaboración propia en Canva

2.2) Discretización

Entendemos la discretización como el proceso que convierte datos continuos (con un rango infinito de posibles valores) en datos discretos, es decir, transformamos una variable continua y dividimos sus valores en categorías o intervalos discretos, donde cada valor dentro de un intervalo se considera equivalente. Habitualmente se utiliza esta técnica cuando nos interesa agrupar las observaciones con el fin de diferenciar mejor nuevos grupos, de tal forma que por ejemplo podríamos transformar la variable edad en *n* categorías ordinales. El número de nuevos valores discretos y los puntos de corte en la variable continua suelen depender del criterio del investigador, aunque existen algoritmos o técnicas estadísticas (como la distribución por cuantiles) que pueden optimizar el proceso. Podemos observar gráficamente este proceso en la siguiente figura (*Ilustración 2*).

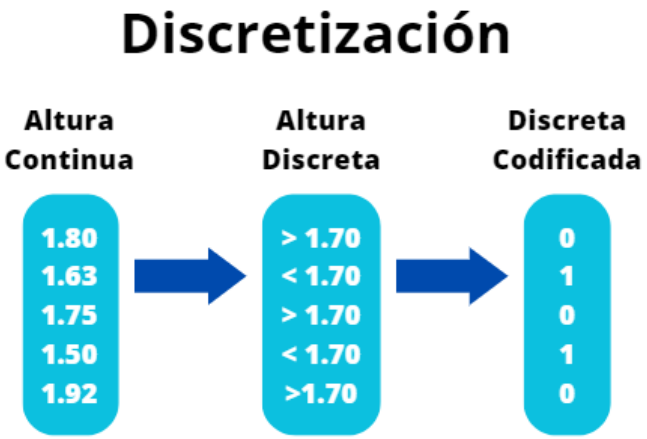

Ilustración 2. Discretización y codificación. Fuente: elaboración propia en Canva

2.3) Escalado

El objetivo de esta técnica es ajustar los valores de las variables de modo que se encuentren en una escala uniforme y comparable. Cuando los datos de entrada tienen diferentes magnitudes o unidades, algunos algoritmos pueden dar más peso a las variables con valores más grandes, lo que puede afectar negativamente el rendimiento del modelo, por lo que el escalado es muy importante, ya que puede mejorar la precisión y la eficacia de estos modelos. Además, algunos algoritmos, como los algoritmos basados en distancias (*k-means* por ejemplo), requieren que los datos estén en una escala común para su correcto funcionamiento. Algunos de los métodos de escalado (Ahsan et al., 2021) más populares son los siguientes (*Ilustración 3*):

- **Estandarización:** es uno de los métodos más utilizados y útiles, supone que los datos siguen una distribución normal. Consiste en restar a cada observación la media de la variable μ y dividir por la desviación típica σ de tal modo que las nuevas variables tendrán $\mu = 0$ y $\sigma = 1$. Para i variables (columnas) y j individuos (filas) las fórmulas serían:

$$\mu_i := \frac{1}{n}\sum_{j=1}^{n} x_i^{(j)} \qquad \sigma_i := \sqrt{\frac{1}{n}\sum_{j=1}^{n}\left(x_i^{(j)} - \mu_i\right)^2}$$

$$x_i := \frac{x_i - \mu_i}{\sigma_i}$$

- **Escalado Min-Max**: es útil cuando los valores extremos son importantes en nuestro estudio. Todos los valores quedarán comprendidos en el intervalo [0,1]. La principal limitación de esta técnica es que es muy sensible a los *outliers* o valores extremos. Para *i* variables y *j* individuos las fórmulas serían:

$$max = \max_{0<j<n} x_i^{(j)} \qquad min = \min_{0<j<n} x_i^{(j)}$$

$$x_i := \frac{x_i - min}{max - min}$$

- **Escalado Min-Max corregido:** técnica análoga a la anterior, pero sobre el intervalo [-1,1]. Para *i* variables la fórmula sería:

$$x_i := \frac{2(x_i - min)}{max - min} - 1$$

- **Escalado Robusto:** técnica similar al estandarizado, pero ignorando los *outliers*. Consiste en restar la mediana Q_2 y escalar entre el primer y el tercer cuartil [Q_1, Q_3]. Para *i* variables la fórmula sería:

$$x_i := \frac{x_i - Q_2}{Q_3 - Q_1}$$

- **Normalización a norma unitaria:** normalizamos cada variable de forma individual para que tenga longitud unitaria respecto a alguna norma, normalmente la euclídea (L_2). Para *i* variables la fórmula sería:

$$x := \frac{x}{\sqrt{\sum_i (x_i)^2}}$$

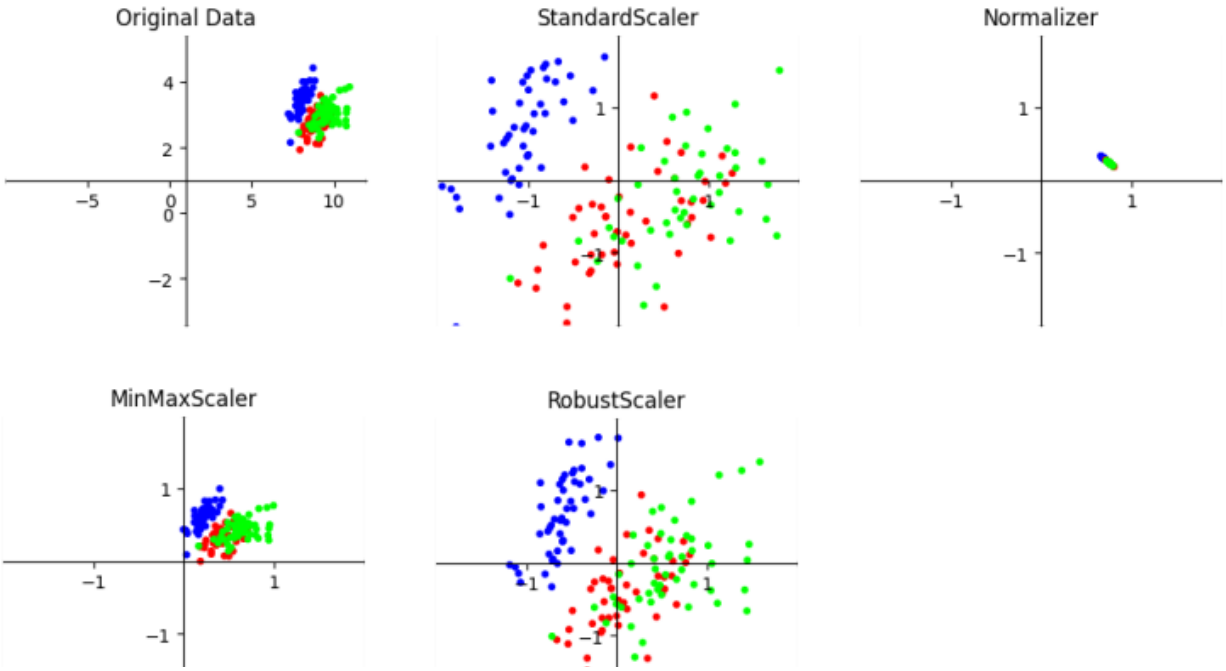

Ilustración 3. Tipos de escalado. Fuente: Google Images

2.4) Tratamiento de valores perdidos

Es habitual trabajar con bases de datos incompletas, es decir, en las que por diversos motivos faltan algunos valores (Emmanuel et al., 2021). Dependiendo de la cantidad y la importancia de dichos valores perdidos, podemos emplear distintas técnicas:

- **Eliminar la/las observaciones:** si el número de observaciones es grande, y existen sólo unas pocas observaciones con valores perdidos, podemos optar por eliminarlas, suponiendo pequeña la pérdida de información.

- **Eliminar la/las variables:** podemos tomar esta decisión si consideramos que la variable no es útil y posee demasiados valores perdidos, aunque en general esta técnica es poco utilizada ya que supone una pérdida grande de información.

- **Sustituir los valores perdidos por medidas de centralización:** es la técnica más utilizada ya que conlleva una pérdida de información mucho menor que las anteriores alternativas. Es muy común utilizar la media y sobre todo la mediana (menos sensible a valores extremos) de una variable cuantitativa para remplazar los valores perdidos correspondientes a dicha variable. Por otra parte, se puede utilizar el valor más frecuente (la moda) para sustituir valores perdidos en variables cualitativas.

- **Utilizar algoritmos de ML para predecir los valores perdidos**: por ejemplo, podemos utilizar algoritmos como los *K* vecinos más cercanos o *K-nearest neighbors* (Knn) que transforma los valores perdidos por medidas de centralización de las variables de los *k* vecinos u observaciones más cercanas.

- **No hacer nada:** ya que los valores perdidos aportan información por sí mismos y muchas veces es conveniente mantener su estatus, además, algunos algoritmos pueden trabajar con la inexistencia de estos valores sin afectar a su rendimiento.

2.5) Selección de variables

Es muy importante elegir bien que variables vamos a utilizar o introducir en nuestro modelo, ya que con frecuencia trabajaremos con conjuntos de datos muy grandes en los que no todas las variables resultarán relevantes para nuestro estudio. Además, existen algoritmos que trabajan mejor con conjuntos de datos con un número reducido de variables y, aunque existen técnicas de reducción de la dimensionalidad de las que hablaremos en el siguiente capítulo, siempre podemos comenzar por seleccionar simplemente aquellas variables que aporten más información.

En algunos casos eliminaremos variables que presenten una correlación extremadamente alta respecto a otra y, específicamente en el caso del aprendizaje supervisado, aquellas variables en los que sus distintos valores, no afecten a la variable respuesta, mediante el uso de contrastes estadísticos como por ejemplo el Chi^2 para variables cualitativas o el *ANOVA* para variables cuantitativas si la respuesta tiene más de dos categorías, aunque existen muchos más ejemplos. Tras estos test tenderemos a eliminar las variables que muestren poca o ninguna variabilidad.

3) Aprendizaje no supervisado

El aprendizaje no supervisado, como habíamos visto, es una rama del ML que se centra en encontrar patrones, estructuras y relaciones en los datos sin la necesidad de una etiqueta o una variable respuesta que predecir (Gentleman & Carey, 2008). En este capítulo, exploraremos en profundidad las técnicas y principales algoritmos del aprendizaje no supervisado, y cómo pueden ser utilizados para resolver problemas concretos en diversas áreas.

3.1) Reducción de la dimesionalidad

La reducción de la dimensionalidad es una técnica utilizada para simplificar conjuntos de datos, o lo que es lo mismo, reducir el número de variables, manteniendo la mayor cantidad de información posible (Sorzano et al., 2014). Estas técnicas pueden ser útiles para muchos objetivos, como pueden ser: reducir el ruido en los datos (valores atípicos o valores perdidos), mejorar la eficiencia de los algoritmos de ML o posibilitar la visualización de los datos en espacios de baja dimensionalidad (representaciones en 2 o 3 dimensiones). Existen a grandes rasgos, dos maneras para llevar a cabo esta tarea:

- **La selección de variables:** como vimos en el anterior capítulo, mediante técnicas estadísticas para eliminar variables de menor variabilidad y quedarnos sólo con aquellas que aporten más información, aunque también se pueden utilizar algoritmos de ML para esta tarea.

- **La extracción de características:** mediante algoritmos de ML, busca transformar el conjunto de datos original en uno nuevo de dimensión reducida manteniendo la mayor cantidad de información posible. Dentro de este enfoque, podemos encontrar numerosos algoritmos de ML, tanto de aprendizaje supervisado como no supervisado, siendo estos últimos más populares. Al estar dentro del capítulo de aprendizaje no supervisado, vamos a centrarnos en estas técnicas. Así pues, algunos de estos algoritmos son:

    - **Análisis de componentes principales:** esta técnica es una de las más populares (Hotelling, 1933). Describiremos detalladamente su funcionamiento y aplicación más adelante, pero la idea general es transformar el conjunto de datos original en un nuevo conjunto más pequeño en el que las nuevas variables son combinaciones lineales de las variables originales, procurando explicar la mayor cantidad de variabilidad posible.

    - **Análisis de componentes independientes,** por sus siglas en inglés ICA, es similar al análisis de componentes principales, este algoritmo construye las variables del nuevo conjunto de datos de dimensión reducida a partir de combinaciones lineales de las variables originales que sean lo más independientes posible entre sí (Hyvarinen, 1999).

- ***t-Distributed Stochastic Neighbor Embedding* (t-SNE):** utilizado para visualizar datos de alta dimensión en un espacio de menor dimensión. Se diferencia de otras técnicas ya tiende a enfocarse en la estructura de vecindad en los datos originales, preservando así las relaciones locales entre las observaciones (Van der Maaten & Hinton, 2008).

En la siguiente sección explicaremos detalladamente el análisis de componentes principales al considerarla la técnica más representativa del campo de la reducción de la dimensionalidad.

3.1.1) Análisis de componentes principales

El análisis de componentes principales (PCA, por sus siglas en inglés), es una técnica estadística empleada en análisis multivariante qué, como habíamos indicado anteriormente, se utiliza para reducir el número de variables transformando las variables, originales mediante combinaciones lineales de estas en un nuevo conjunto de variables denominadas "componentes principales".

Este algoritmo fue desarrollado por primera vez por el matemático y estadístico británico Harold Hotelling (Hotelling, 1933), basándose en las ideas de Karl Pearson para reducir la dimensionalidad transformando variables ortogonales (Pearson, 1901).

La idea principal de esta técnica es buscar la dirección en la que los datos presentan la mayor varianza entendiendo que a mayor varianza, más información. Una vez encontrada esta dirección, se deberán proyectar los datos originales sobre esta, obteniendo así la primera "componente". De manera análoga, se busca la segunda dirección de mayor varianza y se proyecta sobre ella, de tal forma que, si el rango es máximo e igual al número de variables, se obtienen tantas componentes principales como variables originales, con la particularidad de que cada componente explica un porcentaje de variabilidad (y por tanto de información), mayor que la siguiente componente (*Ilustración 4*).

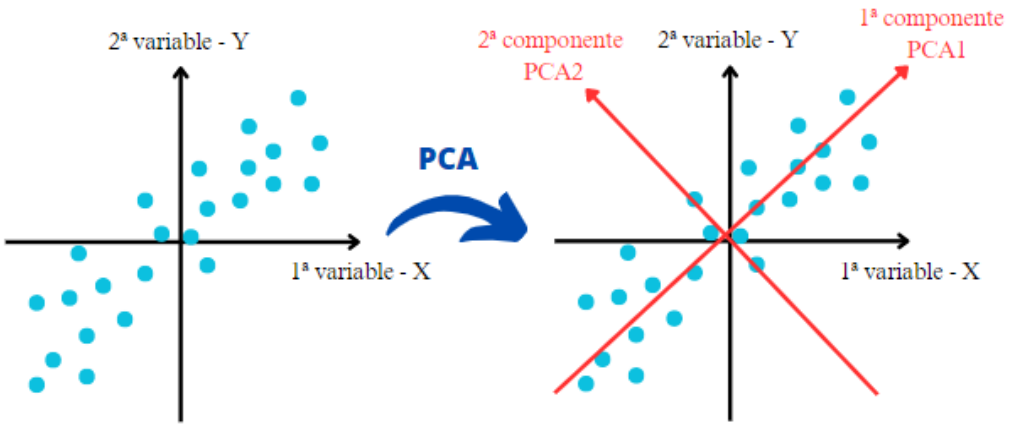

Ilustración 4. Obtención de las componentes principales. Fuente: elaboración propia con Canva

Desarrollo teórico

Como hemos indicado anteriormente, la idea es buscar transformaciones ortogonales de las variables originales para conseguir un nuevo conjunto de variables incorreladas que maximicen la varianza y por tanto la información (Shlens, 2014).

Supongamos sin pérdida de generalidad que todas las variables originales tienen media 0 y sea *X* la matriz de los datos con *n* filas y *p* columnas, de tal forma que cada fila se corresponde con una observación y cada columna con una de las variables originales.

