UNIVERSIDAD MIGUEL HERNÁNDEZ DE ELCHE

ESCUELA POLITÉCNICA SUPERIOR DE ELCHE

GRADO EN INGENIERÍA INFORMÁTICA EN
TECNOLOGÍAS DE LA INFORMACIÓN

"SEEK HEALTH: SISTEMA INTELIGENTE BASADO EN DEEP LEARNING PARA LA AYUDA EN EL DIAGNÓSTICO MÉDICO"

TRABAJO DE FIN DE GRADO

Septiembre – 2021

AUTOR: Manuel López Martínez

DIRECTOR: Antonio Peñalver Benavent

“La mejor forma de predecir el futuro es creándolo”

Peter Drucker.

Resumen

Desde que se tiene uso de razón, el ser humano ha sido consciente de estar expuesto a innumerables enfermedades de todo tipo: infecciosas, crónicas, degenerativas, genéticas, etc. A medida que se ha ido evolucionando y adquiriendo el conocimiento necesario, se ha hecho hincapié en desarrollar medicaciones y/o tratamientos que ayuden a eliminar o reducir los trastornos que estas enfermedades provocan en el organismo.

Este trabajo de fin de grado, haciendo uso de la inteligencia artificial, pretende dar una visión preventiva al personal sanitario de centros médicos, sobre la tendencia de los pacientes a la hora de desarrollar determinadas enfermedades. Proporcionando de esta forma, una ayuda para el diagnóstico, pues la verdadera eficacia de un tratamiento reside en la pronta diagnosticación.

El desarrollo de este trabajo incluye: la elaboración de un modelo de red neuronal, basado en Deep Learning, para la predicción de infartos cerebrovasculares; el diseño y desarrollo de una aplicación multiplataforma para la visualización de los resultados y el diseño y desarrollo de una API Rest, proporcionando esos resultados a la aplicación, haciendo de intermediaria entre el modelo y la misma.

Para el entrenamiento de la red neuronal, se han realizado varios experimentos con diferentes arquitecturas y configuración de parámetros, con el fin de evaluar el desempeño del modelo.

Agradecimientos

La elaboración de este trabajo ha sido el empujón que necesitaba para descubrir y adentrarme en el maravilloso mundo de las redes neuronales, así como el haber podido nutrirme de nuevos lenguajes de programación, como son Python y Dart, e iniciarme en el desarrollo de aplicaciones multiplataforma, de la mano de Flutter.

Este trabajo de fin de grado supone el final de una etapa importante para mí. Si durante esta etapa, dijera que no ha habido momentos difíciles, estaría mintiendo, pues no es fácil compaginar los estudios con el trabajo. Pero una vez llegas hasta aquí te das cuenta de que si algo quieres debes luchar con todas tus fuerzas por conseguirlo.

Gracias, en primer lugar, a mis padres por apuntarme con doce años a un curso de ofimática avanzada, pues gracias a vosotros comencé a interesarme por el fascinante mundo de la informática. Gracias, a mi hermana Marina, por ser la persona más fuerte y valiente que conozco y porque contigo he aprendido que el querer es poder.

Gracias a mis abuelos, en especial a mi abuelo Manolo, tú me enseñaste lo importante que es el tener perseverancia en conseguir todo lo que uno se propone y aunque ya no estés conmigo, sigues siendo fuente de inspiración para mí.

Gracias a ti Cris, no solo por aguantarme si no por hacerlo en los momentos de bajón y estar siempre ahí.

Gracias a mi tutor Antonio Peñalver por su confianza puesta en mí, sus buenos consejos y dedicación en este proyecto.

Gracias a mis compañeros de empresa, en especial a Lorca y Juanmi, por hacerme más llevadero la compaginación de estudios y trabajo.

Índice

Índice de ilustraciones

Índice de tablas

Índice de ecuaciones

CAPÍTULO 1: INTRODUCCIÓN

1.1. Salud y enfermedad

Los términos salud y enfermedad que se muestran a continuación, han sido definidos por la Organización Mundial de la Salud (OMS).

1.1.1. ¿Qué es la salud?

La salud es un estado de completo bienestar físico, mental y social, y no solamente la ausencia de afecciones o enfermedades [1].

1.1.2. ¿Qué es la enfermedad?

La enfermedad es una parte más de la salud y de la vida en general. Es la alteración o desviación del estado fisiológico en una o varias partes del cuerpo, por causas en general conocidas, manifestada por síntomas y signos característicos, y cuya evolución es más o menos previsible [1].

La enfermedad es considerada como cualquier estado donde haya un deterioro de la salud del organismo humano. Todas las enfermedades implican un debilitamiento del sistema natural de defensa del organismo [1].

1.1.3. Tipos de enfermedades

Dependiendo de los órganos o tejidos afectados y del motivo de su aparición, las enfermedades se pueden clasificar en quince tipos:

1. **Enfermedades oncológicas**

Más conocidas como cáncer. Son caracterizadas por el desarrollo de tumores, siendo éstos causados por el desarrollo anormal de un grupo de células que se dividen de manera descontrolada, teniendo la capacidad de infiltrarse y destruir los tejidos del organismo [2].

2. **Enfermedades infecciosas y parasitarias**

Son causadas por microorganismos, tales como bacterias, virus, parásitos u hongos [2].

3. **Enfermedades de la sangre**

La sangre está constituida por glóbulos rojos (células que transportan el oxígeno por el cuerpo), glóbulos blancos y plaquetas.

Las enfermedades sanguíneas afectan a estos componentes, así como a las células que se encargan de generarlos e impiden que cumplan sus funciones [2].

4. **Enfermedades del sistema inmunitario**

El sistema inmunitario es el encargado de proteger el cuerpo contra las enfermedades e infecciones. Hay ocasiones, en que este se puede debilitar o alterar, provocando que no pueda cumplir sus funciones de forma correcta, haciendo que la persona desarrolle una inmunodeficiencia o una enfermedad autoinmune [2].

5. **Enfermedades endocrinas**

El sistema endocrino está formado por ocho glándulas distribuidas por todo el cuerpo y produce más de 20 hormonas. Las hormonas actúan como mensajeros clínicos y viajan hacia los tejidos y órganos a través del torrente sanguíneo. Cumplen labores tales como: función sexual, modulación del estado de ánimo, crecimiento y desarrollo, regulación del metabolismo.

Las enfermedades endocrinas aparecen cuando la producción de hormonas se ve alterada, ya sea por un déficit o superávit en el desarrollo de las hormonas [2].

6. **Trastornos mentales, del comportamiento y del desarrollo**

Se caracterizan por causar una alteración en la cognición, la regulación emocional o el comportamiento de las personas, siendo sus causas muy variadas. Pueden deberse a un factor genético, y, por lo tanto, pueden ser hereditarias, como consecuencias del estrés o incluso de la alimentación. En algunas ocasiones, puede deberse a infecciones perinatales o a riesgos ambientales [2].

7. **Enfermedades del sistema nervioso**

El sistema nervioso es el centro de comunicación de nuestro cuerpo y está formado por el cerebro, médula espinal y nervios.

Es uno de los sistemas más complejos y recibe la información de los órganos sensoriales a través de los nervios, la transmite a través de la médula espinal y en último lugar es procesada por el cerebro. Coordina los sentidos, el movimiento y la capacidad de pensar y razonar.

La sintomatología de estas enfermedades neurológicas vendrá condicionada por la zona del sistema nervioso que se vea afectada. Pueden ser afecciones de carácter

degenerativo o aparecer de forma repentina como respuesta a una lesión, por ejemplo, a causa de un accidente [2].

8. Enfermedades oftalmológicas y de la visión

Los ojos son la continuación de nuestro sistema nervioso y son los encargados de captar estímulos sensitivos.

Las enfermedades oculares varían según la parte del ojo afectado, por lo que las causas pueden diferir considerablemente. A veces, pueden estar originadas por problemas vasculares, traumatismos o procesos degenerativos [2].

9. Enfermedades auditivas

Las orejas alojan el sistema auditivo del cuerpo. Estas enfermedades pueden estar provocadas por afectaciones al tímpano (membrana que actúa como tambor), aunque también pueden ser producto de afecciones neurosensoriales [2].

10. Enfermedades cardiovasculares

Los trastornos que afectan al corazón y a los vasos sanguíneos son la principal causa de muerte en todo el mundo. La gravedad de estas enfermedades reside en que el sistema circulatorio, es el encargado de hacer llegar oxígeno y nutrientes a todos los tejidos del organismo. Por ello, si falla esta función, el organismo se ve gravemente comprometido.

Las causas más comunes son: tabaco y alcohol, dietas altas en sodio y grasas y defectos congénitos [2].

11. Enfermedades respiratorias

Los pulmones son los órganos más susceptibles y sensibles del cuerpo, pues están constantemente expuestos a los patógenos y contaminantes del medio externo. Algunas de las causas de este tipo de enfermedades son: bacterias, virus, tabaquismo, contaminación del aire por polvo y gases tóxicos [2].

12. Enfermedades del sistema digestivo

El aparato digestivo está formado por muchos órganos distintos, por lo tanto, son afecciones que pueden ocurrir en el estómago, los intestinos, el esófago, la boca, etc. Los trastornos digestivos causan problemas en el procesamiento y la eliminación de la comida [2].

13. Enfermedades de la piel

La piel es el órgano más grande del organismo. Los trastornos en este órgano suelen ser muy visibles y, en la mayoría de los casos, no ser muy graves.

Como la piel está en constante contacto con el medio exterior, la lista de factores que la pueden dañar es muy amplia. Entre ellos se pueden destacar: tomar el sol sin protección, el tabaco, la utilización de fórmulas cosméticas no respetuosas, sequedad ambiental y una mala calidad del agua. Existen también irritaciones de la piel que pueden ser debidas a condiciones genéticas o de condición autoinmune [2].

14. Enfermedades del aparato genitourinario

Son aquellas que afectan al sistema urinario (riñón, uréter, vejiga, uretra) y al aparato reproductor (pene, útero, matriz) [2].

15. Enfermedades congénitas y alteraciones cromosómicas

Forman un amplio grupo de enfermedades causadas por alteraciones genómicas que pueden ser heredables. En él, forman parte las patologías causadas por deformidades y anomalías cromosómicas, las cuales, se generan por mutaciones en ciertos genes o en anormalidades en el conjunto de cromosomas que presenta el individuo [2].

1.1.4. Infarto cerebrovascular

En el apartado anterior, se enumeraron las distintas categorías en las que se pueden clasificar las enfermedades. En este apartado, se habla del infarto cerebrovascular, enfermedad en la que se centra este proyecto, y sobre el que se tratará en el apartado 5.2.1. Red neuronal.

¿Qué es un infarto cerebrovascular?

El infarto cerebrovascular, o también llamado ataque cerebral, sucede cuando se interrumpe el flujo sanguíneo de parte del cerebro, que al no poder recibir el oxígeno y nutrientes que necesita, las células comienzan a morir, pudiendo provocar un daño severo al cerebro, discapacidad permanente e incluso la muerte [3].

Factores que aumentan el riesgo de sufrirlo

Algunos de los factores que más aumentan el riesgo de sufrir un infarto cerebrovascular son [3]:

- **Diabetes**.
- **Presión arterial alta** (uno de los principales factores de riesgo).
- **Enfermedades del corazón**.
- **Fumar**.
- **Antecedentes familiares**.
- **Edad**: su riesgo aumenta a medida que se envejece.
- **Raza y etnicidad**: por ejemplo, los afroamericanos tienen un mayor riesgo de padecer ataques cerebrales.

1.2. Introducción a las redes neuronales

En los siguientes apartados se pretende dar una visión global sobre las redes neuronales artificiales y su símil con las redes neuronales biológicas. Además, se definirán una serie de conceptos básicos de gran ayuda para la comprensión de este trabajo de fin de grado.

1.2.1. Redes neuronales biológicas

1.2.1.1. El cerebro

El cerebro es el órgano más complejo del cuerpo y está implicado en todas las tareas y funciones que se llevan a cabo día a día. Desde hablar, pensar o razonar, hasta tareas tan básicas como el respirar.

1.2.1.1.1. ¿Cuál es el funcionamiento del cerebro?

A través de los sentidos, el cerebro recibe una gran cantidad de información del entorno exterior. Esta información, el cerebro la procesa y hace que cobre significado.

Internamente, cuando se toman decisiones y/o se experimentan emociones, en el cerebro se produce una compleja mezcla de procesos químicos y eléctricos [4].

1.2.1.1.2. Estructura del cerebro

El cerebro está formado por diversas partes o estructuras que realizan funciones individuales, pero trabajando de forma conjunta y coordinada. Esta coordinación es posible gracias a los miles de conexiones que se establecen entre sí y con el resto del cuerpo [4].

El cerebro, forma parte del Sistema Nervioso Central (SNC), de donde se pueden distinguir dos partes: la médula espinal y el encéfalo.

- La médula espinal es un largo cordón blanquecino localizado en el canal vertebral, cuya tarea es la de actuar como si de una carretera se tratara, llevando la información proporcionada por el cerebro al resto del cuerpo.
- El encéfalo es la parte central del SNC y se encuentra encerrada y protegida dentro del cráneo. El cerebro sería tan sólo una parte de todo el encéfalo.

La corteza, o córtex cerebral, es la superficie externa del cerebro y tiene una gran extensión, aproximadamente el equivalente a una o dos hojas de periódico. Sólo un tercio de la corteza está expuesta superficialmente, el resto está oculto en la profundidad de los surcos. Esto hace que se aproveche mucho mejor el espacio que si el córtex fuese liso, ya que permite que diferentes regiones del cerebro se comuniquen más rápida y fácilmente, por el hecho de estar más cerca [4].

El cerebro está dividido en dos grandes partes, el hemisferio derecho y el hemisferio izquierdo, ambos conectados entre sí por un conjunto de fibras, que constituyen el cuerpo calloso. Así mismo, cada uno de los hemisferios cuenta con cuatro lóbulos: frontal, parietal, temporal y occipital. Cada uno de estos lóbulos contribuye de manera diferente a las distintas funciones que desempeña el cerebro [4].

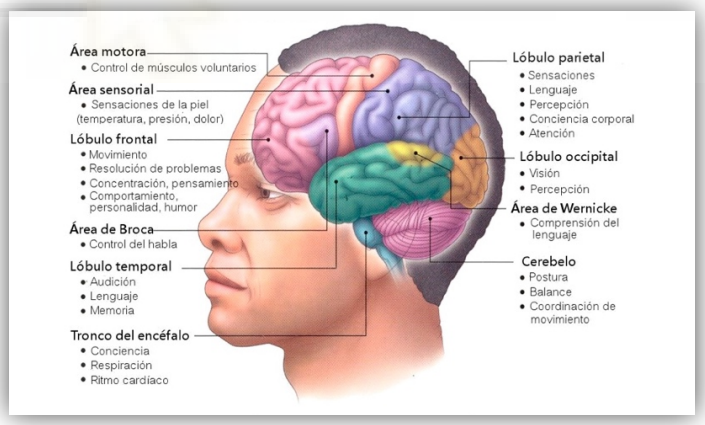


Ilustración 1: Partes del cerebro humano y sus funciones

1.2.1.2. La neurona biológica

Las neuronas (también llamadas células nerviosas) son las unidades fundamentales del cerebro. Son las responsables de la recepción de los inputs sensoriales provenientes del mundo exterior, así como de enviar órdenes a distintas partes del cuerpo.

Constan de tres partes elementales:

- **El cuerpo celular** o **soma**: en él se encuentra el núcleo que contiene el ADN y es el lugar donde se elaboran las proteínas.
- **El axón**: parecido a un cable, es el elemento de la célula que conduce los mensajes electroquímicos.
- **Las dendritas**: son las encargadas de establecer conexiones con otras células. Las dendritas reciben los mensajes a través de los neurotransmisores que liberan los axones de otras células nerviosas. En la parte inicial del axón se genera un breve impulso eléctrico que viaja a lo largo de este y provoca la liberación de neurotransmisores en la sinapsis, el punto donde se produce esta liberación y la recepción del mensaje por otra neurona, permitiendo así la comunicación entre ellas [4].

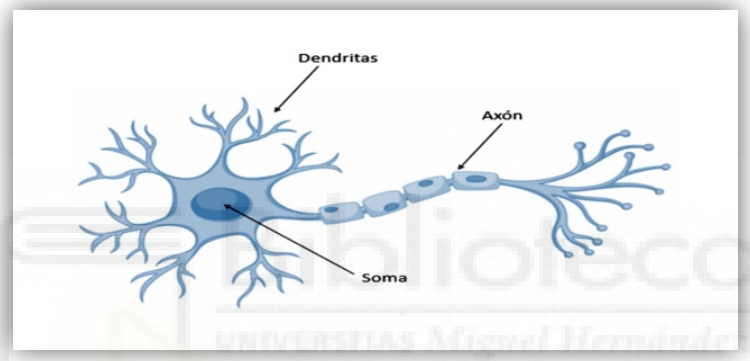


Ilustración 2: Estructura de una neurona biológica

Para que una neurona permanezca sana y activa es imprescindible que haya una conexión con otro tipo de células, pues sin dicha conexión morirían [4].

La neurona por sí sola tiene un comportamiento muy simple, pero al conectarse con otras neuronas, se generan enlaces que permiten realizar cálculos y/o tareas muy complejas.

1.2.2. Redes neuronales artificiales

La capacidad que tiene el cerebro humano de recordar, pensar y resolver problemas ha inspirado a muchos científicos a intentar modelar en el ordenador el funcionamiento de este. Siendo este el principal motivo del nacimiento de las redes neuronales artificiales.

Aunque el concepto de redes neuronales puede parecer reciente debido al “boom” que están experimentando, no lo es, pues en los años 50 ya se asentaron sus bases.

Una de las razones por la que no tuvo tanta repercusión como hoy en día, fue debido a la escasa potencia de cómputo que había, haciendo que las capacidades de las redes neuronales estuvieran muy limitadas.

En la actualidad, gracias a las grandes capacidades y tecnologías de cómputo que tenemos hacen que las herramientas y funcionalidades que se le pueden dar a las redes neuronales sean muy diversas. Es por ello por lo que, su popularidad y funcionalidad están creciendo a pasos agigantados.

1.2.2.1. ¿Qué son los modelos?

El mundo en el que se vive es complejo, caótico y lleno de ruido. A pesar de ello, a lo largo de la historia, la inteligencia humana ha conseguido dar sentido a todo ese caos, en una búsqueda por la elegancia de la simetría que se esconde en los patrones que se identifican en la realidad. Esto ha provocado que se esté en constante evolución, debido a esta capacidad que tiene el ser humano en saber detectar patrones y poder utilizarlos a su favor [5].

La ciencia ha permitido explotar la capacidad de observar el mundo de manera más simplificada, convirtiendo todo ese ruido en conocimiento, es decir, construyendo la realidad a través de modelos [5].

Un modelo es una construcción conceptual simplificada de una realidad más compleja, donde se busca el equilibrio entre aproximarse a representar correctamente la realidad y ser simple para poder utilizarlo.

Un claro ejemplo de un modelo puede ser una partitura, que no es más que la simple representación de cómo los diferentes instrumentos deben combinarse, sincronizarse y ajustar su frecuencia para producir siempre la misma canción.

En el Machine Learning todo se estudia en base a modelos. A partir de los datos se puede construir el modelo que más se aproxime a la realidad que se quiere estudiar. Una vez se tiene el modelo se puede ajustar para probar nuevos escenarios y así evaluar su desempeño.

Existen tres elementos clave en un modelo:

1. **Datos**: gracias a las mediciones que se realizan, son la toma de contacto con la realidad. A partir de estos datos se extrae la información para construir el modelo.

Un concepto importante para tener en cuenta es que los datos son multidimensionales [5].

2. **Parámetros:** dan la flexibilidad al modelo para que este pueda ajustarse a los datos [5].
3. **Error**: tal y como citaba Peter Drucker, "*Aquello que no se mide no se puede mejorar*", por tanto, es necesario definir una función de error (pérdida) que pueda proporcionar una visión de cómo el modelo se ajusta a los datos [5].

1.2.2.2. El perceptron: origen de las redes neuronales

Los Ordenadores Digitales, con las contribuciones de John Von Neuman; la Inteligencia Artificial (IA), con los Sistemas Expertos de Marvin Minsky y; el funcionamiento del ojo, con la conocida red Perceptron de Frank Rosenblatt, así como otros múltiples campos, tienen su fundamento, es decir, la base de su avance en el artículo *A logical calculus of Ideas Imminent in nervous Activity* publicado en 1943 por Warren McCulloch, neurobiólogo, y Walter Pitss, estadístico. En 1956 Minsky, McCarthy, Rochester y Shanon, los precursores de la IA organizaron la primera ponencia de redes neuronales. En dicha conferencia Nayhaural Rochester, perteneciente al grupo de investigación de IBM, anunció el que puede tenerse en cuenta como el *software* inicial de simulación de redes neuronales artificiales [6].

Fue en 1957 cuando salía publicado el que, hasta ese momento, era considerado el más completo y desarrollo trabajo de investigación en computación neuronal. Su autor, Frank Rosenblatt; el elemento desarrollado, el Perceptron, un sistema para catalogar patrones distinguiendo los que son geométricos de los abstractos, por lo que halla el hiperplano que distingue linealmente dos conjuntos. El Perceptron original podía asimilar, es decir, le era posible aprender, por ello su pauta cambiaba únicamente si surgían fallos en los elementos que conformaban el sistema. Una de sus características era la flexibilidad, con lo que su conducta seguía siendo adecuada, aunque varias celdas quedaran destruidas [6].

El proyecto original del Perceptron fue ideado para la identificación visual de patrones. Una rejilla de cuatrocientas fotocélulas, correspondientes a las neuronas de la retina sensibles a la luz, percibe el estímulo óptico. Dichas fotocélulas se encuentran enlazadas a componentes asociativos, que reúnen los impulsos eléctricos producidos desde las fotocélulas. Los vínculos que resultan entre los elementos asociativos y las fotocélulas son ejecutados de manera aleatoria. Cuando las células cuentan con un

valor de entrada más grande que el umbral determinado, el componente asociativo provoca una salida [6].

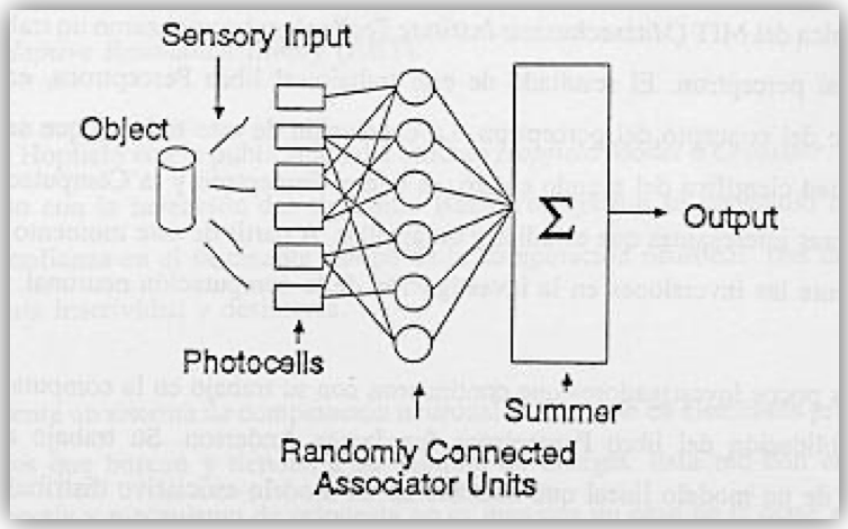


Ilustración 3: Esquema de la red perceptron

Un perceptron toma varias entradas binarias y produce una sola salida binaria. Con la necesidad de poder computar esta salida, Rosenblatt introduce el concepto de “pesos”, es decir, un número real que manifiesta la relevancia de la correspondiente entrada con la salida. Dicha salida de la neurona será 1 o 0 si, la suma de la multiplicación de pesos por entradas es mayor o menor a un determinado umbral. Al tratarse de un instrumento en proceso de desarrollo, el perceptron contaba con determinadas restricciones. La limitación más importante fue revelada por Minsky y Papert años más tardes, dejando clara la imposibilidad del perceptron en ponerle solución a ciertas tareas o cuestiones simples; como por ejemplo la función lógica OR exclusivo. El perceptron de Rossenblatt sufrió cambios durante la década de los 60. Una de las modificaciones más relevantes es la creación perfeccionada de sistemas multicapa, que son capaces de aprender y categorizar datos complejos [6].

En 1959, Bernard Windrow, en Stanford, formula un componente adaptativo lineal con el nombre de “Adaline”. Adaline y Madaline, siendo esta última una versión de dos capas, fueron empleadas en diferentes usos como por ejemplo la identificación de voz y caracteres, pronóstico del tiempo, el control adaptativo y, sobre todo, en el desarrollo de filtros adaptativos que eliminen los ecos de las líneas telefónicas [6].

A mediados de la década de los años sesenta, Minsky y Papert del Laboratorio de Investigación de Electrónica del MIT (Massachusetts Institute of Technology) iniciaron un trabajo hondo y transcendente de análisis crítico al perceptron. *Perceptrons*, fue el

título del libro publicado como consecuencia de dicho trabajo. En él dejaron constancia de un completo estudio matemático de la noción del perceptron. El resultado de ese trabajo, que fue presentado a la comunidad científica mundial, exponía que el Perceptron y la Computación Neuronal no presentaban la relevancia necesaria para ser aprendidas y desarrolladas. Como consecuencia de ello, las investigaciones relacionadas con la computación neuronal decayeron enérgicamente [6].

James Anderson fue uno de los escasos investigadores que, tras la publicación degradándola, decidió proseguir con su trabajo en la computación neuronal. Su trabajo estaba relacionado con el desarrollo de un modelo lineal que se fundamenta en un modelo asociativo distribuido. Dicho arquetipo se asentó sobre el principio de Hebb, por el que las conexiones son fortalecidas cada vez que son accionadas las neuronas. Una versión dilatada de este modelo lineal es el llamado modelo Brain-State-in- a Box (BSB) [6].

En la década de los setenta, la computación neuronal continuó siendo impulsada, siendo uno de los más importantes responsable Teuvo Kahonen, de la Universidad de Helsinki. De su trabajo de investigación sobresalen dos contribuciones. En primer lugar, la explicación y estudio de una considerable clase de reglas adaptativas. En dichas reglas los enlaces ponderados se transforman de manera subordinada de los valores anteriores y posteriores de la sinapsis. En segundo lugar, el principio de aprendizaje competitivo por el que los componentes luchan por responder a un estímulo de entrada, y el vencedor se aclimata a él mismo para contestar con más efecto al impulso [6].

Otro investigador que prosiguió con su trabajo de investigación en el ámbito de la computación neuronal, a pesar del mal presagio que indicaron Minsky y Papert, fue Stephen Grossberg, quién estaba principalmente atraído por el empleo de datos de la neurología para erigir modelos de computación neuronal. Po ese motivo, en el trabajo de Grossberg, la gran parte de las leyes y principios provinieron de estudios fisiológicos. Su trabajo ha fundamentado un vasto estímulo en la investigación del diseño y construcción de modelos neuronales. Una de estas clases de redes es la Adaptative Resonance Theory (ART) [6].

Después de dos décadas, en las que prácticamente no hubo actividad ni interés, fue en 1982, con la publicación del artículo Hopfield Model o Crossbar Associative Network, de John Hopfield, y la creación del algoritmo Backpropagation, cuando se logró restituir el interés y la seguridad en el deslumbrante terreno de la computación neuronal [6].

1.2.2.3. La neurona artificial

Similar a una neurona biológica, es la unidad básica de procesamiento dentro de una red neuronal. Tal y como se explicó en el apartado 1.2.1.2. La neurona biológica, la estructura básica la forman: el núcleo, las dendritas (por donde reciben los estímulos) y axones (por donde se transmite la respuesta). La neurona artificial se diseñó siguiendo este mismo patrón, donde posee: conexiones de entrada (simulan a las dendritas) que reciben la información, un elemento procesador (núcleo) y conexiones de salida (imitando los axones de la neurona biológica).

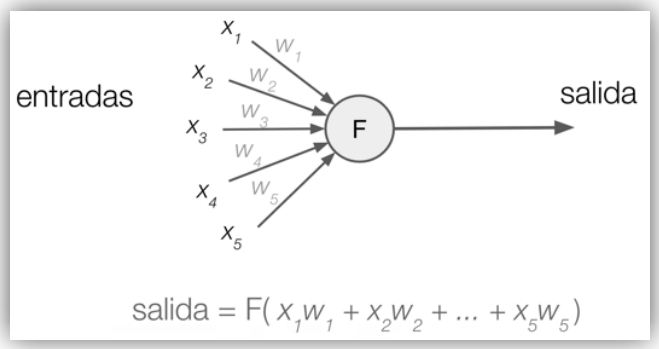


Ilustración 4: Estructura de la neurona artificial

Con los valores de entrada, internamente la neurona realiza una suma ponderada. La ponderación de cada una de las entradas viene dada por el peso (Wx) que se asigna a cada una de las conexiones de entrada. Es decir, cada conexión que llega a la neurona tendrá asociado un valor que servirá para definir con qué intensidad cada variable de entrada afecta a la neurona.

Estos pesos son los parámetros del modelo, y serán los valores que se podrán ajustar para que la red pueda aprender. Este procedimiento de ajustar los valores para un correcto desempeño del modelo se conoce como optimización, aunque normalmente se le llama entrenamiento o ajuste del modelo.

En realidad, una neurona artificial no es más que una abstracción de una función matemática de regresión lineal.

$$y = W_0 + W_1X_1 + W_2X_2 + \cdots + W_nX_n$$

Ecuación 1: Suma ponderada de una neurona artificial (regresión lineal)

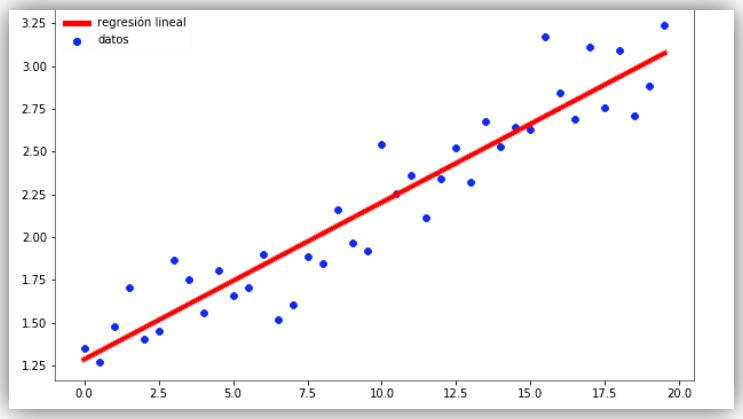


Ilustración 5: Ejemplo de regresión lineal

1.2.2.4. La red neuronal

Existen diferentes maneras de organizar las neuronas. Una manera sería colocarlas en la misma capa. Dos neuronas que se encuentran en la misma capa recibirán la misma información de entrada de la capa anterior y los cálculos que realicen lo pasarán a la capa siguiente. A la primera capa donde están las variables de entrada se le denomina **capa de entrada** y a la última **capa de salida**. A las capas intermedias se les denomina **capas ocultas**.

Cuando se colocan dos neuronas de forma secuencial, una de ellas recibe la información procesada por la neurona anterior. Esto aporta la ventaja de que la red pueda aprender conocimiento jerarquizado.

Cuantas más capas, más complejo puede ser el conocimiento que se elabora. Aunque, tal y como se mostrará en los diferentes experimentos realizados en el apartado 5.2.1.4.3. Entrenamiento de la red neuronal, no siempre por añadir más capas implica que la red se comporte mejor, siendo en algunos casos contraproducente.

La profundidad en la cantidad de capas es lo que se conoce como ***Deep Learning***.

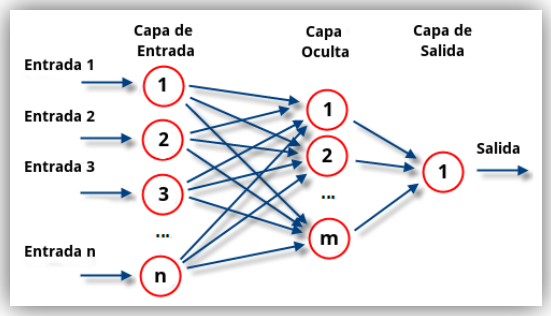


Ilustración 6: Ejemplo de la estructura de una red neuronal de n capas

1.2.2.5. Función de activación

Tal y como se mostró en el apartado 1.2.2.3 La neurona artificial, lo que hace una neurona es un problema de regresión lineal $y = wx + b$. Por ende, al conectar varias neuronas y formar una red neuronal, en verdad lo que se está haciendo es concatenar diferentes operaciones de regresión lineal. Esto presenta un problema y es que, matemáticamente, el hecho de sumar muchas operaciones de regresión lineal (líneas rectas) es equivalente al haber hecho una única operación.

Para conseguir que esa suma no colapse, es necesario que la suma dé como resultado algo diferente a una línea recta. Esto se consigue convirtiendo esa salida lineal en una no lineal, a través de pequeñas manipulaciones. Estas manipulaciones, las llevarán a cabo las funciones de activación [5].

Una función de activación es una función matemática que se aplica al valor de salida de la neurona artificial con el objetivo de distorsionarlo, añadiéndole deformaciones no lineales. De esta manera, se pueden conectar de forma efectiva la computación de varias neuronas, haciendo que las operaciones conjuntas de todas ellas tengan sentido y finalidad [5].

Cada una de estas funciones, además de aportar la no linealidad, ofrecen una serie de características y, darán mejores o peores resultados en función del objetivo que se quiera conseguir con la red neuronal.

Las principales funciones de activación son las siguientes:

- **Escalonada**: se llama escalonada porque el cambio de valor se produce instantáneamente y no de forma gradual. Para un valor de entrada mayor a un umbral, provoca un uno a la salida y si es inferior al umbral, la salida es cero. Esta función se suele usar cuando se quiere clasificar o cuando se tiene salidas categóricas [7] [8].

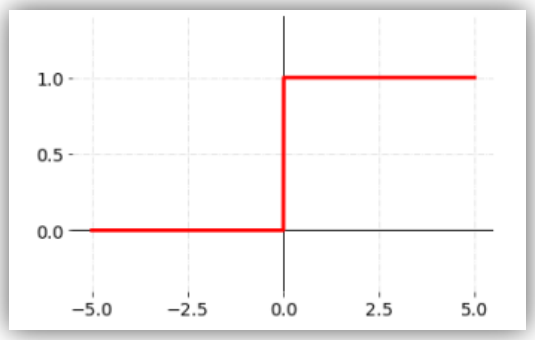


Ilustración 7: Gráfica de la función de activación escalonada

- **Sigmoide**: la distorsión que produce hace que los valores muy grandes se saturen a 1 y los valores muy pequeños se saturen a 0. Por tanto, con esta función no solo se consigue añadir la deformación que se necesita, si no que también sirve para representar probabilidades que siempre vienen en el rango 0 - 1. Esta función, tiene un uso limitado, siendo su principal aplicación la clasificación binaria [7] [8].

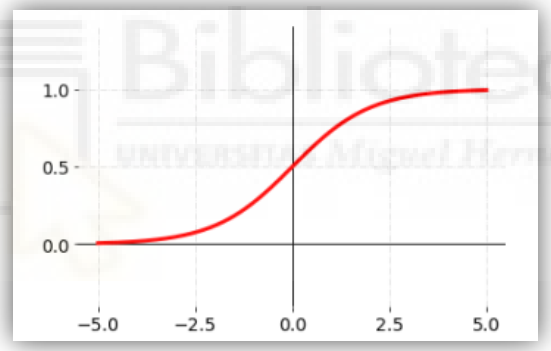


Ilustración 8: Gráfica de la función de activación sigmoide

Tal y como se observa en la ilustración anterior, la función se satura a 1 cuando la entrada es muy alta, y a 0 cuando es muy baja. Esto hace que, durante el entrenamiento, usando el método del gradiente descendente, los gradientes calculados sean muy pequeños, dificultando así la convergencia del algoritmo.

- **Tangente hiperbólica (TANH)**: es similar a la sigmoide, pero su rango varía de -1 a 1.

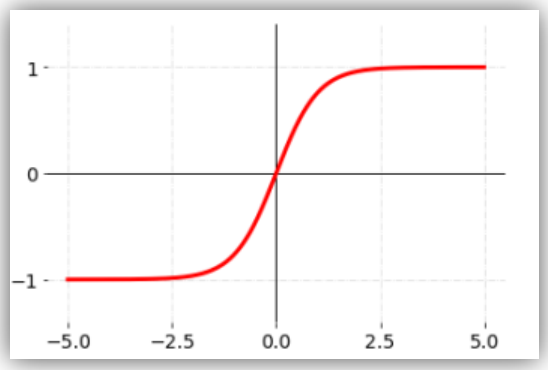


Ilustración 9: Gráfica de la función de activación TANH

En este caso, la función también sufre de saturación, pero ofrece la ventaja de tener una salida simétrica, facilitando así el proceso de entrenamiento.

- **RELU (Unidad rectificada lineal)**: básicamente se comporta como una función lineal cuando es positiva y constante a 0 cuando el valor de entrada es negativo.

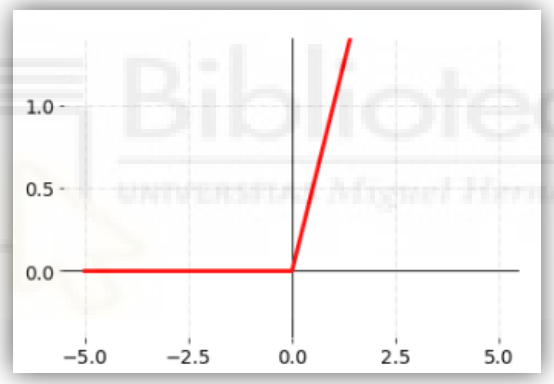


Ilustración 10: Gráfica de la función de activación RELU

Es la más utilizada debido principalmente a las dos siguientes ventajas:

  - No existe saturación, haciendo que el algoritmo del gradiente descendente converja mucho mejor, facilitando así el entrenamiento.

  - Computacionalmente, es más sencillo y eficiente de implementar en comparación con SIGMOIDE y TANH

- **Lineal**: también conocida como función de identidad. Se suele utilizar en la última capa de la red neuronal, siendo su principal característica no aplicar ninguna transformación a los datos de entrada. Es útil para problemas de regresión [7] [8].

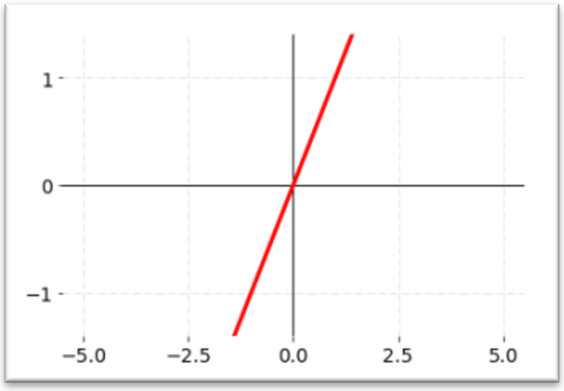


Ilustración 11: Gráfica de la función de activación lineal

1.2.2.6. Función de pérdida

El primer hito que se pretende conseguir al implementar una red neuronal es minimizar la diferencia entre el resultado estimado y el real obtenido. Esta diferencia es lo que se conoce como función de pérdida (o coste).

Propiedades de la función de pérdida

- Para cada algoritmo, la función de pérdida no es única.
- La función de pérdida se puede utilizar para evaluar la calidad del modelo. Cuanto menor es la función de pérdida, mejor se ajusta el modelo y los parámetros a los datos.

Existen diferentes funciones de pérdida y la selección de una de ellas depende de varios factores como, por ejemplo, el algoritmo seleccionado, pero principalmente depende del objetivo del modelo implementado.

A continuación, se presentan algunas de las funciones de pérdida más utilizadas en la regresión y clasificación:

Funciones de pérdida para regresión

- **Error Cuadrático medio (*Mean Square Error*):** es la función de pérdida más básica y utilizada en regresión. Representa a la raíz cuadrada de la distancia cuadrada promedio entre el valor esperado y el valor obtenido. El hecho de elevar al cuadrado hace penalizar con mayor intensidad aquellos puntos que están más alejados del valor esperado y, con menor intensidad, a los que se encuentran más próximos. Siendo los valores más bajos los que indican un mejor ajuste [9].

