Trabajo Fin de Grado
Grado en Ingeniería Aeroespacial

Automatización de diagnósticos mediante el uso de técnicas de Machine Learning

Autor: Esdras Sánchez Sánchez

Tutor: María Ángeles Martín Prats

Dpto.Ingeniería Electrónica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2021

Trabajo Fin de Grado
Grado en Ingeniería Aeroespacial

Automatización de diagnósticos mediante el uso de técnicas de Machine Learning

Autor:
Esdras Sánchez Sánchez

Tutor:
María Ángeles Martín Prats
Profesor Titular

Dpto.Ingeniería Electrónica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2021

Trabajo Fin de Grado: Automatización de diagnósticos mediante el uso de técnicas de Machine Learning

Autor: Esdras Sánchez Sánchez
Tutor: María Ángeles Martín Prats

El tribunal nombrado para juzgar el trabajo arriba indicado, compuesto por los siguientes profesores:

Presidente:

Vocal/es:

Secretario:

acuerdan otorgarle la calificación de:

El Secretario del Tribunal

Fecha:

Agradecimientos

A mis padres, por ser mi pilar incondicional, mi apoyo día tras día y transmitirme siempre felicidad. Por enseñarme valores tan importantes como el respeto, la solidaridad, la constancia y el trabajo. Sois mi ejemplo a seguir.
A mi hermana, por ser una persona extraordinaria en todos los sentidos. Sigue siendo como eres porque vas a conseguir todo lo que te propongas.

A Carmen, por haberte cruzado en el camino durante esta aventura y llenarme de alegría día tras día. Esto es solo el comienzo de todo lo que nos espera.

A mi familia y amigos, muchas gracias.

Al grupo docente de la Universidad y en especial a mi tutora María Ángeles, por haberme abierto los ojos y ayudarme a descubrir mi pasión. Muchas gracias por todo el tiempo y esfuerzo que ha dedicado ha sido un placer trabajar con usted durante este tiempo.

Resumen

El avance de los estudios realizados en el campo de la Inteligencia Artificial, así como su implementación directa en la industria está suponiendo un cambio inminente. En la industria aeronáutica la seguridad es un pilar fundamental. Los accidentes aéreos se producen debido tanto al factor humano como al fallo de la maquinaria. Aquí donde el Machine Learning tiene la oportunidad de detectar estos errores y mitigarlos. Algunas de las aplicaciones pueden ser usadas para predecir el mantenimiento de ciertas partes del avión, otras para liberar de carga de trabajo al piloto a la hora de tomar decisiones en situaciones de elevado estrés mediante la indicación por parte de la máquina de la acción más favorable que debe realizar. Mediante el empleo de estas técnicas se puede reducir potencialmente el número de accidentes. En una situación similar a la del piloto se encuentra la tarea de un médico, que también se encarga de proteger y salvar vidas, y para ello la toma de decisiones es fundamental. La motivación del presente trabajo es abordar problemas cercanos a las personas, del día a día. Esto se llevará a cabo mediante el empleo de técnicas de aprendizaje automático, también conocido como Machine Learning.

Abstract

The advance of the studies carried out in the field of artificial intelligence as well as the direct implementation in the industry is assuming an imminent change. In the aeronautical industry, safety is a fundamental. Air accidents occur due to both the human factor and the failure of machinery. This is where Machine Learning has the opportunity to detect these errors and mitigate them. Some of the examples may be for the maintenance of certain parts of the aircraft or to relieve the pilot of workload when making decisions in highly stressing situations by indicating him the most favorable action he should take. Through the use of these techniques, the number of accidents can potentially be reduced.A situation which is similar to aircrafts pilots is the role of a doctor, who is also in charge of protecting and saving lives, and for this, decision-making in critical situations is essential. The motivation of this project is to solve problems close to day-to-day life through the use of Machine Learning.

Índice

Índice de Figuras

Notación

ML	Machine Learning
IA	Inteligencia Artificial
DL	Deep Learning
$SVMs$	Support Vector Machines
SVC	Support Vector Classifier
SVR	Support Vector Regression
KNN	K-Nearest-Neighbor, K-Vecinos más cercanos
RNA	Red Neuronal Artificial
RNC	Red Neuronal Convolucional
TP	True Positive, Verdadero positivo
TN	True Negative, Verdadero negativo
FP	False Positive, Falso positivo
FN	False Negative, Falso negativo

1 Introducción

La necesidad de automatización de procesos, de asistentes para la toma de decisiones y de proporcionar predicciones de distintos problemas son las bases sobre las que se sustenta el presente trabajo. Cada vez más, aparece la necesidad de disminuir el número de fallos de sistemas y el factor humano, y es por ello que la detección de estos errores con la suficiente antelación supone una gran ventaja en todos los sentidos; tanto económico, de seguridad como por supuesto de salud. Esto se lleva a cabo con el famoso "Big Data" y la Inteligencia Artificial. Se comenzará con el estudio del estado del arte de la Inteligencia Artificial, más concretamente de la rama de Machine Learning. Dentro del apartado se hablará de cómo estas tecnologías están aplicándose en la industria aeronáutica y se comentarán alguno de sus ejemplos. El siguiente punto será el estado del arte en el ámbito de la medicina. Se tratarán ambos sectores debido a que son dos de los que tienen un contacto más cercano con la vida de personas y el factor humano juega un papel muy relevante. Posteriormente se hablará del Machine Learning y se explicarán las técnicas que se usan para resolver este tipo de problemas. Se dividirá el estudio en algoritmos de aprendizaje supervisado [24] y no supervisado, y el primero se dividirá en algoritmos de regresión y de clasificación. Dentro de este apartado también se hablará del Deep Learning o Aprendizaje profundo [23] que son algoritmos que simulan el funcionamiento del cerebro humano. Seguidamente se explicará cómo aplicar la teoría mediante el uso del lenguaje de programación *Python* explicando cuáles son las librerías necesarias para resolver estos problemas.

Para poner estos conocimientos en práctica, se han tenido que realizar búsquedas de datos y pese a hacerlo en primera instancia con bases de datos del sector aeronáutico, no se ha podido encontrar nada que fuese fiable. Esto se debe principalmente a que la mayoría de empresas del sector aún están comenzando a aplicar este tipo de tecnologías y no tienen una cantidad de datos suficientes o si alguna ya lo tiene, no está disponible. Otra opción que se planteó fue hacer uso de algún simulador de vuelo, pero finalmente se concluyó que esto no era una buena opción debido a la inexperiencia para usar el software. La consecuencia que esto tendría es que no se podría simular lo que un piloto haría en distintas situaciones y lo que interesa es obtener datos de pilotos experimentados para que así la máquina aprenda de las distintas situaciones que ocurren durante un vuelo y cómo actúa el piloto. Por lo que si lo hace una persona sin experiencia el avión no volará de forma correcta y por tanto no serían válidos los datos obtenidos.

Aunque la metodología que se va a emplear es aplicable en cualquier sector (ya que se trata de aplicar algoritmia a un conjunto de datos), debido a la estrecha relación que guardan los problemas del sector sanitario con aviación en la toma de decisiones en situaciones con elevado estrés, y debido a que se han encontrado datos de mayor veracidad, se ha decidido que los casos prácticos estén orientados a la salud. El primero ellos se tratará de un proceso de clasificación de pacientes según el estado de gravedad tras sufrir una amputación, para que de esta forma se pueda aplicar el protocolo de asistencia que más favorezca a su estado de salud.

Seguidamente se determinará la probabilidad que un paciente con unas determinadas características padezca o no cáncer de mama. Como último caso, se estudiará el análisis de lesiones cutáneas mediante el empleo de técnicas de tratamiento de imágenes.

Para finalizar, se analizarán las distintas conclusiones obtenidas a lo largo del trabajo, así como las líneas futuras en las que se podría continuar trabajando.

2 Estado del Arte Inteligencia Artificial

La inteligencia artificial definida como la inteligencia de las máquinas, está siendo el pilar fundamental del progreso tecnológico de la industria e incluso de nuestro día a día, y esto viene impulsado por un uso eficiente de los datos. Además, a día de hoy no es necesario almacenar estos datos en un lugar físico debido al avance en las tecnologías *cloud* que permiten acelerar la evolución tecnológica. El objetivo principal que tiene esta *Ciencia del Dato* es a partir de algoritmos poder tomar decisiones, lo que se podría llamar ciencia de decisión. A continuación, se hablará de dos industrias en las que estas técnicas están presentes.

2.1 Industria Aeroespacial

La aplicación de técnicas de inteligencia artificial en el sector aeroespacial es algo inminente pese a que como anteriormente se ha comentado, las empresas lo están llevando a cabo desde hace muy poco tiempo. Todos los sensores que tiene cada avión generan una gran cantidad de datos por segundo lo cual permite generar una gran base de datos para posteriormente utilizarla aplicando algoritmos predictivos. Con esta información se podría determinar con una alta precisión el momento en el que un sensor dejase de funcionar. Otro objetivo fundamental que tendrían estas técnicas, sería la determinación de mantenimiento de distintos sistemas, subsistemas o elementos. De esta manera se reducirían los costes ya que se tendría conocimiento de la probabilidad que tiene un determinado sistema de fallar y permitiría a los operarios actuar en consecuencia. Empresas como Rolls Royce [26] utilizan este tipo de técnicas para aprovechar los datos que genera el motor en funcionamiento para que si en algún momento se detecta algo que está fuera de lo común, se pueda actuar con la mayor rapidez posible. Otro ejemplo sin duda muy relacionado con lo que posteriormente se verá en el caso de medicina, es la capacidad de eliminar situaciones que puedan suponer al piloto un elevado estrés. Igual en un futuro se verán aviones volando sin piloto ya que será todo manejado por un supercomputador, pero hasta que llegue ese momento (si finalmente ocurre), se tiene que buscar una solución al problema de que una gran parte de los accidentes se deben al fallo humano. Esto puede ser debido a que psicológicamente haya días en los que no se realice el checklist por completo, o que no se tengan los mismos reflejos, y es por eso que un gran aporte sería la implementación de un asistente que permitiese eliminar esta carga cognitiva al piloto para facilitar así su labor. Un ejemplo de este tipo de proyecto es Harvis, en el cual participa la empresa sevillana Skylife Engineering [30]. Este proyecto europeo consiste en establecer un "single pilot", es decir, que en la cabina haya un único piloto y una computadora que mediante el aprendizaje a través de numerosos vuelos, aprendiendo de las acciones que realza el piloto, pueda asistir definiendo cuál es la acción más favorable a realizar. En definitiva, se trata de un asistente que ha aprendido lo que un piloto hace en las distintas fases de un vuelo gracias a que ha sido entrenado por pilotos con gran experiencia, y de esta forma en cualquier situación podrá dar indicaciones acerca de la tarea que se tiene que realizar. Otra aplicación que tendrá un gran uso en el futuro debido al aumento en el número de vuelos que se tiene previsto, será la optimización de las trayectorias [11]. Esto es de vital importancia para los momentos en los que haya una elevada congestión del tráfico ya que el algoritmo optimizará la ruta de los distintos vehículos evitando posibles accidentes.

2.2 Medicina

En el caso de Medicina la mayor aplicación que tiene es en medicina predictiva. Para que se pueda predecir si un paciente con ciertas características tiene una determinada enfermedad, es necesario contar con un gran volumen de datos de registros que se tengan de pacientes previos de los cuales se conozcan una serie de características y se haya obtenido un diagnóstico en función de estas entradas. Para que esto sea posible es fundamental llevar a cabo un proceso de digitalización. De esta manera, el hospital o centro de salud tendrá registrado a todos los pacientes de modo que estén presentes en formato digital y así el sistema pueda alimentarse de esta información y poder llevar a cabo un algoritmo predictivo que sea eficaz. En definitiva, la máquina aprenderá de los diagnósticos que hacen los médicos para que cuando se hayan registrado una cantidad suficiente de pacientes, el algoritmo actúe con una tasa de acierto elevada o lo que es lo mismo, que tome la misma decisión que hubiese tomado un especialista.

En este trabajo se presentarán una serie de casos en los que esto se pone en práctica. Un punto muy positivo es que dado que el algoritmo lo que lee son características de los pacientes (bien sean numéricas o imágenes), se podrá adelantar a la aparición de esa enfermedad y así actuar de manera que, al hacerlo previo a la aparición de forma grave del problema, la recuperación sea más rápida y el paciente corra el menor riesgo posible.
Otro punto a favor de esto es para aquellos médicos que hayan terminado sus estudios y comiencen a trabajar. La experiencia que el algoritmo tiene tras miles o decenas de miles de pacientes (cuantos más mejor) ayudarán en cuanto a la toma de decisión de un diagnóstico más certero y un tratamiento correcto.

Se ha hablado ya de prevención de enfermedades, lo cual es muy interesante por las razones que se han expuesto. Otra aplicación que tiene la inteligencia artificial en medicina es la asignación de un tratamiento a un paciente [20]. Un paciente con unos determinados síntomas tendrá que ser recetado con una medicación. Si se tuviese el registro de los pacientes a los que se les ha recetado una medicación dada una serie de características presentadas, se almacenase haciendo uso de las tecnologías Big Data y se aplicase el algoritmo predictivo de Machine Learning más adecuado, se conocería el medicamento que el paciente necesita, así como las dosis que serían adecuadas. Se trataría de un proceso de automatización y toma de decisiones que ayudaría a trabajar a los médicos. Todo esto concluye a que se prevé un futuro en el que el objetivo será desarrollar aplicaciones centradas en la salud del paciente y poder así tener una atención médica más personalizada y efectiva.

El objetivo de la medicina de precisión [22] es seleccionar tratamientos que tengan más probabilidad de ayudar a pacientes de acuerdo a su genética. Esto se conoce también como medicina personalizada. El problema que existe a día de hoy es que una persona a la que se le diagnostica un cáncer, es tratada del mismo modo que otra persona que presenta el mismo tipo y estadio de cáncer. Esto es un problema debido a que cada individuo responde de forma diferente. Los pacientes presentan variaciones genéticos que hace que la respuesta al tratamiento sea distinta. El objetivo principal de la medicina de precisión es que con las pruebas genéticas que se realicen y mediante un algoritmo predictivo, se pueda determinar qué tratamiento es el que tiene una mayor probabilidad de curar un tumor.

3 Machine Learning

Se define la modelización predictiva como un conjunto de algoritmos de la rama de la estadística que cuando se aplican a datos históricos es capaz de devolver una función matemática que tiene que ser útil para resolver un determinado problema [12]. En definitiva, se busca predecir un resultado basándose en una serie de parámetros de entrada sobre los que nuestro modelo opera y predice de modo que en el futuro cuando se tengan nuevos datos de entrada, la salida del algoritmo permita tomar decisiones.

La estadística juega un papel muy relevante; dice el volumen de los datos, cómo están distribuidos (si hay un valor central o si en cambio los datos están dispersos), si existe correlación entre variables o si son independientes. En general la estadística ayuda a responder a estas y muchas más cuestiones. Los algoritmos generarán ecuaciones matemáticas basados en datos históricos que ayudarán a predecir datos a posteriori. Se pueden clasificar en dos grandes grupos.

3.1 Algoritmos de aprendizaje supervisado

Los algoritmos de aprendizaje supervisado trabajan con datos etiquetados, intentando encontrar una función que, dadas las variables de entrada les asignen la etiqueta de salida adecuada. Dicho algoritmo se entrena con un "histórico" de datos y así "aprende" a asignar la etiqueta de salida adecuada a un nuevo valor, es decir, predice el valor de salida. De los algoritmos que se verán a continuación, algunos se usarán para clasificación, otros para regresión y algunos para ambos.

En los problemas de clasificación se tratará de asignar la nueva muestra en una clase. Por ejemplo, un reloj inteligente cuando clasifica la actividad que la persona está realizando tendría que escoger entre andar, correr, montar en bicicleta... Otro ejemplo podría ser si la acción de una determinada empresa subirá o no.

Por otro lado, en los algoritmos de regresión la salida es un valor numérico. Algunos ejemplos podrían ser en cuánto está valorada una propiedad, determinar el número de ventas que una panadería va a tener en un día o el tiempo que un cliente va a permanecer en una empresa, son algunas de las aplicaciones que tienen estos algoritmos.

- **Regresión**: Regresión Lineal, SVMs, Árbol de decisión, Redes neuronales.
- **Clasificación**: Regresión Logística, SVMs, Árbol de decisión, Redes neuronales.

Regresión lineal

Se define como la modelización de una función que relaciona de forma lineal una serie de variables de entrada con la variable de salida. Si solo se tuviese una variable de entrada, se estaría hablando de regresión lineal simple y si es más de una variable de entrada, se define como regresión lineal múltiple.

En la regresión lineal las relaciones de cada variable con la salida se modela utilizando funciones de predicción lineal con parámetros que se estiman a partir de los datos históricos. Es el más simple.

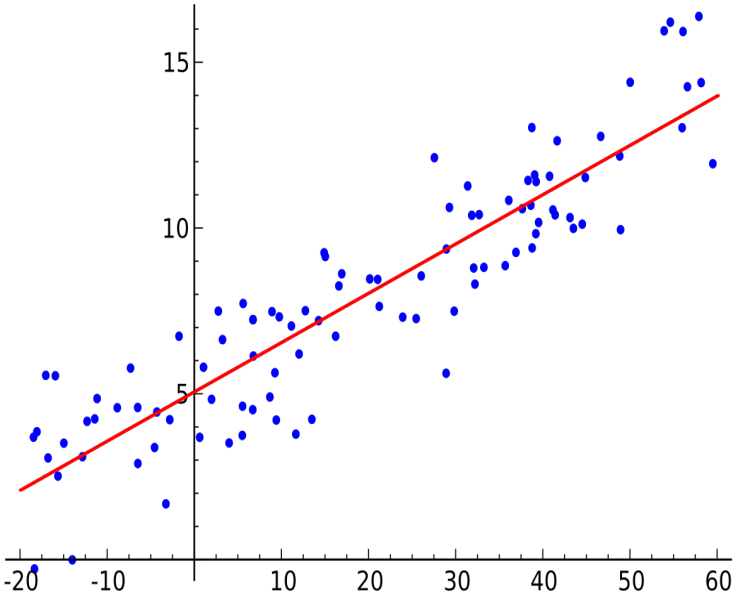

Figura 3.1 Modelo de regresión lineal [8].

Para ajustar y mejorar los modelos de regresión lineal se suele aplicar el método de los mínimos cuadrados.

- Y es la variable dependiente, la que se busca predecir.
- X_1,X_2,...,X_m son las variables independientes, serían las columnas del data set.
- β_0,β_1,β_2,...,β_p son los parámetros del modelo e indican el peso o importancia que tiene cada variable X en nuestra función.

$$y_i = \beta_0 + \beta_1 x_{i1} + ... + \beta_p x_{ip} + \varepsilon_i, \; para \;\; i = 1,...,n \tag{3.1}$$

Agrupando las x_i y β en un vector, nos queda la expresión siguiente:

$$y_i = {x_i}^T \beta + \varepsilon_i \tag{3.2}$$

Si trabajamos para cada i, podemos definir un modelo genérico como sigue:

$$y = X\beta + \varepsilon \tag{3.3}$$

Donde podemos definir:

$$y = \begin{pmatrix} y_1 \\ y_2 \\ ... \\ y_n \end{pmatrix} \tag{3.4}$$

$$X = \begin{pmatrix} {x_1}^T \\ {x_2}^T \\ ... \\ {x_n}^T \end{pmatrix} \tag{3.5}$$

$$\beta = \begin{pmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ \dots \\ \beta_p \end{pmatrix}, \varepsilon = \begin{pmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \dots \\ \varepsilon_n \end{pmatrix} \tag{3.6}$$

Se puede observar cómo dado un dataset, la regresión lineal asume que la relación entre la variable dependiente y y la variable independientes x es lineal. Esta relación se modela con un error ε el cual añade ruido a dicha relación.

Este modelo no siempre es útil debido a que puede que la salida no se ajuste correctamente con una relación lineal por lo que otro método sería más preciso.

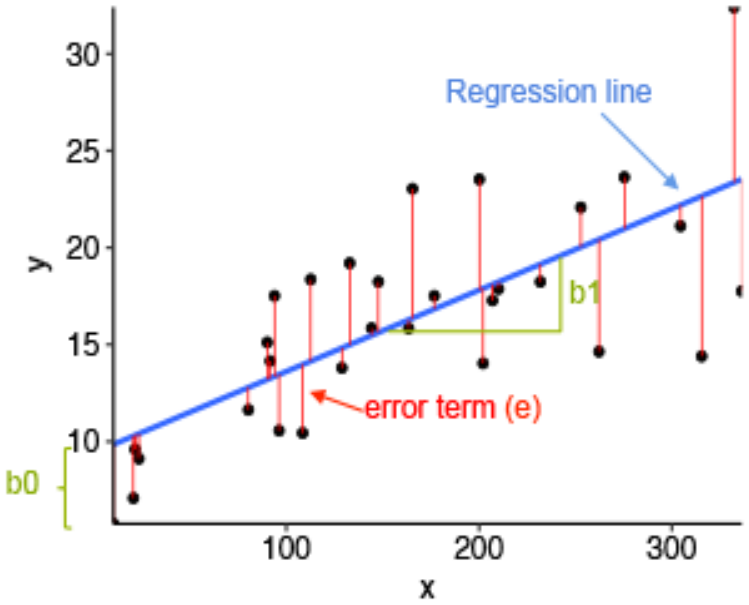

Figura 3.2 Visualización de errores regresión lineal [5].

Regresión logística

Se define como el modelo que permite determinar la probabilidad que hay de que un suceso ocurra o no, pase/no pase, gane/pierde. De forma básica, se puede entender como regresión logística al modelo estadístico que sirve para modelar una variable dependiente binaria. El modelo logístico binario tiene una variable dependiente con dos valores posibles que está representada por una variable que tiene valores ”0” o ”1”. La salida del algoritmo será la probabilidad entre ”0” y ”1”, y en el caso que se obtenga que es ”0”, se podrá asegurar que dicha salida es con total seguridad la ”0”. Como su propio nombre indica, la función que define esta salida es la logística.