Buscamos la matriz de puntuaciones en componentes principales Y tal que $Y = XV$, es decir, buscamos combinaciones lineales de las variables originales.
Para la obtención de la primera componente principal Y_1, buscamos maximizar su varianza

$$Var(y_1) = \frac{1}{n} Y_1^T Y_1 = \frac{1}{n} v_1^T X^T X v_1 = v_1^T S v_1$$

Siendo S la matriz de covarianzas de X. Debemos incluir una restricción de la norma $v_1^T v_1 = 1,\ V^T V = I$ para obtener una solución única. Así pues, aplicando el multiplicador de Lagrange λ para esta restricción, debemos maximizar la siguiente función:

$$L(v_1) = v_1^T S v_1 - \lambda(v_1^T v_1 - 1)$$

Derivamos e igualamos a 0, para obtener el máximo.

$$\frac{L(v_1)}{\partial v_1} = 2Sv_1 - 2\lambda v_1 = 0$$

De donde obtenemos que $Sv_1 = \lambda v_1$, por lo que deducimos que v_1 tiene que ser un vector propio de la matriz de covarianzas S con valor propio asociado λ.

Como $Var(Xv_1) = v_1^T S v_1 = v_1^T \lambda v_1 = \lambda$, entonces λ tiene que ser λ_1, el valor propio más grande y v_1 su vector propio asociado. Es decir, el primer valor propio de la matriz de covarianzas se corresponde con la varianza de la primera componente principal.

Podemos deducir que, de manera análoga, las componentes principales se obtienen de la descomposición en autovalores (valores propios) y vectores propios de la matriz de covarianzas. Así pues, concluiríamos que las componentes principales son incorreladas, obteniéndose en orden decreciente de importancia. Además, observamos que obtenemos tantas componentes como variables originales (si el rango es máximo e igual al número de variables), aunque la mayor parte de la varianza quede explicada por las primeras componentes y sea labor del investigador decidir con cuantas se queda finalmente.

Resumiendo, podemos definir el algoritmo PCA para una matriz de datos *X* en los siguientes pasos:

1. **Escalado:** antes de aplicar el algoritmo, los datos deben estar escalados para asegurarnos que todas las variables están en la misma escala. Después de aplicar algún método como la estandarización, obtenemos una matriz estandarizada *Z*.

2. **Cálculo de la matriz de covarianza *S*:** calculamos la matriz de covarianza *S* sobre los datos estandarizados *Z*. Esta matriz nos mostrará las relaciones entre las variables. Definimos la covarianza de X_1, X_2 como:

$$Cov(X_1, X_2) = \frac{1}{n}\sum_{i=1}^{n}(x_{i1} - \overline{x_1})(x_{i2} - \overline{x_2})$$

Con $\overline{x_1}, \overline{x_2}$ las medias de las variables X_1, X_2 respectivamente. Para el conjunto de datos centrado *Z*, su matriz de covarianza es:

$$S = \frac{1}{n}Z^T Z$$

3. **Cálculo de los vectores y valores propios de la matriz de covarianzas *S***: podemos calcularlos resolviendo la siguiente ecuación $Sv = \lambda v$ con λ, v valor y vector propio respectivamente y *S* continúa siendo la matriz de covarianzas. Los vectores propios nos indicarán las direcciones de máxima variabilidad de los datos mientras que los valores propios nos mostrarán el porcentaje de variabilidad explicado en dichas direcciones.

4. **Selección de las componentes**: en función de distintos criterios que explicaremos posteriormente, el investigador deberá decidir con cuantas componentes se queda. Hay que tener en cuenta que la primera componente (correspondiente al valor propio más grande) explicará más varianza que la siguiente y así sucesivamente, de tal forma que las primeras componentes recogen casi toda la información.

5. **Proyección de los datos sobre las componentes:** reducimos la dimensionalidad proyectando los datos centrados *Z* en una nueva matriz *Y* con el número de componentes (columnas) seleccionado

$$Y = ZV_k$$

Es decir, la matriz de datos proyectados Y se obtiene al multiplicar la matriz de datos originales centrados Z por la matriz de componentes principales V_k, en la que k indica el número de componentes seleccionadas.

Criterios para la evaluación y selección de variables

Es tarea del investigador decidir con cuantas componentes se queda, existiendo varios métodos que le guiarán a la hora de tomar esta decisión evaluando cuanta información se pierde.

- **Criterio de varianza explicado**: se establece un umbral mínimo de varianza explicada (en torno al 70% o el 80%) y se seleccionan suficientes componentes principales para alcanzar ese umbral, teniendo en cuenta que las primeras componentes explican la mayor proporción de la varianza.

- **Criterio del codo**: se representa gráficamente el porcentaje de varianza explicada (eje Y) por cada componente principal ordenada (eje X). El número de componentes a elegir nos lo indicará el punto que representa el cambio drástico en la pendiente (o "codo") de la gráfica, indicando que seleccionar más componentes no supondría un aumento considerable de la varianza explicada. Este gráfico se denomina grafico de sedimentación o *Scree Plot* (*Ilustración 5*).

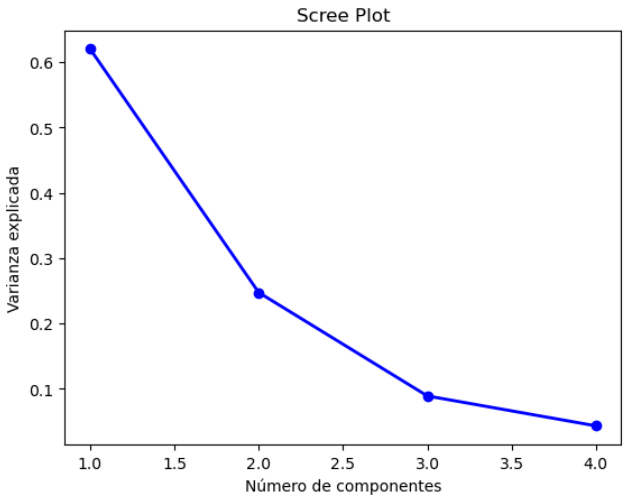

Ilustración 5. Sree Plot. Número óptimo de componentes 2. Fuente: elaboración propia en Python a partir del siguiente dataset ***https://raw.githubusercontent.com/JWarmenhoven/ISLR-python/master/Notebooks/Data/USArrests.csv***

- **Criterio de interpretabilidad:** criterio personal en función del problema, el investigador debe seleccionar un número de componentes principales fácilmente interpretable y relevante.

3.2) Clustering

Se trata de otra de las técnicas de ML no supervisado más populares. Consiste en agrupar un conjunto de datos en subconjuntos, conglomerados o "*clusters*" en función de su similitud, de tal forma que los individuos pertenecientes a cada *cluster* sean más similares entre ellos mismos que frente a los de los otros *clusters*. Así pues, el objetivo principal del agrupamiento, más conocido como *clustering* es identificar patrones o estructuras en los datos, que pueden ser útiles para la toma de decisiones o para descubrir información valiosa, como por ejemplo para la segmentación de clientes en base a sus hábitos de compra (Madhulatha, 2012). Para evaluar la similitud se emplean distintas medidas de distancia dependiendo de los datos y también del algoritmo empleado. Aunque existen infinidad de medidas, algunas de las más utilizadas son (Thant et al., 2020) (*Ilustración 6*):

- **Distancia de Manhattan:** también conocida como distancia ciudad, mide la distancia entre puntos como la suma de las diferencias absolutas de sus coordenadas. Recibe este nombre ya que, para dos dimensiones, las variables pueden trazarse sobre una cuadrícula que puede compararse con las calles de la ciudad, de tal forma que la distancia entre dos puntos equivaldría al número de manzanas que recorrería una persona. Para dos puntos A y B, con coordenadas X_i e Y_i respectivamente donde $i = 1, \dots, n$ indica la dimensión, la distancia es:

$$d(A,B) = \sum_{i=1}^{n} |X_i - Y_i|$$

- **Distancia Euclídea**: o norma L_2, es la distancia más utilizada en *clustering* y recibe este nombre en honor al matemático griego Euclides, considerado como el padre de la

geometría. Es la longitud del vector que conecta dos puntos en el espacio *n*-dimensional, es decir, la distancia más corta entre dos puntos en línea recta. Para dos puntos A y B se calcula como la raíz cuadrada de la suma de los cuadrados de las diferencias entre sus coordenadas:

$$d(A,B) = \sqrt{\sum_{i=1}^{n}(X_i - Y_i)^2}$$

- **Distancia de Minkowski**: o métrica L_p es una generalización de las distancias anteriores y fue propuesta por el matemático alemán Hermann Minkowski en 1908. Se define como:

$$d(A,B) = \left(\sum_{i=1}^{n}|X_i - Y_i|^p\right)^{\frac{1}{p}}$$

donde p indica el tipo de distancia (con $p = 1$ tenemos la distancia de Manhattan y con $p = 2$ la distancia Euclídea).

- **Distancia de Chebyshev:** distancia del ajedrez o métrica máxima L_∞ fue propuesta por el matemático ruso Pafnuty Chebyshev en 1854 y es especialmente útil cuando se emplea sobre datos en forma de tablero, como en el caso del ajedrez, donde los movimientos se realizan en una cuadrícula. Se define como la máxima diferencia entre las coordenadas de dos puntos en un espacio *n*-dimensional:

$$d(A,B) = \max_i|X_i - Y_i|$$

- **Distancia de Mahalanobis:** a diferencia de las distancias anteriores, que consideran que las dimensiones son independientes y tienen la misma importancia, esta distancia considera que las variables pueden estar correlacionadas y que algunas variables pueden ser más importantes que otras para medir la similitud entre los objetos. Fue propuesta por el estadístico indio Prasanta Chandra Mahalanobis, fundador del Instituto de Estadística de la India, en 1936. Sea *S* la matriz de covarianzas entre *A* y *B*, definimos la distancia como:

$$d(A,B) = \sqrt{(X_i - Y_i)S^{-1}(X_i - Y_i)^T}$$

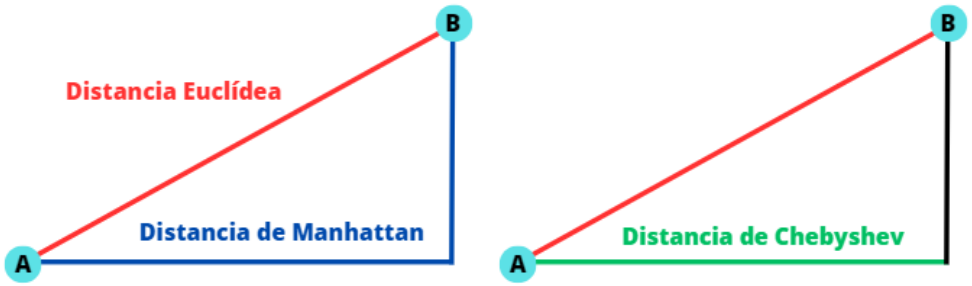

Ilustración 6. Distancias. Fuente: elaboración propia con Canva, basado en ilustración de la revista Mètode 2017 - 94. Sapiens - Verano 2017

Al existir muchas formas distintas de agrupar los datos en función de muchas medidas diferentes, cuando hablamos de *clustering* no hablamos de un único algoritmo, sino de una técnica de agrupamiento de objetos que engloba numerosos algoritmos, cada uno con sus propias características, ventajas e inconvenientes y, en función de los datos, el investigador debe decidir cuál es el más apropiado para cada caso. A grandes rasgos, se pueden agrupar todos los algoritmos en los siguientes grupos que describiremos más detalladamente en los siguientes apartados (Madhulatha, 2012):

- *Clustering* jerárquico.
- *Clustering* particional.
- *Clustering* basado en densidad.
- *Clustering* basado en modelo.
- *Clustering* en cuadrículas.

3.2.1) *Clustering* jerárquico

Este tipo de *clustering* agrupa las observaciones o individuos en una estructura jerárquica de *clusters*, de manera que crea *clusters* sucesivos utilizando otros previamente establecidos. No existe un único descubridor como tal de este tipo de algoritmos, aunque se considera que los primeros artículos relacionados con el *clustering* jerárquico aparecen en la década de los 60 (Johnson, 1967; Sokal & Sneath, 1963). Dentro de este tipo de *clustering*, los algoritmos se dividen a su vez en dos categorías:

- **Aglomerativos:** comienzan considerando cada individuo como un *cluster* y los van fusionando entre ellos formando así *clusters* cada vez más grandes. Una vez elegida la medida de distancia (por ejemplo, siendo *d* la distancia euclídea), comenzamos uniendo los dos *clusters* (en este caso individuos) más cercanos en función de alguno de los siguientes criterios:

    - **Distancia máxima** entre los elementos de cada *cluster* también conocida como enlace completo o *complete linkage*. Dados dos *clusters A, B* con $x_i \in A$ e $y_j \in B$ los elementos o individuos pertenecientes a los *clusters*, teniendo en cuenta que en la primera iteración cada punto es un *cluster* y por tanto cada *cluster* está formado por un único individuo ($i, j = 1$), y sea d la medida de distancia escogida; se define la distancia máxima entre los *clusters A* y *B* como la distancia entre las observaciones más alejadas $x_i \in A$ e $y_j \in B$:

      $$\max_{i,j} d(x_i, y_j), x_i \in A, y_j \in B$$

      De tal forma que, aunque en las primeras iteraciones se unan los puntos más próximos, en cuanto los *clusters* empiecen a estar formados por más de una observación (*i* o *j* > 1), se buscará minimizar la distancia entre las observaciones más alejadas para cada par de conglomerados.

    - **Distancia mínima** entre los elementos de cada *cluster*, enlace simple o *single linkage*. De manera análoga. Se define la distancia mínima entre los *clusters A* y *B* como la distancia más pequeña para cada par $x_i \in A$ e $y_j \in B$:

      $$\min_{i,j} d(x_i, y_j), x_i \in A, y_j \in B$$

- **Distancia media** entre los elementos de cada *cluster*, enlace promedio o *average linkage*. De manera análoga, se define la distancia media entre los *clusters* A y B como el promedio de todas las distancias entre cada par $x_i \in A$ e $y_j \in B$:

$$\frac{1}{nm}\sum_{i=1}^{n}\sum_{j=1}^{m} d(x_i, y_j)$$

- **Distancia Ward** (Ward Jr, 1963): una de las distancias más populares en *clustering*, tiene en cuenta la varianza dentro de cada grupo y busca minimizar el incremento de esta al combinar dos grupos en uno solo. Comienza considerando las distancias entre *clusters* (recordemos que en la primera iteración cada punto es un *cluster*) como el cuadrado de la distancia euclídea y, en cada iteración se busca el par de *clusters* con cuya unión se minimice esta varianza.

De manera iterativa se van uniendo los *clusters* siguiendo uno de los anteriores criterios. El investigador puede establecer una distancia máxima o un número de *clusters* mínimo para finalizar el proceso. Podemos apreciar este proceso en el siguiente dendograma (*Ilustración 7*).

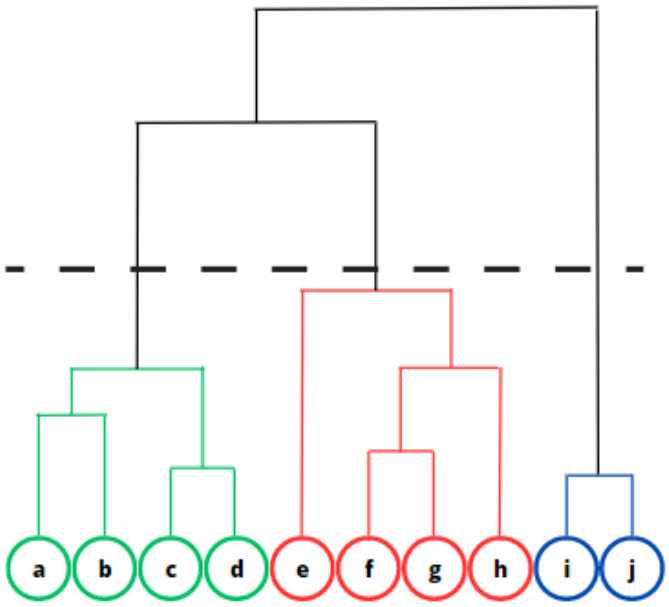

Ilustración 7. Dendograma de cluster jerárquico aglomerativo.
Fuente: elaboración propia en Canva

- **Divisivos**: en contraposición a los algoritmos aglomerativos, se comienza considerando a todos los individuos como un único *cluster* y se van dividiendo sucesivamente en *clusters* cada vez más pequeños atendiendo a criterios como los vistos anteriormente. Estos algoritmos son más complejos que los citados aglomerativos, ya que para cada iteración suelen requerir de un segundo algoritmo que vaya subdividiendo cada *cluster*; aunque suelen ser más eficientes.