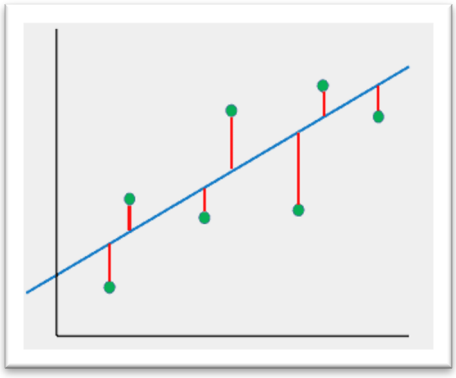


Ilustración 12: Ejemplo del error cuadrático medio

$$RMSE = \sqrt{\frac{1}{n}\sum_{j=1}^{n}(y_j - \hat{y}_j)^2}$$

Ecuación 2: Error cuadrático medio

- **Error absoluto medio (MAE):** es el promedio de la suma de las diferencias absolutas entre el valor objetivo y el valor predicho. Esta función de pérdida presenta un problema a la hora de entrenar a la red neuronal, y es que, su gradiente es una constante relativamente grande, pudiendo hacer que el valor mínimo desaparezca al final del entrenamiento [10].

$$MAE = \frac{\sum_{i=1}^{n}\left|y_i - y_i^p\right|}{n}$$

Ecuación 3: Error absoluto medio

- **Error absoluto medio uniforme (Huber)**: la pérdida de Huber es menos sensible a valores atípicos en los datos en comparación con el MAE. Cuando el error es pequeño, se convierte en error cuadrático. Lo pequeño que el error pueda convertirse en una función dependerá de los hiperparámetros. Cuando el hiperparámetro es próximo a cero, el método de pérdida de Huber está cerca de MAE; cuando el hiperparámetro es próximo a ∞, el método está cerca de MSE. Esta función de pérdida es útil porque reduce el gradiente cerca del mínimo y es más robusto que el MSE [10].

- **Pérdida de log-Cosh:** es el logaritmo del coseno hiperbólico del error de predicción. Funciona de forma muy similar a MSE, pero, no se ve afectado por valores atípicos ocasionales [10].

$$L(y, y^p) = \sum_{i=1}^{n} \log\left(\cosh\left(y_i^p - y_i\right)\right.$$

Ecuación 4: Log-Cosh

- **Pérdida de cuantiles**: es útil cuando se quiere predecir un intervalo en lugar de solo un punto. El intervalo de predicción de la regresión de mínimos cuadrados se basa en el supuesto de que el residual tiene una varianza constante entre los valores de las respectivas variables [10].

$$L_\gamma(y, y^p) = \sum_{i=y_i<y_i^p} (\gamma - 1) \cdot \left|y_i - y_i^p\right| + \sum_{i=y_i<y_i^p} (\gamma) \cdot \left|y_i - y_i^p\right|$$

Ecuación 5: Pérdida de cuantiles

Funciones de pérdida para clasificación

- **Multi clase SVM Loss (Hinge Loss)**: esta función de pérdida, normalmente, se usa para la clasificación de margen máximo, más concretamente, para máquinas de vectores de soporte. Es una función convexa que facilita la tarea a los optimizadores convexos [11].

$$SVM\ Loss = \sum_{j \neq y_i} \max(0, s_j - s_{yi} + 1)$$

Ecuación 6: SVM Loss

- **Entropía cruzada (*Cross Entropy Loss*)**: se trata de la función más utilizada para problemas de clasificación. La pérdida de entropía cruzada aumenta a medida que la probabilidad prevista diverge del valor real. Un aspecto importante en esta función de pérdida es que penaliza fuertemente las predicciones que son seguras pero erróneas [11].

$$CrossEntropyLoss = -(y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)$$

Ecuación 7: Cross Entropy Loss

1.2.2.7. Descenso del gradiente

Es un algoritmo (optimizador) clave dentro del *machine learning* y se encuentra en el corazón de la gran mayoría de sistemas de inteligencia artificial que se desarrollan hoy en día.

Antes de pasar a definir en detalle el funcionamiento del algoritmo, es conveniente dar un repaso a algunos conceptos básicos sobre el tipo de funciones matemáticas.

- **Función convexa**: de encontrar un punto mínimo, se puede asegurar que dicho punto será un ínfimo global de la función. Es decir, no se encontrará otro punto de la función que sea inferior a ese.

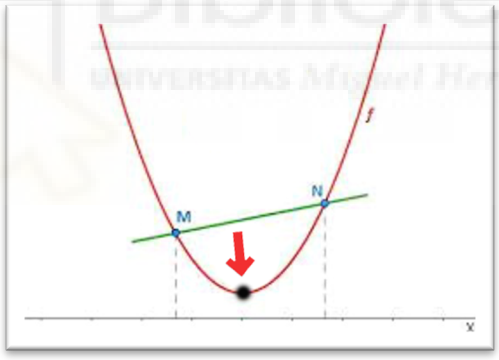


Ilustración 13: Gráfica de una función convexa

- **Función cóncava**: a efectos prácticos, una función cóncava, si se invierte mediante un símbolo negativo, se convierte en una función convexa.

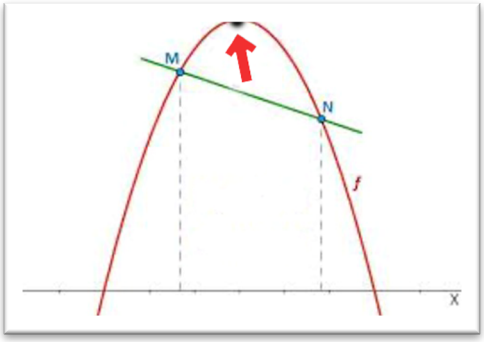


Ilustración 14: Gráfica de una función cóncava

- **Funciones no convexas**: este tipo de funciones no cumplen la propiedad de las funciones convexas, ya que en estas es posible encontrar un punto mínimo que no sea el ínfimo global de la función.

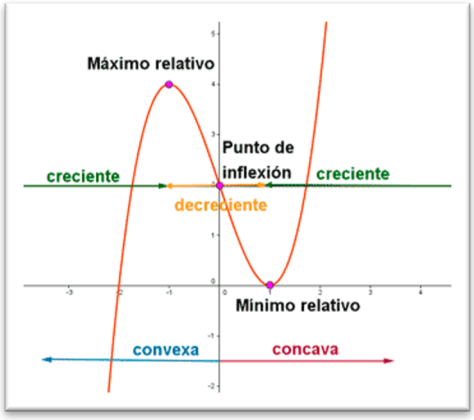


Ilustración 15: Gráfica de una función no convexa

En este tipo de funciones no convexas donde se tienen varios puntos mínimos es donde entra en juego el algoritmo del descenso del gradiente.

Si se echa la vista atrás, en las clases de matemáticas del instituto, se enseña que para encontrar los mínimos de una función se deben aplicar los siguientes pasos:

1. **Derivar la función**: si se deriva una función, la derivada indica la pendiente de la función en ese punto.

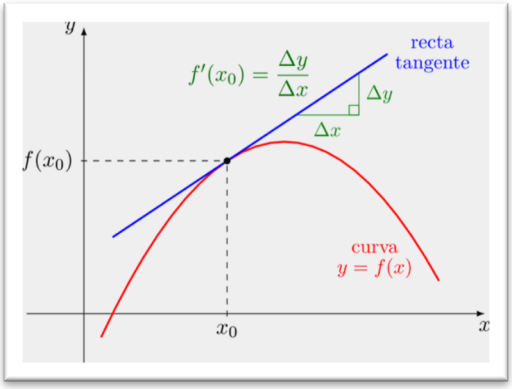


Ilustración 16: Gráfica de la derivada de una función

2. **Igualar a cero**: una vez se tiene la derivada, se iguala la pendiente a cero y se resuelve dicha ecuación, de esta forma se obtiene el punto donde la pendiente es nula. En las funciones convexas, dado que sólo hay un punto mínimo, únicamente se deberá resolver una ecuación. Pero, en las funciones no

convexas sí pueden presentarse múltiples puntos mínimos, y, por lo tanto, habría múltiples ecuaciones a resolver. Y no sólo eso, sino que también puede haber zonas de la función con pendiente nula, como, por ejemplo: máximos locales, puntos de inflexión o puntos de silla. Esto hace que se tenga un sistema de ecuaciones enorme, bastante ineficiente de resolver.

Para explicar el funcionamiento del descenso del gradiente se hará uso de la siguiente ilustración, donde representa la función de coste de un modelo cualquiera.

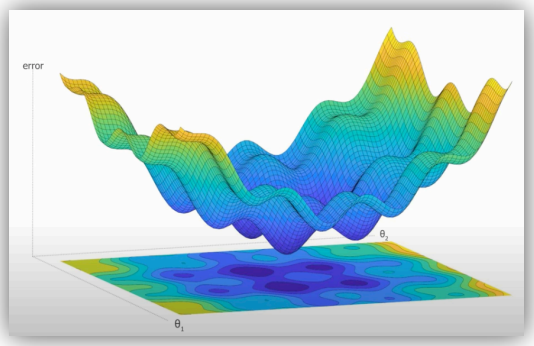


Ilustración 17: Ejemplo gráfico de una función de coste cualquiera

Cuando se empieza el entrenamiento de la red neuronal, los parámetros se inicializan con un valor aleatorio, esto equivale a iniciar en la función en un punto cualquiera, por ejemplo:

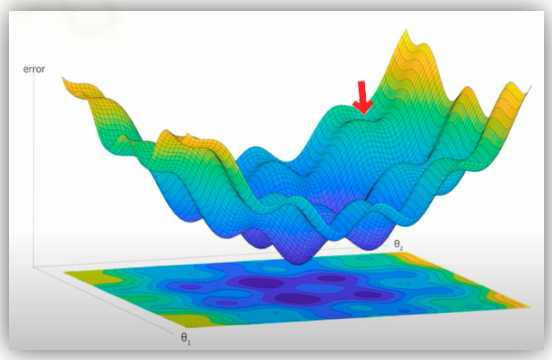


Ilustración 18: Ubicación aleatoria en una función

Ahora, aprovechando la funcionalidad de las derivadas, se evalúa la pendiente de la posición en la que se encuentre. Como la función en este caso es multidimensional, se deben calcular las derivadas parciales para cada uno de los parámetros. Cada uno de estos valores, informará de cuál es la pendiente en el eje de dicho parámetro [5].

Todas estas derivadas parciales forman un vector que indica la dirección hacia la que la pendiente asciende. A este vector se le denomina **gradiente** [5].

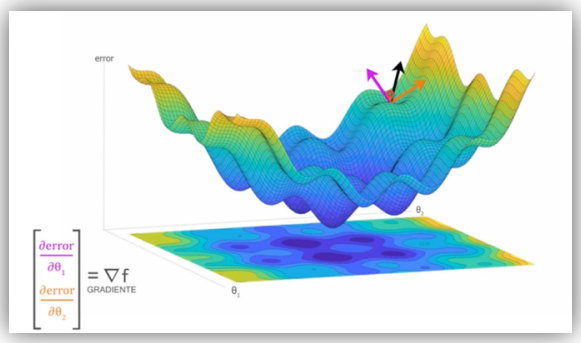


Ilustración 19: Representación gráfica del vector gradiente

Pero, lo que se pretende realizar con el algoritmo del descenso del gradiente es descender, por lo que se debe tomar este vector en sentido contrario, es decir, si el gradiente indica como actualizar los parámetros del modelo para subir, lo que se debe hacer es restarlo [5].

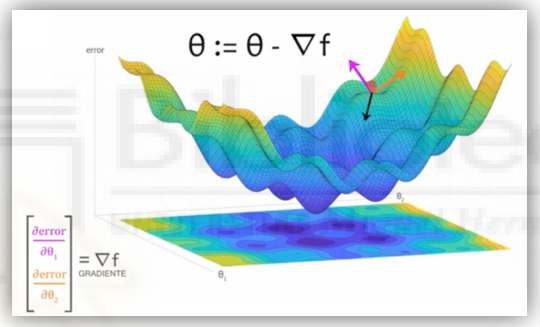


Ilustración 20: Representación gráfica del vector gradiente inverso

Esto daría lugar a un nuevo conjunto de parámetros y, por tanto, se ubicaría en un nuevo lugar de la función, donde se deberá repetir este proceso múltiples veces de forma iterativa hasta llegar a una zona donde el moverse ya no suponga una variación destacada del coste [5].

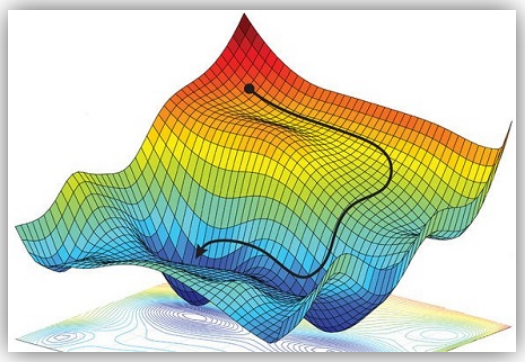


Ilustración 21: Ejemplo del recorrido realizado en el descenso del gradiente

Junto al algoritmo del gradiente existe otro parámetro muy importante en el desempeño de este, conocido como **ratio de aprendizaje**, que viene a definir completamente el comportamiento del algoritmo.

El ratio de aprendizaje define cuanto afecta el gradiente a la actualización de los parámetros en cada iteración. Es decir, cuanto se avanza en cada paso.

Este parámetro es muy importante y se deberá encontrar un valor adecuado para cada problema, ya que, si se le asigna un valor muy alto, los pasos que realiza el algoritmo en cada iteración serán muy largos y será incapaz de introducirse dentro de la zona de mínimo coste, siendo imposible converger en dicho punto, quedando el proceso de optimización en un bucle infinito. Si, por el contrario, se le asigna un valor muy pequeño, sí se aproximará al mínimo coste, pero a consecuencia de realizar muchas iteraciones, por lo que el proceso de entrenamiento será más lento e ineficiente.

1.2.2.8. Optimizadores

Un optimizador es un algoritmo cuya finalidad es ajustar los parámetros (pesos) de la red neuronal, con el objetivo de minimizar la función de pérdida, es decir, encontrar el mínimo error posible. El optimizador es utilizado para entrenar a la red y utiliza la función de pérdida como guía para saber si está en el camino adecuado para conseguir el mínimo global de la función.

Además del optimizador del descenso del gradiente, existen otros optimizadores que, basados en él, intentan mejorar su rendimiento. Y, al igual que ocurría con la función de pérdida, dependiendo del problema que se quiera resolver, será más conveniente utilizar uno u otro, pues una mala elección puede tener un impacto en el resultado, principalmente en los tiempos del entrenamiento de la red.

Tipos de optimizadores

- **Stochastic Gradient Descent (SGD)**: este optimizador simplifica el cálculo, considerando sólo una muestra escogida de forma aleatoria cada vez que realiza el cálculo del gradiente, es decir, se pasa una única muestra, se calcula la función de error asociada y, a partir de esta, el gradiente y las modificaciones a aplicar. Este funcionamiento supone que el tiempo necesario para realizar el cálculo sea muchísimo menor, aunque el algoritmo irá moviéndose hacia el mínimo de forma menos coherente en cada iteración [12] [13].

- **Mini-batch Gradient Descent**: calcula el gradiente, no a partir de todo el conjunto de entrenamiento ni a partir de una muestra, sino a partir de un pequeño subconjunto aleatorio de muestras llamadas mini-batches. Este algoritmo tarda menos en alcanzar el mínimo que SGD, pero corre el riesgo de caer en un mínimo local más fácilmente [12] [13].

- **Momentum**: este optimizador recuerda el incremento aplicado a las variables en cada iteración y determina la siguiente actualización como una combinación lineal entre el gradiente y el incremento anterior. Es decir, aplica a los incrementos cierta "inercia" de forma que varíen más lentamente, pero con algo más de criterio [12] [13].

- **Nesterov Accelerated Gradient (NAG)**: añade también la inercia del optimizador Momentum, pero intenta que los incrementos disminuyan cuando el algoritmo se acerca al mínimo buscado [12] [13].

- **AdaGrad (Adaptative Gradient Algorithm)**: basándose en Stochastic Gradient Descent, este utiliza diferentes tasas de aprendizaje para las variables, teniendo en cuenta el gradiente acumulado en cada una de ellas. Este optimizador plantea un problema y es que, en ocasiones, puede ocurrir que la tasa de aprendizaje para una variable decrezca demasiado rápido debido a la acumulación de altos valores del gradiente al comienzo del entrenamiento, provocando que no sea capaz de aproximarse al mínimo en dicha dimensión [12] [13].

- **RMSProp**: es una variación del AdaGrad, pero, en lugar de mantener el acumulado de los gradientes, utiliza el concepto de "ventana" para considerar sólo los gradientes más recientes, actuando como una caché [12] [13].

- **Adam**: es una combinación de Momentum y RMSProp, calculando una combinación lineal entre el gradiente y el incremento anterior, y considera los gradientes recientemente aparecidos en las actualizaciones para mantener diferentes tasas de aprendizaje por variable. Es uno de los optimizadores más usados ya que está demostrado ser uno de los que mejores resultados aporta [12] [13].

A continuación, y para cerrar este apartado se muestra una gráfica con la comparación del coste de entrenamiento de los diferentes optimizadores citados. Donde se observa

que en líneas generales el optimizador Adam, es más liviano computacionalmente, además de ofrecer una buena respuesta en el entrenamiento de las redes neuronales.

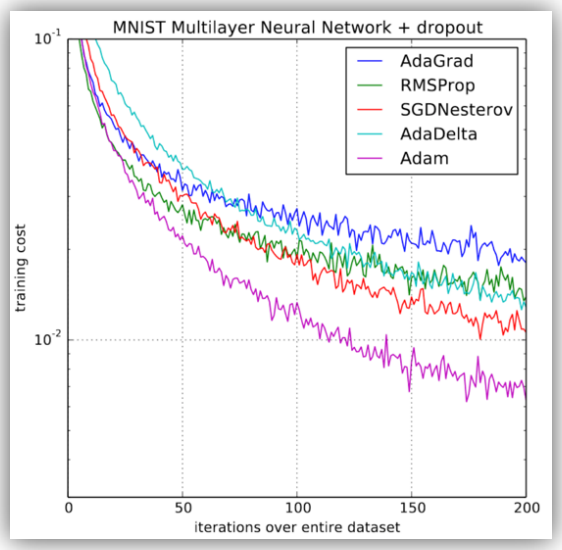


Ilustración 22: Gráfico comparativo de los optimizadores en el entrenamiento

1.2.2.9. Backpropagation

La solución que viene a dar el algoritmo de backpropagation es que en muchos casos el cálculo de los parámetros que minimizan la función de pérdida puede resultar computacionalmente inabordable. En el caso de una red neuronal, donde se tengan miles de pesos (bias), elementos cuyos valores hay que determinar en el entrenamiento de la red, no es factible el cálculo infinitesimal, por ello se suele recurrir a algoritmos como el descenso del gradiente. Pero, aplicar únicamente el algoritmo del descenso del gradiente implica calcular el gradiente de la función de pérdida en un punto determinado para unos parámetros concretos, lo que a su vez implica calcular la derivada parcial de la función de pérdida con respecto a todos y cada uno de los parámetros. Si este funcionamiento, se aplica a una red neuronal donde dicha red está compuesta por varias capas ocultas de miles de neuronas en cada una de ellas, hace que este cálculo sea muy complejo [5].

Anteriormente se utilizaban métodos basados en fuerza bruta que estimaban la derivada parcial en cuestión, modificando el valor del peso y viendo el efecto que tenía en la salida de la red, pero esto no solo era poco fiable, sino que era computacionalmente muy costoso.

El algoritmo de backpropagation se introdujo originalmente en la década de 1970, pero su importancia no se apreció completamente hasta que en 1986 David Rumelhart, Geoffrey Hinton y Ronal Williams publicaron el artículo “Learning representations by back-propagating errors”. Dicho artículo, describe que en redes neuronales la retropropagación tiene un funcionamiento más eficiente que los métodos que se conocían antes, haciendo posible utilizar redes neuronales para resolver problemas que anteriormente habían sido insolubles [13].

Funcionamiento matemático del algoritmo backpropagation

Partiendo de una red neuronal, con unos parámetros inicializados de forma aleatoria y, por lo tanto, un error elevado, se va a utilizar ese error para entrenar a la red.

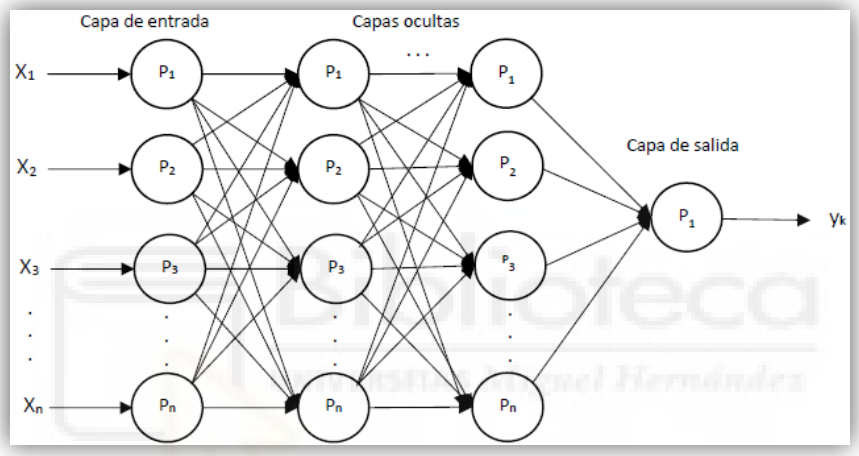


Ilustración 23: Ejemplo de red neuronal multicapa

El objetivo es calcular para cada parámetro dentro de la red, la derivada parcial del coste respecto a cada uno de los parámetros (incluido el de sesgo).

El primer paso sería, calcular las derivadas de los parámetros de la última capa (capa de salida). Para ello se analiza cual es el camino que conecta el valor del parámetro y el coste final [5].

$$\frac{\partial C}{\partial b^L} \frac{\partial C}{\partial w^L}$$

Ecuación 8: Derivadas de los parámetros de la última capa

El parámetro “w” en una neurona artificial participa en la suma ponderada. Para esta explicación se notará como “Z”. Esta a su vez, es pasada por la función de activación “a”. El resultado de las activaciones de la neurona en la última capa conforma el

resultado de la capa que es evaluada por la función de pérdida “C”, determinando así el error de la red neuronal [5].

$$C(a(Z^L))$$

Ecuación 9: Composición de funciones

De esta forma, se llega a una composición de funciones donde para calcular la derivada, se debe multiplicar cada una de las derivadas intermedias, de donde se obtienen las siguientes fórmulas:

$$\frac{\partial C}{\partial w^L} = \frac{\partial C}{\partial a^L} \cdot \frac{\partial a^L}{\partial z^L} \cdot \frac{\partial z^L}{\partial w^L}$$

$$\frac{\partial C}{\partial b^L} = \frac{\partial C}{\partial a^L} \cdot \frac{\partial a^L}{\partial z^L} \cdot \frac{\partial z^L}{\partial b^L}$$

$$Z^L = W^L a^{L-1} + b^L$$

Ecuación 10: Conjunto de derivadas parciales de la última capa

Desglosando cada una de las derivadas parciales se tiene:

$$\frac{\partial C}{\partial w^L}$$

Ecuación 11: Derivada de la activación respecto al coste

Derivada de la activación con respecto al coste, es decir, cómo varía el coste de la red neuronal cuando se varía la activación de las neuronas en la última capa. En la última capa, la activación de las neuronas es la salida de la red.

$$\frac{\partial a^L}{\partial z^L}$$

Ecuación 12: Derivada de la activación con respecto a la suma ponderada de la neurona

Derivada de la activación con respecto a Z, es decir, cómo varía la salida (output) de la neurona cuando se varía la suma ponderada de la neurona. Como lo único que separa

a Z con la activación de la neurona es la función de activación, únicamente se debe derivar la función de activación que se esté utilizando.

$$\frac{\partial z^L}{\partial w^L}$$
$$\frac{\partial z^L}{\partial b^L}$$

Ecuación 13: Derivada de la suma ponderada con respecto al término de sesgo

Representan como varía la suma ponderada (Z) con respecto a una variación en los parámetros. La derivada de la suma ponderada con respecto al término de sesgo es 1, ya que es un término independiente constante. Respecto a "w" la derivada es el valor de entrada de la neurona que conecta esa conexión para el que el parámetro hace referencia. De esta forma, los valores de entrada de la neurona (a_i^{L-1}) se corresponden con el output de las neuronas de la capa anterior (capa L-1) [5].

$$\frac{\partial z^L}{\partial w^L} = a_i^{L-1}$$

Ecuación 14: Relación de la entrada de la neurona con la salida de la capa anterior

Ahora bien, las derivadas parciales que representan la variación del error en función del valor de Z y la suma ponderada dentro de la neurona se puede simplificar de la siguiente forma:

$$\frac{\partial C}{\partial a^L} \cdot \frac{\partial a^L}{\partial z^L} = \frac{\partial C}{\partial z^L}$$

Ecuación 15: Variación del error en función de Z

De esta forma, la derivada resultante indica en qué grado se modifica el error cuando se produce un pequeño cambio en la suma de la neurona. Si la derivada es grande es que ante un pequeño cambio en el valor de la neurona este se verá reflejado en el resultado final. Pero, si, por el contrario, la derivada es pequeña da igual como se varíe el valor de la suma, ya que este no afectará al error de la red. En definitiva, esta derivada indica la responsabilidad que tiene la neurona en el resultado final y, por lo tanto, en el error. Este error de imputación de la neurona se suele interpretar como δ^L.

Por tanto, simplificando la expresión en función del error de las neuronas de la capa L:

$$\frac{\partial C}{\partial b^L} = \delta^L \cdot \frac{\partial C}{\partial a^L}$$
$$\frac{\partial C}{\partial w^L} = \delta^L \cdot \frac{\partial C}{\partial w^L}$$

Ecuación 16: Error de las neuronas en la capa última capa (L)

De esta forma, se tiene, por un lado, que la derivada del coste respecto al término de sesgo es igual al error de las neuronas y por otro, la derivada del coste respecto a w que es igual al error de las neuronas multiplicado por la activación de la capa previa.

$$\delta^L = \frac{\partial C}{\partial a^L} \cdot \frac{\partial C}{\partial a^L}$$

Ecuación 17: Error de imputación en la última capa L

Con este procedimiento explicado, se consiguen obtener las derivadas de la última capa. Ahora, simplemente para calcular las derivadas parciales de las siguientes capas se deberá aplicar el mismo procedimiento. Básicamente, lo que se va haciendo es ir moviendo el error de una capa a la capa anterior, distribuyendo el error en función de cuáles son las ponderaciones de las conexiones.

Aquí es donde se refleja la eficacia del algoritmo, ya que lo obtenido en la capa L-1 es extensible al resto de capas de la red, simplemente aplicando la misma lógica [5]:

1. Computar el error de la última capa (L).

$$\delta^L = \frac{\partial C}{\partial a^L} \cdot \frac{\partial a^L}{\partial z^L}$$

Ecuación 18: Error de la última capa (L)

2. Propagar el error a la capa anterior (L-1).

$$\delta^{l-1} = W^{l}\delta^{l} \cdot \frac{\partial a^{l-1}}{\partial z^{l-1}}$$

Ecuación 19: Error de la capa anterior (L-1)

3. Calcular las derivadas de la capa usando el error.

$$\frac{\partial C}{\partial b^{l-1}} = \delta^{l-1}$$

$$\frac{\partial C}{\partial w^{l-1}} = \delta^{l-1} a^{l-2}$$

Ecuación 20: Derivadas de la capa usando el error

1.2.2.10. Interrelación

Introducidos los conceptos de función de activación, función de pérdida (o coste), optimizador y backpropagation, en la siguiente ilustración se muestra cómo, internamente, se relacionan entre sí en la fase de entrenamiento de una red neuronal.

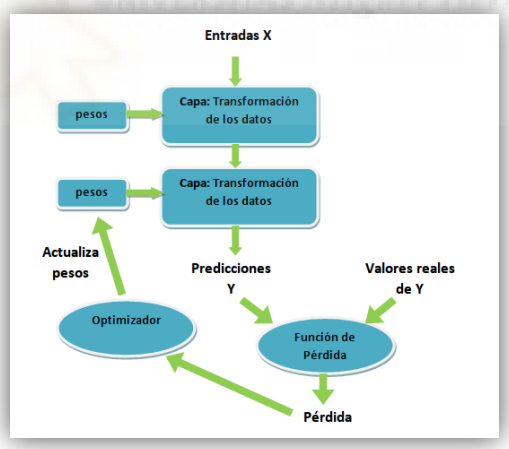


Ilustración 24: Interrelación de los componentes de una red neuronal

1.2.2.11. Overfitting y Underfitting

Antes de entrenar una red es importante realizar una separación del conjunto de datos (conocidos como dataset), de forma que haya dos grupos claramente diferenciados. Por un lado, los datos de entrenamiento y por otro los datos de validación. Normalmente, suele ser el 70% para entrenamiento y el resto para validación, aunque estas proporciones pueden variar en función del tamaño del dataset con el que se esté

trabajando. Esta separación de los datos no siempre garantiza unos resultados óptimos y en ocasiones se pueden presentar dos situaciones nada agradables como son el Overfitting y Underfitting.

Overfitting

Coloquialmente llamado “sobrentrenamiento o sobreajuste”. Este se produce cuando en la fase de entrenamiento, el modelo “memoriza” los datos en lugar de “entenderlos”, provocando un desempeño aparentemente muy bueno. Pero, sin embargo, a la hora de probar el modelo con nuevos datos, este no es capaz de adaptarse a ellos, provocando que su rendimiento sea muy inferior al esperado.

Existen diferentes técnicas para minimizar el overfitting:

1. **Dropout**: la funcionalidad del *dropout* es anular al azar la actividad de ciertas neuronas de la red durante el entrenamiento. Esto provoca que la red no memorice los datos de entrenamiento y aprenda a ser flexible y adaptarse a los datos de entrada [14].

2. **Regularizadores**: tratan de penalizar pesos y umbrales con valores absolutos demasiado grandes, sumando a la función de pérdida otra función que dependa de dichos parámetros. Los regularizadores más utilizados son *Lasso Regression* y *Ridge Regression* [14].

3. **Early-Stopping**: su funcionamiento básicamente es ir guardando el mínimo error obtenido en la validación. Ya que en ocasiones puede ocurrir que a medida que se avance en el número de iteraciones el error llegue un momento que tienda a aumentar [14].

Underffiting

También llamado “infrajuste”, este se produce cuando el modelo no es capaz de identificar los patrones. Al igual que ocurría con el Overfitting, existen diferentes técnicas y/o pautas para prevenirlo [14]:

1. Tratar los datos correctamente, eliminando los outliers y variables innecesarias.
2. Aumentar la complejidad del modelo.
3. Ajustar los parámetros del modelo.
4. Aumentar las iteraciones.

Representación gráfica del Overfitting y Underfitting

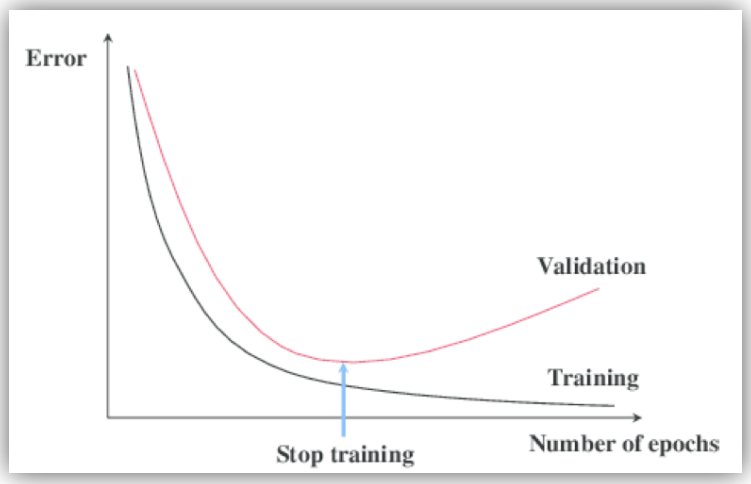


Ilustración 25: Overfitting y Underfitting

En la ilustración anterior, se aprecian dos zonas, la primera es en la que el error de validación baja a medida que avanza el entrenamiento y la segunda en la que comienza a aumentar. A estas zonas se les denomina Underfitting y Overfitting respectivamente [14].

1.2.2.12. Tipos de redes neuronales

Aunque este proyecto se centra en la red neuronal multicapa (Perceptrón multicapa), existen otros tipos de redes neuronales, donde la elección de una u otra arquitectura dependerá del problema a resolver. A continuación, se citan algunas de las redes neuronales más clásicas.

Red neuronal monocapa – Perceptrón simple

Es la red neuronal más simple, compuesta únicamente por una capa de neuronas que proyectan las entradas a una capa de neuronas de salida donde se realizan los determinados cálculos [16].

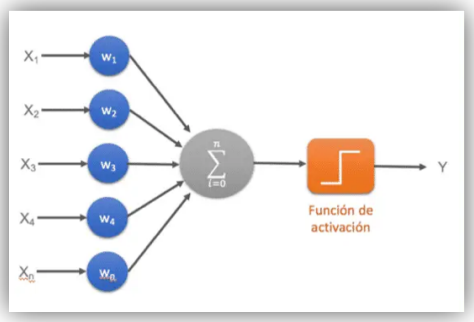


Ilustración 26: Estructura de red neuronal monocapa

Red neuronal multicapa – Perceptrón multicapa

Es una generalización de la red neuronal monocapa, la diferencia reside en que, a diferencia de la monocapa, que está compuesta por una capa de neuronas de entrada y otra capa de neuronas de salida, esta dispone de un conjunto de capas intermedias (capas ocultas) entre la capa de entrada y la de salida [16].

Dependiendo del número de conexiones, la red puede estar total o parcialmente conectada.

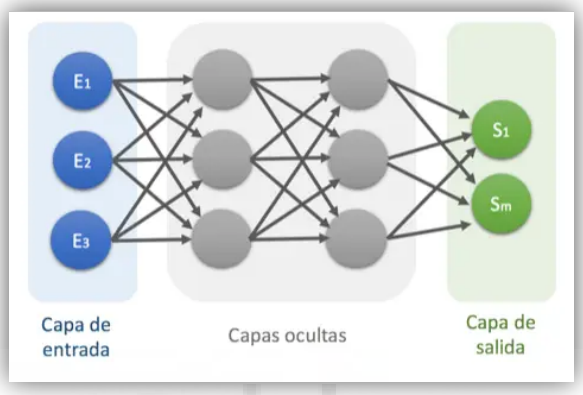


Ilustración 27: Estructura de una red neuronal multicapa

Red neuronal Convolucional (CNN)

La diferencia de este tipo de redes neuronales frente a las redes neuronales multicapa, reside en que cada neurona no se conecta con todas y cada una de las capas siguientes, sino solo con un subgrupo de ellas (se especializa), con esto se consigue reducir el número de neuronas necesarias y la complejidad computacional necesaria para su ejecución [16].

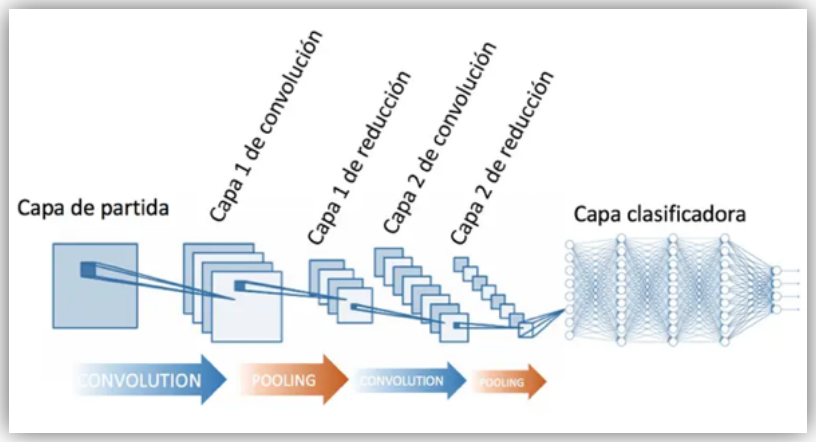


Ilustración 28: Estructura de una red neuronal convolucional

Red neuronal recurrente (RNN)

Este tipo de redes no tienen una estructura de capas, sino que permiten conexiones arbitrarias entre las neuronas, incluso pudiendo crear ciclos, con esto se consigue crear una temporalidad, permitiendo que la red tenga memoria [16].

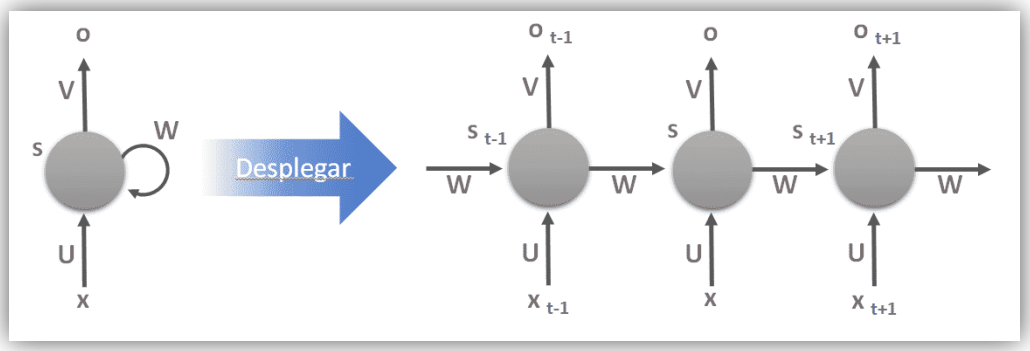


Ilustración 29: Estructura de una red neuronal recurrente

Red de base radial (RBF)

Las redes de base radial calculan la salida de la función en base a la distancia sobre un punto denominado centro. La salida es una combinación lineal de las funciones de activación radiales utilizadas para las neuronas individuales. Estas redes tienen la ventaja de no presentar mínimos locales donde la retropropagación pueda quedarse bloqueada [16].

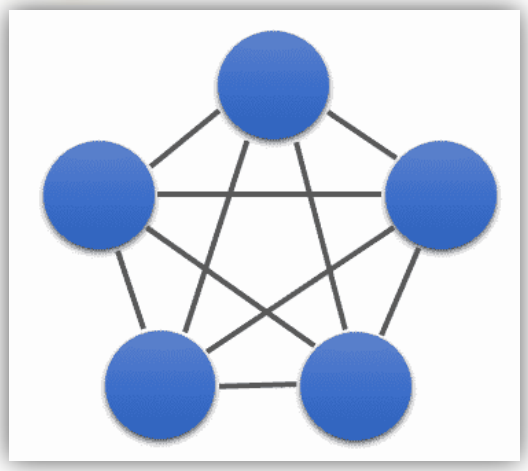


Ilustración 30: Representación gráfica de una red de base radial

1.3. Motivación del proyecto

Desde temprana edad siempre me ha gustado indagar en el porqué de las cosas; origen del universo, origen y motivo de la existencia humana, funcionamiento interno de las máquinas, etc. Pero, sin duda, una de las curiosidades que siempre me ha gustado investigar ha sido el cuerpo humano y todo lo relacionado con la salud.

Seguramente, de no haber estudiado ingeniería informática me habría decantado por la Nutrición y dietética, pues la alimentación, sin duda alguna constituye la pieza angular de la salud.

Mucha gente piensa que la mayoría de las enfermedades se deben a patologías que tienen que ver con la genética y cae en la mala costumbre de creer que, si un familiar directo tiene o tuvo cáncer, tú también lo tendrás hagas lo que hagas y te cuides lo que te cuides. No voy a mentir, yo también creía que era así, hasta que un médico, a quien admiro mucho, me lo aclaró.

Hace aproximadamente 4 años, en una analítica me dieron unos valores de transaminasas muy altos, aproximadamente 96, cuando unos valores normales oscilan entre 42 y 46. Luego quedó demostrado que en parte este valor tan alto se debía a la alta actividad física que por aquel entonces llevaba, pero claro, como en mi familia había antecedentes de cáncer de hígado, saltaron las alarmas.

Decidí acudir a Valencia a uno de los mejores médicos deportivos de España, un especialista en medicina ortomolecular. Recuerdo que una de las preguntas que le hice fue: ¿Cómo afecta la genética al desarrollo de enfermedades? A lo que él me respondió: “Los antecedentes genéticos sólo afectan un 2% al desarrollo de enfermedades, tiene muchísimo más impacto el estilo de vida que llevamos”.

Con la realización de este proyecto, pretendo aportar mi granito de arena a la salud, asentando las bases de una herramienta que contribuya al personal médico en la elaboración del diagnóstico y toma de decisiones. Haciendo ver al paciente lo importante que es llevar un estilo de vida saludable para prevenir el desarrollo de enfermedades.

CAPÍTULO 2: ESTADO DE LA CUESTIÓN

2.1. El diagnóstico médico

El diagnóstico médico es una de las tareas fundamentales de los médicos y la base para un tratamiento eficaz. Siendo este un medio imprescindible para establecer el tratamiento adecuado [17].

El diagnóstico se basa en el análisis de unos datos seguros. La decisión final solo será válida cuando descanse sobre nociones exactas y hechos precisos, pues cuando no se cumplan estos principios los resultados siempre serán erróneos [17].

Existen algunas premisas básicas sobre las que se apoya el diagnóstico médico [17]:

- **Relación médico-paciente**: fundamental para obtener la información que el médico necesita en el proceso de diagnóstico. Con una buena relación médico-paciente se logra dar tranquilidad y seguridad, así como una mejor exposición de los síntomas por parte del paciente.

- **La anamnesis:** base fundamental para el diagnóstico de los problemas de salud de los pacientes. Del 50 al 75% de los diagnósticos se hacen a través de preguntas-respuestas.

