REVISIÓN DE TÉCNICAS DE CLASIFICACIÓN SUPERVISADA Y SU APLICACIÓN A LA PREDICCIÓN DE RANGOS DE PRECIOS DE TELÉFONOS MÓVILES

IGNACIO FERNÁNDEZ SÁNCHEZ-PASCUALA

GRADO DE MATEMÁTICAS. FACULTAD DE CIENCIAS MATEMÁTICAS.
UNIVERSIDAD COMPLUTENSE DE MADRID

Trabajo Fin de Grado

27 de Junio de 2023

Director:

Juan Tinguaro Rodríguez González

Autorización de difusión

IGNACIO FERNÁNDEZ SÁNCHEZ-PASCUALA

18 de Marzo de 2023

El/la abajo firmante, matriculado/a en el Grado de Matemáticas de la Facultad de Ciencias Matemáticas, autoriza a la Universidad Complutense de Madrid (UCM) a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a su autor el presente Trabajo Fin de Máster: "REVISIÓN DE TÉCNICAS DE APRENDIZAJE AUTOMÁTICO Y APLICACIÓN A UN PROBLEMA PRÁCTICO", realizado durante el curso académico 2022-2023 bajo la dirección de Juan Tinguaro Rodríguez, y a la Biblioteca de la UCM a depositarlo en el Archivo Institucional E-Prints Complutense con el objeto de incrementar la difusión, uso e impacto del trabajo en Internet y garantizar su preservación y acceso a largo plazo.

Resumen

En este trabajo de final de grado se realiza una revisión minuciosa de técnicas de clasificación supervisada y su aplicación en la predicción de rangos de precios de teléfonos móviles.

En primer lugar, se introduce el concepto de aprendizaje automático y se exploran sus subtipos y diversas aplicaciones en el mundo actual. Se destaca la relevancia de esta disciplina en la resolución de problemas prácticos de vital importancia para la sociedad.

A continuación, se profundiza en los procesos implicados en el aprendizaje supervisado, poniendo especial énfasis en su aplicación para clasificación. Se examinan los aspectos fundamentales relacionados con la selección y preparación de datos y las distintas etapas del proceso de entrenamiento y evaluación de modelos.

Posteriormente, se desarrollan exhaustivamente algunas de las técnicas de aprendizaje más reconocidas para clasificación, como el método *k-NN*, los árboles de decisión, las redes neuronales artificiales y la regresión logística ordinal. Esta última técnica se aborda en particular debido a la naturaleza ordinal de la variable respuesta en el problema de predicción de rangos de precios.

Para corroborar la eficacia de las técnicas, se lleva a cabo un estudio computacional utilizando *Python* y un conjunto de datos específico de la plataforma *Kaggle* relacionado con la predicción de rangos de precios de teléfonos móviles. Se comparan y evalúan los resultados obtenidos por cada técnica, destacando el rendimiento superior alcanzado por la regresión logística ordinal respecto a las demás, que respalda su efectividad sobre conjuntos de datos de esta naturaleza.

Palabras clave

Aprendizaje Automático, Aprendizaje Supervisado, Clasificación, Redes Neuronales Artificiales, Arboles de Decisión, Regresión Logística Ordinal, K-Vecinos-Cercanos, Algoritmos.

Abstract

In this degree project, a thorough review of supervised classification techniques and their application in the prediction of mobile phone price ranges is carried out.

First, the concept of machine learning is introduced and its subtypes and various applications in today's world are explored. The relevance of this discipline in the resolution of practical problems of vital importance to society is highlighted.

Next, the processes involved in supervised learning are explored further,with special emphasis on its application for classification. The main aspects related to data selection and preparation and the different stages of the training and evaluation model process are examined.

Subsequently, some of the most recognized learning techniques for classification are exhaustively developed, such as the k-NN method, decision trees, artificial neural networks and ordinal logistic regression. The latter technique is addressed in particular because of the ordinal nature of the response variable in the price range prediction problem.

To corroborate the effectiveness of the techniques, a computational study is carried out using Python and a specific dataset of the Kaggle platform related to the prediction of mobile phone price ranges. The results obtained by each technique are compared and evaluated, highlighting the superior performance achieved by ordinal logistic regression with respect to the others, which supports its effectiveness on datasets of this nature.

Keywords

Machine Learning, Supervised Learning, Classification, Artificial Neural Networks, Decision Trees, Ordinal Logistic Regression, K-Nearest-Neighbours, Algorithms.

Índice general

Agradecimientos

En agradecimiento especial a mi familia, a mis padres y a mi hermano, los cuales me han proporcionado los recursos y confianza necesaria para realizar este grado y el desarrollo de este trabajo, sin ellos nada de esto hubiera sido posible.

Por otra parte, un especial agradecimiento a mis compañeros de clase durante la carrera, que me han aguantado en el día a día, y en especial a Javi, mi amigo y compañero con el que he compartido viajes, trabajos y dudas durante todos estos años.

A Juan Tinguaro, mi tutor del trabajo que me ha facilitado las fuentes necesarias para realizar este TFG, y con el que me he sumergido a aprender un poco sobre el mundo inmenso de conocimientos que se encuentran en este área.

Por último y no menos importante, agradecer a mis amigos y a María, que han estado en mis mejores y peores momentos personales, y siempre me han apoyado y confiado en mis capacidades para seguir a delante pese a cualquier situación.

Capítulo 1

Introducción

En un mundo donde cada vez se genera más volumen de datos, la capacidad de aprovechar y monetizar esta enorme cantidad de información se ha convertido en una necesidad inminente. El campo del aprendizaje automático ha nacido como una rama fundamental para enfrentar este desafío, proporcionando técnicas potentes para extraer conocimientos y patrones ocultos en los datos. Dentro de la inmensa variedad de técnicas dentro del aprendizaje automático, las técnicas de aprendizaje supervisado para clasificación han demostrado ser especialmente útiles en aplicaciones concretas. Este Trabajo de Fin de Grado se centra en el estudio y aplicación de algunas de estas técnicas con el objetivo de predecir el rango de precio de los teléfonos móviles a partir de sus características. Mediante este enfoque, se pretende no solo obtener predicciones precisas sobre el rango de precio de los móviles, sino también proporcionar una visión general de las técnicas de aprendizaje automático y su aplicación práctica en un problema relevante y en constante evolución.

Los principales objetivos de este proyecto son:

- Introducir el concepto de aprendizaje automático. Conocer su evolución histórica, su contexto actual, sus aplicaciones y los diferentes tipos de aprendizaje automático.
- Detallar todos los pasos y procesos necesarios para el correcto funcionamiento de un algoritmo de aprendizaje supervisado.
- Explicar algunas de las técnicas de aprendizaje supervisado para clasificación y sus posibles modelos en función del objetivo del estudio a realizar.
- Adaptar los datos y técnicas de aprendizaje según el interés de un estudio en particular.
- Predecir el rango de precio de un móvil a partir de sus características mediante los algoritmos presentados en el trabajo.
- Comparar el rendimiento de los algoritmos introducidos en el estudio realizado.

El TFG sigue la siguiente estructura: en el Capítulo 2 se presentan los fundamentos teóricos del aprendizaje automático, sus distintas subdivisiones y los procesos necesarios para el correcto funcionamiento del aprendizaje supervisado. El Capítulo 3 explora diversas técnicas de clasificación supervisada para clasificación. El Capítulo 4 presenta el estudio computacional sobre el conjunto de datos aplicando las técnicas descritas y analizando sus resultados. Y el Capítulo 5 recopila las reflexiones finales del trabajo. Además, se incluye un anexo con el código de *Python* con el que se ha realizado el estudio.

Capítulo 2

Preliminares

El *Machine Learning*, o aprendizaje automático, es una rama de la inteligencia artificial cuya finalidad es transmitir a las máquinas la capacidad de aprender a través de su entrenamiento y experiencia. Los algoritmos de Machine Learning permiten a los ordenadores aprender mediante unos datos de entrada, identificando los patrones de interés en ellos, a veces imposibles de observar por el humano, con el objetivo de predecir futuros datos. Es decir, convierten los datos en información y con esta información pueden tomar decisiones. Estos algoritmos se van nutriendo a medida que se van entrenando con más datos. Para ello es muy importante la calidad de los datos seleccionados y la cantidad de ellos, para poder realizar un estudio robusto con resultados más precisos.

El aprendizaje automático ha adquirido una especial relevancia en los últimos años. Herramientas como ChatGPT, basada en redes neuronales, están en boca de todos debido a su capacidad de procesar el lenguaje natural y generar respuestas como un humano, casi por arte de magia. Esta creciente relevancia se asocia a que la cantidad de datos generados por la sociedad es cada vez mayor, por lo que las empresas acumulan grandes volúmenes de datos procedentes de diversas fuentes, tecnología conocida como *Big Data*. Esto ha llevado a un aumento en la demanda de expertos en Machine Learning capaces de procesar y analizar estos datos y convertirse en una tecnología clave en muchos campos, incluyendo la informática, como en el desarrollo de sistemas autónomos y robots, la medicina, con la mejora de la atención médica, la industria, optimizando los procesos empresariales, el comercio o la identificación de patrones climáticos, entre otros.

Desde un punto de vista matemático, el aprendizaje automático se puede entender como un conjunto de técnicas y métodos estadísticos y/o computacionales que permiten modelar la relación entre variables y realizar predicciones a partir de un conjunto de datos. En particular, este aprendizaje utiliza técnicas de optimización para ajustar los parámetros de los modelos a los datos de partida, con el objetivo de minimizar algún criterio de error.

A diferencia de la programación tradicional, dónde se dispone de unos datos de entrada a los cuales se les ejecuta un programa con un modelo, para así obtener resultados, en el aprendizaje automático se da de entrada una serie de datos y sus resultados esperados, y es la propia máquina la que induce un modelo a partir de éstos, como ilustra la Figura 2.1.

En este capítulo se introduce el aprendizaje automático, tanto sus orígenes, como sus aplicaciones y subtipos. Se pondrá el foco en el aprendizaje supervisado para clasificación, explicando los procedimientos y técnicas que intervienen en su proceso de aprendizaje.

Un poco de historia

Más allá de su reciente popularidad, el aprendizaje automático existe desde hace unas décadas. Los orígenes de esta idea se dieron en 1943[2], a través del matemático Walter Pitts y el neurofisiólogo Warren McCulloh, aunque no fue hasta 1950 cuando se empezó a estudiar en más profundidad, con Arthur Samuel como pionero, creando en 1952 el primer

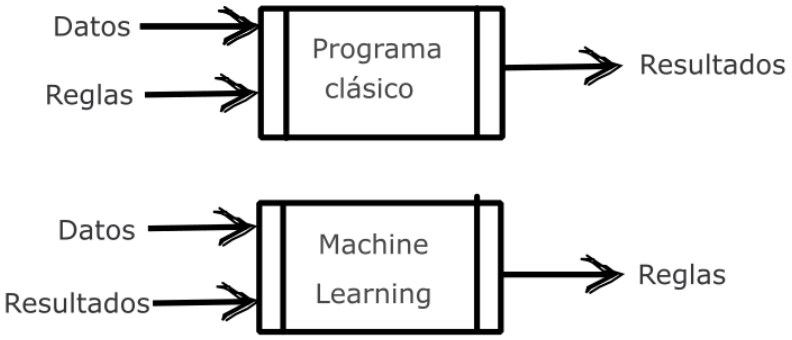


Figura 2.1: Esquema Programación Tradicional vs Machine Learning[1]

programa de cómputo, capaz de jugar a las damas[3]. Un año antes, en 1951 se inventó la primera red neuronal artificial[4]. En 1956, se crea el término de *"Artificial Intelligence"* en una conferencia científica en Darthmouth[5]. Fue en la década de 1960 cuando el *machine learning* se consolidó como un área de investigación importante dentro de la inteligencia artificial, gracias al desarrollo de nuevas técnicas estadísticas y computacionales. Se fue expandiendo durante las décadas de 1970 y 1980, apareciendo nuevos modelos como "*the nearest neighbour algorithm*" o redes neuronales con varias capas. Aun así, todavía era una rama emergente con aplicaciones limitadas.

A partir de 1990, se introdujeron técnicas más sofisticadas como algoritmos de "*Boosting*" para mejorar la predicción de un modelo, o los "*Random Decision Trees*". Además, en este momento se comienza a llevar a la práctica estas técnicas, sobretodo en la industria, un ejemplo es un reconocimiento de caracteres en cheques bancarios creado por Yann LeCunn en 1989[6]. En las décadas posteriores, se ha producido un crecimiento exponencial debido a los avances en tecnología de la información y computación, así como el aumento de datos disponibles. En la actualidad, el aprendizaje automático tiene diversidad de aplicaciones que veremos a continuación. Y, esto es solo el principio de su desarrollo. Se prevé que cada vez tenga mayor trascendencia en la sociedad.

Algunas aplicaciones

Como se menciona anteriormente, el auge en el conocimiento sobre aprendizaje automático ha conducido a su aplicación en numerosas áreas, cada vez de forma más útil y precisa. Algunas de estas aplicaciones son:

- Recomendaciones de productos:[7] Los sitios web de comercio electrónico ahora utilizan el comportamiento anterior de un comprador para ofrecer recomendaciones de productos que podrían gustarle. Por ejemplo, en el caso de plataformas como Netflix o HBO, se basan en las búsquedas y comportamientos de sus clientes para destacarles nuevas películas o series que coincidan con sus gustos. Ahora la publicidad en cualquier servicio web se adapta al usuario y no es general para todos, algo que no sucedía antes.

- Coches autónomos:[8] La conducción autónoma se refiere a un vehículo completamente

autónomo que está controlado por un sistema de conducción automático y no necesita la intervención de conductores físicos. En estos vehículos, los modelos obtenidos mediante aprendizaje automático recopilan datos de su entorno a través de cámaras y sensores, interpretan estos datos y toman decisiones sin intervención humana. En este momento compañías como Tesla se centran en su desarrollo, aunque por ahora solo han salido a mercado vehículos semiautónomos.

- Asistentes Virtuales:[9] Los asistentes personales virtuales son una de las aplicaciones más comunes entre la población. Entre ellos destacan Siri o Alexa, que ya aparecen en muchos de nuestros dispositivos para interactuar con nosotros, usados como métodos de consulta de información. Funcionan por reconocimiento de voz, y disponen de toda la información almacenada en la web. Se ajustan a nuestras preferencias para ofrecer el mejor servicio a cada usuario.

2.1. Tipos de *Machine Learning*

Es habitual distinguir diferentes tipos o modalidades de aprendizaje automático en función de la naturaleza de los datos disponibles y de la tarea que se necesite abordar. Entre ellos caben destacar el aprendizaje supervisado, el aprendizaje no supervisado y el aprendizaje por refuerzo[10].

2.1.1. Aprendizaje supervisado

El aprendizaje supervisado se enfoca en la tarea de aprender una función que toma inputs/variables explicativas, y los convierte en un output/variable respuesta, tratando de ajustarse a las relaciones entre inputs y outputs observadas en los datos. El objetivo es encontrar una función que pueda predecir la salida de un sistema a partir de un conjunto de datos de entrada[11].

Para ello, se considera un conjunto de datos de la forma $T = \{(x_1, y_1), \ldots, (x_n, y_n)\}$, donde n indica el número de ejemplos en los datos. Por un lado, $x_i = (x_{i1}, \ldots, x_{ip})$ $\forall i \in \{1, \ldots, n\}$ representan las variables explicativas del dato i, siendo p su dimensionalidad, e y_i representa su variable respuesta. La tarea del aprendizaje supervisado es encontrar una función f que pueda predecir el valor de y a partir de los valores de x:

$$\begin{aligned} f : X &\to Y \\ x &\mapsto f(x) = \hat{y} \end{aligned} \tag{2.1}$$

donde $X = (X_1, \ldots, X_p)$.

El objetivo de este aprendizaje es encontrar la mejor función f que pueda predecir de manera precisa los valores de la variable respuesta para nuevas instancias en los inputs que no se han visto antes. Para lograr esto, es necesario disponer de un conjunto de datos de entrenamiento con un tamaño de datos suficiente para aportar la información necesaria, y así lograr un buen rendimiento en el problema a modelar.

El proceso de aprendizaje implica ajustar los parámetros de la función f para minimizar la diferencia entre las predicciones de f y los valores reales de la variable respuesta Y en

el conjunto de entrenamiento, generalmente se utiliza una *función de perdida* para evaluar esta diferencia.

Esta función de perdida puede ser de diferentes tipos en función del algoritmo de aprendizaje usado y la naturaleza de la variable respuesta. Algunos algoritmos de aprendizaje se describen en el Capítulo 3, tanto su funcionamiento como la elección apropiada de sus parámetros en cada iteración en el proceso de aprendizaje y su función de perdida correspondiente. La elección del modelo o del tipo función dependerá de la naturaleza del problema y de los datos disponibles.

Dependiendo del tipo de variable respuesta existen dos tipos de tareas:

- Clasificación: La variable respuesta es una variable categórica o discreta que puede tomar un número finito de valores. Si solo hay 2 valores posibles ("Sí" o "No") se considera clasificación binaria, si hay más (por ejemplo en variables cualitativas: colores, forma, emociones...), se denomina clasificación multiclase. En ambos casos el modelo es de la forma introducida en la ecuación 2.1. Además, en clasificación, realizando una codificación de variables categóricas si es necesario, se tiene que $Y = \{1, \ldots, L\}$, siendo L el número de clases.

  La clasificación se utiliza, entre otras muchas cosas, para el reconocimiento de objetos en imágenes, reconocimiento de voz, clasificación de correos electrónicos como "spam" o "no spam", clasificación de fraude en transacciones financieras, etc. Por ejemplo, para el reconocimiento de dígitos numéricos manuscritos[12], la variable respuesta son los distintos números del 0 al 9 y las variables explicativas son las características de la imagen del dígito que se utilizan para hacer la predicción: La intensidad de los píxeles, la forma y orientación de los trazos, la relación entre los componentes del dígito...

  Algunas técnicas de aprendizaje comunes de clasificación son el *K-Nearest-Neighbours*, los árboles para clasificación y las redes neuronales artificiales, descritos detalladamente en el Capítulo 3 junto a algunas otras.

- Regresión: La variable respuesta es una variable numérica que puede tomar cualquier valor dentro de un rango continuo, es decir, $Y \subset \mathbb{R}$. En este caso el modelo es de la forma:

$$\hat{y} = f(x) + \varepsilon \tag{2.2}$$

  donde ε representa el error aleatorio, debido a la variabilidad de la variable respuesta al ser continua.

  La regresión se puede utilizar para la predicción del valor futuro de una inversión en función de los datos históricos del mercado, predicción del clima para modelar la relación entre variables meteorológicas, como la temperatura, la presión atmosférica y la humedad, etc[13]. Por ejemplo, en la predicción del precio de una casa, la variable respuesta podría ser el precio en euros y las variables explicativas podrían ser el tamaño de la casa, el número de habitaciones, la presencia de garaje, la ubicación, etc.

  Algunos modelos comunes de regresión son la regresión lineal, la regresión polinómica, la regresión logística y los árboles para regresión.

En los posteriores puntos del trabajo, se pondrá el punto de mira en el aprendizaje supervisado, más concretamente en clasificación, desde las fases previas, llegando a la inducción de los distintos modelos a partir del conjunto de datos de entrenamiento y su aplicación práctica.

2.1.2. Aprendizaje no supervisado

A diferencia del aprendizaje supervisado, el aprendizaje no supervisado se enfoca en descubrir patrones y estructuras en un conjunto de datos, que puedan ser utilizados para generar conocimiento útil y relevante o para simplificar los datos, sin la necesidad de una variable respuesta, trabajando solo con variables explicativas.