Los métodos jerárquicos presentan una limitación y es que una vez la división o fusión de *clusters* se ha realizado, ya no se puede deshacer. Este hecho reduce los costes computacionales,

pero en numerosos casos puede resultar un inconveniente. A continuación, mostramos el resto de las ventajas y desventajas de estos algoritmos.

Entre las ventajas destacamos que no es necesario indicar el número de grupos o *clusters* deseados de antemano y su fácil interpretación visual.

Por otro lado, algunos de los puntos débiles de estos algoritmos son que no son adecuados para grandes conjuntos de datos ya que, aunque se pueden aplicar, al aumentar el número de observaciones, el coste computacional y el tiempo de ejecución también aumentan considerablemente.

3.2.2) *Clustering* particional

Estos algoritmos se basan en la especificación de un número inicial de grupos o *clusters* (diferenciándose así del *clustering* jerárquico), y en la reasignación iterativa de observaciones, individuos o puntos entre los grupos hasta la convergencia. Además, estos algoritmos suelen determinar todos los grupos a la vez. Destacan principalmente dos algoritmos de este tipo:

K-Means

Este algoritmo, desarrollado también durante la década de los 60 (MacQueen, 1967), describe una técnica para clasificar observaciones en *k clusters*, donde *k* es un número predeterminado de grupos.

La idea general del algoritmo es asignar cada punto u observación al *cluster* con el centro, denominado centroide, más cercano. Este centroide es la media de todos los puntos pertenecientes al *cluster*, es decir, sus coordenadas se corresponden con la media aritmética para cada variable por separado sobre todos los puntos del *cluster* y por eso el algoritmo recibe su nombre traducido como k-medias.

Así pues, el desarrollo general del algoritmo sería el siguiente (*Ilustración 8*):

1. Elegir el número *k* de *clusters*.

2. Seleccionar aleatoriamente *k* observaciones del conjunto de datos como los centroides iniciales. Algunas variaciones del algoritmo seleccionan puntos aleatorios del espacio de las variables como centroides, sin necesidad de que estos sean observaciones.

3. Asignar cada observación al *cluster* cuyo centroide sea el más cercano.

4. Calcular los nuevos centroides para cada *cluster* como la media de todas las observaciones pertenecientes a cada *cluster.*

5. Repetir los pasos 3 y 4 hasta la convergencia, es decir, hasta que los centroides no cambien de una iteración a otra o se cumpla algún criterio de tolerancia.

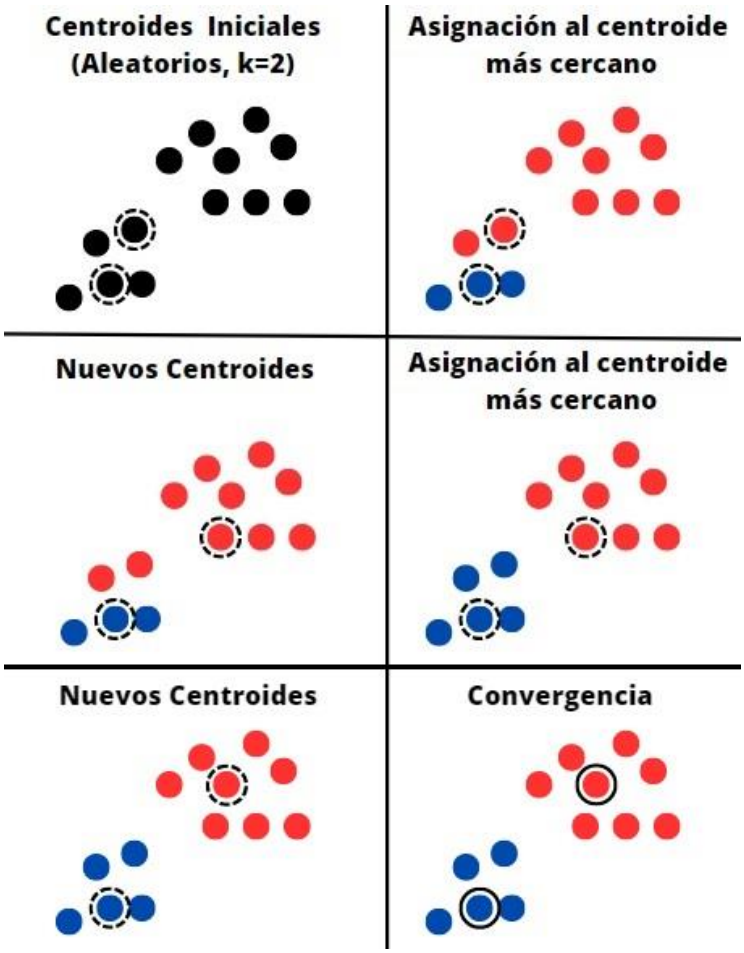

Ilustración 8. K-Means para k=2. Fuente: elaboración propia en Canva.

Debido a su simplicidad y bajo coste computacional, tanto en tiempo de ejecución como en memoria utilizada, este algoritmo es uno de los más populares no sólo dentro del *clustering*, sino en todo el campo del ML. No obstante, también presenta ciertos inconvenientes como son la elección del número de *clusters* (al igual que el resto de los algoritmos de este tipo) y, más en particular, este algoritmo es sensible a la elección inicial de los centroides, pudiéndose ver afectado el resultado final por una decisión aleatoria. Además, debido a su naturaleza, tampoco es adecuado para tratar datos categóricos, aunque hay alternativas para dichos casos.

K-Medoids

Este algoritmo (Kaufmann, 1987) es muy similar al *k-means* tanto en su objetivo final cómo en su funcionamiento. Su principal diferencia reside en la forma en la que se seleccionan los centros de los *clusters*. En lugar de seleccionar los centroides (medias aritméticas), el *k-medoids* selecciona los medoides como centros de los *clusters*.

Un medoide es una observación de un *cluster* que minimiza la suma de las distancias del resto de observaciones al mismo *cluster*. Es decir, un medoide es el objeto que se encuentra más

cerca de todos los demás objetos en el *cluster*, siendo así el objeto más representativo de dicho *cluster*. Por tanto, el pseudo-código del algoritmo sería el siguiente:

1. Elegir el número *k* de *clusters*.

2. Seleccionar aleatoriamente *k* observaciones del conjunto de datos como los medoides iniciales.

3. Asignar cada observación al *cluster* cuyo medoide sea el más cercano.

4. Calcular los nuevos medoides para cada *cluster*. Dentro de un *cluster*, el medoide será la observación que minimiza la suma de las distancias a todas las demás observaciones dentro de ese mismo cluster.

5. Repetir los pasos 3 y 4 hasta la convergencia, es decir, hasta que los medoides se mantengan iguales de una iteración a otra o se cumpla algún criterio de tolerancia.

Este algoritmo es más robusto que el *k-means* a la hora de trabajar con valores atípicos, ruidosos u *outliers* ya que, por construcción, los medoides son menos sensibles a estos valores que los centroides. Por otra parte, el *k-medoids* es más lento y presenta un mayor coste computacional que *el k-means* al ser un método más complejo.

3.2.3) *Clustering* basado en densidad

Este tipo de *clustering* identifica *clusters* en regiones densas de objetos, separadas por regiones de baja densidad. La idea es, a partir de un umbral mínimo de densidad, agrupar los puntos u observaciones que se encuentren en regiones del espacio de datos que superen dicho umbral, dando lugar así a *clusters* con formas y tamaños arbitrarios. Es decir, cada *cluster* se corresponderá con una región del espacio de datos que supere el citado umbral y por todas las observaciones que se encuentren en dicha región. El DBSCAN es el principal algoritmo de *clustering* de este tipo.

DBSCAN

El nombre de este algoritmo (Ester et al., 1996) es el acrónimo de *Density-Based Spatial Clustering of Applications with Noise*, es decir, *clustering* espacial basado en la densidad de aplicaciones con ruido.

En este algoritmo, las regiones con una alta densidad de puntos indican la existencia de *clusters*, mientras que las regiones con una baja densidad de puntos indicarían clusters de ruido o clusters de valores atípicos. Así pues, la idea clave es que, para cada punto de un *cluster*, la vecindad de un radio dado tiene que contener al menos un número mínimo de puntos. Por lo tanto, este algoritmo necesita de al menos dos hiper-parámetros fijados de antemano:

- Radio (*r*): longitud que delimita el área, región o vecindario para cada punto.

- Puntos mínimos *(MinP):* número mínimo de puntos u observaciones que deben existir en una región o vecindario para que sea considerada un *cluster*.

Una vez explicados estos conceptos, el desarrollo del algoritmo sería el siguiente:

1. Seleccionar aleatoriamente un punto inicial o semilla perteneciente al conjunto de datos.

2. Identificar todos los puntos en su vecindario, es decir, a una distancia menor o igual que su radio r.

3. Recontar el número de puntos n_1 en dicho vecindario. Si supera o iguala el umbral ($n_1 \geq MinP$), consideramos el punto inicial como un punto central y su región como un nuevo *cluster*.

4. Para cada punto perteneciente al nuevo *cluster*, se deben repetir los pasos 2 y 3. Los puntos pertenecientes a este *cluster* que no superen el umbral ($n_i < MinP$) serán considerados puntos límite o borde.

5. Seleccionar un nuevo punto semilla.

6. Repetir los pasos del 2 al 5 hasta haber estudiado todos los puntos del conjunto de datos. Todos los puntos que no formen parte de un *cluster* serán considerados ruido (*Ilustración 9*).

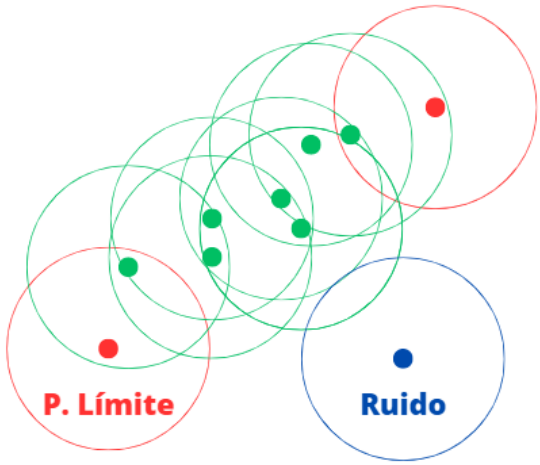

Ilustración 9. DBSCAN. Fuente: elaboración propia con Canva.

Como podemos deducir, los valores de los hiper-parámetros *r* y *MinP* son de vital importancia para poder formar *clusters* significativos y que no haya exceso de ruido. Esta sería una de sus principales desventajas, ya que el resultado final dependerá en gran medida de estos valores y su elección muchas veces no resulta sencilla. Además, no funciona bien para todos los conjuntos de datos, viéndose alterado por la distribución o la dimensionalidad de estos.

Por otra parte, entre sus principales ventajas destaca su eficiencia tanto en tiempo como en memoria, especialmente para grandes conjuntos de datos; su robustez, ya que no es sensible al ruido y que no requiere fijar el número de *clusters* de antemano.

3.2.4) *Clustering* basado en modelo

Este tipo de *clustering* presenta un enfoque estadístico para agrupar las observaciones en función de su similitud. Su funcionamiento subyace del supuesto de que los datos son generados por una mezcla finita de distribuciones de probabilidad, de tal forma que cada subconjunto de dicha mezcla representa un *cluster*. Es decir, se asume que los datos observados son una combinación de múltiples subpoblaciones, cada una de las cuales sigue una distribución diferente y configura un *cluster* distinto.

El objetivo de estos algoritmos es encontrar el número óptimo de *clusters*, así como las distribuciones que mejor se ajustan a las observaciones. Para lograr dicho objetivo, se debe maximizar una función de verosimilitud que describa la probabilidad de observar los datos dados los parámetros del modelo. De entre todos los algoritmos basados en modelo, destaca el MCLUST (Fraley & Raftery, 2002).

3.2.5) *Clustering* en cuadrículas (*Grid-Based*)

El último subtipo de *clustering* que se va a explicar es el basado en cuadrículas o *Grid-Based*. La idea general de estos algoritmos es dividir el espacio de observaciones en un conjunto finito y homogéneo de cuadrículas o celdas. Los *clusters* se formarán en función de la densidad y de la proximidad de aquellas cuadrículas que contengan observaciones.

Si trabajamos con observaciones sobre las que se han medido dos variables numéricas, resulta fácil proyectar los puntos sobre los ejes de abscisas y ordenadas y, posteriormente, dividir este espacio en una rejilla formada por cuadrados homogéneos. Una vez representada la rejilla, así como los puntos que se encuentran en cada celda, existen distintos algoritmos de este tipo con distintas maneras de operar. Una idea general, común para casi todos estos algoritmos, sería agrupar las celdas adyacentes que superen un determinado umbral mínimo de densidad para formar distintos *clusters*. Dada su naturaleza, todos estos algoritmos son especialmente útiles para trabajar con datos geoespaciales, formando *clusters* sobre mapas.

3.2.6) Métricas de evaluación y selección del número de *clusters*

Una vez aplicados uno o varios de los algoritmos que hemos visto en esta sección sobre un determinado conjunto de datos, en la mayoría de los casos debemos decidir cuantos *clusters* compondrán nuestro modelo finalmente. Existen algunos algoritmos como el MCLUST que ya incluyen un paso en el propio código para decidir el número óptimo de *clusters*. Otros, como el *K-Means*, precisan indicar el número de *clusters k* previamente, aunque podemos implementar estos algoritmos varias veces modificando el valor de *k* y posteriormente comparar los

resultados. En general, para casi todos los algoritmos de *clustering*, debemos utilizar distintas técnicas y métricas para determinar cuál es el número de *clusters* óptimo en cada caso, así como para evaluar la calidad de las agrupaciones. Es conveniente utilizar varias métricas a la hora de evaluar la calidad de los *clusters* ya que no existe una sola métrica universalmente aceptada para esta tarea y, dependiendo de cada caso concreto, priorizaremos distintas características de los *clusters*. Algunas de estas métricas son:

- **Coeficiente de *Silhouette*** (Rousseeuw, 1987): mide la similitud de cada observación con su propio *cluster* en comparación con el resto de *clusters*. Este valor varía entre -1 y 1, de tal forma que un valor cercano a 1 indica que el punto está asignado correctamente, un valor de 0 indicaría indiferencia y un valor negativo indicaría que la observación estaría mejor agrupada en otro *cluster*. La fórmula de esta métrica, para un punto *x* es la siguiente:

$$s(x) = \frac{d_{ext}(x) - d_{int}(x)}{max(d_{int}, d_{ext})}$$

  donde $d_{int}(x)$ sería la distancia media entre el punto x y el resto de los puntos de su *cluster*, mientras que $d_{ext}(x)$ es la distancia media entre el punto x y los puntos del *cluster* más cercano al que no pertenezca.

- **Inercia (SSE)**: también conocida como suma de cuadrados interna o del error, se utiliza principalmente para evaluar el *k-means* o el *k-medoids* ya que, mide la suma de las distancias al cuadrado entre cada punto y el punto central (centroide, medoide…) de su *cluster* correspondiente. Un valor bajo de esta métrica indicará que los puntos están más cerca de su centroide, es decir, los *clusters* serán más compactos. Elegida una medida de distancia (*d*) y siendo *k* el número de *clusters*, su fórmula es la siguiente:

$$SSE = \sum_{i=1}^{k} \sum_{x \in C_i} d^2(m_i, x)$$

  Donde *x* es un unto perteneciente al *cluster* C_i cuyo punto central es m_i con $i = 1, \cdots, k$

- **Separación (SSB):** se corresponde con otra suma de cuadrados, pero en este caso nos indicará como de lejos o cuan separados están unos *clusters* de otros. De nuevo elegida una medida de distancia (*d*) y siendo *k* el número de *clusters*, su fórmula sería la siguiente:

$$SSB = \sum_{i=1}^{k} n_i \cdot d^2(m_i, \bar{x})$$

  Con n_i el número de elementos pertenecientes al *cluster i*, m_i el centroide de dicho *cluster* y $\bar{x}$ la media o punto medio del conjunto total de datos.