- **El examen físico**: complementa a la anamnesis, pues los signos físicos son marcas objetivas y verificables de la enfermedad y representan hechos sólidos e indiscutibles.

- **Asociación de los síntomas y signos**: es muy común en los médicos tratar de agrupar los síntomas y signos para realizar el diagnóstico, construyendo determinadas asociaciones: tríadas, tétradas, síndromes, etc.

Sin ninguna duda, el diagnóstico es una de las tareas más importantes que realiza un médico y, en ocasiones, muy dificultosa. Todo este proceso, puede traer consigo la realización de diferentes pruebas médicas: análisis de sangre, radiografías, cardiogramas, electros, resonancias, etc. Evidentemente, la realización de estas pruebas conlleva un gasto de tiempo que en ocasiones puede ser crucial para la correcta recuperación del paciente.

Actualmente, con la gran cantidad de datos generados cada día, los grandes avances tecnológicos y las capacidades de cómputo, sumado al auge que está experimentando la Inteligencia Artificial (IA) aportando grandes ventajas y utilidades en diferentes

ámbitos de la vida, ha hecho que se abra un abanico de posibilidades por descubrir, siendo la medicina uno de los campos que más están apostando por la IA. Diversos investigadores intentan indagar en el análisis de grandes cantidades de datos relacionados con la salud, aplicando la IA con el fin de encontrar patrones que puedan llevar al descubrimiento de nuevos tratamientos o formas de mejorar los diagnósticos de los pacientes.

2.2. Usos reales de la inteligencia artificial en la medicina

A continuación, se citan algunos de los usos reales de como la inteligencia artificial puede mejorar cualquier área de la medicina y la salud [18].

1. **Análisis médicos y de imágenes**: analizar diferentes pruebas médicas (resonancias, analíticas, estudios genéticos…).
2. **Diagnósticos médicos**: sobre todo para enfermedades que se desarrollan muy rápido y un pronto diagnóstico puede ser clave.
3. **Tratamientos farmacológicos**: analizar las secuencias genéticas para hallar vacunas.
4. **Genética**: detectar enfermedades raras y trastornos genéticos.
5. **Embarazos**: detección de malformaciones.
6. **Prótesis**: memorizar los patrones de movimiento de la persona.

La idea de ***Seek Health***, aplicación desarrollada en este proyecto, es la de ofrecer una ayuda en el diagnóstico de enfermedades haciendo uso de la inteligencia artificial, concretamente con una red perceptron multicapa.

2.3. Algunas aplicaciones relacionadas con la salud

Aunque se pueda pensar que el uso de la IA en la medicina es una posible realidad aún a medio y largo plazo, no es así. De hecho, sin ir más lejos, con toda esta situación que se está viviendo actualmente con el COVID-19, se ha aplicado inteligencia artificial, desarrollándose algunas aplicaciones y/o algoritmos que han sido de gran ayuda, como ejemplo:

MONAI (*Medical Open Network for AI*)

Haciendo uso de técnicas de *Deep Learning* e inteligencia artificial, esta aplicación permite detectar lesiones pulmonares en la segmentación de imágenes en 3D de las tomografías computarizadas de pacientes producidas por el COVID-19 [19]. Esta aplicación está disponible en código abierto, de manera gratuita en *GitHub*: https://github.com/Project-MONAI/MONAI.

Insights

Es un *software* desarrollado por la compañía canadiense BlueDot. Este *software* utiliza la Inteligencia Artificial para realizar predicciones y mapear el avance de las enfermedades infecciosas prácticamente en tiempo real. Convirtiéndose en un punto de apoyo importante durante la pandemia provocada por el COVID-19 [19].

AlphaFold

Desarrollada por DeepMind de Google, permite calcular la forma 3D de una proteína a partir de su secuencia de aminoácidos. El plegamiento de proteínas está considerado como uno de los problemas fundamentales de la biología [19].

LinearFold

Se trata de un algoritmo de IA para científicos y equipos médicos dedicados a la lucha contra el coronavirus. Este algoritmo, predice la estructura secundaria de la secuencia de ácido ribonucleico (RNA) de un virus. LinearFold es capaz de predecir la estructura secundaria de una secuencia de SARS-CoV-2 RNA en tan solo 27 segundos, 120 veces más rápido que utilizando métodos ya existentes [19].

Este algoritmo ha significado un punto de inflexión en la lucha contra el virus, gracias a la creación de un nuevo tipo de vacunas de RNA mensajero. En lugar del método tradicional, que consiste en insertar una pequeña porción del virus en un paciente para activar una respuesta inmune determinada. El RNA mensajero les dice directamente a las células cómo sintetizar una proteína que pueda provocar una respuesta inmunitaria determinada. De esta forma, se reduce significativamente el tiempo necesario para el desarrollo, testeo y aprobación de las nuevas vacunas. Sin ir más lejos, la vacuna contra el COVID-19 de la farmacéutica Pfizer utiliza este método [19].

MoCo (Momentum Contrast)

Un estudio de Facebook AI junto con *NYU Langone health´s Predictive Analytics Unit and Department of Radiology*, ha desarrollado modelos de *machine learning* que utilizan datos clínicos como las radiografías de tórax para predecir qué pacientes tienen mayor

riesgo de deterioro provocado por el COVID-19. En concreto, se trata de tres modelos [19]:

- Basándose en una radiografía, predice el deterioro del paciente.
- Apoyándose en una secuencia de rayos X, predice el deterioro del paciente.
- En base a una radiografía, predice la cantidad de oxígeno suplementario que podría necesitar un paciente.

CAPÍTULO 3: OBJETIVOS

3.1. Organización de los objetivos

En este apartado, se enumeran los diferentes objetivos que se pretenden alcanzar con la elaboración de este proyecto. Para ello, se han organizado en distintas categorías:

- **Generales**: están relacionados con el proyecto en sí, con el resultado que se espera conseguir del sistema final.
- **Técnicos**: se trata de las metas a alcanzar en la metodología de la implementación del proyecto a lo largo de su desarrollo.
- **Didácticos**: son aquellos en los que se han indagado para la elaboración del proyecto y que puedan ser útiles para el desarrollo de proyectos relacionados con el tema en cuestión.
- **Personales**: son aquellos que han significado un reto a abordar por el autor del proyecto.

3.1.1. Objetivos Generales

- Desarrollar las bases de un sistema que permita ayudar en el proceso de diagnóstico al personal sanitario de centros médicos.
- Implementar un modelo de red neuronal que permita la predicción de infartos cerebrovasculares.
- Desarrollar el prototipo funcional de una aplicación móvil multiplataforma con una interfaz sencilla e intuitiva.
- Proporcionar un servicio para la visualización de los datos de los pacientes; resultado y detalle de las analíticas, historiales médicos, etc.
- Desarrollar una API Rest que permita brindar la información necesaria.

3.1.2. Objetivos Técnicos

- Diseñar y desarrollar las bases de una infraestructura que permita albergar múltiples modelos, previamente entrenados, con el fin de predecir diversas enfermedades.

3.1.3. Objetivos Didácticos

- Instalación de las librerías necesarias para la elaboración del proyecto en el entorno de Anaconda.
- Instalar y configurar una base de datos Oracle en un sistema operativo Ubuntu.

- Preparar un entorno de trabajo en Visual Studio Code para el desarrollo de aplicaciones móviles en Flutter.
- Configurar git y Jupyter Notebook en Visual Studio Code.

3.1.4. Objetivos Personales

- Aprender el funcionamiento interno de las redes neuronales, tanto desde un punto de vista conceptual como matemático.
- Aprender los conceptos básicos de los lenguajes de programación Python y Dart.
- Iniciarse en el desarrollo de aplicaciones multiplataforma con Flutter, uno de los SDK más utilizados en la actualidad por las empresas más punteras.
- Elaborar el diseño de un proyecto de gran envergadura en distintos ámbitos.
- Crear un prototipo de alta fidelidad en adobe Xd.
- Ofrecer una pequeña aportación al mundo de la medicina y salud.

3.2. No objetivos

No son objetivos de este proyecto los siguientes puntos:

- Experimentar con todas y cada una de las arquitecturas de redes neuronales, así como con diferentes librerías para su construcción.
- Obtener un modelo de red neuronal con una eficiencia del 100%.
- Desarrollar una aplicación completamente funcional.
- Testear en distintos tamaños de pantalla la aplicación móvil para validar su correcta adaptación a esta. Es decir, garantizar una respuesta 100% adaptativa.
- Visualizar la aplicación móvil en distintos idiomas en función de la configuración del dispositivo.
- Desarrollar una aplicación que cumpla con la normativa 1112/2018 sobre la accesibilidad de sitios webs y aplicaciones para dispositivos móviles.

CAPÍTULO 4: HIPÓTESIS DE TRABAJO

En este capítulo se indican todas y cada una de las tecnologías, herramientas, lenguajes y configuraciones utilizadas para la elaboración de este proyecto.

4.1. Herramientas generales utilizadas

4.1.1. Visual Studio Code

Prácticamente, todos los sistemas operativos traen por defecto un editor de texto plano, con el que se permite modificar archivos o tomar notas fácilmente. En el caso de Windows, por ejemplo, viene con el Bloc de notas. En macOS, con TextEdit. Aunque estos dos editores cumplen sus funciones, la verdad es que están muy limitados, por ejemplo, para usuarios programadores que requieren de herramientas con más funcionalidad. Aquí es donde entra en juego Visual Studio Code (VSC).

Visual Studio Code es un editor de código fuente liviano y muy potente. Este se encuentra disponible para Windows, macOS y Linux. Una de las grandes ventajas que incorpora esta herramienta es la cantidad de extensiones que permite acoplarle, consiguiendo adaptarse a prácticamente cualquier desarrollo que se requiera [20].

Sin duda alguna, esta aplicación ha sido de las más importantes en la elaboración de este proyecto, ya que ha estado presente en todos y cada uno de los desarrollos llevados a cabo (Red neuronal, Aplicación móvil y API Rest).

4.1.1.1. Extensiones utilizadas de VSC

- **Awesome Flutter Snippets**: es una colección de fragmentos y accesos directos para funciones y clases de Flutter de uso común.
- **Bracket Pair Colorizer 2**: permite personalizar los colores de los elementos estructurales del código, dando una mayor legibilidad al código.
- **Dart**: permite dar soporte al lenguaje de programación Dart en VSC y proporciona herramientas para editar, refactorizar, ejecutar y recargar las aplicaciones móviles Flutter.
- **Flutter**: soporte para el desarrollo de aplicaciones en Flutter con VSC.
- **GitHub**: permite integrar GitHub en VSC, además permite realizar las acciones (commit, push…) a través de botones de una forma muy sencilla y rápida.
- **Jupyter**: da soporte básico a cuadernos Jupyter (extensión jpynb).
- **VS Code Jupyter Notebook Previewer**: esta extensión es un complemento a la anterior extensión. Permite previsualizar cuadernos Jupyter dentro de VSC.

- **Material Icon Theme**: este plugin es más una ayuda visual que funcional, ya que proporciona una colección de iconos para VSC.
- **Pupspec Assist**: permite añadir y/o actualizar de una forma muy sencilla y rápida las dependencias de un proyecto Dart o Flutter.

4.1.2. SQLDeveloper

Oracle SQL Developer es una interfaz gráfica gratuita que permite a usuarios expertos y no expertos de base de datos realizar tareas de una forma más rápida y sencilla que mediante código SQL.

4.1.3. Oracle DataBase

Es un sistema gestor de base de datos relacionales, desarrollado por la empresa Oracle Corporation, siendo ampliamente utilizado en el ámbito profesional. En la actualidad (septiembre 2021) cuenta con siete ediciones diferentes.

- Lite Edition (LE)
- Personal Edition (PE)
- Express Edition (XE)
- Standard Edition 1 (SE1)
- Standard Edition 2 (SE2)
- Stardard Edition (SE)
- Enterprise Edition (EE)

A excepción de la edición XE (Express Edition) todas son de pago. XE es la utilizada en este proyecto.

4.1.4. Cacoo

Cacoo es una herramienta online que permite crear de manera individual o colaborativa gran variedad de organizadores gráficos, tales como: mapas mentales, wireframes, diagramas UML, etc. La herramienta permite compartir el diseño a través de varias plataformas o exportarlo a .svg, .png y otros tipos de formatos [21].

4.1.5. Control de versiones

Un sistema de control de versiones ayuda a llevar un registro y administrar cualquier cambio en el código del proyecto software. Tener un control de versiones en los proyectos no es obligatorio, pero es altamente recomendable.

4.1.5.1. Git

Git es el sistema de control de versiones distribuido por excelencia. Además, es gratuito y de código abierto. Permite manejar desde proyectos pequeños hasta muy grandes de una manera rápida y efectiva [22].

Para realizar la instalación se debe descargar el instalador de la web oficial de git (https://git-scm.com/downloads) y elegir el sistema operativo. Para el caso de este proyecto, se ha utilizado el instalador para el sistema operativo de Apple, macOS.

Ilustración 31: Zona de descarga oficial de git

Una vez finalizada la instalación, se debe validar. Para ello, en una terminal se debe escribir el siguiente comando:

Ilustración 32: Comando para visualizar la versión instalada de git

4.1.5.2. GitHub

GitHub es un servicio en la nube que ofrece un lugar donde almacenar y administrar el código. Además, permite llevar un registro y control de cualquier cambio realizado sobre dicho código.

Esta herramienta ha sido de gran utilidad en la elaboración de este trabajo de fin de grado.

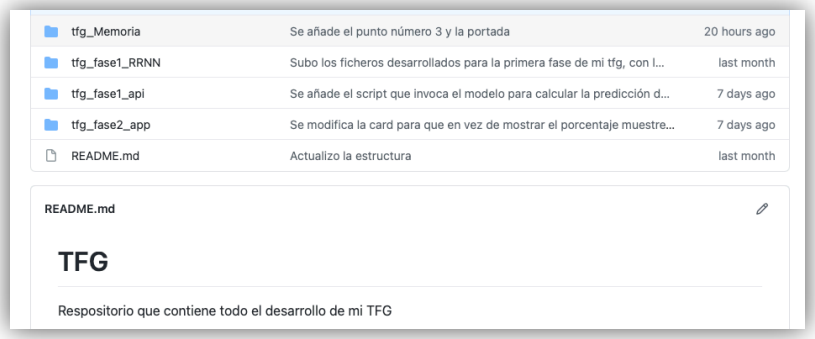


Ilustración 33: Estructura de directorios del repositorio para el TFG

Una vez introducidas las herramientas generales utilizadas, se procede a exponer los lenguajes y librerías usadas en cada fase del proyecto.

4.2. Red Neuronal

4.2.1. Lenguaje Python

Python es un lenguaje de programación de alto nivel, orientado a objetos e interpretado, es decir, el código escrito no se traduce a un formato legible por el ordenador en tiempo de ejecución. Fue creado por Guido Van Rossum en los años 90 [23].

Uno de los beneficios más importantes de Python es que tanto la librería estándar como el intérprete están disponibles gratuitamente, tanto en forma binaria como en forma de fuente. De hecho, viene de forma nativa en el núcleo de Linux [23].

Ilustración 34: Comando para visualizar la versión instalada de Python

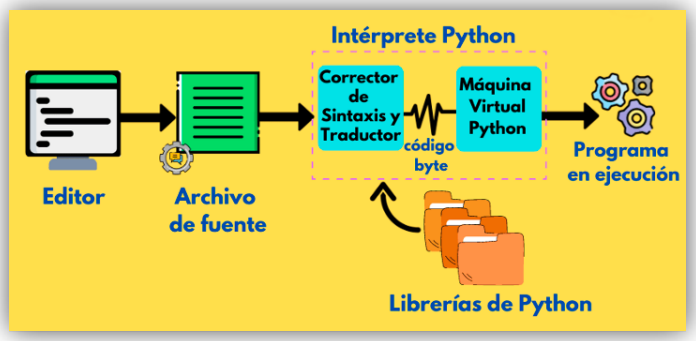


Ilustración 35: Funcionamiento del intérprete de Python

Python se ha convertido en un lenguaje de propósito general y es uno de los más conocidos y gustados. De las características más importantes se pueden destacar las siguientes [23] [24]:

- Lenguaje interpretado, no compilado.
- Fuertemente tipado.
- Es multiplataforma.
- Multiparadigma: soporta varios paradigmas de programación como orientado a objetos, estructurada, programación imperativa y en menor medida, programación funcional.
- Gran flexibilidad.
- Sintaxis muy sencilla e intuitiva.
- Rápida curva de aprendizaje.

En la inteligencia artificial es uno de los lenguajes más utilizados, dada la alta variedad de librerías que existen para este lenguaje.

Para el desarrollo de la red neuronal se han utilizado las librerías y paquetes que se detallan a continuación.

4.2.1.1. Jupyter Notebook

Es un entorno de trabajo interactivo y de código abierto que permite desarrollar código en Python de manera dinámica, a la vez que permite integrar en un mismo documento bloques de texto, gráficos, imágenes y vídeos [25].

Principalmente fue pensado para seguir una estructura cliente-servidor por el que se pudiera acceder vía web. Para el desarrollo de este proyecto, se ha optado por incrustarlo en Visual Studio Code, a través de los correspondientes plugins.

4.2.1.2. NumPy

Es un paquete de Python cuyo significado es “Numerical Python”. Es una librería principalmente utilizada en la informática científica, proporcionando potentes estructuras de datos, implementando matrices multidimensionales [26].

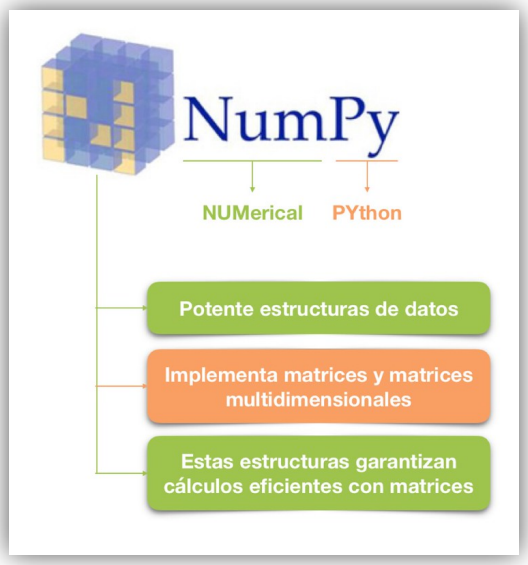


Ilustración 36: Librería Numpy de Python

4.2.1.3. Pandas

Es una librería de Python especializada en el manejo y análisis de estructuras de datos. Las principales características que posee la librería son [27]:

- Define nuevas estructuras de datos basados en los arrays de la librería NumPy, pero añadiéndole nuevas funcionalidades.
- Permite leer de forma muy rápida y fácil ficheros en formato CSV, Excel y Base de datos SQL.
- Permite el acceso a los datos mediante el uso de índices, nombre de filas y/o columnas.
- Ofrece métodos para reordenar, dividir y combinar conjuntos de datos.
- Permite el trabajo de serie temporales

4.2.1.4. Matplotlib

Es una librería de Python especializada en la creación de gráficos de dos dimensiones [28].

Algunos de los gráficos que permite crear y personalizar son:

- Histogramas
- Diagrama de colores
- Diagrama de barras
- Diagramas de caja y bigotes

- Diagrama de violín
- Diagramas de dispersión
- Diagramas de líneas
- Diagramas de áreas
- Diagramas de contorno
- Mapas de color

4.2.1.5. Scikit-Learn (Sklearn)

Es una librería de código abierto y es reutilizable en varios contextos. Esta librería está constituida sobre SciPy (Scientific Python) e incluye las siguiente librerías o paquetes [29]:

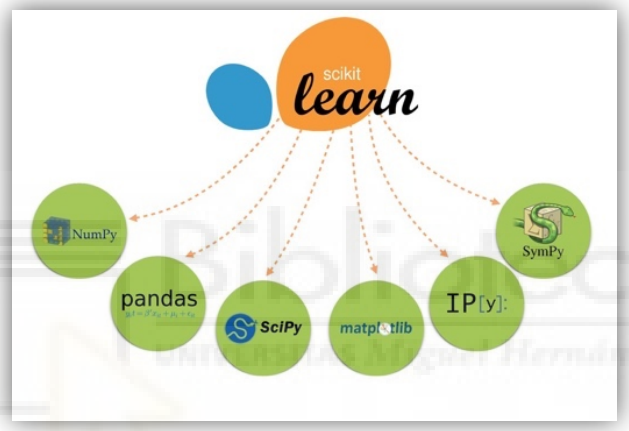


Ilustración 37: Librería Scikit-Learn

4.2.1.6. Tensorflow

Es una biblioteca de software de código abierto para computación numérica que permite crear y utilizar gráficos de flujo de datos, es decir, estructuras que describen como los datos se mueven a través de un gráfico, o una serie de nodos de procesamiento. Cada nodo del gráfico representa una operación matemática, y cada conexión entre nodos es una matriz de datos multidimensional, llamada tensor. El cálculo de matrices es el fuerte de las GPUs, por ello Tensorflow incorpora la aceleración por GPU [30] [31].

Es ampliamente utilizado para el aprendizaje automático. Cuenta con un ecosistema integral y flexible de herramientas, bibliotecas y recursos de la comunidad.

4.2.1.7. Keras

Es una biblioteca de código abierto de redes neuronales desarrollada en Python por François Chollet, ingeniero de Google [32].

Keras es una abstracción para la creación de modelos de aprendizaje sobre Tensorflow. Dado que las redes neuronales son un tipo de gráfico de flujo de datos, keras y Tensorflow ofrecen una combinación perfecta, haciendo un tándem que ofrece sencillez de uso, potencia y rapidez en la construcción de redes neuronales [33].

4.2.2. Anaconda Distribution como IDE

Anaconda es sin duda, una de las grandes herramientas descubiertas en la elaboración de este proyecto.

Anaconda es una suite completa de código abierto enfocada a la ciencia de datos con Python. Anaconda se agrupa en cuatro sectores: Anaconda Navigator, Anaconda Proyect, Librerías de Ciencia de datos y Conda. Todas ellas vienen implícitamente con la instalación [34].

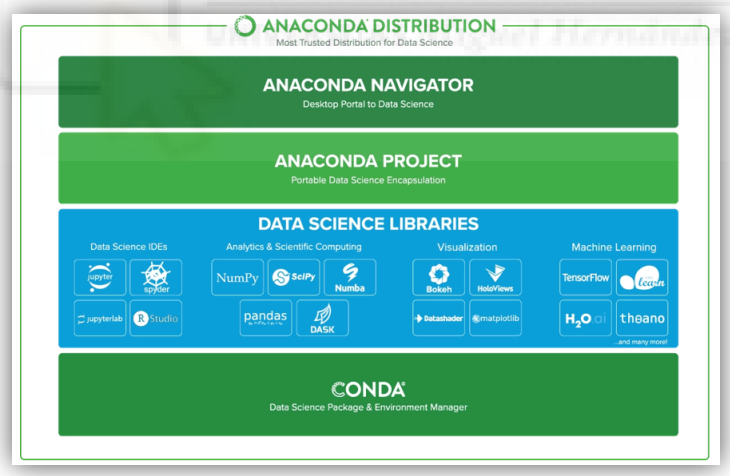


Ilustración 38: Estructura Anaconda Distribution

De las cuatro secciones mencionadas, la que gira en torno a este proyecto es **Conda**. Conda permite generar entornos virtuales personalizados. En dichos entornos, se pueden instalar las librerías o paquetes que se necesiten, de manera que estos entornos sean totalmente independientes entre sí.

Antes de pasar a explicar el proceso de instalación es necesario instalar la herramienta "pip" mediante línea de comandos:

```
sudo easy_install install pip
```

4.2.2.1.1. Instalación

La instalación de Anaconda se puede realizar de dos formas: Gráfica o mediante línea de comandos. En este caso, se utilizó el entorno gráfico. Dado que se ha trabajado con el sistema operativo macOS, la extensión del instalable es .dmg y se puede descargar desde la web oficial: https://www.anaconda.com/products/individual

Paso 1: instalar Anaconda. Su instalación es muy sencilla, del tipo siguiente, siguiente.

Para validar que la instalación ha finalizado correctamente, se puede abrir la terminal y automáticamente se ubicará en el entorno (base) por defecto que conda crea con la instalación.

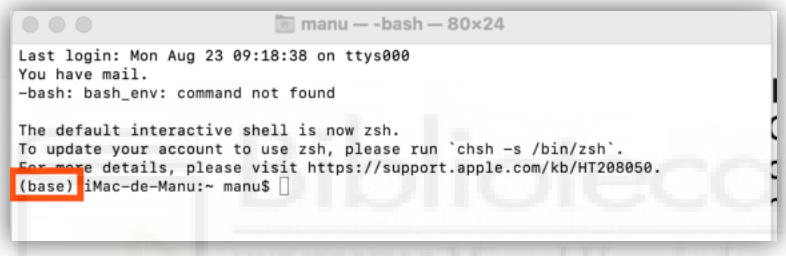


Ilustración 39: Validación de la instalación de Conda

Paso 2: se puede trabajar sin problemas en el entorno "base" pero en este caso, se ha creado uno nuevo específico para este trabajo de fin de grado. Desde el terminal:

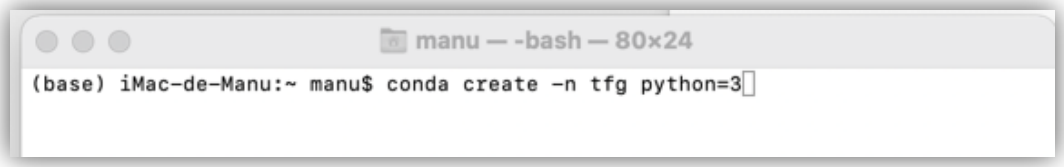


Ilustración 40: Comando para la creación del entorno del tfg en Conda

Ahora, una vez creado el entorno, se activa, para ello:

Ilustración 41: Comando para ubicarse en el entorno tfg de Conda

4.2.2.1.2. Configuración

Una vez se está ubicado en el entorno, se precede a su configuración, instalando las librerías y paquetes necesarios.

Paso 1: instalar el paquete Jupyter.

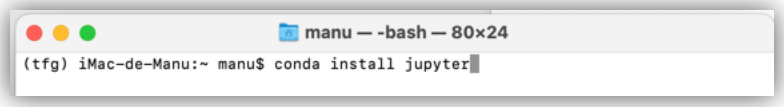


Ilustración 42: Comando para instalar jupyter en Conda

Paso 2: instalar la librería TensorFlow.

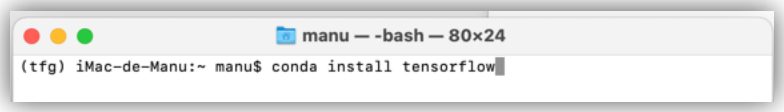


Ilustración 43: Comando para instalar la librería TensorFlow en Conda

Implícitamente la instalación de TensorFlow lleva consigo la librería numpy.

Paso 3: instalar la librería pandas.

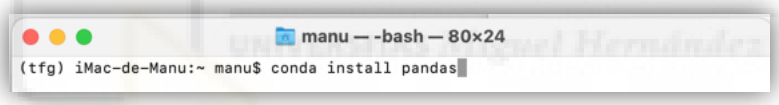


Ilustración 44: Comando para instalar la librería Pandas en Conda

Paso 4: instalar ipython.

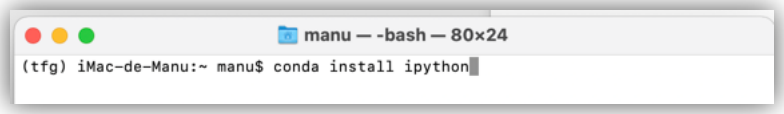


Ilustración 45: Comando para instalar la librería ipython en Conda

Paso 5: instalar la librería matplotlib.

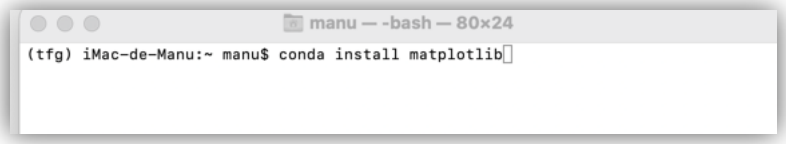


Ilustración 46: Comando para instalar la librería matplotlib en Conda

Paso 6: instalar versión 2.0 de Tensorflow.

Para poder utilizar el framework keras es necesario actualizar Tensorflow a su versión 2.0.

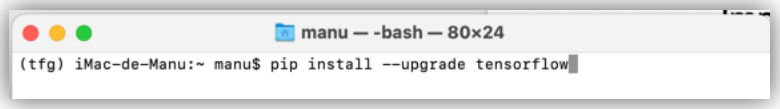


Ilustración 47: Comando para actualizar a TensorFlow 2.0

Paso 7: instalar la librería sklearn.

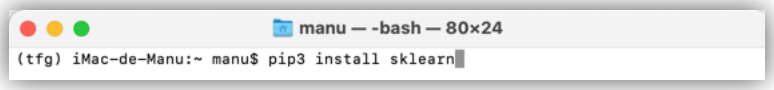


Ilustración 48: Comando para instalar la librería sklearn

Una vez completada la instalación de todas las librerías necesarias, llega el momento de enlazarlo con Visual Studio Code.

Cuando se abre o crea un fichero de Jupyter (.jpynb) Visual Studio Code comienza a activar las extensiones necesarias (previamente instaladas), detectando el entorno, en este caso el entorno 'tfg', creado en el apartado 4.2.2.1.1 Instalación.

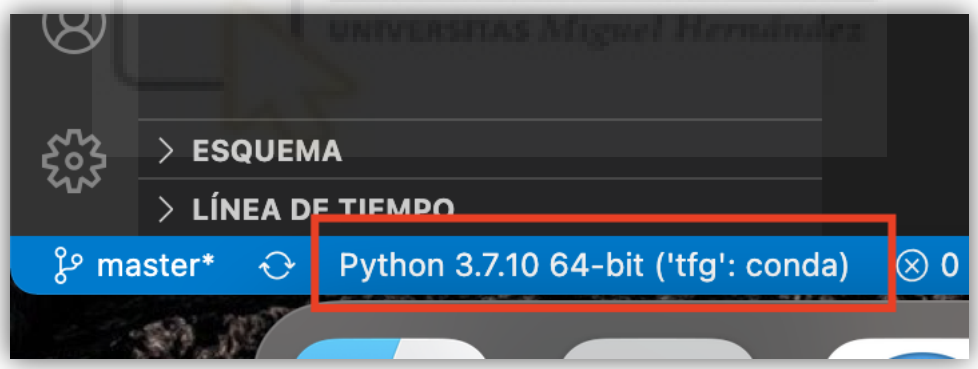


Ilustración 49: Detección del entorno tfg en VSC

En ocasiones, es posible que al abrir VSC no detecte el entorno de forma automática, por lo que se deba "forzar" su elección.

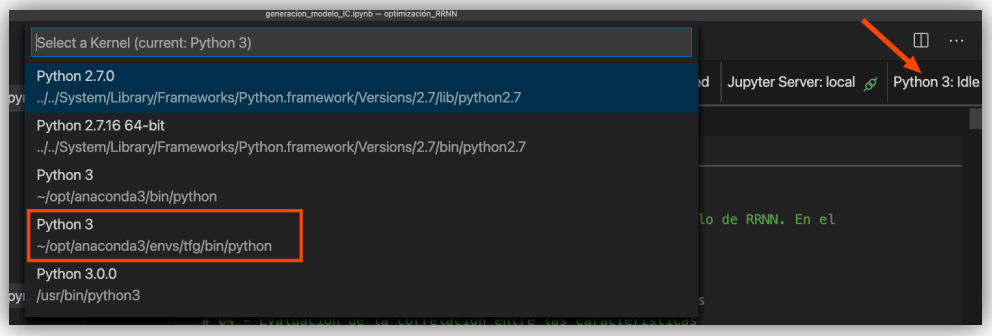


Ilustración 50: Selección del entorno conda en VSC

Por último, se valida la ejecución del fichero para ver si se han enlazado bien conda y VSC.

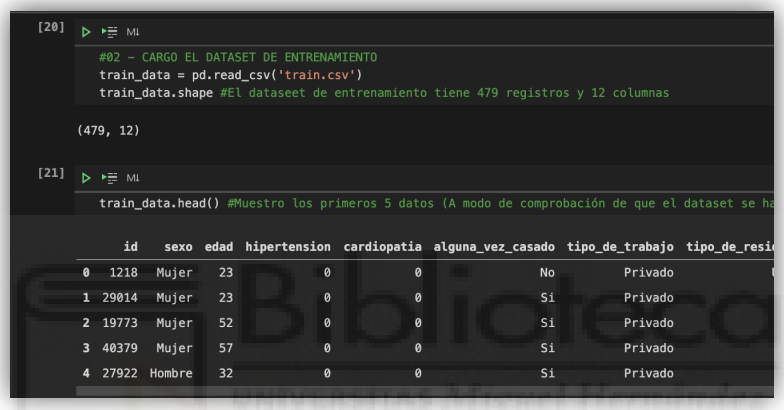


Ilustración 51: Carga del dataset en VSC con Pandas

4.3. API Rest

Una API es un conjunto de definiciones y protocolos que se utiliza para desarrollar e integrar el *software* de las aplicaciones. La API vendría a ser como el mediador entre los usuarios y los recursos o servicios web que se quieren obtener. Rest vendría a definir un conjunto de límites de arquitectura [35].

Cuando se envía una solicitud del cliente a través de una API de RESTful, esta transfiere una representación del estado del recurso requerido a quien lo haya solicitado. La información, se entrega por medio de HTTP, normalmente en uno de estos formatos: JSON, HTML, XLT, Python, PHP o texto sin formato.

Para que una API se considere de RESTful, debe cumplir los siguientes criterios [35]:

- Arquitectura cliente-servidor.

- Comunicación entre el cliente y el servidor sin estado, no almacenándose la información del cliente en las solicitudes.
- Interfaz uniforme entre los elementos, para una trasferencia de información estandarizada.
- Un sistema de capas que organiza en jerarquías invisibles para el cliente cada uno de los servidores.
- Código disponible según se solicite, es decir, capacidad de enviar códigos ejecutables del servidor al cliente cuando se requiera.

4.3.1. Librerías utilizadas

4.3.3.1. Librería Flask de python

Flask es un "mini" *Framework* escrito en Python y concebido para facilitar el desarrollo de Aplicaciones o servicios Web bajo el patrón Modelo-Vista-Controlador (conocido por sus siglas como MVC) [36].

La palabra "mini" en este caso no se viene a referir a que con esta librería únicamente se puedan realizar proyectos pequeños, si no que al instalar Flask ya se disponen de las herramientas necesarias para crear aplicaciones funcionales. Obviamente si se necesitan más funcionalidades se deberá complementar mediante extensiones (plugins) [36].

Ventajas de utilizar Flask [36]**:**

1. Permite desarrollar una aplicación básica de forma ágil y rápida.
2. Incluye un servidor web de desarrollo: por lo que no se necesita una infraestructura extra con un servidor web para probar las aplicaciones.
3. Dispone de un depurador y soporte integrado para pruebas unitarias.
4. Es perfectamente compatible con Python 3.
5. Existe una gran documentación.
6. Muy buen manejo de las rutas de la aplicación.
7. Permite el uso de sesiones.
8. Permite construir servicios web como APIs REST).
9. Es Open Source.

4.3.3.2. cx_Oracle

Es un módulo de extensión de Python que permite el acceso a Base de datos de Oracle.

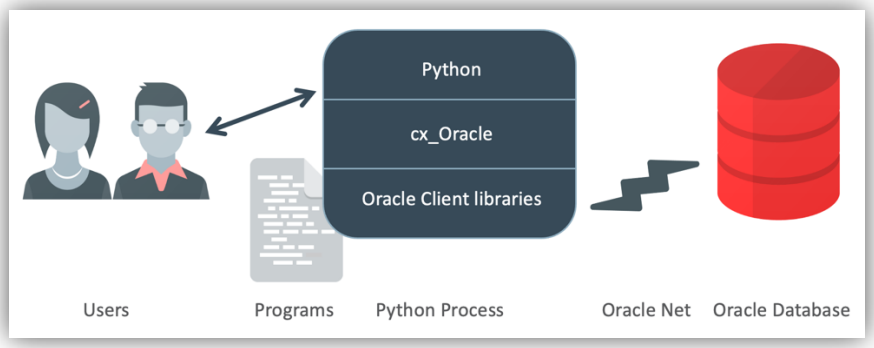


Ilustración 52: Arquitectura de la librería cx_Oracle

En este caso, dado que la base de datos está en remoto, ha sido necesario realizar los siguientes pasos extras [37] [38]:

1. Descarga del DMG básico de 64 bits para Oracle.
2. Montar (con doble clic) el DMG descargado en el punto 1.
3. Ejecutar en la terminal el script install_ic.sh incluido en el paquete.
4. Para hacer referencia al paquete descargado, es necesario indicar en el import de donde se quiera usar, la siguiente referencia:

```
import cx_Oracle
cx_Oracle.init_oracle_client(lib_dir="/Users/manu/Downloads/instantclient_19_8
")
```

Ilustración 53: Importación de la librería cx_Oracle

Una vez se han realizado estos pasos, es necesario instalar la librería cx_Oracle en el entorno de conda.

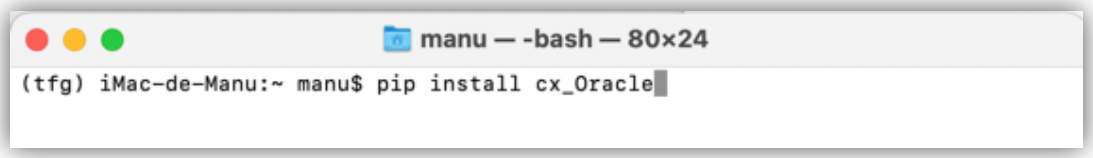


Ilustración 54: Comando para instalar la librería cx_Oracle en Conda

4.3.2. Herramienta utilizada para validar la API

La API Rest construida con Flask en este proyecto, se ha programado para que brinde los resultados en formato JSON. Para validar el correcto funcionamiento de cada funcionalidad (rutas) se ha hecho uso de la aplicación Postman, que permite invocar a cada una de ellas.

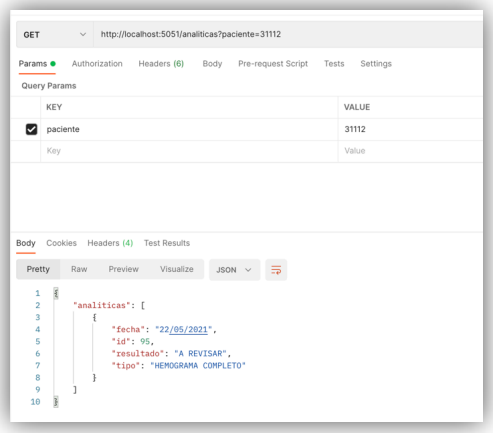


Ilustración 55: Ejemplo de uso de la aplicación Postman

4.4. Aplicación móvil

4.4.1. Lenguaje Dart

Es un lenguaje de programación optimizado para aplicaciones rápidas en múltiples plataformas. Fue desarrollado en 2021 por Google y es utilizado para crear aplicaciones móviles, de escritorio, backend y web [39].

Dart es un lenguaje orientado a objetos, definido por clases, usando una sintaxis al estilo de C, aunque en algunos aspectos recuerda a JavaScript, python y Java.

Características de Dart [39]:

- **Programación estructurada y flexible**: se diseñó de manera que pudiera ser utilizado en proyectos desde una sola persona hasta proyectos muy complejos.
- **Lenguaje familiar y fácil de entender**.
- **Permite adaptarse a cualquier navegador web**: Dart se puede ejecutar de dos maneras; en una máquina virtual, o en un motor de JavaScript utilizando un compilador para traducir el código, permitiendo adaptarse a cualquier navegador.
- **Basado en clases e interfaces o POO**: se facilita la encapsulación y la reutilización del código.

4.4.1.1. Instalación

Para realizar la instalación de dart en macOS se han realizado los siguientes pasos:

Paso 1: instalar la herramienta brew mediante el siguiente comando.

```
/bin/bash -c "$(curl -
fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
```

Paso 2: verificar la instalación de brew

```
brew tap dart-lang/dart
```

Paso 3: instalar dart mediante brew

```
brew install dart
```

Paso 4: verificar instalación de dart

```
dart --version
```

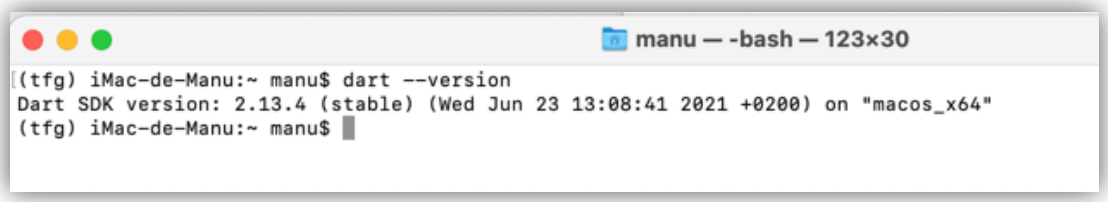


Ilustración 56: Comando para verificar la instalación de Dart

4.4.2. Flutter

Flutter es un SDK desarrollado por Google con la finalidad de crear aplicaciones móviles multiplataforma, es decir, tanto para Android como para iOS (Apple). Está escrito en C, C++ y mayormente en Dart. Inicialmente fue desarrollado como un software para uso interno de la compañía, pero dado el enorme potencial que tenía se lanzó como proyecto de código libre. Hoy en día es uno de los proyectos de desarrollo de aplicaciones móviles que más está creciendo. Además, desde la última versión estable, también es posible realizar aplicaciones Web y de escritorio para Windows y Mac [40].

