Sección de Matemáticas
Universidad de La Laguna

Alba Maura Candelario Brito

Fundamentos de la Ciencia de Datos

Fundamentals of Data Science

Trabajo Fin de Grado
Grado en Matemáticas
La Laguna, junio de 2023

Dirigido por
Rodrigo Trujillo González

Rodrigo Trujillo González
Departamento de Análisis Matemático
Universidad de La Laguna
38200 La Laguna, Tenerife

Agradecimientos

Quiero agradecer a mi familia por ser mi mayor fuente de motivación a lo largo de este desafiante viaje académico. En especial, a mis padres y mi hermano, por su constante apoyo emocional, sus palabras de aliento y su confianza en mí. Gracias por animarme siempre. Este logro es tan suyo como mío.

Muchas gracias a todos mis amigos de siempre, y a todos los amigos que me ha dado esta etapa. Han sido mi compañia a lo largo de este arduo camino y espero tenerles cerca en el siguiente. Gracias por prestarse siempre a ayudar, ser un apoyo en los momentos de incertidumbre y por todos los maravillosos recuerdos que me guardo.

También quiero agradecer a mi tutor por sus conocimientos, orientación y paciencia a lo largo del proceso de realización de este Trabajo de Fin de Grado. Gracias por su disposición para escuchar mis ideas, por su asesoramiento y por ser una guía en cada etapa de este proyecto. Y, en general, quiero agradecer a todos los profesores de que aman las matemáticas, y más aún, a aquellos que aman la docencia.

Alba Maura Candelario Brito
La Laguna, 22 de mayo de 2023

Resumen · Abstract

Resumen

La Ciencia de Datos es una disciplina que busca extraer conocimiento y generar información valiosa a partir de grandes volúmenes de datos. En este trabajo se han explorado tres de los principales algoritmos utilizados en esta área: k-vecinos más cercanos, k-medias, regresión lineal y logística. Estos algoritmos son utilizados para la clasificación y agrupamiento de datos y la predicción de valores.
Se ha llevado a cabo una descripción exhaustiva de estos algoritmos, destacando su importancia y aplicaciones. Además, se han explorado los fundamentos matemáticos que los sustentan, lo cual es fundamental para comprender su funcionamiento. Asimismo, se ha utilizado la herramienta MATLAB para trabajar con distintos ejemplos, aprovechando su potencial en análisis de datos y visualización. El principal objetivo ha sido comprender y describir de manera clara y concisa los algoritmos mencionados, enfocándose en sus fundamentos matemáticos y aplicaciones prácticas. Se ha brindado una visión general de cómo estos algoritmos pueden contribuir al campo de la Ciencia de Datos y su relevancia en la toma de decisiones.

Palabras clave: *Ciencia de Datos – Regresión lineal – Regresión logística – k-NN – k-medias*

Abstract

Data Science is a discipline that seeks to extract knowledge and generate valuable information from large volumes of data. In this project we have explored the three of the main algorithms used in this area: k-nearest neighbours, k-means, linear and logistic regression. They are used for classification and clustering of data and prediction of values.

A comprehensive description of these algorithms has been carried out, highlighting their importance and applications. In addition, the mathematical foundations underpinning them have been explored, which is fundamental to understanding how they work. The MATLAB tool has also been used to work with different examples, taking advantage of its potential in data analysis and visualisation.

The main objective has been to understand and describe clearly and concisely the algorithms mentioned, focusing on their mathematical foundations and practical applications. An overview of how these algorithms can contribute to the field of Data Science and their relevance in decision making.

Keywords: *Data Science – Algorithms – Linear regression – Logistic regression – Classification – k-NN – k-means – MATLAB*

Contenido

Agradecimientos ... III

Resumen/Abstract ... V

Introducción ... IX

1. Métodos de clasificación ... 1
1.1. k-nearest neighbors (k-NN) ... 1
1.1.1. Distancias ... 2
1.1.2. Algoritmo de k-NN ... 5
1.1.3. Elección de k ... 7
1.1.4. Aplicaciones de k-NN en aprendizaje automático ... 7
1.1.5. Ventajas y desventajas ... 8
1.1.6. Regresión con k-NN ... 9
1.1.7. Aplicación a un ejemplo real ... 10
1.2. k-medias (k-means) ... 12
1.2.1. Aplicaciones ... 12
1.2.2. El algoritmo k-medias ... 14
1.2.3. Criterio de parada ... 17
1.2.4. Elección de k ... 17
1.2.5. Ventajas y desventajas ... 18
1.3. Ejemplo de clasificación con k-medias: Compresión de imágenes ... 18

2. Métodos de regresión ... 21
2.1. Regresión lineal ... 21
2.1.1. Métodos de minimización: Descenso de gradiente estocástico ... 23
2.1.2. Tasa de aprendizaje ... 26
2.1.3. Convexidad ... 27
2.2. Regresión logística ... 28
2.2.1. Límites de decisión ... 30
2.2.2. Función de activación: la función sigmoide ... 31

2.2.3. Función de costo para la regresión logística 33
2.2.4. Interpretación probabilística 34
2.2.5. Aplicación del descenso de gradiente a la regresión logística . 35
2.2.6. Interpretación 36
2.2.7. Regresión logística polinómica y regularizada 37
2.2.8. Problemas de la regresión logística 38
2.3. Ejemplo de regresión lineal 39
2.4. Ejemplo de regresión logística 40
2.5. Ejemplo de regresión logística polinómica y regularizada 41

Bibliografía 45

Poster 49

Introducción

Actualmente, la Ciencia de Datos (*Data Science*) se ha convertido en una pieza fundamental para la toma de decisiones en diversos campos, desde la industria hasta la investigación científica. Dentro de este contexto, los algoritmos de clasificación y regresión juegan un papel crucial al permitirnos comprender, predecir y modelar los patrones subyacentes en los datos. Estos algoritmos son fundamentales para extraer información valiosa a partir de grandes volúmenes de datos y han ganado una importancia significativa en el campo del aprendizaje automático y la ciencia de datos.

La Ciencia de Datos es un campo de estudio emergente que se sustenta sobre las ciencias de la computación y la estadística. Hay varios motivos por los que su desarrollo es relativamente actual y sigue evolucionando:

- **Desarrollo de nueva tecnología:** hace posible la obtención y el almacenamiento de grandes cantidades de datos.
- **Avances en computación:** hacen posible analizar gran número de datos, en diferentes lenguajes de programación (Python, R, C++, Java...), junto con el desarrollo de nuevos, más rápidos y eficientes algoritmos y modelos.
- **Aparición de grandes compañias tecnológicas (Google, Facebook...):** permiten obtener gran variedad de datos de sus usuarios. Anteriormente a esto, los conjuntos de datos reales eran un recurso escaso y requería mucho esfuerzo obtenerlos.
- **Proceso de "datificación":** es el proceso de convertir todos los aspectos de la vida en datos. Casi cualquier movimiento en nuestra vida actual es un proceso generador de datos (GPS de dispositivos, búsquedas en Internet, "cookies"...).
- **Desarrollo de la inteligencia artificial:** a través de algoritmos dota a los ordenadores de la capacidad de realizar tareas específicas sin necesidad de ser programados, es decir, ser autónomos.

Hay dos tipos de problemas que se repiten a lo largo de la historia, la clasificación, asignar una etiqueta a un objeto entre un conjunto discreto de

posibilidades, y la regresión, pronosticar un resultado numérico futuro, en base a los valores previos. El estudio de los algoritmos de clasificación y regresión es de vital importancia debido a su amplia aplicabilidad en numerosos campos. Se utilizan en áreas como el marketing, la medicina, la biología, la economía, la investigación social y muchas otras disciplinas.

El desarrollo de algoritmos más sofisticados y precisos permite a las organizaciones y empresas obtener ventajas competitivas, mejorar la eficiencia operativa y tomar decisiones estratégicas basadas en datos fiables. Además, el estudio de estos algoritmos contribuye al avance de la ciencia y la investigación en general, al proporcionar herramientas poderosas para analizar y modelar fenómenos complejos.

Es importante a lo hora de trabajar con algoritmos tener claro los siguientes conceptos:

- **Aprendizaje supervisado:** los algoritmos de clasificación y regresión se entrenan utilizando datos etiquetados, es decir, datos que tienen una respuesta o variable objetivo conocida
- **Aprendizaje no supervisado:** se exploran datos sin etiquetas o respuestas conocidas, con el objetivo de descubrir patrones y estructuras ocultas.

Estos enfoques ofrecen distintas perspectivas para comprender y analizar los datos, según la disponibilidad de información previa.

Al estudiar estos algoritmos es fundamental utilizar conjuntos de entrenamiento ("training sets") y conjuntos de prueba ("test sets"). El conjunto de entrenamiento se utiliza para entrenar y ajustar el modelo, mientras que el conjunto de prueba se utiliza para evaluar el rendimiento del modelo en datos no vistos previamente, lo que nos permite estimar su capacidad para generalizar y hacer predicciones precisas. Se utiliza el porcentaje de aciertos respecto del total como medida de precisión. Esta división garantiza que el modelo no se sobreajuste a los datos de entrenamiento y pueda generalizar correctamente para nuevos datos.

También son importantes los conceptos de sobreajuste y subajuste. El sobreajuste ("overfitting") ocurre cuando un modelo se ajusta demasiado bien a los datos de entrenamiento, capturando incluso el ruido o las peculiaridades irrelevantes de los datos. Como resultado, el modelo puede tener un rendimiento deficiente al enfrentarse a nuevos datos, ya que no ha logrado capturar las tendencias o patrones generales en los datos subyacentes. El sobreajuste suele ocurrir cuando el modelo es demasiado complejo en relación con la cantidad limitada de datos de entrenamiento disponibles.

Por otro lado, el subajuste ("underfitting") ocurre cuando un modelo no logra capturar las relaciones y estructuras complejas presentes en los datos de

entrenamiento. En lugar de ajustarse adecuadamente, el modelo puede ser demasiado simplista o no tener suficientes características para capturar la complejidad de los datos. Como resultado, el modelo puede tener un rendimiento deficiente tanto en los datos de entrenamiento como en los datos de prueba o validación. El objetivo ideal es encontrar un equilibrio entre el sobreajuste y el subajuste, lo que se conoce como ajuste óptimo. Un modelo con ajuste óptimo generaliza bien a nuevos datos y, es capaz de capturar las relaciones y patrones relevantes en los datos, sin ajustarse demasiado a los detalles irrelevantes o sin ser demasiado simplista. Para lograr un ajuste óptimo, se pueden aplicar técnicas avanzadas como la validación cruzada, la regularización y el ajuste de parámetros, que se escapan a los objetivos de este trabajo.

Existen un amplia gama de algoritmos en Ciencia de Datos entre los que se incluyen:

- **Algoritmos de clasificación:** identificamos una serie de datos con etiquetas. Forman parte de este grupo la regresión logística, los árboles de decisión, el algoritmo de k vecinos más cercanos, "naives bayes"...
- **Algoritmos de regresión:** aquellos que mediante un conjunto de datos predicen una variable continua. Destaca la regresión lineal.
- **Algoritmos de clustering:** se dedican a agrupar los datos en diferentes grupos. Destaca el algortimo k-medias.
- **Algoritmos de reducción de dimensionalidad:** reducen el número de variables de nuestro conjunto de datos. Lo que hacen es crear nuevas variables que son combinaciones lineales de las anteriores maximizando la varianza. Se utilizan para compresión de imágenes, astronomía...
- **Algoritmos de "forecasting":** son algoritmos predictivos que utilizan las autocorrelaciones para encontrar patrones en los datos y realizar previsiones temporales. Se utilizan en previsión de ventas, logística, biología (predecir cambios poblacionales de especies en un ecosistema)...
- **Algoritmos de "deep learning":** funcionan a través de las redes neuronales. Estas redes están compuestas por unidades básicas llamadas "nodos", que están interconectadas entre sí. Cada uno realiza un cálculo en base a una función de activación, que toma como entrada una combinación lineal de los valores de entrada y produce una salida. Estas salidas son transmitidas a otras neuronas, creando así una red interconectada de nodos. Han demostrado excelentes resultados en áreas como el procesamiento del lenguaje natural, la clasificación de imágenes, el reconocimiento de voz o la traducción automática, y muchas otras aplicaciones.
- **Algoritmos de "Reinforcement learning":** son algoritmos basados en la psicología conductista. Funcionan a base de prueba, error y recompensa. Por ejemplo, para un robot cuyo objetivo es aprender a andar hasta una pared,

los movimientos que lo acerquen serán guardados positivamente; y los que no lo acerquen o lo alejen serán guardados negativamente, y por tanto, no se repetirán.

El objetivo principal de este Trabajo de Fin de Grado es comprender y analizar los algoritmos de clasificación y regresión en el contexto del aprendizaje automático y la Ciencia de Datos. Estudiar los fundamentos matemáticos sobre los que se sustentan, explorar su funcionamiento, fortalezas y limitaciones, así como su aplicabilidad en diferentes situaciones.

En el primer capítulo, se trabajará con dos algoritmos de clasificación: k-NN y k-medias. Estos algoritmos son ampliamente utilizados en el campo de la Ciencia de Datos para clasificar y agrupar datos en función de su similitud. Se analizará en detalle cómo funcionan estos algoritmos y cómo se pueden aplicar en diferentes escenarios.

En el segundo capítulo, se abordará la regresión lineal como una introducción a la regresión logística, utilizada para problemas de clasificación binaria. Se examinarán los principales fundamentos teóricos.

Cada uno de estos capítulos se cerrará con un ejemplo de aplicación de cada algoritmo para mostrar cómo se pueden aplicar estos conceptos en la práctica. Utilizaremos MATLAB como principal herramienta para la visualización y análisis de datos. Finalmente, se presentarán las conclusiones extraídas del estudio.

1

Métodos de clasificación

Como apuntamos en la introducción, los métodos de clasificación son algoritmos utilizados para categorizar objetos o datos en diferentes clases o categorías.

Existen varios tipos de métodos de clasificación, pero en general se pueden dividir en dos grandes categorías: supervisados y no supervisados. En la clasificación supervisada, se utiliza un conjunto de datos previamente etiquetados para entrenar un modelo de clasificación, que luego se utiliza para predecir la clase de nuevos datos. Algunos de los más comunes incluyen el árbol de decisión, la regresión logística, las redes neuronales y el algoritmo de k-vecinos más cercanos.

En la clasificación no supervisada, el modelo de clasificación se entrena sin etiquetas previas, lo que significa que el algoritmo debe encontrar patrones y estructuras en los datos para clasificarlos. Algunos de los métodos más populares son el algoritmo k-medias y la agrupación jerárquica.

Cada método de clasificación tiene sus propias ventajas y desventajas. La elección del método adecuado dependerá de las características de los datos y del objetivo específico de la clasificación.

En particular, en este capítulo estudiaremos los algoritmos de k-vecinos más cercanos y k-medias.

1.1. *k*-nearest neighbors (*k*-NN)

El algoritmo de k-vecinos más cercanos es un algoritmo de aprendizaje supervisado desarrollado por primera vez por Evelyn Fix y Joseph Hodges en 1951. Utiliza la proximidad para hacer clasificaciones de nuevas muestras o predicciones. Si bien se puede usar para problemas de regresión, generalmente se usa como algoritmo de clasificación.

En la clasificación, la salida es la etiqueta de una clase. A una nueva observación se le asigna como etiqueta la que tiene la mayoría de las observaciones más

cercanas (vecinos). Dependiendo de cuántos vecinos consideremos obtendremos un resultado u otro.

En la regresión, la salida es el valor promedio de los valores de los k-vecinos más cercanos.

Figura 1.1. Ejemplo del algoritmo (K-NN) para la clasificación [27]

En definitiva, el objetivo del algoritmo es identificar a los vecinos más cercanos de una nueva observación, de modo que podamos asignar una etiqueta de clase a ese punto o predecir su valor, como puede observarse en la Figura 1.1. Para automatizarlo tenemos que tener dos conceptos claros: cómo definir cercanía y cuántos vecinos deberíamos tomar como referencia para asignar la etiqueta.

En primer lugar, dado que este algoritmo se basa en la distancia para la clasificación, si las variables se presentan en diferentes unidades físicas o vienen en escalas muy diferentes, es conveniente normalizar los datos de entrenamiento para mejorar la precisión.

Luego, para trabajar con estos datos, tenemos que escoger la distancia más apropiada, dependiendo de cada caso. Veamos los distintos tipos de distancias que podemos utilizar.

1.1.1. Distancias

La noción de cercanía y similaridad varía según el contexto. Los matemáticos probablemente hayan pensado en la distancia euclídea o la distancia Manhattan en estos momentos. Sin embargo, en muchos contextos estas distancias no son aplicables. Por ejemplo, si nos referimos a la cercanía en una red social, no podemos usar estas distancias usuales, la podríamos definir como el número de amigos en común.