- **Coeficiente de Calinski-Harabasz** (Caliński & Harabasz, 1974)**:** esta métrica propuesta por T. Calinski y J. Harabasz en 1974, es una fusión de las dos anteriores. Mide la dispersión intra *cluster* y la dispersión entre los distintos *clusters*, de tal forma que un valor alto de este coeficiente indica una mejor calidad de *clustering*. Atiende a la siguiente fórmula.

$$CH = \frac{SSB/(k-1)}{SSE/(n-k)}$$

donde *n* representa el número total de observaciones y *k* el de *clusters*.

- **Índice de Davies-Bouldin (DBI)** (Davies & Bouldin, 1979)**:** evalúa la compacidad y separación de los *clusters*, de tal forma que un valor bajo de esta métrica indicaría calidad en el agrupamiento. Para *k clusters*, su fórmula es la siguiente:

$$DBI = \frac{1}{k} \sum_{i=1, i \neq j}^{k} max\left(\frac{\bar{d}_i + \bar{d}_j}{d(m_i, m_j)}\right)$$

Con $\bar{d}_i, \bar{d}_j$ la distancia media entre cada punto del *cluster i* o *j* respectivamente a su centroide; m_i y m_j dichos centroides y $d(m_i, m_j)$ la distancia entre ambos.

Así pues, en función de los valores obtenidos en estas métricas podremos decidir si el ajuste de nuestros modelos es bueno y podremos compararlos entre ellos. Para decidir el número de *clusters* si este no se conoce previamente, habitualmente se comienza empleando modelos aglomerativos y observando sus dendogramas. Los modelos que presenten dendogramas que presenten un número razonable de conglomerados y estos no parezcan muy descompensados (es decir, que la mayoría de las observaciones no pertenezcan a un único *cluster*), podrán servir como indicativo. Podemos evaluar y comparar estos modelos con las anteriores métricas para decidir cual es el número apropiado de agrupaciones. Una vez decidido este valor *k*, podemos comprobar si para otro tipo de modelos como los particionales, también conseguimos los mejores resultados evaluando estos algoritmos sobre un rango de valores entre los que se encuentre nuestro valor *k*.

4) Aprendizaje Supervisado

En este capítulo, desarrollaremos detalladamente las principales técnicas y algoritmos de este tipo de aprendizaje. Como habíamos visto anteriormente el aprendizaje supervisado se caracteriza por trabajar con conjuntos de datos etiquetados, es decir, en los que existe una variable respuesta (denominada habitualmente por la letra *Y*) cuyo valor buscamos predecir con la mayor precisión y exactitud posibles (Singh et al., 2016). Dependiendo de si esta variable *Y* toma un valor numérico o categórico, emplearemos algoritmos de regresión o de clasificación respectivamente. Por ejemplo, para predecir el precio de una casa utilizaríamos algoritmos de regresión en función del resto de características del inmueble; mientras que para determinar si dicha casa se venderá o no, deberíamos utilizar algoritmos de clasificación binaria, es decir con dos posibles valores para la variable respuesta (Sí/No).

A diferencia de los algoritmos de aprendizaje no supervisado vistos anteriormente, todos estos algoritmos deberán ser entrenados previamente antes de poder evaluar su desempeño sobre conjuntos de datos nuevos. Para llevar a cabo este entrenamiento y la posterior evaluación, lo que se hace habitualmente es dividir (de manera aleatoria o teniendo en cuenta la distribución de la variable respuesta) el conjunto total de datos en dos subconjuntos: un subconjunto de entrenamiento y otro de test. Un porcentaje de división comúnmente aceptado es en torno al 70-80% para el subconjunto de entrenamiento frente al 30-20% restante para el test, aunque dependiendo del tamaño del conjunto de datos estos porcentajes pueden variar enormemente.

La idea general para todos los modelos que emplean algoritmos de aprendizaje supervisado es utilizar el subconjunto de entrenamiento y sus correspondientes variables respuesta (ya sean numéricas o categóricas) para "enseñarles" a predecirlas, de tal modo que generalmente cuantos más datos etiquetados tengamos en este subconjunto, mejor aprenderá nuestro modelo. Durante esta fase de entrenamiento el modelo generará una respuesta $\hat{Y}$ que será comparada con la real *Y.* El objetivo de estos algoritmos es ir minimizando la diferencia entre ambas (o error) en cada iteración. Una vez nuestro modelo ya esté entrenado y pueda predecir con precisión y exactitud las variables respuesta del conjunto de entrenamiento, debemos evaluar su desempeño sobre datos que no ha visto anteriormente, es decir, prediciendo los valores de la variable respuesta sobre el subconjunto de test. Para evaluar la exactitud, precisión y veracidad de los modelos estudiados, emplearemos diversas métricas distintas, aunque una primera aproximación será comparar los valores reales de la variable respuesta (Y) frente a los valores predichos ($\hat{Y}$). Siempre debemos emplear un subconjunto de test con datos no utilizados anteriormente para evaluar nuestros modelos, ya que de no ser utilizado corremos peligro de que se produzca *"overfitting*" o sobre ajuste y, que un modelo que creemos que funciona bien, solo ofrezca buenos resultados sobre un mismo conjunto de datos, reduciendo por completo su utilidad (*Ilustración 10*).

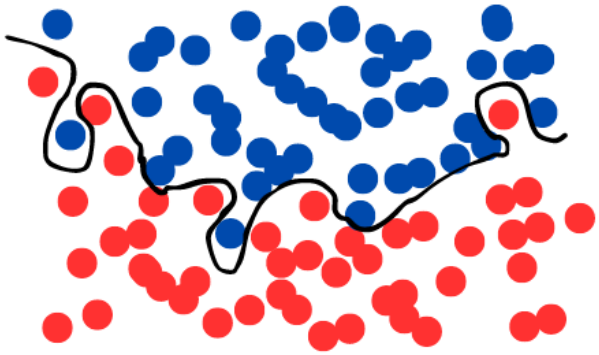

Ilustración 10. Clasificador binario con sobreajuste u overfitting. Fuente: elaboración propia en Canva.

Además, la inmensa mayoría de algoritmos de aprendizaje supervisado, poseen una serie de hiper-parámetros, como podía ser el radio (r) en el DBSCAN, cuyos valores deben ser escogidos manualmente. En muchos casos, como veíamos en los algoritmos de *clustering*, la elección de estos valores condiciona considerablemente el resultado final. Es por este motivo por el que, en lugar de la división del conjunto total de datos en subconjuntos de test y entrenamiento, habitualmente se añade un tercer subconjunto, conocido como conjunto de validación, para evaluar distintas configuraciones de hiper-parámetros. En este caso, un porcentaje de división aceptable sobre el conjunto total de datos podría ser del 60% para el subconjunto de entrenamiento, un 20% para el test y otro 20% para el subconjunto de validación; aunque de nuevo estos porcentajes son solo orientativos y dependerá de cada caso. La idea general es entrenar un modelo bajo distintas configuraciones de hiper-parámetros sobre el subconjunto de entrenamiento, creando por tanto distintos modelos (uno por cada configuración distinta). Una vez entrenados estos modelos, evaluamos sus distintos desempeños sobre el conjunto de validación y seleccionamos el modelo con la configuración de hiper-parámetros que mejores resultados obtenga. Por último, evaluamos este modelo final sobre el conjunto de test. Otra práctica muy extendida es entrenar el modelo final sobre el conjunto de entrenamiento más el de validación para mejorar así sus resultados.

Una vez explicadas estas características propias del aprendizaje supervisado, pasemos a estudiar sus principales algoritmos, agrupados en dos grandes grupos: regresión y clasificación.

4.1) Regresión

Los modelos de regresión buscan predecir una variable respuesta cuantitativa en función del resto de variables, también cuantitativas. Dependiendo del número de variables explicativas o regresoras del modelo trabajaremos con modelos de regresión simple (una sola variable explicativa) o regresión múltiple (más de una variable explicativa). Aunque existen distintos algoritmos de regresión, a en este trabajo nos centraremos en describir la regresión lineal, el modelo más simple que representa la relación de las variables en forma de línea recta. Comenzaremos explicando el funcionamiento del modelo simple para después generalizarlo al múltiple.

4.1.1) Regresión lineal – Enfoque clásico

Consideremos dos variables cuantitativas *X* e *Y*. El modelo muestral (Montgomery et al., 2021) que explica el comportamiento de *Y* frente a *X* es:

$$Y_i = \hat{\beta}_0 + \hat{\beta}_1 X_i + \hat{u}_i$$

Donde $i = 1, \dots, n$ representa el número de observaciones; $\hat{\beta}_0, \hat{\beta}_1$ los coeficientes del modelo y $\hat{u}_i$ el residuo o término aleatorio. El objetivo es construir un modelo de predicción.

$$\hat{Y}_i = \hat{\beta}_0 + \hat{\beta}_1 X_i$$

Para estimar el valor de los coeficientes, buscaremos minimizar la suma de los cuadrados de los residuos que se corresponden con la diferencia entre el valor real (*Y)* y el esperado ($\hat{Y}$). Elevamos los residuos al cuadrado para evitar los posibles efectos derivados de sus signos. Este enfoque se conoce como mínimos cuadrados ordinarios.

$$\min G = \min \sum_{i=1}^{n} {\hat{u}_i}^2 = \min \sum_{i=1}^{n} \left(Y_i - \hat{Y}_i\right)^2 = \min \sum_{i=1}^{n} \left(Y_i - \hat{\beta}_0 - \hat{\beta}_1 X_i\right)^2$$

Derivando respecto de los coeficientes e igualando a 0, obtenemos las ecuaciones normales desde las que podemos deducir el valor de los coeficientes.

$$0 = \frac{\partial G}{\partial \beta_0} = -2 \sum_{i=1}^{n} \left(Y_i - \hat{\beta}_0 - \hat{\beta}_1 X_i\right)$$

$$0 = \frac{\partial G}{\partial \beta_1} = -2 \sum_{i=1}^{n} \left(Y_i - \hat{\beta}_0 - \hat{\beta}_1 X_i\right) X_i$$

Desarrollando la primera ecuación deducimos el valor de $\hat{\beta}_0$

$$0 = \sum_{i=1}^{n} Y_i - n\hat{\beta}_0 - \hat{\beta}_1 \sum_{i=1}^{n} X_i$$

De donde finalmente despejando obtenemos

$$\hat{\beta}_0 = \frac{\sum_{i=1}^{n} Y_i - \hat{\beta}_1 \sum_{i=1}^{n} X_i}{n} = \frac{\sum_{i=1}^{n} Y_i}{n} - \hat{\beta}_1 \frac{\sum_{i=1}^{n} X_i}{n} = \overline{Y} - \hat{\beta}_1 \overline{X}$$

Donde $\overline{\mathrm{X}}, \overline{\mathrm{Y}}$ representan las medias muestrales de las variables *X, Y*. Por otro lado, desarrollando la segunda ecuación normal deducimos

$$0 = \sum_{i=1}^{n} Y_i X_i - \hat{\beta}_0 \sum_{i=1}^{n} X_i - \hat{\beta}_1 \sum_{i=1}^{n} X_i^2$$

Y despejando obtenemos

$$\hat{\beta}_1 = \frac{\sum_{i=1}^{n} Y_i X_i - \hat{\beta}_0 \sum_{i=1}^{n} X_i}{\sum_{i=1}^{n} X_i^2}$$

Dividiendo por *n* y sustituyendo en la anterior ecuación el valor de $\widehat{\beta_0}$ obtenemos

$$\hat{\beta}_1 = \frac{\overline{XY} - \hat{\beta}_0\overline{X}}{\overline{X^2}} = \frac{\overline{XY} - \left(\overline{Y} - \hat{\beta}_1\overline{X}\right)\overline{X}}{\overline{X^2}} = \frac{\overline{XY} - \bar{Y}\bar{X} + \hat{\beta}_1\overline{X}^2}{\overline{X^2}}$$

Y, por último, despejando

$$\hat{\beta}_1 = \frac{\overline{XY} - \bar{Y}\bar{X}}{\overline{X^2} - \overline{X}^2} = \frac{S_{XY}}{S_X^2}$$

Donde S_{XY} es la covarianza entre las dos variables y S_X^2 la varianza de la variable independiente. Generalizando estos resultados para el caso de la regresión lineal múltiple con k variables independientes, la forma matricial del modelo muestral sería

$$Y = X\beta + u$$

con:

$$Y = \begin{pmatrix} Y_1 \\ \vdots \\ Y_n \end{pmatrix},\ X = \begin{pmatrix} 1 & x_{1,1} & \dots & x_{1,k} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n,1} & \dots & x_{n,k} \end{pmatrix},\ \beta = \begin{pmatrix} \beta_0 \\ \vdots \\ \beta_k \end{pmatrix},\ u = \begin{pmatrix} u_1 \\ \vdots \\ u_n \end{pmatrix}$$

Aplicando de nuevo mínimos cuadrados obtenemos

$$\min G = \min u^T u = \min(Y - X\beta)^T (Y - X\beta)$$

Derivando e igualando a 0 obtenemos las ecuaciones normales para la regresión lineal múltiple, cuya forma matricial es

$$0 = \frac{\partial G}{\partial \beta} = -2X^T Y + 2X^T X\beta$$

Luego finalmente despejando obtenemos

$$\hat{\beta} = (X^T X)^{-1} X^T Y$$

Utilizamos el método de mínimos cuadrados para calcular los estimadores β ya que, cumpliendo una serie de hipótesis previas, el teorema de Gauss-Markov nos garantiza que estos estimadores son óptimos, es decir, insesgados y de varianza mínima. Las citadas hipótesis que debe cumplir el modelo son las siguientes:

- **Linealidad**: la variable dependiente depende linealmente de las independientes.

$$Y = X\beta + u$$

- **Homocedasticidad**: todos los residuos o errores tienen la misma varianza.

$$Var(u_i) = \sigma^2, \forall i = 1, \dots, n$$

- **Independencia**: los residuos son independientes entre sí.

$$Cov(u_i, u_j) = 0, \forall i \neq j$$

- **Normalidad**: los residuos siguen una distribución normal.

$$u \sim N(0, \sigma^2)$$

Aunque a la hora de trabajar con casos reales, muchas veces no se cumplan todas las hipótesis, es importante tenerlas en cuenta para poder encontrar los coeficientes óptimos. (*Ilustración 11*)

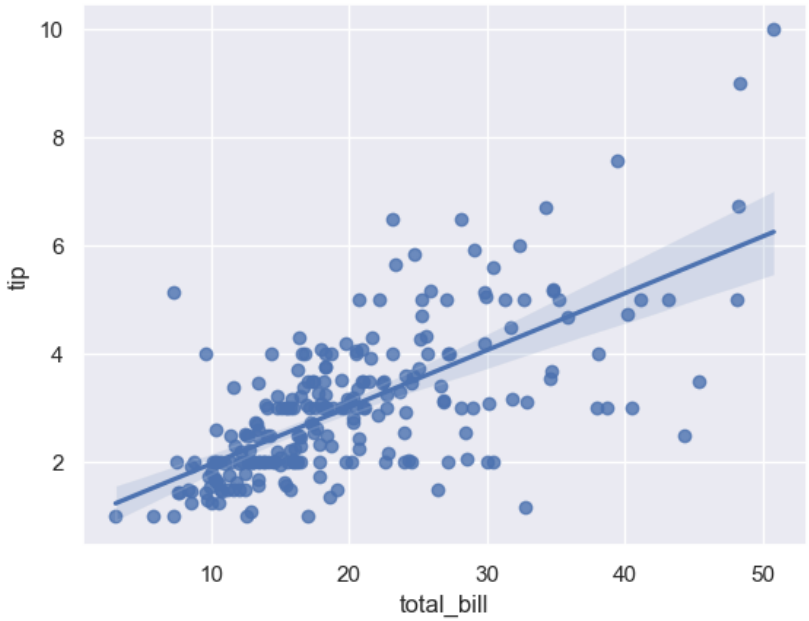

Ilustración 11. Gráfico de regresión lineal simple. Fuente: elaboración propia en Python a partir del siguiente dataset https://github.com/mwaskom/seaborn-data/blob/master/tips.csv

4.1.2) Regresión Lineal - Enfoque del ML

Aunque hemos visto el enfoque clásico de la regresión para "ajustar el modelo", en ML el enfoque consiste en ir "entrenando" el modelo con cada iteración y observación. La idea es la misma, elegimos una función de coste L y buscamos minimizarla. Para el modelo simple, sin pérdida de generalidad, tenemos:

$$y = h(x) = \theta_0 + \theta_1 x$$

Elegimos el MSE o error cuadrático medio como función de coste L

$$L(\theta_0, \theta_1) = \frac{1}{2n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2 = \frac{1}{2n}\sum_{i=1}^{n}(h(x_i) - y_i)^2$$

Buscamos los valores de θ_0, θ_1 que minimicen la función de coste L. Para ello, aplicaremos un algoritmo muy popular en ML y que podremos utilizar en otros casos aparte de la regresión: el **descenso de gradiente**.