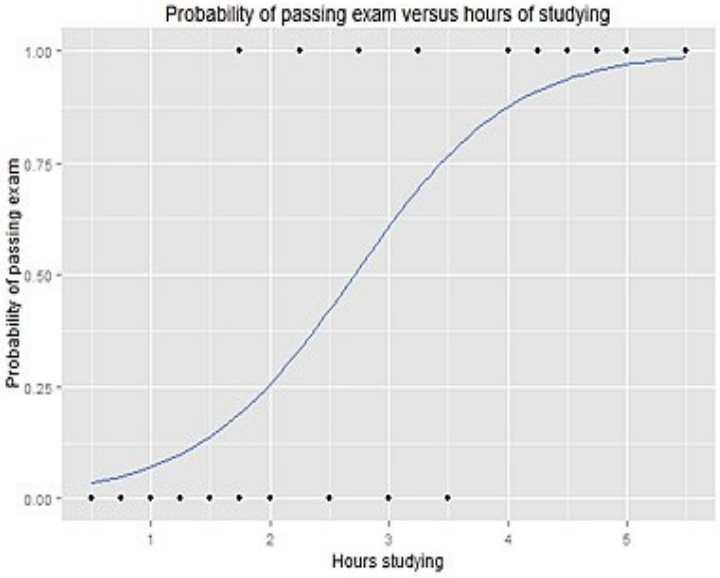

Figura 3.3 Regresión logística para determinar la probabilidad de aprobar un examen en función de las horas de estudio [9].

Se puede observar como aumenta la probabilidad de aprobar el examen a medida que aumentan las horas de estudio, y se puede también apreciar que existe un tramo donde la función evoluciona de forma prácticamente lineal.

La función que se usa para definir el modelo es una función sigmoidea la cual tiene una entrada que se define como t. La función logística estándar se define como sigue:

$$\sigma(t) = \frac{e^t}{e^t + 1} = \frac{1}{1 + e^{-t}} \tag{3.7}$$

Asumiendo a continuación que t es una función lineal definida por la variable x,

$$t = \beta_0 + \beta_1 x \tag{3.8}$$

Por lo que redefiniendo la ecuación (2.7) se obtiene que:

$$p(x) = \sigma(t) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x)}} \tag{3.9}$$

Siendo $p(x)$ la probabilidad que toma la salida.

Support Vector Machines (SVMs)

El algoritmo SVM es uno de los métodos de predicción más robustos. El algoritmo se encarga de construir un hiperplano o un conjunto de hiperplanos en un espacio con una dimensión elevada y puede usarse tanto para clasificar, como una regresión o para determinar *outliers* o puntos atípicos. Al igual que en los casos anteriores, a partir de datos históricos se prepara el modelo para poder utilizarlo en el futuro.

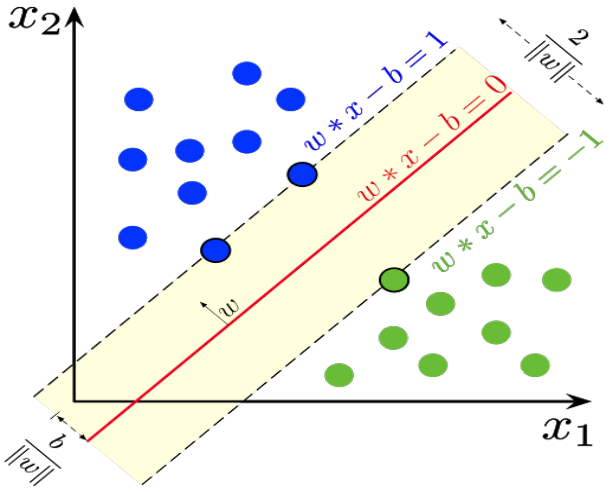

Figura 3.4 Ejemplo Support Vector Machine lineal [10].

Dada la imagen anterior, un algoritmo SVM ayudaría a predecir a qué grupo pertenece el nuevo punto que se tiene que clasificar. Los puntos están representados en dos dimensiones, entonces se va a buscar un plano que esté en $p-1$ dimensión que sería dimensión 1 lo que sería una recta, que separe a los conjuntos de puntos. El ancho del hiperplano se define como la mayor distancia que existe al punto de entrenamiento más cercano de cualquier clase (es el llamado margen funcional). El mejor hiperplano será aquel que consiga la mayor separación entre las dos clases. En este caso los puntos son linealmente separables. Matemáticamente para el caso donde la separación se pueda establecer de forma lineal, cualquier punto x que esté en el hiperplano separador, satisface la siguiente ecuación:

$$x^T w + b = 0 \tag{3.10}$$

Se define:

- w es un vector perpendicular a la línea central.
- b desplaza el hiperplano respecto del origen.
- La distancia del hiperplano al origen se define como $\frac{b}{\|w\|}$

Si se definen los puntos azules como $y_i = 1$ y los puntos de color verde como $y_i = -1$, se tiene que:

$$x_i^T w + b \geq +a, y_i = +1 \tag{3.11}$$

$$x_i^T w + b \leq +a, y_i = -1 \tag{3.12}$$

$$y_i(x_i^T w + b) \leq a \tag{3.13}$$

Por lo que ahora se puede determinar la anchura del canal de la siguiente forma:

$$M = d_+ + d_- = \frac{2a}{\|w\|} \tag{3.14}$$

Se fija $a = 1$ para maximizar dicho corredor.
El objetivo será maximizar el valor de M.

Árbol de decisión

El aprendizaje por decisión de árbol es un método comúnmente utilizado en la minería de datos. Se tiene como propósito crear un modelo que prediga la variable objetivo en función de las variables de entrada (igual que en los casos anteriores). En un árbol de decisión cada nodo interno está etiquetado con una característica o variable de entrada. Las uniones que provienen de cada nodo se etiquetan con los posibles valores que pueda tomar dicho nodo. La estructura básica de un árbol de decisión es como se indica en la figura,

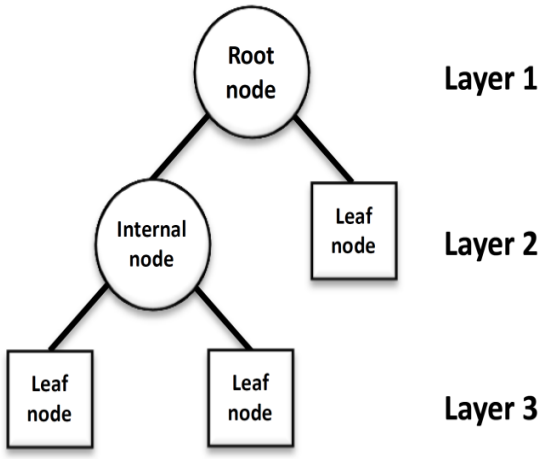

Figura 3.5 Estructura árbol de decisión [19].

Es decir, un conjunto de nodos, de ramas y de hojas.

- **Nodos raíz**: son los puntos a través de los cuales van saliendo las ramas a partir de las cuales se ramifica el árbol. Es el punto donde nace el árbol.
- **Nodo hoja**: es aquel nodo que no tiene tras él ninguna hoja que haga que siga creciendo el árbol. Aquí se lleva a cabo la decisión final por lo que es el punto donde se realizará la clasificación.

K-Vecinos más cercanos

El algoritmo de los K-Vecinos más cercanos o K-Nearest-Neighbor, es un algoritmo que puede utilizarse para clasificación o para predecir una salida continua (como lo hace por ejemplo la regresión lineal). "K" significa la cantidad de "puntos vecinos" que el algoritmo tiene en cuenta para clasificar al nuevo punto. Este método busca las observaciones más cercanas a la nueva que se está tratando de predecir.

Es un algoritmo basado en instancia, esto quiere decir que no aprende un modelo, sino que memoriza las instancias de entrenamiento que se usan para la fase de predicción. Luego al utilizar todo el dataset para entrenar cada uno de los puntos, requiere de mucha memoria. Es por ello que este método funciona de forma correcta si el dataset no es muy grande y no tiene demasiadas columnas. El funcionamiento se puede resumir a lo siguiente:

- Calcular la distancia entre el nuevo punto (a clasificar) y el resto de items del dataset de entrenamiento.
- Elegir los "k" elementos más cercanos que serán los que estén a menor distancia del nuevo punto. La forma más usual de medir estas distancias es mediante la distancia euclídea.
- Se clasificará de una forma u otra dependiendo de lo que la mayoría de los "k" vecinos digan.

Tras conocer los pasos que sigue el algoritmo, para la predicción es muy importante definir correctamente el valor de "k". Una opción coherente sería tomar los valores impares para que de esta forma no haya empate. Aunque tomar más puntos no requiere necesariamente mayor precisión, lo que está claro es que el coste computacional será mayor.

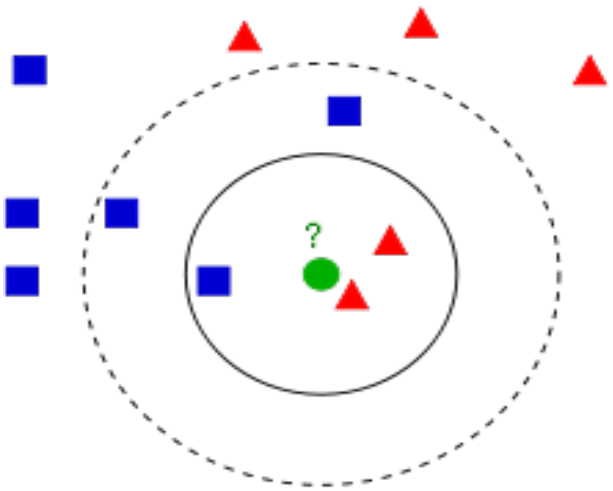

Figura 3.6 KNN para clasificación [7].

3.2 Algoritmos de aprendizaje no supervisado

Estos algoritmos trabajan sin variables de salida de los datos históricos. El aprendizaje no supervisado tiene lugar cuando no se dispone de datos "etiquetados" para el entrenamiento. Sólo se conocen los datos de entrada, pero no existen datos de salida que correspondan a un determinado input. Por tanto, sólo se puede describir la estructura de los datos para tratar así de encontrar algún tipo de organización que simplifique el análisis. Por ello, tienen un carácter exploratorio.

3.2.1 Clustering

Es un algoritmo que categoriza las entradas del dataset en clusters o segmentos donde las entradas que pertenecen a un mismo cluster son similares. En definitiva:

- Se busca agrupar los datos que presenten semejanzas entre los miembros del cluster (que sean parecidos).
- Los datos que pertenezcan a grupos diferentes, deben tener rasgos suficientemente diferentes entre sí.

Por ejemplo, a la hora de realizar campañas de marketing, si se tienen distintos cluster o grupos de clientes identificados se podrá realizar una campaña específica para cada uno de ellos. Existen diversos tipos de segmentación, y para ello se verán a continuación distintos modelos utilizados para medir la similitud y diferencias.

Distancia Euclídea

Como se ha dicho anteriormente, es fundamental conocer la similitud que hay entre los puntos de modo que esto permita clasificar el nuevo dato en un determinado grupo. Para poder realizar esta medición, se hace uso de la distancia y aunque hay distintas formas de medirla, se va a explicar la distancia euclídea. Siendo x las variables de entrada de cada punto de cada set, se define la distancia entre un punto y otro como sigue:

$$D(x_i,x_j) = \sqrt{(x_{i1}-x_{j1})^2+(x_{i2}-x_{j2})^2+...+(x_{in}-x_{jn})^2} = \sqrt{\sum_{k=1}^{n}(x_{ik}-x_{jk})^2} \tag{3.15}$$

Como se puede observar, n es el número de columnas del data set.

Clustering K-Means

Este algoritmo no supervisado [6] permite dividir un número de observaciones en determinados grupos. En principio, el número de grupos "k" será conocido, pero también se verá que existe una manera de optimizar el número de clusters o grupos que se conoce como el método del codo.

Se definen los centroides del cluster o grupo como el punto medio de la nube de puntos que lo define. Con respecto a este punto se medirá la distancia. El proceso que llevará a cabo el modelo es el siguiente:

$$SS_w(C_j) = \sum_{x\in C_j}(x-c_j)^2 \tag{3.16}$$

Una vez se incluye un nuevo punto al cluster, hay que volver a calcular el centroide debido a que el "centro de masas" habrá variado. Para ello se aplica lo siguiente:

$$\widetilde{SS_W} = \sum_{j=1}^{k}\frac{SS_W(C_j)}{\sum_{i=1}^{n}(x_i-\bar{x})^2} \tag{3.17}$$

Se repetirán sucesivamente estos dos últimos pasos hasta que ya no se pueda variar el centroide. El objetivo es reducir la distancia intracluster SS_W. La ecuación por tanto a minimizar será como sigue:

$$SS_W(k) = \sum_{j=1}^{k}SS_w(C_j) = \sum_{j=1}^{k}\sum_{x\in C_j}(x-c_j)^2 \tag{3.18}$$

Donde como se ha dicho antes:

- k : número de clusters.
- x_i: muestra *i-ésima*.

- c_j: centroide del cluster *j-ésimo*.

A continuación, se va a hablar de cómo encontrar el valor óptimo de k, que es una de las partes más difíciles a la hora de elaborar un algoritmo por K-Means. A continuación, se presentará el **método del codo**.

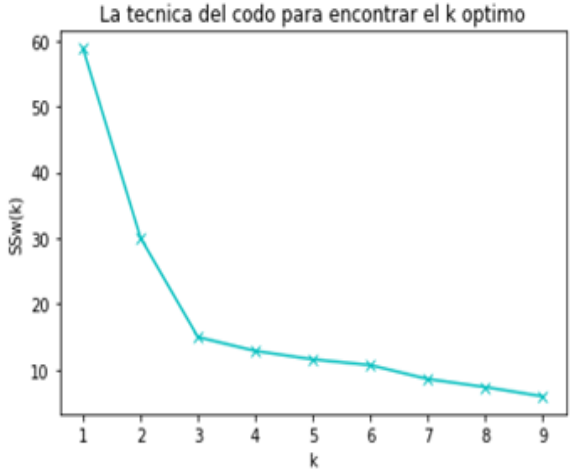

Figura 3.7 Método del codo. Optimización del número de clusters [25].

Es evidente que al aumentar el número de clusters, la distancia intra-cluster disminuirá paulatinamente. El caso extremo sería que se tuvieran tantos clusters como puntos en el dataset y por tanto la distancia sería nula. Esta técnica se denomina método del codo debido a que en casi todos los casos suele haber una variación máxima donde se observa que la curva tras ese punto decrece más lentamente que en los puntos anteriores y prácticamente se estabiliza la curva. En el caso de la imagen el punto en el que esto ocurre es en $k = 3$. En consecuencia, ese punto sería el que marcaría el número de clusters que se debería tomar.

3.3 Deep Learning

El pionero del aprendizaje profundo o red neuronal es Geoffrey Hinton. Fue el que comenzó a estudiar esta rama del Machine Learning y actualmente se encuentra trabajando en el departamento de Inteligencia Artificial de Google. Mucha de la información que se encontrará en adelante se fundamenta en estudios que realizó Geoffrey y se pueden consultar en sus *papers*.

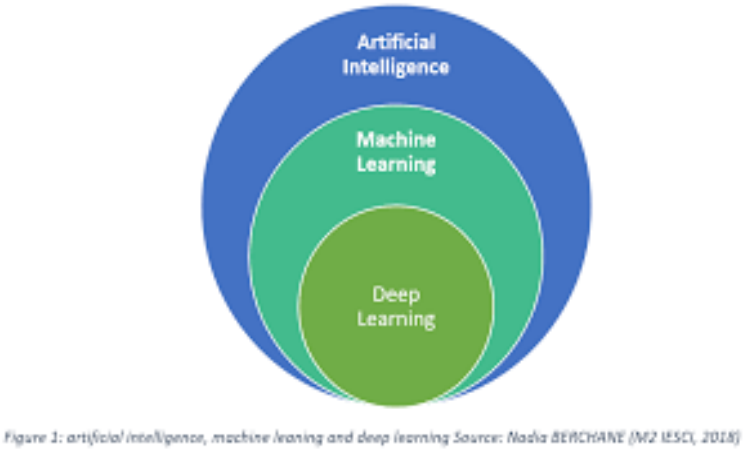

Figura 3.8 Situación de la red neuronal [13].

Se puede observar en la imagen anterior cómo el Deep Learning es una rama del Machine Learning, y éste a su vez una rama de la Inteligencia Artificial. La principal diferencia que existe entre los algoritmos de Machine Learning y Deep Learning es que estos últimos detectan por sí mismos las distintas características de los datos. En aquellos problemas de clasificación de imágenes, las redes neuronales son capaces de determinar cuáles son los rasgos más característicos que les permitirán realizar una clasificación con un elevado porcentaje de acierto. En cambio, al algoritmo de Machine Learning se le tendría que decir cuáles son estas características. Por ejemplo, en el caso de un vehículo, se le tendría que decir que tiene ruedas, puertas, espejos... Y sin embargo la red neuronal lo detectaría sin necesidad de que se le indicase. Si fuese un problema con variables numéricas o categóricas, la red neuronal se encargaría de dar más o menos importancia a las variables (variando su peso) para así obtener el mejor modelo.

La idea detrás del aprendizaje profundo es simular al cerebro humano y hacer uso de la neurociencia para establecer dichas leyes que rigen dicho funcionamiento. La tendencia que tienen las neuronas en el cerebro humano es conectarse entre ellas, ya que por sí mismas no tienen utilidad. En un cerebro humano existen alrededor de 100.000 millones de neuronas juntas.

La arquitectura típica de red neuronal es como sigue:

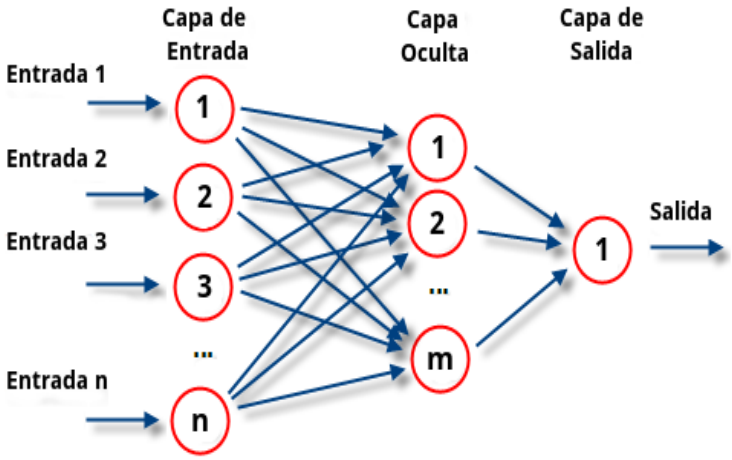

Figura 3.9 Arquitectura red neuronal [4].

- Capa de entrada: son los inputs o variables de entrada.
- Salida: es la predicción que se quiere realizar.
- Capas ocultas: nodos intermedios que son las neuronas que conecta y transmite la información de la entrada a la salida. Puede haber más de una capa oculta. Aquí es donde se realiza el aprendizaje.

3.3.1 RNA: Red Neuronal Artificial

Ya se ha definido anteriormente la arquitectura de la red neuronal. Ahora se va a proceder a explicar la red neuronal artificial, la cual es un algoritmo de Machine Learning supervisado. Un detalle a tener en cuenta es que, en la capa de entrada, los distintos inputs que se tengan deben estar todos en el mismo rango para que el algoritmo no tenga preferencia por ninguna de estas variables. Esto se puede llevar a cabo mediante la estandarización.

El valor de salida puede ser tanto un valor continuo como un valor categórico dependiendo del problema que se esté estudiando. En el caso de que sea categórico, se tendrán distintos valores de salida y cada uno de ellos será la categoría que se está tratando de asignar.

- **Primer paso**: la neurona va a ponderar cada variable de entrada mediante la aplicación de distintos pesos a cada una de ellas:

$$\sum_{i=1}^{m} w_i x_i \tag{3.19}$$

  La red neuronal aprenderá ajustando los distintos pesos.

- **Segundo paso**: una vez se tenga la suma ponderada se aplicará una función de activación que servirá para saber si la información de esa neurona se tiene que transmitir o en su defecto (al no ser muy buena), que esta información no se transmita a la siguiente capa. Dependerá en algunos casos de la función de activación si la neurona transmite o no dicha información.
- **Tercer paso**: la transmisión de dicha información.

El poder que tiene la red neuronal es que cada neurona entenderá los datos a su modo. La sinapsis, que es la línea que unen datos con neuronas, permite transmitir la información que realmente interese a la neurona.

La función de activación

La función de activación decide si la información obtenida por la neurona tras realizar la ponderación tiene que ser transmitida o no. Existen distintas funciones de activación y a continuación se hablará de cuatro de ellas.

Función escalón

La función escalón crea un salto en torno al valor 0 de forma que si al hacer el sumatorio de la multiplicación de los pesos de la red neuronal por cada una de las variables sale negativo, la función de activación tomará el valor nulo. En cambio, si el sumatorio es positivo, la función escalón tomará el valor unitario.

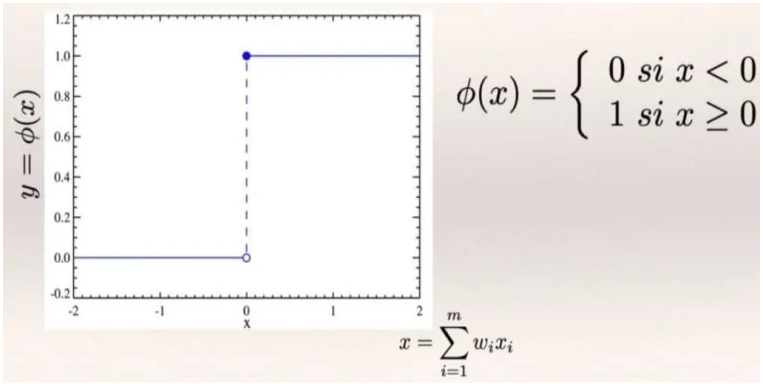

Figura 3.10 Función de activación escalón [18].

Función sigmoide

La función que define dicha activación viene dada como sigue:

$$\phi(x) = \frac{1}{1+e^{-x}} \tag{3.20}$$

$$x = \sum_{i=1}^{m} w_i x_i \tag{3.21}$$

Se puede observar que para el valor del sumatorio nulo, la función de activación toma el valor $\phi = 0.5$. Con esta función se puede definir la probabilidad que tiene una neurona de activarse. Además, la fórmula es igual que la de la regresión logística.

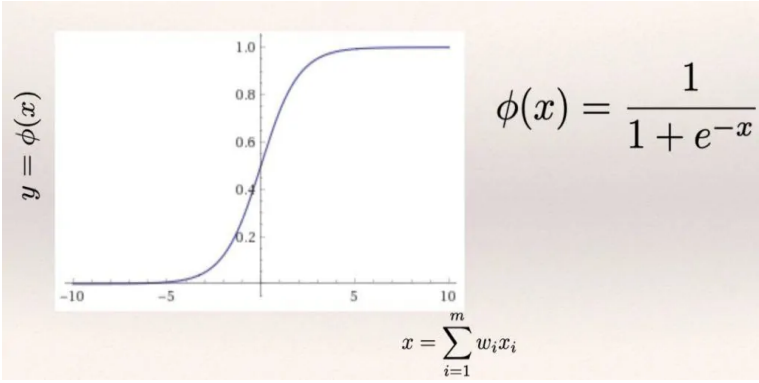

Figura 3.11 Función de activación sigmoidea [18].