Para determinar qué observaciones están más cerca de un punto consulta determinado, será necesario calcular la distancia entre este punto y los del

conjunto de entrenamiento. Veamos las distancias más relevantes en la práctica [11], [27]:

- **Distancia euclídea:** la podemos utilizar para variables cuantitativas reales en el plano o en un espacio multidimensional. En general, la distancia euclidiana entre los puntos $P = (p_1, p_2, \cdots, p_n)$ y $Q = (q_1, q_2, \cdots, q_n)$ del espacio euclídeo n-dimensional, se define como,

$$d_E(P, Q) = \sqrt{\sum_{i=1}^{n} (p_i - q_i)^2}$$

- **Distancia Manhattan (taxi):** también se utiliza para dos vectores reales n-dimensionales. La forma más clara de visualizar esta distancia es imaginando un taxi conduciendo a través de una ciudad. No mide la distancia en línea recta, sino que considera la distancia como la suma de los catetos, como puede observarse en la Figura 1.2. Formalmente, se define como:

$$d_M(P, Q) = \sum_{i=1}^{n} |p_i - q_i|$$

Figura 1.2. La línea azul representa la distancia Euclídea, mientras que la línea roja representa la distancia Manhattan [11]

- **Similitud coseno:** mide el ángulo entre dos vectores (tomemos $\vec{P}$ y $\vec{Q}$) en un espacio que posee un producto interior, para saber si apuntan, o no, en la misma dirección. El resultado es un valor entre -1 (exactamente opuestos) y 1 (en la misma dirección). Si son ortogonales el valor es 0. Se calcula de la siguiente forma:

$$\cos(\vec{P}, \vec{Q}) = \frac{\vec{P} \cdot \vec{Q}}{\|\vec{P}\| \cdot \|\vec{Q}\|}.$$

Como queremos que la distancia entre dos vectores que tengan el mismo sentido y la misma dirección sea 0, debemos tomar

$$d_{SC} = 1 - \cos(\vec{P}, \vec{Q}).$$

Esta distancia se emplea frecuentemente en los sistemas de recomendación online. Funciona situando los vectores en el espacio considerado, por ejemplo, si tomamos unas películas, siendo su medida el "valor rating" de cada usuario a esa película. Luego calcula el ángulo entre los vectores partiendo del cero. Cuanto menor sea el ángulo entre vectores, más cerca estarán los usuarios en cuanto a su gusto en cine. También se usa en el *pose matching*, donde se busca establecer la similitud entre dos posiciones de una imagen.
La similitud coseno no debe ser considerada como una métrica debido a que no cumple la desigualdad triangular.

- **Coeficiente de Jaccard:** mide el grado de similitud entre dos conjuntos, sea cual sea el tipo de elementos. Se define como la cardinalidad de la intersección de ambos conjuntos dividida por la cardinalidad de su unión, es decir,

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|}.$$

Por ejemplo, si tenemos la lista de amigos de Alba y de Belinda,

$$A = \{Daniel, Pablo, Carla, Marta\}$$

$$B = \{Oliver, Carla, Daniel, David, Paula\}$$

la distancia entre ellas es,

$$J(A, B) = \frac{2}{7}.$$

Siempre toma valores entre 0 y 1, y este último indica la igualdad entre ambos conjuntos. Se utiliza, por ejemplo, en ecología, para medir la relación de presencia-ausencia entre el número de especies comunes en dos áreas y en el número total de especies. Puede verse el siguiente artículo [3] en el que hacen uso de dicha distancia.

- **Distancia Hamming:** se usa para medir la distancia entre dos cadenas o palabras de la misma longitud. Se trata de revisar si los elementos de cada posición son iguales, si no lo son, se suma una unidad. Por ejemplo, la distancia entre 1234 y 1284 es 1, o la distancia entre gato y rata es 2. Cuanto mayor sea esta diferencia, menor es la posibilidad de que un código válido se transforme en otro código válido por una serie de errores. Si dos palabras de código difieren en una distancia d, se necesitan d errores para convertir una en la otra. Como se puede apreciar en la Figura 1.3.

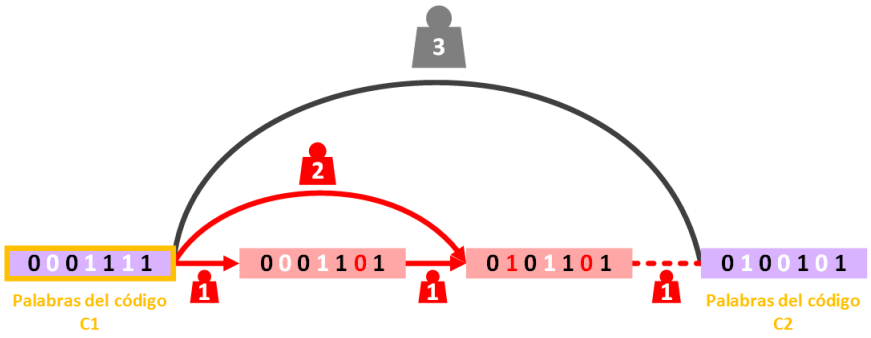

Figura 1.3. Distancia Hamming [14]

Esta distancia se utiliza para definir algunas nociones esenciales en teoría de códigos, tales como códigos detectores y correctores de errores.

Existen muchas más distancias disponibles dependiendo del tipo de datos con el que estemos trabajando, como puede ser la distancia Mahalanobis, la distancia de Minkowski... Pueden encontrarse más ejemplos en [27] y [9].

Ya conocemos las principales medidas de distancia que se suelen utilizar en el algoritmo k-NN. Ahora cabría preguntarse cuándo deberíamos usar cada una de ellas. La distancia que usemos dependerá del tipo de dato que tengamos, las dimensiones y el objetivo del estudio.

Cuando tenemos un nivel de dimensionalidad alto, es decir, cuando hay muchas variables, la distancia de Manhattan funciona mejor que la distancia euclídea. Otra opción suele ser mirar la dirección de los vectores, es decir, usar la distancia del coseno.

Por otro lado, se puede utilizar la distancia Jaccard si no importa la duplicidad de los datos, ya que estamos trabajando con conjuntos. Esto significa que no se considera que los elementos duplicados sean relevantes para el cálculo de similitud. Y, en caso contrario, se puede usar otra distancia como la del coseno.

Ahora que ya conocemos las diferentes medidas de distancia y cuándo usar cada una de ellas, vamos a usar estas distancias para encontrar a los k vecinos más cercanos y, a partir de ahí, ver cómo se hace la predicción.

1.1.2. Algoritmo de k-NN

Imaginemos que tenemos la situación de la Figura 1.4. Los puntos azules y rojos son observaciones del conjunto de entrenamiento que ya están clasificadas. Dada una nueva observación, punto verde, queremos clasificarlo en la clase azul o roja, dependiendo de sus k vecinos más cercanos.

La intuición nos sugiere tomar la distancia euclídea para medir la cercanía, en este contexto, del punto verde a todas las demás observaciones. Podemos tomar, por ejemplo, $k = 3$. Se puede observar en la Figura 1.5 que el punto

verde está rodeado de observaciones rojas, por lo que su etiqueta será 'ROJO'. Podría decirse que se han obtenido 3 votos para la clase roja y 0 votos para la clase azul.

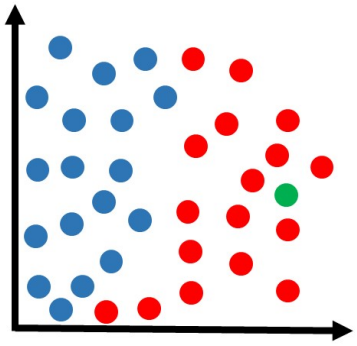

Figura 1.4. Punto verde sin clasificar [33]

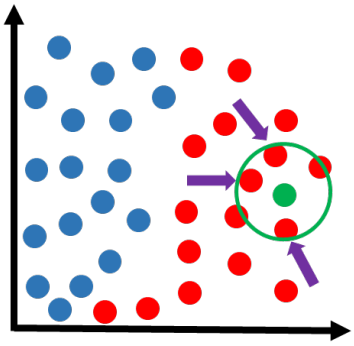

Figura 1.5. Punto verde sin clasificar k=3

Sin embargo, este ha sido un caso bastante sencillo donde no hay lugar a dudas. Vamos a fijarnos en el ejemplo de la Figura 1.6. Usando el método para $k = 3$, obtenemos 2 votos para la clase roja y 1 voto para la clase azul, por lo que su etiqueta será 'ROJO'.

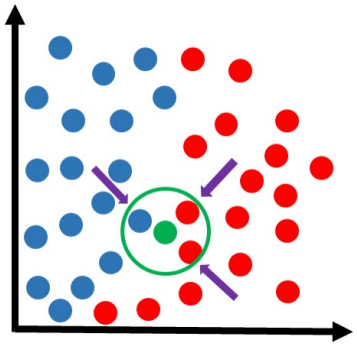

Figura 1.6. Punto verde sin clasificar k=3

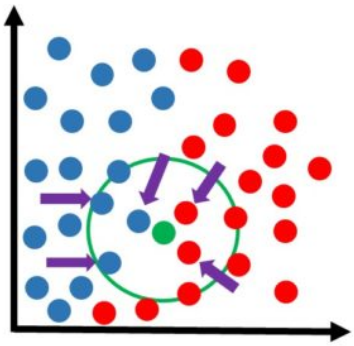

Figura 1.7. Punto verde sin clasificar k=5

Sin embargo, si usamos el método para $k = 5$, como puede verse en la figura 1.7, obtenemos 3 votos para la clase azul y 2 votos para la clase roja, por lo que su etiqueta será 'AZUL'.

Por lo tanto, para este segundo caso, hemos obtenido diferentes resultados dependiendo del número de vecinos que consideremos. Queda claro entonces que la elección de este valor es muy importante para los resultados del estudio.

Cuando k es pequeño, la clasificación es más sensible a regiones muy cercanas (puede ocurrir un problema de sobreajuste). Con k grande, la clasificación puede considerarse más robusta, pero si k es demasiado grande, puede haber un problema de subajuste. Por ello, veremos la forma de escoger el valor de k en la siguiente subsección.

Podemos resumir el proceso realizado al aplicar el algoritmo k-NN en los siguientes pasos:

1. Decidir cómo vamos a considerar la cercanía y escoger la distancia más adecuada.
2. Decidir el valor de k.
3. Recibir un dato sin clasificar y medir su distancia a cada uno de los demás datos que ya están clasificados.
4. Seleccionar las k distancias más pequeñas.
5. Contar el número de veces que apareció cada clase en esas k seleccionadas.
6. Clasificar el nuevo dato como perteneciente a la clase que más se repitió.

En cualquier algortimo de aprendizaje automático, lo más habitual es tener una fase de entrenamiento donde se crea el modelo y se entrena. Luego, hay una fase de prueba donde se usan nuevos datos para ver cómo de bien funciona el modelo.

En el caso de k-NN, la fase de entrenamiento consiste en leer los datos ya clasificados. Para valorar lo bien que funciona el modelo, necesitamos datos ya clasificados, que se extraen del conjunto de entrenamiento para la fase de prueba. Una vez hemos comprobado que funciona correctamente, podemos tratar de clasificar nuevos datos de los que no conocemos su etiqueta [33], [18], [6], [27].

1.1.3. Elección de k

El valor k en el algoritmo k-NN define cuántos vecinos se tendrán en cuenta para determinar la clasificación de un punto de consulta específico. Por ejemplo, si $k = 1$, se asignará la misma clase que su vecino más cercano. Algunos valores de k pueden llevar a un ajuste excesivo o insuficiente, como sucedía en la Figura 1.6.

La elección de k dependerá en gran medida de los datos de entrada, ya que los datos con más valores atípicos o ruido probablemente funcionarán mejor con valores más altos de k. En general, se recomienda tener un número impar para evitar empates en la clasificación.

1.1.4. Aplicaciones de k-NN en aprendizaje automático

El algoritmo k-NN tiene variedad de aplicaciones:

- **Preprocesamiento de datos:** los conjuntos de datos suelen tener valores faltantes, el algoritmo k-NN puede estimar esos valores en un proceso conocido como imputación de datos faltantes. El objetivo es proporcionar una representación completa de los datos para que puedan ser utilizados en análisis estadísticos o modelos de aprendizaje automático. [34]
- **Motores de recomendación:** utilizando datos de flujo de clics de sitios web, el algoritmo se ha utilizado para proporcionar recomendaciones automáticas a los usuarios sobre contenido adicional. Un usuario está asignado a un grupo en particular y, en función del comportamiento del usuario de ese grupo, se le da una recomendación. Sin embargo, dados los problemas de escala con k-NN (que veremos en la próxima subsección), este enfoque puede no ser óptimo para conjuntos de datos muy grandes.
- **Finanzas:** se ha utilizado en una variedad de casos de uso económico y financiero. Por ejemplo, el uso de k-NN en datos crediticios puede ayudar a los bancos a evaluar el riesgo de un préstamo para una organización o individuo, es decir, para determinar la solvencia crediticia de un solicitante de préstamo.
- **Salud:** se ha aplicado dentro de la industria de la salud, haciendo predicciones, por ejemplo, sobre el riesgo de ataques cardíacos y cáncer de próstata. En [5] se hace uso del algoritmo para dividir a los pacientes que sobreviven a un infarto de miocardio en grupos de riesgo arrítmico alto y bajo. Distinguir entre estos dos grupos es de crucial importancia ya que se ha demostrado que el grupo de alto riesgo se beneficia de la inserción de un desfibrilador cardioversor implantable; un procedimiento quirúrgico costoso con complicaciones potenciales y sin ventajas comprobadas para el grupo de bajo riesgo.
- **Reconocimiento de patrones:** esto ha sido particularmente útil para identificar números escritos a mano.

Se puede utilizar para conocer la cercanía de nuevos productos a los similares de sus competidores. Clasificar estudiantes según su asistencia para predecir si aprobarán o no. Estos conocimientos permitirían a la administración tomar medidas oportunas. En definitiva, se trata de un algoritmo muy versátil. Se pueden encontrar más ejemplos en [27] y [11].

1.1.5. Ventajas y desventajas

El algoritmo k-NN destaca por ser fácil de implementar y de entender. Además, se ajusta para tener en cuenta cualquier dato nuevo, ya que todos los datos de entrenamiento se almacenan en la memoria, y solo requiere escoger el valor de k y la métrica a utilizar.

Sin embargo, no funciona bien con entradas de datos grandes y de alta dimensión, ocupa más memoria y almacenamiento de datos en comparación con otros clasificadores, lo que aumenta el costo comercial y el tiempo de procesado.

1.1.6. Regresión con k-NN

El algoritmo k-NN en regresión se basa en tomar las observaciones más cercanas y calcular una media. Como se observa en la Figura 1.8.

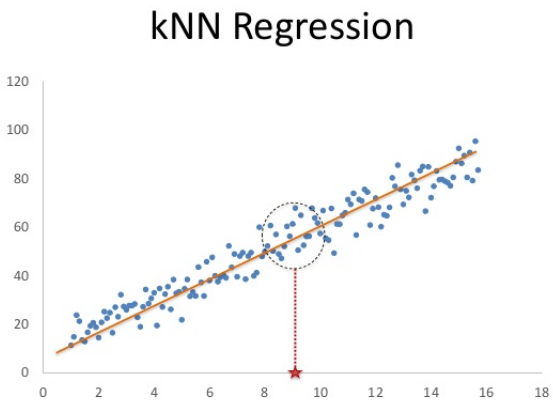

Figura 1.8. Aplicación de k-NN a regresión lineal [10]

Se utiliza para estimar una variable continua X. El modelo entrenado es una función, $\hat{X}$, que a cada elemento, v, de fuera del conjunto de entrenamiento, M, asigna la media de $\{X(m_i)|1 \leq i \leq k\}$ donde $\{m_i|1 \leq i \leq k\}$ son los k elementos de la muestra más próximos a v.

La regresión lineal utiliza una función para modelar la relación entre las variables independientes y la variable dependiente. Mientras que el algoritmo k-NN se basa en la similitud de las muestras de entrenamiento para predecir el valor de la variable dependiente. Se utiliza la información de las k muestras de entrenamiento más cercanas (vecinos) para realizar una predicción. Respecto a la regresión lineal presenta una ventaja, si los datos contienen valores atípicos o la relación no es lineal, los resultados de la regresión lineal pueden verse afectados. Mientras que el algoritmo k-NN es menos sensible a los valores atípicos, ya que se basa en la similitud con las muestras de entrenamiento cercanas y no en suposiciones específicas sobre la relación.

Vamos a tener un buen ajuste si hay mucha densidad de puntos y bajo ruido. Sin embargo, sin los suficientes datos, no tiene sentido llevar a cabo una regresión con el algoritmo k-NN [10].

1.1.7. Aplicación a un ejemplo real

Ya sabemos los conceptos generales sobre el algoritmo k-NN: las distintas medidas de distancia que se puede usar y cuándo usar cada una de ellas, cómo elegir la k... Así que vamos a aplicar el método a un ejemplo con datos reales.

Tomaremos 257 registros con opiniones de usuarios sobre una aplicación. Utilizaremos 2 variables para poder graficar en 2 dimensiones, pero es aplicable a cualquier dimensión. Las variables que utilizaremos serán: la cantidad de palabras utilizadas, que van desde 1 a 103; y emoción, con un valor entre -4 y 4 que indica si el comentario fue negativo o positivo. Nuestras etiquetas, serán las estrellas que dieron los usuarios a la aplicación, que son valores discretos del 1 al 5. Teniendo la información representada en la Figura 1.9.