Descenso de Gradiente

El descenso de gradiente es un proceso de optimización iterativo que busca minimizar la función de coste elegida y, por tanto, la diferencia entre el valor real y el predicho. La idea original de este algoritmo para encontrar el mínimo de una función ya fue desarrollada por Cauchy en el siglo XIX, aunque su aplicación al ML no sería hasta el siglo siguiente (McCulloch & Pitts, 1943). Un factor muy importante que debemos tener en cuenta en este algoritmo es el valor del hiper-parámetro α, que designa la tasa de aprendizaje o ***learning rate***. Este hiper-parámetro con valor constante positivo, será el encargado de regular la velocidad de aprendizaje del modelo. Es de vital importancia ajustar su valor correctamente, ya que si elegimos un valor demasiado pequeño el modelo necesitará demasiadas iteraciones para minimizar el error, aumentando así el coste computacional; mientras que si el valor es demasiado grande nos "pasaremos de largo" y no se alcanzará el mínimo como vemos en la siguiente figura. El pseudocódigo del algoritmo es el siguiente (*Ilustración 12*):

1. Se asigna un valor aleatorio a los parámetros θ_0, θ_1, por ejemplo (0,0).

2. Se calcula la derivada parcial de la función de coste L con respecto a cada uno de los parámetros y se construye el vector gradiente.

$$\nabla_\theta L = \left[\frac{\partial L}{\partial \theta_0}, \frac{\partial L}{\partial \theta_1}\right]$$

3. Se actualizan a la vez los valores de los parámetros en la dirección contraria al valor de la derivada parcial para minimizar el error.

$$\Delta\theta = -\alpha \cdot \nabla_\theta L = -\alpha \cdot \left[\frac{\partial L}{\partial \theta_0}, \frac{\partial L}{\partial \theta_1}\right]$$

$$\theta = (\theta_0, \theta_1) \coloneqq \theta + \Delta\theta$$

4. Se repiten los pasos 2 y 3 hasta la convergencia, es decir, hasta que se alcanza un mínimo.

Generalizando el descenso de gradiente para cualquier algoritmo con parámetros

$\theta = (\theta_0, \dots, \theta_n)$, debemos actualizar los parámetros θ_j de forma simultánea y reiterar el proceso hasta convergencia bajo la siguiente fórmula:

$$\theta_j \coloneqq \theta_j - \alpha \frac{\partial}{\partial \theta_j} L(\theta_0, \dots, \theta_n)$$

Este algoritmo nos garantiza convergencia a un mínimo, pero dependiendo de los valores iniciales que asignemos a los parámetros θ, este puede ser un mínimo local y no optimizar correctamente la función. En la práctica, trabajando con modelos con múltiples variables, lo que se suelen dar es puntos de silla y, aunque ya no se pueda descender en una de las variables, si se podrá en el resto, llegando casi siempre a una solución óptima.

En líneas generales, cuando contamos con pocas variables k, es preferible el enfoque clásico utilizando las ecuaciones normales para calcular los estimadores. Sin embargo,

computacionalmente es mucho más lento y complejo, sobre todo si contamos con muchas variables debido principalmente al cálculo de la inversa $(X^T X)^{-1}$.

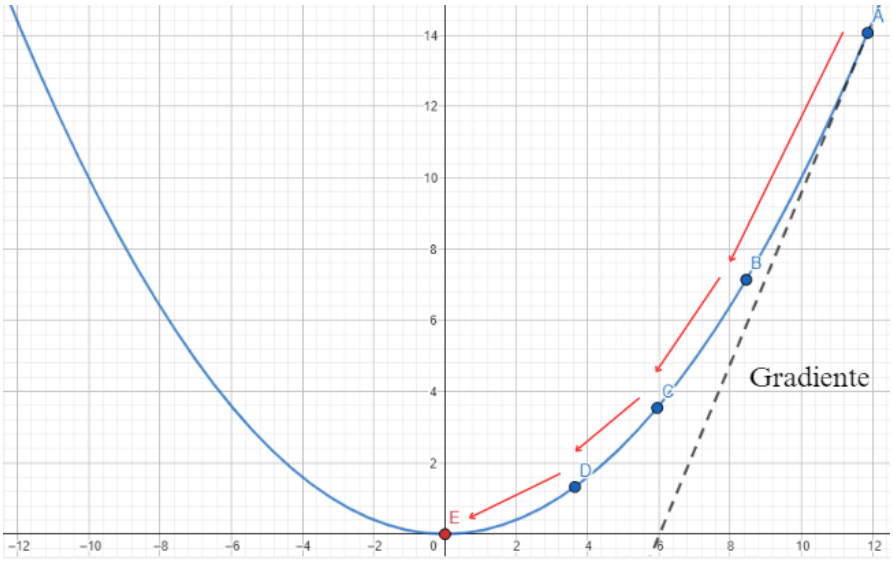

Ilustración 12. Descenso de gradiente. Fuente: elaboración propia con Canva y Geogebra. basado en imagen de IArtificial.net.

4.1.3) Métricas de evaluación

Las principales métricas para evaluar si las predicciones de un modelo de regresión $(\hat{Y})$ se ajustan y se aproximan a los valores reales (Y) son las siguientes:

- **Coeficiente de determinación** R^2: este coeficiente indica cuánta varianza de la variable dependiente (Y) puede ser explicada por el modelo de regresión lineal. El coeficiente de determinación se calcula como la proporción de la variabilidad total de la variable dependiente que es explicada por el modelo.

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

  donde SC_{Res} representa la suma de cuadrados de los residuales, es decir, la diferencia entre los valores reales u observados y los predichos; y SC_{Tot} representa la suma de cuadrados total, es decir, la diferencia entre los valores observados y su media $(\bar{y})$.

  El coeficiente de determinación R^2 puede tomar valores entre 0 y 1. De tal forma que, un valor igual a 0 indica que el modelo no explica ninguna variación en la variable dependiente, mientras que un valor de 1 indicaría que el modelo explica toda la variación en la variable dependiente. Aunque siempre es preferible un valor de R^2 alto, ya que indicará que el modelo se ajusta bien a los datos, esto no refleja necesariamente un buen valor predictivo del modelo.

- **Error cuadrático medio** *(MSE)***:** es una de las métricas más utilizadas en regresión lineal y se define como el promedio de los errores cuadrados entre los valores observados y los valores predichos por el modelo. Su fórmula es la siguiente:

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

  Esta métrica mide la diferencia entre los valores observados y los predichos, por tanto, cuanto menor sea su valor, mejor se ajustará nuestro modelo a los datos, aunque de nuevo esto no es suficiente indicador para saber si la predicción de valores nuevos será la adecuada. Además, esta medida es sensible a los *outliers* o valores extremos.

- **Raíz del error cuadrático medio** *(RMSE):* se corresponde con la raíz de la anterior métrica. Se suele utilizar la raíz cuadrada, en lugar *MSE*, ya que es más fácil de interpretar, pues el *RMSE* se mide en las mismas unidades que la variable dependiente *Y*. De manera análoga, se busca minimizar su valor.

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

- **Error absoluto medio** (*MAE*): similar al *MSE*, esta métrica tiene en cuenta las diferencias en valor absoluto entre valores reales y predichos, en lugar de los cuadrados de tal forma que es menos sensible a los valores atípicos. Atiende a la siguiente fórmula:

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

4.2) Clasificación

Definimos los algoritmos de clasificación como aquellos algoritmos de aprendizaje supervisado utilizados para predecir una clase o categoría de una variable dependiente cualitativa. Es decir, como habíamos visto anteriormente, los modelos que empleen estos algoritmos serán entrenados con un conjunto de datos etiquetados en los que la variable respuesta será de tipo cualitativo. Si esta variable respuesta es dicotómica, es decir que sólo acepta dos categorías, trabajaremos con algoritmos de clasificación binaria. Por otro lado, si la variable acepta más de dos posibles respuestas, la clasificación será multinomial. Un ejemplo de clasificación binaria podría ser predecir si un tumor es maligno o benigno en función de sus características, mientras que un ejemplo de clasificación multinomial sería predecir el estadio de dicho tumor (5 posibles modalidades, del 0 al 4).

La clasificación es el área dentro del ML en la que se han desarrollado (y se siguen desarrollando) más algoritmos. En esta sección, describiremos tan sólo algunos de los más populares.

4.2.1) Regresión logística

La regresión logística es una extensión de la regresión lineal para el caso de variables dependientes cualitativas binarias, aunque puede ser también utilizada en problemas de clasificación múltiples. Esta popular técnica fue desarrollada en la década de 1940 (Berkson, 1944) y extendida en años posteriores (Cox, 1958).

Considerando que la variable respuesta *Y* admita dos posibles modalidades, de tal forma que tomando el ejemplo del tumor estos serían maligno y benigno, podemos codificar rápidamente la variable respuesta para que tome los valores 0 (benigno) y 1 (maligno). El uso de la regresión lineal para predecir este tipo de problemas no sería adecuado ya que se vería muy afectado por los *outliers* y predeciría respuestas fuera del rango, como podemos ver en la figura (*Ilustración 13*).

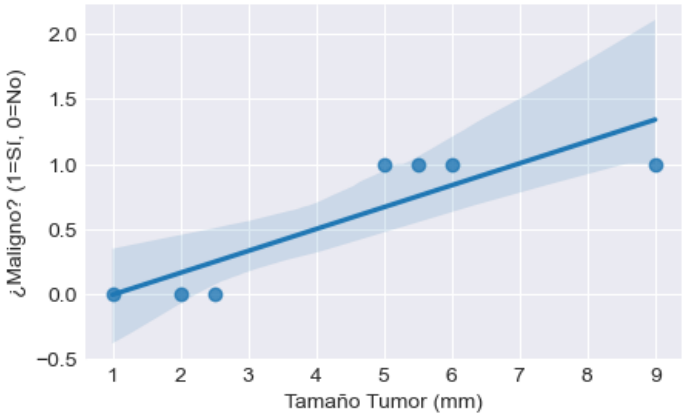

Ilustración 13. Regresión lineal aplicada a problemas de clasificación. Fuente: elaboración propia con Python y dataset propio.

Por este motivo, se buscó, a partir de los conceptos básicos de la regresión lineal, una curva que se ajuste mejor a este tipo de problemas y que permita predecir probabilidades, es decir, salidas en el intervalo [0,1]. La función capaz de llevar a cabo esta tarea es la función sigmoide o logística que tiene dominio $\mathbb{R}$, es decir, acepta cualquier valor de entrada y recorrido o salidas concentradas en el intervalo [0,1] (*Ilustración 14*). Su fórmula es la siguiente:

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

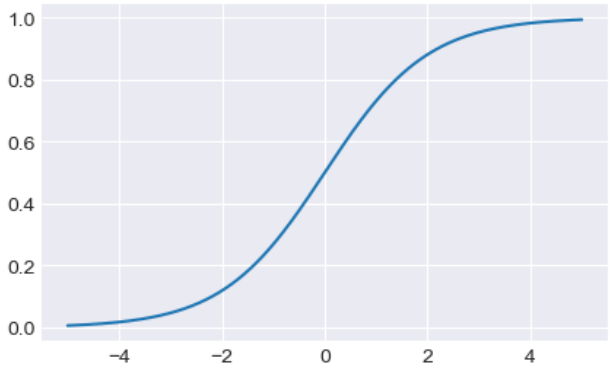

Ilustración 14. Función sigmoide. Fuente: elaboración propia con Python.

Uniendo este concepto al de la regresión lineal vista anteriormente y suponiendo sin pérdida de generalidad un modelo simple, $y = \beta_0 + \beta_1 x$, siendo *y* la variable respuesta o dependiente y *x* la variable independiente, la fórmula para la regresión logística se corresponderá con la probabilidad de que la variable salida $\widehat{Y}$ tome el valor 1 para cada observación *i*.

$$P_i = P(Y = 1) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x_i)}}$$

Esta función, proporcionará una curva con forma de S que se adapte mucho mejor a las observaciones, de tal forma que si $P_i > 0{,}5$, entonces $\widehat{Y}_i = 1$; mientras que para valores de $P_i < 0{,}5$, tendremos que $\widehat{Y}_i = 0$. Por convenio, para valores de $P_i = 0{,}5$ le asignaremos también el grupo 1.

En la regresión logística, a diferencia de la regresión lineal, no podemos utilizar el error cuadrático medio (*MSE*) como función de coste *L* ya que no estamos trabajando con funciones lineales y casi siempre caeríamos en mínimos locales sin llegar a optimizar nunca la función. Por lo que como estamos trabajando con probabilidades, la estrategia será buscar los parámetros que maximicen la función de verosimilitud. La idea principal es encontrar los valores de los parámetros que maximizan la probabilidad (verosimilitud) de obtener los datos observados. Para ello, comenzaremos por definir nuestra variable salida *Y*, que sigue una distribución de *Bernoulli*.

La distribución de *Bernoulli* es un modelo de probabilidad discreta, cuyo nombre proviene del matemático suizo Jacob Bernoulli y que se utiliza para describir un experimento o evento con solo dos posibles resultados (1 o 0, en este caso). En esta distribución, se define un parámetro *p*, que representa la probabilidad de éxito ($p = 1$) en un único ensayo, de tal forma que $1 - p$, representa el fracaso. Podemos expresar su función de probabilidad como

$$P(X = x) = p^x \cdot (1 - p)^{1-x}$$

Así pues, en el caso de la regresión logística tenemos que $Y \sim Ber(P)$, donde *P* representa la función sigmoide logística y por tanto para *n* observaciones tenemos la función de verosimilitud

$$J(\beta) = \prod_{i=1}^{n} P_i^{Y_i} \cdot (1 - P_i)^{1-Y_i}$$

Y tomando logaritmos, obtenemos la versión más utilizada de la función de verosimilitud que debemos maximizar para obtener los mejores estimadores β

$$ln(J) = \sum_{i=1}^{n} [Y_i \cdot ln(P_i) + (1 - Y_i) \cdot ln(1 - P_i)]$$

Como habíamos visto, en ML generalmente no se busca maximizar una función de verosimilitud *J*, sino minimizar una función de coste *L*. Teniendo en cuenta que $Max\big(ln(x)\big) = Min\big(-ln(x)\big)$ y dividiendo por el número de observaciones *n*, obtenemos la función de coste *log-loss* o de entropía logarítmica

$$L = -\frac{1}{n} \sum_{i=1}^{n} [Y_i \cdot ln(P_i) + (1 - Y_i) \cdot ln(1 - P_i)]$$

Finalmente, una vez que tenemos definida la función de coste que nos ayudará a minimizar los errores y optimizar el modelo, podemos implementar de nuevo el algoritmo de descenso de gradiente para estimar de manera iterativa el valor de los parámetros.