Se define como un proceso en el que se busca encontrar una función f que pueda asignar una entrada x a una salida y, previamente desconocida, ya que no hay variable respuesta en los datos. En lugar de eso, se trabaja con un conjunto de datos $T = \{x_1, x_2, \ldots, x_n\}$ en el que cada x_i representa un vector de variables explicativas que puede ser utilizado para entrenar y ajustar el modelo f.

Algunos de los métodos más utilizados en aprendizaje no supervisado son los siguientes[11]:

- Reducción de la dimensionalidad: Este método se utiliza para reducir el número de variables explicativas en los datos, manteniendo al mismo tiempo la información relevante. Algunos de los métodos más comunes son Análisis de Componentes Principales (ACP), *t-Distributed Stochastic Neighbor Embedding* (t-SNE) y el Escalado Multidimensional Métrico[14].

- Clustering: Es el más común y se utiliza para agrupar los datos en diferentes clusters o grupos en función de la similitud entre ellos. Algunos de los métodos de clustering más populares son: *Hierarchical clustering*, *K-means* y *Density-Based Spatial Clustering of Applications with Noise* (DBSCAN)[15].

- Reglas de asociación: Se utiliza para identificar patrones entre variables, especialmente en el ámbito del análisis de mercado y ventas. El algoritmo más conocido es el Apriori[16].

El aprendizaje no supervisado se utiliza en el análisis de redes sociales para detectar comunidades o grupos de usuarios con intereses y comportamientos similares, en la detección de anomalías o outliers, o la clasificación automática de documentos, entre otras aplicaciones.

2.1.3. Aprendizaje por refuerzo

Por último, el aprendizaje por refuerzo se basa en recompensar los comportamientos deseados y penalizar los no deseados. En este caso, no utiliza pares de datos de entrada-salida y la evaluación del sistema ocurre de forma concurrente con el aprendizaje. Un agente es capaz de percibir e interpretar el entorno, ejecutar acciones y retro alimentarse a través de prueba y error. Este proceso se denomina exploración. El comportamiento del agente debe de ser tal que escoja acciones que tiendan a incrementar a largo plazo la suma de las recompensas totales, lo que implica aprender una política de decisión en cada estado. Ésta se puede representar como una función que asigna un estado a una acción. Este proceso se

denomina explotación. El objetivo es encontrar un equilibrio adecuado entre la explotación y la exploración.

En el aprendizaje por refuerzo, se usan distintos algoritmos o métodos, como los métodos de Montecarlo, el algoritmo *Q-learning* o los procesos de decisión de Markov, entre otros. Cada uno utiliza diferentes técnicas para ajustar la política de decisión y maximizar la recompensa[17].

El juego es uno de los campos más utilizados para poner a prueba el aprendizaje por refuerzo, por ejemplo al enseñar a una máquina a jugar a videojuegos o juegos de ordenador[18]. Este tipo de aprendizaje se aplica también en otras áreas como la robótica, la optimización de recursos, como la gestión de sistemas de energía y suministro de agua, o sistemas de control.

2.2. Aprendizaje para clasificación supervisada

Como se menciona anteriormente en el Capítulo 2, en este trabajo se pone el foco en el aprendizaje supervisado para clasificación, o clasificación supervisada. Para llevar a cabo este proceso de aprendizaje que se quiere inducir a la máquina sobre unos datos en particular, hay que tener en cuenta una serie de conceptos y atravesar por distintas fases que se tratarán en esta sección, cada una de ellas importante para que el resultado final sea el esperado y la función f que asocie futuros inputs a outputs sea la más correcta en la medida de lo posible.

2.2.1. Preprocesos

En primer lugar, es primordial que los datos utilizados para inducir la función de predicción sean los apropiados. Los datos son el punto de partida del proceso de aprendizaje y una mala selección e interpretación de estos puede conducir a múltiples errores. Por ello, se realizan fases previas para la selección y limpieza de los datos:

- **Recopilación de los datos**

  En esta primera fase se seleccionan los datos sobre los cuáles se quiere realizar el estudio. Es importante entender qué estudio se va a realizar y cuál es su objetivo antes de realizar la selección, para así elegir correctamente la variable respuesta y las variables explicativas utilizadas. Estos datos pueden ser recolectados por fuentes propias, como encuestas, o por medios externos, generalmente por vía de bases de datos de estatutos oficiales y reconocidos, donde los datos cuentan con la verificación de expertos en la materia. Para que estos datos sean capaces de predecir futuras respuestas correctamente, tienen que ser descritos de una forma adecuada y precisa. Además es importante que estos datos reflejen de forma representativa la realidad para así intentar reproducirla con la mayor exactitud posible. Datos aleatorios o caóticos tienden a producir resultados confusos en los algoritmos.

  También hay que tener en cuenta la cantidad y variedad de estos, cuánta más información esté disponible mejor será la predicción y el funcionamiento del algoritmo. Por ejemplo si la variable respuesta posee 4 clases distintas, es conveniente que haya suficientes ejemplos de cada una de ellas. Aun así se indicará cómo solventar este tipo

de carencias en muestras no balanceadas, por medio de técnicas aplicadas al conjunto de entrenamiento, definido más adelante.

- **Preparación y Limpieza de los datos**[19]

  Tan importante es la selección de los datos como un análisis y filtrado sobre ellos. Sea cual sea la fuente de los datos, pueden sufrir de imperfecciones que el analista debe entender. Se requiere que los datos para entrenar a la máquina sean suficientes y relevantes, ya que de no ser así, la capacidad de generalización del modelo se puede ver afectada, y provocar subajuste o sobreajuste, conceptos introducidos en la siguiente sección.

  Por un lado, la presencia de variables explicativas irrelevantes en la muestra, la ausencia de otras importantes o la aparición de variables redundantes puede alterar la precisión de los resultados esperados. Por ejemplo, se puede estar prediciendo el precio de una casa y no tener en cuenta el número de habitaciones o por el contrario, tener en cuenta la altura del comprador, que no debería influir en el precio, o por ejemplo, usar la edad de una persona y su fecha de cumpleaños a la vez puede confundir a la máquina. Una buena técnica para hacer un orden de importancia de las variables explicativas son los árboles de decisión, introducidos en el Capítulo 3.

  Otro problema son los datos "missing", consiste en datos a los que les faltan algún valor en alguna variable explicativa, la mayoría de las técnicas no son capaces de procesar estos datos. Una opción de solucionar este problema es inducir un nuevo clasificador a partir de los demás datos sin escasez de valores en esa variable explicativa para predecir éstos que faltan, donde en este caso la variable respuesta será la variable explicativa de la que faltan datos. Otra forma de solventar este problema es asignarle el valor más común en la muestra o un valor que coincida con el de otro ejemplo similar.

  Por último, es posible que haya ruido sobre las etiquetas/clases que se quieren clasificar, algunos ejemplos se mueven en una "nube gris" entre 2 clases, lo que provoca que futuros resultados de ejemplos parecidos sean clasificados incorrectamente.

2.2.2. Generalización a nuevos datos

En el aprendizaje supervisado para clasificación, el objetivo es crear un modelo/función que pueda clasificar de manera precisa nuevas observaciones que no se utilizaron en el proceso de entrenamiento. Para lograr este objetivo, es necesario que el modelo tenga la capacidad de generalizar, es decir, que pueda capturar patrones subyacentes en los datos y utilizarlos para hacer predicciones correctas en nuevas observaciones, eliminando a su vez el ruido y los patrones irrelevantes.

Para ello, es necesario primero elegir el algoritmo de aprendizaje y sus hiperparámetros adecuados, desarrollado en el Capítulo 3. Durante el entrenamiento, se van ajustando los parámetros del modelo para minimizar una función de pérdida o maximizar una función objetivo, según el algoritmo elegido, para así generalizar bien a nuevos datos posteriormente. Hay que tener en cuenta que el objetivo principal es que el modelo clasifique bien futuras

entradas, no los datos de entrenamiento. Para cumplir este objetivo es necesario evitar el sobreajuste y subajuste.

Sobreajuste y Subajuste[20]

Una de las principales preocupaciones en el aprendizaje supervisado en clasificación es evitar el sobreajuste, también denominado "overfitting", que ocurre cuando el modelo se ajusta demasiado bien a los datos de entrenamiento y no puede generalizar bien para nuevos datos. En este caso, el modelo se adapta demasiado a las particularidades de los datos de entrenamiento, incluyendo el ruido y las características irrelevantes, y no puede identificar patrones que sean relevantes para nuevos datos. Este fenómeno se puede observar en la primera gráfica de la figura 2.2, donde la función clasifica correctamente todos los datos de entrenamiento, sin capturar la tendencia general en los datos.

El subajuste, también llamado "underfitting", por otro lado, se produce cuando el modelo es demasiado simple y no es capaz de aprender patrones complejos en los datos. Éste se ve representado en la tercera gráfica de la figura 2.2, donde se realiza una clasificación demasiado simple de los datos, sin apenas capturar ningún patrón en ellos.

Es importante encontrar un equilibrio entre el sobreajuste y el subajuste para crear un modelo que pueda generalizar de manera efectiva a nuevos datos. Este equilibrio se puede ver reflejado en la segunda gráfica de la figura 2.2, donde se ve un balance entre las otras dos gráficas. En general, esto se logra ajustando la complejidad del modelo y mediante el uso de técnicas de regularización (por ejemplo, nuevos hiperparámetros en la función de pérdida)[21].

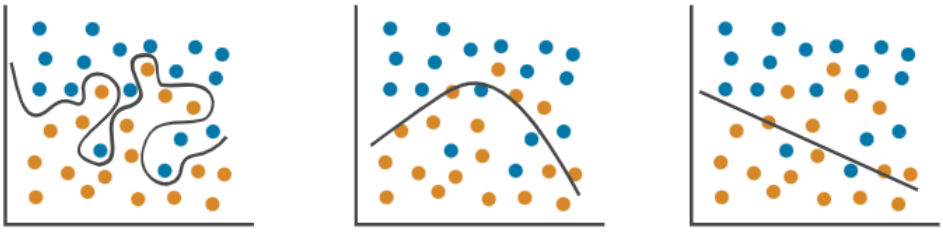


Figura 2.2: Representación Sobreajuste vs Ajuste Correcto vs Subajuste para clasificación[22]

Complejidad del modelo[19]

La capacidad del modelo para generalizar correctamente depende de su complejidad, directamente relacionada con el subajuste y sobreajuste. Esta complejidad se refiere a la capacidad del modelo para ajustarse a los datos. Un modelo demasiado simple no será capaz de capturar patrones complejos en los datos, y puede tener un mal rendimiento tanto en los datos de entrenamiento como en futuros datos, tendiendo al subajuste, mientras que un modelo demasiado complejo puede sobreajustar los datos. La complejidad de un modelo viene determinada principalmente por el algoritmo de aprendizaje utilizado y sus hiperparámetros.

Dependiendo del estudio que queramos realizar y el conjunto de datos disponible, será

más conveniente utilizar algoritmos de aprendizaje más complejos, como redes neuronales profundas, o más simples, como los árboles de decisión, ambos estudiados en el Capítulo 3.

Además del algoritmo de aprendizaje, hay que tener en cuenta los hiperparámetros utilizados en el modelo, ya que pueden influir en la complejidad del mismo y su capacidad para generalizar. Estos parámetros se establecen antes de entrenar el modelo y se deben seleccionar de forma cuidadosa para evitar el sobreajuste y subajuste.

Por ejemplo, en modelos basados en redes neuronales, la complejidad del modelo dependerá del número de neuronas utilizado y el número de capas. También en árboles de decisión, la profundidad del árbol puede controlar la complejidad del modelo. Un árbol de decisión más profundo suele conducir a un modelo más complejo, que se ajusta mejor a los datos. Se introducirán técnicas más adelante para seleccionar de manera correcta el algoritmo de aprendizaje y los hiperparámetros iniciales, como la validación cruzada.

También es necesario tener en cuenta que ante dos explicaciones igual de eficaces del mismo fenómeno, suele ser mejor la más simple, esto se denomina "principio de parsimonia". Esto se debe a que los modelos más simples suelen ser más fácil de entender e interpretar y más eficientes computacionalmente, con un menor costo en el entrenamiento y predicción.

El objetivo es encontrar un equilibrio entre la complejidad del modelo y su capacidad para generalizar. Por esto, surge la necesidad de poder evaluar la capacidad de generalización de un modelo, y así comparar la actuación sobre los datos entre distintos modelos. Para ello son necesarios datos nuevos donde ponerlos en práctica y ver cuál es su capacidad de predicción.

Separación de los datos

Para evaluar la capacidad de generalización del modelo, es común dividir los datos en conjunto de entrenamiento (*"training data"*) y prueba (*"testing data"*). El conjunto de entrenamiento será el utilizado por nuestro modelo de aprendizaje supervisado para aprender a través de alguna de las técnicas descritas en el Capítulo 3. Por otro lado, en el conjunto de prueba/test es donde se comprueba la efectividad del clasificador inducido en el conjunto de entrenamiento, es decir, una estimación de la capacidad de generalización que se produciría al entrenar el modelo con todos los datos disponibles. Esta efectividad se puede calcular por distintas métricas, introducidas en la sección 2.3, que determinarán la capacidad de predicción del modelo.

Diferentes conjuntos de entrenamiento y prueba conducen a diferentes funciones de predicción y, por ende, a distintos resultados de evaluación en futuras actuaciones, por tanto la precisión del modelo cambia. De cara a obtener una estimación más robusta, es decir, menos dependiente de la división aleatoria de los datos en entrenamiento y test, se puede realizar el proceso repetidas veces, tomando diferentes muestras de entrenamiento y test, y promediar resultados. En particular se puede usar distintos tipos de validación cruzada para esto[19]:

- 5x2 Cross Validation: Sea un conjunto de datos $T = \{(x_1, y_1), \ldots, (x_n, y_n)\}$, donde (x_i, y_i) es un par entrada-salida. Se separan los datos en 2 subconjuntos aleatorios de mismo tamaño, T_1, T_2. En primer lugar, se usa T_1 como conjunto de entrenamiento, y T_2 como conjunto de prueba, obteniendo el resultado de evaluación E_{11}. Después se usa

T_2 como conjunto de entrenamiento y T_1 como conjunto de prueba, obteniendo E_{12}. El resultado de evaluación medio para esta partición será entonces de $E_1 = (E_{11} + E_{12})/2$. Repitiendo esta técnica 5 veces, para distintas particiones aleatorias del conjunto de datos en 2 mitades, se consigue llegar al resultado de evaluación general, dado por:

$$E = \frac{E_1 + E_2 + E_3 + E_4 + E_5}{5} \tag{2.3}$$

- K-fold Cross Validation: En este caso, se divide el conjunto de datos T en K subconjuntos de mismo tamaño, $T_1, T_2, \ldots, T_K$. En la primera iteración se usa T_1 como conjunto de entrenamiento, y la unión de los K-1 subconjuntos restantes como conjunto de entrenamiento, obteniendo el resultado de evaluación E_1. Seguidamente, se usa T_2 como conjunto de prueba y así sucesivamente, hasta conseguir K resultados de evaluación diferentes. Una vez estimadas las evaluaciones parciales, se calcula el resultado de evaluación total como la media de todas:

$$E = \frac{\sum_{i=1}^{K} E_i}{K} \tag{2.4}$$

Fold	**Conjunto entrenamiento**	**Conjunto prueba**	**Evaluación**
1	2-3-4	1	E_1
2	1-3-4	2	E_2
3	1-2-4	3	E_3
4	1-2-3	4	E_4

Cuadro 2.1: Esquema aplicación técnica 4-Fold Cross Validation.

Dependiendo del conjunto de datos disponible y del objetivo del aprendizaje, será conveniente utilizar una técnica de validación cruzada u otra. Se debe tener en cuenta que los conjuntos de entrenamiento deben tener la suficiente información relevante para poder generalizar bien en futuros datos. Si no es así, se pueden aplicar algunas técnicas para mejorar conjuntos de entrenamiento no balanceados, como el "Undersampling" o el "Oversampling"[23].

Sin embargo, uno de los objetivos de la separación de datos es comparar el balance complejidad/generalización entre distintos modelos con distintos algoritmos de aprendizaje e hiperparámetros, por ello, normalmente no se ajusta un único modelo con la muestra de entrenamiento, sino muchos. Y entonces, si se selecciona el modelo/hiperparámetros con mejores resultados en los conjuntos de test, se corre riesgo de que la métrica obtenida no refleje realmente la capacidad de generalización y se produzca sobreajuste, ya que se usan los datos de los conjuntos de prueba para la selección, y entonces la métrica no es independiente de esta selección[24].

Por esto, siempre que se ajusten modelos distintos, se suele usar una nueva validación cruzada sobre los datos de entrenamiento, en este caso con el método *K-fold Cross Validation*, para así obtener una métrica para cada modelo con la que realizar la selección previa a la

evaluación del modelo seleccionado sobre test. Los datos de test no se tocan hasta que tenemos un modelo final del que queremos evaluar qué tal va a generalizar.

Como tenemos varios conjuntos de entrenamiento de la primera separación del conjunto de datos, se realiza este proceso para cada uno de ellos, obteniendo el resultado de la evaluación por validación cruzada para cada modelo y conjunto de entrenamiento. Posteriormente, se selecciona el modelo con mejores resultados de predicción medios, y se evalúa sobre los conjuntos de test reservados.

El proceso global se puede ver resumido en el cuadro 2.2.

Modelo	**Train 1**	**Train 2**	**...**	**Train *k***	Resultado
1	CV_{11}	CV_{21}	...	CV_{k1}	$\overline{CV_1}$
2	CV_{12}	CV_{22}	...	CV_{k2}	$\overline{CV_2}$
...	...	...	...	...	...
n	CV_{1n}	CV_{2n}	...	CV_{kn}	$\overline{CV_n}$

Cuadro 2.2: Representación separación de los datos y selección mejor modelo, donde $\overline{CV_j} = (\sum_{i=1}^{k} CV_{ij})/k$ es el resultado de la evaluación del modelo j.

El modelo seleccionado será el $m = \arg\min_{j=1}^{n} CV_j$ ó $m = \arg\max_{j=1}^{n} CV_j$, en función de la métrica de evaluación elegida. Tras la selección, se entrena el modelo m en los distintos conjuntos de entrenamiento y lo evaluamos en sus respectivos conjuntos de test, obteniendo así la estimación de la capacidad de generalización final del modelo.

2.3. Métricas de evaluación

Una vez separado el conjunto de datos, se debe evaluar el rendimiento del modelo sobre el conjunto de test, sobre el cuál conocemos la categoría real a la que pertenecen. Existen diversas métricas de evaluación que permiten medir este rendimiento[19].

Se considera en primer lugar un conjunto de datos cuya variable respuesta es binaria, y sus 2 clases son "positivo" y "negativo", para posteriormente generalizarlo a variables respuesta multiclase.

Se presenta la *Matriz de Confusión* como:

	Predicción positiva	**Predicción negativa**
Real positivo	TP	FN
Real negativo	FP	TN

Cuadro 2.3: Matriz de confusión para un clasificador binario

Donde TP es el número de ejemplos positivos clasificados correctamente por el modelo en el conjunto de prueba, TN el número de ejemplos negativos clasificados como tal, FP los ejemplos negativos clasificados como positivos y FN los ejemplos positivos clasificados como negativos.