Función rectificadora

En el rectificador lineal unitario (RELU). Se puede observar que la mitad de la función es totalmente nula mientras que la otra mitad crece de manera lineal. Es una de las funciones más populares para redes neuronales artificiales ya que al igual que ocurría en la función escalón, todo aquello que sea negativo lo convierte en nulo. Por otro lado, todo lo positivo se queda tal cual queda el resultado de la ponderación, y esa es la razón por la cual el valor de ϕ varía entre el valor nulo y el máximo de $\sum_{i=1}^{m} w_i x_i$.

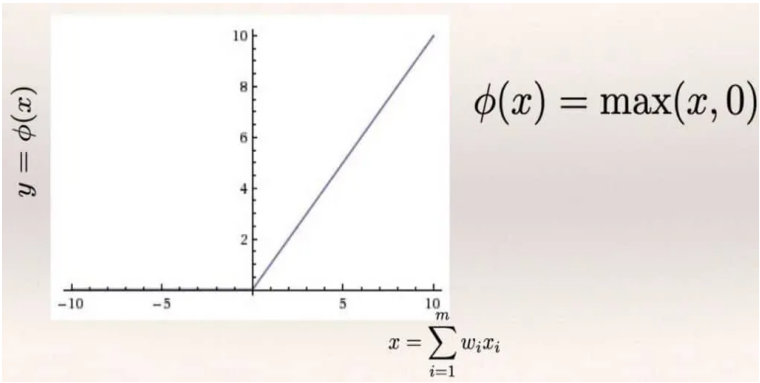

Figura 3.12 Función de activación rectificador [18].

Función tangente hiperbólica

Tiene una forma muy similar a la sigmoidea pero esta función comienza en -1 y llega hasta el valor positivo unitario.

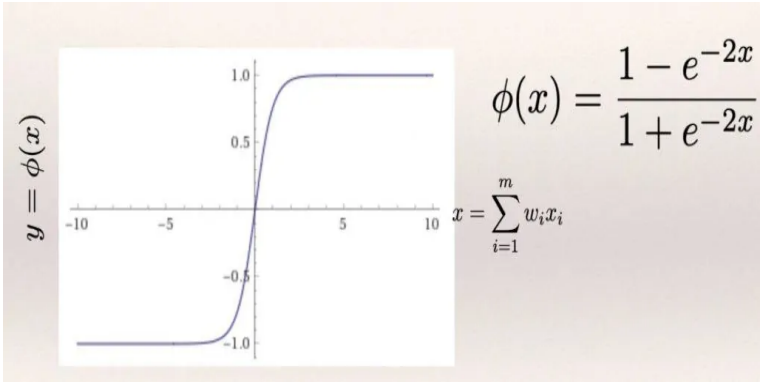

Figura 3.13 Función de activación tangente hiperbólica [18].

Función softmax

La función softmax también conocida como la función exponencial normalizada, es una generalización de la función logística usada para múltiples dimensiones. Se utiliza normalmente como la última función de activación en una red neuronal para así normalizar la salida de la red para que se obtenga la distribución de probabilidad. De esta forma la salida de esta función estará siempre dentro del intervalo $(0,1)$. Se define de la siguiente forma:

$$\sigma(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \tag{3.22}$$

Se da para $i = 1,...,K$ y $z = (z_1,...,z_K)$ en el intervalo $\in \mathbb{R}^K$

Cabe destacar también que cada neurona puede tener una función de activación diferente y de esta forma filtrar la información a su manera. Es bastante usual utilizar en la capa de salida la función sigmoidea debido a que como se ha explicado anteriormente, esta función permitiría conocer qué tan probable es que la salida pertenezca a una u otra clase o qué tan probable es que la salida tome un determinado valor.

Método de aprendizaje de las redes neuronales

Una vez ejecutada la red neuronal y obtenido el valor de salida, $\hat{y}$ se comparará con el resultado real y. Se define la función de costes como sigue:

$$C = \frac{1}{2}(\hat{y} - y)^2 \tag{3.23}$$

El objetivo será minimizar dicha función de costes. Para conseguirlo, lo único que la red neuronal puede hacer es variar el valor de los pesos que multiplica a cada variable. Se trata de un trabajo de ida y vuelta constante corrigiendo los pesos.

Gradiente descendente

Ya se ha adelantado anteriormente que el objetivo que tiene toda red neuronal es disminuir el valor de la función de costes. Es por ello que se va a hablar ahora del gradiente descendente. Lo primero y más importante, para que la solución converja, la función de costes tiene que ser convexa. Se busca el mínimo global de la función.

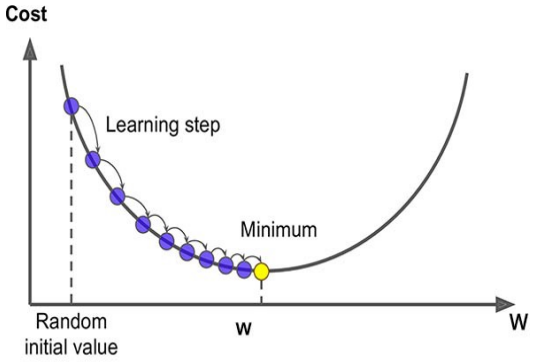

Figura 3.14 Reducción del valor de la función de coste tras realizar iteraciones [28].

El gradiente se define como la tangente a la curva y se calcula mediante la derivada. Si la pendiente es nula se estaría en el mínimo, luego ya habría convergido el algoritmo. En aquellas situaciones en las que la función de costes no sea convexa y existan fluctuaciones es más aconsejable hacer uso del gradiente estocástico. En este caso se busca el mínimo de la función de costes, pero no necesariamente será el mínimo global.

Entrenamiento Red Neuronal Artificial

- **Paso 1**: inicializar los pesos de las variables de entrada de forma aleatoria con valores próximos a 0.
- **Paso 2**: introducir el primer punto del dataset en la capa de entrada. Como se ha dicho anteriormente, cada nodo de entrada es una de las columnas del dataset.
- **Paso 3**: se propaga de izquierda a derecha. Las neuronas se irán activando de acuerdo a las funciones de activación y se obtendrá la predicción.
- **Paso 4**: se compara la predicción obtenida con el resultado real. Se mide el error.
- **Paso 5**: se propaga el error hacia atrás para que de esta forma se puedan actualizar los pesos.
- **Paso 6**: se repiten los pasos del 1 a 5 para cada una de las observaciones del dataset y se actualizan los pesos.
- **Paso 7**: una vez se ha barrido todo el dataset, se vuelve a repetir el proceso el número de veces que más interese mediante la variación del número de *epochs*.

3.3.2 Red Neuronal Convolucional

Los algoritmos de redes neuronales de convolución [2] son los utilizados por excelencia para el tratamiento de imágenes ya que son capaces de identificar las características más importantes que la definen sin necesidad de que se indique. Se suministrará una imagen y el algoritmo dará la salida que nos indique a qué categoría pertenece la imagen. La red neuronal convolucional tratará de obtener rasgos de las imágenes. Un ejemplo muy sencillo sería enviar imágenes categorizadas como gatos y perros de tal manera que cada imagen tiene una etiqueta asignada. Tras tratar con gran cantidad de ejemplos de cada una de estas clases, la RNC habrá identificado los rasgos más característicos de cada una de las dos especies de modo que cuando se le pase una nueva imagen ya sea de un perro o de un gato, ésta sea capaz de clasificarla correctamente.

Lo primero que hay que realizar antes de suministrar datos a la red, es normalizar los valores. Los píxeles toman valores de 0 a 255 por lo que se transformará cada pixel dividiendo el valor de cada uno entre 255 y de esta forma se tendrá todo entre 0 y 1. Si la imagen estuviese en blanco y negro habría dos canales, y si es a color habría tres canales rojo, verde y azul. A continuación, se explican los pasos que se siguen para construir la red neuronal convolucional:

- **Paso 1: Convolución:** en este paso se llevan a cabo las convoluciones. Se trata de seleccionar grupos de píxeles de la imagen de entrada y realizar productos escalares con una matriz que es el llamado *kernel*. Dicho *kernel* recorre todas las neuronas de entrada y genera una nueva matriz a la salida que será la nueva capa de neuronas. Realmente se aplican más de un *kernel*, es decir se tendrán un conjunto de filtros en cada capa. Se tendrán entonces tantas matrices como filtros haya. Por lo tanto, el número de neuronas por cada capa vendrá dado por el número de *kernels* multiplicado por la dimensión de la imagen (si es 28x28, el número de *kernels* multiplicado por 784). En definitiva, estos filtros lo que buscan es identificar características de la imagen original. Se aplica ahora la función de activación, normalmente se utiliza para estas capas el rectificador lineal unitario.

- **Paso 2: Max Pooling:** se encarga de reducir la dimensión del problema. Tomando las matrices definidas anteriormente por los filtros, reducen su tamaño. Por ejemplo, si se tiene un Max Pooling de 2x2, se seleccionan 4 pixeles de la imagen y solamente se pone el que tiene el valor más alto. Se reduce mucho el número de neuronas y se supone que se sigue almacenando la información que define las características de las imágenes.

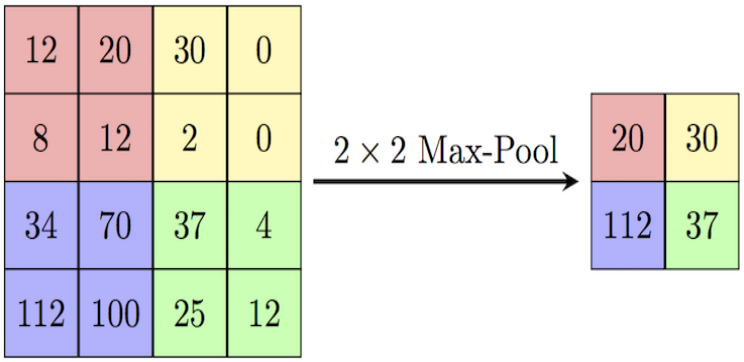

Figura 3.15 MaxPooling [1].

Este proceso se aplicará tantas veces como capas se quieran.

- **Paso 3: Flattening:** tras haber realizado las distintas fases de Max Pooling, para unirse con la red neuronal se necesita que se tenga como variables de entrada valores escalares. Esta capa se encarga de transformar la última matriz obtenida en el Max Pooling en un vector cuyas componentes serán las entradas en la red neuronal artificial.

- **Paso 4: Full Connection:** entrada de una red neuronal.

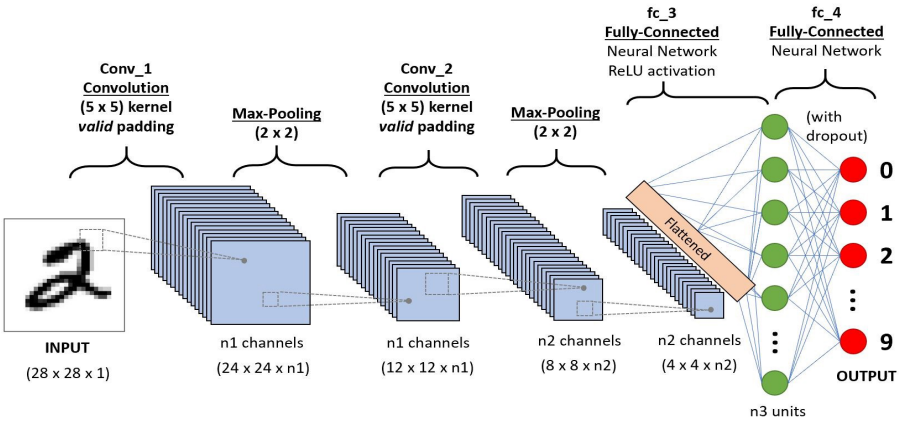

Figura 3.16 Arquitectura de la red neuronal convolucional [27].

3.4 Determinar efectividad del modelo

Existen distintas formas de determinar lo bien que se comporta un modelo y determinar si cumple o no con las expectativas de un modelo robusto y fiable. Dependiendo del tipo de problema que se esté abordando (si es un problema supervisado, no supervisado, de regresión, clasificación...) se usará un método de evaluación u otro. Antes de comenzar con los modelos se define:

- Verdadero positivo (True Positivo, TP): predicción como positivo y era positivo.
- Verdadero negativo (True Negative, TN): predicción como negativo y era negativo.
- Falso positivo (False Positive, FP): predicción como positivo y era negativo.
- Falso negativo (False Negative, FN): predicción como negativo y era positivo.

3.4.1 Matriz de confusión

La matriz de confusión se utiliza en problemas de clasificación (machine learning supervisado: agrupación en clases). Permite visualizar el número de veces que el algoritmo ha clasificado correctamente o por el contrario aquellas situaciones en las que el algoritmo ha clasificado en una clase distinta a la que debería haberlo hecho. A continuación, se muestra un ejemplo:

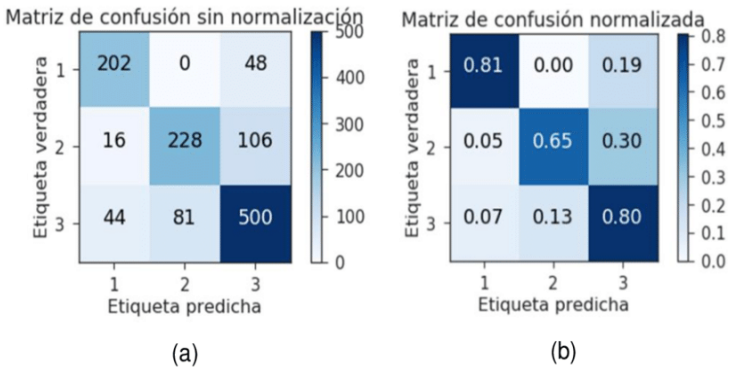

Figura 3.17 Ejemplo matriz de confusión [17].

La matriz de confusión es siempre cuadrada debido a que las variables que están en las filas y en las columnas son las mismas. Aquellos elementos que se sitúen en la diagonal son los que el algoritmo ha clasificado de forma correcta. Si se observa una columna, para aquellos elementos que estén en una posición distinta a la diagonal, son el número de veces que el algoritmo ha clasificado una muestra en una clase que no era realmente la suya.

3.4.2 Exactitud

Se define la exactitud como el porcentaje de aciertos del algoritmo. Esto es el cociente entre los aciertos y la suma de aciertos y fallos (el total de las predicciones).

$$Exactitud = \frac{TP+TN}{TP+TN+FP+FN} \tag{3.24}$$

Este tipo de evaluación del modelo para aquellos casos en los que el problema no sea balanceado (haya alguna de las clases dominantes) no se comportará correctamente. Por ejemplo, determinar cuando fallará un sensor de un avión se trata de un problema no balanceado debido a que igual el fallo se produce una vez cada 100.000 datos tomados. Entonces en este caso la precisión sería $\frac{99999}{100000}$ lo cual es una exactitud muy elevada pero el algoritmo no está resolviendo el problema que se está plantenado. Es por ello que la exactitud para algunos casos no se puede aplicar ya que no es un buen indicador.

Para este tipo de casos se podría balancear el modelo bien reduciendo el número de casos de la clase dominante, aumentando el del resto de clases o ambos. Si esto no tiene resultado, lo que se puede hacer es utilizar otra medida del rendimiento del modelo que sería sensibilidad (recall).

3.4.3 Precisión

Indica el porcentaje de aciertos de predicciones positivas, es por ello que será el cociente entre el número de verdaderos positivos y el número de predicciones hechas como positivos (suma de verdaderos y falsos positivos):

$$Precisión = \frac{TP}{TP+FP} \tag{3.25}$$

3.4.4 Sensibilidad

Representa el porcentaje de verdaderos positivos. Es decir, es el cociente entre los verdaderos positivos y la suma de todos los positivos que hay.

$$Sensibilidad = \frac{TP}{TP+FN} \tag{3.26}$$

3.4.5 Especifidad

Representa el porcentaje de verdaderos negativos. Es igual que la sensibilidad, pero para el caso de las predicciones negativas.

$$Especifidad = \frac{TN}{TN+FP} \tag{3.27}$$

3.4.6 F1 score

Es un porcentaje entre medias de la sensibilidad y la precisión, por lo que tiene en cuenta ambos valores. Es por ello que a medida que aumente el valor de $F1$ será mejor el modelo.

$$\frac{2}{\frac{1}{precisión}+\frac{1}{sensibilidad}} = \frac{2*precisión*sensibilidad}{precisión+sensibilidad} \tag{3.28}$$

De forma general, se le da la misma importancia a la precisión y la sensibilidad, pero se podrían cambiar los pesos que multiplican a cada uno de estos valores para cambiarlo. Puede ser útil para ciertos problemas donde interesen más los positivos o los negativos.

3.4.7 Curva ROC

Es una de las curvas más importantes que se utilizan para medir el rendimiento de un algoritmo de Machine Learning.

- ROC: características de funcionamiento del receptor.
- AUC: área bajo la curva.

Lo que mide la curva ROC es la capacidad que tiene el modelo de distinguir entre una clase u otra. Si se tuviese un problema cuyo objetivo es la clasificación de una muestra en 1 de 2 clases, se dirá que el modelo es bueno cuando es capaz de distinguir con precisión que una muestra pertenece a una o a otra clase. Se representa la $sensibilidad$ frente a $1-especificidad$.

El AUC proporciona una buena idea de lo bien que está funcionando el modelo.

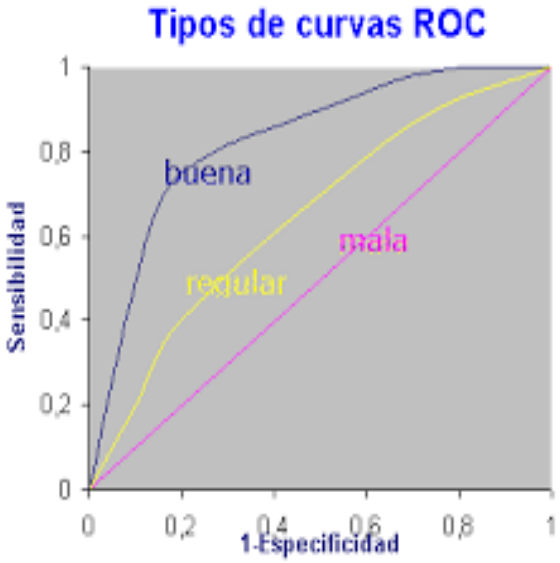

Figura 3.18 Ejemplos curvas ROC [32].

La curva definida como "mala" es la peor situación. En ella el AUC toma el valor en torno a 0.5 por lo que el modelo no tiene capacidad para distinguir entre las dos clases.

Lo ideal sería que la curva tuviese área 1 ya que en ese caso se tendría el perfecto funcionamiento del algoritmo.

3.5 Proceso de resolución de un problema de Machine Learning

A continuación, se van a explicar los distintos pasos que hay que seguir para elaborar un proyecto de Machine Learning. Aunque sin ninguna duda la parte más llamativa de este proceso se trate de la implementación del algoritmo y el análisis de los resultados obtenidos, cabe destacar que alrededor del 80 % del tiempo que dedican los científicos de datos en un proyecto de estas características se basa en la obtención de datos, la limpieza y la organización de estos.

3.5.1 Entender el problema de negocio

Lo primero de todo es entender el problema que se quiere resolver. Normalmente los Ingenieros de Machine Learning o Científicos de Datos a lo largo de sus trayectorias tienen que resolver una amplia variedad de problemas cada uno independiente e incluso de distintos sectores. Es por ello que una componente fundamental es conocer qué es lo que el cliente demanda para poder así enfocar de la manera correcta el problema y que los resultados obtenidos tras la aplicación del modelo permitan a la empresa tomar una acción que suponga una ventaja competitiva. Además de todo esto, conocer bien la finalidad del problema permite tomar unas variables de entrada que aporten valor al modelo. La perspectiva de negocio es imprescindible en este tipo de proyectos.

3.5.2 Obtención y carga de los datos

Como se ha adelantado anteriormente, esto es una de las tareas más difíciles del proyecto. Obtener datos que sean fiables y de calidad y además seleccionar las variables relevantes, no es tarea fácil. También puede ocurrir que mientras se está llevando a cabo esta toma de datos, haya algún problema en alguno de los sensores de medición y se tenga que volver a empezar.

3.5.3 Análisis exploratorio y limpieza de datos

Una vez se tienen los datos cargados en la plataforma donde se vaya a trabajar (en este caso será Python), es imprescindible llevar a cabo un análisis exploratorio de los datos. Esto permitirá conocer las características de cada una de las variables teniendo en cuenta todas las muestras tomadas. Además, con esto se podrá examinar si para alguna de estas muestras falta algún valor y en tal caso sería necesario sustituir esa incógnita por algún número extremo para que el algoritmo no lo tenga en cuenta (por ejemplo 99999), aunque otra opción sería sustituir el valor de esa variable por la media de las muestras. Son distintas opciones las que se pueden usar y dependerá en mayor o menor medida del problema que se esté estudiando. Una parte muy importante que se realiza en este punto es determinar si el problema está o no balanceado. Si se trata de un problema de clasificación y una de las clases es dominante frente al resto, es probable que el algoritmo no trabaje correctamente y por ello habrá que balancear bien eliminando muestras de la clase con más casos, o repitiendo de las menos numerosas. Debido a todo lo expuesto, se concluye que es un paso muy importante para poder avanzar correctamente.

3.5.4 División del conjunto de datos para el entrenamiento

Tras haber realizado la carga de datos, el análisis exploratorio del problema y la limpieza, se procede a dividir el conjunto de datos en una parte que se dedicará al entrenamiento y otra parte se usará para validar el modelo. Normalmente se dedica en torno al $70\,\% - 85\,\%$ de los datos totales para entrenar. De esta forma el algoritmo entrenará a base de ese conjunto de datos y se valida con el resto de datos para que de esta forma no se tenga que esperar a un futuro a que se tengan nuevos datos.

3.5.5 Elección del modelo y entrenamiento

Se elegirá el modelo de acuerdo al problema que se quiere resolver. Se usarán también distintos modelos en la medida de lo posible para resolver el problema y de esta forma analizar el que tenga el mejor rendimiento.