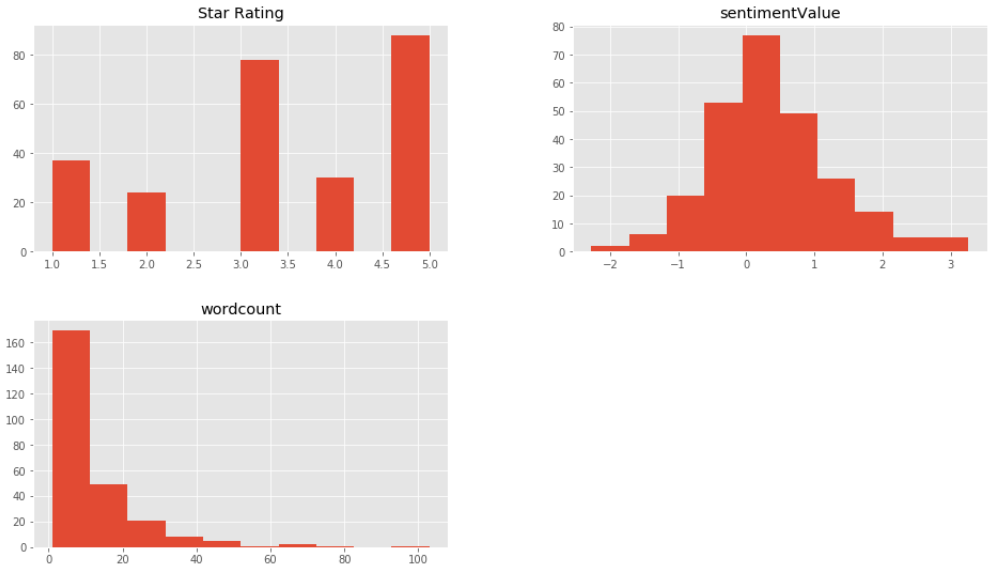

Figura 1.9. Histogramas de las variables comentadas [8]

Vemos que la distribución de estrellas no está balanceada, los valores que más se repiten son 3 y 5 estrellas. Es esencial para garantizar la precisión y objetividad del estudio. La cantidad de palabras se centra sobre todo de 0 a 10.

Vamos a tomar el valor de $k = 7$ y la distancia euclídea. La gráfica con la clasificación obtenida nos ayudará a ver donde caerán las predicciones.

Tomamos una parte de los datos como conjunto de entrenamiento, este se toma ya clasificado; y otro como conjunto de prueba. Representamos los datos de entrenamiento en función de la emoción y el número de palabras. Como disponemos de las etiquetas, se van a distinguir 5 zonas en la representación (Figura 1.10). Ahora, podemos clasificar los datos del conjunto de prueba mediante sus

7 vecinos más cercanos, que caerá en una de las 5 zonas marcadas. Se tiene una precisión del 86 % para el conjunto de prueba.

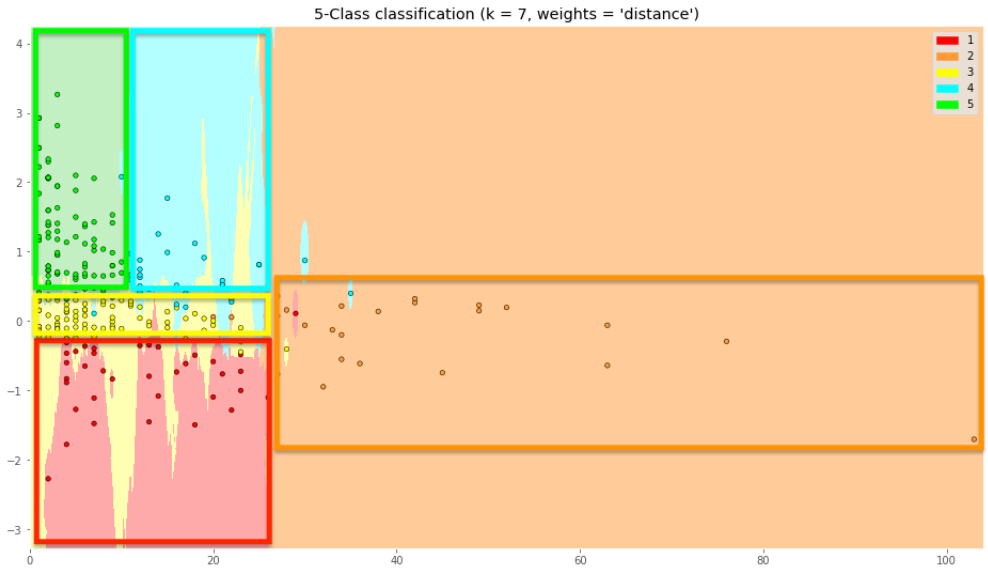

Figura 1.10. Clases obtenidas en la representación

Con estos resultados, podemos intuir ciertas características de los usuarios que usan y valoran la aplicación:

- Los usuarios que ponen 1 estrella tienen emoción negativa y escriben hasta 25 palabras.
- Los usuarios que ponen 2 estrellas dan muchas explicaciones (hasta 100 palabras) y su emoción tiende a ser moderadamente negativa.
- Los usuarios que ponen 3 estrellas son bastante neutrales en emoción, están en torno al cero y escriben de 0 a 25 palabras.
- Los usuarios que dan 5 estrellas tienen una impresión positiva y escriben pocas palabras (menos de 10).

Es decir, ya podemos intuir la puntuación que dejará cualquier nuevo usuario en la aplicación teniendo en cuenta el número de palabras y la intención del texto de su comentario. Por ejemplo, un usuario que comente "Fácil de usar", lo clasificamos en el grupo de 5 estrellas.

Para obtener el mejor valor de k, ejecutamos el algoritmo para distintos valores y comparamos la precisión, es decir, el porcentaje de observaciones etiquetadas correctamente del conjunto de prueba, respecto del total. En la figura 1.11 vemos que con valores entre $k = 7$ y $k = 14$ obtenemos la mayor precisión (86 %) [8].

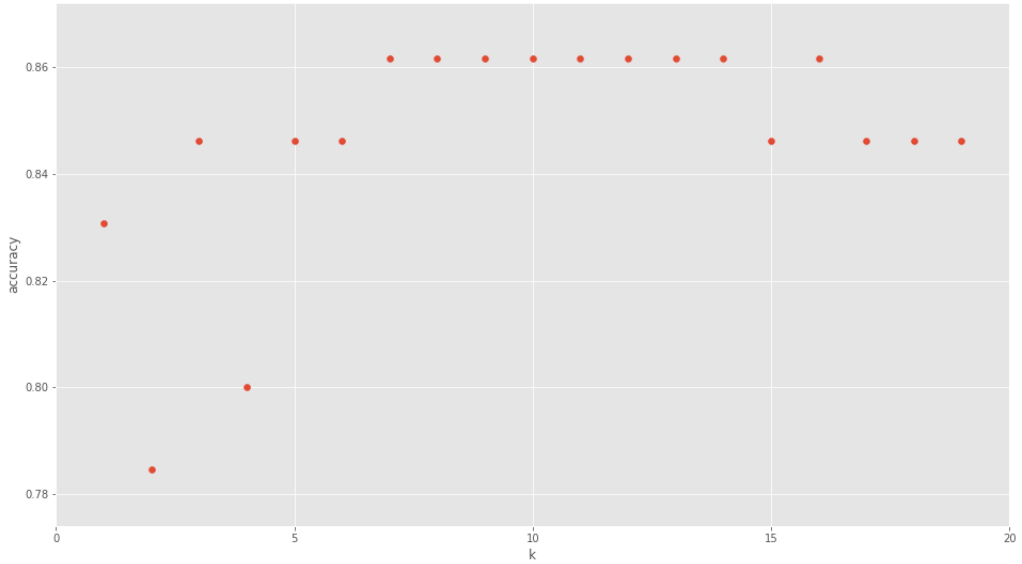

Figura 1.11. Precisión obtenida al variar el valor de k

1.2. *k*-medias (*k*-means)

El algoritmo k-medias es un método de aprendizaje automático no supervisado, es decir, el algoritmo procesa un conjunto de datos sin que se le proporcionen etiquetas o resultados esperados. Su objetivo es encontrar k grupos (o clústeres) de puntos de datos similares en un conjunto dado.

El resultado final es una partición de los puntos de datos en k grupos, donde los puntos dentro de cada grupo son similares entre sí y diferentes respecto a los de los otros grupos. Es conocido por su simplicidad, eficiencia y facilidad de implementación.

1.2.1. Aplicaciones

Se puede utilizar en una amplia variedad de aplicaciones como:

- **Segmentación de clientes:** puede utilizarse para dividir a los clientes en diferentes grupos en función de sus características o comportamientos (historial de compras, actividades en la web...). Esto puede ser útil para campañas de marketing más personalizadas o para tomar decisiones de negocio.
- **Clasificación de texto:** como la agrupación de noticias según su contenido o la clasificación de documentos según su temática. Por ejemplo, en el estudio [4] se trata de determinar la cercanía lingüística entre distintos medios de prensa aplicando el algoritmo k-medias.
- **Detección de anomalías:** puede utilizarse para detectar patrones anormales en un conjunto de datos y señalar posibles problemas o errores. Se ha utilizado

en la identificación de patrones en los datos de transacciones financieras, lo que puede ayudar a detectar fraudes. Para ello, se agrupa la actividad válida y se detectan los valores atípicos.

- **Análisis de datos biológicos:** como la identificación de patrones en datos de expresión génica o la segmentación de imágenes de células. Por ejemplo, en [1] se utiliza el algoritmo k-medias para el reconocimiento de células de leucemia linfoblástica aguda en imágenes microscópicas.
- **Reconocimiento de patrones de tráfico:** para identificar congestiones o patrones de movimiento de vehículos en ciertas áreas, lo que puede ayudar a la planificación de la infraestructura vial.
- **Segmentación de imágenes:** como la clasificación de píxeles en áreas de diferentes colores o texturas, lo que puede ayudar en la identificación de objetos o patrones en las imágenes. Se agrupan los píxeles para separar los elementos significativos de una imagen, y así, poder extraer cierta información de alguno de ellos. Por ejemplo, calcular el tamaño de un tumor a partir de imágenes médicas, el porcentaje de mica en una roca granítica o el área de un lago a partir de una foto aérea (Figura 1.12). Veremos un ejemplo aplicado al final del capítulo.

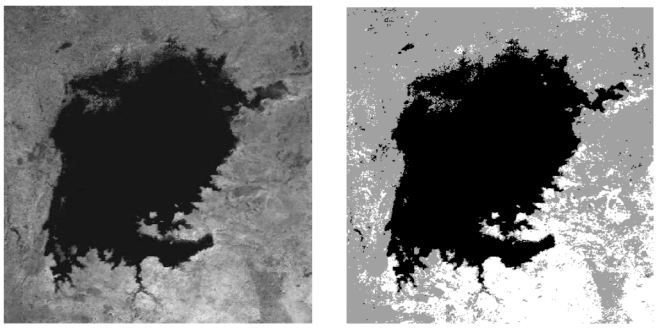

Figura 1.12. La imagen de la izquierda muestra la imagen original en blanco y negro. La imagen de la derecha se han agrupado los píxeles en tres clústeres (blanco, gris y negro) y se ha reconstruido la imagen utilizando la intensidad asignada a cada píxel. Contando el número de píxeles negros se puede calcular el porcentaje respecto al número total de píxeles de la foto y obtener el porcentaje del área del lago respecto al área total representada por la foto.[29]

Como ilustramos con estos ejemplos, el método k-medias es un algoritmo con muchos usos potenciales y muy versátil ya que puede ser utilizado para casi cualquier tipo de clustering de datos [20], [13].

1.2.2. El algoritmo *k*-medias

Como se había mencionado, el algoritmo k-medias es un método de agrupamiento que divide un conjunto de datos en k grupos. Para ello, primero se especifica el número de clústeres deseados (k), por ejemplo, al establecer $k = 4$, el conjunto de datos se agrupará en 4 grupos.

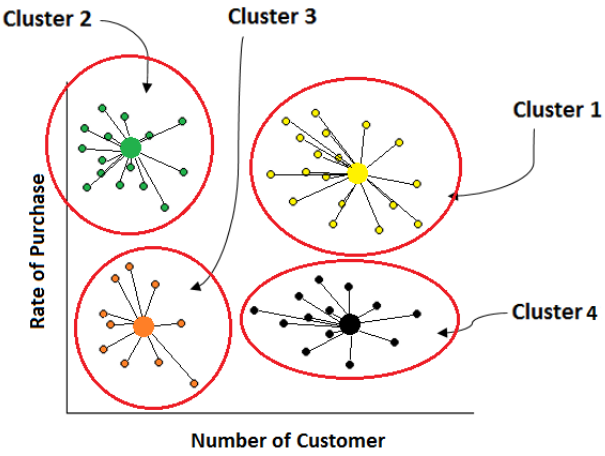

Figura 1.13. Resultado del algoritmo k-medias [26]

Cada grupo está representado por su centro o centroide (los puntos más gruesos de la Figura 1.13), que corresponde a la media aritmética de los puntos de datos asignados al grupo.

De esta manera, el algoritmo funciona a través de un proceso iterativo hasta que cada punto de datos está más cerca del centroide de su propio grupo que de los centroides de otros grupos, minimizando la distancia del punto dentro del grupo en cada paso.

Este es un algoritmo versátil que puede ser utilizado para cualquier tipo de agrupación. Por ejemplo, imagina a un bibliotecario que quiere ordenar un montón de libros diferentes en 4 estanterías. El objetivo es colocar libros similares en la misma estantería. Lo que haría es escoger 4 libros, uno para cada estantería, para establecer un tema. Estos libros dictarán cuál de los libros restantes irá en cada grupo.

Cada vez que se toma un libro nuevo, se compara con los primeros, y se clasifica en la estantería que tenga libros similares, repitiendo este proceso hasta que todos los libros hayan sido colocados.

Una vez que haya terminado, cabe destacar que cambiar el número de estanterías y tomar diferentes libros iniciales (cambiando el tema para cada estantería), aumentaría la eficacia con la que se agrupan los libros. Por lo tanto,

se repite el proceso con la esperanza de un mejor resultado. El algoritmo k-medias trabaja de esta forma.

El algoritmo k-medias resuelve un problema de optimización, siendo la función a optimizar (minimizar) la suma de las distancias cuadráticas de cada punto al centroide de su clúster.

Dado un conjunto de observaciones $(\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n)$ donde cada observación es un vector real de d dimensiones. El algoritmo k-medias construye una partición en k conjuntos ($k \leq n$) con el fin de minimizar la suma de los cuadrados dentro de cada grupo, $\boldsymbol{S} = \{S_1, \ldots, S_k\}$. Es decir,

$$\underset{\boldsymbol{S}}{\operatorname{argmin}} = \sum_{i=1}^{k} \sum_{\boldsymbol{x}_j \in S_i} \|\boldsymbol{x}_j - \boldsymbol{\mu}_i\|^2$$

donde $\boldsymbol{\mu}_i$ es la media de los puntos de S_i.

Paso a paso funciona de la siguiente forma ([20], [30]):

1. **Especificar el número de clústeres k.**
2. **Seleccionar k puntos al azar del conjunto de datos como los centroides iniciales de cada clúster:** sean $\boldsymbol{m}_1^1, \ldots, \boldsymbol{m}_k^1$, un conjunto inicial de k centroides.
3. **Asignar cada punto del conjunto de datos al clúster cuyo centroide esté más cerca:** Para hacer esto, se calcula la distancia entre cada punto y cada centroide, y se asigna el punto al clúster cuyo centroide tenga la menor distancia con el punto.

$$S_i^{(t)} = \{\boldsymbol{x}_p / \|\boldsymbol{x}_p - \boldsymbol{m}_i^t\| \leq \|\boldsymbol{x}_p - \boldsymbol{m}_j^t\|, \forall\, 1 \leq j \leq k\}$$

donde cada $\boldsymbol{x}_p$ va exactamente dentro de un $S_i^{(t)}$, incluso aunque pudiera ir en dos de ellos.

4. **Recalcular los centroides de cada clúster:** Se recalculan como la media de todos los puntos del clúster. Esto significa que se actualiza la posición del centroide para reflejar la nueva agrupación.

$$\boldsymbol{m}_i^{(t+1)} = \frac{1}{|S_i^{(t)}|} \sum_{\boldsymbol{x}_j \in S_i^{(t)}} \boldsymbol{x}_j$$

5. **Repetir los pasos 3 y 4 hasta que se alcance el criterio de parada.**

Este proceso se ve reflejado en las Figuras 1.14, 1.15 y 1.16.