En cuanto a su uso, la regresión logística presenta ciertas ventajas frente a otros algoritmos de clasificación como son su fácil interpretabilidad y su bajo coste computacional, especialmente útil para grandes conjuntos de datos. Además, las salidas proporcionadas por estos modelos se corresponden con probabilidades de pertenencia a la clase positiva y nos resulta muy útil a la hora de clasificar nuevas muestras.

Por otro lado, este algoritmo, es sensible a los valores atípicos u *outliers* además de basarse en un supuesto de linealidad que, de no cumplirse, puede provocar un mal ajuste del modelo al conjunto de datos.

4.2.2) Máquinas de soporte vectorial

Las máquinas de soporte vectorial (SVM, por sus siglas en inglés), son uno de los mejores y más utilizados algoritmos de clasificación, aunque también pueden ser utilizadas en regresión. Este conjunto de algoritmos fue presentado por primera vez por Vladimir Vapnik y su equipo en los laboratorios *At&T* durante la década de los 90 (Boser et al., 1992; Cortes & Vapnik, 1995).

Su principal objetivo es encontrar el hiperplano que mejor separe las muestras de las diferentes clases en el espacio, es decir, busca el hiperplano que proporcione la máxima separación. Esta separación o margen se define como la distancia perpendicular entre el citado hiperplano y los puntos más próximos a él de las distintas clases, que reciben el nombre de vectores soporte. Para conjuntos de datos con *d* variables, el hiperplano que separará el espacio en dos será de dimensión *d-1*, de tal forma que por ejemplo y sin pérdida de generalidad una línea recta (1D) separará un plano (2D). Resumiendo, para un problema de clasificación binario (Carmona, 2016), buscamos separar las n observaciones $x_i \in \mathbb{R}^d, (i = 1, \ldots, n)$; en función de sus respectivas etiquetas y_i (en este caso para facilitar los cálculos, -1 o 1) por un hiperplano *H* tal que:

$$H = \{x \in \mathbb{R}^d \mid a^T x - b = 0\}$$

Con $a \in \mathbb{R}^d$ y $b \in \mathbb{R}$ sin especificar y de tal forma que se divida el espacio de observaciones en dos partes para separar las dos clases

$$H_+ = \{x \in \mathbb{R}^d \mid a^T x - b \geq 0\}$$

$$H_- = \{x \in \mathbb{R}^d \mid a^T x - b < 0\}$$

Sin pérdida de generalidad, asignando cada clase a una mitad tendríamos que

$$y = \begin{cases} 1, \text{ si } x \in H_+, \text{ es decir, } a^T x \geq b \\ -1, \text{ si } x \in H_-, \text{ es decir, } a^T x < b \end{cases}$$

El objetivo es encontrar el hiperplano *H* que maximice la distancia o margen con los elementos más cercanos de cada clase. Por definición de máxima separación, buscamos que este margen sea igual para las dos clases y por tanto para cada observación $x_i \in \mathbb{R}^d, (i = 1, \ldots, n)$ y dado un valor $\delta \geq 0$ tenemos

$$\begin{cases} a^T x_i - b \geq \delta, \text{ si } y_i = 1 \\ a^T x_i - b \leq -\delta, \text{ si } y_i = -1 \end{cases}$$

O lo que es lo mismo

$$y_i(a^T x_i - b) \geq \delta$$

Proporcionándonos así dos nuevos hiperplanos paralelos, definidos por las siguientes ecuaciones

$$m_1 = \{x \in R^d \mid a^T x - b_1 = a^T x - (b + \delta) = 0\}$$

$$m_2 = \{x \in R^d \mid a^T x - b_2 = a^T x - (b - \delta) = 0\}$$

La idea es maximizar la distancia entre estos dos hiperplanos. Tomando un punto aleatorio $x_1 \in m_1$ desde el que seguimos la dirección del vector normal *a* hacia m_2, obtenemos un punto $x_2 \in m_2$ tal que para algún $t \in \mathbb{R}$

$$x_2 = x_1 + t \cdot a$$

Y por tanto como $x_2 \in m_2$; $x_1 \in m_1$ tenemos que

$$b_2 = a^T(x_1 + t\, a) = a^T x_1 + t\, a^T a = b_1 + t\|a\|_2^2$$

Donde $\|a\|_2$ representa la norma euclídea, pudiendo así despejar el valor de *t* como

$$t = \frac{b_2 - b_1}{\|a\|_2^2}$$

Así pues, finalmente la distancia entre los hiperplanos sería

$$\|x_1 - x_2\|_2 = \left\| x_1 + \frac{b_2 - b_1}{\|a\|_2^2} \cdot a - x_1 \right\|_2 = \frac{|b_2 - b_1| \cdot \|a\|_2}{\|a\|_2^2} = \frac{|b_2 - b_1|}{\|a\|_2}$$

Sustituyendo b_2 y b_1 por sus valores obtenemos el valor de la distancia

$$\|x_1 - x_2\|_2 = \frac{|b_2 - b_1|}{\|a\|_2} = \frac{|b - \delta - (b + \delta)|}{\|a\|_2} = \frac{2\delta}{\|a\|_2}$$

De esta forma el hiperplano *H* paralelo a m_1, m_2 que pase por el punto medio de ambos, es decir, que se encuentre a una distancia de $(\delta/\|a\|_2)$ que vamos a designar como *D,* de los otros dos hiperplanos, será el que garantice la separación máxima. Así pues, podemos afrontar ahora esta tarea como un problema de optimización de tal forma que buscamos los parámetros δ, a, b que maximicen la distancia

$$max\ D\ = \max_{\delta,a,b} \frac{\delta}{\|a\|_2},\ \text{tal que}\ y_i(a^T x_i - b) \geq \delta,\ i = 1, \dots, n$$

Gráficamente podemos comprender mejor estos conceptos en la siguiente figura (*Ilustración 15*).

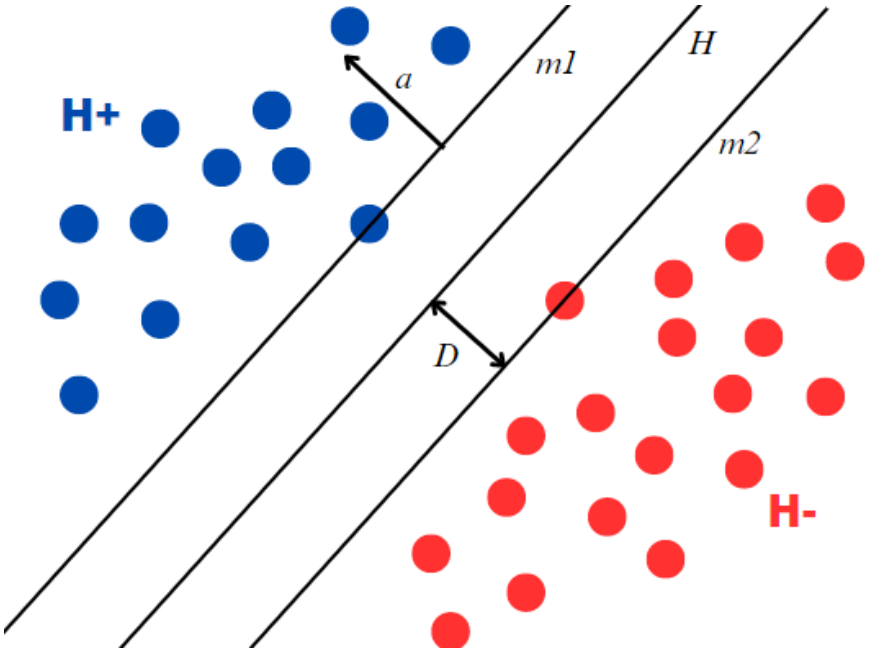

Ilustración 15. Hiperplanos de separación en SVM. Fuente: elaboración propia en Canva basada en el artículo (Carmona, 2016).

Teniendo en cuenta que el valor óptimo de δ debe ser positivo y realizando algunas transformaciones tenemos la fórmula final del problema de optimización.

$$\min_{a,b} \|a\|_2\ ,\ \text{tal que}\ y_i(a^T x_i - b) \geq 1,\ i = 1, \dots, n$$

Habitualmente no se trabaja directamente con esta ecuación debido a su complejidad, sino que se afronta el problema dual más sencillo definido por la siguiente fórmula:

$$\max_{\delta \geq 0} \sum_{i=1}^{n} \delta_i - \frac{1}{2} \sum_{i,j=1}^{n} \delta_i\, \delta_j\, y_i\, y_j\, x_i^T x_j,\ \text{tal que}\ \sum_{i=1}^{n} \delta_i\, y_i = 0$$

Con $\delta = (\delta_1, \dots, \delta_n)^T \in \mathbb{R}^n$ multiplicadores de Lagrange.

En términos generales cuando estamos trabajando con datos linealmente separables, es decir, si trabajando en un plano se pueden separar las clases por una línea recta, la regresión logística y el SVM lineal se comportarán de manera muy similar. La principal diferencia entre los algoritmos es que la regresión logística afronta el problema desde una perspectiva probabilística y el SVM busca el hiperplano de máxima separación. Al tratarse de un algoritmo más complejo, las SVM proporcionarán habitualmente mejores resultados, aunque su gasto computacional también será mayor.

Por otra parte, y aquí reside la polivalencia de las SVM, cuando estemos trabajando con datos no linealmente separables las SVM permiten utilizar ***kernels*** para afrontar estos problemas. Un *kernel* es una función matemática que se utiliza para transformar el espacio de características original en un espacio de características de mayor dimensionalidad, buscando que en este nuevo

espacio las clases si sean linealmente separables. La elección de este *kernel* resultará determinante para el resultado final.

Así pues, modificamos mediante una función ϕ: $\mathbb{R}^d \longrightarrow \mathbb{R}^r$ con $r > d$, de mapeo de características que convierta los vectores x_i, x_j de la ecuación dual en linealmente separables. Debemos pues encontrar una función *Kernel* K: $\mathbb{R}^d x \mathbb{R}^d \longrightarrow \mathbb{R}$ tal que:

$$\phi(x_i)^T \phi(x_i) = K(x_i, x_j)$$

se trabaja así con el siguiente problema de optimización:

$$\max_{\delta \geq 0} \sum_{i=1}^{n} \delta_i - \frac{1}{2} \sum_{i,j=1}^{n} \delta_i \, \delta_j \, y_i \, y_j \, \phi(x_i)^T \phi(x_j) = \max_{\delta \geq 0} \sum_{i=1}^{n} \delta_i - \frac{1}{2} \sum_{i,j=1}^{n} \delta_i \, \delta_j \, y_i \, y_j \, K(x_i, x_j)$$

Algunos de los *kernels* más utilizados son:

- **Lineal**: como habíamos dicho, consiste en no realizar transformaciones y se utiliza cuando los datos ya son linealmente separables.

$$K(x_i, x_j) = x_i \cdot x_j$$

- **Polinómico:** se utiliza una función polinómica para mapear los datos a un espacio de mayor dimensión. El grado *d* de este polinomio se puede ajustar.

$$K(x_i, x_j) = (x_i \, x_j + c)^d$$

- ***Radial Basis Function*** **(RBF):** es uno de los *kernels* más populares. Transforma los datos a un espacio de dimensionalidad infinita utilizando funciones de base radial.

$$K(x_i, x_j) = exp\left(-\gamma \left\| x_i - x_j \right\|^2\right)$$

  El parámetro gamma (γ) controla el alcance de la influencia de cada ejemplo de entrenamiento.

- **Sigmoide:** aplica la función tangente hiperbólica (*tanh*) a los vectores de características. Esta función tiene características de función sigmoide con el rango de valores de salida en el intervalo [-1,1]

$$K(x_i, x_j) = tanh(x_i x_j + c)$$

4.2.3) Métricas de evaluación

Existen numerosas técnicas o métricas distintas para evaluar el desempeño de los algoritmos de clasificación, pero todas parten de una idea común: buscan recompensar los aciertos, es decir, las coincidencias entre la categoría real y la categoría predicha o, en otras palabras, la igualdad de valores entre Y e $\hat{Y}$. Algunas de las métricas o herramientas más populares son (Grandini et al., 2020):

- **Matriz de confusión (*Confusion Matrix*):** no se trata de una métrica como tal, sino de una tabla cruzada que recoge el número de observaciones clasificadas correctamente para cada clase. Por conveniencia, las filas se corresponderán con las categorías reales y las columnas por las predichas. Para un ejemplo de clasificación binario, la matriz de confusión quedaría de la siguiente forma (*Tabla 1*):

Tabla 1. Matriz de Confusión.

		Predicciones ($\hat{Y}$)		
Categorías Reales (Y)	**Clases**	**Positivo (1)**	**Negativo (0)**	**Total**
	Positivo (1)	Verdaderos Positivos (VP)	Falsos Negativos (FN)	VP+FN
	Negativo (0)	Falsos Positivos (FP)	Verdaderos Negativos (VN)	VN+FP
	Total	VP+FP	VN+FN	N =VP+FN+VN+FP

donde VP representa las observaciones con etiqueta positiva ($Y_i = 1$) clasificadas correctamente ($\hat{Y}_i = 1$), VN las observaciones con etiqueta negativa ($Y_i = 0$) clasificadas correctamente ($\hat{Y}_i = 0$), FN las observaciones con etiqueta positiva ($Y_i = 1$) clasificadas como negativas ($\hat{Y}_i = 0$) y FP las observaciones con etiqueta negativa ($Y_i = 0$) clasificadas como positivas ($\hat{Y}_i = 1$).

- **Exactitud (*Accuracy*):** se define como la proporción de observaciones correctamente clasificadas. Obtenemos su valor dividiendo las predicciones correctamente clasificadas ($Y = \hat{Y}$) entre el número total de predicciones. Habitualmente se expresa este resultado multiplicado por 100 como un porcentaje.

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

- **Precisión (*Precision*):** se define como la proporción de verdaderos positivos sobre el total de predicciones positivas. Se utiliza para evaluar la capacidad del modelo para no clasificar como positivas aquellas observaciones que no lo son.

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

- **Sensibilidad (*Recall*):** mide la capacidad del modelo para detectar correctamente las observaciones positivas y se calcula como la proporción de verdaderos positivos dividido por el total de observaciones realmente positivas.

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

- **Especificidad (*Specificity*):** de manera análoga a la sensibilidad, esta métrica evalúa la capacidad para clasificar correctamente las observaciones negativas. Se calcula dividiendo los verdaderos negativos entre el total de negativos.

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

- **Exactitud balanceada (*Balanced Accuracy*):** similar a la exactitud estándar, esta métrica es especialmente útil cuando trabajamos con *datasets* desbalanceados, es decir, en los que una categoría domina sobre el resto. Se calcula como la suma de las proporciones de las observaciones correctamente clasificadas para cada categoría, dividido por el número total de categorías. Tomando como referencia la matriz de confusión, tenemos:

$$Balanced\_Accuracy = \frac{\frac{VP}{VP + FN} + \frac{VN}{VN + FP}}{2}$$

- **F1-Score:** es una de las métricas más utilizadas en clasificación, y se define como la media armónica entre la sensibilidad y la precisión del modelo.

$$F1 - Score = \frac{2}{precision^{-1} + recall^{-1}} = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

En general, se tiende a evaluar el desempeño de los modelos aplicando estas métricas sobre el subconjunto de test. Otra técnica alternativa y muy popular es la **validación cruzada**. Consiste en dividir el conjunto de datos en *k* subconjuntos del mismo tamaño, repitiendo el proceso de entrenamiento y test *k* veces, utilizando cada uno de los subconjuntos como conjunto de evaluación y el resto como conjunto de entrenamiento (*Ilustración 16*). Por último, se calcula la media de los resultados obtenidos en cada iteración para obtener una estimación final del modelo.

