

Programación de Inteligencia Artificial

Conforme a contenidos del «Curso de Especialización en Inteligencia Artificial y Big Data»



Universidad de Castilla-La Mancha

Escuela Superior de Informática
Ciudad Real

Índice general

1. Lenguajes de programación	1
1.1. Introducción a la programación	1
1.1.1. El proceso de desarrollo de software	2
1.1.2. Aspectos fundamentales de desarrollo	8
1.1.3. Lenguajes de programación compilados e interpretados	12
1.1.4. Resumen	16
1.2. Programación de Inteligencia Artificial	16
1.2.1. Consideraciones previas	17
1.2.2. Características deseables en un lenguaje de programación para IA	18
1.2.3. Caso práctico. Búsqueda de patrones en archivos de texto	28
1.2.4. Resumen	31
1.3. Lenguajes de programación de IA	32
1.3.1. Visión general	32
1.3.2. Python como lenguaje de referencia para IA	33
1.3.3. R: más allá de la estadística	38
1.3.4. Caso práctico en Python: <i>Piedra, Papel o Tijeras!</i>	47
1.3.5. Resumen	59
1.4. Procesamiento de datos, objetos y lenguajes de marcas	60
1.4.1. Los lenguajes de marcas	60
1.4.2. Presentación de la información mediante HTML	62
1.4.3. Estructuración de los datos mediante XML	83

1.4.4. Procesamiento de datos mediante XSLT	86
2. Plataformas de Inteligencia Artificial	97
2.1. Introducción a las Plataformas de IA	98
2.1.1. Computación en la nube para la IA	98
2.1.2. Beneficios de la computación en la nube	100
2.1.3. El concepto de plataforma de IA	103
2.1.4. Desarrollo de aplicaciones de IA en entornos cloud	104
2.1.5. Oportunidades de las plataformas de IA	110
2.1.6. Resumen	111
2.2. Detalles de las Plataformas de IA	112
2.2.1. Visión general	112
2.2.2. Fundamentos básicos	113
2.2.3. Aspectos fundamentales de las plataformas de IA	115
2.2.4. Comparativa	124
2.2.5. Elección de la plataforma	125
2.2.6. Resumen	127
2.3. IA mediante servicios en la nube	128
2.3.1. Consideraciones previas	129
2.3.2. Problemática	131
2.3.3. Arquitectura propuesta	132
2.3.4. Implementación	134
2.3.5. Consideraciones finales	144
2.3.6. Resumen	145
3. Entornos de modelado de IA	147
3.1. Herramientas de modelado	147
3.1.1. Google Cloud AI	150
3.1.2. Azure Machine Learning	157
3.1.3. Amazon SageMaker	174
3.2. Bibliotecas de modelado de IA	193
3.2.1. Configuración del entorno	196
3.2.2. Caso práctico: prediciendo fugas de clientes	200
3.2.3. Caso práctico: serialización y despliegue de modelos	210
3.3. Modelos y algoritmos predefinidos	215

3.3.1.	Inteligencia Artificial	215
3.3.2.	Definición de modelos	220
3.3.3.	Ejemplo de algoritmos de Aprendizaje Automático	222
3.3.4.	Entornos de modelado para Inteligencia Artificial	228
3.4.	Recolección de datos	231
3.4.1.	Visión general	231
3.4.2.	El tamaño y la calidad del <i>dataset</i>	232
3.4.3.	Combinando fuentes de datos	234
3.4.4.	Etiquetado	235
3.4.5.	Muestreo de datos	236
3.4.6.	Datos no balanceados	236
3.4.7.	División de los datos	238
3.4.8.	Caso práctico: construcción de un <i>dataset</i>	238
3.5.	Manipulación de datos	247
3.5.1.	Visión general	247
3.5.2.	Transformación de datos numéricos	249
3.5.3.	Transformación de datos categóricos	252
3.5.4.	Caso práctico: entrenamiento de modelo de regresión lineal	254
3.6.	Evaluación de Resultados	266
3.6.1.	Introducción	266
3.6.2.	Validación cruzada	268
3.6.3.	Evaluación de modelos de clasificación	270
3.6.4.	Ejemplo de análisis completo: clasificación de cáncer de mama como maligno o benigno	281
3.7.	Modelado de redes neuronales	289
3.7.1.	Fundamentos básicos	289
3.7.2.	Estructura de una red neuronal	290
3.7.3.	Topologías de redes neuronales	294
3.7.4.	Proceso de aprendizaje de una red neuronal	295
3.7.5.	Caso práctico: implementación de una red neuronal para clasificar flores	299
3.7.6.	Aplicaciones generales	307
3.7.7.	Resumen	308
3.8.	Modelado avanzado de redes neuronales	309
3.8.1.	Técnicas avanzadas de aprendizaje	309

3.8.2. Módulos predefinidos	316
3.8.3. Caso práctico: entrenando una red neuronal con <i>Tensorflow</i>	317
3.8.4. Resumen	325
3.9. Herramientas de generación de código para crear IA	327
3.9.1. Visión general	327
3.9.2. Beneficios del enfoque “IA sin código”	328
3.9.3. Herramientas de generación de IA “sin código”	330
3.9.4. Ejemplo: detección de rostros con mascarillas usando Lobe	331
3.9.5. Oportunidades de la generación de “IA sin código”	338
3.9.6. Resumen	339
4. Convergencia Tecnológica	341
4.1. Introducción a la convergencia tecnológica	341
4.2. Visión general	342
4.3. Ventajas de la convergencia tecnológica	346
4.4. Plataformas para la conexión tecnológica	347
4.4.1. Power Platform	349
4.5. Análisis de Power Automate	351
4.5.1. Ejemplo 1. Hola Mundo y uso de plantillas	352
4.5.2. Ejemplo 2. Creación de flujo automatizado (Creación de Archivo a Excel)	354
4.5.3. Ejemplo 3. Creación de flujo programado y utilización de variables de flujo	355
4.5.4. Ejemplo 4. Creación de flujo de escritorio	356
4.5.5. Ejemplo de llamado mediante API REST	358
4.5.6. Ejemplo 5. Uso de condicionales	359
4.5.7. Ejemplo 6. Captura de eventos de escritorio	359
4.5.8. Ejemplo 7. Procesamiento por lotes	362
4.5.9. Ejemplo 8. Operaciones cognitivas	364
4.6. AI Builder	369
4.6.1. Ejemplo 1. Uso de modelos predefinidos	371
4.6.2. Ejemplo 2. Extracción de información de documentos	373
4.6.3. Ejemplo 3. Detección de objetos	376
4.7. Sistemas Blockchain	377
4.7.1. Dificultades de los sistemas descentralizados	377

4.7.2. El concepto de bloque	379
4.7.3. Qué es el <i>Proof of Work</i> y ajustes	380
4.7.4. Seguridad en Blockchain	382
4.7.5. ¡Blockchain es una estructura de datos descentralizada! . .	383
4.8. Introducción a Ethereum	384
4.8.1. Ethereum: Un blockchain de propósito general	386
4.8.2. Aplicaciones Descentralizadas (DApps)	388
4.8.3. Desarrollando en Solidity	388
4.8.4. Siguientes pasos	401
4.9. Conexión entre tecnologías	403
4.10. Agentes conversacionales	404
4.10.1. Introducción a Power Virtual Agents	404
4.10.2. Temas, Variables, Entidades y Control de Flujo	405
4.10.3. Un ejemplo de conexión con Power Automate	406
4.10.4. Publicación del Agente	408
4.11. OpenAI	409
4.11.1. Motores y modelos	410
4.11.2. Ejemplo de uso en Python	412
4.12. Sistemas IoT	415
4.13. Seguridad en la convergencia tecnológica	417
5. Evaluación de modelos	419
5.1. Estrategias corporativas	419
5.1.1. Visión general	419
5.1.2. Productos inteligentes	422
5.1.3. Servicios inteligentes	426
5.1.4. Procesos de negocio inteligentes	431
5.1.5. Reflexión	433
5.2. Modelos de negocio	433
5.2.1. Introducción	433
5.2.2. Reinvención del modelo de negocio	433
5.2.3. Caso práctico: rehabilitación remota de ejercicios físicos basada en IA	435
5.2.4. Caso práctico: MediaPipe Pose con Python	448
5.3. Modelos de automatización	455

5.3.1.	El concepto de <i>automatización inteligente</i>	456
5.3.2.	Caso práctico: reconocimiento automático de matrículas .	457
5.4.	Gestión de recursos y activos	472
5.4.1.	Adoptando IA en la empresa	473
5.4.2.	Necesidades humanas y culturales	474
5.4.3.	Inteligencia colaborativa entre humanos e IA	477
5.4.4.	Caso práctico: Google App Engine	479

Listado de acrónimos

AIAAS	Artificial Intelligence as a Service
ALPR	Automatic License Plate Recognition
ANPR	Automatic Number Plate Recognition
API	Application Programming Interface
ANN	Artificial Neural Network
AWS	Amazon Web Services
BASIC	Beginner's All-purpose Symbolic Instruction Code
CLI	Command-Line Interface
CPD	Centro de Procesamiento de Datos
CPU	Computer Processing Unit
CPU	Central Processing Unit
DEVOPS	Development and Operations
DTW	Dynamic Time Warping
FANN	Fast Artificial Neural Network
FSF	Free Software Foundation
GAE	Google App Engine
GCP	Google Cloud Platform
GCC	GNU Compiler Collection
GPL	GNU General Public License
GPU	Graphics Processing Unit
GUI	Graphical User Interface
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IAAS	Infrastructure-As-A-Service
IA	Inteligencia Artificial
JSON	JavaScript Object Notation
KISS	Keep it Simple Stupid!
LISP	List Processing

MIT	Massachusetts Institute of Technology
ML	Machine Learning
MLOPS	Machine Learning Operations
MVP	Minimum Viable Product
NHS	National Health Service (UK)
NIST	National Institute of Standards and Technology
NLTK	Natural Language Toolkit
OOV	Out of Vocab
OPENCV	Open Source Computer Vision Library
PAAS	Platform-As-A-Service
SAAP	Service-As-A-Platform
PEP	Python Enhancement Proposal
PLN	Procesamiento del Lenguaje Natural
POO	Programación Orientada a Objetos
PROLOG	Logic Programming
PSF	Python Software Foundation
RAAS	Robot-As-A-Service
RAM	Random Access Memory
RPA	Robotic Process Automation
REST	Representational State Transfer
SAAS	Software-As-A-Service
SRP	Single-Responsibility Principle
SDK	Software Development Kit
SDL	Simple DirectMedia Layer
SQL	Structured Query Language
SSH	Secure Shell
TDD	Test-Driven Development
TI	Tecnologías de la Información
TRL	Technology Readiness Levels
URL	Uniform Resource Locator

1

Capítulo

Lenguajes de programación

David Vallejo Fernández, Santiago Sánchez Sobrino

Este capítulo sirve como punto introductorio a la programación, abordándola desde la perspectiva general del **proceso de desarrollo de software** y aumentando así el alcance que el primer término realmente tiene. El contexto para el cual se irá particularizando, no obstante, es el desarrollo de aplicaciones de IA (Inteligencia Artificial). Adicionalmente, se realiza un recorrido a alto nivel de aspectos fundamentales y buenas prácticas de desarrollo. En dicho recorrido, se discuten los conceptos generales de los lenguajes de programación compilados e interpretados, ofreciendo al lector una comparativa y estableciendo directrices que faciliten la toma de decisiones, a nivel de tecnología, cuando se aborda un proyecto de IA.

En este sentido, se discute un conjunto básico de las principales **características deseables de un lenguaje de programación para aplicaciones de IA**, con un especial énfasis en las que resultan relevantes para el programador. Posteriormente, y desde un punto de vista más concreto, se aborda la introducción de dos lenguajes de programación de referencia, **Python** y **R**, con respectivos casos prácticos que ejemplifican sus bondades.

Finalmente, el capítulo termina detallando aspectos más específicos que generalmente son transversales en el desarrollo de aplicaciones de IA, como el **procesamiento de datos** y el uso de lenguajes de marcas.

1.1. Introducción a la programación

Esta sección ofrece una visión general del proceso de desarrollo de software, contextualizándolo para destacar la importancia que tiene más allá de la fase puramente vinculada a la programación. En este recorrido general y sintetizado, se abordará la relevancia de la capacidad de manejar abstracciones y de plantear

soluciones simples. Asimismo, esta visión se complementará con un listado aconditado de buenas prácticas de desarrollo y generación de *clean code*. Por otro lado, esta sección resume las principales características de los lenguajes de programación compilados e interpretados, en el marco de herramientas para el desarrollo de proyectos de IA.

1.1.1. El proceso de desarrollo de software

La construcción de software es un proceso complejo y sofisticado que va más allá de lo que comúnmente conocemos como programación. En este apartado se ofrece una visión general de las fases más relevantes en las que este proceso se estructura. Asimismo, también se introducen dos aspectos que están relacionados con destrezas fundamentales a la hora de abordarlo: i) la capacidad de abstracción mediante el uso de metáforas y ii) la importancia de la simplicidad a la hora de plantear cualquier tipo de solución.

Fases principales

La industria del desarrollo o construcción de software ha madurado significativamente en los últimos 30 años, siendo posible identificar un conjunto de fases que conforman dicho desarrollo. Así, y desde una perspectiva de la ingeniería del software, las principales fases son las siguientes [McC04]:

- **Definición del problema.** En esta fase se especifica el prerequisito más relevante de todo proyecto software, el cual consiste en una identificación clara y precisa del problema que se pretende resolver. En este punto, no se hace referencia a posibles soluciones. La salida esperada es un breve documento, de una o dos páginas, que defina el problema.
- **Especificación de requisitos.** Esta fase consiste en generar una descripción detallada acerca de lo que el sistema software debe hacer. Uno de los aspectos fundamentales de esta fase reside en garantizar que la funcionalidad del sistema esté guiada por el usuario final, y no por el programador. En otras palabras, un listado exhaustivo de requisitos representa el contrato entre el cliente y el desarrollador y permite que el cliente lo revise y valide antes de abarcar otras fases. Es importante prestar mucha atención a esta fase con el objetivo de minimizar el número de cambios sobre el sistema, especialmente en las etapas de diseño y desarrollo.
- **Planificación del desarrollo.** Esta fase tiene como objetivo detallar la planificación del resto de fases de la construcción de software. Existe una dependencia clara entre esta fase y la anterior, ya que el alcance del proyecto vendrá determinado por el número y complejidad de los requisitos previamente identificados.

- **Arquitectura software o diseño de alto-nivel.** Esta fase trata el diseño global de un proyecto software, entendido como la actividad que sirve de nexo entre la especificación de requisitos y la codificación. En proyectos complejos, la arquitectura software define el mapa global sobre el que se apoyará un conjunto de aspectos de diseño más específicos que, posteriormente, se puedan utilizar como punto de referencia a la hora de programar. En esta fase se manejan conceptos como módulos, subsistemas o paquetes.
- **Diseño detallado.** Esta fase persigue la especificación de un diseño más cercano al código, considerando las entidades que forman parte del mismo y sus relaciones. En esta fase se manejan conceptos como el tradicional diagrama de clases, en caso de utilizar un paradigma de programación orientada a objetos. Tanto en esta fase, como en la anterior, debe prestarse atención a características como la reducción de la complejidad, la facilidad de mantenimiento, el bajo acoplamiento, la extensibilidad o la capacidad de reutilización, entre otras.
- **Codificación y depuración.** Esta fase engloba tanto el proceso de programación en sí, entendido como la escritura de un conjunto de sentencias que persiguen la generación de un resultado, como el proceso de depuración o identificación y corrección de las causas que originan errores. En función del paradigma de programación empleado, el nivel de abstracción empleado será diferente. Por ejemplo, en la programación estructurada, el concepto de rutina o función representa un elemento fundamental, mientras que en la programación orientada a objetos la clase es la herramienta esencial para programar.
- **Pruebas unitarias.** Esta fase aborda la escritura de código que posibilite la prueba de cierto código, como por ejemplo una clase, de manera independiente al resto de código que conforma el sistema. La fase de pruebas tiene una gran relevancia para reducir el número de errores que se puedan producir en tiempo de ejecución. De hecho, existen procesos de desarrollo de software que giran alrededor del concepto de prueba, como por ejemplo TDD (Test-Driven Development) [Bec04].
- **Pruebas de integración.** Esta fase tiene como objetivo combinar la ejecución y pruebas de dos o más clases, módulos o subsistemas, de forma que se puedan evaluar las interacciones existentes entre ellas. Es importante realizar pruebas de integración desde el principio hasta el final del desarrollo del proyecto.
- **Integración.** Esta fase aborda la problemática de mezclar nuevo código con código existente. Al igual que ocurre con la fase de pruebas de integración, resulta recomendable abordar esta fase lo antes posible para mitigar la complejidad de la misma cuando el proyecto crece de tamaño. La idea de integración continua se ha acuñado en los últimos años como filosofía de trabajo [HF10].

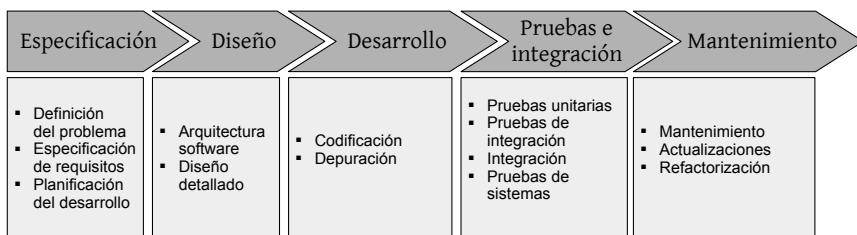


Figura 1.1: Fases globales del proceso de desarrollo de software.

- **Pruebas de sistemas.** Esta fase representa la última actividad relativa a pruebas de software antes de llevar a cabo el despliegue de un sistema, o una actualización del mismo, en producción.
- **Mantenimiento.** Esta última fase está relacionada con la actualización del código, e incluso en ocasiones del diseño, lo cual deriva en procesos de refactorización, que conforman el núcleo de un proyecto de desarrollo de software. Las actualizaciones de código son una consecuencia de los cambios introducidos en un proyecto después de su lanzamiento. Aunque pueda resultar sorprendente, los costes de mantenimiento suelen representar la mayor partida económica de un proyecto software a largo plazo.



Más allá del mantenimiento. De acuerdo a Fred Books, “el problema fundamental del mantenimiento de los programas es que arreglar un defecto tiene una probabilidad considerable (20-50 %) de introducir otro. Por lo tanto, todo el proceso consiste en dar dos pasos adelante y uno atrás”.

Como puede apreciar, en el exhaustivo listado anterior se reflejan las 4 grandes fases generales que probablemente ya tuviera interiorizadas: i) especificación, ii) diseño, iii) desarrollo y iv) pruebas. La figura 1.1 muestra este listado de manera visual. Desafortunadamente, no es sencillo encontrar un equilibrio entre el nivel de formalidad del anterior listado y la posibilidad de relajar u obviar alguna de las fases mencionadas. De hecho, es bastante probable que, internamente, las haya agrupado alrededor del **concepto de programación**, especialmente si hasta ahora ha trabajado en proyectos personales o prototipos que no vayan a alcanzar un nivel significativo de madurez tecnológica.

Si bien la fase de programación o codificación representa una de las principales actividades en proyectos de prototipado rápido, es importante **conocer el contexto general del proceso completo de construcción de software**. Esto es especialmente relevante tanto para saber dónde poner el esfuerzo en cada momento como para no olvidar las dependencias, y consecuencias, existentes entre fases. A modo de ejemplo, un error en la fase de diseño puede acarrear consecuencias desastrosas en la fase de codificación y depuración, como el incremento de la complejidad de la solución o el aumento en el tiempo empleado en programar la misma.



Niveles de Madurez Tecnológica. Procedentes del término en inglés TRL (Technology Readiness Levels), representan una medida de la madurez de una tecnología, facilitando el uso de una nomenclatura consistente y uniforme. Los niveles TRL utilizados en la Unión Europea van de 1 a 9: 1) principios básicos observados, 2) concepto de tecnología formulado, 3) prueba experimental de concepto, 4) tecnología validada en laboratorio, 5) tecnología validada en un entorno relevante, 6) tecnología demostrada en un entorno relevante, 7) demostración del prototipo del sistema en un entorno operativo, 8) sistema completo y calificado, 9) sistema real probado en el entorno operativo.

Abstracción y uso de metáforas

Una de las capacidades más relevantes que un desarrollador debe adquirir es la **capacidad de abstracción**, es decir, la capacidad de generalizar y reutilizar ideas independientemente del contexto particular. En este sentido, el uso de metáforas ha resultado ser tremadamente útil conforme la industria del desarrollo de software ha ido evolucionando. De hecho, el concepto de modelado software está inherentemente relacionado con la aplicación de metáforas o analogías, las cuales facilitan la generación de resultados cuando se compara un tema o dominio que no se entiende del todo bien, al menos al principio, con otro que sí se entiende de manera efectiva.

Existe un amplio listado de metáforas que han servido para simplificar la complejidad asociada al desarrollo de software desde una perspectiva integral [McC04]. Quizá una de las más representativas viene derivada de la expresión *escribir código*, la cual sugiere que desarrollar un programa es similar a escribir una carta. Otra metáfora bastante conocida es la que compara el hecho de cultivar una cosecha con la creación de software. Esta se basa en la idea de plantear soluciones incrementales: diseño de una parte del sistema, codificación de esa parte, pruebas e integración. En esencia, plantear una solución que se va generando poco a poco minimiza el riesgo de encontrarse en una situación donde la complejidad subyacente es tal que dificulta enormemente la gestión de un proyecto.



La importancia de documentar código. En relación a la analogía entre escribir código y escribir una carta, en este punto se anima al lector a recordar la clásica afirmación *documentation is a love letter that you write to your future self*.

En el **contexto del desarrollo de aplicaciones de inteligencia artificial**, la utilización de metáforas es especialmente relevante. Considere, por ejemplo, que el concepto de red neuronal es en realidad una analogía, aunque inexacta, de cómo funcionan y se comunican las neuronas en un cerebro biológico. Hoy en día, las redes neuronales se emplean en programas capaces de reconocer patrones y aprender de forma automática, entre otras funciones, y representan el corazón del *deep learning*.

Otro caso relevante es el de los sistemas basados en reglas, ya sean definidas por un experto humano o aprendidas de manera automática, los cuales recrean la forma en la que los humanos solemos pensar: si se da una determinada situación, definida por un conjunto de condiciones, entonces se lanzan una o varias acciones. Una metáfora más reciente es la de los *web crawlers*, entendidos como programas que analizan y extraen enlaces o hipervínculos de páginas web, no dejan de ser una analogía de una telaraña explorada por una cazadora (araña) en búsqueda de presas (información). El concepto de *web scrapping*, relacionado con el anterior, se centra en la extracción de datos o contenido.

Incluso existen teorías científicas, como la lógica difusa o *fuzzy logic* [Zad99], que acercan la forma en la que nosotros razonamos a la definición de modelos matemáticos que permitan su uso por parte de máquinas o programas. En esencia, la lógica difusa es una forma de lógica multi-valuada en el que los valores de verdad de las variables pueden tomar valores en el rango [0, 1]. Por el contrario, en la lógica booleana las variables solo pueden tomar los valores enteros de 0 o 1. La lógica difusa se fundamenta en el hecho de que las personas toman decisiones atendiendo a información imprecisa o vaga (típicamente, no numérica). Así, surge la idea de conjunto difuso como medio matemático para representar dicha información. Una de las aplicaciones clásicas de la lógica difusa, combinada con los sistemas basados en reglas, está representada por los sistemas de control. De hecho, una parte significativa de los productos de consumo, como por ejemplo el controlador del aire acondicionado, se basan en lógica difusa.

Otra metáfora significativa es la de agente inteligente, definido en el ámbito de la IA como cualquier entidad capaz de percibir lo que ocurre en el medio o contexto en el que habita, mediante la ayuda de sensores, y actuar en consecuencia, mediante la ayuda de actuadores, generando normalmente algún tipo de cambio en dicho medio o contexto (ver figura 1.2). En este caso, la metáfora de agente se hace directamente con el concepto de ser humano o individuo. Cuatro son las características fundamentales de un agente inteligente: i) autonomía, de manera que un agente actúa sin la intervención directa de terceras partes, ii) habilidad social, de forma que los agentes interactúan entre sí y se comunican para alcanzar un objetivo común, iii) reactividad, de manera que un agente actúa en función de las percepciones del entorno, y iv) proactividad, de manera que un agente puede tomar la iniciativa en lugar de ser puramente reactivo. Los agentes inteligentes conforman sistemas multi-agente, los cuales están asociados al dominio de la inteligencia artificial distribuida e incluso al concepto de *programación orientada a agentes*.

La importancia de la simplicidad

Si la capacidad de pensar, modelar y desarrollar de forma abstracta es una de las habilidades más importantes de un ingeniero software, **la habilidad de plantear soluciones sencillas** quizás sea la que más impacto y alcance tiene cuando se aborda cualquier proyecto software. Piense que los problemas que tendrá que resolver, desde el punto de vista de la construcción de software, son lo suficientemente

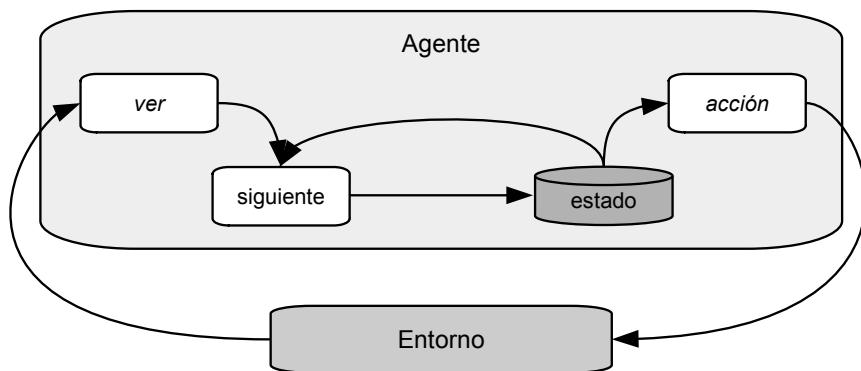


Figura 1.2: Visión abstracta del funcionamiento interno de un agente.

complejos como para que sus soluciones también lo sean. Recuerde, además, y como se ha mencionado anteriormente, que los costes de mantenimiento se suelen llevar la mayor parte de costes económicos de un proyecto software a largo plazo. Todo guarda relación con la simplicidad.

Por otro lado, el desarrollo de soluciones basadas en IA no deja de ser un caso particular de construcción de software, por lo que el planteamiento de soluciones simples sigue teniendo la misma o más importancia. De hecho, el desarrollo de soluciones basadas en IA, al menos en lo que se refiere a proyectos experimentales, está íntimamente relacionado con el **concepto de prototipado**, donde el pragmatismo y la simplicidad deberían ser protagonistas. Suele ser bastante común prototipar una solución en un lenguaje como Python o LUA para, posteriormente, generar una solución con más alcance en C++ (especialmente si existen restricciones en términos de eficiencia, como se discutirá más adelante).



Tomando decisiones. Ante un problema para el cual resulta posible aplicar diferentes soluciones, ¿cuánto tiempo dedica a evaluar qué solución es la más sencilla y, por lo tanto, ofrece mejores perspectivas de desarrollo y mantenibilidad de código?

Puede que el lector esté familiarizado con el principio KISS (Keep it Simple Stupid!), el cual pone de manifiesto que la mayoría de sistemas funcionan mejor si se mantienen lo más sencillos posible. En otras palabras, la simplicidad se convierte en un objetivo principal en el diseño, obviando cualquier atisbo de complejidad innecesaria.



Simplicidad como regla general. La navaja de Ockham es un principio metodológico y filosófico según el cual “en igualdad de condiciones, la explicación más sencilla suele ser la más probable”. En el ámbito de desarrollo de software, cuando dos soluciones generan, potencialmente, el mismo resultado, el más sencillo tendría que prevalecer sobre el otro. Incluso Leonardo Da Vinci afirmó que “la simplicidad representa el máximo nivel de sofisticación”.

1.1.2. Aspectos fundamentales de desarrollo

La elección del lenguaje de programación

En este punto se incluye una breve descripción de algunos de los lenguajes de programación que más protagonismo han tenido y que más se utilizan en la actualidad [McC04]. El objetivo que se persigue es el de ofrecer al lector una visión general de dichos lenguajes de programación. El listado que se presenta a continuación está ordenado alfabéticamente, y no por orden de importancia o relevancia.

- **Ada.** Lenguaje de programación de propósito general, orientado a objetos y con un soporte nativo para la construcción de sistemas concurrentes. Desarrollado originalmente por el Departamento de Defensa de los Estados Unidos, Ada está particularmente pensado para sistemas empotrados y sistemas de tiempo real. Una de las características más relevantes del lenguaje es el de la encapsulación de datos, forzando al programador a elegir entre aquellos elementos que son públicos o privados de cada clase o paquete. Actualmente, el uso principal de Ada está vinculado con los sectores militar y aeroespacial.
- **C.** El lenguaje C es considerado como un lenguaje de programación con un nivel de abstracción medio, originalmente asociado al desarrollo del sistema operativo UNIX. Sin embargo, C ofrece características de alto nivel, como las estructuras o las propias sentencias de control incluidas en el lenguaje.
- **C++.** Este lenguaje de programación orientado a objetos, compatible con C, ofrece características como el soporte a clases, polimorfismo, plantillas y una biblioteca estándar altamente potente y flexible. Es considerado como el estándar de facto en el ámbito de las aplicaciones gráficas interactivas.
- **C#.** Este lenguaje de programación, desarrollado por Microsoft, tiene una sintaxis similar a otros lenguajes populares, como C++ o Java, ofrece mecanismos de alto nivel, como la orientación a objetos, y proporciona herramientas de desarrollo especialmente pensadas para plataformas Microsoft.

- **Ensamblador.** Se trata de un lenguaje de bajo nivel caracterizado por establecer una relación directa entre instrucciones máquina y sentencias del propio lenguaje. Precisamente, debido a esta característica, un lenguaje ensamblador siempre estará vinculado inherentemente a un determinado procesador. Con carácter general, no se hará uso de lenguaje ensamblador salvo que existan unas restricciones enormes en lo que se refiere a optimización de código (tiempo de ejecución o tamaño del código).
- **Java.** Lenguaje de programación, similar en sintaxis a C y C++, creado por Sun Microsystems y que ofrece un entorno de desarrollo de alto nivel. Una de las principales características del lenguaje es la portabilidad, gracias al concepto de *Java Virtual Machine* y a la posibilidad de convertir el código fuente Java en *byte code*. Este código se puede ejecutar en cualquier plataforma que tenga disponible una máquina virtual. El auge de Java estuvo principalmente vinculado, en su momento, al desarrollo de aplicaciones web.
- **JavaScript.** Lenguaje de *scripting* interpretado que se ha utilizado principalmente para el desarrollo de la funcionalidad relativa a la parte del cliente en aplicaciones web.
- **PHP.** Lenguaje de *scripting* que se ha utilizado esencialmente para el desarrollo de la parte del servidor en aplicaciones web.
- **Python.** Lenguaje de alto nivel, versátil, interpretado y con soporte para la programación orientada a objetos, que se utiliza en un amplio rango de dominios, desde el desarrollo de la parte del servidor de aplicaciones web hasta el prototipado rápido de componentes de inteligencia artificial.
- **SQL.** Lenguaje estándar de referencia en lo que se refiere a consultas, actualización y gestión de bases de datos relacionales. De hecho, SQL es el acrónimo de *Structured Query Language*. Se trata de un lenguaje declarativo, es decir, que no se basa en el uso explícito de una secuencia de operaciones como tal, sino que está guiado por los resultados de las operaciones que se aplican.
- **Visual Basic.** Lenguaje de programación de alto nivel, orientado a objetos, con soporte a la programación visual y que deriva de la versión de BASIC creada para aplicaciones de escritorio en entornos Microsoft. Su acrónimo, BASIC, significa *Beginner's All-purpose Symbolic Instruction Code*.



Popularidad de los lenguajes de programación. Existen diversos índices que reflejan la demanda actual de los lenguajes de programación más representativos. Un ejemplo relevante es el del índice TIOBE, utilizado desde principios de siglo.

En el listado anterior, se ha mencionado el atributo *interpretado*, asociados al concepto de lenguaje de programación. Esta característica, junto al concepto de lenguaje de programación *compilado*, serán objeto de estudio más adelante, ya que dichas características resulta especialmente relevante a la hora de escoger un lenguaje de programación para el desarrollo de IA.

Convenciones de desarrollo

En el desarrollo de software debe existir una relación entre la arquitectura general propuesta para resolver un problema, materializada mediante el diseño, y la implementación final que le da soporte y que se realiza a través de un lenguaje de programación concreto. Esta coherencia también ha de reflejarse a nivel interno, de forma que la solución a nivel de código sea consistente. Precisamente, esta consistencia es la que idealmente ha de derivar en el uso de guías o convenciones de desarrollo a la hora de utilizar, por ejemplo, nombres para las clases, las funciones, las variables e incluso para los comentarios de código.

Piense en un proyecto en el que participan, concurrentemente, un equipo de 8 desarrolladores software. Resulta evidente asumir la existencia de una **guía que sirva como referencia para escribir un código que sea consistente**. De otro modo, cada ingeniero utilizaría, por ejemplo, una convención de nombrado diferente para su código. Esta situación dificultaría tanto la coordinación interna del equipo como el futuro mantenimiento del código.

Un ejemplo de guía de estilo para el desarrollo de código en C++ es la utilizada internamente por Google¹. Otro ejemplo representativo, para el caso del lenguaje de programación Python, es la guía de estilo PEP (Python Enhancement Proposal) 8².

Visión general sobre buenas prácticas de desarrollo

En este apartado se pretende ofrecer al lector una visión general, y a muy alto nivel, de buenas prácticas de desarrollo cuando se aborda un proyecto. Debido a la profundidad de la temática, a continuación se esbozan algunas de las cuestiones más importantes [McC04].

A nivel de **desarrollo de código**:

- Identifique qué aspectos del diseño son susceptibles de abordar de manera previa a la codificación y cuáles se pueden trabajar en el momento de escribir el código.
- Utilice un convenio de nombrado de acuerdo al lenguaje de programación elegido.

¹<https://google.github.io/styleguide/cppguide.html>

²<https://www.python.org/dev/peps/pep-0008/>

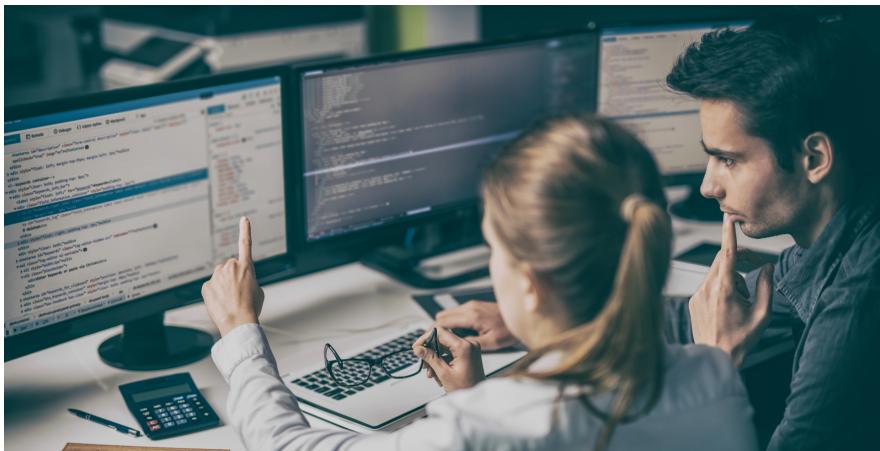


Figura 1.3: Paradigma de programación en parejas o *pair programming*.

- Adopte prácticas de programación que tengan en cuenta cómo se van a gestionar posibles errores o excepciones, cuestiones de seguridad de código y diseño de interfaces, entre otras.
- Evalúe el impacto del uso de tecnologías más maduras o más recientes en el contexto de su proyecto.

A nivel de **coordinación y trabajo en equipo**:

- Defina un esquema de integración de código, considerando los pasos que han de darse antes de desplegar código en un sistema en producción.
- Elija una técnica de desarrollo de software que encaje con su filosofía de trabajo, como por ejemplo *Pair Programming* [BA04] (ver figura 1.3).

A nivel de **pruebas y calidad**:

- Defina si los programadores usarán un modelo de desarrollo dirigido por tests, como TDD [Bec04], lo cual implica codificar primero las pruebas.
- Defina si los programadores codificarán pruebas unitarias.
- Defina si los programadores llevarán a cabo pruebas de integración antes de promocionar su código.
- Defina si los programadores realizarán una revisión cruzada de código.

A nivel de **herramientas y entorno de desarrollo**:

- Elija y use un sistema de control de versiones, como Git³.

³<https://git-scm.com/>

- Elija lenguaje, versión del lenguaje y versión del compilador (o intérprete) del lenguaje.
- Elija y valore las ventajas de usar un *framework* de desarrollo concreto.
- Identifique y valore herramientas adicionales que puedan facilitar el proceso de desarrollo, como el depurador, el *framework* de pruebas o incluso la generación automática de esqueletos de código.

Principios de *Clean Code*

Si usted conoce el concepto de deuda técnica, probablemente también le resulte familiar el término *clean code*. Si no es así, probablemente intuya que una mala decisión en el diseño o incluso en la programación de código puede acarrear consecuencias que son difíciles de gestionar conforme pasa el tiempo. Implícitamente, ya hemos tratado el **concepto de deuda técnica** cuando comentamos que la etapa de mantenimiento de un proyecto software es la que suele estar relacionada con la mayor partida económica de un proyecto software a largo plazo. Incluso si se trata de un proyecto personal o de un prototipo interno, la diferencia entre un buen código y un mal código siempre marca la diferencia.

Una de las referencias bibliográficas más relevantes en el ámbito del clean es precisamente el libro que lleva su nombre: *Clean Code, A Handbook of Agile Software Craftsmanship* [Mar08]. Se anima al lector a revisar los principios fundamentales del código limpio y a descubrir la analogía existente entre ellos y la importancia de la simplicidad previamente expuesta.



Código limpio. La definición de *Clean Code* puede tener múltiples acepciones. Aquí se incluye la traducción de una de ellas, formulada por Bjarne Stroustrup, el creador del lenguaje de programación C++: “Me gusta que mi código sea elegante y eficiente. La lógica del código debería ser sencilla para dificultar la depuración de errores, con un número mínimo de dependencias para facilitar el mantenimiento, con una gestión de errores completa de acuerdo a una estrategia bien definida, y con un rendimiento cercano al óptimo para evitar que otros desarrolladores generen código complejo como consecuencia del primero. El código limpio hace una cosa y la hace bien.”

1.1.3. Lenguajes de programación compilados e interpretados

Visión general

A la hora de abordar el diseño de una aplicación, tendrá que tomar la decisión de utilizar un lenguaje de programación compilado o interpretado para escribir su código fuente. Como puede imaginar, cada tipo de lenguaje tiene sus fortalezas y debilidades. Esencialmente, la decisión de emplear un lenguaje interpretado estará relacionada con las restricciones de tiempo existentes a la hora de abordar un

proyecto software y la facilidad de realizar futuros cambios en el mismo. La cara negativa de la moneda tiene que ver con el rendimiento. Así, cuando se emplea un lenguaje de programación interpretado, se está incurriendo en unos costes de ejecución elevados para emplear una herramienta que acelera la velocidad de desarrollo. En otras palabras, un lenguaje interpretado puede ser más adecuado para peticiones de desarrollo sobrevenidas, mientras que un lenguaje compilado sería una mejor opción, al menos a priori, para una petición predefinida.

Entrando más en detalle, un **lenguaje compilado** se basa en la escritura de programas que, una vez compilados, se expresan en instrucciones de una arquitectura máquina concreta. A modo de ejemplo, la operación de suma de dos valores numéricos en el lenguaje de programación C++ sería trasladada directamente en la operación ADD en código máquina. Si bien un programa se podría implementar utilizando directamente código máquina, también denominado ejecutable o binario, el nivel de abstracción tan bajo no haría práctico el desarrollo de proyectos complejos. Además, sería necesario realizar una implementación diferente para cada arquitectura hardware. Esta es una de las principales razones de la existencia de los compiladores. Un compilador es un programa que traduce código fuente, programado en un lenguaje de programación de alto nivel, en código máquina para una determinada arquitectura.

El proceso de compilación se estructura, a su vez, en varios pasos intermedios, tal y como se muestra en la figura 1.4. Se pueden distinguir dos fases principales. La fase de *frontend* realiza el análisis léxico, sintáctico y semántico de los ficheros de entrada que representan el código fuente del proyecto. La salida de esta fase es un *código intermedio* independiente de la arquitectura hardware final. La fase de *backend*, y particularmente el *optimizador*, toma como entrada dicho *código intermedio* y lo mejora mediante diversas estrategias, como la eliminación de código muerto. Posteriormente, el *generador de código* ofrece como salida el código binario vinculado a una determinada arquitectura, el cual también es optimizado por el generador.

Resulta muy común enlazar bibliotecas de código ya existente con el código binario generado por el compilador para aprovecharse de las ventajas de la reutilización de código. Así, el enlazador, otro componente de los compiladores modernos, posibilita el **proceso de enlazado** de código a través de dos formas generales: i) estático, donde las dependencias de código se resuelven en tiempo de enlazado, generando un ejecutable autocontenido que no requiere la instalación de bibliotecas externas en la máquina destino, y ii) dinámico, donde las dependencias de código se resuelven en tiempo de ejecución, generando un ejecutable final de menor tamaño pero que requiere la instalación previa de bibliotecas externas. En [VM15] se ofrece una descripción en detalle del proceso de compilación para el caso particular del conjunto de compiladores GCC (GNU Compiler Collection).

Por el contrario, un **lenguaje interpretado** es aquel en el que las instrucciones generadas no se ejecutan directamente sobre una máquina destino, sino que se leen y ejecutan por otro programa, normalmente escrito en el lenguaje de la máquina destino. Si retoma el ejemplo anterior de la suma de dos valores numéricos, en este

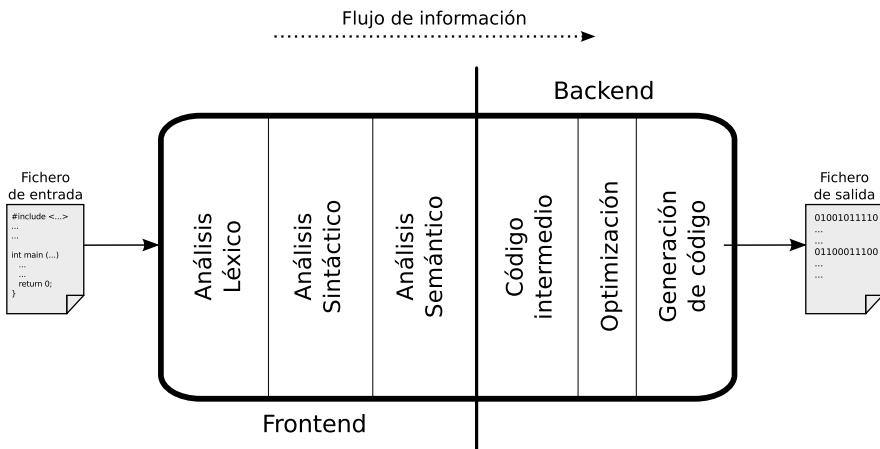


Figura 1.4: Fases del proceso de compilación [VM15].

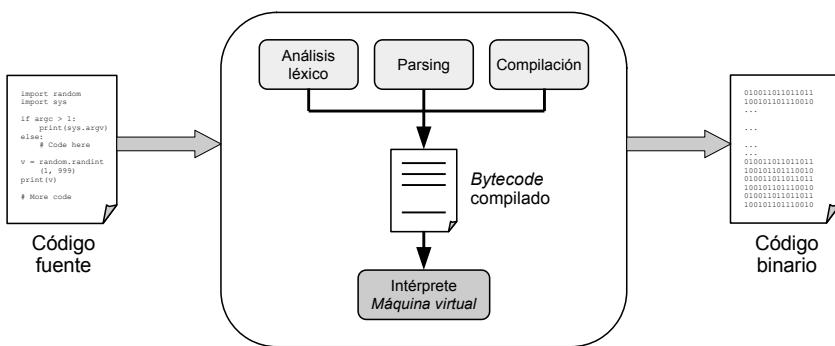


Figura 1.5: Visión general de las fases principales de un intérprete Python.

caso la misma operación de suma sería reconocida por el intérprete en tiempo de ejecución, el cual la traduciría a una función del tipo `add(a, b)` y que, posteriormente, ejecutaría a través de la instrucción máquina ADD. La figura 1.5 muestra, de manera gráfica, las principales fases abordadas por un intérprete Python.



El lenguaje de programación Java. Algunos autores afirman que los lenguajes de programación, en sí mismos, no se pueden clasificar en compilados o interpretados. Son las implementaciones específicas de un lenguaje las que realmente reflejan esta característica. En el caso de Java, están involucrados componentes como la máquina virtual de Java (*Java Virtual Machine*), compiladores nativos como *gcj* e intérpretes para el propio lenguaje, como *BeanShell*. Así pues, ¿qué tipo de lenguaje es Java?

Comparativa

En primer lugar, es importante destacar que, desde el punto de vista funcional, es posible implementar mediante un lenguaje interpretado cualquier programa que se pueda implementar con un lenguaje compilado, y viceversa. Sin embargo, cada opción, como se ha introducido anteriormente, tiene sus ventajas y desventajas. Estas se resumen a continuación.

Con respecto a los lenguajes compilados,

- Generación de código eficiente, que se puede ejecutar un número arbitrario de veces, para la máquina destino. En otras palabras, la sobrecarga incurrida por el proceso de compilación, una vez que el código ha sido validado, se reduce a un único proceso.
- Posibilidad de aplicar optimizaciones durante el proceso de compilación.
- Incremento del rendimiento con respecto a un lenguaje interpretado.

Con respecto a los lenguajes interpretados,

- Facilidad a la hora de implementar y prototipar código.
- Con carácter general, reducción de la complejidad a la hora de emplearlos como toma de contacto en relación a un lenguaje compilado.
- El código se puede ejecutar *al vuelo*, sin necesidad de un proceso previo de compilación.

Teniendo como referencia estas cuestiones, se establecen **dos escenarios generales que pueden servir como referencia** a la hora de elegir un lenguaje de programación compilado o interpretado:

1. Si el contexto del proyecto lo permite, un lenguaje interpretado resultaría más adecuado para llevar a cabo tareas de prototipado rápido o para generar una primera versión de un proyecto con agilidad. Como se ha introducido en este capítulo, las aplicaciones de IA en las que se necesiten probar y adaptar comportamientos con cierta rapidez representan un candidato ideal con respecto a dicha elección.
2. Las tareas más intensas, desde el punto de vista computacional, o aquellas que se ejecuten con más frecuencia dentro de un proyecto software, se podrían relacionar con un lenguaje compilado. Por el contrario, aquellas menos intensas, como por ejemplo una interfaz de usuario, se podrían relacionar con un lenguaje interpretado.



No todo es blanco o negro. Note que en un mismo proyecto software pueden convivir varios lenguajes de programación. Por ejemplo, se podría utilizar un lenguaje compilado para implementar el núcleo funcional del proyecto y un lenguaje interpretado para aquellas cuestiones menos relevantes desde el punto de vista computacional.

1.1.4. Resumen

En esta sección se ha ofrecido una visión general del proceso de desarrollo de software, entendido como un proceso complejo que va más allá de la programación. Particularmente, se han agrupado las fases identificadas en cuatro grandes etapas: especificación, ii) diseño, iii) desarrollo y iv) pruebas. Asimismo, se ha establecido la relación de este proceso con dos destrezas fundamentales que deberían interiorizarse por parte de cualquier ingeniero software: i) el uso de metáforas como mecanismo para manejar de manera eficaz la abstracción y ii) la importancia de la simplicidad como eje fundamental para diseñar, programar y mantener código. Esta introducción general ha quedado complementada con una visión general sobre buenas prácticas de desarrollo y referencias a cuestiones relacionadas con *clean code*.

Por otro lado, se ha ofrecido al lector una comparativa general de lenguajes de programación compilados e interpretados, relacionando la misma con su uso como herramienta para desarrollar proyectos de IA. En la siguiente sección se abordarán las características deseables en un lenguaje de programación para IA.

1.2. Programación de Inteligencia Artificial

La programación de aplicaciones de IA comparte la gran mayoría de aspectos y cuestiones a considerar por parte de otro tipo de proyecto donde el desarrollo software sea el principal protagonista. Sin embargo, es posible identificar un conjunto de características concretas que han de valorarse cuando se aborda un proyecto de IA. Particularmente, en esta sección se discute un listado de 5 características específicas a tener en cuenta a la hora de elegir un lenguaje de programación. Este listado se complementa con fragmentos de código y con un caso práctico diseñado para ilustrarlas y compararlas cuando se usan los lenguajes de programación C y Python, respectivamente.

1.2.1. Consideraciones previas

En los últimos años, resulta evidente que la transformación digital y la automatización han causado una revolución en términos industriales. Uno de los actores cuyo peso ha ido creciendo significativamente es la Inteligencia Artificial. Por lo tanto, la demanda de profesionales que sepan cómo diseñar, desarrollador, validar y mantener proyectos basados en IA se irá incrementando progresivamente en el futuro.



Ética en IA. Más allá de la parte técnica vinculada al desarrollo de software e IA, existen otras consideraciones, como la ética, que están inherentemente vinculadas al uso de IA. Este capítulo, no obstante, está centrado en la parte técnica y de desarrollo.

En este contexto, a la hora de **aprender las competencias necesarias para la programación de software para IA**, existen diversos caminos que se podrían recorrer (ver figura 1.6):

1. Aprender a programar software de manera simultánea a conocer los fundamentos en los que se basa la IA.
2. Aprender a programar software de manera previa a conocer los fundamentos de IA.
3. Aprender los fundamentos de IA de manera previa a abordar los aspectos que conforman el núcleo de la programación actual.

Cada opción tiene sus ventajas y desventajas. La primera opción puede resultar demasiado ambiciosa, debido a que sería necesario abordar tanto la complejidad que representa el aprendizaje de la programación, incluyendo la capacidad de pensar de forma abstracta, como los fundamentos de las principales técnicas y algoritmos de IA. No obstante, si el aprendizaje está guiado por proyectos aplicados y concretos, es posible abordarlo desde un punto de vista incremental.

La segunda opción es probablemente la más extendida en gran cantidad de currículos vinculados al ámbito de *Computer Science*. En este sentido, disponer de una buena base de programación (particularmente de algoritmia, análisis de complejidad y diseño de estructuras de datos), facilita el estudio y entendimiento de los fundamentos en los que se apoyan gran parte de las técnicas de IA existentes. En este sentido, el hecho de disponer de la programación como herramienta lista para usarse en el ámbito de la IA puede simplificar la comprensión y aplicación práctica de dichas técnicas a problemas concretos.

Por otro lado, la tercera opción estaría más vinculada con un perfil donde los fundamentos matemáticos pueden tener más presencia, al menos en primera instancia, de forma que el aprendiz domine primero la parte algorítmica y, posteriormente, aprenda a programar tanto a nivel transversal como a nivel más aplicado a la IA.

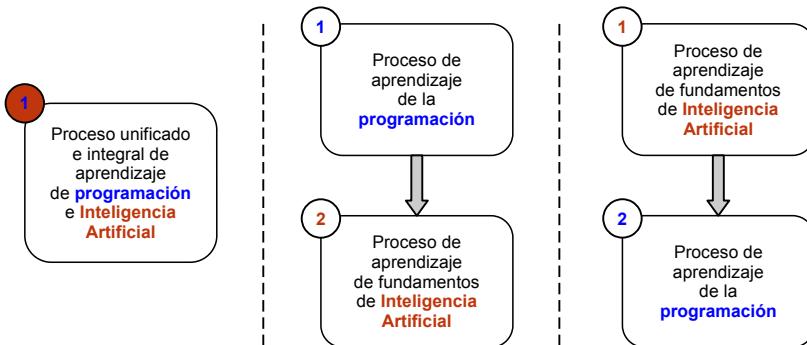


Figura 1.6: Esquema conceptual de los diferentes caminos de aprendizaje a la hora de aunar IA y desarrollo software.



Introduction to Machine Learning. Este término es uno de los más buscados cuando una persona decide abordar el aprendizaje automática desde una perspectiva de la programación. Le recomiendo que revise y reflexione sobre cómo se aborda en la literatura existente, incluyendo las asignaturas que llevan por título su nombre. A modo de referencia, la asignatura del MIT (Massachusetts Institute of Technology) *Introduction to Machine Learning* tiene 3 prerequisitos: i) Programación (mencionando específicamente el lenguaje Python), ii) Cálculo y iii) Álgebra Lineal.

1.2.2. Características deseables en un lenguaje de programación para IA

En este apartado se describen un conjunto mínimo de propiedades que, idealmente, todo lenguaje de programación utilizado para un proyecto de IA debería poseer. No se trata de ofrecer al lector un listado exhaustivo de características, sino una **referencia inicial** que sirva como punto de partida a la hora de elegir el lenguaje a utilizar para resolver un determinado problema. Para ello, se han escogido las siguientes 5 características:

1. **Simplicidad**, debido a la importancia de la misma para escribir y mantener código.
2. **Capacidad de prototipado rápido**, debido a la propia naturaleza de las aplicaciones de IA, que suelen variar considerablemente su implementación en las primeras etapas.
3. **Legibilidad**, debido a la necesidad de acercar lo máximo posible al programador y al código, considerando que un número significativo de desarrollos en este ámbito parten de pseudocódigo o algoritmos previamente diseñados.

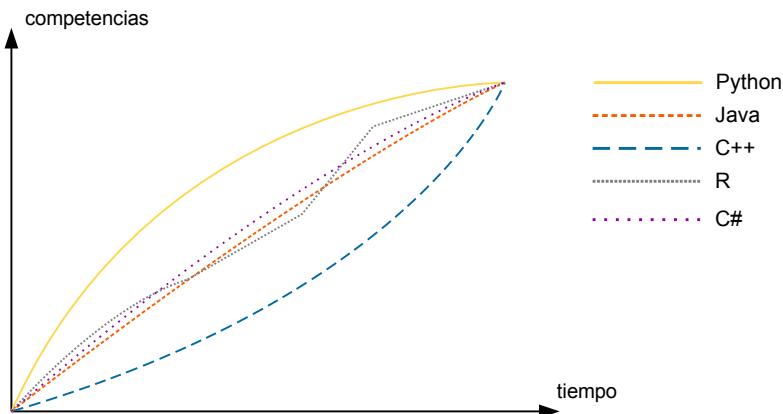


Figura 1.7: Curvas de aprendizaje de 5 lenguajes de programación utilizados para programar IA.

4. **Existencia de bibliotecas para IA**, debido a la importancia de reutilizar código existente que acelere la generación de prototipos.
5. **Comunidad de desarrollo**, debido a las ventajas que ofrece compartir experiencias y soluciones a problemas previamente abordados por otra persona.

A continuación, estas características se abordan con más detalle.

Simplicidad

La importancia de la simplicidad, en un contexto general de desarrollo de software, ya introdujo en la sección 1.1.1. La simplicidad como características deseable de un lenguaje de programación está relacionada tanto con la curva de aprendizaje necesaria para dominarlo como con la sencillez a la hora de aplicar mecanismos propios del lenguaje.

Respecto a la **curva de aprendizaje**, un lenguaje de programación para IA debería facilitar un nivel básico de productividad incluso para principiantes. Esta capacidad se justifica con la necesidad de probar y validar conceptos de manera ágil, incluso por desarrolladores que no tenga una gran experiencia en el ámbito de la programación. En el dominio de la IA, al menos en las etapas iniciales de desarrollo, el foco debe estar en la técnica o enfoque de IA aplicado, y no tanto en la eficiencia o calidad del código fuente en su versión inicial. No obstante, es deseable que el programador tenga interiorizadas las competencias necesarias para resolver problemas, de manera independiente al lenguaje de programación, y pensar de forma abstracta. La figura 1.7 muestra, de manera gráfica, la curva de aprendizaje de 5 lenguajes vinculados al desarrollo de aplicaciones de IA.

En relación a los **mecanismos ofrecidos por el propio lenguaje de manera nativa**, es decir, los que forman parte de su estándar, idealmente deberían ser fáciles de entender y utilizar de manera productiva. Los mecanismos que están diseñados para mejorar la productividad de un desarrollador suelen implicar un periodo de tiempo en el que la curva de aprendizaje se hace más pronunciada, ofreciendo beneficios a más largo plazo. Un ejemplo clásico es el mecanismo de plantillas ofrecido por el lenguaje de programación C++ para manejar la programación genérica [Str13], es decir, independiente del tipo de datos que manejen las funciones o las clases. A modo de ejemplo, el lector podría pensar en un algoritmo que funciona para clasificar, automáticamente, tipos de objetos diferentes en base a una función de utilidad. Si bien el beneficio de esta característica resulta evidente, la depuración de código con plantillas introduce una complejidad no desdeñable. El listado 1.1 muestra un ejemplo de uso de plantillas de función en C++. Se invita al lector a probarlo y a introducir, a propósito, un error de compilación.

Capacidad de prototipado rápido

El desarrollo de aplicaciones de IA suele iniciarse con una fase de prototipado que permita validar un concepto o algoritmo de la forma más ágil posible. En otras palabras, los primeros prototipo software no suelen prestar demasiada atención a la calidad del código o a su rendimiento.

Listado 1.1: Ejemplo de función en C++ que hace uso de plantillas

```
1 #include <iostream>
2 using namespace std;
3
4 template <class T>
5 T GetMax (T a, T b) {
6     T result;
7     result = (a > b) ? a : b;
8
9     return (result);
10 }
11
12 int main () {
13     int i = 5, j = 6, k;
14     long l = 10, m = 5, n;
15
16     k = GetMax<int>(i, j);
17     cout << k << endl;
18
19     n = GetMax<long>(l, m);
20     cout << n << endl;
21
22     return 0;
23 }
```

Por el contrario, resulta más relevante que el lenguaje escogido facilite tanto la generación de la primera versión como su adaptación. Esta características de prototipado rápido se puede relacionar con tres aspectos fundamentales:

1. La **experiencia del programador de IA con un determinado lenguaje** de programación. Por ejemplo, si el programador tiene una gran experiencia con Java, podría escoger este lenguaje para implementar su primer prototipo, ya que le resultará relativamente fácil programar el código de un determinado algoritmo. Si la experiencia del programador con un lenguaje específico no es relevante, el segundo y el tercer aspecto, descritos a continuación, toman más relevancia.
2. La capacidad del lenguaje para **ejecutar y probar código de manera ágil**. Este aspecto está directamente vinculado con el uso de un lenguaje interpretado, donde el código se ejecuta *al vuelo* sin necesidad de un proceso de compilación previo.
3. La existencia de **código que se pueda reutilizar** por parte del desarrollador de manera directa. Debido a la importancia de esta cuestión en el contexto de desarrollo de aplicaciones de IA, la misma se discute más adelante con mayor nivel de detalle.

A modo de ejemplo, considere la necesidad de desarrollar un prototipo que permita modelar un sencillo autómata o máquina de estados. Por cuestiones de simplicidad, puede asociar un autómata con un modelo de comportamiento basado en estados, eventos y acciones. Un estado sirve para representar una situación; un evento es un suceso que dispara el cambio de un estado a otro; una acción es un tipo de comportamiento. La figura 1.8 muestra un sencillo autómata para modelar el comportamiento de un enemigo artificial en un juego de espionaje.

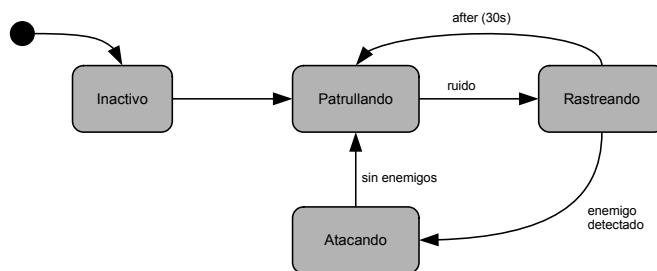


Figura 1.8: Máquina de estados que define el comportamiento de un enemigo en un juego.

Si necesitara ofrecer una implementación que soportara la máquina de estados de la figura 1.8, en un contexto de prototipar y validar el comportamiento de un enemigo artificial, probablemente optaría por un lenguaje que facilitara ambas acciones. En este contexto, el listado 1.2 muestra una posible implementación, en el

lenguaje Python, de una clase base que representa un estado genérico. Como se puede apreciar, en la línea ③ se declara la lista que contendrá el nombre de los estados a los que se puede transitar desde el estado actual. Por otro lado, la función `switch`, implementada entre las líneas ⑤ y ⑩, es la responsable de la transición efectiva entre estados, si esta es posible (vea la condición de la línea ⑥).

Listado 1.2: Implementación sencilla de una máquina de estados en Python (I)

```

1 class State(object):
2     name = "General State"
3     allowed_transitions = []
4
5     def switch(self, new_state):
6         if new_state.name in self.allowed_transitions:
7             self.__class__ = new_state
8             print 'Current:', self, '=> switched to new state', new_state.name
9         else:
10            print 'Current:', self, '=> switching to', new_state.name, 'is not allowed.'
11
12    def __str__(self):
13        return self.name

```

Esta clase base se podría especializar, a través de relaciones de herencia, para modelar los estados contemplados en la figura 1.8, así como para definir las transiciones permitidas. Estos aspectos se reflejan en el listado 1.3. Como puede comprobar, y a modo de ejemplo, desde el estado *Patrolling* solo se pueden realizar transiciones hasta el estado *Tracking*.

Listado 1.3: Implementación sencilla de una máquina de estados en Python (II)

```

1 class Inactive(State):
2     name = "Inactive"
3     allowed = ['Patrolling']
4
5 class Patrolling(State):
6     name = "Patrolling"
7     allowed = ['Tracking']
8
9 class Tracking(State):
10    name = "Tracking"
11    allowed = ['Patrolling', 'Attacking']
12
13 class Attacking(State):
14    name = "Attacking"
15    allowed = ['Patrolling']

```

Finalmente, ya es posible modelar y utilizar una máquina de estados, considerando la definición general de estado y la funcionalidad básica de transitar de un estado a otro. El listado 1.4 muestra el código, junto con un conjunto de instrucciones de prueba. Note cómo en la línea ⑤ se define la función `change` que, a su vez, delega en la función `switch` del estado actual el cambio de estado hacia *new_state*.

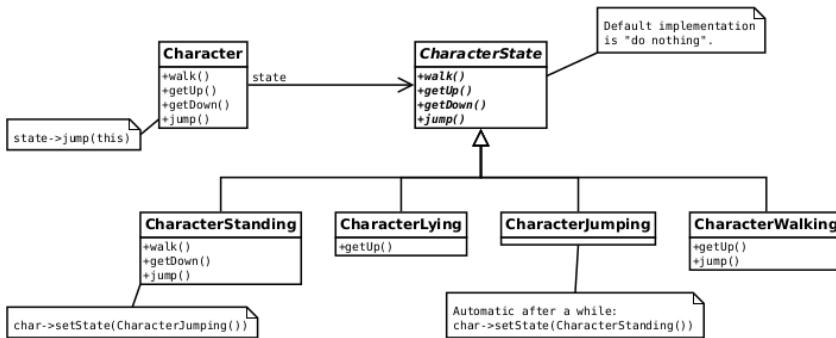


Figura 1.9: Representación gráfica del patrón de diseño State aplicada al modelado de la IA de un enemigo virtual en un videojuego [VM15].

Listado 1.4: Implementación sencilla de una máquina de estados en Python (III)

```

1  class StateMachine(object):
2      def __init__(self):
3          self.state = Inactive()
4
5      def change(self, new_state):
6          self.state.switch(new_state)
7
8  if __name__ == "__main__":
9      enemy_basic_AI = StateMachine()
10     enemy_basic_AI.change(Inactive)
11     enemy_basic_AI.change(Patrolling)
12     enemy_basic_AI.change(Tracking)
13     enemy_basic_AI.change(Attacking)
  
```

Como ha podido comprobar, gracias a unas pocas líneas de código y en relativamente poco tiempo, es posible ofrecer una implementación base de lo que sería el patrón de diseño *State* [GHJV94], ampliamente utilizado en la programación de IA y representado visualmente en la figura 1.9. En este ejemplo concreto se ha utilizado el lenguaje de programación Python; se invita al lector a prototipar este ejemplo en el lenguaje de programación que le resulte más cercano y a comparar el código obtenido con el discutido previamente.



Transiciones dirigidas por eventos y transiciones temporales. ¿Cómo aumentaría el prototipo introducido para manejar transiciones que ocurren cuando se dispara un evento o cuando pasa un determinado tiempo?

Legibilidad

Los lenguajes de programación utilizados mayoritariamente para el desarrollo de aplicaciones de IA son lenguajes de programación de alto nivel. En otras palabras, ofrecen un alto **nivel de abstracción** para que el programador no tenga que preocuparse de dónde y cómo se ejecuta el código que implementa. La cantidad de abstracción proporcionada determina cómo de alto, medio o bajo es el nivel vinculado al lenguaje. Así, y con carácter general, un lenguaje de alto nivel tiene un grado de legibilidad alto, ya que las abstracciones ofrecidas por el mismo estarán más cercanas a la hora de entender código.

El precio a pagar por utilizar un lenguaje de programación de alto nivel reside en las limitaciones existentes a la hora de optimizar código para una determinada máquina o considerando restricciones de implementación (memoria necesaria o tiempo de ejecución). No obstante, y con carácter general, esta contrapartida no tiene un impacto relevante en el contexto del desarrollo de aplicaciones de IA debido a dos motivos: i) en las primeras fases del desarrollo de IA el rendimiento del código no suele ser relevante, y ii) las herramientas que generan código máquina a partir de código fuente de alto nivel, especialmente los compiladores, son capaces de generar un resultado lo suficientemente optimizado en la mayoría de los casos.

Por otro lado, la legibilidad está relacionada con las facilidades que ofrece un lenguaje para facilitar el seguimiento del código de una manera lógica. Un ejemplo concreto sería la indentación de código en Python, la cual se refiere la uso de espacios al principio de una línea de código. Python utiliza sangrías para resaltar bloques de código. El espacio en blanco es el carácter utilizado para incluir sangrías. Así, todas las sentencias con la misma distancia a la derecha pertenecen al mismo bloque de código. Si un bloque tiene que ser anidado en un nivel de profundidad mayor, simplemente se indentá más a la derecha.

El listado 1.5 muestra un ejemplo de indentación en Python. El simple hecho de introducir espacios ya facilita el seguimiento de código. Note, por ejemplo, cómo en la línea **④ 12** y sucesivas se aprecia visualmente el bloque de código asociado al bucle *while*.

Existencia de bibliotecas para IA

La IA es un campo de trabajo amplio que se apoya sobre un conjunto de dominios o materias. Algunas están relacionadas con las matemáticas, como por ejemplo la lógica, la probabilidad, las matemáticas continuas; otras con aspectos de obtención y procesamiento de información, como la percepción multi-sensorial, el razonamiento, el aprendizaje; otras con cuestiones relativas a ciencias sociales, como las relaciones de confianza, la seguridad, la ética, el bien social.

Listado 1.5: Ejemplo en Python que refleja las bondades del uso de indentación

```
1 # Incluir módulos de pygame...
2 WIDTH, HEIGHT = 1200, 800
3 FPS = 30
4
5 if __name__ == "__main__":
6     pygame.init()
7
8     clock = pygame.time.Clock()
9     soccerField = SoccerField(WIDTH, HEIGHT)
10
11    while True:
12        tick_time = clock.tick(FPS)
13        for event in pygame.event.get():
14            if event.type == QUIT:
15                pygame.quit()
16                sys.exit()
17
18    soccerField.render()
19    pygame.display.update()
```

Algunos autores, como Russell y Norvig en el libro Artificial Intelligence: A Modern Approach [RN20], unifican todas estas cuestiones alrededor del concepto de agente inteligente, entendiendo así la IA como el estudio de los agentes que reciben estímulos de un entorno y llevan a cabo acciones (ver figura 1.2). La gestión y procesamiento de estos estímulos determina el comportamiento de estos agentes, existiendo agentes reactivos, agentes que planifican en tiempo real, agentes basados en la teoría de la decisión o agentes basados en algoritmos de *deep learning*. En cualquier caso, es importante destacar que gran parte de estos fundamentos se basan en **algoritmos que han sido implementados en múltiples ocasiones**, empleando diferentes lenguajes de programación.



Programas = Algoritmos + Estructuras de Datos. Si bien en términos de reutilización de código para la programación de IA solemos pensar en términos de algoritmos ya implementados, también es importante considerar el soporte que ofrece un lenguaje en relación a las estructuras de datos que ofrece. Recuerde que una estructura de datos bien diseñada facilitará la programación de un algoritmo, independientemente del lenguaje de programación empleado.

Por ejemplo, considere un algoritmo de búsqueda binaria sobre un array ya ordenado. El listado 1.6 muestra una aproximación en Python. Como puede imaginar, resultaría ventajoso disponer de una versión lista para usarse de dicho algoritmo, en lugar de implementarlo desde cero.

Listado 1.6: Posible implementación de un algoritmo de búsqueda binaria

```
1 def binary_search (vector, low, high, x):
2
3     # Caso base de la recursividad
4     if high >= low:
5         mid = (high + low) / 2
6
7         # ¿Elemento en la posición central del array?
8         if vector[mid] == x:
9             return mid
10
11        # Si el elemento central es mayor que x,
12        # x solo puede estar en el subarray izquierdo
13        elif vector[mid] > x:
14            return binary_search(vector, low, mid - 1, x)
15
16        # Si no, x solo puede estar en el subarray derecho
17        else:
18            return binary_search(vector, mid + 1, high, x)
19
20    # x no está en el vector de entrada
21    else:
22        return -1
```

En este sentido, la existencia de bibliotecas o módulos que faciliten la programación de IA se convierte en una característica deseable a la hora de elegir, o utilizar, un determinado lenguaje de programación. Particularmente, debería prestar especial atención a la **existencia de bibliotecas que cubran los siguientes temas:**

- Redes neuronales.
- Aprendizaje supervisado y no supervisado.
- Procesamiento natural.
- Procesamiento de textos.
- Modelado y caracterización de sistemas expertos.
- Procesamiento matemático, particularmente estadística y probabilidad.
- Visión por computador, debido a la relación del análisis y procesamiento de imágenes con la IA.
- Simuladores y motores de física.

Comunidad de desarrollo

La **popularidad de un lenguaje** es definitivamente una característica que ha de evaluar a la hora de elegir un lenguaje de programación como herramienta para solucionar un problema, ya esté relacionado o no con la IA. Si un lenguaje es popular, el número de programadores que lo estén usando será elevado. Cuanto mayor sea el número de usuarios de un lenguaje de programación, mayor será la probabilidad de que alguno de ellos se haya enfrentado a un problema similar al que usted esté afrontando en un momento determinado.



El efecto Stack Overflow. En los últimos años han proliferado los sitios web que permiten lanzar preguntas y ofrecer respuestas relacionadas con aspectos del desarrollo de software, desde una perspectiva transversal. Una de las referencias, por excelencia, es Stack Overflow⁴.

Así pues, la existencia de una comunidad de desarrollo amplia y activa es una característica deseable a la hora de elegir un lenguaje de programación para IA. Como ya se introdujo anteriormente en este mismo capítulo, existen índices, como el índice TIOBE, que nos permiten conocer la demanda de los lenguajes de programación actuales, factor que se puede utilizar como posible indicador del nivel de comunidad existente en torno a ellos.

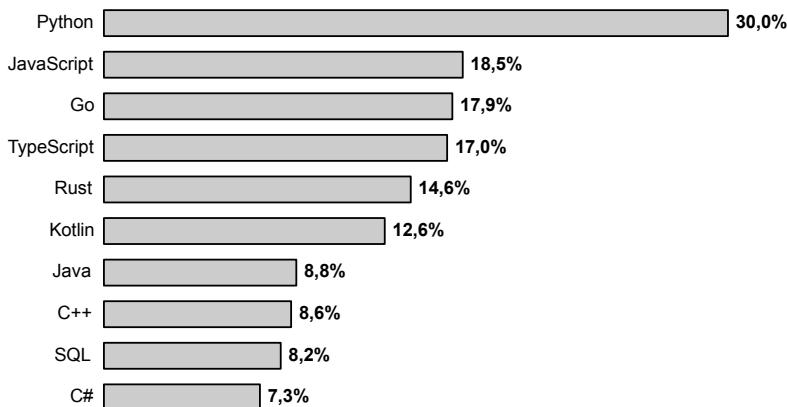


Figura 1.10: Listado de los 10 lenguajes de programación más deseados, de acuerdo a un estudio realizado por StackOverflow, representado en porcentaje de desarrolladores que no están desarrollando con una tecnología concreta pero que han expresado un interés en desarrollar con ella. Fuente: Stack Overflow Developer Survey 2020.