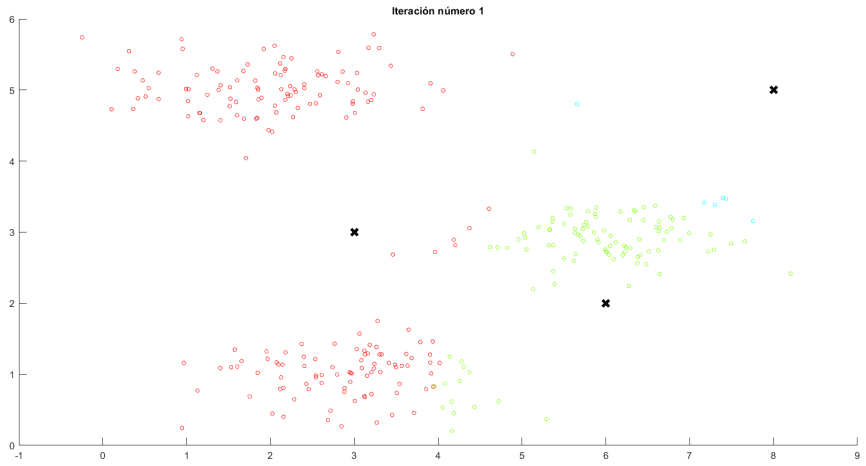

Figura 1.14. Primera iteración

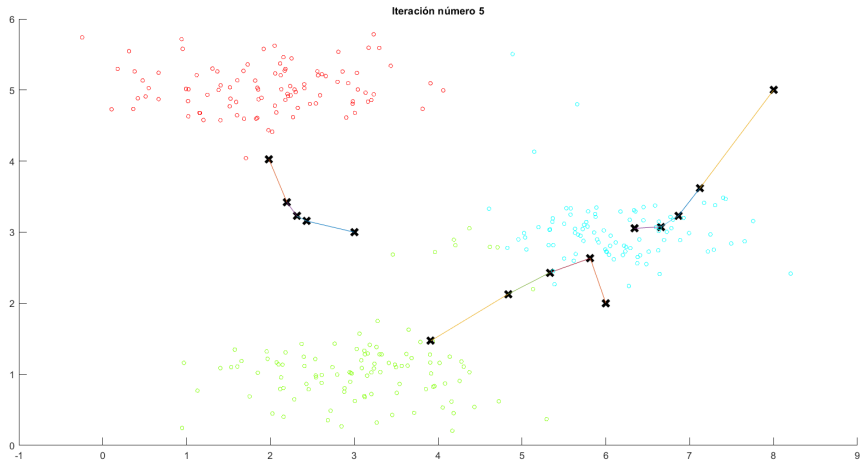

Figura 1.15. Quinta iteración

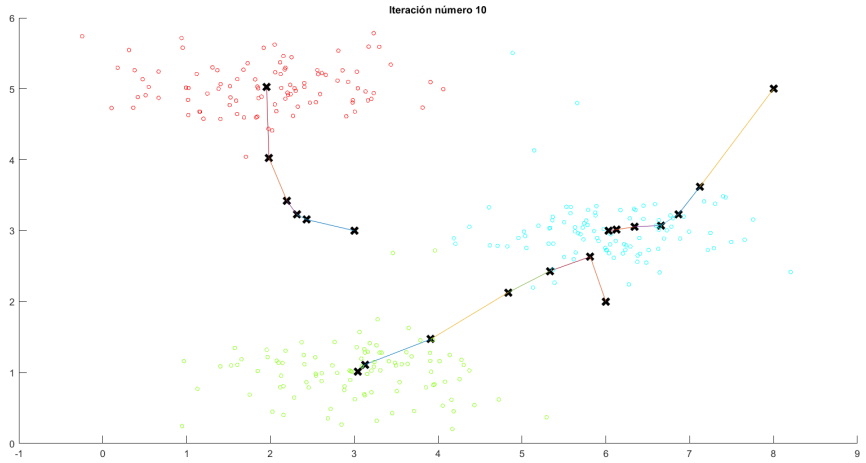

Figura 1.16. Décima iteración

1.2.3. Criterio de parada

El criterio de parada indica a nuestro algoritmo cuándo dejar de actualizar los clústeres. Establecer un buen criterio de parada no necesariamente devolverá los mejores clústeres, pero nos asegura que devuelve grupos razonablemente buenos.

Hay tres criterios de parada principales, que se pueden adoptar para detener el algoritmo [13]:

- **Los centroides de los clústeres ya no cambian.**
- **Los puntos permanecen en el mismo grupo.**
- **Se alcanza el número máximo de iteraciones** (previamente establecido).

1.2.4. Elección de k

En este algoritmo elegimos el número de grupos en el que se van a dividir los datos. Como esta es una decisión relevante en el algoritmo, existen varias formas de encontrar el valor óptimo de k [20]:

- **Método del codo:** se grafican los valores de k junto con la suma de los errores cuadrados (la suma de la distancia de cada punto al centroide de su clúster) para cada valor de k. A medida que aumenta k, el error disminuye, pero a un ritmo cada vez menor. El punto en el que disminuye más lentamente se conoce como el codo y es el punto óptimo para elegir k. Puede observarse en la Figura 1.17.
- **Técnicas de validación cruzada:** implica dividir el conjunto de datos en k subconjuntos y entrenar el algoritmo k veces, cada vez utilizando un subconjunto diferente. Comparando los resultados obtenidos en cada caso.

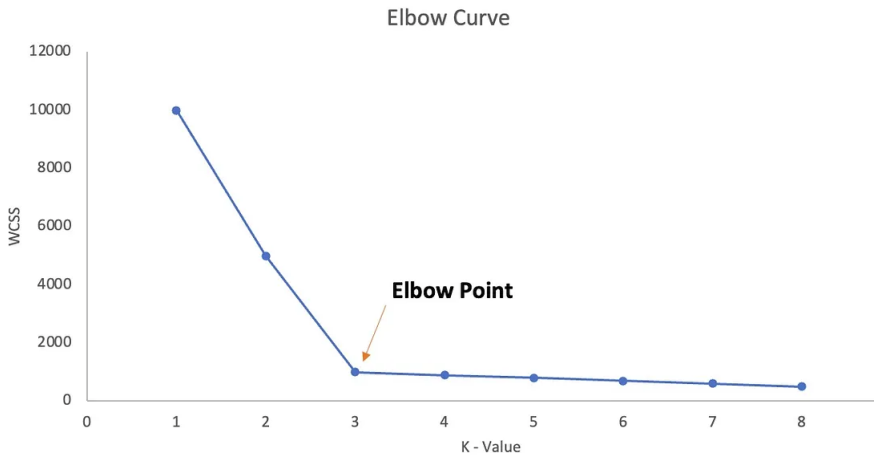

Figura 1.17. Método del codo [24]

1.2.5. Ventajas y desventajas

Este algoritmo presenta algunas desventajas [13]:

- Es necesario decidir personalmente el número de clústeres y esta decisión puede afecta los resultados.
- Puede converger a un mínimo local en lugar del óptimo global, lo que puede llevar a resultados diferentes si se ejecuta varias veces con diferentes inicializaciones.
- Cada clúster tiene, aproximadamente, la misma cantidad de observaciones, y no siempre es lo adecuado.
- Es muy sensible a los valores atípicos y a los datos ruidosos. Adicionalmente, asume que los puntos de datos en cada clúster están localizados dentro de una esfera alrededor del centroide del clúster, por lo que no es adecuado para clústeres de forma irregular.
- En general, la interpretabilidad del resultado final puede suponer un problema.

Sin embargo, también presenta una serie de ventajas:

- Es relativamente fácil de implementar y es eficiente en términos de tiempo de procesamiento, especialmente para conjuntos de datos de gran tamaño.
- Se adapta fácilmente a nuevas observaciones.
- Se puede generalizar a clústeres de diferentes tamaños y formas, como clústeres elípticos.
- Tiene amplias aplicaciones en múltiples campos: marketing, computación...

1.3. Ejemplo de clasificación con k-medias: Compresión de imágenes

En una imagen, cada píxel se representa como tres enteros de 8 bits sin signo (que van de 0 a 255) que especifican los valores de intensidad de rojo, verde y azul. Esta codificación a menudo se denomina la codificación RGB. Una imagen contiene miles de colores, lo que hace el algoritmo de k-medias es reducir este número de colores [2]. Al hacer esta reducción, es posible comprimir la foto de manera eficiente.

Utilizaremos el algoritmo k-medias para agrupar por similitud todos los colores que aparecen en una imagen en 16 colores (centroides). Estos colores son los representantes de cada clase y cada color de la imagen será sustituido por el color de su clase, obteniendo así, la imagen comprimida. De esta manera, se reduce significativamente el número de bits que se requieren para describir la imagen.

Primero, cargamos la imagen en un formato array de dimensiones: altura x ancho x canales. Estos canales son los RGB.

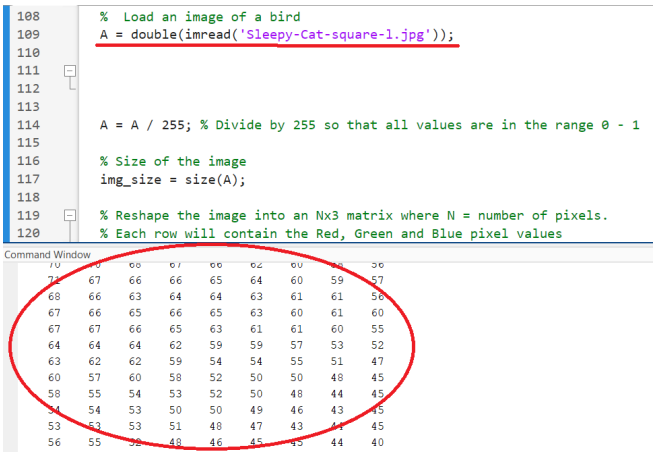

Figura 1.18. Visualización de los píxeles con sus bits

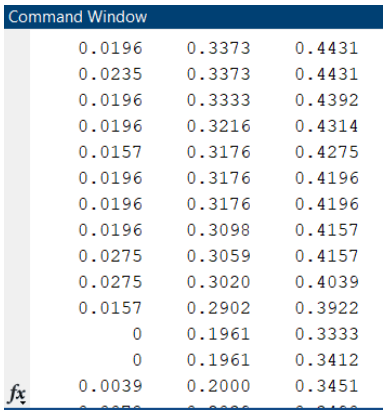

Figura 1.19. Matriz con los colores de cada píxel

Transformamos nuestra imagen a una matriz de $nx3$ donde n es la cantidad de bits de la imagen (128x128 en este caso). Cada fila tendrá la intensidad del píxel Rojo, Azul o Verde (los valores se muestran de 0 a 1 puesto que los hemos dividido entre 255). Sobre esta matriz utilizaremos el algoritmo k-medias (Figura 1.19).

Como usamos 16 centroides, tomamos 16 puntos al azar de nuestros datos y ejecutamos el algoritmo k-medias 10 veces (ya que lo hemos prefijado así) con la distancia euclídea, de modo que obtenemos el cluster más cercano a cada píxel. Con este algoritmo guardamos los 16 centroides (que representan un color) y el índice de la clase a la que pertenece cada píxel (Figura 1.20).

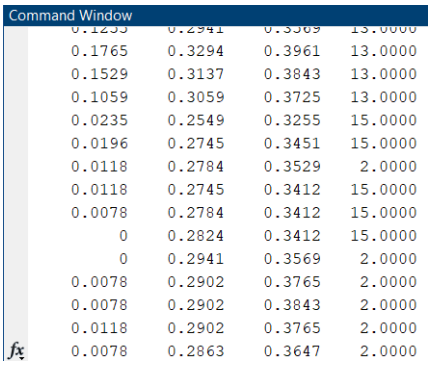

Figura 1.20. Las tres primeras columnas representan el valor del píxel y la última el cluster al que está asociado

Ahora podemos recuperar la imagen desde los índices (el cluster al que corresponde cada píxel) mapeando cada píxel al valor del centroide de dicho píxel. Obteniendo la Figura 1.21. Se pueden ver los efectos de la compresión reconstru-

yendo la imagen basada con las asignaciones del algoritmo, reemplazando cada píxel con el valor del centroide asignado.

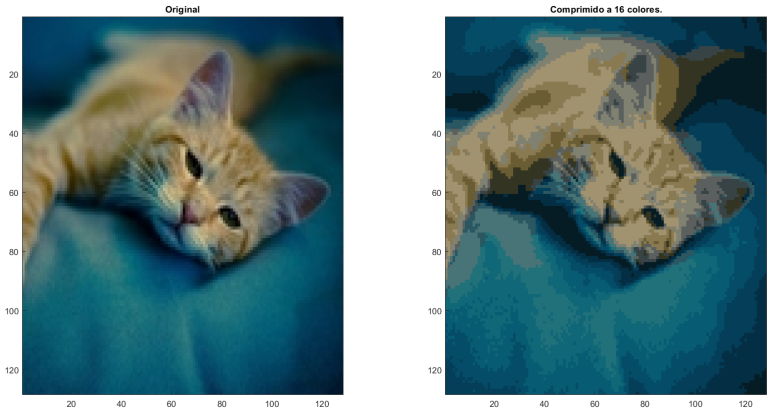

Figura 1.21. Imagen original a la izquierda. Imagen comprimida a 16 colores a la derecha

Por ejemplo, si la imagen original requiere 24 bits para cada una de las ubicaciones de 128×128 píxeles, resulta en un total de $128 \times 128 \times 24 = 393.216$ bits. La nueva representación requiere un diccionario de 16 colores que necesita 24 bits por cada color y 4 bits por ubicación de píxel, por lo tanto, $16 \times 24 + 128 \times 128 \times 4 = 65.920$ bits. Lo que corresponde a comprimir la imagen original por un factor de 6.

Si comprimimos a menos colores (Figura 1.22) se requerirán menos bits para representarla pero la imagen será peor.

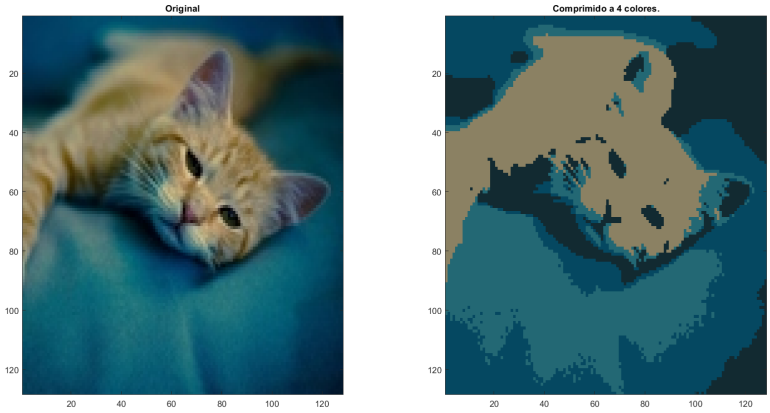

Figura 1.22. Imagen original a la izquierda. Imagen comprimida a 4 colores a la derecha

2

Métodos de regresión

En este capítulo vamos a estudiar los métodos de regresión lineal y logística. Tanto la regresión como la clasificación, vista en el capítulo anterior, son técnicas utilizadas en el análisis de datos y en la construcción de modelos predictivos.

La regresión se utiliza para predecir un valor numérico continuo, como por ejemplo el precio de una casa en función de sus características. Mientras, que la regresión logística es un método de clasificación que utiliza la regresión para predecir la probabilidad de que una observación pertenezca a una categoría específica de la variable de salida. En general, para predecir una variable categórica binaria, como por ejemplo, si una persona es diabética o no.

Pese a ser un método de clasificación la técnica de regresión logística se incluye en esta sección porque se comprende mejor en comparación con la regresión lineal.

2.1. Regresión lineal

La regresión lineal es el método de aprendizaje automático más representativo y conocido a la hora de construir modelos de predicción de valores. Posee una gran base teórica detrás, pero suele ser descartada en favor de métodos más optimizados. Presenta limitaciones al tratar con datos que se ajustan a otras formas no lineales. Sin embargo, es una buena manera de empezar a tratar con datos, ya que estos modelos son fáciles de construir y de interpretar.

El modelo de regresión lineal trata de encontrar el hiperplano que mejor aproxima un conjunto de n puntos dados. Es muy útil para la visualización de los datos, ya que sustituimos un gran conjunto de puntos por una recta de la que conocemos su expresión. Vamos a construir dicho modelo.

Supongamos que tenemos un conjuntos de puntos de $\mathbb{R}^{n+1}$:

$$\{(\boldsymbol{x^{(1)}}, y^{(1)}), (\boldsymbol{x^{(2)}}, y^{(2)}), \ldots, (\boldsymbol{x^{(n)}}, y^{(n)})\}$$

donde $\boldsymbol{x^{(i)}} = (x_1^{(i)}, \ldots, x_n^{(i)})$, que representan las variables predictoras. Y denotaremos por simplicidad en algunos casos $(x_1^{(i)}, \ldots, x_n^{(i)}) = (x_1, \ldots, x_n)$.

Queremos estimar el valor de y_i como combinación lineal de las variables predictorias. Y como parámetros a estimar tendremos $\boldsymbol{\theta} \in \mathbb{R}^{n+1}$, $\boldsymbol{\theta} = (\theta_0, \ldots, \theta_n)$. Por conveniencia, se define $x_0 = 1$ para incluir el término constante en la ecuación, lo que permite que el modelo se ajuste mejor a los datos y mejore la precisión de las predicciones.

Luego, para hacer la estimación se utiliza una ecuación lineal:

$$h_\theta(x) = \theta_0 + x_1\theta_1 + \ldots + x_n\theta_n \tag{2.1}$$

Nuestro objetivo es encontrar la combinación lineal de coeficientes, $\boldsymbol{\theta}$, que mejor aproxime los valores $y^{(i)}$, es decir, $\hat{y}^{(i)} = h_\theta(\boldsymbol{x^{(i)}}) \approx y^{(i)}$.

La intuición nos sugiere buscar el hiperplano que minimiza la distancia entre todos los puntos y este.