A partir de esta matriz, se presentan las siguientes métricas de evaluación:

- Exactitud: Es la proporción de predicciones correctas del modelo respecto al número total de ejemplos en el conjunto de prueba.

$$Acc = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.5}$$

  Es una métrica adecuada para conjuntos equilibrados, donde el número de ejemplos de cada categoría es parecido. A veces, se prefiere trabajar con la cantidad opuesta, denominada *tasa de error*, E.

$$E = \frac{FP + FN}{TP + TN + FP + FN} = 1 - Acc \tag{2.6}$$

- Recall: Es la proporción de ejemplos positivos que el modelo ha predicho correctamente respecto al número total de ejemplos positivos que había en el conjunto de prueba.

$$Re = \frac{TP}{TP + FN} \tag{2.7}$$

  Esta métrica es útil para conjuntos de datos desequilibrados o en los cuáles una clase es más importante que otra. En este caso se desea minimizar los falsos negativos. Por ejemplo en aplicaciones donde los costos de los falsos negativos pueden ser altos, como en la detección de enfermedades graves.

- Precisión: Es la proporción de verdaderos positivos sobre el total de predicciones positivas.

$$Pr = \frac{TP}{TP + FP} \tag{2.8}$$

  Se centra en la precisión de las predicciones positivas, es decir, cuántas de las instancias clasificadas como positivas son realmente positivas.

- F1-Score: Combina precisión y recall, es la media armónica de ambas métricas.

$$F_1 = \frac{2 \cdot Pr \cdot Re}{Pr + Re} \tag{2.9}$$

  Es una medida útil cuando se desea tener un equilibrio entre la precisión y el recall, es decir, cuando es importante tanto clasificar correctamente los casos positivos como minimizar los falsos negativos.

Es importante elegir la métrica de evaluación adecuada para el contexto de la aplicación y tener en cuenta que a veces una medida sola no es suficiente para evaluar el rendimiento del modelo.

Sea ahora un conjunto de datos cuya variable respuesta es multiclase. En este caso la métrica más común es la Exactitud, que corresponde al porcentaje de aciertos en las predicciones. También se pueden calcular algunas de las medidas anteriores de forma independiente

a cada una de las clases, suponiendo en cada caso como ejemplos positivos los de la categoría en cuestión y negativos todos los que pertenecen a las demás clases. Así se consideran las siguientes métricas de clasificación:

- Macro-Recall:

$$Re^M = \frac{1}{L}\sum_{i=1}^{L} Re_i \tag{2.10}$$

- Macro-Precision:

$$Pr^M = \frac{1}{L}\sum_{i=1}^{L} Pr_i \tag{2.11}$$

- Macro-F1:

$$F_1^M = \frac{2 \cdot Pr^M \cdot Re^M}{Pr^M + Re^M} \tag{2.12}$$

donde L es el número de clases en la variable respuesta.

En estas métricas, se le da la misma importancia a cada una de las clases, y se suelen utilizar en muestras balanceadas.

Desde otro enfoque, a cada clase se le da la importancia equivalente a su frecuencia en el conjunto de datos, esto puede ser útil en conjunto de datos no balanceados, aparecen así las siguientes métricas:

- Micro-Recall:

$$Re^\mu = \frac{\sum_{i=1}^{L} TP_i}{\sum_{i=1}^{L}(TP_i + FN_i)} \tag{2.13}$$

- Micro-Precisión:

$$Pr^\mu = \frac{\sum_{i=1}^{L} TP_i}{\sum_{i=1}^{L}(TP_i + FP_i)} \tag{2.14}$$

- Micro-F1:

$$F_1^\mu = \frac{2 \cdot Pr^\mu \cdot Re^\mu}{Pr^\mu + Re^\mu} \tag{2.15}$$

Si la variable respuesta es categórica ordinal, es decir, sus categorías tienen un orden ascendente o descendente, y se le ha asignado a cada categoría un número según su orden, es recomendable contemplar la media de la diferencia en valor absoluto de los rangos predichos respecto a los reales:

$$MDP = \frac{\sum_{i=1}^{m} |y_i - \hat{y}_i|}{m} \tag{2.16}$$

donde m es el número de datos en el conjunto de test.

2.4. Estudio a realizar

Para poner en práctica los conceptos sobre *machine learning* explicados en este trabajo, se realizará un estudio sobre un problema real que puede ser resuelto por algoritmos de aprendizaje supervisado para clasificación. En este caso, el estudio seleccionado es el siguiente[25]: Un empresario ha creado su nueva marca de móviles, y quiere hacer competencia a las grandes compañías como Apple, Samsung etc. Como no sabe cómo estimar el precio de sus móviles, ha investigado los datos de venta de distintas empresas, recogiendo el rango de precio de venta y las características de los móviles.

El objetivo del estudio será clasificar el rango de precio de un móvil dada sus características, mediante el uso de las técnicas introducidas en el Capítulo 3. La presentación del conjunto de datos, obtenido de la plataforma *Kaggle* y su estudio computacional será realizado en el Capítulo 4.

Capítulo 3

Técnicas de aprendizaje

En este capítulo se aborda el proceso de aprendizaje mediante el uso de distintos algoritmos para clasificación. Como se introdujo en el Capítulo 2, se parte de un conjunto de datos de la forma $T = \{(x_1, y_1), \ldots, (x_n, y_n)\}$, donde $x_i = (x_{i1}, \ldots, x_{ip}) \in X$ e $y_i \in Y$ $\forall i \in \{1, \ldots, n\}$, con $Y = \{1, \ldots, L\}$, siendo L el número de clases. Se presentará la inducción de distintas funciones a partir del conjunto de entrenamiento para predecir una nueva observación $\mathbf{x}$.

3.1. Árboles para clasificación

Los árboles para clasificación se basan en la búsqueda de patrones en los datos mediante una serie de "preguntas" sobre sus variables explicativas $X_1, \ldots, X_p$, con un orden determinado ,y cada una dependiente de la anterior, que conducen a distintas subdivisiones en los datos, hasta llegar finalmente a un subconjunto de datos al que se le asigna una categoría de la variable respuesta Y. Cada uno de estos subconjuntos finales equivale por tanto a una fórmula lógica sobre sus variables explicativas, y una vez inducido el árbol, un nuevo ejemplo será clasificado según la instancia lógica que satisfaga.

Esta secuencia de preguntas se representa mediante una estructura de árbol formada por nodos y ramas que los conectan. La clasificación de una nueva observación $\mathbf{x}$ empieza en el nodo raíz, que pregunta sobre el valor de una variable explicativa en particular. Las diferentes ramas desde este nodo corresponden a las distintas respuestas sobre los valores en esta variable. Según la respuesta, se sigue el enlace apropiado para acceder a un nodo inferior, dónde se repite nuevamente este proceso hasta llegar a un nodo final denominado hoja, que asocia finalemente una categoría a la observación. Esta estructura se puede ver representada en la figura 3.1.

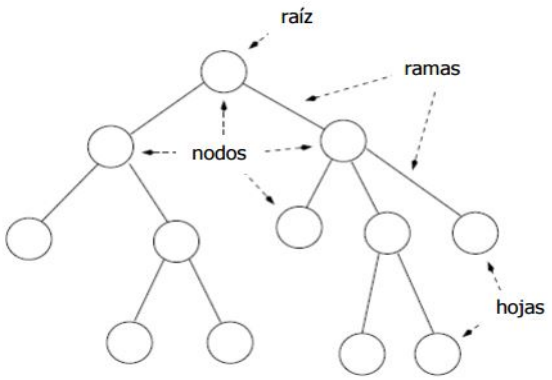


Figura 3.1: Esquema gráfico estructura de un árbol para clasificación genérico[26]

Una de las mayores ventajas de esta técnica respecto a las demás es su gran capacidad de interpretación, ya que no solo clasifica nuevas observaciones, si no que te indica el motivo de su clasificación. Además, su clasificación es inmediata, simplemente es seguir una ruta de decisiones lógicas por el árbol.

En los siguientes puntos se entra en los detalles de la creación de un árbol de decisión a partir de un conjunto de entrenamiento mediante el algoritmo *CART*.

3.1.1. *CART*

Una de las preguntas a la hora de crear un árbol de decisión es cómo ordenar los distintos tests de evaluación que se realizan en los nodos del árbol. Cada decisión en un nodo divide el conjunto de entrenamiento en subconjuntos cada vez más pequeños. Sería ideal si todos los ejemplos de un subconjunto perteneciesen a la misma categoría, en este caso se denomina subconjunto puro. De no ser así, hay que decidir si seguir dividiendo el subconjunto por otra propiedad o asignarle una categoría a éste por medio de algún criterio.

El algoritmo *CART* proporciona un marco general que puede ser implementado de varias formas para producir distintos árboles de decisión. En este algoritmo se usan diferentes criterios en la creación de un árbol, como se ve a continuación[27].

3.1.2. Número de divisiones en un nodo

El número de divisiones en un nodo está relacionado con la propiedad X_j, $j \in \{1, \ldots, p\}$ que se va a evaluar en él. Generalmente se establece por el usuario y varía según la propiedad.

El número de ramas descendentes desde un nodo se denomina factor de ramificación, y se denota como B. En el caso de variables categóricas, B suele ser el número de categorías, y en variables continuas, se suele dividir la muestra en dos por el punto de corte más óptimo.

También, cada decisión en un nodo se puede simplificar en una sucesión de decisiones binarias ("Sí" o "No"). Para ello, es necesario realizar una serie de manipulaciones sobre las variables explicativas. En algunos casos es común usar este tipo de árboles, debido a su poder expresivo y simplicidad.

3.1.3. ¿Qué test realizar en cada nodo?

Gran parte del trabajo en el diseño de un árbol viene de qué propiedad o variable explicativa evaluar en cada nodo. El objetivo es encontrar la propiedad X_t para cada nodo z que maximiza la "pureza" de sus nodos descendentes. Para ello, necesitamos medir el grado de pureza o impureza en un nodo z. La medida más popular para medir la impureza es la denominada *entropía de la información*[19]:

$$i(T) = -\sum_{j=1}^{L} P(j) \log_2 P(j) \tag{3.1}$$

donde T es el subconjunto de datos en el nodo z, $P(j)$ es la proporción de ejemplos en T de clase j. Si todos los ejemplos pertenecen a la misma categoría en T, entonces $i(T) = 0$, de no ser así, $i(T)$ alcanza su mayor valor cuando todas las categorías son equiprobables.

También se usa en ocasiones el *índice de Gini* como medida, sobretodo para predecir

variables binarias:

$$i(T) = \sum_{i \neq j} P(i)P(j) \tag{3.2}$$

Como el objetivo es medir la impureza de sus nodos descendentes según la propiedad elegida, calculamos la media en función de la propiedad X_j seleccionada:

$$i(T, X_j) = \sum_{b \in B_{X_j}} P(b) \cdot i(T_b) \tag{3.3}$$

donde B_{X_j} es el número de divisiones producidas por la propiedad X_j, T_b el subconjunto de T al que se llega tras la división b, y $P(b) = \frac{|T_b|}{|T|}$.

Finalmente, se selecciona X_t tal que maximice la cantidad de información del nuevo árbol, o lo que es lo mismo, la reducción de impureza en el árbol tras su crecimiento, representada por $\triangle(T, X_j)$:

$$\begin{aligned} \triangle(T, X_j) &= i(T) - i(T, X_j) \\ X_t &= \arg\max_{X_j} \triangle(T, X_j) \end{aligned} \tag{3.4}$$

En el caso de realizar una división sobre variables continuas, se debe elegir el umbral óptimo en el que realizar la división de la variable X_j, es decir el parámetro θ_j que divide una nueva observación $\mathbf{x}$ en el nodo según $x_j \leq \theta_j$ ó $x_j > \theta_j$. Para ello, se siguen los siguientes pasos[19]:

1. Ordenamos los valores x_{ij} de la variable X_j de menor a mayor en el conjunto de entrenamiento.

2. Elegimos los umbrales candidatos $\theta_{ij} = \frac{x_{ij} + x_{i+1j}}{2}$ tal que x_i e x_{i+1} pertenecen a clases distintas.

3. Para cada umbral θ_{ij} seleccionado, calculamos $\triangle(T, X_j)$ según su partición.

4. Seleccionamos el umbral θ_{ij} que maximice la cantidad de información del nuevo árbol tras la división.

Una de las ventajas de los árboles de decisión es que, tras construir el árbol, se pueden ordenar las propiedades según su orden de importancia. Esta importancia se determina sumando la ganancia de información en todos los nodos en los que se utiliza la propiedad para hacer divisiones.

3.1.4. Criterio de parada

Por un lado, los árboles poco profundos de menos niveles ganan en capacidad de interpretación y evitan capturar información irrelevante, pero pueden provocar un subajuste de los datos. Por otro lado, árboles muy profundos tienden a sobreajustar los datos. Debido a esto,

se debe considerar un criterio de parada, es decir, hasta qué punto debe seguir creciendo el árbol y subdividiendo los datos.

Una forma de controlar esto es mediante un hiperparámetro β de tamaño pequeño, de tal forma que un conjunto de datos T se siga subdividiendo por la propiedad X_j en un nodo si y solo si $\triangle(T, X_j) > \beta$.

Otra forma habitualmente usada, y compatible con la anterior, es añadir un hiperparámetro m, de tal forma que si llegamos a un subconjunto de datos en un nodo tal que $|T| \leq m$, no se subdivide más este subconjunto. Este número m suele representar un porcentaje del número total de datos en el conjunto de entrenamiento, en torno a un 5 %.

En ambos casos, tras la parada, se consideran estos nodos como hojas, a las que se asigna la categoría más común en el subconjunto de datos correspondiente. Además, estas hojas no tienen por qué estar todas en el mismo nivel. La idea es ir variando estos hiperparámetros en el modelo y verificar con cuál se consiguen mejores resultados en la validación cruzada.

3.1.5. Poda del árbol

En algunos casos no es necesario establecer un criterio de parada al crear un árbol de decisión. En su lugar, puede ser mejor construir todo el árbol, de tal forma que todas sus hojas estén formadas por subconjuntos puros. Al no detener la subdivisión temprano, se permite que el árbol explore todas las posibles subdivisiones y no se pierdan oportunidades de mejorar el rendimiento global del árbol.

Posteriormente, se puede aplicar una poda para simplificar el árbol y mejorar su generalización en datos nuevos. La poda consiste en sustituir uno o varios nodos internos por hojas, a las que se les asigna la categoría más común de los subconjuntos^{9} en estos nodos internos.

Una técnica de poda es el denominado *post-pruning* y se lleva a cabo mediante varios pasos. Primero se intercambia un nodo interno por un nodo hoja, luego otro, así hasta que todos los intercambios parezcan beneficiosos de acuerdo a algún criterio.

En este caso, el criterio será controlar la tasa de error ponderada de el árbol, definido por recursión sobre la estructura del árbol:

$$E = \begin{cases} \frac{e+1}{N+|T|} & \text{si el árbol es una hoja} \\ \sum_{b \in B} P(b) \cdot E_b & \text{si no es una hoja} \end{cases} \tag{3.5}$$

donde e representa el número de ejemplos mal clasificados en la hoja, T el subconjunto de datos que se encuentra en la hoja, N el número total de datos en el conjunto de entrenamiento, B el número de hijos del nodo y E_b el error ponderado del subárbol b. A veces, se usa como medida la impureza del árbol total.

Como se puede observar, esta estimación del error tiene en cuenta el tamaño relativo de las hojas sobre el árbol total. Con esta medida, se puede analizar la variación del error entre el árbol ya podado y el anterior a la poda:

$$\triangle E = E_{podado} - E_{anterior} \tag{3.6}$$

Y así, de las distintas alternativas de poda, elige reiteradamente la que minimice el valor de $\triangle E$, realizándose siempre y cuando se satisfaga $\triangle E < \delta$, donde δ es un umbral establecido por el usuario, que determina cuánta degradación del error en el conjunto de entrenamiento se puede tolerar a cambio de una posible mejor generalización en nuevos datos. La elección del hiperparámetro δ adecuado se lleva a cabo por validación cruzada.

Sin embargo, a veces esta técnica de poda puede ser muy costosa para grandes tamaños de datos, ya que puede necesitar de muchos pasos y cálculos. Por ello, también se puede reducir a un paso, utilizando la técnica de poda por *coste-complejidad*. En este caso, para cada posible árbol podado, se calcula su valor de coste-complejidad:

$$E_\alpha = E + \alpha \cdot K \tag{3.7}$$

donde α es un hiperparámetro seleccionado por el usuario, E el error definido anteriormente y K el número de hojas del árbol. Posteriormente, se selecciona el árbol podado que minimice el valor de E_{alpha}.

3.2. *K-Nearest-Neighbors*

A diferencia de los árboles para clasificación, el método de los *K-Nearest-Neighbours*, también denominado k-NN, no es útil para entender las relaciones entre las variables explicativas y la variable respuesta. Sin embargo, en algunos casos puede ser muy efectivo, ya que suele conseguir una buena generalización a nuevos datos[19].

El funcionamiento de este algoritmo es bastante simple. La idea es, dada una nueva observación $\mathbf{x}$, busca sus k ejemplos más similares , x^j con $j \in \mathbb{K} = \{1, \ldots, k\}$, en el conjunto de entrenamiento, tras esto, se le asigna a $\mathbf{x}$ la clase más votada entre estos ejemplos, denominados también como "vecinos".

3.2.1. Similitud entre observaciones

Para poder poner en práctica este algoritmo, se necesita definir alguna forma de medir la similitud entre dos vectores de variables explicativas[28]. Si todas las variables explicativas son cuantitativas, éstas se pueden representar como un punto p-dimensional en el espacio, y su similitud es medida por la distancia entre ambos puntos. De forma general, se utiliza la distancia euclídea:

$$d_E(\mathbf{x}, \mathbf{z}) = \sqrt{\sum_{i=1}^{p} (\mathbf{x}_i - \mathbf{z}_i)^2} \tag{3.8}$$

En el caso de aparecer variables ordinales, se puede realizar una codificación de las variables, de tal forma que se asignan valores numéricos en función del orden de las categorías. Así, se pueden tratar ahora como variables cuantitativas y aplicar la ecuación 3.8.

Por último, si alguna de las variables es categórica nominal, no se puede aplicar la distancia euclídea. En este caso, el grado de similitud entre las categorías de la variable tiene que ser determinado explícitamente. Si l es el número de categorías de la variable, los grados de similitud pueden venir representados en una matriz simétrica $B \in \mathbb{M}_{(l \times l)}$,

donde B_{ij} representa el grado de similitud de la categoría i-ésima con la j-ésima, se debe cumplir que $B_{jj} = 0\ \forall j \in \{1, \ldots, l\}$ y $B_{ij} \geq 0$ para $i \neq j$. La elección más común es $B_{ij} = 1$ para $i \neq j$, mientras que otras elecciones se pueden usar para enfatizar algunas diferencias más que otras.

Tratando los grados de similitud entre variables categóricas definidos anteriormente como distancias, podemos llegar a la siguiente fórmula general, combinando todo tipo de variables:

$$d(\mathbf{x}_i, \mathbf{z}_i) = \begin{cases} (\mathbf{x}_i - \mathbf{z}_i)^2 & \text{v.cuantitativa o v.ordinal} \\ B_{x_i, z_i} & \text{v.categórica} \end{cases}$$
$$d(\mathbf{x}, \mathbf{z}) = \sqrt{\sum_{i=1}^{p} d(\mathbf{x}_i, \mathbf{z}_i)} \tag{3.9}$$

3.2.2. Normalización de variables

Al aplicar la ecuación 3.9, puede darse que una variable cuantitativa o ordinal tenga más relevancia que otra sobre la distancia total, debido a que esté medida en una escala mayor. Para evitar esto, antes de calcular distancias, se puede realizar una normalización en la escala de las variables cuantitativas y ordinales codificadas en el conjunto de entrenamiento, para así pertenecer todas al mismo intervalo.