Las tres principales ventajas que ofrece Flutter respecto a otros SDK de desarrollo de aplicaciones multiplataforma son las siguientes:

1. Compila en nativo tanto en Android como en iOS: por lo que no necesita una máquina virtual para funcionar, esto hace que su rendimiento sea mucho mayor.
2. La creación de interfaces gráficas es muy flexible.
3. El desarrollo es muy rápido.

Funcionamiento de Flutter

Flutter utiliza un motor gráfico, llamado Skia, que renderiza en 2D los elementos gráficos. La capa del motor está escrita en C++ y la de los Widgets en Dart [40].

Todo en Flutter es un Widget. Widget es la palabra más repetida cuando se hace referencia a Flutter, se refiere a los elementos gráficos que componen una vista. Por ejemplo, un botón, un texto o una imagen. Aunque también hay Widgets más complejos formados por otros Widgets.

Flutter utiliza Dart como lenguaje de programación y, quizá esto pueda ser una desventaja ya que Dart no es un lenguaje muy popular por lo que sería una limitación. Lo bueno que tiene es que es muy parecido a Java y C# por lo que si se tiene experiencia en alguno de estos lenguajes su curva de aprendizaje es muy rápida [40].

4.4.2.1. Instalación

Se exponen los pasos necesarios para llevar a cabo la instalación de Flutter en macOS. Antes de nada, es necesario disponer de mínimo la versión 2.1 de git.

Paso 1: instalar Android Studio (incluido el SDK) y xCode.

Paso 2: ir a la página oficial de Flutter https://flutter.dev/docs/get-started/install/macos. Una vez ahí se pulsa en install y se elige el sistema operativo, macOS en este caso.

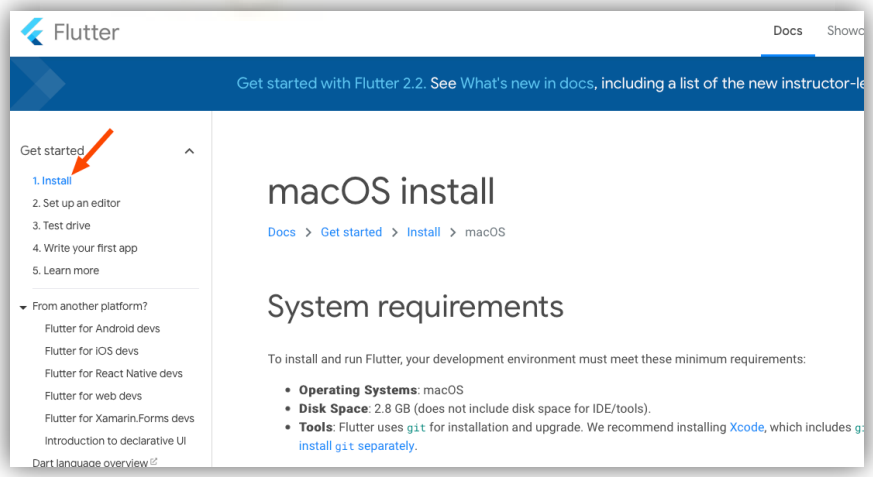


Ilustración 57: Guía de instalación de Flutter oficial

Paso 3: descargar el SDK de Flutter.

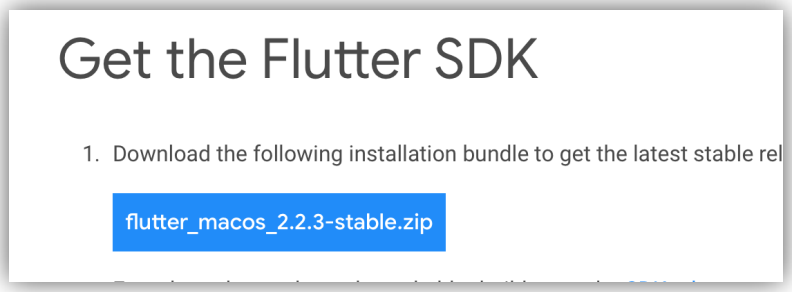


Ilustración 58: Descarga de Flutter para macOS desde la web oficial de Flutter

Paso 4: extraer su contenido.

Paso 5: crear carpeta **development** (es el nombre que recomienda la comunidad de Flutter) en el home del usuario.

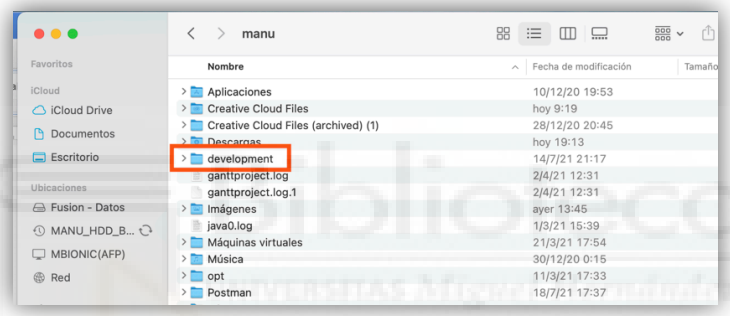


Ilustración 59: Creación de la carpeta development en home del usuario

Paso 6: mover el directorio extraído en el paso 4 a la carpeta development creada en el paso anterior.

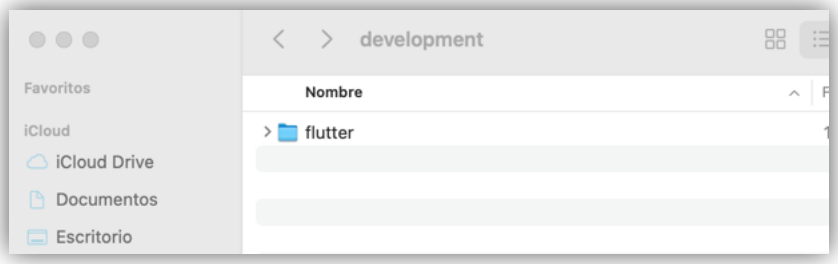


Ilustración 60: Ubicación del directorio Flutter en la carpeta development

Paso 7: abrir la terminal y ubicarse en la carpeta principal del usuario mediante el comando “cd”.

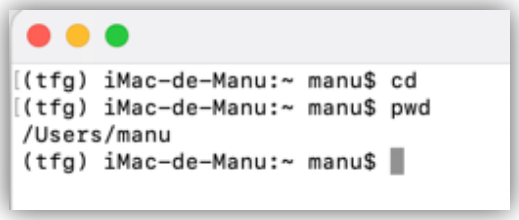


Ilustración 61: Comando para ubicarse en el home del usuario

Paso 8: en este caso, al estar utilizando el sistema operativo de Big Sur hay que escribir en la terminal `vim .profile` y añadir las siguientes dos líneas.

Ilustración 62: Configuración path Flutter en el profile del usuario

Paso 9: reiniciar la terminal para que vuelva a cargar el *profile.*

Paso 10: desde la terminal, se valida la instalación de Flutter mediante el comando `flutter --version`

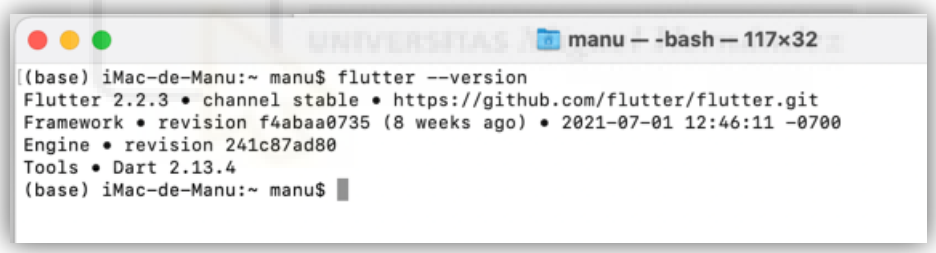


Ilustración 63: Comando para validar la instalación de Flutter

Paso 11: ejecutar la aplicación xCode desde la línea de comando.

```
sudo xcode-select --switch /Applications/Xcode.app/Contents/Developer
```

```
sudo xcodebuild –runFirstLaunch
```

Paso 12: aceptar la licencia de xCode.

```
sudo xcodebuild –license
```

Paso 13: instalar cocapods.

```
sudo gem install cocoapods
```

Paso 14: abrir Android Studio y crear una máquina virtual Android.

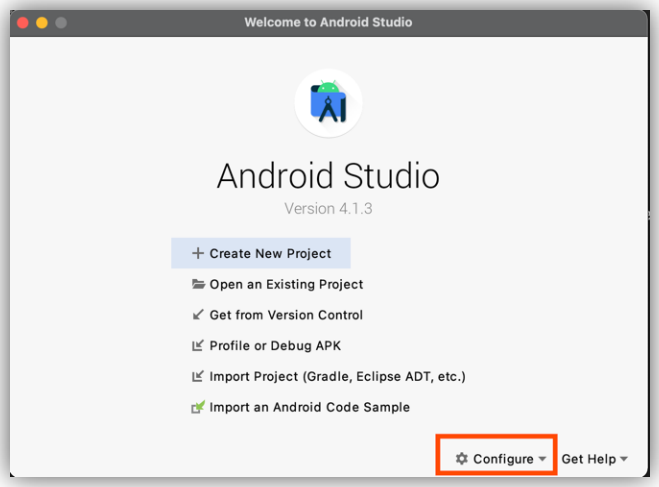


Ilustración 64: Pantalla de bienvenida de Android Studio

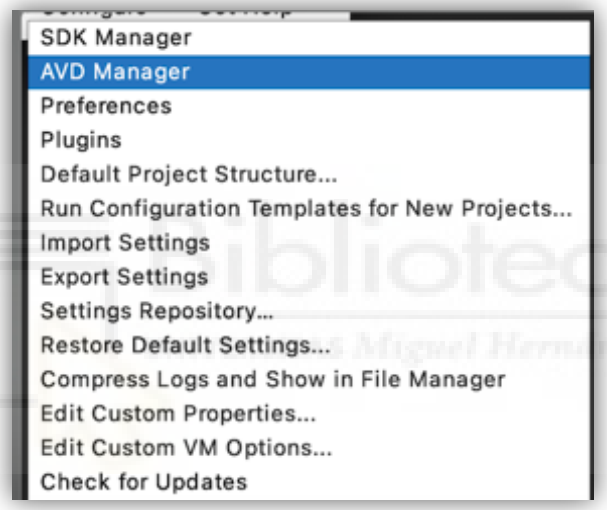


Ilustración 65: AVD Manager de Android Studio

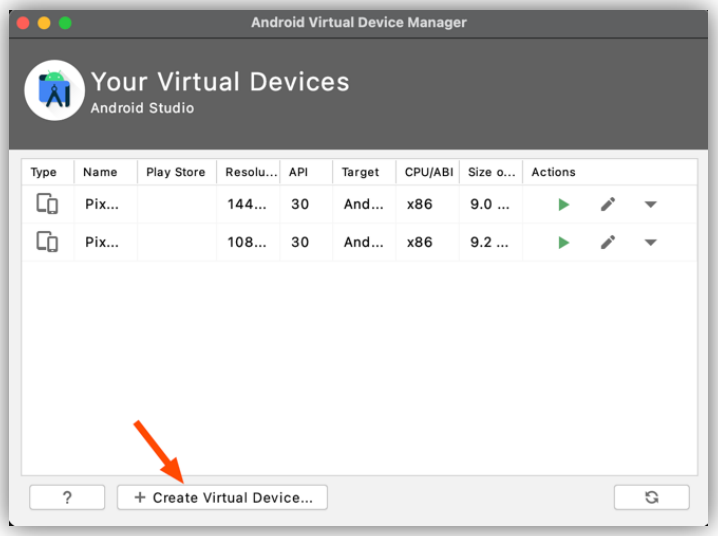


Ilustración 66: Creación de máquina virtual en Android Studio

Paso 15: desde la terminal, se debe aceptar la licencia de Android.

```
flutter doctor --android-licenses
```

Paso 16: validación final, probar que todo esté correcto. Flutter trae consigo la herramienta "`flutter doctor`" que realiza un chequeo de la configuración a modo de comprobar si falta alguna herramienta necesaria para su funcionamiento.

Ilustración 67: Comando para chequear la configuración de Flutter

Una vez validado que todo en Flutter está bien configurado, se debe abrir la paleta de comandos de Visual Studio Code:

Ilustración 68: Paleta de comandos de Visual Studio Code

Seguidamente, se debe escribir "Flutter" y pulsar sobre la opción "new proyect"

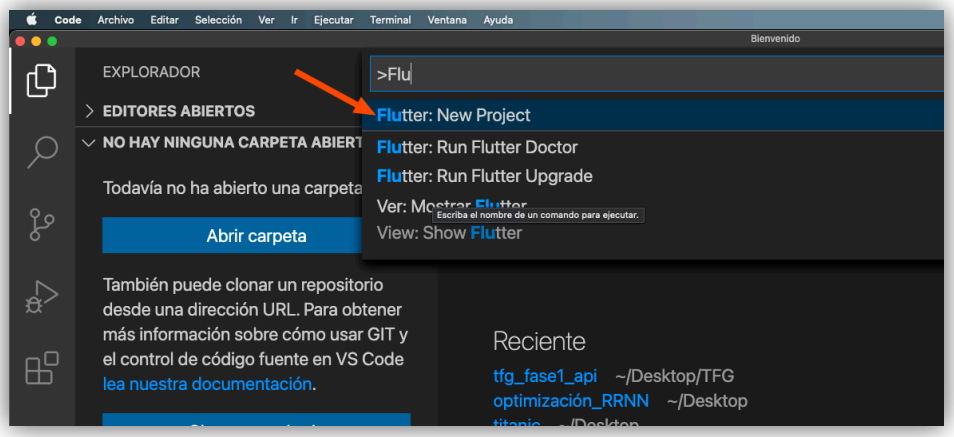


Ilustración 69: Crear nuevo proyecto en Flutter desde VSC

Seguidamente, se establece el nombre del proyecto y su ubicación para guardarlo.

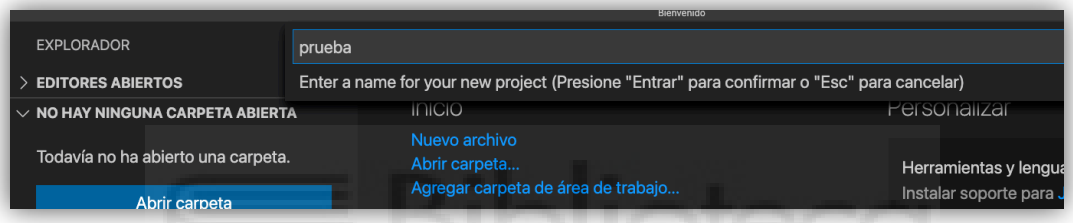


Ilustración 70: Introducción del nombre del proyecto de Flutter en VSC

Una vez finalice el proceso, se tendrá creado el esqueleto de una aplicación básica.

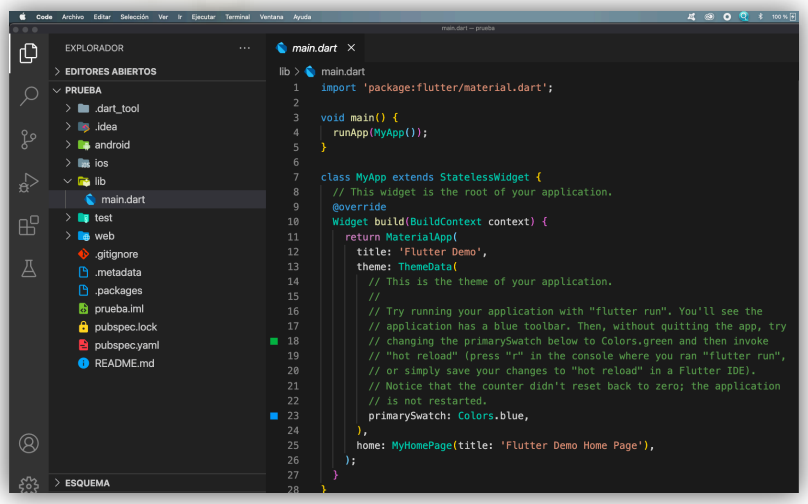


Ilustración 71: Directorios y ficheros de una aplicación básica de Flutter

Para desplegar la aplicación se debe en primer lugar, seleccionar el emulador.

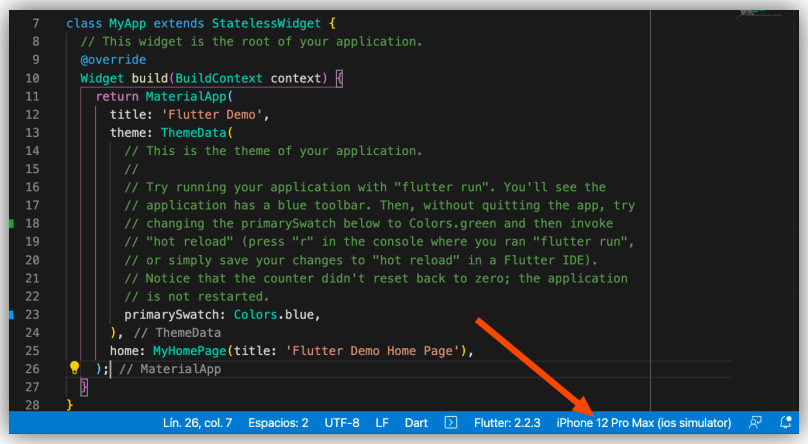


Ilustración 72: Dispositivo seleccionado en VSC

En este caso, se tiene por defecto el de iOS, pero si se pulsa sobre él se abren más opciones para seleccionar.

Ilustración 73: Selección de dispositivos en VSC

Por último, para lanzar la aplicación, simplemente, se debe ir al menú **Ejecutar → Iniciar depuración** (o se pulsa directamente el botón F5).

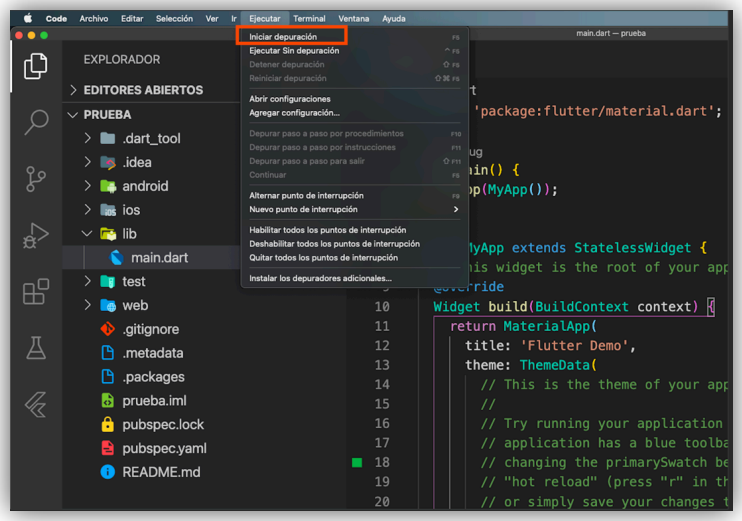


Ilustración 74: Despliegue de una aplicación de Flutter en VSC

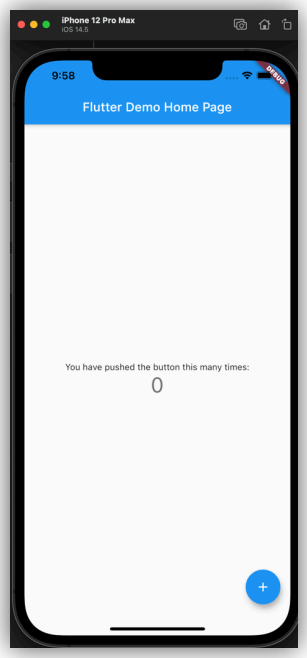


Ilustración 75: App de ejemplo desplegada en emulador

4.4.3. Quicktype.io

Cuando se trabaja con aplicaciones móviles, una de las funcionalidades a dotar es la del consumo de servicios web vía http para la obtención de datos, con el objetivo de consumir esa información. Los Web Service son una solución adecuada para la lograr la interoperabilidad entre aplicaciones.

Cuando, en este caso, el dispositivo móvil obtiene esos datos a través de la API Rest, es necesario realizar un tratamiento con ellos, para ello se debe realizar un mapeo de ellos, creando las correspondientes estructuras de datos. Aquí es donde entra en aparición Quicktype.io

Quicktype.io es una herramienta en línea muy útil que permite mapear la respuesta de una API, seleccionando el tipo de salida (JSON en este caso). Por otro lado, se selecciona el lenguaje (Dart) de esta forma genera automáticamente la clase correspondiente. Esta utilidad ha sido de gran utilidad ya que se ha conseguido ahorrar mucho tiempo en la realización de todos los mapeos de la aplicación. Es una forma de agilizar la construcción de las clases.

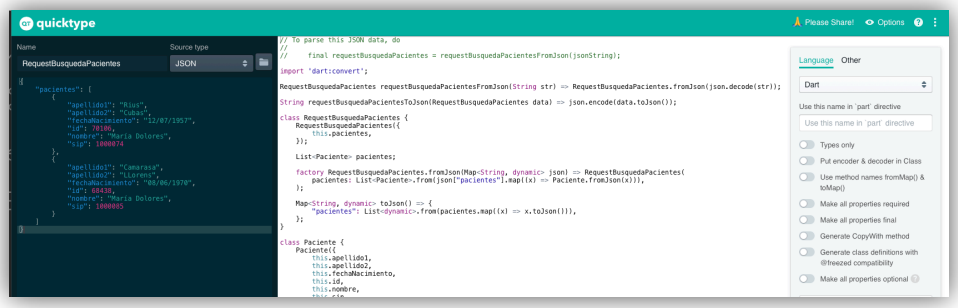


Ilustración 76: Ejemplo de uso de la herramienta Quicktype.io

4.4.4. Adobe Xd

Es un software destinado a crear y compartir interfaces para webs y aplicaciones, con el foco puesto en la experiencia de usuario. También permite compartir prototipos y así poder obtener el feedback sobre el producto real por parte de los usuarios [41].

Esta aplicación ha sido utilizada para la elaboración de la interfaz de la aplicación móvil.

4.5. Características de los equipos utilizados

4.5.1. Equipo local

Tanto el desarrollo de la red neuronal, API y aplicación móvil han sido desarrolladas en local por un iMac de 27” con un monitor extra HP también de 27”.

4.5.1.1. Características técnicas

- **Procesador**: Intel® Core i7 de 4 núcleos a 4,2GHz
- **Memoria RAM**: 8GB 2400 MHz DDR4
- **Tarjeta gráfica**: Radeon PRO-580 8GB
- **Sistema Operativo**: macOS Big Sur 11.2.3

4.5.2. Servidor Base de datos

La base de datos Oracle Express Edition ha sido instalada en un servidor VPS propio, contratado a la empresa IONOS.

4.5.2.1. Instalación Base de datos Oracle

En este apartado se detalla la instalación de la Base de datos Oracle Express Edition en el servidor VPS.

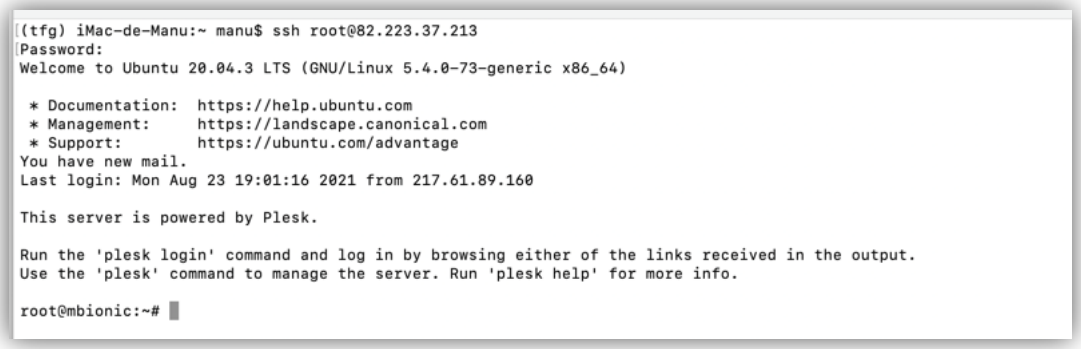


Ilustración 77: Acceso al servidor VPS

Paso 1: descargar desde la web oficial de Oracle, la Base de datos "Oracle DataBase 18c Express Edition".

```
wget https://download.oracle.com/otn-pub/otn_software/db-
    express/oracle-database-xe-18c-1.0-1.x86_64.rpm
```

Como la instalación se va a hacer en un servidor bajo Ubuntu (basada en Debian), es necesario convertir el fichero descargado .rpm (para distribuciones en Red Hat). Pero para eso, se necesita hacer uso de la herramienta *alien*.

Paso 2: instalar la herramienta *alien* junto con las dependencias necesarias.

```
sudo apt install alien libaio1 unixodbc
```

Paso 3: convertir el paquete descargado en el paso 1.

```
sudo alien –scripts -d Oracle-database-xe-18c-1.0-1.x86_64.rmp
```

Paso 4: añadir las siguientes líneas de código al script /sbin/chkconfig

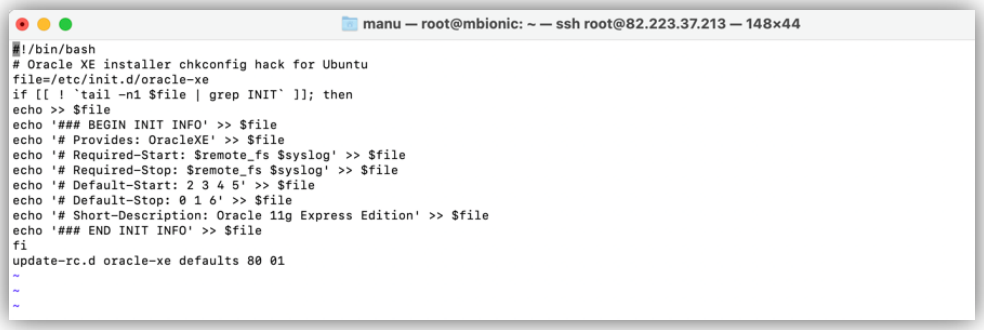


Ilustración 78: Configuración del script /sbin/chkconfig

Paso 5: otorgar permisos necesarios al archivo /sbin/chkconfig

```
sudo chmod 755 /sbin/chkconfig
```

Paso 6: Oracle 18c XE requiere una configuración peculiar en el kernel para poder funcionar. Por ello, es necesario editar el fichero sudo `vi /etc/sysctl.d/60-oracle.confg` y añadir las siguientes líneas:

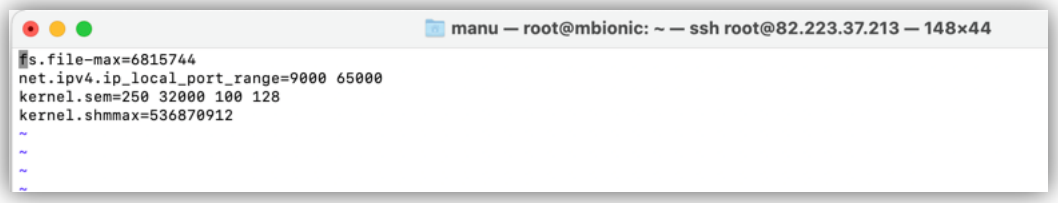


Ilustración 79: Configuración del fichero /etc/sysctl.d/60-oracle.confg

Paso 7: reiniciar los parámetros del kernel.

```
sudo service procps start
```

Paso 8: instalar paquete Oracle 18c.

```
sudo dpkg -i Oracle-database-xe-18c_1.0-2_amd64.deb
```

Paso 9: una vez el proceso del paso 8 finalice (aproximadamente 10 minutos) se debe añadir el siguiente parámetro al fichero `/etc/init.d/oracle-xe-18c`.
Se busca la línea de texto:

```
<span lang="es-ES"><i><b>Doracle.assistants.dbca.validate.DBCredentials=false</b></i></span>
```

Y se le añadimos lo siguiente:

```
<span lang="es-ES"><i><b>-</b></i></span><span lang="es-ES"><i><b>J-</b></i></span><span
lang="es-ES"><i><b>Doracle.assistants.dbca.validate.ConfigurationParams=false</b></i></span>
```

Quedando de la siguiente forma:

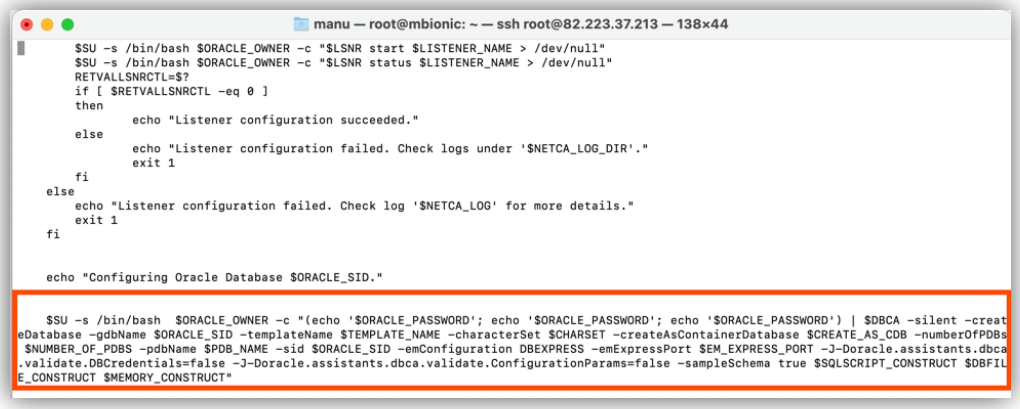


Ilustración 80: Configuración del fichero /etc/init.d/oracle-xe-18c

Paso 10: configurar contraseña del administrador de la base de datos.

```
sudo /etc/init.d/oracle-xe-18c configure
```

Paso 11: añadir las variables de entorno.

```
vi ~/.bashrc
```

se añaden las siguientes líneas al fichero:

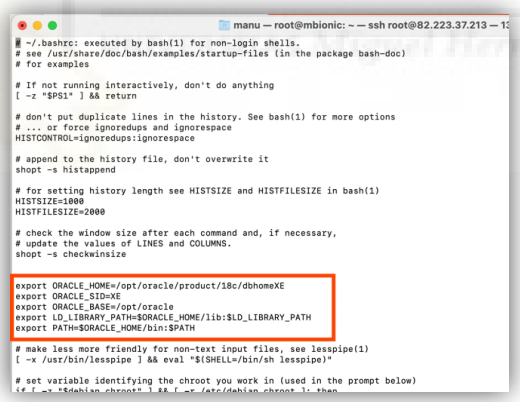


Ilustración 81: Configuración de variables de entorno Oracle

Paso 12: reiniciar el servicio de Oracle 18c.

```
sudo service oracle-xe-18c restart
```

Una vez instalada la base de datos, es necesario configurar los usuarios, para ello se han realizado las siguientes acciones:

Paso 1: validar la configuración.

```
sql /nolog
connect SYSTEM
```

Paso 2: tras el inicio de sesión con el usuario del sistema, es buena práctica tener otro usuario aparte con privilegios de dba.

```
ALTER SESSION SET "_ORACLE_SCRIPT"=true

CREATE USER MANU_DBA IDENTIFIED BY password
    GRANT CONNECT, RESOURCE, DBA TO
```

Paso 3: crear usuario para albergar los objetos de este proyecto.

```
CREATE USER TFG IDENTIFIED BY password
GRANT CONNECT, RESOURCE, DBA TO MANU_DBA;
```

Paso 4: crear tablespace de datos. Un tablespace es una estructura donde se almacenarán los objetos del esquema de base de datos.

```
CREATE TABLESPACE DATOS_TFG DATAFILE 'data_tfg.dbf' SIZE 1g;
```

Paso 5: asignar al usuario cuota de escritura sobre el tablespaces creado en el paso 4.

```
ALTER USER TFG QUOTA UNLIMITED on DATOS_TFG;
```

Paso 6: una vez se tiene todo esto, se procede abrir el sqldeveloper para configurar la conexión con el usuario. Para abrir sqldeveloper en el sistema operativo Big Sur, hay que hacerlo mediante la línea de comando de la siguiente forma:

```
zsh
/Applications/SQLDeveloper.app/Contents/Resoruces/sqldeveloper/s
qldeveloper.sh
```

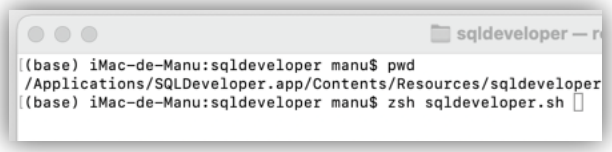


Ilustración 82: Ejecución de sqldeveloper en Big Sur

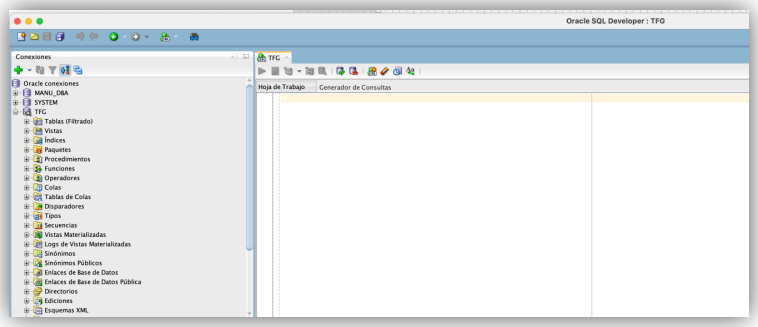


Ilustración 83: Pantalla principal de la herramienta sqldeveloper

4.5.2.2. Características técnicas

- **Tipo**: Servidor Cloud
- **Procesador**: Intel® Xeon® Gold 5120 CPU @ 2.20GHz con 4 núcleos.
- **Memoria RAM**: 8GB
- **Tarjeta gráfica**: N/A
- **Sistema Operativo**: Ubuntu 20.04
- **IP**: 82.223.37.213

Uno de los aspectos más importantes de un servidor, es el sistema de Backups, por ello se ha contratado el servicio de Backups de 50GB, suficientes para las copias periódicas de la Base de datos y configuración del servidor.

CAPÍTULO 5: METODOLOGÍA Y RESULTADOS

En este capítulo se detalla la metodología aplicada, así como los procesos llevados a cabo para el desarrollo de las distintas partes que forman este proyecto.

5.1. Planificación del proyecto

La gestión de proyectos es un conjunto de metodologías destinadas a planificar y dirigir los procesos de un proyecto. Un proyecto comprende una suma específica de operaciones diseñadas para logar un objetivo con un alcance, recursos y fechas de inicio y fin establecidas.

A menudo se cae en el error de pensar que la gestión de proyectos es únicamente relevante para proyectos donde el desarrollo se realiza entre un grupo de personas. Pero, no es así, pues un proyecto unipersonal como es este trabajo de fin de grado también requiere de una gestión y planificación.

5.1.1. Desglose de tareas (WBS)

Las tareas de un proyecto son los “pasos” que se requieren dar para poder conseguir el objetivo. La elaboración de este proyecto se ha desglosado en las siguientes tareas (cada una pertenece a una determinada fase del proyecto), siendo estas a su vez categorizadas en: estudio, infraestructura, desarrollo, análisis, testing y documentación.

Cada tarea tiene asociada una o varias tareas precedentes, que indican una relación entre la finalización de estás y el comienzo de la/s siguiente/s.

Nº	FASE	CATEGORÍA	TAREA	DURACIÓN (h)	PRECEDENTE
1	RRNN	Estudio	Estudio del lenguaje python	22	-
2	RRNN	Estudio	Estudio teoría RRNN	25	-
3	RRNN	Estudio	Estudio librerías: TensorFlow, Keras, Pandas	12,5	1, 2
4	RRNN	Infraestructura	Construcción entorno de trabajo IA	3,5	3
5	RRNN	Estudio	Construcción mi primera RRNN	5,5	4
6	RRNN	Desarrollo	Búsqueda dataset	3	5
7	RRNN	Análisis	Diseño arquitectura scripts python	3	5
8	RRNN	Estudio	Estudio sobre técnicas de análisis y limpieza de datos	4,5	5
9	RRNN	Desarrollo	Limpieza y preparación del dataset	7,5	6, 8
10	RRNN	Desarrollo	Creación RRNN, entrenamiento y optimización	25,5	9
11	RRNN	Desarrollo	Generación de predicciones	1	10
12	RRNN	Desarrollo	Guardado del modelo implementado	0,5	11
13	APP	Infraestructura	Construcción entorno desarrollo app	3	12
14	APP	Estudio	Estudio lenguaje Dart	10,5	12
15	APP	Análisis	Análisis de requisitos	12	12
16	APP	Estudio	Estudio framework Flutter	24	13, 14
17	APP	Diseño	Diseño BBDD	16,5	15
18	APP	Diseño	Diseño de IU	3,5	15, 16
19	API	Desarrollo	Desarrollo API REST	15,5	17
20	APP	Desarrollo	Codificación APP	52	17, 18
21	APP	Testing	Testeo	4	19, 20
22	DOCUMENTACIÓN	Documentación	Generación de la memoria final	66	21

Ilustración 84: Desglose de tareas del proyecto

5.1.2. Métricas

Las métricas son una herramienta muy importante dentro de la gestión de un proyecto. Una métrica es cualquier tipo de variable que pueda ser utilizada para medir el desempeño de algún aspecto del proyecto que sea importante y se quiera controlar.

Este proyecto tenía principalmente dos restricciones:

1. Disponibilidad horaria debido a la compaginación de estudios y trabajo.
2. Tiempo límite para la presentación del proyecto.

Para poder realizar una evaluación y seguimiento a estas dos restricciones, se creó un fichero excel con 4 hojas:

- **Incurridos**: calendario donde se indica el tiempo (horas) dedicadas a cada tarea.
- **Listado tareas**: mediante una tabla dinámica, recoge el tiempo total dedicado a cada una de las tareas que forman el proyecto.
- **Informe duración tareas:** leyendo de la tabla dinámica comentada en el punto anterior, genera un gráfico con la duración de cada tarea.
- **Informe duración por categoría y fase:** genera dos gráficos con los porcentajes que se han dedicado a cada categoría y fase.

La elaboración de estos informes se realizó con el objetivo de obtener, por un lado, una visión global de la evolución del proyecto en cuanto a tiempos se refiere. Y por otro, el tiempo dedicado tanto a nivel de tarea, categoría y fase. Esto último es muy útil a modo de referencia para futuros trabajos, pues el tener estos datos hace más sencillo poder realizar una estimación del tiempo que pueda llevar el realizar un nuevo proyecto de este tipo.

A continuación, se muestran los gráficos de dichos informes.