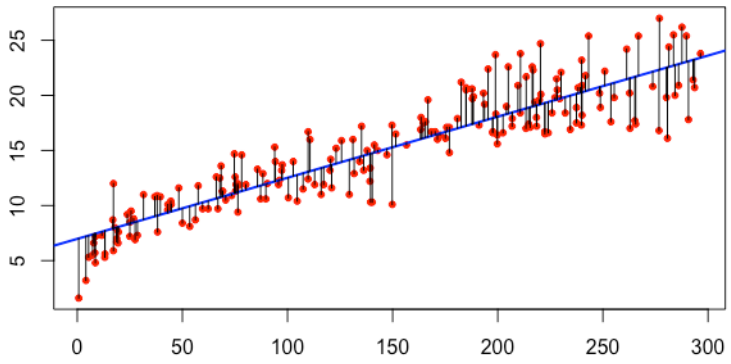

Figura 2.1. Ejemplo de regresión lineal [17]

Podemos observar una tendencia lineal en los datos de la Figura 2.1. Este ejemplo se muestra en dos dimensiones, por lo que variando los parámetros θ_0 y θ_1 obtendríamos todas las posibles rectas.

Con todas estas posibilidades, necesitamos un criterio objetivo para decidir cuál es la recta que mejor aproxima el conjunto. Para ello, la regresión lineal busca minimizar la suma de los cuadrados de las distancias verticales entre los valores aproximados y los observados, como se aprecia en la Figura 2.1. Este método se denomina "estimación de mínimos cuadrados".

Es decir, se basa en minimizar los errores cometidos, también llamados residuales, en la aproximación:

$$e^{(i)} = y^{(i)} - \hat{y}^{(i)}.$$

Por ello, necesitamos una función que mida el error cometido entre los valores predichos y los dados en el conjunto de entrenamiento original, $(y^{(i)})$, para obtener los valores de $\boldsymbol{\theta}$ más óptimos. Necesitamos introducir el concepto de *función de coste*.

Definición 1 *La* ***función de coste*** *es una función que trata de determinar el error cometido entre los valores estimados y los valores reales, con el fin de optimizar los parámetros de la expresión.*

Para la regresión lineal, existen distintas funciones de costo. En nuestro caso, vamos a tomar el **error cuadrático medio (MSE)**, $J : \mathbb{R}^{n+1} \longrightarrow \mathbb{R}$, que se define de la siguiente forma:

$$J(\boldsymbol{\theta}) = \frac{1}{2n}\sum_{i=1}^{n}(\hat{y}^{(i)} - y^{(i)})^2 = \frac{1}{2n}\sum_{i=1}^{n}(h_\theta(\boldsymbol{x^{(i)}}) - y^{(i)})^2 \tag{2.2}$$

Como podemos observar en (2.2), cuando $h_\theta(\boldsymbol{x^{(i)}}) \approx y^{(i)}$, la función de costo satisface que $J(\boldsymbol{\theta}) \approx 0$.

Además, como todos los términos de nuestra función de costo son positivos,

$$J(\boldsymbol{\theta}) \approx 0 \Longleftrightarrow h_\theta(\boldsymbol{x^{(i)}}) \approx y^{(i)}$$

.

Por lo tanto, encontrando $\boldsymbol{\theta^*}$ que minimiza la función de costo, conseguiremos nuestro objetivo. Es decir, debemos encontrar,

$$\boldsymbol{\theta^*} = \arg\min_{\boldsymbol{\theta}\in\mathbb{R}^{(n+1)}} J(\boldsymbol{\theta})$$

Una vez obtengamos este valor, el error que cometemos es el mínimo y nuestro modelo de regresión lineal, $h_{\boldsymbol{\theta^*}}(x)$, puede ser usado para predecir nuevos valores [22].

2.1.1. Métodos de minimización: Descenso de gradiente estocástico

Ahora, nuestro objetivo consiste en hallar dicho mínimo. Esto se puede hacer de forma analítica, calculando las derivadas parciales de la función respecto a los parámetros e igualando a cero. Sin embargo, las soluciones analíticas normalmente requieren operaciones de álgebra lineal complejas, como la inversión de matrices, que son muy costosas desde el punto de vista computacional y pueden ser numéricamente inestables. Calcular el descenso de gradiente es mucho más simple y más rápido. Además, en muchos casos no es posible encontrar una solución analítica.

Otra opción consiste en utilizar métodos de optimización numérica. Por ejemplo, el método de descenso Newton converge más rápido que el descenso de

gradiente. Sin embargo, no tiene una tasa de aprendizaje ajustable (más adelante detallaremos en qué consiste exactamente) y utiliza la matriz hessiana en su expresión, pero no se garantiza que exista. Incluso si existe, es muy costosa de calcular. Esto significa que aunque el descenso de Newton requiere menos pasos para converger, el descenso de gradiente sigue siendo generalmente más rápido para los casos de uso en el aprendizaje automático.

Por todos estos motivos, *el descenso de gradiente estocástico (Stochastic Gradient Descend)*, ofrece una combinación única de eficiencia computacional y adaptabilidad a diferentes modelos, que lo convierten en la mejor opción para encontrar predictores en problemas de aprendizaje automático y optimización [25].

Para visualizar cómo funciona vamos a considerar la función de costo de la Figura 2.2. Tomamos un solo parámetro, θ_1, para poder visualizar la función de costo en dos dimensiones.

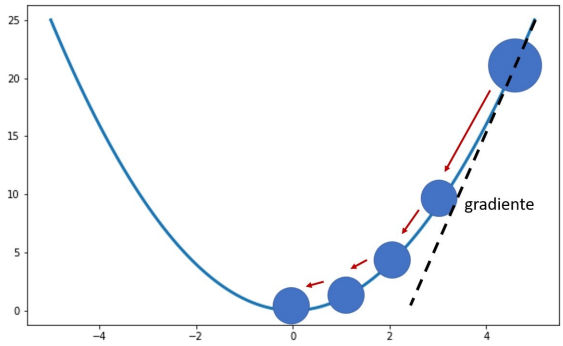

Figura 2.2. Algoritmo de descenso de grandiente [16]

La función de costo en este caso es una parábola, que tiene un único mínimo global, al igual que cualquier superficie convexa. El objetivo del algoritmo es alcanzar de algún modo dicho mínimo, que define los parámetros para el mejor ajuste de regresión. Por lo tanto, es recomendable disponer de una función de costo convexa, ya que el mínimo es global. Veremos en próximas secciones lo que ocurre para funciones no convexas.

Para tener una intuición de cómo funciona, podemos imaginarnos que estamos en la cima de una montaña, y nuestro objetivo es llegar al valle. El descenso de gradiente nos indica la dirección en la que debemos movernos para llegar al valle.

Comenzamos en una posición aleatoria en la montaña y calculamos la pendiente en ese punto. Luego, nos movemos en la dirección que nos lleve a un punto con una pendiente más baja, es decir, en la dirección que apunta hacia el valle más cercano. Este proceso se repite una y otra vez hasta que llegamos al valle.

Matemáticamente, se parte de un punto arbitrario, x_0. Como en nuestro ejemplo nos estamos restringiendo a una dimensión, solo podemos movernos hacia la izquierda o hacia la derecha. Por lo tanto, tomamos un pequeño paso en cada dirección, es decir, tomamos los valores $x_0 - \varepsilon$ y $x_0 + \varepsilon$. Ahora:

- Si $J(x_0 - \varepsilon) < J(x_0)$, entonces, tenemos que movernos hacia la izquierda para descender. Como en la Figura 2.2.
- Si $J(x_0 + \varepsilon) < J(x_0)$, entonces, tenemos que movernos hacia la derecha para descender.
- Si no se da ningún caso anterior, entonces, no podemos reducir el valor de J y hemos encontrado el mínimo.

La dirección del descenso está marcada por la pendiente, $\frac{\partial J}{\partial \theta}$, es decir, la recta tangente a la función en ese punto. Una pendiente positiva indica que el mínimo debe de estar hacia la izquierda, mientras, una pendiente negativa, indica que debe de estar hacia la derecha.

La magnitud de esta pendiente describe la inclinación de esta caída, es decir, cuanto difiere $J(x_0 - \varepsilon)$ de $J(x_0)$, por ejemplo.

La pendiente puede ser aproximada encontrando la recta que pasa por los puntos $(x_0, J(x_0))$ y $(x_0, J(x_0 - \varepsilon))$. Esto es exactamente lo que se hace al computar una derivada, que en cada punto especifica la tangente a la curva. Ya tenemos el primer elemento de nuestro algoritmo: las derivadas.

A medida que avanzamos más allá de una dimensión, ganamos la libertad de movernos en un rango mayor de direcciones. En lugar de calcular una pendiente en una sola dirección, necesitamos calcular una tasa de cambio en cada una de las posibles. Encontrar estar direcciones requiere computar las derivadas parciales de la función objetivo. La magnitud de las derivadas parciales define la inclinación en cada dirección.

Lo único que falta para terminar de construir el algoritmo es fijar el parámetro que determina el tamaño del paso que da el algoritmo en cada dirección para cada iteración. Lo llamaremos **tasa de aprendizaje (α)**. Si la tasa de aprendizaje es demasiado baja, el modelo podría tardar demasiado en converger, es decir, en llegar al valor mínimo de la función de costo. En contraste, si la tasa de aprendizaje es demasiado alta, el modelo podría "oscilar" entre diferentes valores y nunca converger. La tasa de aprendizaje se puede ajustar mediante prueba y error o mediante el uso de técnicas más avanzadas. Lo veremos con detenimiento en el siguiente apartado. Luego, el algoritmo de descenso de gradiente para cualquier dimensión se define como:

Definición 2 *El método de descenso de gradiente encuentra el mínimo de una función mediante un proceso iterativo siguiendo los siguientes pasos:*

1. *Se toma un* $\boldsymbol{\theta^{(t)}} \in \mathbb{R}^{n+1}$ *como valor inicial para* $\boldsymbol{\theta^*}$

2. *Se repite el siguiente proceso hasta que* $\|\boldsymbol{\theta}^{(t+1)} - \boldsymbol{\theta}^{(t)}\| < \delta$, $\boldsymbol{\theta}^{(t)} \in \mathbb{R}^{n+1}$, *donde* δ *se define como la tolerancia de convergencia:*

$$\boldsymbol{\theta}^{(t+1)} := \boldsymbol{\theta}^{(t)} - \alpha \nabla_\theta J(\boldsymbol{\theta}^{(t)}) \tag{2.3}$$

Donde:

- $t = 0, 1, ...$ *es el número de iteración*
- α *es la tasa de aprendizaje que determina el paso que toma el algoritmo en cada iteración.*
- ∇_θ *es el gradiente de* $J(\boldsymbol{\theta})$ *con respecto a los parámetros* $\boldsymbol{\theta}$*, y se define como:*

$$\nabla_\theta J(\boldsymbol{\theta}) = \left[\frac{\partial J(\boldsymbol{\theta})}{\partial \theta_0}, \quad \frac{\partial J(\boldsymbol{\theta})}{\partial \theta_1}, \quad \cdots, \quad \frac{\partial J(\boldsymbol{\theta})}{\partial \theta_n} \right]^T$$

Luego, el descenso de gradiente para la función de costo que consideramos en regresión lineal consiste en:

$$\frac{\partial J}{\partial \theta_0} = \frac{2}{2n} \sum_{i=1}^{n} (\hat{y}^{(i)} - \theta_0 - \theta_1 x_1 - ... - \theta_n x_n)(-1) = \frac{-1}{n} \sum_{i=1}^{n} (\hat{y}^{(i)} - y^{(i)})$$

$$\frac{\partial J}{\partial \theta_1} = \frac{2}{2n} \sum_{i=1}^{n} (\hat{y}^{(i)} - \theta_0 - \theta_1 x_1 - ... - \theta_n x_n)(-x_1) = \frac{-1}{n} \sum_{i=1}^{n} x_1 (\hat{y}^{(i)} - y^{(i)})$$

Y, por lo tanto,

$$\frac{\partial J}{\partial \theta_j} = \frac{-1}{n} \sum_{i=1}^{n} x_j^{(i)} (\hat{y}^{(i)} - y^{(i)}), \quad j \in \{0, \cdots, n\} \tag{2.4}$$

Luego, insertando la expresión (2.4) en la ecuación (2.3) obtenemos:

$$\theta_j^{(t+1)} := \theta_j^{(t)} - \frac{\alpha}{n} \sum_{i=1}^{n} (\hat{y}^{(i)} - y^{(i)}) x_j^{(i)} \tag{2.5}$$

Recordamos que el error se define como: $e^{(i)} = y^{(i)} - \hat{y}^{(i)}$. Por lo tanto, estamos actualizando $\theta_j^{(t)}$ por un desplazamiento lineal que es proporcional a la estimación del error ([22], [12]).

2.1.2. Tasa de aprendizaje

La derivada de la función de costo nos indica la dirección a seguir para alcanzar el mínimo, que especifica los parámetros para resolver nuestro problema de regresión. Pero no especifica cómo de grandes tienen que ser los pasos. Por ejemplo, si queremos ir desde La Laguna a Los Cristianos, sabemos que el

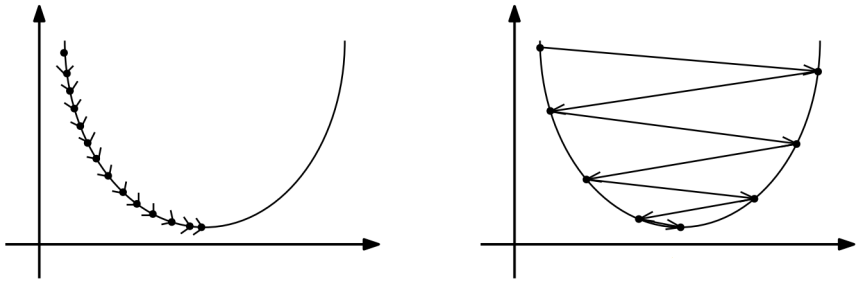

Figura 2.3. Efecto del tamaño del paso [6]

camino más rápido es tomar la autopista, pero cuando nos estemos acercando, necesitaremos instrucciones más detalladas para llegar a nuestro destino.

El descenso de gradiente encuentra la mejor dirección, toma un paso y repite hasta que encuentra el mínimo. Si tomamos pasos muy pequeños, esta convergencia será muy lenta. Sin embargo, si tomamos un paso muy grande, cabe la posibilidad de saltarnos el mínimo, como podemos observar en la Figura 2.3.

En general, queremos una tasa de aprendizaje grande al principio de nuestra búsqueda (la autopista del ejemplo anterior) y que vaya decreciendo a medida que nos acercamos al mínimo. Para ello, necesitamos controlar el valor de la función de costo a lo largo de la optimización. Las librerías para el descenso de gradiente disponen de algoritmos construidos para ajustar la tasa de aprendizaje. Por ejemplo, *TensorFlow* [1] ofrece la implementación del descenso de gradiente y varios algoritmos de optimización, como el "AdamOptimizer", que adapta automáticamente la tasa de aprendizaje durante el entrenamiento. O *Scikit-learn* es una librería de aprendizaje automático de propósito general en Python. Proporciona el optimizador "SGDRegressor" y "SGDClassifier", que utilizan el descenso de gradiente estocástico y permiten ajustar la tasa de aprendizaje [6].

2.1.3. Convexidad

Como habíamos comentado anteriormente, la forma de la superficie de la función de costo es de vital importancia a la hora de encontrar el mínimo global a partir del descenso de gradiente. Si la función es convexa, el algoritmo funciona muy bien, como se puede apreciar en la Figura 2.4. Por ejemplo, si tenemos una superficie con forma de bol, pero relativamente plano (como un plato hondo), el proceso hasta encontrarlo será más lento que en un bol más pronunciado. Cuando la pendiente de la función de costo es igual o cercana a cero, el modelo deja de iterar ya que el término que resta en la expresión de actualización se

[1] https://www.tensorflow.org

hace cero y los parámetros coinciden, por lo que su norma será menor que δ. En este caso, conviene parar y tomar estos valores como solución.

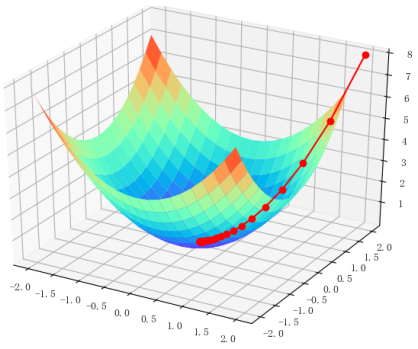

Figura 2.4. Descenso de gradiente con función convexa [28]

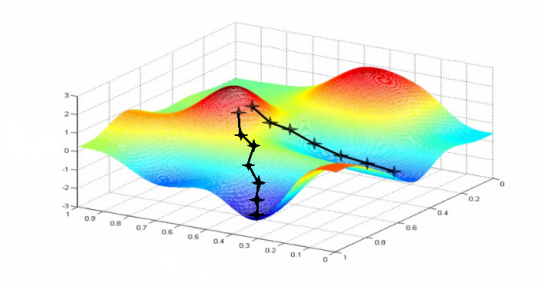

Figura 2.5. Descenso de gradiente con función no convexa

El mayor problema ocurre cuando la función de costo no es convexa, esto significa que pueden existir varios mínimos locales, como puede observarse en la Figura 2.5. Este no es el caso de la regresión lineal, pero ocurre para otras funciones interesantes de problemas de aprendizaje automático. Por ejemplo, la función de error cuadrático medio en la regresión logística no es convexa. Veremos en detalle este algoritmo próximamente.