3.5.6 Análisis de resultados

Después de haber realizado todo el proceso descrito anteriormente llega el momento de determinar si el algoritmo cumple con las expectativas. En el caso que no sea así, se volvería al paso anterior para tratar de buscar otro modelo o incluso volver al inicio porque puede que se necesiten más variables, no se haya hecho bien la limpieza... En el caso de que sea positivo el resultado, lo que habrá que hacer será transmitir lo

obtenido a la parte de negocio de la empresa o al cliente, para que de esta manera se pueda tomar la acción que desde el principio se buscaba ya que supondrá una ventaja competitiva.

4 Machine Learning en Python

Para la programación de estos algoritmos y su aplicación en los distintos casos que se verán en el siguiente apartado, se ha hecho uso del lenguaje de programación Python. Para poder ejecutar las distintas acciones desde la limpieza de los datos hasta el entrenamiento del modelo, en Python es necesario implementar librerías. A continuación, se va a realizar una explicación de estas librerías (todas ellas de código abierto).

4.1 Numpy

Numpy es la base sobre la que se apoya el Machine Learning en Python. Esta librería permite realizar operaciones con estructuras típicas de problemas de Machine Learning como vectores, matrices y tensores. Por lo tanto, es una librería especializada en el cálculo numérico y el análisis de datos. Incorpora una clase de objetos los cuales se denominan arrays que permiten representar colecciones de datos de un mismo tipo en distintas dimensiones y preparar funciones eficientes para la manipulación posterior.

4.2 Pandas

Es una librería de Python especializada en el manejo y análisis de estructuras de datos. Gracias a esta librería se pueden leer archivos en formato Excel, CSV y bases de datos de SQL. Una vez cargada la tabla de datos lo siguiente es realizar las tareas de limpieza.

4.3 Matplotlib

Es una librería que sirve para crear visualizaciones estáticas, animadas e interactivas en 2D en Python.

4.4 Scikit-Learn

Es una librería [29] que sirve para resolver problemas de regresión, clasificación, clustering y reducción de dimensionalidad. También es la encargada de dividir el conjunto de datos en la parte de entrenamiento y la parte de testing. Es compatible con las librerías mencionadas anteriormente. Una de las principales ventajas de esta librería es que cuenta con la mayor parte de los algoritmos principales que se usan para Machine Learning. Realmente no es necesario ajustar los parámetros para que se pueda ejecutar el algoritmo ya que por defecto el propio scikit-learn los tiene activados. Por lo que a continuación se explicarán de los principales algoritmos y cuáles son los parámetros que el usuario puede cambiar en función del problema que esté abordando.

4.4.1 División de datos

Esta parte es fundamental. Una vez se realiza la carga de datos y la limpieza de estos, se divide el conjunto de datos en entrenamiento y testing para poder entrenar el algoritmo con la primera división. Los datos que hay que proporcionar son el dataset que se vaya a dividir, el porcentaje de estos datos que se destinará a entrenar el modelo, el porcentaje destinado a validarlo y por último el método que se quiera para mezclar los datos.

Esto último es importante debido a que no interesa coger por ejemplo el 80 % primero, sino que es mejor que esté todo mezclado.

4.4.2 Modelos predictivos

Como ya se ha introducido anteriormente, en la actualidad Sklearn (Scikit-Learn) es una de las librerías más potentes para Machine Learning. La implementación del algoritmo se realiza una vez se ha separado el modelo con la función anterior.

Regresión lineal

Para este algoritmo los parámetros que se pueden configurar son los siguientes:

- *fit_intercept*: True/False. Esto lo que hará será hacer el valor de la constante que traslada la recta nula o distinta de 0. En el caso de que dicha constante fuese nula, la recta pasaría por el origen.
- *normalize*: True/False. Para normalizar o no los datos. Generalmente se normaliza/estandariza antes de llegar a este punto, pero si antes no se ha realizado, se sabe que aquí es posible hacerlo y de esta forma el algoritmo trabajará correctamente.

Regresión logística

Para este algoritmo los parámetros que se pueden configurar son los siguientes:

- *fit_intercept*: True/False. Al igual que antes.
- *penalty*: sirve como regulador. Tiene los siguientes tipos:
  - L1: aquellas variables de entrada que no afecten de forma significativa a la salida las toma como nulas, o lo que es lo mismo $\beta = 0$.
  - L2: se conoce como Ridge. En este caso en lugar de eliminar las β, las minimiza y así elimina el sobreajuste.
  - elasticnet: aplica las dos regulaciones citadas anteriormente. Hay que indicar el peso o importancia que se quiere tener de cada una de las dos.
- *tol*: sirve para que una vez se alcanza la tolerancia deseada (convergencia del algoritmo), el programa deje de iterar.

Árbol de decisión

A modo de ejemplo, se verán los parámetros para la regresión. Para este algoritmo los parámetros que se pueden configurar son los siguientes:

- *criterion*: selección del criterio de división que hace que los datos se dividan de la mejor manera posible. Por defecto está "mse" que es la más importante. "mse" es la media del error cuadrado. Para una versión actualizada de la librería también se encuentra disponible "mae", la media del error absoluto.
- *splitter*: estrategia utilizada para la división en cada nodo. "best" que es la mejor opción y está por defecto y "random" que lo realiza de forma aleatoria.
- max depth: la profundidad máxima del árbol.
- main samples split

Support Vector Machines

Como se ha explicado anteriormente, los algoritmos de SVMs son usados para clasificación, regresión y detección de outliers.

- Para clasificar se utiliza SVC, Support Vector Classifier. Para poder obtener el algoritmo de SVC primero hay que cargar el de SVM mediante la librería scikit-learn y posteriormente el de SVC.
- Para la regresión se utiliza SVR, Support Vector Regression. El método para utilizarlo sería igual que el anterior, es decir, cargar el algoritmo de SVM mediante la librería de scikit-learn y posteriormente cargar SVR.

4.5 Keras

Es la librería [21] por excelencia que se utiliza para elaborar algoritmos de *Deep Learning* o aprendizaje profundo. Se utilizarán principalmente los siguientes módulos de la librería:

- Sequential: permite que las capas se agrupen de forma secuencial.
- Dense: permite conectar las distintas capas de la red neuronal.
- SGD: optimizador del algoritmo mediante el gradiente descendente.

5 Casos de estudio

Se van a aplicar las técnicas descritas de Machine Learning en Python. Como ya se ha expuesto anteriormente, debido a la falta de datos fiables del sector aeroespacial (aún están comenzando a implementar estas técnicas por lo que no se tienen suficientes datos y las empresas que los tengan no los ceden a terceros), ya que la toma de decisiones de personas clave del sector como es el caso del piloto es muy similar a la de un médico y además hay fuentes fiables de datos, se van a abordar problemas del sector sanitario.

Se analizarán tres casos. El primero de ellos se trata de un problema de Machine Learning no supervisado, el segundo un problema de Machine Learning supervisado, y el último se trata de un problema de tratamiento de imágenes en el que se aplican técnicas de Deep Learning.

5.1 Caso I. Clasificación de pacientes con amputaciones traumáticas en distintas clases según su gravedad

Se ha realizado el estudio de la agrupación de pacientes con hemorragia externa de distintas clases, ordenadas de menor a mayor gravedad para su posterior tratamiento con un protocolo de actuación previamente definido [16]. Como variables de entrada se han utilizado las siguientes:

5.1.1 Descripción de cada clase

Clase I

- Estimación de sangre: Hasta 750 *ml*.
- Frecuencia cardiaca: Hasta 100 *lpm*.
- Presión pulsos: Normal.
- Relleno capilar: Normal.
- Frecuencia respiratoria: entre: 14 *rpm* y 20 *rpm*.
- Diuresis: >30 (*ml/h*).
- Nivel de conciencia: Ansiedad leve.
- Coloración piel: Normal.

En el programa se introduce un vector cuyas componentes serían las siguientes:

$$Clase\ I = [0,0,0,0,0,0,0,0] \tag{5.1}$$

Clase II

- Estimación de sangre: Entre 750 *ml* y 1500 *ml*.
- Frecuencia cardiaca: Entre 100 *lpm* y 120 *lpm*.
- Presión pulsos: Disminuido.
- Relleno capilar: Retrasado.

- Frecuencia respiratoria: entre 20 *rpm* y 30 *rpm*.
- Diuresis: entre: 20 (*ml/h*) y 30 (*ml/h*).
- Nivel de conciencia: Ansiedad o agresividad.
- Coloración piel: Pálida.

En el programa se introduce un vector cuyas componentes serían las siguientes:

$$Clase\ II = [1,1,0,1,1,1,1,1] \tag{5.2}$$

Clase III

- Estimación de sangre: Entre 1500 *ml* y 2000 *ml*.
- Frecuencia cardiaca: Entre 120 *lpm* y 140 *lpm*.
- Presión pulsos: Disminuido.
- Relleno capilar: Retrasado.
- Frecuencia respiratoria: entre 30 *rpm* y 35 *rpm*.
- Diuresis: entre: 10 (*ml/h*) y 20 (*ml/h*).
- Nivel de conciencia: Ansiedad y confusión.
- Coloración piel: Pálida.

En el programa se introduce un vector cuyas componentes serían las siguientes:

$$Clase\ III = [2,2,1,1,2,2,2,1] \tag{5.3}$$

Clase IV

- Estimación de sangre: 2000 *ml* o más.
- Frecuencia cardiaca:140 *lpm* o más.
- Presión pulsos: Disminuido.
- Relleno capilar: Retrasado.
- Frecuencia respiratoria: más de 35 *rpm*.
- Diuresis: entre: 0 (*ml/h*) y 10 (*ml/h*).
- Nivel de conciencia: Confusión + letargia.
- Coloración piel: Grisácea.

En el programa se introduce un vector cuyas componentes serían las siguientes:

$$Clase\ IV = [3,3,1,1,3,3,3,2] \tag{5.4}$$

5.1.2 Estudio del problema

Cada clase está determinada por un paciente ideal. Se define el centroide como el paciente ideal de cada clase. La labor que se tiene que realizar es que, tomando los datos anteriores con diferentes sensores, el programa clasifique al nuevo paciente para que ingrese en una clase u otra. Para ello lo que se van a evaluar son las características que tenga el nuevo paciente para así poder compararlas con cada una de las clases y de este modo agruparlo en la que sea más similar. En el problema, se ha definido que la mayoría de las variables son categóricas (cada una tiene una categoría diferente). Esto supondrá tener que afrontar el problema de una forma diferente a si las variables no fuesen clases y fuesen variables continuas.

5.1.3 Modelos predictivos posibles

Se han propuesto tres posibles métodos de asignación de pacientes.

Matriz de distancias, *K-Means* sin variación del centroide

Se adjunta una representación gráfica en la que se basará la posterior explicación del problema,

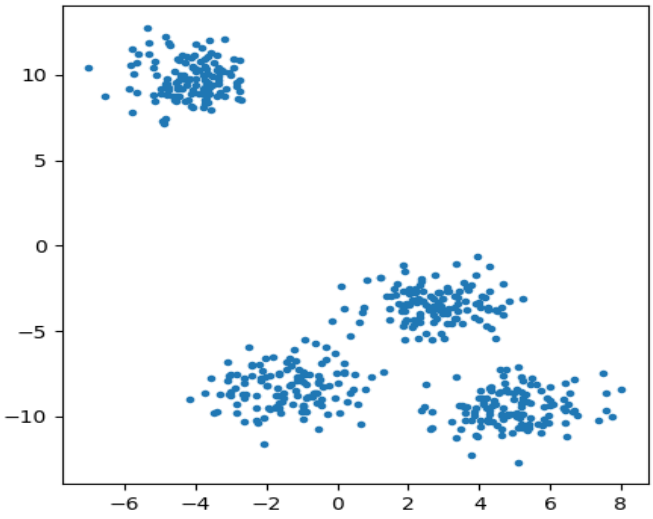

Figura 5.1 Representación gráfica de distintos pacientes y la correspondiente clasificación en cada grupo.

Como se puede apreciar en la imagen, hay cuatro grupos diferenciados (en este caso estos vienen definidos y no se pueden alterar debido a que para cada uno de ellos hay distintos protocolos de actuación). Como ya se ha dicho, el centroide de cada clase es el paciente ideal.

Lo más importante en esta clasificación:

- El paciente ideal siempre es el mismo. Se ha considerado que lo más importante es el protocolo médico.
- Para medir cuál es el grupo al que pertenece el paciente, se calculará la distancia euclídea que hay a cada paciente ideal y aquella que sea menor, será a la clase a la que pertenezca el paciente evaluado. Si solo se tuviesen dos variables de medición x e y, la distancia se obtendría de la siguiente forma:

$$d_1 = \sqrt{(x - x_{claseI})^2 + (y - y_{claseI})^2}$$

  Siendo x e y el valor de la variable que tendría el nuevo paciente.

Tampoco se puede determinar la eficacia del modelo empleado ya que no hay valores para validar los resultados obtenidos. Para mejorar el modelo sería conveniente establecer un criterio que permitiese tener unas variables de entrada con mayor peso y de esta forma la distancia estaría ponderada.

K-Modes

Este método de clasificación se aplica si las variables con las que se trabajan son categorías como ocurre en este caso.

Se utiliza un concepto de distancia diferente al anterior. Hay que maximizar la función de similaridad. Esta se define de forma que si una de las variables del nuevo paciente coincide con la del paciente ideal sumaría 1 a dicha distancia. Por lo tanto, cuantas más veces coincida con las variables del paciente ideal mayor será esta función. De esta manera estudiando esta función con respecto a cada una de las clases, aquella que el sumatorio sea mayor, será a la que pertenezca el paciente. Claramente este modelo no es el elegido debido a que:

- No se tiene en cuenta el ascenso de gravedad que se tiene en aquellas variables que no se puede agrupar únicamente en dos grupos. Es decir si, por ejemplo, se compara la estimación de sangre perdida, el modelo trataría igual si comparamos con $< 750ml$ a cualquier paciente que estuviera fuera de ese rango. El problema que hay, como ya se ha comentado, es que es muy importante tener en cuenta que

un paciente de clase *IV* debe ser tratado diferente que uno de clase *II* el cual es bastante similar a la primera clase que hemos comentado.

- Otro inconveniente sería a la hora de actualizar el paciente ideal debido a que tomará aquel que sea la moda (y no la media como es en el caso de $K-Mean$).

K-Means

Al igual que en el caso anterior, para determinar el grupo al que se le asigna un paciente, se tomará la menor de las distancias con respecto a los pacientes ideales. La diferencia que existe es que, es posible que el centro de cada clase varíe. Para poder llevar a cabo este estudio, sin que el paciente tipo sufra mucha variación, ya que podría alterar el protocolo, se ha supuesto que previo al ingreso de nuevos pacientes, habían llegado 100 pacientes ideales de cada una de las clases. De esta forma se consigue que una nueva actualización de un paciente no varía el centro del grupo. Por lo tanto, se ha considerado vital que los pacientes que caracterizan a cada clase no se desplacen de forma muy agresiva.

Se va a explicar el procedimiento matemático:

- 1) Asignación de cada observación al cluster con la distancia euclídea. Este es el paso que se realiza en el primer método que se ha explicado.
- 2) Recalcular el centro de cada clasificación (volver a calcular el nuevo paciente ideal) una vez se ha añadido un nuevo paciente.

5.1.4 Razones de elección del modelo

Tras la descripción de cada uno de los modelos, en primera instancia se ha descartado el método de las $K-Modes$ por las razones expuestas. Por lo que queda elegir entre el método de la matriz de distancias y *K-Means*, poniendo en este último 100 pacientes previos asignados en cada grupo para que el paciente ideal no varíe bruscamente.

Ambos procedimientos se consideran válidos y dependerá de la flexibilidad que se tenga para alterar al paciente que define a cada clase ya que igual según el criterio de los médicos, esto es inalterable.

5.1.5 Análisis de resultados

Para realizar el estudio pertinente, se ha utilizado *Python* y las librerías *numpy*, *pandas*, *kmodes.kmodes* y *sklearn.cluster*.

Se han añadido 149 pacientes aleatorios y se ha comparado el método de la Matriz de distancias con el de *K-Means*. Los resultados han sido los siguientes:

- Coincidencias = 138 pacientes asignados al mismo grupo = 92.62 %
- Diferencia de 1 clase = 8 pacientes = 5.37 %
- Diferencia de 2 clases = 3 pacientes = 2.01 %

Se puede observar que, como se quería, los centroides apenas han variado y es por ello que ambos métodos han coincidido en la asignación de la gran mayoría de pacientes. A continuación, se adjuntan los 3 pacientes que han tenido una diferencia de dos clases a la hora de clasificarlos. Se usarán estas muestras para que contando con la ayuda de personal médico cualificado, se sepa el grupo al que realmente debería pertenecer.

$$[2,0,1,0,2,1,0,0]$$

- *KMeans* fijo = Clase I
- *KMeans* = Clase III

$$[0,1,1,0,0,3,0,1]$$

- *KMeans* fijo = Clase I
- *KMeans* = Clase III

$$[1,1,1,0,0,2,0,1]$$

- *KMeans* fijo = Clase I
- *KMeans* = Clase III

5.1.6 Usabilidad

El método de uso sería evaluar variable a variable al nuevo paciente que se quiera asignar e introducir los valores en el programa. Una vez hecho esto se tendría la clase a la que pertenece y a continuación se actuaría en función del protocolo descrito.

5.1.7 Mejora del modelo

Para un primer modelo del algoritmo, se ha tenido en cuenta que todas las variables tienen el mismo peso. Realmente esto no es algo preciso ya que hay ciertas variables que son más críticas y afectan en mayor medida a la agrupación del paciente. La mejora se podría realizar de diversas formas y aquí se plasman dos de ellas:

- Acudir a un especialista médico que permita conocer desde su experiencia cuáles son las mediciones a las que se les da más importancia. Incluso puede ser que haya variables que permitan descartar algún grupo. Debido a esto, la experiencia de un profesional es crucial.
- Tener un registro de pacientes y conocer en cada caso a qué grupo se le ha asignado. Esto supondría pasar de algoritmos de Machine Learning No Supervisados (como se ha afrontado el problema) a Supervisados. Se podría entrenar un modelo de *K-Nearest Neighbours* lo cual sería mucho más eficaz.

5.2 Caso II. Determinación lesión cáncer de mama

Se ha realizado el estudio de la determinación de la probabilidad que tiene un paciente dado una serie de rasgos, de padecer cáncer de mama benigno o maligno. Para ello se ha empleado el dataset proveniente del Hospital Universitario de Wisconsin proporcionado por el doctor William H. Wolberg. La fecha en la que se hizo público esta base de datos fue el 8 de Enero de 1991 [15].

Se llevó a cabo un registro de 699 pacientes. De cada uno de ellos se tiene la información de las variables que se explican a continuación:

- Radio: media de las distancias desde el centro hasta los puntos del perímetro.
- Textura: desviación estándar de los valores de la escala de grises.
- Perímetro
- Área
- Suavidad: variación local en longitudes de radio.
- Compacidad: $\frac{permetro^2}{rea} - 1$
- Puntos cóncavos: número de porciones cóncavas del contorno.
- Simetría
- Dimensión fractal
- Resultado: cáncer benigno o maligno.

Se tienen 9 variables de entrada que toman un valor entre el $1 - 10$ ya que los datos vienen escalados. Como variable objetivo o variable a predecir se tiene "Resultado", en la que se registra la decisión que el médico tomó para cada paciente. Debido a esto, se puede concluir que se trata de un problema de Machine Learning Supervisado. En este caso el problema consiste en una clasificación y se tienen dos categorías posibles.

5.2.1 Análisis exploratorio

A continuación, se adjunta una imagen donde se puede observar el porcentaje de casos benignos y malignos.

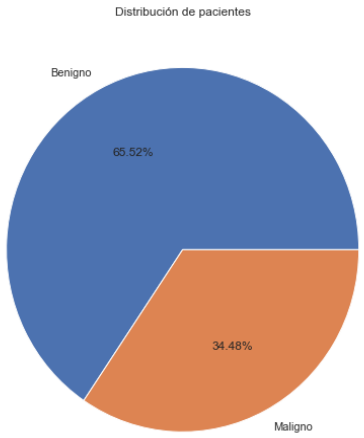

Figura 5.2 Distribución resultados de evaluación.

Como se puede observar en la imagen anterior, el 65.52 % de los casos tuvieron como resultado benigno, mientras que el 34.48 % restante fueron malignos. Debido a que se examinaron un total de 699 pacientes, se tiene que:

- 458 pacientes tuvieron cáncer benigno.

- 241 pacientes tuvieron cáncer maligno.

Son cifras que permiten asegurar que se trata de un problema cuyos datos de salida están balanceados ya que no existe una gran diferencia entre las dos clases.

A continuación, se adjunta la previsualización que se obtuvo tras haber realizado la carga de datos.

	V1	V2	V3	V4	V5	V6	V7	V8	V9	Class
0	5	1	1	1	2	1	3	1	1	2
1	5	4	4	5	7	10	3	2	1	2
2	3	1	1	1	2	2	3	1	1	2
3	6	8	8	1	3	4	3	7	1	2
4	4	1	1	3	2	1	3	1	1	2
...	...	...	...	...	...	...	...	...	...	...
694	3	1	1	1	3	2	1	1	1	2
695	2	1	1	1	2	1	1	1	1	2
696	5	10	10	3	7	3	8	10	2	4
697	4	8	6	4	3	4	10	6	1	4
698	4	8	8	5	4	5	10	4	1	4

Figura 5.3 Previsualización del dataset tras la carga de datos.

Se observa que, como se ha dicho anteriormente, se tienen pacientes registrados desde el paciente 0 al 698 quedando un total de 699 pacientes.

Otro punto positivo que tiene este dataset es que todas las variables de entrada que se han definido previamente están dentro de un mismo rango de modo que no es necesario estandarizar o normalizar, se podría hacer. De todos modos, se realiza la estandarización de las variables de entrada para que el rango esté entre 0 y 1.

Siguiendo con la etapa del análisis exploratorio de los datos, se adjuntan las principales características que tiene cada una de las variables tanto para el caso de cáncer benigno como maligno.

	V1	V2	V3	V4	V5	V7	V8	V9	Class
count	458.000000	458.000000	458.000000	458.000000	458.000000	458.000000	458.000000	458.000000	458.0
mean	2.956332	1.325328	1.443231	1.364629	2.120087	2.100437	1.290393	1.063319	2.0
std	1.674318	0.907694	0.997836	0.996830	0.917130	1.080339	1.058856	0.501995	0.0
min	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	2.0
25%	1.000000	1.000000	1.000000	1.000000	2.000000	1.000000	1.000000	1.000000	2.0
50%	3.000000	1.000000	1.000000	1.000000	2.000000	2.000000	1.000000	1.000000	2.0
75%	4.000000	1.000000	1.000000	1.000000	2.000000	3.000000	1.000000	1.000000	2.0
max	8.000000	9.000000	8.000000	10.000000	10.000000	7.000000	9.000000	8.000000	2.0

Figura 5.4 Características variables de entrada para cáncer benignos.