Otro aspecto a considerar está relacionado a la **existencia de bibliotecas software** que no formen parte del estándar del lenguaje a considerar. En el caso de aplicaciones de IA, la referencia directa a estudiar sería el número de bibliotecas o proyectos activos en dicho dominio. En esencia, es posible inferir que si existe

una comunidad de desarrolladores preocupados por mejorar la funcionalidad asociada a un lenguaje, entonces será más probable que la propia comunidad sea más receptiva a la hora de dar soporte a otros programadores que están comenzando a aprender un lenguaje.

Finalmente, otro indicador relevante que se puede estudiar es el **número de programadores en activo** vinculados a cada lenguaje. No obstante, la popularidad de un lenguaje en un determinado momento temporal puede representar una mejor aproximación a la hora de buscar contenido actualizado.

1.2.3. Caso práctico. Búsqueda de patrones en archivos de texto

En este apartado, se discute un caso práctico en el que se pretenden desarrollar y comparar varias soluciones, utilizando diferentes lenguajes de programación, para resolver un sencillo problema de búsqueda de patrones en archivos de texto. Se pretende proporcionar un ejemplo práctico en el que sea posible identificar las características deseables para la programación de IA, previamente discutidas, a la hora de elegir un lenguaje de programación. Aunque se trata de un problema sencillo y con poco alcance, nos servirá como referencia inicial, considerando especialmente las propiedades de simplicidad, capacidad de prototipado rápido y legibilidad.

Para caracterizar el caso práctico se asumen los siguientes **requisitos funcionales**:

1. El texto a procesar se encuentra en un archivo de texto. El nombre de este archivo se proporciona por la línea de comandos como primer argumento.
2. El programa debe ser capaz de identificar e imprimir cuántas veces se encuentra un determinado patrón en el contenido del archivo previamente mencionado. El patrón se proporciona por la línea de comandos como segundo argumento.
3. El programa debe ofrecer un control de errores mínimo. Particularmente, si no se proporciona un nombre de archivo o no se especifica el patrón a buscar, entonces el programa debe mostrar un mensaje de error y finalizar.

El listado 1.7 muestra una posible **implementación en C** del problema planteado, de acuerdo a los requisitos preestablecidos.

Listado 1.7: Implementación en lenguaje C del programa que busca un patrón de texto

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include <unistd.h>
5
6 int main (int argc, char *argv[]) {
7     FILE *fp;
8     char *line, *pattern;
9     int occurrences;
10    size_t len;
11    ssize_t read;
12
13    if (argc < 3) {
14        fprintf(stderr, "Error! You need to provide the filepath and the pattern.\n");
15        fprintf(stderr, "Synopsis: ./02_simple_patterns_error_control <file> <pattern>\n");
16        exit(EXIT_FAILURE);
17    }
18
19    pattern = argv[2];
20    occurrences = 0;
21    len = 0;
22
23    if ((fp = fopen(argv[1], "r")) == NULL) {
24        fprintf(stderr, "Error opening file %s!\n", argv[1]);
25        exit(EXIT_FAILURE);
26    }
27
28    while ((read = getline(&line, &len, fp)) != -1) {
29        if (strstr(line, pattern) != NULL) {
30            occurrences++;
31        }
32    }
33
34
35    fclose(fp);
36
37    printf("The word %s appeared %d times in the file %s\n",
38          argv[2], occurrences, argv[1]);
39
40    return 0;
41 }
```

A continuación, se exponen algunas reflexiones con respecto a las **características deseables** previamente comentadas:

1. Simplicidad. El código introduce una cierta complejidad inherente a la hora de manejar el lenguaje C, como por ejemplo el uso de punteros (variables que almacenan direcciones de memoria) para gestionar las líneas del archivo, el patrón y la propia línea de comandos. El programador tendría que estar familiarizado con sus fundamentos y con el uso de los operadores unarios * y &.

2. Capacidad de prototipado rápido. En el caso del lenguaje C, esta característica está acoplada al nivel de experiencia del programador. El hecho de utilizar un lenguaje compilado, implica la re-compilación del código cada vez que se introduce un nuevo cambio. Adicionalmente, ha sido posible emplear funciones existentes para leer el contenido del archivo y buscar un patrón, respectivamente, pero quizás se echa en falta una mayor agilidad a la hora de procesar archivos y controlar errores.
3. Legibilidad. A nivel global, el código resulta legible y el flujo principal se sigue gracias a los bloques de código definidos. No obstante, la llamada a la función getline() en la línea **②8** no ofrece un alto nivel de abstracción. El código relativo al control de errores no está demasiado cerca, desde el punto de vista semántico, de lo que realmente se pretende controlar, es decir, de si se proporcionaron un nombre de archivo y un patrón.

Por otro lado, el listado 1.8 muestra una posible **implementación en Python** del problema planteado, de acuerdo a los requisitos pre establecidos.

Listado 1.8: Implementación en Python del programa que busca un patrón de texto

```
1 #!/usr/bin/python3
2
3 # Import module sys
4 import sys
5
6 # Variable to store how many times the pattern was found
7 occurrences = 0
8
9 try:
10     # Open the file whose name is specified in the first argument
11     with open(sys.argv[1]) as f:
12         for line in f:
13             # The words of every line are stored as a list
14             words = line.split()
15             # Check for matches
16             for w in words:
17                 # The pattern to be found is specified in sys.argv[2]
18                 if w == sys.argv[2]:
19                     occurrences += 1
20
21     print("The word", sys.argv[2], "appeared",
22           occurrences, "times in the file", sys.argv[1])
23     f.close()
24
25 except IndexError:
26     print("Error! You need to provide the filepath and the pattern.")
27     print("Sinopsis: python3 02_simple_patterns_error_control <file> <pattern>")
```

A continuación, se exponen algunas reflexiones con respecto a las **características deseables** previamente comentadas:

1. Simplicidad. El código es sencillo y compacto, incluso aunque en esta solución se hayan introducido dos bucles for anidados. El número total de líneas es 27, en comparación a las 41 de la solución en lenguaje C.
2. Capacidad de prototipado rápido. En el caso del lenguaje Python, la curva de aprendizaje es asequible, y resulta posible escribir programas como el del listado 1.8 con cierta agilidad. Hubiera sido posible utilizar directamente el intérprete de Python para prototipar el código.
3. Legibilidad. A nivel global, el código resulta muy legible y el flujo principal se sigue gracias a los bloques de código definidos. Note cómo la función de apertura de archivos condiciona, y facilita, la estructura del código. El control de errores mediante bloques try-except también incrementa el nivel semántico en comparación a la solución anterior en C.

Con respecto a las características deseables vinculadas a la existencia de bibliotecas y a la comunidad de desarrollo, el presente caso práctico no resulta lo suficientemente específico para establecer una comparativa más concreta. Por una parte, las bibliotecas utilizadas en las soluciones planteadas forman parte de los estándares de los lenguajes C y Python. Por otra parte, resultaría sencillo identificar tanto en libros académicos como en portales web orientados al desarrollo problemas similares al aquí abordado.



Más allá de C y Python. En este punto, se invita al lector a desarrollar una solución al problema discutido en este apartado, empleando un lenguaje de programación diferente a los aquí utilizados.

1.2.4. Resumen

En esta sección se han identificado y discutido 5 características específicas que han de considerarse a la hora de utilizar un lenguaje de programación en el contexto de un proyecto de IA. Estas características son las siguientes: i) simplicidad, ii) capacidad de prototipado rápido, iii) legibilidad, iv) existencia de bibliotecas para IA, y v) comunidad de desarrollo. Esta discusión se ha completado con un caso práctico, vinculado a la búsqueda de patrones de texto e implementado en los lenguajes C y Python.

Por otro lado, se ha invitado al lector a que reflexione acerca de cómo enfocar el aprendizaje de los fundamentos de IA y la programación de software. Esta cuestión no ha de estar relacionada directamente con un determinado lenguaje de programación.

1.3. Lenguajes de programación de IA

En la sección 1.2 se discutieron 5 características relevantes a la hora de elegir un lenguaje de programación para IA: simplicidad, capacidad de prototipado rápido, legibilidad, existencia de bibliotecas para IA y comunidad de desarrollo. Esta sección pone el foco en dos de los lenguajes de programación más populares en el contexto de aplicaciones de IA, Python y R, al mismo tiempo que se pretende ilustrar al lector, mediante **ejemplos prácticos y código**, cómo estos lenguajes hacen gala de las características previamente mencionadas.

La sección 1.3.4 discute una solución en Python, concebida de **manera incremental**, para implementar la IA del **clásico juego piedra, papel o tijeras**. En ella no solo se refleja la potencia y simplicidad de Python como lenguaje para IA, sino que también sirve para mostrar algunas de las características generales del propio lenguaje.

1.3.1. Visión general

Tradicionalmente, y como se ha mencionado antes en este capítulo, los lenguajes de *scripting* han estado vinculados a la programación de aplicaciones de IA, debido a las características previamente expuestas y discutidas. No obstante, algunos lenguajes compilados también se usan de manera extensiva como lenguajes para IA, debido esencialmente a su popularidad, a la existencia de una gran comunidad de desarrolladores con experiencia en dichos lenguajes y a su relevancia histórica. Particularmente, los siguientes lenguajes han cobrado especial relevancia, dentro del contexto de desarrollo de IA, en los últimos años:

- **Python.** Probablemente, es el lenguaje de referencia para IA, y que será abordado, con más detalle y en el contexto de un ejemplo práctico, en la presente sección.
- **LISP (List Processing).** Definido, comúnmente, como el lenguaje para IA más antiguo. Actualmente, se utiliza principalmente para problemas de lógica y de aprendizaje automático.
- **PROLOG (Logic Programming).** Similar a LISP en cuanto a trascendencia y evolución históricas, se utiliza más frecuentemente para procesamiento de lenguaje natural.
- **R.** Lenguaje vinculado al desarrollo de IA debido a su facilidad para tratar y visualizar datos, combinando con el auge e importancia actual de *data science*.
- **Haskell.** Lenguaje funcional y de tipado estático, que se caracteriza por la importancia en la detección de errores en tiempo de compilación. Esta decisión de diseño es uno de los rasgos más relevantes del lenguaje desde su concepción.

- **Java.** Lenguaje de referencia para multitud de dominios y aplicaciones. Debido a su importancia histórica y popularidad, siempre será considerado una opción a la hora de desarrollar un proyecto de IA.
- **C++.** Se trata, probablemente, de uno de los lenguajes de programación más completo y versátiles en la actualidad. Los motivos de usar C++ como lenguaje para IA son similares a los expuestos para el caso de Java, aunque se debe considerar el incremento notable de rendimiento y las capacidades de desarrollo a medio nivel.

1.3.2. Python como lenguaje de referencia para IA

Los lenguajes de programación para IA van y vienen. Sin embargo, Python se mantiene como uno de los principales lenguajes de referencia debido a dos características principales: i) su **simplicidad** como entorno de programación, y ii) la **accesibilidad** que ofrece en términos de funcionalidad ya existente. Estas dos características representan el núcleo de las 5 características deseables de un lenguaje de programación para IA que se han abordado en este libro.

Debido a su naturaleza interpretada, es posible instalar y ejecutar Python en las principales plataformas de desarrollo. No existe la necesidad de disponer de un compilador nativo para traducir las instrucciones, ya que el intérprete de Python se encarga de ello. Además, Python es un lenguaje fácil de usar y entender, incluso para personas que no tienen una formación técnica en el ámbito del desarrollo software. A modo de ejemplo, y desde el punto de vista de la sintaxis del lenguaje, no hay necesidad de puntos y comas constantes, ni de largas declaraciones de tipos, ni de reinventar funciones comunes.

Pero, probablemente, una de las ventajas más importantes de usar Python deriva de su **amplio ecosistema funcional**. Existe todo un conjunto de bibliotecas externas que facilitan la vida a los desarrolladores de aplicaciones. A la hora de abordar un desarrollo de IA, será prácticamente seguro que ya exista una biblioteca en Python que dé soporte, al menos a nivel general.

Fundamentos del lenguaje

Python es un lenguaje de programación **multi-paradigma**, donde la programación estructurada (particularmente procedural) y la POO (Programación Orientada a Objetos) tienen mayor presencia que respecto a la programación funcional. Además, Python es un lenguaje basado en el **tipado dinámico**, es decir, la comprobación de los tipos de datos se hace en tiempo de ejecución. Por otro lado, la gestión de memoria está delegada en un **recolector de basura**. La biblioteca estándar de Python ofrece un conjunto muy amplio de estructuras de datos y de funcionalidad, que hace que a Python se le asocie el término de *batteries included*.

A continuación, se incluye un listado de las principales características de Python:

- **Sintaxis elegante**, que facilita la escritura y lectura de programas.
- **Facilidad de uso**, que convierte a Python en un lenguaje ideal para prototipar y construir soluciones a medida, sin sacrificar el rendimiento.
- **Amplia biblioteca estándar**, que da soporte a las tareas de desarrollo más comunes.
- **Lenguaje pegamento**, que simplifica la extensión del mismo mediante el desarrollo de nuevos módulos escritos en lenguajes compilados, como C o C++.
- **Modo interactivo**, que facilita la prueba de pequeños fragmentos de código.
- **Alta portabilidad**, siendo posible ejecutar Python en Windows, Linux, Mac OS X, y Unix.
- **Licencia *open-source***. Tanto el software de Python como su documentación está licenciado mediante el PSF (*Python Software Foundation*) License Agreement⁵, por lo que es posible usar y modificar Python en proyectos comerciales.

Por otro lado, algunas de las características más destacables, a nivel de lenguaje de programación, son las siguientes:

- **Estructuras de datos variadas y de alto nivel**, destacando especialmente las listas y los diccionarios.
- **Tipado fuerte y dinámico**, arrojando excepciones cuando en una misma operación se mezclan dos tipos de datos diferentes (por ejemplo, una cadena y un número).
- **Todo es un objeto**, lo cual ofrece un enfoque homogéneo y consistente.
- **Soporte para POO**, incluyendo herencia múltiple.
- **Características avanzadas**, como *generators* y *list comprehensions*.
- **Gestión automática de memoria**, de forma que el desarrollador se despreocupe de la asignación y liberación de memoria.

⁵<https://docs.python.org/3/license.html#psf-license>

Módulos Python para IA

El listado de módulos o bibliotecas que Python ofrece puede resultar abrumador. El lector solo tiene que ejecutar la ayuda interactiva del intérprete de Python y teclear *modules* para comprobarlo:

```
$ python  
>>> help()  
  
Welcome to Python help utility!  
  
help> modules
```

En el ámbito particular de bibliotecas Python para IA, destacan los que se exponen a continuación, los cuales se han clasificado en grandes categorías vinculadas a las temáticas más populares cuando se aplican técnicas de IA:

■ Redes neuronales.

- FANN (Fast Artificial Neural Network)⁶. Biblioteca de redes neuronales de código abierto, que permite utilizar redes neuronales artificiales multi-capa. Es multi-plataforma y ofrece un entorno de trabajo que facilita el proceso de entrenamiento de la red neuronal. Además de proporcionar binding para Python, es posible utilizar FANN con más de una decena de lenguajes de programación.
- ffnet⁷. Biblioteca de IA en Python para implementar redes neuronales pre-alimentadas (*feed-forward*). Los datos de entrenamiento se pueden visualizar mediante una interfaz gráfica de usuario específica. Integra una función de normalización automática de datos, que ahorra tiempo en la etapa de preprocesamiento. ffnet implementa sus funciones principales en Fortran, aumentando el rendimiento en comparación con soluciones que están implementadas en Python de manera nativa.
- PyTorch⁸. Biblioteca optimizada para *deep learning* que hace uso de GPUs y CPUs. PyTorch se puede entender como un sustituto del paquete NumPy⁹ que se aprovecha de la capacidad computacional ofrecidas por las GPUs, en lugar de utilizar CPUs.

⁶<https://github.com/libfann/fann>

⁷<https://github.com/mrkwjc/ffnet>

⁸<https://pytorch.org/>

⁹<https://numpy.org/>

■ Aprendizaje automático.

- scikit-learn¹⁰. Biblioteca de IA en Python, construida sobre las bibliotecas NumPy, SciPy¹¹ y matplotlib¹², que da soporte a la implementación de algoritmos de aprendizaje automático. Tiene algoritmos incorporados para clasificar objetos, construir regresiones, agrupar objetos similares en conjuntos (*clustering*), reducir la cantidad de variables aleatorias, preprocesar datos e incluso elegir el modelo de aprendizaje final.
- TensorFlow¹³. Plataforma de código abierto que ofrece mecanismos de soporte para el aprendizaje automático, incluyendo *deep learning*. TensorFlow ofrece un ecosistema integral y flexible de herramientas, bibliotecas y recursos diseñados para crear aplicaciones de *machine learning*. Aunque los cálculos se pueden expresar en Python, internamente se ejecutan en C++ para incrementar el rendimiento, ahorrando tiempo de ejecución y aumentando la velocidad de los programas desarrollados con esta biblioteca.
- Keras¹⁴. API diseñada con un foco especial en la usabilidad, ofreciendo mecanismos para reducir la carga cognitiva y APIs consistentes y simples. Uno de sus principales objetivos es el de minimizar el número de acciones del usuario requeridas para desarrollar los casos de uso comunes. Además, facilita enormemente el proceso de depuración, cuenta con una amplia documentación y es uno de los *frameworks* para *deep learning* más populares.

■ Procesamiento de lenguaje natural.

- NLTK (Natural Language Toolkit)¹⁵. Biblioteca de propósito general para el procesamiento de texto escrita en Python. Simplifica el procesamiento lingüístico y ofrece funcionalidad para gestionar la *tokenización* y el etiquetado de texto. Además, permite la identificación de entidades con nombre e incluso la visualización de árboles de análisis sintáctico.
- Gensim¹⁶. Biblioteca gratuita de código abierto en Python para representar documentos como vectores semánticos, de una eficiente, en términos de consumo de recursos, y sencilla (para facilitar su procesamiento). Gensim está diseñado para procesar textos digitales sin estructurar (texto plano) utilizando algoritmos de aprendizaje automático no supervisado.

¹⁰<https://scikit-learn.org/stable/>

¹¹<https://www.scipy.org/>

¹²<https://matplotlib.org/>

¹³<https://www.tensorflow.org/>

¹⁴<https://keras.io/>

¹⁵<http://www.nltk.org/>

¹⁶<https://radimrehurek.com/gensim/>

- spaCy¹⁷. Biblioteca gratuita y de código abierto para el procesamiento avanzado del lenguaje natural en Python. spaCy está diseñado específicamente para su uso en producción y ayuda a crear aplicaciones que procesan y *comprenden* grandes volúmenes de texto. Puede utilizarse para construir sistemas de extracción de información o de comprensión del lenguaje natural, o para preprocesar texto para algoritmos de *deep learning*.

■ **Modelado y caracterización de sistemas expertos.**

- PyCLIPS¹⁸. Extensión para el lenguaje Python que incorpora la funcionalidad completa de CLIPS¹⁹ en aplicaciones de Python. Así, es posible dotar a Python de un motor de inferencia sólido, fiable, ampliamente utilizado y bien documentado. CLIPS representa una de las implementaciones de referencia en lo que se refiere a sistemas expertos. Tiene un sistema de inferencia basado en reglas de encadenamiento hacia adelante, así como todas las construcciones imperativas y orientadas a objetos, lo cual permite un control total del flujo de ejecución.
- Experta²⁰. Motor de reglas que empareja un conjunto de hechos con un conjunto de reglas basadas en esos hechos. A continuación, se ejecutan acciones basadas en estas reglas. Todos los hechos y las reglas son gestionados por el motor de conocimiento, responsable de generar las salidas. Al igual que ocurre con PyCLIPS, Experta también está inspirado en CLIPS.

■ **Visión por computador.**

- OPENCV (Open Source Computer Vision Library)²¹. Biblioteca de referencia para el desarrollo de aplicaciones de visión computador, la cual ofrece *bindings* para los lenguajes de programación más populares, incluyendo Python. Se puede definir como una biblioteca de software de visión por computador y aprendizaje automático de código abierto. OPENCV se construyó para proporcionar una infraestructura común para las aplicaciones de visión por computador y para acelerar el uso de la percepción automática en los productos comerciales. La biblioteca cuenta con más de 2.500 algoritmos optimizados, que incluyen un amplio conjunto de algoritmos de visión por ordenador y de aprendizaje automático, tanto clásicos como de última generación. Estos algoritmos pueden utilizarse para detectar y reconocer rostros, identificar objetos, clasificar acciones humanas en vídeos, seguir los movimientos de la cámara, rastrear objetos en movimiento, extraer modelos 3D de objetos, producir nubes de puntos 3D a partir de cámaras estereoscópicas, unir imágenes para producir una imagen de alta resolución de

¹⁷<https://spacy.io/>

¹⁸<http://pyclips.sourceforge.net/web/>

¹⁹<http://www.clipsrules.net/>

²⁰<https://pypi.org/project/experta/>

²¹<https://opencv.org/>

toda una escena, encontrar imágenes similares a partir de una base de datos de imágenes, eliminar los ojos rojos de las imágenes tomadas con flash, seguir los movimientos de los ojos, reconocer paisajes y establecer marcadores para superponerlos a la realidad aumentada, entre otras cuestiones.

1.3.3. R: más allá de la estadística

R es un lenguaje y un entorno de desarrollo que pone un énfasis especial en dos ámbitos concretos: la **estadística** y los **gráficos**²². R ofrece soporte para un amplio abanico de i) técnicas estadísticas, como por ejemplo la modelización lineal y no lineal, pruebas estadísticas clásicas, análisis de series temporales, clasificación, o agrupación, y ii) gráficas. R se caracteriza, además, por ser muy extensible. Uno de los puntos fuertes de R es la facilidad con la que puede generar gráficos de calidad, incluyendo símbolos matemáticos y fórmulas cuando sea necesario. R está disponible como software libre bajo los términos de la licencia GPL (GNU General Public License) de la FSF (Free Software Foundation). Además, R se puede compilar y ejecutar en una amplia variedad de plataformas UNIX, Windows y MacOS.

En esta sección se ofrece una introducción general a R, haciendo especial hincapié en sus principales características como lenguaje orientado al procesamiento y visualización de datos.



Figura 1.11: Logo oficial del proyecto R.

Fundamentos del lenguaje

R se puede definir como un **conjunto integrado de programas** diseñados para la manipulación de datos, el cálculo y la visualización gráfica. Entre otras cuestiones, R se caracteriza por los siguientes aspectos:

- un eficaz sistema de manipulación y almacenamiento de datos,
- un conjunto de operadores para el cálculo de arrays, en particular de matrices,
- una colección amplia, coherente e integrada de herramientas para el análisis de datos,
- un sistema gráfico para el análisis y la visualización de datos,

²²<https://www.r-project.org/>

- un lenguaje de programación bien desarrollado, sencillo y eficaz (basado en el lenguaje S), que incluye sentencias condicionales, bucles, funciones recursivas definidas por el usuario y funciones de entrada y salida.

El término *entorno* pretende caracterizar R como un sistema totalmente planificado y coherente, en lugar de una agrupación incremental de herramientas específicas y con baja flexibilidad, como suele ocurrir con otros programas de análisis de datos. R representa, en gran medida, un vehículo para el **desarrollo de nuevos métodos de análisis de datos interactivos**. Se ha desarrollado rápidamente y, en los últimos años, se ha ampliado con una gran colección de paquetes. Sin embargo, la mayoría de los programas escritos en R son esencialmente específicos, creados para un único análisis de datos con unas características concretas.

¿Por qué usar R?

En este apartado se ofrece al lector un listado acotado de características por las considerar el uso de R, prestando especial atención a perfiles del tipo *data analyst*, *data scientist* y *business analyst*. En este sentido, R es uno de los lenguajes de scripting más demandado en la actualidad en el **ámbito data science**.

- Nivel de estandarización. R se considera una herramienta estándar para llevar a cabo tareas estadísticas y de análisis de datos.
- Capacidad de visualización. R facilita enormemente la generación de gráficos de alta calidad.
- Captura de datos. R ofrece mecanismos y funcionalidad para simplificar el proceso de captura de datos en múltiples fuentes de información.
- Portabilidad. R se puede ejecutar en los sistemas operativos más populares en la actualidad.
- Extensibilidad. R se puede integrar con diversas tecnologías de almacenamiento de datos, y facilita la inclusión de paquetes con funcionalidad específica.
- Libertad de uso. R está liberado bajo licencia GNU, incrementando su accesibilidad y disponibilidad para proyectos de desarrollo.
- Relevancia en dominios clave. En campos como la sanidad, la banca o el comercio electrónico, entre otros, R representa una de las herramientas de trabajo más populares en la actualidad.
- Nexo con el aprendizaje automático. R se puede utilizar como base tecnológica para desarrollar soluciones en ámbitos donde técnicas como el análisis predictivo, el análisis de sentimientos u otros métodos de aprendizaje automático tienen cabida. Por ejemplo, Facebook usa R para realizar análisis predictivo y de sentimientos en lo que se refiere a sus usuarios.



Popularidad de R. De acuerdo al índice TIOBE, el lenguaje R alcanzó su pico de popularidad en 2020, el cual ha ido bajando conforme ha pasado el tiempo. Sin embargo, se considera una de las opciones más acertadas en proyectos de IA donde el análisis y la visualización de datos cobran especial importancia.

Caso práctico en R: Inferencia difusa y representación gráfica

En la sección 1.1.1 se introdujo el concepto de lógica difusa como herramienta para razonar con valores vagos o imprecisos [Zad99], en el contexto del uso de metáforas que acercan la forma en la que los humanos pensamos a los mecanismos de representación de información que pueden ser utilizados por las máquinas. En este apartado, se discute un caso práctico que hace uso de la lógica difusa para medir el grado de esfuerzo que un paciente de rehabilitación física necesita aplicar conforme su terapia avanza, de acuerdo a un conjunto de variables de entrada. Esta discusión se materializa con una posible implementación en R. No se pretende ofrecer una discusión detallada de las características de este lenguaje ni de los fundamentos de la lógica difusa. Lo que se persigue, principalmente, es poner de manifiesto la **fácilidad con la que es posible manejar y representar, visualmente, información con R**.

Suponga una terapia de rehabilitación física donde un paciente tiene como objetivo recuperar la movilidad en un brazo. El paciente tiene asignada una rutina diaria, compuesta de un número arbitrario de ejercicios que debe realizar. Cada ejercicio tendrá asociado un número de repeticiones. Idealmente, el paciente debería ejecutar cada ejercicio de la forma más parecida posible a cómo lo haría su terapeuta. Simplificando, esta relación de parecido tiene una dimensión espacial, de forma que el paciente siga la misma trayectoria que la definida por su terapeuta, y una dimensión temporal, de forma que el paciente realice el ejercicio en un tiempo razonable. La figura 1.12 muestra una comparación visual de las trayectorias realizadas por un paciente y un terapeuta.

Se pretende construir un sistema artificial que sea capaz de monitorizar cómo de adecuada es la ejecución de una rutina de rehabilitación, de forma que sea posible detectar cuándo el paciente se está esforzando en exceso o, por el contrario, con demasiada facilidad. En este último caso, tendría sentido asignarle un ejercicio, o rutina, más ambicioso que fomente la recuperación de movilidad en el miembro a rehabilitar.

Para representar tanto las variables que conforman el sistema como su núcleo de inferencia, se opta por utilizar un sistema basado en lógica difusa. En esencia, la lógica difusa permite que una máquina razonne gestionando valores imprecisos. El clásico ejemplo que se utiliza cuando se estudia lógica difusa es el de la altura. Por ejemplo, usted usaría, comúnmente, una expresión del tipo “esa persona es *muy alta*”, en lugar de afirmar “esa persona mide 1,90 metros”. También usaría expresiones del tipo “si una persona es *muy alta*, entonces está en *buenas* condiciones para jugar a baloncesto”. En términos de uso de lógica difusa, la **variable** que se

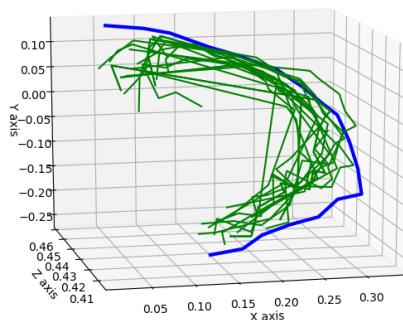


Figura 1.12: Representación en el espacio 3D de las trayectorias asociadas a un ejercicio de rehabilitación por parte de un paciente (verde) y un terapeuta (azul).

estaría definiendo es la altura, empleando para ello una serie de **etiquetas difusas**, como *alto* o *muy alto*, que tendrían asociadas sendos **conjuntos difusos** con el rango de valores que las definen. Por otro lado, también es posible definir **reglas difusas del tipo si-entonces**, usando las variables y etiquetas difusas anteriores, para establecer así un sistema de inferencia difusa.



Razonamiento aproximado. La lógica difusa permite resolver problemas reales de forma imprecisa con el uso de términos lingüísticos.

De este modo, el sistema artificial discutido en este apartado constaría de tres variables espacio-temporales de entrada (las dos primeras relativas al espacio y la tercera al tiempo), a través de las cuales se monitoriza la ejecución de una rutina por parte de los pacientes. También integraría una variable de salida que determina el grado de adecuación previamente introducido. En particular, las **variables de entrada** son las siguientes:

- Número de pasos (*number steps*). Determina la diferencia entre el número de pasos para culminar una rutina, considerando la ejecución del terapeuta (sujeto sano) y el paciente.
- Desviación acumulada (*accumulated deviation*). Representa la desviación espacial acumulada entre paciente y terapeuta. Cuanto más distinto es el recorrido de movimientos realizado por el paciente con respecto al terapeuta, mayor será el valor de esta variable. Por el contrario, una desviación acumulada baja significa que el recorrido del paciente se ajusta en gran medida al definido por el terapeuta.
- Tiempo invertido (*time*). Expresa la diferencia de tiempo invertido en la ejecución de la rutina entre paciente y terapeuta.

Por otro lado, existe una **variable de salida** cuyo significado es el siguiente:

- Grado de adecuación (*adequacy*). Determina el grado de adecuación de la rutina a la situación particular del paciente. Cuanto mayor es el valor de esta variable, mayor es el grado de adecuación.

La figura 1.13 representa, visualmente, la definición de cada una de las variables en su universo particular. Cada universo de discurso representa el rango de valores que la variable puede tomar y, en lo que atañe a este caso práctico, cada uno de ellos consta de cinco conjuntos difusos cuyas etiquetas lingüísticas son VL (*very low*), L (*low*), M (*medium*), H (*high*) y VH (*very high*). Cada uno de los conjuntos tiene asociado una función miembro, que determina el grado de pertenencia de un caso en función del valor de la variable de entrada. Según sea el valor de la variable, un determinado caso puede pertenecer a distintos conjuntos con diferentes grados de pertenencia. Es importante destacar que la figura 1.13 se ha generado automáticamente gracias a la funcionalidad ofrecida por R y, particularmente, por el paquete *sets*²³. Dicho paquete integra estructuras de datos y operaciones básicas tanto para conjuntos ordinarios como para conjuntos difusos.

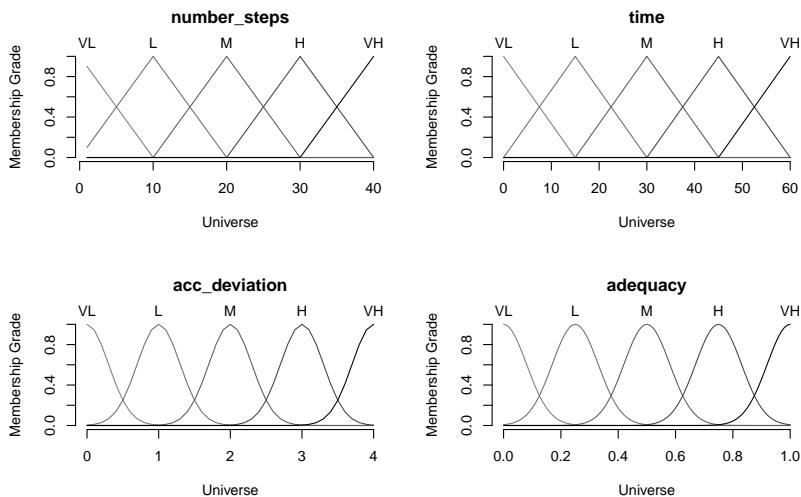


Figura 1.13: Representación gráfica de las variables difusas utilizadas en el caso práctico.

²³<https://cran.r-project.org/web/packages/sets/sets.pdf>



Funciones para conjuntos difusos. Para el lector que tenga conocimientos de lógica difusa, o para el que desee profundizar en este tema, resulta interesante destacar que se ha optado por funciones triangulares en aquellas variables enteras cuyos valores varían de forma discreta, mientras que se ha optado por funciones sinusoidales para aquellas cuya variación es más gradual y varían de forma continua. De esta forma, se amplía el rango de valores del dominio que implican un grado de pertenencia alto a cada conjunto (cima del conjunto), y el cambio entre conjuntos es más suave.

En este sentido, el código del listado 1.9 muestra la definición de los universos del discurso (líneas **④ 6-9**) asociados a las variables previamente descritas (líneas **④ 12-28**).

Listado 1.9: Definición en R de las variables *number_steps*, *acc_deviation*, *time* y *adequacy*.

```

1 # https://cran.r-project.org/web/packages/sets/sets.pdf
2 library(sets)
3
4 # Universes of discourse
5 # Range of values that the fuzzy variables may take
6 unv_number_steps = seq(from=1.0, to=40.0, by=1.0)
7 unv_acc_deviation = seq(from=0.0, to=4.0, by=0.1)
8 unv_time = seq(from=0.0, to=60.0, by=1.0)
9 unv_adequacy = seq(from=0.0, to=1.0, by=0.01)
10
11 # Set of variables
12 variables <- set(
13   number_steps = fuzzy_partition(
14     varnames=c(VL=0.0, L=10.0, M=20.0, H=30.0, VH=40.0),
15     FUN=fuzzy_cone, radius=10, universe=unv_number_steps),
16
17   acc_deviation = fuzzy_partition(
18     varnames= c(VL=0.0, L=1.0, M=2.0, H=3.0, VH=4.0),
19     sd=0.3, universe=unv_acc_deviation),
20
21   time = fuzzy_partition(
22     varnames= c(VL=0.0, L=15.0, M=30.0, H=45.0, VH=60.0),
23     FUN=fuzzy_cone, radius=15, universe=unv_time),
24
25   adequacy = fuzzy_partition(
26     varnames=c(VL=0, L=0.25, M=0.5, H=0.75, VH=1.0),
27     sd=0.08, universe=unv_adequacy)
28 )

```

El código para obtener la representación visual de estos conjuntos es realmente sencillo, a la par que útil. Este se muestra en el listado 1.10, en el cual se ejemplifica cómo se puede instanciar un modelo difuso en R y la facilidad para obtener una representación visual de las variables que lo conforman.

Listado 1.10: Renderizado de las variables *number_steps*, *acc_deviation*, *time* y *adequacy*.

```

1 model <- fuzzy_system (variables, rules)
2 print(model)
3 plot(model)

```

Volviendo a la lógica del dominio del caso práctico, la tabla 1.1 muestra la situación que representan cada una de las etiquetas situadas en los extremos, para cada una de las variables contempladas en el sistema diseñado.

Variable	VL (very low)	VH (very high)
Número de pasos	El paciente necesita casi los mismos pasos que el terapeuta para realizar una rutina	El paciente necesita muchos más pasos que el terapeuta para realizar una rutina
Desviación acumulada	El recorrido del paciente y terapeuta son muy similares	El recorrido y movimientos que hace el paciente, dista en gran medida de los del terapeuta.
Tiempo	El tiempo invertido por el paciente es muy similar al del terapeuta.	El paciente ha invertido mucho más tiempo que el terapeuta en realizar el ejercicio.
Adecuación (salida)	La rutina no se adapta a la situación del paciente	La rutina se ajusta perfectamente a la situación del paciente.

Tabla 1.1: Situación que representa cada etiqueta extrema en cada una de las variables

Por otro lado, el sistema incluye un conjunto de reglas difusas de tipo IF-THEN que clasifican el grado de adecuación en función de los tres valores de entrada. De hecho, uno de los motores de inferencia más comunes es el basado en reglas difusas de tipo SI-ENTONCES. Este tipo de reglas está formado por un conjunto de antecedentes conectados mediante operadores lógicos (AND, OR, NOT) y uno o varios consecuentes. Cada antecedente a su vez consiste en una variable difusa y el rango de etiquetas lingüísticas que esta debería tomar para que se produzca la activación de la regla.



Reglas SI-ENTONCES. SI distancia {muy corta, corta} Y fuerza_enemigo es {baja, muy baja, media} ENTONCES Ataque {agresivo}. En el contexto de la IA de un bot en un videojuego, a modo de ejemplo, para que se active la regla anterior la distancia entre el enemigo y el personaje principal debe tener un grado de pertenencia superior a cero en los conjuntos, *corta* o *muy corta*; lo mismo sucede con la fuerza del enemigo, en donde al menos se debe dar uno de los tres casos siguientes: *baja*, *muy baja* o *media*. En caso afirmativo, la regla se activa y el consecuente es atacar con un determinado grado de pertenencia a ataque *agresivo*.

Algunos ejemplos de las reglas contempladas en el presente caso de estudio se indican a continuación:

- IF *number_steps* is VL AND *acummulated_deviation* is VL THEN *adequacy* is VL.
- IF *number_steps* is VL AND *time* is VL THEN *adequacy* is VL.
- IF *acummulated_deviation* is L AND *time* is L THEN *adequacy* is L.
- IF *acummulated_deviation* is VH THEN *adequacy* is VL.
- IF *number_steps* is M AND *acummulated_deviation* is M AND *time* is M THEN *adequacy* is M.

El listado de código 1.11 muestra cómo es posible usar R y el paquete *sets* para llevar a cabo la representación de algunas de estas reglas, las cuales modelan el núcleo de razonamiento del sistema.

Listado 1.11: Representación de reglas difusas IF-THEN en R.

```

1 # Set of rules
2 rules <- set(
3   fuzzy_rule(number_steps %is% VH && acc_deviatiion %is% VH && time %is% VH,
4             adequacy %is% VL),
5   fuzzy_rule(number_steps %is% VL && acc_deviatiion %is% VL && time %is% VL,
6             adequacy %is% VL),
7   fuzzy_rule(number_steps %is% L && acc_deviatiion %is% L && time %is% L,
8             adequacy %is% L),
9   fuzzy_rule(number_steps %is% H && acc_deviatiion %is% H && time %is% H,
10            adequacy %is% H),
11  fuzzy_rule(number_steps %is% M && acc_deviatiion %is% M && time %is% M,
12            adequacy %is% M),
13
14  # IF the accumulated deviation is VH,
15  # THEN the patient is not performing the routine properly.
16  fuzzy_rule(acc_deviatiion %is% VH,
17              adequacy %is% VL),
18
19  # More rules here...
20 )

```

Finalmente, el **resultado ofrecido por el sistema de razonamiento difuso**, vinculado al concepto de salida difusa, se calcula mediante el método de inferencia de Mamdani [Mam74]. Particularmente, se hace uso de una defuzzificación, o conversión de un valor difuso a un valor numérico (conocido como *crisp*), mediante el denominado método del centroide [KM01]. Este método permite obtener un valor numérico que representa el grado de adecuación que se desea obtener para evaluar la propia adecuación de la rutina de rehabilitación al paciente. El proceso de defuzzificación es necesario sólo si se desea un valor real o *crisp* de salida en el sistema propuesto. El proceso más sencillo y ampliamente utilizado se conoce como método de *centro de gravedad*, en el que se obtiene el punto central del área generada en el consecuente de la regla.

Todas aquellas ejecuciones de rutinas que sean catalogadas con un grado de adecuación VL (*very low*) o L (*bajo*), generarían alertas que serían supervisadas por el terapeuta responsable de la evolución del paciente, siendo aquellas con un grado de adecuación VL las que tienen una prioridad mayor. De esta forma, el terapeuta puede centrarse directamente en anomalías que se produzcan durante el proceso de rehabilitación y tomar medidas en consecuencia. En este sentido, el terapeuta cambiaría o ajustaría con respecto a un conjunto de rutinas que se adapten mejor al estado actual del paciente.

A continuación, se muestran **dos ejemplos que representan ejecuciones distintas** de rutinas, reflejando el comportamiento interno del sistema.

En el primer ejemplo, el paciente no necesita muchos más pasos que el terapeuta para realizar la rutina, invierte un tiempo medio, pero la trayectoria vinculada a los ejercicios de rehabilitación difieren considerablemente entre lo que hace el terapeuta y el paciente. Unos valores de entrada que podrían representar tal situación serían los siguientes: *number_steps* = 8, *time* = 30 y *acc_deviation* = 3,6.

La figura 1.14 representa la salida obtenida tras el proceso de inferencia difuso. Tal como se puede apreciar, el grado de adecuación es predominantemente *very_low*. Esto es debido principalmente a la existencia de la regla difusa

If accumulated deviation is VH then adequacy is VL

en la que se representa que una desviación acumulada demasiado alta implica un grado de adecuación bajo, independientemente del valor de cualquier otra variable. El valor de salida tras el proceso de defuzzificación es de 0.0773.

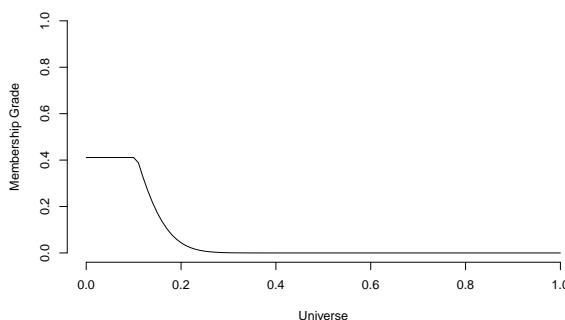


Figura 1.14: Salida difusa obtenida con respecto al primer ejemplo.

En el segundo ejemplo, el paciente necesita “unos cuantos” pasos más que el terapeuta para realizar la rutina, invierte “mucho más” tiempo, y la trayectoria entre paciente y terapeuta difiere, pero no en exceso. Unos valores de entrada que podrían representar tal situación serían los siguientes: *number_steps* = 23, *time* = 50,

acc_deviation = 1.5. Ante tal situación, el sistema genera una salida difusa que se muestra en la figura 1.15, con un valor de 0.5793. En dicha figura se puede apreciar como la situación podría ser catalogada con un grado de conveniencia entre *medio* y *alto*, estando más próximo a *medio*.

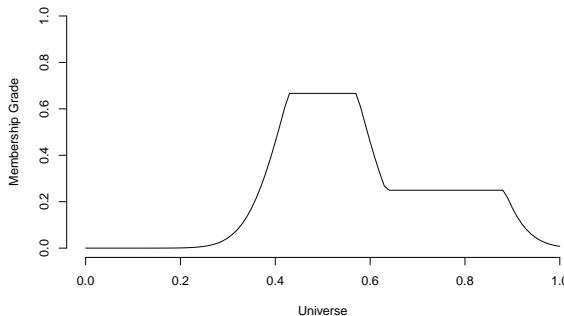


Figura 1.15: Salida difusa obtenida con respecto al segundo ejemplo.

El listado de código 1.12 muestra cómo recrear la inferencia vinculada a los dos ejemplos anteriores, incluyendo la generación de la salida difusa ofrecida por el sistema.

Listado 1.12: Representación de reglas difusas IF-THEN en R.

```

1 example.1 <- fuzzy_inference(model, list(number_steps = 8,
2                               time = 30, acc_deviation = 3.6))
3 gset_defuzzify(example.1, "centroid")
4 plot(example.1)
5
6 example.2 <- fuzzy_inference(model, list(number_steps = 23,
7                               time = 50, acc_deviation = 1.5))
8 gset_defuzzify(example.2, "centroid")
9 plot(example.2)
```

1.3.4. Caso práctico en Python: Piedra, Papel o Tijeras!

Esta sección discute un caso práctico en el que se plantea un desarrollo incremental del clásico juego *piedra, papel o tijeras*²⁴. Se trata de un juego de manos en el que participan dos jugadores. En cada ronda, cada jugador forma, simultáneamente, una de estas tres formas con la mano extendida. Estas formas son i) piedra, representada por un puño cerrado, ii) papel, representada mediante la mano abierta, y iii) tijeras, representada por un puño con los dedos índice y corazón extendidos,

²⁴En inglés, este juego se conoce como *rock paper scissors*.

formando una V. El juego solo tiene dos resultados posibles. Uno de ellos es el empate, si ambos jugadores eligen la misma forma. El otro se define por una situación en la que un jugador gana y el otro pierde. En este sentido, la piedra vence (o rompe) a las tijeras, el papel vence (o envuelve) a la piedra y, finalmente, las tijeras vencen (o cortan) al papel.



Figura 1.16: Representación visual de las tres opciones de juego para un jugador en piedra, papel o tijeras (foto tomada por <https://www.flickr.com/photos/marktee/>).

Desde un punto de vista más formal, y en el contexto de la teoría de juegos no competitivos, se trata de un **juego de suma cero**, ya que una situación de victoria para un jugador se equilibra, o se traduce, en una situación de derrota para el jugador contrario. La suma cero es un caso particular de uno más general, el de suma constante, donde los beneficios y las pérdidas de los jugadores suman el mismo valor. Esto se debe a que un jugador recibe como beneficio la misma cantidad que pierde el jugador contrario. Por otro lado, los juegos de suma no nula son aquellos en los que se producen situaciones en las que los jugadores pueden ganar o perder de manera simultánea.

Debido a que en cada ronda un jugador solo tiene 3 opciones de juego (piedra, papel o tijeras), es posible obtener una buena tasa de victorias jugando de manera puramente aleatoria. No obstante, y a diferencia de los métodos de selección verdaderamente aleatorios, este juego se puede jugar aplicando una estrategia basada en la detección y aprovechamiento del comportamiento no aleatorio del jugador contrario.

En los apartados que se incluyen a continuación se discute una **sencilla implementación** de este juego, **diseñada de forma incremental** mediante el lenguaje de programación Python con el objetivo de ofrecer al lector un punto de partida para modificar y mejorar el código fuente.

Código base y comportamiento aleatorio de la máquina

El listado 1.13 muestra la implementación de partida de una posible solución que recrea el juego *piedra, papel o tijeras*. Como se describirá más adelante, el **comportamiento** de la máquina en esta primera versión es **totalmente aleatorio**.

Listado 1.13: Implementación base del juego *piedra, papel o tijeras*.

```
1 #!/usr/bin/python3
2
3 import random
4
5
6 ROCK = 'rock'
7 PAPER = 'paper'
8 SCISSORS = 'scissors'
9
10
11 def assess_game(user_action, computer_action):
12     if user_action == computer_action:
13         print(f"User and computer picked {user_action}. Draw game!")
14
15     # You picked Rock
16     elif user_action == ROCK:
17         if computer_action == SCISSORS:
18             print("Rock smashes scissors. You won!")
19         else:
20             print("Paper covers rock. You lost!")
21
22     # You picked Paper
23     elif user_action == PAPER:
24         if computer_action == ROCK:
25             print("Paper covers rock. You won!")
26         else:
27             print("Scissors cuts paper. You lost!")
28
29     # You picked Scissors
30     elif user_action == SCISSORS:
31         if computer_action == ROCK:
32             print("Rock smashes scissors. You lost!")
33         else:
34             print("Scissors cuts paper. You won!")
35
36
37
38 def main():
39     game_actions = [ROCK, PAPER, SCISSORS]
40
41     while True:
42         user_action = input("\nPick a choice: rock, paper or scissors: ")
43         computer_action = random.choice(game_actions)
44
45         print(f"\nYou picked {user_action}. The computer picked {computer_action}\n")
46         assess_game(user_action, computer_action)
47
48
49 if __name__ == "__main__":
50     main()
```

Se invita al lector a descargar y probar el código mediante la siguiente instrucción, o directamente desde su editor visual de código.

```
$ python3 01_RPS_Basic.py
```

En este primera versión de código, solo existen dos funciones: i) *main()*, que comprende las líneas **④38-46** y que incluye el código de control para jugar infinitas rondas de juego, y ii) *assess_game()*, que está definida en las líneas **④11-34** y que es la responsable de mostrar el resultado de cada ronda de juego tras evaluar las decisiones tomadas por el jugador y por la máquina.

En la función *main()* destacan las siguientes cuestiones:

- Las posibles acciones de juego, modeladas mediante constantes en las líneas **④6**, **④7** y **④8**, respectivamente, se añaden a la lista *game_actions* (línea **④39**) para facilitar la generación aleatoria de las decisiones de la máquina. Esta generación se delega en la función *choice()* del módulo *random*, como se muestra en la línea **④43**.
- El usuario introduce su movimiento mediante una cadena de texto, después de que el programa se lo solicite. Para ello, se hace uso de la función *input()* en la línea **④42**. El programa espera que el jugador introduzca una de las siguientes cadenas de texto: *rock*, *paper* o *scissors*.
- Como se ha comentado anteriormente, la evaluación de la ronda de juego se delega en la función *assess_game()* en la línea **④46**, que recibe como argumentos la acción del jugador y la de la máquina. Recuerde que las acciones se han modelado mediante cadenas de texto.

Por otra parte, en la función *assess_game()* destacan los siguientes aspectos a nivel de código:

- La comparación de las decisiones del jugador y de la máquina se lleva a cabo mediante un conjunto de sentencias *if-else* anidadas.
- Al mismo tiempo que se realiza la evaluación, se muestra información por la salida estándar con respecto a dicha evaluación y a su justificación.

Añadiendo control de errores

El código discutido anteriormente no contemplaba una situación en la que el jugador introdujera, usando el teclado, una entrada de texto diferente a *rock*, *paper* o *scissors*. En este contexto, el listado 1.14 muestra el código extra añadido a la versión anterior para incluir un control de errores básico. A continuación, se listan los aspectos más destacables del nuevo código:

- Se ha definido la excepción *IncorrectOptionException*, modelada a través de una clase en Python que hereda de la clase *Exception*. Esta última clase la proporciona el lenguaje con el objetivo de que todas las excepciones definidas por el usuario hereden de la misma. La relación de herencia, tal y como se muestra en la línea **④ 1**, se expresa indicando entre paréntesis el nombre de la clase base. La clase derivada aparece después de la palabra clave *class*.
- En la línea **④ 15** se incluye una conversión explícita a minúsculas de la cadena de texto introducida por el jugador, utilizando para ello la función *lower()* vinculada a las cadenas de texto en Python. Esta adaptación permite gestionar situaciones en las que el jugador introduce ROCK o Rock, en lugar de rock, que es lo que espera el programa (tal y como se refleja en la línea **④ 6** del listado 1.13).
- La sentencia condicional que engloba las líneas **④ 16-17** comprueba, en primer lugar, si la cadena de texto introducida por el usuario es alguna de las tres almacenadas en la lista *game_actions*. Note la facilidad que ofrece Python para realizar esta comprobación sin la necesidad de iterar sobre la lista. En segundo lugar, si la cadena introducida no está en *game_actions*, entonces se crea y se lanza una excepción del tipo *IncorrectOptionException*.
- La parte relevante del código de la función *main()* se encuentra enmarcado en un bloque *try-except*, con el objetivo de capturar y gestionar una posible excepción del tipo *IncorrectOptionException*. La gestión, delegada de manera efectiva en el bloque *except* de las líneas **④ 22-23**, simplemente recuerda al jugador que debe introducir *rock*, *paper* o *scissors*.

Listado 1.14: Código de control de errores y uso de excepciones

```
1 class IncorrectOptionException(Exception):
2     pass
3
4
5 def assess_game(user_action, computer_action):
6     # Previous code here
7     # ...
8
9
10 def main():
11     game_actions = [ROCK, PAPER, SCISSORS]
12
13     while True:
14         try:
15             user_action = input("\nPick a choice: rock, paper or scissors: ").lower()
16             if user_action not in game_actions:
17                 raise IncorrectOptionException
18             computer_action = random.choice(game_actions)
19
20             print(f"\nYou picked {user_action}. The computer picked {computer_action}\n")
21             assess_game(user_action, computer_action)
22         except IncorrectOptionException:
23             print("\nYou can only picked rock, paper or scissors!")
```

Mejorando la mantenibilidad del código

Hasta ahora, el código discutido estaba más centrado en la lógica de dominio, en un contexto general de prototipar una solución rápida que permitiera jugar a *piedra, papel o tijeras* con la máquina. En este apartado, se abordan algunos **aspectos que aumentarán la calidad y mantenibilidad** del código, facilitando así potenciales modificaciones futuras.

En primer lugar, se introduce el uso de los **tipos enumerados en Python**²⁵. Una enumeración se puede definir como un conjunto de nombres simbólicos que están asociados, de manera unívoca, a valores constantes. En el contexto de una enumeración, los miembros o elementos de la misma se pueden comparar por su identidad. Además, es posible iterar sobre la propia enumeración.

En el código anterior, se hizo uso de 3 constantes para modelar las 3 acciones posibles: *rock*, *paper* o *scissors*. En este punto, se va a optar por el uso de un tipo enumerado, denominado *GameAction*, para representar dichas acciones. El listado 1.15 muestra el tipo enumerado *GameAction*, el cual hereda de la clase *enum.IntEnum*, definida en Python como clase base para crear constantes enumeradas que también son una subclase de *int*.

Listado 1.15: Tipo enumerado *GameAction*

```
1 from enum import IntEnum
2
3 class GameAction(IntEnum):
4     Rock = 0
5     Paper = 1
6     Scissors = 2
```

Por otro lado, resulta muy aconsejable estructurar el código del apartado anterior en funciones, con el objetivo de acotar responsabilidades y aumentar la mantenibilidad global del mismo. Es por ello que en la tercera versión de la solución planteada en esta sección se han incluido nuevas funciones, que serán llamadas desde la función principal *main()*. Precisamente, el listado 1.16 muestra el nuevo aspecto de dicha función, la cual introduce las siguientes novedades:

- La acción de juego elegida por el jugador se obtiene a través de una llamada a la nueva función *get_user_action()*, tal y como se muestra en la línea Ⓛ 4.
- Se ha generalizado la impresión de las opciones de juego a través de la cadena *range_str* de la línea Ⓛ 6. Particularmente, en la cadena *range_str* se almacenará el texto [0, 2], ya que la expresión *len(GameAction)* devuelve el número de elementos definidos en el tipo enumerado *GameAction*. Si en un futuro se añaden más miembros a *GameAction*, entonces no será necesario modificar la línea Ⓛ 6. En este sentido, se está planteando una solución escalable a nivel de código.

²⁵<https://docs.python.org/3/library/enum.html>

- La función `get_user_action()`, como se discutirá más adelante, arrojará una excepción del tipo `ValueError` si el jugador introduce una acción o valor numérico que no esté comprendido en el rango $[0, 2]$. Esta excepción se captura en el bloque `try` de la línea $\Theta 3$, y se gestiona en el bloque `except` de la línea $\Theta 5$. Resulta recomendable usar las excepciones que ofrece el propio lenguaje, en lugar de usar excepciones propias, tal y como se discutió en el apartado anterior a través de la excepción `IncorrectOptionException`.
- La acción de juego elegida por la máquina se obtiene a través de la nueva función `get_computer_action()`. Como se puede apreciar, y al igual que ocurre con la función `get_user_action()`, se está aplicando el principio de responsabilidad única, aumentando el nivel de estructuración y mantenibilidad del código.
- El código ofrece, de manera explícita, la posibilidad de jugar, o no, otra ronda al jugador. Esta lógica se materializa en las líneas $\Theta 13-14$, y se delega en la nueva función `play_another_round()`. Note el uso del operador `not`.

Listado 1.16: Función `main()` delegando funcionalidad

```
1 def main():
2     while True:
3         try:
4             user_action = get_user_action()
5         except ValueError as e:
6             range_str = f"[0, {len(GameAction) - 1}]"
7             print(f"Invalid selection. Pick a choice in range {range_str}!")
8             continue
9
10            computer_action = get_computer_action()
11            assess_game(user_action, computer_action)
12
13            if not play_another_round():
14                break
```



Principio de responsabilidad única. Recuerde que, de acuerdo al concepto SRP (Single-Responsibility Principle), todo módulo, clase o función de su código debería tener una responsabilidad sobre una única parte de la funcionalidad de su programa, encapsulándola de manera adecuada.

La función `get_user_action()` se muestra en el listado 1.17. Destaca, particularmente, el uso de la construcción sintáctica denominada *list comprehension* en la línea $\Theta 3$ que Python ofrece. En esencia, esta característica representa una forma elegante de definir y crear listas basadas en listas existentes. La ventaja de esta característica se deriva de la iteración que se realiza sobre los miembros del tipo enumerado `GameAction`, de forma que si, en el futuro, se añaden más miembros (como en la variante del juego *rock paper scissors lizard spock*), simplemente sería necesario modificar la definición de `GameAction`.

La lista `game_choices` de la línea **③** se usa para mostrar por pantalla las opciones de juego disponibles, en la línea **⑥**. Posteriormente, la acción escogida por el jugador se crea como tipo enumerado (línea **⑦**) y se devuelve (línea **⑨**).

Listado 1.17: Función `get_user_action()`

```

1 def get_user_action():
2     # Scalable to more options (beyond rock, paper and scissors...)
3     game_choices = [f"{game_action.name}::{game_action.value}""
4                     for game_action in GameAction]
5     game_choices_str = ", ".join(game_choices)
6     user_selection = int(input("\nPick a choice ({game_choices_str}): "))
7     user_action = GameAction(user_selection)
8
9     return user_action

```

El código relativo a la toma de decisiones de la máquina se ha encapsulado, en esta nueva versión del código, en la función `get_computer_action()`, la cual se muestra en el listado 1.18. Simplemente destaca, con respecto a la versión anterior, el uso del tipo enumerado para gestionar la decisión aleatoria tomada por la máquina.

Listado 1.18: Función `get_computer_action()`

```

1 def get_computer_action():
2     computer_selection = random.randint(0, len(GameAction) - 1)
3     computer_action = GameAction(computer_selection)
4     print(f"Computer picked {computer_action.name}.")
5
6     return computer_action

```

La última función que faltaría por discutir es `play_another_round()`, que se muestra en el listado 1.19. Note cómo tanto la comparación de cadenas como la devolución de comparación se efectúan en una única línea, la línea **③**.

Listado 1.19: Función `play_another_round()`

```

1 def play_another_round():
2     another_round = input("\nAnother round? (y/n): ")
3     return another_round.lower() == 'y'

```

Añadiendo IA básica

El juego *piedra, papel o tijeras* admite la integración de diversas soluciones basadas en IA que maximicen el porcentaje de victorias de un jugador controlado por la máquina. En este apartado se incluye una posible implementación de una IA básica, considerando que existe todo un abanico de soluciones que giran en torno a la **detección de patrones de comportamiento** por parte del jugador humano.

Así, la solución planteada como primera alternativa se apoya sobre dos premisas que se pueden considerar como parte de la *cultural general* del juego²⁶:

1. Es probable que, si el jugador ha ganado la ronda de juego anterior (por ejemplo eligiendo *papel*), entonces repita la última elección que hizo (eligiendo de nuevo *papel*).
2. Es probable que, si el jugador perdió en la última ronda, entonces cambie de elección. Además, esta elección suele ser la que representa el siguiente elemento de la serie. Por ejemplo, si el jugador perdió con piedra, se supone que la siguiente acción que elegirá será papel.

Para contemplar las dos reglas anteriores, que servirán de guía a la IA implementado, es necesario almacenar tanto el resultado de las diferentes rondas de juego como el histórico de decisiones tomadas por el jugador humano como . La implementación que se muestra en el listado 1.20 incluye las listas `game_history` y `user_actions_history`, en las líneas **②-3**, como estructura de datos para dicho almacenamiento. Note cómo esta información se guarda en las líneas **⑦** y **⑧**, respectivamente.

A la hora de evaluar la decisión por parte de la máquina, se vuelve a hacer uso de la función `get_computer_action()`, modificada para recibir como argumentos sendas listas (ver línea **⑭**).

Listado 1.20: Integración de histórico de acciones y resultados

```

1 def main():
2     game_history = []
3     user_actions_history = []
4
5     while True:
6         computer_action = get_computer_action(user_actions_history, game_history)
7
8         try:
9             user_action = get_user_action()
10            user_actions_history.append(user_action)
11        except ValueError as e:
12            range_str = f"[0, {len(GameAction) - 1}]"
13            print(f"Invalid selection. Pick a choice in range {range_str}!")
14            continue
15
16        game_result = assess_game(user_action, computer_action)
17        game_history.append(game_result)
18
19        if not play_another_round():
20            break

```

²⁶Estas suposiciones simplemente responden a observaciones realizadas sobre la forma de jugar de un conjunto arbitrario de individuos, sin que se apoyen en un estudio concreto.



Ilusión de inteligencia. Hace más de medio siglo, en 1950, Alan Turing propuso la denominada *Prueba de Turing*, basada en la incapacidad de una persona de distinguir entre hombre o máquina a la hora de evaluar un programa de ordenador. En concreto, un programa pasaría el test si un evaluador humano no fuera capaz de distinguir si las respuestas a una serie de preguntas formuladas eran o no de una persona.

Por otro lado, en el listado 1.21 se muestran dos estructuras de datos incorporadas a la nueva versión del código:

1. *GameResult*, definida como un tipo enumerado para gestionar valores enteros que permite modelar los posibles resultados tras una ronda de juego: victoria (*victory*), derrota (*defeat*) o empate (*tie*).
2. *Victories*, definida como un diccionario para almacenar qué acción genera una situación de victoria con respecto a la acción efectuada por un contrincante. Por ejemplo, y como se refleja en la línea **⑧**, *paper* derrota a *rock*.

Un **diccionario** se puede entender como un conjunto de pares clave-valor, con el requisito de que las claves son única en el contexto de un diccionario. En Python²⁷, es posible crear un diccionario vacío con un par de llaves {}, o crearlos e inicializarlos con una lista de pares clave-valor separadas por coma, tal y como se puede apreciar en el listado 1.21 (ver líneas **⑦-11**). La indexación de los diccionarios se realiza mediante las claves. En Python, el tipo asociado a la clave debe ser inmutable; por ejemplo, las cadenas de texto o los números pueden actuar como claves.

Listado 1.21: Estructuras de datos para gestionar resultados y decisiones ganadoras

```

1 class GameResult(IntEnum):
2     Victory = 0
3     Defeat = 1
4     Tie = 2
5
6
7 Victories = {
8     GameAction.Rock: GameAction.Paper,
9     GameAction.Paper: GameAction.Scissors,
10    GameAction.Scissors: GameAction.Rock
11 }
```

En este punto, ya es posible discutir la nueva implementación de la función *get_computer_action()*, mostrada en el listado 1.22, la cual incluye las dos premisas introducidas previamente como referencia a la hora de identificar patrones básicos de comportamiento en el jugador humano.

²⁷<https://docs.python.org/3/tutorial/datastructures.html>

En esta implementación, destacan los siguientes puntos:

- Si no existe histórico, entonces la máquina generará una decisión de manera aleatoria (ver líneas ④ 3-4).
- Si el jugador humano ganó la última ronda, entonces la máquina tomará la decisión de elegir la acción que derrota a la última acción elegida por el jugador (ver líneas ④ 12-13). A modo de ejemplo, si el jugador ganó la última ronda eligiendo *papel*, entonces la máquina supondrá que el jugador va a elegir *papel* de nuevo y, por lo tanto, jugará con *tijeras*. Si el jugador ganó con piedra, la máquina elegirá papel en la próxima ronda. Como puede imaginar, la función *get_winner_action()* simplemente indexará el diccionario *Victories* para obtener una posición ganadora con respecto a la decisión que, presumiblemente, tomará el jugador. Note cómo se indexa la lista *user_actions_history* en la línea ④ 13 para obtener el último elemento de una lista en Python.
- Si el jugador humano perdió la última ronda, entonces la máquina tomará la decisión de elegir la acción que derrota al siguiente elemento en la serie *rock-paper-scissors* (ver líneas ④ 15-18). Además de indexar la lista *user_actions_history* para obtener el valor de la última acción elegida por el jugador, dicho valor se actualiza para *apuntar* al siguiente miembro del tipo enumerado.
- Si el jugador empató en la última ronda, entonces la máquina jugará la ronda actual de manera aleatoria.

Listado 1.22: Integración de IA básica

```
1 def get_computer_action(user_actions_history, game_history):  
2     # No previous user actions =>random computer choice  
3     if not user_actions_history or not game_history:  
4         computer_action = get_random_computer_action()  
5     # Basic AI functionality  
6     # 1) If the user won the last round, he may repeat the last choice  
7     # 2) If the user lost the last round,  
8     # he may change to the next action in the sequence  
9     # 3) If user and computer tied in the last round, then get a random computer choice  
10    else:  
11        # Path 1)  
12        if game_history[-1] == GameResult.Victory:  
13            computer_action = get_winner_action(user_actions_history[-1])  
14        # Path 2)  
15        elif game_history[-1] == GameResult.Defeat:  
16            computer_action = get_winner_action(  
17                GameAction((user_actions_history[-1].value + 1) % len(GameAction)))  
18        # Random choice  
19        else:  
20            computer_action = get_random_computer_action()  
21  
22    print(f"Computer picked {computer_action.name}.")  
23  
24    return computer_action
```

La siguiente salida muestra un ejemplo de ejecución de la versión actual del código. Como puede apreciar, el jugador humano pierde en la segunda ronda después de haber elegido *rock*. En la siguiente ronda, la máquina supone que el jugador va a elegir *paper* (el siguiente elemento en la serie).

```
$ python3 04_RPS_Basic_AI.py  
Pick a choice (Rock[0], Paper[1], Scissors[2]): 1  
Computer picked Paper.  
User and computer picked Paper. Draw game!  
  
Another round? (y/n): y  
  
Pick a choice (Rock[0], Paper[1], Scissors[2]): 0  
Computer picked Paper.  
Paper covers rock. You lost!  
  
Another round? (y/n): y  
  
Pick a choice (Rock[0], Paper[1], Scissors[2]): 1  
Computer picked Scissors.  
Scissors cuts paper. You lost!  
  