Esta normalización se puede realizar mediante la siguiente transformación sobre las variables X_j, $j \in \{1, \ldots, p\}$ de este tipo:

$$x'_{ij} = \frac{x_{ij} - MIN_j}{MAX_j - MIN_j} \tag{3.10}$$

donde $MIN_j = \text{mín}_i\, x_{ij}$ es el menor valor de la variable explicativa X_j y $MAX_j = \text{mín}_i\, x_{ij}$ el mayor. Esta técnica se denomina escalamiento *Min-Max* y ajusta los datos de la variable al intervalo $[0, 1]$. Otra forma alternativa es realizar una estandarización de los datos a una distribución normal estándar $N(0, 1)$ de la siguiente forma:

$$x'_{ij} = \frac{x_{ij} - \bar{x}_j}{\sigma_j} \tag{3.11}$$

donde $\bar{x}_j$ es la media de la variable X_j, y σ_j su desviación estándar. Esta técnica se ajusta los datos de la variable al intervalo $(-1, 1)$.

En el caso de variables categóricas, no es necesario realizar ninguna transformación, simplemente definir los grados de similitud en las matrices de tal forma que $B_{ij} \leq 1\ \forall i, j$.

3.2.3. Elección del hiperparámetro *k*

Antes de ajustar el modelo, hay que decidir el valor de k, es decir, el número de ejemplos más cercanos que se van a tener en cuenta para la elección de la clase más votada.

Como se observa en la figura 3.2, dada una observación $\mathbf{x}$, dependiendo del valor de k,

se le asocia una clase u otra, en el caso de $k = 1$, la clase mayoritaria es la representada por el cuadrado, para $k = 2$, hay un empate entre la clase representada por el cuadrado y el círculo, y para $k = 3$, la clase mayoritaria es la representada por el círculo.

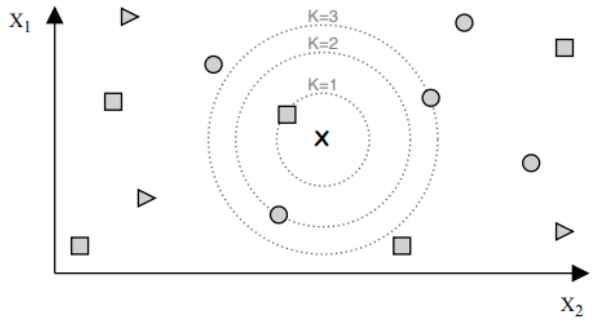


Figura 3.2: Ejemplo gráfico de clasificación k-NN[29].

Esta selección del hiperparámetro k generalmente se realiza mediante validación cruzada, por medio del proceso desarrollado en la sección 2.2. Su valor óptimo depende de varios factores, como el número de datos, el número de variables explicativas o el número de clases de la variable respuesta. Generalmente para valores de k pequeños se produce sobreajuste, a medida que k aumenta, el modelo k-NN se vuelve menos sensible a datos individuales y más influenciado por la mayoría de los vecinos, provocando una mejor generalización de los datos. Sin embargo, para valores de k altos, también existe el riesgo de que el modelo pierda detalles y producirse subajuste.

3.2.4. Mecanismos de votación

Como mecanismo de votación, normalmente se asigna a $\mathbf{x}$ la clase mayoritaria entre los k vecinos.

$$\hat{\mathbf{y}} = f(\mathbf{x}) = \arg \max_{y \in \mathbb{Y}} \sum_{j \in \mathbb{K}} \delta_{y,y^j} \tag{3.12}$$

donde $\mathbb{Y}$ representa el conjunto de clases de los k vecinos de $\mathbf{x}$, δ es una delta de *Kronecker* cuyo valor es 1 cuando las dos salidas que compara son iguales, e y^j la clase de la variable respuesta del vecino j en los datos de entrenamiento[29].

Para evitar empates entre clases, y compensar la cercanía de unos vecinos sobre otros, en ocasiones se utiliza una votación ponderada por pesos, donde a cada ejemplo de los k más cercanos se le asocia un peso w_j, que representa su grado de cercanía con la observación $\mathbf{x}$. Suponiendo que los k vecinos están ordenados acorde a sus distancias de menor a mayor, $D = \{d_1, \ldots, d_k\}$, se determinan los pesos:

$$w_j = \begin{cases} \frac{d_k - d_j}{d_k - d_1} & d_1 \neq d_k \\ 1 & d_1 = d_k \end{cases} \tag{3.13}$$

En este caso, se selecciona la clase respuesta en $\mathbf{x}$ por medio de la siguiente regla:

$$\hat{\mathbf{y}} = f(\mathbf{x}) = \arg\max_{y \in \mathbb{Y}} \sum_{j \in \mathbb{K}} w_j \delta_{y,y^j} \tag{3.14}$$

3.3. Redes Neuronales Artificiales

Una de las técnicas más populares en aprendizaje supervisado son las redes neuronales artificiales, formadas por muchas unidades simples, llamadas neuronas, que están interconectadas entre sí por enlaces con un peso asociado, formando potentes estructuras con gran capacidad de predicción sobre los datos. A diferencia de otras técnicas, las redes neuronales usan funciones no lineales en su construcción, lo que las dota de una mayor adaptabilidad a los datos. Se verá en esta sección los mecanismos elementales para la construcción de las redes neuronales *perceptrón multicapa (MLP)*, uno de los modelos más conocidos[27].

3.3.1. Estructura

La estructura de un *perceptrón multicapa* depende del número de capas y neuronas. La capa de entrada recibe los inputs, en las capas ocultas/intermedias se encuentran las neuronas ocultas, y la capa de salida devuelve los outputs. Generalmente se trabaja con perceptrones de solo una capa oculta, ya que *Kolmogorov* demostró que toda función real puede ser aproximada por este tipo de redes eligiendo los correctos parámetros[30]. Esta estructura se puede ver en la figura 3.3

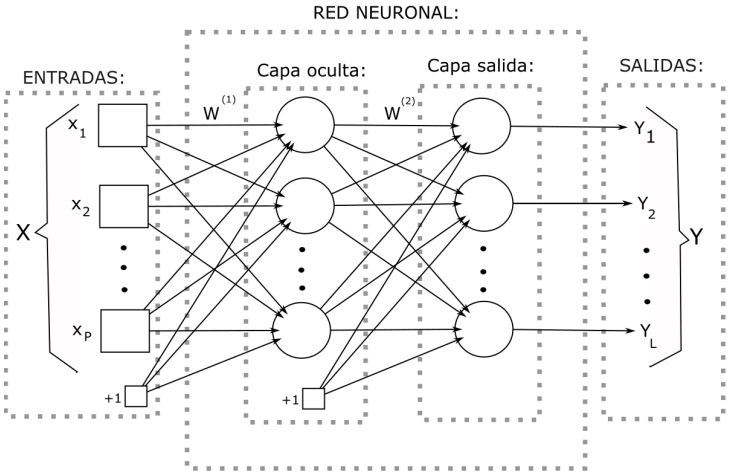


Figura 3.3: Representación de la estructura general de una red neuronal perceptrón multicapa (MLP) con una sola capa oculta.

3.3.2. *Forward Propagation*

Dada una nueva observación $\mathbf{x} = (x_1, \ldots, x_p)$, el proceso realizado dentro de la red neuronal hasta devolver sus outputs, se denomina *forward propagation* o propagación hacia delante[27]. En primer lugar, cada una de las neuronas ocultas procesa la suma ponderada de

las características/inputs de la observación, formando una red:

$$red_j^{(1)} = \sum_{i=1}^{p} x_i w_{ji}^{(1)} + w_{j0}^{(1)} = \sum_{i=0}^{p} x_i w_{ji}^{(1)} \equiv (\mathbf{w}_j^{(1)})^t \mathbf{x} \tag{3.15}$$

donde $j \in \{1, \ldots, n_H\}$ siendo n_H el número de neuronas ocultas, w_{ji} el peso asociado de la neurona oculta j con el input i y $x_0 = 1$. Ahora cada una de las neuronas ocultas produce una señal a partir de sus redes, siendo ésta la imagen de una función de transformación no lineal:

$$h_j = \sigma(red_j^{(1)}) \tag{3.16}$$

La función de transformación puede ser de diversas formas. Sin embargo, la función comúnmente usada en problemas de clasificación es la función *sigmoide* logística (ecuación 3.17), ya que cumple con todas las propiedades necesarias[31].

$$\sigma(v) = \frac{1}{1 + e^{-v}} \tag{3.17}$$

En ocasiones se usa también la función $ReLU$, la función *softmax*, u otras.

Análogamente, cada neurona de salida recibe las señales de cada una de las neuronas ocultas y las procesa:

$$red_k^{(2)} = \sum_{j=1}^{n_H} h_j w_{kj}^{(2)} + w_{k0}^{(2)} = \sum_{j=0}^{n_H} h_j w_{kj}^{(2)} \equiv (\mathbf{w}_k^{(2)})^t \mathbf{h} \tag{3.18}$$

donde $k \in \{1, \ldots, L\}$, siendo L el número de categorías de la variable respuesta, w_{kj} el peso asociado de la neurona de salida k con la neurona oculta j y $h_0 = 1$. Ahora cada neurona de salida emite su respectivo output, cada uno de ellos asocia un valor a una categoría de la variable respuesta, mediante la misma transformación anterior:

$$y_k = \sigma(red_k^{(2)}) \tag{3.19}$$

en algunos modelos se usan distintas funciones de transformación en las capas, de hecho es común utilizar la función *softmax* en la capa de salida, ya que trata los outputs como probabilidades.

Todas las ecuaciones anteriores se pueden generalizar de la siguiente forma:

$$g_k(\mathbf{x}) \equiv y_k = \sigma\Big(\sum_{j=1}^{n_H} w_{kj}^{(2)} \sigma\Big(\sum_{i=1}^{p} x_i w_{ji}^{(1)} + w_{j0}^{(1)}\Big) + w_{k0}^{(2)}\Big) \tag{3.20}$$

llegando finalmente a la clasificación de la observación $\mathbf{x}$ como:

$$\hat{\mathbf{y}} = f(\mathbf{x}) = \arg\max_k g_k(\mathbf{x}) \tag{3.21}$$

3.3.3. *Backpropagation*

El objetivo al inducir una red neuronal es establecer los pesos apropiados entre neuronas para obtener los outputs deseados para cada nueva observación. Este proceso se realiza mediante el método de *backpropagation* o propagación hacia atrás.

En primer lugar, se necesita saber cuáles son los outputs asociados a cada ejemplo en el conjunto de entrenamiento. Esto se recoge en el denominado *target vector*, $\mathbf{t}(\mathbf{x}) = (t_1(\mathbf{x}), \ldots, t_L(\mathbf{x}))$, donde generalmente $t_k(\mathbf{x})$ vale 1 si la observación $\mathbf{x}$ pertenece a la clase k, y 0 en caso contrario. Los valores de este vector pueden ser modificado en función del interés del estudio.

El aprendizaje parte de una red neuronal sin entrenar, con unos pesos iniciales apropiados, y consiste en, dado un ejemplo del conjunto de entrenamiento, modificar los pesos de la red neuronal para que los outputs se aproximen a los valores de su *target vector*. Estos parámetros son modificados en una dirección concreta, para así minimizar el valor de una función de perdida/error, que cuantifica la diferencia entre los outputs del modelo y los deseados del *target vector*.

En función del estudio propuesto, se utiliza una función de pérdida u otra. En problemas de clasificación, la más utilizada es la función de *entropía cruzada*[28]:

$$R(\mathbf{w}) \equiv -\sum_{i=1}^{N}\sum_{k=1}^{L} t_k(x_i)\log g_k(x_i) \tag{3.22}$$

donde $\mathbf{w} = (\mathbf{w}^{(1)}; \mathbf{w}^{(2)})$ y N es el número de datos en el conjunto de entrenamiento. En algunos otros modelos se usa a menudo la función del error cuadrático medio *MSE*.

La regla de aprendizaje del método *backpropagation* se basa en el descenso del gradiente:

$$\begin{aligned} \triangle\mathbf{w} &= -\eta\frac{\partial R}{\partial \mathbf{w}} \\ \triangle w_{mn} &= -\eta\frac{\partial R}{\partial w_{mn}} \end{aligned} \tag{3.23}$$

donde η es la tasa de aprendizaje, un hiperparámetro que indica el tamaño de los cambios en los pesos. De tal forma que los pesos se actualizan mediante el siguiente proceso iterativo:

$$\mathbf{w}(m+1) = \mathbf{w}(m) + \triangle\mathbf{w}(m) \tag{3.24}$$

con m es el número de iteraciones.

Considerando en primer lugar los pesos w_{kj} entre las neuronas ocultas y las de salida, aplicando la regla de la cadena de la derivada:

$$\frac{\partial R}{\partial w_{kj}} = \frac{\partial R}{\partial red_k}\frac{\partial red_k}{\partial w_{kj}} = \delta_k\frac{\partial red_k}{\partial w_{kj}} \tag{3.25}$$

donde $\delta_k \equiv -\partial R/\partial red_k$ es el grado de responsabilidad de la neurona de salida k sobre el

error total. Derivando sobre la ecuación 3.18 se llega a:

$$\frac{\partial red_k}{\partial w_{kj}} = h_j \tag{3.26}$$

Por tanto, juntando todo, se tiene:

$$\triangle w_{kj} = \eta \delta_k h_j \tag{3.27}$$

La derivación respecto de los pesos w_{ji} entre los inputs y las neuronas ocultas es más sutil, pero clave en el aprendizaje. Aplicando la regla de la cadena de nuevo:

$$\frac{\partial R}{\partial w_{ji}} = \frac{\partial R}{\partial h_j} \frac{\partial h_j}{\partial red_j} \frac{\partial red_j}{\partial w_{ji}} \tag{3.28}$$

De las ecuaciones 3.15 y 3.16 se deriva que:

$$\begin{aligned} \frac{\partial h_j}{\partial red_j} &= \sigma'(red_j) \\ \frac{\partial red_j}{\partial w_{ji}} &= x_i \end{aligned} \tag{3.29}$$

A su vez, de las ecuaciones 3.20 y 3.22 se deriva:

$$\frac{\partial R}{\partial h_j} = -\sum_{k=1}^{L} w_{kj} \delta_k \tag{3.30}$$

Juntando todo, se llega a la norma de aprendizaje para los pesos w_{ji}:

$$\triangle w_{ji} = \eta \delta_j x_i \tag{3.31}$$

donde $\delta_j \equiv \sigma'(red_j) \sum_{k=1}^{L} w_{kj} \delta_k$ es la sensibilidad de la neurona oculta j sobre el error total.

3.3.4. Entrenamiento

Una vez conocido como se actualizan los pesos mediante *backpropagation*, hay varias metodologías sobre cómo realizar el entrenamiento de la red. Los métodos de entrenamientos más conocidos son los de *batch*, estocástico y online. En este trabajo se trabaja con el método de *batch*, aunque en la práctica cualquiera puede ser válido en general.

El algoritmo de *batch* se basa en recorrer uno a uno los ejemplos del conjunto de entrenamiento, realizando la actualización de pesos correspondiente para cada ejemplo siguiendo los siguientes pasos[19]:

1. Dado un ejemplo $\mathbf{x}$, y su target vector $\mathbf{t}$, se propaga por la red y calcula sus outputs.
2. Para cada neurona de salida, se calcula su responsabilidad δ_k, y para cada neurona oculta, su respectiva δ_j.

3. Se actualizan los pesos:

$$\begin{aligned} w_{ji} &= w_{ji} + \eta\delta_j x_i \\ w_{kj} &= w_{kj} + \eta\delta_k h_j \end{aligned} \tag{3.32}$$

Cuando se han recorrido todos los ejemplos del conjunto de entrenamiento, se dice que se ha completado una etapa. El número de etapas se puede usar como criterio de parada, e introducírselo al modelo mediante el hiperparámetro m. En ocasiones no se recorren todos los ejemplos del entrenamiento en cada etapa, si no un grupo de ellos que va variando. Esto es conocido como algoritmo de *mini-batch*.

3.3.5. Técnicas para mejorar *backpropagation*

Se presentan a continuación una serie de técnicas y aspectos a tener en cuenta antes de la inducción de una red neuronal:

- Tratamiento previo variables explicativas: En el caso de variables continuas o categóricas ordinales, es necesario realizar una normalización de sus datos, de forma análoga a cómo se realizó en la sección 3.2.2. En el caso de variables categóricas nominales, se debe llevar a cabo una codificación *one-hot*[32], cada categoría se representa como una variable binaria, donde se asigna el valor 1 si la observación pertenece a esa categoría y 0 en caso contrario. Se crean tantas variables nuevas en el conjunto de datos como categorías.

- Mínimos Locales: Los pesos se cambian de tal manera que garantiza el descenso del gradiente por la pendiente más pronunciada. Pero una vez que se alcanza un mínimo local, puede que la red se quede allí atrapada, aunque el objetivo final sea aproximarse al mínimo global[19]. Para corregir esto, se puede controlar la tendencia de la función de perdida de una etapa a otra, de tal forma que se entre a una nueva etapa si y solo si $\nabla R(w) < \theta$, siendo θ un hiperparámetro seleccionado por el usuario. O por ejemplo, parar el entrenamiento si la función de perdida no mejora en varias etapas consecutivas.

- Inicialización de los pesos: Los pesos se inicializan con valores muy pequeños, generalmente elegidos aleatoriamente de una distribución uniforme centrada en cero y con una varianza específica, que depende del número de unidades de entrada y salida de cada capa.

- Tasa de aprendizaje: La tasa de aprendizaje puede ser constante, tal que $\eta \in (0, 1)$, o se puede ajustar en función del tiempo de entrenamiento:

$$\eta(t) = \eta(0)e^{-\beta t} \tag{3.33}$$

  Esto es debido a que normalmente se desea que al principio aprenda más rápido, y así superar los mínimos locales. En redes más sofisticadas se utiliza el algoritmo de

Adam como regla de aprendizaje en *backpropagation*, que combina las ventajas del descenso del gradiente con adaptaciones que ajustan la tasa de aprendizaje para cada parámetro de la red.

- Número de neuronas ocultas: El número de neuronas ocultas es crucial en el poder expresivo de la red, usar muchas neuronas provoca que la red se ajuste demasiado a los datos, y pocas, que la red se quede atrapada en un mínimo local. Dependiendo de la naturaleza de los datos, más o menos neuronas serán necesarias para conseguir un buen rendimiento. Para determinar el número óptimo de neuronas es necesario realizar validación cruzada.

- Regularización: Un método de simplificar la red y evitar el sobreajuste es imponer que los pesos sean pequeños. En este enfoque se busca que los pesos vayan decayendo durante el entrenamiento, y se lleva a cabo modificando la función de perdida para que penalice los pesos más grandes:

$$R_{ef}(\mathbf{w}) = R(\mathbf{w}) + \frac{\alpha}{2}\mathbf{w}^t\mathbf{w} \tag{3.34}$$

  aunque no esté demostrado que este método consiga mejores rendimientos, sí se ha visto que en muchos casos ayuda a una mejor generalización.

3.4. Regresión Logística Ordinal

Los modelos de regresión logística son capaces de predecir la probabilidad de sucesos en función de las variables explicativas, y pueden ser usados para clasificación a partir de las probabilidades obtenidas para cada categoría, como se aplicará en este caso.