	V1	V2	V3	V4	V5	V7	V8	V9	Class
count	241.000000	241.000000	241.000000	241.000000	241.000000	241.000000	241.000000	241.000000	241.0
mean	7.195021	6.572614	6.560166	5.547718	5.298755	5.979253	5.863071	2.589212	4.0
std	2.428849	2.719512	2.562045	3.210465	2.451606	2.273852	3.350672	2.557939	0.0
min	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	4.0
25%	5.000000	4.000000	4.000000	3.000000	3.000000	4.000000	3.000000	1.000000	4.0
50%	8.000000	6.000000	6.000000	5.000000	5.000000	7.000000	6.000000	1.000000	4.0
75%	10.000000	10.000000	9.000000	8.000000	6.000000	7.000000	10.000000	3.000000	4.0
max	10.000000	10.000000	10.000000	10.000000	10.000000	10.000000	10.000000	10.000000	4.0

Figura 5.5 Características variables de entrada para cáncer malignos.

Observando la media de cada una de las variables para ambos casos se aprecia que, para el cáncer benigno, esta media no supera el valor 3 para ninguna de las variables y sin embargo en el caso del cáncer maligno una variable llega a superar el valor de 7. También se pueden observar las características anteriores mediante un gráfico de cajas, el cual consiste en lo siguiente:

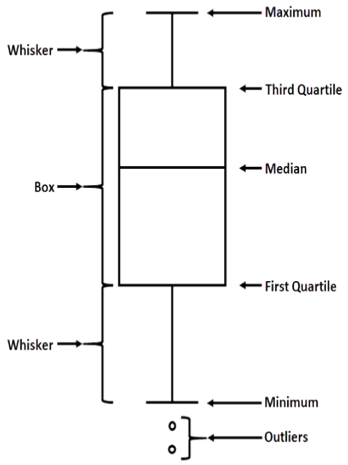

Figura 5.6 Explicación visual del gráfico de cajas.

El resultado obtenido para las variables del problema es el siguiente:

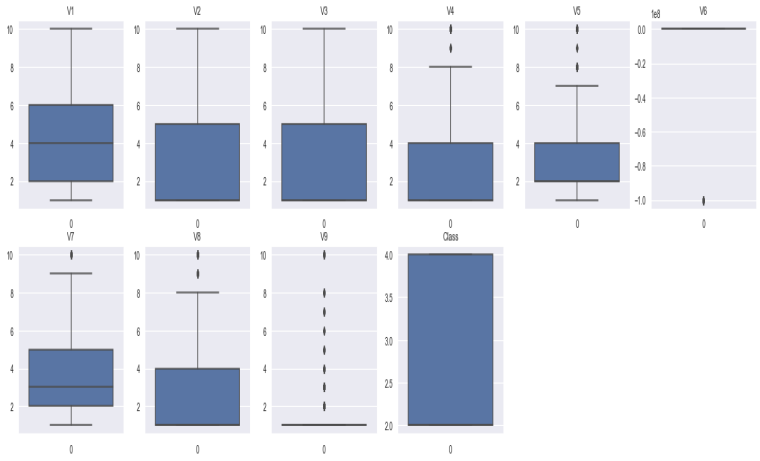

Figura 5.7 Gráfico de cajas para las variables del problema.

Para poder observar la relación que tienen unas variables con otras, se usa la matriz de correlación. La diagonal es 1, debido a que ahí se está comparando la variable consigo misma.

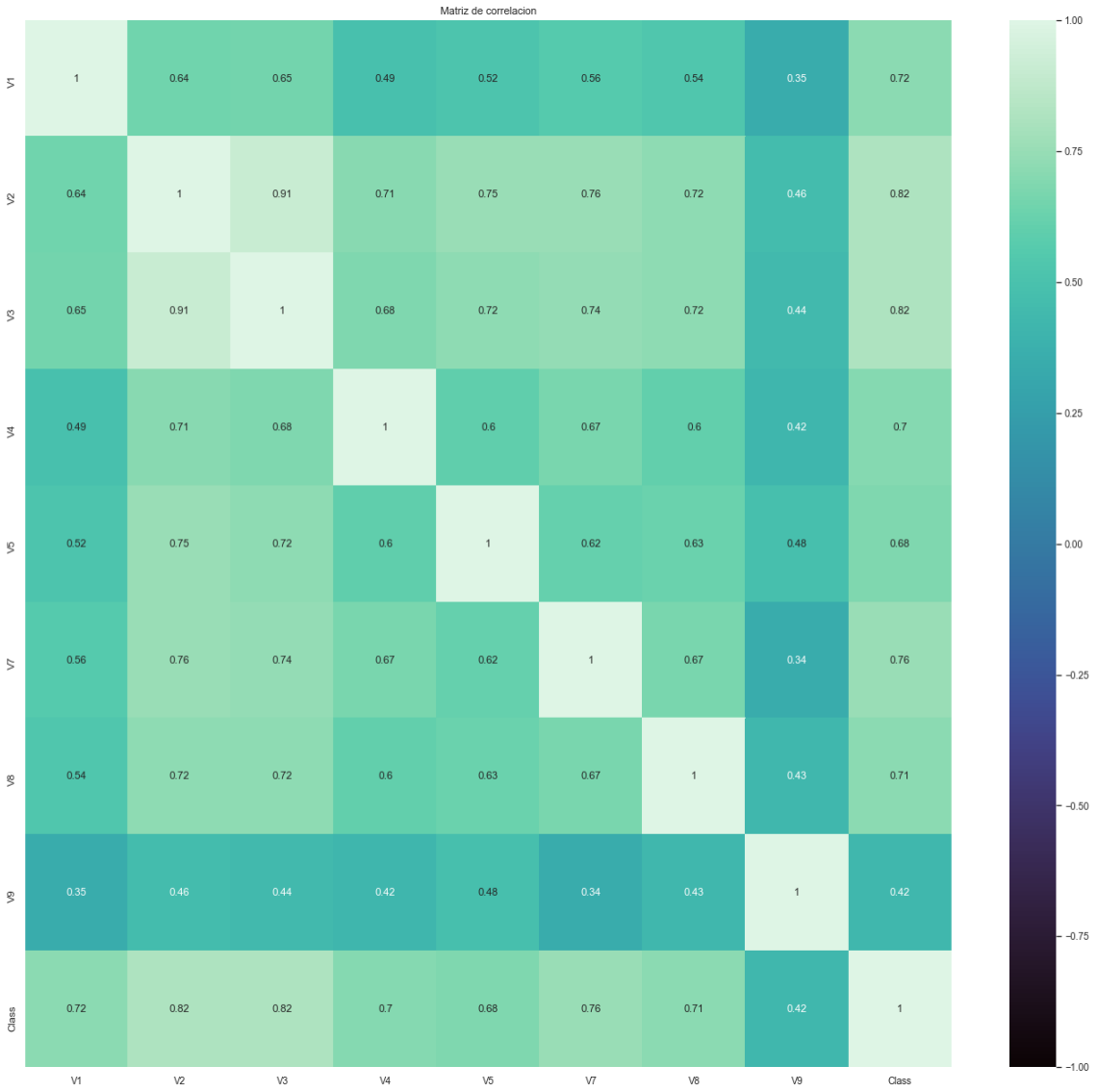

Figura 5.8 Matriz de correlación de las variables.

Se observa que las variables V2 y V3 están altamente relacionadas a pesar de que aparentemente no existe relación entre la textura y el perímetro. Cabe también destacar, que en el caso que el dataset tenga una cantidad considerable de columnas (variables de entrada), se hará uso de algoritmos como el de PCA (Componentes Principales) para que, usando la relación entre variables, se creen otras nuevas a partir de aquellas que estén altamente relacionadas. También se podría reducir el número de variables eliminando aquellas que se observe que no tienen mayor implicación en el algoritmo. Como en este caso el número de variables no es excesivo y el coste computacional no es elevado, no es necesario realizar lo que se acaba de explicar.

5.2.2 Entrenamiento del modelo

Una vez realizado el análisis exploratorio se procede a dividir el modelo en dos partes, una parte que se encargará de crear el modelo, el conjunto de entrenamiento, y otra que se encargará de evaluar el modelo, el conjunto de testing. Para ello, se usa la siguiente línea de código:

```
X_train, X_test, Y_train, Y_test = train_test_split(X,Y,test_size = 0.2)
```

Figura 5.9 División en conjunto de entrenamiento y testing.

Como se puede observar, el 80 % de los datos serán de entrenamiento y el resto de prueba. Se ha optado por entrenar y validar varios modelos para así comparar cuál de todos ellos tiene mayor eficacia. Dicha implementación se ha realizado de la siguiente forma:

```
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import LinearSVC, SVC
from sklearn.neural_network import MLPClassifier
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
```

Figura 5.10 Carga de las librerías usadas para entrenar el modelo.

Se puede observar que se han añadido además de los algoritmos explicados en la segunda sección, el *Random Forest* y el *Gradient Boosting*. El primero de ellos es un conjunto de *Árboles de decisión* y pese a que computacionalmente sean más rápidos que estos, son más complicados de entender. En definitiva es un algoritmo que se usa para Machine Learning Supervisado en algoritmos tanto de regresión como de clasificación [14]. Por otro lado, el Gradient Boosting se compone de una secuencia de *Árboles de decisión* entrenados de forma que los errores de uno de ellos son reducidos por el siguiente. También se pueden aplicar tanto para regresión como para clasificación [3].

5.2.3 Análisis de resultados y elección del modelo

Para medir el rendimiento de los algoritmos con el conjunto de datos se calcula el *accuracy* o exactitud. Finalmente, y como se comprueba en la siguiente imagen, el método del bosque aleatorio (random forest) es el que tiene mayor eficacia.

```
                                Logistic Regression: 95.00%
                                K-Nearest Neighbors: 94.29%
                                      Decision Tree: 94.29%
Support Vector Machine (Linear Kernel): 95.00%
   Support Vector Machine (RBF Kernel): 94.29%
                                   Neural Network: 95.00%
                                    Random Forest: 96.43%
                                Gradient Boosting: 94.29%
```

Figura 5.11 Análisis de resultados.

Cualquiera de los modelos empleados da unos resultados extremadamente favorables lo cual es muy bueno. Como conclusión se llega a que el programa clasifica de forma correcta en 19 de 20 casos

5.2.4 Usabilidad

Para poder agilizar el proceso de determinación de si es benigno o maligno y actuar con mayor o menor rapidez, el médico tendría que rellenar los valores de entrada que el programa necesita, lo cual es un proceso muy rápido y como se ha podido observar es altamente fiable. De esta forma se tendría una primera opinión basada en la experiencia de otro médico especialista en el campo ya que la máquina aprende del médico que creó estos datos.

5.3 Caso III. Predicción problemas cutáneos

Se ha realizado el estudio del problema de clasificación de imágenes cuyos pacientes tienen problemas cutáneos (bien sean malignos o benignos). Para ello se ha hecho uso del dataset "The HAM10000" proveniente de la base de datos de Harvard [31], y como bien se dice en su página web, las imágenes provienen de agrupaciones de distintos datasets. Los distintos problemas cutáneos que se tratan son los siguientes:

- **Nevus melanocítico**: es una lesión cutánea benigna muy frecuente que se da en un porcentaje elevado de la población. Es un tumor benigno derivado de los melanocitos (células responsables de la pigmentación de la piel). Puede ser una lesión plana o bien sobreelevada y el color puede variar, tanto color carne como marrón. Este tipo de lesión se encuentra altamente influenciada por el grado de exposición solar al que ha sido expuesta la piel.
- **Melanoma**: es el tipo más grave de cáncer de piel. Se origina en las células que producen en los melanocitos (que son las células que producen melanina, lo que da color a la piel). El riesgo de tener melanoma aumenta con la exposición a la radiación ultravioleta (UV) de luz solar o lámparas por lo que limitar esta exposición puede ayudar a reducir el riesgo de tener este tipo de enfermedad.
- **Enfermedades benignas, Queratosis**: la queratosis en este caso seborreica, es un crecimiento cutáneo común no canceroso y se tiende a aumentar el número de estas manchas a medida que la gente envejece. Suele ser de color marrón o también negro. Los crecimientos tienen un textura cerosa, son escamosos y también se encuentran ligeramente elevados.
- **Carcinoma de células basales**: es un tipo de cáncer de piel que comienza en las células basales, un tipo de células que se encuentran en la piel y cuya función es la de generar células cutáneas nuevas a medida que las viejas mueren.
- **Queratosis actínica**: la queratosis en este caso actínica es una mancha áspera y escamosa en la piel la cual se presenta tras años de exposición al sol. Esta se desarrolla lentamente y por lo general aparece por primera vez en personas mayores de 40 años. Al igual que los anteriores, para reducir la probabilidad de que este tipo de lesión aparezca, hay que reducir el tiempo de exposición al sol, así como proteger la piel de los rayos ultravioletas.
- **Lesiones vasculares**: las lesiones vasculares incluyen aquellas adquiridas a lo largo de la vida (como el granuloma piógeno), y las que están presentes al nacer o aparecen poco después del nacimiento (marcas de nacimiento vasculares). Las marcas vasculares congénitas incluyen los tumores vasculares (como el hemangioma de la lactancia) y malformaciones vasculares.
- **Dermatofibroma**: es un tumor cutáneo benigno muy frecuente. La mayor parte de los especialistas consideran que la dermatofibroma tiene un origen tumoral aunque hay una parte que lo considera como carácter reactivo (en este caso refiriéndose a una picadura de insecto o un traumatismo previo a su aparición). Suele ser una lesión de pequeño tamaño, con consistencia firme y coloración variable.

A continuación, se muestran imágenes de cada una de las clases que tiene el dataset.

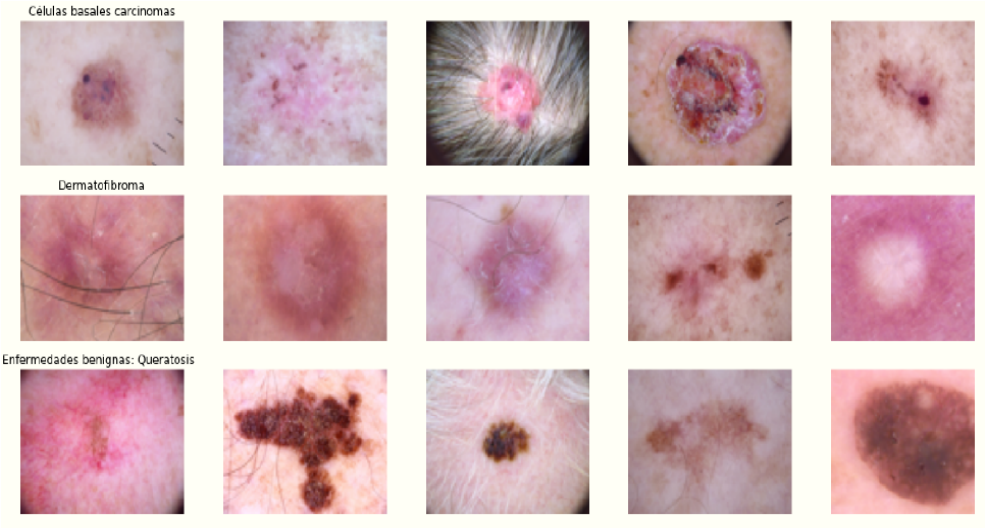

Figura 5.12 Visualización de imágenes I.

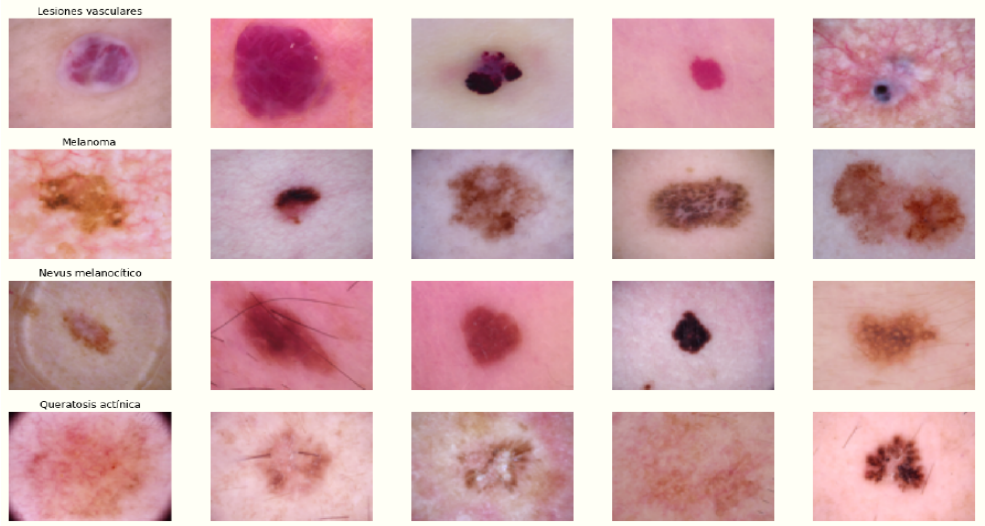

Figura 5.13 Visualización de imágenes II.

La finalidad de este problema será crear una red neuronal convolucional, el método por excelencia para el tratamiento de imágenes, que sea capaz de, con una precisón elevada, predecir el problema cutáneo que pueda tener un paciente.

5.3.1 Análisis exploratorio

El dataset original contiene 10015 imágenes, pero a continuación se verá que hay imágenes que están repetidas. Además de dichas imágenes, se tiene información que, aunque no se usará para entrenar el modelo sirve para entender más el problema. Son los siguientes datos:

- Edad
- Sexo
- Localización de la mancha
- Método usado para diagnosticar el problema

Se observa que inicialmente existen 57 pacientes que no tienen registrada la edad. Dichos valores se reemplazarán con la media de edades de todo el dataset usando la siguiente línea de código:

```
skin_df.isnull().sum()
```

```
lesion_id         0
image_id          0
dx                0
dx_type           0
age              57
sex               0
localization      0
path              0
cell_type         0
cell_type_idx     0
dtype: int64
```

```
skin_df['age'].fillna((skin_df['age'].mean()), inplace=True)
skin_df.isnull().sum()
```

```
lesion_id        0
image_id         0
dx               0
dx_type          0
age              0
sex              0
localization     0
path             0
cell_type        0
cell_type_idx    0
dtype: int64
```

Figura 5.14 Determinación variables nulas.

Se cargan las imágenes y se reajustan a un tamaño de $(64,64)$. Se incluyen en el dataset mediante el uso de una función que recorra los dos Winrar que contienen las imágenes. Una vez todo está definido en el dataset se procede a identificar en las imágenes que se han cargado si existe alguna duplicada.

```
def get_duplicates(x):
    unique_list = list(df_undup['lesion_id'])
    if x in unique_list:
        return 'unduplicated'
    else:
        return 'duplicated'

skin_df['duplicates'] = skin_df['lesion_id']
skin_df['duplicates'] = skin_df['duplicates'].apply(get_duplicates)
```

```
skin_df['duplicates'].value_counts()
skin_df_nd = skin_df[skin_df['duplicates'] == 'unduplicated']
```

```
skin_df['duplicates'].value_counts()
```

```
unduplicated    5514
duplicated      4501
Name: duplicates, dtype: int64
```

Figura 5.15 Estudio de imágenes duplicadas.

Como se puede observar, 4501 imágenes están duplicadas, lo que supone el 44.94 % del total del dataset. Una vez se eliminan dichas duplicidades, se pasa a visualizar el número de imágenes que el dataset contiene de cada uno de los grupos.

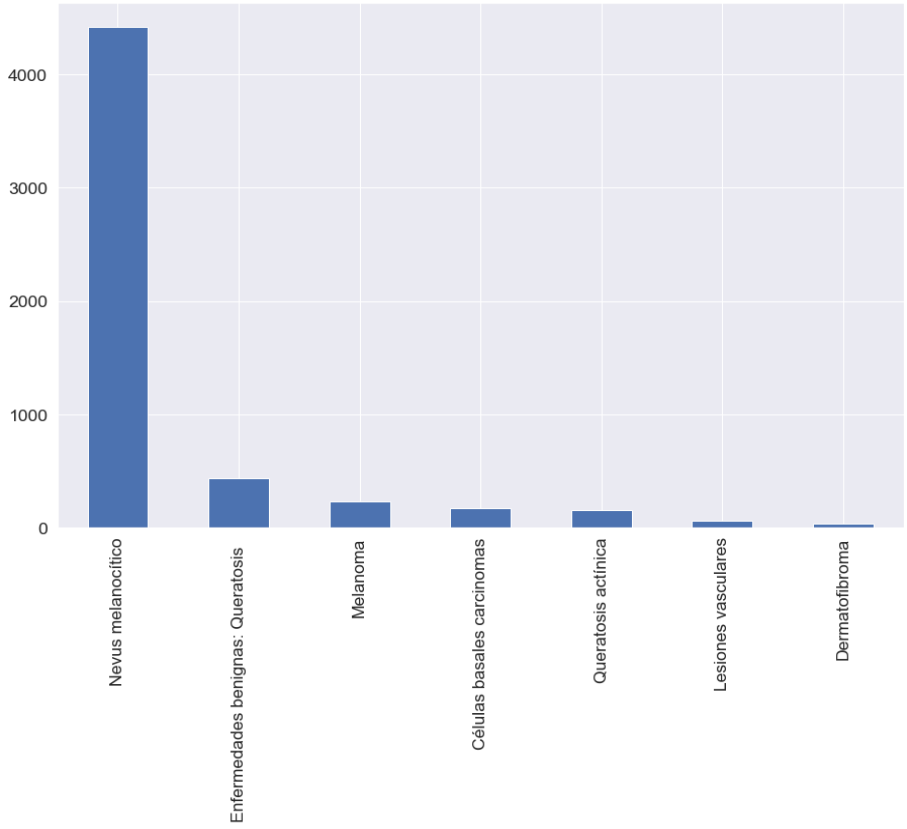

Figura 5.16 Distribución de casos según problema cutáneo.