Another round? (y/n): n
```

IA alternativa

El código discutido hasta el momento se puede modificar fácilmente para ajustar la IA de la máquina y mejorar la tasa de victorias con respecto a un jugador humano. De hecho, existen competiciones basadas en implementar bots de IA que compiten entre sí para averiguar quién se comporta mejor. El denominador común de todos ellos suele basarse en la detección de patrones, problemática que se puede abordar desde diferentes puntos de vista.

A modo de ejemplo, en este último apartado se incluye una nueva modificación de la función *get_computer_action()*, que gira en torno a la **suposición de que hay jugadores que optan por repetir una acción más que otra**. En este sentido, y como se muestra en el listado 1.23, la máquina tomaría la decisión de elegir la acción que derrota a la acción más frecuentemente usada por el jugador en el pasado inmediato.

La implementación planteada pasa por obtener la sublista que contiene las NUMBER_RECENT_ACTIONS más recientes, constante que se establece con un valor de 5 en la línea **③** del listado 1.23. Esta sublista se usa como argumento de la función *mode*, contenida en el paquete *statistics* de Python, para obtener el elemento más repetido, tal y como se refleja en las líneas **⑬-14**.

Finalmente, resulta relevante destacar la facilidad que Python ofrece para prototipar y modificar código de manera ágil, como el lector ha podido comprobar a lo largo de esta sección, en la que se ha planteado una solución incremental al clásico juego *piedra, papel o tijeras*.

Listado 1.23: Modificación de IA básica

```
1 from statistics import mode
2
3 NUMBER_RECENT_ACTIONS = 5
4
5
6 def get_computer_action(user_actions_history, game_history):
7     # No previous user actions => random computer choice
8     if not user_actions_history or not game_history:
9         computer_action = get_random_computer_action()
10    # Alternative AI functionality
11    # Choice that would beat the user's most frequent recent choice
12 else:
13     most_frequent_recent_computer_action = \
14         GameAction(mode(user_actions_history[-NUMBER_RECENT_ACTIONS:]))
15     computer_action = get_winner_action(most_frequent_recent_computer_action)
16
17 print(f"Computer picked {computer_action.name}.")
18
19 return computer_action
```



Extendiendo Rock Paper Scissors. Se deja como ejercicio propuesto para el lector la modificación del código final de esta sección para incluir la extensión *lizard spock* del juego original, diseñada por Sam Kass y Karen Bryla (<http://www.samkass.com/theories/RPSSL.html>) antes de que se hiciera popular en la conocida serie de televisión *The Big Bang Theory*. También se plantea, como posible mejora, la integración de soluciones artificiales basadas en el aprendizaje automático derivado de conjuntos de entrenamiento que alberguen partidas del juego.

1.3.5. Resumen

En esta sección se ha profundizado con respecto a dos lenguajes de programación, **R** y **Python**, que actualmente gozan de un protagonismo relevante en el ámbito de la IA. Particularmente, Python se ha tomado como referencia principal a la hora de elegir un lenguaje de programación para IA, y se han discutido sus características más destacables.

Por otro lado, se han descrito **dos casos prácticos** que sirven para ejemplificar la potencia de estos dos lenguajes. Resulta especialmente interesante que el lector estudio el caso práctico con el lenguaje Python, en el que se ha abordado una implementación incremental del clásico juego *piedra, papel o tijeras*. Esta solución ha servido para reflejar, de una manera pragmática, las bondades de Python como lenguaje de programación para IA. Se anima al lector a modificar dicho código fuente y a integrar nuevas variantes del comportamiento de la IA incluida en la versión actual.

1.4. Procesamiento de datos, objetos y lenguajes de marcas

Los lenguajes de marcas nos proporcionan una forma estructurada de organizar la información para su posterior procesamiento en algún lenguaje de programación, *framework* o herramienta. En el ámbito de la IA, los lenguajes de marcas nos permiten, por ejemplo, especificar los parámetros de configuración de una red neuronal artificial para un determinado problema o incluso crear aplicaciones complejas a través de una descripción declarativa de la funcionalidad deseada.

En esta sección se presentan los lenguajes de marcas más utilizados en la actualidad, una clasificación de los mismos en función de su objetivo, y cómo aplicar transformaciones a los datos que contienen.

1.4.1. Los lenguajes de marcas

Los lenguajes de marcas o marcado nacen de la necesidad de aportar o anotar información sobre el contenido de un documento. Por ejemplo, si se tiene una lista de productos y se quiere saber cuáles son los precios y *stock*, es necesario indicar de alguna forma qué parte de cada elemento de la lista se corresponde con el precio y qué parte con el *stock*.

Para conseguirlo, se pueden emplear diversas formas de codificación de dicho documento. Una de ellas es utilizando etiquetas o marcas que añadan esa información extra que necesitamos. El lenguaje definido por el conjunto de etiquetas es lo que se denomina lenguaje de marcas. A su vez, será necesario establecer una serie de reglas que le den validez al lenguaje de marcas que vayamos a utilizar. Estas reglas vendrán introducidas por la sintaxis del lenguaje y su semántica.

A diferencia de los lenguajes de programación, los lenguajes de marcas únicamente se utilizan para estructurar y representar información. Es decir, no son lenguajes que nos permitan construir aplicaciones que realicen alguna funcionalidad lógica. Sin embargo, la línea divisoria entre lo que es un lenguaje de programación y lo que es un lenguaje de marcado puede resultar difusa, por lo que en caso de duda, podemos referirnos a ambos tipos como *lenguajes informáticos*.

Los lenguajes de marcas más extendidos son HTML y XML, aunque existen multitud de derivados del segundo que lo toman como base para construir otros lenguajes de marcas. Es por ello que XML suele definirse como un meta-lenguaje de marcas.

Clasificación de los lenguajes de marcas

Existen tres categorías principales de lenguajes de marcado, propuestas en 1987 [CRD87] y 2003 [Bra]:

- **Presentacionales:** se trata de aquellos lenguajes de marcas utilizados en procesadores de texto tradicionales, es decir, aquellos que utilizan códigos binarios empotrados en los documentos junto al texto para producir una salida modificada. Estas marcas normalmente se encuentran ocultas al usuario que está trabajando en el documento, abstrayéndole del procesamiento interno que realiza el software para construir el documento. El aspecto presentacional se consigue mediante la conversión de esos códigos binarios internos a aspectos del documento como, por ejemplo, palabras en negrita, incrustado de imágenes en línea y construcción de tablas de contenido, entre otros. Un ejemplo de software que utiliza este tipo de lenguajes de marcas podría ser Microsoft Word.
- **Procedurales:** las marcas son incrustadas en línea con el texto del documento para indicar las instrucciones que otros programas tendrán que realizar cuando procesen el texto. De esta forma, se espera que el procesador lea el archivo completo de principio a fin, siguiendo las instrucciones que encuentre para manipular el texto. Los archivos de texto que utilizan este tipo de marcas suelen editarse con las marcas visibles para que puedan ser manipuladas a discreción del usuario. Un ejemplo sería el lenguaje PostScript, utilizado como estándar en los sistemas de impresión para imprimir documentos complejos.
- **Descriptivos:** también conocidos como *semánticos*, este tipo de lenguajes de marcas se utiliza para realizar anotaciones que describan partes del documento, en lugar de indicar cómo deberían procesarse. Así, el objetivo es el de desacoplar la estructura del documento de cualquier tratamiento o interpretación particular del mismo. Como buena práctica, se recomienda que este tipo de lenguajes de marcas se utilice para describir el texto conceptualmente (p. ej. indicando que el texto es una citación bibliográfica), en lugar de visualmente (p. ej. indicando que el texto debería mostrarse en negrita). Un ejemplo de este tipo lenguajes sería HTML, utilizado para crear páginas web.

En los últimos años se han desarrollado una serie de pequeños lenguajes de marcado, generalmente no estandarizados, que permiten a los usuarios crear texto con formato a través de los navegadores web, como los utilizados en las *wikis* y en los foros web y comúnmente denominados *lenguajes de marcas ligeros*. Por ejemplo, Markdown y BBCODE pertenecerían a esta familia de lenguajes.

Editores de texto

Atendiendo a la clasificación anterior, podemos dar algunas indicaciones sobre los editores de texto a utilizar para crear archivos que utilicen lenguajes de marcas.

Los lenguajes de marcas presentacionales necesitarán algún software que permita su construcción. Al ser archivos típicamente binarios, cada uno de estos archivos necesitará un software concreto que sea capaz de procesarlo, aunque en la práctica, la mayoría de estos formatos pueden ser procesados por varios sistemas,

ya sea porque se ha liberado su especificación o porque se ha realizado un esfuerzo de ingeniería inversa para *concretar* dicha especificación. Un ejemplo de esto puede ser el formato de archivo ‘DOC’, utilizado por versiones previas a Microsoft Word 2007; aunque el formato era propietario de Microsoft (la especificación se liberó posteriormente en 2008), era posible procesar dichos documentos con otros softwares de terceros como, por ejemplo, LibreOffice Writer.

En el caso de los lenguajes de marcas procedurales o descriptivos, existe flexibilidad completa en los editores de texto que podemos utilizar para crearlos o editarlos, siempre que utilicemos editores de texto plano. Es decir, cualquiera de estos archivos que utilicen estos tipos de lenguajes de marcas podrá ser editado, por ejemplo, con el «Bloc de Notas» que incluye Microsoft Windows, «TextEdit» en el caso de macOS, o «Gedit» en el caso de Ubuntu (GNU/Linux). Este tipo de editores suele carecer de algunas funciones útiles típicas de editores de texto más avanzados como, por ejemplo, resaltado de sintaxis y apertura/cierre automática de las marcas, entre otras. Así, podemos utilizar otros editores de texto para manipular estos archivos, como «Notepad++» en el caso de Microsoft Windows o «CotEditor» en el caso de macOS; el editor de texto mencionado para Ubuntu ya es lo suficientemente avanzado como para sustituirlo por otro similar. En cualquier caso, podemos optar por una solución multiplataforma y extensible como «Visual Studio Code», que incluye además la posibilidad de añadir otros plugins que faciliten el trabajo con tipos de archivos específicos de lenguajes de marcas.

1.4.2. Presentación de la información mediante HTML

HTML es un lenguaje de marcas orientado a la creación de páginas web. Por ello, su función fundamental es la de crear la estructura de una página, dar formato a los elementos que la componen y añadir información semántica.

La extensión común de estos archivos es *.html* y suelen ser interpretados por los navegadores web o cualquier aplicación que disponga de un navegador web integrado (p. ej. visores de HTML).

Inicialmente surgió con fines divulgativos y el estándar carecía de especificaciones para integrar contenidos multimedia. Es por ello que cada navegador web integraba sus propias extensiones del lenguaje HTML para añadir funcionalidad adicional como, por ejemplo, los antiguos controles ActiveX utilizados en el navegador Internet Explorer.



La versión actual utilizada de HTML se denomina HTML5, la cual ha permitido simplificar el desarrollo web estableciendo un estándar común al que el resto de navegadores web tienen que adaptarse.

Estructura de HTML

En el lenguaje de marcas HTML, las marcas son etiquetas que se colocan en línea con el texto, añadiéndole significado. Habitualmente, estas etiquetas consisten en un nombre encerrado entre signos de menor qué (<) y mayor qué (>). Además, suelen aparecer por parejas, de manera que la primera abre un contexto y la segunda lo cierra, afectando a todo lo que queda dentro de dicho contexto.

El nombre que se le haya dado a la etiqueta será el que le dé un significado u otro. Por ejemplo, <p> indicará que estamos escribiendo un párrafo y indicará que queremos realzar la importancia de un texto. Además, se permite el anidamiento de etiquetas, es decir, pueden aparecer unas dentro del contexto de otras, por ejemplo: <p>Esto es importante. Esto ya no lo es.</p>. Así, un documento HTML estará compuesto por un conjunto de etiquetas, que se utilizarán para definir la forma que se quiere aplicar a cada parte del documento. Será el navegador el que se encargue de interpretar todas estas etiquetas y mostrar el contenido con el formato definido por ellas.

Concretamente, todo documento HTML tendrá que estar delimitado por las etiquetas <html> y </html>. Dentro de estas etiquetas, se pueden distinguir dos partes principales:

- **Cabecera:** delimitada por <head> y </head>, donde se colocarán meta-etiquetas y otras etiquetas informativas como, por ejemplo, el título de nuestra página.
- **Cuerpo:** delimitador por <body> y </body>, y que contendrá el contenido de la página.

Además de lo anterior, es necesario indicar al navegador que interpretará el documento HTML la versión de HTML que se está utilizando en el documento. Cada versión de HTML tiene soporte para unas u otras etiquetas. En cualquier caso, podemos indicar que nuestro documento utiliza la versión actual (HTML5) utilizando la etiqueta <!DOCTYPE html> en el preámbulo de nuestro documento.

Utilizando estas dos partes y la versión de HTML que utilizará nuestro documento, podemos definir la estructura típica de un documento HTML de la siguiente forma:

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4      <title>Estructura de un documento HTML</title>
5      <!-- Etiquetas y contenidos de la cabecera -->
6  </head>
7  <body>
8      <!-- Etiquetas y contenidos del cuerpo -->
9  </body>
10 </html>
```

En este código, podemos ver cómo en las líneas **⑤** y **⑧** se incluyen dos comentarios en el documento. Estos comentarios no serán mostrados cuando se interprete el documento por el navegador (es decir, serán ignorados) pero nos servirán internamente para añadir algún tipo de anotación sobre el archivo. Los comentarios en HTML se definen entre las etiquetas `<!--` y `-->`, y pueden abarcar múltiples líneas.

Elementos, etiquetas y atributos

Un documento HTML está formado por un conjunto de **elementos**, cada uno de ellos con un nombre (etiqueta), un contenido y unas características (atributos). Al ser un lenguaje de marcas, son las etiquetas las que se usan para delimitar el contenido de estos elementos.

Estas **etiquetas** pueden ser de tres tipos:

- **De apertura:** indicadas mediante `<nOMBRE_etiqueta>`.
- **De cierre:** indicadas mediante `</nOMBRE_etiqueta>`.
- **Autocerradas:** indicadas mediante `<nOMBRE_etiqueta />`. Sirven para indicar elementos sin contenido (p. ej. saltos de línea).



En la versión actual de HTML las etiquetas autocerradas se encuentran en desuso. Por ejemplo, los saltos de línea se pueden escribir como `
` o `
`, aunque se recomienda utilizar la primera forma.

Los nombres de las etiquetas no diferencian entre mayúsculas y minúsculas, por lo que pueden escribirse con cualquier capitalización. Sin embargo, se recomienda que se escriban siempre en minúsculas.

Además, cada uno de los elementos puede tener una serie de propiedades que se denominan **atributos**, y se indican en la etiqueta de apertura de dicho elemento, asignándoles un valor entre comillas (simples o dobles), de la forma: `atributo="valor"`. Todos los atributos deben escribirse separados tanto de la etiqueta como de otros atributos.



Es posible utilizar atributos vacíos sin indicar un valor, lo que se traducirá por defecto en un valor de cadena vacía.

Los atributos posibles y el valor que se le puede dar a cada uno de ellos vendrá determinado por la especificación de la versión de HTML que se esté usando, siendo posibles valores de tipo textual o numérico. Éstos últimos, pueden aparecer en diferentes bases (decimal, hexadecimal...) y con diferentes unidades (sin unidades, en píxeles, en porcentajes, etc.).

Así, la sintaxis de un elemento es la siguiente:

```
1 <nombre_etiqueta atributo1="valor1" atributo2="valor2" ...>
2   contenido_elemento
3 </nombre_etiqueta>
```

Por ejemplo, como se vio anteriormente, el elemento principal en un documento HTML es el delimitado por `<html>` y `</html>`. Su contenido es el propio contenido del documento, compuesto a su vez por los elementos `<head>` y `<body>`.

El principal atributo de `<html>` es `lang`, que identifica el idioma en el que está escrito nuestro documento. Por tanto, para un documento en español, se escribiría `<html lang="es">`, mientras que para un documento en inglés `<html lang="en">`.

Elementos de la cabecera

La primera parte de un documento HTML es su cabecera, delimitada por el elemento `<head></head>`. Algunas de las etiquetas que podemos encontrarnos como contenido de este elemento son:

- **<title>**: sirve para indicar el título del documento (contenido dentro del elemento entre las etiquetas) que mostrará el navegador. Resulta obligatorio indicarlo para que el documento HTML sea válido.
- **<meta>**: sirve para declarar meta-etiquetas empleadas para definir propiedades del documento. Dichas propiedades se especifican habitualmente mediante un atributo `name` para indicar la propiedad que queremos indicar y otro atributo `content` para indicar su valor. Estas etiquetas no requieren de otra etiqueta de cierre. Por ejemplo, mediante el elemento `<meta charset=UTF-8>` podemos indicar la codificación de caracteres del documento, lo cual es recomendable si queremos mostrar caracteres con tildes o eñes.
- **<link>**: se emplea para enlazar con archivos externos como, por ejemplo, hojas de estilos. No lleva etiqueta de cierre.
- **<script>**: se utiliza para incluir código que ejecutará el propio navegador, normalmente en lenguaje JavaScript. También puede usarse para incluir archivos externos. En este último caso, no lleva etiqueta de cierre.

Un ejemplo más completo de documento HTML considerando lo anterior podría ser el siguiente:

```
1 <!DOCTYPE html>
2 <html lang="es">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <meta name="description" content="Estructura de un documento HTML">
7   <meta name="author" content="Santiago">
8   <title>Documento HTML</title>
```

```
9  </head>
10 <body>
11     <!-- Cuerpo del documento HTML; lo que mostrará el navegador en pantalla -->
12 </body>
13 </html>
```



La línea **⑤** nos permitirá hacer que nuestro documento HTML se ajuste automáticamente al ancho de la pantalla del dispositivo con el que lo estemos visualizando, resultando así en una página adaptable o *responsive*.

Elementos del cuerpo

La otra parte que podemos encontrar en un documento HTML es el cuerpo, contenida en el elemento `<body></body>`. En esta parte del documento se incluirán todos los elementos que el navegador mostrará por pantalla como, por ejemplo, texto, imágenes y tablas, entre otros.

El aspecto con el que se mostrarán los elementos del cuerpo vendrá definido por el navegador web que se utilice para visualizar el documento, a no ser que se especifiquen manualmente los estilos de los elementos. Estos estilos se definen en cualquier etiqueta mediante el atributo `style`. Cada propiedad de estilo se escribe de la forma `propiedad: valor`, y se pueden especificar tantos estilos como se desee, separándolos con puntos y comas:

```
1 <nOMBRE_etiqueta style="propiedad1: valor1; propiedad2: valor2 ...">
2   contenido_elemento
3 </nOMBRE_etiqueta>
```



Resulta obligatorio aclarar aquí que esta forma de aplicar estilos al texto (conocida como estilos en línea) no es la recomendable, sino que se suelen utilizar hojas de estilos en cascada (CSS) enlazados con la etiqueta `link` en la cabecera para ello. Por simplicidad, el resto de la documentación seguirá utilizando estilos en línea para ilustrar los ejemplos. Si el lector desea saber más sobre la aplicación de estilos CSS, se recomienda visitar el recurso de aprendizaje disponible en la MDN Web Docs:  Enlace: developer.mozilla.org/en-US/docs/Learn/CSS .

A continuación se introducen algunas de las etiquetas más relevantes que podemos utilizar en el cuerpo de un documento HTML.

Párrafos

Podemos crear párrafos en nuestro documento utilizando el elemento `<p></p>`. Por defecto, el texto representado mediante párrafos añadirá un pequeño margen inferior para separar los párrafos entre sí.

Los párrafos pueden alinearse, centrarse o justificarse utilizando estilos, por ejemplo: `<p style="text-align: left">Texto del párrafo</p>`. En el Listado 1.24 se muestra un ejemplo completo con posibles configuraciones de párrafos, mientras que en la Figura 1.17 puede verse el resultado de abrir dicho documento en un navegador web.

Listado 1.24: Ejemplo de creación de párrafos en HTML y posibles alineaciones.

```
1 <!DOCTYPE html>
2 <html lang="es">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <title>Ejemplo de párrafos</title>
7 </head>
8 <body>
9   <p style="text-align: left">Texto alineado a la izquierda</p>
10  <p style="text-align: center">Texto centrado</p>
11  <p style="text-align: right">Texto alineado a la derecha</p>
12  <p style="text-align: justify">Texto justificado a ambos márgenes del documento. Para
      ver el efecto, es necesario incluir varias líneas de texto. Puede verse cómo
      el navegador ampliará el ancho de los espacios en blanco para ajustar las
      líneas de texto y que queden justificadas.</p>
13 </body>
14 </html>
```

Encabezados

Permiten definir títulos en las páginas utilizando el elemento `<hx></hx>`, siendo `x` cualquier número comprendido entre 1 y 6. Utilizando estos elementos podemos indicar encabezados con mayor o menor importancia y tamaño; el encabezado de primer nivel, denotado por la etiqueta `<h1>` tendrá mayor importancia que el `<h2>`, y así sucesivamente con los siguientes. La etiqueta `<h3>` se correspondería con el tamaño estándar del texto.

Idealmente, se suele utilizar el encabezado de primer y segundo nivel para el título principal de la página, y el resto de tamaños menores para otros títulos de segunda categoría.

En el Listado 1.25 y la Figura 1.18 se muestra un ejemplo completo.

Listado 1.25: Ejemplo de creación de encabezados en HTML.

```
1 <!DOCTYPE html>
2 <html lang="es">
3 <head>
```

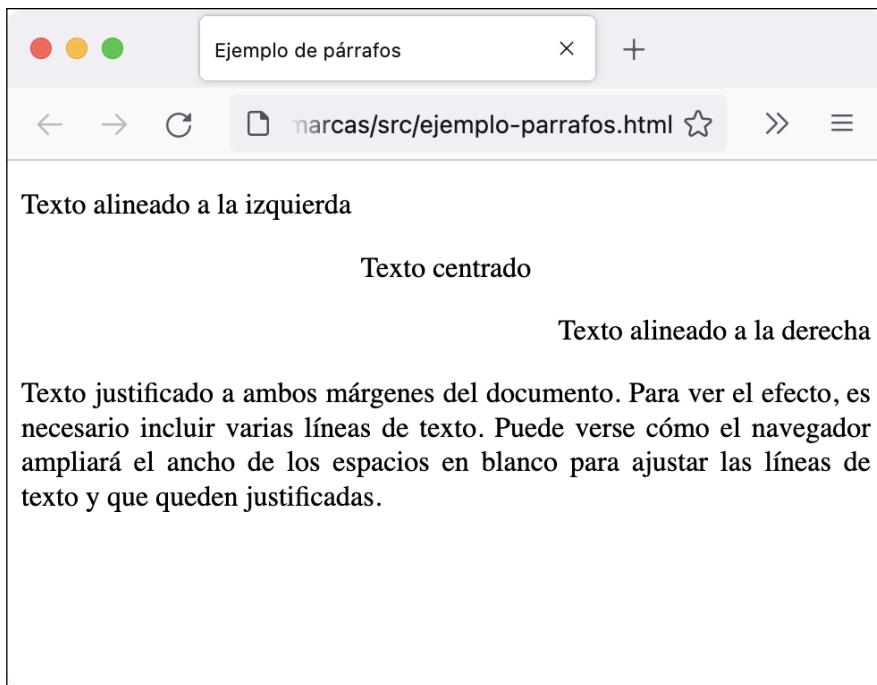


Figura 1.17: Resultado de abrir el documento del Listado 1.24 en un navegador web.

```
4 <meta charset="UTF-8">
5 <meta name="viewport" content="width=device-width, initial-scale=1.0">
6 <title>Ejemplo de encabezados</title>
7 </head>
8 <body>
9   <h1>Título 1</h1>
10  <h2>Título 2</h2>
11  <h3>Título inferior 3</h3>
12  <h4>Título inferior 4</h4>
13  <h5>Título inferior 5</h5>
14  <h6>Título inferior 6</h6>
15 </body>
16 </html>
```

Líneas horizontales

Son líneas que se usan para separar secciones de texto. Se usa la etiqueta `<hr>`, sin etiqueta de cierre. Por defecto, la línea ocupará todo el ancho de la página o el bloque en el que se encuentre, a no ser que se especifique lo contrario en su estilo (ver Listado 1.26 y Figura 1.19).

Listado 1.26: Ejemplo de creación de líneas horizontales en HTML.

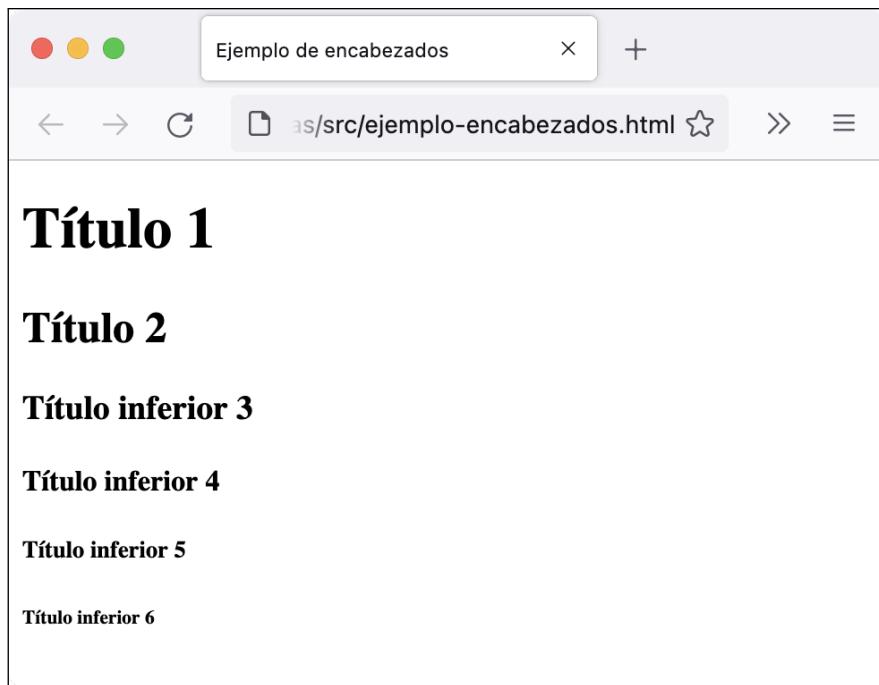


Figura 1.18: Resultado de abrir el documento del Listado 1.25 en un navegador web.

```
1 <!DOCTYPE html>
2 <html lang="es">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <title>Ejemplo de líneas horizontales</title>
7 </head>
8 <body>
9   <p>Primer párrafo de texto</p>
10  <hr>
11  <p>Segundo párrafo de texto</p>
12  <hr style="width: 75%">
13  <p>Tercer párrafo de texto</p>
14  <hr style="width: 50%">
15  <p>Cuarto párrafo de texto</p>
16  <hr style="width: 200px">
17  <p>Quinto párrafo de texto</p>
18 </body>
19 </html>
```



Resulta interesante destacar como es posible definir la longitud de las líneas utilizando distintas unidades como, por ejemplo, porcentajes o píxeles.

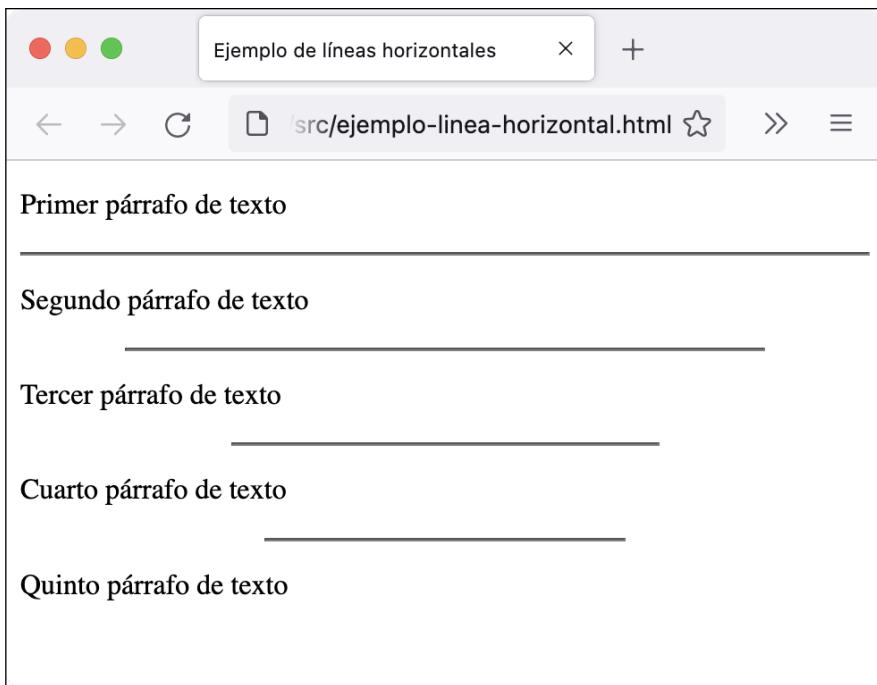


Figura 1.19: Resultado de abrir el documento del Listado 1.26 en un navegador web.

Efectos sobre el texto

Es posible dotar al texto de un documento HTML de algún significado semántico mediante un conjunto específico de etiquetas. A su vez, los navegadores interpretan estas etiquetas de una manera especial, representando el texto afectado utilizando diferentes estilos y sirviendo a los lectores de pantalla de ayuda para utilizar un tono adecuado a la hora de leerlas en voz alta. Estas etiquetas son:

- ****: sirve para indicar que el texto es importante; los navegadores suelen representar este tipo de textos en negrita.
- ****: sirve para aplicar cierto énfasis al texto; los navegadores suelen representar estos textos en cursiva.
- ****: sirve para indicar que el texto se ha eliminado; los navegadores suelen representar estos textos como tachados.

Para el resto de estilos que queramos aplicar sobre el texto, lo haremos mediante el atributo `style` como hemos visto anteriormente. Combinando dicho atributo con el elemento ``, es posible aplicar estilos a palabras o líneas completas contenidas en algún otro elemento como, por ejemplo, párrafos, de tal forma que podamos realizar un ajuste fino de cada texto del documento.

En el Listado 1.27 y la Figura 1.20 se muestra un ejemplo completo.

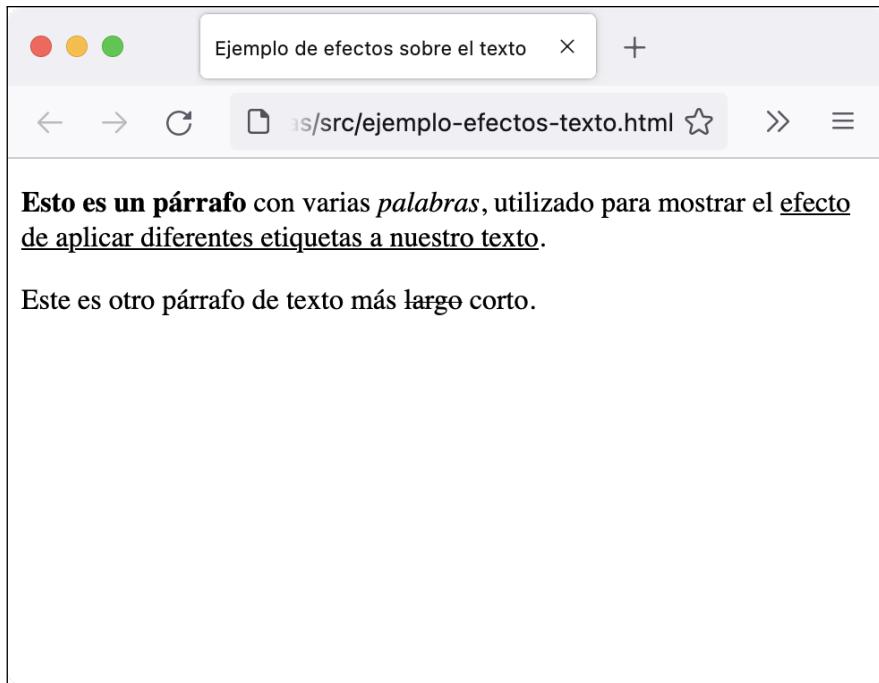


Figura 1.20: Resultado de abrir el documento del Listado 1.27 en un navegador web.

Listado 1.27: Ejemplo de creación de diferentes efectos de texto en HTML.

```
1 <!DOCTYPE html>
2 <html lang="es">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <title>Ejemplo de efectos sobre el texto</title>
7 </head>
8 <body>
9   <p><strong>Esto es un párrafo</strong> con varias <em>palabras</em>, utilizado para
     mostrar el <span style="text-decoration: underline">efecto de aplicar
     diferentes etiquetas a nuestro texto</span>. </p>
10  <p>Este es otro párrafo de texto más <del>largo</del> corto.</p>
11 </body>
12 </html>
```

Texto preformatado

HTML no conserva los espacios a partir del primero, ni las tabulaciones, ni los saltos de línea para que el usuario pueda organizar el código HTML como desee sin que afecte al resultado final. Es por ello que para insertar más espacios se puede utilizar la entidad `&nbsp` y para insertar saltos de línea la etiqueta `
`.

Sin embargo, puede haber ciertas situaciones que queramos escribir un texto tal cual se quiere que aparezca, por ejemplo, para mostrar el fragmento de un *dataset* en formato CSV o el código fuente de algún programa con el que estemos trabajando. Para ello, existe el elemento `<pre></pre>` que define el elemento de texto preformatado, de forma que todo su contenido será mostrado igual que se escribió (ver Listado 1.28 y Figura 1.21).

Listado 1.28: Ejemplo de creación de texto preformatado en HTML mostrando las 25 primeras filas de un dataset del medallero olímpico de Tokio 2021.

```
1 <!DOCTYPE html>
2 <html lang="es">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <title>Ejemplo de texto preformatado</title>
7 </head>
8 <body>
9   <p>Esto es un texto con muchos &nbsp;&nbsp;&nbsp;&nbsp;&nbsp; espacios
     utilizando la entidad &nbsp; A continuación se insertan varios saltos de
     línea dentro del mismo párrafo usando la etiqueta &lt;br&gt;;:
10  <br>
11  <br>
12  <br>
13  <br>
14  Fin de los saltos de línea. Ejemplo de contenido usando las etiquetas &lt;pre&gt;:</p>
15 <pre>
16   Rank,Team/NOC,Gold,Silver,Bronze,Total,Rank by Total
17   1,United States of America,39,41,33,113,1
18   2,People's Republic of China,38,32,18,88,2
19   3,Japan,27,14,17,58,5
20   4,Great Britain,22,21,22,65,4
21   5,ROC,20,28,23,71,3
22   6,Australia,17,7,22,46,6
23   7,Netherlands,10,12,14,36,9
24   8,France,10,12,11,33,10
25   9,Germany,10,11,16,37,8
26   10,Italy,10,10,20,40,7
27   11,Canada,7,6,11,24,11
28   12,Brazil,7,6,8,21,12
29   13>New Zealand,7,6,7,20,13
30   14,Cuba,7,3,5,15,18
31   15,Hungary,6,7,7,20,13
32   16,Republic of Korea,6,4,10,20,13
33   17,Poland,4,5,5,14,19
34   18,Czech Republic,4,4,3,11,23
35   19,Kenya,4,4,2,10,25
36   20,Norway,4,2,2,8,29
37   21,Jamaica,4,1,4,9,26
38   22,Spain,3,8,6,17,17
39   23,Sweden,3,6,0,9,26
40   24,Switzerland,3,4,6,13,20
41   25,Denmark,3,4,4,11,23
42 </pre>
43 </body>
```

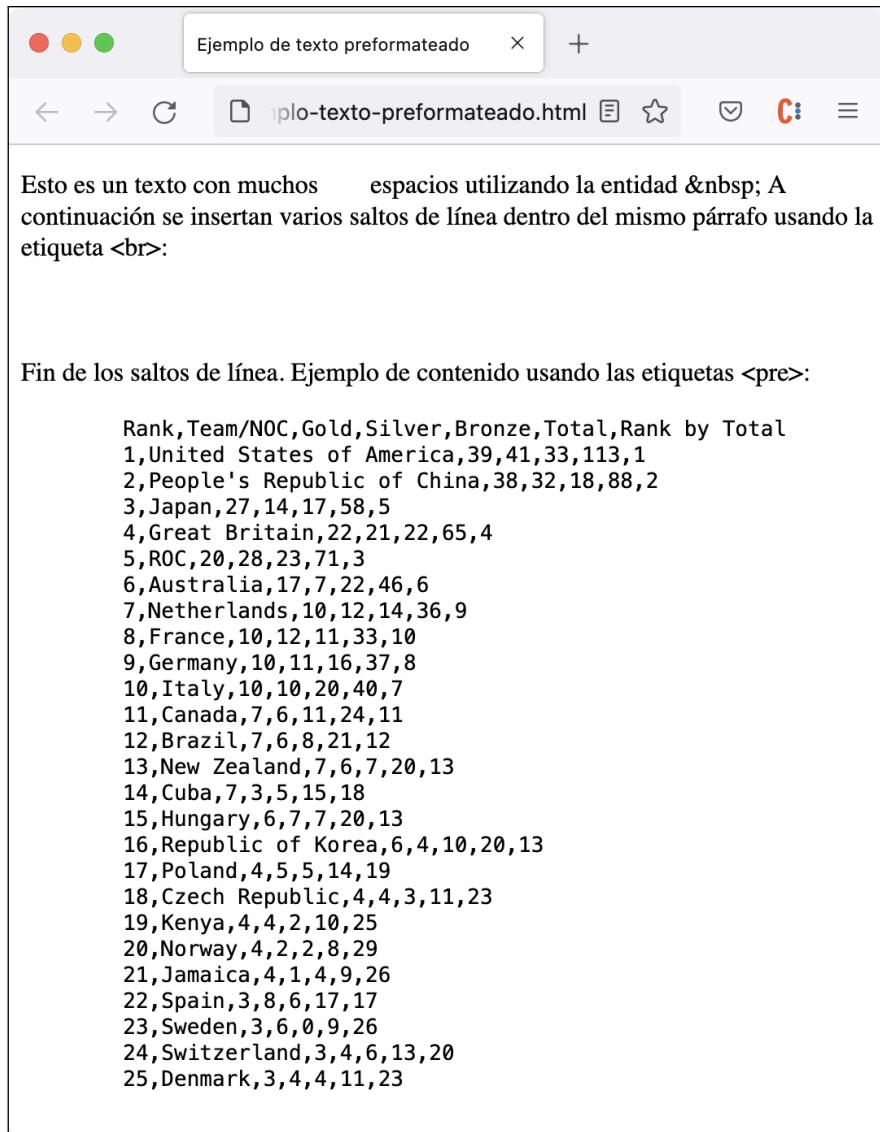


Figura 1.21: Resultado de abrir el documento del Listado 1.28 en un navegador web.

Hipervínculos o enlaces

Los hipervínculos o enlaces permiten conectar documentos HTML con otros, dotándolos de relaciones en lugar de ser documentos aislados. El elemento utilizado para poder crear enlaces es `<a>`. El contenido del elemento será lo que sirva como enlace, que puede ser un texto o cualquier otro elemento que se pueda definir en HTML como, por ejemplo, una imagen. De esta forma, al hacer clic sobre este texto o imagen, el navegador accederá al contenido apuntado por dicho enlace.

Un documento HTML puede enlazarse consigo mismo o con otro, con documentos dentro de la misma web o fuera de ella. En caso de que el documento enlazado sea de un tipo reconocible por el navegador, éste lo abrirá. En caso contrario, preguntará si se desea descargar dicho archivo.

La dirección del documento vendrá especificada en el atributo `href` (hypertext reference) y habitualmente suele indicarse mediante una URL. Otro atributo que resulta interesante mencionar es `target`, que indica al navegador dónde se debe abrir el documento apuntado por el enlace y que puede tomar los valores `_blank` (para abrir el enlace en una pestaña o ventana nueva del navegador) o `_self` (para abrir el enlace en la misma pestaña o ventana del navegador; por defecto).

Dependiendo de la naturaleza del enlace, podemos clasificarlos en:

- **Enlaces absolutos:** se especifican mediante un enlace completo a alguna dirección web concreta.
- **Enlaces relativos:** se especifican mediante un enlace relativo a alguna página alojada en nuestro sitio web.
- **Enlaces internos:** sirven para desplazarse dentro del documento actual a alguna sección concreta. Se necesita definir dos componentes para que funcionen correctamente: i) un elemento HTML ubicado en la sección hacia la que queremos desplazarnos (denominado *ancla*), que vendrá especificada mediante el atributo `id` y un nombre descriptivo como su valor; y ii) el mismo valor que hemos utilizado en el identificador del ancla precedido por el símbolo de almohadilla (#), que será insertado como valor en el atributo `href` del enlace.

En el Listado 1.29 y la Figura 1.22 se muestra un ejemplo completo de todo lo anterior.

Listado 1.29: Ejemplo de enlaces de distintos tipos en HTML.

```
1 <!DOCTYPE html>
2 <html lang="es">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <title>Ejemplo de enlaces</title>
7 </head>
8 <body>
```

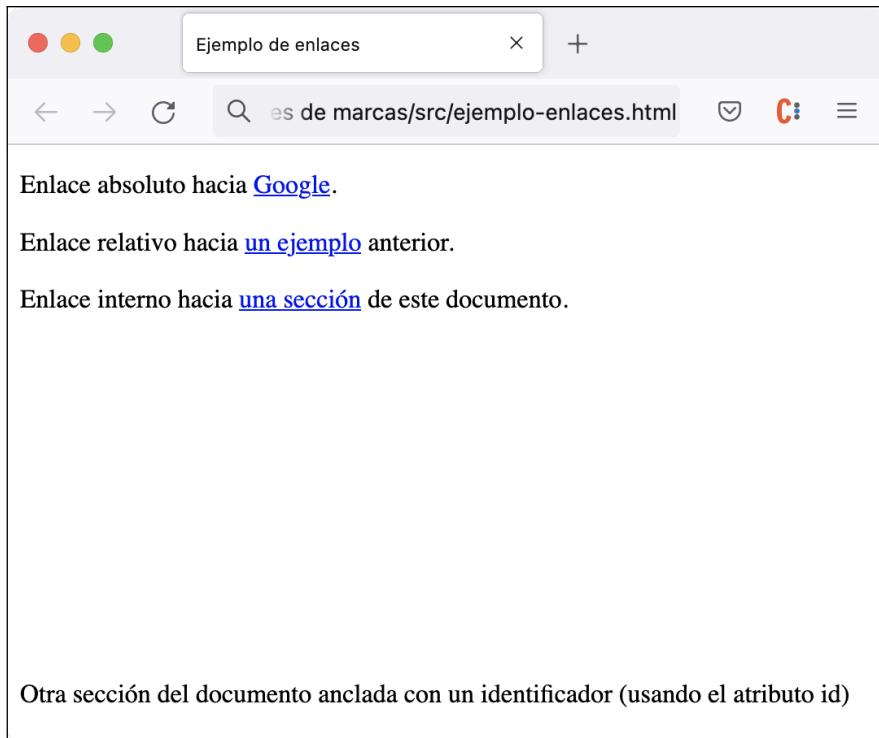


Figura 1.22: Resultado de abrir el documento del Listado 1.29 en un navegador web. Si la barra de desplazamiento (*scroll*) estuviera visible y la sección inferior no se mostrara, pulsar sobre el tercer enlace desplazaría el documento hasta la sección inferior.

```
9      <p>Enlace absoluto hacia <a href="https://www.google.com" target="_blank">Google</a>.  
10     </p>  
11     <p>Enlace relativo hacia <a href="ejemplo-texto-preformatado.html">un ejemplo</a>  
12         anterior.</p>  
13     <p>Enlace interno hacia <a href="#seccion">una sección</a> de este documento.</p>  
14     <br><br><br><br><br><br><br><br>  
15     <p id="seccion">Otra sección del documento anclada con un identificador (usando el  
atributo id)</p>  
16 </body>  
17 </html>
```

Listas

En HTML podemos crear tres tipos de listas diferentes: desordenadas (o de balazos), ordenadas (o numéricas) y de descripción.

Las **listas desordenadas** son aquellas en las que no importa el orden de los elementos, por lo que no llevan numeración. El elemento utilizado para crearlas es `` (abreviatura de *unordered list*) y cada entrada de la lista se definirá mediante el elemento `` (abreviatura de *list item*). Por ejemplo, una lista con 2 elementos se crearía como:

```
1 <ul>
2   <li>Elemento 1</li>
3   <li>Elemento 2</li>
4 </ul>
```



Es posible controlar el estilo de la viñeta utilizando la propiedad de estilo `list-style-type` en la etiqueta `` (aplicaría el estilo a todos los elementos de la lista) o en la etiqueta `` (aplicaría el estilo a un elemento concreto de la lista), y alguno de los valores disponibles: `disc` (por defecto), `circle` (círculo sin relleno), `square` (cuadrado) o `none` (ninguna viñeta).

Las **listas ordenadas** tienen en cuenta el orden de los elementos y lo muestran junto a cada elemento de la lista en lugar de utilizar las viñetas de las listas desordenadas. El elemento utilizado para crearlas es `` (abreviatura de *ordered list*) y se mantiene el uso de los elementos `` para indicar sus entradas. Repitiendo el ejemplo anterior:

```
1 <ol>
2   <li>Elemento 1</li>
3   <li>Elemento 2</li>
4 </ol>
```



La etiqueta `` puede configurarse utilizando diferentes atributos: i) `start`, para indicar cuál será el primer número de la lista (1, 2, 3...); y ii) `type`, para indicar el tipo de numeración a utilizar (1: decimal; a: letras minúsculas; A: letras mayúsculas; i: números romanos en minúsculas; I: números romanos en mayúsculas). Esta configuración puede aplicarse también a nivel de elemento sobre las etiquetas `` en lugar de hacerlo de manera común para todos los elementos.



Es posible anidar listas ordenadas y listas desordenadas para incluir diferentes niveles de listas.

Por último, las **listas de descripción** sirven para enumerar conjuntos de términos con sus respectivas descripciones. En este caso, se utilizará el elemento `<dl></dl>` (abreviatura de *description list*) para definir la lista, `<dt></dt>` para cada término y `<dd></dd>` para cada descripción de cada término. Por ejemplo:

```
1 <dl>
2   <dt>Término 1</dt>
3   <dd>Descripción del término 1</dd>
4   <dt>Término 2</dt>
5   <dd>Descripción del término 2</dd>
6 </dl>
```

En el Listado 1.30 y la Figura 1.23 se muestra un ejemplo completo de todo lo anterior, incluyendo además listas anidadas.

Listado 1.30: Ejemplo de listas de distintos tipos en HTML.

```
1 <!DOCTYPE html>
2 <html lang="es">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <title>Ejemplo de listas</title>
7 </head>
8 <body>
9   <p>Lista de la compra</p>
10  <ul>
11    <li>Nata</li>
12    <li>Jamón
13      <ul>
14        <li>Curado</li>
15        <li>York</li>
16      </ul>
17    </li>
18    <li>Huevos</li>
19  </ul>
20  <p>Receta</p>
21  <ol>
22    <li>Batir los huevos</li>
23    <li>Añadir la nata</li>
24    <li>Picar el jamón</li>
25  </ol>
26  <dl>
27    <dt>Jamón</dt>
28    <dd>Lo necesitamos para picarlo y después servirlo aparte</dd>
29    <dt>Nata</dt>
30    <dd>Se mezclará con los huevos para seguir la receta</dd>
31  </dl>
32 </body>
33 </html>
```

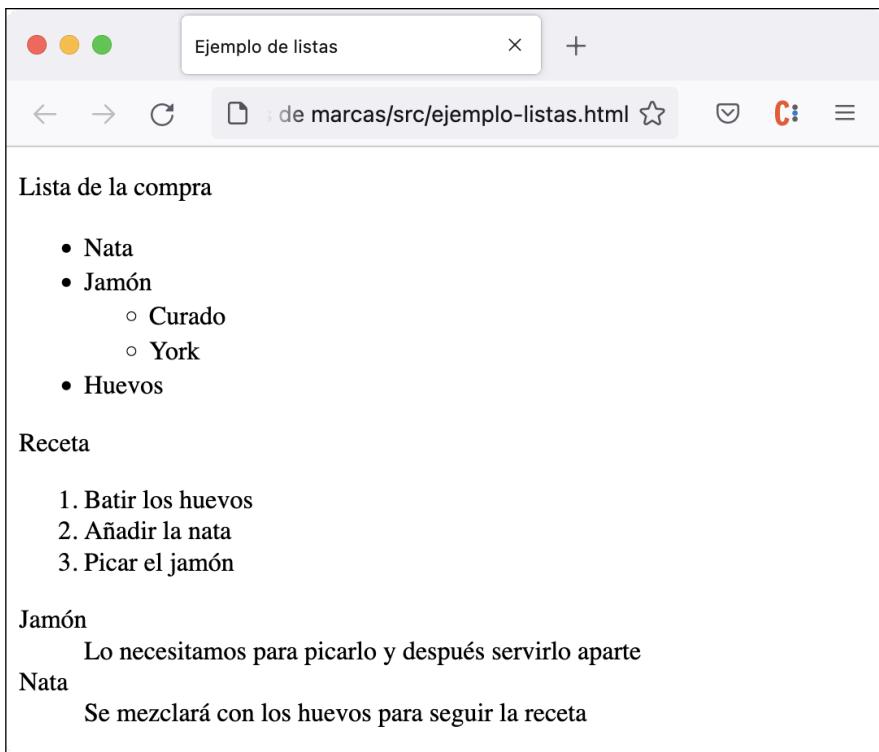


Figura 1.23: Resultado de abrir el documento del Listado 1.30 en un navegador web.

Imágenes

Podemos incrustar imágenes en un documento HTML utilizando la etiqueta `` y los atributos `src` (para indicar la dirección donde se encuentra la imagen, ya sea relativa o absoluta) y `alt` (para indicar una descripción textual de la imagen, utilizada por lectores de pantalla o dispositivos que no puedan cargar imágenes). Por ejemplo:

```
1 
```

Utilizando el atributo `style`, podemos especificar las propiedades `width` y `height` para definir el ancho y el alto de la imagen respectivamente. Del mismo modo, también podemos utilizar otras propiedades como, por ejemplo, `float`, para indicar si queremos que la imagen se encuentre a algún lado del texto utilizando los valores `left` o `right` para ubicar la imagen a la izquierda o a la derecha del texto, respectivamente.

Por último, como se ha mencionado anteriormente, es posible utilizar imágenes como enlaces a otros documentos HTML utilizando la etiqueta `` como contenido del elemento `<a>`. Por ejemplo:

```
1 <a href="https://www.google.com/search?q=gato+negro+tumbado+hierba">
2   
3 </a>
```

En el Listado 1.31 y la Figura 1.24 se muestra un ejemplo completo.

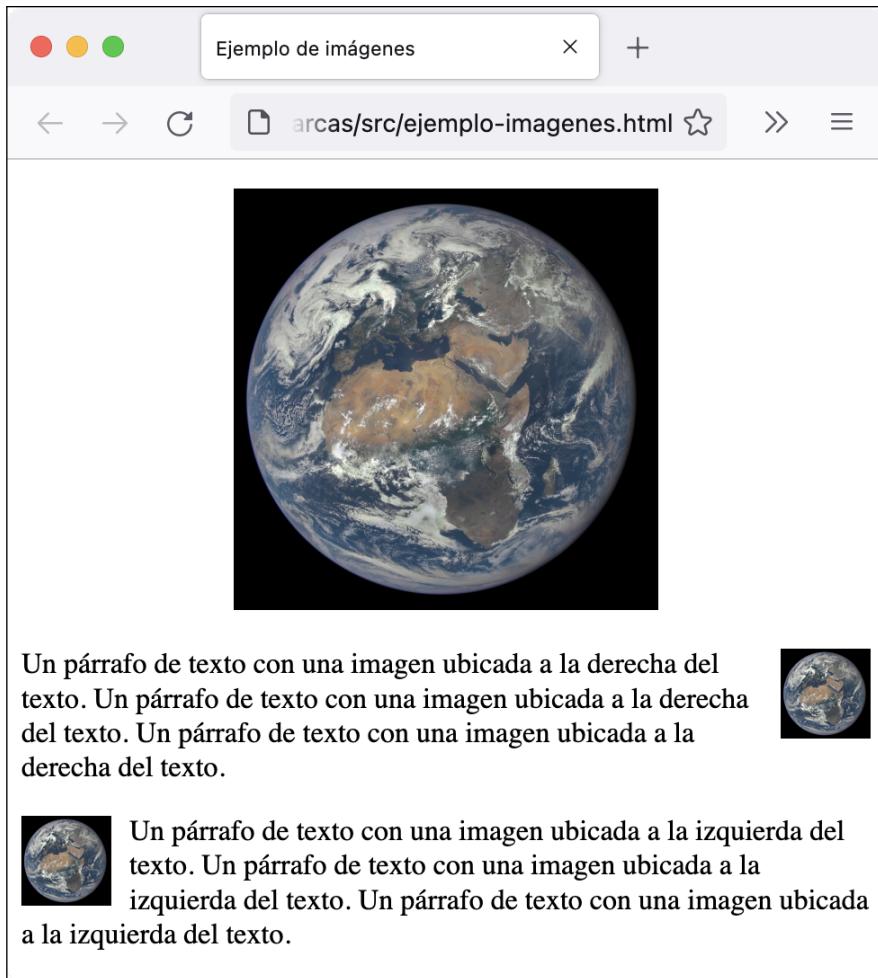
Listado 1.31: Ejemplo de inserción de imágenes en HTML.

```
1 <!DOCTYPE html>
2 <html lang="es">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <title>Ejemplo de imágenes</title>
7 </head>
8 <body>
9   <p style="text-align: center">
10    
13
14   Un párrafo de texto con una imagen ubicada a la derecha del texto. Un párrafo de
15   texto con una imagen ubicada a la derecha del texto. Un párrafo de texto con
16   una imagen ubicada a la derecha del texto.
17
18   <p>
19   
22
23   Un párrafo de texto con una imagen ubicada a la izquierda del texto. Un párrafo de
24   texto con una imagen ubicada a la izquierda del texto. Un párrafo de texto con
25   una imagen ubicada a la izquierda del texto.
26
27   </p>
28 </body>
29 </html>
```

Tablas

Las tablas sirven fundamentalmente para presentar información de manera ordenada. Pueden definirse mediante el elemento `<table></table>` junto a las filas y columnas que deseemos. Esto se realiza mediante el uso de los elementos `<tr></tr>` para definir filas y `<td></td>` para definir celdas dentro de ellas. También podemos especificar un tipo especial de celdas mediante el elemento `<th></th>`, que servirán como títulos de la columnas. Por ejemplo:

```
1 <table>
2   <tr>
3     <th>Título de columna 1</th>
```



Un párrafo de texto con una imagen ubicada a la derecha del texto. Un párrafo de texto con una imagen ubicada a la derecha del texto. Un párrafo de texto con una imagen ubicada a la derecha del texto.



Un párrafo de texto con una imagen ubicada a la izquierda del texto. Un párrafo de texto con una imagen ubicada a la izquierda del texto. Un párrafo de texto con una imagen ubicada a la izquierda del texto.

Figura 1.24: Resultado de abrir el documento del Listado 1.31 en un navegador web.

```
4      <th>Título de columna 2</th>
5  </tr>
6  <tr>
7      <td>Celda 1,1</td>
8      <td>Celda 1,2</td>
9  </tr>
10 <tr>
11     <td>Celda 2,1</td>
12     <td>Celda 2,2</td>
13 </tr>
14 </table>
```

Si quisieramos combinar celdas o hacer que una celda ocupe varias columnas o filas, podemos utilizar los atributos `colspan` y `rowspan` indicando el número de columnas o filas que abarcará la celda, respectivamente. Por ejemplo:

```
1 <table>
2   <tr>
3     <td rowspan="3">Celda abarcando 3 filas</td>
4     <td colspan="2">Celda abarcando 2 columnas</td>
5   </tr>
6   <tr>
7     <td>Celda 1,1</td>
8     <td>Celda 1,2</td>
9   </tr>
10  <tr>
11    <td>Celda 2,1</td>
12    <td>Celda 2,2</td>
13  </tr>
14 </table>
```



Es posible añadir secciones a las tablas mediante los elementos `<thead></thead>`, `<tbody></tbody>` y `<tfoot></tfoot>`, con el objetivo de separar semánticamente la cabecera, el cuerpo y el pie de la tabla, respectivamente.



Por defecto, las tablas aparecerán sin bordes. Podemos añadírselos utilizando el atributo `style` y la propiedad `border`, por ejemplo `style="border: 1px solid black"`. Cabe destacar que esto tendremos que aplicarlo tanto a la etiqueta `<table>` como a las etiquetas `<td>` si queremos que tanto la tabla como las celdas tengan borde.

En el Listado 1.32 y la Figura 1.25 se muestra un ejemplo completo.

Listado 1.32: Ejemplo de creación de tablas en HTML.

```
1 <!DOCTYPE html>
2 <html lang="es">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <title>Ejemplo de tablas</title>
7 </head>
8 <body>
9   <table style="width: 100%; border: 1px solid black">
10    <thead>
11      <tr>
12        <th style="border: 1px solid black">Nombre</th>
13        <th style="border: 1px solid black">Edad</th>
14        <th style="border: 1px solid black">Sexo</th>
15      </tr>
```

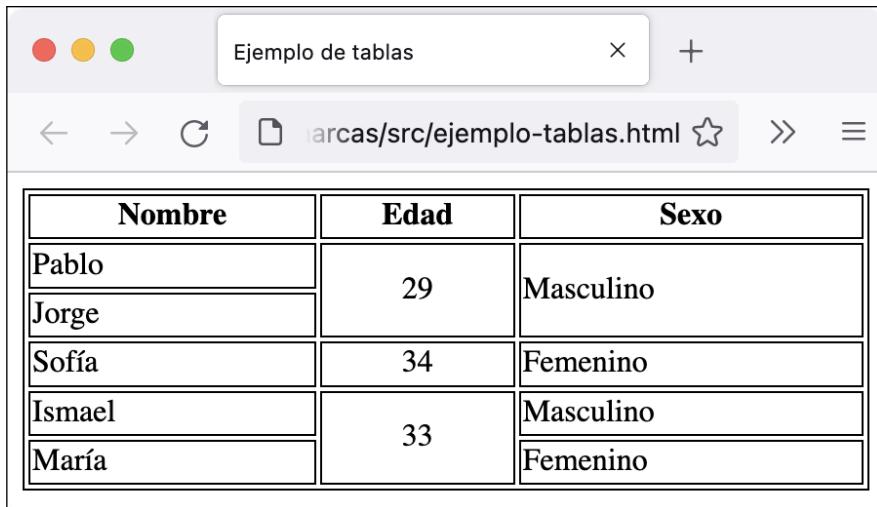


Figura 1.25: Resultado de abrir el documento del Listado 1.32 en un navegador web.

```
16 </thead>
17 <tbody>
18     <tr>
19         <td style="border: 1px solid black">Pablo</td>
20         <td rowspan="2" style="text-align: center; border: 1px solid black">29</
21             td>
22         <td rowspan="2" style="border: 1px solid black">Masculino</td>
23     </tr>
24     <tr>
25         <td style="border: 1px solid black">Jorge</td>
26     </tr>
27     <tr>
28         <td style="border: 1px solid black">Sofía</td>
29         <td style="text-align: center; border: 1px solid black">34</td>
30         <td style="border: 1px solid black">Femenino</td>
31     </tr>
32     <tr>
33         <td style="border: 1px solid black">Ismael</td>
34         <td rowspan="2" style="text-align: center; border: 1px solid black">33</
35             td>
36         <td style="border: 1px solid black">Masculino</td>
37     </tr>
38     <tr>
39         <td style="border: 1px solid black">María</td>
40         <td style="border: 1px solid black">Femenino</td>
41     </tr>
42 </tbody>
43 </table>
```

1.4.3. Estructuración de los datos mediante XML

A diferencia de HTML que nos permite mostrar algún tipo de información en forma de documento web, XML es un lenguaje de marcas que únicamente nos permite almacenar o transportar datos de manera estructurada; su presentación se realizará con alguna otra herramienta.

Concretamente, podemos decir que XML es un meta-lenguaje de marcas, ya que no dispone de etiquetas propias como sí ocurre en HTML (p. ej. las etiquetas `<head>`, `<body>` o `<p>`, por nombrar algunas). XML nos servirá entonces para crear nuestro propio conjunto de marcas que procesaremos con alguna herramienta externa preparada para consumir dicho documento.

Otra diferencia importante con respecto de HTML, es que todas las etiquetas en XML deben cerrarse, ya sea especificando su pareja o mediante etiquetas autocerradas. Así mismo, XML sólo permite un elemento raíz del que colgarán, jerárquicamente, todos los demás. Por ejemplo:

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <carta>
3   <producto>
4     <nombre>Ración de paella</nombre>
5     <precio>9.90</precio>
6   </producto>
7   <producto>
8     <nombre>Bocadillo de lomo</nombre>
9     <precio>3.95</precio>
10  </producto>
11 </carta>
```

Cabeceras XML

Al igual que ocurría en HTML, XML dispone también de una declaración especial que debe aparecer al inicio del documento para especificar algunas propiedades, como la versión del lenguaje que se utilizará (actualmente la versión 1.0) o la codificación de caracteres del documento (por defecto UTF-8), por ejemplo:

```
1 <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
```

El atributo `standalone` es opcional, y se utiliza para indicar si el documento hace referencias a otros documentos externos o no. En este último caso, se asignará el valor `yes`.

Seguidamente a la declaración del archivo XML, se pueden realizar otras como, por ejemplo, la de los ficheros de estilo o transformación que se usarán en la representación del documento:

```
1 <?xml-stylesheet type="text/xsl" href="stylesheet.xsl"?>
```

A lo largo de este capítulo veremos cómo usar dichos ficheros.



La referencia oficial del estándar XML puede encontrarse en [Enlace](http://www.w3.org/TR/xml/):
www.w3.org/TR/xml/.

Elementos de XML

Un documento XML puede contener comentarios, elementos y/o entidades.

Los **comentarios** permiten añadir anotaciones internas al documento XML y se definen del mismo modo que en el lenguaje HTML, utilizando los caracteres `<!--` y `-->` para delimitarlos, por ejemplo:

```
1 <ejemplo>
2   <detalles>Deabajo de este elemento se incluye un comentario</detalles>
3   <!-- Ejemplo de comentario incluido en el documento -->
4 </ejemplo>
```

Los **elementos** definen el contenido principal del documento XML. Se encuentran definidos mediante etiquetas de apertura y de cierre delimitadas mediante los caracteres `<` y `>`. A diferencia de las etiquetas HTML, XML no define unas etiquetas específicas, sino que será el creador del documento quien las defina dependiendo de la información a almacenar. Por ejemplo, podríamos almacenar la información relativa a los colores en formato hexadecimal mediante una etiqueta `<color>`:

```
1 <color>
2   <formato>hexadecimal</formato>
3   <valor>#FF0000</valor>
4 </color>
```

Además de contenido, los elementos pueden definir también sus propios atributos, por ejemplo:

```
1 <color formato="hexadecimal">#FF0000</color>
```

No existe una regla concreta para decidir cuándo usar atributos y cuando usar elementos, aunque por simplicidad podemos utilizar los atributos para especificar meta-information del elemento, y otros elementos para incluir contenido en el documento.

Los elementos también pueden aparecer sin contenido, lo que se denominaría elemento vacío. Esto puede ocurrir cuando ese elemento no dispone de ninguna información. Puede especificarse mediante un elemento autocerrado o mediante un elemento sin contenido, por ejemplo:

```
1 <!-- Elemento vacío sin contenido: -->
```

```
2 <color></color>
3
4 <!-- Elemento vacío autocerrado: -->
5 <color />
```

Por último, las **entidades** nos permiten crear constantes con valores en el preámbulo del documento, para reemplazar las ocurrencias de dichas constantes en el resto del documento con el valor que indicamos y reducir así la introducción repetitiva de la misma información. Por ejemplo, podemos definir una entidad con el nombre `xml` y el valor "Extensible Markup Language", para poder así utilizar la entidad `&xml;` a lo largo del documento y reemplazar sus ocurrencias con el valor textual que hemos indicado.

Estas entidades deben definirse en el interior de una declaración DOCTYPE, similar a la que se utiliza en HTML, pero indicando el nombre de nuestro elemento raíz en lugar del elemento `<html></html>`. Después, se pueden incluir tantas declaraciones ENTITY como entidades queramos definir.



La declaración DOCTYPE forma parte del DTD o *Document Type Definition*, el cual sirve para definir la estructura del documento y poder validarla.

En el Listado 1.33 y la Figura 1.26 se muestra un ejemplo completo de documento XML.

Listado 1.33: Ejemplo de documento XML.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE menu [
3     <!ENTITY fechaActual "2021">
4     <!ENTITY cargo "5">
5 ]>
6 <menu fecha="&fechaActual;">
7     <comida>
8         <nombre>Ración de paella mixta</nombre>
9         <precio>9.00 + &cargo; %</precio>
10    </comida>
11    <comida>
12        <nombre>Ración de paella de pollo</nombre>
13        <precio>9.00 + &cargo; %</precio>
14    </comida>
15    <comida>
16        <nombre>Ración de paella de marisco</nombre>
17        <precio>12.00 + &cargo; %</precio>
18    </comida>
19    <comida>
20        <nombre>Bocadillo de lomo</nombre>
21        <precio>4.95 + &cargo; %</precio>
22    </comida>
23    <comida>
24        <nombre>Bocadillo de jamón</nombre>
25        <precio>5.95 + &cargo; %</precio>
```

```
26    </comida>
27    <comida>
28        <nOMBRE>Bocadillo de tortilla</nOMBRE>
29        <precio>3.95 + &cargo; %</precio>
30    </comida>
31 </menu>
```

1.4.4. Procesamiento de datos mediante XSLT

Como hemos visto hasta ahora, XML es un formato de archivo utilizado para estructurar e intercambiar información. No obstante, un sistema no tiene por qué necesitar siempre todos los datos contenidos en un XML, o puede requerirlos en un formato diferente.

Para dar solución a esto, surge XSLT (*Extensible Stylesheet Language Transformations*), una especificación que permite transformar un documento XML en otro de forma sencilla, pudiendo así obtener un documento con los datos que se requieran sin tener que reescribirlo. Igualmente, también es posible asociar una hoja de estilos a un archivo XML, permitiendo que su contenido se pueda mostrar con el aspecto de un documento HTML.

Por ejemplo, aplicando XSLT sería posible hacer que un documento que almacena información sobre un menú, con sus correspondientes productos y precios, se transformara en un documento HTML en el que apareciera el producto y su precio formateado como una tabla.

Este proceso de transformación puede ocurrir en diferentes entornos: i) sobre un proceso externo, ii) sobre un servidor web, o iii) sobre un navegador web. En el último caso, será el navegador web el que realice la transformación y ofrezca al usuario el documento XML transformado.

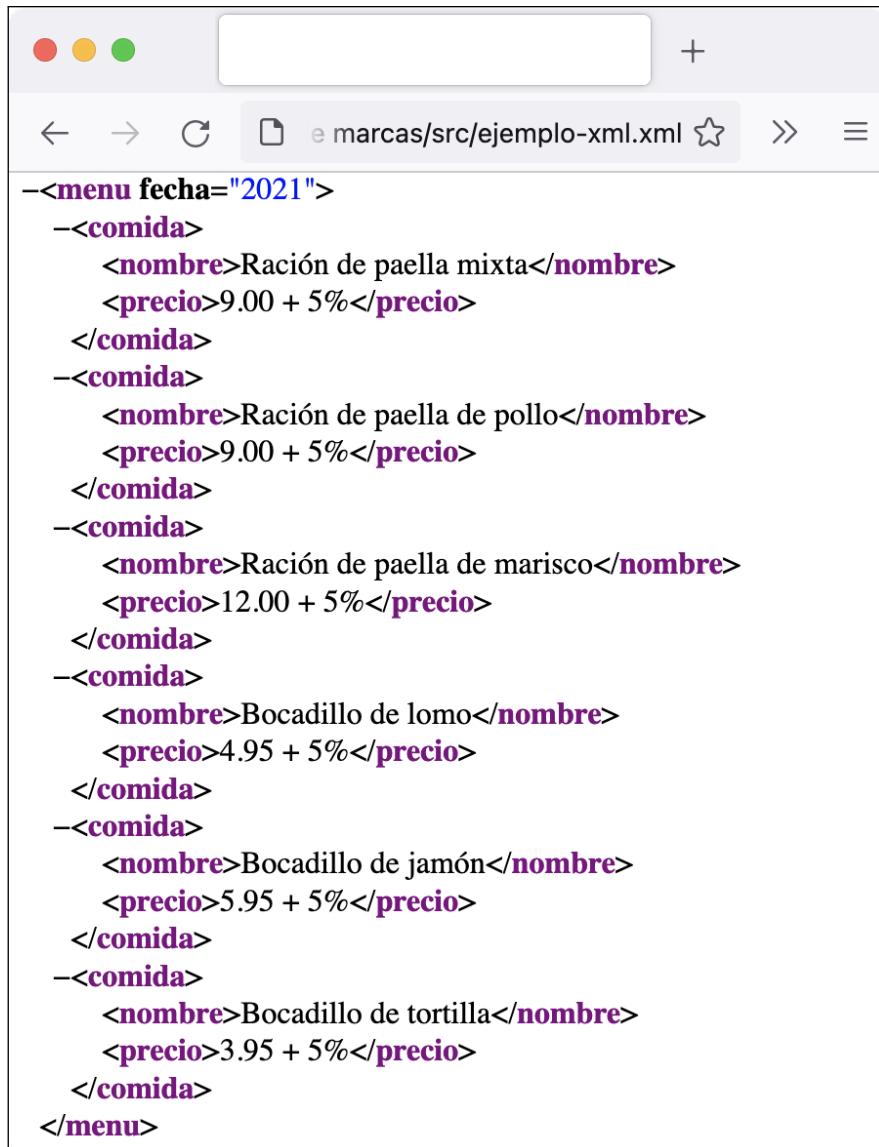
Definición de transformaciones

Un documento XSL es un documento XML con el espacio de nombres <http://www.w3.org/1999/XSL/Transform> y cuyo elemento raíz es `xsl:stylesheet`. Se utiliza para definir las transformaciones que se aplicarán sobre el documento XML que contiene los datos. Su sintaxis es la de XML y su vocabulario consta de etiquetas especiales que representan operaciones concretas a realizar con el texto del documento fuente. Así, su cabecera quedaría como:

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <xsl:stylesheet version="1.0" xmlns:xsl= "http://www.w3.org/1999/XSL/Transform" />
```

Por otra parte, habría que editar el documento XML al que se deseen aplicar las transformaciones del documento XSL para realizar la asociación entre ambos:

```
1 <?xml version="1.0" encoding="UTF-8"?>
```



The screenshot shows a web browser window with the address bar containing 'marcas/src/ejemplo-xml.xml'. The main content area displays an XML document with the following structure:

```
--<menu fecha="2021">
  --<comida>
    <nombre>Ración de paella mixta</nombre>
    <precio>9.00 + 5%</precio>
  --</comida>
  --<comida>
    <nombre>Ración de paella de pollo</nombre>
    <precio>9.00 + 5%</precio>
  --</comida>
  --<comida>
    <nombre>Ración de paella de marisco</nombre>
    <precio>12.00 + 5%</precio>
  --</comida>
  --<comida>
    <nombre>Bocadillo de lomo</nombre>
    <precio>4.95 + 5%</precio>
  --</comida>
  --<comida>
    <nombre>Bocadillo de jamón</nombre>
    <precio>5.95 + 5%</precio>
  --</comida>
  --<comida>
    <nombre>Bocadillo de tortilla</nombre>
    <precio>3.95 + 5%</precio>
  --</comida>
</menu>
```

Figura 1.26: Resultado de abrir el documento del Listado 1.33 en un navegador web. Nótese como las entidades han sido reemplazados por sus valores asignados.

2 <?xmlstylesheet type="text/xsl" href="archivo-de-transformaciones.xsl"?>



La referencia oficial de los estándares XSLT puede encontrarse en [Enlace: www.w3.org/TR/xslt/all/](http://www.w3.org/TR/xslt/all/).

XPath

Podemos escribir expresiones regulares para recorrer y procesar un documento XML mediante el uso de XPath, un estándar que forma parte de XSLT y que sirve para navegar por los elementos y atributos de un documento XML. Por ejemplo, si tenemos el siguiente elemento de un documento XML:

```
1 <producto>Tortilla</producto>
```

Y el documento XSL contiene una plantilla como:

```
1 <xsl:template match="producto">
2   <h2><xsl:value-of /></h2>
3 </xsl:template>
```

Cuando el procesador XSLT encuentre dicho elemento en el XML, lo transformará en un elemento `<h2></h2>` incluyendo el contenido del elemento gracias a la expresión `xsl:value-of />`, resultando en un documento HTML:

```
1 <h2>Tortilla</h2>
```

También podremos acceder a elementos anidados utilizando el carácter de la barra (/), como si nos estuviéramos moviendo por los directorios del sistema operativo utilizando la terminal. En el caso concreto de los atributos, podremos referirnos a ellos utilizando el carácter @.

Plantillas y manipulación de datos

Las instrucciones de plantillas son las encargadas de describir las reglas que especifican la transformación a aplicar a los elementos del documento XML:

- **<xsl:template>**: utilizada para definir las reglas de transformación para los nodos del documento XML que coinciden con la expresión XPath especificada en el atributo `match`.
- **<xsl:apply-templates>**: utilizada para aplicar todas las plantillas posibles a los elementos coincidentes con la expresión XPath que acompaña al atributo `select`. Si no se especifica este atributo, todos los nodos hijos del nodo actual serán seleccionados.

Así, podemos definir plantillas utilizando la expresión `<xsl:template>` y aplicarlas en determinados puntos de nuestro documento mediante la expresión `<xsl:apply-templates>`.

Por otra parte, podemos utilizar la expresión `<xsl:value-of>` para extraer el contenido del nodo proporcionado mediante el atributo `select`, o el valor del atributo de dicho nodo. En este último caso, la ruta al nodo estará precedida por el carácter @.

En el Listado 1.34 se muestra un archivo de transformaciones que aplicaremos sobre el documento XML del Listado 1.35. Nótese como se ha enlazado el documento XSL en el documento XML.

Listado 1.34: Ejemplo de transformaciones con XSLT.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <xsl:stylesheet version="1.0"
3   xmlns:xsl= "http://www.w3.org/1999/XSL/Transform">
4
5 <xsl:template match="/">
6   <html lang="es">
7     <head>
8       <meta charset="UTF-8" />
9       <meta name="viewport" content="width=device-width, initial-scale=1.0" />
10      <title>Ejemplo de transformaciones de datos</title>
11    </head>
12    <body>
13      <h2>Menú del restaurante</h2>
14      <ul>
15        <xsl:apply-templates />
16      </ul>
17    </body>
18  </html>
19 </xsl:template>
20
21 <xsl:template match="/menu/comida">
22   <li><xsl:value-of select="nombre" /></li>
23 </xsl:template>
24
25 </xsl:stylesheet>
```

Listado 1.35: Ejemplo de documento XML que será transformado.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <?xml-stylesheet type="text/xsl" href="ejemplo-transformaciones-de-datos.xsl"?>
3 <!DOCTYPE menu [
4   <!ENTITY fechaActual "2021">
5   <!ENTITY cargo "5">
6 ]>
7 <menu fecha=&fechaActual;">
8   <comida>
9     <nombre>Ración de paella mixta</nombre>
10    <precio>9.00 + &cargo; %</precio>
11  </comida>
12  <comida>
13    <nombre>Ración de paella de pollo</nombre>
14    <precio>9.00 + &cargo; %</precio>
15  </comida>
16  <comida>
17    <nombre>Ración de paella de marisco</nombre>
18    <precio>12.00 + &cargo; %</precio>
19  </comida>
20  <comida>
21    <nombre>Bocadillo de lomo</nombre>
```

```
22      <precio>4.95 + &cargo; %</precio>
23  </comida>
24  <comida>
25      <nOMBRE>Bocadillo de jamón</nOMBRE>
26      <precio>5.95 + &cargo; %</precio>
27  </comida>
28  <comida>
29      <nOMBRE>Bocadillo de tortilla</nOMBRE>
30      <precio>3.95 + &cargo; %</precio>
31  </comida>
32 </menu>
```

Tras aplicar la transformación al documento XML, se obtendrá un documento HTML como el del Listado 1.36.

Listado 1.36: Resultado de transformar el documento XML del Listado 1.35 en un documento HTML.

```
1 <html lang="es">
2   <head>
3     <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
4     <meta charset="UTF-8" />
5     <meta name="viewport" content="width=device-width, initial-scale=1.0" />
6     <title>Ejemplo de transformaciones de datos</title>
7   </head>
8   <body>
9     <h2>Menú del restaurante</h2>
10    <ul>
11      <li>Ración de paella mixta</li>
12      <li>Ración de paella de pollo</li>
13      <li>Ración de paella de marisco</li>
14      <li>Bocadillo de lomo</li>
15      <li>Bocadillo de jamón</li>
16      <li>Bocadillo de tortilla</li>
17    </ul>
18  </body>
19 </html>
```



Es posible que el navegador no pueda enlazar el documento XSL al abrir el documento XML a transformar, debido a la política de seguridad *CORS*. La solución consiste en ejecutar un servidor web en el directorio de los ejemplos utilizando alguna herramienta, por ejemplo, mediante el comando `$ python -m http.server`. Así, podremos dirigirnos a la dirección `http://localhost:8000` para abrir los ejemplos correctamente.

Instrucciones iterativas

Una de las principales ventajas de XSL es la posibilidad de efectuar operaciones de control sobre el documento. Para ello, consta de instrucciones XSLT que permiten realizar tanto iteraciones como estructuras condicionales. Una estructura iterativa se representa en XSLT mediante el elemento `<xsl:for-each></xsl:for-each>` y el atributo `select` para indicar el nodo del documento XML sobre el que se debe iterar. Además, también se pueden especificar condiciones para filtrar los resultados. El contenido del elemento será aquello que se repetirá en el documento tras la transformación.

Podemos ordenar los resultados utilizando la expresión `<xsl:sort>` y los atributos `select` para indicar elemento o atributo sobre el que queremos ordenar, `order` para indicar si el sentido de la ordenación será ascendente (`ascending`) o descendente (`descending`) y `data-type` para especificar si queremos realizar la ordenación de manera alfabética (`text`) o numérica (`number`).

En el Listado 1.37 se muestra un archivo de transformaciones que aplicaremos sobre el documento XML del Listado 1.38. Nótese como en el primero se utiliza una condición para filtrar aquellos productos con un precio mayor o igual a 9 euros.

Listado 1.37: Ejemplo de instrucciones iterativas con XSLT.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <xsl:stylesheet version="1.0"
3     xmlns:xsl= "http://www.w3.org/1999/XSL/Transform">
4
5 <xsl:template match="/">
6     <html lang="es">
7         <head>
8             <meta charset="UTF-8" />
9             <meta name="viewport" content="width=device-width, initial-scale=1.0" />
10            <title>Ejemplo de instrucciones iterativas</title>
11        </head>
12        <body>
13            <h2>Menú de paellas</h2>
14            <table>
15                <xsl:for-each select="menu/comida[precio >= 9]">
16                    <xsl:sort select="precio" order="descending" data-type="number" />
17                    <tr>
18                        <td><xsl:value-of select="nombre" /></td>
19                        <td><xsl:value-of select="precio" /></td>
20                    </tr>
21                </xsl:for-each>
22            </table>
23        </body>
24    </html>
25 </xsl:template>
26
27 </xsl:stylesheet>
```

Listado 1.38: Ejemplo de documento XML que será transformado.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <?xml-stylesheet type="text/xsl" href="ejemplo-instrucciones-iterativas.xsl"?>
3 <menu>
4     <comida>
5         <nOMBRE>Ración de paella mixta</nOMBRE>
6         <precio>9.00</precio>
7     </comida>
8     <comida>
9         <nOMBRE>Ración de paella de pollo</nOMBRE>
10        <precio>9.00</precio>
11    </comida>
12    <comida>
13        <nOMBRE>Ración de paella de marisco</nOMBRE>
14        <precio>12.00</precio>
15    </comida>
16    <comida>
17        <nOMBRE>Bocadillo de lomo</nOMBRE>
18        <precio>4.95</precio>
19    </comida>
20    <comida>
21        <nOMBRE>Bocadillo de jamón</nOMBRE>
22        <precio>5.95</precio>
23    </comida>
24    <comida>
25        <nOMBRE>Bocadillo de tortilla</nOMBRE>
26        <precio>3.95</precio>
27    </comida>
28 </menu>
```

Tras aplicar la transformación al documento XML, se obtendrá un documento HTML como el del Listado 1.39.

Listado 1.39: Resultado de transformar el documento XML del Listado 1.38 en un documento HTML.