La regresión logística es uno de los métodos estadísticos usados para analizar las relaciones entre variables explicativas y variable respuesta cuando la variable respuesta es categórica. Si la variable respuesta tiene 3 o más categorías que siguen un orden, entonces el modelo es denominado Regresión Logística Ordinal, y debido a la naturaleza de los datos del estudio que se va a realizar, será éste el modelo descrito[33].

3.4.1. Modelo

Sea L el número de categorías de la variable respuesta Y. La probabilidad acumulada en Y es la probabilidad de que Y esté por debajo de un valor, dada una observación x. En este caso estos valores se refieren a la codificación de las categorías según su orden, desde 1 hasta L. Se denota como:

$$P(Y \leq j \mid x) = \pi_1(x) + \cdots + \pi_j(x), \quad j \in \{1, \ldots, L\} \tag{3.35}$$

donde $\pi_j(x)$ representa la probabilidad de que x pertenezca a la clase j, por lo que $P(Y \leq 1|x) \leq P(Y \leq 2|x) \leq \cdots \leq P(Y \leq L|x) = \sum_{j=1}^{L} \pi_j(x) = 1$. Como $P(Y \leq L|x) = 1$ siempre, es redundante en el modelo, asique basta considerar las L-1 primeras categorías.

Estas probabilidades acumuladas se modelan mediante su transformación por medio de la función *logit*, llegando a los denominados *logits acumulados* L_j:

$$L_j = logit[P(Y \leq j \mid x)] = log\left(\frac{P(Y \leq j)}{1 - P(Y \leq j)}\right) = log\left(\frac{\pi_1 + \cdots + \pi_j}{\pi_{j+1} + \cdots + \pi_L}\right), \quad j \in \{1, \ldots, L-1\} \tag{3.36}$$

Estos *logits* L_j se pueden tratar de diversas maneras. Suponiendo que las probabilidades acumuladas de las categorías ordenadas siguen una distribución logística, es decir, la razón entre logits acumulados de 2 categorías consecutivas es constante, entonces cada logit acumulado L_j puede ser representado como un modelo de regresión logística binario, donde las categorías de la 1 a la j forman una sola categoría, y desde la $j+1$ hasta la L otra[34]:

$$L_j = \beta_{0j} + \beta_1 x_1 + \cdots + \beta_p x_p, \quad j \in \{1, \ldots, L-1\} \tag{3.37}$$

Se observa que $\beta_1, \ldots, \beta_p$ se mantienen constantes para todo índice j. Además, este modelo asume que el efecto de x es idéntico sobre los L-1 logits acumulados. Por tanto, de la ecuación 3.37 podemos despejar la probabilidad de que una observación x pertenezca a una categoría j:

$$\pi_j(x) = \frac{e^{\beta_{0j} + \sum_{k=1}^{p} \beta_k x_k}}{1 + e^{\beta_{0j} + \sum_{k=1}^{p} \beta_k x_k}} - \frac{e^{\beta_{0j-1} + \sum_{k=1}^{p} \beta_k x_k}}{1 + e^{\beta_{0j-1} + \sum_{k=1}^{p} \beta_k x_k}}, \quad j \in \{2, \ldots, L-1\} \tag{3.38}$$

Una vez establecidas las probabilidades de las distintas categorías en la variable respuesta a partir de una observación $\hat{\pi}(x) = (\pi_1(x), \ldots, \pi_L(x))$, la observación x se clasifica a la categoría con mayor probabilidad:

$$\hat{y} = f(x) = \arg\max_j \pi_j(x) \tag{3.39}$$

3.4.2. Estimación de parámetros

La clave en el modelo de regresión logística ordinal es encontrar los parámetros $\hat{\beta} = (\hat{\beta}_{01}, \ldots, \hat{\beta}_{0L-1}, \hat{\beta}_1, \ldots, \hat{\beta}_p)$ apropiados. Una forma de estimarlos es mediante la función de máxima verosimilitud(MLE), que mide qué tan probable es que los datos observados sean generados por el modelo propuesto. Se buscan los valores $\hat{\beta}$ en los parámetros que maximicen la función *log-verosimilitud*:

$$L(\beta) = \sum_{i=1}^{N} \sum_{j=1}^{L} y_{ji} ln(\pi_j(x_i)) \tag{3.40}$$

donde y_{ij} representa si el ejemplo i pertenece a la clase j y N el número de ejemplos en el conjunto de entrenamiento.

Como estas ecuaciones no se pueden resolver explícitamente derivando e igualando a 0, la búsqueda de los parámetros óptimos se realiza mediante el algoritmo numérico de *Newton-Raphson*, véase[35]. Así, los parámetros se van actualizando en cada iteración como

sigue[36]:

$$\beta(m+1) = \beta(m) - \left(\frac{\partial^2 L(\beta)}{\partial \beta^2}\right)^{-1} \frac{\partial L(\beta)}{\partial \beta} \tag{3.41}$$

donde $H = \frac{\partial^2 L(\beta)}{\partial \beta^2}$ representa la matriz hessiana de L en β (debe ser invertible), y $\nabla L(\beta) = \frac{\partial L(\beta)}{\partial \beta}$ su gradiente, ambos evaluados en el valor $\beta(m)$ de la anterior iteración. En el primer paso se inicializan los parámetros de forma aleatoria y se van iterando hasta conseguir una buena estimación de estos mediante la convergencia del algoritmo numérico.

3.4.3. Significancia estadística del modelo

Tras la construcción del modelo, hay que examinar la significancia estadística de los parámetros finales estimados $\hat{\beta}$ para determinar si las variables explicativas tienen un efecto significativo en la variable respuesta. Hay algunos tests para medir esta significancia estadística:

- Test de Razón de Verosimilitudes: Se compara un modelo completo con otro formado por solo constantes (sin variables explicativas). La hipótesis nula es que los parámetros restringidos son igual a cero, mientras que la hipótesis alternativa es que al menos uno de los parámetros restringidos es distinto de cero[37]:

$$\begin{aligned} H_0 &: \hat{\beta}_1 = \hat{\beta}_2 = \cdots = \hat{\beta}_p = 0 \\ H_1 &: \exists k \quad t.q \quad \hat{\beta}_k \neq 0 \quad k \in \{1, \ldots, p\} \end{aligned} \tag{3.42}$$

  El estadístico utilizado es:

$$G = -2ln\left[\frac{l_0}{l_1}\right] \tag{3.43}$$

  donde l_0 es la verosimilitud del modelo restringido, y l_1 la verosimilitud del modelo completo. Bajo la hipótesis nula, G se distribuye asintóticamente según una distribución chi-cuadrado χ^2_p. En caso de rechazar la hipótesis nula, cuando el p-$valor = Pr(G < \chi^2_p) < \alpha$, con α el grado de significación elegido, se concluye que al menos una de las variables explicativas afectan a la variable respuesta.

- Test de Wald: Tras verificar que el modelo es significante en el Test de Razón de Verosimilitudes, se aplica el Test de Wald, donde la significancia de cada variable explicativa es evaluada por separado para así comprobar cuáles tienen efecto en la variable respuesta. Es decir, se examina la significancia individual de los coeficientes en el modelo estimado:

$$\begin{aligned} H_0 &: \hat{\beta}_k = 0 \\ H_1 &: \hat{\beta}_k \neq 0 \end{aligned} \tag{3.44}$$

En este caso se utiliza el estadístico:

$$W_k = \frac{\hat{\beta}_k}{\hat{SE}(\hat{\beta}_k)} \quad k \in \{1, \ldots, p\} \tag{3.45}$$

donde $\hat{SE}(\hat{\beta}_k)$ es la desviación estándar del parámetro estimado. Se calcula el correspondiente $p - valor = Pr(|z| > W_k)$, donde z sigue una distribución normal estándar. Si el p-valor$< \alpha$ se rechaza la hipótesis nula, lo que significa que la variable explicativa k es significante en el modelo estimado. Se elimina la variable explicativa no significativa de mayor p-valor y se repite el proceso, ajustando de nuevo el modelo con las variables restantes. Así hasta llegar a un modelo con todas las variables significativas. Esta técnica se conoce como el método de *backward*.

3.4.4. Multicolinealidad

La multicolinealidad se refiere a la presencia de una alta correlación entre dos o más variables explicativas y puede tener efectos negativos en el modelo como la inestabilidad en los parámetros o una mala interpretación de las variables explicativas en el modelo final. Para evitar la correlación, es conveniente realizar un estudio previo sobre las variables explicativas. Una forma es calculando la matriz de correlación entre las variables y si se encuentra una alta correlación entre dos variables, se considera eliminar una de ellas o combinarlas para crear una nueva variable que capture su esencia.

3.4.5. Bondad de ajuste

Una vez ajustado el modelo/modelos finales, se debe evaluar la adaptación del modelo a los datos observados, es decir, cómo de bien se aproximan los outputs del modelo a las observaciones en los datos de entrenamiento. Una de las pruebas utilizadas es el *Hosmer-Lemeshow*[38] test. Esta prueba se basa en la comparación de las frecuencias observadas y esperadas en diferentes grupos creados a partir de los puntajes ordinales asignados a las observaciones.
Se asigna un puntaje ordinal a cada observación utilizando la fórmula:

$$s_i = \hat{\pi} i1 + 2\hat{\pi} i2 + \ldots + L\hat{\pi}_{iL} \quad i \in \{1, \ldots, n\} \tag{3.46}$$

Las observaciones se ordenan de acuerdo con los puntajes ordinales y se dividen en G grupos. El número de grupos puede variar, pero comúnmente se utiliza $G = 10$. Se calculan las frecuencias observadas O_{gj} y las frecuencias esperadas E_{gj} para cada grupo k y cada categoría j. El estadístico de prueba del Hosmer-Lemeshow test es el estadístico χ^2 de Pearson C_g

$$C_G = \sum_{g=1}^{G} \sum_{j=1}^{L} \frac{(O_{gj} - E_{gj})^2}{E_{gj}}, \tag{3.47}$$

Bajo la suposición de un modelo de odds proporcionales correctamente ajustado, C_g se aproxima bien a una distribución χ^2 con $(G-2)(L-1) + (L-2)$ grados de libertad. Si C_G

es menor que el valor crítico, no hay evidencia suficiente para rechazar la hipótesis nula, y el modelo se ajusta bien a los datos observados.

3.4.6. Interpretación del modelo

Al igual que en los árboles para clasificación, los modelos de regresión logística ordinal se pueden usar para fines explicativos. Para ello, se necesita medir el impacto de cada variable explicativa sobre la variable respuesta[39]. Una forma de medir esta relevancia son los denominados *odds ratio*:

$$\hat{OR}_k = e^{\hat{\beta}_k}, \quad k \in \{1, \dots, p\} \tag{3.48}$$

Cada odd ratio mide cómo varía la probabilidad relativa de pertenecer a una categoría superior en comparación con la categoría de referencia cuando una variable explicativa X_k aumenta en una unidad, manteniendo todas las demás variables constantes. Así se puede interpretar su efecto en las probabilidades relativas de pertenecer a diferentes categorías de la variable respuesta. Si el odd ratio es mayor que 1, indica que un aumento en el valor de la variable explicativa está asociado con un aumento en la probabilidad de pertenecer a una clase superior en comparación con la clase de referencia. Y si es menor que 1, al contrario.

Si queremos evaluar el odd ratio entre 2 observaciones concretas a y b, se puede generalizar mediante la siguiente fórmula:

$$\hat{OR}(a, b) = \frac{Pr(Y = j \mid x = a)/Pr(Y = 1 \mid x = a)}{Pr(Y = j \mid x = b)/Pr(Y = 1 \mid x = b)} \tag{3.49}$$

3.4.7. Tratamiento previo variables explicativas

En el caso de variables cuantitativas, si se quiere comparar la importancia de las variables en la predicción mediante los odds ratio, puede ser útil normalizarlas antes de incluirlas en el modelo como se hizo en la sección 3.2.2, aunque no es estrictamente necesario en este caso. Por otro lado, dada una variable categórica, es necesario convertirla en variables *dummy* o variables binarias, análogamente a cómo se realizó en las redes neuronales. Para una variable X_j de l categorías, se crean l-1 variables *dummy*, $X_{j1}, \dots, X_{jl-1}$, para así evitar la multicolinealidad. Cada X_{ji} representa la presencia, con un 1, o no, con un 0, de la categoría i en la variable j.

Capítulo 4

Estudio Computacional

En este último capítulo se realiza el estudio computacional introducido en la sección 2.4, la predicción del rango de precios de un móvil en función de sus características. Como la variable respuesta es categórica ordinal, se aplicarán las técnicas de clasificación desarrolladas durante el trabajo, y se ajustarán y compararán distintos modelos. Durante el capítulo, se presentará el conjunto de datos, se indicará la configuración y funcionamiento del estudio, y posteriormente, sus respectivos resultados y conclusiones. El estudio se llevará a cabo mediante el lenguaje *Python*.

4.1. Conjunto de datos

El conjunto de datos ha sido obtenido de *Kaggle*, una plataforma online que ofrece una amplia gama de recursos relacionados con la ciencia de datos y el aprendizaje automático. El formato de los datos es *.csv*. El conjunto consta de 21 columnas: 20 variables explicativas, que representan las características del móvil, y 1 variable respuesta, que indica el rango de precio del móvil. Se han recogido los datos de 2000 móviles.
Las características/variables explicativas recogidas son las siguientes:

- **battery_power**: Potencia de la batería del móvil. Variable cuantitativa.
- **blue**: Presencia de Bluetooth. Variable binaria.
- **clock_speed**: Velocidad de la CPU. Variable cuantitativa.
- **dual_sim**: Presencia de dual SIM. Variable binaria.
- **fc**: Número píxeles cámara frontal. Variable cuantitativa.
- **four_g**: Presencia de 4G. Variable binaria.
- **int_memory**: Capacidad de la memoria interna. Variable cuantitativa.
- **m_deep**: Grosor del dispositivo. Variable cuantitativa.
- **mobile_wt**: Peso del dispositivo. Variable cuantitativa.
- **n_cores**: Número de núcleos en el procesador. Variable cuantitativa.
- **pc**: Número píxeles cámara trasera. Variable cuantitativa.
- **px_height**: Resolución pantalla a lo alto. Variable cuantitativa.
- **px_width**: Resolución de pantalla a lo ancho. Variable cuantitativa.
- **ram**: Capacidad RAM. Variable cuantitativa.
- **sc_h**: Altura pantalla. Variable cuantitativa.
- **sc_w**: Anchura pantalla. Variable cuantitativa.

- **talk_time**: Tiempo máximo de conversación. Variable cuantitativa.
- **three_g**: Presencia de 3G. Variable binaria.
- **touch_screen**: Presencia de pantalla táctil. Variable binaria.
- **wifi**: Presencia de conectividad Wifi. Variable binaria.

Y la variable respuesta:

- **price_range**: Rango de precio del móvil. Variable categórica ordinal (0-Precio bajo, 1-Precio medio, 2-alto, 3-Precio muy alto).

Veamos una representación de los 8 primeros ejemplos del conjunto:

```
   battery_power  blue  clock_speed  ...  touch_screen  wifi  price_range
0            842     0          2.2  ...             0     1            1
1           1021     1          0.5  ...             1     0            2
2            563     1          0.5  ...             1     0            2
3            615     1          2.5  ...             0     0            2
4           1821     1          1.2  ...             1     0            1
5           1859     0          0.5  ...             0     0            1
6           1821     0          1.7  ...             0     1            3
7           1954     0          0.5  ...             1     1            0
```

Un breve resumen estadístico de las variables:

```
    [8 rows x 21 columns]
       battery_power       blue  ...         wifi  price_range
count    2000.000000  2000.0000  ...  2000.000000  2000.000000
mean     1238.518500     0.4950  ...     0.507000     1.500000
std       439.418206     0.5001  ...     0.500076     1.118314
min       501.000000     0.0000  ...     0.000000     0.000000
25%       851.750000     0.0000  ...     0.000000     0.750000
50%      1226.000000     0.0000  ...     1.000000     1.500000
75%      1615.250000     1.0000  ...     1.000000     2.250000
max      1998.000000     1.0000  ...     1.000000     3.000000
```

Estudio presencia de valores missing, tipo de cada variable y número de valores únicos por cada una:

```
 #   Column         Non-Null Count  Dtype    Unique Conts
---  ------         --------------  -----    -------------
 0   battery_power  2000 non-null   int64            1094
 1   blue           2000 non-null   int64               2
 2   clock_speed    2000 non-null   float64            26
 3   dual_sim       2000 non-null   int64               2
 4   fc             2000 non-null   int64              20
 5   four_g         2000 non-null   int64               2
```

```
 6   int_memory     2000 non-null   int64              63
 7   m_dep          2000 non-null   float64            10
 8   mobile_wt      2000 non-null   int64             121
 9   n_cores        2000 non-null   int64               8
 10  pc             2000 non-null   int64              21
 11  px_height      2000 non-null   int64             137
 12  px_width       2000 non-null   int64             109
 13  ram            2000 non-null   int64             562
 14  sc_h           2000 non-null   int64              15
 15  sc_w           2000 non-null   int64              19
 16  talk_time      2000 non-null   int64              19
 17  three_g        2000 non-null   int64               2
 18  touch_screen   2000 non-null   int64               2
 19  wifi           2000 non-null   int64               2
 20  price_range    2000 non-null   int64               4
dtypes: float64(2), int64(19)
```

Observamos que no hay ningún dato *missing* en los datos. A priori, aunque hay variables cuantitativas con pocos valores únicos, las trataremos todas como continuas en el estudio.

4.2. Configuración Experimental

En primer lugar separaremos el conjunto de datos en conjuntos de entrenamiento y test mediante las separaciones del método *5x2 Cross Validation* (sección 2.2.2). Para cada conjunto de entrenamiento, se evalúan los modelos con distintos hiperparámetros en cada algoritmo. El mejor modelo será el que consiga mejores evaluaciones medias de los *5-Fold Cross-Validation* aplicado sobre los conjuntos de entrenamiento. Una vez conseguido el mejor modelo para cada algoritmo, compararemos sus evaluaciones en los tests. Los algoritmos utilizados y sus hiperparámetros serán los siguientes:

1. Árboles para clasificación con parada: Compararemos distintos valores en β (reducción mínima de la impureza) y m (ejemplos mínimos en la partición de un nodo).

2. Árboles para clasificación con poda: Compararemos los distintos valores de α (parámetro de poda coste-complejidad).

   En ambos arboles utilizaremos la entropía de la información para medir la impureza. Usaremos la función *DecisionTreeClassifier()* de la biblioteca *sklearn* para ajustar los modelos.

3. k-Nearest-Neighbors: Compararemos el número de vecinos, el mecanismo de voto (uniforme o ponderado), y la normalización previa o no de las variables. Se implementa mediante la función *KNeighborsClassifier()* de *sklearn*.

4. Redes Neuronales MLP: Realizaremos una estandarización previa de las variables en los conjuntos de entrenamiento mediante la función *StandardScaler()*. Usaremos la

función de transformación sigmoide y el algoritmo de entrenamiento de *mini-batch*. Compararemos las distintas formas de la tasa de aprendizaje, el número de etapas, el número de neuronas y los valores del parámetro α de regularización. Se implementa mediante la función *MLPClassifier()* de *sklearn*.

5. Regresión logística ordinal: En este caso no hay hiperparámetros, por tanto no tiene sentido realizar ninguna comparación. Haremos un análisis previo de la multicolinealidad en las variables. Luego, para cada entrenamiento ajustaremos un modelo significativo, mediante el método de *backward*. Evaluaremos cada modelo en test, y el que mejores resultados consigan, lo consideraremos como el modelo final, al que analizaremos su bondad de ajuste y odds ratio. Este método se implementa con *OrderedModel()* de la librería *statsmodel*.