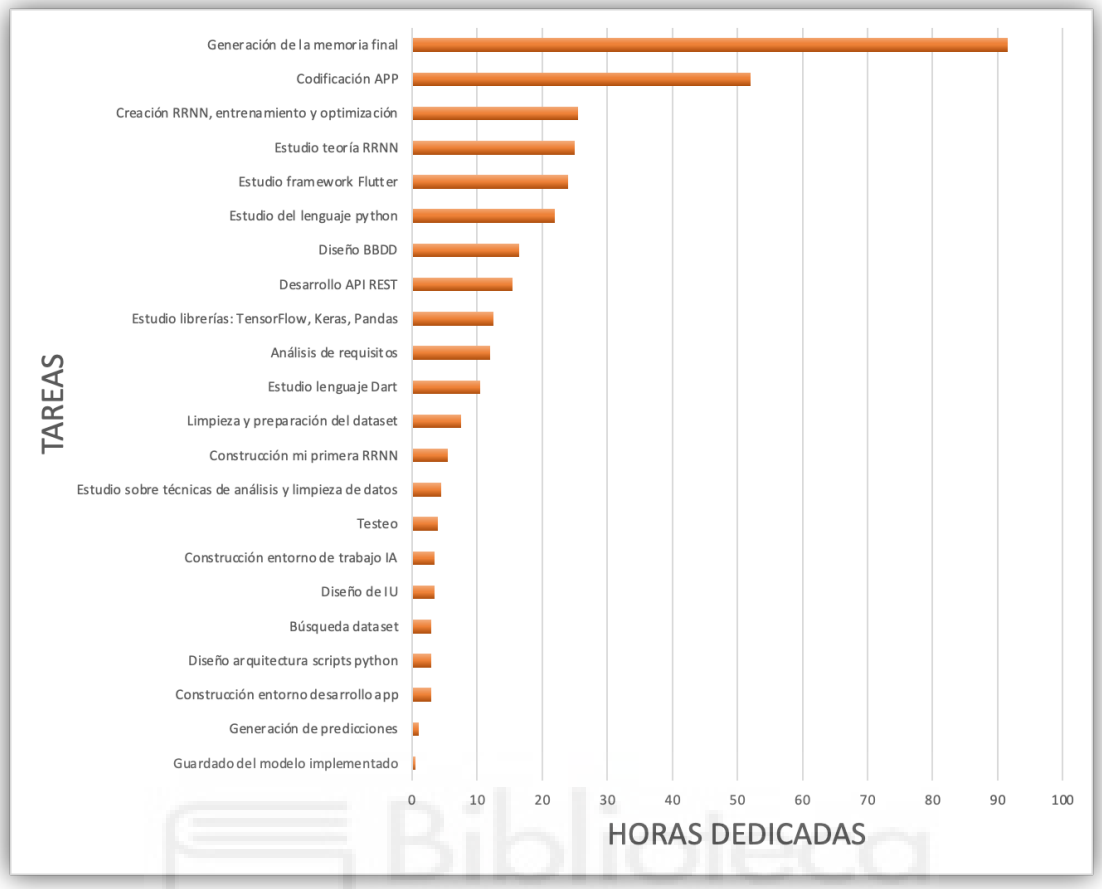


Ilustración 85: Tiempo dedicado a las tareas del proyecto

Ilustración 86: Tiempo dedicado por categoría

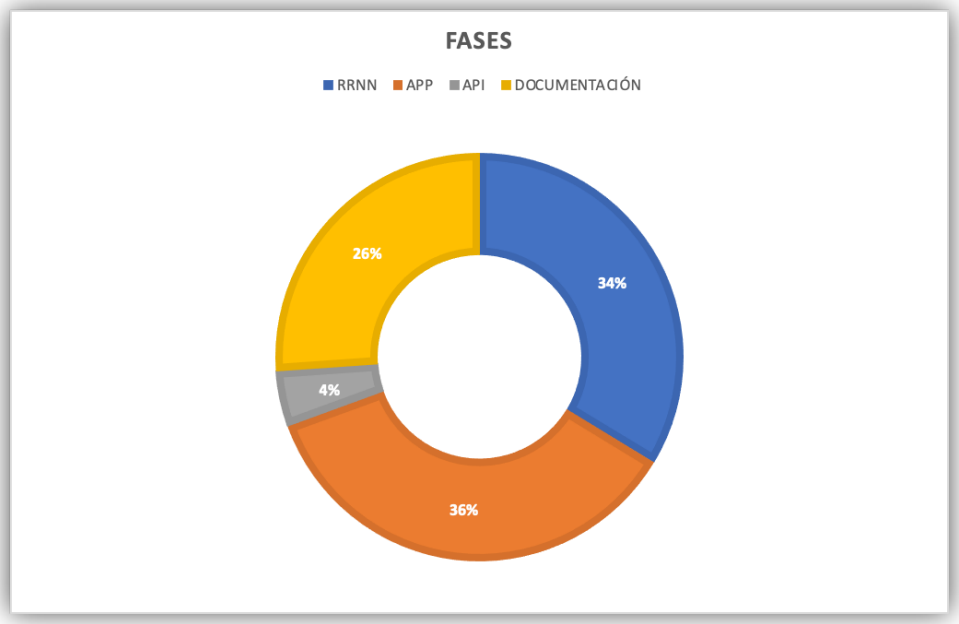


Ilustración 87: Tiempo dedicado por fase

5.1.3. Diagrama de Gantt

En base al desglose de tareas presentadas en el apartado 5.1.1. Desglose de tareas, junto con la imputación de las horas dedicadas a cada una de ellas, se muestra a continuación el diagrama de Gantt del proyecto.

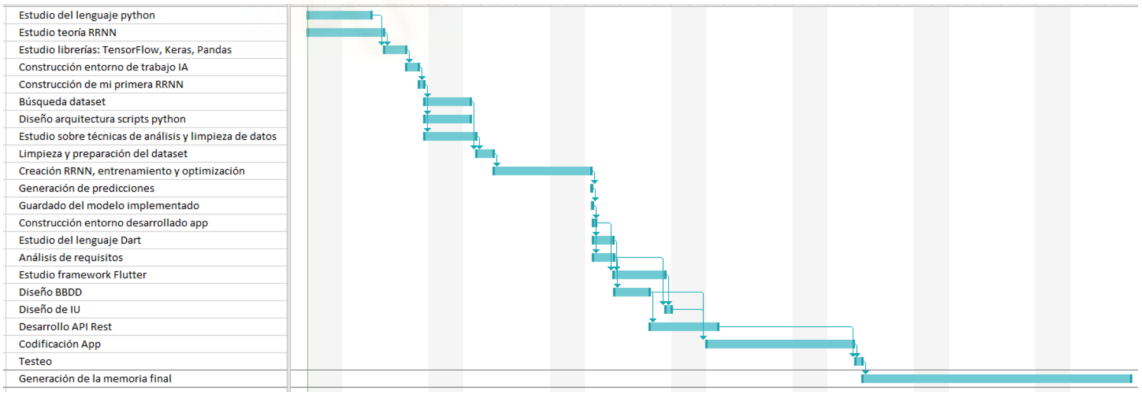


Ilustración 88: Diagrama de Gantt del proyecto

5.2. Desarrollos

5.2.1. Red Neuronal

El principal objetivo de la construcción de la red neuronal que se expone a continuación es la elaboración de un modelo que permita realizar predicciones, en términos de probabilidad, sobre si determinados pacientes desarrollarán un infarto cerebrovascular.

Para ello, se hará uso de una red neuronal multicapa cuyo objetivo sea aplicar un clasificador binario.

5.2.1.1. Dataset utilizado

Para la elaboración del modelo, se ha utilizado un dataset descargado desde la web oficial de Kaggle. Kaggle es una comunidad online (subsidiaria de Google) de científicos de datos y profesionales del aprendizaje automático.

El dataset empleado se encuentra documentado y se puede descargar gratuitamente desde la siguiente URL:

https://www.kaggle.com/fedesoriano/stroke-prediction-dataset

5.2.1.1.1. Características del dataset

El dataset consta de 4.996 registros conformados con las siguientes 12 características:

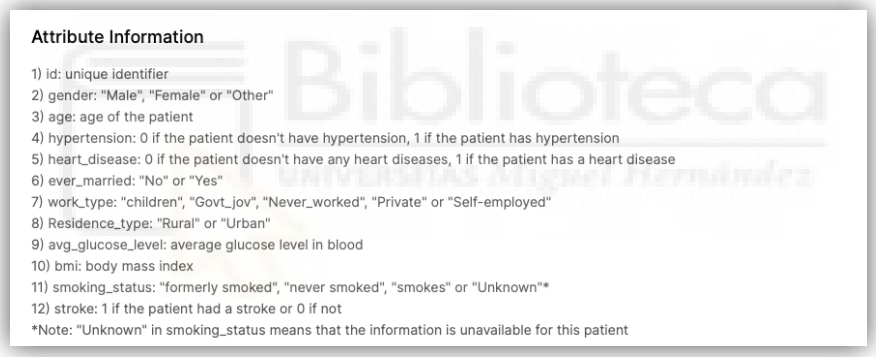


Ilustración 89: Características del dataset utilizado

5.2.1.2. Estructura y finalidad de los scripts desarrollados

En la construcción de un modelo, existen dos partes claramente diferenciadas en el proceso: la limpieza y preparación de los datos y la creación y entrenamiento en sí de la red neuronal. Para cubrir este cometido se generaron dos *scripts*, separando el desarrollo en dos partes:

1. Limpieza y preparación del dataset (script **limpieza_IC.ipynb**), en este script se realiza:
   1.1. Traducción del dataset.
   1.2. Estudio de la distribución de la variable objetivo.
   1.3. Tratamiento de registros problemáticos.
   1.4. Preparación del conjunto de datos

2. Creación y entrenamiento del modelo (script **modelo_infarto_cerebral.ipynb**):
    2.1. Estudio de las características
    2.2. Construcción de la red neuronal.
    2.3. Entrenamiento de la red neuronal.
    2.4. Generación de predicciones.
    2.5. Reutilización del modelo.

En los siguientes apartados se explica en detalle los pasos llevados a cabo en cada uno de los *scripts* desarrollados.

5.2.1.3. Limpieza y preparación del dataset

A la hora de desarrollar un modelo, el desempeño que este tenga dependerá en gran medida de los datos con los que se haya entrenado. Una de las tareas más costosas en tiempo y esfuerzo es sin duda la limpieza y preparación de los datos.

En este apartado se exponen los pasos realizados para la preparación y adecuación del dataset.

Paso 1: traducción del dataset.

Partiendo del dataset comentado en el apartado 5.1.1.1 Dataset utilizado se observa que este se encuentra en inglés. El primer paso llevado a cabo fue realizar la traducción de este (tanto de las columnas como de los datos), a modo de una mejor comprensión.

```
#Visualizo el dataset pra verificar que se ha cargado correctamente
dataset.head()
```

	id	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	stroke
0	9046	Male	67.0	0	1	Yes	Private	Urban	228.69	36.6	formerly smoked	1
1	51676	Female	61.0	0	0	Yes	Self-employed	Rural	202.21	NaN	never smoked	1
2	31112	Male	80.0	0	1	Yes	Private	Rural	105.92	32.5	never smoked	1
3	60182	Female	49.0	0	0	Yes	Private	Urban	171.23	34.4	smokes	1
4	1665	Female	79.0	1	0	Yes	Self-employed	Rural	174.12	24.0	never smoked	1

Ilustración 90: Visualización del dataset original en inglés

```
#Visualizo el dataset pra verificar que se ha cargado correctamente
dataset.head()
```

	id	sexo	edad	hipertension	cardiopatia	alguna_vez_casado	tipo_de_trabajo	tipo_de_residencia	nivel_medio_de_glucosa	indice_masa_co
0	9046	Hombre	67	0	1	Si	Privado	Urbano	228.69	
1	51676	Mujer	61	0	0	Si	Cuenta propia	Rural	202.21	
2	31112	Hombre	80	0	1	Si	Privado	Rural	105.92	
3	60182	Mujer	49	0	0	Si	Privado	Urbano	171.23	
4	1665	Mujer	79	1	0	Si	Cuenta propia	Rural	174.12	

Ilustración 91: Visualización del dataset traducido

Paso 2: visualizar la distribución de la variable objetivo.

Una de las primeras acciones que se debe realizar es estudiar la distribución de la variable objetivo, en este caso ***riesgo_ataque***.

Nada más cargar el dataset se aprecia que existe un desbalanceo muy importante de los datos, pues tal y como se puede ver en las ilustraciones 92 y 93, existen pacientes donde el 95% pertenecen a un grupo mayoritario (riesgo_ataque = 0) y apenas el 5% al grupo minoritario (riesgo_ataque = 1):

```
dataset['riesgo_ataque'].hist()
```

```
<AxesSubplot:>
```

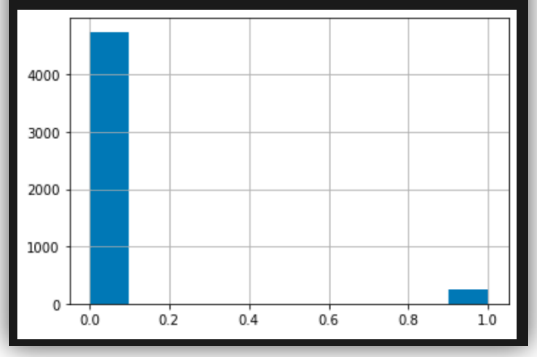


Ilustración 92: Gráfico de barras con la distribución de la variable objetivo

```
dataset['riesgo_ataque'].value_counts()

0    4745
1     249
```

Ilustración 93: Recuento de registros por valores de la variable objetivo

¿Qué impacto tienen unos datos desbalanceados?

Este desbalanceo de los datos presenta un gran problema, pues provocará que el modelo en la fase de entrenamiento tenga un desempeño aparentemente muy bueno, pero en el momento de asignarle nuevos datos este no se comporte de forma adecuada y entre en un estado de overfitting.

En líneas generales, dado que en este caso la mayoría de los datos son de pacientes "negativos" esto generará el problema de que cuando el modelo intente realizar una nueva generalización, este perjudicará a la clase minoritaria (pacientes positivos). Esto es debido a que el modelo se habrá entrenado prácticamente en su totalidad con datos donde la variable objetivo es 0.

Corrección del desbalanceo de datos

Ante un problema de desbalanceo de datos existen principalmente dos técnicas para abordar su corrección: Submuestreo y sobremuestreo.

Submuestreo

Ilustración 94: Distribución de datos en el submuestreo aleatorio

Consiste en eliminar de forma aleatoria registros de la clase mayoritaria y asignarla a la clase minoritaria, sin necesidad de rellenar el vacío creado en la clase mayoritaria. Este proceso presenta unas ventajas e inconvenientes [42]:

Ventajas

- Cuando el dataset es muy grande, ayuda a mejorar el tiempo de ejecución del modelo y resolver los problemas de memoria al reducir el número de muestra de los datos de entrenamiento.

Inconvenientes

- El eliminar de forma aleatoria registros del dataset hace que se pueda descartar información que puede ser útil para el modelo.

Sobre-muestreo

Ilustración 95: Distribución de datos en el sobremuestreo

Consiste en aumentar las instancias de la clase minoritaria replicándolas hasta un grado consistente, sin disminuir el número de instancias asignadas a la clase mayoritaria [42].

Esta técnica también presenta unas ventajas e inconvenientes:

Ventajas

- No conduce a la pérdida de información.

Inconvenientes

- Aumenta la probabilidad de un exceso de equipamiento ya que replica los eventos de la clase minoritaria.

Técnica aplicada al modelo en cuestión

Dada la alta cantidad de registros mayoritarios que hay, se optó por aplicar un **submuestreo**, eliminando instancias de forma aleatoria.

Para ello se realizaron las siguientes acciones:

1. Se generaron dos dataset separados por el valor de la variable objetivo.

```
condicion_no = dataset['riesgo_ataque']==0
condicion_si = dataset['riesgo_ataque']==1
dataset_no = dataset[condicion_no]
dataset_si = dataset[condicion_si]
```

Ilustración 96: Separación del dataset por variable objetivo

2. En el dataset con las instancias mayoritarias y haciendo uso de la función **sample** de la librería de **pandas**, se obtuvo una muestra aleatoria de 350 registros.

```
dataset_no = dataset_no.sample(350)
```

Ilustración 97: Generación de dataset con muestras aleatorias

3. Fusión de ambos dataset.

```
#Junto los dos dataset generados en uno solo
dataset = pd.concat([dataset_si,dataset_no])
```

Ilustración 98: Fusión instancias mayoritarias y minoritarias

Consiguiendo de esta forma que los datos ahora sí estén balanceados, con un 58% clase mayoritaria y 42% clase minoritaria.

```
dataset['riesgo_ataque'].value_counts()

0    350
1    249
```

Ilustración 99: Balanceo de datos en el dataset

4. Permutación de los registros.

Al concatenar ambos dataset ha provocado que los datos de ambas instancias estén correlativos:

al	estado_fumador	riesgo_ataque
.6	fumaba	1
aN	Nunca	1
.5	Nunca	1
.4	fuma	1
.0	Nunca	1

Ilustración 100: Datos correlativos según la variable objetivo tras la concatenación

Por lo que, en este punto, se realizó una permutación para mezclarlos de forma aleatoria.

```
nrows = dataset.shape[0]
p = np.random.permutation(nrows)
dataset = dataset.take(p)
```

Ilustración 101: Permutación de los registros del dataset

indice_masa_corporal	estado_fumador	riesgo_ataque
18.2	Desconocido	0
49.3	Nunca	0
NaN	Desconocido	1
29.7	fuma	0
36.5	fumaba	1
27.8	Desconocido	0
24.2	Nunca	0
27.0	fumaba	1
26.5	Nunca	0
37.8	Nunca	0

Ilustración 102: Visualización de instancias del dataset preparado

Paso 3: tratamiento de registros que puedan dar problemas en el entrenamiento de la red.

El siguiente paso fue observar por cada una de las características que forman el dataset, la existencia de registros duplicados (no hubo ninguno) y/o nulos.

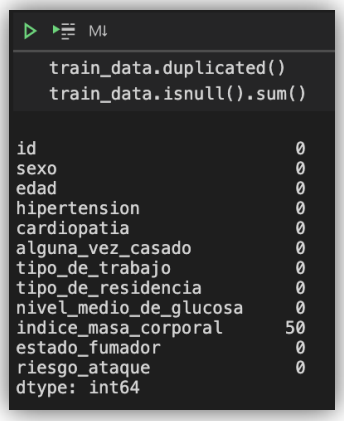


Ilustración 103: Conteo de registros nulos en el dataset

Se aprecia que únicamente existen campos nulos (50) en la característica *índice_masa_corporal*. El principal motivo de la existencia de datos nulos en un dataset es una incorrecta implementación en el sistema de recogida de la información. Para que, en el entrenamiento, el modelo no presente problemas, es necesario imputar estos registros.

Existen diferentes técnicas para imputar registros nulos, como, por ejemplo: tomar el valor más repetido, establecer la media, eliminar el registro. Para este caso se optó por imputar los campos nulos con el valor de la media. Para ello se realizaron las siguientes acciones:

1. Generar un dataset sin los registros nulos.

```
#Genero un dataset sin los nulos
filtro_sin_nulos = dataset[ dataset['indice_masa_corporal'].isnull()].index
dataset_sin_nulos = dataset.drop(filtro_sin_nulos)
```

Ilustración 104: Generación de dataset sin registros nulos

2. Calcular la media.

```
media = round(dataset_sin_nulos.indice_masa_corporal.mean())
```

Ilustración 105: Cálculo de la media de la característica índice_masa_corporal

3. Imputar los valores nulos con la media calculada.

```
dataset['indice_masa_corporal'].fillna(media, inplace=True)
```

Ilustración 106: Imputación de los registros nulos con la media

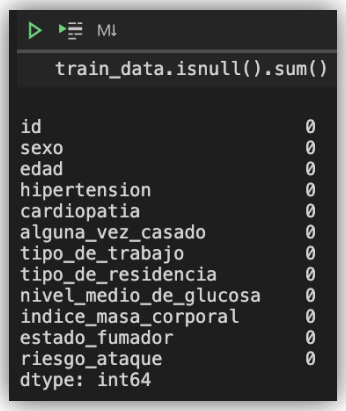


Ilustración 107: Confirmación de la no existencia de registros nulos en el dataset

Paso 4: preparación del conjunto de datos.

Una vez se realizaron los anteriores pasos comentados y teniendo el dataset preparado, se generaron 2 dataset, uno para el entrenamiento y otro para la validación del modelo.

Para este cometido, la librería **sklearn** dispone del método **train_test_split** que permite realizarlo de una forma muy sencilla y rápida.

La división que se realizó fue del 80% de datos para el entrenamiento y el 20% para la validación.

```
#Genero los dataset de entrenamiento (train.csv) y testeo (test.csv)
train, test = train_test_split(dataset, test_size = 0.20)

train.to_csv('train.csv',index=False) #Con el index=False se eliminan los índices
test.to_csv('test.csv',index=False) #Con el index=False se eliminan los índices
```

Ilustración 108: Creación de los dataset de entrenamiento y validación

5.2.1.4. Creación y entrenamiento de la red neuronal

5.2.1.4.1. Estudio de las características del dataset

Antes de la creación de la red neuronal se realizó un estudio de las características del dataset que pudieran ser útiles, es decir, aquellas que pudieran generar cierta potencia predictiva. Estas características serán las encargadas de alimentar a la red neuronal.

Para ello, se hizo uso de una herramienta muy útil y ampliamente utilizada en el análisis de datos, la **matriz de correlación**.

Dicha matriz, representa las correlaciones entre pares de variables en un dato dado. Donde cada fila y columna simboliza una variable, y cada valor es el **coeficiente de correlación** entre las variables representadas por la fila y columna correspondiente [43].

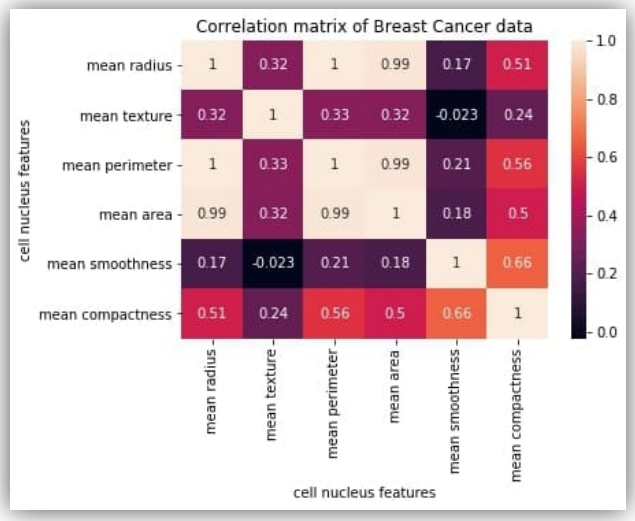


Ilustración 109: Ejemplo de matriz de correlación

Coeficiente de correlación

Es un número que oscila en el rango de [-1,1] e indica la fuerza de la relación entre dos variables. Los valores cercanos a 1 indican la presencia de una fuerte relación positiva entre X e Y. Por otro lado, los cercanos a -1 indican una fuerte relación negativa entre X e Y. En cambio, los valores cercanos a cero indican que no existe ninguna relación entre X e Y [43].

Existen varios tipos de coeficientes de correlación, siendo el más común el de Pearson, que es definido como la varianza entre dos variables dividida por el producto de la desviación estándar de las variables dadas.

$$\rho(X,Y) = \frac{COV(X,Y)}{\theta_X \theta_Y}$$

Ecuación 21: Coeficiente de correlación de Pearson

$$COV\ (X,Y) = \frac{E[(X - \mu_X)\ (Y - \mu_Y)]}{\theta_X \theta_Y}$$

Ecuación 22: Covarianza

La librería **pandas** desde el propio dataframe (tipo de dato que representa el dataset en python), ofrece el método **corr()** que permite calcular de una forma muy rápida y sencilla la matriz de correlación. Pero, para poder aplicar dicho método, es necesario que todos los datos sean numéricos.

Estudio del tipo de dato de las características

Revisando el dataset se observa que existen tanto datos numéricos, como categóricos:

Característica	Tipo
id	numérico
Sexo	Categórico
Edad	Numérico
Hipertensión	Numérico binario
Cardiopatía	Numérico binario
Alguna_vez_casado	Categórico
Tipo_de_trabajo	Categórico
Tipo_de_residencia	Categórico
Nivel_medio_glucosa	Numérico
Estado_fumador	Categórico

Tabla 1: Tipo de dato de las características

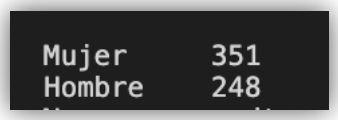


Ilustración 110: Valores categóricos de la característica sexo

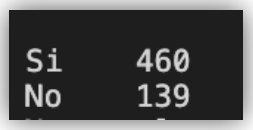


Ilustración 111: Valores categóricos de la característica alguna_vez_casado

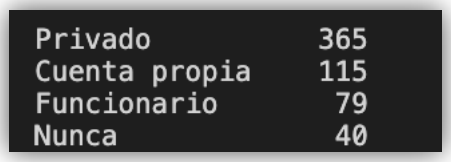


Ilustración 112: Valores categóricos de la característica tipo_de_trabajo

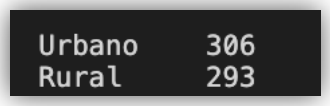


Ilustración 113: Valores categóricos de la característica tipo_de_residencia

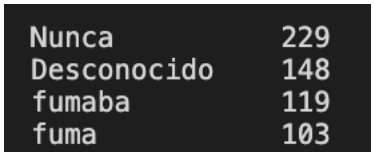


Ilustración 114: Valores categóricos de la característica estado_fumador

A la hora de convertir variables categóricas a numéricas se puede realizar de distintas formas:

- Convertir la variable a binaria, por ejemplo, en la característica sexo, cuyos valores pueden ser dos.
- Dumificar la variable. La librería **pandas** dispone del método **ged_dummies** para tal fin. Este método genera de forma automática una lista de números, cada uno correspondiente a una categoría particular.
- Asignando a cada posible categoría un valor numérico.

A continuación, se presenta la conversión llevada a cabo por cada una de las variables categóricas. Donde, para las características con dos valores posibles como son ***sexo*** y ***alguna_vez_casado*** se convirtieron a una variable binaria. El resto de las características categóricas se dumificaron.

```
dataset['sexo'].replace('Hombre',1,inplace=True)
dataset['sexo'].replace('Mujer',0,inplace=True)
dataset['sexo'].value_counts()
# alguna_vez_casado --> Convierto a 0 y uno (Si = 1, No = 0)
dataset['alguna_vez_casado'].replace('Si',1,inplace=True)
dataset['alguna_vez_casado'].replace('No',0,inplace=True)
dataset['alguna_vez_casado'].value_counts()
#tipo_de_trabajo
dataset = pd.concat([dataset, pd.get_dummies(dataset['tipo_de_trabajo'],prefix='tipo_de_trabajo')],axis=1)
dataset.drop(['tipo_de_trabajo'],axis=1, inplace=True)
#tipo_de_residencia
dataset = pd.concat([dataset, pd.get_dummies(dataset['tipo_de_residencia'],prefix='tipo_de_residencia')],axis=1)
train_features.drop(['tipo_de_residencia'],axis=1, inplace=True)
#estado_fumador
dataset = pd.concat([dataset, pd.get_dummies(dataset['estado_fumador'],prefix='estado_fumador')],axis=1)
dataset.drop(['estado_fumador'],axis=1, inplace=True)
```

Ilustración 115: Conversión de las variables categóricas a numéricas en el dataset

Aplicando la matriz de correlación al dataset

Una vez que todas las características son numéricas, se puede calcular la matriz de correlación.

```
corr_df = dataset.corr()
corr_df.style.background_gradient(cmap='coolwarm')
```

	sexo	edad	hipertension	cardiopatia	alguna_vez_casado	nivel_medio_de_glucosa	riesgo_ataque	tipo_de_trabajo_Cuenta propia	tipo_de_trabajo_Funci
sexo	1.000000	0.057112	-0.009366	0.141582	0.073601	0.119851	0.052152	-0.062259	-0.
edad	0.057112	1.000000	0.285346	0.280065	0.574830	0.283928	0.542299	0.291432	0.
hipertension	-0.009366	0.285346	1.000000	0.061913	0.093616	0.172973	0.247019	0.170978	0.
cardiopatia	0.141582	0.280065	0.061913	1.000000	0.093368	0.255815	0.259166	0.009695	-0.
alguna_vez_casado	0.073601	0.574830	0.093616	0.093368	1.000000	0.195714	0.231581	0.159786	0.
nivel_medio_de_glucosa	0.119851	0.283928	0.172973	0.255815	0.195714	1.000000	0.282338	-0.001767	0.
riesgo_ataque	0.052152	0.542299	0.247019	0.259166	0.231581	0.282338	1.000000	0.142732	-0.
tipo_de_trabajo_Cuenta propia	-0.062259	0.291432	0.170978	0.009695	0.159786	-0.001767	0.142732	1.000000	-0.
tipo_de_trabajo_Funcionario	-0.039776	0.095166	0.043436	-0.016665	0.093353	0.020359	-0.032120	-0.194832	1.
tipo_de_trabajo_Nunca	0.004854	-0.557086	-0.120885	-0.094924	-0.501161	-0.099051	-0.198904	-0.136184	-0.
tipo_de_trabajo_Privado	0.075779	-0.015779	-0.107144	0.052632	0.064105	0.038593	0.008356	-0.612106	-0.
tipo_de_residencia_Rural	0.081513	0.009868	0.030808	-0.015924	0.057724	0.018900	-0.057837	0.046992	-0.
tipo_de_residencia_Urbano	-0.081513	-0.009868	-0.030808	0.015924	-0.057724	-0.018900	0.057837	-0.046992	0.
estado_fumador_Desconocido	0.034918	-0.301477	-0.190415	-0.045328	-0.287181	-0.096719	-0.119200	-0.118877	-0.
estado_fumador_Nunca	-0.124038	0.097453	0.117743	-0.092034	0.022470	-0.004314	-0.020314	0.072328	0.
estado_fumador_fuma	0.084146	-0.005772	0.034349	0.106201	0.103767	0.035243	-0.011193	-0.022524	0.
estado_fumador_fumaba	0.032985	0.212842	0.030452	0.059987	0.184664	0.076304	0.163853	0.062010	0.

Ilustración 116: Matriz de correlación del dataset

Observando la matriz, se puede apreciar que las características que más relación parecen tener con la variable objetivo (riesgo_ataque) son:

- edad
- hipertensión
- cardiopatía
- nivel_medio_glucosa
- alguna_vez_casado

Dado este resultado, se tuvo una idea de que características descartar de cara a las primeras pruebas de entrenamiento.

En los siguientes apartados se exponen los diferentes experimentos llevados a cabo en la creación y entrenamiento de la red neuronal en base a este análisis previo de las características. Pero, antes es conveniente definir una serie de términos que son utilizados en el entrenamiento de una red neuronal (y, por lo tanto, aparecen en los experimentos realizados), que hasta ahora no han sido definidos. A estos dos términos se les conocen como **hiperparámetros** de una red neuronal.

- **Batch_size**: número de datos que tiene cada iteración de un ciclo (epoch). Es útil dado que la red neuronal hace que actualice los parámetros más veces. Cuando se tienen grandes cantidades de datos es necesario el uso de computadores con más recursos (memoria) [44].

- **Epoch**: define el número de veces que se ejecutará el algoritmo de backpropagation. En cada ciclo (epoch) todos los datos de entrenamiento pasan por la red neuronal con el objetivo de aprender de ellos. Por ejemplo, si existen 10 ciclos y 1000 registros de datos, en cada ciclo los 1000 registros pasarán por la red neuronal. Si se especifica el batch_size cada ciclo (epoch) tendrá más ejecuciones internas, donde si tenemos por ejemplo un batch_size de 100, se tendrán 10 iteraciones para completar un ciclo [44].

5.2.1.4.2. Diseño de la arquitectura de la red neuronal

Existen algunas pautas para tener en cuenta de cara al diseño de una red neuronal [45].

- **Capa de entrada**: el número de neuronas vendrá definido por las características que se utilicen.
- **Problemas canónicos**
  - **Regresión**
    - **Head** = Dense (1)
    - **Optimizador** = Adam
    - **Función de pérdida** = mean_squered_error
  - **Clasificación binaria**
    - **Head** = Dense (1) – sigmoid
    - **Optimizador** = Adam
    - **Función de pérdida** = binary_crossentropy
  - **Clasificación multi-clase**
    - **Head** = Dense (#clases) softmax
    - **Optimizador** = Adam
    - **Función de pérdida** = categorical_crosentropy

Las pautas comentadas, sirven como punto de referencia en la creación de las capas de entrada y salida. Pero, para el diseño de las capas ocultas no existe una ciencia exacta. En el sentido de que, si el problema a resolver es de una forma, la arquitectura deberá tener una determinada estructura. En la gran mayoría de casos, el diseño de la arquitectura de la red es un proceso altamente empírico de prueba y error donde básicamente se realizan modificaciones del tipo:

- Se añaden y/o eliminan capas.
- Se cambia el número de neuronas.

- Se añaden dropout para evitar el overfitting.
- Se cambia la función de activación.
- Se modifica el *learning rate*.

Haciendo combinaciones de todas estas posibilidades se va obteniendo un cierto conocimiento de cómo los datos se van comportando.

En el siguiente apartado, se muestra el entrenamiento de la red neuronal, donde se realizan diferentes experimentos con distintas arquitecturas de red, con el fin de ir mejorando las predicciones.

5.2.1.4.3. Entrenamiento de la red neuronal

Para cada experimento, se mostrará el detalle de la arquitectura de la red neuronal utilizada, así como del valor de los diferentes hiperparámetros de la misma.

5.2.1.4.3.1. Entrenamiento 1

Características
Edad
Hipertensión
Cardiopatía
Alguna_vez_casado
Nivel_medio_de_glucosa

Ilustración 117: Características usadas en el entrenamiento 1

Optimizador	Adam
Capas ocultas	1 capa oculta con 9 neuronas
Función de activación capa oculta	Relu
Grado del dropout	0,10
Learning_rate	0,001
Batch size	15
Epochs	500
Validation_split	0,30
Función de activación capa salida	sigmoid

Ilustración 118: Arquitectura utilizada para el entrenamiento 1

Resultados obtenidos

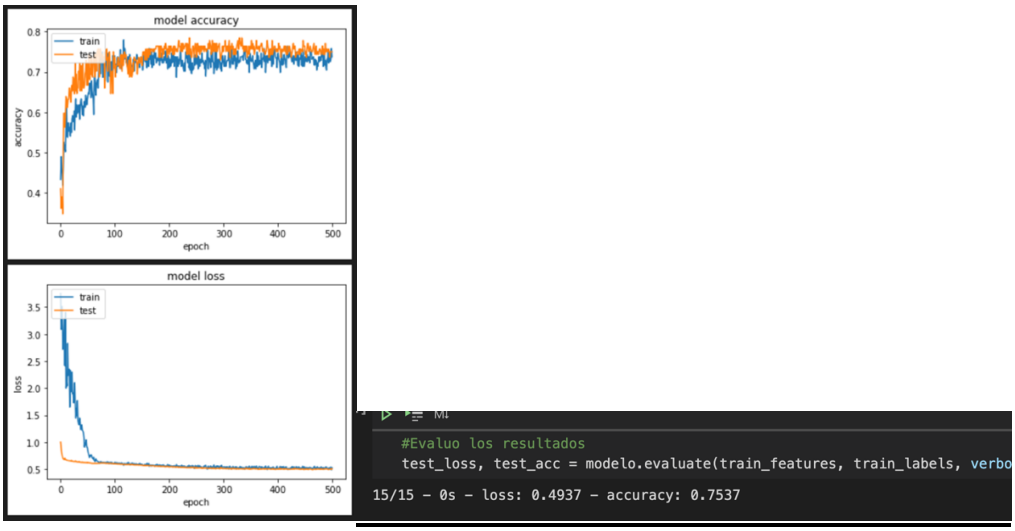


Ilustración 119: Evaluación de los resultados del entrenamiento 1

5.2.1.4.3.2. Entrenamiento 2

Para este segundo entrenamiento se han añadido más características al modelo y modificado el número de neuronas en la capa oculta, el *batch_size*, *epochs* y *validation_split*.

Características
Edad
Hipertensión
Cardiopatía
Alguna_vez_casado
Nivel_medio_de_glucosa
Tipo_de_trabajo
Estado_fumador

Tabla 2: Características usadas en el entrenamiento 2

Optimizador	Adam
Capas ocultas	1 capa oculta con 9 neuronas
Función de activación capa oculta	Relu
Grado del dropout	0,10
Learning_rate	0,001
Batch size	10
Epochs	300
Validation_split	0,20
Función de activación capa salida	sigmoid

Tabla 3: Arquitectura utilizada para el entrenamiento 2

Resultados obtenidos

Se observa que añadiendo las características de *edad* y *estado_fumador* los resultados mejoran ligeramente.

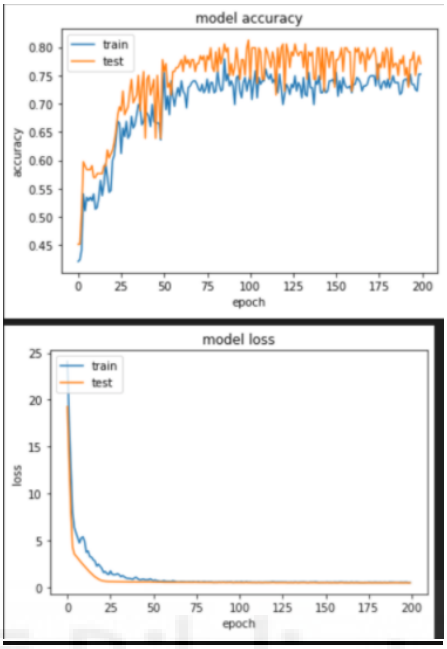


Ilustración 120: Evaluación de los resultados del entrenamiento 2

5.2.1.4.3.3. Entrenamiento 3

En este tercer entrenamiento de la red, se mantienen las características utilizadas en el entrenamiento 2 y se modifican los hiperparámetros *learning_rate*, *epochs* y *validation_split*. Adicionalmente, se reduce el número de neuronas a 9.

Características
Edad
Hipertensión
Cardiopatía
Alguna_vez_casado
Nivel_medio_de_glucosa
Tipo_de_trabajo
Estado_fumador

Tabla 4: Características usadas en el entrenamiento 3

Optimizador	Adam
Capas ocultas	1 capa oculta con 9 neuronas
Función de activación capa oculta	Relu
Grado del dropout	0,10
Learning_rate	0,002
Batch size	10
Epochs	200
Validation_split	0,30
Función de activación capa salida	sigmoid

Tabla 5: Arquitectura utilizada para el entrenamiento 3

Resultados obtenidos

Con la modificación de los hiperparámetros comentados, se consigue mejorar ligeramente el porcentaje de acierto y reducir la tasa de error.

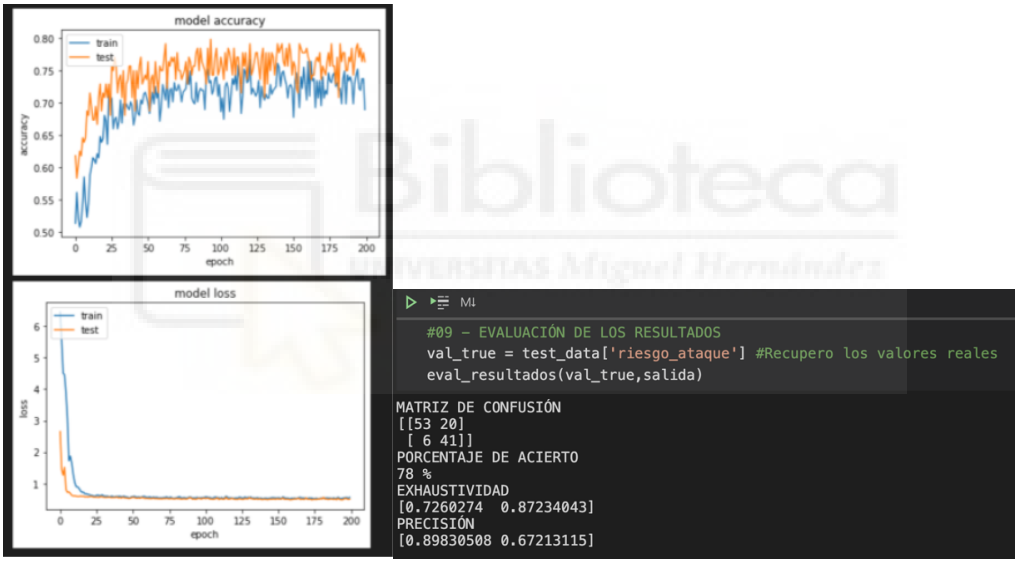


Ilustración 121: Evaluación de los resultados del entrenamiento 3

5.2.1.4.3.4. Entrenamiento 4: Aplicando PCA

La matriz de correlación es una ayuda, no una solución final, pues en ocasiones puede generar más oscuridad que luz. Sobre todo, cuando las variables están muy relacionadas entre sí (no solo con la variable objetivo), provocando que estas puedan meter "ruido" al entrenamiento del modelo. En el dataset por ejemplo existen varias características que presentan este problema, como, por ejemplo, *alguna_vez_casado* que está relacionada sobre todo con la edad. Para minimizar este problema se hizo una nueva batida de entrenamientos aplicando el algoritmo PCA.

Descripción del algoritmo PCA (Análisis de componentes principales)

Se trata de un método para la reducción de la dimensionalidad que permite simplificar la complejidad de espacios con múltiples dimensiones a la vez que conserva la información. El método de PCA permite "condensar" la información aportada por múltiples variables en solo unas pocas componentes [46].

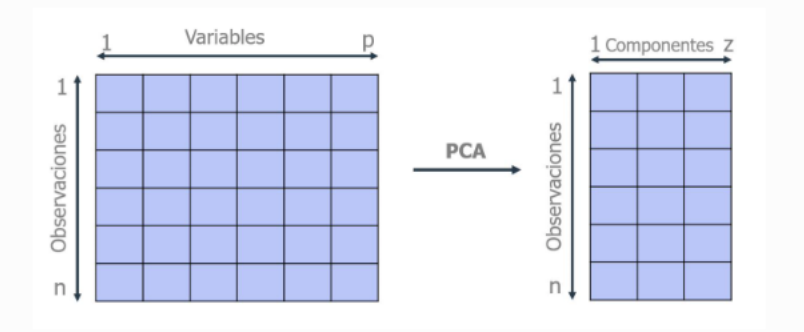


Ilustración 122: Representación gráfica de la finalidad del algoritmo PCA

La librería **scikitlearn** contiene la clase **sklearn.decomposition.PCA** que implementa la mayoría de las funcionalidades necesarias para crear y utilizar modelos PCA.