```
1 <html lang="es">
2 <head>
3     <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
4     <meta charset="UTF-8" />
5     <meta name="viewport" content="width=device-width, initial-scale=1.0" />
6     <title>Ejemplo de instrucciones iterativas</title>
7 </head>
8 <body>
9     <h2>Menú de paellas</h2>
10    <table>
11        <tbody>
12            <tr>
13                <td>Ración de paella de marisco</td>
14                <td>12.00</td>
15            </tr>
16            <tr>
17                <td>Ración de paella mixta</td>
18                <td>9.00</td>
19            </tr>
20            <tr>
21                <td>Ración de paella de pollo</td>
22                <td>9.00</td>
```

```
23         </tr>
24     </tbody>
25   </table>
26 </body>
27 </html>
```

Instrucciones condicionales

XSLT nos permite especificar instrucciones condicionales del tipo `if` y `switch` mediante los elementos `<xsl:if test="..."/></xsl:if>` y `<xsl:choose></xsl:choose>` respectivamente. El primero nos permitirá aplicar una transformación siempre y cuando se cumpla la condición especificada en el atributo `test`. El segundo nos permitirá definir múltiples casos mediante elementos `<xsl:when test="..."/></xsl:when>` y un caso que se ejecutará por defecto en caso de que no se cumpla ninguna condición, definido por el elemento `<xsl:otherwise></xsl:otherwise>`.

En el Listado 1.40 se muestra un archivo de transformaciones que aplicaremos sobre el documento XML del Listado 1.41. Nótese como en el primero se utiliza una condición para filtrar aquellos productos con un precio mayor o igual a 9 euros.

Listado 1.40: Ejemplo de instrucciones condicionales con XSLT.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <xsl:stylesheet version="1.0"
3      xmlns:xsl= "http://www.w3.org/1999/XSL/Transform">
4
5  <xsl:template match="/">
6      <html lang="es">
7          <head>
8              <meta charset="UTF-8" />
9              <meta name="viewport" content="width=device-width, initial-scale=1.0" />
10             <title>Ejemplo de instrucciones condicionales</title>
11         </head>
12         <body>
13             <h2>Menú de bocadillos</h2>
14             <table>
15                 <xsl:for-each select="menu/comida">
16                     <xsl:if test="precio < 6">
17                         <tr>
18                             <td><xsl:value-of select="nombre" /></td>
19                             <td><xsl:value-of select="precio" /></td>
20                         </tr>
21                     </xsl:if>
22                 </xsl:for-each>
23             </table>
24
25             <h2>Menú de Paellas baratas</h2>
26             <table>
27                 <xsl:for-each select="menu/comida">
28                     <xsl:choose>
29                         <xsl:when test="precio >= 9 and precio < 10">
30                             <tr>
31                                 <td><xsl:value-of select="nombre" /></td>
```

```
32             <td><xsl:value-of select="precio" /></td>
33         </tr>
34     </xsl:when>
35     <xsl:otherwise>
36         <tr>
37             <td><del><xsl:value-of select="nombre" /></del></td>
38             <td><del><xsl:value-of select="precio" /></del></td>
39         </tr>
40     </xsl:otherwise>
41   </xsl:choose>
42 </xsl:for-each>
43 </table>
44 </body>
45 </html>
46 </xsl:template>
47
48 </xsl:stylesheet>
```

Listado 1.41: Ejemplo de documento XML que será transformado.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <?xml-stylesheet type="text/xsl" href="ejemplo-instrucciones-condicionales.xsl"?>
3 <menu>
4     <comida>
5         <nombre>Ración de paella mixta</nombre>
6         <precio>9.00</precio>
7     </comida>
8     <comida>
9         <nombre>Ración de paella de pollo</nombre>
10        <precio>9.00</precio>
11    </comida>
12    <comida>
13        <nombre>Ración de paella de marisco</nombre>
14        <precio>12.00</precio>
15    </comida>
16    <comida>
17        <nombre>Bocadillo de lomo</nombre>
18        <precio>4.95</precio>
19    </comida>
20    <comida>
21        <nombre>Bocadillo de jamón</nombre>
22        <precio>5.95</precio>
23    </comida>
24    <comida>
25        <nombre>Bocadillo de tortilla</nombre>
26        <precio>3.95</precio>
27    </comida>
28 </menu>
```

Tras aplicar la transformación al documento XML, se obtendrá un documento HTML como el del Listado 1.42.

Listado 1.42: Resultado de transformar el documento XML del Listado 1.41 en un documento HTML.

```
1 <html lang="es">
2 <head>
3   <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
4   <meta charset="UTF-8" />
5   <meta name="viewport" content="width=device-width, initial-scale=1.0" />
6   <title>Ejemplo de instrucciones condicionales</title>
7 </head>
8 <body>
9   <h2>Menú de bocadillos</h2>
10  <table>
11    <tbody>
12      <tr>
13        <td>Bocadillo de lomo</td>
14        <td>4.95</td>
15      </tr>
16      <tr>
17        <td>Bocadillo de jamón</td>
18        <td>5.95</td>
19      </tr>
20      <tr>
21        <td>Bocadillo de tortilla</td>
22        <td>3.95</td>
23      </tr>
24    </tbody>
25  </table>
26  <h2>Menú de Paellas baratas</h2>
27  <table>
28    <tbody>
29      <tr>
30        <td>Ración de paella mixta</td>
31        <td>9.00</td>
32      </tr>
33      <tr>
34        <td>Ración de paella de pollo</td>
35        <td>9.00</td>
36      </tr>
37      <tr>
38        <td><del>Ración de paella de marisco</del></td>
39        <td><del>12.00</del></td>
40      </tr>
41      <tr>
42        <td><del>Bocadillo de lomo</del></td>
43        <td><del>4.95</del></td>
44      </tr>
45      <tr>
46        <td><del>Bocadillo de jamón</del></td>
47        <td><del>5.95</del></td>
48      </tr>
49      <tr>
50        <td><del>Bocadillo de tortilla</del></td>
51        <td><del>3.95</del></td>
52      </tr>
53    </tbody>
54  </table>
55 </body>
56 </html>
```


2

Capítulo

Plataformas de Inteligencia Artificial

Cristian Gómez Portes

En este capítulo se lleva a cabo una introducción general a las plataformas de IA desde la perspectiva de la **computación en la nube**. En primer lugar, se define el término de computación en la nube y se pone en contexto la importancia de este enfoque para el desarrollo de aplicaciones de IA. Adicionalmente, se discuten los beneficios como consecuencia de usar la nube.

Esta introducción, además, se complementa con una visión general sobre el desarrollo de aplicaciones de IA mediante estas plataformas. Primero, se describe un **flujo de trabajo** típico y los retos comunes de pasar una aplicación en la fase de prototipo a producción. Además, se comparten una serie de buenas prácticas sobre cómo mejorar las etapas de su desarrollo. Es importante destacar que estas dos cuestiones son meramente orientativas, dado que este capítulo se centra en el desarrollo de aplicaciones mediante servicios de IA. Más tarde, se introducen algunas de las principales **plataformas** que hacen posible la construcción de soluciones con comportamiento inteligente. Y, posteriormente, se discuten algunas de las **oportunidades** que estas plataformas brindan con el objetivo de ofrecer al lector una visión general sobre sus beneficios.

Por otra parte, este capítulo profundiza en los detalles de las plataformas de IA desde un punto de vista general, es decir, los **aspectos fundamentales** que caracterizan a estos entornos. En dicho recorrido se introducen algunos ejemplos prácticos para mostrar al lector sus particularidades. Por otro lado, se realiza una comparativa de las plataformas existentes en base a sus características, y se ofrece al lector una serie de recomendaciones para elegir la plataforma más adecuada según la aplicación a desarrollar.

Este capítulo termina con un caso de estudio que plantea el diseño e implementación, desde cero, de una aplicación conectada a un servicio inteligente en la nube. En este caso se usa el proveedor de servicios en la nube AWS (Amazon Web Services) para **facilitar** y **acelerar** el desarrollo de aplicaciones de IA al considerarse uno de los más usados hasta la fecha.

2.1. Introducción a las Plataformas de IA

2.1.1. Computación en la nube para la IA

La inteligencia artificial hoy en día vive su momento de esplendor, pero eso no significa que sea una tecnología plenamente novedosa. De hecho, durante décadas ha habido numerosos avances significativos que han puesto esta tecnología en el podio de soluciones más prometedoras. Sin embargo, la ausencia de máquinas capaces de ejecutar livianamente estas soluciones implicó que esta tecnología perdiera repercusión progresivamente. La explosión de la IA no ha tenido lugar hasta día de hoy, principalmente por las limitaciones técnicas de la época y la complejidad inherente de la tarea.

Es por esto que la revolución de la IA en la actualidad pasa por la **evolución del hardware**, el cual ha permitido incrementar las capacidades de estas soluciones. Los sistemas basados en IA requieren de grandes conjuntos de datos que solo pueden ser procesados por computadores con arquitecturas notables. Esto lo que permite es entrenar los modelos subyacentes de los sistemas de IA para tomar así las mejores decisiones. En definitiva, simular en la medida de lo posible un comportamiento lo más fiel al humano.



Hardware como pilar fundamental. La innovación y desarrollo en hardware también supone un aspecto clave en el campo de la IA.

No obstante, en este momento, la IA se encuentra en un punto muy similar al que estaban las bases de datos relacionales a principios de la década de 1990. El mundo del software era conocedor de que esta tecnología iba a ser verdaderamente útil para cualquier compañía. Sin embargo, solo un número reducido de individuos o empresas tenían la capacidad de aprovecharse de ellas por su alta grado de complejidad.

En este sentido, los grandes gigantes tecnológicos, los cuales tienen los medios suficientes para hacer uso de la IA, llevan realizando en los últimos años considerables inversiones para crear soluciones que permitan acercar esta tecnología al mayor número de personas posibles. La premisa es hacer que la IA sea accesible y fácil de usar tanto a profesionales expertos como a gente totalmente inexperta en este campo.

El vehículo para hacer esto posible es a través de la computación en la nube o *cloud computing* (en inglés). Este término no es más que una abstracción o metáfora para hacer referencia a las capas de abstracción ofrecidas bajo hardware virtual en sistemas complejos distribuidos por todo el mundo. Particularmente estas capacidades son las que hacen posible crear **plataformas basadas en IA**, las cuales tratan de impulsar este tecnología al reducir barreras y facilitar su desarrollo al ofrecer un enfoque de todo integrado.



Figura 2.1: Ejemplo de localizaciones de centros de datos repartidos por todo el mundo.

Computación en la nube. Aquí se proporciona una traducción de una definición más oficial del NIST (National Institute of Standards and Technology), Agencia del Departamento de Comercio de los Estados Unidos: “La computación en la nube es un modelo que permite el acceso ubicuo, cómodo y a la carta a un conjunto compartido de recursos informáticos configurables (p. ej., redes, servidores, almacenamiento, aplicaciones y servicios) que pueden ser rápidamente aprovisionados y liberados con un mínimo esfuerzo de gestión o de interacción con el proveedor de servicios.”



Es cierto que la IA empezó mucho antes que la computación en la nube, pero ésta y sus tecnologías han mejorado y contribuido a hacer realidad soluciones que antes eran inimaginables. La computación en la nube ha sido un **catalizador eficaz** para la IA.

2.1.2. Beneficios de la computación en la nube

La computación en la nube habilita el suministro de recursos TI (Tecnologías de la Información) **bajo demanda** a través de internet mediante pago por su uso. La principal ventaja de este enfoque es que un usuario no necesita comprar, disponer o mantener un CPD (Centro de Procesamiento de Datos) o *data center* (en inglés) para desplegar sus soluciones, sino que solo necesita adquirir potencia de cómputo, almacenamiento, bases de datos y otros servicios en función de sus necesidades. Una analogía con esto que acabamos de ver podría reflejarse en el funcionamiento de una compañía eléctrica cuando se envía electricidad en el momento en el que accionamos un interruptor de luz en nuestra casa. En este caso, la nube suministra recursos informáticos a la carta con el clic de un botón o la invocación de una API. Veamos a continuación el beneficio del cloud computing en el contexto de la IA.

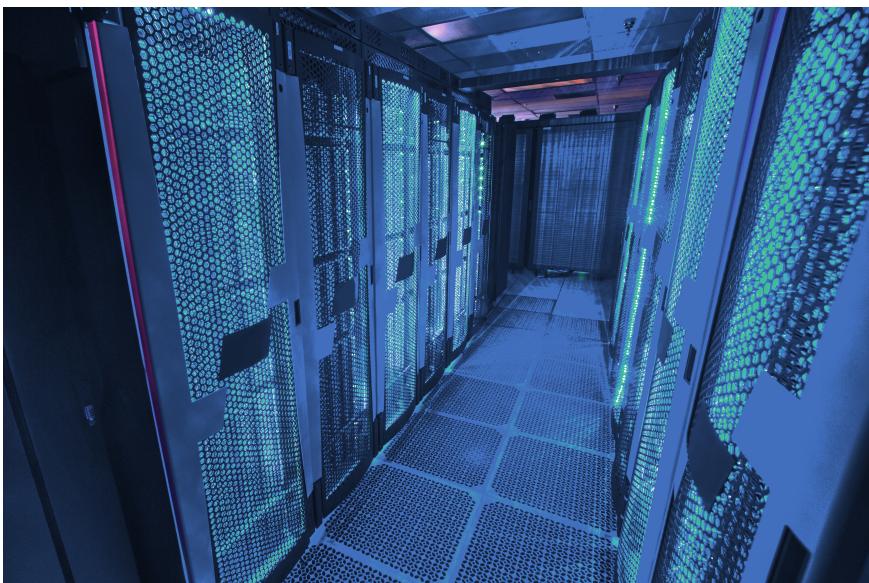


Figura 2.2: Parte de un CDP con servidores a los lados del pasillo. Imagen: Freepick.com.

Agilidad

La computación en la nube nos permite disponer de **recursos a medida que se necesitan**. Esto permite que usuarios puedan experimentar con diferentes servicios de manera rápida sin la necesidad de realizar grandes desarrollos. Imagina que queremos entrenar un modelo de procesamiento del lenguaje natural. El proveedor de servicios en la nube habilita una infraestructura en cuestión de minutos con diez, cien o mil servidores para ejecutar nuestra tarea. En el caso de que el experimento

falle, podemos eliminar los recursos sin ningún riesgo. También es posible **automatizar despliegues** mediante código para crear redes, iniciar clusters de servidores virtuales o desplegar una base de datos relacional, entre otros. La automatización aumenta la fiabilidad y mejora la eficiencia.

Ahorro de costes

La computación en la nube se basa en el modelo de **pago por uso**. La ventaja de esto es que un usuario no tiene que hacer frente a inversiones a largo plazo en infraestructuras que pueden quedarse obsoletas en pocos meses. Así, el usuario contrata los recursos necesarios para su tarea y solo paga por su consumo, ya sea por horas de uso, cantidad de tráfico o almacenamiento. Además, esto hace que la tolerancia a fallos sea asequible para pequeños presupuestos, dado que, por ejemplo, un sistema puede dividirse en dos partes más pequeñas con la misma capacidad.

Elasticidad

Una de las características más útiles de la computación en la nube es la capacidad de automáticamente **escalar recursos** en función de la carga de la aplicación. En otras palabras, la infraestructura en la nube monitoriza la carga de los recursos operativos, incrementándolos o decrementándolos en base a las necesidades de la aplicación. Por ejemplo, imagina un servidor que aloja un modelo para predecir matrículas de coches. La infraestructura de nuestra aplicación en la nube puede automáticamente escalar en el caso que las peticiones aumenten considerablemente. Similarmente, el número de recursos puede decrementar si el pico de solicitudes desciende. La nube apenas tiene límites de capacidad. No es necesario pensar en el espacio de los servidores, fuentes de alimentación, etc. Es decir, si el volumen de datos crece siempre se puede añadir nueva capacidad de almacenamiento. Esto evita que un usuario contrate más recursos de los que necesita para hacer frente a potenciales picos de carga en el futuro. El propio entorno cloud es quien se encarga de estas tareas.

Innovación rápida

La computación en la nube permite innovar con mayor rapidez, ya que el foco está en desarrollar aplicaciones que diferencien nuestro negocio más que dedicar tiempo a la ardua tarea de gestionar la infraestructura hardware y software. En esencia, la nube ayuda a **experimentar** con nuevos algoritmos, frameworks y hardware en segundos frente a meses.

Despliegue en minutos

La nube ofrece la posibilidad de desplegar aplicaciones en cuestión de minutos. La economía global en la que vivimos obliga a estar cerca de los clientes. La mayoría de proveedores de servicios en la nube hacen uso del concepto de **región**, el cual es una zona física geográfica donde se encuentran los recursos computacionales contratados. Esto ayuda necesariamente a **mejorar el rendimiento** de las aplicaciones con tiempos de respuesta extremadamente rápidos y **cumplir con las restricciones de privacidad de datos** de cada región. A modo de ejemplo, imagina que una empresa española está desarrollando un modelo de IA para la conducción autónoma en Reino Unido. Es evidente que las respuestas que genere el modelo deben ser prácticamente instantáneas para controlar adecuadamente el vehículo y evitar accidentes. Para ello, el modelo debe de estar desplegado en un *cluster*¹ en el área de Reino Unido para ofrecer un alto rendimiento y un buen balanceo de carga frente al potencial masivo conjunto de peticiones. Además, los datos que recoja el modelo para retroalimentarse estarán bajo las leyes de Reino Unido y no de España.

Transición fluida de prototipado a producción

La creación de prototipos suele realizarse en entornos de desarrollo locales o en cuadernos en la nube (p. ej. Jupyter Notebooks²). Este enfoque funciona bien para pequeños conjuntos de datos. Sin embargo, en el caso opuesto los recursos de CPU (Central Processing Unit) y RAM (Random Access Memory) de una sola máquina se disparan rápidamente, perjudicando el rendimiento. Además, cuando se quiere acelerar el entrenamiento de modelos de IA suele hacerse uso de GPUs, lo cual implica utilizar diferentes máquinas.

Otra problemática existente está relacionada con desplegar los modelos desarrollados como prototipos en entornos de producción. Por un lado, la aplicación necesita manejar miles o millones de peticiones en tiempo real. Por otro lado, el despliegue de estas aplicaciones requiere de la colaboración entre diferentes equipos de la empresa, como puede ser el de ciencia de datos, ingeniería de datos, DEVOPS³, etc. Aparte, una vez la aplicación es exitosamente desplegada se requiere de una continua monitorización con el objetivo de reaccionar a incidentes que puedan ocurrir relacionados con el rendimiento, calidad y acceso, entre otros.

Por lo tanto, uno de los beneficios que ofrece la nube es **flexibilizar** los mecanismos para mover un modelo prototípico a un entorno en producción con una infraestructura física robusta. Además, los servicios en la nube proporcionan herramientas para automatizar y facilitar el flujo de trabajo y despliegue de modelos en un entorno de producción escalable y de alto rendimiento.

¹Conjunto de dispositivos físicos unidos entre sí mediante una red de gran velocidad que actúan como un solo dispositivo lógico.

² Enlace: <https://jupyter.org/>

³ Enlace: <https://es.wikipedia.org/wiki/DevOps>



No hay balas de plata. Si bien es cierto que la tecnología en la nube ofrece una infinidad de beneficios para el desarrollo de aplicaciones y, concretamente, en IA, no creas que esta solución resuelve cualquier problema de manera mágica. Es importante tener claro el objetivo de nuestra solución y emplear los recursos necesarios para ello; puede ser que nuestro desarrollo no vaya a tener un impacto global y sencillamente un servidor alojado en casa sea suficiente para desplegar nuestra aplicación.

2.1.3. El concepto de plataforma de IA

En el campo de la informática es muy común que constantemente aparezcan nuevas soluciones para facilitar, sobre todo, el desarrollo software. Es por eso que existen multitud de herramientas que tratan de facilitar el rol de los desarrolladores en la tarea de generar código. Particularmente, en el contexto de los videojuegos, ha habido una evolución exponencial respecto al desarrollo de motores de juegos que faciliten y reduzcan considerablemente la complejidad de crearlos de un modo relativamente ágil y sencillo. Algo parecido ha ocurrido con el desarrollo de aplicaciones basadas en IA con la llegada de plataformas que ofrecen servicios en la nube.

No existe exactamente un concepto que defina oficialmente una plataforma de IA, pero, a grandes rasgos, esto se asocia a un ecosistema completo cargado de herramientas y servicios que facilitan el desarrollo de aplicaciones inteligentes y que **ocultan la complejidad inherente de la tarea**. Estas plataformas no son exclusivas de la nube, pero este capítulo se centra en este aspecto por ser el vehículo común para desarrolladores con el que beneficiarse de la IA.

El origen de estas plataformas de IA es algo relativamente novedoso. Estos ecosistemas surgen de la mano de las infraestructuras cloud por disponer de recursos masivos TI con los que alimentar modelos de aprendizaje automático. Debido a su naturaleza, cuantos más datos reciba el sistema, mejores serán las decisiones a tomar. Es por esto que las plataformas de IA en la nube ofrecen soluciones completamente sofisticadas.

En el caso de las aplicaciones de IA, la **fiabilidad** y **efectividad** son dos factores que han de cumplirse para satisfacer las necesidades del usuario, y así evitar que esta se perciba simplemente como algo novedoso sin una finalidad y utilidad concreta. En este contexto, los servicios y herramientas de las plataformas de IA deben satisfacer estas exigencias, y permitir que el comportamiento de las aplicaciones desarrolladas bajo estos entornos cumplan unos criterios aceptables de funcionamiento. En otras palabras, estas aplicaciones están marcadas por el resultado que arrojan, el cual debe cumplir una serie de requisitos en base a la finalidad de la misma para no perder realismo. Obviamente, las plataformas no hacen todo este trabajo solas. Debe haber un compromiso entre desarrollador y entorno para lograr una sensación acorde a lo que el usuario espera.

Este enfoque, entornos cloud para IA, se ha expandido considerablemente a lo largo de los años y ha ido evolucionando en consecuencia. Particularmente por el surgimiento de nuevas técnicas que han dado lugar a servicios y/o herramientas en la nube. Su finalidad es que puedan ser utilizadas por cualquiera, independientemente de su tamaño o sofisticación técnica, y que ofrezcan, generalmente, resultados sorprendentes. También, en parte, por la urgencia comercial de un número reducido de compañías por dominar este mercado y establecerse como líder de la IA en la nube.

Como conclusión final, y desde un punto de vista general, es importante resaltar que el desarrollo de una aplicación de IA implica considerar diversos factores, al margen de la elección de la plataforma, que incrementan la complejidad del mismo, como garantizar la seguridad de los datos, fiabilidad, efectividad, etc. El desarrollador debe tener en cuenta estas particularidades para ofrecer una adecuada experiencia al usuario.

2.1.4. Desarrollo de aplicaciones de IA en entornos cloud

Flujo de trabajo para el desarrollo de aplicaciones de IA

El flujo de trabajo de una aplicación de IA, en líneas generales, involucra pasos complejos, multidisciplinares e iterativos. Sin embargo, estos pasos pueden variar considerablemente en base al comportamiento de la aplicación. Por lo general, una aplicación de IA se diferencia en dos escenarios: i) si el modelo que se necesita debe de ser desarrollado desde cero, es decir, el modelo es personalizado o, por el contrario, ii) puede utilizarse uno estándar ya preentrenado. El segundo caso será en el que profundice este capítulo de Plataformas de Inteligencia Artificial. No obstante, con el objetivo de ofrecer al lector una visión general de las posibilidades de estas plataformas, esta sección aborda los pasos del desarrollo y despliegue de un modelo de IA desde cero. El siguiente capítulo profundizará en detalle alguno de los aspectos del primer escenario que ahora veremos de manera sintetizada.

Tomemos como ejemplo un flujo de trabajo típico para desarrollar modelos de aprendizaje automático. El primer paso consiste en la preparación de los datos. Más tarde es necesario pasar el modelo a la fase de entrenamiento y ajuste con el objetivo de verificar que el modelo es adecuado para resolver el problema dado. Finalmente, se despliega el modelo a un entorno de producción. Cada uno de estos pasos consta de varias subtareas, como se muestra en la figura 2.3.

La primera fase del flujo de trabajo, es decir, aquella relacionada con la **preparación de los datos**, se divide en un conjunto de pasos. En primer lugar, para modelos de aprendizaje automático personalizados, se comienza con la ingestión de datos en bruto y su exploración, incluyendo análisis de datos, evaluación de calidad de los datos, estadísticas, valores perdidos, cálculos de cuantiles, análisis de sesgo de los datos, análisis de correlación, etc.

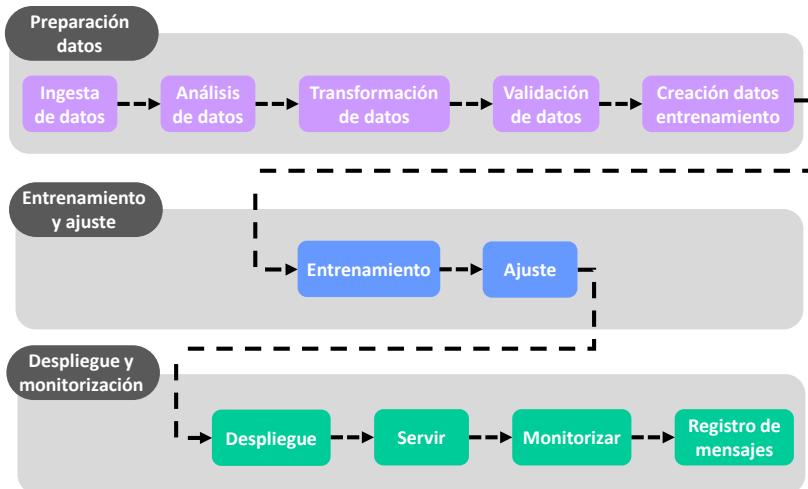


Figura 2.3: Flujo de trabajo para el desarrollo de una aplicación de aprendizaje automático.

En esta misma fase, se define el tipo de problema de aprendizaje automático, es decir, regresión, clasificación, agrupamiento o *clustering* (en inglés), etc. Después de identificar el tipo de problema, se selecciona el algoritmo que mejor se adapta al problema, por ejemplo, árboles de decisión, regresión logística, *random forest* o *k-means*, entre otros. Dependiendo del algoritmo elegido es necesario seleccionar un subconjunto de datos para entrenar, validar y probar el modelo. Generalmente, los conjuntos de datos usados para modelos de aprendizaje automático necesitan ser transformados en vectores matemáticos para optimizaciones numéricas y el entrenamiento del modelo. Por ejemplo, se podría transformar columnas categóricas en unos y ceros (*one-hot encoding vector*) o incluso convertir columnas en texto a vectores numéricos (*word-embedding vector*). Una vez transformado el subconjunto de datos brutos en características, estas deben ser divididas en conjuntos de entrenamiento, validación y prueba para preparar el entrenamiento, el ajuste y la prueba del modelo.

Durante la fase de **entrenamiento y ajuste**, se entrena el modelo con el algoritmo elegido utilizando el conjunto de datos de entrenamiento. Esto sirve para verificar que el código del modelo y el algoritmo es adecuado para resolver el problema identificado. Posteriormente, se ajustan los parámetros del algoritmo y se evalúa el rendimiento del modelo contra el conjunto de validación; esto relativo a la fase de ajuste. Estos pasos se repiten tantas veces como sea necesario hasta que el modelo arroja los resultados esperados en el conjunto de pruebas. Es importante destacar que los resultados deben estar en línea con los objetivos de negocio antes de desplegar el modelo en producción.

Por último, en la fase de **despliegue y monitorización**, el paso final es servir el prototipo en una infraestructura en la nube que garantice seguridad, disponibilidad, acceso, rendimiento, etc. También es importante monitorizar el estado del modelo, así como registrar los mensajes que se generan durante su ejecución. Esta fase suele ser una de las partes más complejas para los profesionales de IA.



No reinventes la rueda. Si la aplicación a implementar necesita de un modelo de IA estándar, por ejemplo, reconocimiento o análisis de imágenes, no es necesario que sigas los pasos anteriormente especificados. En este caso, los proveedores de servicios en la nube cuentan con modelos preentrenados para satisfacer este tipo de necesidades. El desarrollador solo tiene que contratar el servicio en la nube y conectarlo a su aplicación. Más adelante describiremos los pasos sobre cómo realizar esto último.

Principio MLOps como estrategia efectiva

Anteriormente se han discutido los pasos que se requieren para desplegar una aplicación de IA en producción. Es obvio que existe una complejidad alta desde implementar el código de un modelo hasta entrenarlo. Sin embargo, el verdadero reto consiste en tener el modelo funcionando en un entorno de producción que escale y que pueda lidiar con datos cambiantes. En otras palabras, el objetivo es que el modelo se encuentre en un sistema integrado y operarlo en producción continuamente. Se recomienda al lector revisar el artículo *Machine Learning: The High Interest Credit Card of Technical Debt* [S⁺14] donde se estudian las principales razones por las cuales los sistemas de IA fallan normalmente en entornos de producción.

La necesidad de hacer frente a estos retos y a otros matices menores de la implantación de sistemas de IA ha dado lugar al concepto, relativamente nuevo, de MLOPS (Machine Learning Operations). A rasgos generales, MLOPS es un conjunto de buenas prácticas con el objetivo de automatizar el ciclo de vida de los sistemas de IA para **mitigar la deuda técnica**. Es decir, unificar los pasos de validación, transformación, entrenamiento, alojamiento y monitorización de aplicaciones con el objetivo de simplificar el proceso de despliegue ante un cambio, ya sea en el código o en los datos⁴.



MLOPS. Este enfoque ha sido el germen de soluciones completas por parte de proveedores de servicios en la nube que integran herramientas para la gestión del ciclo de vida de aplicaciones de IA.

⁴Se recomienda al lector visitar: Enlace: <https://ml-ops.org/>

Una vez visto el concepto de MLOPS y el beneficio que aporta al desarrollo de aplicaciones de IA, veamos a continuación algunas de las mejores prácticas para resolver retos como la puesta en marcha satisfactoria de aplicaciones, la seguridad, la fiabilidad, la eficiencia del rendimiento y la optimización de los costes.

- **Puesta en marcha satisfactoria.** La creación de modelos o aplicaciones de IA que funcionen de manera satisfactoria en producción requieren seguir una serie de recomendaciones. Veamos a continuación las más destacables: i) haz uso de conjuntos de datos de alta calidad. La mala calidad de los datos provoca el fracaso de muchos proyectos; ii) utiliza soluciones existentes. Hoy en día existen una variedad de servicios y herramientas que integran algoritmos y modelos preentrenados, lo cuales facilitan enormemente el desarrollo de aplicaciones. Comienza con un enfoque simple y beneficiate de esta soluciones; iii) mantén un control de versiones tanto del código como de los datos. Esto es de vital ayuda cuando hay nuevos cambios en la aplicación y ocurren fallos no esperados; iv) define métricas de rendimiento del modelo. Asignar métricas de rendimiento y supervisarlas continuamente permite detectar problemas en el modelo y tomar medidas en consecuencia; finalmente, v) automatiza los flujos de trabajo. Construir procesos de orquestación robustos permite reducir errores humanos y liberar el tiempo del equipo para centrarse en otros asuntos.
- **Seguridad.** La seguridad y cumplimiento son una responsabilidad compartida entre el proveedor de servicios en la nube y el cliente. En otras palabras, el proveedor ofrece una serie de servicios y/o herramientas seguras, mientras que el cliente es responsable de establecer una configuración para cada una de ellas acorde a sus necesidades, de tal forma que el entorno en su totalidad sea seguro. Entre las consideraciones más comunes de seguridad que deben adoptarse se encuentran el acceso a los datos o aplicaciones, métodos de encriptación o aislamiento de redes para prevención de ataques, entre otros. También es interesante implementar una estrategia de gobierno de datos para asegurar la integridad, seguridad y disponibilidad de los datos.
- **Fiabilidad.** Se refiere a la capacidad de un entorno o sistema para recuperarse ante la caída de un servicio o infraestructura, adquirir recursos TI dinámicamente en función de la demanda o mitigar problemas de red, entre otros. Como solución eficaz ante situaciones de riesgo destaca el alojamiento tanto del código como de los datos en un sistema con capacidad para almacenar un histórico de cambios. Es decir, el sistema debe ser lo suficientemente autónomo e inteligente para, ante una situación de caída por un nuevo cambio, construir la aplicación con una versión anterior a la actual para asegurar el funcionamiento del servicio a los usuarios.
- **Eficiencia del rendimiento.** Esto está estrechamente ligado al uso eficiente de los recursos computacionales. También al hecho de mantener esa eficiencia según lo requiera la demanda y la tecnología evolucione. Para ello es necesario conocer fundamentalmente los requisitos de nuestra aplicación para que funcione en el mejor y peor de los casos. En otras palabras, conocer los

requisitos de rendimiento de latencia y ancho de banda de la red de los modelos de IA y usar, en consecuencia, los recursos necesarios como instancias basadas en GPUs, colas de mensajes, unidades aritmético-lógicas, etc. Puede darse el caso en el que sea necesario desplegar los modelos de IA cerca de los clientes para mejorar el rendimiento o cumplir con la normativa sobre privacidad de datos. Ante esta situación, puede ser interesante hacer uso de la “computación en el borde” o *edge computing* (en inglés) para analizar y procesar los datos desde el punto en el que se crearon.

- **Optimización de costes.** Este es uno de los retos más complejos cuando se aborda una aplicación en general. Siempre existe un grado de incertidumbre cuando se despliega un sistema en la nube dado que es difícil estimar el número de clientes que potencialmente pueden consumir el producto. Atendiendo a este asunto, existen diferentes alternativas en la actualidad que tratan de hacer frente a este problema. En primer lugar, prácticamente todos los proveedores que ofrecen servicios en la nube adoptan el enfoque de pago por uso, donde la capacidad de las instancias, el tráfico o su consumo determinan el precio a pagar. En este caso, una mala configuración de la infraestructura puede derivar en sobrecostes. Otra opción a considerar es el uso de instancias de acceso puntual. Abarcan cargas de trabajo flexibles y tolerantes a fallos que no son sensibles al tiempo; adecuadas para análisis de datos, entrenamiento de modelos o tareas opcionales, entre otros. Por último, algunos proveedores incluyen planes de ahorros cuya ventaja respecto a las anteriores es reducir los costes a cambio de comprometerse a un uso específico durante un periodo de tiempo. Como resultado, y siempre que sea posible, la clave para abaratar costes es hacer uso de los diferentes planes en función de las necesidades del productor a implantar.



IA con MLOPS. Este principio resulta útil cuando se desarrolla modelos de IA personalizados, siguiendo incrementalmente las fases del aprendizaje automático. Sin embargo, este capítulo deja a un lado esta parte y se centra en evaluar plataformas de IA desde la perspectiva de servicios inteligentes previamente entrenados listos para su uso. Se recomienda al lector que revise la asignatura de *Sistemas de Aprendizaje Automático* y reflexione sobre cómo abordar el flujo de trabajo de una aplicación de IA usando el enfoque MLOPS.

Ecosistemas finales para el desarrollo de aplicaciones de IA

En este apartado se describe un conjunto de ecosistemas en la nube con servicios y herramientas que soportan, por un lado, las etapas del desarrollo de aplicaciones de IA y, por otro, servicios inteligentes listos para su uso. Es importante resaltar que el objetivo no reside en ofrecer un listado exhaustivo, sino una visión general de los ecosistemas más relevantes en el mercado que les sirvan al lector como punto de partida. Entre los más destacados se encuentran:

- **Amazon AI⁵**. Entorno propiedad de Amazon, la compañía estadounidense de comercio electrónico y servicios de computación en la nube. Ofrece un catálogo extenso de soluciones completas a disposición de cualquier desarrollador, científico de datos y profesional experto. Según la prestigiosa consultora Gartner⁶, Amazon, o el servicio en la nube AWS, se considera actualmente el proveedor líder de la nube. AWS oferta servicios de IA que, a golpe de clics, se integran en aplicaciones de terceros para ocuparse de casos comunes, como recomendación de productos, reconocimiento de imágenes, conversión de voz a texto, conversión de texto a voz, traducción, procesamiento del lenguaje natural, etc. Además, soporta los entornos de trabajo más populares para la creación de modelos de aprendizaje automático. Por otro lado, integra herramientas para guiar, gestionar y acelerar el ciclo de vida de modelos o aplicaciones de IA.
- **Azure AI⁷**. Plataforma propietaria de la compañía Microsoft. Según el informe anterior, Azure se encuentra en la segunda posición como proveedor de soluciones en la nube. Azure AI mantiene una cartera de servicios inteligentes muy similar a la de Amazon AI. Entre ellas destacan los servicios de aprendizaje automático, minería de datos, aplicaciones basadas en inteligencia artificial y agentes virtuales. También soporta la mayoría de entornos para el desarrollo de modelos y contiene soluciones para MLOPS.
- **Google Cloud AI⁸**. Ofrece una variedad de soluciones potentes para hacer realidad proyectos de IA. Según Gartner, Google Cloud se encuentra en la tercera posición como proveedor de servicios en la nube. Google Cloud AI ofrece servicios para el desarrollo de aplicaciones de IA, modelos de IA para ayudar a agentes humanos, IA con funciones de búsqueda de empleo, etc. Al igual que sus competidores, cuenta con los entornos de trabajo más usados para el desarrollo de aplicaciones de IA y mantiene herramientas para MLOPS.
- **IBM Watson⁹**. Es la plataforma de IA de la compañía IBM que ofrece servicios inteligentes. De acuerdo al informe de Gartner, IBM Watson se encuentra en la cuarta posición como proveedor de servicios en la nube. Al igual que sus competidores, esta plataforma en la nube mantiene un catálogo extenso de servicios para el desarrollo de IA, como procesamiento del lenguaje natural, conversión de voz a texto, conversión de texto a voz, reconocimiento de imágenes, etc. Cuenta con un soporte menor de entornos de trabajo al de la competencia, pero ofrece integraciones con los más populares. También contiene herramientas para MLOPS.

⁵ Enlace: <https://aws.amazon.com/es/machine-learning/>

⁶ Enlace: <https://www.gartner.com/reviews/market/public-cloud-iaas>

⁷ Enlace: <https://azure.microsoft.com/es-es/overview/ai-platform/>

⁸ Enlace: <https://cloud.google.com/products/ai>

⁹ Enlace: <https://www.ibm.com/es-es/watson>

2.1.5. Oportunidades de las plataformas de IA

Desde hace no muchos años, la IA está tomando peso en la sociedad. Cada vez son más las aplicaciones que hacen uso de esta tecnología para resolver problemas en la sociedad gracias a la cantidad ingente de datos de los que hoy en día disponemos. De hecho, son muchas las organizaciones que apuestan por soluciones inteligentes con el fin de satisfacer las necesidades de los clientes y predecir la demanda. Incluso la batalla de países por coronarse como primera potencia mundial pasa por liderar el desarrollo tecnológico en IA.



175 zettabytes. IDC estima que crezca el volumen de datos producidos en el mundo a 175 zettabytes en 2025 (un zettabyte es un billón de gigabytes).

Las compañías tecnológicas más punteras en la actualidad están apostando fuertemente por soluciones que aceleren el desarrollo de aplicaciones y mejorar así la industria. En definitiva, liderar el crecimiento económico. Estas soluciones son plataformas compuestas de herramientas y APIs para el desarrollo de aplicaciones basadas en IA.

Desde 2019, según IDC¹⁰, el mercado de las plataformas de software de IA ha crecido exponencialmente hasta superar los 3.5 millones de dólares, es decir, tres veces el producto interior bruto de España. Y se espera que siga creciendo, dado que las organizaciones, cada vez más, necesitan software basado en recomendaciones, asistentes digitales y conversacionales, predicciones y automatizar procesos de negocio, entre otros. Además, es evidente que estas soluciones ayudan a las organizaciones a evolucionar rápidamente por su facilidad de pasar de un producto en fase experimental a producción.

Desde el punto de vista de las **ofertas de trabajo** se espera una alta demanda de personal cualificado para responder a las soluciones que exige la industria. Prueba de ello es el nuevo rol de MLOPS que muchas empresas necesitan cubrir para automatizar y orquestar el ciclo de vida de una aplicación de IA.

Con respecto a la **competitividad de las empresas** destacan herramientas que no requieren de código para implementar soluciones basadas en traducción automática, procesamiento y clasificación de imágenes, o reconocimiento del habla, entre otras funciones. Estas herramientas ofrecen la posibilidad de simplemente conectarse a aplicaciones existentes para proporcionar capacidades de IA sin la necesidad de entrenar modelos o desarrollarlos; especialmente útil para pequeñas empresas sin un equipo dedicado a estas tareas.

¹⁰🔗

Enlace: <https://www.ibm.com/downloads/cas/RGN4EOZK>

En esta misma línea entra en juego la **democratización de la IA**, poniéndola a disposición de un grupo cada vez más amplio de desarrolladores y empresas. Esto reduce la barrera que hasta día de hoy existía, brindando a cualquier individuo u organización a experimentar con IA al beneficiarse de algoritmos que permiten construir modelos en infraestructuras cloud. Esto significa que no es necesario realizar grandes inversiones para adentrarse en el mundo de la IA, sino tan solo acceso a internet.

En cuanto a **tendencias** destacan los frameworks *open source* basados en IA con un catálogo muy variado de herramientas gratuitas y con opción a modificar si se requiere. Estas herramientas son utilizadas por una gran mayoría de organizaciones para desarrollar sus propios modelos de aprendizaje automático. Sin embargo, estos frameworks a menudo necesitan de capacidades adicionales por lo que muchas plataformas de IA están comenzando a ofrecer soporte como parte de su opción tecnológica. En este sentido, los proveedores que consigan combinar herramientas propietarias con tecnologías open source a un precio razonable serán los ganadores de este mercado a largo plazo.

2.1.6. Resumen

En este apartado se remarcán los aspectos clave abordados en este bloque titulado Introducción a las Plataformas de IA:

- La **computación en la nube** ha sido un catalizador eficaz para servir IA. En los últimos años los entornos cloud han sufrido una evolución sustancial y con ellos hoy en día es posible crear soluciones inteligentes sin invertir grandes cantidades de dinero.
- Las soluciones de IA basadas en la nube ofrecen una serie de **beneficios** al delegar los recursos computacionales en un proveedor de TI. Entre ellos, destaca *agilidad* por disponer de recursos a medida que se necesitan; *ahorro de costes* al no hacer frente a grandes inversiones en infraestructuras; *elasticidad* por el escalado automático de recursos; *innovación* por ofrecer mayor rapidez para experimentar con algoritmos, frameworks y hardware; *despliegue* de aplicaciones en minutos; y *transición* fluida de productos en fase experimental a producción.
- Las **plataformas de IA** facilitan enormemente el desarrollo de aplicaciones basadas en IA. Destacan por ofrecer soluciones integradas de alta calidad para satisfacer las necesidades de los desarrolladores y gestionar el ciclo de vida de los desarrollos. Además, se basan en la idea de mitigar la **deuda técnica**. Las más populares se encuentran bajo las grandes compañías tecnológicas.

- La **previsión de oportunidades** de las plataformas de IA es optimista. Por un lado, se espera la creación de nuevos puestos de trabajo para hacer frente a la demanda de nuevas soluciones que mejoren la productividad. También se espera que estas plataformas mejoren la competitividad de las empresas y se democratice la IA poniéndola a disposición de cualquier individuo o equipo. Por otro lado, las soluciones de código abierto basadas en IA pueden resultar ser la pieza clave para liderar el mercado de las plataformas de IA.

2.2. Detalles de las Plataformas de IA

Hoy en día la red cuenta con infinidad de ejemplos, tutoriales, libros, en resumen, una gran cantidad de recursos para aprender cualquier temática que nos sea de interés. El campo de la IA no es una cuestión menor. De hecho, si realizamos una búsqueda en alguno de los motores más populares posiblemente encontraremos un número alto de resultados, siendo la mayoría accesibles de manera gratuita para todo el mundo. En el caso del tema que nos concierne es cierto que no existe una gran cantidad de plataformas en la nube para el desarrollo de IA dado que, mayoritariamente, son ofertadas por las grandes compañías tecnológicas. En este sentido es importante conocer las características de estas plataformas y realizar una evaluación que nos permita elegir en consecuencia, ya que este proceso puede llegar a ser verdaderamente complejo. Es por ello que esta sección, desde un punto de vista general, pretende abordar esta problemática, proporcionando al lector una serie de aspectos que caracterizan estas plataformas. Además, se añade una comparativa de las más populares y se proporcionan una serie de recomendaciones a tener en cuenta a la hora de su elección.

2.2.1. Visión general

El término plataforma en informática es un concepto que tiende a ser, en general, abstracto e impreciso. La definición que propone la RAE es una prueba de ello, donde plataforma, en el campo de la informática, lo consideran como «un entorno informático determinado, que utiliza sistemas compatibles entre sí». En el contexto en el que se plantea una plataforma de IA en este capítulo, la definición que propone la RAE se entiende como un **sistema base que actúa sobre un hardware y software** para ofrecer servicios y herramientas. Ejemplos a destacar son Amazon AI, Google Cloud AI, IBM Watson o Azure AI, entre muchos otros.

Entre los aspectos clave bajo la filosofía de estas plataformas de IA, desde el punto de vista del desarrollador, resaltan la flexibilidad y los costes. Por un lado, una plataforma de IA permite satisfacer las necesidades de un equipo de desarrollo para una compañía grande y, al mismo tiempo, para un desarrollador individual. En línea con lo anterior, cualquier individuo, a un coste reducido, tiene a su disposición algoritmos o modelos con alto grado de sofisticación que de otro modo sería impensable. Estos hechos hacen que las plataformas de IA destaque por conseguir un **compromiso tanto en accesibilidad como en facilidad de uso**.

Por otra parte, las plataformas de IA también destacan por su excelente **soporte y documentación**. Por un lado, las plataformas más populares soportan una gran diversidad de frameworks, bibliotecas, lenguajes, etc. El objetivo es que los desarrolladores cuenten con un catálogo extenso de opciones que les permita desarrollar soluciones con las tecnologías que más cómodos se sienten o aquellas que son requisito indispensable para un desarrollo específico. En relación a la documentación, las propias plataformas cuentan con una gran cantidad de manuales, tutoriales y ejemplos disponibles de forma gratuita en Internet. También existen cursos y libros, los cuales pueden encontrarse de libre acceso o de pago. Además, la comunidad de usuarios que hacen uso de estas plataformas ha crecido exponencialmente, lo cual ha facilitado especialmente la rápida búsqueda de respuestas ante problemas comunes que pueden encontrarse durante la creación de una aplicación.

Antes de entrar en detalle sobre las cuestiones clave que caracterizan a una plataforma de IA en la nube es importante conocer los conceptos fundamentales para poder así beneficiarnos de estos recursos tecnológicos de tendencia. Estos se resumen a continuación.

2.2.2. Fundamentos básicos

Como bien se ha visto en la parte introductoria de este capítulo, las plataformas de IA son entornos que están constituidos bajo un entorno cloud. Esto quiere decir que existe una infraestructura hardware, invisible para el usuario, que es la encargada de ejecutar el software bajo los servicios publicados. Desde la perspectiva de la IA, el software podría entenderse como algoritmos, modelos o simplemente servicios ya preentrenados para casos de uso específicos.

Estas infraestructuras son gestionadas por proveedores que son responsables de manejar los recursos de TI, integrar aplicaciones, o desarrollar nuevas capacidades así como funcionalidades.

Esta gestión de los recursos deriva en diferentes **modelos de servicios** que los proveedores suministran a los clientes para satisfacer sus necesidades. En este apartado se detallan los modelos más ampliamente conocidos, ofreciendo al lector una descripción de cada uno de ellos en el contexto de la IA.

- **Infraestructura como servicio.** Término conocido en inglés como *Infrastructure as a service (IaaS)*. Ofrece recursos fundamentales como computación, almacenamiento y capacidades de red. Ayuda a los profesionales de IA a disponer de un entorno completo y complejo (CPU, memoria, red, discos, sistema operativo, etc.) de forma sencilla, evitando esperas por parte del equipo de infraestructura a que prepare el entorno.

- **Plataforma como servicio.** Término conocido en inglés como *Platform as a service (PaaS)*. En general, proporciona plataformas para desplegar aplicaciones personalizadas en la nube. Desde la perspectiva de la IA, ofrece a profesionales un conjunto de herramientas y servicios con los que construir, desplegar y mantener aplicaciones (p. ej. suite de herramientas para el desarrollo de aplicaciones de *machine learning*). El objetivo es facilitar el desarrollar de aplicaciones de nueva generación.
- **Software como servicio.** Término conocido en inglés como *Software as a service (SaaS)*. Combina la infraestructura y el software que se ejecuta en la nube. Ayuda a profesionales de IA a consumir servicios dentro de una aplicación, por ejemplo, agentes conversacionales para mejorar el servicio de un cliente.

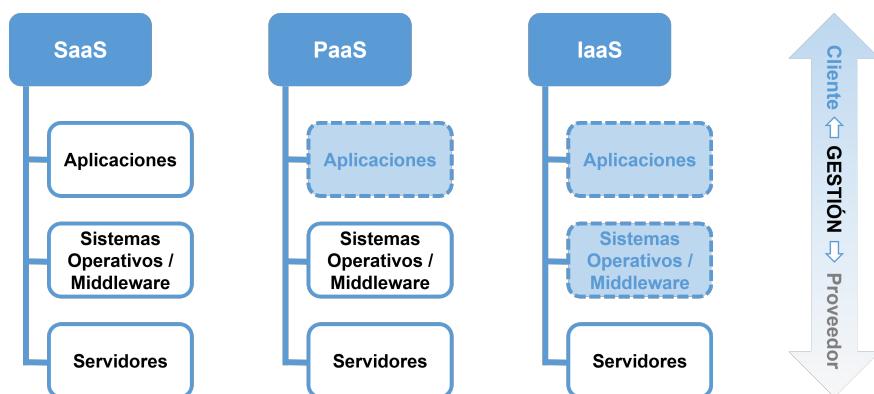


Figura 2.4: Nivel de gestión por parte del cliente o proveedor entre los diferentes modelos de servicio.



La magia de los contenedores. Una tecnología complementaria a las soluciones de los proveedores de la nube es lo que se conoce como contenedor. La esencia de este concepto reside en empaquetar software y desplegarlo en cualquier servidor. A modo de referencia, se recomienda al lector el estudio de las tecnologías *Docker* y *Kubernetes*, las cuales hacen posible, entre su abanico de posibilidades, la creación de entornos de IA personalizados.

Esta clasificación de modelos de servicios de computación en la nube, a su vez, se agrupan en tres categorías superiores. Se diferencian por la cantidad de gestión que requieren del cliente, así como el nivel de seguridad suministrado.

- **Nube pública.** La infraestructura es operada por una organización que ofrece servicios al público en general.
- **Nube privada.** Los servicios son ofrecidos a un grupo de usuarios y no al público general. La infraestructura es íntegramente gestionada por una organización.

- **Nube híbrida.** Es la combinación de dos o más nubes individuales que pueden ser públicas o privadas.

Uno de los libros de referencia sobre computación en la nube es: *Cloud Computing: Concepts, Technology & Architecture* [EPM13]. Se anima al lector a revisar los conceptos, modelos y mecanismos fundamentales de este paradigma, y a profundizar sobre la tecnología de contenedores, así como otras también de interés que puedan ser de utilidad para proyectos de IA.

2.2.3. Aspectos fundamentales de las plataformas de IA

Interacción con la plataforma

En el campo del desarrollo de software es muy habitual contar con herramientas que ofrece, generalmente, un fabricante, ya sea de hardware, de sistemas operativos o de lenguajes de programación, para facilitar a desarrolladores la creación de aplicaciones. Algo muy similar a esto serían los kits o cajas de herramientas que usan los profesionales de toda la vida, por ejemplo, un fontanero, para desempeñar su labor de manera más ágil y efectiva. En el contexto del desarrollo de aplicaciones, esto se conoce como **kits de desarrollo software** o **SDK (Software Development Kit)**, en inglés.



Destacando del resto. No todos los fabricantes utilizan el nombre SDK para referirse a un kit de desarrollo software. Por ejemplo, la compañía Oracle llama JDK (Java Development Kit) a su paquete para el lenguaje de programación Java.

En líneas generales, las plataformas de IA basadas en la nube cuentan con un SDK para proporcionar a los desarrolladores diferentes opciones con las que interactuar con la herramienta y, en definitiva, crear sus aplicaciones. Por lo general, los SDKs de las plataformas de IA suelen ser multiplataforma, es decir, que las herramientas software funcionan en cualquier arquitectura, por ejemplo, Windows, macOS o Linux. Además, estos SDKs, en la mayoría de casos, incluyen un compilador, un depurador, herramientas de línea de comandos o CLI (Command-Line Interface), una API, documentación y bibliotecas, aunque también pueden contar con entornos de desarrollo y controladores, entre otros. A la vista está que incluyen todos los elementos necesarios para crear aplicaciones nuevas para un producto y ecosistema específico.

La premisa de estos SDKs es facilitar al usuario la interacción con la plataforma, ocultar los detalles de implementación y, en definitiva, la manera en la que consumir los servicios que ofrecen estas plataformas basadas en la nube. Mayoritariamente, todas ellas comparten los mecanismos más comunes para interactuar con sus servicios. Entre las herramientas disponibles destacan las **consolas web**, las **herramientas de línea de comandos** y las **bibliotecas para el cliente**.

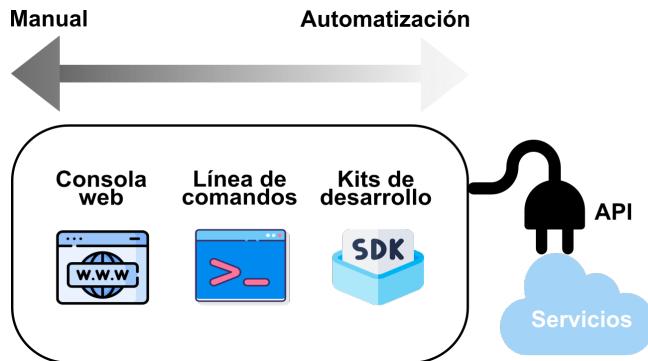


Figura 2.5: Interacción entre componentes software y servicios de IA.

La primera opción consiste en una consola web para interactuar con la plataforma de IA. Básicamente es una interfaz gráfica de usuario o GUI (Graphical User Interface), en inglés, que puede correr prácticamente en cualquier navegador web moderno, por ejemplo, Chrome, Firefox o Safari. Aunque bien es cierto que esta herramienta está fuera de lo que se considera un SDK, dado que es un entorno web, suelen estar presente en la mayoría de plataformas por su alta utilidad. Esta alternativa suele ser interesante para empezar a experimentar con la plataforma, dado que ayuda a obtener una visión general de los diferentes servicios y a conseguir resultados rápidamente. La ventaja de esta herramienta reside en el hecho de que, a golpe de clics de ratón, es muy sencillo gestionar los recursos para levantar un servicio, y viceversa. Sin embargo, este entorno no está preparado para describir el comportamiento de la aplicación.

La segunda opción suele estar enfocada a un grupo de usuarios con un conocimiento más profundo de los servicios y, en definitiva, de la plataforma. Las herramientas de línea de comandos o CLI, en inglés, permiten controlar cualquier aspecto de la plataforma basada en la nube, al igual que también lo hacen las consolas web, pero con la diferencia que todo acceso es mediante comandos desde un terminal. Una de las ventajas de esta herramienta es la rapidez para probar servicios y realizar pruebas de concepto. Es decir, no es necesario desarrollar código para probar una funcionalidad concreta de la plataforma, sino que utilizando un par de comandos es posible comprobar el comportamiento del servicio y decidir si es lo adecuado para un desarrollo. Por el contrario, esta herramienta no permite, de un modo sencillo, crear un comportamiento personalizado como sí podría hacerse mediante el uso de un lenguaje de programación. A modo de ejemplo, considere la necesidad de probar un servicio que sintetice un fragmento de texto a voz. Por cuestiones prácticas, se ha utilizado AWS, concretamente el servicio Amazon Polly¹¹, por catalogarse como uno de los proveedores de IA en la nube de referencia.

¹¹ Enlace: <https://aws.amazon.com/es/polly/>

```
$ aws polly synthesize-speech  
--output-format mp3  
--voice-id Lucia  
--text 'Cualquier tecnología suficientemente avanzada es indistinguible  
de la magia' lucia-ia.mp3
```

Por último, la tercera opción está relacionada con bibliotecas para la parte del cliente. En este caso, el foco está puesto en desarrolladores con cierto bagaje por considerarse una parte mucho más técnica que las anteriores. Estas bibliotecas proporcionan una experiencia optimizada a los desarrolladores con las convenciones y estilos de cada lenguaje compatible. A continuación, se enumeran los lenguajes comunes admitidos por la mayoría de plataformas de IA.

- | | | |
|--------------|----------|--------------|
| ■ Go | ■ Python | ■ C# |
| ■ Java | ■ Ruby | ■ Typescript |
| ■ JavaScript | ■ PHP | ■ C++ |

Por otro lado, las bibliotecas controlan todos los detalles a bajo nivel de comunicación con el servidor. En esencia, esta solución reduce al máximo el código a escribir, ya que están diseñadas teniendo en cuenta las particularidades de los servicios ofertados por la plataforma. El hecho de utilizar bibliotecas supone como ventaja conseguir aplicaciones con un comportamiento determinado, dado que el uso de código ofrece una mayor versatilidad frente a la opción anterior en cuanto a personalización de un servicio se refiere. Con el objetivo de reflejar la bondad de esta opción considere la modificación del ejemplo anterior para sintetizar texto a voz y enviar el resultado a un nodo de almacenamiento. Date cuenta que las claves de acceso a AWS en las líneas ⑧-9 no están puestas en el código por motivos de seguridad.

Listado 2.1: Ejemplo en Python que refleja las bondades del uso de bibliotecas

```
1 #!/usr/bin/python3  
2  
3 # Import the AWS SDK for Python language  
4 import boto3  
5  
6 # Create the service Polly and assign credentials  
7 polly_client = boto3.Session(  
8     aws_access_key_id='aws_access_key_id',  
9     aws_secret_access_key='aws_secret_access_key',  
10    region_name='eu-west-2').client('polly')  
11  
12 # Assign parameters and call the service  
13 try:  
14     response = polly_client.start_speech_synthesis_task(  
15         VoiceId='Lucia',  
16         OutputS3BucketName='my-bucket',  
17         OutputFormat='mp3',  
18         Text='Cualquier tecnología suficientemente avanzada es ' +
```

```
19     'indistinguible de la magia',  
20     TextType='text',  
21     SampleRate='22050')  
22  
23     print("Audio file is saved into {} ".format(response['SynthesisTask']['OutputUri']))  
24 except:  
25     print("Oops! An unexpected error was raised")
```

Como conclusión final es importante destacar que cada opción es útil para un caso determinado. Todas ellas cuentan con sus ventajas e inconvenientes. Por lo tanto, se recomienda al lector buscar un compromiso entre utilidad y adecuación. En otras palabras, el desarrollador debe ser consciente de las características de cada herramienta y aplicarlas en el contexto adecuado para lograr la mayor efectividad posible.



El saber no ocupa lugar. Se propone como ejercicio al lector modificar el fragmento de código anterior para, dado un artículo o noticia, convertirlo a voz y descargar el resultado como archivo .mp3.

Mecanismos de comunicación

En el apartado anterior se ha realizado un recorrido sobre las herramientas disponibles para la interacción con las plataformas de IA. Sin embargo, aun no se ha puesto en conocimiento los mecanismos que establecen la comunicación entre los desarrolladores y sus servicios. Veamos a continuación los ingredientes que hacen esto posible.

El mecanismo por excelencia que todos los proveedores cloud utilizan para que los desarrolladores consuman sus servicios es el uso de APIs bajo comunicaciones basadas en web. El procedimiento por referencia a la hora de escribir código es dirigir el desarrollo hacia una arquitectura REST (Representational State Transfer)¹² donde el desarrollador organiza su código en llamadas a APIs publicadas por el proveedor de servicios en la nube. A la petición se le pasa una serie de parámetros que varía según el servicio y un servidor retorna un resultado, por lo general, en formato JSON (JavaScript Object Notation), el cual se suele comprobar para confirmar la operación.



Computación sin servidor. Las llamadas a las APIs de IA también pueden realizarse siguiendo este modelo, el cual suele denominarse Funciones como Servicio o *Functions as a Service* (FaaS), en inglés. El proveedor de la nube asigna recursos dinámicamente para ejecutar código sin estado en el lado del servidor en base a la demanda.

¹²🔗

Enlace: https://es.wikipedia.org/wiki/Transferencia/_de/_Estado/_Representacional

Las APIs que soportan los proveedores de servicios en la nube suelen estar basadas en esquemas de autorización, como por ejemplo OAuth 2.0¹³, para garantizar la autenticación y la seguridad del usuario. En esencia, este mecanismo de autenticación permite o deniega las peticiones en función de la validación de un *token*, el cual no es más que una cadena de texto única que actúa como llave para acceder a los servicios del proveedor de la nube.

A modo de ejemplo, considere el listado 2.1 en Python donde se realiza una petición a Amazon Polly para sintetizar una cadena de texto a voz. En las líneas Ⓛ 7-10 se crea la sesión del cliente que representa *Amazon Polly*, con la biblioteca boto3, y se asignan los tokens que dan acceso al servicio. Por otro lado, en las líneas Ⓛ 14-21 se realiza una petición a un servidor donde se aloja un modelo preentrenado para la tarea en cuestión. Si por casual los *tokens* fueran incorrectos, la llamada no se realizaría y, en consecuencia, se obtendría un error de autenticación. Se observa, además, que la llamada recibe una serie de parámetros que contienen detalles de lo que el servicio debe realizar. Sólo como curiosidad, la llamada al método start_speech_synthesis_task realiza una llamada a la API y crea una tarea asíncrona de síntesis, lo cual significa que la llamada es bloqueante, es decir, que la ejecución del programa se bloquea hasta que no recibe un resultado o, alternativamente, se produce un error.

Cuando el cliente accede a la API lo hace a través de una abstracción de código, en este caso una biblioteca que, de manera transparente, se conecta a una URL (Uniform Resource Locator) o *endpoint* siguiendo una conexión HTTPS (Hypertext Transfer Protocol Secure)¹⁴. Por lo general, el tráfico viene predeterminado por este sistema de comunicación que añade una capa más de seguridad al protocolo HTTP (Hypertext Transfer Protocol)¹⁵ para intercambiar datos entre el receptor y el destinatario. Dado que las peticiones vía URL siguen el modelo REST, los métodos de petición HTTP¹⁶ se definen como parte de la especificación de la API. En otras palabras, el catálogo de servicios de la API debe cumplir con los requisitos asociados al método de petición del protocolo HTTP. Por ejemplo, si se desea recuperar una lista de voces que están disponibles para su uso cuando se solicita la síntesis de voz, la API debe haber diseñado este servicio con el método GET, el cual representa una acción de recuperación que no tiene ningún efecto secundario visible para el cliente.

En el contexto de desarrollo de aplicaciones mediante mecanismos de comunicación siempre es importante tener presente ciertas **recomendaciones o buenas prácticas** para obtener un rendimiento óptimo y la mejor experiencia posible. Dado que la naturaleza para consumir servicios de IA en la nube es a través de APIs, tal y como se ha visto en el listado 2.1, se recomienda, siempre que sea posible,

¹³ ⓘ Enlace: <https://oauth.net/2/>

¹⁴ ⓘ Enlace: https://es.wikipedia.org/wiki/Protocolo_seguro_de_transferencia_de_hipertexto

¹⁵ ⓘ Enlace: <https://developer.mozilla.org/es/docs/Web/HTTP/Overview>

¹⁶ ⓘ Enlace: <https://developer.mozilla.org/es/docs/Web/HTTP/Methods>

la **reutilización del objeto de la sesión del cliente** para subsecuentes peticiones. Esto debería primar frente a la creación de nuevas instancias. Ya no solo por el hecho de tener un código incoherente, sino por cuestiones más relacionadas con el rendimiento.

Listado 2.2: Ejemplo en Python de un mal uso de una API

```
1 # Previous code here
2
3 try:
4     # Previous code here
5
6     # Bad practice of using another instance of Polly object
7     polly_client = boto3.Session(
8         aws_access_key_id='aws_access_key_id',
9         aws_secret_access_key='aws_secret_access_key',
10        region_name='region').client('polly')
11
12     taskId = response['SynthesisTask']['TaskId']
13
14     task_status = polly_client.get_speech_synthesis_task(TaskId = taskId)
15
16     print("The status of the task '{}' is '{}' ".format(taskId, task_status['
17     SynthesisTask']['TaskStatus']))
17 except:
18     print("Oops! An unexpected error was raised")
```

En esencia, la petición inicial que realiza una instancia de la sesión del cliente ejecuta la autenticación, autorización y, como resultado, genera un *token* de acceso para mantener activa dicha sesión. Aunque el tiempo en llevarse a cabo esta tarea no es excesivamente alto, si puede tomar varios segundos, pues este proceso sucede en la red y, como consecuencia, es contraproducente para aplicaciones que requieren bajos tiempos de espera. Por lo tanto, esta recomendación habilita la reutilización del *token* de acceso, siempre que este sea válido. Por el contrario, en el caso de su expiración, una nueva llamada con el mismo objeto actualizaría automáticamente la clave, ocultando así los detalles de bajo nivel a los programadores de IA.

Por otro lado, una estrategia que es imprescindible adoptar cuando se utilizan este tipo de enfoques es **ocultar las claves de acceso** de los servicios de IA en el propio código fuente. Su fuga o copia podría causar diversos problemas de seguridad, puesto que una persona tiene la posibilidad de realizar cualquier acción en una cuenta donde estas credenciales están asociadas. Por ejemplo, un usuario podría manipular el comportamiento de un servicio, acceder a datos sensibles o incluso incluir fallos de seguridad en las aplicaciones, entre otras muchas opciones. En definitiva, tal y como entregar las llaves de tu casa a un desconocido. Como solución ante este problema, las **variables de entorno** proporcionan un mecanismo de seguridad para mantener ocultas estas claves de acceso y ofrecer una comunicación fiable con el servicio. A continuación se muestra al lector un ejemplo de cómo recuperar en Python variables de entorno creadas en sistemas Windows y Linux.