Claramente se trata de un problema no balanceado,

- Nevus melanocítico: 4415 imágenes
- Enfermedades benignas, Queratosis: 440 imágenes
- Melanoma: 230 imágenes
- Carcinoma de células basales: 175 imágenes
- Queratosis actínica: 151 imágenes
- Lesiones vasculares: 64 imágenes
- Dermatofibroma: 39 imágenes

Cierto es que en las primeras aproximaciones del modelo no se balanceó el problema y como era de esperar no dio un buen resultado. A modo de ejemplo, uno de los problemas más comunes que tienen que estudiar los científicos de datos en el sector bancario es la determinación de transacciones fraudulentas. Si hubiese 1 transacción fraudulenta diaria y 100.000 registradas en un día, si el algoritmo dice que todas son correctas, tendría una exactitud del 99.999 %. ¿Esto querría decir que el modelo funciona correctamente? No, ya que no está resolviendo el problema.

Para llevar esta labor a cabo, se utiliza de la librería de *sklearn* la opción de *resample*. Se generarán 500 imágenes de cada uno de los casos. Para el Nevus melanocítico de las 4415 imágenes se seleccionarán 500 y del resto lo que se hará será repetirse el número de imágenes hasta llegar a 500. Este paso es clave para garantizar la eficacia del modelo pese a que se pueda llegar a disminuir la precisión por lo que se ha explicado anteriormente.

```
# Se va a proceder a reescalar los casos
from sklearn.utils import resample

df_0 = skin_df[skin_df.cell_type_idx == 0]
df_1 = skin_df[skin_df.cell_type_idx == 1]
df_2 = skin_df[skin_df.cell_type_idx == 2]
df_3 = skin_df[skin_df.cell_type_idx == 3]
df_4 = skin_df[skin_df.cell_type_idx == 4]
df_5 = skin_df[skin_df.cell_type_idx == 5]
df_6 = skin_df[skin_df.cell_type_idx == 6]
```

```
n_samples = 500
df_0_balanced = resample(df_0, replace=True, n_samples=n_samples, random_state=42)
df_1_balanced = resample(df_1, replace=True, n_samples=n_samples, random_state=42)
df_2_balanced = resample(df_2, replace=True, n_samples=n_samples, random_state=42)
df_3_balanced = resample(df_3, replace=True, n_samples=n_samples, random_state=42)
df_4_balanced = resample(df_4, replace=True, n_samples=n_samples, random_state=42)
df_5_balanced = resample(df_5, replace=True, n_samples=n_samples, random_state=42)
df_6_balanced = resample(df_6, replace=True, n_samples=n_samples, random_state=42)


skin_df_balanced = pd.concat([df_0_balanced, df_1_balanced,
                              df_2_balanced, df_3_balanced,
                              df_4_balanced, df_5_balanced, df_6_balanced])
```

Figura 5.17 Código de escalado de las muestras.

Quedando como resultado lo siguiente,

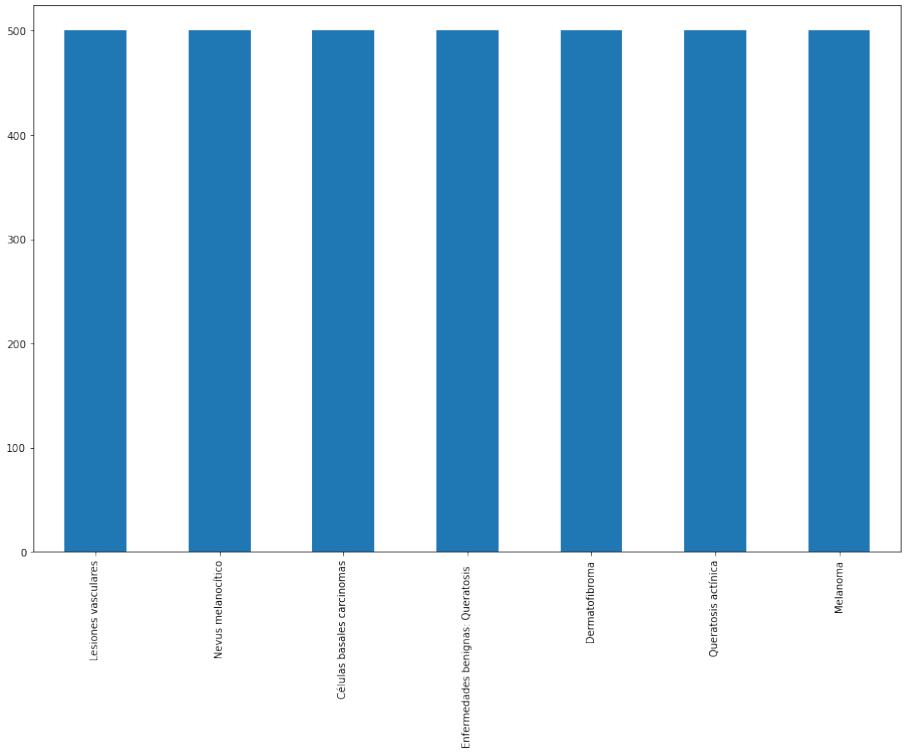

Figura 5.18 Muestras escaladas.

5.3.2 Entrenamiento del modelo

Antes de pasar a comentar la estructura de la red neuronal, lo primero que hay que realizar es definir la X y la salida y. En este caso la X será la información que ofrece la imagen (la cual hay que redimensionar dividiendo el vector entre el máximo de las componentes que es 255). La y será la enfermedad que ha tenido el paciente y al tener 7 clases se han generado 7 columnas distintas, las cuales se rellenarán para cada caso con 0 si no es el problema que tiene el paciente, o con 1 si lo es. Se ha utilizado el código *to_categorical*. Una vez hecho esto, se ha procedido a dividir el conjunto en entrenamiento y testing, mediante la función

train_test_split (al igual que en el caso anterior), y esta vez se ha destinado un 75 % de los datos para el conjunto de entrenamiento y el resto para el testing. No hay una regla que marque este porcentaje por lo que se podría variar y probar el que de un mejor resultado.

```
model = Sequential()
model.add(Conv2D(256, (3, 3), activation="relu", input_shape=(SIZE, SIZE, 3)))
#model.add(BatchNormalization())
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Dropout(0.3))

model.add(Conv2D(128, (3, 3),activation='relu'))
#model.add(BatchNormalization())
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Dropout(0.3))

model.add(Conv2D(64, (3, 3),activation='relu'))
#model.add(BatchNormalization())
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Dropout(0.3))
model.add(Flatten())

model.add(Dense(32))
model.add(Dense(7, activation='softmax'))
model.summary()
```

Figura 5.19 Arquitectura red neuronal.

- La primera capa de neuronas "Convolucional de 2 Dimensiones" es donde entrarán las imágenes. En esta capa se aplican 256 filtros (kernel) de tamaño $3x3$ los cuales detectan ciertas características de la imagen. Como función de activación se utilizará el rectificador lineal unitario.

- A continuación se hace un MaxPooling de $2x2$ que reduce el tamaño de la imagen que entra a la mitad, manteniendo las características que detectó cada kernel.

- Para evitar el *overfitting* se añade la capa de Dropout. La capa de Dropout consiste en eliminar la información que aporta un determinado porcentaje de neuronas dentro de cada una de las capas de la red neuronal y se utiliza con el fin de reducir el sobreajuste. El sobreajuste u overfitting, es el efecto causado por el sobreentrenamiento del algoritmo para unos ciertos datos.

- En las siguientes dos capas se sigue la misma metodología. La diferencia es que en cada capa se reducen los filtros del kernel a la mitad. Aunque se podían haber mantenido en todas las capas el número del kernel fijo, se ha decidido hacer de esta forma, porque en cada capa se reduce la dimensión de la imagen a la mitad.

- A continuación se aplana con "Flatten()" los 32 últimos filtros del kernel y se crea una capa de 32 neuronas tradicionales mediante "Dense()".

- Finalmente se pasan estas 32 neuronas a las 7 clases que son las que se tienen que clasificar. Se usa la función de activación *softmax* para esta capa de salida.

```
model.compile(loss='categorical_crossentropy', optimizer='Adam', metrics=['acc'])
```

Figura 5.20 Arquitectura red neuronal II.

```
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_3 (Conv2D)            (None, 62, 62, 256)       7168
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 31, 31, 256)       0
_________________________________________________________________
dropout_3 (Dropout)          (None, 31, 31, 256)       0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 29, 29, 128)       295040
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 14, 14, 128)       0
_________________________________________________________________
dropout_4 (Dropout)          (None, 14, 14, 128)       0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 12, 12, 64)        73792
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 6, 6, 64)          0
_________________________________________________________________
dropout_5 (Dropout)          (None, 6, 6, 64)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 2304)              0
_________________________________________________________________
dense_2 (Dense)              (None, 32)                73760
_________________________________________________________________
dense_3 (Dense)              (None, 7)                 231
=================================================================
Total params: 449,991
Trainable params: 449,991
Non-trainable params: 0
_________________________________________________________________
```

Figura 5.21 Resumen red neuronal.

Se han elegido 75 epochs para que así se pueda visualizar la evolución del algoritmo y cómo va refinando conforme aumenten las iteraciones y se corrijan los pesos. Se ajusta el modelo y comienza el entrenamiento.

5.3.3 Análisis de resultados

Para medir el rendimiento de los algoritmos con el conjunto de datos se calcula el *accuracy* o exactitud. Se ha obtenido una exactitud del 75.2 %. Ahora se verá si los resultados son coherentes.

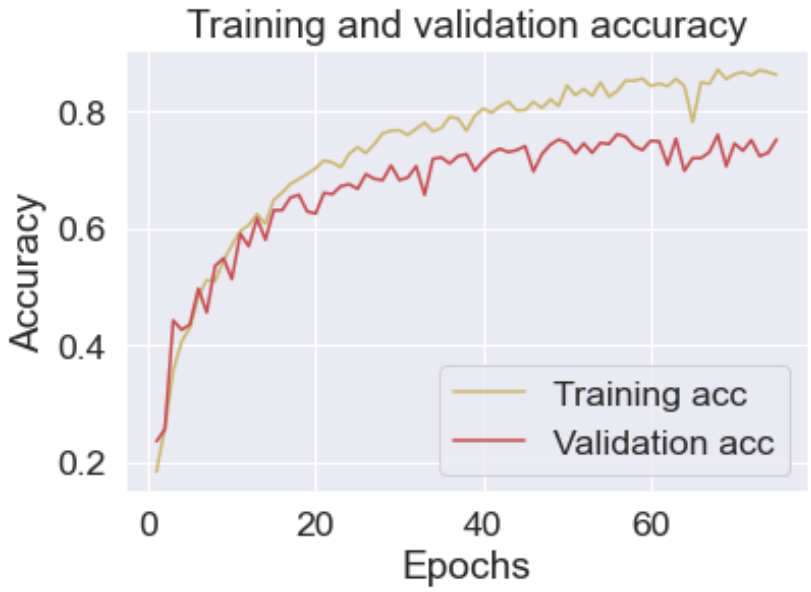

Figura 5.22 Evolución de la exactitud de la red neuronal a medida que aumenta el número de epochs.

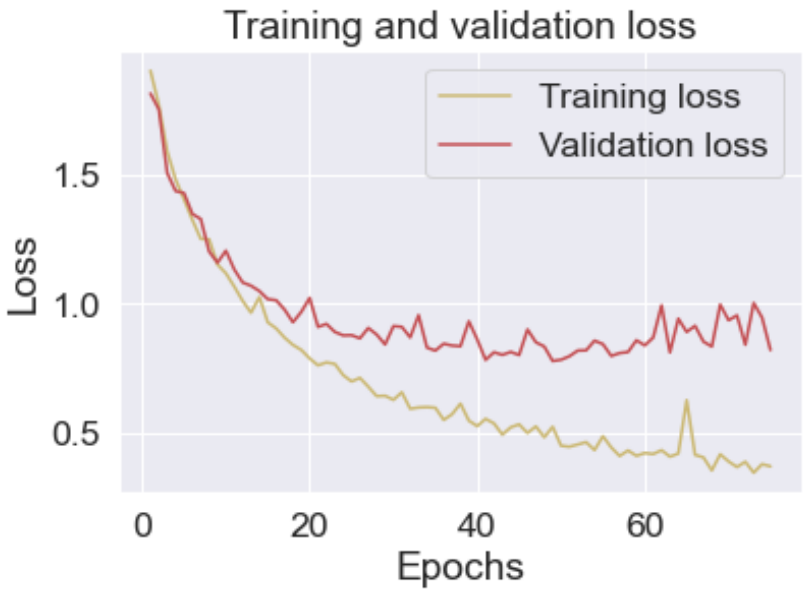

Figura 5.23 Evolución del error de la red neuronal a medida que aumenta el número de epochs.

Ambas representaciones son coherentes debido a que a medida que aumentan los epochs, aumenta la exactitud y disminuye el error. También se puede observar que mientras que el algoritmo en el conjunto de entrenamiento da mejores resultados (la exactitud llega cerca del 90 %), la validación se ha quedado algo estancada. Se podría probar a aumentar al $80-85\,\%$ el porcentaje de los datos de entrenamiento y evaluar cómo responde el modelo.

A continuación, se va a mostrar la matriz de confusión. Esta matriz muestra el número de veces que el algoritmo ha encasillado una imagen en un grupo o en otro. Aquella clasificación que se sitúe en la diagonal será correcta, mientras que la que se sitúe fuera de la diagonal es incorrecta.

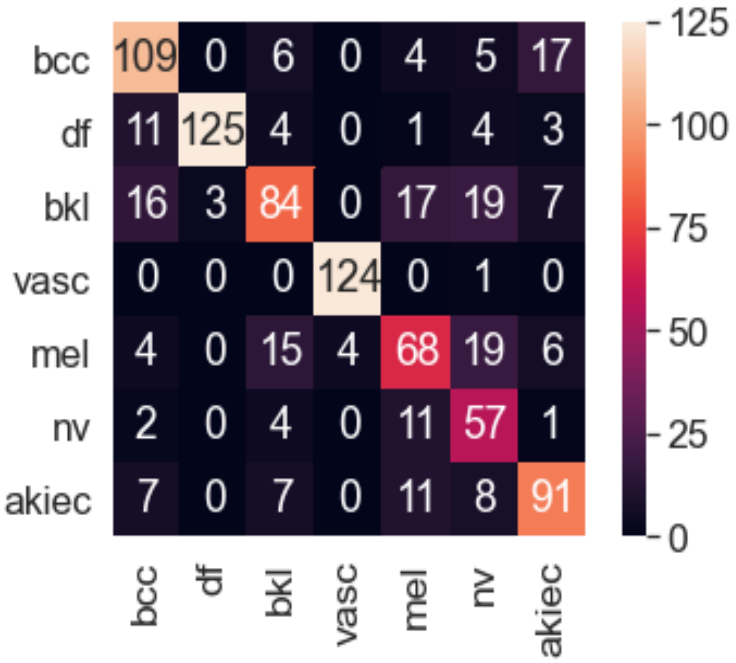

Figura 5.24 Matriz de confusión.

Lesión	Tasa de acierto
Células basales carcínomas	73.15 %
Dermatofibroma	97.65 %
Queratosis benigna	70 %
Lesiones vasculares	96.875 %
Melanoma	73.15 %
Nevus melanocítico	60.714 %
Queratosis actínica	72.8 %

Como se puede observar los resultados no son del todo malos. Es cierto que el nevus melanocítico se lleva en este caso la peor parte. Es por ello que podría estar bien aumentar el número de imágenes de este tipo para que así el algoritmo sea capaz de detectar de manera correcta los rasgos.

A continuación, se va a adjuntar una imagen de la matriz de confusión de una aproximación inicial del algoritmo. En este caso no se habían eliminado las imágenes duplicadas ni se había solucionado el problema del balanceo de casos.

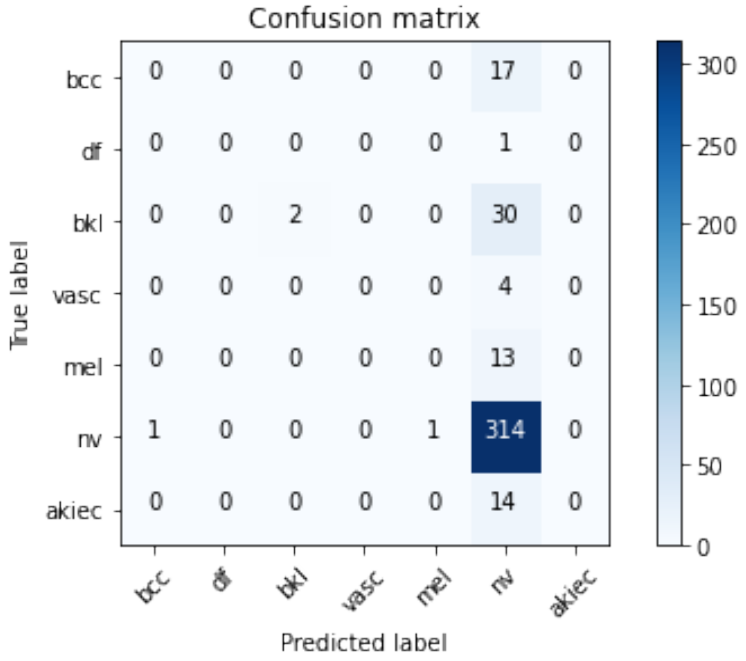

Figura 5.25 Matriz de confusión incorrecta.

Como se puede observar, el algoritmo coloca prácticamente todas las imágenes como nevus melanocítico por lo que, se llegó a la conclusión de que no estaba trabajando correctamente. Es igual que lo que se ha comentado antes, a pesar de que el algoritmo no está trabajando correctamente, la exactitud es elevada debido a que existe un tipo de imágenes que es prácticamente el 80 % de todos los ejemplos.

5.3.4 Método de uso

Como se ha podido observar, el algoritmo acierta en 3 de cada 4 casos en esta aproximación. Mejoraría analizando la red neuronal, aumentando el número de imágenes...

Para utilizarlo lo único que habría que hacer sería subir una imagen de la mancha. Podría diseñarse una aplicación sencilla que al subir la imagen devolviese lo siguiente:

```
La probabilidad que sean células basales carcinomas es del 73.92107 %
La probabilidad que sea dermatofibroma es del 0.00067146344 %
La probabilidad que sea benigno (carcinomas) es del 22.588985 %
La probabilidad que sea una lesión vascular es del 0.07236279 %
La probabilidad que sea  melanoma es del 2.3154552 %
La probabilidad que sea  nevus melanocítico es del 0.34931758 %
La probabilidad que sea  queratosis actínica es del 0.75213933 %
```

Figura 5.26 Respuesta de la aplicación.

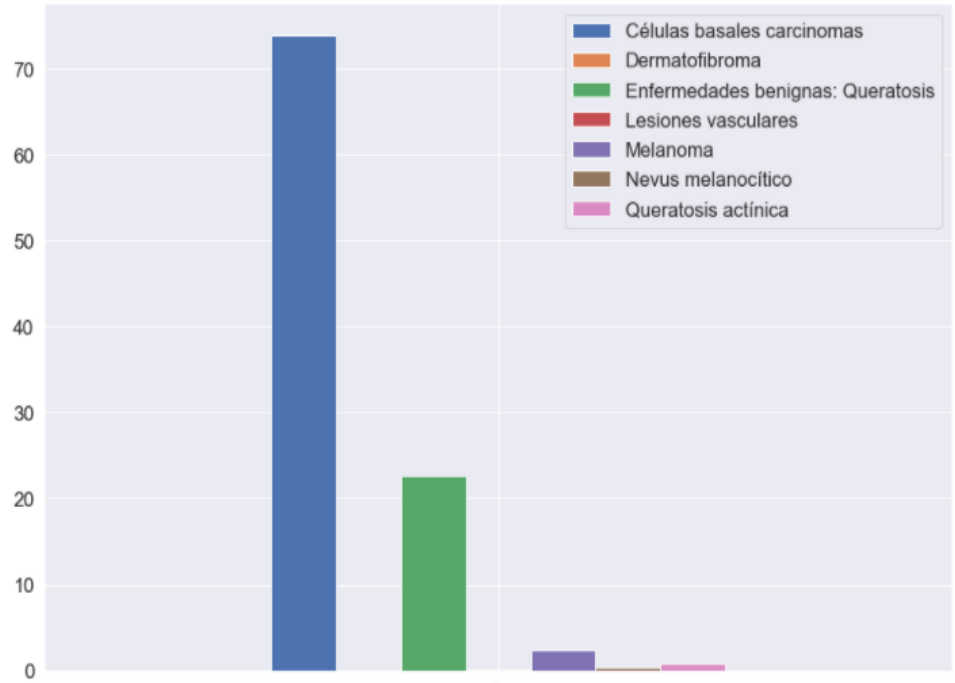

Figura 5.27 Respuesta de la aplicación.

5.3.5 Mejora del modelo

Viendo que la efectividad del modelo es del 75.2 % y observando que en la matriz de confusión algunas tasas de aciertos eran bajas, se ha tratado de mejorar el modelo. Para ello las modificaciones que se han realizado han sido las siguientes:

- Se han reescalado los datos de la siguiente forma:

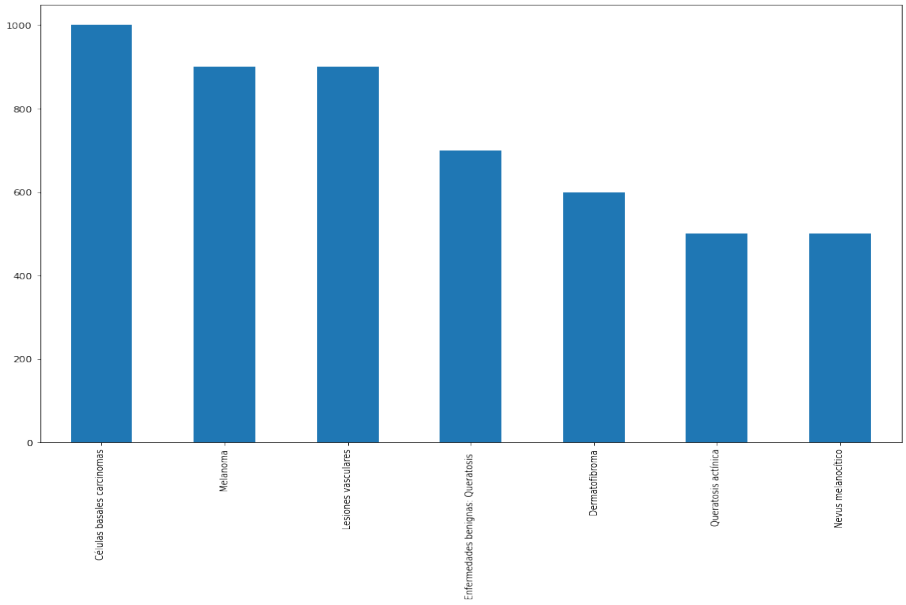

Figura 5.28 Escalado de las imágenes.