Partiendo de la arquitectura y características utilizadas en el entrenamiento anterior, se ha aplicado el algoritmo del PCA.

PCA de 2 componentes

```
pca = PCA(n_components=2)
#Entrenamos la variable pca con los datos
pca.fit(train_features)
#Se reduce las dimensiones a 2 dimensiones
transformada = pca.transform(train_features)
```

Ilustración 123: Reducción de la dimensionalidad a dos componentes aplicando PCA

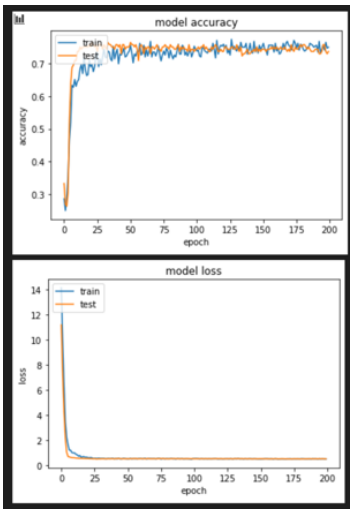


Ilustración 124: Evaluación de los resultados del PCA para dos componentes

Únicamente con dos componentes se puede observar que el desempeño de la red neuronal es ligeramente peor.

PCA de 3 componentes

```
pca = PCA(n_components=3)
#Entrenamos la variable pca con los datos
pca.fit(train_features)
#Se reduce las dimensiones a 3
transformada = pca.transform(train_features)
```

Ilustración 125: Reducción de la dimensionalidad a tres componentes aplicando PCA

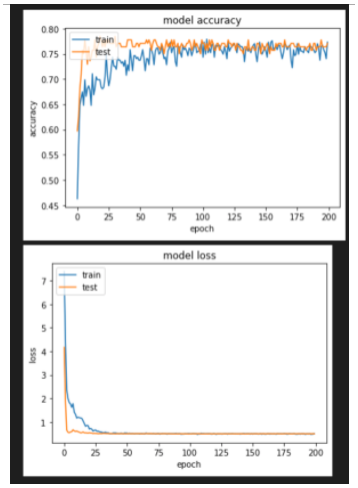


Ilustración 126: Evaluación de los resultados del PCA para tres componentes

Utilizando tres componentes tampoco se aprecia mejoría.

PCA de 4 componentes

Por último, se probó a reducirlo a cuatro componentes, donde se obtuvo el mejor resultado hasta ese momento, llegando a un 80% de acierto y reduciendo ligeramente el error.

```
pca = PCA(n_components=4)
#Entrenamos la variable pca con los datos
pca.fit(train_features)
#Se reduce las dimensiones a 4 dimensiones
transformada = pca.transform(train_features)
```

Ilustración 127: Reducción de la dimensionalidad a cuatro componentes aplicando PCA

```
#09 - EVALUACIÓN DE LOS RESULTADOS
val_true = test_data['riesgo_ataque'] #Recupero los valores reales
eval_resultados(val_true,salida)

MATRIZ DE CONFUSIÓN
[[59 14]
 [10 37]]
PORCENTAJE DE ACIERTO
80 %
EXHAUSTIVIDAD
[0.80821918 0.78723404]
PRECISIÓN
[0.85507246 0.7254902 ]
```

Ilustración 128: Evaluación de los resultados del PCA para cuatro componentes

También se probó con 5 componentes, pero se producía un empeoramiento de los resultados.

5.2.1.4.3.5. Entrenamiento 5: Añadiendo más dropout

Hasta este momento, en los experimentos se ha utilizado únicamente un *dropout* en la capa oculta de la red neuronal. En este nuevo experimento, se realizó una prueba añadiendo un dropout adicional a la capa de entrada. Adicionalmente, se realizaron unas ciertas modificaciones en los hiperparámetros y además se añadió uno nuevo: **MaxNorm**. Este nuevo parámetro, restringe los pesos incidentes a cada neurona de la capa oculta, para tener una norma menor o igual a un valor deseado.

Características
Edad
Hipertensión
Cardiopatía
Alguna_vez_casado
Nivel_medio_de_glucosa
Tipo_de_trabajo
Estado_fumador

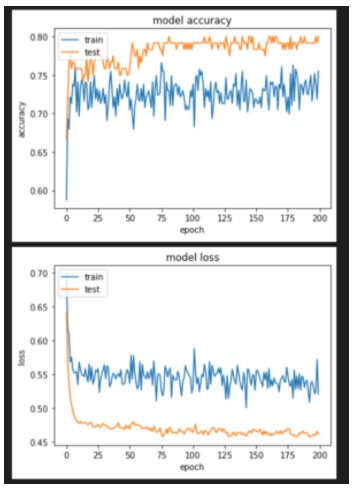
Tabla 6: Características usadas en el entrenamiento 5

Optimizador	Adam
Capas ocultas	1 capa oculta con 9 neuronas
Función de activación capa oculta	Relu
Grado del dropout	0,20 (en capa oculta y capa de entrada)
Learning_rate	0,002
Batch size	10
Epochs	200
Validation_split	0,15
Función de activación capa salida	sigmoid

Tabla 7: Arquitectura utilizada para el entrenamiento 5

Resultados obtenidos

Con estas ligeras modificaciones, se consigue reducir el error e incrementar el acierto a un 82%.




```
#09 - EVALUACIÓN DE LOS RESULTADOS
val_true = test_data['riesgo_ataque'] #Recupero los valores reales
eval_resultados(val_true,salida)
```

```
MATRIZ DE CONFUSIÓN
[[59 14]
 [ 8 39]]
PORCENTAJE DE ACIERTO
82 %
EXHAUSTIVIDAD
[0.80821918 0.82978723]
PRECISIÓN
[0.88059701 0.73584906]
```

Ilustración 129: Evaluación de los resultados del entrenamiento 5

Como se ha podido ver en los diferentes experimentos, únicamente se ha hecho uso de una capa oculta. Durante los experimentos se realizaron pruebas añadiendo más capas

y jugando con el número de neuronas de estas pero los resultados no mejoraban, de hecho, en la mayoría de los casos empeoraban. Los mejores resultados obtenidos fueron con una única capa oculta de 9 neuronas.

Con los resultados obtenidos en este último experimento, donde el porcentaje de acierto ha mejorado ligeramente al reducir (entre otras modificaciones) el *split_validation*, se puede deducir que el motivo por el que cada vez cuesta más reducir el error se debe principalmente a que el dataset de 599 registros es demasiado pequeño y hace que la red llegue un momento que no pueda "aprender" más, debido a la falta de datos de ejemplo.

Sobre PCA comentar un tema importante y es que, no se puede aplicar a un único paciente, si no que requiere un mínimo de registros en función del número de componentes a calcular. Una de las características del sistema que se está desarrollando en este proyecto es la posibilidad de poder predecir a voluntad la probabilidad de las enfermedades en un único paciente. Puesto que esto no fue posible utilizando PCA, se realizaron otros experimentos adicionales con el fin de encontrar una alternativa.

En dichos entrenamientos se hizo uso de la arquitectura utilizada en el entrenamiento 5.2.1.4.3.5. Entrenamiento 5: Añadiendo más dropout (hasta este punto la que mejor resultados ha dado) pero incorporando nuevas características y eliminando otras.

5.2.1.4.4. Otros entrenamientos realizados

Optimizador	Adam
Capas ocultas	1 capa oculta con 9 neuronas
Función de activación capa oculta	Relu
Grado del dropout	0,20 (en capa oculta y capa de entrada)
Learning_rate	0,001
Batch size	10
MaxNorm	3
Epochs	200
Validation_split	0,15
Función de activación capa salida	sigmoid

Tabla 8: Arquitectura utilizada para los entrenamientos adicionales

5.2.1.4.4.1. Entrenamiento adicional 1

Revisando las características utilizadas en el modelo, se puede ver que presentan una diferencia de rangos. Pues, por ejemplo, *hipertensión*, *cardiopatía* y *alguna_vez_casado*

son binarios (valores de 1 y 0), la *edad* el valor mayor que alcanza en el dataset es 84 y, por último, el *nivel_medio_glucosa* donde hay valores superiores a 200. En estos casos donde existe una disparidad en los rangos, es conveniente realizar una normalización o escalado de los mismos. De esta manera, los datos estarán en el mismo rango, haciendo que el modelo no sienta un predominio por la diferencia de estos.

En este caso la diferencia de rangos no es muy grande y la mejoría es muy pequeña, pero es conveniente escalarlos.

Hasta ahora no se había aplicado el escalado debido a que PCA ya lo realiza implícitamente antes de aplicar el algoritmo.

La librería **scikitlearn** dispone de la clase **sklearn.preprocessing** donde es posible importar las funciones necesarias para estandarizar y normalizar datos. Existen varias funciones, siendo las más utilizadas:

- **StandarScaler**: estandariza los datos eliminando la media y escalando los datos de forma que su varianza sea igual a 1 [47].
  Esta función, trabaja a nivel de columna, por lo que es necesario aplicarla a un conjunto de datos, no pudiéndose aplicar a un único registro.

- **MinMaxScaler**: transforma las características escalándolas a un rango dado, por defecto es (0,1), aunque este rango puede ser definido [47].

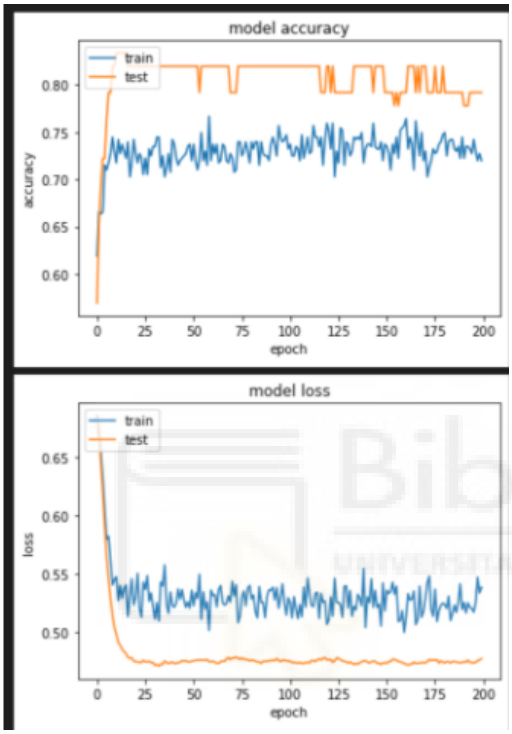
- **Normalizer**: este a diferencia de los dos anteriores, funciona a nivel de filas, no en columnas. Por lo que este si es posible utilizarlo en un único registro (paciente). Esta función transforma todas las características en valores entre -1 y 1 [48].

A continuación, se muestran las características utilizadas y los resultados obtenidos de este primer entrenamiento adicional:

Características
Sexo
Edad
Hipertensión
Cardiopatía

Tabla 9: Características usadas en el entrenamiento adicional 1

Resultados obtenidos




```
MATRIZ DE CONFUSIÓN
[[52 21]
 [10 37]]
PORCENTAJE DE ACIERTO
74 %
EXHAUSTIVIDAD
[0.71232877 0.78723404]
PRECISIÓN
[0.83870968 0.63793103]
```

Ilustración 130: Evaluación de los resultados del entrenamiento adicional 1

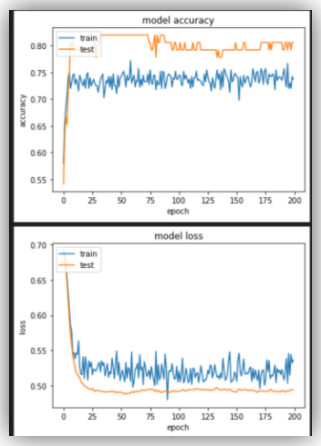
5.2.1.4.4.2. Entrenamiento adicional 2

En este entrenamiento, se añade la característica *nivel_medio_glucosa.*

Características
Sexo
Edad
Hipertensión
Cardiopatía
Nivel_medio_glucosa

Tabla 10: Características usadas en el entrenamiento adicional 2

Resultados obtenidos




```
MATRIZ DE CONFUSIÓN
[[55 18]
 [11 36]]
PORCENTAJE DE ACIERTO
76 %
EXHAUSTIVIDAD
[0.75342466 0.76595745]
PRECISIÓN
[0.83333333 0.66666667]
```

Ilustración 131: Evaluación de los resultados del entrenamiento adicional 2

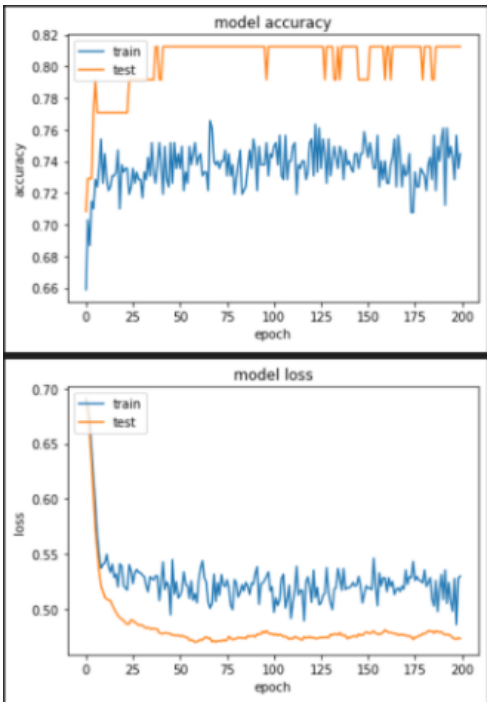
5.2.1.4.4.3. Entrenamiento adicional 3

En este cuarto entrenamiento, se utilizan las mismas características que en el entrenamiento anterior, pero con la diferencia de que se modifica el *split_validation* del entrenamiento de un 0.15 a un 0.10.

Características
Sexo
Edad
Hipertensión
Cardiopatía
Nivel_medio_glucosa

Tabla 11: Características usadas en el entrenamiento adicional 3

Resultados obtenidos




```
MATRIZ DE CONFUSIÓN
[[55 18]
 [ 8 39]]
PORCENTAJE DE ACIERTO
78 %
EXHAUSTIVIDAD
[0.75342466 0.82978723]
PRECISIÓN
[0.87301587 0.68421053]
```

Ilustración 132: Evaluación de los resultados del entrenamiento adicional 3

A medida que se ha ido subiendo el valor del *split_validation* más del 0.1 la precisión ha ido bajando ligeramente. Esto es debido a que al aumentar el *split_validation* se está disminuyendo directamente el número de datos de entrenamiento y si ya de por sí el dataset es pequeño, el hecho de quitar más datos de entrenamiento perjudica.

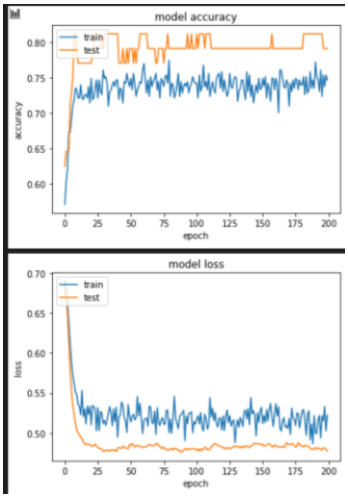
5.2.1.4.4.4. Entrenamiento adicional 4

Para este cuarto entrenamiento, se parte de la base del entrenamiento anterior, pero con la diferencia de que se añade la característica *índice_masa_corporal*.

Características
Sexo
Edad
Hipertensión
Cardiopatía
Nivel_medio_glucosa
Índice_masa_corporal

Tabla 12: Características usadas en el entrenamiento adicional 4

Resultados obtenidos




```
MATRIZ DE CONFUSIÓN
[[55 18]
 [ 8 39]]
PORCENTAJE DE ACIERTO
78 %
EXHAUSTIVIDAD
[0.75342466 0.82978723]
PRECISIÓN
[0.87301587 0.68421053]
```

Ilustración 133: Evaluación de los resultados del entrenamiento adicional 4

Como se aprecia, el resultado ha sido prácticamente igual que en el entrenamiento anterior, por lo que la característica *índice_masa_corporal* es irrelevante y no aporta mejoría al modelo. Además, se hizo otro experimento añadiendo la característica *estado_fumador* y el resultado fue también el mismo.

En base a los últimos experimentos realizados las características que mejor han definido al modelo han sido:

Características
Sexo
Edad
Hipertensión
Cardiopatía
Nivel_medio_glucosa

Tabla 13: Características que mejor definen al modelo implementado

5.2.1.4.4.5. Entrenamiento adicional 5: confirmando el overfitting

En esta última prueba, haciendo uso de la arquitectura del experimento anterior, se pretende evidenciar lo comentado en el apartado 5.2.1.3. Limpieza y preparación del dataset sobre el problema del desbalanceo de datos en el dataset original.

Características
Sexo
Edad
Hipertensión
Cardiopatía
Nivel_medio_glucosa

Tabla 14: Características usadas en el entrenamiento adicional 5

Resultados obtenidos

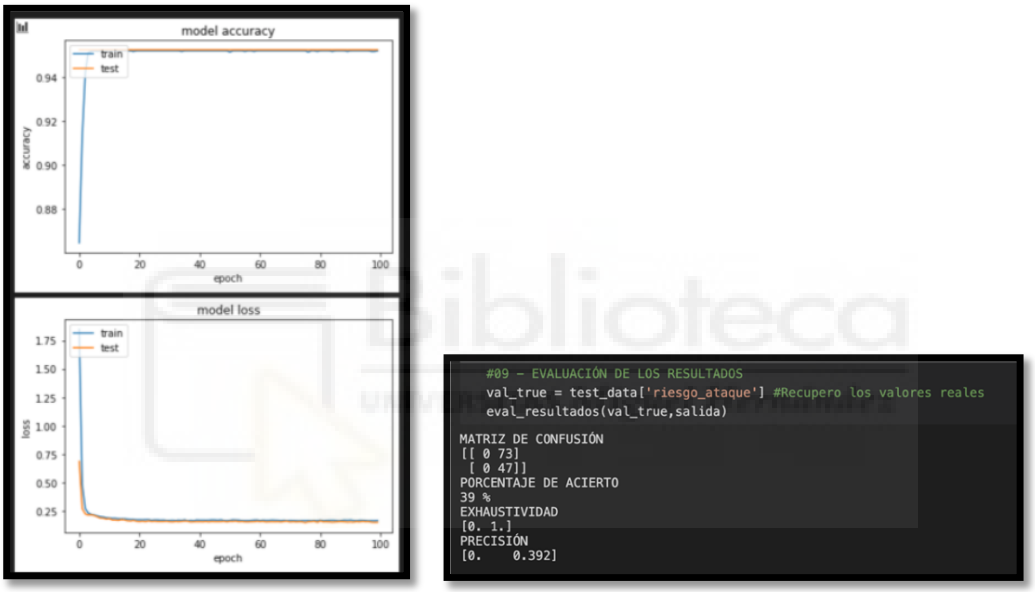


Ilustración 134: Evaluación de resultados del entrenamiento adicional 5

Tal y como se puede apreciar en la ilustración 134 (izquierda), utilizando prácticamente la totalidad el dataset original con los más de 4.900 pacientes, se observa que en el entrenamiento se produce un acierto del 95% y un error de apenas un 0,15% pero a la hora de realizar las predicciones no es capaz de realizar una correcta clasificación.

Como evidencia, se cargó el dataset de validación (120 registros) utilizado en los anteriores entrenamientos citados y se intentó realizar una predicción, obteniendo los resultados mostrados en la ilustración 134 (derecha) y confirmando las sospechas de que el modelo había entrado en **overfitting**.

5.2.1.5. Reutilización de la red neuronal

Una vez se tiene el modelo que mejor se ajusta a los datos del dataset utilizado, es necesario guardarlo para poder reutilizarlo. Keras incorpora el método **save** que permite su guardado para posteriormente cargarlo en cualquier script e invocarlo.

```
#Guardo el modelo
modelo.save('modelo_IC.h5')
```

Ilustración 135: Sentencia para el guardado del modelo entrenado en Keras

Un aspecto muy importante ha tener en cuenta es que las nuevas predicciones que se realicen con el modelo deben adaptarse a las mismas escalas y transformaciones con las que fueron entrenados.

Para este modelo, se desarrolló un script (**modelo_IC.ipynb**) que es el encargado de cargar el modelo, escalar los datos acorde al entrenamiento y realizar la predicción. Estas pautas se deberán mantener para todo modelo que se quiera añadir en el sistema.

Para finalizar este bloque, se muestra un ejemplo donde desde el script creado, se cargan los datos de un único paciente y se obtiene la predicción.

	sexo	edad	hipertension	cardiopatia	nivel_medio_de_glucosa
0	Hombre	77	0	0	57.6

Ilustración 136: Visualización de los datas de un paciente

```
scaler = Normalizer()
datos_paciente_escalados = scaler.fit_transform(datos_paciente)
datos_paciente = pd.DataFrame(datos_paciente_escalados)
```

Ilustración 137: Adecuación de los datos del paciente

```
#CARGO EL MODELO
modelo = keras.models.load_model('modelo_IC.h5')
```

Ilustración 138: Carga del modelo en Keras

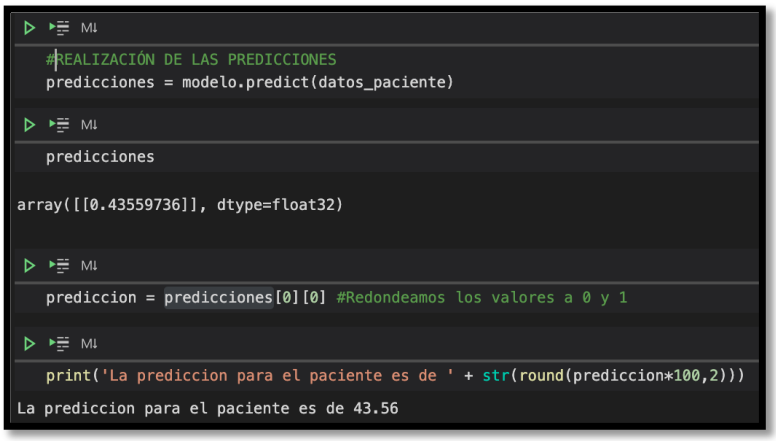


Ilustración 139: Predicción para un paciente

5.2.2. APP

En este segundo bloque de desarrollos, se muestra todo el proceso de diseño y desarrollo del prototipo funcional de la aplicación.

Para el desarrollo de la aplicación se llevaron a cabo las siguientes fases:

1. Análisis de requisitos
2. Diseño BBDD
3. Diseño de la interfaz
4. Implementación
5. Testing

5.2.2.1. Análisis de requisitos

Un análisis de requisitos consiste en el estudio de las necesidades de los usuarios con el fin de definir los requerimientos del sistema, tanto a un nivel de hardware como software. Para la obtención de los requerimientos del sistema, se realiza lo que se conoce como toma de requisitos, que consisten en reuniones llevadas a cabo con el cliente. En dichas reuniones, el cliente expone la funcionalidad y características que quiere que tenga el sistema a desarrollar. Para ello, el analista (persona encargada de llevar a cabo el análisis) podrá ir formulando preguntas al cliente con el fin de enriquecer la toma de requisitos.

En este caso, dadas las circunstancias que se han presentado por la pandemia del COVID-19, no ha sido posible contar con la colaboración expresa de un cliente (centro

médico en este caso), por lo que, todo el análisis de requisitos se ha realizado en base a las características que se han considerado útiles que incorpore este primer prototipo funcional.

5.2.2.1.1. Requisitos funcionales

Aunque para este proyecto se haya desarrollado un primer prototipo funcional, los requisitos funcionales que se han definido para su desarrollo se han realizado pensando en que el diseño en sí sea escalable en el futuro. Estos requisitos son:

- OBJ-01 GESTIÓN DE USUARIOS
- OBJ-02 GESTIÓN DE CENTROS MÉDICOS
- OBJ-03 GESTIÓN DE FICHAS CLÍNICAS
- OBJ-04 GESTIÓN DE ANÁLISIS MÉDICOS
- OBJ-05 GESTIÓN DE MODELOS RRNN
- OBJ-06 GESTIÓN DE ENFERMEDADES
- OBJ-07 GESTIÓN DE CARACTERÍSTICAS

5.2.2.1.2. Requisitos no funcionales

- Diseño de una aplicación multiplataforma
- Interfaz sencilla e intuitiva para ser utilizada por personal médico y no técnicos.

5.2.2.1.3. Actores

En este apartado, se definen los actores que van a interactuar con esta primera versión de la aplicación.

ACT-01	**No registrado**
Descripción	Actor que representa un usuario que no tiene una cuenta o bien todavía no se ha identificado en el sistema.
Comentarios	N/A
Casos de uso relacionados	RF-01

Tabla 15: Actor no registrado

ACT-02	Médico
Descripción	Actor que representa a un médico.
Comentarios	Se trata del actor principal en este primer prototipo desarrollado.
Casos de uso relacionados	RF-01, RF-02, RF-03, RF-04, RF-05 y RF-06

Tabla 16: Actor médico

Todos los diseños y desarrollos mostrados a continuación giran en torno al actor **médico**.

5.2.2.1.4. Diagramas

En las siguientes figuras se exponen las acciones que puede llevar a cabo cada uno de los actores presentados en el anterior apartado.

Diagrama gestión de usuarios

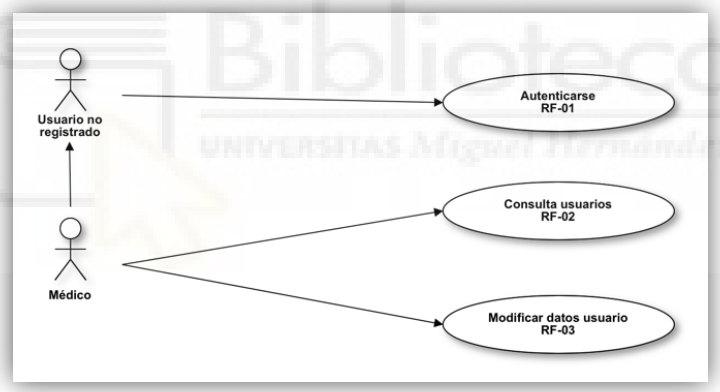


Ilustración 140: Diagrama de gestión de usuarios

Diagrama gestión fichas clínicas

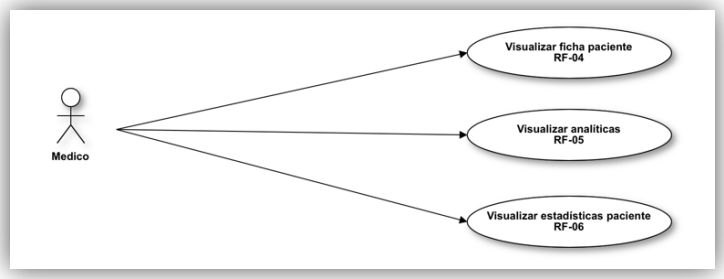


Ilustración 141: Diagrama de gestión de fichas clínicas

5.2.2.1.5. Requisitos de almacenamiento de información

RI-01	**Información sobre usuarios**
Objetivos asociados	OBJ-01 - Gestionar los usuarios
Requisitos asociados	RF-01 – Autentificarse RF-02 – Consulta usuarios RF-03 – Modificación datos usuario
Descripción	El sistema deberá almacenar la información correspondiente a los usuarios
Datos específicos	• Nombre • Apellido 1 • Apellido 2 • Fecha nacimiento • Población • Código Postal • Dirección • Médico asociado • Nº colegiado • Teléfono • Mail • Username • Password • Centro médico al que pertenece • Características demográficas
Comentarios	

Tabla 17: Requisitos de información para usuarios

RI-02	**Información sobre fichas clínicas**
Objetivos asociados	OBJ-02 - Gestionar las fichas clínicas
Requisitos asociados	RF-04 – Visualizar ficha paciente RF-05 – Visualizar analíticas RF-06 – Visualizar estadísticas pacientes
Descripción	El sistema deberá almacenar la información correspondiente a las fichas clínicas de los pacientes, para su posterior explotación por parte del personal sanitario.
Datos específicos	• Fecha realización analítica • Resultado analíticas • Listado de características-valor
Comentarios	

Tabla 18: Requisitos de información para fichas clínicas

5.2.2.1.6. Casos de uso

Casos de uso del subsistema **gestión de usuarios.**

<table>
<tr><td>RF-01</td><td>Autentificación</td></tr>
<tr><td>Objetivos asociados</td><td>OBJ-01 - Gestionar los usuarios</td></tr>
<tr><td>Requisitos asociados</td><td>RI-01 Información sobre usuarios</td></tr>
<tr><td>Descripción</td><td>El sistema deberá comportarse tal y como se describe en el siguiente caso de uso cuando cualquier actor que intente autentificarse.</td></tr>
<tr><td>Precondición</td><td>Desempeñar cualquier acción especificada en los casos de uso.</td></tr>
<tr><td>Secuencia normal</td><td><table><tr><td>Paso</td><td>Acción</td></tr><tr><td>1</td><td>El actor solicita al sistema comenzar el proceso de autentificación.</td></tr><tr><td>2</td><td>El sistema solicita los datos de acceso al actor.</td></tr><tr><td>3</td><td>El sistema comprueba que los datos introducidos en el paso nº 2 son correctos.</td></tr><tr><td>4</td><td>El sistema permite el acceso para que el actor pueda interactuar con los casos de uso.</td></tr></table></td></tr>
<tr><td>Postcondición</td><td>El actor se encuentra autentificado en el sistema.</td></tr>
<tr><td>Excepciones</td><td><table><tr><td>Paso</td><td>Acción</td></tr><tr><td>2</td><td>Si el actor solicita cancelar la operación, el sistema la cancela, a continuación, este caso de uso termina.</td></tr><tr><td>3</td><td>Si los datos introducidos por el actor son incorrectos este vuelve a solicitarlos.</td></tr></table></td></tr>
<tr><td>Rendimiento</td><td><table><tr><td>Paso</td><td>Cota de tiempo</td></tr><tr><td>3</td><td>2 segundos</td></tr></table></td></tr>
<tr><td>Estabilidad</td><td>Alta</td></tr>
<tr><td>Comentarios</td><td>N/A</td></tr>
</table>

Tabla 19: Caso de uso para autentificación en el sistema

RF-02	**Consulta usuarios**
Objetivos asociados	OBJ-01 - Gestionar los usuarios
Requisitos asociados	RI-01 Información sobre usuarios
Descripción	El sistema deberá comportarse tal y como se describe en el siguiente caso de uso cuando cualquier actor intente consultar usuarios.
Precondición	Estar logueado en el sistema como médico
Secuencia normal	**Paso** | **Acción**

	1 | El actor solicita al sistema visualizar sus pacientes. 2 | El sistema muestra los pacientes
Postcondición	Aparecen los pacientes
Excepciones	Ninguna
Rendimiento	**Paso** | **Cota de tiempo** 1 | 1 segundo 2 | 3 segundos (depende del número de registros en la BBDD)
Estabilidad	Alta
Comentarios	Adicionalmente, a la visualización de los pacientes de un médico, este podrá solicitar al sistema la búsqueda de un paciente en concreto.

Tabla 20: Caso de uso para la consulta de usuarios

RF-03	**Modificar datos de usuario**
Objetivos asociados	OBJ-01 - Gestionar los usuarios
Requisitos asociados	RI-01 Información sobre usuarios
Descripción	El sistema deberá comportarse tal y como se describe en el siguiente caso de uso cuando cualquier actor intente modificar sus datos de usuario.
Precondición	Estar logueado en el sistema.
Secuencia normal	**Paso** | **Acción** 1 | El actor solicita al sistema modificar sus datos 2 | El sistema habilita la edición de los datos 3 | El actor modifica sus datos y solicita al sistema actualizarlos. 4 | El sistema actualiza los datos.
Postcondición	Datos del actor actualizados
Excepciones	**Paso** | **Acción** 2 | El actor cancela la edición, el caso de uso finaliza. 3 | El sistema al validar los datos detecta datos incorrectos y vuelve a solicitarlos al actor.
Rendimiento	**Paso** | **Cota de tiempo** 1 | 2 segundos
Estabilidad	Alta

Comentarios	Este caso de uso por ser menos relevante para esta primera versión y por limitaciones de tiempo, no se implementará.

Tabla 21: Caso de uso para la modificación de datos de usuario

Casos de uso del subsistema **gestión fichas clínicas.**

RF-04	**Visualizar ficha paciente**
Objetivos asociados	OBJ-03 - Gestionar las fichas clínicas
Requisitos asociados	RI-02 Información sobre fichas clínicas
Descripción	El sistema deberá comportarse tal y como se describe en el siguiente caso de uso cuando cualquier actor intente visualizar la ficha de un paciente.
Precondición	Estar logueado en el sistema como médico
Secuencia normal	**Paso** / **Acción** 1 / El actor solicita al sistema visualizar la ficha clínica de un paciente. 2 / El sistema muestra la ficha clínica del paciente.
Postcondición	Aparecen los datos del paciente, analíticas y opción de ver sus estadísticas.
Excepciones	Ninguna
Rendimiento	**Paso** / **Cota de tiempo** 1 / 1 segundo 2 / 3 segundos (depende del número de registros en la BBDD)
Estabilidad	Alta
Comentarios	N/A

Tabla 22: Caso de uso para la visualización de la ficha del paciente

RF-05	**Visualizar analíticas paciente**
Objetivos asociados	OBJ-03 - Gestionar las fichas clínicas
Requisitos asociados	RI-02 Información sobre fichas clínicas
Descripción	El sistema deberá comportarse tal y como se describe en el siguiente caso de uso cuando el médico desee visualizar las analíticas de un paciente.
Precondición	Estar logueado en el sistema como médico y haber accedido a la ficha del paciente.
Secuencia normal	**Paso** / **Acción**

	1 | El actor solicita al sistema visualizar las analíticas de un paciente. 2 | El sistema muestra las analíticas del paciente.
Postcondición	Aparecen las analíticas del paciente.
Excepciones	Ninguna
Rendimiento	**Paso** | **Cota de tiempo** 1 | 1 segundo 2 | 2 segundos (depende del número de registros en la BBDD)
Estabilidad	Alta
Comentarios	N/A

Tabla 23: Caso de uso para la visualización de analíticas del paciente

RF-06	**Visualizar estadísticas paciente**
Objetivos asociados	OBJ-03 - Gestionar las fichas clínicas
Requisitos asociados	RI-02 Información sobre fichas clínicas
Descripción	El sistema deberá comportarse tal y como se describe en el siguiente caso de uso cuando el médico desee visualizar las estadísticas del paciente.
Precondición	Estar logueado en el sistema como médico y haber accedido a la ficha del paciente.
Secuencia normal	Paso | Acción 1 | El actor solicita al sistema visualizar las estadísticas de un paciente. 2 | El sistema muestra las estadísticas del paciente.
Postcondición	Aparecen las estadísticas del paciente.
Excepciones	Paso | Acción 2 | En caso de que el paciente no tenga todas las características necesarias para el modelo, se notificará al actor (médico).
Rendimiento	Paso | Cota de tiempo 1 | 1 segundo 2 | 3 segundos (depende del número de registros en la BBDD)
Estabilidad	Alta

Comentarios	N/A

Tabla 24: Caso de uso para la visualización de las estadísticas del paciente

5.2.2.1.7. Diagramas de actividad

Un diagrama de actividad es un tipo de esquema gráfico que define el orden cronológico de las actividades que se producen al llevar a cabo una determinada acción.

En este apartado, se exponen los diagramas de actividad básicos diseñados.

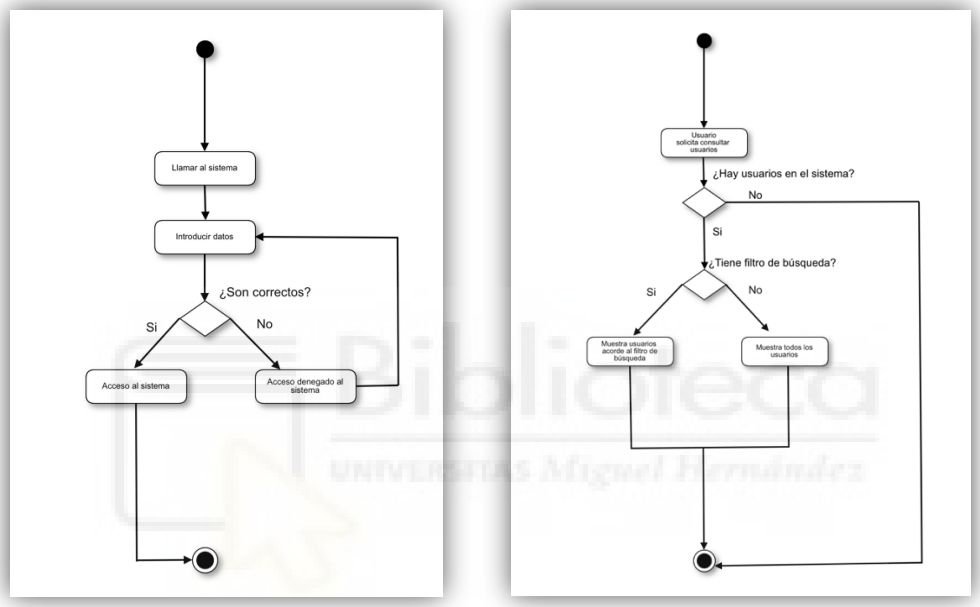


Ilustración 142: Diagramas de actividad para la autentificación y consulta de pacientes

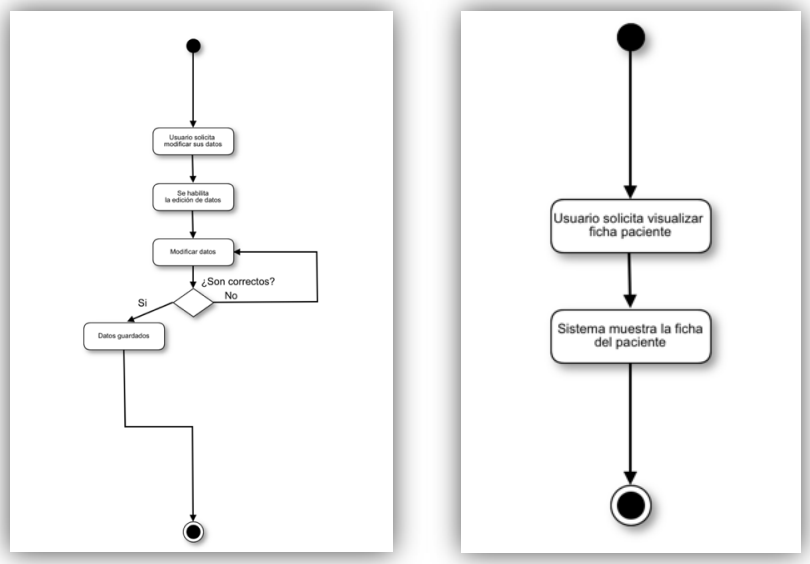


Ilustración 143: Diagramas de actividad para la modificación y visualización de datos de pacientes

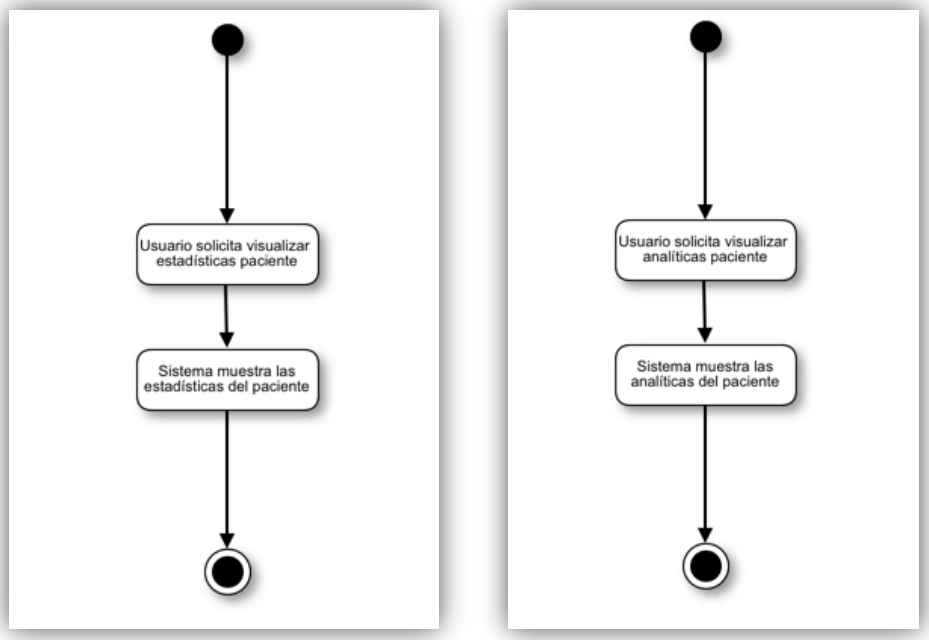


Ilustración 144: Diagramas de actividad para la visualización de estadísticas y analíticas

5.2.2.2. Diseño Wireframe

Una vez definidos los requisitos se procedió a realizar un boceto (o Wireframe) de alta fidelidad, que reflejara el esqueleto del diseño de las distintas pantallas que se pretendía conseguir. El diseño de la interfaz se realizó con el programa **adobe Xd**, donde también se implementó la interoperabilidad entre las distintas pantallas. Dicho diseño, se ideó con el objetivo de que la interfaz resultara atractiva, intuitiva, fácil de usar y que provocara una agradable experiencia al usuario. Para ello, se tomó como referencia los distintos widgets (componentes) que ofrece Flutter.