```
# Linux  
$ export ACCESS_KEY_ID="ACCESS_KEY_ID"  
  
# Windows  
C:\> SET ACCESS_KEY_ID=ACCESS_KEY_ID
```

Listado 2.3: Ejemplo en Python que refleja cómo acceder a una variable de entorno

```
1 #!/usr/bin/python3  
2  
3 import os  
4  
5 # Get the environment variable specified  
6 # Return 'KeyError' exception if the env variable doesn't exist  
7 print(os.environ['ACCESS_KEY_ID'])  
8  
9 # Return 'None' if the env variable doesn't exist  
10 # It doesn't throw the 'KeyError' exception  
11 print(os.environ.get('ACCESS_KEY_ID'))  
12  
13 # os.getenv is equivalent to the above function  
14 # It is possible to add a default value instead of 'None'  
15 print(os.getenv('ACCESS_KEY_ID', 'access_key_id'))
```

Por último, tampoco hay que olvidar seguir las **pautas generales del desarrollo de software**, pues esto beneficia tanto al desarrollador como a la aplicación. De hecho, organizar el código tiende a evitar, por ejemplo, duplicidad en las llamadas a la API, lo cual puede perjudicar considerablemente el rendimiento de la aplicación. También, optar por un enfoque desacoplado brinda la posibilidad de mantener un código a prueba de cambio de versiones en las APIs, algo que sucede con bastante frecuencia. En definitiva, el desarrollador debe poner también de su parte para conseguir una solución adecuada que satisfaga las necesidades del usuario.

Opciones de terceros

En el contexto de desarrollo de aplicaciones mediante plataformas de IA suele ser muy común apoyarse sobre paquetes o entornos de referencia. Principalmente por el hecho de sacar un mayor partido a estas soluciones cloud y ofrecer la mayor flexibilidad posible a los desarrolladores.

Como se ha mencionado en anteriores ocasiones, estas plataformas de IA, al basarse en entornos en la nube, cuentan con multitud de opciones que hacen de ellas un **enfoque de todo integrado**. En otras palabras, la construcción de una solución puede realizarse perfectamente con las herramientas que proporciona la propia plataforma. De hecho, existen soluciones ya integradas, exclusivas del proveedor o externas, que habilitan la creación de modelos de IA personalizados, tal y como se discutió en la parte introductoria de este capítulo. Ese flujo de trabajo puede seguir-

se completamente paso a paso con este tipo de entornos sin necesidad de hacer uso de otras herramientas. Este enfoque, no obstante, se encuentra fuera del alcance de este capítulo. Se invita al lector a profundizar en este tema siguiendo otra de las asignaturas de este curso como ya se mencionó anteriormente.

Por otro lado, el comportamiento inteligente, sin lugar a dudas, también puede venir determinado por la plataforma, principalmente por habilitar una funcionalidad compleja fácil de consumir a través de su exposición en un API. Sin embargo, el envoltorio que rodea este comportamiento no tiene por qué estar implementado con las soluciones del proveedor cloud, sino que puede usarse cualquier otra opción que la tecnología lo permita.

Atendiendo estas cuestiones, considere como ejemplo una mejora al listado 2.1 para ejecutar la llamada a la API *Polly* una vez un servidor recibe una petición. Para más información sobre la API, se recomienda al lector consultar el sitio web:  Enlace: boto3.amazonaws.com/v1/documentation/api/latest/index.html. Obsérvese que, en este caso, la lógica donde reside la llamada al servicio de síntesis de voz se encuentra dentro del código que rodea la implementación de un servidor. No obstante, antes de mostrar su implementación, véase primero el listado 2.4, el cual encapsula en una función (líneas  22-35) el comportamiento encargado de realizar un llamada a la API de IA. Su diseño se debe a mantener el comportamiento lo más desacoplado posible, aunque en este ejemplo no sea realmente necesario por su simplicidad. Sin embargo, se ha optado por esta opción para mostrar al lector un ejemplo de solución cuidada desde el punto de vista del diseño de software. Nótese que dicha función recibe un argumento que es el cuerpo del mensaje que se envía durante la petición al servidor. A modo de curiosidad, véase que las variables de entorno son cargadas usando la función `load_dotenv` (línea  8) que lee de un archivo `.env`. Para más información consulte el siguiente sitio web:  Enlace: pypi.org/project/python-dotenv/.

Listado 2.4: Encapsulación de la llamada al servicio *Polly* mediante una función en Python

```
1 #!/usr/bin/python3
2
3 # Import packages
4 import boto3, os
5 from dotenv import load_dotenv
6
7 # Take environment variables from .env file
8 load_dotenv()
9
10 # Get env variables
11 accessKeyId = os.environ.get('ACCESS_KEY_ID')
12 secretKey = os.environ.get('ACCESS_SECRET_KEY')
13 region = os.environ.get('REGION')
14 bucketName = os.environ.get('BUCKET_NAME')
15
16 # Create the service Polly and assign credentials
17 polly_client = boto3.Session(
18     aws_access_key_id=accessKeyId,
19     aws_secret_access_key=secretKey,
20     region_name=region).client('polly')
```

```

21
22 def speechSynthesis(request):
23     # Assign parameters and call the service
24     try:
25         response = polly_client.start_speech_synthesis_task(
26             VoiceId=request['voiceId'],
27             OutputS3BucketName=bucketName,
28             OutputFormat='mp3',
29             Text=request['text'],
30             TextType='text',
31             SampleRate='22050')
32
33     print("Audio file is saved into {} ".format(response['SynthesisTask']['OutputUri']
34     )))
34 except:
35     raise Exception("Oops! An unexpected error was raised")

```

Por otro lado, el listado 2.5 muestra un ejemplo básico de la implementación de un servidor escrito en Python con el framework *Flask*¹⁷. Con respecto a las partes más importantes a considerar, la línea ⑧ crea una instancia de la aplicación y le asigna un nombre en base al módulo donde esta se encuentra; en este caso simplemente “application”. Después, las líneas ⑪-⑯ definen el comportamiento cuando un usuario realiza una llamada POST a la ruta o *endpoint* ‘api/speech-synthesis’. Nótese que línea ⑭ es una llamada a una función aislada en una carpeta con nombre ‘api’ y cuyo archivo es *speech_synthesis*. El nombre de la función se ha cambiado a *textToSpeech* para no interferir con la función *speechSynthesis*. Como puede observarse, esta función, definida en el listado 2.4, se le pasa como argumento *request.get_json()*, el cual contiene la información enviada en la petición. El cuerpo del mensaje se envía en formato JSON con las siguientes propiedades: *voiceId* y *text*. El primero mantiene un identificador de la voz a usar durante la síntesis y el segundo se corresponde con la entrada de texto a sintetizar. Por cuestiones de simplicidad, los aspectos relacionados con cuestiones de seguridad han sido obviados. No obstante, en un desarrollo real es más que evidente que esto debe de ser abordado desde las primeras fases del desarrollo.

Listado 2.5: Implementación del servidor usando Flask

```

1 #!/usr/bin/python3
2
3 # Import packages
4 from flask import Flask, request, abort, Response
5 from api.speech_synthesis import speechSynthesis as textToSpeech
6
7 # Create Flask application
8 application = Flask(__name__)
9
10 # api/speech-synthesis endpoint
11 @application.route('/api/speech-synthesis', methods=['POST'])
12 def speechSynthesis():
13     try:
14         textToSpeech(request.get_json())

```

¹⁷🔗 Enlace: <https://flask.palletsprojects.com/en/2.0.x/>

```
15     except Exception as error:
16         print(error)
17         abort(500)
18     return Response(status=200)
19
20 # Run the app
21 if __name__ == "__main__":
22     application.debug = True
23     application.run() # Running on http://127.0.0.1:5000/
```



Más allá de AWS. ¿Cómo se realizaría este mismo ejemplo sobre otra plataforma de IA? ¿Crees que el desarrollo variaría considerablemente? Se invita al lector a reflexionar sobre estas dos cuestiones.

Como conclusión final, esta sección ha tratado, mediante un ejemplo, mostrar al lector la facilidad de aunar diferentes tecnologías. La plataforma de IA, por un lado, ofrece el comportamiento inteligente, mientras que las opciones de terceros brindan la posibilidad de encapsular esa funcionalidad en un contexto real. Se invita al lector a construir este mismo ejemplo con otro lenguaje de programación, y a reflexionar si la complejidad reside en la tecnología utilizada, en lugar del diseño y desarrollo del comportamiento inteligente.

2.2.4. Comparativa

Como pudimos ver en secciones anteriores, la evolución del hardware y los nuevos retos tecnológicos han dado lugar a la creación de soluciones basadas en la nube de una calidad sustancial, como son las que ofrecen Google, Amazon, IBM o Microsoft, entre otras. Aunque todas ellas pueden llegar a ofrecer servicios muy similares entre sí, existen algunas diferencias que reflejan las ventajas de usar uno u otro proveedor. Esta sección trata de poner en relieve aquellas consideraciones generales más importantes que diferencian a los proveedores de IA en la nube. Estas se resumen a continuación.

- **Localizaciones.** Con esto nos referimos al número de CPDs repartidos por todo el mundo que sostienen las infraestructuras de los proveedores cloud. Aquí la cuestión no reside tanto en el número, sino en dónde se encuentran localizados estos CPDs que contendrán la información de nuestras aplicaciones. Anteriormente se ha discutido que la zona física geográfica donde se instala una aplicación influye en las restricciones a adoptar en materia de privacidad de datos.
- **SDKs.** En concreto, los lenguajes de programación bajo las bibliotecas que proporcionan las plataformas de IA suponen un aspecto totalmente diferenciador. No es que una solución solo pueda programarse con único lenguaje, sino que sus características pueden acelerar el desarrollo frente a un problema concreto.

- **Herramientas y/o servicios de IA.** En el contexto del desarrollo de aplicaciones de IA este aspecto supone un factor clave en cuanto a diferencias entre una plataforma u otra. Particularmente por la calidad, facilidad de uso, integración, etc.
- **Precio.** Al igual que con cualquier otro producto, el precio es otro de los factores que también diferencian a estas soluciones. Es importante resaltar que una solución más barata que otra no supone que sea peor, y viceversa.
- **Documentación.** Siempre se valora positivamente que las soluciones estén soportadas por una documentación accesible, exhaustiva, explicativa y, sobre todo, útil. En definitiva, una guía que ayude a usuarios a utilizar, en este caso, las plataformas de IA proporcionadas por los proveedores de la nube.

2.2.5. Elección de la plataforma

Elegir una plataforma de IA no es para nada una tarea sencilla, principalmente por el abanico de opciones que ofrecen para resolver un mismo problema. Un ejemplo que podría reflejar esta situación sería el momento en el que una persona decide suscribirse a una plataforma de *streaming*. Si no partimos con una idea del contenido que ofrece cada una de ellas es posible que nos sintamos abrumados a la hora de la elección, dado que todas mantienen un catálogo lo bastante extenso como para perderse en él. Algo similar ocurre cuando se nos plantea la opción de elegir una plataforma de IA.

En primera instancia es necesario plantearse una serie de preguntas antes de comenzar a buscar por la red y encontrar potenciales soluciones. Tomando como referencia las consideraciones de la sección anterior, la formulación de las preguntas sería la siguiente:

- ¿El desarrollo a realizar involucra utilizar técnicas de IA? Si la respuesta es sí, continuemos hacia la siguiente pregunta.
- ¿Qué servicios de IA necesita el desarrollo?
- ¿Cuál es el presupuesto disponible para el desarrollo de la solución?
- ¿Debemos usar una localización concreta?
- ¿Partimos de la premisa de utilizar un lenguaje de programación concreto?

Las respuestas a las preguntas anteriores orientan, de algún modo, la elección de la plataforma de IA. Esto permite obtener un conjunto de alternativas que, posteriormente, es necesario valorar. Como punto inicial puede ser interesante, dado que ayuda a descartar opciones o incluso a valorar otras que antes podrían pasar desapercibidas. Sin embargo, este planteamiento no finaliza aquí, sino que es imprescindible profundizar en cada una de ellas estableciendo un orden de criterios. A continuación se proponen, según orden de importancia, 5 criterios tomando de nuevo como referencia las consideraciones de la sección anterior:

- Solución de IA
- Precio de la solución
- Regiones
- SDKs
- Documentación

En primer lugar, el criterio más importante, sin lugar a dudas, son las **herramientas y/o servicios de IA**. Resulta obvio que si un proveedor de la nube no ofrece una solución para el problema a resolver inmediatamente se descarte del conjunto de alternativas. Esta opción se considera la más importante principalmente porque es el enfoque con el que se resolverá el problema planteado. Adicionalmente, otras cuestiones que suelen valorarse, y que tienen un peso considerable en la elección, están relacionadas con la calidad del servicio, características, etc.

El **precio** también es una cuestión muy importante a considerar. Por lo general, los servicios de IA que ofrecen los proveedores de la nube no suelen ser excesivamente costosos. De hecho, como ya vimos en la parte introductoria de este capítulo, una de las ventajas de estas soluciones es el ahorro de costes al no tener que hacer frente a inversiones en infraestructuras. Sin embargo, si es cierto que el coste del servicio, ante todo, debe de cuadrar dentro el presupuesto del desarrollo.

En tercera posición, la **región o zona de la infraestructura cloud** donde se agrupan los CPDs supone tres aspectos a considerar. El primero, y más importante, es que el servicio elegido esté disponible en la localización donde se realice el despliegue. En línea con lo anterior, la zona física de despliegue es importante en términos de latencia, dado que es recomendable que esté lo más cerca del cliente objetivo para reducir los tiempos de espera al consumir un servicio. El tercero, por otro lado, se enmarca en cuestiones de privacidad, dado que la zona física geográfica elegida implica implementar una política de privacidad asociada a dicha región.

En cuarto lugar, si bien es cierto que desarrollar una solución mediante un **lenguaje de programación** no debería de suponer un impedimento, el lenguaje ayuda, sin lugar a dudas, a facilitar cuestiones de programación. Por ejemplo, Python es un lenguaje muy adecuado para tratar información de grandes volúmenes de datos, mientras que JavaScript es muy útil para gestionar la comunicación entre servidor y cliente. En resumen, el lenguaje no es la piedra angular para resolver el problema, pero sí el medio con el que ser más efectivo a la hora de construir nuestro código.

Por último, y por razones obvias, la **documentación** es la cuestión menos importante a tener en cuenta. Principalmente por el hecho de que esta opción no resuelve el problema, sino que sirve de ayuda a los profesionales a entender cómo utilizar la herramienta. Por lo general, todas las plataformas de IA basadas en la nube proporcionan una documentación rica y exhaustiva, tratando de reducir su línea de aprendizaje y permitiendo al profesional de IA centrarse en asuntos importantes, por ejemplo, diseño de la arquitectura, diseño de algoritmos, optimizaciones, pruebas de rendimiento, etc.

A modo de ejemplo, considere el siguiente escenario para reflejar cuál sería la elección de una plataforma de IA ante una situación real. Imaginemos que una Universidad española dispone de un sitio web donde aloja cursos online para sus alumnos. En líneas generales, la plataforma no está siendo explotada como debería, principalmente porque son pocos los alumnos que hacen uso de ella. Ante esta situación, surge la necesidad de realizar un desarrollo en IA que permita recomendar, en función de la rama de conocimiento de cada alumno, un curso con el que complementar sus estudios. En primer lugar, se realizaría un estudio exhaustivo de las plataformas de IA existentes, tomando como referencia servicios capaces de ofrecer recomendaciones personalizadas. Entre los más destacables se encuentra el servicio de Google, denominado *Recommendations AI*¹⁸; *Personalizer*¹⁹, de Azure; y *Amazon Personalize*²⁰. Después, teniendo en cuenta los precios de cada solución, en primera instancia, la elección a considerar sería Amazon Personalize, pues es el servicio más barato hasta la fecha. No obstante, resulta importante destacar que la diferencia entre ellos es minúscula. En términos de regiones geográficas, los tres cuentan con CPDs en España, por lo que cualquiera de las opciones disponibles sigue siendo un buen candidato. Además, el servicio de recomendaciones está disponible para todas las localizaciones en todos los proveedores cloud. Finalmente, y dejando a un lado la documentación, la opción que cuenta con un abanico más amplio de lenguajes es Amazon, aunque el resto soporta, al menos, aquellos más populares. Por todo ello, la solución a considerar sería Amazon Personalize que, aparte de todo lo anterior, es un servicio mucho más maduro y consolidado que el resto de opciones. Proporciona una gran cantidad de opciones para ofrecer a los clientes recomendaciones totalmente personalizadas.



Donde cabe uno, caben dos, caben tres... Mientras algunos desarrollos requieren la utilización de un solo proveedor en la nube, otros combinan diferentes soluciones para sacar partido de las ventajas de cada uno. Lo segundo, desde luego, no suele ser lo más común, pero resulta interesante mencionarlo para que el lector lo tenga en cuenta para futuros proyectos.

2.2.6. Resumen

A continuación se remarcán los aspectos clave abordados en este bloque titulado Detalles de las Plataformas de IA:

¹⁸ Enlace: <https://cloud.google.com/recommendations>

¹⁹ Enlace: <https://azure.microsoft.com/es-es/services/cognitive-services/personalizer/>

²⁰ Enlace: <https://aws.amazon.com/es/personalize/>

- Las plataformas de IA basadas en la nube, desde el punto de vista del desarrollador, son un enfoque verdaderamente potente por i) hacer **accesible** la IA a cualquier individuo y ii) conseguir **ocultar** los detalles complejos de implementación. Además, la **integración** de tecnologías y su buena **documentación** hacen que esta solución sea considerada como un enfoque de **todo integrado**.
- La **nube**, que habilita la existencia de plataformas de IA, puede catalogarse en *pública, privada o mixta* con **modelos de servicios** que los proveedores cloud suministran a los clientes: *infraestructura como servicio (SaaS), plataforma como servicio (PaaS)* y *software como servicio (SaaS)*.
- El **modelo de interacción** con las plataformas de IA se puede clasificar en *consolas web, herramientas de línea de comandos o bibliotecas para el cliente*. Las dos últimas suelen estar integradas en SDKs que usan como **mecanismo de comunicación** las conocidas **APIs** para consumir los servicios de IA. Además, suele considerarse el uso de **opciones de terceros** para crear aplicaciones totalmente funcionales sin depender, en excesiva medida, de los proveedores cloud.
- Las **5 cuestiones** más generales que permiten **diferenciar a una plataforma de IA** basada en la nube son: i) la *localización* de los CPDs del proveedor cloud, ii) las *herramientas y/o servicios* de IA, iii) sus *precios*, iv) SDKs o, concretamente, el soporte de *lenguajes de programación* y, por último y no tan importante, v) su *documentación*.
- La **elección de una plataforma de IA** debería sustentarse en **5 criterios ordenados** de la siguiente manera: 1) *solución* de IA proporcionada por un proveedor cloud, 2) *precio* de la solución, 3) *región* donde esa solución está disponible, 4) *lenguajes de programación* que soporta esa solución y 5) su *documentación*. El último criterio, por lo general, no es excluyente, si no orientativo.

2.3. IA mediante servicios en la nube

En la sección anterior se han discutido los diferentes tipos de interacción con las plataformas de IA: i) consolas web, ii) línea de comandos y iii) SDKs. Además, se ha realizado un breve recorrido sobre los mecanismos de comunicación utilizados para conectar aplicaciones con servicios en la nube. Esta sección parte de ese contenido para mostrar al lector, mediante un **ejemplo práctico** sencillo, cómo utilizar una plataforma de IA en la nube para un caso real.

Si bien es cierto que comenzar a usar una nueva tecnología siempre supone un reto, esta sección trata de servir como una toma de contacto, desde el punto de vista del desarrollo y del uso de la plataforma en sí. Para esto último se ha considerado la utilización de la plataforma en la nube **Amazon AI** enmarcada en la colección AWS por su popularidad, uso y oferta de servicios. Como lenguaje de programación se ha elegido **Python** al considerarse el lenguaje por referencia de este curso, cuyos beneficios y justificación han sido previamente discutidos.

La solución al problema propuesto se ha seguido manteniendo un **enfoque incremental**, desde implementaciones básicas a una final totalmente funcional. Además, las decisiones de diseño se justifican para ofrecer al lector recomendaciones que puedan servirle a la hora de realizar un proyecto de IA.

2.3.1. Consideraciones previas

Antes de adentrarnos en los detalles de implementación del ejemplo propuesto, es preciso realizar un pequeño recorrido sobre la plataforma en la nube que vamos a utilizar, Amazon AI. Por supuesto, este sección no profundiza en el lenguaje de programación Python, el cual ya se discutió en secciones previas.

Amazon AI en detalle

Este catálogo de servicios en la nube dentro de AWS está principalmente destinado para casos de uso común. En otras palabras, Amazon AI ofrece una **variedad de servicios** cuyos modelos de aprendizaje automático ya están **preentrenados y listos para su uso**. Amazon también proporciona herramientas para la creación de modelos desde cero. Sin embargo, esta consideración no se cubre en esta sección, dado que se encuentra fuera del alcance de este caso de estudio.

La principal ventaja de este enfoque es que, a través de pocos clics, tenemos a nuestra disposición servicios de IA que podemos conectar a nuestras aplicaciones a través de simples llamadas a APIs. Indudablemente, la experiencia en este caso sobre aprendizaje automático no es necesaria. Sin embargo, nunca está de más poseer unos mínimos conocimientos en el campo de la IA para aprovecharlos al máximo de estas tecnologías.

Uno de los principales detalles por los cuales Amazon AI se encuentran entre las tres primeras opciones, en cuanto a servicios de IA en la nube se refiere, es por el conjunto de alternativas que ofrece. Veamos a continuación un listado de los servicios más interesantes.

- **Búsqueda inteligente.** El servicio *Amazon Kendra* permite realizar búsquedas de lenguaje natural a los sitios web y aplicaciones.
- **Recomendaciones personalizadas.** Con *Amazon Personalize* es posible personalizar experiencias a los clientes, como por ejemplo, recomendaciones de productos, reclasificación de productos o marketing directo personalizado.
- **Inteligencia de centro de contacto.** *Amazon Contact Lens* es el servicio que ofrece una solución para agregar IA a la atención al cliente.
- **Análisis y extracción de datos automatizados**
 - Procesamiento de documentos. *Amazon Textract* es el servicio que Amazon proporciona para permitir identificar, comprender y extraer datos de grandes volúmenes de archivos de texto.

- Análisis de documentos. La extracción de información a partir de texto no estructurado con PLN (Procesamiento del Lenguaje Natural) se realiza mediante *Amazon Comprehend*.
- Traducción automática. La traducción de textos a diferentes idiomas es posible mediante el servicio *Amazon Translate*.

■ Visión

- Análisis de imagen y vídeo. *Amazon Rekognition* es el servicio capaz de reconocer objetos, personas, texto y actividades, además de detectar contenido inapropiado.

■ Lenguajes de IA

- Bots y agentes virtuales. *Amazon Lex* es el servicio ofrecido por Amazon AI para crear interfaces de conversación con voz y texto.
- Voz a texto. El servicio vinculado a esta comportamiento es ofrecido por *Amazon Transcribe*.
- Texto a voz. *Amazon Polly* es el servicio contrario a *Amazon Transcribe*, transformando un texto dado a voz.

No obstante, Amazon AI también es atractivo por la flexibilidad de sus servicios a la hora de poder aplicarse en diferentes escenarios.

■ Sanidad

- Compresión de datos. Con *Amazon HealthLake*, proveedores del sector médico pueden almacenar, consultar, transformar y analizar datos sanitarios.
- Transcripciones médicas. Amazon ofrece *Amazon Transcribe Medical* para obtener el mejor diagnóstico posible a partir de las conversaciones entre el proveedor de atención médica y el paciente.

Se recomienda al lector consultar la web de Amazon²¹ sobre aprendizaje automático para tener una visión general sobre las posibilidades de la plataforma.



Popularidad de AWS. Según el informe de 2021 de Stack Overflow ([Enlace: insights.stackoverflow.com/survey/2021#section-most-popular-technologies-cloud-platforms](https://insights.stackoverflow.com/survey/2021#section-most-popular-technologies-cloud-platforms)), AWS mantiene su liderazgo como la plataforma en la nube más utilizada.

2.3.2. Problemática

Los **datos compartidos** en la red incluyen, a menudo, detalles sensibles, lo cual conlleva, y genera, **problemas de privacidad**. La mayoría de los usuarios de internet tienen cierto grado de conciencia sobre la protección de la privacidad. Por ejemplo, cuando un usuario sube una foto a las redes sociales puede elegir una configuración para que la foto sólo sea visible para sus amigos. Sin embargo, ellos, a menudo, no consiguen la privacidad que desean.

La principal solución que adoptan la mayoría de aplicaciones (p. ej., las redes sociales) para resolver la privacidad en las imágenes es dejar en manos de los usuarios el control sobre quién puede acceder al contenido. No obstante, este enfoque, y otros similares, como des-etiquetar la foto, abordan problemas de privacidad, impiendo que otras personas no deseadas vean sus fotos.

Atendiendo a esta situación, se pretende construir un sistema artificial que sea capaz de **analizar imágenes**, de forma que sea posible detectar cuando hay una cara a la que aplicarle una filtro de difuminado y **ocultar el rostro de una persona**. Con este enfoque, parte de la foto se oculta, pixelando o distorsionando la cara de la persona para evitar su identificación. El desenfoque es el método más utilizado y ampliamente estudiado para controlar la divulgación del contenido de las fotos.

Para facilitar el diseño y desarrollo de comportamientos inteligentes en este caso en particular, este caso práctico discute las bases de la integración y validación de estos comportamientos de la mano de AWS. De este modo, el lector podrá dedicar principalmente su esfuerzo a la creación y *testing* de dichos comportamientos con el objetivo de adquirir las habilidades necesarias para desarrollar IA con una plataforma en un contexto real.

²¹ Enlace: aws.amazon.com/es/machine-learning/



Figura 2.6: Rostros de personas difuminadas.

2.3.3. Arquitectura propuesta

Antes de entrar en los detalles de implementación, veamos el aspecto de la arquitectura de este sencillo sistema. El objetivo es visualizar cómo los servicios colaboran entre sí para ofrecer el comportamiento inteligente deseado. La figura 2.7 muestra la estructura general del sistema.

Por otro lado, los siguientes pasos describen el flujo de trabajo general:

1. Una imagen es alojada en un nodo de almacenamiento origen para su posterior análisis.
2. Una cadena con el identificador de la imagen almacenada a analizar es enviada a un servidor al endpoint `<url>/api/analyze` mediante una petición POST.
3. El servicio de análisis de imágenes de Amazon identifica si hay algún rostro en la imagen.
4. Si la imagen contiene al menos una cara de una persona se realizan las siguientes acciones:
 - a) Un proceso localiza los rostros de las personas en la imagen y les aplica un filtro para difuminarlas.
 - b) Cuando las imágenes son procesadas, el resultado es enviado a un nodo de almacenamiento destino.

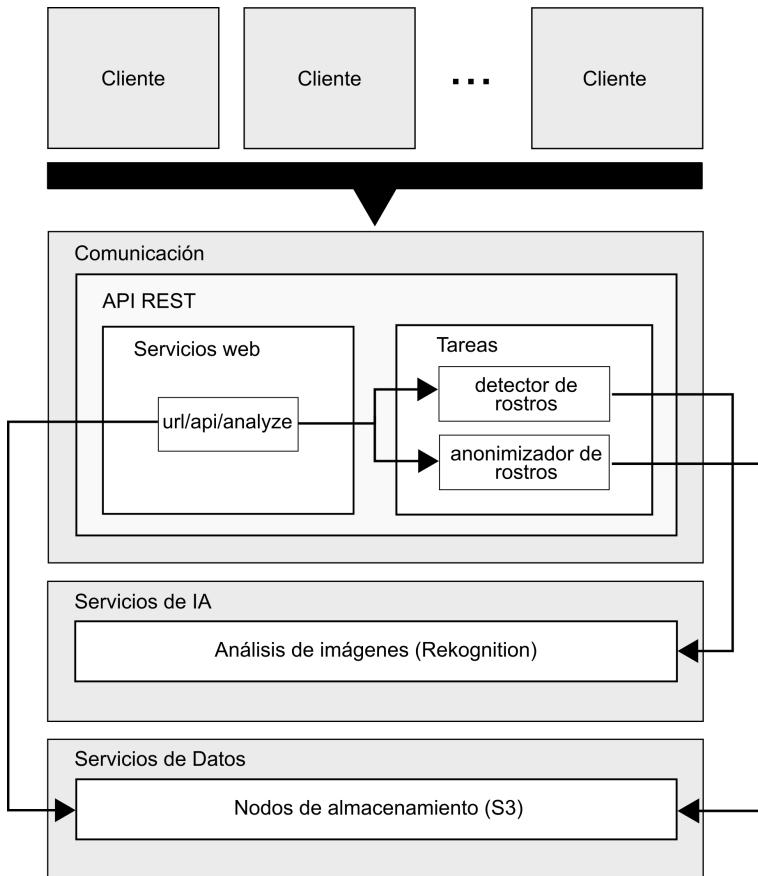


Figura 2.7: Visión general de la arquitectura propuesta.

Comunicación

Internamente, el sistema utiliza como mecanismo de comunicación un modelo basado en la arquitectura REST. Este enfoque permite modelar servicios desacoplados sin saber cómo se recibe el recurso ni de dónde proviene.

En el contexto del sistema discutido en este caso de estudio, esta arquitectura ha ayudado a diseñar una API simple que implica un sistema servidor donde se alojan los servicios. Un cliente envía imágenes y el servidor las procesa, almacenando el resultado en un nodo del almacenamiento como si de una base de datos se tratase. Para este caso de estudio se ha utilizado *Flask* con el objetivo de agilizar la creación de la API. Los detalles del cliente se han omitido, dado que su función puede materializarse manualmente: i) alojando una imagen en el nodo de almacenamiento y ii) realizando una petición al servidor desde la línea de comandos.

Servicios web

Este servicio está disponible como endpoint al que se accede a través de una API expuesta en la red:

- POST <url>/api/analyze. Este endpoint toma una cadena de texto que identifica si hay alguna cara visible para aplicarle un filtro de difuminado.

Para activar el análisis, partimos de que un usuario toma imágenes en situaciones aleatorias. Estas son enviadas a un nodo de almacenamiento para el análisis de caras y, en caso de que el resultado sea favorable, una nueva tarea se encarga de difuminar los rostros que han sido detectados.

Tareas

La tarea de detección de rostros proporciona una interfaz a *Amazon Rekognition*, enviando imágenes para su análisis y recogiendo los resultados. Por el contrario, la tarea de difuminado únicamente se encarga de recibir una imagen con la ubicación 2D de las caras de las personas para aplicar un filtro a los rostros. Para esto último se utiliza la biblioteca OPENCV para facilitar la manipulación de las imágenes.

Servicios de IA

Este sistema utiliza un único servicio de IA, *Amazon Rekognition*²². Este servicio ofrece varios modos de reconocimiento de imágenes, como la detección de objetos y escenas, el reconocimiento facial, el análisis facial, el reconocimiento de celebridades, la detección de texto en imágenes y la detección de equipamiento de protección. Para este caso de estudio se utiliza la API de detección de caras.

Servicio de Datos

A nivel de Servicios de Datos, este caso de estudio utiliza el servicio de almacenamiento simple, es decir, *Amazon S3*²³. Esto es suficiente para las necesidades de este prototipo.

2.3.4. Implementación

Esta sección parte de que el lector cuenta una experiencia previa mínima en programación, particularmente con el lenguaje Python. Por lo tanto, se obvian cuestiones básicas del lenguaje y otras relacionadas con la instalación de paquetes, entre otras cuestiones.

²² Enlace: <https://aws.amazon.com/es/rekognition/>

²³ Enlace: <https://aws.amazon.com/es/s3/>

Código base del sistema

En el listado 2.6 se muestra una implementación base del servidor en *Flask* que recibe peticiones en el endpoint <url>/api/analyze (línea ⑩). Como se puede observar, en la línea ⑪ se encuentra el método analyzeImage enlazado al endpoint anterior. Este código, más adelante, se extenderá para incluir la lógica necesaria para realizar el comportamiento inteligente.

Listado 2.6: Ejemplo base del servidor *Flask* en Python

```
1 #!/usr/bin/python3
2
3 # Import packages
4 from flask import Flask, request, Response
5
6 # Create Flask application
7 application = Flask(__name__)
8
9 # api/analyze endpoint
10 @application.route('/api/analyze', methods=['POST'])
11 def analyzeImage():
12     print(request.get_json())
13     return Response(status=200)
14
15 # Run the app
16 if __name__ == "__main__":
17     application.debug = True
18     application.run() # Running on http://127.0.0.1:5000/
```

Se invita al lector a probar el código base anterior, realizando una llamada al endpoint definido para comprobar que el servidor funciona correctamente y recibe peticiones de manera satisfactoria. A modo de nota aclaratoria, se requiere de dos sesiones de terminal para hacer funcionar el ejemplo.

```
# Terminal 1
$ Python3 application.py

# Terminal 2
$ curl localhost:5000/api/analyze
-H 'Content-Type: application/json'
-X POST -d '{
    "message": "This is a test"
}'
```

Integrando IA básica

El esquema básico planteado hasta ahora no es más que la base para la implementación de los componente de IA. En otras palabras, el código fuente discutido anteriormente representa un servidor base que permanece a la escucha de nuevas peticiones. El objetivo es que el servidor atienda dichas peticiones para realizar una llamada a la tarea de análisis de imagen de Amazon y procesar su resultado. En definitiva, delegar las tareas complejas en otros actores.

Veamos ahora cómo integrar IA básica al servidor. Como se ha comentado anteriormente, el objetivo de este caso de estudio es anonimizar las caras de personas en una imagen. El servicio de Amazon realiza un análisis completo sobre la imagen y retorna un resultado en caso de que exista al menos una cara. Sin embargo, el difuminado del rostro no es una operación que este servicio ofrezca. Por lo tanto, resulta obligatorio utilizar un paquete de terceros que, por un lado, facilite esta tarea y, por otro, la complejidad no se vea alterada. OPENCV es una opción verdaderamente adecuada, ya que es una biblioteca de referencia para visión por computador. En el contexto de este caso de estudio, su uso se basa en la gestión de la imagen a anonimizar y el aplicado de filtros para conseguir un efecto de difuminado. El listado de código 2.7 muestra una función que se encarga de la parte relativa al difuminado del rostro de una persona.

Listado 2.7: Función que transforma una imagen aplicando un filtro de difuminado a los rostros de las personas detectadas

```
1 #!/usr/bin/python3
2
3 import numpy as np
4 import cv2
5
6 def anonymize_face(buffer, response):
7     # Read image from buffer
8     nparr = np.fromstring(buffer, np.uint8)
9     img = cv2.imdecode(nparr, cv2.IMREAD_COLOR)
10
11     height, width, _ = img.shape
12
13     for faceDetail in response['FaceDetails']:
14         box = faceDetail['BoundingBox']
15         x = int(width * box['Left'])
16         y = int(height * box['Top'])
17         w = int(width * box['Width'])
18         h = int(height * box['Height'])
19
20         # Get region of interest
21         roi = img[y:y+h, x:x+w]
22
23         # Applying a gaussian blur over this new rectangle area
24         roi = cv2.GaussianBlur(roi, (83, 83), 30)
25
26         # Impose this blurred image on original image to get final image
27         img[y:y+roi.shape[0], x:x+roi.shape[1]] = roi
28
```

```
29     print("Blurred face task has been successfully run")
30
31     # Encode image and return image with blurred faces
32     _, res_buffer = cv2.imencode('.jpg', img)
33
34     return res_buffer
```

Los parámetros que recibe la función son una imagen en *bytes* y la respuesta que *Amazon Rekognition* retorna como consecuencia de haber detectado un rostro. La sintaxis de la respuesta se encuentra en el listado 2.8.

Listado 2.8: Respuesta de *Amazon Rekognition* como consecuencia de llamar a la tarea de detección de caras

```
1  {
2      "FaceDetails": [
3          {
4              "AgeRange": {
5                  "High": number,
6                  "Low": number
7              },
8              "Beard": {
9                  "Confidence": number,
10                 "Value": boolean
11             },
12             "BoundingBox": {
13                 "Height": number,
14                 "Left": number,
15                 "Top": number,
16                 "Width": number
17             },
18             "Confidence": number,
19             "Emotions": [
20                 {
21                     "Confidence": number,
22                     "Type": "string"
23                 }
24             ],
25             "Eyeglasses": {
26                 "Confidence": number,
27                 "Value": boolean
28             },
29             "EyesOpen": {
30                 "Confidence": number,
31                 "Value": boolean
32             },
33             "Gender": {
34                 "Confidence": number,
35                 "Value": "string"
36             },
37             "Landmarks": [
38                 {
39                     "Type": "string",
40                     "X": number,
41                     "Y": number
42                 }
43             ],
44         ]
45     }
```

```
44     "MouthOpen": {
45         "Confidence": number,
46         "Value": boolean
47     },
48     "Mustache": {
49         "Confidence": number,
50         "Value": boolean
51     },
52     "Pose": {
53         "Pitch": number,
54         "Roll": number,
55         "Yaw": number
56     },
57     "Quality": {
58         "Brightness": number,
59         "Sharpness": number
60     },
61     "Smile": {
62         "Confidence": number,
63         "Value": boolean
64     },
65     "Sunglasses": {
66         "Confidence": number,
67         "Value": boolean
68     }
69 }
70 ],
71     "OrientationCorrection": "string"
72 }
```

Como se puede observar, la respuesta contiene atributos con información de todo tipo, como género, rango de edad, ojos abiertos, gafas o vello facial. Incluso también proporciona predicciones sobre la emoción de la cara detectada. En este pequeño ejemplo se obvian estas características, pero se utiliza el atributo *BoundingBox* que contiene una aproximación de la zona de la imagen donde se encuentran los rostros reconocidos.

Un cuadro delimitador o *BoundingBox* (en inglés) tiene las siguientes propiedades:

- Altura (*Height*): La altura del cuadro delimitador como proporción de la altura total de la imagen.
- Izquierda (*Left*): La coordenada izquierda del cuadro delimitador como proporción de la anchura total de la imagen.
- Parte superior (*Top*): La coordenada superior del cuadro delimitador como proporción de la altura total de la imagen.
- Anchura (*Width*): La anchura del cuadro delimitador como proporción de la anchura total de la imagen.

Cada propiedad del cuadro delimitador (*BoundingBox*) tiene un valor entre 0 y 1. Sus valores son una proporción de la anchura total de la imagen (*Left* y *Width*) y de la altura (*Height* y *Top*). Por ejemplo, si la imagen de entrada es de 800 x 300 píxeles, y la coordenada superior izquierda del cuadro delimitador es de 360 x 60 píxeles, la API devuelve un valor *Left* de 0,45 (360/800) y un valor *Top* de 0,2 (60/300). La figura 2.8 muestra una representación del rango de una imagen que cubre cada propiedad del cuadro delimitador.

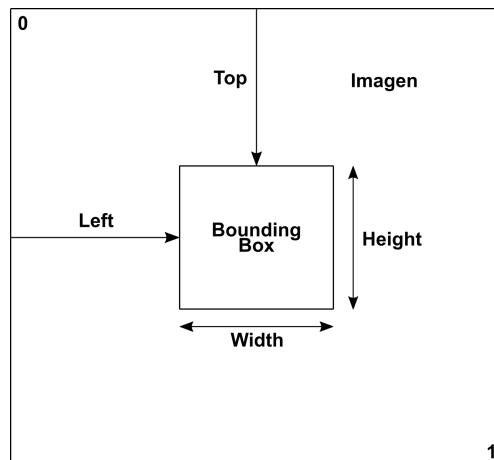


Figura 2.8: Detalles del cuadro delimitador (*BoundingBox*).

Para obtener el cuadro delimitador con la ubicación y el tamaño correctos es necesario realizar una pequeña operación aritmética con los valores del *Bounding-Box*, junto con la anchura y altura de la imagen. Básicamente esto sirve para obtener sus valores en píxeles. Antes de entrar en detalle, primero es necesario convertir el buffer de *bytes* a imagen, cuya operación se realiza en las líneas Ⓛ 8-9. Fundamentalmente, esta transformación se realiza para cargar la imagen en memoria. Después, las líneas Ⓛ 14-18 realizan una multiplicación con los valores del *BoundingBox*, y la anchura y la altura de la imagen, cuyos valores son recuperados en la línea Ⓛ 11 con la función `image.shape`. Todo esto está englobado en un bucle por si la respuesta de *Amazon Rekognition* contiene más de un rostro identificado. Además, véase que el resultado de las operaciones aritméticas se convierten a enteros, básicamente porque cada píxel de la imagen representa un número.

El resto del código obtiene el cuadro delimitador de la imagen y le aplica un filtro para difuminar dicha región. Concretamente, la línea Ⓛ 21 realiza la operación de obtener la región de interés de la imagen, la cual resulta ser el cuadro delimitador. Considera (0,0) como la esquina superior izquierda de la imagen total con la coordenada *x* de izquierda a derecha y la coordenada *y* de arriba a abajo. Si tene-

mos $(x1, y1)$ como el vértice superior izquierdo y $(x2, y2)$ como el vértice inferior derecho de una región de interés, el área del rectángulo de la imagen se obtiene al iterar desde y hasta $y+h$ (filas de la lista `img` con los píxeles del *BoundingBox*), y desde x hasta $x+w$ (columnas de la lista `img` con los píxeles del *BoundingBox*).

Por otro lado, en la línea **②24** se aplica la técnica del desenfoque gaussiano, mayoritariamente utilizada para reducir el ruido en una imagen. En este caso, su uso sirve para difuminar la región de interés (`roi`) que contiene el rostro de una persona. En líneas generales, y sin entrar demasiado en detalle, este algoritmo escanea cada píxel de la región de interés proporcionada y recalcula el valor del píxel basándose en aquellos que lo rodean. El área que se escanea alrededor de cada píxel es una matriz, cuyo valor es representado por el segundo parámetro de la función. El tercero especifica la desviación estándar del filtro en el eje X. Se invita al lector a modificar los valores de estos parámetros, ya que la solución propuesta no es única. Además, para más información, se recomienda consultar la función `GaussianBlur` en la API de OPENCV²⁴.

Por último, en la línea **②27**, se copia el contenido de la región de interés a la imagen original `y`, en la línea **③2**, su resultado se codifica para retornarlo en `bytes`. Es necesario almacenar la nueva imagen en memoria para enviar dicho contenido al nodo de almacenamiento de destino.

Integrando IA avanzada

Aunque la IA incluida hasta ahora podría parecer un tanto elaborada, su aplicación se basa en el resultado de realizar una operación compleja para detectar la región de una imagen donde se encuentran los rostros de las personas. Esta solución se apoya en un **servicio en la nube**, el cual presenta una serie de ventajas:

- El desarrollador no necesita conocimientos profundos en IA.
- La integración de esta funcionalidad en una aplicación se realiza a través de una API expuesta en la nube.
- Con muy pocas líneas de código es posible aplicar técnicas de IA.
- Y un largo etc.

La implementación que se muestra en el listado de código 2.9 refleja las ventajas anteriores, realizando una llamada a la API de *Amazon Rekognition* para detectar, dada una imagen, rostros de personas.

En esta implementación destacan los siguientes puntos:

- Las líneas **⑩-13** recuperan las variables de entorno depositadas en un archivo `.env`. Como se discutió en anteriores secciones, este enfoque permite añadir una capa de seguridad al código, ya que las claves están aisladas en un archivo separado que no se comparte.

²⁴🔗 Enlace: docs.opencv.org/master/index.html

- La sesión de *Amazon Rekognition* está fuera de la definición de la función para crear una sola vez la instancia del servicio de análisis de imagen (líneas ⑩ 16-19). En el caso que dicha sesión expirase, el SDK automáticamente actualiza los detalles subyacentes cuando se realice una nueva llamada al servicio, facilitando así la labor del desarrollador.
- La función `detect_faces` recibe como parámetro un cadena que identifica la imagen alojada en un nodo S3. La extensión también se debe proporcionar.
- La ejecución de la IA avanzada se encuentra en la línea ⑩ 24, invocando a la función de detección de rostros de la API *rekognition*. Como indica su nombre, esta función está destinada a reconocer rostros de una imagen. El parámetro `Image` especifica que la imagen está en un nodo S3. Por otro lado, el atributo `Attributes` indica que solo deben ser retornados los atributos faciales por defecto. Es decir, en la respuesta se obviarán muchas de las propiedades contenidas en el listado 2.8.

Listado 2.9: Función que analiza imágenes mediante el servicio *Amazon Rekognition*.

```
1 #!/usr/bin/python3
2
3 import boto3, os
4 from dotenv import load_dotenv
5
6 # Load env variables from .env file
7 load_dotenv()
8
9 # Get env variables
10 accessKeyId = os.environ.get('ACCESS_KEY_ID')
11 secretKey = os.environ.get('ACCESS_SECRET_KEY')
12 bucket = os.environ.get('BUCKET_SOURCE')
13 region = os.environ.get('REGION')
14
15 # Create the service Rekognition and assign credentials
16 rekognition_client = boto3.Session(
17     aws_access_key_id=accessKeyId,
18     aws_secret_access_key=secretKey,
19     region_name=region).client('rekognition')
20
21 def detect_faces(img):
22     # Assign parameters and call the service
23     try:
24         response = rekognition_client.detect_faces(
25             Image={'S3Object': {'Bucket': bucket, 'Name': img}}, Attributes=['DEFAULT'])
26
27         print("Image task recognition has been successfully run")
28     except:
29         raise Exception("An unexpected error was raised recognizing an image")
30
31     return response
```

En este punto, ya es posible introducir la nueva implementación del servidor, mostrada en el listado 2.10, el cual introduce las siguientes novedades más notables:

- Se han importado nuevos elementos en el código. Destacan principalmente las funciones de detectar y anonimizar rostros, en las líneas **⑥-7**.
- Se ha creado el servicio S3 de Amazon (líneas **⑯-20**) por dos motivos. El primero para acceder al nodo de almacenamiento y recuperar la imagen (líneas **⑳-33**), cuyo identificador está contenido en el cuerpo de la petición de la línea **⑳**. El segundo con el objetivo de enviar la imagen anonimizada al nodo de destino (línea **⑳**).
- Para evitar problemas relacionados con la petición, se ha añadido un control de error en las líneas **㉖-㉗** que gestione que el contenido del cuerpo del mensaje. De lo contrario, se retorna un código de estado 400 para informar al cliente que la petición es incorrecta.
- La función `detect_faces` retorna una respuesta como consecuencia de realizar una llamada al servicio de análisis de imágenes en la nube. Este contenido es usado como parámetro, junto con la imagen en `bytes`, en la función `anonymize_face` de línea **㉔**. Toda esta lógica se ha englobado en el bloque `try`, comenzando en la línea **㉙**, y finalizando en el bloque `except`, de la línea **㉜**. Un error procedente de alguno de los servicios anteriores retornará un código de estado 500. Su uso sirve para avisar al cliente que ha ocurrido un error interno en el servidor.
- En las líneas **㉞-㉟** se transforma la imagen comprimida `buffer_anon_img` en `bytes` para poder subir el archivo anonimizado al nodo de almacenamiento de destino.

Listado 2.10: Implementación final del servidor *Flask* en Python

```
1 #!/usr/bin/python3
2
3 import boto3, os, base64
4 from flask import Flask, request, Response, abort
5 from dotenv import load_dotenv
6 from detect_faces import detect_faces
7 from blur_faces import anonymize_face
8
9 load_dotenv()
10
11 accessKeyId = os.environ.get('ACCESS_KEY_ID')
12 secretKey = os.environ.get('ACCESS_SECRET_KEY')
13 bucket_source = os.environ.get('BUCKET_SOURCE')
14 bucket_dest = os.environ.get('BUCKET_DEST')
15
16 application = Flask(__name__)
17
18 # Create the s3 service and assign credentials
19 s3 = boto3.Session(aws_access_key_id=accessKeyId,
20                     aws_secret_access_key=secretKey).resource('s3')
21
22 # api/analyze endpoint
23 @application.route('/api/analyze', methods=['POST'])
24 def analyzeImage():
```

```
25     key = request.get_json()['key']
26     if key is None:
27         abort(400)
28
29     try:
30         response = detect_faces(key)
31
32         fileObject = s3.Object(bucket_source, key).get()
33         fileContent = fileObject['Body'].read()
34         buffer_anon_img = anonymize_face(fileContent, response)
35
36         img_enc = base64.b64encode(buffer_anon_img)
37         img_dec = base64.b64decode(img_enc)
38         s3.Object(bucket_dest, f"result_{key}").put(Body=img_dec)
39     except Exception as error:
40         print(error)
41         abort(500)
42     return Response(status=200)
43
44 # Run the app...
```

Resultado

A modo de guía para probar el sistema discutido en este caso de estudio, se incluye, a continuación, una serie de pasos que sirvan como referencia al lector para poner el sistema en funcionamiento.

En primer lugar, se requiere realizar una configuración previa en la nube de Amazon para crear los nodos de almacenamiento donde se guarden las imágenes origen (aquellas a analizar) y las de destino (aquellas ya analizadas). Se incluye, por tanto, los pasos necesarios haciendo uso de la línea de comandos de AWS previamente discutida. A modo de nota aclaratoria, el paso (3) requiere la ruta de la imagen a subir y el nombre que se le asigna cuando se añade al nodo de almacenamiento. Ambos archivos con su extensión.

```
# (1) Crear nodo de almacenamiento de origen
$ aws s3 mb s3://bucket-name-source --region region-name

# (2) Crear nodo de almacenamiento de destino
$ aws s3 mb s3://bucket-name-destination --region region-name

# (3) Añadir imagen al nodo de almacenamiento de origen
$ aws s3 cp image-location s3://bucket-name-source/image-name
```

Finalmente, solo queda ejecutar el servidor y realizar una petición al endpoint `localhost:5000/api/analyze`, dado que no se ha considerado desplegar esta API en la nube para no complicar demasiado el ejemplo. Nótese que el valor asociado a la clave `key` del mensaje JSON es el nombre de la imagen almacenada en el nodo de almacenamiento de origen. El resultado se almacenará en el nodo de destino.