En estos casos, podemos quedar atrapados en un mínimo local, a no ser que exista un mecanismo para volver hacia atrás que nos libere del mínimo local para seguir avanzando hacia el objetivo global. Aunque pueda existir este valor, lo más común a la hora de aplicar el descenso de gradiente en funciones no convexas es repetir el proceso desde diferentes puntos iniciales y usar el mejor mínimo local para definir nuestra solución.

Además, muchos problemas que resolvemos con aprendizaje automático, suelen presentar lo que se denomina punto de silla. En el que la pendiente es cero pero no se trata de un extremo local (máximo o mínimo) ya que la elevación es máxima en una dirección y mínima en la dirección perpendicular. Por lo que el proceso, puede seguir iterando [6].

2.2. Regresión logística

Podemos aplicar regresión lineal a problemas de clasificación convirtiendo las clases en números, siempre que tenga sentido hacerlo. En esta sección, trataremos problemas de clasificación binaria. La conversión se produce asociando, en general, "la clase positiva" al cero y "la clase negativa" al uno:

- spam=1 / no spam=0

- cáncer=1 / benigno=0

En los casos en los que no existe esta distinción, se elige de forma arbitraria. Supongamos que tenemos información acerca del tamaño de un tumor y diagnóstico, que se muestra en la Figura 2.6.

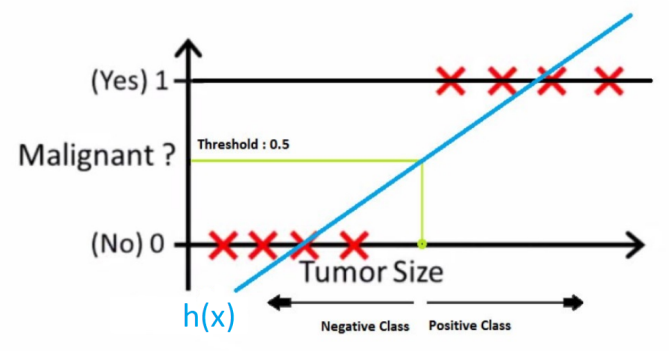

Figura 2.6. Ejemplo de regresión lineal a problemas de clasificación binaria [19]

Estamos interesados en aplicar regresión lineal. Ajustamos nuestra recta de regresión a través del proceso explicado anteriormente y obtenemos la recta azul ($h(x)$). Nos interesa buscar una regla para predecir si los tumores de futuros pacientes van a ser malignos o no.

Para tomar una decisión se necesita establecer un umbral, que será un valor entre 0 y 1. Si la salida continua del modelo es mayor que el valor del umbral, se clasifica en la clase del 1; de lo contrario, se clasifica en la otra clase, la del 0. El valor del umbral suele ser establecido en 0.5 por defecto.

Sin embargo, puede ajustarse para mejorar la precisión y el rendimiento del modelo, según los requisitos del problema específico. Por ejemplo, se puede aumentar el umbral para minimizar los falsos positivos (clasificar como positivo cuando en realidad es negativo), a costa de aumentar los falsos negativos (clasificar como negativo cuando en realidad es positivo). Este es el caso más peligroso de error.

Vamos a tomar como valor umbral 0.5, es decir:

$$si\ h(x) > 0.5 \longrightarrow 1\ (tumor\ maligno)$$
$$si\ h(x) < 0.5 \longrightarrow 0\ (tumor\ no\ maligno)$$

Hasta aquí, todo parece funcionar correctamente. Sin embargo, si añadimos un valor más alejado del resto, la recta de mejor ajuste en la regresión lineal cambia para ajustarse a ese punto, dando lugar a la situación de la Figura 2.7.

Manteniendo la misma regla de decisión, es decir, que el umbral es 0.5, observamos que la línea verde que representa este límite de decisión, separa

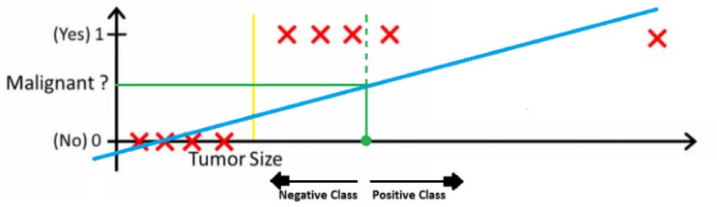

Figura 2.7. Ejemplo de regresión lineal a problemas de clasificación binaria con valor alejado

los tumores malignos y benignos. En este caso, la recta de regresión clasificará algunos datos que antes estaban en la clase positiva, en la clase negativa. Por lo tanto, pacientes a los que antes diagnosticábamos su tumor como maligno, hemos pasado a clasificarlo como benigno. Esto es un grave problema, es lo que se denomina en estadística un modelo poco robusto.

Como conclusión, incluso un único valor alejado de lo normal puede desequilibrar las estimaciones de regresión lineal. El fracaso de la regresión lineal para clases discretas de resultados nos lleva a tener que definir otro tipo de modelo, partiendo de la regresión vista. Para superarlo, la regresión logística transforma la recta de regresión en una curva con valores en el rango $[0, 1]$, usando la función sigmoide, que veremos a continuación ([21]).

La regresión logística es otro famoso algoritmo de aprendizaje automático supervisado. Su principal diferencia con la regresión lineal radica en el tipo de problemas para los que son usados. La regresión lineal se emplea en problemas de predicción de valores, mientras que la regresión logística se utiliza en problemas de clasificación binaria. Se aplica en una gran variedad de campos, incluyendo la medicina, las ciencias sociales y la ingeniería.

Para entender este método, veámoslo aplicado al siguiente problema con dos variables. Se tiene un conjunto de 100 datos donde cada uno viene representado con dos características, y además, pertenece a una de dos posibles categorías: 0 o 1. Los 100 datos se encuentran distribuidos de la manera que se muestra en la Figura 2.8.

El objetivo de la regresión logística es clasificar de forma automática nuevos datos en las dos categorías utilizando el conjunto ya clasificado. Esto equivale a encontrar una frontera que permita separar los datos en dos agrupaciones diferentes, como se muestra en la Figura 2.9.

2.2.1. Límites de decisión

La forma de pensar en clasificación es dividir el espacio en regiones en las que todos los puntos de una región son asignados a la misma etiqueta. Las regiones están definidas por sus límites.

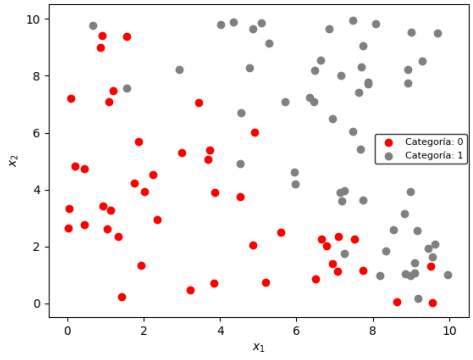

Figura 2.8. Representación del conjunto de datos y sus categorías [23]

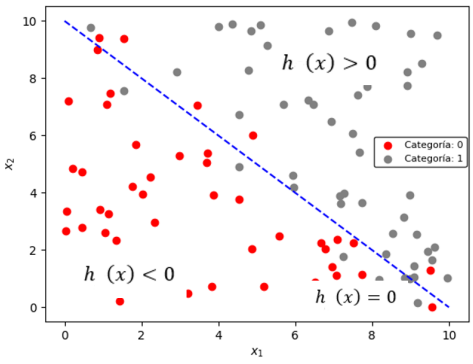

Figura 2.9. Representación de la frontera, línea azul

La línea azul de la Figura 2.9 representa el límite de decisión. En la figura, los datos ubicados encima de la frontera serán clasificados como "1", y los datos ubicados debajo como "0".

Tenemos que buscar un criterio para que cuando $h_\theta(x) \gg 0$ implique sin lugar a dudas que $\hat{y} = 1$; y cuando, $h_\theta(x) \ll 0$, implique que $\hat{y} = 0$. Es decir, modificar el modelo para no cometer el error del ejemplo que vimos anteriormente con el diagnóstico de los tumores y la regresión lineal.

Necesitamos utilizar la regresión lineal para establecer la frontera de decisión utilizando los datos, y después usar una función para obtener los valores en el intervalo $[0, 1]$, y que nos ayude a interpretarlos como grado de posibilidad (probablilidad) de ser "0" o "1". Para nuestro ejemplo:

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2$$

Con dicha transformación se logra una separación inicial de los datos, una recta de regresión, para su posterior clasificación ([32], [23]).

2.2.2. Función de activación: la función sigmoide

La función $h_\theta(x)$ tiene un rango continuo de valores. Para el proceso de clasificación se requiere que estén en el intervalo $[0, 1]$, el cual se puede obtener con la **función sigmoide**. Se puede observar en la Figura 2.10. Se define como:

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

Por lo tanto, de obtener el límite de decisión se encarga la regresión lineal, y obtener los valores de los puntos en el intervalo $[0, 1]$, la función sigmoide. El dominio de la función son todos los valores reales $(-\infty \leq z \leq \infty)$, y su

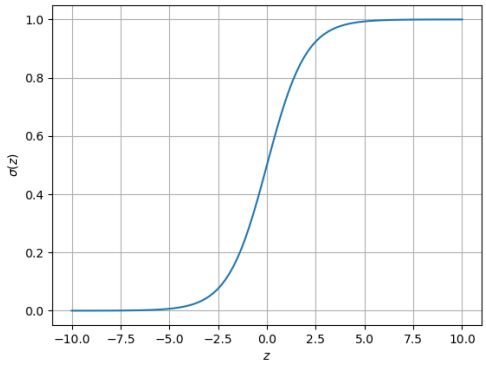

Figura 2.10. Función sigmoide

imagen se restringe al intervalo $[0, 1]$, es decir, se podría interpretar como una probabilidad. Verifica las siguientes propiedades:

$$\lim_{x\to\infty} \sigma(x) = \frac{1}{1+e^{-\infty}} = 1$$
$$\lim_{x\to-\infty} \sigma(x) = \frac{1}{1+e^{\infty}} = \frac{1}{\infty} = 0$$

De hecho, $\sigma(h_\theta(x))$ mide la probabilidad de que un punto de nuestro ejemplo se clasifique como 1. Tomaremos como valor umbral 0.5, es decir, que si para un punto $\sigma(h_\theta(x)) = 0.75$, el punto tendrá como etiqueta 1, puesto está por encima del umbral. En ese caso, la probabilidad de que el punto deba ser etiquetado como 1 es de 0.75.

Para valores cercanos a 0.5 la valoración es poco precisa. Se pueden tomar decisiones más inequívocas cuanto más nos alejemos de este valor ($h_\theta(x) \gg 0$ o $h_\theta(x) \ll 0$).

Luego, el clasificador que vamos a utilizar surge de componer nuestra recta de regresión con la función sigmoide:

$$f(x) = \frac{1}{1+e^{-h_\theta(x)}}, \tag{2.6}$$

donde, $\boldsymbol{\theta}$ son los parámetros de los que tenemos que estimar su valor óptimo para obtener el separador de los datos.

Esto nos recuerda al caso de regresión lineal, en el que llevamos a cabo los siguientes pasos:

- Definición de la función de costo.
- Inicialización aleatoria de los parámetros del modelo.
- Definición de la tasa de aprendizaje y el número de iteraciones.

- Actualización de los parámetros del modelo usando el algoritmo del gradiente descendente.

Es decir, este proceso de entrenamiento consistía en aplicar el gradiente descendente para calcular los parámetros del modelo que minimizan la pérdida. En la regresión logística se procede de la misma manera. La única diferencia es que ya no nos sirve el error cuadrático medio como función de costo. Para hacer un estudio más completo veremos dos formas de construir la función de costo para la regresión logística.

2.2.3. Función de costo para la regresión logística

La función de costo mide la diferencia existente entre las categorías, y, a las que realmente pertenece cada uno de los datos (0 o 1), y las que se obtienen durante el entrenamiento del modelo, $\hat{y}$.

En la regresión logística no se utiliza el error cuadrático medio, pues al ser variables binarias, obtendríamos una función no convexa (se trataría de una función escalonada que ni siquiera es continua). Por tanto, en este caso, se usa una función de costo conocida como la entropía cruzada *(cross-entropy)*. Vamos a deducir su expresión.

Tenemos una etiqueta de clase cero/uno de $y^{(i)}$ para cada vector de entrada $\boldsymbol{x^{(i)}}$. Necesitamos una función de costo que atribuya el valor apropiado a $f(\boldsymbol{x^{(i)}}) = \hat{y}^{(i)}$ como la probabilidad de que $y^{(i)} = 1$.

Primero, vamos a considerar el caso de que $y^{(i)} = 1$. Entonces, de forma ideal, $f(\boldsymbol{x^{(i)}}) = 1$. Queremos penalizarlo si es menor que uno, es decir, cuando $f(\boldsymbol{x^{(i)}}) \longrightarrow 0$, ya que esto significaría que el punto i tiene pocas posibilidades de estar en la clase 1, cuando ese es, en realidad, el caso correcto.

La función logarítmica resulta ser una buena función de penalización. Cuando $y^{(i)} = 1$, definimos $cost(\boldsymbol{x^{(i)}}, 1) = -\log(f(\boldsymbol{x^{(i)}}))$. Como $\log(1) = 0$, tenemos un error cero cuando $f(\boldsymbol{x^{(i)}}) = 1$, como debe ser para identificar correctamente $y^{(i)} = 1$.

Además, $\log(x) \longrightarrow -\infty$ cuando $x \longrightarrow 0$, es decir, esta función penaliza de forma muy severa cuanto peor clasificamos $y^{(i)}$, lo que la hace muy apropiada para nuestro propósito.

Si ahora, consideramos el caso en el que $y^{(i)} = 0$, queremos penalizar cuanto más se acerque $f(\boldsymbol{x^{(i)}})$ a 1. Por lo tanto, podemos tomar $cost(\boldsymbol{x^{(i)}}, 0) = -\log(1 - f(\boldsymbol{x^{(i)}}))$. Se puede apreciar este efecto en la Figura 2.11.

Nos falta juntar estas dos expresiones. Observamos que $y^{(i)}$ tiene solo dos posibles valores, 1 o 0. Por cada clasificación errónea tenemos que sumar la penalización del caso. Si multiplicamos $cost(\boldsymbol{x^{(i)}}, 1)$ por $y^{(i)}$, obtenemos el efecto deseado, ya que la penalización se aplica para los $y^{(i)} = 1$. De igual forma, multiplicando $cost(\boldsymbol{x^{(i)}}, 0)$ por $(1 - y^{(i)})$, obtenemos el efecto contrario, aplicamos la penalización para $y^{(i)} = 0$.

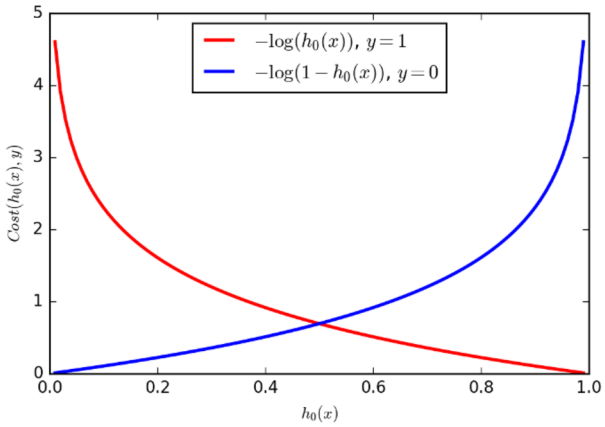

Figura 2.11. Funciones de penalización para el caso $y = 1$ en rojo, y para $y = 0$, en azul [6]

Luego, podemos definir la función de costo para la regresión logística como:

$$\begin{aligned} J(\boldsymbol{\theta}) &= \frac{1}{n}\sum_{i=1}^{n} cost(f(\boldsymbol{x^{(i)}}), y^{(i)}) \\ &= -\frac{1}{n}\sum_{i=1}^{n} y^{(i)} \log(f(\boldsymbol{x^{(i)}})) + (1 - y^{(i)}) \log(1 - f(\boldsymbol{x^{(i)}})) \end{aligned} \tag{2.7}$$

Un gran aspecto positivo de esta función es que es convexa, como se puede apreciar en la Figura 2.11. Lo que significa que podemos encontrar los parámetros $\boldsymbol{\theta}$ más óptimos usando descenso de gradiente, y además, el mínimo es global. Por lo tanto, podemos encontrar el mejor separador lineal entre dos clases de esta forma, ya que estos parámetros son los de la recta [7], [32].

Sin embargo, es importante comprender que la regresión logística definida de esta manera siempre genera una frontera de decisión lineal. Es decir, la frontera siempre será un hiperplano con varias variables. Esto implica que si los datos tienen una distribución diferente (no lineal), tendremos que utilizar regresión logística con características polinómicas. Esto implica generar nuevas variables mediante combinaciones polinómicas de las variables originales. Al introducir interacciones no lineales, la regresión logística puede encontrar una frontera de decisión más compleja. Esto lo abordaremos en la sección 2.2.7.