Ilustración 145: Diseño de la interfaz de la aplicación móvil

5.2.2.3. Implementación

La implementación de la aplicación móvil se ha realizado con la característica de que su uso debe ser enteramente online. Es decir, la aplicación necesita expresamente que exista conexión con la API. Por lo que no está soportado su uso en modo offline.

La totalidad de la aplicación se basa en widgets (componentes de Flutter) y servicios (a través del paquete *http* de Flutter), que son los encargados de operar con la API. En determinadas ocasiones es necesario ejecutar tareas de forma asíncrona, de manera que no permitan un bloqueo de la aplicación mientras dura dicha tarea. Para este cometido se han usado objetos de tipo *Future,* que implementan los async/await. Estos objetos han sido utilizados en todas y cada una de las comunicaciones entre la aplicación y la API.

Cuando un médico inicia sesión en la aplicación, dicho usuario se queda guardado en las preferencias de la aplicación (hasta que se cierre sesión). Por lo que si se cierra y vuelve a abrir la aplicación no solicitará de nuevo las credenciales.

El patrón que se ha seguido a la hora de desarrollar la aplicación ha sido el MVC (modelo vista controlador).

Este método se caracteriza por separar los componentes de una aplicación en tres grupos (o capas) principales: el modelo, la vista y el controlador, y describe cómo se relacionarán entre ellos de cara a mantener una estructura limpia y organizada [49].

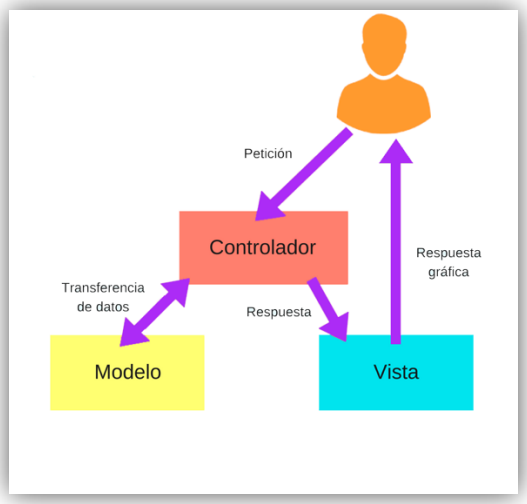


Ilustración 146: Arquitectura MVC

Extrapolando este patrón a la aplicación que se ha desarrollado en Flutter, se tendría, por un lado, los modelos que sirven para almacenar la información que se trae desde la API y que posteriormente el controlador utilizará para construir y alimentar la vista.

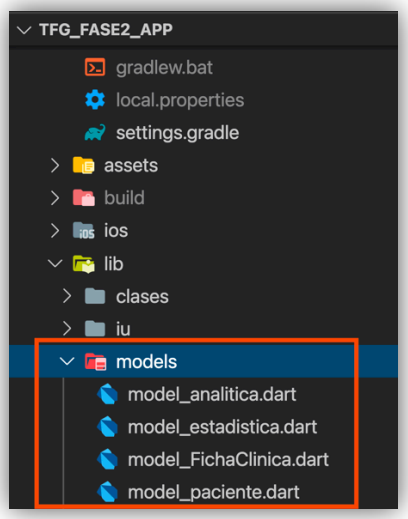


Ilustración 147: Listado de algunos modelos implementados en la aplicación

Por otro lado, las vistas (pantallas), que son las encargadas de generar la interfaz de la aplicación, es decir, representan el estado del modelo en un momento concreto. Todas las vistas desarrolladas en la aplicación están compuestas por un conjunto de widgets. Estos widgets forman el controlador, que actúan como intermediario entre el usuario y el sistema, siendo capaz de capturar las acciones de este sobre la vista, como puede ser, por ejemplo: realizar un scroll sobre la lista de los pacientes, pulsar el botón de login, etc.

Un claro ejemplo de controlador podría ser el widget *FutureBuilder*, este widget ha sido ampliamente utilizado, ya que proporciona de una forma natural tareas de transformación de información de datos y comunicación, haciendo que la vista y el modelo se “entiendan” a la perfección.

Comunicación aplicación y API

Para la comunicación entre la aplicación y la API se ha desarrollado la clase *Wservice* que contiene atributos y métodos estáticos. Dicha clase contiene la IP (localhost en este caso) de acceso a la API, así como el puerto por el que acceder. De esta forma se tiene centralizado el acceso a la API, pudiendo modificarse si fuese necesario desde un único punto y haciéndose extensible a todos los sitios donde se haga referencia en la aplicación.

Por otro lado, los métodos estáticos de la clase *Wservice* representan cada uno de los recursos (rutas) de la API, pudiéndose utilizar en donde se necesite sin la necesidad de crear múltiples objetos.

Una vez presentados los pasos de la implementación de la aplicación, se muestran imágenes de las diferentes pantallas desarrolladas. Los despliegues se han realizado en dos dispositivos emulados. Siendo el iPhone 12 Pro Max para iOS y el Pixel 4 XL para Android.

Pantalla de bienvenida

Se trata de la primera pantalla que aparece al abrir la aplicación. Aunque cabe destacar que si previamente se ha iniciado sesión (y esta se ha quedado guardada, es decir no se ha realizado el cierre) esta pantalla ya no aparecería y, en su lugar, se mostraría la pantalla de inicio correspondiente al tipo de usuario logueado. En este caso, dado que se ha desarrollado la aplicación en base al rol "médico" si este se encuentra ya logueado, la primera pantalla que aparecería al abrir la aplicación sería la de sus pacientes (inicio).

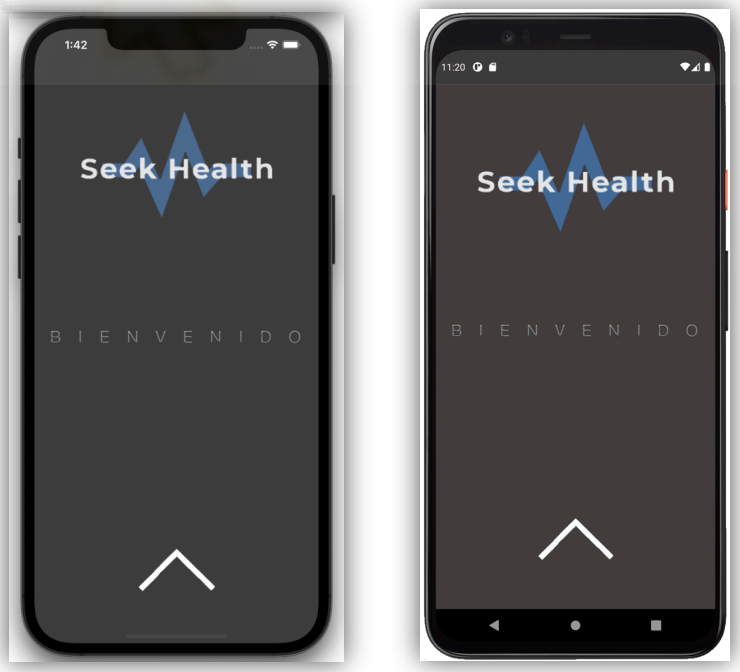


Ilustración 148: Pantalla de bienvenida de Seek Health en iOS y Android

Pantalla de login

Una vez la pantalla de bienvenida es deslizada, aparece la pantalla de login. Desde esta, el usuario puede iniciar sesión con sus credenciales. En caso de no recordarlas, podrá solicitarlas indicando su usuario.

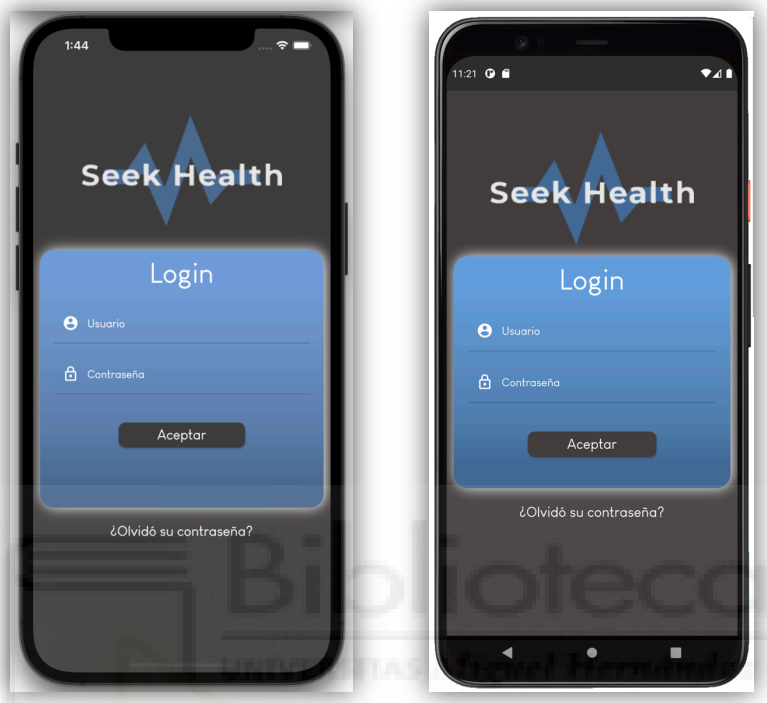


Ilustración 149: Pantalla de login de Seek Health en iOS y Android

Pantalla de pacientes

Una vez el médico se ha logueado, la primera pantalla que aparece es la lista de sus pacientes. Pulsando sobre los pacientes se accedería a su ficha clínica. Desde el menú inferior de esta pantalla se puede ir a la zona de búsqueda de pacientes y al perfil del usuario logueado. Por otro lado, el menú superior indica el nombre de la zona en la que se encuentra el usuario. Adicionalmente, ofrece en todo momento la posibilidad de realizar el cierre de sesión.

Ilustración 150: Pantalla de inicio del médico de Seek Health en iOS y Android

Búsqueda de pacientes

Esta pantalla permite realizar la búsqueda de pacientes. En el menú superior se dan dos opciones: volver a la zona desde la que se llamó y limpiar el cuadro de búsqueda.

Ilustración 151: Pantalla de búsqueda pacientes de Seek Health en iOS y Android

Pantalla perfil cuenta

Esta pantalla muestra los datos del médico logueado. La interfaz se ha dividido en dos zonas donde aparecen los datos personales y algunos datos médicos. Cabe destacar que la pantalla implementa un *SingleChildScrollView* (widget de Flutter). De manera que se pueden añadir todos los datos que sean necesario.

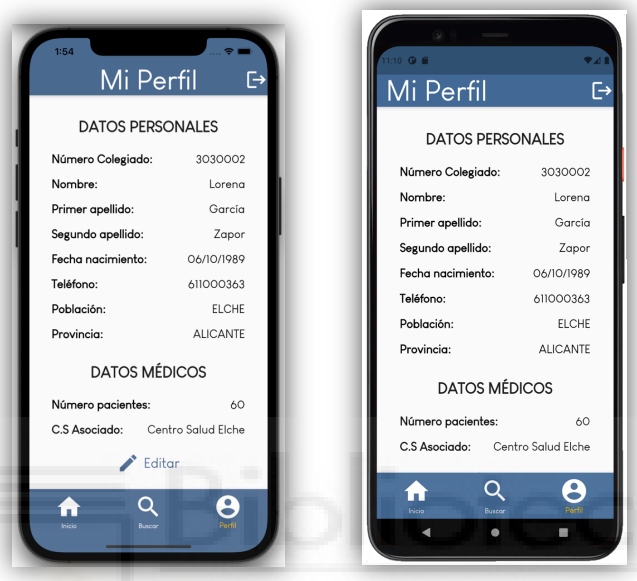


Ilustración 152: Pantalla perfil cuenta médico de Seek Health en iOS y Android

Como se puede apreciar, la pantalla del emulador de Android es ligeramente inferior, por ello el botón editar no aparece directamente, aquí es donde entra en juego el widget *SingleChildScrollView* que permite mostrarlo haciendo *scroll*.

Pantalla ficha clínica paciente

Esta pantalla aparece cuando desde la zona de *inicio* (listado de los pacientes) se pulsa sobre un paciente cualquiera, cargando de esta forma su ficha clínica. En este caso, el menú inferior cambia, ofreciendo 3 nuevas opciones. Por un lado, la opción *Ficha*, desde donde se indican los datos personales y datos clínicos (primera opción del menú y la que aparece por defecto). Segunda: el listado de las analíticas que se ha realizado el paciente, y, como última opción del menú: *Estadísticas,* desde donde se accede a los datos estadísticos del paciente. Al igual que la pantalla anterior esta también incorpora un *SingleChildScrollView,* que permite poder añadir cuantos datos se necesite a la vista.

Ilustración 153: Pantalla ficha clínica paciente de Seek Health en iOS y Android

Pantalla de las analíticas paciente

Esta pantalla muestra el listado de las analíticas que se ha realizado el paciente, ordenadas por la más reciente. Pulsando en cada analítica se accede a la pantalla del detalle de esta.

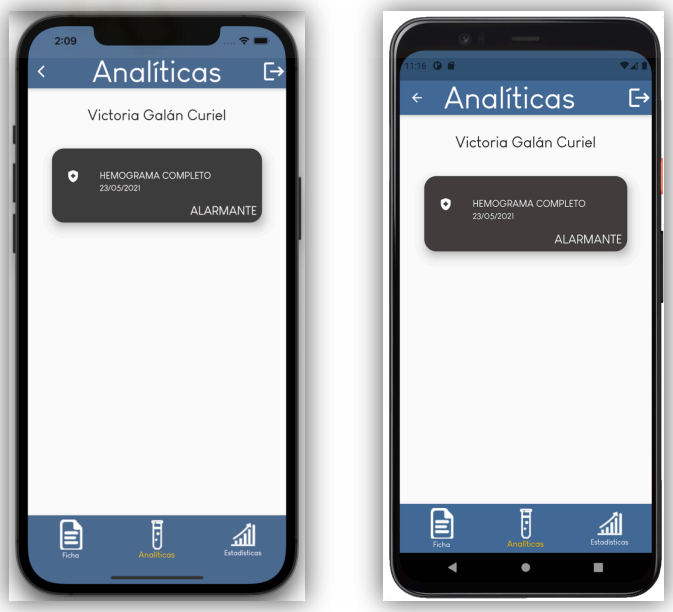


Ilustración 154: Pantalla de analíticas paciente de Seek Health en iOS y Android

Detalle de analítica paciente

Esta pantalla muestra todos y cada uno de los parámetros que constituye la analítica. Así mismo, esta pantalla también incorpora un *SingleChildScrollView*, haciendo que se puedan visualizar haciendo *scroll* todos y cada uno de los parámetros que conformen la analítica.

Ilustración 155: Pantalla de detalles analítica de Seek Health en iOS y Android

Pantalla de las estadísticas del paciente

Esta pantalla es una de las más importante de la aplicación, desde ella se indica la probabilidad que tiene el paciente en cuestión de desarrollar las enfermedades que se tengan definidas en el sistema. En este caso, la enfermedad infarto cerebral. Se ha definido un baremo para que en función de la probabilidad aparezca en un color u otro, simbolizando un nivel de alerta.

Ilustración 156: Pantalla de estadísticas paciente de Seek Health en iOS y Android

Para una visualización más exhaustiva de la aplicación, se ha realizado un manual de usuario, disponible en el apartado 8.1 Manual de usuario app Seek Health del anexo.

5.2.2.4. Testing

El *testing* es un proceso que consiste en una serie de pruebas cuyos principales objetivos son:

- Demostrar que el software creado **tiene** errores.
- Validar que el software realiza las funciones para las que fue creado.

El testeo de un software también sirve para llegar a conocer que tan bueno es, en otras palabras, actúa como una medida de detección de calidad.

Un aspecto que hay que tener en cuenta de cara afrontar un proceso de *testing* es que, es imposible realizar una prueba completa, debido al alto número de posibles entradas y flujos que puede tener un software.

Pautas de cara a elaborar un testing [50]:

1. Antes de empezar con las pruebas es necesario definir los resultados esperados.

2. La persona que elabore el *testing* debe evitar (siempre en la medida de lo posible) probar su propio desarrollo.
3. Se deben revisar las pruebas de manera profunda.
4. Las pruebas deben incluir tanto entradas inválidas e inesperadas como entradas válidas y esperadas.
5. Se debe revisar que el software hace lo que se espera que haga, pero también lo que se espera que no haga.

Actividades principales del proceso de testing

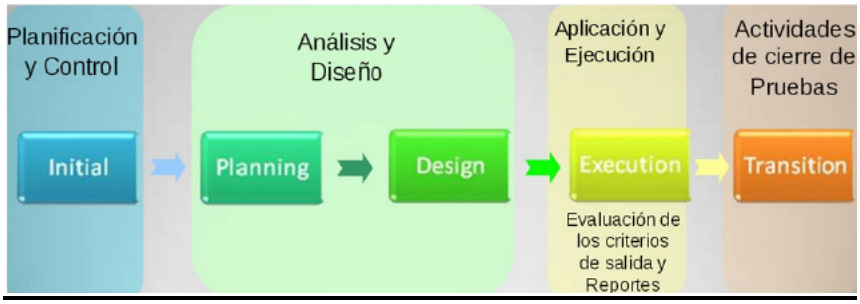


Ilustración 157: Actividades principales del proceso de testing

Testing de la aplicación móvil

Dado que la aplicación no está pensada para trabajar de forma offline, se han realizado una serie de pruebas mínimas de *testing* en esta primera versión con el fin de detectar errores que pueda provocar en la aplicación la no disponibilidad o conexión con la API.

La elaboración de las pruebas de *testing* pueden demorar mucho tiempo, por lo que, en este caso las pruebas realizadas se han enfocado en las funcionalidades más relevantes de la aplicación.

Test 1: Médico sin pacientes y búsqueda si coincidencias

Para el primer caso, se valida que, si un médico que inicia sesión no ha tratado todavía pacientes, el sistema debe mostrar un mensaje donde informe al usuario de forma clara. En el segundo caso, se valida que, en el momento de realizar la búsqueda de pacientes, si el filtro que se ha introducido no produce ninguna coincidencia en base de datos, el sistema debe informar al usuario con un mensaje coherente a la situación. En ambos casos el funcionamiento fue el esperado.

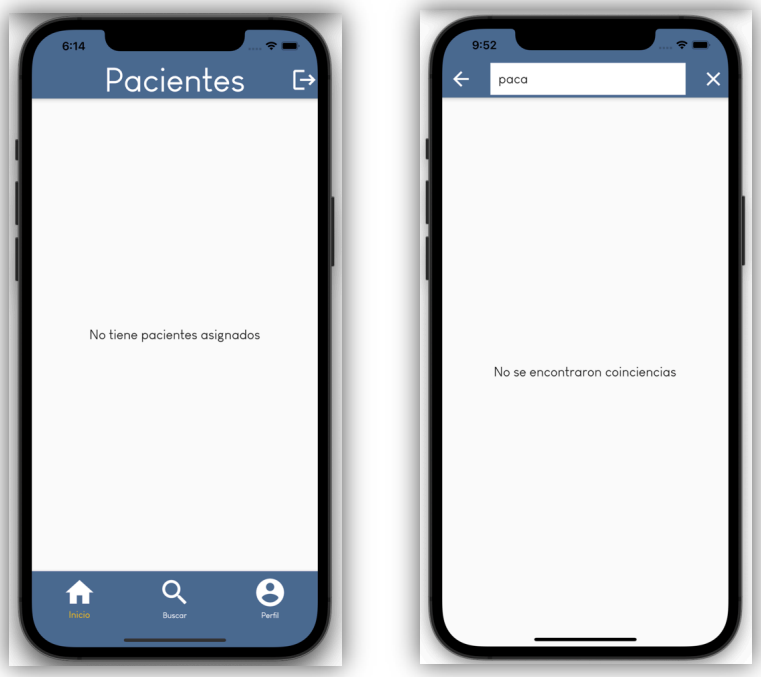


Ilustración 158: Resultados del test 1 – Pantalla de inicio del médico

Test 2: Sin conexión con la API

En esta prueba, se valida que en caso de que la aplicación no pueda conectar con la API, ya sea porque no ha sido arrancada o por algún problema en la comunicación, la aplicación no se cuelgue y en su defecto alerte al usuario.

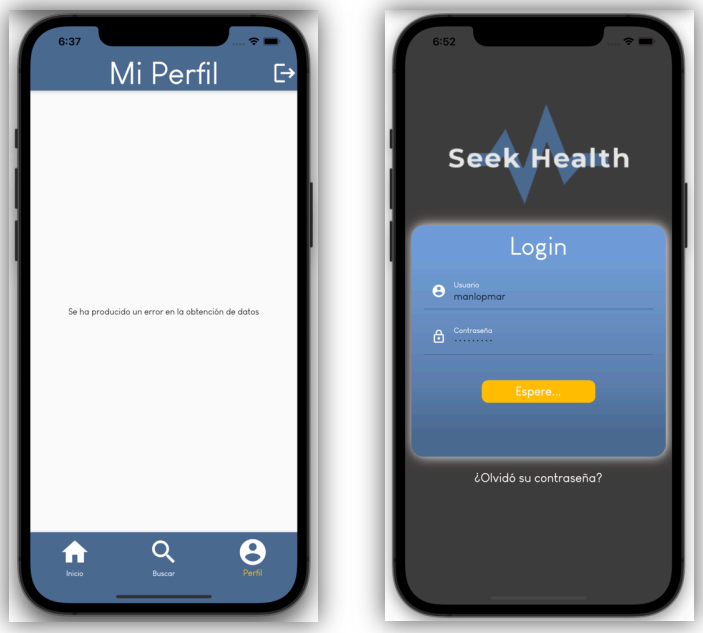


Ilustración 159: Resultados del test 1 - Pantallas perfil médico y login

En el caso del primer ejemplo (imagen izquierda de la ilustración 159) donde se intenta consultar el perfil del usuario logueado mientras está la API caída, muestra un mensaje informando al usuario, por lo que su funcionamiento es el esperado. Pero, en el caso de la pantalla de login (imagen derecha de la ilustración 159) se obtuvo un bug en el que al intentar iniciar sesión cuando la API no está en ejecución generó que la aplicación se quedase bloqueada sin mostrar ningún mensaje al usuario.

Por último, se validó el comportamiento de la no disponibilidad de la API en el resto de las pantallas.

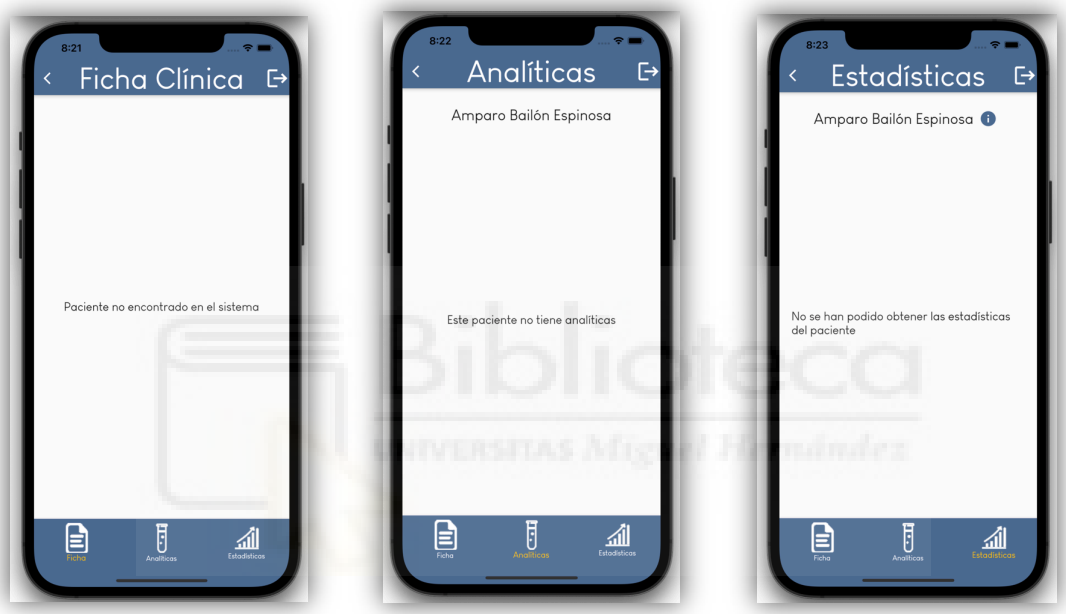


Ilustración 160: Resultados del test 2 - Pantallas de ficha clínica, analíticas y estadísticas

En esta última también se detectaron comportamientos no esperados de la aplicación a la hora de consultar la ficha del paciente, analíticas y estadísticas cuando la API está caída. El problema que es que la ejecución no estaba entrando en el flujo que debía, por lo que los mensajes que mostraba al usuario no eran coherentes al problema que se estaba produciendo.

Corrección de bugs

Una vez se detectaron los bugs comentados en el párrafo anterior, se comprobó que el problema fuera reproducible, una vez se confirmó este escenario, se evaluaron las posibles modificaciones en el código con el fin de corregir el problema, de manera que

los mensajes que se mostrasen fueran coherentes a la situación que se presentara en cada caso.

5.2.3. Base de datos

El diseño de base de datos es un aspecto muy importante dentro de cualquier desarrollo de software. Pues, una elaboración equivocada de él repercutirá negativamente en el rendimiento computacional del sistema en su conjunto.

Para el almacenamiento de la información se planteó el diseño como una interoperabilidad entre dos zonas claramente diferenciadas:

1. **Gestión de centros médicos**: datos de centros médicos, usuarios de acceso al sistema y datos médicos de los pacientes.
2. **Gestión de modelos**: almacenamiento de los modelos y enfermedades dados de alta en el sistema. Así como las diferentes características que éstos necesitan.

En el diseño, caben destacar dos entidades centrales:

- **USUARIO**: esta entidad representa a los actores del sistema. A su vez, es la entidad padre de una generalización, donde de ella cuelgan los diferentes tipos usuarios del sistema (subentidades), en este caso: **PACIENTE** y **MÉDICO**.
- **CARACTERISTICA**: al igual que la entidad USUARIO, esta entidad es entidad padre de una generalización, donde de ella cuelgan las entidades: **CARACT_ANALITICA** y **CARACT_DEMOGRAFICA**.

La implementación de estas generalizaciones dota al sistema de una flexibilidad de cara a añadir nuevos tipos de usuarios (así como sus correspondientes atributos) y tipos de características.

En el siguiente apartado, se muestra en detalle el diseño completo del modelo de entidad-relación.

5.2.3.1. Modelo entidad-relación

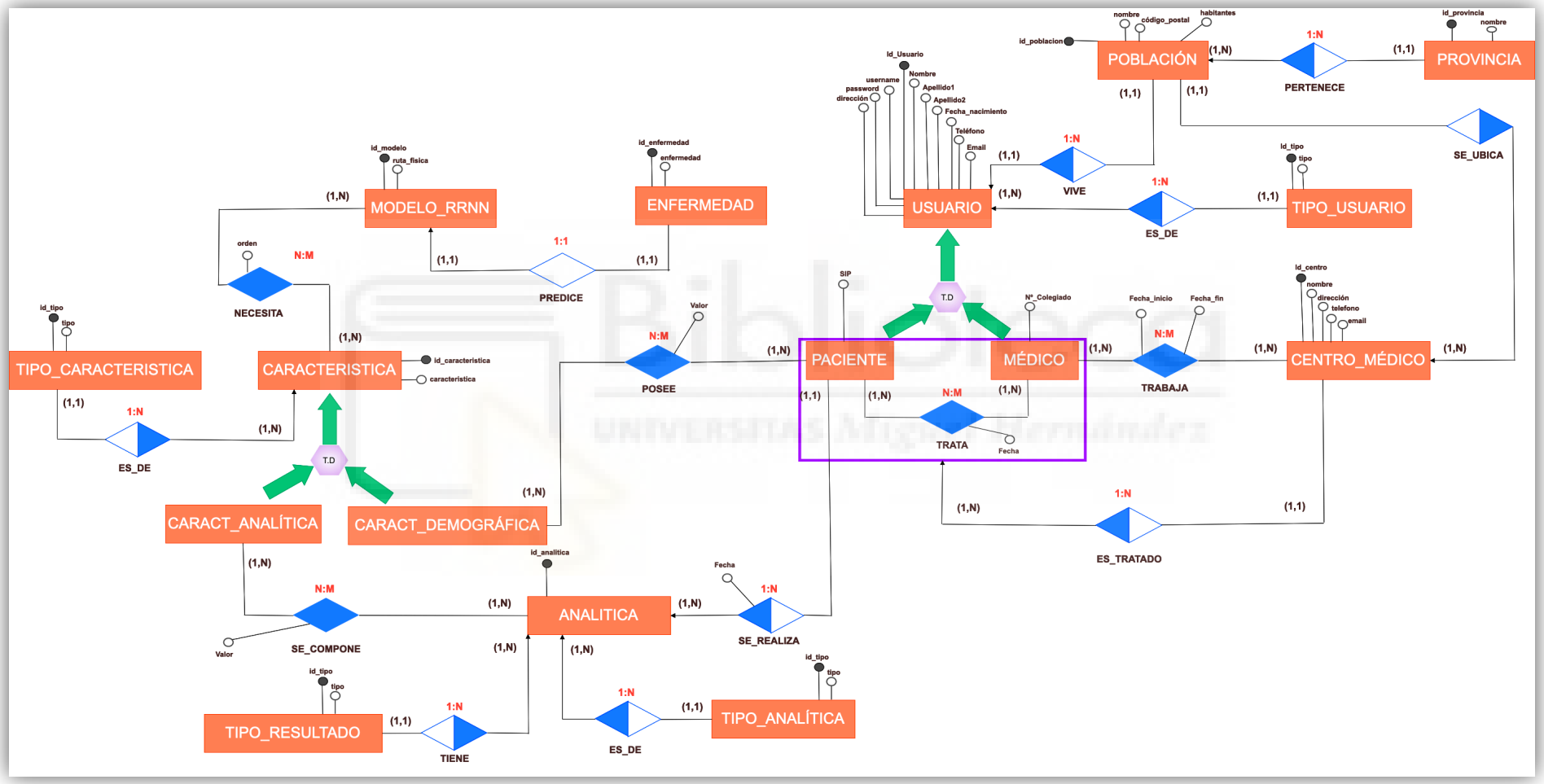


Ilustración 161: Diseño Entidad-Relación de la base de datos

Las relaciones más destacables del diagrama de entidad-relación son:

- **Predice**: un modelo de red neuronal puede realizar la predicción de una única enfermedad.
- **Modelo necesita características**: para que en el momento de las nuevas predicciones los resultados no se vean alterados, el orden de las características con las que se debe llamar al modelo debe ser el mismo que con el que fueron entrenadas. Por ello, se ha incorporado el atributo “orden” en dicha relación, para mantener esta coherencia.
- **Paciente trata médico**: un médico a lo largo de su carrera profesional podrá tratar a pacientes en distintos centros médicos.

5.2.3.2. Modelo relacional

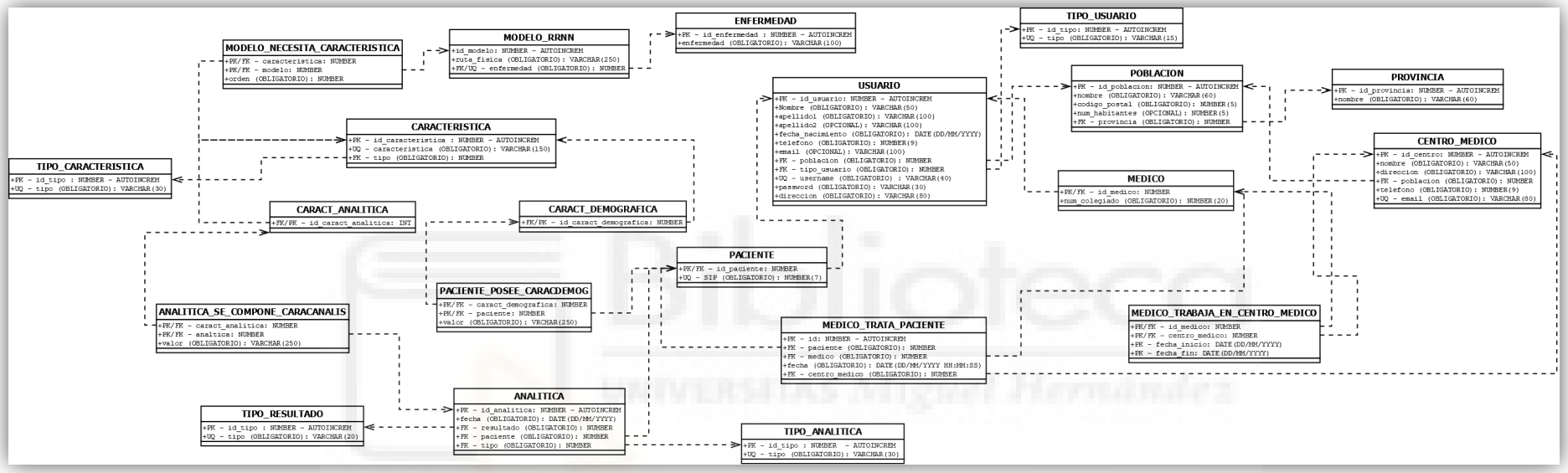


Ilustración 162: Modelo relacional de la base de datos

En el diagrama entidad-relación, las entidades con relación de tipo N:M tras el paso al modelo relacional, generan nuevas tablas que hasta ahora sólo habían sido visibles de una forma conceptual. A continuación, se describen estas nuevas tablas.

- **MODELO_NECESITA_CARACTERISTICA**: almacena las características necesarias para el modelo de red neuronal previamente entrenado, así como el orden de cada una de estas características.

- **ANALITICA_SE_COMPONE_CARACANALIS**: contiene por cada analítica dada de alta en el sistema, la lista de valores que forman los parámetros de una analítica.

- **PACIENTE_POSEE_CARACDEMOG**: almacena los valores de las diferentes características demográficas de los pacientes.

- **MEDICO_TRATA_PACIENTE**: contiene la información de cada consulta llevada a cabo en cada uno de los centros dados de alta en el sistema.

- **MEDICO_TRABAJA_CENTRO_MEDICO**: almacena la información correspondiente a la vida laboral de los médicos. En este primer prototipo se ha supuesto que un médico puede estar actualmente activo únicamente en un centro médico. Esto se lleva a cabo indicando en el campo *fecha_fin* el valor '31/12/9999'. Este valor es muy utilizado en entornos de BI.

Como se mostró en el apartado 5.2.1.3. Limpieza y preparación del dataset, el sistema de recogida de información, así como las comprobaciones y/o restricciones que se establezcan, juegan un papel fundamental de cara a mantener unos datos limpios y coherentes. En el diseño de la base de datos esto se ha tenido en cuenta, por ello, se han definido aquellos campos obligatorios que, sí o sí deben serlo, de cara a evitar este tipo de casuísticas y prevenir por tanto largas y costosas fases de limpieza de datos en la creación de nuevos modelos.

5.2.3.3 Objetos de base de datos

En este último apartado del diseño de la base de datos, se muestran todos los objetos creados en la misma: tablas, vistas, secuencias y triggers.

5.2.3.3.1. Tablas

En base al diseño relacional presentado en el apartado 5.2.3.2. Modelo relacional, en este apartado, se muestra la imagen del correspondiente paso a tablas en la base de datos.

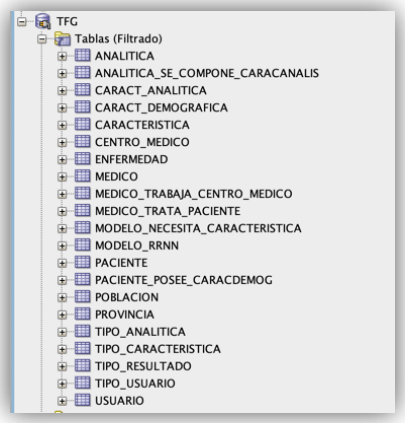


Ilustración 163: Conjunto de tablas creadas en base de datos

5.2.3.3.2. Vistas

Una vista es la representación de una tabla lógica que se basa en una/s tabla/s u otra/s vista/s.

El uso de vistas permite varias ventajas, algunas de ellas son:

- Restringir el acceso (tanto a un nivel de tabla como de columnas).
- Realizar consultas complejas de una manera más fácil.
- Obtener una independencia de los datos.
- Presentar diferentes vistas de los mismos datos.

En este caso se ha creado la vista **V_CARACT_NECESARIAS_MODELO**. El principal motivo de dicha creación ha sido para facilitar la lectura en algunas consultas complejas, en concreto en la obtención de los datos de los pacientes en función de las características que necesita el modelo a predecir.

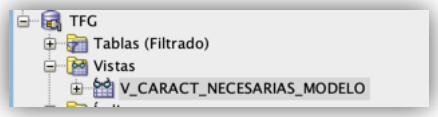


Ilustración 164: Vista creada en base de datos

5.2.3.3.3. Triggers

Un trigger es un trozo de código que se ejecuta automáticamente cuando ocurre algún evento, como, por ejemplo: una inserción, actualización o borrado sobre una tabla o vista.

Para mantener la integridad de las dos generalizaciones comentadas en el apartado 5.2.3.1. Modelo entidad-relación, se crearon dos triggers. Estos triggers están asociados a las tablas CARACTERISTICA y USUARIO. De manera que cuando se produce una inserción en alguna de estas, automáticamente se realiza una inserción en su correspondiente tabla hija.

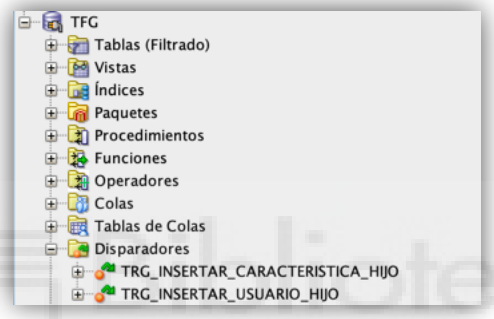


Ilustración 165: Triggers creados en base de datos

5.2.3.3.4. Secuencias

La finalidad de una secuencia es proporcionar una lista de números consecutivos.

Para la asignación de claves únicas a tablas, se han creado 14 secuencias, con 1 como valor de inicio y 9999999 como valor final. De esta forma, en la inserción de nuevos registros se invocan estas secuencias, donde se les va asignando de forma automática el valor de las claves primarias.

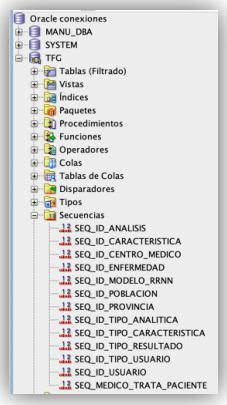


Ilustración 166: Secuencias creadas en base de datos

5.2.4. API Rest

5.2.4.1. Descripción y diseño de las funciones de la API

En este último bloque de desarrollos llevados a cabo en el proyecto, se describe el diseño y puesta en ejecución de la API Rest, que es la encargada de operar con la base de datos y el/los modelo/s, proporcionando los datos a la aplicación móvil vía petición http. Esta API, nace con el concepto de ser una pieza o engranaje de todos y cada uno de los desarrollos expuestos.

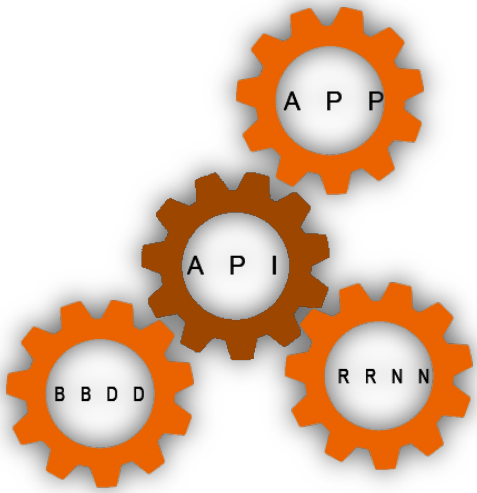


Ilustración 167: Ecosistema del proyecto desarrollado

Para el desarrollo de la API en esta primera versión de la aplicación móvil y con el objetivo de cubrir las necesidades planteadas en el apartado 5.2.2.1. Análisis de requisitos, se han implantado siete rutas accesibles por http por medio de métodos *get*.