```
# Terminal 1  
$ Python3 application.py  
  
# Terminal 2  
$ curl localhost:5000/api/analyze  
-H 'Content-Type: application/json'  
-X POST -d '{  
    "key": "<image-name-uploaded>"  
}'
```

La figura 2.9 muestra un ejemplo de la salida del sistema, pixelando los rostros de las personas que aparecen en la imagen como consecuencia de utilizar el servicio *Amazon Rekognition*.



Figura 2.9: Resultado de enviar una imagen al servidor para su anonimización.

2.3.5. Consideraciones finales

En esta sección se ha planteado una implementación inicial de un problema real, con el objetivo de proporcionar una enfoque general relativo al desarrollo de aplicaciones inteligentes con plataformas de IA. A partir de aquí, es necesario prestar especial atención a los siguientes aspectos para incrementar la funcionalidad del prototipo discutido:

- Por motivos de simplicidad, los aspectos relacionadas con cuestiones de seguridad han sido obviados. Sin embargo, con vistas a tener el sistema desplegado en la nube sería interesante incluir **mecanismos de autenticación** para restringir el uso de la API.
- El código relativo a la IA encargado de detectar rostros de personas se podría modificar para conseguir un **comportamiento inteligente más elaborado**. Por ejemplo, se podría utilizar la propiedad de la edad en la respuesta de *Amazon Rekognition* para pixelar los rostros de personas dentro de un rango especificado.
- El comportamiento interno de la aplicación debería de modelarse mediante un **enfoque dirigido por eventos**. Así, cada vez que una imagen sea introducida en el nodo S3 se dispare una acción que analice dicha imagen y se aplique la técnica de difuminado.
- La consideración anterior da pie a utilizar un **enfoque basado en colas de mensajes**, enviando una notificación desde Amazon S3 a una implementación basado en colas de AWS. Este mecanismo, además, permite controlar el caudal de mensajes enviados al servicio de análisis facial y desacoplar los componentes del sistema.
- En línea con lo anterior, se recomienda utilizar una **comunicación sin servidor**, convirtiendo las tareas en **funciones como servicios**. En AWS esto se conoce como funciones *lambda*.

2.3.6. Resumen

En esta sección se remarcán los aspectos más importantes abordados durante el caso de estudio enmarcado en el bloque titulado AI mediante Servicios en la Nube.

- Se ha realizado un recorrido a alto nivel por los **servicios de IA en la nube de Amazon**, destacando aquellos más relevantes e interesantes.
- Este bloque ha discutido la problemática existente sobre la privacidad de las personas en las imágenes en internet. En este contexto, se ha planteado un **caso práctico** sencillo para mostrar el desarrollo de una aplicación bajo un caso real utilizando una plataforma de IA en la nube. El objetivo es analizar una imagen y detectar si existe el rostro de una persona para aplicarle un filtro de difuminado.
- Se han mostrado los aspectos más importantes de la **arquitectura** propuesta, junto con los servicios utilizados de la nube de Amazon.
- En este caso práctico se ha discutido, de modo incremental, el código de la **implementación del sistema**, profundizando en los detalles relativos al comportamiento inteligente.
- Se ha incluido una serie de **consideraciones finales**, a modo de trabajo futuro, que animen al lector a modificar el código fuente y mejorar tanto la arquitectura del sistema como el comportamiento inteligente.

3

Capítulo

Entornos de modelado de IA

Santiago Sánchez Sobrino, Jesús Serrano Guerrero
Javier Albusac Jiménez, Cristian Gómez Portes

En este capítulo se introducen los entornos de modelado; herramientas y bibliotecas que nos permiten aplicar el marco de trabajo completo del aprendizaje automático a un problema concreto, en resumen, recolección de los datos y su preparación, entrenamiento del modelo, evaluación del modelo y despliegue del modelo para su integración en algún sistema de negocio. Este proceso puede realizarse de manera automatizada y guiada o de manera manual. En este último caso, seremos nosotros mismos los que indiquemos la naturaleza de los datos a analizar y decidamos qué estadísticos aplicar en el proceso de aprendizaje automático.

3.1. Herramientas de modelado

Como se ha visto a lo largo del capítulo anterior, existen numerosas herramientas en la nube que nos permiten aplicar procesos completos de aprendizaje automático a problemas concretos. Estas herramientas nos permiten ajustar y definir, de manera personalizada, todas las etapas del ciclo de vida de desarrollo con el fin de poder crear, entrenar y desplegar un modelo para su puesta en producción. En la práctica, y a la hora de hablar de sistemas en producción, estas etapas del ciclo de vida incluyen los pasos que ya conocemos y otros previos:

1. Definición y análisis del problema.
2. Planteamiento de las hipótesis.
3. Selección de métricas de salida.
4. Recolección y preparación de los datos.
5. Construcción y creación de los modelos de ML.

6. Entrenamiento de los modelos.
7. Evaluación y despliegue de los modelos.
8. Repetición del ciclo de vida.

Un equipo de *data science* puede poseer la disciplina para definir el problema y la hipótesis, realizar el tratamiento inicial de los datos, seleccionar las propiedades adecuadas basándose en las instrucciones del experto en la materia, elegir la familia de modelos adecuada, optimizar los *hiperparámetros* del modelo, revisar los resultados y las métricas de salida, y por último optimizar los modelos. Este proceso se repite iterativamente, de tal forma que el científico de datos tendrá que asegurarse de que los datos, la versión del modelo y los futuros reentrenamientos sean llevados a cabo. También debe poner salvaguardas para garantizar que el rendimiento del modelo sea supervisado. Además de todo lo anterior, también se suelen llevar a cabo competiciones y experimentos A/B en producción para encontrar el mejor modelo.

Con todo lo anterior, surge el concepto de aprendizaje automático automatizado o *AutoML*, el cual trata facilitar el trabajo de los expertos mediante la automatización de aquellas tareas que puedan ser triviales y repetitivas, para que los expertos puedan enfocarse en las tareas que realmente importan. El conjunto de tareas susceptible de ser automatizadas incluye:

- Definición de propiedades (*feature engineering*) mediante el uso de técnicas que incluyen la expansión/reducción, las transformaciones de organización jerárquica, el metaaprendizaje y el aprendizaje por refuerzo.
- Búsqueda de arquitecturas neuronales mediante el uso de algoritmos evolutivos, meta-aprendizaje, aprendizaje por refuerzo y técnicas de búsqueda local, entre otros.
- Optimización de hiperparámetros mediante optimizaciones Bayesianas, algoritmos evolutivos, funciones lipschitzianas, técnicas de búsqueda local y optimización de enjambres de partículas, por citar algunas.



El concepto de **hiperparámetro** será introducido con más detalle posteriormente. De momento, es suficiente con saber que los modelos de aprendizaje automático tienen *parámetros internos* (o parámetros del modelo), intrínsecos al modelo y que se estiman durante el proceso de entrenamiento con los conjuntos de datos (p.ej., los pesos en una ANN o los coeficientes en una regresión lineal, entre otros); y *parámetros externos* (o hiperparámetros), utilizados para configurar el proceso de entrenamiento de antemano (p. ej., número de capas de una ANN, ratio de aprendizaje o el número de iteraciones, entre otros).



No hay que abrumarse por la cantidad de conceptos que se mencionan en este capítulo, ya que el enfoque de AutoML facilita la creación y entrenamiento de modelos abstrayendo al usuario de estos conceptos. En capítulos posteriores se irán detallando cada uno de los conceptos que se vean aquí, por lo que se recomienda al lector volver a este capítulo para entender realmente la abstracción que propone el paradigma AutoML sobre el paradigma tradicional de aprendizaje automático.

Consideremos un ejemplo donde tratamos de construir un modelo de aprendizaje automático que nos permita reconocer los dígitos de una imagen. El modelo tomará las imágenes como entrada y producirá una salida con los dígitos de la imagen. A grandes rasgos, para construir el modelo necesitaremos un *dataset* para entrenar y validar el modelo, el algoritmo que aplicaremos y los parámetros que proporcionaremos al algoritmo.

Un enfoque típico que podemos aplicar para seleccionar el algoritmo que vamos a utilizar, es buscar otros problemas similares al que estemos resolviendo e identificar un conjunto de algoritmos utilizados en esos problemas que puedan ser susceptibles de ser utilizados en nuestro problema. Una vez dispongamos de este conjunto de algoritmos, tendremos que probarlos uno a uno con nuestro *dataset* y elegir el que mejor se desenvuelva en nuestro problema de reconocer dígitos.

El proceso anterior parece intuitivo y sistemático, pero realizarlo manualmente puede llevarnos horas o días si no tenemos la suficiente experiencia a la hora de, por ejemplo, revisar el estado del arte, comprender cómo se implementan los algoritmos e integrarlos con los datos de nuestro *dataset*. Un enfoque basado en AutoML nos permitiría automatizar este proceso mediante la búsqueda automática del algoritmo de aprendizaje automático que mejor resultado ofrezca en el contexto de los datos y el problema al que nos enfrentamos. Así, una implementación básica de un algoritmo basado en AutoML que nos proporcione el mejor algoritmo de aprendizaje automático podría contar con los siguientes pasos:

1. Seleccionar un algoritmo de aprendizaje automático del conjunto de algoritmos disponible.
2. Construir un modelo de aprendizaje automático utilizando el algoritmo seleccionado.
3. Evaluar el modelo para obtener un resultado.
4. Almacenar el resultado obtenido y el modelo construido, y repetir el proceso para el resto de algoritmos.
5. Finalmente, seleccionar el mejor modelo de los construidos en base a los resultados obtenidos.

Así, los entornos de AutoML estarían formados por tres elementos principales:

- Espacio de búsqueda: estaría formado por el conjunto de algoritmos de aprendizaje automático y el tipo de algoritmo.
- Estrategia de búsqueda: típicamente se definirá mediante una estrategia secuencial que itere sobre todos los algoritmos de aprendizaje automático siguiendo un enfoque de prueba y error. Una mejor estrategia de búsqueda puede ayudar a obtener un mejor algoritmo dentro del mismo intervalo de tiempo, así como permitir utilizar un espacio de búsqueda aún mayor reduciendo el tiempo de búsqueda y los costes computacionales.
- Estrategia de evaluación de rendimiento: se trata de una estrategia que nos permita evaluar el rendimiento de un algoritmo de aprendizaje automático concreto con respecto de otro.

Los entornos de AutoML suelen encapsular estos tres elementos a través de una API que utilice un espacio de búsqueda y un algoritmo de búsqueda por defecto, abstrayendo así al usuario de tener que configurar estos elementos manualmente. Con todo esto, el usuario solo tendría que encargarse de proporcionar los datos al entorno de AutoML para construir y entrenar los modelos.



Dado que existen usuarios que parten de diferentes experiencias y conocimientos, la mayoría de entornos de AutoML permiten realizar ajustes avanzados en caso de que el usuario desee modificar el espacio de búsqueda de algoritmos de aprendizaje automático o la estrategia de búsqueda para encontrar el más adecuado.

Existen multitud de herramientas de AutoML, algunas de ellas de bajo nivel e implementadas como bibliotecas que puedan utilizarse desde Python u otros lenguajes de programación (p.ej., TPOT, Auto-SKLearn, AutoKeras, AutoGluon, PyCaret, etc.), y otras de alto nivel que pueden ser ejecutadas de manera visual en entornos de desarrollo completos. En el resto de la sección se introducen tres de estos entornos de AutoML que pueden ser ejecutados directamente en la nube. Finalmente, se plantea un caso práctico en el que se revisan todos los pasos para construir, entrenar y desplegar un modelo de aprendizaje automático siguiendo el enfoque de AutoML.

3.1.1. Google Cloud AI

La plataforma de Google Cloud Computing (GCP) ofrece una gran variedad de servicios en la nube que podemos utilizar para construir aplicaciones inteligentes. Concretamente, puede verse una lista completa y actualizada de todos los servicios ofrecidos en Enlace: <https://cloud.google.com/terms/services>.

Google se posicionó como una de las primeras empresas en ofrecer servicios de inteligencia artificial en la nube mediante una amplia oferta donde construir modelos de aprendizaje automático, utilizar cuadernos de desarrollo, realizar etiquetado de datos, generar tareas de aprendizaje automático y flujos de trabajo, entre otros.

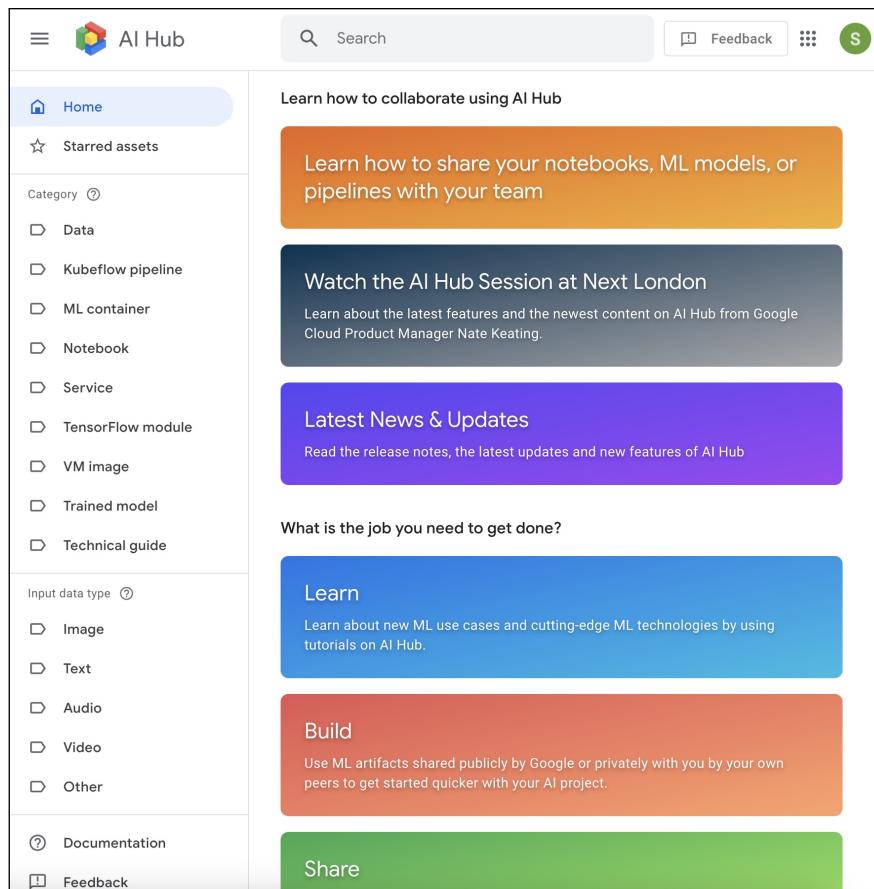


Figura 3.1: Vista principal del AI Hub de Google Cloud.

El punto de entrada a todos estos servicios es el *Google Cloud AI Hub*¹, el cual permite desplegar sistemas completos de IA en la nube a golpe de click (ver Figura 3.1).

El sitio de Google Cloud AI Hub proporciona un punto de entrada a todo el desarrollo de IA mediante kits para principiantes, las últimas noticias sobre casos de uso de ML, tutoriales y tecnologías de ML emergentes. Así mismo, el entorno proporciona acceso a cientos de ejemplos concretos que podemos ejecutar directamente sobre la plataforma.

En lo que respecta al aprendizaje automático automatizado, Google Cloud proporciona un conjunto de servicios que permiten construir modelos de aprendizaje automático mediante un enfoque de AutoML²:

¹ Enlace: <https://aihub.cloud.google.com/>

² Enlace: <https://cloud.google.com/automl>

- *AutoML Natural Language*³: permite a los usuarios clasificar mensajes de entrada con etiquetas personalizadas (clasificación supervisada). Los usuarios pueden ajustar los modelos a sus propios dominios o casos de uso.
- *AutoML Tables*⁴: permite construir y desplegar modelos de aprendizaje automático entrenados a partir de datos estructurados, a gran velocidad y escala.
- *AutoML Translation*⁵: se trata de una solución simple y escalable que permite a usuarios con experiencia limitada en tecnologías de aprendizaje automático personalizar y ajustar modelos de *Google Neural Machine Translation* (GNMT) a sus propios dominios o casos de uso.
- *AutoML Video*⁶: permite a los usuarios entrenar modelos basados en vídeo de manera personalizada y escalable para su propio dominio o caso de uso.
- *AutoML Vision*⁷: permite a los usuarios entrenar modelos basados en visión por computador de manera personalizada y escalable para su propio dominio o caso de uso.
- *Recommendations AI*⁸: permite construir sistemas personalizados de recomendaciones basados en modelos de aprendizaje profundo, sin necesitar un experto en aprendizaje automático o arquitecturas de sistemas de recomendaciones.

De todos estos servicios, **AutoML Tables** es el que nos permite construir, entrenar y desplegar modelos de aprendizaje automático. Partiendo de los datos estructurados, el servicio analiza las propiedades de entrada, selecciona el modelo, realiza el ajuste de hiperparámetros y evalúa el comportamiento del modelo repetidamente para asegurar la consistencia, precisión y fiabilidad de éste.



A fecha de redacción de este capítulo, Google se encuentra en un proceso de unificación de todos sus servicios de IA en lo que ha denominado **Vertex AI** (🔗 Enlace: <https://cloud.google.com/vertex-ai>), una solución que aúna el enfoque de AutoML para el entrenamiento de modelos y el entrenamiento personalizado.

Caso práctico: Google Cloud AutoML Tables

Como se ha mencionado, AutoML Tables se encarga de analizar de manera automática las diferentes combinaciones de modelos de aprendizaje automático e hiperparámetros hasta dar con un modelo óptimo para el dominio del problema. Para ello, el primer paso será importar el *dataset* de entrenamiento a partir de una

³🔗 Enlace: <https://cloud.google.com/natural-language/automl>

⁴🔗 Enlace: <https://cloud.google.com/automl-tables>

⁵🔗 Enlace: <https://cloud.google.com/translate/automl>

⁶🔗 Enlace: <https://cloud.google.com/video-intelligence/automl>

⁷🔗 Enlace: <https://cloud.google.com/vision/automl>

⁸🔗 Enlace: <https://cloud.google.com/recommendations>

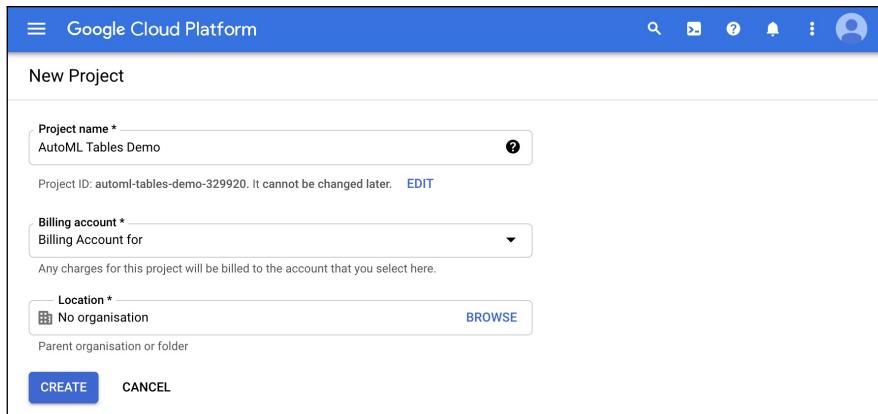


Figura 3.2: Creación de un proyecto utilizando la consola de Google Cloud.

tabla *BigQuery*, un archivo ubicado en Google Cloud Storage, o un archivo local de nuestro computador. Este primer paso llevará algo de tiempo, ya que el sistema tendrá que analizar las columnas del *dataset*. Una vez haya finalizado, la herramienta permitirá editar el esquema autogenerado para cambiar los tipos de las propiedades (y si alguna de ellas puede ser *null*) y seleccionar la propiedad sobre la que realizaremos las predicciones, es decir, la variable dependiente. Después de este análisis inicial, podremos pasar al entrenamiento automático del modelo, donde podremos realizar algunos ajustes como, por ejemplo, la cantidad máxima de horas que puede durar el entrenamiento. Finalmente, una vez se haya completado el entrenamiento, tendremos que realizar la evaluación analizando las métricas sobre el rendimiento del modelo y desplegarlo para poder obtener predicciones.

Una vez introducido el proceso que seguiremos, vamos a ver cómo ponerlo en práctica con un modelo que nos permitirá predecir el nivel de riesgo de una solicitud de préstamo. Para ello, basaremos el ejemplo en un *dataset* de riesgo crediticio, elaborado por el Dr. Hans Hofmann del Institut für Statistik und Ökonometrie, Universität Hamburg y disponible para su descarga en [Enlace: https://bml.io/W2SpyF](https://bml.io/W2SpyF). Este *dataset* contiene campos como el estado de la cuenta, la duración del crédito solicitado, el historial crediticio, la finalidad del crédito, el importe del crédito y la capacidad de ahorro del cliente, entre otros.

Lo primero que haremos será acceder a la consola de Google Cloud Platform y crearemos un nuevo proyecto ([Enlace: https://console.cloud.google.com/projectcreate](https://console.cloud.google.com/projectcreate)) (ver Figura 3.2). Después, nos dirigiremos al sitio de AutoML Tables ([Enlace: https://console.cloud.google.com/automl-tables](https://console.cloud.google.com/automl-tables)), seleccionaremos el proyecto que acabamos de crear y activaremos la API para poder usar el entorno de Cloud AutoML (ver Figura 3.3).

Continuaremos con la importación del *dataset* seleccionando la entrada **Data-sets** del menú izquierdo, pulsando sobre el botón **New Data Set** y completando los campos de la ventana de diálogo que nos aparecerá (ver Figura 3.4). Después, importaremos el *dataset* del riesgo crediticio que descargamos anteriormente selec-

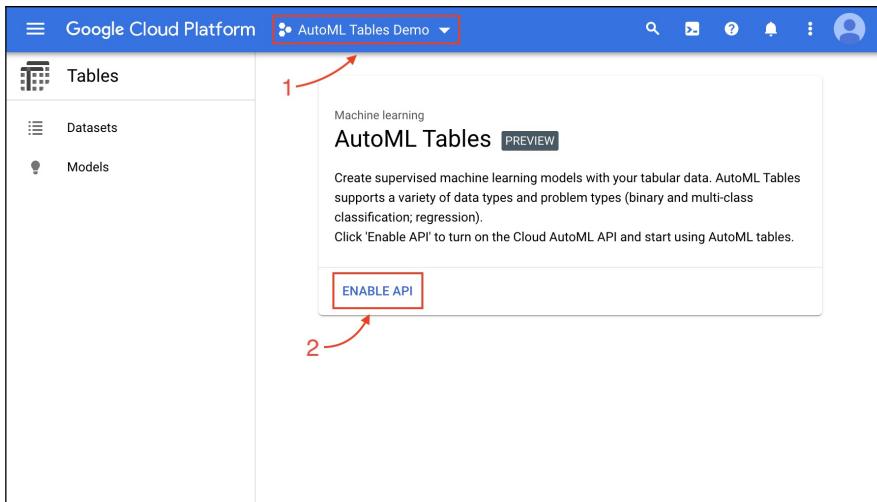


Figura 3.3: Segundo paso para importar el *dataset*; 1) selección del proyecto y 2) activación de la API.

cionándolo desde nuestro equipo y siguiendo los pasos tal y como se muestra en la Figura 3.5. Si no tenemos un destino de almacenamiento, tendremos que crear uno pulsando sobre la opción **Browse**; en la sección que aparecerá, simplemente tendremos que indicar un nombre único para el almacenamiento y seguir el asistente (podemos dejar las opciones por defecto). Tras confirmar la importación, el proceso se iniciará y nos avisará de que puede llegar a tardar hasta 1 hora (ver Figura 3.6); al terminar, recibiremos un email con un enlace que podemos seguir para continuar el proceso.



En caso de que estemos trabajando con otro *dataset*, este deberá contener un mínimo de 1000 filas para que el proceso de importación funcione correctamente. En cualquier otro caso, la plataforma nos arrojará un error indicándonoslo.

Tras terminar la importación, el proceso continuará automáticamente a la pestaña **Train**. Aquí, podremos ver un resumen de los datos y podremos modificar los tipos de columnas y si pueden ser *null* o no. Si seleccionamos alguna de las columnas, podremos ver algunas estadísticas y gráficos adicionales. Para continuar, seleccionaremos la columna objetivo, es decir, la columna sobre la que realizaremos las predicciones, que en nuestro caso será la columna **class** y pulsaremos sobre el botón **Train Model** (ver Figura 3.7). Al pulsar sobre este botón, tendremos que ajustar algunas opciones adicionales como, por ejemplo, el número de horas máximo que queremos entrenar al modelo o los objetivos de optimización que se buscarán durante el entrenamiento, entre otros. En cualquier caso, y tras realizar los ajustes que deseemos, pulsaremos sobre el botón **Train Model** para iniciar el proceso.

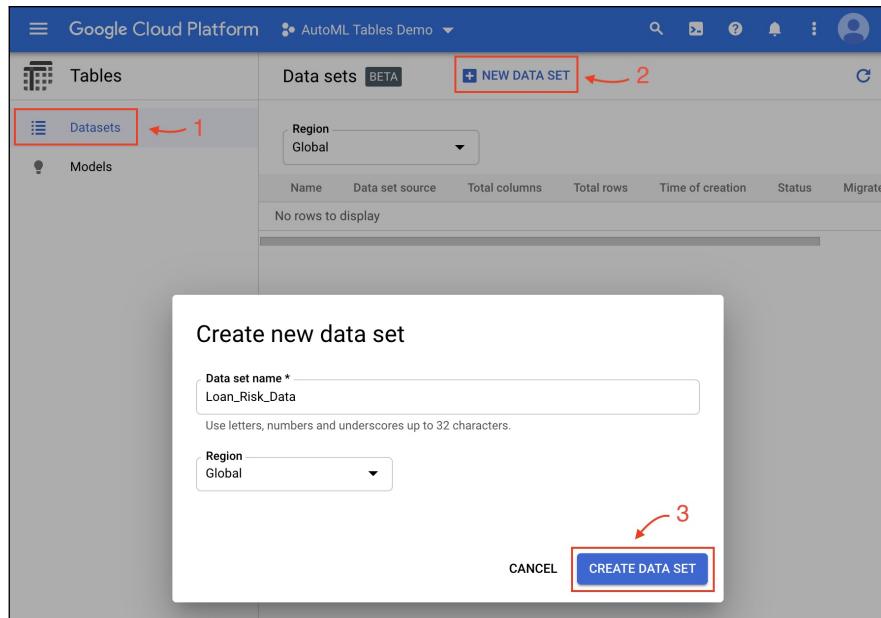


Figura 3.4: Tercer paso para importar el *dataset*; 1) sección de *Datasets*, 2) botón para crear el *dataset* y 3) confirmación para crear el *dataset*.

Una vez se haya completado el entrenamiento, podremos ver las métricas generadas sobre el rendimiento del modelo que nos permitirán evaluar cómo de precisos a la hora de realizar predicciones (ver Figura 3.8).

En la pestaña **Evaluate** tendremos una vista más detallada de las métricas generadas y otros detalles (ver Figura 3.9), a destacar, la precisión del modelo, la matriz de confusión y las propiedades que más impacto han tenido en el entrenamiento del modelo. En este último caso, podemos ver que el estado de la cuenta bancaria, la duración y el propósito del crédito son las tres propiedades que más relevancia han tenido.

Tras realizar analizar los resultados de la evaluación y confirmar que el modelo generado cumple las expectativas, llega la hora de desplegarlo para realizar predicciones sobre nuevos datos de entrada. Para ello, nos dirigiremos a la pestaña **Test & Use**. Desde aquí podremos realizar predicciones en lote (*batch prediction*) a partir de archivos CSV de entrada o desde el servicio de Google Cloud BigQuery, o predicciones utilizando una API online (*online prediction*). Realizaremos algunas pruebas utilizando este último tipo de predicciones, pero antes, tendremos que desplegar el modelo tal y como se muestra en la Figura 3.10.

Tras finalizar el proceso, podremos realizar las predicciones online utilizando el modelo que acabamos de desplegar. Por ejemplo, cambiando los datos de los parámetros veremos que obtenemos recomendaciones concretas con un valor de confianza dado.

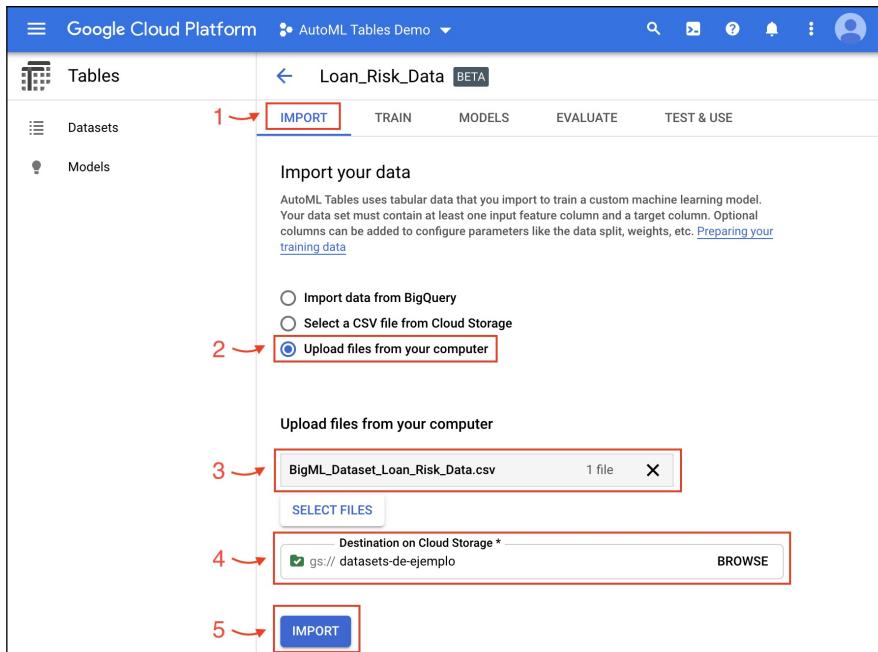


Figura 3.5: Cuarto paso para importar el *dataset*; 1) selección de la pestaña para importarlo, 2) elección para carga local, 3) selección del archivo, 4) almacenamiento en Google Cloud Storage donde se almacenará y 5) confirmación del proceso

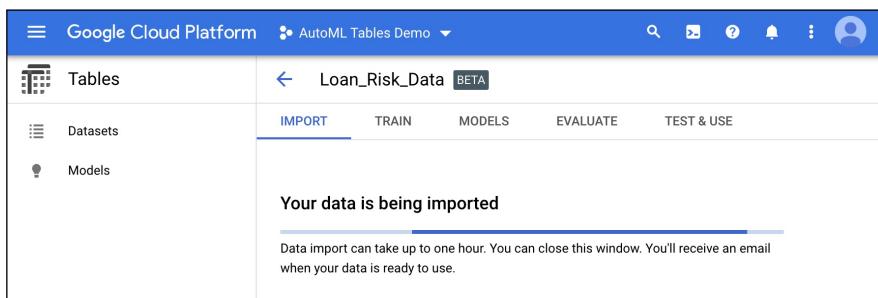


Figura 3.6: Quinto paso para importar el *dataset*.



Una vez se hayan terminado las pruebas con el modelo es conveniente eliminar su despliegue para no incurrir en posibles costes. Para ello, simplemente habrá que pulsar sobre el botón **Remove Deployment**.

Por último, la herramienta nos permite exportar los modelos que hayamos generado como paquetes de TensorFlow que puedan ser ejecutados mediante un contenedor Docker (ver Figura 3.11).

The screenshot shows the Google Cloud Platform AutoML Tables Demo interface. The 'TRAIN' tab is selected. On the left, there's a summary of the dataset: Total columns: 21, Total rows: 1,000. Below this is a table of column statistics:

Column name	Data type	Nullability	Missing% (Count)	Invalid values	Distinct values	Correlation with target
age	Numeric	Nullable	0% (0)	0% (0)	53	0.
checking_status	Categorical	Nullable	0% (0)	0% (0)	4	0.
class Target	Categorical	Nullable	0% (0)	0% (0)	2	
credit_amount	Numeric	Nullable	0% (0)	0% (0)	921	0.
credit_history	Categorical	Nullable	0% (0)	0% (0)	5	0.
duration	Numeric	Nullable	0% (0)	0% (0)	33	0.

Figura 3.7: Primer paso del entrenamiento del modelo; 1) selección de la columna objetivo y 2) confirmación para iniciar el entrenamiento del modelo

3.1.2. Azure Machine Learning

Azure Machine Learning forma parte del ecosistema de Azure AI, que ayuda a acelerar el ciclo de vida de aprendizaje automático de principio a fin utilizando la potencia de la plataforma y los servicios de Windows Azure.

Se trata de un servicio completamente gestionado que incorpora entornos de cómputo, alojamiento de cuadernos Jupyter y funcionalidades para la gestión de modelos, control de versiones y reproducibilidad de modelos. Además, se integra completamente con el resto de servicios de Azure permitiendo, por ejemplo, conectar entornos de cómputo y alojamiento ya existentes al ciclo de vida completo de ML.

El servicio puede ser utilizado mediante el SDK de Python, los cuadernos Jupyter, R y la consola por línea de comandos. O también puede ser utilizado de manera visual mediante el diseñador interactivo para construir, probar y desplegar modelos de forma fácil y rápida utilizando algoritmos de aprendizaje automático preconstruidos.

En el contexto de AutoML, Azure Machine Learning proporciona un entorno completo para construir modelos que resuelvan problemas de clasificación, regresión, predicción de series temporales y visión por computador.

Vamos a ver un ejemplo completo de cómo se construiría y entrenaría un modelo de aprendizaje automático utilizando la herramienta visual de Azure Machine Learning Designer y después el mismo ejemplo utilizando la herramienta AutoML que proporciona el ecosistema de Azure AI.

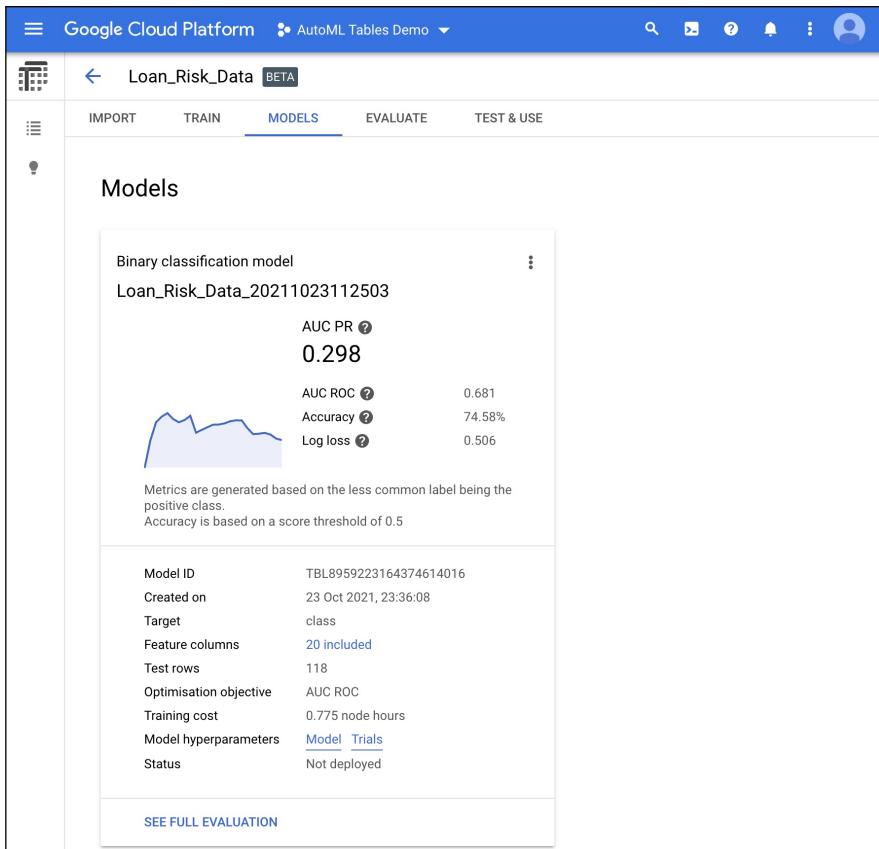


Figura 3.8: Entrenamiento del modelo completado

Caso práctico: Azure Machine Learning Designer

Azure Machine Learning Designer⁹ es un entorno visual que nos permite construir, evaluar y desplegar modelos de aprendizaje automático en la nube. Se trata de una herramienta *no-code* por lo que toda la interacción con la herramienta se producirá mediante el uso del ratón sin que tengamos que escribir ninguna línea de código.

Para empezar, lo primero que haremos es dirigirnos al sitio web de Microsoft Azure Machine Learning Studio, disponible en ⁹ Enlace: <https://ml.azure.com/>. Si es la primera vez que accedemos y todavía no disponemos de un *workspace*, entonces crearemos uno siguiendo el enlace **Create a new workspace** (ver Figura 3.12). En la nueva vista, rellenaremos los campos indicando el **Resource group** que utilizaremos y el nombre que le daremos al *workspace*. El resto de campos se llenarán automáticamente (ver Figura 3.13). A partir de aquí, podemos dejar las

⁹ Enlace: <https://azure.microsoft.com/en-us/services/machine-learning/designer>

3.1. Herramientas de modelado

[159]

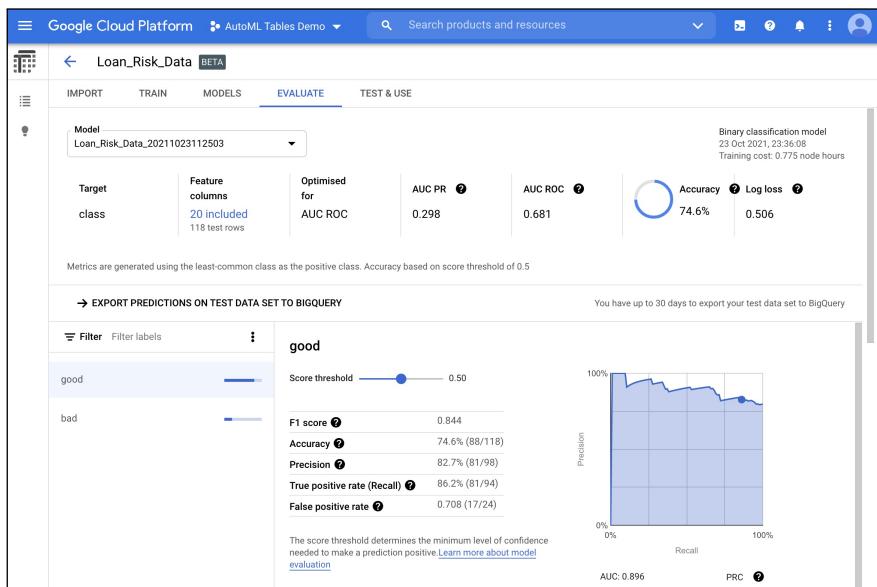


Figura 3.9: Evaluación del modelo

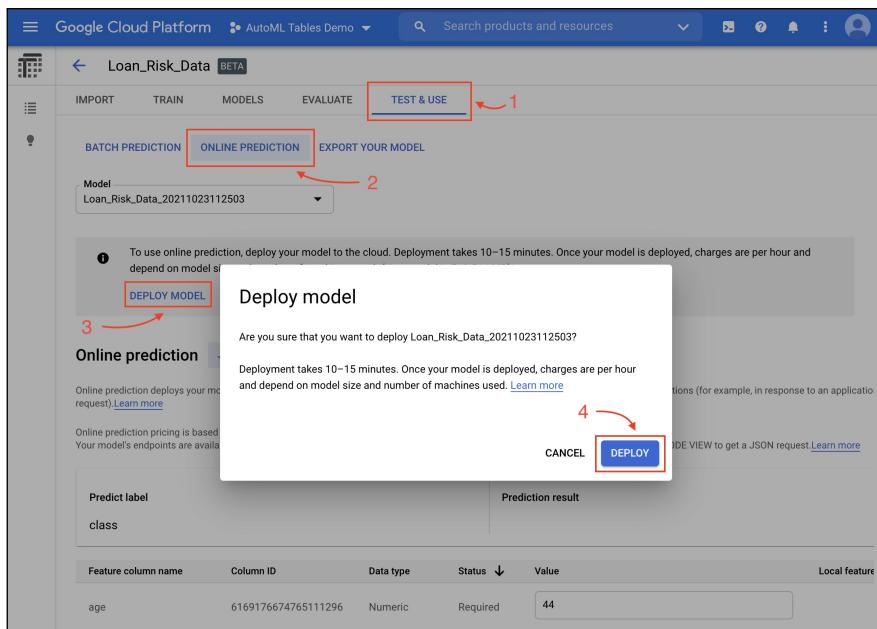


Figura 3.10: Despliegue del modelo para realizar predicciones online

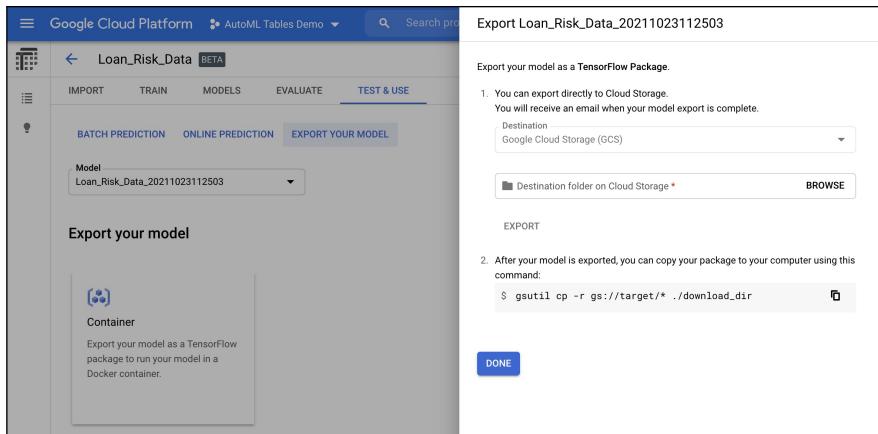


Figura 3.11: Exportación del modelo

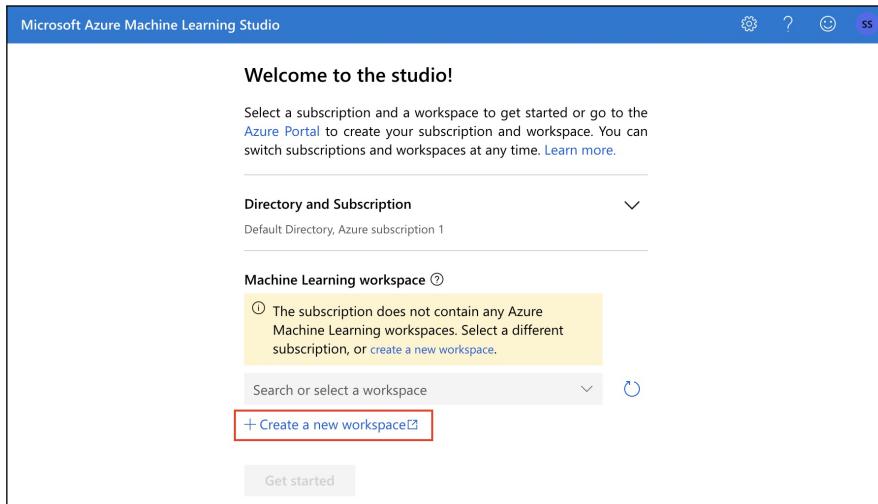


Figura 3.12: Vista inicial del sitio web de Microsoft Azure Machine Learning Studio

opciones por defecto en el resto de pasos y finalizar la creación del *workspace* pulsando sobre el botón **Review + create** o **Create** (ver Figura 3.14). Una vez haya finalizado la creación del *workspace*, volveremos al sitio inicial para seleccionar el *workspace* que acabamos de crear y pulsaremos sobre el botón **Get started** (ver Figura 3.15).

En la nueva vista, accederemos finalmente al ML Studio, desde donde podremos realizar todas las operaciones de aprendizaje automático que nos ofrece la herramienta. Antes de nada, importaremos el *dataset* que utilizaremos durante el ejemplo, el del riesgo de conceder un préstamo que utilizamos anteriormente. Para ello, nos dirigiremos a la sección **Datasets** utilizando el menú lateral izquierdo y

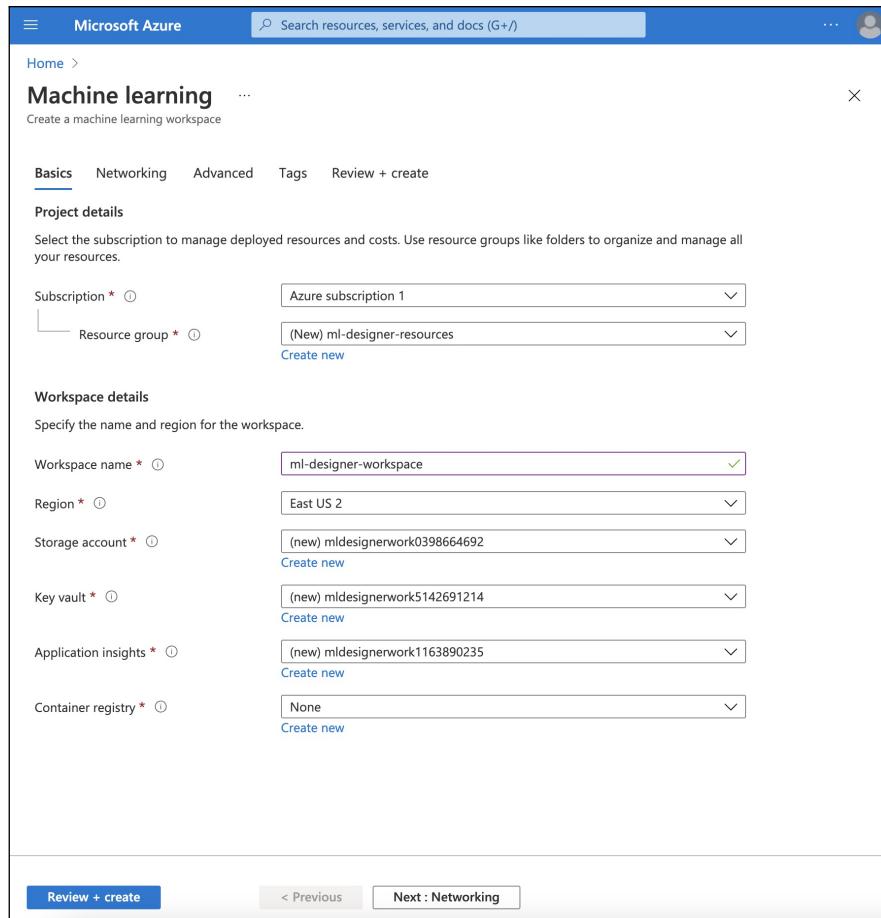


Figura 3.13: Creación del *workspace*

seleccionando la opción **Create dataset >From local files** (ver Figura 3.16). En la nueva vista, le daremos un nombre al *dataset*, seleccionaremos el tipo **Tabular**, el *datastore* donde se almacenará y el archivo que importaremos desde disco con el *dataset*. El resto de opciones las dejaremos por defecto y terminaremos el proceso.

Una vez importado el *dataset*, continuaremos seleccionando la entrada **Designer** del menú lateral izquierdo (ver Figura 3.17) y crearemos una nueva *pipeline* eligiendo la opción **Easy-to-use prebuilt modules** (ver Figura 3.18). En la nueva vista, podremos cambiar el nombre de nuestra *pipeline* a algo más apropiado.

The screenshot shows the 'Machine learning' configuration page in Microsoft Azure. At the top, there's a green banner indicating 'Validation passed'. Below it, tabs for 'Basics', 'Networking', 'Advanced', 'Tags', and 'Review + create' are visible, with 'Review + create' being the active tab. The 'Basics' section lists various configuration details:

Subscription	Azure subscription 1
Resource group	(New) ml-designer-resources
Region	East US 2
Workspace name	ml-designer-workspace
Storage account	(new) mldesignerwork0398664692
Key vault	(new) mldesignerwork5142691214
Application insights	(new) mldesignerwork1163890235
Container registry	None

The 'Networking' section shows 'Connectivity method' set to 'Public endpoint (all networks)'. The 'Advanced' section includes 'Identity type' (System assigned), 'Encryption type' (Microsoft-managed keys), and 'Enable HBI Flag' (Disabled). At the bottom, there are buttons for 'Create' (highlighted with a red box), '< Previous', 'Next >', and 'Download a template for automation'.

Figura 3.14: Resumen de la configuración del *workspace*



Las *pipelines* de Azure Machine Learning permiten organizar múltiples pasos de aprendizaje automático y procesamiento de datos en un único recurso, pudiendo así organizar, gestionar y reutilizar flujos de trabajo de aprendizaje automático complejos entre proyectos y usuarios.

Antes de continuar, deberemos seleccionar la instancia de cómputo que ejecutará la *pipeline* que vamos a crear. Para ello, seleccionaremos el icono del engranaje en la parte superior y seleccionaremos una instancia de cómputo. Si no tenemos ninguna, seguiremos el enlace de **Create Azure ML compute instance** y completaremos los campos (ver Figura 3.19). En este último paso, tendremos que esperar unos 5 minutos hasta que la instancia se haya creado completamente para poder seleccionarla como instancia de uso por la *pipeline*.

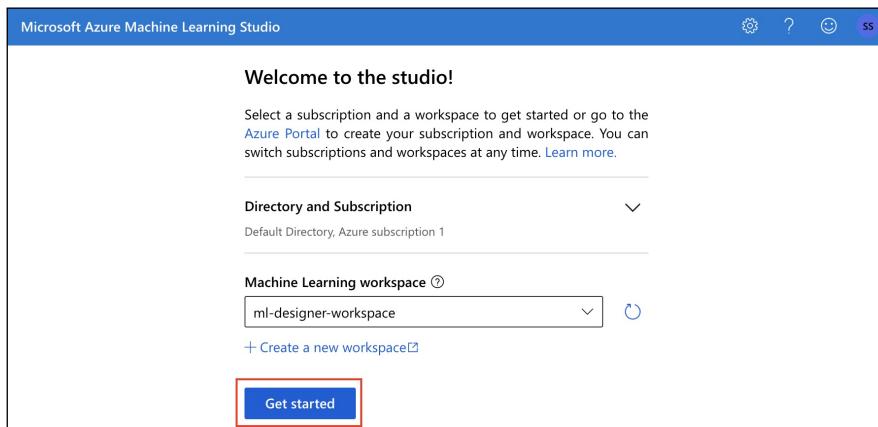


Figura 3.15: Finalización del proceso inicial en el ML Studio

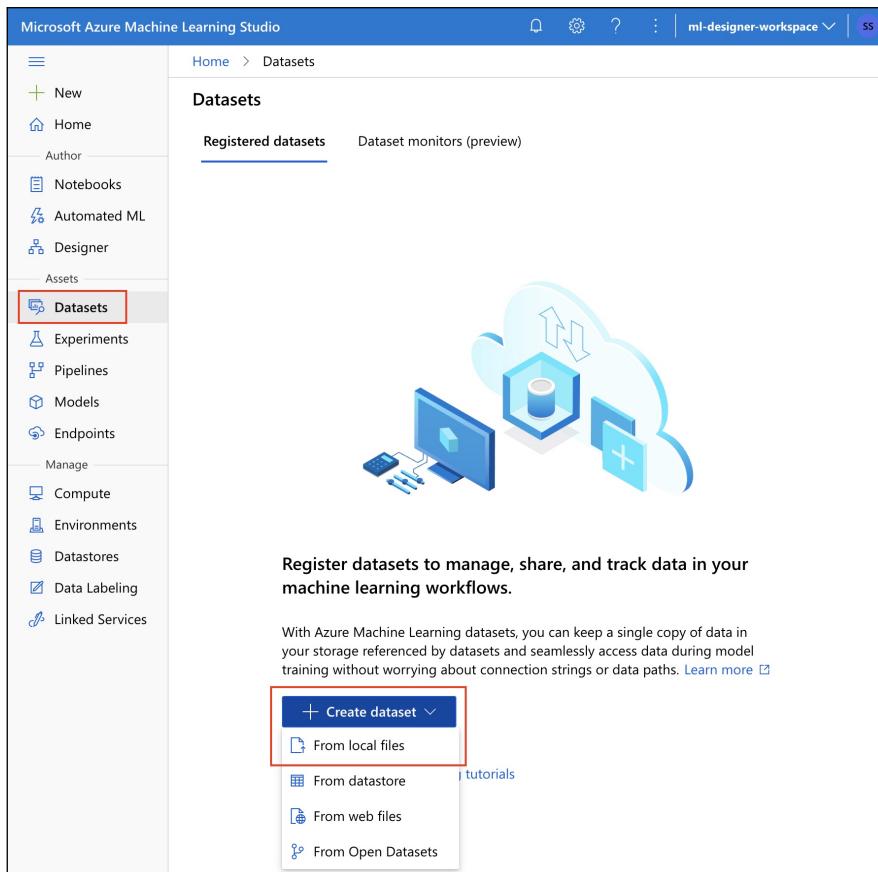


Figura 3.16: Selección del origen del *dataset* a importar

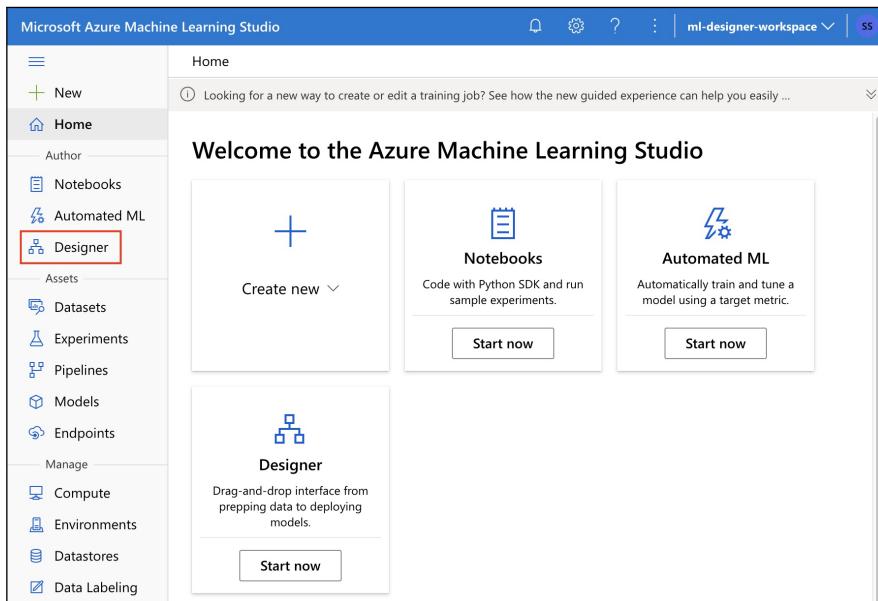


Figura 3.17: Vista principal del ML Studio

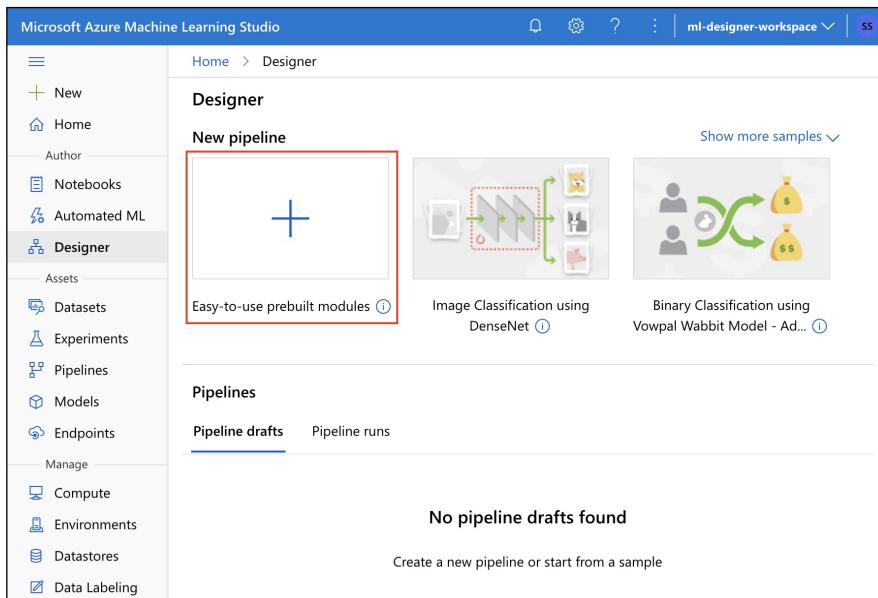


Figura 3.18: Elección del tipo de *pipeline* a crear

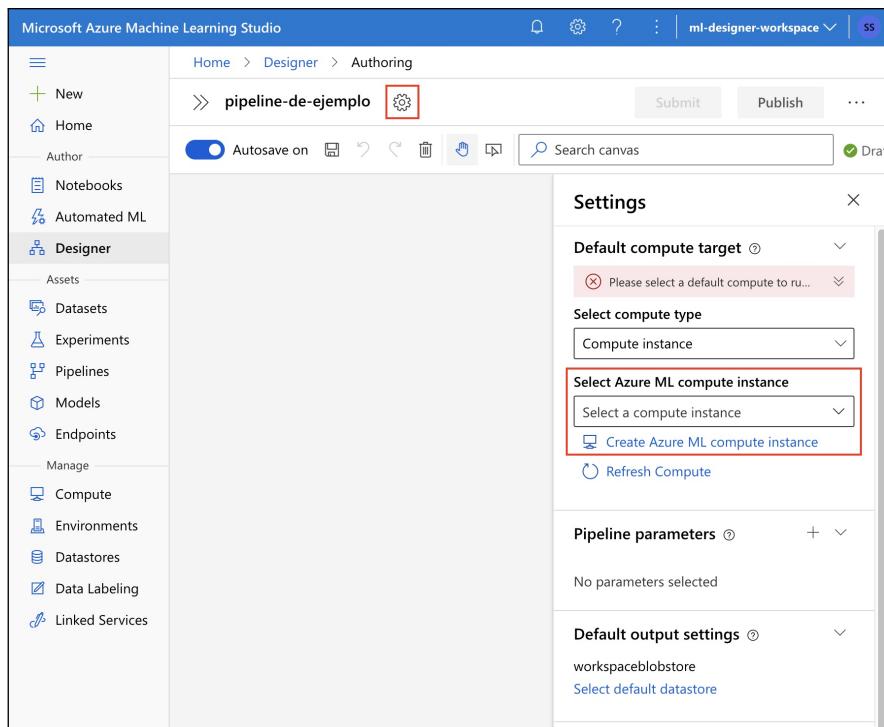


Figura 3.19: Selección de la instancia de cómputo

Una vez tengamos la *pipeline* configurada con un entorno de ejecución, comenzaremos con el proceso de aprendizaje automático. Para ello, el primer paso consistirá en cargar el *dataset* que importamos previamente. Seleccionaremos un nodo **Import Data** de la paleta de recursos del menú lateral izquierdo y lo añadiremos a nuestra *pipeline*. Después, seleccionaremos el nodo que acabamos de añadir y ajustaremos las opciones del menú lateral derecho para cargar el *dataset* (ver Figura 3.20).

Ahora, pasaremos a dividir los datos en datos de entrenamiento (70 %) y datos de prueba (30 %). Para ello, añadiremos un nodo **Split Data**, conectaremos su entrada con la salida del nodo **Import Data** y ajustaremos el ratio de división al 0.7. De esta forma, el pin izquierdo de salida incluirá los datos de entrenamiento (el 70 % de los datos del *dataset*) y el pin derecho de salida incluirá el resto de datos (el 30 % de los datos del *dataset*) (ver Figura 3.21).

A continuación, pasaremos al proceso de entrenamiento del modelo utilizando los datos de entrenamiento. El algoritmo construirá un modelo que explique la relación entre las propiedades y el nivel de riesgo de conceder el préstamo. Para ello, añadiremos un nodo de tipo **Train Model**, configuraremos la columna objetivo o aquella sobre la que queremos realizar las predicciones (en nuestro caso, la columna *class*) y conectaremos su pin derecho de entrada con el pin izquierdo de salida

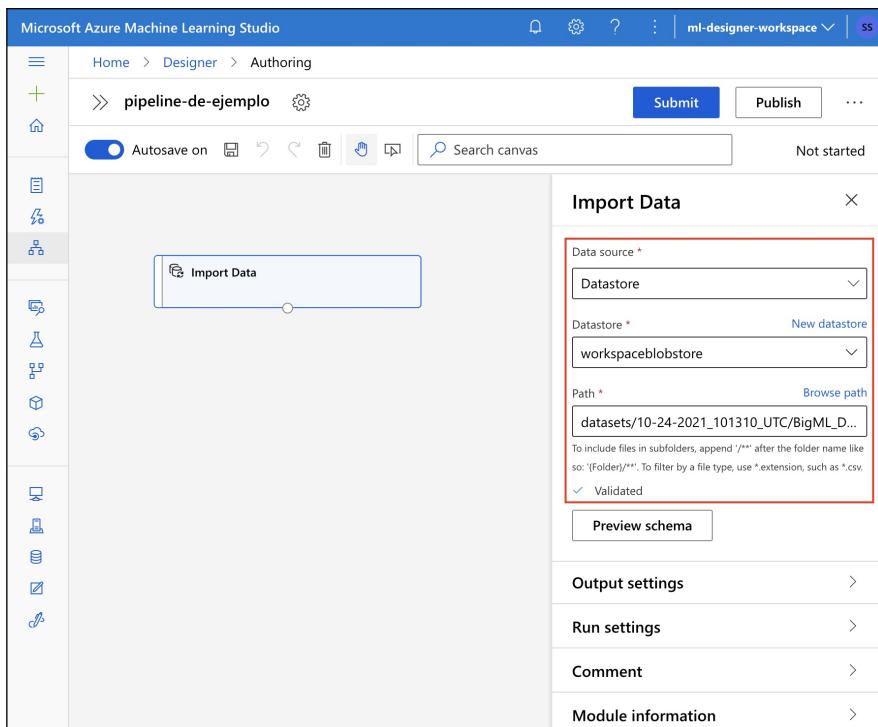


Figura 3.20: Importación del *dataset* en la *pipeline*

del nodo **Split Data**. Después, tendremos que seleccionar el algoritmo de aprendizaje que utilizaremos. Como se trata de un problema de clasificación, tendremos que añadir uno de los algoritmos apropiados para ello. En nuestro caso, hemos seleccionado el algoritmo **Two-Class Neural Network**, ya que se trata de un problema de clasificación binaria. Lo añadiremos a la *pipeline* y conectaremos su pin de salida con el pin de entrada izquierdo del nodo **Train Model** (ver Figura 3.22).



En este punto podemos ajustar los hiperparámetros del algoritmo de aprendizaje automático. Como se mencionó al inicio de la sección, estos ajustes serán detallados en próximas secciones.

Después, añadiremos el nodo **Score Model** a la *pipeline*. Utilizaremos este nodo para generar métricas sobre los datos de prueba (el otro 30 % de los datos que reservamos). Estas métricas las utilizaremos para alimentar un nodo **Evaluate Model**, que nos informará sobre la precisión que ha conseguido el modelo que acabamos de entrenar (ver Figura 3.23 para revisar las conexiones entre nodos).

3.1. Herramientas de modelado

[167]

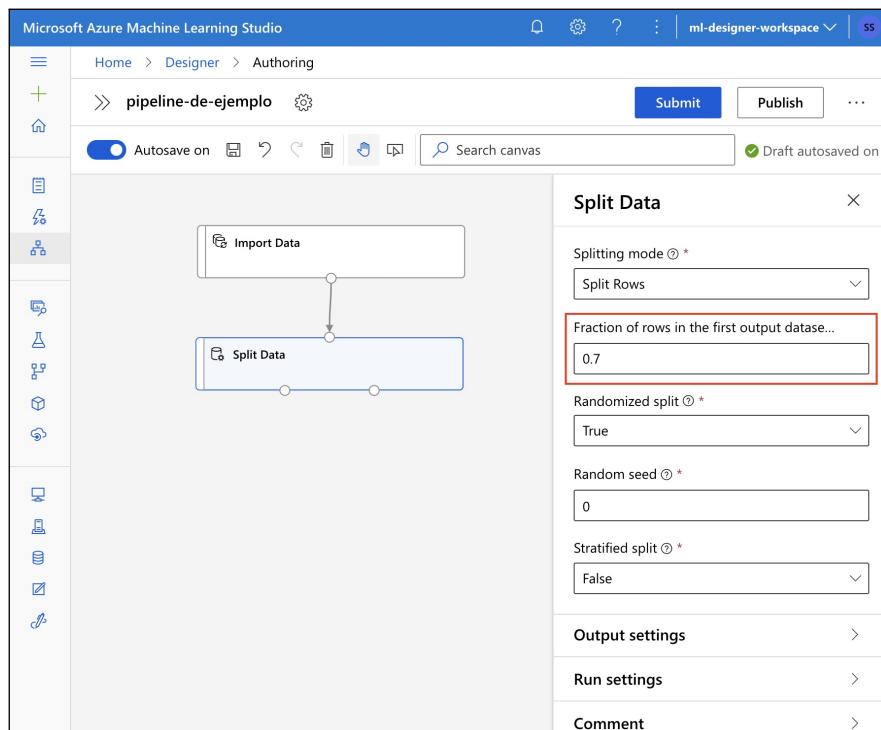


Figura 3.21: División de los datos de entrada

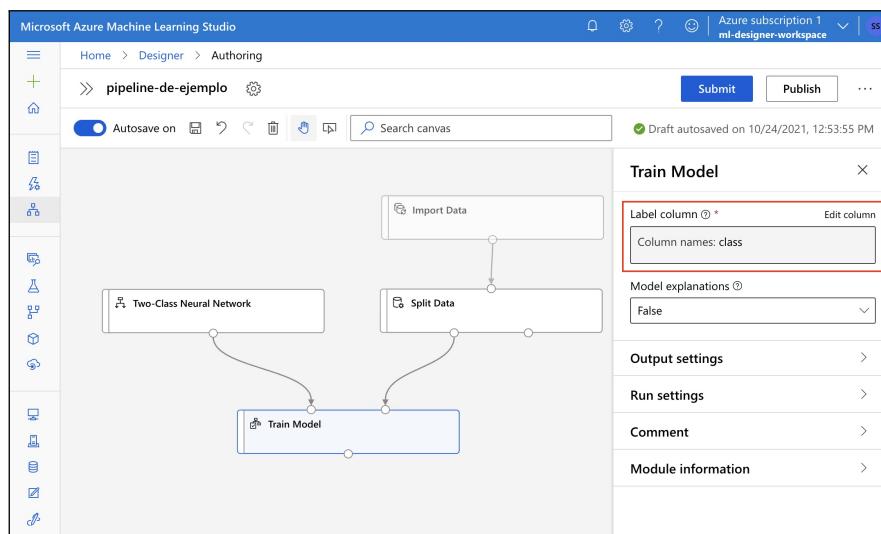


Figura 3.22: Entrenamiento del modelo

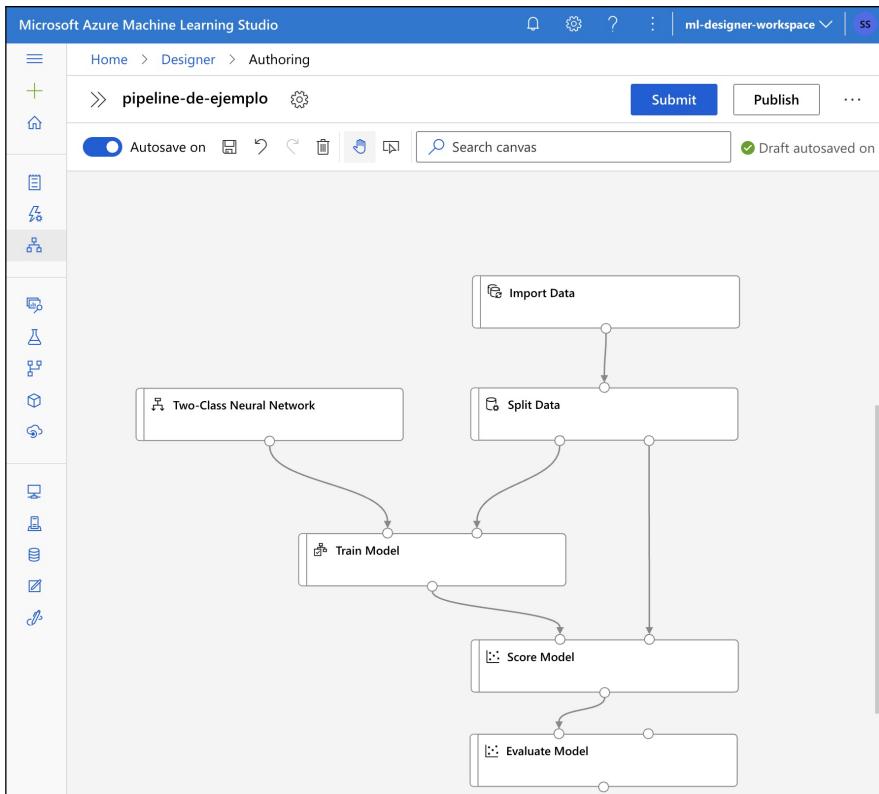


Figura 3.23: Evaluación de la precisión del modelo entrenado

En este momento, la *pipeline* ya estaría completada, por lo que restaría ejecutarla. Para ello, pulsaremos sobre el botón **Submit** y crearemos un nuevo experimento completando los campos (ver Figura 3.24). Durante este paso, podremos ver cómo se va ejecutando la *pipeline* visualmente en cada nodo hasta terminar.

Una vez haya finalizado el proceso, podremos explorar las métricas generadas pulsando con el click derecho del ratón sobre el nodo **Score Model** y seleccionando la opción **Preview data >Scored dataset**. Si desplazamos la tabla hasta la izquierda, podremos ver las columnas **Scored Labels** y **Scored Probabilities**, o lo que es lo mismo, la etiqueta que obtendrá el modelo para los datos de prueba y la probabilidad de que dicha predicción sea correcta, respectivamente (ver Figura 3.25).

Por último, podremos ver los resultados de la evaluación repitiendo el proceso anterior pero esta vez sobre el nodo **Evaluate Model**. Aquí, podremos ver información sobre la precisión del modelo y la matriz de confusión, entre otros (ver Figura 3.26).

Para finalizar, y si los datos de la evaluación son adecuados, podríamos desplegar nuestra *pipeline* para empezar a realizar predicciones sobre nuevos datos.

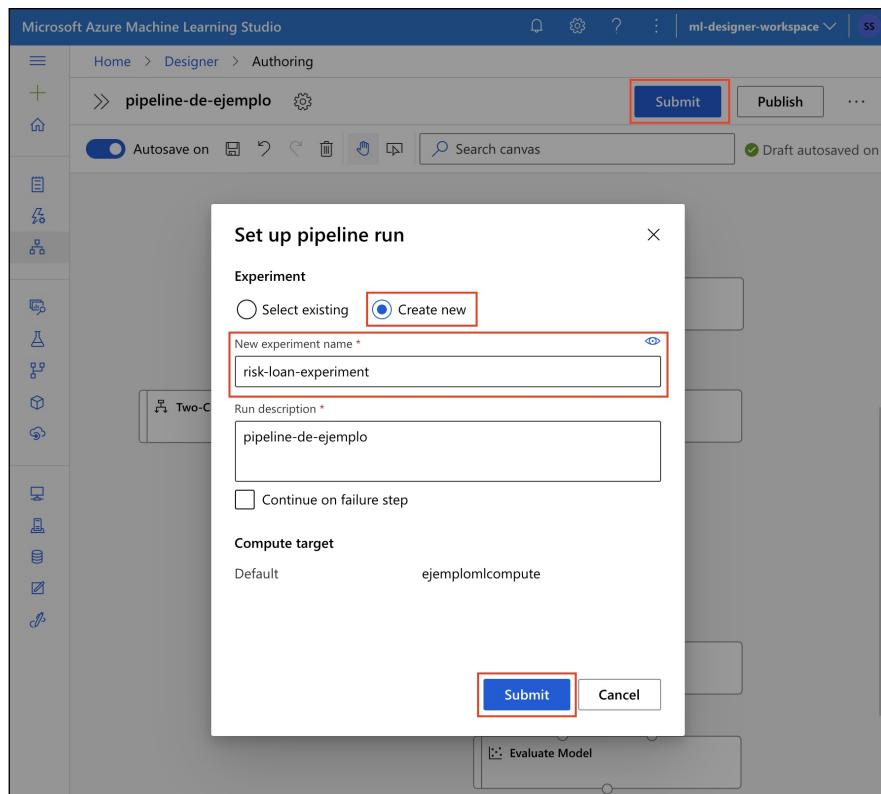


Figura 3.24: Ejecución de la *pipeline*

Caso práctico: Azure Machine Learning Automated ML

La herramienta de modelado Azure Machine Learning Designer nos ha permitido realizar todo el proceso de aprendizaje automático de una manera simple y rápida sin tener que escribir ninguna línea de código. Sin embargo, su flexibilidad obliga al usuario a tomar ciertas decisiones, como la elección del algoritmo de aprendizaje automático o el ajuste de hiperparámetros. Como se ha visto anteriormente, las herramientas de AutoML tratan de reducir esta complejidad abstraéndonos de ello. Vamos a ver cómo lo consigue la plataforma de Azure Machine Learning mediante su herramienta Automated ML¹⁰.

En primer lugar nos dirigiremos al portal de Microsoft Azure Machine Learning Studio (Enlace: <https://ml.azure.com/>), seleccionaremos la entrada de **Automated ML** del menú lateral izquierdo y pulsaremos sobre el botón **New Automated ML run** para iniciar el proceso de aprendizaje automático mediante AutoML (ver Figura 3.27).

¹⁰ Enlace: <https://azure.microsoft.com/es-es/services/machine-learning/automatedml/>

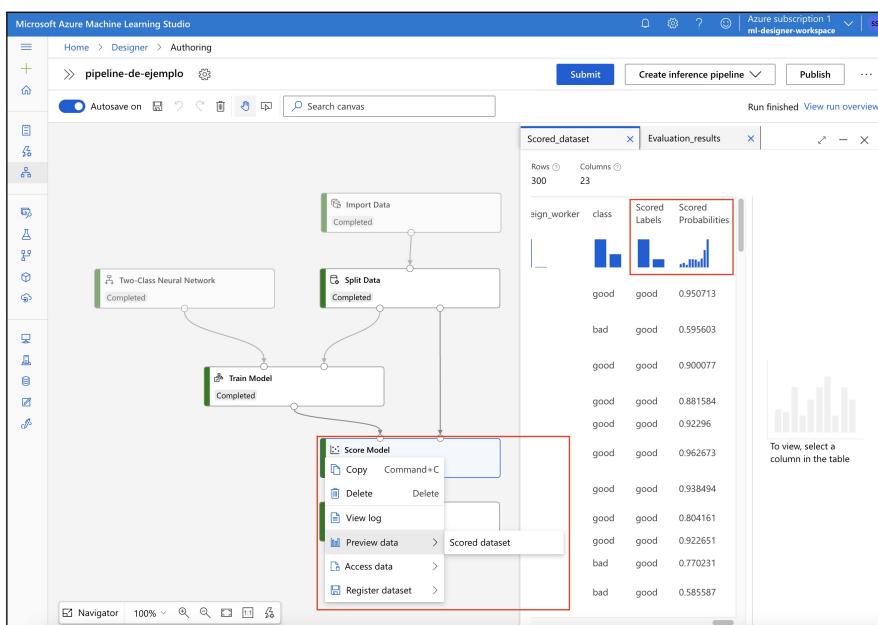


Figura 3.25: Predicciones obtenidas para los datos de prueba

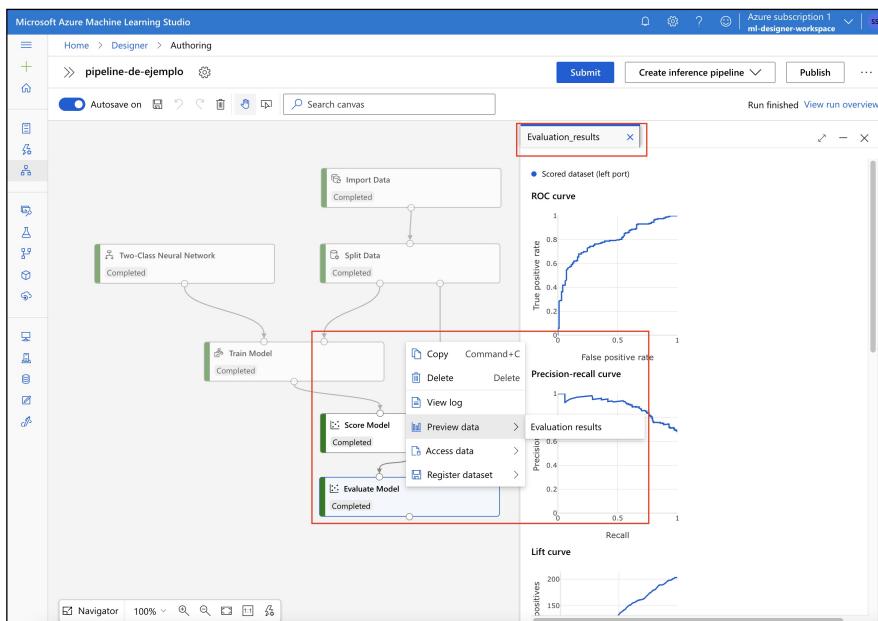


Figura 3.26: Detalles sobre la evaluación del modelo

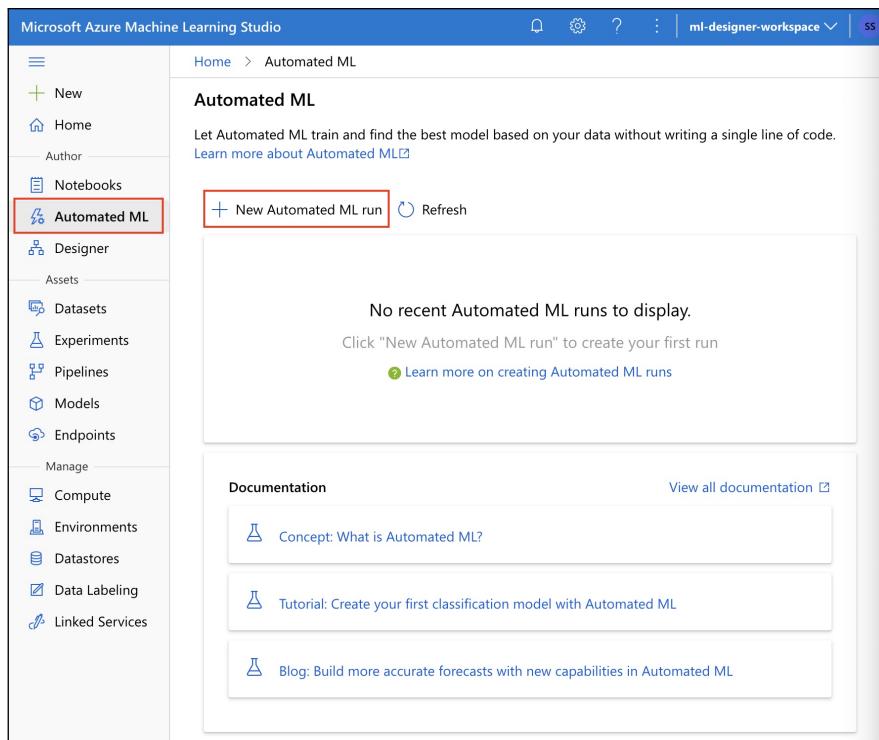


Figura 3.27: Inicio de la herramienta Automated ML

En la nueva sección, seleccionaremos el *dataset* del riesgo de conceder un préstamo. Si hemos seguido el caso práctico anterior, podremos seleccionarlo directamente, ya que se encontrará importado (ver Figura 3.28). En cualquier otro caso, tendremos que importarlo pulsando sobre el botón **Create dataset >From local files**.

En el siguiente paso configuraremos el experimento que utilizaremos para realizar las ejecuciones, la columna del *dataset* sobre la que realizaremos las predicciones (*class*) y la instancia que se ocupará de realizar el entrenamiento del modelo. Si tenemos un experimento y una instancias ya creados (por ejemplo, por haber seguido los pasos del caso práctico anterior), podemos reutilizarlos de nuevo sin tener que crear nuevos recursos (ver Figura 3.29).

Antes de completar el experimento, tendremos que seleccionar el tipo de problema al que nos enfrentamos (en nuestro caso, es un problema de clasificación) y algunos ajustes adicionales. Estos ajustes pueden modificarse a través del enlace **View additional configuration settings**, a destacar, la métrica sobre la que se tratará de optimizar el modelo, los algoritmos que no queremos evaluar, la duración del entrenamiento (en nuestro caso, 1 hora) y el porcentaje de los datos del *dataset* reservados para las pruebas (en nuestro caso, el 30 %) (ver Figura 3.30). Una vez configurado el experimento, pulsaremos sobre el botón **Finish**.

The screenshot shows the Microsoft Azure Machine Learning Studio interface. On the left, there's a sidebar with various options like 'New', 'Home', 'Author', 'Notebooks', 'Automated ML' (which is currently selected), 'Designer', 'Assets', 'Datasets', 'Experiments', 'Pipelines', 'Models', 'Endpoints', 'Manage', 'Compute', 'Environments', 'Datastores', 'Data Labeling', and 'Linked Services'. The main area is titled 'Create a new Automated ML run' and has a sub-section 'Select dataset'. It says 'Select an input dataset from the list below, or create a new dataset. Automated ML currently only supports tabular data for authoring runs.' There's a search bar, a refresh button, and a toggle switch for 'Show supported datasets only'. Below that, it says 'Showing 1-1 of 1 datasets'. A table lists one dataset: 'risk-loan-dataset' (selected with a checked checkbox), 'Tabular' (dataset type), and 'Oct 24, 2021' (created on). At the bottom are 'Back', 'Next', and 'Cancel' buttons.

Figura 3.28: Selección del *dataset* con el que trabajaremos



Si establecemos un tiempo de entrenamiento demasiado bajo la ejecución no será capaz de realizar el entrenamiento y por tanto fallará.

Una vez haya finalizado la ejecución, podremos explorar las diferentes pestañas para revisar los detalles. Por ejemplo, en la pestaña **Child runs** podremos ver las distintas ejecuciones que se han realizado para entrenar los modelos. En la pestaña **Models** podremos ver qué grado de precisión lograron cada uno de los modelos entrenados en las distintas ejecuciones. En la tabla podremos ver los algoritmos utilizados en el entrenamiento del modelo. Si pulsamos sobre alguno de ellos, podremos acceder a todas las métricas generadas durante su ejecución (ver Figura 3.31).

En la pestaña **Data guardrails** podremos ver cada una de las salvaguardias que automáticamente define la herramienta sobre las propiedades del *dataset* para asegurar la calidad de los datos de entrada (ver Figura 3.32).

Una vez haya finalizado el experimento y se haya encontrado el mejor modelo posible, en la pestaña **Details** podremos ver un resumen sobre los detalles del mejor modelo (ver Figura 3.33).

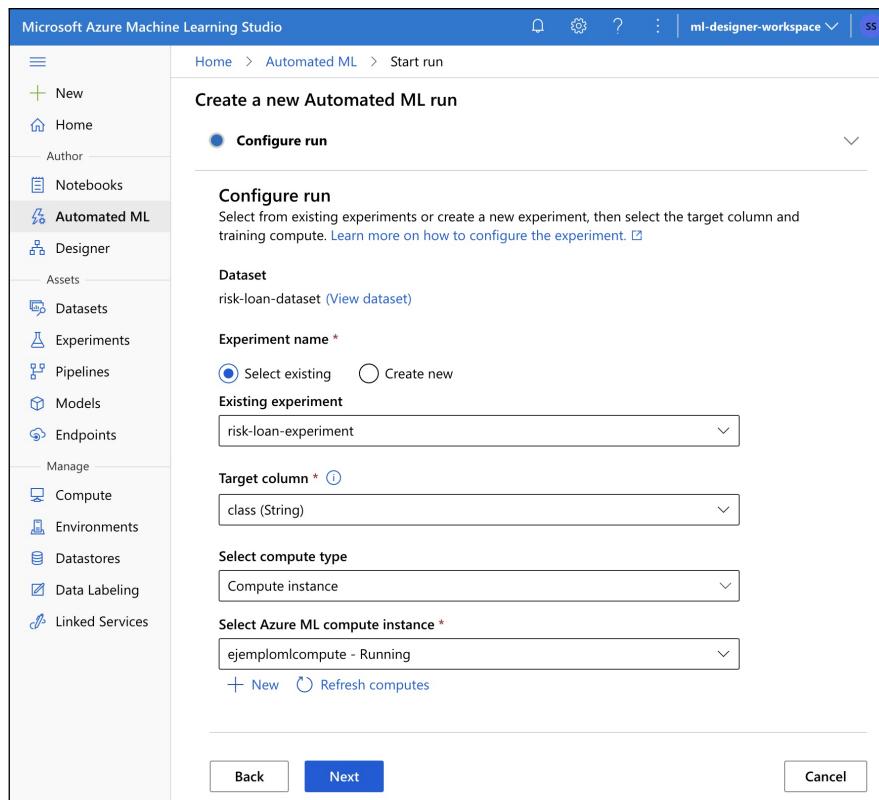


Figura 3.29: Configuración el experimento

Si pulsamos sobre el enlace de su nombre, seremos redirigidos a la vista completa de detalles del modelo. Aquí, podremos ver algunas explicaciones sobre el modelo en la pestaña **Explanations** (es decir, aquellas propiedades que tienen mayor impacto sobre las predicciones) y las métricas generadas en la pestaña **Metrics** que podremos utilizar para evaluar la precisión de nuestro modelo (ver Figuras 3.34 y 3.35).

Por último, y si el modelo se ajusta a nuestras necesidades, la herramienta nos permite desplegarlo en la nube de Google Cloud o descargarlo en formato PKL junto a una pequeña implementación en Python para utilizarlo sobre el entorno Anaconda.



Una vez se hayan completado todos los ejemplos, resulta conveniente eliminar cualquier recurso creado para no incurrir en posibles costes. Para ello, tendremos que eliminar el *workspace*, los *resource groups* que hayamos creado y los posibles despliegues que hayamos realizado.

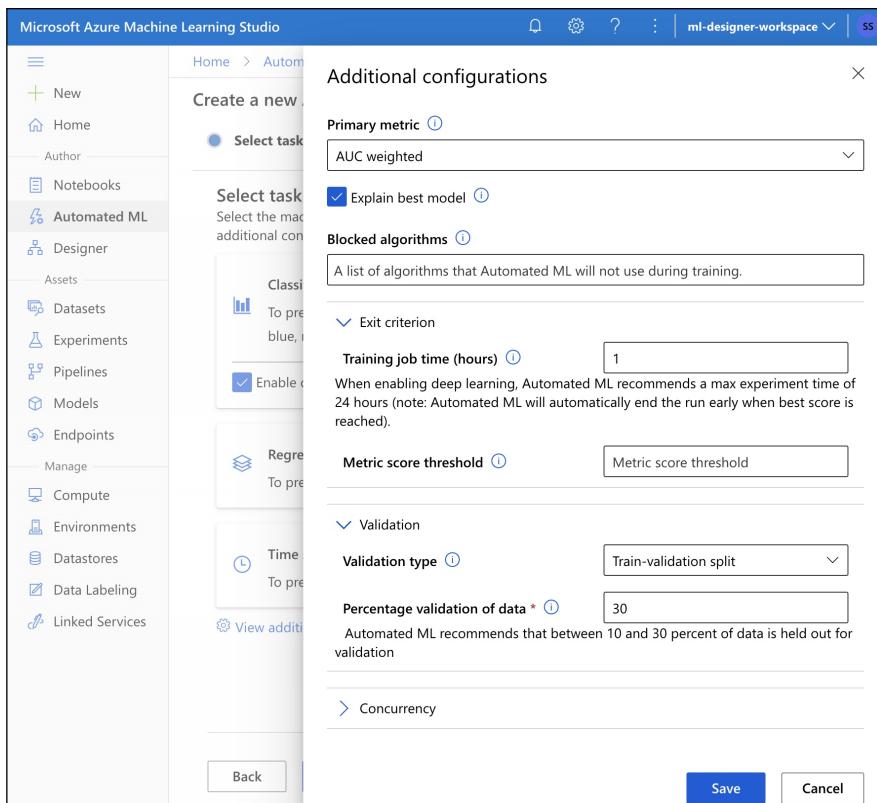


Figura 3.30: Ajustes adicionales del experimento

3.1.3. Amazon SageMaker

Para finalizar la sección, vamos a introducir la herramienta que nos proporciona Amazon para creación y entrenamiento de modelos de aprendizaje automático: **Amazon SageMaker**¹¹.

La herramienta se presenta como una plataforma de ML en la nube completamente gestionada, que proporciona funcionalidades de preparación de datos, desarrollo de modelos, entrenamiento, optimización y despliegue. La plataforma destaca además por **Amazon SageMaker Studio**, un entorno de desarrollo integrado orientado a la preparación, construcción y despliegue de modelos de aprendizaje automático.

¹¹ Enlace: <https://aws.amazon.com/sagemaker/>

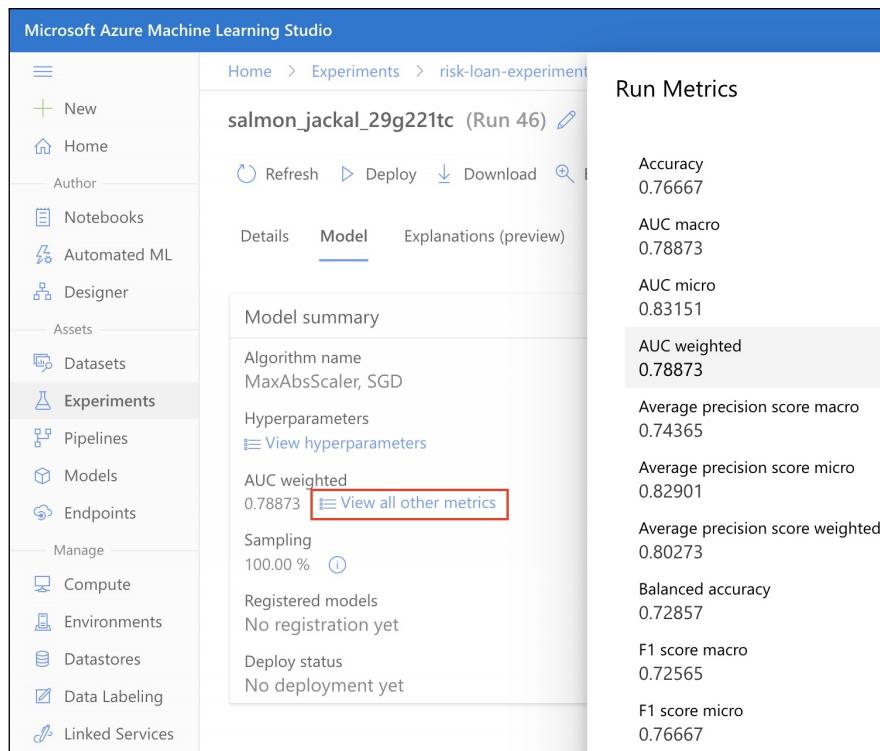


Figura 3.31: Métricas generadas para un modelo durante su entrenamiento

Entre los servicios que ofrece se encuentran los cuadernos de Jupyter en la nube, SageMaker Experiments para ejecutar experimentos similares a los que hemos visto en las otras plataformas, SageMaker Debugger para ayudar a inspeccionar parámetros y datos, SageMaker Model Monitor para monitorizar los modelos que tengamos desplegados en entornos de producción, SageMaker Neo para separar la infraestructura de entrenamiento de la de predicciones y SageMaker AutoPilot, que permite aplicar un enfoque de AutoML al ciclo de vida de ML, entre otros.

SageMaker AutoPilot¹² se diferencia del resto de herramientas que hemos visto hasta ahora en que la solución propuesta no es una *caja negra* que se ocupe de manera transparente de todo el proceso de ML hasta entrenar un modelo que se ajuste a nuestras necesidades. En este caso, la herramienta actúa como una *caja blanca*, exponiendo las *pipelines* utilizadas para entrenar los modelos resultantes.

¹² Enlace: <https://aws.amazon.com/sagemaker/autopilot>

The screenshot shows the Microsoft Azure Machine Learning Studio interface. The top navigation bar displays the path: Home > Automated ML > risk-loan-experiment > silver_school_z8c74rd7 (Run 8). The main content area is titled "silver_school_z8c74rd7 (Run 8)" with a pencil icon. Below the title are buttons for Refresh, Cancel, and Delete. A horizontal menu bar contains "Details", "Data guardrails" (which is underlined and bolded), "Models", "Outputs + logs", "Child runs", and "Snapshot". A descriptive text below the menu states: "Data guardrails are run by Automated ML when automatic featurization is enabled. This is a sequence of checks over the input data to ensure high quality data is being used to train model." Three data guardrail entries are listed in a table:

Type	Status	Description
Class balancing detection	Passed	Your inputs were analyzed, and all classes are balanced in your training data. Learn more about imbalanced data.
Missing feature values imputation	Passed	No feature missing values were detected in the training data. Learn more about missing value imputation.
High cardinality feature detection	Passed	Your inputs were analyzed, and no high cardinality features were detected. Learn more about high cardinality feature detection.

Figura 3.32: Salvaguardas definidas por la herramienta para garantizar la calidad de los datos de entrenamiento

Caso práctico: Amazon SageMaker AutoPilot

En este ejemplo construiremos un modelo de aprendizaje automático, utilizando la herramienta Amazon SageMaker AutoPilot, que nos permitirá predecir el nivel de riesgo de una solicitud de préstamo a partir del *dataset* que hemos venido utilizando hasta ahora.

Lo primero que haremos será dirigirnos a la consola de Amazon SageMaker en Enlace: <https://console.aws.amazon.com/sagemaker/home#dashboard> y abrir la herramienta SageMaker Studio pulsando sobre el botón (ver Figura 3.37).

En la nueva sección que se nos abrirá, seleccionaremos la opción **Quick start** para que la configuración de la herramienta en nuestra cuenta de Amazon se realice automáticamente, y crearemos un rol de ejecución para el usuario con las opciones por defecto (ver Figura 3.38). Finalmente, pulsaremos sobre el botón **Submit**.

Una vez finalizado el proceso inicial de configuración, podremos acceder a SageMaker Studio siguiendo el enlace **Open Studio** tal y como se muestra en la Figura 3.38.

The screenshot shows the Microsoft Azure Machine Learning Studio interface. The top navigation bar includes 'Home', 'Automated ML', 'risk-loan-experiment', and 'silver_school_z8c74rd7 (Run 8)'. Below the navigation is a toolbar with 'Refresh', 'Cancel', and 'Delete' buttons. A sidebar on the left contains various icons for managing experiments, runs, and datasets. The main content area has tabs for 'Details', 'Data guardrails', 'Models', 'Outputs + logs', 'Child runs', and 'Snapshot'. The 'Details' tab is selected, displaying the 'Properties' section. This section includes fields for 'Status' (Completed), 'Created' (Oct 24, 2021 2:06 PM), 'Started' (Oct 24, 2021 2:06 PM), 'Duration' (22m 31.32s), 'Compute duration' (22m 31.32s), 'Compute target' (ejemplomlcompute), 'Run ID' (AutoML_0a79e9ca-5912-414f-8ef0-a793c2ec3b5d), 'Script name' (--), 'Created by' (Santiago Sanchez), and 'Input datasets'. To the right of the properties is a red-bordered box containing the 'Best model summary' and 'Run summary' sections. The 'Best model summary' section lists the algorithm as 'MaxAbsScaler, SGD', hyperparameters (View hyperparameters), AUC weighted (0.78873, View all other metrics), Sampling (100.00 %), Registered models (No registration yet), and Deploy status (No deployment yet). The 'Run summary' section lists Task type (Classification, View configuration settings), Featurization (Auto), Primary metric (AUC weighted), and Experiment name (risk-loan-experiment).

Figura 3.33: Vista de resumen de los detalles del mejor modelo encontrado



Es posible que durante este proceso de configuración inicial podamos ver algún mensaje de error. Es completamente seguro ignorarlos, ya que suelen aparecer mientras se realiza el proceso de configuración debido a problemas de permisos que todavía no se han terminado de solucionar. Una vez termine el proceso, no volverán a aparecer.

La primera vez que accedamos a SageMaker Studio tardará unos minutos en finalizar los ajustes iniciales hasta que podamos acceder a la herramienta. Una vez haya cargado la herramienta, crearemos un nuevo cuaderno Jupyter sobre una imagen de SageMaker basada en **Data Science** (ver Figura 3.40).

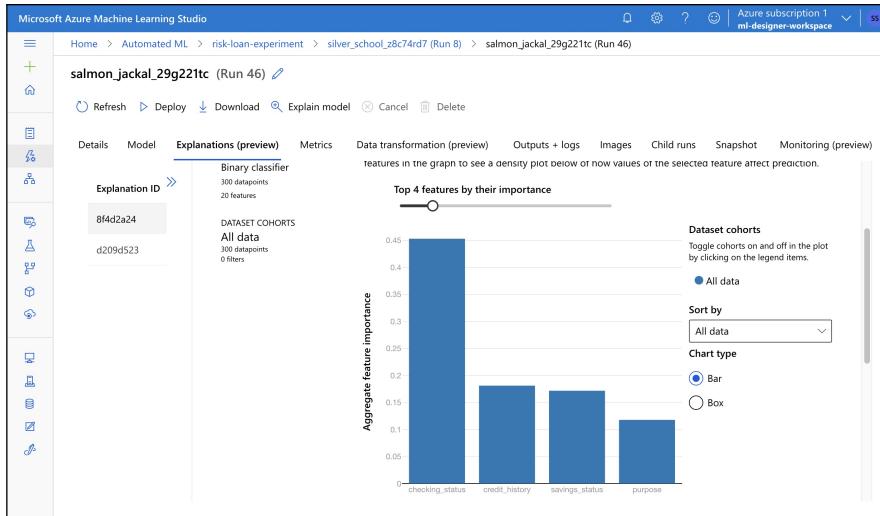


Figura 3.34: Explicación del modelo

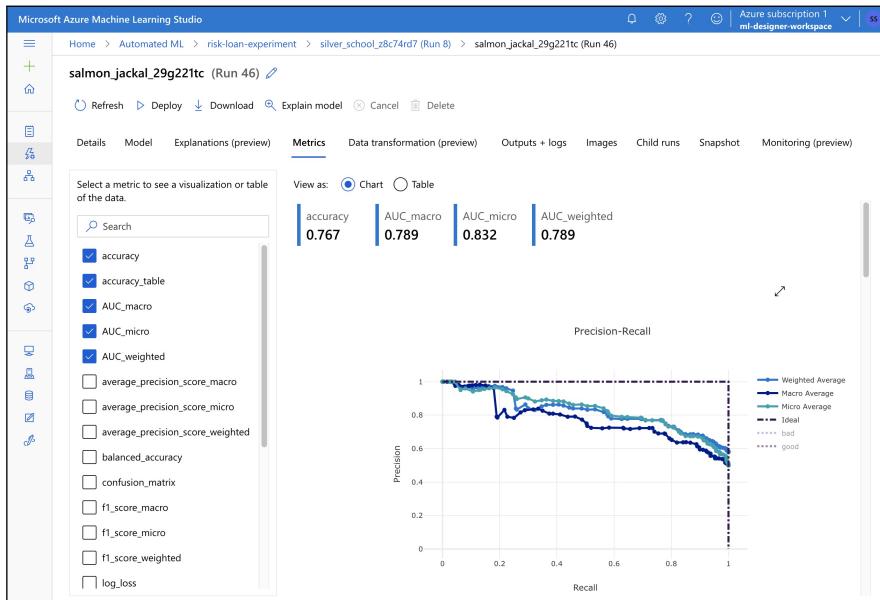


Figura 3.35: Métricas generadas

Una vez creado el cuaderno, podremos empezar a importar nuestro *dataset*. Para ello, simplemente tendremos que subir nuestro archivo CSV al sistema de archivos del entorno. Después, podremos utilizar pandas para cargarlo y previsualizar las primeras filas para confirmar que todo ha ido bien (ver Figura 3.41).

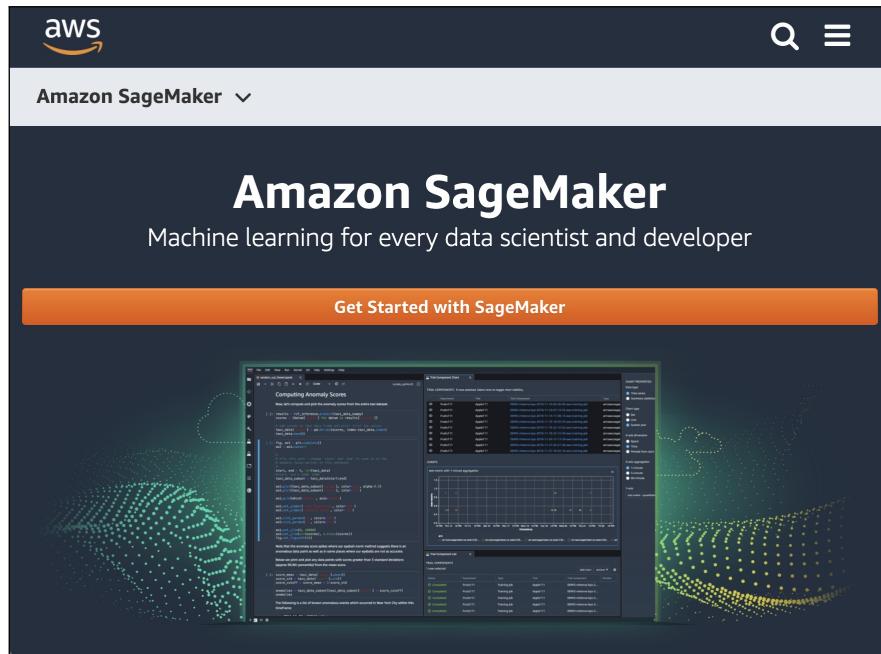


Figura 3.36: Página principal de Amazon SageMaker

The screenshot shows the Amazon SageMaker Dashboard. At the top, the AWS logo, Services dropdown, and navigation bar with 'Amazon SageMaker' and 'Dashboard' are visible. To the right is a search bar, a bell icon, and links for 'Sandbox', 'London', and 'Support'. The main area is divided into sections: 'Dashboard' (highlighted in orange), 'Search', 'SageMaker Domain', 'Studio', 'Images', and a sidebar with links like 'Ground Truth', 'Notebook', 'Processing', 'Training', 'Inference', 'Augmented AI', and 'AWS Marketplace'. The 'Dashboard' section has a 'Overview' card with four icons: 'Prepare' (cloud with gear), 'Build' (cloud with document), 'Train & Tune' (cloud with neural network), and 'Deploy & Manage' (laptop with person). Below this is a 'Studio' section with a 'Open Studio' button and a table of services: Data Wrangler, Studio Notebooks, One-click Training, One-click Deployment; Processing, Feature Store, Built-in and Bring-your-own Algorithms, Experiments, Multi-Model Endpoints; Clarify, Clarify, Autopilot, Automatic Model Tuning, Model Monitor; Inference, JumpStart, Debugger, Managed Spot Training, Pipelines; Augmented AI, Post-processing Job.

Figura 3.37: Dashboard de Amazon SageMaker

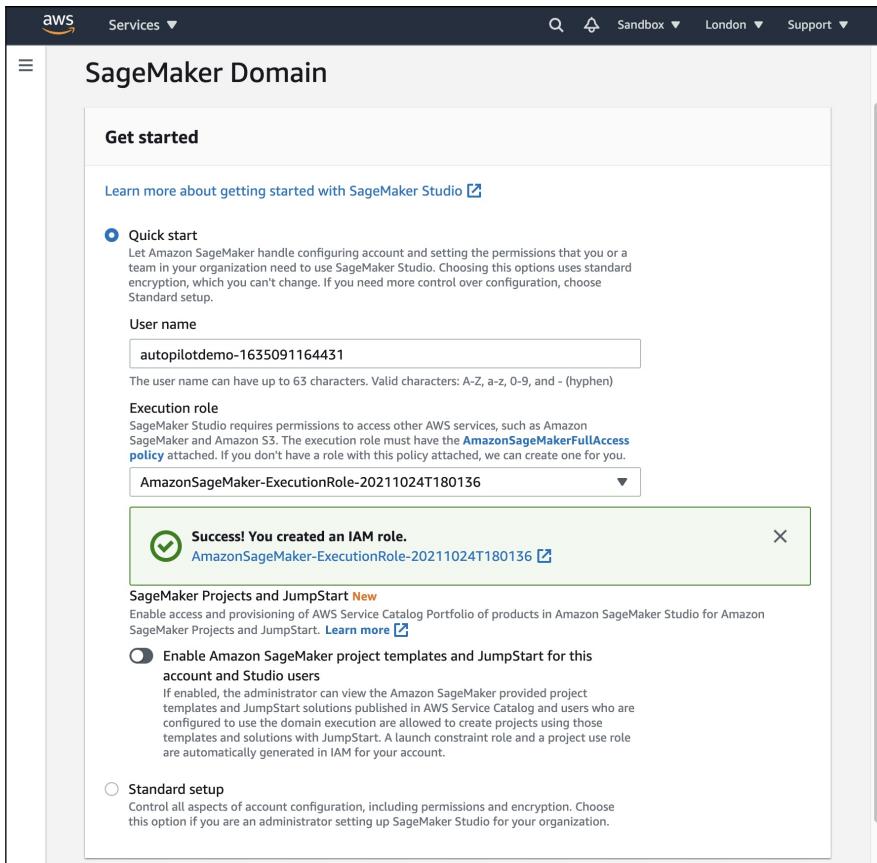


Figura 3.38: Configuración inicial del servicio de Amazon SageMaker para la cuenta del usuario

A continuación, utilizaremos NumPy para dividir el *dataset* en un conjunto de entrenamiento (70 %) y un conjunto de prueba (30 %). Estos dos conjuntos de datos los almacenaremos en el sistema de archivo. Después, crearemos una sesión utilizando la API de SageMaker para subir los datos de entrenamiento a un contenedor S3 (ver Figura 3.42).

Una vez tengamos los datos de entrenamiento disponibles en nuestro contenedor, podremos pasar a crear el experimento de AutoML. Para ello, seleccionaremos la opción **File >New >Experiment** de la barra de menús y ajustaremos las opciones del experimento. El lector se debería sentir más que familiarizado con todas las opciones que se muestran en el asistente, a destacar, la ruta al *dataset* de entrenamiento que acabamos de cargar en el contenedor, la propiedad sobre la que realizaremos las predicciones (en nuestro caso, *class*), el contenedor y la ruta donde se almacenarán los resultados de la ejecución, el tipo de problema que se pretende solucionar (en nuestro caso, un problema de clasificación binaria, aunque podemos dejar la opción **Auto** para que la propia herramienta lo infiera) y el tiempo máximo

Figura 3.39: Configuración inicial del servicio y acceso a SageMaker Studio disponible

del experimento, entre otras. Las Figuras 3.43, 3.44 y 3.45 muestran la configuración seleccionada para este ejemplo. Nótese cómo la opción de auto-desplegar el modelo ha sido desactivada; esto lo haremos manualmente en caso de que la evaluación del modelo se ajuste a nuestras necesidades.

Confirmaremos los ajustes y se iniciará la ejecución del experimento. Este proceso puede durar como máximo 1 hora, dependiendo del tiempo máximo que hayamos indicado en los ajustes del experimento. Mientras el experimento finaliza, podremos ver algunos detalles como, por ejemplo, el tipo de problema que ha identificado la herramienta y las pruebas ejecutadas con distintos modelos (pestaña **Trials**) (ver Figura 3.46).

Una vez haya finalizado el proceso, veremos cuál es el modelo que ha logrado la mejor puntuación en la métrica objetivo, denotado con la etiqueta **Best model** (ver Figura 3.47). Si exploramos sus detalles (click derecho >**Open in model details**), podremos ver algunos detalles como el algoritmo de entrenamiento y las propiedades que mayor impacto han tenido sobre la propiedad objetivo (ver Figura 3.48).

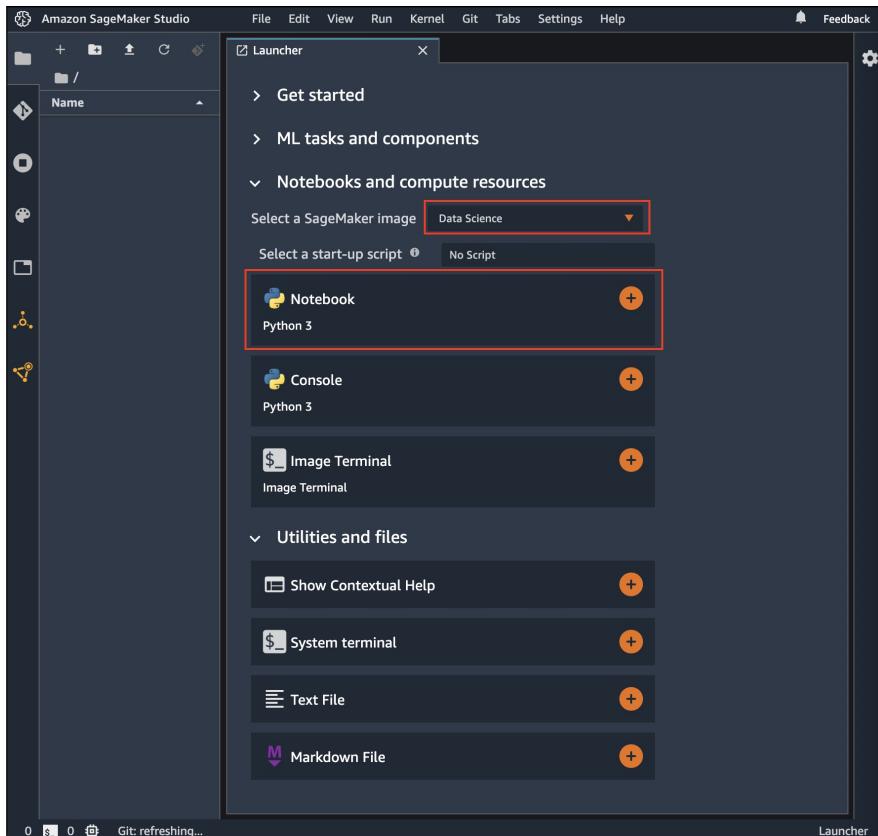


Figura 3.40: Creación de un cuaderno Jupyter en SageMaker Studio basado en una imagen de *Data Science*



La herramienta nos permite revisar el análisis del *dataset* realizado mediante el acceso al cuaderno denominado **Data exploration notebook**. Del mismo modo, el cuaderno **Candidate definition**, nos permite revisar los pasos que se han realizado durante el experimento.

Una vez entrenado el modelo, podemos desplegarlo pulsando sobre el botón **Deploy model**. En la nueva pestaña, seleccionaremos el apartado **Real Time Predictions** para construir un *endpoint* que nos permita realizar predicciones de datos sobre el modelo mediante peticiones HTTP. El resto de opciones las configuraremos a nuestra preferencia y pulsaremos sobre el botón **Deploy model** (ver Figura 3.49). En la pestaña **AWS settings** de la nueva sección, podremos ir viendo el estado en el que se encuentra la creación del *endpoint*.

The screenshot shows the Amazon SageMaker Studio interface. On the left, the file browser displays a folder named 'BigML_Dataset_Loan...' and a file named 'Untitled.ipynb'. A red arrow labeled '1' points to the upload icon in the top toolbar. The main area shows a Jupyter notebook cell with the code:

```
[2]: import pandas as pd  
data = pd.read_csv('BigML_Dataset_Loan_Risk_Data.csv')  
data.describe()  
data[:10]
```

A red box highlights the output of the last two lines, which is a table showing the first 10 rows of the dataset. Red arrows labeled '2' and '3' point to the play button and the table respectively. The table has columns: checking_status, duration, credit_history, purpose, credit_amount, and savings. The data includes various categories like 'no checking', 'existing paid', and 'radio/tv', along with numerical values for duration and credit amount.

Figura 3.41: 1) Importación de los datos del *dataset* y 2,3) previsualización de las primeras filas con *pandas*

Una vez se haya desplegado el *endpoint*, podremos realizarle peticiones utilizando nuestro cuaderno Jupyter del principio. Para ello, añadiremos una nueva celda de código, crearemos una sesión usando la API de SageMaker y le enviaremos peticiones con los datos de prueba que generamos al comienzo del experimento. En la Figura 3.50 puede verse el código completo y el resultado de la ejecución. En nuestro caso, mostramos la petición completa (sin la columna **class**), la propiedad **class** (utilizada para compararla con la predicción realizada) y la predicción obtenida como respuesta.

The screenshot shows the Amazon SageMaker Studio interface. On the left, there's a file browser pane with a tree view of datasets. The main area is a Jupyter notebook cell containing Python code. The code performs the following steps:

- Imports numpy.
- Splits the dataset into train_data and test_data using np.split with a 70/30 split ratio and random_state=123.
- Saves train_data to 'BigML_Dataset_Loan_Risk_Data_train.csv' with index=False, header=True, and sep=','.
- Saves test_data to 'BigML_Dataset_Loan_Risk_Data_test.csv' with index=False, header=True, and sep=','.
- Imports sagemaker.
- Creates a Session.
- Uploads the training data to S3 with the key_prefix 'sagemaker/loan-risk-data/input/BigML_Dataset_Loan_Risk_Data_train.csv'.

The status bar at the bottom indicates the session is idle, the kernel is idle, and the instance has 2 vCPU + 4 GiB of memory.

Figura 3.42: División del *dataset* en el conjunto de entrenamiento y pruebas, y subida de los datos de entrenamiento a un contenedor S3

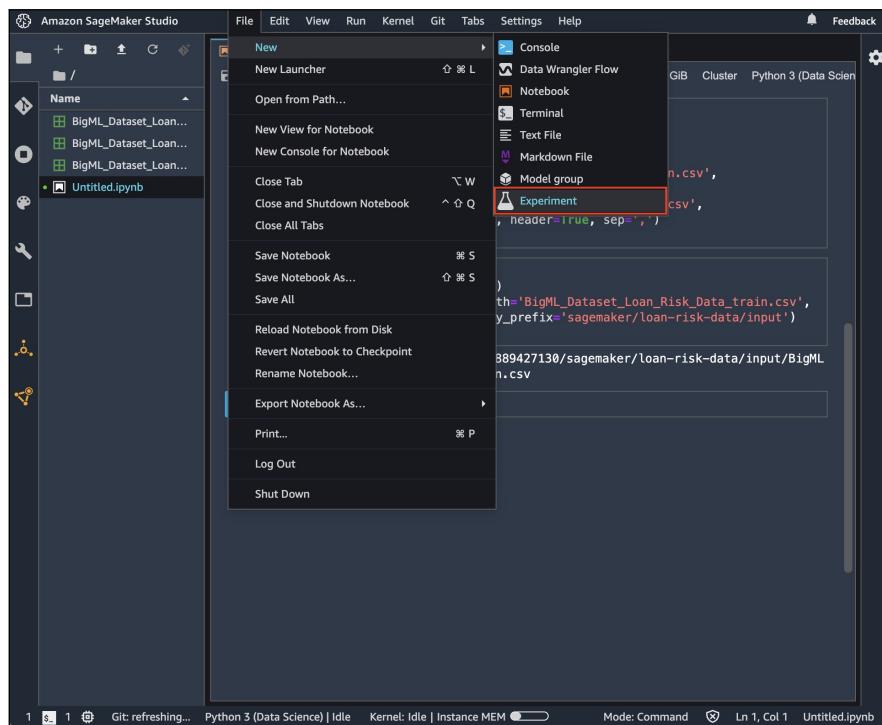


Figura 3.43: Creación del experimento de AutoPilot

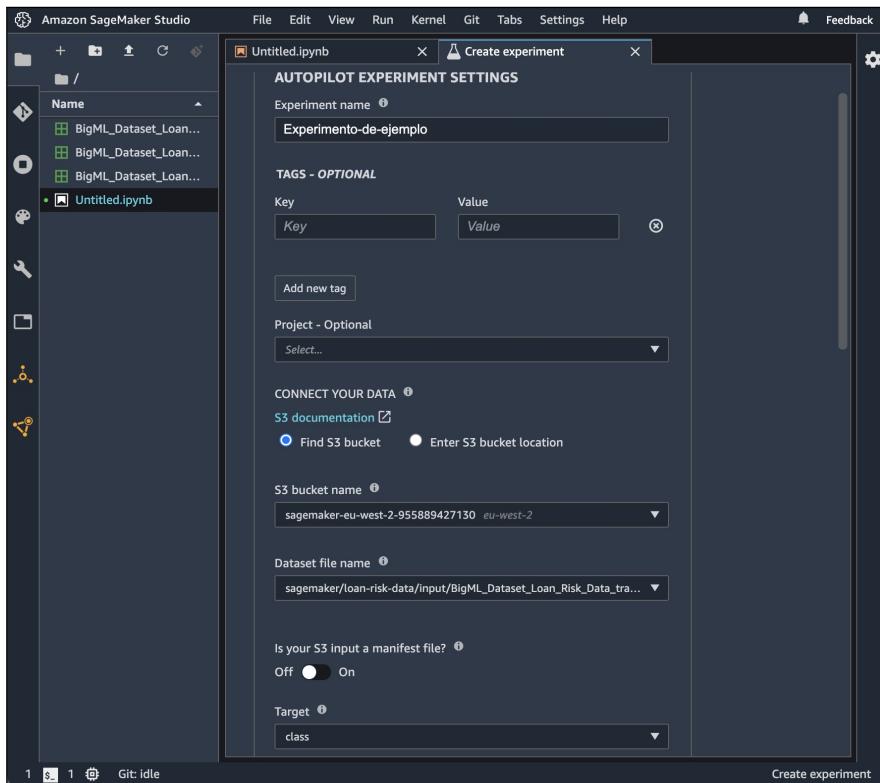


Figura 3.44: Configuración del experimento (1/2)

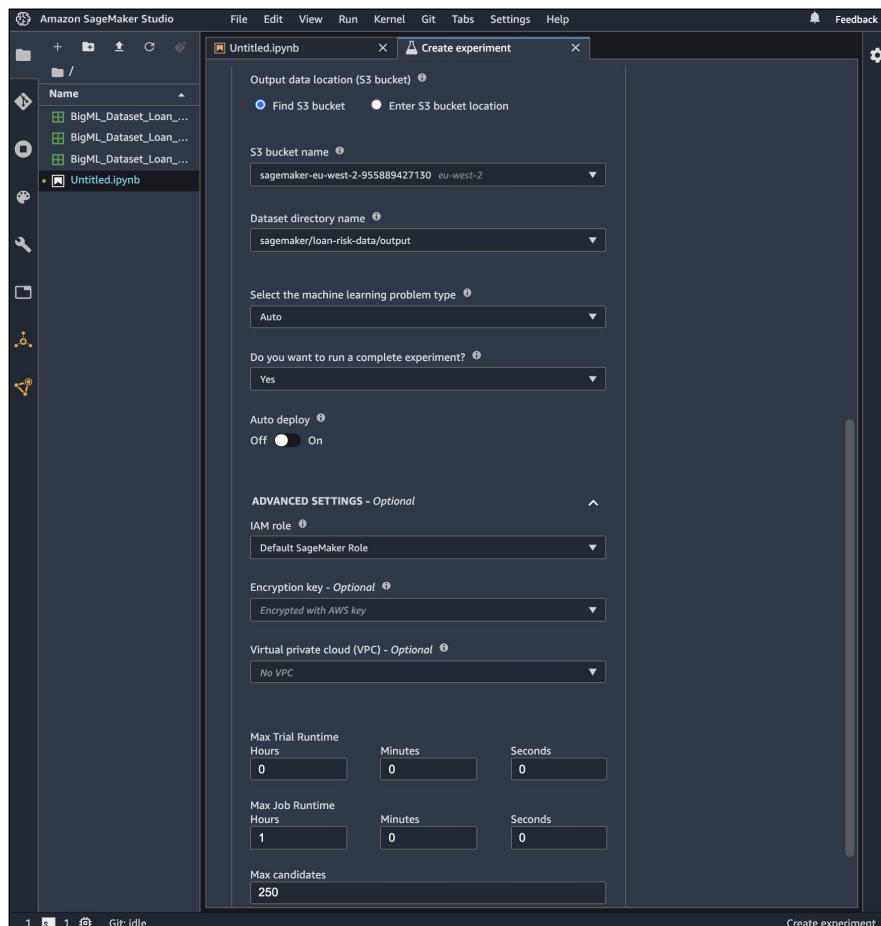


Figura 3.45: Configuración del experimento (2/2)

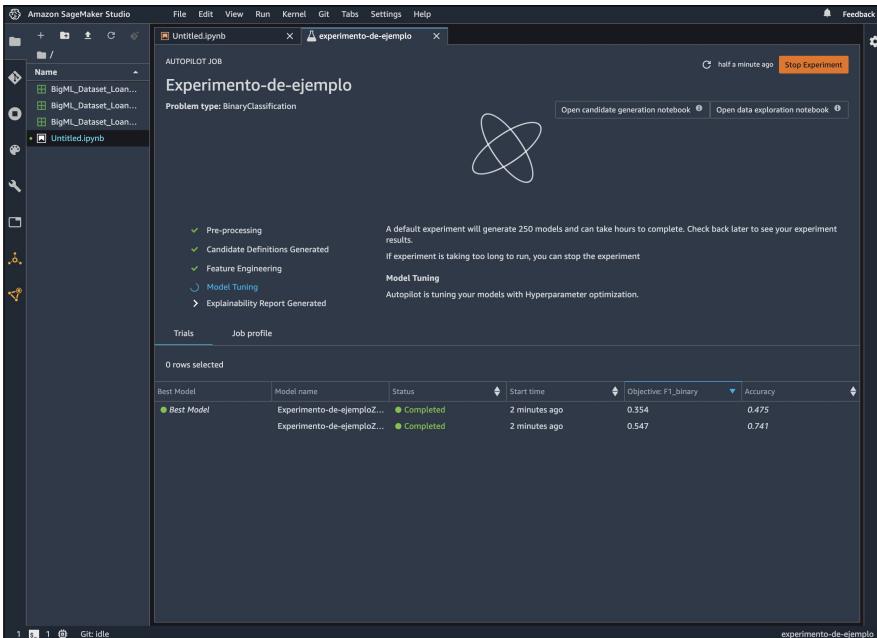


Figura 3.46: Ejecución del experimento

The screenshot shows the Amazon SageMaker Studio interface. The top navigation bar includes File, Edit, View, Run, Kernel, Git, Tabs, Settings, and Help. A central workspace shows an Untitled.ipynb file and an experiment named 'experimento-de-ejemplo'. The experiment details indicate it's a BinaryClassification problem type, last updated 9 minutes ago, with a Deploy model button. The 'Trials' tab is active, showing a table of 30 trials. The first trial, labeled 'Best Model', has its status set to 'Deploy model' and the 'Open in model details' button highlighted with a red box. The table columns include Best Model, Model name, Status, Start time, Objective: F1_binary, and Accuracy. The accuracy values range from 0.532 to 0.746.

Best Model	Model name	Status	Start time	Objective: F1_binary	Accuracy
● Best Model	Experimento-de-ejemplo...	Deploy model	9 mins ago	0.562	0.746
	Experimento-de-ejemplo...	Open in model details	9 mins ago	0.559	0.741
	Experimento-de-ejemplo...	Copy cell contents	9 mins ago	0.557	0.736
	Experimento-de-ejemplo...	Shift+Right Click for Browser Menu	9 mins ago	0.547	0.741
	Experimento-de-ejemplo...	● Completed	2 hours ago	0.546	0.744
	Experimento-de-ejemplo...	● Completed	2 hours ago	0.545	0.76
	Experimento-de-ejemplo...	● Completed	2 hours ago	0.544	0.751
	Experimento-de-ejemplo...	● Completed	2 hours ago	0.543	0.754
	Experimento-de-ejemplo...	● Completed	2 hours ago	0.542	0.736
	Experimento-de-ejemplo...	● Completed	2 hours ago	0.539	0.746
	Experimento-de-ejemplo...	● Completed	2 hours ago	0.538	0.751
	Experimento-de-ejemplo...	● Completed	2 hours ago	0.538	0.733
	Experimento-de-ejemplo...	● Completed	2 hours ago	0.538	0.744
	Experimento-de-ejemplo...	● Completed	2 hours ago	0.538	0.739
	Experimento-de-ejemplo...	● Completed	2 hours ago	0.537	0.733
	Experimento-de-ejemplo...	● Completed	2 hours ago	0.537	0.753
	Experimento-de-ejemplo...	● Completed	3 hours ago	0.536	0.731
	Experimento-de-ejemplo...	● Completed	2 hours ago	0.536	0.743
	Experimento-de-ejemplo...	● Completed	2 hours ago	0.535	0.74
	Experimento-de-ejemplo...	● Completed	2 hours ago	0.535	0.75
	Experimento-de-ejemplo...	● Completed	2 hours ago	0.535	0.734
	Experimento-de-ejemplo...	● Completed	2 hours ago	0.534	0.75
	Experimento-de-ejemplo...	● Completed	3 hours ago	0.533	0.753
	Experimento-de-ejemplo...	● Completed	2 hours ago	0.532	0.747

Figura 3.47: Mejor modelo entrenado

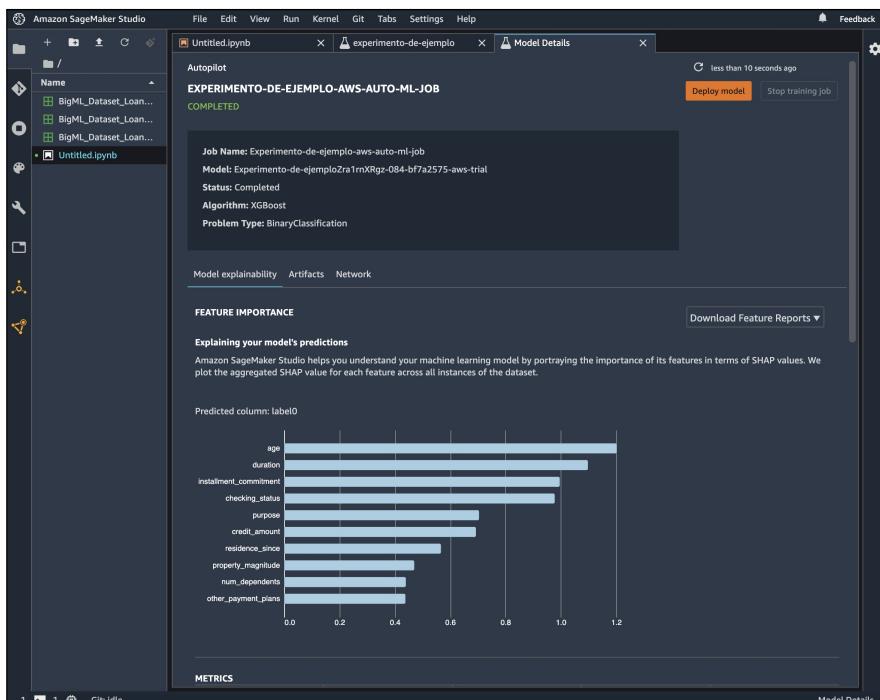


Figura 3.48: Detalles del modelo

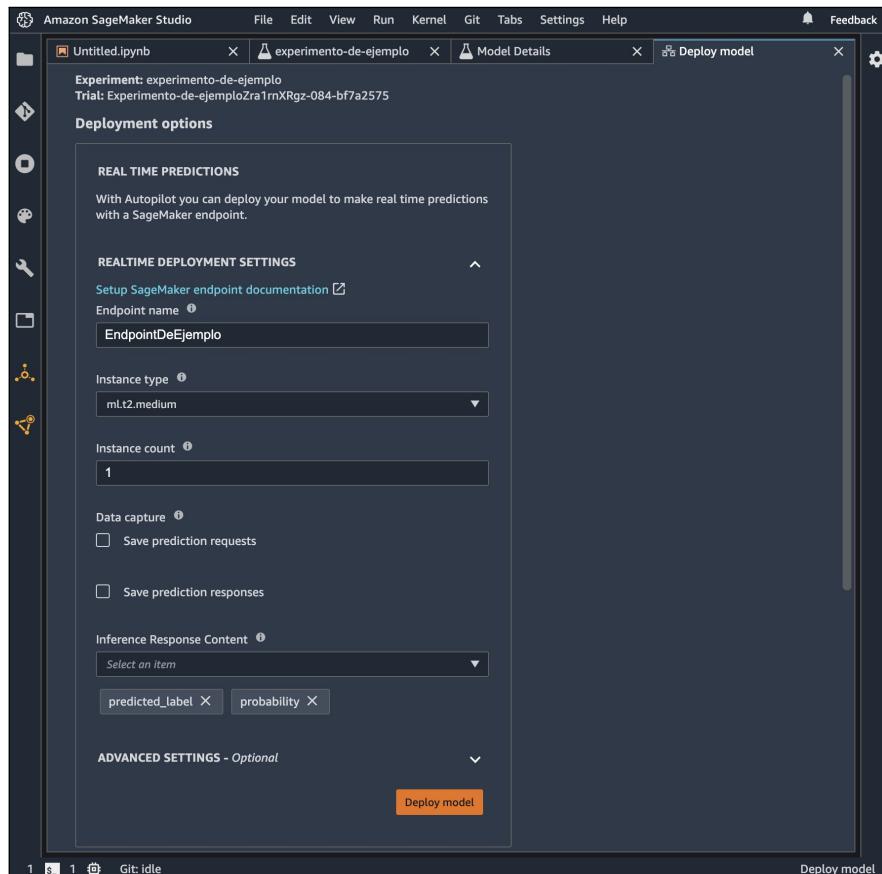
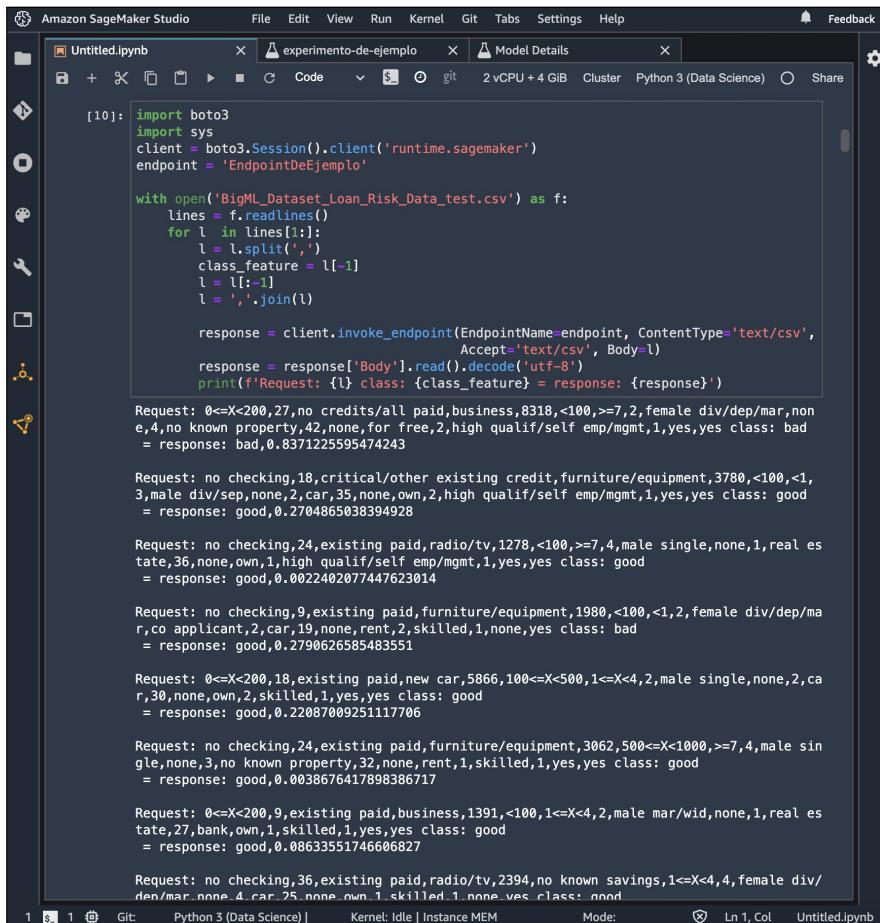


Figura 3.49: Despliegue del modelo



The screenshot shows a Jupyter notebook cell in Amazon SageMaker Studio. The cell contains Python code that reads a CSV file, processes its rows, and sends them to an endpoint named 'EndpointDeEjemplo'. The code uses the boto3 library to interact with the SageMaker runtime. The output of the cell shows several requests being sent to the endpoint, each with specific feature values and a classification result ('good' or 'bad') and a confidence score.

```
import boto3
import sys
client = boto3.Session().client('runtime.sagemaker')
endpoint = 'EndpointDeEjemplo'

with open('BigML_Dataset_Loan_Risk_Data_test.csv') as f:
    lines = f.readlines()
    for l in lines[1:]:
        l = l.split(',')
        class_feature = l[-1]
        l = l[:-1]
        l = ','.join(l)

        response = client.invoke_endpoint(EndpointName=endpoint, ContentType='text/csv',
                                           Accept='text/csv', Body=l)
        response = response['Body'].read().decode('utf-8')
        print(f'Request: {l} class: {class_feature} = response: {response}')


Request: 0<=X<200,27,no credits/all paid,business,8318,<100,>=7,2,female div/dep/mar,non e,4,no known property,42,none,for free,2,high qualif/self emp/mgmt,1,yes,yes class: bad = response: bad,0.8371225595474243

Request: no checking,18,critical/other existing credit,furniture/equipment,3780,<100,<1,3,ma le div/sep,none,2,car,35,none,own,2,high qualif/self emp/mgmt,1,yes,yes class: good = response: good,0.2704865038394928

Request: no checking,24,existing paid,radio/tv,1278,<100,>=7,4,ma le single,none,1,real es tate,36,none,own,1,high qualif/self emp/mgmt,1,yes,yes class: good = response: good,0.0022402077447623014

Request: no checking,9,existing paid,furniture/equipment,1980,<100,<1,2,female div/dep/ma r,co applicant,2,car,19,none,rent,2,skilled,1,none,yes class: bad = response: good,0.2790626585483551

Request: no checking,18,existing paid,new car,5866,100<=X<500,1<=X<4,2,ma le single,none,2,ca r,30,none,own,2,skilled,1,yes,yes class: good = response: good,0.22087009251117706

Request: no checking,24,existing paid,furniture/equipment,3062,500<=X<1000,>=7,4,ma le sin gle,none,3,no known property,32,none,rent,1,skilled,1,yes,yes class: good = response: good,0.0038676417898386717

Request: 0<=X<200,9,existing paid,business,1391,<100,1<=X<4,2,ma le mar/wid,none,1,real es tate,27,bank,own,1,skilled,1,yes,yes class: good = response: good,0.08633551746606827

Request: no checking,36,existing paid,radio/tv,2394,no known savings,1<=X<4,4,female div/den/mar,none,4,car,25,none,own,1,skilled,1,none,yes class: good = response: good,0.0038676417898386717
```

Figura 3.50: Solicitando predicciones al *endpoint* con el modelo desplegado

3.2. Bibliotecas de modelado de IA

Existen multitud de bibliotecas que podemos utilizar para dotar a nuestras aplicaciones de comportamiento inteligente. La mayoría de ellas serán aplicables en función del contexto del problema que se pretenda resolver, y otras actuarán como dependencias de otras para construir bibliotecas más completas.

Así, podemos clasificar las bibliotecas de IA en aquellas utilizadas en el campo del i) aprendizaje automático, ii) procesamiento del lenguaje natural, iii) redes neuronales, iv) visión por computador, v) sistemas expertos y vi) robótica, entre otros:

- Aprendizaje automático:

- **scikit-learn**¹³: biblioteca que facilita la implementación de algoritmos de aprendizaje automático. Incluye algoritmos para clasificar objetos, construir regresiones, agrupamiento de objetos similares (*clustering*), preprocesamiento de datos y selección automática de modelos. La biblioteca está basada en *NumPy*¹⁴, *SciPy*¹⁵ y *matplotlib*¹⁶.
- **TensorFlow**¹⁷: *framework* creado por Google para facilitar el uso de algoritmos complejos de aprendizaje automático y aprendizaje profundo basados en ANN. Los desarrolladores crean flujos de datos en los que cada nodo (o «neurona») representa un cálculo concreto especificado por el programador, y se elige entre uno de los muchos algoritmos de aprendizaje automático/profundo ya implementados en la biblioteca TensorFlow para ejecutarlo. Todo esto se facilita mediante la API de alto nivel para aprendizaje profundo llamada **Keras**¹⁸.
- **XGBoost**¹⁹: XGBoost son las siglas de «eXtreme Gradient Boosting». Esta biblioteca se centra en ayudar a los desarrolladores a clasificar datos y construir regresiones utilizando algoritmos de árboles de decisión potenciados. Estos árboles están formados por hijos de modelos de regresión más débiles (que representan diferentes tareas de cálculo). A medida que se entrena el modelo, se añaden nuevos modelos de regresión más débiles para *rellenar los huecos* hasta que no se puedan hacer más mejoras.

- Procesamiento del lenguaje natural:

¹³ Enlace: <https://scikit-learn.org/>

¹⁴ Enlace: <https://numpy.org/>

¹⁵ Enlace: <https://www.scipy.org/>

¹⁶ Enlace: <https://matplotlib.org/>

¹⁷ Enlace: <https://www.tensorflow.org/>

¹⁸ Enlace: <https://keras.io/>

¹⁹ Enlace: <https://xgboost.readthedocs.io/>

- **NLTK**²⁰: NLTK son las siglas de «Natural Language Toolkit». Es una biblioteca que simplifica el uso de la lengua gracias a una serie de funciones e interfaces definidas. Desde la *tokenización* y el etiquetado de texto, hasta la identificación de entidades con nombre e incluso la visualización de árboles de análisis sintáctico, NLTK es una biblioteca de PLN de propósito general que encaja en cualquier proyecto basado en el análisis del lenguaje.
- **spaCy**²¹: esta biblioteca hace, a través de su API extremadamente sencilla, que el procesamiento de grandes cantidades de texto sea rápido y eficiente. Al proporcionar e integrar el tokenizador, el etiquetador, el analizador sintáctico, los vectores de palabras preentrenados y las funciones de reconocimiento de entidades con nombre en una sola biblioteca, spaCy puede ayudar a los programas a comprender todos los aspectos de un texto, o simplemente a preprocesarlo para que alguna de las otras bibliotecas de IA se ocupe de ello posteriormente.
- **Gensim**²²: el objetivo de Gensim es facilitar el proceso de identificación del tema subyacente de un texto (conocido como *modelado de temas*). Se encarga de todo el proceso de modelización, desde el procesamiento del texto (en un diccionario de tokens) hasta la construcción del propio modelo temático, todo ello sin tener que cargar todo el texto en la memoria.

■ Redes neuronales:

- **FANN**²³: Fast Artificial Neural Network Library, o FANN, implementa redes neuronales artificiales en C (lo que hace que sea hasta 150 veces más rápida que otras bibliotecas), al mismo tiempo que las hace accesibles en distintos lenguajes, incluido Python. Es increíblemente fácil de usar, ya que permite crear, entrenar y ejecutar una red neuronal artificial con sólo tres llamadas a funciones.
- **PyTorch**²⁴: esta biblioteca está construida para tareas de cálculo de *tensores* (utilizando la aceleración de la GPU) y la construcción de redes neuronales profundas más duraderas, haciendo que las redes neuronales construidas no tengan que volver a crearse cada vez que cambia el caso de uso, lo que mejora la velocidad y la escalabilidad. Sus principales casos de uso son la sustitución de NumPy para aprovechar la potencia de las GPUs (frente a las CPUs), y como plataforma de investigación de aprendizaje profundo altamente personalizable y rápida.

■ Visión por computador:

²⁰ Enlace: <https://www.nltk.org/>

²¹ Enlace: <https://spacy.io/>

²² Enlace: <https://radimrehurek.com/gensim/>

²³ Enlace: <https://github.com/libfann/fann>

²⁴ Enlace: <https://pytorch.org/>

- **OpenCV**²⁵: Open Source Computer Vision Library (OpenCV) proporciona a los desarrolladores más de 2.500 algoritmos optimizados para una gran variedad de casos de uso relacionados con la visión por computador. Desde la detección/reconocimiento de rostros hasta la clasificación de acciones humanas, OpenCV hace que la comprensión de la información visual sea tan simple como llamar a la función correcta y especificar los parámetros adecuados.
- **SimpleCV**²⁶: mientras que OpenCV se centra en la exhaustividad y la personalización, SimpleCV se centra en hacer que la visión por computador sea sencilla. La curva de aprendizaje es mucho menor hasta el punto de que obtener imágenes de una cámara es tan sencillo como inicializar una cámara (usando `Camera()`) y obtener su imagen (usando `Camera.getImage()`). De esta forma, el desarrollador puede enfocarse en el dominio del problema y realizar prototipos de la solución de forma rápida y sencilla.

■ Sistemas expertos:

- **PyCLIPS**²⁷: PyCLIPS proporciona un motor de inferencia basado en reglas como módulos binarios dentro de la biblioteca a los que se accede mediante clases y funciones. El motor en sí mismo permanece residente en un espacio de memoria separado del espacio de Python, por lo que las inferencias y las reglas se mantienen a medida que el programa crece en funcionalidad.
- **PyKnow**²⁸: También inspirado en CLIPS, esta biblioteca es un motor de reglas que asocia un conjunto de hechos con un conjunto de reglas basadas en esos hechos. Después, se ejecutan acciones basadas en estas reglas. Todos los hechos y las reglas son mantenidos por el motor de conocimiento implementado que determina la salida del sistema experto cuando es invocado.

■ Robótica:

- **AirSim**²⁹: AirSim es un simulador basado en Unity/Unreal Engine construido por Microsoft, que permite a los desarrolladores probar y experimentar con algoritmos de vehículos autónomos sin necesidad de poseer el hardware físico para ello. De este modo, proporciona un entorno de pruebas para vehículos autónomos sin los costes y los problemas de seguridad que habría que superar en el mundo real.

²⁵ Enlace: <https://opencv.org/>

²⁶ Enlace: <http://simplecv.org/>

²⁷ Enlace: <http://pyclips.sourceforge.net/>

²⁸ Enlace: <https://github.com/buguroo/pyknow>

²⁹ Enlace: <https://microsoft.github.io/AirSim/>

- **Carla**³⁰: mientras que AirSim puede atender a una amplia variedad de vehículos autónomos (como coches y drones), Carla se dirige específicamente a la investigación de la conducción autónoma. Cuenta con características más específicas para el conductor, como sensores flexibles para el vehículo, y condiciones ambientales, así como una amplia variedad de edificios y vehículos ya implementados.

Todas las bibliotecas mencionadas incluyen soporte para el lenguaje Python, ya sea de forma interna por los desarrolladores o de forma externa por la comunidad. Dado el rico ecosistema de bibliotecas que existe, la curva de aprendizaje del lenguaje y el soporte de plataformas de ejecución, Python se ha erigido como el lenguaje *de facto* para el desarrollo de sistemas de IA.

En el resto de la sección se introducirá la biblioteca **scikit-learn** mediante casos prácticos de entrenamiento de un modelo para resolver un problema de clasificación. Pero antes de eso, veamos como configurar el entorno para ejecutar los casos prácticos.

3.2.1. Configuración del entorno

A la hora de ejecutar ejemplos de código, seguir casos prácticos o desarrollar un proyecto, resulta conveniente mantener las dependencias entre bibliotecas bien organizadas. Python nos proporciona para ello los entornos virtuales o *virtualenvs*. Mediante esta herramienta, podemos aislar las versiones de las dependencias entre proyectos sin alterar la instalación global de nuestra máquina. Sin embargo, tratar con los entornos virtuales directamente plantea ciertas desventajas o problemáticas como, por ejemplo, la separación de dependencias entre los entornos de producción y desarrollo, la resolución de dependencias, la generación automatizada de archivos *requirements.txt*, o el empaquetado de proyectos, entre otros.

A lo largo de los años han surgido diversas herramientas que tratan de facilitar la gestión de dependencias y el uso de entornos virtuales en los proyectos Python como, por ejemplo, *Pipenv*³¹, *pip-tools*³², *Conda*³³, o *Poetry*³⁴, entre otros. De todos los anteriores, *Poetry* nos proporciona un entorno integrado para la gestión de paquetes, la resolución de dependencias y el empaquetado de nuestros proyectos siguiendo las últimas recomendaciones del lenguaje, con el objetivo de poder compartir nuestros proyectos de código y crear entornos que sean reproducibles para otros usuarios.

Para empezar a utilizar *Poetry* tendremos que descargarlo desde su sitio web oficial siguiendo las instrucciones disponibles en Enlace: <https://python-poetry.org/docs/#installation>. Una vez instalado, podemos ejecutar el siguiente comando para comprobar que todo ha ido bien y comprobar la versión instalada de la herramienta:

³⁰ Enlace: <http://carla.org/>

³¹ Enlace: <https://github.com/pypa/pipenv>

³² Enlace: <https://pip-tools.readthedocs.io/>

³³ Enlace: <https://docs.conda.io/>

³⁴ Enlace: <https://python-poetry.org/>

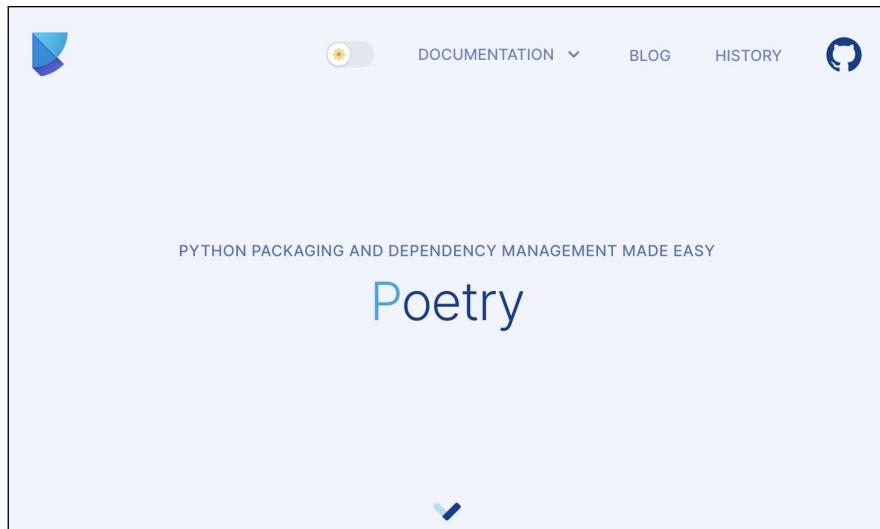


Figura 3.51: Sitio web oficial de la herramienta *Poetry*

```
$ poetry --version  
Poetry version 1.1.11
```

Ahora, y preparando el caso práctico que veremos después, podremos crear un nuevo proyecto con las dependencias que vayamos a utilizar:

```
$ poetry new caso-practico  
Created package caso_practico in caso-practico
```

Esto creará una estructura de archivos inicial similar a la siguiente:

```
$ tree  
. .  
└── README.rst  
└── caso_practico  
    └── __init__.py  
└── pyproject.toml  
└── tests  
    └── __init__.py  
    └── test_caso_practico.py  
2 directories, 5 files
```

El archivo *pyproject.toml* será el que mantenga toda la configuración del proyecto, así como las dependencias de producción y desarrollo que se vayan instalando. Vamos ahora a proceder a instalar algunas de ellas:

```
$ poetry add scikit-learn pandas ipykernel
Using version ^1.0.1 for scikit-learn
Using version ^1.3.4 for pandas
Using version ^6.4.2 for ipykernel
```

```
Updating dependencies
Resolving dependencies...
```

El comando *poetry add* permite añadir nuevas bibliotecas a nuestro proyecto, en nuestro caso, *scikit-learn* para la creación de un modelo de aprendizaje automático, *pandas* para el tratamiento de los datos y *ipykernel* para la ejecución interactiva de Python en los cuadernos Jupyter.

Tras instalar las bibliotecas por primera vez, se creará un nuevo directorio *.venv* en nuestro proyecto para mantener la configuración del entorno virtual, y un archivo *poetry.lock*, para bloquear todas las dependencias de nuestro proyecto. Cabe destacar que estas dependencias solo existirán en el entorno virtual que se acaba de crear y no estarán disponibles a nivel global. En el futuro, podemos añadir más bibliotecas al proyecto utilizando el mismo comando; esto actualizará los archivos *pyproject.toml* y *poetry.lock* automáticamente. En caso de que queramos compartir el proyecto con alguien más o configurarlo en otra máquina, simplemente habrá que ejecutar el comando **poetry install** para instalar las dependencias del proyecto; en primer lugar se resolverán desde el archivo *poetry.lock* y si este archivo no existe, se resolverán desde el archivo *pyproject.toml* y se generará un nuevo archivo *poetry.lock*.



Si queremos ejecutar nuestros *scripts* de Python utilizando el entorno virtual creado por la herramienta, podemos hacerlo mediante el comando **\$ poetry run python script.py**. Del mismo modo, también podemos activar completamente el entorno virtual mediante el comando **\$ poetry shell**. Desde ese momento, podremos utilizar el binario de *python* del entorno virtual para ejecutar nuestro proyecto haciendo uso de las dependencias instaladas. Para abandonar el entorno, simplemente habrá que ejecutar el comando **\$ exit** o pulsar la combinación de teclas **Ctrl+D**.

Una vez hayamos instalado todas las bibliotecas que vamos a utilizar, abriremos el directorio raíz del proyecto con Visual Studio Code. Desde la aplicación, crearemos un directorio **notebooks** y un archivo en su interior llamado **Pruebas.ipynb**. Para disponer de soporte para cuadernos Jupyter desde VS Code, instalaremos además el pack de extensiones llamado **Jupyter**³⁵. Por último, abriremos nuestro nuevo archivo **notebooks/Pruebas.ipynb**, seleccionaremos el *kernel* de ejecución pulsando sobre el botón **Select Kernel** y elegiremos el entorno de ejecución de Python cuya ruta coincida con la de nuestro proyecto; esto permitirá conectar la instan-

³⁵ Enlace: <https://marketplace.visualstudio.com/items?itemName=ms-toolsai.jupyter>

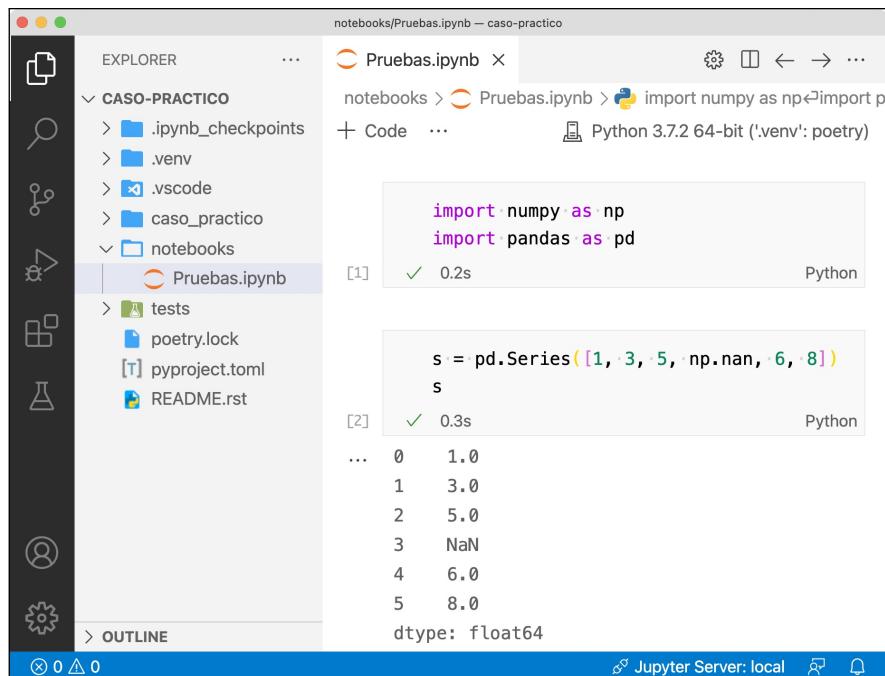


Figura 3.52: Comprobación del entorno de ejecución

cia de VS Code con nuestro entorno virtual, por lo que todas las ejecuciones que se realicen sobre los cuadernos Jupyter se realizarán sobre nuestro entorno virtual. Con todo esto, ya podemos escribir algunas líneas de código en el cuaderno para ver si todo se ha configurado correctamente (ver Figura 3.52).

En la estructura de archivos que nos ha generado *Poetry*, tendremos un subdirectorio con el mismo nombre del proyecto, en nuestro caso, *caso_practico*. En este directorio será donde añadamos los *scripts* que formen parte del dominio de la aplicación, y desde los cuadernos Jupyter podremos realizar pruebas sobre ellos. Sin embargo, con la configuración actual no podremos importar módulos de ese subdirectorio desde nuestros cuadernos Jupyter, a no ser que ubiquemos los cuadernos en la raíz del proyecto, lo cual no es deseable. Para solucionarlo, deberemos cambiar la configuración de VS Code para que el directorio raíz del cuaderno no sea el directorio donde se encuentra el archivo del cuaderno, sino el directorio raíz del proyecto. Para ello, basta con abrir el archivo de configuración de VS Code y añadir la siguiente línea:

```
"jupyter.notebookFileRoot": "$workspaceFolder"
```

3.2.2. Caso práctico: prediciendo fugas de clientes

A lo largo del resto de la sección se plantea un caso práctico en el que se intentará predecir mediante un modelo de aprendizaje automático qué clientes son propensos a dejar de utilizar los servicios que ofrece una determinada compañía. Saber esto es crucial para una empresa, ya que le permitiría emprender acciones para retener al cliente antes de que se produzca la pérdida definitiva.

A partir de los datos de fugas de clientes antiguos, es posible crear un modelo para identificar los clientes potenciales de cancelar su suscripción con los servicios que ofrece la compañía. Por tanto, se trata de un problema de clasificación binaria, donde se intenta predecir si un determinado cliente se dará de baja o no.

Para ilustrar el ejemplo se utilizará un análisis de los datos mediante un modelo estadístico de **regresión logística**, por ser uno de los más simples y rápidos de utilizar pero que al mismo tiempo nos permitirá realizar predicciones sobre variables categóricas (aquellas que pueden adoptar un número finito de categorías).

Preparación de los datos

Para este caso práctico vamos a partir de un *dataset* que contiene toda la información que necesitamos. Concretamente, utilizaremos el *dataset* de *Telco Customer Churn*, disponible en el sitio web de Kaggle desde  Enlace: <https://www.kaggle.com/blastchar/telco-customer-churn> y que contiene los datos de los clientes de una empresa que ofrece servicios de telecomunicaciones. De acuerdo a su descripción, el *dataset* incluye información de:

- Los clientes que se han dado de baja en el último mes (columna *Churn*).
- Servicios contratados por cada cliente: teléfono, líneas múltiples, Internet, seguridad en línea, copias de seguridad en línea, protección de dispositivos, asistencia técnica, TV y películas.
- Información de la cuenta del cliente: cuánto tiempo ha sido cliente, contrato, método de pago, facturación digital, cargos mensuales y cargos totales.
- Información demográfica sobre los clientes: sexo, rango de edad, y si tienen pareja y personas a su cargo.

Una vez descargado, lo descomprimiremos y almacenaremos el archivo CSV contenido en un nuevo directorio **datasets** en la raíz de nuestro proyecto.

Vamos ahora a realizar un análisis inicial de los datos. Para ello, empezaremos creando un nuevo cuaderno Jupyter llamado **CasoPractico.ipynb** donde realizaremos el análisis. Una vez creado, importaremos algunas bibliotecas que necesitaremos después y cargaremos el *dataset* que acabamos de descargar (ver Figura 3.75).

Tras importar los datos, podemos ver que el *dataset* contiene 7043 filas y varias columnas, cada una de ellas con un tipo de dato concreto:

- CustomerID: el identificador del cliente.

```

import pandas as pd
df = pd.read_csv('datasets/WA_Fn-UseC_-Telco-Customer-Churn.csv')
print(len(df))
df.head().T
... 0.5s
... 7043
</>
   0      1      2      3      4
customerID 7590-VHVEG 5575-GNVDE 3668-QPYBK 7795-CFOCW 9237-HQITU
gender       Female     Male     Male     Male     Female
SeniorCitizen 0         0         0         0         0
Partner       Yes        No        No        No        No
Dependents    No        No        No        No        No
tenure        1         34        2         45        2
PhoneService  No        Yes       Yes       No        Yes
MultipleLines No phone service No        No        No phone service No
InternetService DSL        DSL       DSL       DSL     Fiber optic
OnlineSecurity No        Yes       Yes       Yes       No
OnlineBackup   Yes        No        Yes       No        No
DeviceProtection No       Yes       No        Yes       No
TechSupport    No        No        No        Yes       No
StreamingTV    No        No        No        No        No
StreamingMovies No       No        No        No        No
Contract      Month-to-month One year Month-to-month One year Month-to-month
PaperlessBilling Yes        No        Yes       No        Yes
PaymentMethod  Electronic check Mailed check Mailed check Bank transfer (automatic) Electronic check
MonthlyCharges 29.85      56.95      53.85      42.3      70.7
TotalCharges   29.85      1889.5     108.15     1840.75    151.65
Churn         No        No        Yes       No        Yes

```

Figura 3.53: Importación y previsualización del *dataset*

- Gender: masculino o femenino (*male/female*).
- SeniorCitizen: si el cliente es una persona mayor (*0/1*).
- Partner: si el cliente vive en pareja (*yes/no*).
- Dependents: si el cliente tiene dependientes (*yes/no*).
- Tenure: número de meses desde que se inició el contrato (*numérico*).
- PhoneService: si el cliente tiene línea de teléfono (*yes/no*).
- MultipleLines: si el cliente tiene varias líneas telefónicas (*yes/no/no phone service*).
- InternetService: el tipo de servicio de internet contratado (*no/fiber/optic*).
- OnlineSecurity: si la seguridad está activada (*yes/no/no internet*).
- OnlineBackup: si el servicio de copias de seguridad online está activado (*yes/no/no internet*).

- DeviceProtection: si el servicio de protección de dispositivos está activado (*yes/no/no internet*).
- TechSupport: si el cliente tiene contratado el servicio de soporte técnico (*yes/no/no internet*).
- StreamingTV: si el servicio de TV está activado (*yes/no/no internet*).
- StreamingMovies: si el servicio de películas está activado (*yes/no/no internet*).
- Contract: el tipo de contrato (*monthly/yearly/two years*).
- PaperlessBilling: si la facturación es digital (*yes/no*).
- PaymentMethod: la forma de pago (*electronic check/mailed check/bank transfer/credit card*).
- MonthlyCharges: el importe total cobrado (*numérico*).
- Churn: si el cliente se ha dado de baja (*yes/no*).

Entre las propiedades anteriores, la más relevante para nuestro caso práctico será **Churn**, la cual estableceremos como variable objetivo o como aquella sobre la que nuestro modelo realizará las predicciones.

Cuando Pandas importa el *dataset*, intenta determinar automáticamente el tipo de dato para cada columna. Podemos ver esta información inspeccionando el atributo `dtypes` (ver Figura 3.76).

Podemos ver como casi todos los tipos de datos se han identificado correctamente (el tipo *object* equivale a un tipo de cadena de texto), aunque podemos ver ciertas particularidades en dos columnas:

- SeniorCitizen: esta columna se ha identificado como tipo numérico debido a que puede tomar valores de 0 o 1. Esto no afecta realmente al resto del caso práctico, por lo que podemos ignorarlo.
- TotalCharges: esta columna se ha identificado como una cadena de texto debido a que, algunas filas, contienen un espacio (« ») para representar un valor que falta.

Podemos cambiar el tipo de la columna *TotalCharges* mediante la función `to_numeric` de Pandas y especificar que aquellos valores que falten sean sustituidos por el valor `NaN`, que en Python representa un valor numérico (ver Figura 3.77).

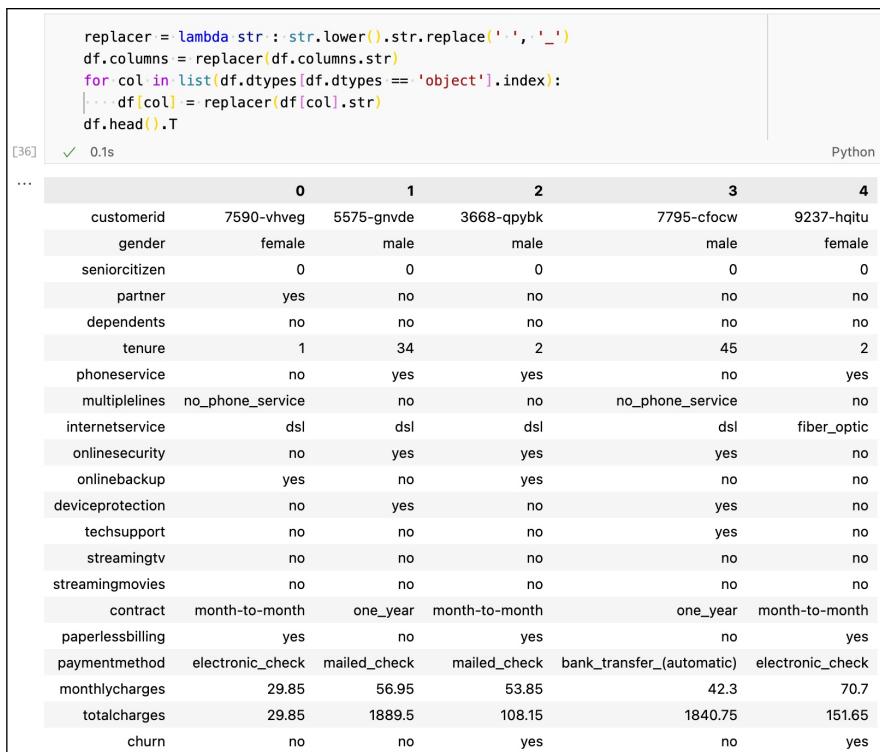
También se habrá podido observar que los nombres de algunas columnas no son consistentes con el resto, por ejemplo, empezando algunas de ellas con una letra minúscula mientras que otras lo hacen en mayúsculas. También podemos ver hay espacios en los valores de las columnas. Así, renombraremos los nombres de las columnas para cambiarlos por minúsculas y sustituiremos los espacios en los valores de las columnas por guiones bajos (ver Figura 3.78).

df.dtypes	
[10]	✓ 0.1s
...	
customerID	object
gender	object
SeniorCitizen	int64
Partner	object
Dependents	object
tenure	int64
PhoneService	object
MultipleLines	object
InternetService	object
OnlineSecurity	object
OnlineBackup	object
DeviceProtection	object
TechSupport	object
StreamingTV	object
StreamingMovies	object
Contract	object
PaperlessBilling	object
PaymentMethod	object
MonthlyCharges	float64
TotalCharges	object
Churn	object
dtype:	object

Figura 3.54: Inspeccionando los tipos de datos identificados por Pandas al importar el *dataset*

[26]	✓ 0.3s	Python	
<pre>total_charges = pd.to_numeric(df.TotalCharges, errors='coerce') df.TotalCharges = pd.to_numeric(df.TotalCharges, errors='coerce') df.TotalCharges = df.TotalCharges.fillna(0) df[total_charges.isnull()][['customerID', 'TotalCharges']]</pre>			
...			
	customerID	TotalCharges	
	488	4472-LVYGI	0.0
	753	3115-CZMZD	0.0
	936	5709-LVOEQ	0.0
	1082	4367-NUYAO	0.0
	1340	1371-DWPAZ	0.0
	3331	7644-OMVMY	0.0
	3826	3213-VVOLG	0.0
	4380	2520-SGTTA	0.0
	5218	2923-ARZLG	0.0
	6670	4075-WKNIU	0.0
	6754	2775-SEFEE	0.0

Figura 3.55: Corrección de tipos de datos para la columna *TotalCharges*



The screenshot shows a Jupyter Notebook cell with the following content:

```

replacer = lambda str: str.lower().str.replace(' ', '_')
df.columns = replacer(df.columns.str)
for col in list(df.dtypes[df.dtypes == 'object'].index):
    df[col] = replacer(df[col].str)
df.head().T

```

[36] ✓ 0.1s

Python

	0	1	2	3	4
customerid	7590-vhveg	5575-gnvde	3668-qpybk	7795-cfocw	9237-hqitu
gender	female	male	male	male	female
seniorcitizen	0	0	0	0	0
partner	yes	no	no	no	no
dependents	no	no	no	no	no
tenure	1	34	2	45	2
phoneservice	no	yes	yes	no	yes
multiplelines	no_phone_service	no	no	no_phone_service	no
internetservice	dsl	dsl	dsl	dsl	fiber_optic
onlinesecurity	no	yes	yes	yes	no
onlinebackup	yes	no	yes	no	no
deviceprotection	no	yes	no	yes	no
techsupport	no	no	no	yes	no
streamingtv	no	no	no	no	no
streamingmovies	no	no	no	no	no
contract	month-to-month	one_year	month-to-month	one_year	month-to-month
paperlessbilling	yes	no	yes	no	yes
paymentmethod	electronic_check	mailed_check	mailed_check	bank_transfer_(automatic)	electronic_check
monthlycharges	29.85	56.95	53.85	42.3	70.7
totalcharges	29.85	1889.5	108.15	1840.75	151.65
churn	no	no	yes	no	yes

Figura 3.56: Cambio de los nombres de las columnas a minúsculas y sustitución de espacios por guiones bajos en los valores

Por último, en los problemas de clasificación binaria, los modelos suelen esperar los valores de la variable objetivos como valores numéricos, por lo que modificaremos los valores de columna *churn* para sustituir las cadenas de texto «yes» por un 1 y las cadenas de texto «no» por un 0 (ver Figura 3.79).

Una vez tengamos el *dataset* preparado, podemos proceder a dividirlo en datos de entrenamiento y datos de prueba. Para ello, utilizaremos la función `train_test_split` de scikit-learn para mantener un 80 % de los datos como datos de entrenamiento y el restante 20 % como datos de prueba. Esta función mezclará los datos del *dataset* aleatoriamente y después los dividirá en los dos conjuntos. Para que esta mezcla se realice de la misma manera en invocaciones sucesivas, proporcionaremos un valor constante al tercer parámetro de la función (`random_state`). Repetiremos el proceso de nuevo sobre los datos de entrenamiento para reservar un 33 % de los datos de entrenamiento para su validación. Por último, eliminaremos la columna *churn* de los datos de entrenamiento para asegurarnos de que dicha columna no se utiliza accidentalmente durante el entrenamiento del modelo, no sin antes hacer una copia para utilizarla posteriormente (ver Figura 3.58).

```
df.churn = (df.churn == 'yes').astype(int)
df.churn.head()
```

[55] ✓ 0.2s Python

	0	1
0	0	0
1	0	1
2	1	0
3	0	1
4	1	0

Name: churn, dtype: int64

Figura 3.57: Sustitución de los valores de la variable objetivo por valores numéricos

```
from sklearn.model_selection import train_test_split
df_train_full, df_test = train_test_split(df, test_size=0.2, random_state=1)

df_train, df_val = train_test_split(df_train_full, test_size=0.33, random_state=1)
y_train = df_train.churn.values
y_val = df_val.churn.values

del df_train['churn']
del df_val['churn']

df_train.head()
```

[69] ✓ 0.3s Python

	4204	7034	5146	5184	1310
customerid	4395-pzmsn	0639-tsiqw	3797-fkogg	7570-welny	6393-wryze
gender	male	female	male	female	female
seniorcitizen	1	0	0	0	0
partner	no	no	no	yes	yes
dependents	no	no	yes	no	no
tenure	5	67	11	68	34
phoneservice	yes	yes	yes	yes	yes
multiplelines	no	yes	yes	yes	yes
internetservice	fiber_optic	fiber_optic	fiber_optic	fiber_optic	fiber_optic
onlinesecurity	no	yes	no	yes	no
onlinebackup	yes	yes	no	yes	no
deviceprotection	no	yes	no	no	no
techsupport	no	no	no	no	no
streamingtvtv	no	yes	no	no	yes
streamingmovies	yes	no	yes	no	yes
contract	month-to-month	month-to-month	month-to-month	two_year	month-to-month
paperlessbilling	yes	yes	no	yes	yes
paymentmethod	electronic_check	credit_card_(automatic)	electronic_check	bank_transfer_(automatic)	electronic_check
monthlycharges	85.55	102.95	86.2	84.7	97.65
totalcharges	408.5	6886.25	893.2	5711.05	3207.55

Figura 3.58: Sustitución de los valores de la variable objetivo por valores numéricos

Análisis de importancia de propiedades

Antes de pasar al proceso de entrenamiento, tenemos que identificar qué variables tienen un mayor impacto sobre la variable objetivo que pretendemos predecir. En el caso de las variables categóricas, podemos estudiar su importancia calculando el grado de dependencia entre ella y la variable objetivo. Si dos variables son dependientes, conocer el valor de una de ellas nos dará cierta información sobre la otra. Por otro lado, si una variable es completamente independiente de la variable objetivo, no nos será útil, por lo que podremos eliminarla con seguridad del conjunto de datos.

Para las variables categóricas, una de estas métricas es la *información mutua*, que nos indica cuánta información obtenemos sobre una variable si conocemos el valor de otra. Esta métrica se utiliza a menudo en el aprendizaje automático para medir la dependencia mutua entre dos variables: a mayor valor de información mutua, mayor será la dependencia entre ambas variables (y por tanto dicha variable será relevante para predecir el objetivo).

Utilizando la función `mutual_info_score` de scikit-learn, podemos calcular el valor de información mutua entre la variable objetivo y cada una de nuestras variables categóricas (ver Figura 3.59).

De este análisis, podemos ver que las variables *contract*, *onlinesecurity* y *tech-support* se encontrarían entre las propiedades más importantes.

Como se ha mencionado, la información mutua nos serviría para cuantificar el grado de dependencia entre dos variables categóricas. En nuestro caso, nos quedaría por cuantificar el grado de dependencia de tres variables numéricas, por lo que tenemos que aplicar alguna otra técnica para ello. Un método estadístico que podemos aplicar es el coeficiente de correlación de Pearson entre dos variables numéricas. En nuestro caso, podemos aplicarlo asumiendo que los valores de la variable objetivo se han convertido a valores numéricos (0 y 1). El coeficiente puede tomar valores entre -1 y 1:

- Una correlación positiva implica que cuando el valor de una variable aumenta, también lo hace el de la otra variable.
- Una correlación de cero indica que no hay relación entre las dos variables.
- Una correlación negativa implica que cuando el valor de una variable aumenta, el valor de la otra variable disminuye.

Así, podemos aplicar el coeficiente de correlación a cada una de nuestras tres variables numéricas para estudiar la correlación de cada una con nuestra variable objetivo mediante la función `corrwith` de Pandas (ver Figura 3.60).

De estos coeficientes podemos deducir que cuanto más paga al mes un cliente por los servicios contratados, mayor es la posibilidad de que se dé de baja en los servicios de la empresa.

```
from sklearn.metrics import mutual_info_score

calculate_mi = lambda col: mutual_info_score(col, df_train_full.churn)

df_mi = df_train_full[categorical].apply(calculate_mi)
df_mi = df_mi.sort_values(ascending=False).to_frame(name='MI')
df_mi
```

[88] ✓ 0.1s Python

...

	MI
contract	0.098320
onlinesecurity	0.063085
techsupport	0.061032
internetservice	0.055868
onlinebackup	0.046923
deviceprotection	0.043453
paymentmethod	0.043210
streamingtv	0.031853
streamingmovies	0.031581
paperlessbilling	0.017589
dependents	0.012346
partner	0.009968
seniorcitizen	0.009410
multiplelines	0.000857
phoneservice	0.000229
gender	0.000117

Figura 3.59: Cálculo de la información mutua entre la variable objetivo y el resto de variables categóricas

Ingeniería de propiedades

El último paso antes de proceder a entrenar nuestro modelo será el de la ingeniería de propiedades (o *feature engineering*, en inglés). Los modelos de aprendizaje automático trabajan con matrices numéricas, por lo que deberemos convertir todas las variables categóricas a variables numéricas que podamos codificar en forma de matriz de datos.

Para ello, podemos simplemente aplicar la técnica de codificación de etiquetas y darle un valor numérico a cada cadena de texto. Sin embargo, esto podría presentar el problema de que los valores numéricos sean malinterpretados por algunos algoritmos. Por ello, surge la técnica de codificación *one-hot*, cuya estrategia se define mediante la creación de una columna para cada valor único que exista en la propiedad que estamos codificando y, para cada registro, marcar con un 1 la columna a la que pertenezca dicho registro y dejar a 0 las demás.