Para obtener las evaluaciones mediante *K-Fold Cross-Validation* de distintos hiperparámetros sobre los conjuntos de entrenamiento, usaremos el método *GridSearchCV()* de la librería *sklearn*.

Como métricas de evaluación utilizaremos Exactitud, que es la métrica más común en clasificación, Macro-F1, ya que hace un balance entre Recall-Precisión y considera todas las clases como igual de importantes, y MDP, recomendable para tratar variables ordinales.

4.3. Resultados y Conclusiones

Elección modelos

-Arboles con parada: Se consigue las mejores evaluaciones medias con $m = 9$ y $\beta = 0{,}001$ (figura 4.1).
-Arboles con poda: Se escoge el valor de $\alpha = 0{,}007$ (figura 4.2).
-K-Nearest-Neighbors: Tras varias pruebas, hemos obtenido resultados notablemente mejores cuando no se normalizan las variables cuantitativas. Esto suele ser debido a que las variables con mayor escala son buenas predictoras de la variable respuesta, ya que tienen una gran influencia sobre la distancia con los vecinos. De hecho, basta con las características 'battery_power','int_memory','pc','mobile_wt','px_height','px_width'y 'ram' para conseguir buenos resultados (figura 4.3). Elegiremos este modelo, usando votación ponderada y 20 vecinos. Además, al considerar el modelo con solo estas variables, ya sí hemos conseguido resultados más cercanos al normalizar, lo que refuerza las hipótesis anteriores.
-Redes Neuronales: Tras probar varios modelos, hemos conseguido mejores evaluaciones cuando la tasa de aprendizaje era constante ($\eta = 0{,}001$) y con un valor en el parámetro de regularización de $\alpha = 0{,}0001$. Respecto al número de etapas, a medida que hemos ido aumentado su valor, hemos conseguido una mejor evaluación, hasta llegar a 1000, donde ya no mejora tanto. Respecto a las neuronas, usaremos 400 en el modelo final(figura 4.4).
-Regresión logística: En primer lugar, se eliminan las variables 'three_g', 'sc_h' y 'fc' para evitar la multicolinealidad, ya que guardan una alta correlación con 'four_g', 'sc_w' y 'pc' respectivamente. Aunque hubiésemos preferido usar variables normalizadas, para así medir su grado de relevancia sobre la variable respuesta en la misma escala, se han conseguido

resultados mucho mejores sin normalizarlas. El modelo que más se repite y que mejores evaluaciones consigue es el formado por las variables 'battery_power', 'px_height', 'px_width', 'ram' y 'mobile_wt' sin normalizar, por tanto lo consideraremos como modelo final.

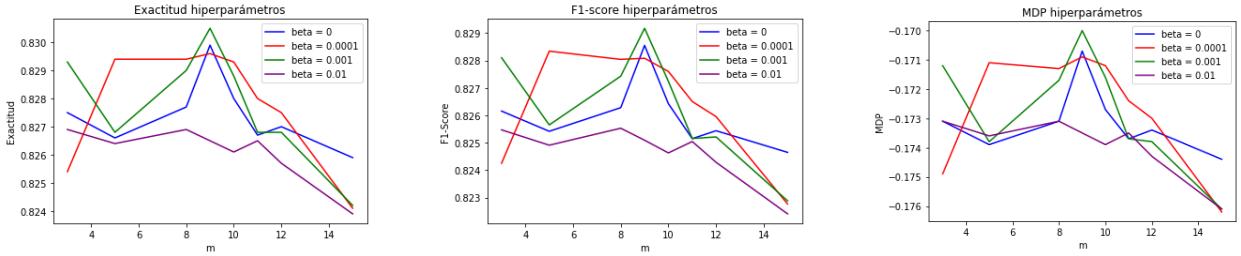


Figura 4.1: Evaluaciones medias en función de los hiperparámetros m y β en arboles con parada.

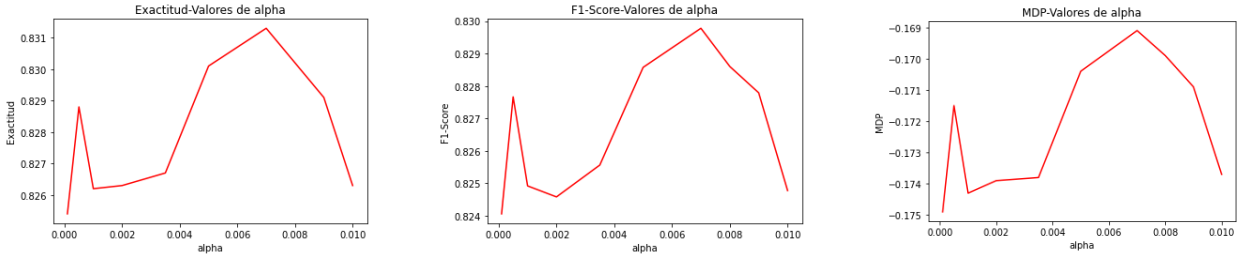


Figura 4.2: Evaluaciones medias en función del valor de α en arboles con poda.

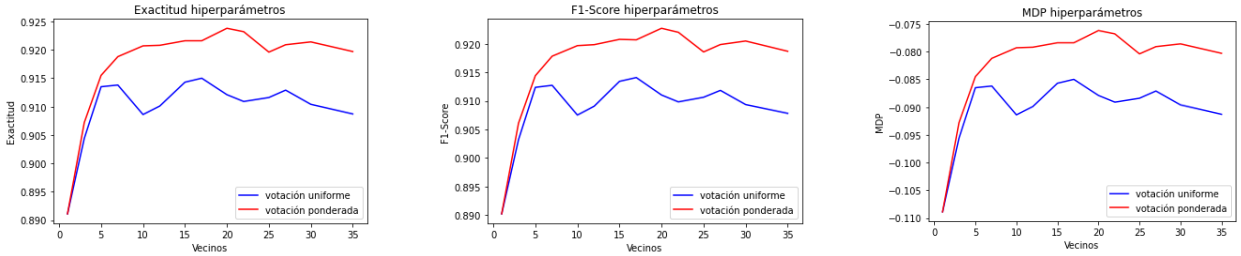


Figura 4.3: Evaluaciones medias en función de hiperparámetros en k-NN sin normalización.

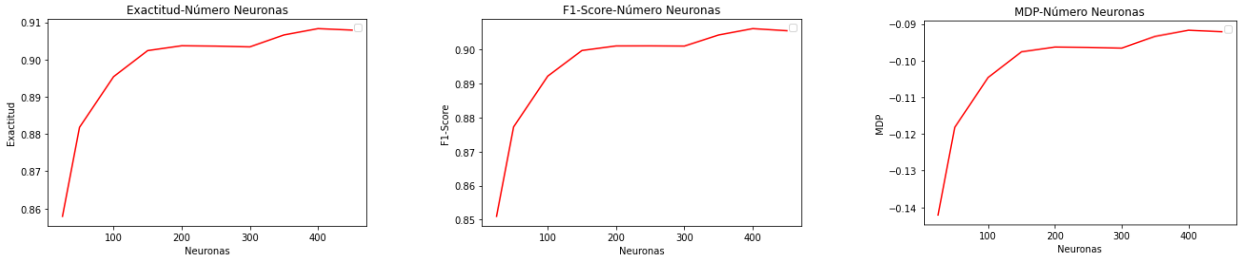


Figura 4.4: Evaluaciones medias en función de hiperparámetros redes neuronales.

Comparación mejores modelos

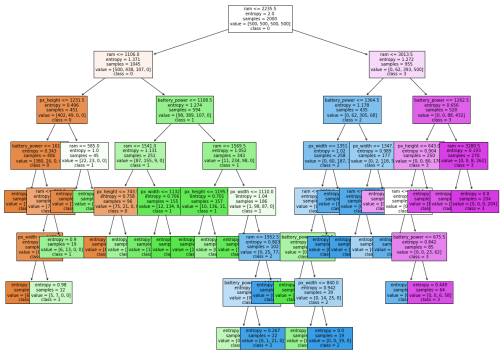
Tras evaluar los modelos finales de cada algoritmo en los tests, hemos obtenido los siguientes resultados de evaluación:

Modelos Finales	Exactitud	Macro F1-Score	MDP
Árbol parada	0.83	0.83	0.16
Árbol poda	0.84	0.84	0.15
k-NN	0.92	0.92	0.07
Red Neuronal	0.93	0.93	0.06
Reg.Log.Ordinal	0.98	0.98	0.02

Cuadro 4.1: Evaluaciones *5x2CV* modelos finales

se puede observar que los valores en Exactitud y Macro-F1 son similares (con redondeo), lo que manifiesta que las clases están balanceadas.

Respecto a los árboles, tienen una gran capacidad explicativa, las características se ordenan en función de su importancia y es el algoritmo menos costoso computacionalmente de los estudiados. En este caso, el árbol con poda consigue unas ligeras mejores evaluaciones usando menos variables y con una menor profundidad que el de parada.



```
ram: 0.781234274768333
battery_power: 0.10878301796643593
px_width: 0.06597903968486062
px_height: 0.044003667580370 49
```

Figura 4.5: Árbol podado final con orden de importancia de sus características.

Como se observa en la figura 4.5, el árbol podado final solo necesita 4 características para predecir la variable respuesta, con una gran importancia de la variable 'ram', qué es la que más información aporta en la predicción. Sin embargo, su capacidad de generalización es la peor de entre los algoritmos, ésta podría mejorar induciendo otro tipo de técnicas que combinan varios árboles para mejorar la evaluación, como los *Random Forest*[40].

En cuanto al modelo *k-NN*, consigue una muy buena generalización a nuevos datos. Además, al haber conseguido los mejores resultados con las variables de mayor escala sin normalizar, nos hace indicar que éstas son las más determinantes en la predicción. Sin embargo, no llega al detalle explicativo de los árboles.

Por otro lado, la red neuronal, aunque tiene un alto grado de acierto en sus predicciones, hemos necesitado 1000 etapas y 400 neuronas para lograrlo, lo que ha provocado un alto coste computacional. Además, no logra ninguna capacidad explicativa sobre la clasificación en los datos, y en este caso en concreto, podría ser útil para crear el mejor móvil posible con el menor precio.

En cualquier caso, cabe destacar que a pesar de que estos 3 algoritmos no son específicos para variables respuesta ordinales, consiguen un MDP acorde con su exactitud y macro-F1, lo que indica que en caso de fallar en la predicción, tienden a hacerlo a una de sus clases cercanas.

Finalmente, el modelo de regresión logística ordinal ha conseguido las mejores evaluaciones en todas las métricas con considerable diferencia. Además, se consigue una muy buena bondad de ajuste, con un valor de $C_G = 0{,}22 < 38{,}88 = \chi^2_{(0,95,26)}$, y se llega a un modelo significativo con solo 5 variables explicativas, que son las que determinan la predicción:

Coeficientes		Odds ratio	
battery_power	0.032393	battery_power	1.032924e+00
mobile_wt	-0.072205	mobile_wt	9.303403e-01
px_height	0.019161	px_height	1.019346e+00
px_width	0.018766	px_width	1.018943e+00
ram	0.052302	ram	1.053694e+00
0/1	127.390004	0/1	2.112398e+55
1/2	3.889689	1/2	4.889565e+01
2/3	3.884086	2/3	4.862250e+01

donde 0/1, 1/2 y 2/3 representan los interceptos β_{01}, β_{02} y β_{03}. En este caso, los odds ratio nos ofrecen menos información, ya que las variables no están normalizadas y la escala influye en los resultados. Sin embargo, observamos que el odd ratio de 'mobile_wt' es cercano a 0, lo que significa que el aumento de peso provoca una reducción en el precio. Por otro lado, en las demás variables son ligeramente superiores a 1, lo que indica que el precio crece ligeramente al aumentar en estas características, en relación a su escala.

Definitivamente la regresión logística ordinal es el mejor modelo de los planteados, ya que combina una muy buena capacidad explicativa de la relación entre características y respuesta con la mejor capacidad de generalización. Además, su construcción es simple, en cuánto a no necesitar de la selección de hiperparámetros. Esto muestra lo importante que puede ser a veces usar un modelo que se ajuste bien a las características del problema.

Capítulo 5

Conclusiones

En conclusión, podemos afirmar que mediante este TFG se han cumplido los objetivos propuestos, ya que hemos logrado transmitir la importancia del *machine learning* en la sociedad actual, recorriendo sus diversas áreas y aplicaciones, hemos conocido los fundamentos del aprendizaje supervisado y entendido el funcionamiento de algunas de las técnicas más utilizadas para clasificación. Así, posteriormente hemos podido aplicar estas técnicas sobre un estudio real, en este caso la predicción del rango de precio de un móvil a partir de sus características. Además, durante el estudio hemos aprendido la importancia de adaptar el proceso de aprendizaje al problema y conjunto de datos que se van a a abordar en concreto, para así poder obtener los mejores resultados de evaluación posibles.

En el estudio realizado, hemos podido comparar la actuación de las técnicas propuestas en el marco teórico, ajustando distintos modelos para cada una de ellas y seleccionando los mejores. Tras un profundo análisis, se ha determinado que el mejor rendimiento se consigue con la técnica de regresión logística ordinal, que ha logrado unos resultados fascinantes, con una capacidad explicativa bastante notable. También, se puede observar que la mayoría de técnicas han coincidido en que las características más determinantes son la capacidad RAM, la potencia de la batería, el peso del dispositivo y la resolución de la pantalla. Estos resultados fortalecen aún más la figura del aprendizaje automático, ya que sugieren que es posible predecir el rango de precio de un móvil con casi total precisión, siempre y cuando se realice un estudio analítico previo sobre el conjunto de datos y se utilicen las técnicas apropiadas.

A pesar de haber estudiado una gran cantidad de nociones y conceptos, solo es una pequeña parte de todo lo que engloba el aprendizaje automático, y quedan infinidad de ellos por aprender. Ha sido una experiencia muy enriquecedora adentrarme en este mundo, que está en constante evolución y parece que todavía estamos ante el principio de su desarrollo. Por ello, me gustaría seguir investigando sobre diversos puntos dentro de él. Uno de ellos es profundizar sobre el procesamiento previo y limpieza de los datos, ya que en este caso, los datos obtenidos eran bastante limpios, y las variables, al ser cuantitativas o binarias, no nos han generado demasiados inconvenientes, algo que no suele suceder en la mayoría de problemas. Por otro lado, sería interesante estudiar otras técnicas de clasificación que no hemos podido abordar en este trabajo, como los *Random Forests*, los métodos de *Support Vector Machine* y los algoritmos de *Boosting*, que combinan distintas técnicas para mejorar la capacidad de generalización. Por último, otra meta es indagar en otras áreas y problemas dentro del aprendizaje automático: tratar conjuntos de datos de distinta naturaleza, investigar la tarea de regresión y sus diversos algoritmos, e incluso, profundizar más en el aprendizaje no supervisado y aprendizaje por refuerzo.

Bibliografía

[1] CampusMVP. *Los 4 mejores lenguajes de programación para inteligencia artificial y machine learning*. https://www.campusmvp.es/recursos/post/los-4-mejores-lenguajes-de-programacion-para-inteligencia-artificial-machine-learning.aspx. Accedido el 13 de mayo de 2023. 2021.

[2] D. Hinestroza Ramírez. "El Machine Learning a través de los tiempos, y los aportes a la humanidad". En: *Revista Científica Agustín Codazzi* 16.1 (2018), págs. 8-10.

[3] Luis Barrera Arrestegui. "Fundamentos históricos y filosóficos de la inteligencia artificial". En: *UCV-HACER. Revista de Investigación y Cultura* 1.1 (2012), págs. 87-92.

[4] C. G. Sandoval Vásquez y C. V. Marín Escobar. "Artificial neural network simulator-ANNS V1.1 Diseño de software de simulación de redes neuronales artificiales (Parte II)". En: *Revista Técnica de la Facultad de Ingeniería Universidad del Zulia* 31.3 (2008), págs. 225-232.

[5] ROBERTO Cordeschi. "AI's half century. On the thresholds of the Dartmouth conference". En: *IA Retrospettiva* 3 (2006), págs. 1-2.

[6] Wikipedia. *Yann LeCun — Wikipedia, La enciclopedia libre*. [Internet; descargado 19-abril-2023]. 2022. URL: \url{https://es.wikipedia.org/w/index.php?title=Yann_LeCun&oldid=144356697}.

[7] S. A. Atencio Manyari et al. "Propuesta de segmentación de clientes aplicando técnicas de Machine Learning para mejorar la experiencia de compra mediante un sistema de recomendación de productos de Tottus". En: *Revista de Investigación Académica* 15.1 (2022).

[8] I. J. Badenes Villena. "Conceptualización, implementación y desarrollo de un coche autónomo desde cero mediante machine learning". Tesis doct. Universitat Politècnica de València, 2022.

[9] M. M. E. Torres y R. Manjarrés-Betancur. "Asistente virtual académico utilizando tecnologías cognitivas de procesamiento de lenguaje natural". En: *Revista Politécnica* 16.31 (2020).

[10] Jose Antonio Sánchez, experto en Ciencia de Datos. *Machine Learning y sus diferentes tipos*. https://datos.gob.es/es/blog/como-aprenden-las-maquinas-machine-learning-y-sus-diferentes-tipos. Acedido: 18 de marzo de 2023.

[11] Julianna Delua. *Supervised vs. Unsupervised Learning: What's the Difference?* https://www.ibm.com/cloud/blog/supervised-vs-unsupervised-learning. Acedido: 20 de marzo de 2023.

[12] A. E. Valentin Paucar. "Reconocimiento de dígitos escritos a mano usando redes neuronales". En: (2021).

[13] Danilo Loza Quispillo et al. "Incertidumbre en la predicción de la precipitación a partir de variables meteorológicas utilizando Modelos de Regresión Funcional". En: *Matemática* 20.1 (2022).

[14] María Quílez Miguel, María Alcalá Nalvaiz y José Tomás. "Métodos de reducción de la dimensionalidad: ACP vs t-SNE". En: (2022).

[15] Matthias Fuchs y Wolfram Höpken. "Clustering: Hierarchical, k-Means, DBSCAN". En: *Applied Data Science in Tourism: Interdisciplinary Approaches, Methodologies, and Applications*. Springer, 2022, págs. 129-149.

[16] Fernando Berzal. "Reglas de asociación". En: *Obtenido de http://elvex. ugr. es/idbis/dm/slides/2 % 20Association. pdf* (2016).

[17] E. Morales y J. González. *Aprendizaje por refuerzo*. Presentacion En Linea en: https://ccc.inaoep.mx/rales/Cursos/Aprendizaje2/Acetatos/refuerzo.pdf. 2012.

[18] Antonio David Ponce Martínez et al. "Desarrollo de jugadores automáticos mediante aprendizaje profundo por refuerzo para videojuegos". En: (2020).

[19] Miroslav Kubat. *An Introduction to Machine Learning*. Vol. 2nd Ed. Springer International Publishing AG, 2017.

[20] M. Cárdenas-Montes. *Sobreajuste–overfitting*. Ciemat (Centro de Investigaciones Energéticas, Medioambientales y Tecnológicas). 2006.

[21] AprendeIA. *Sobreajuste y subajuste en machine learning*. https://aprendeia.com/sobreajuste-y-subajuste-en-machine-learning/. Fecha de acceso: 2023.