  Esto se ha realizado con la intención de aumentar el porcentaje de acierto de aquellas clases que tenían una tasa más baja.
- El conjunto de entrenamiento en vez de ser el 75 % de los datos, esta vez pasa a ser el 80 %.
- Se ha aumentado el número de epochs a 150.

Los resultados obtenidos son los siguientes:

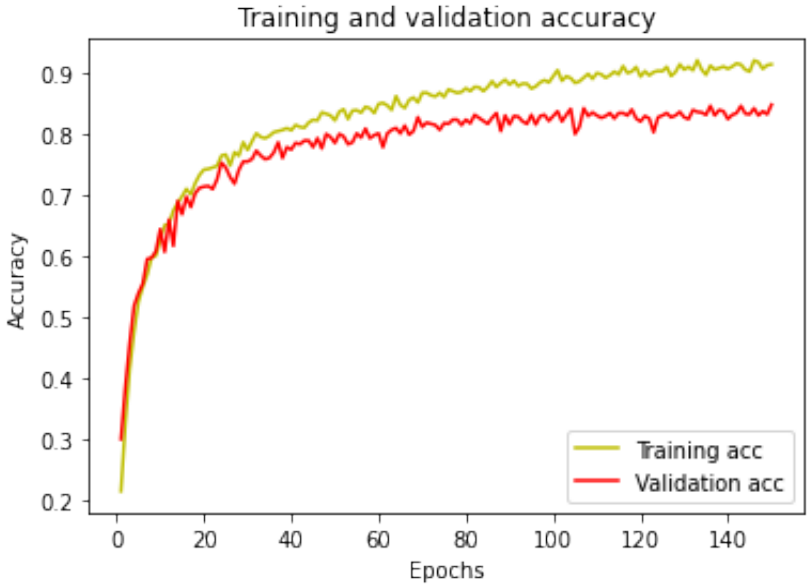

Figura 5.29 Evolución de la exactitud de la red neuronal a medida que aumenta el número de epochs. Caso mejorado.

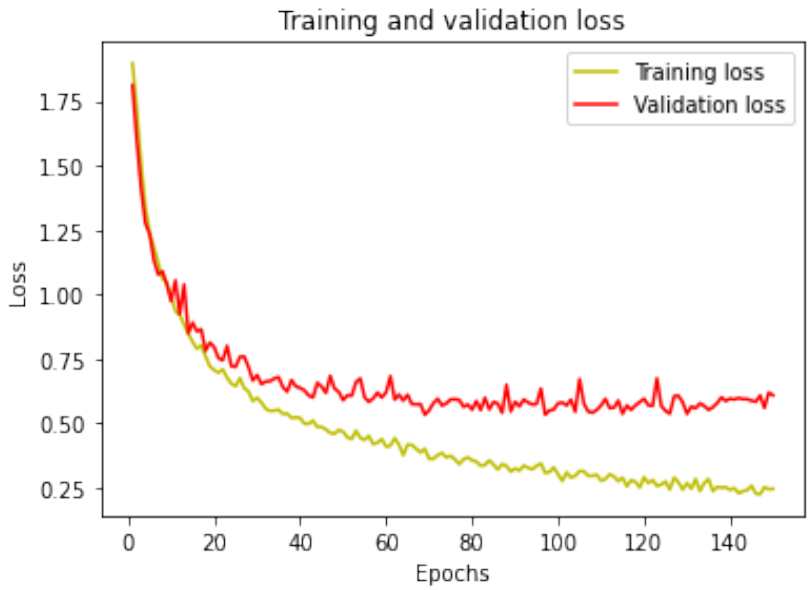

Figura 5.30 Evolución del error de la red neuronal a medida que aumenta el número de epochs. Caso mejorado.

El modelo alcanza una exactitud del 84.90 %, lo que supone una mejora del 13 % con respecto al modelo anterior. La matriz de confusión que se obtiene es la siguiente:

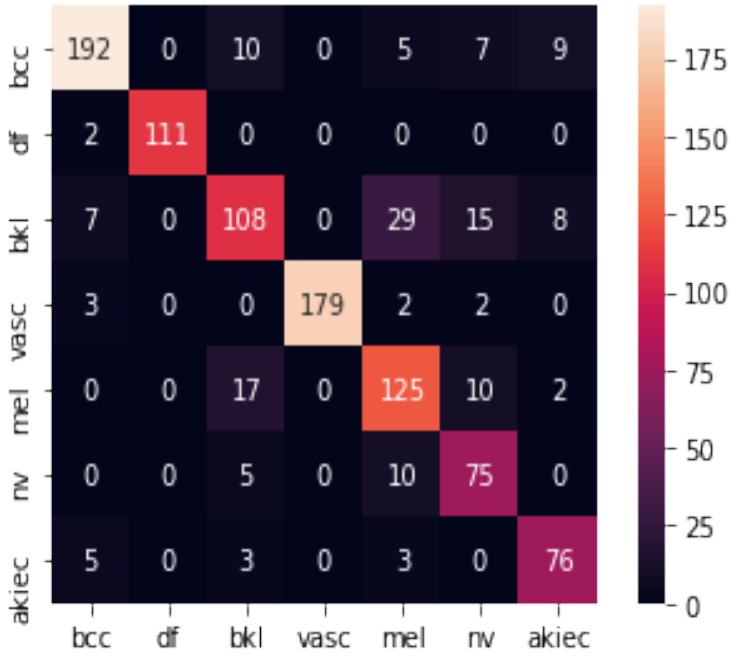

Figura 5.31 Matriz de confusión.

Lesión	Tasa de acierto
Células basales carcínomas	91.866 %
Dermatofibroma	100 %
Queratosis benigna	75.524 %
Lesiones vasculares	100 %
Melanoma	71.839 %
Nevus melanocítico	68.807 %
Queratosis actínica	80 %

Se puede observar cómo los resultados en todas las lesiones han mejorado excepto en el caso del melanoma el cual ha disminuido (aunque nuy poco) la tasa de acierto. El modelo ha mejorado bastante y tiene una tasa de acierto elevada.

6 Conclusiones y líneas de trabajo futuras

Como se ha ido observando a lo largo de todo el trabajo, las técnicas de Aprendizaje Automático tienen unas aplicaciones realmente interesantes. Una vez dominadas dichas técnicas de Machine Learning, gracias al avance de las aplicaciones de programación, así como a la potencia que tienen las librerías de uso libre, se pueden llevar a cabo estudios a un gran nivel de profundidad. Es fundamental entender que este tipo de algoritmos es aplicable en todo tipo de empresas, desde banca, empresas energéticas o redes sociales.
La componente ética y cercana a problemas del día a día como puede ser la predicción de enfermedades en el caso de Medicina, o reducir el número de fallos en vuelo, ayuda a verlo como una herramienta que se debe explotar al máximo para obtener los mejores resultados posibles y de esta forma seguir avanzando. Algo muy importante que hay que tener en cuenta antes de abordar un problema de este tipo, es contar con datos que aporten valor para el problema a resolver y que sean fiables. Es por ello, que si se quisiera aplicar este tipo de algoritmos en la industria aeroespacial se necesitaría disponer de datos de calidad y elevada fiabilidad para poder aplicar los algoritmos que se han ido explicando.

Líneas de trabajo futuras

Dentro de los problemas resueltos en los casos anteriores, para el primero de los casos como se ha mencionado en dicho apartado, se tendrían que tomar datos reales y contrastar lo que se obtiene con el algoritmo con lo que el especialista haría. Del dataset de cáncer de mama se han obtenido muy buenos resultados por lo que, en ese caso, no hay mucho margen de mejora. Por último, el caso de las lesiones cutáneas se mejoraría mediante el aumento de imágenes de aquellas clases menos numerosas para que de esta forma no esté tan desbalanceado el problema y se pueda obtener un mayor rendimiento.

Una vez mejorado el algoritmo, el siguiente paso sería crear una aplicación por cada caso (ya sea una aplicación de ordenador o de teléfono móvil) para poder aplicar estas investigaciones en casos reales.

Anexo

1 Código

Se adjunta a continuación el código empleado para cada uno de los casos estudiados.

1.1 Caso I

```
#!/usr/bin/env python
# coding: utf-8

# In[1]:


import pandas as pd
import numpy as np


# In[2]:


mainpath="/Users/Usuario/Desktop/Skylife/INTRODUCCIÓN/"
filename= "AMPUTACIÓN DE EXTREMIDADES/Hemorragia_1.csv"
fullpath=mainpath+"/"+filename
data = pd.read_csv(fullpath,sep=";")
data = data.drop(['user_id'],axis = 1)
df=data
df_mean=data


# # 1.Centroides fijos

# ### 1.1 Definimos los centroides: casos ideales de cada clase

# In[3]:


data = pd.DataFrame(data).to_numpy()
clust = (data[1,:],data[102,:],data[202,:],data[302,:])
c1_f = clust[0]
c2_f = clust[1]
c3_f = clust[2]
```

```
c4_f = clust[3]
c4_f


# ### 1.2 Normalizamos los datos para calcular correctamente la distancia euclí
    dea

# In[4]:


data_normed = data / data.max(axis=0)
clust_centers = (data_normed[1,:],data_normed[102,:],data_normed[202,:],data_
    normed[302,:])


#    ### 1.3 Utilizamos el método de los k-mean: al tener fijos los centroides,
     aplica distancia euclidea

# In[5]:


from scipy.cluster.vq import vq


# In[56]:


[cluster_f,distance_f] = vq(data_normed, clust_centers)
cluster_f = np.where(cluster_f==0,10,cluster_f)
cluster_f = np.where(cluster_f==1,20, cluster_f)
cluster_f = np.where(cluster_f==2,30, cluster_f)
cluster_f = np.where(cluster_f==3,40, cluster_f)
cluster_f[528]


# # 2. K-Modes

# ## 2.1 Añadimos el data set

# In[56]:


df = data


# ## 2.2 Añadimos K-Modes

# In[57]:


from kmodes.kmodes import KModes
km_cao = KModes(n_clusters=4, init = "Cao", n_init = 10, verbose=1)
fitClusters_cao = km_cao.fit_predict(df)
#fitClusters_cao
```

```
# ### Una vez visualizado la pertenencia de cada paciente a cada cluster, se
    cambian los puntos para que se pueda identificar con las clases

# In[58]:


array1 = fitClusters_cao
array1 = np.where(array1==1,10, array1)
array1 = np.where(array1==3,20, array1)
array1 = np.where(array1==0,30, array1)
array1 = np.where(array1==2,40, array1)
cluster_M = array1
cluster_M


# In[59]:


clusterCentroidsDf = pd.DataFrame(km_cao.cluster_centroids_)
centroid = pd.DataFrame(clusterCentroidsDf).to_numpy()
c1_M = centroid[1]
c2_M = centroid[3]
c3_M = centroid[0]
c4_M = centroid[2]


# # 3.Análisis de resultados

# ## 3.1 Comparación de centroides

# In[60]:


c1_f == c1_M


# In[61]:


c2_f == c2_M


# In[62]:


c3_f == c3_M


# In[63]:


c4_f == c4_M


# ## 3.2 Comparación de los clusters a los que pertenece cada punto

# In[64]:
```

```
medidas_nuevas = cluster_f - cluster_M


# In[65]:


medidas_nuevas = medidas_nuevas [400:]
medidas_nuevas


# ### 3.2.1 Se van a estudiar la desviación que tienen las medidas

# In[66]:


len(medidas_nuevas)


# ### 3.2.2 Coincidencias

# In[67]:


coincidentes = medidas_nuevas[medidas_nuevas==0]


# In[68]:


n_coincidentes = len(coincidentes)
n_coincidentes


# #### Porcentaje de coincidencia

# In[69]:


tasa_coincidencia = (n_coincidentes/len(medidas_nuevas))*100
tasa_coincidencia


# ### 3.2.3 Diferencias de 1 cambio de clase

# In[70]:


una_clase_pos = len(medidas_nuevas[medidas_nuevas==10])
una_clase_neg = len(medidas_nuevas[medidas_nuevas==-10])
una_clase = una_clase_pos + una_clase_neg
#n_una_clase = len(una_clase)
una_clase


# In[71]:
```

```
# Vemos las veces que sale negativo y positivo frente a las totales
(una_clase_neg/una_clase)*100


# In[72]:


(una_clase_pos/una_clase)*100


# In[73]:


tasa_una_clase = (una_clase/len(medidas_nuevas))*100
tasa_una_clase


# ### 3.2.3 Diferencias de 2 cambios de clase

# In[74]:


dos_clases_pos =len(medidas_nuevas[medidas_nuevas==(20)])
dos_clases_neg =len(medidas_nuevas[medidas_nuevas==(-20)])
dos_clases = dos_clases_pos + dos_clases_neg
#n_dos_clases = len(dos_clases)
dos_clases


# In[75]:


(dos_clases_neg/dos_clases)*100


# In[76]:


(dos_clases_pos/dos_clases)*100


# In[77]:


tasa_dos_clases = (dos_clases/len(medidas_nuevas))*100
tasa_dos_clases


# ### 3.2.4 Diferencias de 3 cambios de clase

# In[78]:


tres_clases_pos =len(medidas_nuevas[medidas_nuevas==(30)])
tres_clases_neg =len(medidas_nuevas[medidas_nuevas==(-30)])
```

```
tres_clases = tres_clases_pos + tres_clases_neg
#n_dos_clases = len(dos_clases)
tres_clases


# In[79]:


tasa_tres_clases = (tres_clases/len(medidas_nuevas))*100
tasa_tres_clases


# ## Añadiendo el método de los K-Means cambiando el centroide

# In[59]:


from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters=4).fit(df_mean)
centroids = kmeans.cluster_centers_
centroids = np.around(centroids, decimals=1)
print(centroids)


# In[8]:


c4_mean = [3,3,1,1,3,3,3,2]
c2_mean = [1,1,0,1,1,1,1,1]
c3_mean = [2,2,1,1,2,2,2,1]
c1_mean = [0,0,0,0,0,0,0,0]


# In[9]:


c1_mean == c1_f


# In[10]:


c2_mean == c2_f


# In[11]:


c3_mean == c3_f


# In[12]:


c4_mean == c4_f
```

```
# In[13]:


centroids_normed = centroids / centroids.max(axis=0)


# In[18]:


[cluster_mean,distance_mean] = vq(data_normed, centroids_normed)


# In[19]:


cluster_mean


# In[57]:


cluster_mean = np.where(cluster_mean==1,40,cluster_mean)
cluster_mean = np.where(cluster_mean==2,30, cluster_mean)
cluster_mean = np.where(cluster_mean==3,10, cluster_mean)
cluster_mean = np.where(cluster_mean==0,20, cluster_mean)
cluster_mean[528]


# In[35]:


medidas_nuevas_mean = cluster_f - cluster_mean
medidas_nuevas_mean1 = medidas_nuevas_mean
medidas_nuevas_mean = medidas_nuevas_mean [400:]
medidas_nuevas_mean


# In[36]:


len(medidas_nuevas_mean)


# In[37]:


coincidentes_mean = medidas_nuevas_mean[medidas_nuevas_mean==0]
n_coincidentes_mean = len(coincidentes_mean)
n_coincidentes_mean


# In[38]:


tasa_coincidencia_mean = (n_coincidentes_mean/len(medidas_nuevas_mean))*100
tasa_coincidencia_mean
```

```
# In[39]:


una_clase_pos_mean = len(medidas_nuevas_mean[medidas_nuevas_mean==10])
una_clase_neg_mean = len(medidas_nuevas_mean[medidas_nuevas_mean==-10])
una_clase_mean = una_clase_pos_mean + una_clase_neg_mean
#n_una_clase = len(una_clase)
una_clase_mean


# In[40]:


(una_clase_mean/len(medidas_nuevas_mean))*100


# In[41]:


dos_clase_pos_mean = len(medidas_nuevas_mean[medidas_nuevas_mean==20])
dos_clase_neg_mean = len(medidas_nuevas_mean[medidas_nuevas_mean==-20])
dos_clase_mean = dos_clase_pos_mean + dos_clase_neg_mean
#n_una_clase = len(una_clase)
dos_clase_mean
(dos_clase_mean/len(medidas_nuevas_mean))*100


# In[42]:


dos_clase_mean


# In[43]:


tasa_coincidencia_mean+(una_clase_mean/len(medidas_nuevas_mean))*100+(dos_clase
    _mean/len(medidas_nuevas_mean))*100


# In[45]:


#medidas_nuevas_mean1
medidas_nuevas_mean1[np.where(medidas_nuevas_mean1 == 20)]


# In[47]:


np.where(medidas_nuevas_mean1 == -20)


# In[48]:
```

```
np.where(medidas_nuevas_mean1 == 0)


# In[ ]:
```

1.2 Caso II

```
#!/usr/bin/env python
# coding: utf-8

# In[1]:


import numpy as np
from sklearn import neighbors,preprocessing
from sklearn.model_selection import train_test_split
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns; sns.set()
sns.set_style('darkgrid')


# In[2]:


from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import LinearSVC, SVC
from sklearn.neural_network import MLPClassifier
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier


# In[3]:


mainpath="/Users/Usuario/Desktop/Skylife/AI_MEDICINE/"
filename= "CANCER_DE_MAMA/cancer/breast-cancer-wisconsin.data.txt"
fullpath=mainpath+"/"+filename
data = pd.read_csv(fullpath,header=None)
data.describe()


# In[4]:


mainpath="/Users/Usuario/Desktop/Skylife/AI_MEDICINE/"
filename= "CANCER_DE_MAMA/cancer/breast-cancer-wisconsin.data.txt"
fullpath=mainpath+"/"+filename
data = pd.read_csv(fullpath,header=None)
data = data.drop([0],axis = 1)
data.head()


# In[5]:


data.describe()
```

```
# In[6]:


data.columns = ["V1","V2","V3","V4","V5","V6","V7","V8","V9","Class"]
data


# In[7]:


data.replace("?",-99999999,inplace = True)


# ### Exploración de los datos

# In[8]:


dataframe = data.set_index(data["Class"])


# In[9]:


data_benigno = dataframe.loc[2]


# In[10]:


data_benigno.shape


# In[11]:


data_benigno.describe()


# In[12]:


data_maligno = dataframe.loc[4]


# In[13]:


data_maligno.shape


# In[14]:


data_maligno.describe()
```

```
# In[15]:


plt.figure(figsize = (8,8))

plt.pie(data['Class'].value_counts(),autopct = '%.2f%%',labels=["Benigno","
    Maligno"])

plt.title("Distribución de pacientes")


# In[16]:


pip install -U seaborn


# In[17]:


plt.figure(figsize = (25,15))
for i, column in enumerate(data.columns):
    plt.subplot(4,6,i+1)
    sns.boxplot(data=data[column])
    plt.title(column)
plt.show()


# In[18]:


corr = data.corr()
plt.figure(figsize = (24,20))
sns.heatmap(corr, annot = True, vmin = -1.0, cmap = 'mako')
plt.title("Matriz de correlacion")
plt.show()


# In[19]:


X = data[["V1","V2","V3","V4","V5","V6","V7","V8","V9"]]
data['Class'] = data['Class'].replace({
       2: "Benigno",
       4: "Maligno",
    })
Y = data["Class"]
Y


# In[20]:


X_train, X_test, Y_train, Y_test = train_test_split(X,Y,test_size = 0.2)


# In[21]:
```

```
scaler = StandardScaler()
scaler.fit(X_train)
X_train = pd.DataFrame(scaler.transform(X_train), index=X_train.index, columns=
    X_train.columns)
X_test = pd.DataFrame(scaler.transform(X_test), index=X_test.index, columns=X_
    test.columns)
X_train


# In[22]:


models = {
    "                 Logistic Regression": LogisticRegression(),
    "                 K-Nearest Neighbors": KNeighborsClassifier(),
    "                      Decision Tree": DecisionTreeClassifier(),
    "Support Vector Machine (Linear Kernel)": LinearSVC(),
    "   Support Vector Machine (RBF Kernel)": SVC(),
    "                     Neural Network": MLPClassifier(),
    "                      Random Forest": RandomForestClassifier(),
    "                   Gradient Boosting": GradientBoostingClassifier(),
}

for name, model in models.items():
    model.fit(X_train, Y_train)
    print(name + " trained.")


# In[23]:


for name, model in models.items():
    print(name + ": {:.2f}%".format(model.score(X_test, Y_test) * 100))


# In[24]:


classification = neighbors.KNeighborsClassifier()
classification.fit(X_train,Y_train)


# In[25]:


accuracy = classification.score(X_test,Y_test)
accuracy


# In[26]:


medida_nueva = np.array([4,4,4,2,2,3,2,1,1])
medida_nueva = medida_nueva.reshape(1,-1)
```

```
# In[27]:


predict = classification.predict(medida_nueva)


# In[28]:


predict


# In[29]:


from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters=2).fit(X)
centroids = kmeans.cluster_centers_
centroids


# In[30]:


labels = kmeans.predict(X)


# In[31]:


labels


# In[32]:


Y1 = Y.to_numpy()
Y1
```