2.2.4. Interpretación probabilística

La regresión logística puede ser interpretada probabilísticamente utilizando la distribución Bernoulli, que es una distribución de probabilidad discreta utilizada para modelar variables binarias. Vamos a tomar $f(x)$ como la definimos en (2.6).

La probabilidad de que $f(x)$ tome el valor 1 en función de las variables predictoras se puede expresar como:

$$P(y = 1|x; \theta) = f(x)$$

donde θ representa los coeficientes del modelo y x las variables predictoras. La probabilidad de que la variable de que tome el valor 0 se puede expresar como:

$$P(y = 0|x; \theta) = 1 - f(x)$$

La función de verosimilitud se utiliza para estimar los coeficientes del modelo, y se puede expresar como:

$$L(\theta) = \prod f(x)^y (1 - f(x))^{1-y}.$$

Estamos interesados en encontrar los valores de los parámetros del modelo que mejor se ajustan a los datos. En estadística, un enfoque común para la estimación de parámetros es la estimación de máxima verosimilitud (MLE). En la regresión logística, la función de verosimilitud se utiliza para calcular la probabilidad de los datos de entrenamiento en función de los valores de los parámetros.

Para encontrar estos valores se puede utilizar el descenso del gradiente. Este busca el mínimo de una función, pero en este caso, queremos encontrar el máximo de la función de verosimilitud.

Maximizar una función es lo mismo que maximizar el logaritmo de una función (ya que es monótona creciente). Por lo tanto, aplicamos logaritmos sabiendo que todo funciona bien:

$$\begin{aligned} l(\theta) &= \log\left(\prod f(x)^y (1 - f(x))^{1-y}\right) \\ &= \sum y \log(f(x)) + (1 - y) \log(1 - f(x)) \end{aligned}$$

Para utilizar el descenso del gradiente tenemos que multiplicar la función de verosimilitud por -1, para que nuestro máximo ahora coincida con el mínimo. Ahora, podemos aplicar el descenso de gradiente para encontrar los valores de los parámetros que minimizan la función. Multiplicando por -1 y haciendo la media, obtenemos la misma función de costo que en (2.7) ([7], [23]):

$$J(\boldsymbol{\theta}) = -\frac{1}{n} \sum y \log(f(x)) + (1 - y) \log(1 - f(x))$$

2.2.5. Aplicación del descenso de gradiente a la regresión logística

Una vez hemos obtenido la función de costo, nuestro objetivo, al igual que lo era para el caso de la regresión lineal, es encontrar los mejores parámetros

minimizando dicha función. Como se trata de una función convexa, el método más utilizado para hallar el mínimo es el descenso de gradiente.

El único elemento necesario para aplicar el descenso de gradiente es el cálculo de las derivadas parciales de la función de costo. Así,

$$\frac{\partial J}{\partial \theta_0} = \frac{-1}{n}\sum_{i=1}^{n}\left(y^{(i)}\frac{1}{f(x)}\frac{\partial f(x)}{\theta_0} + (1-y^{(i)})\frac{-1}{1-f(x)}\frac{\partial f(x)}{\partial \theta_0}\right)$$

luego,

$$\begin{aligned}\frac{\partial f(x)}{\partial \theta_0} &= -\left(\frac{1}{1+e^{-h_\theta(x)}}\right)^2 (e^{-h_\theta(x)})(-1)\\ &= \frac{1+e^{-h_\theta(x)}-1}{(1+e^{h_\theta(x)})^2}\\ &= \frac{1}{1+e^{-h_\theta(x)}} - \frac{1}{(1+e^{-h_\theta(x)})^2}\\ &= f(x) - f(x)^2 = f(x)(1-f(x))\end{aligned}$$

Entonces,

$$\begin{aligned}\frac{\partial J}{\partial \theta_0} &= \frac{-1}{n}\sum_{i=1}^{n}(y^{(i)}\frac{1}{f(x)}f(x)(1-f(x)) + (1-y^{(i)})\frac{-1}{1-f(x)}f(x)(1-f(x))\\ &= \frac{-1}{n}\sum_{i=1}^{n}\left(y^{(i)}(1-f(x)) - (1-y^{(i)})f(x)\right)\\ &= \frac{1}{n}\sum_{i=1}^{n}\left(f(x)-y^{(i)}\right)\end{aligned}$$

De manera similar,

$$\frac{\partial J}{\partial \theta_j} = \frac{1}{n}\sum_{i=1}^{n}\left(f(\boldsymbol{x^{(i)}}) - y^{(i)}\right)x_j^{(i)}, \quad j \in \{0, \cdots, n\}$$

Que coincide con la expresión (2.4) que obtuvimos en el caso de la regresión lineal. Ya tenemos, entonces, la función de costo y una forma de implementar el descenso de gradiente. Ahora solo es necesario iterarlo hasta que converja.

2.2.6. Interpretación

Una vez obtenidos los coeficientes del modelo, que denotaremos $\hat{\boldsymbol{\theta}}$, se pueden clasificar nuevas observaciones evaluando,

$$f(x) = \frac{1}{1+e^{-h_{\hat{\boldsymbol{\theta}}}(x)}}$$

y asignamos la clase de la siguiente forma:

$$clase = \begin{cases} 0 \text{ si } f(x) \leq 0.5 \\ 1 \text{ si } f(x) > 0.5 \end{cases}$$

O si deshacemos la transformación y nos fijamos en el hiperplano:

$$h_{\hat{\boldsymbol{\theta}}(x)} = \hat{\theta}_0 + \hat{\theta}_1 x_1 + \cdots + \hat{\theta}_n x_n$$

que divide el espacio en dos regiones, como se podía observar en la Figura 2.9 ([15], [7]).

$$clase = \begin{cases} 0 \text{ si } \hat{\theta}_1 x_1 + \cdots + \hat{\theta}_n x_n \leq -\hat{\theta}_0 \\ 1 \text{ si } \hat{\theta}_1 x_1 + \cdots + \hat{\theta}_n x_n > -\hat{\theta}_0 \end{cases}$$

2.2.7. Regresión logística polinómica y regularizada

Como habíamos apuntado anteriormente, existen conjuntos de datos que no pueden ser separados mediante una línea recta en ejemplos positivos y ejemplos negativos, como se puede ver en la Figura 2.12.

Para obtener un ajuste con límites de decisión no lineales, se puede utilizar una técnica conocida como regresión logística polinómica. Consiste en crear nuevas características polinómicas a partir de las variables independientes existentes. Esto implica elevar al cuadrado, al cubo o a potencias superiores las variables continuas, y combinar variables de manera polinómica. Luego, se utiliza la nueva expresión para ajustar un modelo de regresión logística, como se ilustra en la Figura 2.12.

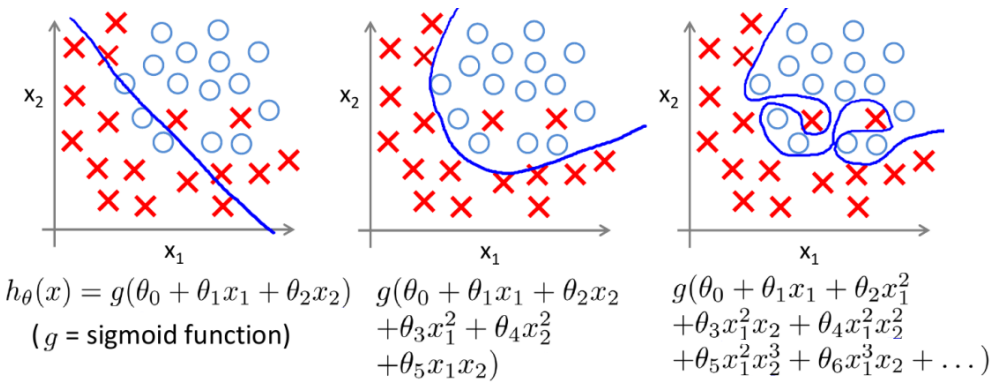

Figura 2.12. Nuevas características

En estos casos, es posible que el modelo se sobreajuste al definir las nuevas características. Esto sucede porque dominan en el modelo los coeficientes

no lineales y de grados mayores, que tenderán a ajustarse más entre los datos para separarlos como refleja la tercera representación de la Figura 2.12. La regularización es una técnica utilizada en el aprendizaje automático para evitar el sobreajuste ("overfitting") de un modelo a los datos de entrenamiento. El sobreajuste da como resultado que nuestro modelo no se generalice. Podemos encontrar que hemos entrenado un modelo en algún conjunto de datos y parece funcionar bien en esos datos, pero cuando lo probamos en un nuevo conjunto, el rendimiento se ve afectado. Esto podrá ocurrir debido a que la muestra de la que derivamos nuestro modelo estaba sesgada de alguna manera.

La regularización introduce una penalización adicional en la función de coste con el objetivo de desalentar a los coeficientes del modelo de tomar valores que dominen unos sobre otros de manera muy sustancial. Esto se logra mediante la adición de un término de regularización a la función de coste original. Puede tener diferentes formas, pero las más comunes son la regularización Lasso y la regularización Ridge.

Este último agrega la suma de los cuadrados de los coeficientes como término de penalización, junto con un parámetro de regularización, generalmente denotado como lambda (λ). Un valor más alto de lambda aumentará la penalización y puede ayudar a evitar el sobreajuste, pero también puede llevar a un subajuste ("underfitting") si se vuelve demasiado alto ([6]).

Luego, obtenemos la siguiente expresión para la función de costo de regresión logística regularizada:

$$J(\boldsymbol{\theta}) = -\frac{1}{n}\sum_{i=1}^{n} y^{(i)} \log(f(\boldsymbol{x^{(i)}})) + (1 - y^{(i)}) \log(1 - f(\boldsymbol{x^{(i)}})) + \frac{\lambda}{2n}\sum_{j=1}^{n} \theta_j^2 \quad (2.8)$$

No se tiene en cuenta θ_0 porque no actúa sobre ninguna variable, ya que es el término independiente.

Y el gradiente de la función de costo se define de la siguiente manera:

$$\frac{\partial J}{\partial \theta_0} = \frac{1}{n}\sum_{i=1}^{n} \left(f(\boldsymbol{x^{(i)}}) - y^{(i)}\right) x_j^{(i)}, \quad j = 0$$

$$\frac{\partial J}{\partial \theta_j} = \frac{1}{n}\sum_{i=1}^{n} \left(f(\boldsymbol{x^{(i)}}) - y^{(i)}\right) x_j^{(i)} + \frac{\lambda}{n}\theta_j, \quad j \geq 1$$

Esto es para el caso que hemos venido estudiando hasta ahora. Para la regresión logística polinómica tendremos m parámetros que dependen de las nuevas características que formemos.

2.2.8. Problemas de la regresión logística

Existen varios inconvenientes que son relevantes tanto para la regresión logística como para otros métodos de aprendizaje automático:

- **Desequilibrio del tamaño de las clases:** Es aconsejable que el tamaño de ejemplos negativos y positivos sea el mismo para que no exista un desequilibrio en el números de ejemplos para el entrenamiento. Puesto que la clase grande contribuirá demasiado a la función de costo en comparación con la clase más pequeña.
- **Clasificación multiclase:** A menudo, los problemas de clasificación implican elegir entre más de dos etiquetas distintas, como vimos en el capítulo anterior. En estos casos, si buscamos aplicar la regresión logística es mejor realizarla entre los elementos de una clase y la unión de los elementos del resto de clases, y realizar este proceso para todas las etiquetas. O aplicar los algoritmos del clasificación estudiados en el capítulo anterior.

2.3. Ejemplo de regresión lineal

Ahora que hemos comprendido los conceptos básicos de la regresión lineal y la regresión logística, es momento de aplicar estos métodos en ejemplos prácticos, utilizando para ello MATLAB.

Vamos a implementar regresión lineal con una variable para predecir las ganancias de un camión de comida. Tenemos una franquicia de restaurantes sobre ruedas y estamos considerando diferentes ciudades para expandirnos. La cadena ya tiene camiones en varias ciudades y tiene datos de ganancias y poblaciones de las ciudades. Nos gustaría usar estos datos para seleccionar a qué ciudad expandirnos ([2]). Los datos representandos se pueden ver en la Figura 2.13.

Ahora ajustaremos los parámetros de regresión lineal, $\boldsymbol{\theta}$, a nuestro conjunto de datos utilizando el descenso de gradiente sobre la función de costo. Inicializamos los parámetros iniciales a 0 y la tasa de aprendizaje, α, a 0.01.

El valor de $J(\theta)$ decrece en cada iteración y converge a un valor estable al final del algoritmo, obteniendo los parámetros para trazar el ajuste lineal (Figura 2.14, Figura 2.15 y Figura 2.16).

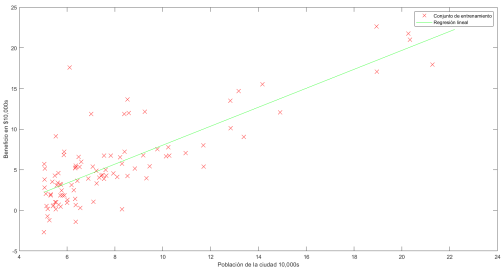

Figura 2.13. Ejemplo de ajuste de regresión lineal

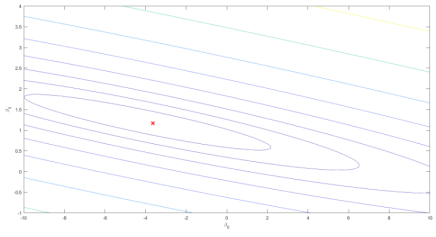

Figura 2.14. Valores óptimos

```
J = 40.2181
J = 37.2379
J = 34.4654
J = 31.9005
J = 29.5433
J = 27.3937
J = 25.4518
...
J = 4.7121
J = 4.7113
...
J = 4.4834
J = 4.4834

Theta found by gradient descent:
 -3.630291
 1.166362
For population = 35,000, we predict a profit of 4519.767868
For population = 70,000, we predict a profit of 45342.450129
```

Figura 2.15. Proceso llevado a cabo por el programa para encontrar los valores óptimos

```
function [theta, J_history] = gradientDescent(X, y, theta, alpha, num_iters)
% Initialize some useful values
m = length(y); % number of training examples
J_history = zeros(num_iters, 1);
J = 0
%theta = [0; 0]
for iter = 1:num_iters
    predictions = X * theta;
    predictions1 = (predictions-y);
    for j = 1: 2
    theta(j, 1) = theta(j, 1) - alpha * sum(predictions1'*X(:,j))/m;
    end;
    % Save the cost J in every iteration
    J_history(iter) = computeCost(X, y, theta);
end

end
```

Figura 2.16. Algoritmo del descenso de gradiente

Por ejemplo, en un área de 70000 personas, se tendrán unos beneficios de aproximadamente 45342 dólares.

En la Figura 2.14 se muestra el punto óptimo para θ_0 y θ_1, y cada paso en el descenso del gradiente se acerca a este punto, haciendo el valor de la función costo menor.

2.4. Ejemplo de regresión logística

Vamos a construir un modelo de regresión logística para predecir la probabilidad de que un estudiante sea admitido en una universidad en función de su resultado en dos exámenes. Para ello, disponemos de datos históricos de solicitantes anteriores que se puede usar como conjunto de entrenamiento para la regresión logística. Cada ejemplo, tiene las puntuaciones del solicitante en dos exámenes y la decisión de admisión ([2]). Podemos visualizar los datos en la Figura 2.18.

Una vez calculados los parámetros de la función de costo para la regresión logística mediante el descenso de gradiente (puede verse en la Figura 2.17), se obtiene el límite de decisión de la Figura 2.18.

Podemos usar el modelo para estimar la probabilidad de que un estudiante sea admitido. Por ejemplo, se puede observar que si ha obtenido una puntuación de 70 en el primer examen y de 50 en el segundo, no estaría admitido. De hecho, la probabilidad es de 0.342644.

Una forma de evaluar la calidad de los parámetros es ver qué tan bien predice el modelo nuestro conjunto de entrenamiento. Para ello, se calcula el porcentaje de ejemplos que acertó. En este caso, tiene una precisión del 89 %.

```
Cost at initial theta (zeros): 0.693147
Gradient at initial theta (zeros):
-0.100000
-12.009217
-11.262842
Local minimum found.
Optimization completed because the size of the gradient is less than the value of the optimality tolerance.
Cost at theta found by fminunc: 0.203498
Theta:
-25.161343
0.206232
0.201472
Train Accuracy: 89.000000
```

Figura 2.17. Proceso llevado a cabo por el programa para encontrar los valores óptimos

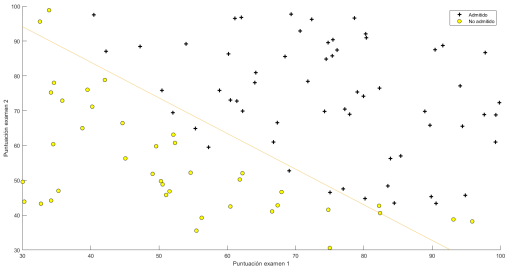

Figura 2.18. Límite de decisión de los datos de entrenamiento

2.5. Ejemplo de regresión logística polinómica y regularizada

Vamos a ver una regresión logística polinómica para predecir si los microchips de una planta de fabricación pasan el control de calidad. Durante el control de calidad, cada microchip pasa por varias pruebas para garantizar está funcionando correctamente. Tenemos los resultados de prueba para algunos microchips en dos pruebas diferentes, con las que queremos determinar si los microchips deben ser aceptados o rechazados ([2]). Para tomar la decisión, se tiene un conjunto de datos de resultados de pruebas en microchips anteriores, a partir de los cuales puede construir un modelo de regresión logística. Sin embargo, dichos datos tienen la forma de la Figura 2.20.