[22] MathWorks. *Overfitting*. https://es.mathworks.com/discovery/overfitting.html.

[23] Mayuri S Shelke, Prashant R Deshmukh y Vijaya K Shandilya. "A review on imbalanced data handling using undersampling and oversampling technique". En: *Int. J. Recent Trends Eng. Res* 3.4 (2017), págs. 444-449.

[24] *Análisis de Errores en Machine Learning*. https://www.iartificial.net/analisis-de-errores-en-machine-learning/.

[25] Batuhan Bilgili. *Mobile Price Classification*. https://www.kaggle.com/code/batuhanbilgili/mobile-price-classification.

[26] *Árbol*. 2023. URL: {https://estructurasite.wordpress.com/arbol/}.

[27] David G.Stork Richard O. Duda Peter E.Hart. *Pattern Classification*. Vol. 2nd Ed.

[28] Trevor Hastie Jerome Friedman Robert Tibshirani. *The Elements of Statistical Learning*. Vol. 2nd Ed. Springer International Publishing AG, 2008.

[29] Adrián Rocha Íñigo. "Codificación de variables categóricas en aprendizaje automático". En: (2020).

[30] Luz Gloria Torres. "Redes neuronales y aproximación de funciones". En: *Boletín de Matemáticas* 1.2 (1994), págs. 35-58.

[31] *Función Sigmoide — Wikipedia, la enciclopedia libre*. https://es.wikipedia.org/wiki/Funci%C3%B3n_sigmoide. 2023.

[32] C Guadalupe Origel-Rivas et al. "Redes neuronales artificiales y árboles de decisión para la clasificación con datos categóricos." En: *Res. Comput. Sci.* 149.8 (2020), págs. 541-554.

[33] Jajang Jajang, Nunung Nurhayati y Suci Jena Mufida. "Ordinal Logistic Regression Model and Classification Tree on Ordinal Response Data". En: *BAREKENG: Jurnal Ilmu Matematika dan Terapan* 16.1 (2022), págs. 075-082.

[34] Alan Agresti. *An Introduction to Categorical Data Analysis*. Vol. 2nd Ed. Wiley-Interscience, 2007.

[35] W.Murray. *Numerical Methods for Unconstrained Optimization*. Academic Press, 1972.

[36] Laura Freijeiro González. "Modelos de predicción y clasi cación con alta dimensión en el número de covariables". En: ().

[37] Ouorou Ganni Mariel Guera et al. "Modelos de Regresión Logística Multinomial Ordinal y Redes Neuronales Artificiales para la clasificación de madera aserrada". En: *Revista Forestal Mesoamericana Kurú* 18.43 (2021), págs. 29-40.

[38] Morten W Fagerland y David W Hosmer. "Tests for goodness of fit in ordinal logistic regression models". En: *Journal of Statistical Computation and Simulation* 86.17 (2016), págs. 3398-3418.

[39] David W.Hosner. *Applied Logistic Regression*. Vol. 3rd Ed. 2013.

[40] Leo Breiman. "Random forests". En: *Machine learning* 45 (2001), págs. 5-32.