1.3 Caso III

Primer resultado que se muestra

```
#!/usr/bin/env python
# coding: utf-8

# ## 1. Importe de librerías

# In[1]:


import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import os
import glob
import seaborn as sns
from PIL import Image
np.random.seed(123)
from sklearn.preprocessing import label_binarize
from sklearn.metrics import confusion_matrix
import itertools
from sklearn.model_selection import train_test_split

import keras
from keras.utils.np_utils import to_categorical # used for converting labels to
     one-hot-encoding
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPool2D
from keras import backend as K
import itertools
from keras.layers.normalization import BatchNormalization
from keras.utils.np_utils import to_categorical # convert to one-hot-encoding
from tensorflow.keras.callbacks import *

from keras.optimizers import Adam
from keras.preprocessing.image import ImageDataGenerator
from keras.callbacks import ReduceLROnPlateau


# ## 2. Carga de imágenes

# In[2]:


mainpath="/Users/Usuario/Desktop/Skylife/SKIN_CANCER"
filename= "skin-cancer-mnist-ham10000"
path = mainpath+"/"+filename
path_imagenes = {os.path.splitext(os.path.basename(x))[0]:x for x in glob.glob(
    os.path.join(path,'*','*.jpg'))}


# ## 3. Carga del documento

# In[3]:
```

```
mainpath ="/Users/Usuario/Desktop/Skylife/SKIN_CANCER"
filename = "skin-cancer-mnist-ham10000/HAM10000_metadata.csv"
fullpath = mainpath+"/"+filename
skin_df = pd.read_csv(fullpath)


# In[4]:


# Visualizamos cómo queda el dataset
skin_df.head()


# In[5]:


# Introducimos el path recorriendo el path de imágenes creado con la image_id
skin_df['path'] = skin_df['image_id'].map(path_imagenes.get)


# In[6]:


# Introducimos otra columna para ver el nombre completo de la lesión
tipo_lesion = {
    'nv':'Nevus melanocítico',
    'mel': 'Melanoma',
    'bkl': 'Enfermedades benignas: Queratosis ',
    'bcc': 'Células basales carcinomas',
    'akiec': 'Queratosis actínica',
    'vasc': 'Lesiones vasculares',
    'df': 'Dermatofibroma'
}
skin_df['cell_type'] = skin_df['dx'].map(tipo_lesion.get)


# In[7]:


# Pasamos las distintas cell_type a categorías
skin_df['cell_type_idx'] = pd.Categorical(skin_df['cell_type']).codes


# ## Exploración del dataset: limpieza y wrangling

# In[8]:


skin_df.isnull().sum()


# In[9]:


skin_df['age'].fillna((skin_df['age'].mean()), inplace=True)
skin_df.isnull().sum()
```

```
# In[10]:


print(skin_df.dtypes)


# In[11]:


SIZE = 64


# In[12]:


skin_df['image'] = skin_df['path'].map(lambda x: np.array(Image.open(x).resize
    ((SIZE,SIZE))))


# In[61]:


skin_df.head()


# In[13]:


skin_df['image'].map(lambda x: x.shape).value_counts()


# In[14]:


skin_df['image']


# ## Eliminación de imágenes duplicadas

# In[15]:


# this will tell us how many images are associated with each lesion_id
df_undup = skin_df.groupby('lesion_id').count()
# now we filter out lesion_id's that have only one image associated with it
df_undup = df_undup[df_undup['image_id'] == 1]
df_undup.reset_index(inplace=True)
df_undup.head()


# In[16]:


def get_duplicates(x):
    unique_list = list(df_undup['lesion_id'])
```

```
    if x in unique_list:
        return 'unduplicated'
    else:
        return 'duplicated'

skin_df['duplicates'] = skin_df['lesion_id']
skin_df['duplicates'] = skin_df['duplicates'].apply(get_duplicates)


# In[17]:


skin_df['duplicates'].value_counts()
skin_df_nd = skin_df[skin_df['duplicates'] == 'unduplicated']


# In[18]:


skin_df['duplicates'].value_counts()


# In[19]:


# Así queda el nº de casos
skin_df_nd['cell_type_idx'].value_counts()


# In[20]:


# Se puede observar cómo está desbalanceado
fig, ax1 = plt.subplots(1, 1, figsize= (15, 10))
skin_df_nd['cell_type'].value_counts().plot(kind='bar', ax=ax1)


# In[21]:


# Se va a proceder a reescalar los casos
from sklearn.utils import resample

df_0 = skin_df[skin_df.cell_type_idx == 0]
df_1 = skin_df[skin_df.cell_type_idx == 1]
df_2 = skin_df[skin_df.cell_type_idx == 2]
df_3 = skin_df[skin_df.cell_type_idx == 3]
df_4 = skin_df[skin_df.cell_type_idx == 4]
df_5 = skin_df[skin_df.cell_type_idx == 5]
df_6 = skin_df[skin_df.cell_type_idx == 6]


# In[22]:



n_samples = 500
```

```
df_0_balanced = resample(df_0, replace=True, n_samples=n_samples, random_state
    =42)
df_1_balanced = resample(df_1, replace=True, n_samples=n_samples, random_state
    =42)
df_2_balanced = resample(df_2, replace=True, n_samples=n_samples, random_state
    =42)
df_3_balanced = resample(df_3, replace=True, n_samples=n_samples, random_state
    =42)
df_4_balanced = resample(df_4, replace=True, n_samples=n_samples, random_state
    =42)
df_5_balanced = resample(df_5, replace=True, n_samples=n_samples, random_state
    =42)
df_6_balanced = resample(df_6, replace=True, n_samples=n_samples, random_state
    =42)


skin_df_balanced = pd.concat([df_0_balanced, df_1_balanced,
                          df_2_balanced, df_3_balanced,
                          df_4_balanced, df_5_balanced, df_6_balanced])


# In[24]:


# Se puede observar cómo está desbalanceado
fig, ax2 = plt.subplots(1, 1, figsize= (15, 10))
skin_df_balanced['cell_type'].value_counts().plot(kind='bar', ax=ax2)


# In[73]:


skin_df_balanced[skin_df_balanced.cell_type_idx == 1]


# In[74]:


# Se puede observar cómo ahora sí nos queda el dataset balanceado
skin_df_balanced['cell_type'].value_counts()


# ## Preparamos los datos para entrenar el modelo

# In[75]:


y = skin_df_balanced['cell_type_idx']
y.shape


# In[76]:


y = to_categorical(y, num_classes = 7)
y.shape
```

```
# In[77]:


X = np.asarray(skin_df_balanced['image'].tolist())
X = X/255.


# In[78]:


x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.25,
    random_state=42)


# In[79]:


model = Sequential()
model.add(Conv2D(256, (3, 3), activation="relu", input_shape=(SIZE, SIZE, 3)))
#model.add(BatchNormalization())
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Dropout(0.3))

model.add(Conv2D(128, (3, 3),activation='relu'))
#model.add(BatchNormalization())
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Dropout(0.3))

model.add(Conv2D(64, (3, 3),activation='relu'))
#model.add(BatchNormalization())
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Dropout(0.3))
model.add(Flatten())

model.add(Dense(32))
model.add(Dense(7, activation='softmax'))
model.summary()


# In[80]:


model.compile(loss='categorical_crossentropy', optimizer='Adam', metrics=['acc
    '])


# In[81]:


model.summary()


# In[82]:


batch_size = 16
```

```
epochs = 75


# In[83]:


history = model.fit(
    x_train, y_train,
    epochs=epochs,
    batch_size = batch_size,
    validation_data=(x_test, y_test),
    verbose=2)aju


# In[160]:


score = model.evaluate(x_test, y_test)
print('Test accuracy:', score[1])


# In[85]:


#plot the training and validation accuracy and loss at each epoch
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(loss) + 1)
plt.plot(epochs, loss, 'y', label='Training loss')
plt.plot(epochs, val_loss, 'r', label='Validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()


# In[86]:


acc = history.history['acc']
val_acc = history.history['val_acc']
plt.plot(epochs, acc, 'y', label='Training acc')
plt.plot(epochs, val_acc, 'r', label='Validation acc')
plt.title('Training and validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()


# In[159]:


# Prediction on test data
y_pred = model.predict(x_test)
# Convert predictions classes to one hot vectors
```

```
y_pred_classes = np.argmax(y_pred, axis = 1)
# Convert test data to one hot vectors
y_true = np.argmax(y_test, axis = 1)


# In[88]:


from sklearn.metrics import confusion_matrix
confusion_mtx = confusion_matrix(y_true, y_pred_classes)
# plot the confusion matrix
plot_labels = ['bcc', 'df', 'bkl', 'vasc', 'mel', 'nv','akiec']
plot_confusion_matrix(confusion_mtx, classes = plot_labels)


# In[100]:


sns.heatmap(confusion_mtx.T, square=True, annot=True, fmt='d', cbar=True,
           xticklabels=plot_labels, yticklabels=plot_labels )


# In[140]:


prediccion = y_pred[1]*100
x_test[1]


# In[121]:


print('La probabilidad que sean células basales carcinomas es del',prediccion
    [0],'%')
print('La probabilidad que sea dermatofibroma es del',prediccion[1],'%')
print('La probabilidad que sea benigno (carcinomas) es del',prediccion[2],'%')
print('La probabilidad que sea una lesión vascular es del',prediccion[3],'%')
print('La probabilidad que sea melanoma es del',prediccion[4],'%')
print('La probabilidad que sea nevus melanocítico es del',prediccion[5],'%')
print('La probabilidad que sea queratosis actínica es del',prediccion[6],'%')


# In[126]:


columns = ['Células basales carcinomas','Dermatofibroma','Benigna',
                                      'Lesiones vasculares','Melanoma','Nevus
                                          melanocítico','Queratosis actínica']


# In[135]:


df1 = pd.DataFrame(prediccion,columns)
df1_transposed = df1.T
df1_transposed
```

```
# In[137]:


fig, ax2 = plt.subplots(1, 1, figsize= (15, 10))
df1_transposed.plot(kind='bar', ax=ax2)


# In[158]:


y_test


# In[155]:



y_pred2 = np.argmax(y_pred,axis=1)

y_test2 = np.argmax(y_test,axis=1)


# In[157]:


plt.figure(figsize=(16,16))
for i in range(9):
    plt.subplot(3,3,i+1)
    index = i+100
    plt.imshow(x_test[index,:,:,::-1])
    label_exp = columns[y_test2[index]] #expected label
    label_pred = columns[y_pred2[index]] #predicted label
    label_pred_prob = round(np.max(y_pred1[index])*100)
    plt.title('Expected:'+str(label_exp)+'\n Pred.:'+str(label_pred)+' ('+str(
        label_pred_prob)+'%)')
plt.ylabel('')
plt.tight_layout()
plt.savefig('final_figure.png',dpi=300)
plt.show()


# In[ ]:
```

Mejora del resultado anterior

```
#!/usr/bin/env python
# coding: utf-8

# ## 1. Importe de librerías

# In[1]:


import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import os
import glob
import seaborn as sns
from PIL import Image
np.random.seed(123)
from sklearn.preprocessing import label_binarize
from sklearn.metrics import confusion_matrix
import itertools
from sklearn.model_selection import train_test_split

import keras
from keras.utils.np_utils import to_categorical # used for converting labels to
     one-hot-encoding
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPool2D
from keras import backend as K
import itertools
from keras.layers.normalization import BatchNormalization
from keras.utils.np_utils import to_categorical # convert to one-hot-encoding
from tensorflow.keras.callbacks import *

from keras.optimizers import Adam
from keras.preprocessing.image import ImageDataGenerator
from keras.callbacks import ReduceLROnPlateau


# ## 2. Carga de imágenes

# In[2]:


mainpath="/Users/Usuario/Desktop/Skylife/SKIN_CANCER"
filename= "skin-cancer-mnist-ham10000"
path = mainpath+"/"+filename
path_imagenes = {os.path.splitext(os.path.basename(x))[0]:x for x in glob.glob(
    os.path.join(path,'*','*.jpg'))}


# ## 3. Carga del documento

# In[3]:
```

```
mainpath ="/Users/Usuario/Desktop/Skylife/SKIN_CANCER"
filename = "skin-cancer-mnist-ham10000/HAM10000_metadata.csv"
fullpath = mainpath+"/"+filename
skin_df = pd.read_csv(fullpath)


# In[4]:


# Visualizamos cómo queda el dataset
skin_df.head()


# In[5]:


# Introducimos el path recorriendo el path de imágenes creado con la image_id
skin_df['path'] = skin_df['image_id'].map(path_imagenes.get)


# In[6]:


# Introducimos otra columna para ver el nombre completo de la lesión
tipo_lesion = {
    'nv':'Nevus melanocítico',
    'mel': 'Melanoma',
    'bkl': 'Enfermedades benignas: Queratosis ',
    'bcc': 'Células basales carcinomas',
    'akiec': 'Queratosis actínica',
    'vasc': 'Lesiones vasculares',
    'df': 'Dermatofibroma'
}
skin_df['cell_type'] = skin_df['dx'].map(tipo_lesion.get)


# In[7]:


# Pasamos las distintas cell_type a categorías
skin_df['cell_type_idx'] = pd.Categorical(skin_df['cell_type']).codes


# ## Exploración del dataset: limpieza y wrangling

# In[8]:


skin_df.isnull().sum()


# In[9]:


skin_df['age'].fillna((skin_df['age'].mean()), inplace=True)
skin_df.isnull().sum()
```

```
# In[10]:


print(skin_df.dtypes)


# In[11]:


SIZE = 64


# In[12]:


skin_df['image'] = skin_df['path'].map(lambda x: np.array(Image.open(x).resize
    ((SIZE,SIZE))))


# In[13]:


skin_df.head()


# In[14]:


skin_df['image'].map(lambda x: x.shape).value_counts()


# In[15]:


skin_df['image']


# ## Eliminación de imágenes duplicadas

# In[16]:


# this will tell us how many images are associated with each lesion_id
df_undup = skin_df.groupby('lesion_id').count()
# now we filter out lesion_id's that have only one image associated with it
df_undup = df_undup[df_undup['image_id'] == 1]
df_undup.reset_index(inplace=True)
df_undup.head()


# In[17]:


def get_duplicates(x):
    unique_list = list(df_undup['lesion_id'])
    if x in unique_list:
```

```
        return 'unduplicated'
    else:
        return 'duplicated'

skin_df['duplicates'] = skin_df['lesion_id']
skin_df['duplicates'] = skin_df['duplicates'].apply(get_duplicates)


# In[18]:


skin_df['duplicates'].value_counts()
skin_df_nd = skin_df[skin_df['duplicates'] == 'unduplicated']


# In[19]:


skin_df['duplicates'].value_counts()


# In[20]:


# Así queda el nº de casos
skin_df_nd['cell_type_idx'].value_counts()


# In[21]:


# Se puede observar cómo está desbalanceado
fig, ax1 = plt.subplots(1, 1, figsize= (15, 10))
skin_df_nd['cell_type'].value_counts().plot(kind='bar', ax=ax1)


# In[22]:


# Se va a proceder a reescalar los casos
from sklearn.utils import resample

df_0 = skin_df[skin_df.cell_type_idx == 0]
df_1 = skin_df[skin_df.cell_type_idx == 1]
df_2 = skin_df[skin_df.cell_type_idx == 2]
df_3 = skin_df[skin_df.cell_type_idx == 3]
df_4 = skin_df[skin_df.cell_type_idx == 4]
df_5 = skin_df[skin_df.cell_type_idx == 5]
df_6 = skin_df[skin_df.cell_type_idx == 6]


# In[23]:



df_0_balanced = resample(df_0, replace=True, n_samples=1000, random_state=42)
df_1_balanced = resample(df_1, replace=True, n_samples=600, random_state=42)
```

```
df_2_balanced = resample(df_2, replace=True, n_samples=700, random_state=42)
df_3_balanced = resample(df_3, replace=True, n_samples=900, random_state=42)
df_4_balanced = resample(df_4, replace=True, n_samples=900, random_state=42)
df_5_balanced = resample(df_5, replace=True, n_samples=500, random_state=42)
df_6_balanced = resample(df_6, replace=True, n_samples=500, random_state=42)


skin_df_balanced = pd.concat([df_0_balanced, df_1_balanced,
                          df_2_balanced, df_3_balanced,
                          df_4_balanced, df_5_balanced, df_6_balanced])


# In[24]:


# Se puede observar cómo está desbalanceado
fig, ax2 = plt.subplots(1, 1, figsize= (15, 10))
skin_df_balanced['cell_type'].value_counts().plot(kind='bar', ax=ax2)


# In[26]:


skin_df_balanced[skin_df_balanced.cell_type_idx == 1]


# In[27]:


# Se puede observar cómo ahora sí nos queda el dataset balanceado
skin_df_balanced['cell_type'].value_counts()


# ## Preparamos los datos para entrenar el modelo

# In[28]:


y = skin_df_balanced['cell_type_idx']
y.shape


# In[29]:


y = to_categorical(y, num_classes = 7)
y.shape


# In[30]:


X = np.asarray(skin_df_balanced['image'].tolist())
X = X/255.


# In[31]:
```

```
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random
    _state=423)


# In[32]:


model = Sequential()
model.add(Conv2D(256, (3, 3), activation="relu", input_shape=(SIZE, SIZE, 3)))
#model.add(BatchNormalization())
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Dropout(0.3))

model.add(Conv2D(128, (3, 3),activation='relu'))
#model.add(BatchNormalization())
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Dropout(0.3))

model.add(Conv2D(64, (3, 3),activation='relu'))
#model.add(BatchNormalization())
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Dropout(0.3))
model.add(Flatten())

model.add(Dense(32))
model.add(Dense(7, activation='softmax'))
model.summary()


# In[33]:


model.compile(loss='categorical_crossentropy', optimizer='Adam', metrics=['acc
    '])


# In[34]:


model.summary()


# In[36]:


batch_size = 16
epochs = 150


# In[37]:


history = model.fit(
    x_train, y_train,
    epochs=epochs,
```

```
    batch_size = batch_size,
    validation_data=(x_test, y_test),
    verbose=2)


# In[38]:


score = model.evaluate(x_test, y_test)
print('Test accuracy:', score[1])


# In[39]:


#plot the training and validation accuracy and loss at each epoch
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(loss) + 1)
plt.plot(epochs, loss, 'y', label='Training loss')
plt.plot(epochs, val_loss, 'r', label='Validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()


# In[40]:


acc = history.history['acc']
val_acc = history.history['val_acc']
plt.plot(epochs, acc, 'y', label='Training acc')
plt.plot(epochs, val_acc, 'r', label='Validation acc')
plt.title('Training and validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()


# In[41]:


# Prediction on test data
y_pred = model.predict(x_test)
# Convert predictions classes to one hot vectors
y_pred_classes = np.argmax(y_pred, axis = 1)
# Convert test data to one hot vectors
y_true = np.argmax(y_test, axis = 1)


# In[44]:


from sklearn.metrics import confusion_matrix
```

```
confusion_mtx = confusion_matrix(y_true, y_pred_classes)
sns.heatmap(confusion_mtx.T, square=True, annot=True, fmt='d', cbar=True,
           xticklabels=plot_labels, yticklabels=plot_labels )
```

Bibliografía

[1] *https://computersciencewiki.org/index.php/Max-pooling_/_Pooling*.

[2] *https://www.aprendemachinelearning.com/como-funcionan-las-convolutional-neural-networks-vision-por-ordenador/*.

[3] *https://www.cienciadedatos.net/documentos/py09_gradient_boosting_python.html*.

[4] *Arquitectura red neuronal*, *https://es.wikipedia.org/wiki/Archivo:RedNeuronalArtificial.png*.

[5] *Errores regresión lineal*, *http://www.sthda.com/english/sthda-upload/images/machine-learning-essentials/linear-regression.png*.

[6] *k-means clustering*, *https://en.wikipedia.org/wiki/K-means_clustering*.

[7] *Knn*, *https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm#/media/File:KnnClassification.svg*.

[8] *Regresión lineal*, *https://en.wikipedia.org/wiki/Linear_regression#/media/File:Linear_regression.svg*.

[9] *Regresión logística*, *https://en.wikipedia.org/wiki/Logistic_regression#/media/File:Exam_pass_logistic_curve.jpeg*.

[10] *Svm*, *https://en.wikipedia.org/wiki/Support-vector_machine*.

[11] Samet Ayhan, *DART: A Machine-Learning Approach to Trajectory Prediction and Demand-Capacity Balancing*, 2017.

[12] BBVA, *'machine learning': ¿qué es y cómo funciona?*, *https://www.bbva.com/es/machine-learning-que-es-y-como-funciona/*.

[13] Nadia Berchane, *https://master-iesc-angers.com/artificial-intelligence-machine-learning-and-deep-learning-same-context-different-concepts/*.

[14] Builtin, *https://builtin.com/data-science/random-forest-algorithm*.

[15] Madison Wisconsin USA Dr. WIlliam H. Wolberg (physician), University of Wisconsin Hospitals, *Breast cancer wisconsin (original) data set*, 1992, *https://archive.ics.uci.edu/ml/datasets/breast+cancer+wisconsin+(original)*.

[16] Dr.Juan Francisco García Regalado Dra. Norma de la Torre Peredo, *Escalas de valoración del paciente traumatológico*, 2016.

[17] Felix García, *ClasificaciÓn de fallas en rodamientos de mÁquinas rotativas utilizando aprendizaje de mÁquinas*, *https://www.researchgate.net/figure/Figura-40-a-Matriz-de-confusion-sin-normalizar-b-Matriz-de-confusion-normalizada_fig11_334821433*.

[18] José David Villanueva García, *Redes neuronales desde cero (i) – introducción*, *https://www.iartificial.net/redes-neuronales-desde-cero-i-introduccion/*.

[19] Research Gate, *Árbol de decisión*, *https://www.researchgate.net/figure/Basic-structure-of-a-decision-tree-All-decision-trees-are-built-through-recursion_fig3_295860754*.

[20] Craig Lambert Ioana Bica, Ahmed M. Alaa and Mihaela van der Schaar, *From real-world patient data to individualized treatment effects using machine learning: Current and future methods to address underlying challenges*, *https://ascpt.onlinelibrary.wiley.com/doi/epdf/10.1002/cpt.1907*.

[21] Keras, *https://keras.io/*.

[22] Celik S. Logsdon Lee, SI., *Machine learning approach to integrate big data for precision medicine in acute myeloid leukemia*, *https://doi.org/10.1038/s41467-017-02465-5*.

[23] Carlos García Moreno, *¿qué es el deep learning y para qué sirve?*, *https://www.indracompany.com/es/blogneo/deep-learning-sirve*.

[24] S. B. Kotsiantis Department of Computer Science and Greece Technology University of Peloponnese, *Supervised machine learning: A review of classification techniques*, *http://www.informatica.si/index.php/informatica/article/viewFile/148/140*.

[25] Jonathan Ramirez, *K-means: Elbow method and silhouette*, *https://medium.com/@jonathanrmzg/k-means-elbow-method-and-silhouette-e565d7ab87aa*.

[26] R^2 Data Labs.Rolls Royce, *https://www.rolls-royce.com/products-and-services/r2datalabs.aspx*.

[27] Sumit Saha, *A comprehensive guide to convolutional neural networks — the eli5 way*, *https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53*.

[28] Agnes Sauer, *Quick guide to gradient descent and it's variants*, *https://morioh.com/p/15c995420be6*.

[29] Scikit-Learn, *https://scikit-learn.org/stable/*.

[30] Skylife Engineering S.L, *Proyecto harvis*, *https://skylife-eng.com/projects/harvis-es/*.

[31] Philipp Tschandl, *The HAM10000 dataset, a large collection of multi-source dermatoscopic images of common pigmented skin lesions*, 2018, https://doi.org/10.7910/DVN/DBW86T.

[32] Hospital Universitario Ramón y Cajal, *Curvas roc*, *http://www.hrc.es/bioest/roc_1.html*.