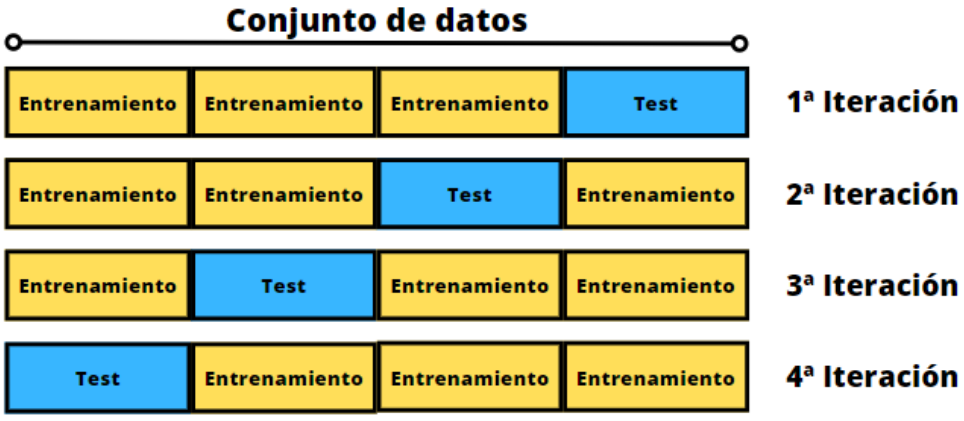

Ilustración 16. Validación cruzada para k = 4. Fuente: elaboración propia con Canva.

5) Aplicaciones Web

En este capítulo describiremos brevemente cuales son las principales herramientas (lenguajes de programación, software específico…) hoy en día para desarrollar aplicaciones web, ya que al fin y al cabo ese es el objetivo de este trabajo.

Hay que tener en cuenta que actualmente existen infinidad de opciones para desarrollar una página web; desde programarla desde cero, hasta utilizar interfaces específicas como WordPresss en las que, una vez que hemos conseguido el dominio (la dirección de la web) solo debemos introducir los datos y elegir las plantillas y elementos estéticos que más nos gusten sin necesidad de programar nada.

Aun así, debemos señalar la importancia en cualquier caso de HTML (HyperText Markup Language). Este lenguaje de programación es la base fundamental de cualquier página web y proporciona una estructura semántica para organizar y presentar información en forma de documentos hipertexto, es decir, es un conjunto de elementos y etiquetas que los rodean que se utilizan para definir la estructura lógica de una página web y facilitan su interpretación para el navegador web. Por ejemplo, la etiqueta <img> indicaría la presencia de una imagen y <h1> la entrada de un título.

Otro lenguaje fundamental para el diseño de páginas web es CSS (Cascading Style Sheets). Si bien HTML nos permite definir la estructura de la página web, es con el lenguaje CSS con el que podremos diseñar y definir los elementos estéticos de la página. CSS se basa en reglas que definen cómo se debe presentar cada elemento HTML en un documento. Por ejemplo, combinando ambos lenguajes, con la siguiente línea de código estaríamos escribiendo el título ML de color naranja.

```
<h1 style="color:Orange;">ML</h1>
```

Además, cabe destacar la creciente relevancia en los últimos años de Javascript. Se trata de un lenguaje de programación de alto nivel, interpretado y orientado a objetos que se utiliza principalmente para crear interactividad en sitios web. A diferencia de los mencionados anteriormente, que se centraban en la estructura y el diseño de la página, la función de Javascript consiste en añadir comportamiento dinámico y funcionalidad a las aplicaciones web, creando así experiencias interactivas que mejoran la experiencia de los usuarios.

Por último, vamos a señalar distintas herramientas o librerías que permiten la creación de páginas web desde lenguajes de programación de alto nivel que no están específicamente diseñados para esta función

En Python, algunas de las principales alternativas para desarrollar aplicaciones web son:

- **Flask**: se centra en la simplicidad, proporcionando únicamente lo esencial para construir aplicaciones web. Por estos motivos es muy popular para crear aplicaciones web simples, pequeñas y rápidas

- **Django**: es una herramienta más compleja y completa enfocada principalmente en la eficiencia y la seguridad. Permite crear aplicaciones web escalables de cualquier tamaño.

- **Streamlit**: esta biblioteca permite crear fácilmente aplicaciones web interactivas y está enfocada principalmente para científicos de datos que quieran mostrar sus análisis de una manera atractiva. Además, actualiza automáticamente la interfaz a medida que los datos cambian, facilitando la creación de aplicaciones en tiempo real. En este trabajo hemos optado por utilizar esta herramienta.

Por su parte, en R, el principal paquete o librerías que facilitan esta tarea es **Shiny**. Este paquete permite desarrollar aplicaciones web completas utilizando código R, sin necesidad de conocimientos de lenguajes de programación web como los mencionados a principio del capítulo. Una ventaja de este paquete es que, al igual que con Streamlit, los elementos de la interfaz se actualizan automáticamente cuando los datos subyacentes a partir de los que esté trabajando cambien. Además, permite crear gráficos interactivos o tablas dinámicas y añadir elementos como campos de entrada o botones para interactuar con los datos.

Así pues, como mencionábamos anteriormente, nuestra aplicación web está escrita en Python con la ayuda de la librería especializada en desarrollo de webs Streamlit. Podemos acceder a ella en la siguiente dirección:

https://tfg-estadistica-juanmarcosdiaz.streamlit.app/

Además, podemos encontrar todos los archivos utilizados en su desarrollo en el siguiente repositorio de Github:

https://github.com/JMarksss/TFG_Streamlit_JuanMarcosDiaz

La aplicación web cuenta con una página inicial **Inicio** (Inicio.py) y otras siete páginas a las que podemos acceder mediante un índice desplegable en el margen izquierdo. Las páginas contienen la siguiente información:

- **Inicio** (Inicio.py): página principal que sirve de introducción, incluye las definiciones de ML, así como de los subtipos supervisado y no supervisado. También incluye un cuestionario interactivo con el que, en función de nuestras respuestas, nos recomendará el tipo de análisis que debemos realizar.

- **Dataset y Análisis Descriptivo** (Dataset_y_Análisis_Descriptivo.py): en esta página introducimos el *dataset* de prueba con el que pondremos a prueba los distintos modelos de *Machine Learning*. El *dataset* lo obtenemos del famoso repositorio de ML Kaggle, en la siguiente dirección:

  https://www.kaggle.com/datasets/rashikrahmanpritom/heart-attack-analysis-prediction-dataset

  Elegimos este *dataset* ya que por su composición lo creemos adecuado para poder practicar todos los algoritmos mencionados anteriormente, además de tratar un tema tan interesante como pueden ser los problemas de corazón. Además, realizamos un análisis descriptivo del conjunto de datos con distintas tablas y gráficos. Después de este análisis concluimos que los datos no parecen ajustarse a la realidad ya que muchos de los indicadores que habitualmente se consideran como factores de riesgo cardiacos, parecen no influir o influir negativamente en el desarrollo de esta enfermedad. Aun así, los seguimos considerando adecuados para aplicar las distintas técnicas de ML.

- **Preprocesado** (Preprocesado.py): en esta página, definimos y describimos técnicas de preprocesado como la codificación, la discretización o el escalado. También aplicamos estas técnicas sobre nuestro conjunto de datos.

- **Reducción de la Dimensionalidad (PCA):**

  (Reducción_de_la_Dimensionalidad_(PCA).py): en esta página describimos detalladamente el PCA, el principal algoritmo para la reducción de la dimensionalidad. Además, lo aplicamos de forma interactiva sobre el conjunto de datos, pudiendo elegir entre el tipo de escalado y el porcentaje de varianza que queremos que explique el modelo. En función del criterio del codo (*Scree Plot*) y la varianza explicada, podemos elegir la mejor transformación.

- **Clustering** (Clustering.py): en esta página, definimos el *clustering* y algunos de sus principales subtipos: el jerárquico aglomerativo, el particional y el basado en densidad; así como las métricas necesarias para evaluarlos. Además, implementamos estos algoritmos de manera interactiva sobre nuestro conjunto de datos para poder elegir el modelo que más nos convenza. Gracias a los modelos aglomerativos, concluimos que dos es el número correcto de *clusters*, aunque es con el *K-Means* con el algoritmo que obtenemos mejores resultados.

- **Regresión** (Regresión.py): en esta página describimos la regresión lineal, tanto la simple como la múltiple, y sus métricas. Aplicamos sobre nuestro conjunto de datos y de forma interactiva distintos modelos de regresión, aunque los resultados obtenidos no son nada buenos principalmente porque las variables del *dataset* no están pensadas para este tipo de análisis.

- **Clasificación** (Clasificación.py): en esta página describimos dos de los algoritmos más populares para la clasificación: la regresión logística y las SVM. También introducimos las métricas y sistemas de evaluación (división train-test o validación cruzada) para estos modelos. Además, aplicamos sobre nuestro conjunto de datos y de manera interactiva, los distintos modelos obteniendo en general muy buenos resultados.

- **Tu propio análisis** (Tu_propio_análisis.py): en la última página permitimos a cualquier usuario subir sus propios datos (ficheros en formato .csv y hasta 200MB) y realizar cualquier análisis de los vistos anteriormente. En primer lugar, la página permitirá visualizar la tabla de datos, y realizar un breve análisis descriptivo de las variables. Posteriormente, en la sección de preprocesado podremos observar la tabla de valores perdidos y decidir que variables deseamos eliminar. También deberemos seleccionar, si existiera, la variable dependiente *Y* (que se codificaría si hiciera falta), así como el tipo de escalado que deseamos. Además, los datos perdidos se imputarán por la mediana o moda de la variable dependiendo de si esta es numérica o categórica respectivamente. Una vez transformados los datos podemos seleccionar el tipo de análisis y evaluarlo de manera análoga a lo que habíamos visto en el resto de las páginas.

6) Resultados

En este capítulo recogeremos y describiremos brevemente los resultados obtenidos en la aplicación web, en la que implementamos las técnicas y algoritmos estudiados a lo largo de la memoria. Aunque en la propia aplicación web podremos encontrar todos los detalles, a continuación, presentamos un resumen del análisis realizado.

El objetivo principal de nuestro estudio era implementar y evaluar los distintos algoritmos detallados en los anteriores capítulos sobre un conjunto de datos sencillo, pero que a su vez contuviera variables numéricas para estudiar los modelos de regresión y una variable categórica binaria que sirviera como variable dependiente en los modelos de clasificación. En este *dataset* la variable salida describirá la probabilidad de padecer una cardiopatía distinguiendo dos grupos: pacientes con mayor probabilidad y pacientes con menor probabilidad de sufrir este tipo de problema. Además, contaremos con variables numéricas como la edad o la frecuencia cardíaca máxima.

Sobre este conjunto de datos realizamos en primer lugar, un análisis descriptivo exploratorio. En este primer análisis, para las variables numéricas analizamos distintas medidas como la media, la mediana, los cuartiles o los valores extremos deduciendo que los pacientes no son jóvenes y que la mayoría presenta valores alarmantes en los niveles de tensión arterial y colesterol (*Tabla 2*).

Tabla 2. Análisis descriptivo de las variables numéricas. Fuente: aplicación web del TFG.

	Edad	Tensión	Colesterol	Frecuencia_Max	Depresión_ST	Vasos_col
count	303	303	303	303	303	303
mean	54.3663	131.6238	246.264	149.6469	1.0396	0.7294
std	9.0821	17.5381	51.8308	22.9052	1.1611	1.0226
min	29	94	126	71	0	0
25%	47.5	120	211	133.5	0	0
50%	55	130	240	153	0.8	0
75%	61	140	274.5	166	1.6	1
max	77	200	564	202	6.2	4

Por otra parte, para cada variable categórica realizamos distintos gráficos de tarta para estudiar su distribución (*Ilustración 17*).

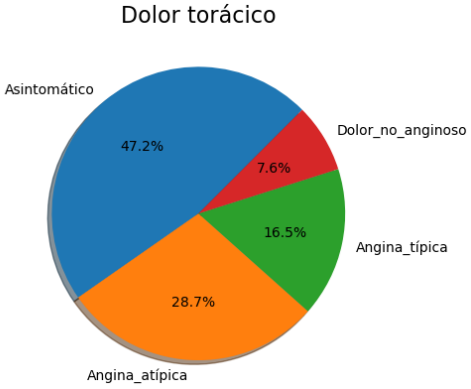

Ilustración 17. Gráfico de tarta de la distribución del dolor torácico. Fuente: app web del TFG

Para concluir este análisis descriptivo estudiamos la relación entre la variable salida y el resto de variables. Para las de tipo numérico utilizamos *boxplots* o gráficos de caja (*Ilustración 18*) y para las variables categóricas empleamos tablas de contingencia normalizadas (*Tabla 3*).

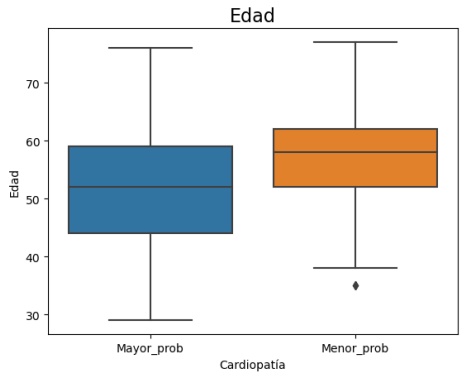

Ilustración 18. Boxplot para la edad y la cardiopatía. Fuente: app web

Tabla 3. Tabla de contingencia normalizada para la edad y la cardiopatía. Fuente: app web

Sexo	Mayor_prob	Menor_prob
Hombre	0.4493	0.5507
Mujer	0.75	0.25

Las conclusiones obtenidas de este primer análisis exploratorio son muy sorprendentes ya que muchos de los indicadores que habitualmente se consideran como factores de riesgo cardiacos (como alto el nivel de colesterol o la avanzada edad), parecen no influir o influir negativamente en el desarrollo de esta enfermedad. Deducimos por tanto que, o bien se trata de una muestra pequeña que no recoge suficientes observaciones o estamos trabajando con un *dataset* artificial que no se ajusta a la realidad.

Tras este primer análisis aplicamos sobre nuestro conjunto de datos las distintas técnicas de preprocesado descritas en la propia aplicación: codificación *One-Hot* para las variables categóricas y distintos tipos de escalado (estándar, robusto, min-max (*Tabla 4*) o sin escalar) para las variables numéricas. Estos nuevos conjuntos de datos son sobre los que aplicaremos los modelos de ML.

Tabla 4. Primeras filas (observaciones) y columnas (variables) del dataset codificado con escalado min-max. Las 5 primeras columnas se corresponden con las variables numéricas en el orden previo. Fuente: app web

0	1	2	3	4	5	6
0.7083	0.4811	0.2443	0.6031	0.371	0	0
0.1667	0.3396	0.2831	0.8855	0.5645	0	0
0.25	0.3396	0.1781	0.771	0.2258	0	1
0.5625	0.2453	0.2511	0.8168	0.129	0	0
0.5833	0.2453	0.5205	0.7023	0.0968	0	1
0.5833	0.434	0.1507	0.5878	0.0645	0	0
0.5625	0.434	0.3836	0.626	0.2097	0	1

Una vez transformados los datos procedemos a aplicar las distintas técnicas y algoritmos de ML estudiados en la memoria. Si comenzamos por la reducción de la dimensionalidad con PCA, concluimos que necesitamos 5 (6 para el escalado robusto) componentes para explicar al menos el 70% de la varianza. Por otro lado, los *ScreePlots* (*Ilustración 19*) nos indican que con 3 componentes y el escalado min-max estaríamos haciendo un ajuste bastante bueno conservando así en torno al 60% de la varianza.