- /fichaclinica
- /médico
- /pacientes
- /estadísticas
- /valores_analítica
- /analíticas
- /login

La librería *Flask* utilizada para la creación de la API en python, ofrece una gran rapidez y facilidad a la hora de crear las rutas. Simplemente, de la forma que sigue a continuación, donde dentro de la propia función se recoge la lógica de cada ruta.

```
@app.route('/fichaclinica')
def fichaClinica():
```

Ilustración 168: Ejemplo de la creación de una ruta en Flask

Todas y cada una de las rutas (funciones) desarrolladas de la API, siguen el siguiente flujo genérico de ejecución:

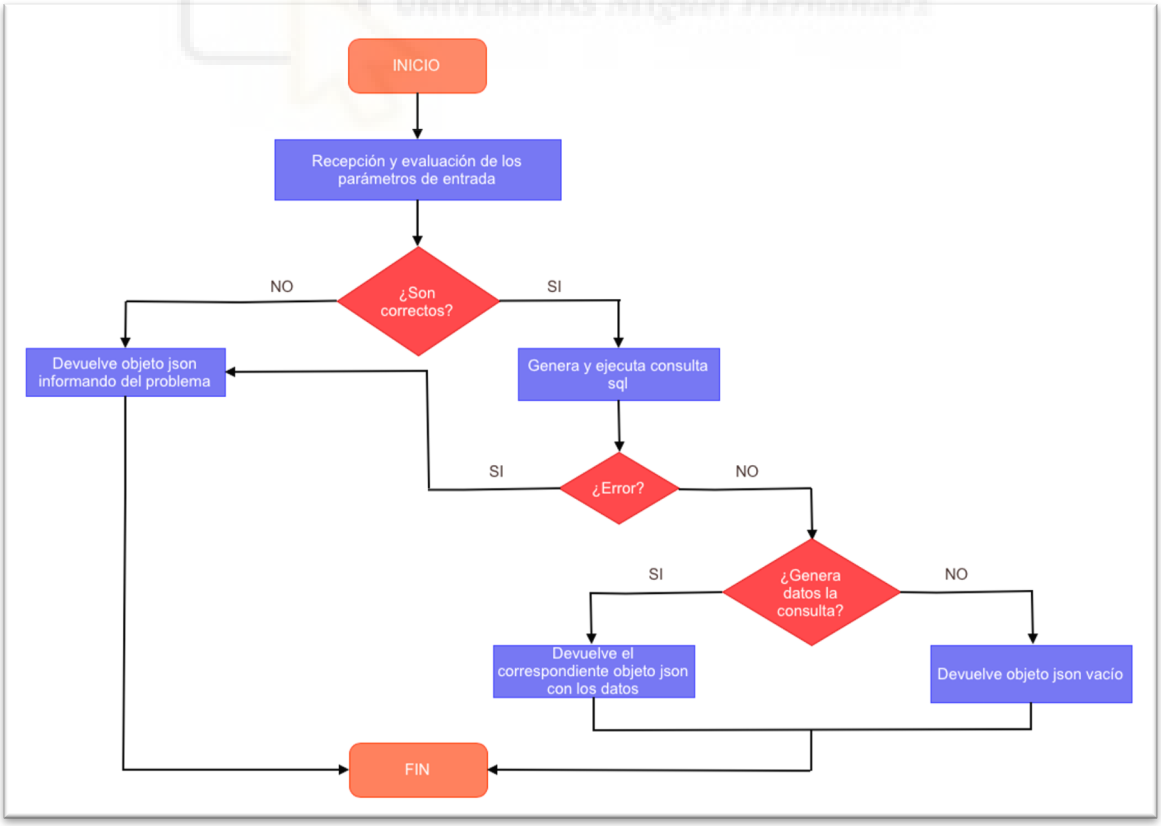


Ilustración 169: Diagrama de flujo genérico de las funciones de la API implementada

A continuación, se indican las rutas, así como sus correspondientes características:

RUTA ACCESO	PARÁMETROS	DESCRIPCIÓN
/fichaclinica	Id del paciente	Devuelve los datos (personales y médicos) del paciente pasado por parámetro.

Tabla 25: Descripción de la ruta /fichaclinica de la API

RUTA ACCESO	PARÁMETROS	DESCRIPCIÓN
/médico	Id del médico	Devuelve los datos del médico pasado por parámetro.

Tabla 26: Descripción de la ruta /médico de la API

Tabla 27: Ejemplo de datos devueltos por la ruta /médico de la API

RUTA	PARÁMETROS	DESCRIPCIÓN
/estadísticas	Id del paciente	Dado el id del paciente, esta función invoca a los determinados modelos configurados en el sistema y devuelve la predicción calculada por cada uno de ellos.

Tabla 28: Descripción de la ruta /estadísticas de la API

Ilustración 170: Ejemplo de datos devueltos por la ruta /estadísticas de la API

RUTA	PARÁMETROS	DESCRIPCIÓN
/valores_analítica	Id de analítica	Dado el id de la analítica, esta función devuelve los parámetros que la forman.

Tabla 29: Descripción de la ruta /valores_analitica de la API

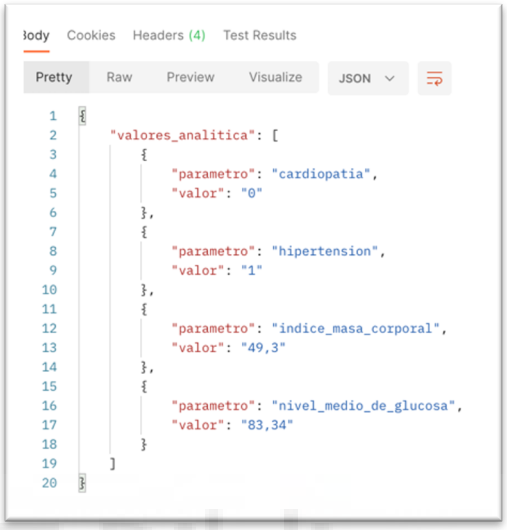


Ilustración 171: Ejemplo de datos devueltos por la ruta /valores_analítica de la API

RUTA	PARÁMETROS	DESCRIPCIÓN
/analíticas	Id del paciente	Dado el id de un paciente, esta función devuelve las analíticas que se ha realizado el paciente ordenado por fecha más reciente.

Tabla 30: Descripción de la ruta /analíticas de la API

Ilustración 172: Ejemplo de datos devueltos por la ruta /analíticas de la API

RUTA	PARÁMETROS	DESCRIPCIÓN
/pacientes	Id del médico, filtro [opcional]	Esta función, si recibe la variable filtro construye una consulta donde busca coincidencias de los pacientes (es invocada desde la pantalla de búsqueda de pacientes) por nombre, apellidos o SIP. Si, por el contrario, se invoca a la función sin la variable filtro, esta devuelve todos los pacientes del médico cuyo id coincida con el pasado por parámetro.

Tabla 31: Descripción de la ruta /pacientes de la API

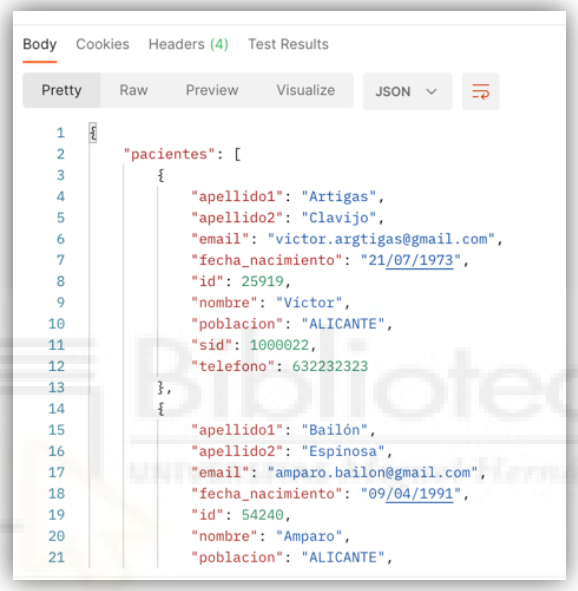


Ilustración 173: Ejemplo de datos devueltos por la ruta /pacientes de la API

RUTA	PARÁMETROS	DESCRIPCIÓN
/login	nickname, password	Dadas las credenciales recogidas, esta función valida que el usuario que intenta hacer login en el sistema existe.

Tabla 32: Descripción de la ruta /login de la API

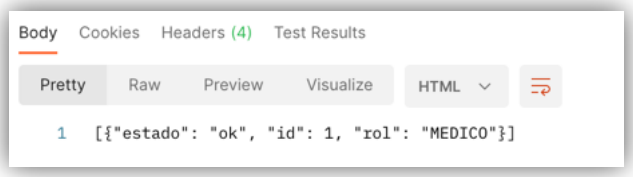


Ilustración 174: Ejemplo de datos devueltos por la ruta /login de la API

Fichero de configuración

En ocasiones, puede ocurrir que la IP de un servidor cambie, ya sea de forma temporal por una actualización u otro tipo de cambio (donde en ese caso se apuntaría a otra máquina de respaldo, si la hubiera, para no dejar el servicio inhabilitado), o, de forma permanente.

Para tener centralizadas todas las credenciales de acceso a los servidores (en este caso: servidor de base de datos en remoto y servidor local donde se ejecuta el web service) y agilizar estos cambios, se ha creado un fichero de configuración (**config.json**) donde se recogen todos los parámetros necesarios para el web service, de esta forma si por ejemplo, se necesitara que este apunte a otra ip y/o puerto, simplemente habría que parar el web service, hacer los correspondientes cambios en el fichero de configuración y volver a arrancarlo. Todo ello sin la necesidad de tocar ni una línea de código.

Este fichero de configuración creado sigue una estructura de tipo json, por lo que su mantenimiento es muy fácil y rápido. Se muestra el formato del fichero en cuestión:

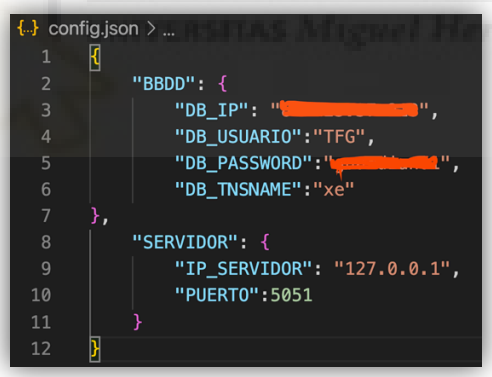


Ilustración 175: Estructura del fichero de configuración de la API

Comunicación con la Base de datos

Para la comunicación con la base de datos, se ha creado un script (**bbdd.py**) que incorpora una clase. Dicha clase está formada por tres métodos:

- **init**: este método, haciendo uso de los parámetros de BBDD contenidos en el fichero de configuración (**config.json**) establece la conexión con la base de datos. Este método (constructor) se llama de forma implícita en el momento que se cree el objeto.

- **select**: este método, recibe como parámetro una sentencia *select* y la ejecuta contra la base de datos.
- **Cerrar_Conexion**: recibe como parámetro la conexión iniciada en el método **init**, y establece el cierre de la comunicación con ella.

De esta forma simplemente creando el correspondiente objeto de la clase, se pueden lanzar las correspondientes consultas que se necesiten.

5.1.4.2. Puesta en ejecución

Una vez desarrollado el web service en su conjunto, llega el momento de ponerlo en ejecución, quedándose a la esperar de recibir peticiones. La forma para ponerla en ejecución es a través de la línea de comandos.

Se indican los pasos para su puesta en ejecución:

Paso 1: ubicarnos en el entorno 'tfg' de conda

Dado que la API hará uso de los modelos de redes neuronales (en este caso, el que predice los infartos cerebrales) es necesario forzar la ejecución de la API en el entorno 'tfg' de conda, para que pueda hacer uso de las librerías que se instalaron para tal fin: pandas, TensorFlow, Keras, etc.

Ilustración 176: comando para activar un entorno de conda

Paso 2: ejecución web service

Una vez se está ubicado en el entorno, se pone en ejecución el web service.

Ilustración 177: Puesta en ejecución de la API

Como se puede ver en la ilustración 177, el servicio está activo en la ip de localhost (127.0.0.1) a través del puerto 5051 que se definió en el diseño.

5.3. Integración de la infraestructura

Una vez presentadas las distintas partes que forman el proyecto, en este apartado, se muestra una representación gráfica de cómo estas se relacionan entre sí, formando la infraestructura que representa el producto final.

Ilustración 178: Infraestructura del sistema implementado

CAPÍTULO 6: CONCLUSIONES Y TRABAJOS FUTUROS

6.1. Conclusiones

A nivel personal este proyecto ha supuesto varios retos. Por un lado, aprender el funcionamiento interno de las redes neuronales, así como el trabajar con dos de las librerías más utilizadas en el ámbito profesional del *machine learning*, como son TensorFlow y Keras. Por otro lado, la adquisición de conocimiento en lenguajes de programación, como son Python y Dart. Prueba de ello, es que el **31%** del tiempo empleado en este proyecto ha ido destinado a una fase de estudio.

La impresión de trabajar con Python en general ha sido muy positiva, es un lenguaje muy intuitivo y sobre todo muy potente. Respecto a Dart, al principio resultó extraño, ya que se hace raro ver en un único lenguaje la mezcla de distintos lenguajes y tecnologías conocidas de antemano. Pero al final, acabó pareciendo interesante y fácil de entender.

Cuando se comenzó a diseñar el sistema presentado en este trabajo de fin de grado, se tuvo dudas de si hacer el desarrollo de la aplicación móvil en Ionic o Flutter. Pero, dado que, con cualquiera de las dos opciones, se partía de cero, se decidió apostar por Flutter, principalmente por el "boom" que estaba experimentando este *SDK*, ya que diversas empresas punteras en diferentes sectores estaban migrando todas sus aplicaciones a Flutter. Lo que se llevó a pensar que sería una puesta de futuro segura y la mejor elección. Una de las mejores características que se ha comprobado que tiene Flutter, es la rapidez que ofrece a la hora de desarrollar, algo que ha sido clave de cara hacer frente a las restricciones de tiempo con las que se afrontaba el proyecto.

En relación con el modelo de red neuronal presentado en el trabajo y, tal y como se vio en los diferentes entrenamientos realizados de la misma, los mejores resultados se obtuvieron aplicando el método de PCA. Pero, esto generó un problema, ya que, el uso de este método conllevaba el no poder utilizar el modelo en un único paciente, debido a que PCA necesita de un conjunto de datos. En su lugar, se buscaron otras alternativas. La primera alternativa que se planteó fue la de guardar las predicciones en la base de datos, en una tabla que relacionase el paciente con la enfermedad y realizar simplemente consultas sobre ella. Pero, esto se descartó debido al alto coste que llevaría mantener la integridad de los datos ante cualquier cambio en alguna de las características de los modelos, esto sin contar que por cada cambio se tendría que realizar el cálculo de predicciones de forma conjunta, dejando temporalmente inaccesible los datos. Finalmente, se optó por realizar un estudio del desempeño del

modelo con las distintas características del dataset, donde aquí se vio que, a la hora de escalar los datos, el que mejores resultados daba era la función *StandarScaler*. Pero aquí se tuvo prácticamente el mismo problema que con PCA y es que esta función se aplica a nivel de columnas, por lo que no es posible aplicarla a un único registro (paciente). Al final, se aplicó la función *Normalizer* donde esta no escala, si no que normaliza los datos. Aquí se apreció que los resultados no eran tan buenos como con el escalado, pero sí permitía aplicarlo a un único paciente. Con esto se dedujo que el escalado funciona mejor para problemas de clasificación y la normalización para problemas de regresión.

Una segunda alternativa que se planteó pero que, por problemas de tiempo no fue posible abordar, fue programar manualmente una función que escalase los datos, es decir, sin usar ninguna librería. El uso de librerías otorga rapidez en el desarrollo, pero en la mayoría de los casos (como este) el alto grado de encapsulamiento que tienen hace que se esté muy limitado.

La escasez de registros del dataset fue el mayor problema, ya que resultó ser el principal impedimento para poder conseguir una mejor precisión en las predicciones. Con un dataset más grande se podrían haber probado otras alternativas como, por ejemplo, el uso de *validaciones cruzadas*, que es una técnica muy utilizada, que, mediante fuerza bruta, valida el modelo a través de subconjunto de datos, obteniendo así los mejores parámetros para ese modelo.

En términos generales, la infraestructura del sistema presentado en este trabajo cumple con los objetivos que se marcaron y expusieron en este proyecto. En primer lugar, se elaboró un modelo de red neuronal capaz de realizar a voluntad predicciones sobre pacientes de forma individualizada, con el fin de calcular la probabilidad de desarrollar un infarto cerebral. En segundo lugar, se desarrolló una base de datos en Oracle, para almacenar tanto toda la parte médica como los modelos y enfermedades dadas de alta en el sistema. En tercer lugar, se desarrolló una aplicación multiplataforma intuitiva y sencilla de utilizar por personal sanitario y no técnico, con el fin de visualizar todos y cada uno de los datos médicos de los pacientes. En cuarto lugar, se desarrolló una API que, haciendo de intermediaria entre los modelos definidos en el sistema y la base de datos, permitiera ofrecer la información necesaria a la aplicación móvil. Y en quinto y último lugar, es que el sistema desarrollado en su conjunto es multimodelo y multienfermedad, por lo que permite albergar diversos modelos con el fin de obtener predicciones de diferentes enfermedades.

Con la elaboración de este proyecto y, aún quedando mucho camino por recorrer, se confirma la gran potencia que tiene el uso de la inteligencia artificial en el mundo de la medicina.

Con este proyecto se deja la puerta abierta para la continuación de un sistema que puede ser de gran ayuda en el ámbito de la salud, concretamente para un acertado y pronto diagnóstico médico.

6.2. Trabajos futuros

En un principio, existía un acuerdo con un laboratorio para desarrollar una aplicación que permitiera detectar la existencia de enfermedades infecciosas en pacientes. Pero, debido a la saturación que ha provocado en la sanidad la pandemia del COVID-19, no fue posible contar con tal colaboración. Esta sería la primera puesta de futuro, colaborar con algún centro médico para poder desarrollar el sistema en modo 100% funcional, adecuándose a un escenario completamente real.

El desarrollo de este trabajo ha supuesto el asentamiento de unas bases sobre las que poder desarrollar más funcionalidades si se precisa. En la elaboración del proyecto, se han centrado los esfuerzos en establecer una infraestructura que sea escalable, con el fin de facilitar la adición de nuevos modelos y funcionalidades, ya que, debido a las limitaciones de tiempo, no era objetivo el desarrollo de un sistema 100% funcional. Por ello, se dejaron a un lado algunas funcionalidades y mejoras que serán implementadas en el futuro.

A continuación, se listan dichas propuestas:

En relación con la base de datos

- Dar de alta nuevos tipos de usuario con la finalidad de poder contar con distintos roles en la aplicación móvil: administrador y personal de laboratorio.
- Diseñar un nuevo conjunto de tablas para guardar las arquitecturas de las redes neuronales y modelos que se van dando de alta en el sistema. Así como de los cambios que se van realizando, esto generará la posibilidad de poder mantener un histórico de los cambios realizados en cada una de las redes neuronales,

además de su desempeño conseguido con cada uno de ellos, dando también la posibilidad de recuperar la configuración que se quiera.
- Cifrar todas las tablas con el fin de aumentar la seguridad a nivel de base de datos.

En relación con los modelos de redes neuronales

- Implementar el escalado de datos de forma manual sin el uso de librerías, de esta forma se podría conseguir aplicar el escalado a una sola fila sin la limitación que impone la librería *scikit-learn* de realizarlo por columnas.
- Implementar una utilidad que, en base a las estructuras y características necesarias de los modelos que se encuentran guardados en base de datos, genere de forma automática los *scripts* que serán cargados y utilizados por la API para realizar las predicciones, consiguiendo una automatización de todo este proceso.

En relación con la aplicación móvil

Aspectos generales

- Realizar un *testing* más exhaustivo, probando la aplicación en distintos tamaños de pantallas, con el objetivo de hacerla completamente adaptable a la interfaz.
- Implementar la aplicación móvil en distintos idiomas. Aunque esto, hoy en día (septiembre 2021) es una limitación de Flutter, ya que no incorpora de una manera sencilla y rápida la implementación del multilenguaje, pero es solo cuestión de tiempo.
- Implementar el reseteo de la contraseña.
- Adaptar la aplicación móvil para que cumpla el Real Decreto 1112/2018 publicado el 7 de septiembre sobre la accesibilidad de los sitios web y aplicaciones para dispositivos móviles en el sector público.
- Permitir el acceso a la aplicación móvil a diferentes tipos de usuario: pacientes, administradores y personal de laboratorio. Donde poder realizar diferentes funciones:

Aspectos por rol de usuario

- **Paciente**
  - Visualización y modificación de sus datos personales.
  - Acceso a la visualización de sus analíticas.
  - Solicitud de citas médicas.

- **Administrador**
  - Visualización y modificación de sus datos personales.
  - Generación de una nueva zona en la aplicación para que los usuarios con rol de administrador puedan diseñar, entrenar y generar modelos de una forma gráfica, consiguiendo de esta manera que la incorporación de nuevos modelos al sistema sea mucho más sencilla y rápida.
  - Generación de una nueva zona de administración desde donde se pueda dar de alta nuevos usuarios, centros médicos y cualquier otro contenido necesario para el funcionamiento del sistema.

- **Personal de laboratorio**
  - Visualización y modificación de sus datos personales.
  - Añadir y/o modificar analíticas al sistema.

- **Médico (añadir más funcionalidad a la zona de estadísticas)**
  - En esta primera versión de la aplicación, se obtiene la probabilidad de que un paciente desarrolle una enfermedad en base a su última analítica. En esta mejora de futuro, se pretende dotar a la aplicación de una nueva funcionalidad que permita visualizar la tendencia (evolución) que experimenta el paciente, en base a todas las analíticas que se haya realizado.
  - Generación de gráficas en base a determinados filtros como, por ejemplo: enfermedad, sexo, edad, etc.
  - Posibilidad de que, tras el inicio de sesión de un médico en la aplicación móvil, este pueda elegir el centro de salud asociado para el que iniciar sesión.

En relación con la API

La seguridad es un aspecto importantísimo en el desarrollo de un producto software, sobre todo cuando hay un intercambio de información donde podría ser susceptible a su intercepción. Para mejorar la seguridad en la API se implementarán las siguientes mejoras:

- Toda la información que viaje desde y hacia la API irá cifrada.
- Implementación de tokens donde en cada petición con la API esta valide la autenticidad del peticionario.

CAPÍTULO 7: BIBLIOGRAFÍA

[1] Página oficial de la organización mundial de la salud – *Como define la OMS la salud* (Consultado en agosto 2021). https://www.who.int/es/about/frequently-asked-questions

[2] Anna Surroca Gibert – Microbióloga y comunicadora científica – *Los 15 tipos de enfermedades, características, síntomas y causas*. (Consultado en agosto 2021). https://medicoplus.com/medicina-general/tipos-enfermedades

[3] Página oficial de MedlinePlus, producido por la Biblioteca Nacional de Medicina de los EE. UU, la biblioteca médica más grande del mundo. (Consultado en marzo 2021). https://medlineplus.gov/spanish/stroke.html

[4] Pascual Maragall, investigador científico en el ámbito del Alzeimer – *Partes del cerebro y anatomía*. (Consultado en agosto 2021) https://www.cognifit.com/es/partes-del-cerebro

[5] Carlos Santana Vega – Ingeniero informático – Divulgador IA – *Introducción a las redes neuronales* (publicado durante los meses de marzo y abril de 2018). (Consultado durante los meses de febrero y marzo 2021). https://www.youtube.com/watch?v=MRIv2IwFTPg&list=PL-Ogd76BhmcB9OjPucsnc2-piEE96jJDQ

[6] Xabier Basogain Olabe – Escuela superior de Bilbao – *Redes neuronales artificiales y sus aplicaciones*. (Consultado en marzo de 2021). https://ocw.ehu.eus/pluginfile.php/40137/mod_resource/content/1/redes_neuro/contenidos/pdf/libro-del-curso.pdf

[7] Estefanía Freire y Silvia Saharí – publicado el 14 noviembre 2019 (Consultado en marzo de 2021). https://bootcampai.medium.com/redes-neuronales-13349dd1a5bb

[8] Miguel Sotaquirá – Ingeniero Electrónico y Doctor en Bioingeniería (publicado en diciembre de 2018. (Consultado en marzo de 2021). https://www.codificandobits.com/blog/funcion-de-activacion/

[9] José Martínez Heras – *Error cuadrático medio para regresión*. Publicado el 10 de octubre de 2020. (Consultado en marzo de 2021). https://www.iartificial.net/error-cuadratico-medio-para-regresion/

[10] Lidgi González – *Evaluando el error en los modelos de regresión*. Publicado en junio de 2018. (Consultado en abril de 2021). https://aprendeia.com/evaluando-el-error-en-los-modelos-de-regresion/

[11] Portal de divulgación científica sitiobigdata – *Funciones comunes de pérdida en el aprendizaje automático*. Publicado el 24 diciembre 2019. (Consultado en abril de 2021). https://sitiobigdata.com/2019/12/24/funciones-comunes-de-perdida-en-el-aprendizaje-automatico/

[12] Raúl E. López Briega – *Introducción al Deep learning*. Publicado el 13 de junio de 2017. (Consultado en abril de 2021). https://relopezbriega.github.io/blog/2017/06/13/introduccion-al-deep-learning/

[13] InteractiveChaos, portal de recursos educativos – *Machine learning y backpropagation*. (Consultado en abril de 2021). https://interactivechaos.com/es/manual/tutorial-de-machine-learning/backpropagation

[14] Arkaitz Bidaurragaza Barrueta – *Estudio de diferentes modelos de redes neuronales para el desarrollo de un clasificador de frases*. Publicado el 19 de junio de 2019. (Consultado en mayo de 2021). https://core.ac.uk/download/pdf/326226202.pdf

[15] Alberto Rubiales, Científico de datos – *¿Qué es Underfitting y Overfitting?* Publicado el 26 de junio de 2020. (Consultado en mayo de 2021). https://rubialesalberto.medium.com/qué-es-underfitting-y-overfitting-c73d51ffd3f9

[16] Diego Calvo – *Clasificación de redes neuronales*. Publicado el 13 de Julio de 2017. (Consultado en mayo de 2021). https://www.diegocalvo.es/clasificacion-de-redes-neuronales-artificiales/

[17] José Díaz Novas, Bárbara Gallego Machado y Aracely León Gonzáles – *El diagnóstico médico: bases y procedimientos.* (Consultado en agosto de 2021). http://scielo.sld.cu/pdf/mgi/v22n1/mgi07106.pdf

[18] Revista digital de divulgación informativa – *Efectos de la inteligencia artificial en la medicina y sus aplicaciones más novedosas*. Publicado el 23 de febrero de 2021. (Consultado en agosto de 2021). https://www.apd.es/aplicaciones-inteligencia-artificial-en-medicina/

[19] techedge, Partner de transformación digital – *Inteligencia artificial y aplicaciones en salud*. Publicado el 03 de marzo de 2021. (Consultado en agosto de 2021). https://www.techedgegroup.com/es/blog/inteligencia-artificial-aplicaciones-en-salud

[20] Página oficial de Visual Studio Code. (Consultado en marzo de 2021). https://code.visualstudio.com/docs

[21] Página oficial de Cacoo. (Consultado en julio de 2021). https://cacoo.com/es/

[22] Página oficial de git. (Consultado en marzo de 2021). https://git-scm.com

[23] Aula21, división de formación en ingeniería industrial – *Python: qué es, para qué sirve y cómo se programa.* (Consultado en agosto de 2021). https://www.cursosaula21.com/que-es-python/

[24] Covantec - *Programación en Python – Nivel básico.* (Consultado en agosto de 2021). https://entrenamiento-python-basico.readthedocs.io/es/latest/leccion1/caracteristicas.html

[25] Página oficial de Jupyter. (Consultado en marzo de 2021). https://jupyter.org

[26] Lidgi González de AprendeIA – *Introducción a la librería NumPy de Python*. (Consultado en abril de 2021). https://aprendeia.com/introduccion-a-numpy-python-1/

[27] Alfredo Sánchez Alberca, profesor de matemáticas y estadística de la universidad CEU de San Pablo y Doctor en Inteligencia Artificial – *La librería Pandas*. (Consultado en abril de 2021). https://aprendeconalf.es/docencia/python/manual/pandas/

[28] Alfredo Sánchez Alberca, profesor de matemáticas y estadística de la universidad CEU de San Pablo y Doctor en Inteligencia Artificial – *La librería Matplotlib*. (Consultado en abril de 2021). https://aprendeconalf.es/docencia/python/manual/matplotlib/

[29] Lidgi González de AprendeIA – *Introducción a la librería Scikit-Learn de Python*. (Consultado en abril de 2021). https://aprendeia.com/libreria-scikit-learn-de-python/

[30] Sitio web oficial de TensorFlow. (Consultado en marzo de 2021). https://www.tensorflow.org/?hl=es-419

[31] Lidgi González de AprendeIA – *¿Qué es TensorFlow? ¿Cómo funciona?* (Consultado en marzo de 2021). https://aprendeia.com/que-es-tensorflow-como-funciona/

[32] Sitio web oficial de keras. Consultado en marzo de 2021. https://keras.io

[33] Luis Llamas – *Machine Learning con Tensorflow y Keras en Python*. Publicado el 10 de febrero de 2019. (Consultado en marzo de 2021). https://www.luisllamas.es/machine-learning-con-tensorflow-y-keras-en-python/

[34] Luigys Toro – *Anaconda Distribution: La Suite más completa para la Ciencia de datos con Python*. (Consultado en febrero de 2021). https://blog.desdelinux.net/ciencia-de-datos-con-python/

[35] Sitio web oficial de Red Hat. Consultado en junio de 2021. https://www.redhat.com/es

[36] José Domingo Muñoz – *Qué es Flask*. Publicado el 17 de noviembre de 2017. (Consultado en junio de 2021). https://openwebinars.net/blog/que-es-flask/

[37] Sitio web oficial de Oracle. (Consultado en junio de 2021). https://oracle.github.io/python-cx_Oracle/

[38] Sitio web oficial del módulo cx_Oracle. (Consultado en junio de 2021). https://cx-oracle.readthedocs.io/en/latest/user_guide/installation.html

[39] Víctor Diví - *¿Qué es el lenguaje de programación Dart?* (Consultado en julio de 2021). https://inlab.fib.upc.edu/es/blog/que-es-el-lenguaje-de-programacion-dart

[40] Consultora aures tic de Alcoy, Alicante – *Flutter. (Consultado en julio 2021).* https://aurestic.es/que-es-flutter/#queesflutter

[41] Sitio web oficial de adobe. (Consultado en junio de 2021). https://www.adobe.com/es/products/xd.html

[42] Lidgi Gonzalez de AprendeIA – *Conjunto de datos desbalanceados* (Consultado en marzo de 2021). https://aprendeia.com/conjunto-de-datos-desbalanceado/

[43] Ebrahim Mokhtar – *La matriz de correlación de Python*. Publicado el 2 de agosto de 2020. (Consultado en marzo de 2021). https://likegeeks.com/es/matrix-correlacion-python/

[44] Vicente Rodríguez – *Conceptos básicos sobre redes neuronales*. Publicado el 30 de octubre 20218. (Consultado en abril de 2021). https://vincentblog.xyz/posts/conceptos-basicos-sobre-redes-neuronales

[45] Luis Velasco, divulgador de información sobre ciencia de datos e inteligencia artificial – *Primer modelo Deep Learning con TensorFlow*. Publicado el 18 de enero de 2021. (Consultado en abril de 2021). https://www.youtube.com/watch?v=NERPPvoj3Go

[46] Joaquín Amat Rodrigo – *PCA con Python*. Publicado en diciembre de 2020. (Consultado en mayo de 2021). https://www.cienciadedatos.net/documentos/py19-pca-python.html

[47] InteractiveChaos, portal de recursos educativos – *Escalado de datos*. (Consultado en abril de 2021). https://interactivechaos.com/es/manual/tutorial-de-machine-learning/standard-scaler

[48] Jeff Hale – *Scale, Standarize, or Normalize with Scikit-Learn*. Publicado el 4 de marzo de 2019. (Consultado en abril de 2021). https://towardsdatascience.com/scale-standardize-or-normalize-with-scikit-learn-6ccc7d176a02

[49] José María Aguilar - *¿Qué es el patrón MVC en programación y por qué es útil?* Publicado el 15 de octubre de 2019. (Consultado en Julio de 2021). https://www.campusmvp.es/recursos/post/que-es-el-patron-mvc-en-programacion-y-por-que-es-util.aspx

[50] John McGregor – A Practical Guide to Testing Object-Oriented Software, Addison Wesley 2001. (Consultado en Julio de 2021). http://index-of.co.uk/Software-Engineering/practical_guide_testing_object_oriented_software.pdf

ANEXO I

Manual de usuario app Seek Health

1. Introducción

Bienvenido a Seek Health, aplicación para la ayuda en el diagnóstico médico.

En este manual encontrará las referencias sobre todo lo que puede realizar en esta primera versión de la aplicación.

A medida que vaya avanzando en la aplicación, se irá mostrando un menú donde podrá interactuar con las distintas zonas de esta. Todas estas opciones se explican en detalle en este manual.

La aplicación cuenta con botones (en forma de flecha) para poder volver a la pantalla anterior, aunque si lo prefiere (y su dispositivo lo dispone) puede hacer uso de el sin problemas.

2. Bienvenida

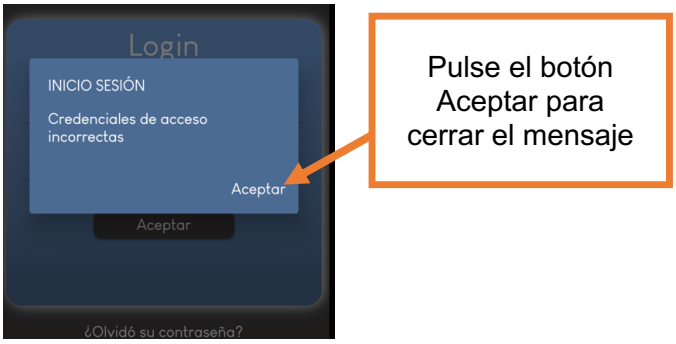
La primera vez que se abre la aplicación muestra una pantalla de bienvenida a la aplicación, donde deslizando hacia arriba se accede a la siguiente pantalla.

3. Iniciar sesión

Si no dispone de una cuenta debe contactar con el administrador de su centro médico para que le habilite una. Si, dispone de una cuenta, pero no recuerda su contraseña de acceso puede hacer uso del botón “¿Olvidó su contraseña?”, donde le aparecerá un cuadro para que introduzca su usuario. Tras aceptar, recibirá unas indicaciones para restablecer su contraseña.

Una vez dispone de sus credenciales puede insertar su nombre de usuario y contraseña y pulsar sobre el botón de “Aceptar”. Es importante tener en cuenta que ambos campos (usuario y contraseña) son **obligatorios** por lo que en caso de no introducir alguno de ellos el sistema le informará de ello.

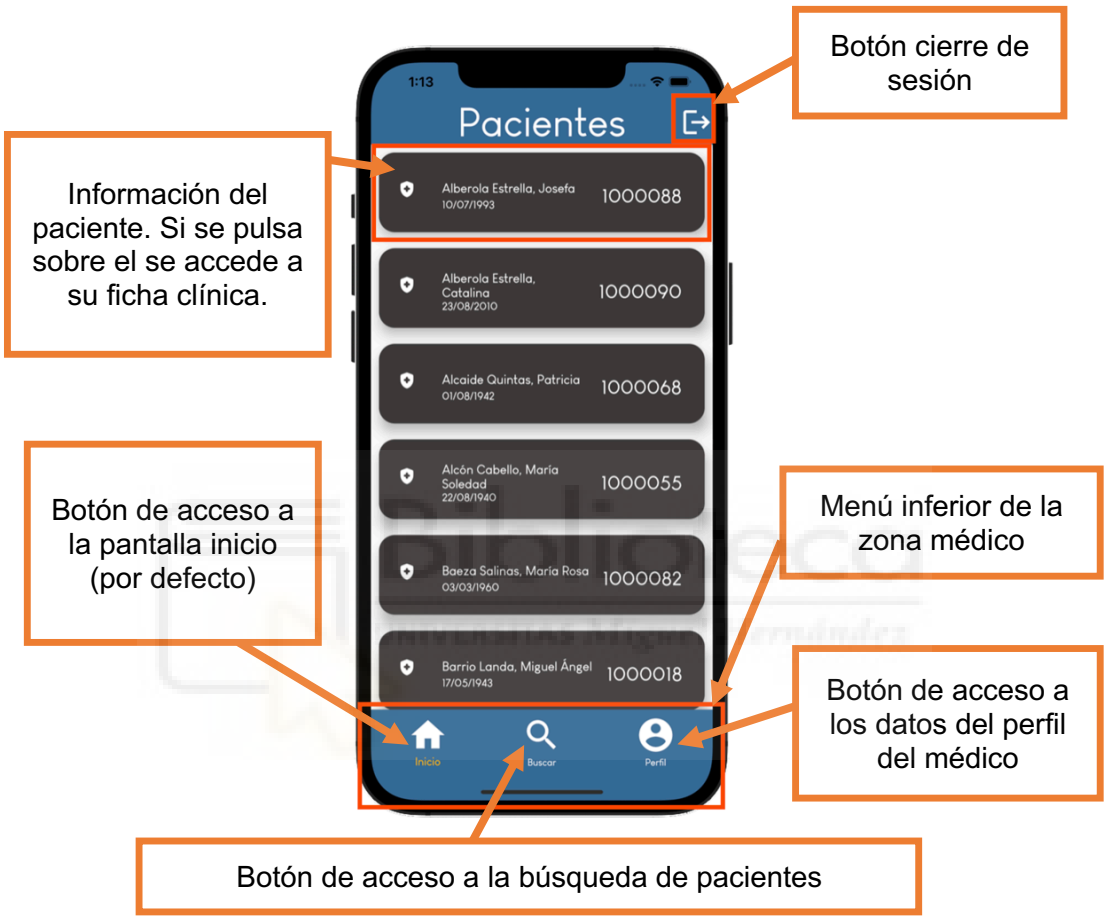
Si tras pulsar el botón de “Aceptar” el sistema detecta que los datos introducidos no son correctos, recibirá el siguiente mensaje y deberá introducir de nuevo las credenciales:



Si, por el contrario, las credenciales introducidas son correctas se le concederá acceso a la aplicación donde se accede a la zona médico.

4. Zona médico

Desde esta zona puede ver un listado de sus pacientes (pantalla inicio por defecto), solicitar la búsqueda de pacientes y visualizar los datos de perfil del médico logueado. Adicionalmente, se da la opción en todo momento de realizar el cierre de sesión de la aplicación.



5. Búsqueda de pacientes

Desde esta pantalla podrá realizar la búsqueda de pacientes. Introduciendo datos en el cuadro de búsqueda el sistema buscará coincidencias por:

- Nombre
- Apellidos
- SIP

6. Perfil

Cuando el médico desea ver su perfil, este se le muestra dividido en dos partes. Por un lado, sus datos personales (tales como nombre, apellidos, teléfono, etc) y datos médicos como, por ejemplo: número de pacientes, centro de salud asociado, etc. Adicionalmente, se muestra un botón “Editar”. Este botón permitirá habilitar un formulario para modificar los datos, pero de momento para esta primera versión de la aplicación no está habilitado.

7. Zona Paciente

Tras pulsar sobre un paciente (bien sea desde la pantalla de inicio de la zona médico o, desde el propio resultado de la búsqueda de pacientes), se nos abrirá la zona del paciente, donde su vista principal es su ficha clínica. Esta zona de paciente dispone de un nuevo menú inferior en el que podemos ir hacia:

- Ficha paciente (pantalla por defecto)
- Analíticas
- Estadísticas

8. Analíticas de un paciente

A esta pantalla se accede desde la ficha del paciente, en ella se muestra el listado de las analíticas que se ha realizado un paciente, ordenadas por fecha más reciente. Para cada una de las analíticas que forman el listado, se muestra: la fecha de realización, tipo de analítica y una referencia al resultado de esta.

Si se quiere visualizar cada uno de los parámetros que forman la analítica se debe pulsar sobre ella. Esto abrirá una nueva pantalla donde se mostrarán los datos.

9. Estadísticas de un paciente

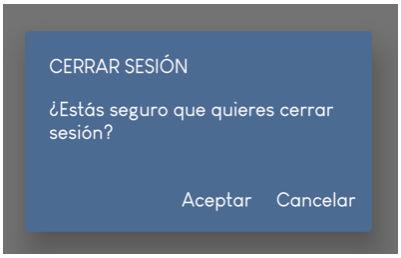
Esta pantalla es una de más interesantes de la aplicación. Al igual que la pantalla de analíticas, a esta también se accede desde el menú inferior de la zona de paciente.

El baremo que sigue la aplicación para mostrar de un color u otro la probabilidad de las enfermedades es el siguiente:

10. Cerrar sesión

A excepción de la zona de búsqueda de pacientes, el cierre de sesión está disponible desde cualquier pantalla de la aplicación a través del botón , situado en la esquina superior derecha del menú.

Tras pulsar este botón, la aplicación pide confirmación para el cierre de sesión.



Desde este cuadro, nos da dos opciones “Aceptar” o “Cancelar”. Si se pulsa cancelar, el cuadro se cierra (sin alterar la pantalla en la que nos encontremos). En cambio, si se pulsa el botón de “Aceptar” se cerrará la sesión del usuario y nos redirigirá a la pantalla de inicio de sesión.