En este caso, obtenemos los parámetros usando la función de costo de la regresión logística polinómica y vamos a regularizarla. Vamos a estudiar cómo cambia el límite de decisión en función del parámetro, λ, para comprender cómo la regularización evita el sobreajuste.

Una forma de ajustar mejor los datos es crear más características a partir de cada dato. Asignaremos las características a todos los términos polinomiales de x_1 y x_2 hasta la sexta potencia. La función tendrá la siguiente forma:

$$h_{\boldsymbol{\theta}}(x) = \sum_{i=0}^{6} \sum_{j=0}^{6} \theta_{ij} x_1^i x_2^j$$

Como resultado, el clasificador tendrá un límite de decisión más complejo y no será lineal cuando se dibuje en nuestra gráfica bidimensional.

Ahora se utiliza la función de costo regularizada y el descenso de gradiente (Figura 2.19), para obtener todos los parámetros y poder pintar el límite de decisión. Luego, podremos variar el parámetro de regularización para entender como funciona.

Con un λ pequeño, el clasificador predice casi exactamente los datos de entrenamiento, creándose una frontera de decisión compleja, por lo que sobre-

```
function [J, grad] = costFunctionReg(theta, X, y, lambda)

% Initialize some useful values
m = length(y); % number of training examples

% You need to return the following variables correctly
J = 0;
grad = zeros(size(theta));

J=(1/m) * sum((-y .* log(sigmoid(X*theta)))- ((1-y) .* log(1-sigmoid(X*theta))));
J=J+ (lambda/(2*m)) * sum(theta(2:size(theta)).*theta(2:size(theta)));

grad(1)=(sum(((sigmoid(X*theta)-y).* X(:,1)))/ m);

for j=2:size(theta)
    grad(j)=(sum(((sigmoid(X*theta)-y).* X(:,j)))/ m)+(lambda * theta(j)/m);
end
```

Figura 2.19. Algoritmo de descenso de gradiente para la regresión logística regularizada

ajusta los datos ("overfitting") (Figura 2.21). La predicción para un nuevo dato no resulta ser correcta.

Con un λ moderadamente más grande, se tiene un límite que todavía separa los positivos y los negativos bastante bien (Figura 2.22). Hace su papel de regularización.

Sin embargo, si λ tiene un valor demasiado alto, crea una frontera de decisión simple, donde no están separados de manera correcta los ejemplos positivos y negativos, subajusta los datos ("underfitting") (Figura 2.23).

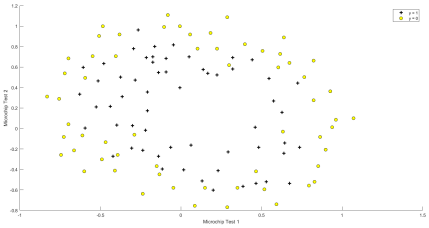

Figura 2.20. Visualización de los datos de entrenamiento, (y=1, aceptado; y=0, rechazado)

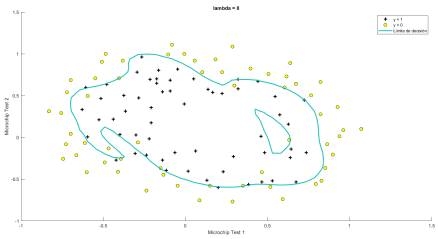

Figura 2.21. Límite de decisión con $\lambda = 0$

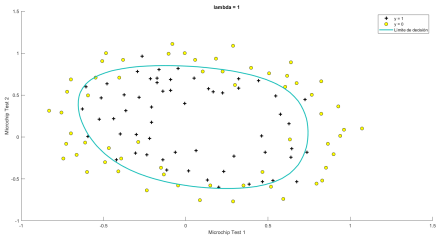

Figura 2.22. Límite de decisión con $\lambda = 1$

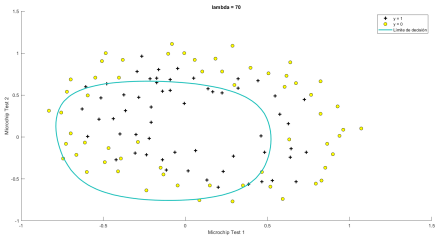

Figura 2.23. Límite de decisión con $\lambda = 70$

Conclusiones

Existe una amplia variedad de algoritmos de Ciencia de Datos disponibles que se pueden utilizar para abordar diferentes tipos de problemas. Tras estos algoritmos se encuentran principios matemáticos sólidos, que son esenciales para su implementación y para interpretar los resultados obtenidos. A lo largo de este trabajo hemos ilustrado que estos algoritmos tienen una amplia gama de aplicaciones en diversas industrias y campos, como el análisis de imágenes, entre otros. Cabe destacar la importancia de elegir el algoritmo adecuado para cada situación. Para ello hay que tener en cuenta la naturaleza de los datos y los objetivos del análisis. Esto requiere comprender las fortalezas y debilidades de cada algoritmo y evaluar su idoneidad para el problema en cuestión. Además, hemos podido proporcionar ejemplos de implementación de estos algoritmos que nos han permitido ilustrar cómo se aplican en la práctica. Estos ejemplos pueden ayudar a los lectores a comprender mejor cómo funcionan los algoritmos y su versatilidad.

Gracias a mi formación en matemáticas, pude comprender y apreciar los fundamentos matemáticos subyacentes en los algoritmos de Ciencia de Datos, lo que me permitió analizarlos. La capacidad de abstracción adquirida en la carrera fue esencial para comprender los conceptos teóricos detrás de los algoritmos. Además, durante la realización de este proyecto he tenido que aprender de forma autónoma nuevos contenidos. He tenido que buscar y analizar diferentes fuentes para obtener información relevante y adecuada, y aplicarla de manera efectiva en el trabajo realizado. También tomar decisiones sobre qué ejemplos y enfoques incluir. Estos nuevos conocimientos los he podido combinar con los adquiridos durante la carrera en asignaturas de estadística y análisis, sobre todo, y con el de MATLAB que aprendí en Métodos Numéricos II.

En definitiva, a través de este trabajo bibliográfico sobre algoritmos de Ciencia de Datos, he podido aprovechar los conocimientos adquiridos en la carrera de Matemáticas y las competencias desarrolladas. Estas habilidades fueron fundamentales para comprender, analizar y aplicar los algoritmos estudiados.

Bibliografía

[1] Amin, M., Kermani, S., Talebi, A. & Oghli M. (2015). Recognition of acute lymphoblastic leukemia cells in microscopic images using k-means clustering and support vector machine classifier. *Journal of medical signals and sensors*, 5(1), 49-58.

[2] Ng, A., Stanford Univ & DeepLearning.AI. (2023). CS229 Lecture Notes.

[3] Badii, M., Landeros, J. & Cerna E. (2008). Patrones de asociación de especies y sustentabilidad. *Daena: International Journal of Good Conscience*, 3(1), 632-660.

[4] Fernández, A. y López, D. (2018). Aplicación en los medios de prensa de un agrupamiento K-Means (Clustering K-Means). *Revista Chilena de Economía y Sociedad*, 12(1).

[5] Kotu, L., Engan, K., Borhani, R., Katsaggelos A., Ørn, S., Woie, L. & EftestØl, T. (2015). Cardiac magnetic resonance image-based classification of the risk of arrhythmias in post-myocardial infarction patients. *Artif Intell Med*, 64(3), 205-215.

[6] Schutt, R. y O'neil, C. (1994). *Doing Data Science: Straight Talk from the Frontline*. Addison-Wesley.

Referencias en línea o digitales con autor/a:

[7] Amat, J. (Noviembre de 2020). *Regresión logística con Python.* Recuperado de https://www.cienciadedatos.net/documentos/py17-regresion-logistica-python.html.

[8] Bagnato, J. (10 de julio de 2018). *Clasificar con K-Nearest-Neighbor ejemplo en Python.* Recuperado de https://www.aprendemachinelearning.com/clasificar-con-k-nearest-neighbor-ejemplo-en-python/.

[9] Berrendero, J. (16 de abril de 2020). *Tan cerca y tan lejos: la distancia de Mahalanobis.* Recuperado de

https://caminosaleatorios.wordpress.com/2020/04/16/tan-cerca-y-tan-lejos-la-distancia-de-mahalanobis/.

[10] Carretero, I. (27 de mayo de 2021). *ADVANCED REGRESSION AND PREDICTION: MACHINE LEARNING TOOLS.* Recuperado de https://bookdown.org/f100441618/bookdown-regresion/ml-tools.html.

[11] Fernández, A. *Programar algoritmo kNN desde cero en R.* Recuperado de https://anderfernandez.com/blog/programar-knn-r/.

[12] García, A. (26 de marzo de 2019). *Regresión lineal y descenso de gradiente con Python.* Recuperado de https://www.ellaberintodefalken.com/2019/03/regresion-lineal-descenso-de-gradiente.html.

[13] González, L. *Algoritmo KMeans.* Recuperado de https://aprendeia.com/algoritmo-kmeans-clustering-machine-learning/.

[14] Invarato, R. (9 de noviembre de 2016) *Hamming* Recuperado de https://jarroba.com/hamming/.

[15] Levin, K. (Noviembre de 2022). *STAT340 Lecture 10: logistic regression.* Recuperado de https://pages.stat.wisc.edu/~kdlevin/teaching/Fall2022/STAT340/lecs/L10_logistic.html.

[16] Martinez, J. (20 de septiembre de 2020). *Gradiente Descendiente para aprendizaje automático.* Recuperado de https://www.iartificial.net/gradiente-descendiente-para-aprendizaje-automatico/.

[17] Mexiquant. (13 de febrero de 2022). *Todo lo que necesitas saber sobre la regresión lineal.* Recuperado de https://mexiquant.substack.com/p/todo-lo-que-necesitas-saber-sobre.

[18] Parra, F. (25 de enero de 2019). *Estadística y Machine Learning con R.* Recuperado de https://bookdown.org/content/2274/metodos-de-clasificacion.html.

[19] Premanand, S. (27 de abril de 2023). *Building an End-to-End Logistic Regression Model.* Recuperado de https://www.analyticsvidhya.com/blog/2021/10/building-an-end-to-end-logistic-regression-model/.

[20] Ramírez, L. (5 de enero de 2023). *Algoritmo k-means: ¿Qué es y cómo funciona?* Recuperado de https://www.iebschool.com/blog/algoritmo-k-means-que-es-y-como-funciona-big-data/.

[21] Saini, A. (26 de agosto de 2021). *Conceptual Understanding of Logistic Regression for Data Science Beginners.* Recuperado de https://www.analyticsvidhya.com/blog/2021/08/.

[22] Shishodiya, I. (8 de abril de 2022). *Gradient Descent (with a little bit of scary maths, just a pinch worth).* Recuperado de https://medium.com/ml-concepts/gradient-descent-6a449eae1095.

[23] Sotaquirá, M. (6 de agosto de 2018). *La Neurona Artificial y la Regresión Logística.* Recuperado de https://www.codificandobits.com/blog/regresion-logistica-y-neurona-artificial/.

[24] Tovar, J. (28 de febrero de 2023). *Método Elbow y método de siluetas para la determinación del número de clústeres en K-means.* Recuperado de https://forum.huawei.com/enterprise/es/.

[25] Tangri, R. (8 de febrero de 2022). *Why Do We Use Gradient Descent for Optimization in Machine Learning?* Recuperado de https://towardsdatascience.com/why-do-we-use-gradient-descent-for-optimization-in-machine.

[26] Towards. (25 de enero de 2021). *Fully Explained K-means Clustering with Python.* Recuperado de https://towardsai.net/p/machine-learning/fully-explained-k-means-clustering-with-python.

Referencias en línea o digitales sin autor/a:

[27] Algoritmo de k vecinos más cercanos. (2020). En IBM https://www.ibm.com/mx-es/topics/knn.

[28] Aprendizaje automático en IA: método de descenso de gradiente. (31 de mayo de 2021). En forum.huawei https://forum.huawei.com/enterprise/es.

[29] El algoritmo k-means aplicado a clasificación y procesamiento de imágenes. En Unioviedo https://www.unioviedo.es/compnum/laboratorios_py/kmeans/kmeans.html.

[30] k-medias. (23 de diciembre de 2022). En Wikipedia https://es.wikipedia.org/wiki/K-medias.

[31] k-nearest neighbors algorithm. (8 de mayo de 2023). En Wikipedia https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm.

[32] Logistic regression. (21 de abril de 2023). En Wikipedia https://en.wikipedia.org/wiki/Logistic_regression.

[33] O que é e como funciona o algoritmo KNN? (2022). En Didática Tech https://didatica.tech/o-que-e-e-como-funciona-o-algoritmo-knn/.

[34] Valores faltantes. Universidad de Valencia https://www.uv.es/webgid/Descriptiva/23_valores_faltantes.html.

Software:

[35] MATLAB *MATLAB R2022a.* 2022. The MathWorks, Inc. https://es.mathworks.com/products/matlab.html.

Fundamentos de la Ciencia de Datos

Alba Maura Candelario Brito
Facultad de Ciencias • Sección de Matemáticas
Universidad de La Laguna
alu0101312293@ull.edu.es

Abstract

Data Science is a discipline that seeks to extract knowledge and generate valuable information from large volumes of data. We have explored the mathematical foundations and practical applications of the main algorithms used in this area: k-nearest neighbours, k-means, linear and logistic regression. They are used for classification and clustering of data and prediction of values.

1. k-nearest neighbors

It uses proximity to make classifications of new samples. A new observation is assigned to the label of the majority of the nearest observations (neighbours) by choosing the most appropriate distance.

Figure 1: *IBM: k-NN diagram*

Distances:

- Euclidean distance
- Manhattan distance
- Cosine similarity
- Jaccard coefficient
- Hamming distance

2. k-means

Aims to find k groups of similar data in a given set. The end result is a partition of the data into k groups, where the points within each group are similar to each other and different from those in the other groups.
It has a wide variety of applications such as image compression:

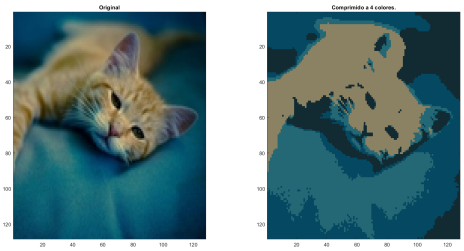

Figure 2: *Four-colour image compression*

3. Linear regression

The cost function is a function that attempts to determine the error between the estimated and actual values in order to optimise the expression parameters. For linear regression we are going to take the mean squared error (MSE):

$$J(\boldsymbol{\theta}) = \frac{1}{2n}\sum_{i=1}^{n}(h_\theta(\boldsymbol{x^{(i)}}) - y^{(i)})^2$$

where $h_{\boldsymbol{\theta}}(\boldsymbol{x^{(i)}}) = \theta_0 + x_1^{(i)}\theta_1 + \ldots + x_n^{(i)}\theta_n$.
The objective is to minimise this function using Stochastic Gradient Descend, the following iterative process:

$$\boldsymbol{\theta^{(t+1)}} := \boldsymbol{\theta^{(t)}} - \alpha\nabla_\theta J(\boldsymbol{\theta^{(t)}})$$

until $\|\boldsymbol{\theta^{(t+1)}} - \boldsymbol{\theta^{(t)}}\| < \delta$, the convergence tolerance.

4. Logistic regression

The objective is to automatically classify new data into the two categories using the already classified set. This is equivalent to finding a boundary that allows the data to be separated into two different groupings. We can define the cost function for the logistic regression such as:

$$J(\boldsymbol{\theta}) = -\frac{1}{n}\sum_{i=1}^{n} y^{(i)}\log(f(\boldsymbol{x^{(i)}})) + (1-y^{(i)})\log(1-f(\boldsymbol{x^{(i)}}))$$

where,

$$f(\boldsymbol{x^{(i)}}) = \frac{1}{1+e^{-h_\theta(\boldsymbol{x^{(i)}})}}$$

being $h_{\boldsymbol{\theta}}(\boldsymbol{x^{(i)}})$ the decision boundary.

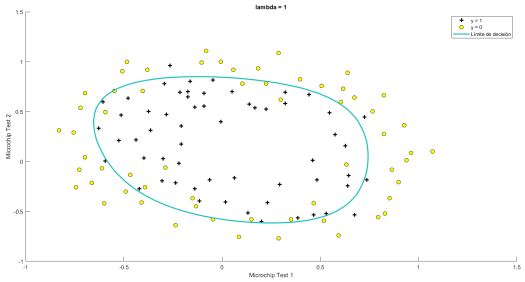

Figure 3: *Non-linear separator for logistic regression*

References

[1] Ng, A., Stanford Univ & DeepLearning.AI. (2023). CS229 Lecture Notes.

[2] Schutt, R. y O'neil, C. (1994). *Doing Data Science: Straight Talk from the Frontline.* Addison-Wesley.