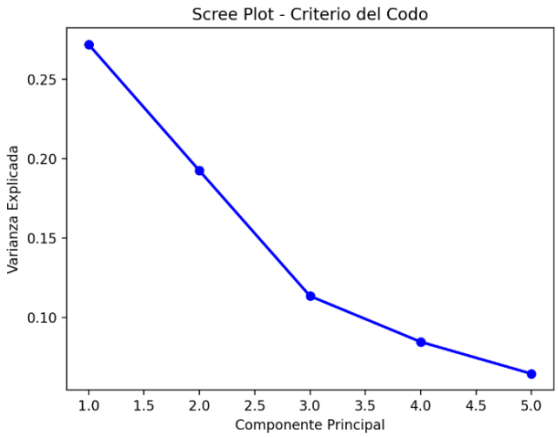

Ilustración 19. Scree Plot para el escalado min-max. Fuente: app web

Para mejorar y simplificar los resultados en el *clustering* trabajaremos ya con los datos transformados tras PCA (en este caso, explicado el 70% de la varianza y con escalado estandar) y comenzaremos aplicando un enfoque jerárquico-aglomerativo ya que desconocemos el número óptimo de *clusters k.* Tras analizar todos los dendogramas (*Ilustración 20*) posibles teniendo en cuenta las distintas combinaciones de medidas (euclídea, Manhattan, Chebyshev y Mahalanobis) y criterios (Ward, distancia máxima, media y mínima) nos parece que el criterio de Ward con 2 grupos es el más convincente. Además, observamos que los criterios de distancia mínima y media presentan *clusters* muy desbalanceados afectados por los *outliers*.

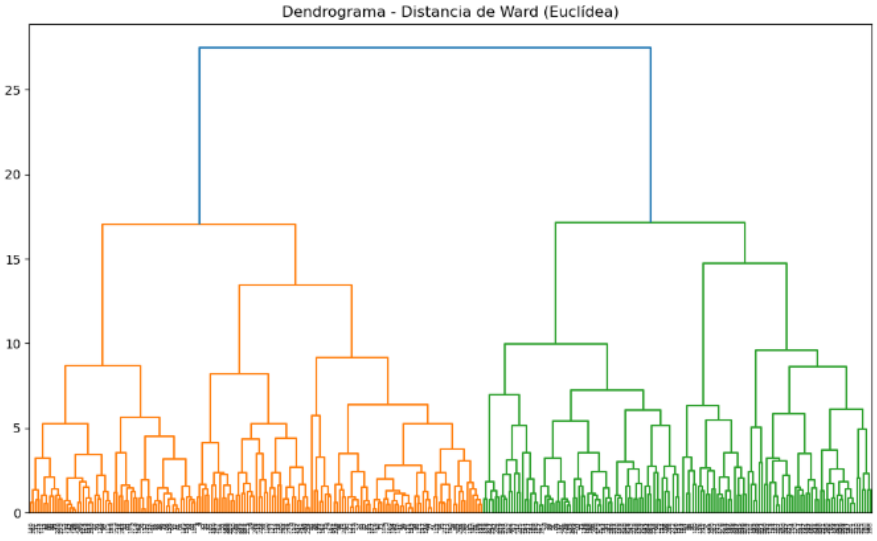

Ilustración 20. Dendograma con el criterio de Ward. Fuente: app web

La evaluación mediante métricas (Silhouette, Calinski-Harabasz y Davies-Bouldin) nos confirma que el criterio de Ward con dos *clusters* es el mejor modelo aglomerativo, aunque los modelos descompensados de distancia mínima obtienen mejores valores en algunas métricas.

Una vez que sabemos que 2 es el número correcto de conglomerados, implementamos los modelos de *clustering* particional con los que obtenemos los mejores resultados. En particular, el K-Means es el modelo que obtiene las mejores evaluaciones.

Desgraciadamente, con el DBSCAN, no conseguimos determinar correctamente el valor de los hiper-parámetros *r* y *MinP,* y no obtenemos buenos resultados.

Pasando ahora ya al aprendizaje supervisado, para los análisis de regresión empleamos el *dataset* sin escalar y obtenemos resultados bastante malos para todas las variables numéricas, tanto para el análisis simple (*Ilustración 21*) como para el múltiple. Esto se debe principalmente a que estas variables no están pensadas originalmente para este tipo de análisis y no parecen guardar mucha relación. Además, la presencia de *outliers* empeora considerablemente la evaluación. También deberíamos tener en cuenta que para un correcto análisis de regresión habría que comprobar las hipótesis de normalidad, independencia y homocedasticidad de los residuos.

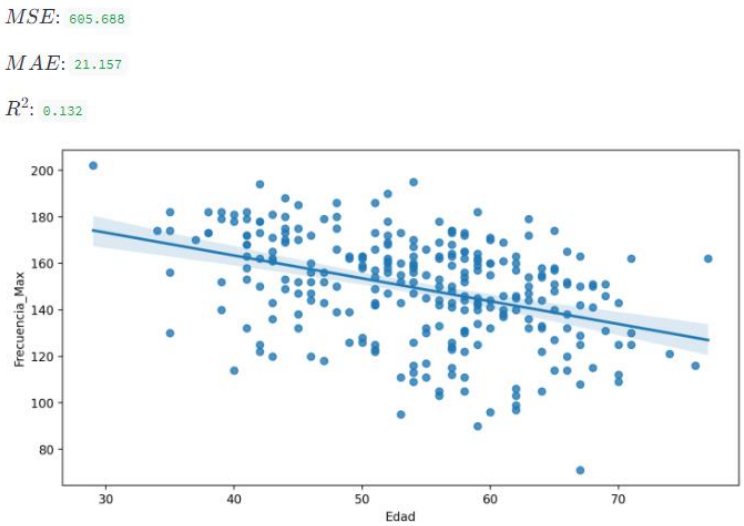

Ilustración 21. Regresión: Frecuencia_Max en función de la Edad. Fuente: app web

Por último, implementando los modelos de clasificación una vez seleccionada la variable cardiopatía como variable dependiente a predecir, podemos concluir que obtenemos buenos resultados en términos generales. Podemos observar como a medida que aumentamos el tamaño del test en la división train-test o reducimos el número de subgrupos *k* en la validación cruzada, empeoran los resultados. En cuanto a los algoritmos, no observamos grandes diferencias entre ninguno de ellos puesto que todos obtienen buenos resultados en torno al 85% de exactitud (*Tabla 5, Ilustración 22*).

Tabla 5. Métricas para la validación cruzada con k=7. Fuente: app web.

Modelo	Accuracy	F1	Recall	Precision	Balanced_Accuracy
Reg_Log	0.848	0.865	0.891	0.844	0.845
SVM_linear	0.845	0.862	0.891	0.838	0.841
SVM_rbf	0.828	0.847	0.872	0.827	0.825
SVM_sigmoid	0.832	0.847	0.866	0.834	0.829
SVM_poly	0.815	0.837	0.878	0.803	0.809

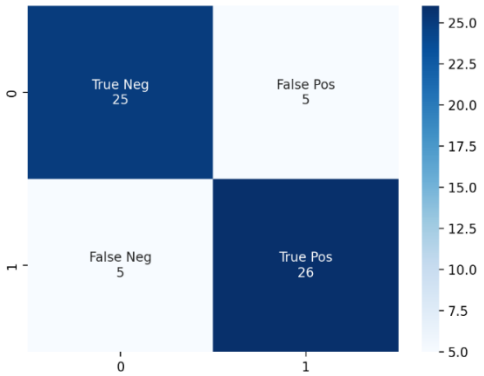

Ilustración 22.. Matriz de confusión para evaluar el modelo de Regresión Logística con un 20% de test. Fuente: app web

7) Conclusiones

Tanto la IA como el ML, son conceptos cada vez más presentes en nuestro día a día. Constantemente leemos, escuchamos o recibimos noticias relacionadas con el desarrollo de algún nuevo algoritmo o aplicación que puede llevar a cabo de manera automática y en poco tiempo tareas que hace años nos parecerían imposibles, como la creación de auténticas obras de arte. El desarrollo en este campo es cada vez más rápido y puede llegar a asustar, ya que aún no se conocen las consecuencias y los cambios que puede depararnos este nuevo avance tecnológico.

Este trabajo no pretende calmar esa creciente preocupación, pero sí dar a conocer y familiarizar a cualquier ciudadano, de una manera simple y atractiva, con algunas de las principales técnicas y algoritmos de ML, que sirven como base para las nuevas y más complejas herramientas mencionadas.

Así pues, las conclusiones obtenidas tanto de la memoria teórica, como de la aplicación web práctica son las siguientes:

1. El ML es una técnica que se utiliza para desarrollar sistemas de IA que permite a las máquinas aprender a partir de los datos. Dentro de este campo de la ciencia encontramos numerosos subtipos muy distintos en función de cómo sea el tipo de aprendizaje, es un área heterogénea.

2. El preprocesado es de vital importancia para la correcta aplicación y evaluación de los modelos. Sin un preprocesado previo no será posible implementar un gran porcentaje de algoritmos y, aquellos que sí puedan, es probable que no obtengan buenos resultados.

3. El PCA es un algoritmo simple y a la vez eficaz para reducir la dimensión de nuestro conjunto de datos, ya sea para poder visualizarlos o para emplearlos en futuros análisis como en el *clustering*.

4. Existen numerosas técnicas y algoritmos de *clustering* distintos, cada uno con sus ventajas e inconvenientes. Por este motivo conviene conocer como son nuestros datos y no limitarse a implementar un único modelo, sino desarrollar y evaluar distintos modelos para poder seleccionar el que mejores resultados obtenga.

5. La regresión lineal es la técnica más simple para poder predecir los valores de una variable numérica. El enfoque clásico para la regresión difiere del enfoque del ML, centrado en la predicción.

6. Hay muchos algoritmos de clasificación distintos para predecir los valores de una variable categórica, pero la regresión logística SVM son algoritmos relativamente sencillos que permiten obtener buenos resultados.

7. La elección de un Kernel para las SVM nos permitirá afrontar problemas no linealmente separables, mejorando así nuestros resultados.

8. Para evaluar cualquier modelo de ML, conviene emplear distintas métricas y no limitarse a utilizar sólo una.

9. Streamlit es una librería de Python muy útil para poder desarrollar aplicaciones web de una manera rápida, sencilla y atractiva.

8) Bibliografía

Ahsan, M. M., Mahmud, M. P., Saha, P. K., Gupta, K. D., & Siddique, Z. (2021). Effect of data scaling methods on machine learning algorithms and model performance. *Technologies*, *9*(3), 52.

Alasadi, S. A., & Bhaya, W. S. (2017). Review of data preprocessing techniques in data mining. *Journal of Engineering and Applied Sciences*, *12*(16), 4102-4107.

Berkson, J. (1944). Application of the logistic function to bio-assay. *Journal of the American statistical association*, *39*(227), 357-365.

Boser, B. E., Guyon, I. M., & Vapnik, V. N. (1992). A training algorithm for optimal margin classifiers. *Proceedings of the fifth annual workshop on Computational learning theory*, 144-152.

Caliński, T., & Harabasz, J. (1974). A dendrite method for cluster analysis. *Communications in Statistics-theory and Methods*, *3*(1), 1-27.

Carmona, E. (2016). *Tutorial sobre Máquinas de Vectores Soporte (SVM)*, 1-17.

Cortes, C., & Vapnik, V. (1995). Support-vector networks. *Machine learning*, *20*, 273-297.

Cox, D. R. (1958). The regression analysis of binary sequences. *Journal of the Royal Statistical Society: Series B (Methodological)*, *20*(2), 215-232.

Davies, D. L., & Bouldin, D. W. (1979). A cluster separation measure. *IEEE transactions on pattern analysis and machine intelligence*, *2*, 224-227.

Emmanuel, T., Maupong, T., Mpoeleng, D., Semong, T., Mphago, B., & Tabona, O. (2021). A survey on missing data in machine learning. *Journal of Big Data*, *8*(1), 1-37.

Ester, M., Kriegel, H.-P., Sander, J., & Xu, X. (1996). A density-based algorithm for discovering clusters in large spatial databases with noise. *Knowledge Discovery and Data Mining*, *96*(34), 226-231.

Fraley, C., & Raftery, A. E. (2002). Model-based clustering, discriminant analysis, and density estimation. *Journal of the American statistical Association*, *97*(458), 611-631.

Gentleman, R., & Carey, V. J. (2008). Unsupervised machine learning. En *Bioconductor case studies* (pp. 137-157). Springer.

Grandini, M., Bagli, E., & Visani, G. (2020). Metrics for multi-class classification: An overview, 1-7. *arXiv preprint arXiv:2008.05756*.

Helm, J. M., Swiergosz, A. M., Haeberle, H. S., Karnuta, J. M., Schaffer, J. L., Krebs, V. E., Spitzer, A. I., & Ramkumar, P. N. (2020). Machine learning and artificial intelligence: Definitions, applications, and future directions. *Current reviews in musculoskeletal medicine*, *13*, 69-76.

Hotelling, H. (1933). Analysis of a complex of statistical variables into principal components. *Journal of educational psychology*, *24*(6), 417.

Hyvarinen, A. (1999). Fast and robust fixed-point algorithms for independent component analysis. *IEEE transactions on Neural Networks*, *10*(3), 626-634.

Jackson, P. (1986). *Introduction to expert systems*, 10-23.

Johnson, S. C. (1967). Hierarchical clustering schemes. *Psychometrika*, *32*(3), 241-254. https://doi.org/10.1007/BF02289588

Kaufmann, L. (1987). Clustering by means of medoids. *Data Analysis Based on the L1 Norm Conference, Neuchatel, 1987*, 405-416.

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *nature*, *521*(7553), 436-444.

Lee, D.-H. (2013). Pseudo-label: The simple and efficient semi-supervised learning method for deep neural networks. *Workshop on challenges in representation learning, ICML*, *3*(2), 896.

MacQueen, J. (1967). Classification and analysis of multivariate observations. *5th Berkeley Symposyum of. Mathematics, Statistics and Probability*, 281-297.

Madhulatha, T. S. (2012). An overview on clustering methods, 1-5. *arXiv preprint arXiv:1205.1117*.

McCarthy, J., Minsky, M. L., Rochester, N., & Shannon, C. E. (2006). A proposal for the dartmouth summer research project on artificial intelligence, august 31, 1955. *AI magazine*, *27*(4), 12-13.

McCulloch, W. S., & Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, *5*, 115-133.

Montgomery, D. C., Peck, E. A., & Vining, G. G. (2021). *Introduction to linear regression analysis*. John Wiley & Sons, 10-147.

O'Shea, K., & Nash, R. (2015). An introduction to convolutional neural networks, 1-10. *arXiv preprint arXiv:1511.08458*.

Pearson, K. (1901). LIII. On lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin philosophical magazine and journal of science*, *2*(11), 559-572.

Samuel, A. L. (1959). Some studies in machine learning using the game of checkers. *IBM Journal of research and development*, *3*(3), 210-229.

Seger, C. (2018). *An investigation of categorical variable encoding techniques in machine learning: Binary versus one-hot and feature hashing*, 6-11

Shlens, J. (2014). *A Tutorial on Principal Component Analysis,* 1-11. (arXiv:1404.1100). arXiv. http://arxiv.org/abs/1404.1100.

Singh, A., Thakur, N., & Sharma, A. (2016). A review of supervised machine learning algorithms. *2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom)*, 1310-1315.

Sokal, R. R., & Sneath, P. H. A. (1963). Principles of numerical taxonomy. *Principles of numerical taxonomy*, 191-199.

Sorzano, C. O. S., Vargas, J., & Montano, A. P. (2014). A survey of dimensionality reduction techniques, 1-21. *arXiv preprint arXiv:1403.2877*.

Thant, A. A., Aye, S. M., & Mandalay, M. (2020). Euclidean, Manhattan and Minkowski Distance Methods For Clustering Algorithms. *International Journal of Scientific Research in Science, Engineering and Technology***,** *7, 553-558.*

Van der Maaten, L., & Hinton, G. (2008). Visualizing data using t-SNE. Journal of machine learning research, *9*(11), 2579-2600.

Van Engelen, J. E., & Hoos, H. H. (2020). A survey on semi-supervised learning. *Machine learning*, *109*(2), 373-440.

Ward Jr, J. H. (1963). Hierarchical grouping to optimize an objective function. *Journal of the American statistical association*, *58*(301), 236-244.

Watkins, C. J., & Dayan, P. (1992). Q-learning. *Machine learning*, *8*, 279-292.

Wiering, M. A., & Van Otterlo, M. (2012). Reinforcement learning. *Adaptation, learning, and optimization*, *12*(3), 729.