Apéndice A

Código

```
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, f1_score, make_scorer
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import KFold, RepeatedKFold
from sklearn.neural_network import MLPClassifier
from sklearn.pipeline import make_pipeline
from sklearn.compose import ColumnTransformer
from sklearn import tree
from statsmodels.miscmodels.ordinal_model import OrderedModel
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import chi2


'''
Extraccion de los datos y analisis previo
'''
data = pd.read_csv('train.csv') #Lectura de los datos
num_filas = data.shape[0]  # N mero de filas
num_columnas = data.shape[1]  # N mero de columnas
#Para ver todas las variables
pd.set_option('display.max_columns', None)

with open('analisis_datos.txt','w') as f:
    #Primeros 8 ejemplos
    f.write(data.head(8).to_string(index=False))
    f.write("\n")
    #Tabla resumen descriptivo variables explicativas
    f.write(data.describe(include="all").to_string(index=False))
    f.write("\n")
    #Descripci n estructura del data frame y valores nulos.
    data.info(buf=f)
    f.write("\n")
    #N mero de datos  nicos  por variable explicativa
    nunique_values = data.nunique()
    f.write(nunique_values.to_string())
    f.write("\n")

#Semilla datos aleatorios
np.random.seed(17)

#Crear graficos de linea entre variable explicativa y respuesta
def graficos_variables(variable):
    media_rango_precios = data.groupby(variable)['price_range'].mean()
    plt.plot(media_rango_precios.index, \
             media_rango_precios, marker='o')
    plt.xlabel(variable)
    plt.ylabel('price_range')
```

```
    plt.show()

# Separacion de los datos en variable respuesta y explicativa
X = data.iloc[:, :-1]
y = data.iloc[:, -1]

#Normalizadores y Variables a normalizar
columns_to_scale = ['battery_power', 'clock_speed', 'fc','int_memory',\
                    'm_dep','mobile_wt','n_cores','pc','px_height',\
                    'px_width','ram','sc_h','sc_w','talk_time']

scaler = StandardScaler()
scaler2=MinMaxScaler()

# Division datos de entrenamiento y test para validacion cruzada 5x2
CV = RepeatedKFold(n_splits=2, n_repeats=5, random_state=5)

#kfold para la seleccion de hiperparametros en GridSearchCV
kfold = KFold(n_splits=5,shuffle=True, random_state=23)

#Definicion auxiliar para la metrica de evaluacion MARE
def mdp(y_test, y_pred):
        diferencia_abs = np.abs(y_test - y_pred)
        suma_dif =  np.sum(diferencia_abs)
        media_diferencias = suma_dif / len(diferencia_abs)
        return media_diferencias

#Metricas que se van a utilzar en seleccion hiperparametros
scoring = {
    'accuracy': make_scorer(accuracy_score),
    'macro_f1': make_scorer(f1_score, average='macro'),
    'mdp': make_scorer(mdp, greater_is_better=False)
}

#Funcion auxiliar para hacer la media de los resultados de CV
def suma_arrays(lista_arrays):
    n_arrays = len(lista_arrays)
    suma = np.zeros_like(lista_arrays[0])
    for array in lista_arrays:
        suma += array
    suma /= n_arrays
    return suma

#Funciones auxiliares preparacion datos para gr ficas
def partir_lista1(lista, n):
    sub_listas = [[] for _ in range(n)]
    for i, elemento in enumerate(lista):
        sub_listas[i % n].append(elemento)
    return sub_listas

def partir_lista2(lista, n):
    sub_listas = []
    for i in range(0, len(lista), n):
        sub_listas.append(lista[i:i+n])
    return sub_listas

#Crear las graficas evaluacion-hiperparametros segun el algoritmo
def crear_grafica(x,y,xlabel, ylabel, title):
    plt.plot(x, y, color='red')
```

```
    plt.xlabel(xlabel)
    plt.ylabel(ylabel)
    plt.title(title)

def crear_grafica_k_NN(x, y, xlabel, ylabel, title):
    y_pesos=partir_lista1(y, 2)
    y_uniform=y_pesos[0]
    y_distance=y_pesos[1]
    plt.plot(x, y_uniform, color='blue', label='votacion␣uniforme')
    plt.plot(x, y_distance, color='red', label='votacion␣ponderada')
    plt.xlabel(xlabel)
    plt.ylabel(ylabel)
    plt.title(title)
    plt.legend()

colores = ['blue', 'red', 'green', 'purple', 'orange', 'cyan', 'magenta',\
           'yellow', 'black', 'gray', 'brown', 'pink', 'teal', 'navy', \
           'olive', 'salmon', 'gold', 'indigo', 'lavender', 'lime']

def crear_grafica_arboles_parada(X,y,xlabel,ylabel\
                                 ,title,min_impurity_decrease,\
                                     min_samples_split):
    y_mod=partir_lista2(y, len(min_samples_split))
    for i in range(len(y_mod)):
        plt.plot(X, y_mod[i], color=colores[i], \
                 label=f'beta␣=␣{min_impurity_decrease[i]}')
    plt.xlabel(xlabel)
    plt.ylabel(ylabel)
    plt.title(title)
    plt.legend()

#Dado un algoritmo, te devuelve las evaluaciones medias por hiperparametro
#de los GridSearchCV()
def comparacion_hiperparametros(model,filename,X,y):
    scores_acc_kfold=[]
    scores_f1_kfold=[]
    scores_mdp_kfold=[]


    for train_index, test_index in CV.split(X):
        X_train= X.iloc[train_index]
        y_train= y.iloc[train_index]

        model.fit(X_train,y_train)

        scores_acc = model.cv_results_['mean_test_accuracy']
        scores_f1 = model.cv_results_['mean_test_macro_f1']
        scores_mdp= model.cv_results_['mean_test_mdp']

        scores_acc_kfold.append(scores_acc)
        scores_f1_kfold.append(scores_f1)
        scores_mdp_kfold.append(scores_mdp)

    params=model.cv_results_['params']
    scores_acc = suma_arrays(scores_acc_kfold)
    scores_f1 = suma_arrays(scores_f1_kfold)
    scores_mdp = suma_arrays(scores_mdp_kfold)

    with open(filename,'w') as f:
```

```
            f.write("\nResultados␣Modelo:\n")
            f.write("\nExactitud:\n")
            f.write(str(scores_acc))
            f.write("\nF1-Score:\n")
            f.write(str(scores_f1))
            f.write("\nMDP:\n")
            f.write(str(scores_mdp))
            f.write("\nParametros:\n")
            f.write(str(params))

        return (scores_acc,scores_f1,scores_mdp)

    #Dado un modelo, te devuelve las evaluaciones medias en los tests de 5x2CV
    def evaluaciones_CV(model,X,y):
        scores_acc_CV=[]
        scores_f1_CV=[]
        scores_mdp_CV=[]

        for train_index, test_index in CV.split(X):
            X_train= X.iloc[train_index]
            y_train= y.iloc[train_index]
            X_test= X.iloc[test_index]
            y_test= y.iloc[test_index]


            model.fit(X_train,y_train)
            y_pred=model.predict(X_test)
            acc= accuracy_score(y_test, y_pred)
            scores_acc_CV.append(acc)
            f1= f1_score(y_test, y_pred, average='macro')
            scores_f1_CV.append(f1)
            mdp_score= mdp(y_test, y_pred)
            scores_mdp_CV.append(mdp_score)

        acc_CV = np.mean(scores_acc_CV)
        f1_CV = np.mean(scores_f1_CV)
        mdp_CV = np.mean(scores_mdp_CV)
        return (acc_CV,f1_CV,mdp_CV)

    #Normalizando previamente los conjuntos de entrenamiento
    def evaluaciones_CV_norm(model,X,y,normalizador):
        scores_acc_CV=[]
        scores_f1_CV=[]
        scores_mdp_CV=[]

        for train_index, test_index in CV.split(X):
            X_train= X.iloc[train_index]
            y_train= y.iloc[train_index]
            X_test= X.iloc[test_index]
            y_test= y.iloc[test_index]

            scaler.fit(X_train[columns_to_scale])

            X_train_scaled = X_train.copy()
            X_train_scaled[columns_to_scale] = scaler.\
                transform(X_train_scaled[columns_to_scale])

            X_test_scaled = X_test.copy()
            X_test_scaled[columns_to_scale] = scaler.\
```

```
            transform(X_test_scaled[columns_to_scale])

        model.fit(X_train_scaled,y_train)
        y_pred=model.predict(X_test_scaled)
        acc= accuracy_score(y_test, y_pred)
        scores_acc_CV.append(acc)
        f1= f1_score(y_test, y_pred, average='macro')
        scores_f1_CV.append(f1)
        mdp_score= mdp(y_test, y_pred)
        scores_mdp_CV.append(mdp_score)

    acc_CV = np.mean(scores_acc_CV)
    f1_CV = np.mean(scores_f1_CV)
    mdp_CV = np.mean(scores_mdp_CV)
    return (acc_CV,f1_CV,mdp_CV)

#Para los modelos de regresion:
def evaluaciones_CV_reg(data_reg_final,max_iter):
    scores_acc_CV=[]
    scores_f1_CV=[]
    scores_mdp_CV=[]
    X_sub=data_reg_final.iloc[:,:-1]

    for train_index, test_index in CV.split(X_sub):
        X_train, X_test = X_sub.iloc[train_index], X_sub.iloc[test_index]
        y_train, y_test = y[train_index], y[test_index]

        model5=OrderedModel(y_train, X_train, distr='logit')
        results=model5.fit(method='nm',maxiter=max_iter)
        predicted = results.model.predict(results.params, X_test)
        y_pred = np.argmax(predicted, axis=1)
        f1=f1_score(y_test, y_pred,average='macro')
        scores_f1_CV.append(f1)
        acc=accuracy_score(y_test, y_pred)
        scores_acc_CV.append(acc)
        mdp_score=mdp(y_test,y_pred)
        scores_mdp_CV.append(mdp_score)

    acc_CV = np.mean(scores_acc_CV)
    f1_CV = np.mean(scores_f1_CV)
    mdp_CV = np.mean(scores_mdp_CV)
    return (acc_CV,f1_CV,mdp_CV)

#Funcion resultados de regresion logistica en un entrenamiento
def entrenamiento_regresion(columns_to_drop_wald,num_entrenamiento,\
                            maxiter):
    subset_data=data_reg.drop(columns=columns_to_drop_wald)
    X_sub=subset_data.iloc[:,:-1]

    train_index, test_index = list(CV.split(X))[num_entrenamiento]
    X_train, X_test = X_sub.iloc[train_index], X_sub.iloc[test_index]
    y_train, y_test = y[train_index], y[test_index]

    model5=OrderedModel(y_train, X_train, distr='logit')
    results=model5.fit(method='nm',maxiter=maxiter)
    print(results.summary())
    predicted = results.model.predict(results.params, X_test)
    y_pred = np.argmax(predicted, axis=1)
    f1=f1_score(y_test, y_pred,average='macro')
```

```
    acc=accuracy_score(y_test, y_pred)
    mdp_score=mdp(y_test,y_pred)
    scores=(acc,f1,mdp_score)

    return scores

#Y normalizando las variables
def entrenamiento_regresion_norm(columns_to_drop_wald,num_entrenamiento,\
                                 maxiter,columns_to_scale):
    subset_data=data_reg.drop(columns=columns_to_drop_wald)
    X_sub=subset_data.iloc[:,:-1]

    train_index, test_index = list(CV.split(X))[num_entrenamiento]
    X_train, X_test = X_sub.iloc[train_index], X_sub.iloc[test_index]
    y_train, y_test = y[train_index], y[test_index]

    scaler.fit(X_train[columns_to_scale])

    X_train_scaled = X_train.copy()
    X_train_scaled[columns_to_scale] = scaler.\
        transform(X_train_scaled[columns_to_scale])
    X_test_scaled = X_test.copy()
    X_test_scaled[columns_to_scale] = scaler.\
        transform(X_test_scaled[columns_to_scale])

    model5=OrderedModel(y_train, X_train_scaled, distr='logit')
    results=model5.fit(method='nm',maxiter=maxiter)
    print(results.summary())
    predicted = results.model.predict(results.params, X_test_scaled)
    y_pred = np.argmax(predicted, axis=1)
    f1=f1_score(y_test, y_pred,average='macro')
    acc=accuracy_score(y_test, y_pred)
    mdp_score=mdp(y_test,y_pred)
    scores=(acc,f1,mdp_score)
    return scores

#Calculo estadistico Bondad de Ajuste Regresion Logistica
def calculo_estadistico_CG(probabilidades,y):
    ordinal_scores = np.sum(probabilidades * np.arange(1, 4+1), axis=1)
    sorted_indices = np.argsort(ordinal_scores)
    probabilidades_sorted=probabilidades[sorted_indices]
    y_sorted = y[sorted_indices]
    probabilidades_groups = np.array_split(probabilidades_sorted, 10)
    y_groups = np.array_split(y_sorted, 10)
    observed_frequencies = []
    expected_frequencies = []
    for g in range(10):
        group_pred = probabilidades_groups[g]
        group_obs = y_groups[g]
        group_pred_t = np.transpose(group_pred)
        group_freq_pred = np.sum(group_pred_t, axis=1)
        group_freq_obs = np.bincount(group_obs, minlength=4)
        expected_frequencies.append(group_freq_pred)
        observed_frequencies.append(group_freq_obs)
    CG=0
    for g in range(10):
        for j in range(4):
            if expected_frequencies[g][j]!=0:
                aux=(observed_frequencies[g][j]-\
```

```
                    expected_frequencies[g][j])**2
                aux2=aux/expected_frequencies[g][j]
                CG=CG+aux2
    return CG

'''
-------------COMPARACION HIPERPARAMETROS------------

Modelo 1: Arboles para clasificacion con parada temprana
'''
np.random.seed(27)

param_grid1 = {
    'min_samples_split': [3,5,8,9,10,11,12,15],
    'min_impurity_decrease': [0,0.0001,0.001,0.01],
}

min_samples_split=param_grid1['min_samples_split']
min_impurity_decrease=param_grid1['min_impurity_decrease']

model1=GridSearchCV(estimator=DecisionTreeClassifier(criterion='entropy'\
                                                     ,splitter='best'),\
                    param_grid=param_grid1,\
                    scoring=scoring,\
                    cv=kfold,\
                    refit=False)

(scores_acc,scores_f1,scores_mdp)=\
    comparacion_hiperparametros(model1,'resultados_arbparada.txt',X,y)

#Graficas
fig_acc=plt.figure()
crear_grafica_arboles_parada(min_samples_split, \
                             scores_acc, 'm',\
                                 'Exactitud',\
                                     'Exactitud hiperparametros',\
                                         min_impurity_decrease,\
                                             min_samples_split)
plt.show(fig_acc)

fig_f1=plt.figure()
crear_grafica_arboles_parada(min_samples_split, scores_f1,\
                        'm','F1-Score','F1-score hiperparametros',\
                            min_impurity_decrease,min_samples_split)
plt.show(fig_f1)

fig_mdp=plt.figure()
crear_grafica_arboles_parada(min_samples_split, scores_mdp,'m','MDP',\
                        'MDP hiperparametros',min_impurity_decrease,\
                            min_samples_split)
plt.show(fig_mdp)

'''
Modelo 2: Arboles para clasificacion con poda
'''
np.random.seed(21)

param_grid2 = {
    'ccp_alpha':[0.0001,0.0005,0.001,0.002,0.0035,\
```

```
                0.005,0.007,0.008,0.009,0.01],
}

ccp_alpha=param_grid2['ccp_alpha']

model2=GridSearchCV(estimator=DecisionTreeClassifier(criterion='entropy'\
                                                     ,splitter='best'),\
                    param_grid=param_grid2,\
                    scoring=scoring,\
                    cv=kfold,\
                    refit=False)

(scores_acc,scores_f1,scores_mdp)=\
    comparacion_hiperparametros(model2,'resultados_arbpoda.txt',X,y)

#Graficas
fig_acc=plt.figure()
crear_grafica(ccp_alpha, scores_acc, 'alpha', 'Exactitud', \
              'Exactitud-Valores de alpha')
plt.show(fig_acc)

fig_f1=plt.figure()
crear_grafica(ccp_alpha, scores_f1, 'alpha', 'F1-Score', \
              'F1-Score-Valores de alpha')
plt.show(fig_f1)

fig_mdp=plt.figure()
crear_grafica(ccp_alpha, scores_mdp, 'alpha', 'MDP', \
              'MDP-Valores de alpha')
plt.show(fig_mdp)

'''
Modelo 3 V1: k-NN (Sin Normalizar Variables)
'''
param_grid3 = {
    'n_neighbors': [1,3,5,7,10,12,15,17,20,22,25,27,30,35],
    'weights': ['uniform','distance'],
}

vecinos=param_grid3['n_neighbors']
pesos=param_grid3['weights']

model3=GridSearchCV(estimator=KNeighborsClassifier(),\
                    param_grid=param_grid3,\
                    scoring=scoring,\
                    cv=kfold,\
                    refit=False)

(scores_acc,scores_f1,scores_mdp)=\
    comparacion_hiperparametros(model3,'resultados_kNN_noNorm.txt',X,y)

#Graficas:
fig_acc=plt.figure()
crear_grafica_k_NN(vecinos, scores_acc, 'Vecinos',\
                   'Exactitud', 'Exactitud hiperparametros')
plt.show(fig_acc)

fig_f1=plt.figure()
crear_grafica_k_NN(vecinos,scores_f1, 'Vecinos',\
```

```
                    'F1-Score','F1-Score hiperparametros')
plt.show(fig_f1)

fig_mdp=plt.figure()
crear_grafica_k_NN(vecinos, scores_mdp ,'Vecinos','MDP',\
                'MDP hiperparametros')
plt.show(fig_mdp)

'''
Modelo 3 V2: k-NN (Seleccion Variables Gran Escala No Normalizadas)

(Tras varias pruebas, dejo seleccionadas las variables con las que\
se consiguen mejores resultados. Esta version es la que mejor
resultados consigue de las 4)
'''
#Selecci n variables mayor escala
subset_datakNN = data[['battery_power','int_memory','pc','mobile_wt',\
                       'px_height','px_width','ram','price_range']]

X_sub = subset_datakNN.iloc[:, :-1]

param_grid3 = {
    'n_neighbors': [1,3,5,7,10,12,15,17,20,22,25,27,30,35],
    'weights': ['uniform','distance'],
}

vecinos=param_grid3['n_neighbors']
pesos=param_grid3['weights']

model3=GridSearchCV(estimator=KNeighborsClassifier(),\
                    param_grid=param_grid3,\
                    scoring=scoring,\
                    cv=kfold,\
                    refit=False)

(scores_acc,scores_f1,scores_mdp)=\
    comparacion_hiperparametros(model3,'resultados_kNN_NoNormImp.txt',\
                                X_sub,y)

#Graficas:
fig_acc=plt.figure()
crear_grafica_k_NN(vecinos, scores_acc, 'Vecinos',\
                    'Exactitud','Exactitud hiperpar metros')
plt.show(fig_acc)

fig_f1=plt.figure()
crear_grafica_k_NN(vecinos, scores_f1, 'Vecinos',\
                    'F1-Score','F1-Score hiperpar metros')
plt.show(fig_f1)

fig_mdp=plt.figure()
crear_grafica_k_NN(vecinos, scores_mdp,'Vecinos','MDP',\
                'MDP hiperpar metros')
plt.show(fig_mdp)

'''
Modelo 3 V3: k-NN (Variables Normalizadas)

(He elegido la normalizacion por estandarizacion)
```

```
'''
param_grid3 = {
    'kneighborsclassifier__n_neighbors': [20, 30, 40, 50, 75, 100,\
                                          120, 150,175,200],
    'kneighborsclassifier__weights': ['uniform', 'distance'],
}

vecinos=param_grid3['kneighborsclassifier__n_neighbors']
pesos=param_grid3['kneighborsclassifier__weights']

preprocessor = ColumnTransformer(
    transformers=[('scaler', scaler, columns_to_scale)],
    remainder='passthrough')

pipeline = make_pipeline(preprocessor, KNeighborsClassifier())

model3=GridSearchCV(pipeline,\
                    param_grid=param_grid3,\
                    scoring=scoring,\
                    cv=kfold,\
                    refit=False)

(scores_acc,scores_f1,scores_mdp)=\
    comparacion_hiperparametros(model3,'resultados_kNN_Norm.txt',\
                                X,y)

#Graficas
fig_acc=plt.figure()
crear_grafica_k_NN(vecinos, scores_acc, 'Vecinos',\
                   'Exactitud','Exactitud␣hiperparametros')
plt.show(fig_acc)

fig_f1=plt.figure()
crear_grafica_k_NN(vecinos, scores_f1, 'Vecinos',\
                   'F1-Score', 'F1-Score␣hiperparametros')
plt.show(fig_f1)

fig_mdp=plt.figure()
crear_grafica_k_NN(vecinos, scores_mdp,'Vecinos','MDP',\
               'MDP␣hiperparametros')
plt.show(fig_mdp)


'''
Modelo 3 V4: k-NN (Normalizacion Variables con mayor escala)

(Tras varias pruebas, dejo seleccionadas las variables con las que \
se consiguen mejores resultados)
'''
param_grid3 = {
    'kneighborsclassifier__n_neighbors':[1,3,5,7,10,12,15,\
                                         17,20,22,25,27,30,35],
    'kneighborsclassifier__weights': ['uniform', 'distance'],
}

#Escogemos en este caso las variables con las que se consiguen
#mejores resultados

subset_datakNN2 = data[['battery_power', 'mobile_wt','px_height',\
```

```
                        'px_width','ram','price_range']]

X_sub2 = subset_datakNN2.iloc[:, :-1]

columns_to_scale2 = ['battery_power',\
                     'mobile_wt','px_height',\
                     'px_width','ram']

vecinos=param_grid3['kneighborsclassifier__n_neighbors']
pesos=param_grid3['kneighborsclassifier__weights']

preprocessor = ColumnTransformer(
    transformers=[
        ('scaler', scaler, columns_to_scale2)
    ],
    remainder='passthrough'
)

pipeline = make_pipeline(preprocessor, KNeighborsClassifier())

model3=GridSearchCV(pipeline,\
                    param_grid=param_grid3,\
                    scoring=scoring,\
                    cv=kfold,\
                    refit=False)

(scores_acc,scores_f1,scores_mdp)=\
    comparacion_hiperparametros(model3,'resultados_kNN_NormImp.txt',\
                                X_sub2,y)

#Graficas:
fig_acc=plt.figure()
crear_grafica_k_NN(vecinos, scores_acc, 'Vecinos',\
                   'Exactitud', 'Exactitud␣hiperparametros')
plt.show(fig_acc)

fig_f1=plt.figure()
crear_grafica_k_NN(vecinos, scores_f1, 'Vecinos',\
                   'F1-Score', 'F1-Score␣hiperparametros')
plt.show(fig_f1)

fig_mdp=plt.figure()
crear_grafica_k_NN(vecinos,scores_mdp,'Vecinos','MDP',\
              'MDP␣hiperparametros')
plt.show(fig_mdp)

'''
Modelo 4: Redes Neuronales

(Aunque hemos realizado la prueba para todas las posibles combinaciones
 de hiperparametros, dejamos seleccionadas ya las mejores y variamos
solo el n mero de neuronas, ya que si no es muy costoso
computacionalmente. El modelo ya hace la creaci n de los target values
de manera correcta)
'''

param_grid4 = {
    'mlpclassifier__hidden_layer_sizes': [(25),(50),(100),(150)\
                                          ,(200),(250),(300),(350),\
```

```
                                                    (400),(450)],
    #'mlpclassifier__alpha':[0.0,0.0001,0.001,0.01,0.1],
    #'mlpclassifier__max_iter':[200,300,500,750,1000],
    #'mlpclassifier__learning_rate':['constant','invscaling'],
    #'mlpclassifier__learning_rate_init':[0.0001,0.001],
    #'early_stopping':[True,False],
}

neuronas=param_grid4['mlpclassifier__hidden_layer_sizes']

preprocessor = ColumnTransformer(
    transformers=[
        ('scaler', scaler, columns_to_scale)
    ],
    remainder='passthrough'
)

pipeline = make_pipeline(preprocessor,\
                         MLPClassifier(max_iter=1000,\
                                       random_state=17,\
                                           learning_rate='constant',\
                                               solver='sgd',\
                                                   activation='logistic',\
                                                       batch_size='auto'))

model4=GridSearchCV(pipeline,\
                    param_grid=param_grid4,\
                    scoring=scoring,\
                    cv=kfold,\
                    refit=False)

(scores_acc,scores_f1,scores_mdp)=\
    comparacion_hiperparametros(model4,'resultados_redes',X,y)

#Graficas:
fig_acc=plt.figure()
crear_grafica(neuronas, scores_acc, 'Neuronas','Exactitud', \
              'Exactitud-Numero␣Neuronas')
plt.show(fig_acc)

fig_f1=plt.figure()
crear_grafica(neuronas,scores_f1, 'Neuronas','F1-Score', \
              'F1-Score-Numero␣Neuronas')
plt.show(fig_f1)

fig_mdp=plt.figure()
crear_grafica(neuronas,scores_mdp,'Neuronas','MDP', \
              'MDP-Numero␣Neuronas')
plt.show(fig_mdp)

'''
Regresion logistica Ordinal:
(Tras aplicar backward, dejamos ya seleccionadas las variables
 que vamos a eliminar del modelo en cada entrenamiento)
'''
matriz_correlacion=data.corr()
#Variables con alta correlacion con otras
columns_to_drop_corr = ['three_g','sc_h','fc']
data_reg=data.drop(columns=columns_to_drop_corr)
```

```
'''
Entrenamiento 1:
battery_power,mobile_wt,px_height,px_width,ram->(0.983,0.982,0.017)
'''
columns_to_drop_wald1=['touch_screen','dual_sim','wifi','m_dep','n_cores',\
                       'blue','four_g','clock_speed','sc_w','int_memory'\
                           ,'pc','talk_time']
scores=entrenamiento_regresion(columns_to_drop_wald1, 0, 6000)
print(scores)

'''
Entrenamiento 2:
battery_power,int_memory,mobile_wt,pc,px_height,px_width,ram,sc_w\
    ->(0.974,0.974,0.026)
'''
columns_to_drop_wald2=['blue','m_dep','dual_sim','touch_screen','wifi',\
                       'four_g','talk_time','n_cores','clock_speed']
scores=entrenamiento_regresion(columns_to_drop_wald2,1, 6000)
print(scores)

'''
Entrenamiento 3:
battery_power,int_memory,mobile_wt,px_height,px_width,ram\
    ->(0.978,0.978,0.022)
'''

columns_to_drop_wald3=['wifi','blue','four_g','m_dep','clock_speed'\
                       ,'touch_screen','sc_w','dual_sim','talk_time',\
                           'n_cores','pc']
scores=entrenamiento_regresion(columns_to_drop_wald3,2, 6000)
print(scores)

'''
Entrenamiento 4:
battery_power,mobile_wt,px_height,px_width,ram->(0.974,0.973,0.026)
'''
columns_to_drop_wald4=['clock_speed','dual_sim','four_g','m_dep',\
                       'n_cores','int_memory','wifi','pc','sc_w',\
                           'blue','talk_time','touch_screen']
scores=entrenamiento_regresion(columns_to_drop_wald4,3, 6000)
print(scores)

'''
Entrenamiento 5:
battery_power,mobile_wt,px_height,px_width,ram,wifi->(0.973,0.973,0.027)
'''
columns_to_drop_wald5=['clock_speed','blue','int_memory','touch_screen'\
                       ,'dual_sim','m_dep','pc','four_g','n_cores',\
                           'talk_time','sc_w']
scores=entrenamiento_regresion(columns_to_drop_wald5,4, 6000)
print(scores)

'''
Entrenamiento 6:
battery_power,mobile_wt,px_height,px_width,ram->(0.98,0.98,0.02)
'''
columns_to_drop_wald6=['pc','dual_sim','four_g','touch_screen','wifi'\
                       ,'n_cores','m_dep','clock_speed','blue',\
```

```
                            'int_memory','talk_time','sc_w']
scores=entrenamiento_regresion(columns_to_drop_wald6,5, 6000)
print(scores)

'''
Entrenamiento 7:
battery_power,mobile_wt,px_height,px_width,ram->(0.973,0.973,0.027)
'''
columns_to_drop_wald7=['touch_screen','m_dep','pc','talk_time','dual_sim'\
                       ,'blue','wifi','sc_w','four_g','n_cores',\
                           'int_memory','clock_speed']
scores=entrenamiento_regresion(columns_to_drop_wald7,6, 6000)
print(scores)

'''
Entrenamiento 8:
battery_power,int_memory,mobile_wt,n_cores,px_height,px_width,ram,sc_w\
    ->(0.94,0.94,0.06)
'''
columns_to_drop_wald8=['m_dep','blue','four_g','wifi','dual_sim','pc',\
                       'touch_screen','talk_time','clock_speed']
scores=entrenamiento_regresion(columns_to_drop_wald8,7, 8000)
print(scores)

'''
Entrenamiento 9:
battery_power,int_memory,mobile_wt,pc,px_height,px_width,ram\
    ->(0.977,0.976,0.023)
'''
columns_to_drop_wald9=['touch_screen','talk_time','blue','dual_sim',\
                       'four_g','clock_speed','m_dep','wifi','n_cores',\
                           'sc_w']
scores=entrenamiento_regresion(columns_to_drop_wald9,8, 8000)
print(scores)

'''
Entrenamiento 10:
battery_power,clock_speed,mobile_wt,px_height,px_width,ram,sc_w,talk_time\
    ->(0.94,0.94,0.06)
'''
columns_to_drop_wald10=['wifi','n_cores','touch_screen','four_g',\
                        'dual_sim','int_memory','blue','m_dep',\
                            'pc']
scores=entrenamiento_regresion(columns_to_drop_wald10,9, 8000)
print(scores)


'''
Prueba con variables normalizadas:

(Llegamos a exactitud de 0.82, muy alejado de los
resultados sin normalizar, asique no hacemos mas
pruebas)
'''

columns_to_drop_wald_norm1=['n_cores','px_width','sc_w','blue','mobile_wt',\
                            'px_height','wifi','clock_speed','int_memory',\
                                'pc','four_g','touch_screen','talk_time',\
                                    'dual_sim','m_dep']
```

```
columns_to_scale1=['battery_power',
                   'ram']

scores=entrenamiento_regresion_norm(columns_to_drop_wald_norm1,0, 6000\
                                    ,columns_to_scale1)
print(scores)


'''
-------COMPARACION MEJORES MODELOS--------

Resultados CV Modelo 1:
'''
modelo_mejor1=DecisionTreeClassifier(criterion='entropy',splitter='best',\
                                     min_samples_split=9,
                                     min_impurity_decrease=0.001)
scores_modelo_mejor1=evaluaciones_CV(modelo_mejor1,X,y)
with open('evaluaciones_finales.txt','w') as f:
    f.write("\n␣Resultados␣mejor␣ rbol ␣con␣parada:\n")
    f.write(str(scores_modelo_mejor1))

#Grafico del  rbol  con todos los datos
modelo_mejor1.fit(X,y)
class_names = list(map(str, modelo_mejor1.classes_))
plt.figure(figsize=(20, 16))
tree.plot_tree(modelo_mejor1, filled=True, fontsize=10,\
               feature_names=X.columns, class_names=class_names)
plt.show()

# Indicadores de importancia de las variables explicativas (ordenados)
importance = modelo_mejor1.feature_importances_
feature_importances = list(zip(importance, X.columns))
feature_importances_sorted = sorted(feature_importances, reverse=True)
for importance, feature in feature_importances_sorted:
    if importance>0:
        print(f"{feature}:␣{importance}")

'''
Resultados CV Modelo 2:
'''
modelo_mejor2=DecisionTreeClassifier(criterion='entropy',splitter='best',\
                                     ccp_alpha=0.007)
scores_modelo_mejor2=evaluaciones_CV(modelo_mejor2,X,y)
with open('evaluaciones_finales.txt','w') as f:
    f.write("\n␣Resultados␣mejor␣arbol␣con␣poda:\n")
    f.write(str(scores_modelo_mejor2))

#Gr fico del  rbol  con todos los datos
modelo_mejor2.fit(X,y)
class_names = list(map(str, modelo_mejor2.classes_))
plt.figure(figsize=(20, 16))
tree.plot_tree(modelo_mejor2, filled=True, fontsize=10,\
               feature_names=X.columns, class_names=class_names)
plt.show()

# Indicadores de importancia de las variables explicativas (ordenados)
importance = modelo_mejor2.feature_importances_
feature_importances = list(zip(importance, X.columns))
feature_importances_sorted = sorted(feature_importances, reverse=True)
```

```
for importance, feature in feature_importances_sorted:
    if importance>0:
        print(f"{feature}: {importance}")

'''
Resultados CV Modelo 3
'''
modelo_mejor3=KNeighborsClassifier(weights='distance',n_neighbors=20)
subset_datakNN = data[['battery_power','int_memory','pc','mobile_wt',\
                    'px_height','px_width','ram','price_range']]
X_sub = subset_datakNN.iloc[:, :-1]
scores_modelo_mejor3=evaluaciones_CV(modelo_mejor3,X_sub,y)
with open('evaluaciones_finales.txt','w') as f:
    f.write("\n Resultados mejor modelo k-NN:\n")
    f.write(str(scores_modelo_mejor3))

'''
Resultados CV Modelo 4
'''
modelo_mejor4=MLPClassifier(max_iter=1000,random_state=17,\
                            learning_rate='constant', solver='sgd',\
                                activation='logistic',batch_size='auto',\
                                    hidden_layer_sizes=(400))
scores_modelo_mejor4=evaluaciones_CV_norm(modelo_mejor4,X,y,scaler)
with open('evaluaciones_finales.txt','w') as f:
    f.write("\n Resultados mejor red neuronal MLP:\n")
    f.write(str(scores_modelo_mejor4))

'''
Resultados CV Modelo 5
'''
columns_to_drop_final=['three_g','sc_h','fc','touch_screen','dual_sim',\
                       'wifi','m_dep','n_cores','blue','four_g',\
                           'clock_speed','sc_w','int_memory'\
                           ,'pc','talk_time']
data_reg_final=data.drop(columns=columns_to_drop_final)

scores_modelo_mejor5=evaluaciones_CV_reg(data_reg_final,8000)

#Entrenamos el mejor modelo con todos los datos
X_reg_final=data_reg_final.iloc[:, :-1]
modelo_mejor5=OrderedModel(y, X_reg_final, distr='logit')
results_final=modelo_mejor5.fit(method='nm',maxiter=6000)
predicted_final = results_final.model.predict(results_final.params,\
                                              X_reg_final)
y_pred = np.argmax(predicted_final, axis=1)
#Test Bondad de Ajuste
CG=calculo_estadistico_CG(predicted_final,y)
df = (10 - 2) * (4 - 1) + (4 - 2)
critical_value = chi2.ppf(1 - 0.05, df)

#Calculo Odds ratio
coeficientes = results_final.params
odds_ratios = np.exp(coeficientes)


with open('evaluaciones_finales.txt','w') as f:
    f.write("\n Evaluaciones mejor modelo Regresion Logistica\
           Ordinal:\n")
```

```
    f.write(str(scores_modelo_mejor5))
    f.write("\n␣Informaci n␣relevante␣modelo␣RLO:\n")
    f.write(str(results_final.summary()))
    f.write("\n␣Coeficientes:\n")
    f.write(str(coeficientes))
    f.write("\nBondad␣de␣Ajuste:\n")
    if CG < critical_value:
        f.write("Hay␣una␣buena␣bondad␣de␣ajuste.\
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣No␣se␣rechaza␣la␣hip tesis␣nula.")
    else:
        f.write("No␣hay␣una␣buena␣bondad␣de␣ajuste.␣\
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣Se␣rechaza␣la␣hip tesis␣nula.")
    f.write("\n␣Odds␣Ratio:\n")
    f.write(str(odds_ratios))
```