```

print(df_train_full[numerical].corrwith(df_train_full.churn))

print(round(df_train_full[df_train_full.tenure <= 2].churn.mean(), 3))
print(round(df_train_full[(df_train_full.tenure > 3) &
|.....|.....|(df_train_full.tenure <= 12)].churn.mean(), 3))
print(round(df_train_full[df_train_full.tenure > 12].churn.mean(), 3))

print(round(df_train_full[df_train_full.monthlycharges < 20].churn.mean(), 3))
print(round(df_train_full[(df_train_full.monthlycharges > 21) &
|.....|.....|(df_train_full.monthlycharges <= 50)].churn.mean(), 3))
print(round(df_train_full[df_train_full.monthlycharges > 50].churn.mean(), 3))

[98]   ✓  0.3s
Python
... tenure      -0.351885
monthlycharges  0.196805
totalcharges    -0.196353
dtype: float64
0.595
0.391
0.176
0.088
0.223
0.325

```

Figura 3.60: Coeficientes de correlación entre las variables numéricas y la variable objetivo

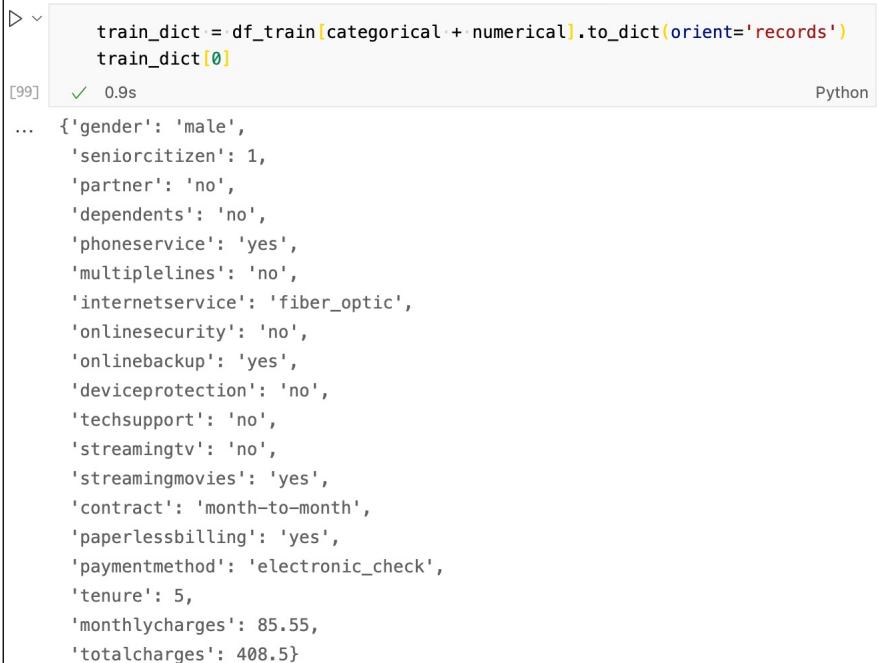
La biblioteca de scikit-learn nos proporciona diversas formas de realizar esta codificación, siendo una de ellas *DictVectorizer*. Para utilizarla, tendremos que convertir nuestro *dataset* a una lista de diccionarios columna → valor mediante la función `to_dict(orient='records')` de Pandas (ver Figura 3.61).

Una vez convertido, podemos pasar a utilizar *DictVectorizer* para realizar la codificación de las propiedades. Para ello, crearemos una instancia de dicha clase y la inicializaremos con los datos de entrenamiento para que infiera los valores para cada propiedad; si la propiedad es categórica, aplica la estrategia de codificación *one-hot* y si es numérica, la dejará intacta. Después, podremos utilizar la función `transform` para convertir la lista de diccionarios a una matriz. Si exploramos el resultado de la operación, veremos como se ha creado una lista de 45 columnas con las posibles combinaciones de los valores de las propiedades categóricas para cada fila de nuestro *dataset* (ver Figura 3.62).

Una vez tengamos nuestras propiedades codificadas en forma de matriz de datos, podríamos proceder a realizar el entrenamiento del modelo.

Entrenamiento del modelo

Para entrenar el modelo utilizaremos una **regresión logística**. Para ello, la biblioteca de scikit-learn nos proporciona la clase `LogisticRegression` que podemos instanciar y el método `fit` que podemos utilizar para realizar el entrenamiento con los datos a partir de la matriz de los datos de entrenamiento (*X_train*) y los valores de la variable objetivo de dicho conjunto de datos (*y_train*) (ver Figura 3.63).



```

train_dict = df_train[categorical + numerical].to_dict(orient='records')
train_dict[0]

```

[99] ✓ 0.9s Python

```

... {'gender': 'male',
'seniorcitizen': 1,
'partner': 'no',
'dependents': 'no',
'phoneservice': 'yes',
'multiplelines': 'no',
'internetservice': 'fiber_optic',
'onlinesecurity': 'no',
'onlinebackup': 'yes',
'deviceprotection': 'no',
'techsupport': 'no',
'streamingtv': 'no',
'streamingmovies': 'yes',
'contract': 'month-to-month',
'paperlessbilling': 'yes',
'paymentmethod': 'electronic_check',
'tenure': 5,
'monthlycharges': 85.55,
'totalcharges': 408.5}

```

Figura 3.61: Conversión del *dataset* a una lista de diccionarios

Una vez entrenado el modelo, ya estaría listo para realizar predicciones sobre nuevos datos utilizando el método `predict_proba`. Para ello, utilizaremos el conjunto de datos de validación que reservamos al comienzo del caso práctico (X_{val}) convirtiéndolo al formato esperado por el modelo (lista de diccionarios transformada a matriz mediante), tal y como se muestra en la Figura 3.64.

El resultado de ejecutar el método `predict_proba` es una matriz bidimensional donde la primera columna contendrá la probabilidad de que el cliente no se de baja (caso negativo) y la segunda columna contendrá la probabilidad de que el cliente sí que se dé de baja (caso positivo). Como únicamente nos interesa el caso positivo, podemos simplificar la estructura eliminando la primera columna de la matriz y discretizar esos valores numéricos de probabilidad a valores booleanos (si el cliente se dará de baja, entonces «verdadero», si no «falso») (ver Figura 3.65). Cabe destacar, que esta discretización de las probabilidades en valores de verdad se realiza estableciendo un punto de corte que utilizaremos para fijar que aquellos valores superiores al punto de corte serán «verdaderos» y aquellos inferiores serán «falsos»; en nuestro ejemplo, se ha establecido que el punto de corte para asumir que un cliente se dará de baja se encontrará a partir de una probabilidad del 50 %.

Una vez obtenidas las predicciones sobre los datos de validación, vamos a ver cómo de certero es nuestro modelo comparando las predicciones obtenidas con los valores reales de la variable objetivo que reservamos al principio, observándose así que el 80 % de las predicciones han sido correctas (ver Figura 3.66).

```

from sklearn.feature_extraction import DictVectorizer

dv = DictVectorizer(sparse=False)
dv.fit(train_dict)

[104] ✓ 0.3s                                         Python
... DictVectorizer(sparse=False)

X_train = dv.transform(train_dict)
X_train[0]

[105] ✓ 0.1s                                         Python
... array([ 1. ,  0. ,  0. ,  1. ,  0. ,  1. ,  0. ,  0. ,
          0. ,  1. ,  0. ,  1. ,  0. ,  85.55,  1. ,  0. ,
          0. ,  0. ,  0. ,  1. ,  1. ,  0. ,  0. ,  0. ,
          1. ,  1. ,  0. ,  0. ,  0. ,  1. ,  0. ,  0. ,
          1. ,  1. ,  0. ,  0. ,  1. ,  1. ,  0. ,  0. ,
          1. ,  0. ,  0. ,  5. ,  408.5 ])
... dv.get_feature_names_out() [-5:] ...

[110] ✓ 0.1s                                         Python
... array(['techsupport=no', 'techsupport=no_internet_service',
         'techsupport=yes', 'tenure', 'totalcharges'], dtype=object)

```

Figura 3.62: Transformación de la lista de diccionarios a una matriz aplicando la estrategia de codificación *one-hot*

```

from sklearn.linear_model import LogisticRegression

model = LogisticRegression(solver='liblinear', random_state=1)
model.fit(X_train, y_train)

[111] ✓ 2.2s                                         Python
... LogisticRegression(random_state=1, solver='liblinear')

```

Figura 3.63: Entrenamiento del modelo

3.2.3. Caso práctico: serialización y despliegue de modelos

Una vez entrenado nuestro modelo de aprendizaje automático deberíamos ser capaces de poder utilizarlo para realizar predicciones bajo demanda, ya sea mediante algún servicio web alojado en un servidor de internet o directamente mediante algún programa que se ejecute en nuestra computadora localmente.

```

val_dict = df_val[categorical + numerical].to_dict(orient='records')
X_val = dv.transform(val_dict)
y_pred = model.predict_proba(X_val)
y_pred

[130] ✓ 0.9s Python
...
... array([[0.99142716, 0.00857284],
       [0.79028939, 0.20971061],
       [0.78364725, 0.21635275],
       ...,
       [0.35664376, 0.64335624],
       [0.81056068, 0.18943932],
       [0.87262068, 0.12737932]])

```

Figura 3.64: Realizando predicciones del modelo sobre el conjunto de datos de validación

```

y_pred = model.predict_proba(X_val)[:, 1]
y_pred

[131] ✓ 0.1s Python
...
... array([0.00857284, 0.20971061, 0.21635275, ..., 0.64335624, 0.18943932,
       0.12737932])

churn = y_pred >= 0.5
churn

[132] ✓ 0.2s Python
...
... array([False, False, False, ..., True, False, False])

```

Figura 3.65: Simplificando las probabilidades obtenidas de las predicciones

```

round((y_val == churn).mean(), 3)

[136] ✓ 0.2s Python
...
... 0.805

```

Figura 3.66: Cálculo de la precisión del modelo

En cualquier caso, el modelo que hemos entrenado hasta ahora reside únicamente en el entorno virtual de Python que hemos preparado. Una vez desactivemos el entorno virtual (por ejemplo, cerrando VS Code) el modelo se perderá y tendremos que volver a entrenarlo de nuevo en caso de que queramos realizar predicciones.

```

import pickle

with open('models/churn-model.pck', 'wb') as f:
    pickle.dump((dv, model), f)

```

[138] ✓ 0.1s Python

Figura 3.67: Serialización del modelo y la instancia del *DictVectorizer* usando el módulo *Pickle*

Para evitar esto, necesitamos alguna forma de, en primer lugar, persistir el modelo entre distintas ejecuciones del entorno virtual y, además, poder integrar los modelos entrenados en algún otro sistema software. Para ello, Python nos proporciona el módulo **Pickle**, que nos permite serializar objetos en formato binario y cargarlos más tarde. En nuestro caso, además del modelo, tendremos que almacenar también la instancia del *DictVectorizer* que inicializamos con el *dataset*, por lo que podemos guardar ambos objetos como una tupla (ver Figura 3.67).

Para el caso del ejemplo, se ha serializado el modelo en el directorio **models** ubicado en la raíz del proyecto. Una vez almacenado nuestro modelo en disco, podemos construir una aplicación que lo importe y sea capaz de realizar predicciones sobre el modelo.

Vamos a crear un pequeño servidor en **Flask** para construir un microservicio que nos permita realizar predicciones mediante peticiones web. Antes de nada, añadiremos la dependencia al proyecto:

```

$ poetry add flask
Using version ^2.0.2 for Flask

Updating dependencies
Resolving dependencies...

```

Una vez instalada, nos dirigiremos al directorio *caso_práctico* y crearemos un nuevo *script* de Python llamado *churn_predict_service.py* con una función que nos permitirá realizar predicciones sobre el modelo:

```

1 def predict_single(customer, dv, model):
2     x = dv.transform([customer])
3     y_pred = model.predict_proba(x)[:, 1]
4     return (y_pred[0] >= 0.5, y_pred[0])

```

Este método nos devolverá una tupla cuyo primer elemento será si el cliente especificado se dará de baja de los servicios y cuyo segundo elemento será la probabilidad de hacerlo.

Después, crearemos un *script* llamado *churn_predict_app.py* en el directorio *caso_práctico* de la raíz del proyecto. Este *script* inicializará el servidor de Flask y expondrá un *endpoint* que podremos consumir para realizar las predicciones sobre el modelo utilizando el servicio anterior:

```
1 import pickle
2
3 from flask import Flask, jsonify, request
4
5 from churn_predict_service import predict_single
6
7 app = Flask('churn-predict')
8
9 with open('models/churn-model.pck', 'rb') as f:
10     dv, model = pickle.load(f)
11
12
13 @app.route('/predict', methods=['POST'])
14 def predict():
15     customer = request.get_json()
16     churn, prediction = predict_single(customer, dv, model)
17
18     result = {
19         'churn': bool(churn),
20         'churn_probability': float(prediction),
21     }
22
23     return jsonify(result)
24
25
26 if __name__ == '__main__':
27     app.run(debug=True, port=8000)
```

Cabe destacar las líneas ⑨-10, utilizadas para cargar el modelo desde disco y pasárselo como parámetro al servicio de predicciones para que lo utilice. El resto del código inicializa una ruta en */predict* que responde a peticiones HTTP de tipo *POST* con la información del cliente en formato JSON. Por último, se inicializa el servidor Flask en el puerto 8000 de la máquina local. Para ejecutarlo, lo haremos a través de la herramienta Poetry:

```
$ poetry run python
caso_practico/churn_predict_app.py
* Serving Flask app 'churn-predict' (lazy loading)
* Environment: production
WARNING: This is a development server. Do not use it in a production deployment.
Use a production WSGI server instead.
* Debug mode: on
* Running on http://127.0.0.1:8000/ (Press CTRL+C to quit)
```

Una vez ejecutado el servidor Flask, podemos poner a prueba nuestra aplicación enviando una petición al servicio con los datos de algún cliente. Para ello, vamos a utilizar la herramienta por consola *curl*:

```
$ curl -request POST 'http://127.0.0.1:8000/predict'
-h 'Content-Type: application/json'
-d '{
    "gender": "female",
    "seniorcitizen": 0,
```

```
"partner": "no",
"dependents": "no",
"tenure": 41,
"phoneservice": "yes",
"multiplelines": "no",
"internetservice": "dsl",
"onlinesecurity": "yes",
"onlinebackup": "no",
"deviceprotection": "yes",
"techsupport": "yes",
"streamingtv": "yes",
"streamingmovies": "yes",
"contract": "one_year",
"paperlessbilling": "yes",
"paymentmethod": "bank_transfer_(automatic)",
"monthlycharges": 79.85,
"totalcharges": 3320.75
}'
```

Para este cliente obtendremos una respuesta como la siguiente:

```
{
  "churn": false,
  "churn_probability": 0.05775414541319264
}
```

Con todo lo anterior, ya tendríamos una aplicación web que podemos utilizar para predecir si un cliente se dará de baja en los servicios de la compañía. Finalmente, podríamos desplegar la aplicación web utilizando los servicios que ofrecen algunas de las plataformas de computación en la nube vistas hasta el momento (por ejemplo, *AWS Lambda*).

3.3. Modelos y algoritmos predefinidos

En esta sección se verán los principales conceptos relativos a la Inteligencia Artificial, su definición, tipos y paradigmas, comentando algunos de los algoritmos más conocidos y algunas plataformas donde ya vienen predefinidos y pueden utilizarse como cajas negras sin necesidad de conocer la mayoría de sus pormenores.

3.3.1. Inteligencia Artificial

De forma genérica, el término de IA en sus orígenes se le otorgaba a un sistema hardware-software cuyo objetivo era simular el modo de funcionamiento de la inteligencia humana; sin embargo, es difícil dar una clara definición de la IA que englobe todos los posibles aspectos que se tratan, así como delimitar todos los campos relacionados con ella. De forma general, la IA puede clasificarse en 3 categorías:

- IA Estrecha: Su propósito es resolver un problema concreto de forma eficaz y eficiente. Es el tipo predominante a día de hoy. Está íntimamente relacionada con los conceptos de Aprendizaje Automático e IA débil que se verán a continuación.
- IA General: Consiste en un tipo de inteligencia que tiene un grado de cognición al nivel del humano, para ello, es necesario poseer propiedades propias del ser humano como el habla, la visión, etc. Por eso, está íntimamente relacionada con áreas como el Procesamiento de Lenguaje de Natural (PLN), la Visión por Computador, la Cibernética, etc. Se presupone que este nivel de inteligencia englobará muchas otras inteligencias artificiales estrechas. Todavía se está lejos de conseguir este nivel, algunos intentos como las máquinas Watson o Deep Blue de IBM pueden entenderse en este sentido, pero están lejos de resolver tareas complejas a la velocidad a la que lo hace el ser humano. En resumen, este grado de inteligencia pretende imitar el nivel de conocimiento humano.
- Super IA: Es el nivel superior al que se aspira a llevar a la IA, el cual debe superar las capacidades del ser humano. Incluye funciones de razonamiento superior, toma de decisiones inteligentes e incluso, la capacidad de establecer relaciones personales. Se está lejos de alcanzar este nivel a día de hoy.

Para seguir entendiendo el concepto de IA, es necesario conocer los paradigmas o modelos de la IA. En este sentido, existen dos grandes paradigmas, la inteligencia simbólica y la sub-simbólica.

Inteligencia Artificial simbólica vs. sub-simbólica

En sus comienzos, la IA se centró en el desarrollo de mecanismos fácilmente interpretables por cualquier ser humano. Así, la inteligencia simbólica engloba todos aquellos métodos basados principalmente en mecanismos como la búsqueda, representaciones lógicas o de alto nivel simbólicas, es decir, son fácilmente entendibles, y los resultados son fácilmente trazables y explicables. Entre las principales herramientas que pueden encontrarse están las redes semánticas, los marcos, los sistemas de reglas o la programación lógica. Estas técnicas fueron utilizadas ampliamente desde los años 50 hasta los 90. A partir del año 85, este conjunto de técnicas fue acuñadas como GOFAI (“Good Old-Fashioned Artificial Intelligence”): Buena Vieja IA [Hau85].

Así surgió el modelo o paradigma sub-simbólico, que permite representar otra serie de mecanismos que no son tan fácilmente seguibles, paso a paso, por el ser humano, pero que pueden proporcionar igual o mayor potencia que las técnicas anteriores y resolver los mismos e incluso nuevos problemas. Los resultados de este tipo de sistemas suelen ser difícilmente explicables debido a su alta complejidad pero, en muchos casos, pueden ser más eficientes y precisos que los alcanzados por los métodos simbólicos. Este paradigma se relaciona con el modelo conexionista, es decir, aquel que intenta representar en cerebro humano mediante la interconexión de neuronas, dando lugar a las redes neurales artificiales. Además, se relaciona con otros métodos como el aprendizaje profundo, las redes bayesianas, los algoritmos genéticos, etc. Tiene especial aplicación en problemas de predicción, clustering, clasificación, PLN, etc.

En resumen, el paradigma simbólico da lugar a conclusiones lógicas y su proceso está claramente guiado por las decisiones del desarrollador. Por el contrario, el paradigma sub-simbólico produce resultados asociativos consecuencia de los datos, en este caso, los algoritmos aprenden y se adaptan a los datos. Para colecciones de datos pequeñas suelen funcionar mejor los algoritmos simbólicos, mientras que cuando las colecciones son grandes y los datos dispersos y con ruido, es decir, pueden ser no representativos del problema a solucionar, el modelo sub-simbólico puede ser más eficiente [Kri21, RN21].

Inteligencia Artificial fuerte vs. débil

Otra posible clasificación de la IA, la subdivide en dos categorías según los límites que se propongan.

La IA fuerte considera que no hay límites y su objetivo debe ser que un computador pueda simular la mente humana, implementando computacionalmente el pensamiento humano de un modo formal y no ambiguo. La IA fuerte puede entenderse como la capacidad del sistema de adecuarse a un entorno caracterizado por la incertidumbre, siendo capaz de abordar la universalidad de las situaciones que pueden aparecer en el dominio de aplicación. Por lo tanto, el ordenador se presupone capaz de alcanzar un estado de auto-consciencia y analizar y entender los elementos de un dominio de forma proactiva y autónoma.

Esta visión de la IA tiene su origen en la máquina de Turing universal, la cual puede resolver todos los problemas que son resolubles por un algoritmo o un método efectivo de computación. Si todos los procesos de resolución de problemas pudiesen ser simplificados a procesos computacionales, entonces una única máquina, con operaciones relativamente simples, podría resolverlos con una adecuada serie de comandos. La máquina requiere una memoria infinita de almacenamiento y que esta pueda ser tratada.

Por otro lado, la IA débil establece claramente los límites dentro de los cuales pretende dar una solución. Se parte de la idea de que no es necesario implementar todas las capacidades de la inteligencia humana, simplemente su objetivo es ejecutar de forma eficaz y eficiente ciertas tareas que un ser humano puede realizar dentro de un dominio bien determinado. Así pues, el objetivo de la IA débil es entender ciertos aspectos de los procesos del razonamiento humano e implementar máquinas que sean capaces de realizar total o parcialmente tareas mecánicas y de alta complejidad para el ser humano.

Son muchos los ejemplos diarios de casos de éxito que pueden verse. El más conocido puede ser el superordenador Deep Blue, desarrollado por IBM, capaz de calcular miles de jugadas de ajedrez, el cual derrotó a Gary Kasparov en 1996, o el sistema Watson, con altas capacidades de PLN, capaz de responder preguntas de alta dificultad, superando a expertos en concursos de televisión como Jeopardy, sin tener conocimiento previo, simplemente mediante búsquedas de información sobre grandes colecciones de textos.

Mecanismos de aprendizaje

Originariamente, la IA pretendía centrarse en aspectos cognitivos, como eran la lógica del razonamiento humano o el modelado del conocimiento. Paralelamente a esta corriente, surgió otra que pretendía centrarse en el Reconocimiento de Patrones, que tenía que ver con una componente más perceptiva, relacionada con los sentidos: la visión, el habla, etc. Así, se propusieron aplicaciones relacionadas con el reconocimiento de figuras, la detección de patrones lingüísticos, etc. Con el fin de integrar ambas corrientes, y alguna más, surgió el Aprendizaje Automático o Machine Learning.

Para entender el Aprendizaje Automático es necesario prestar atención a los mecanismos de aprendizaje que usa un sistema inteligente para resolver un problema. Todo proceso de aprendizaje generalmente parte una serie de datos de aprendizaje o *entrenamiento* que suelen consistir en unos datos de entrada X y unos de salida Y. El proceso de aprendizaje consiste en encontrar el modelo $F : X \rightarrow Y$ que generalice los datos. Normalmente el proceso de “generalizar” simplemente consiste en predecir la salida Y.

En este caso, hay dos grandes tipos de razonamiento:

- Deductivo: Se parte de una serie de instrucciones supervisadas por un experto. De ahí se define una serie de reglas generales que permiten obtener conclusiones particulares. En estos casos se especifican mecanismos causales en la construcción del modelo F . Un ejemplo de estas técnicas es el razonamiento basado en casos que es usado en los **sistemas expertos**. Los sistemas expertos son un modelo de IA que intenta imitar el comportamiento de un ser humano experto en algún dominio concreto.
- Inductivo: En este caso se parte de una serie de datos, porque el conocimiento a priori sobre el problema a resolver es escaso. Esos datos se interpretan como ejemplos o muestras para el aprendizaje de cuál es la salida del sistema para cada entrada. Es necesario desarrollar modelos que analicen dichos ejemplos concretos, los expliquen y permitan llevar a cabo una generalización de los mismos. En estos casos los modelos F actúan como cajas negras, reciben unos datos y generan un salida, por lo que son candidatos a ser “predefinidos” por plataformas de IA como se verá más adelante, dado que el usuario no tiene que tener gran conocimiento de lo que ocurre en el sistema, “simplemente” se les dota de una serie de datos y el sistema genera, por sí solo, la salida. Este es el mecanismo de razonamiento por excelencia del Aprendizaje Automático. Algunos ejemplos típicos de estos modelos son las redes neuronales o los árboles de decisión.

Existen otros tipos de razonamiento dependiendo de la cantidad de información que se tiene de los datos de entrenamiento. La información puede estar completa o incompleta:

- Supervisado: La información está completa, las observaciones son dadas mediante sus datos de entrada X y sus datos de salida Y . Dada una colección de datos previamente clasificados, estos se utilizan para entrenar un algoritmo que será capaz de aprender sus estructuras típicas, para posteriormente clasificar cualquier otro dato. Entre los algoritmos típicos pueden encontrarse los Support Vector Machines, clasificadores naive Bayes, distintos tipos de regresiones, árboles de decisión, etc.
- No supervisado: La información está incompleta, por ejemplo, se conoce la entrada X pero no la salida Y . Conseguir grandes colecciones de datos previamente clasificados para entrenar un algoritmo es una tarea compleja, por lo que, en aquellos casos en los que la información no está completa, existe este otro tipo de algoritmos que intentan analizar la estructura interna de los datos, diferenciar entre ellos e inferir a qué clases/grupos podrían pertenecer o no. Los algoritmos típicos son los algoritmos de *clustering*, análisis de componentes principales (PCA en inglés), Singular Value Decomposition (SVD) o el análisis de componentes independientes.
- Semi-supervisado: Es una combinación de los dos anteriores, hay colección previa de datos, en la que algunos están clasificados pero, otros no.

Existen más tipos de mecanismos de razonamiento:

- Aprendizaje adaptativo: Se parte de un modelo previo cuyos parámetros se van modificando/adaptando en función de nuevos datos de entrenamiento que van entrando en el sistema.
- Aprendizaje activo: Se conoce la entrada X pero no la salida Y. El sistema debe seleccionar cuáles son los mejores datos de las observaciones y estos, los usa para que un humano los etiquete, generando la salida Y.
- Aprendizaje por refuerzo: Puede considerarse como un caso de aprendizaje semi-supervisado. Los algoritmos se marcan una serie de objetivos y reciben recompensas (premios o castigos) según vayan prediciendo las salidas. En esta estrategia de aprendizaje, mediante ensayo y error, el programa itera de forma muy rápida, aprovechando normalmente las capacidades de super-computadores, generando múltiples opciones hasta detectar cuál es la opción ganadora, es decir, la que alcanza el objetivo, ganando la recompensa. Es una estrategia ampliamente utilizada en el mundo de los juegos, siendo su máximo representante el programa que juega al ajedrez AlphaZero de DeepMind Technologies, actualmente absorbida por Alphabet, empresa matriz de Google. Esto supone una explosión de combinaciones que intentan predecir cuál será el desarrollo de una partida de ajedrez por ejemplo, suponiendo los movimientos de ambos jugadores. Mediante el análisis de todas posibilidades, el sistema trata de aprender caminos, patrones, sacar conclusiones o resultados, que le permitan desarrollar la partida con una estrategia ganadora. Normalmente, algunos algoritmos utilizados en esta categoría son los de programación dinámica, Q-Learning y SARSA (state-action-reward-state-action).

Prestando especial atención al razonamiento supervisado o no supervisado, las aplicaciones típicas del Aprendizaje Automático que pueden encontrarse son:

- Clasificación

El objetivo es predecir la salida Y, también puede ser denominada como clase o etiqueta, a partir de los datos de entrada X. Pueden encontrarse distintos tipos de clasificaciones según el número de valores de salida:

- Clasificación binaria: Solo hay dos clases que detectar, por ejemplo, positivo o negativo, o mujer u hombre.
- Clasificación multi-clase: Existen más de dos clases que detectar. Por ejemplo, un tweet puede ser clasificado como positivo, negativo o neutro.
- Clasificación multi-etiqueta: Cada dato puede ser etiquetado con varias clases, no solo una. Si se analizaran los mensajes de texto de una persona para estudiar su personalidad, diversas etiquetas/clases podrían ser asignadas: extrovertida y agresiva, introvertida y simpática, etc.

- Clustering

El clustering es una opción cuando el conjunto de datos de entrenamiento es incompleto, la salida Y no es conocida. Bajo estas circunstancias, el proceso básicamente consiste en la búsqueda de patrones repetitivos dentro de los datos, es decir, buscar subconjuntos de datos similares entre sí de acuerdo a las características de la entrada X .

- Regresión

Es un caso similar al de la clasificación pero la variable de salida Y no es una etiqueta o clase, es decir, una variables categórica (alto-bajo, positivo-negativo, bueno-regular-malo, etc.). El objetivo de este tipo de problemas es la predicción del valor de una variable continua Y , por ejemplo, la altura de una persona, dados unos valores de entrada X , que representarán características sobre esta persona: peso, ejercicio diario, características físicas de sus padres, etc.

Los métodos de aprendizaje se encuadran dentro de la IA débil, dado que el dominio de aplicación está bien delimitado y es conocido.

3.3.2. Definición de modelos

Las técnicas de Aprendizaje Automática se basan principalmente en la definición de un modelo matemático M que representa la realidad R del problema a resolver. Cada realidad depende de quién la observa, es decir, cada persona u observador puede ver distintos problemas o preguntas que resolver respecto a la realidad. Así, un modelo M es un modelo matemático en un dominio o realidad R para un observador O , que debe:

1. Responder a las preguntas del observador acerca de la realidad R , y
2. Generar datos que reproduzcan la realidad R partiendo de las observaciones de que se dispone de la realidad o dominio R .

Véase un ejemplo, en la Figura 3.68, una persona con experiencia en una empresa clasifica manualmente los paquetes que pasan por un cinta transportadora en distintos grupos. La realidad o dominio R es ese proceso de clasificación manual de los paquetes de la cinta. El observador necesita automatizar ese proceso de clasificación, por lo que sus observaciones son, para cada paquete, las clases en las que son clasificados por el humano. El proceso cerebral que lleva el humano para tomar la decisión de, en qué clase categoriza cada paquete es un misterio, es decir, es una caja negra a la cual no se puede acceder. Así, la única información que se tiene son las observaciones que toma el observador, es decir, los datos. Datos puede haber muchos, el tamaño del paquete, el color, la forma, pero solo deben observarse aquellos que pueden ser útiles para entender el proceso de clasificación.

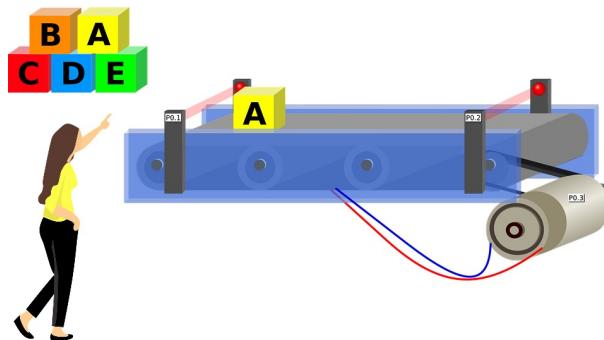


Figura 3.68: Realidad: Proceso de clasificación por un ser humano

Un modelo matemático debe intentar simular el comportamiento del humano, es decir, responder a la pregunta, dado un paquete, a qué categoría corresponde, pero como puede observarse, la función no consiste en reproducir la forma de funcionamiento del cerebro humano, sino simplemente replicar las observaciones que se han obtenido.

En resumen, el cerebro humano es una caja negra que depende de los datos de entrada X para producir una salida Y , formalmente, $\text{cerebro}(X) = Y$.

Un modelo matemático trata de simular ese proceso de clasificación, no el cerebro humano. Para ello, necesita los datos de entrada X , los paquetes, unos hiperparámetros θ que determinarán el tipo de modelo, y además una serie de parámetros sobre el tipo de modelo elegido, α , para ajustar alguna función matemática que generará la salida \bar{Y} , es decir, una predicción que intentará ajustarse lo máximo posible a la salida ideal Y , o lo que es lo mismo, la categoría/clase a la que pertenece cada paquete. Obviamente, el modelo matemático intenta que la categoría predicha \bar{Y} se acerque lo máximo posible a la ideal Y , pero habrá ocasiones en las que falle, por lo que la IA no siempre alcanza la bondad de la humana, aunque puede presentar otras ventajas, por ejemplo, un autómata no se cansa como un humano de clasificar.

Formalmente, un modelo matemático se puede definir como una función:

$$Y = F_\theta(X, \alpha) \quad (3.1)$$

donde tanto Y como X pueden representar uno o un conjunto de valores, y θ representa un conjunto parámetros normalmente conocidos como *hiperparámetros*, que definen la forma del modelo, y α son los parámetros que permiten adaptar el modelo a la realidad que se está trabajando. Este proceso de adaptación se conoce mediante el nombre de *entrenamiento*, es decir, aprendizaje a través de los datos.

Así, dado un conjunto de datos de entrada $X = \{x_1, x_2, x_3\}$ como los valores de un paquete (color, forma, etc.), si el modelo representara una función polinómica de la forma:

$$\bar{Y} = F_\theta(X, \alpha) = \alpha_0 + \alpha_1 x + \alpha_2 x^2 + \dots x^\theta \quad (3.2)$$

dependiendo del grado del polinomio o valor del hiperparámetro $\theta = \{1, 2, 3, \dots\}$ podrían obtenerse funciones con distintas formas: lineales, parabólicas, hiperbólicas, etc., que predijeran con mayor o menor precisión la salida ideal Y .

De forma práctica, los pasos a seguir en la definición de un modelo de IA son:

- Selección del conjunto de datos sobre el que se va a trabajar. Normalmente, los datos originales sufren un proceso de preprocesamiento para seleccionar aquellos que pueden ser más útiles para alcanzar el objetivo buscado.
- Selección del algoritmo de Aprendizaje Automático que mejor se adapte al problema a resolver. Dependiendo del problema a solucionar, el desarrollador debe analizar entre los algoritmos existentes y determinar aquel que mejor se adecúa a la naturaleza del problema y las características de los datos disponibles.
- Configuración de los hiperparámetros del algoritmo. En líneas generales, la mayoría de los algoritmos en IA se caracterizan por una su formulación matemática compleja, que requiere de distintos parámetros conocidos como hiperparámetros, en función de los cuales pueden obtener mejores o peores resultados. El desarrollador debe estudiar cuáles son los valores más adecuados para su colección de datos y objetivo, y configurarlos.

3.3.3. Ejemplo de algoritmos de Aprendizaje Automático

A continuación se van a ver algunos ejemplos de algoritmos de Aprendizaje Automático y sus características; son muchos y de distintos tipos, por lo que solo se analizarán algunos.

Máquinas de Soporte de Vectores (Support Vector Machines)

Este es uno de los algoritmos clásicos basados en razonamientos supervisado, por lo que seguirá un flujo como el descrito en la Figura 3.69. Primero, se partirá de una colección de datos etiquetados, por ejemplo, uno conjunto de emails que serán la entrada X del sistema, con una etiqueta, “spam” o “no spam”, que será la salida del sistema Y . Ese conjunto de datos se puede dividir en dos partes, datos para entrenamiento y datos para testar cuán bueno es el modelo generado. Normalmente, entre el 70-80 % de los datos de la colección se usa para entrenamiento, y el restante 20-30 % se usa para testar el modelo.

Existen dos fases, la de generación del modelo o entrenamiento, que requiere de esa sub-colección de datos de entrenamiento, un algoritmo como los SVMs y la elección de los hiperparámetros típicos de dicho algoritmo. Con ello se genera el modelo matemático (modelo clasificador en la figura) perfectamente ajustado y listo para ser usado. Esta fase solo es necesaria ejecutarla una vez.

Una vez completada esta fase, el modelo está listo para ser usado o testado. Para testarse, se utiliza la otra sub-colección que permitirá medir cuántos aciertos y fallos comete el sistema. Si el modelo se considera que funciona bien, se puede utilizar tal cual, en caso contrario habrá que ajustar los hiperparámetros, añadir nuevos datos de entrenamiento o cambiar el algoritmo con el fin de mejorar el proceso de clasificación.

Cuando el modelo se considera que funciona bien se puede utilizar, es decir, se le pueden pasar nuevos emails de los que se desconoce si son spam o no, el modelo dirá si lo son o no, aunque pueda cometer errores.

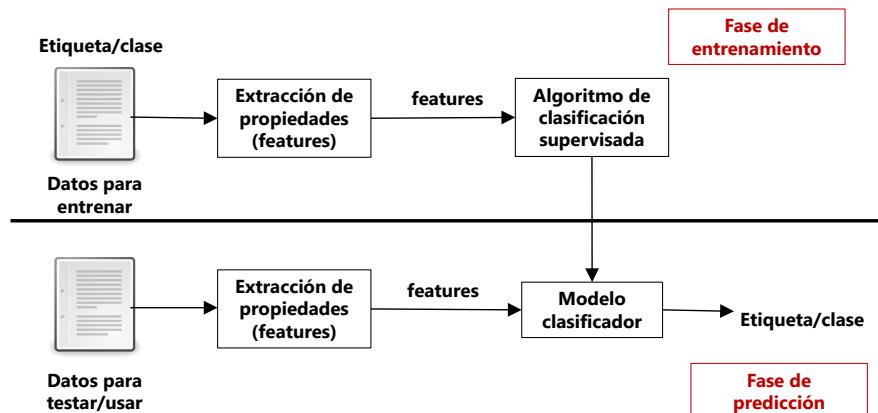


Figura 3.69: Fases de un algoritmo de clasificación supervisado.

En líneas generales, la clasificación mediante SVMs está basada en la búsqueda de un hiperplano (o varios) que permita separar, con la mayor precisión posible, los elementos de una colección de datos según la clase a la que pertenecen.

A modo de ejemplo, véase la Figura 3.70 que contiene varios puntos, los cuales representan datos de la colección que se pueden categorizar en dos clases, azul y verde. Para separar ambas clases se pueden usar varios hiperplanos como se puede ver; sin embargo, algunos son más óptimos que los otros.

Para medir esa optimalidad, se usan los vectores de soporte, que son los puntos que definen el margen máximo de separación entre el hiperplano y las clases que separan, que vienen determinadas por la línea punteada que se puede ver en la Figura 3.71, que se denomina *margen máximo* del hiperplano.

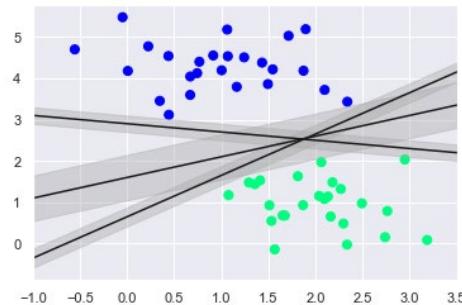


Figura 3.70: Varios hiperplanos para separar clases.

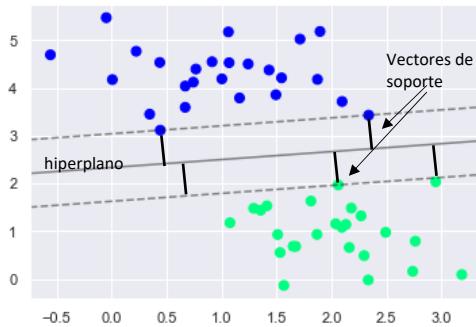


Figura 3.71: Vectores de soporte.

El rendimiento de los SVM es especialmente bueno en aquellas colecciones de datos donde la separación entre clases es relativamente clara; sin embargo, en casos como el de la Figura 3.72 donde la separación entre clases es compleja, el rendimiento decae.

Con el fin de ajustar el hiperplano, se recurre a la modificación de los hiperparámetros típicos de este algoritmo. Entre estos destacan:

- **Kernel:** La función principal del núcleo (kernel) es transformar los datos de entrada en otra forma más útil. Hay varios tipos de funciones como la lineal, la polinómica o la de base radial (RBF). Las funciones polinómicas y RBF son útiles para los hiperplanos no lineales. Los núcleos polinómicos y RBF calculan la línea de separación en la dimensión superior. En algunas aplicaciones, se sugiere utilizar un kernel más complejo para separar las clases que son curvas o no lineales. Esta transformación puede conducir a clasificadores más precisos.

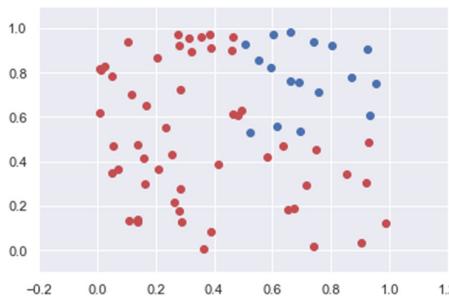


Figura 3.72: Ejemplo de clases difícilmente separables por un hiperplano.

- **Regularización:** El parámetro C se utiliza para establecer el mecanismo de margen/regularización. C es el parámetro de penalización que representa la mala clasificación o el término de error. El término de error o clasificación errónea establece el nivel de error asumible en el proceso de optimización. De esta forma se controla el equilibrio entre el límite de decisión y el término de clasificación errónea. Un valor menor de C crea un hiperplano de margen pequeño y un valor mayor de C crea un hiperplano de margen mayor.
- **Gamma:** Modela la parametrización del kernel. Un valor más bajo de $gamma$ se ajustará de forma imprecisa al conjunto de datos de entrenamiento, mientras que un valor más alto de $gamma$ se ajustará exactamente al conjunto de datos de entrenamiento, lo que provoca un sobreajuste. Es decir, se puede decir que un valor bajo de $gamma$ considera solo los puntos cercanos en el cálculo de la línea de separación, mientras que un valor alto de $gamma$ considera todos los puntos de datos en el cálculo de la línea de separación.

Árboles de decisión

Los árboles de decisión permiten identificar conceptos o clases de objetos a partir de las características de un conjunto de ejemplos que los representan, mediante razonamiento por inducción [Qui96].

La información extraída de los ejemplos se organiza jerárquicamente en forma de árbol, donde los nodos se determinan en función de una pregunta sobre las características (features) de los datos de ejemplo. Las respuestas a estas preguntas son excluyentes entre sí, por lo que cada ejemplo seguirá un único camino y por tanto, se “clasificará” en un único nodo.

Un árbol de decisión tiene un nodo raíz, nodos intermedios y hojas. Las hojas representan los “conceptos” extraídos de manera automática, es decir, una hoja en el árbol corresponde a un conjunto de datos que representa una sola clase. La clase de la hoja que se asigna es la clase de la mayoría de los ejemplos en ella.

Una vez construido un árbol de decisión, un nuevo ejemplo desconocido será representante de la clase en donde caiga recorriendo el árbol desarrollado desde la raíz a las hojas.

La propiedad o feature que se selecciona como separador debe de cumplir el objetivo de que su posición en algún punto del árbol genere un subárbol tan simple como sea posible y dé una concreta clasificación. De esta forma, es necesario tener un criterio para seleccionar las propiedades o features importantes para la clasificación, así como el orden de uso de los mismos. En función de esto, diferentes árboles de decisión pueden construirse.

Los pasos para implementar este algoritmo serían:

1. Calcular la calidad de un nodo: criterio de selección de separadores, impureza, ganancia de información, etc.
2. Si un conjunto de datos en un nodo no tiene suficiente calidad, se calcula el mejor atributo separador, este se añade a la base de reglas, y se utiliza para dividir el conjunto de ejemplos en al menos dos conjuntos/nodos nuevos de mayor calidad.
3. Repetir el proceso hasta que se cumpla el criterio de parada.

Otro aspecto/hiperparámetro importante son los mecanismos de selección de propiedades/features, los cuales dependen de la calidad del nodo (ganancia de información) que generan. Algunos ejemplos son:

- **Entropía:** Si la variable es categórica (ej: si-no, verde-rojo-azul ...), se obtiene sumando el índice de entropía de cada una de sus clases. En cambio, si es numérica, previamente se obtiene uno o varios puntos de corte por métodos iterativos. Se hace con todas las propiedades/features y se elegirá aquella variable que tenga menor índice de entropía. La entropía se define matemáticamente como:

$$H(S) = - \sum_{i=1}^C p_i \log p_i$$

donde p_i es la probabilidad de aparición de la clase i dentro del conjunto de clases C .

- **Índice de Gini:** Mide la probabilidad de no sacar dos registros con el mismo valor para la variable objetivo dentro del mismo nodo. Cuanto menor es el índice de Gini, mayor es la pureza del corte. Por lo tanto, el corte propuesto en primer lugar será el de aquella variable que tenga menor valor del índice de Gini. Matemáticamente, se define como:

$$\text{Gini} = \sum_{i=1}^C p_i (1 - p_i)$$

donde C es el número total de clases y p_i la probabilidad de encontrar elementos de la clase i en el conjunto de elementos que se está analizando.

Además, otro de los aspectos/hiperparámetros más importantes al construir un árbol de decisión es el criterio de parada, cuándo debe dejar de ramificarse el árbol. Algunos de los más habituales son:

- **Máximo número de ramas por nodo:** Número de ramas máximo en que puede dividirse un nodo.
- **Número mínimo de observaciones por nodo final:** Número mínimo de observaciones que tiene que tener un nodo final para que se construya la regla.
- **Número mínimo de observaciones para dividir un nodo:** Número mínimo de observaciones que tiene que tener un nodo para que se pueda cortar por la variable seleccionada.
- **Variables discriminantes:** El hecho de no encontrar una propiedad/feature que sea lo suficientemente discriminante en el nodo es motivo de parada.

Para reducir la complejidad de un árbol, este puede podarse (pruning) quedándose con los nodos más importantes y eliminando los redundantes. Es decir, pueden ponerse criterios/hiperparámetros para evitar una sobre-ramificación de los árboles.

Redes neuronales

Otros de los algoritmos clásicos que se verá en otras secciones son las redes neuronales. Es un algoritmo antiguo que estaba perdiendo auge porque, aunque ofrece buenos resultados, su complejidad computacional lo hace muy costoso. Sin embargo, la aparición de nuevas herramientas hardware más potentes como nuevos microprocesadores o GPUs (graphics processing units), que permiten la parallelización de procesos, han permitido el desarrollo de nuevas arquitecturas de redes neuronales. Este hecho ha dado lugar a un nuevo campo dentro de la Ciencia de Datos como es el Deep Learning. En la Figura 3.73 puede verse una relación entre estos campos. En este caso, la Ciencia de Datos se puede definir como el mero arte de transformar datos en información con el fin de sacarles rédito.

Otros

Existen muchos otros algoritmos: Knn, DBSCAN, random forest, naive Bayes, etc. Para entender en detalle muchos de ellos se recomienda la siguiente bibliografía: [Ger19, Bur19, Bis06].

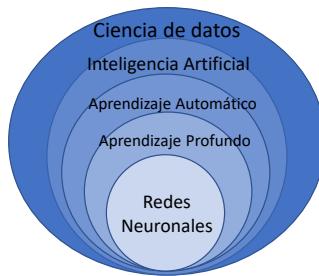


Figura 3.73: Redes neuronales y su relación con el resto de campos

3.3.4. Entornos de modelado para Inteligencia Artificial

Como se ha visto anteriormente, hay muchas librerías, plataformas o servicios en la nube que implementan múltiples soluciones relacionadas con la IA pero que suelen ser dependientes de un lenguaje de programación o una arquitectura hardware concreta. Es por ello, que el desarrollo de modelos de IA puede resultar costoso en este tipo de plataformas, porque aunque el algoritmo en sí, ya viene definido como una caja negra, simplemente puede consistir en invocar una función como puede verse en ejemplos desarrolladas en Python, es necesario implementar las tareas asociadas a la entrada y salida de datos, la visualización de los mismos, y la configuración de los hiperparámetros; y todo ello, depende de la plataforma y lenguaje de programación que se esté utilizando.

Por esta razón, surgen otro tipo de herramientas centradas principalmente en el modelado, siguiendo al filosofía de la ingeniería dirigida por modelos (model-driven engineering) aplicada al campo de la IA. Son herramientas con una orientación más visual, que intenta abstraer al desarrollador de un gran número de tareas como las asociadas a la entrada y salida de datos, la visualización o evaluación de los mismos. De esta forma, el proceso puede centrarse más en el desarrollo del modelo y la configuración de los hiperparámetros.

Este tipo de funcionalidades para el desarrollo rápido de prototipos de Aprendizaje Automático permite desarrollos más ágiles, sencillos y dinámicos; sin embargo, se pierde flexibilidad a la hora configurar ciertas funcionalidad y parámetros, lo que conlleva a aplicaciones menos optimizadas. Por ello, también incorporan funcionalidades adicionales en las que, algunas partes del sistema se puede directamente programar en lenguajes de programación como Python.

Algunas de estas herramientas son:

Azure Machine Learning Studio

Desarrollado por Microsoft, Azure Machine Learning Studio³⁶ es una de las herramientas más completas que pueden encontrarse. Sin necesidad de codificar en ningún lenguaje de programación, simplemente mediante operaciones arrastrar-y-soltar (drag-and-drop) se pueden configurar distintos modelos para diferentes conjuntos de datos de entrada y, se pueden visualizar sus salidas y métricas de evaluación.

SPSS Modeler

Desarrollado por IBM, SPSS Modeler³⁷ es parte del proyecto IBM Watson Studio³⁸, del que se habló anteriormente. El sistema Watson fue diseñado con el objetivo de contestar preguntas gracias a grandes volúmenes de datos y potentes funcionalidades de Procesamiento de Lenguaje Natural e IA incluidas en el IBM Watson Studio. Este último permite el desarrollo de aplicaciones para Ciencia de Datos usando lenguajes como Python o R, pero incluye módulos como SPSS Modeler que permiten el desarrollo visual de modelos sin necesidad de codificar las soluciones.

Sagemaker

Amazon es también la otra gran compañía tecnológica que ha querido desarrollar su propia herramienta, similar a las dos anteriores, en la que se permiten definir flujos (workflows) de Aprendizaje Automático, y que se denominada Sagemaker³⁹.

RapidMiner

Antiguamente conocida como YALE (Yet Another Learning Environment), RapidMiner⁴⁰ es una de las herramientas visuales más potentes que existen para el campo de la Minería de Datos. Ofrece gran flexibilidad para conectarse a múltiples fuentes de información y explotar su información a través de una galería de algoritmos de Aprendizaje Automático, gracias de la inclusión del framework de código abierto desarrollado en Java llamado Weka⁴¹ (Waikato Environment for Knowledge Analysis). Además de incluir herramientas para el diseño de forma visual, incluye otras funcionalidades para la implementación de módulos en Python y R.

³⁶ Enlace: azure.microsoft.com/en-us/services/machine-learning-studio/

³⁷ Enlace: ibm.com/products/spss-modeler

³⁸ Enlace: ibm.com/cloud/watson-studio

³⁹ Enlace: aws.amazon.com/es/sagemaker/

⁴⁰ Enlace: rapidminer.com

⁴¹ Enlace: cs.waikato.ac.nz/ml/weka/

Orange

Surge como un proyecto liderado por la universidad Liubliana, hasta convertir a Orange⁴² en una de las herramientas más potentes que existe para la Minería de Datos. Está desarrollado en C++ aunque permite el desarrollo de módulos y aplicaciones en Python. Su gran cualidad es, otra vez, la facilidad del desarrollo de forma visual e intuitiva sin grandes conocimientos de programación.

Knime

Es un conjunto de herramientas de código abierto gratuito que trae predefinidas un gran número de tareas para los científicos de datos que, permiten el modelado y desarrollo de rápido de aplicaciones. Existen dos versiones, una genérica que incluye funcionalidades típicas de la Ciencia de Datos⁴³ y una especializada en Deep Learning⁴⁴, dado que es una de las técnicas que más importancia tiene en la actualidad. Esta última permite definir cualquier arquitectura de redes neuronales de forma sencilla y visual. Está desarrollada en Java y permite la integración del Knime SDK con Eclipse⁴⁵.

Dianne

Dianne⁴⁶ es otra plataforma de código abierto especializada en redes neuronales que permite el desarrollo de complejas arquitecturas de redes neurales, así como su entrenamiento y validación de una forma sencilla y visual. Aunque originalmente desarrollada en Java, incluye otras librerías nativas especialmente desarrolladas para optimizar el rendimiento de algoritmos y operaciones típicas de las redes neuronales.

Dataiku

Dataiku⁴⁷ es una plataforma comercial, aunque tiene algunas funcionalidades gratuitas, que permite la visualización y el análisis de datos, incluyendo una gran librería de algoritmos típicos del Aprendizaje Automático, que permite el acceso a funcionalidades de otras librerías como Keras o TensorFlow, para el modelado de redes neuronales. Permite el desarrollo visual así como la implementación en lenguajes como R y Python, incluyendo la inclusión de cuadernos de Jupyter, y el

⁴² Enlace: <https://orangedatamining.com>

⁴³ Enlace: knime.com/knime-for-data-scientists

⁴⁴ Enlace: knime.com/deeplearning

⁴⁵ Enlace: eclipse.org

⁴⁶ Enlace: dianne.intec.ugent.be/

⁴⁷ Enlace: <https://www.dataiku.com/>

uso de potentes herramientas para el almacenamiento y procesamiento de grandes colecciones de datos incluidas en el proyecto Hadoop⁴⁸, como Hive o Impala. Las aplicaciones desarrolladas se pueden desplegar fácilmente mediante tecnología API Rest.

3.4. Recolección de datos

Los sistemas de aprendizaje automático nos permiten identificar patrones en los datos que nos permitan realizar predicciones sobre ellos. Para asegurarnos de que esas predicciones sean correctas, debemos construir el *dataset* y transformar los datos de manera apropiada para su utilización.

Los datos que utilizamos para entrenar nuestros modelos representan la piedra angular de los sistemas de aprendizaje automático. Si los datos de partida no son lo suficientemente buenos, no importa el algoritmo o la arquitectura de aprendizaje automático que implementemos, ya que las funciones que obtengamos no se ajustarán a lo que esperamos. La calidad y el tamaño del *dataset* importan mucho más que cualquier ajuste que hagamos sobre los algoritmos de entrenamiento.

Dada es su importancia, que los equipos de desarrollo de servicios como *Google Translate* han obtenido mayores avances utilizando mejores conjuntos de datos que ajustando el modelo de aprendizaje automático. Los *datasets* que contengan datos incorrectos pueden influir en que el modelo aprenda patrones incorrectos, a pesar de las técnicas de modelado utilizadas.

La preparación de los datos y la ingeniería de propiedades se lleva a cabo mediante dos procesos bien definidos, i) la construcción del *dataset* y ii) la transformación de los datos. En este tema se introduce el primero de los procesos a través de la recolección de los datos en bruto, la identificación de las propiedades y su etiquetado, la estrategia de muestreo de datos y la división de los datos.



Típicamente el 80 % del tiempo dedicado a un proyecto de aprendizaje automático se emplea en construir *datasets* y aplicar transformaciones a los datos.

3.4.1. Visión general

Como se ha mencionado, para construir un *dataset* (antes de realizar la transformación de los datos), conviene seguir una serie de pasos:

1. Recolectar los datos en bruto.
2. Identificar las propiedades y realizar su etiquetado.
3. Elegir una estrategia de muestreo de datos.

⁴⁸ Enlace: hadoop.apache.org

4. Dividir los datos.

La forma en que apliquemos estos pasos dependerá en gran medida del problema de aprendizaje automático que intentemos resolver. Por ejemplo, una manera típica de proceder a la hora de seleccionar qué propiedades del *dataset* se utilizarán para entrenar el modelo, es seleccionar una o dos propiedades que tengan una fuerte correlación con la variable que queremos predecir, de tal forma que podamos verificar que nuestro modelo funciona como esperamos. Si empezáramos añadiendo muchas propiedades al proceso de entrenamiento, podríamos estar añadiendo demasiada diversidad y por tanto dificultando el proceso de aprendizaje al modelo.



En términos de aprendizaje automático, i) una propiedad o *feature* representaría una variable de entrada o columna del *dataset*, mientras que ii) una etiqueta o *label* representaría un resultado de salida para un conjunto de propiedades concreto. Así, un *dataset* se dice que está *etiquetado* si contiene una columna que incluye el resultado que queremos obtener para un ejemplo (fila) concreto.

3.4.2. El tamaño y la calidad del *dataset*

Como regla general, el modelo debe entrenarse con al menos un orden de magnitud de ejemplos superior a los parámetros entrenables. Los modelos sencillos en conjuntos de datos grandes suelen superar a los modelos complejos en conjuntos de datos pequeños. Qué entendemos por grandes conjuntos de datos dependerá del proyecto, por ejemplo:

- Conjunto de datos *flor iris*⁴⁹: 150 ejemplos.
- MovieLens⁵⁰: Más de 20 millones de ejemplos.
- Google Gmail SmartReply⁵¹: 238 millones de ejemplos (sólo para conjunto de datos de entrenamiento).
- Google Books Ngram⁵²: 468 mil millones de ejemplos.
- Google Translate: billones de ejemplos.

Además de cantidad, también deberemos asegurarnos de que la calidad de los datos sea adecuada. En este contexto, datos de calidad nos referimos a que los datos sean lo suficientemente buenos como para abordar las tareas a las que nos enfrentamos. Para medir la calidad de los datos podemos valorar las dimensiones de fiabilidad, representación de propiedades y minimización del sesgo.

⁴⁹ Enlace: <https://archive.ics.uci.edu/ml/datasets/iris>

⁵⁰ Enlace: <https://grouplens.org/datasets/movielens/20m/>

⁵¹ Enlace: <https://static.googleusercontent.com/media/research.google.com/en/pubs/archive/45189.pdf>

⁵² Enlace: <https://storage.googleapis.com/books/ngrams/books/datasetsv3.html>

Fiabilidad

La fiabilidad se refiere al grado de confianza en los datos. Un modelo entrenado con un conjunto de datos fiables tiene más probabilidades de producir predicciones útiles que un modelo entrenado con datos poco fiables. Para medir la fiabilidad, hay que determinar:

- Frecuencia de aparición de errores en el etiquetado. Por ejemplo, si los datos han sido etiquetados manualmente, es posible que se hayan cometido errores al hacerlo.
- Si las propiedades son «ruidosas» (por ejemplo, mediciones de un GPS) y, en ese caso, asumirlas o *reducirlas* (por ejemplo, recogiendo más datos).
- Si los datos se encuentran correctamente filtrados para el contexto del problema. Por ejemplo, en caso de querer construir un sistema de detección de *spam*, resultaría apropiado incluir consultas de búsqueda realizadas por *bots*. En otro contexto (por ejemplo, mejoras en los resultados de búsqueda para los usuarios) no sería apropiado incluir dichos datos.

Así, podríamos encontrar ejemplos de *datasets* no fiables que presenten:

- Valores omitidos, por ejemplo, al olvidarse introducir el teléfono de un individuo.
- Ejemplos duplicados, por ejemplo, cuando un servidor ha almacenado por el mismo archivo de *log* por error.
- Etiquetas incorrectas, por ejemplo, cuando alguien ha clasificado una fotografía de un abeto como un olivo.
- Valores de propiedades incorrectos, por ejemplo, cuando se ha escrito incorrectamente el año en una fecha de nacimiento.

Representación de propiedades

La representación de propiedades forma parte de la ingeniería de propiedades, y se encarga de establecer una relación entre los datos originales y los datos que proporcionamos al algoritmo de aprendizaje automático. En este caso, se debería considerar:

- Cómo se van a convertir los datos para que sean utilizados por el modelo.
- Si se deberían de normalizar los valores numéricos.
- Cómo se deberían tratar los valores atípicos.

Minimización del sesgo

A la hora de realizar predicciones sobre un modelo ya entrenado, es posible que no obtengamos los resultados esperados si comparamos con los datos de entrenamiento. Esto es lo que se conoce como el sesgo entre el entrenamiento y el despliegue. Las causas del sesgo pueden ser imperceptibles, pero tienen efectos decisivos en los resultados. Siempre se deben considerar los datos de los que dispone el modelo en el momento de la predicción; durante el entrenamiento, hay que utilizar sólo las propiedades que estén disponibles cuando el modelo se esté utilizando en un entorno de producción, y asegurarse de que el conjunto de entrenamiento es representativo de los datos reales.

3.4.3. Combinando fuentes de datos

Cuando construimos un *dataset* es posible que tengamos que hacerlo a partir de distintas fuentes de datos como, por ejemplo, eventos que suceden en un momento específico (p. ej., la dirección IP desde la que se realizó una consulta en una fecha determinada), o atributos (p. ej., datos demográficos de los usuarios).

Los atributos no son específicos de un evento o un momento en el tiempo, pero pueden ser útiles para hacer predicciones. Para aquellas predicciones que no estén asociadas a un evento específico (p. ej., la pérdida de clientes, que implica un rango de tiempo en lugar de un momento individual), los atributos pueden ser, de hecho, los únicos tipos de datos que tengamos. Estos dos tipos de datos (atributos y eventos) están relacionados. Por ejemplo, se puede crear un tipo de atributo combinando varios eventos mediante la agregación de datos estadísticos. En este caso, es posible analizar varios eventos para crear un único atributo para un usuario para, por ejemplo, calcular la frecuencia con la que el usuario realiza consultas a nuestro sistema o la frecuencia media de *clicks* sobre un anuncio concreto.

Las fuentes de datos suelen encontrarse en ubicaciones diferentes. Cuando recojamos datos para entrenar nuestro modelo de aprendizaje automático tendremos que combinar todas las fuentes de datos de las que dispongamos para crear el *dataset*. Por ejemplo, podríamos aprovechar los archivos de *logs* para extraer el identificador del usuario y la marca de tiempo de un evento concreto para buscar los atributos de un usuario en un momento determinado.

Cabe destacar, que las fuentes de datos pueden ser utilizadas de manera *online* u *offline*, dependiendo de cómo vayamos a entrenar nuestro modelo de aprendizaje automático.

Así, los *modelos dinámicos* serían aquellos que se entranan de manera *online*, es decir, el modelo se entrena continuamente a partir de los datos de entrada que el sistema va recibiendo. En este tipo de entrenamiento resulta necesario considerar la latencia con la que el sistema es capaz de procesar los datos de entrada.

Por otra parte, los *modelos estáticos* serían aquellos que se entrena de manera *offline*, es decir, el modelo se entrena una sola vez y, una vez entrenado, se utilizará durante un tiempo hasta que volvamos a entrenarlo. En este caso se puede repetir el mismo proceso de entrenamiento inicial ya que no existen restricciones de cómputo.

3.4.4. Etiquetado

A la hora de trabajar con datos para entrenar un modelo de aprendizaje automático, resulta mucho más sencillo hacerlo si las etiquetas están bien definidas. Estas etiquetas podemos clasificarlas en directas o derivadas:

- Etiquetas directas: representan los resultados exactos que se quieren predecir. Por ejemplo, si se quisiera predecir si un usuario se va a dar de baja de un servicio, una etiqueta directa podría ser «el usuario se ha dado de baja».
- Etiquetas derivadas: en este caso, no se mide directamente el resultado que se quiere predecir. Por ejemplo, una etiqueta derivada podría ser «el usuario ha cancelado el servicio telefonía móvil». En este caso la precisión del modelo para predecir si un usuario se ha dado de baja dependerá de la relación que exista entre la etiqueta derivada y dicha predicción.

Del mismo modo, la salida de un modelo podría ser un evento o un atributo, resultando en **etiquetas directas para eventos** (p. ej., ¿el usuario seleccionó el primer resultado de búsqueda?) y **etiquetas directas para atributos** (p. ej., ¿el anunciante invertirá más de 50€ la semana que viene?).

En el caso de los eventos, las etiquetas directas suelen ser sencillas, ya que se puede registrar el comportamiento del usuario durante el evento para utilizarlo directamente como etiqueta.

Por otra parte, las etiquetas directas para los atributos suelen ser más complejas de identificar. Si consideramos el ejemplo mencionado anteriormente (¿el anunciante invertirá más de 50€ la semana que viene?), tendremos que utilizar los datos de los días anteriores para predecir lo que ocurrirá en los días siguientes, sin obviar los efectos estacionales o cílicos; por ejemplo, los anunciantes podrían invertir más dinero los fines de semana o en épocas festivas. Por esta razón, podría ser preferible utilizar una ventana de tiempo mayor, o utilizar la fecha como propiedad para que el modelo pueda aprender a reconocer dichos efectos anuales.

En cualquier caso, resulta imprescindible disponer de información del pasado para poder predecir el futuro. Pero, ¿qué ocurre si no disponemos de dicha información del pasado? Este caso podría darse cuando nuestro producto no exista todavía, por lo que no tenemos ningún dato registrado todavía. En ese caso, podríamos realizar alguna(s) de las siguientes acciones:

- Utilizar una heurística para el lanzamiento inicial y luego entrenar un sistema basado en los datos registrados.
- Utilizar los datos de un problema similar para lanzar el sistema inicialmente.

- Utilizar personas reales para que generen los datos necesarios mientras completan tareas concretas.

3.4.5. Muestreo de datos

A menudo resulta complejo reunir suficientes datos para un proyecto de aprendizaje automático. Sin embargo, a veces hay demasiados datos y hay que seleccionar un subconjunto de ejemplos para el entrenamiento. La selección de dicho subconjunto de datos dependerá de lo que se quiera predecir y las propiedades que se utilizarán para ello.

Si los datos contienen información personal identificable, es posible que haya que filtrarla debido a las políticas de protección de datos que se estén aplicando, eliminando así aquellas propiedades que sean poco frecuentes. Este filtrado sesgará la distribución haciendo que se pierda información de los valores más alejados de la media. Esto puede resultar útil porque las propiedades poco frecuentes son complejas de aprender, pero es importante tener en cuenta que el conjunto de datos se encontrará sesgado hacia aquellas propiedades más frecuentes, por lo que el modelo no se comportará correctamente cuando lo utilicemos para realizar predicciones sobre propiedades poco frecuentes.

3.4.6. Datos no balanceados

Un conjunto de datos de clasificación con una proporción de clases sesgada se denomina no balanceado o desequilibrado. Las clases que constituyen una gran proporción del conjunto de datos se denominan clases mayoritarias, mientras que las que representan una proporción menor serían clases minoritarias. Por ejemplo, dado un *dataset* que contiene un 99 % de ejemplos etiquetados como no *spam* y un 1 % etiquetados como *spam*, las etiquetas de *spam* serán la clase minoritaria.

Así, podemos identificar tres niveles de desequilibrio dependiendo de la proporción de clases minoritarias que tengamos en nuestro *dataset*:

- Leve: si la proporción de clases minoritarias representa entre el 20 % y el 40 % del conjunto de datos.
- Moderado: si la proporción de clases minoritarias representa entre el 1 % y el 20 % del conjunto de datos.
- Extremo: si la proporción de clases minoritarias representa menos del 1 % del conjunto de datos.

Dependiendo de cómo de desequilibrado se encuentre nuestro conjunto de datos, tendremos que aplicar alguna técnica de muestreo para reducir dicho desequilibrio. Por ejemplo, considera el ejemplo de la Figura 3.74 donde se muestra un modelo que puede detectar casos de fraude. Los casos donde se detecta fraude ocurren una vez por cada 200 transacciones, por lo que en la distribución real, alrededor del 0,5 % de los datos serán positivos.

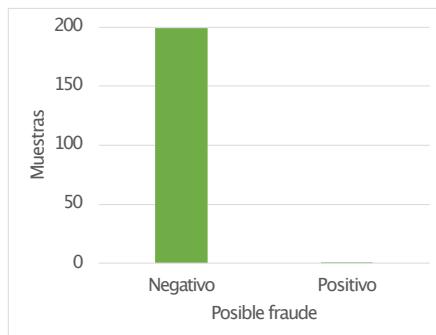


Figura 3.74: Dataset donde los casos de fraude aparecen en 1 de cada 200 transacciones.

Con tan pocos positivos en relación con los negativos, el modelo de entrenamiento pasará la mayor parte de su tiempo en los ejemplos negativos y no aprenderá lo suficiente de los positivos.

Una manera efectiva de tratar con datos no balanceados es reducir la muestra (*downsampling*) y sobreponerla (*upweighting*):

- Reducir la muestra (en este contexto) significa entrenar con un subconjunto mucho menor de los ejemplos de la clase mayoritaria.
- Sobreponer significa añadir un valor de ponderación a la muestra reducida equivalente al factor por el que se ha reducido la muestra.

Veámoslo paso a paso para el ejemplo de la detección de fraude:

- **Paso 1: reducir la muestra de la clase mayoritaria.** Si redujéramos la muestra de la clase mayoritaria en un factor de 20, conseguiríamos mejorar el equilibrio de los datos en 1 positivo por cada 10 negativos (10 %).
- **Paso 2: sobreponer la muestra reducida.** En este paso se añaden los pesos a la muestra reducida; dado que hemos reducido la muestra en un factor de 20, tendremos que sobreponer con un peso de 20.

De esta forma, sobreponer con un peso de 20 significa que el modelo tratará a los ejemplos de fraude negativo como si fueran 20 veces más importantes que antes de reducir la muestra.

Puede parecer extraño añadir pesos después de reducir la muestra, ya que estamos intentando que nuestro modelo mejore la clase minoritaria, ¿por qué sobreponer entonces la clase mayoritaria? Estas son algunas ventajas:

- **Convergencia más rápida:** durante el entrenamiento, vemos la clase minoritaria más a menudo, lo que ayudará a que el modelo converja más rápido.

- **Espacio en disco:** al consolidar la clase mayoritaria en menos ejemplos con mayores pesos, empleamos menos espacio en disco para almacenarlos. Este ahorro permite disponer de más espacio en disco para la clase minoritaria, por lo que podemos recoger un mayor número y una mayor variedad de ejemplos de esa clase.
- **Calibración:** la sobreponderación garantiza que nuestro modelo siga calibrado; los resultados pueden seguir interpretándose como probabilidades.

3.4.7. División de los datos

El objetivo principal de un *dataset* es poder entrenar modelos de aprendizaje automático que se puedan generalizar para nuevos datos de entrada. Para ello, se suele dividir el *dataset* en el conjunto de datos de entrenamiento, el conjunto de datos de validación, y el conjunto de datos de prueba.

Esta división de los datos en los tres conjuntos de datos se suele realizar de manera aleatoria. Sin embargo, en el caso de que los datos se encuentren agrupados, sería necesario definir alguna estrategia de división que nos permita mantener los datos del mismo grupo dentro del mismo conjunto de datos, ya sea el de entrenamiento, validación o pruebas. En este tipo de *datasets*, dividir los datos que pertenecen al mismo grupo en conjuntos de datos distintos puede dar lugar a entrenar modelos que sean menos precisos.

Una posible solución para ello es, por ejemplo, dividir los datos en función de alguna propiedad que indique que los datos se encuentran agrupados temporalmente.

3.4.8. Caso práctico: construcción de un *dataset*

En este caso práctico se plantea un ejemplo de construcción de un *dataset* a partir de un proceso automático que nos permita recopilar información de la base de datos de películas del sitio web de IMDB⁵³. El *dataset* resultante podremos utilizarlo para entrenar un modelo de aprendizaje automático que nos permita realizar algunas predicciones sobre los datos.

Inicialización del entorno

Para construir nuestro *dataset* de películas tendremos que desarrollar una aplicación que sea capaz de analizar el sitio web de IMDB automáticamente. En el entorno de Python existen varias alternativas, aunque nosotros vamos a utilizar el *framework* de Scrapy⁵⁴ para ello. Se trata de una potente herramienta que nos permitirá explorar los sitios web de manera programática utilizando para ello expresiones XPath o selectores CSS.

⁵³ Enlace: <https://www.imdb.com/>

⁵⁴ Enlace: <https://scrapy.org/>

Comenzaremos creando un nuevo proyecto utilizando la herramienta Poetry, la cual ya se ha introducido anteriormente en temas anteriores, y añadiremos las bibliotecas de desarrollo que utilizaremos:

```
$ poetry new caso-practico  
$ cd caso-practico  
$ poetry add scrapy pandas ipykernel
```

En nuestro caso, hemos añadido la biblioteca del *framework* de Scrapy, la biblioteca Pandas para análisis de datos y el módulo *ipykernel* para trabajar con cuadernos Jupyter.

Una vez inicializado el entorno, crearemos un directorio *notebooks* y, dentro de este nuevo directorio, un nuevo cuaderno Jupyter donde realizaremos el desarrollo paso a paso:

```
$ mkdir notebooks  
$ touch notebooks/CasoPractico.ipynb
```

Una vez creado nuestro cuaderno, lo abriremos con Visual Studio Code y configuraremos el entorno de ejecución para que utilice nuestro entorno virtual que acabamos de crear con Poetry. Por último, inicializaremos el directorio *caso_practico* como la raíz del proyecto para nuestro desarrollo con Scrapy:

```
$ poetry run scrapy startproject imdb caso_practico/
```



En lugar de utilizar `$ poetry run` para ejecutar comandos utilizando nuestro entorno virtual, podemos simplemente escribir los comandos que queramos ejecutar en nuestro cuaderno Jupyter precedidos por un signo de cierre de exclamación (!).

Recopilación de los datos

Una vez inicializado nuestro entorno de desarrollo, podemos pasar a la fase de recopilación de datos. Para ello, primero tendremos que saber qué información vamos a recopilar sobre las películas y desde dónde lo vamos a hacer. Esto lo haremos analizando manualmente el sitio web de IMDB para encontrar un punto de entrada que podamos utilizar fácilmente para explorar todas las películas que queramos. El sitio web nos proporciona en Enlace: <https://www.imdb.com/search/title/> una herramienta de búsqueda avanzada donde podremos filtrar por los campos que deseemos. En nuestro caso, filtraremos por *Feature Film*, con fecha de lanzamiento (*release*) entre enero y diciembre de 2020, una duración (*runtime*) como mínimo de 1 minuto (para que la búsqueda únicamente nos muestre películas que tienen una duración definida), y el máximo número de películas por página (en este caso 250) ordenadas por fecha de lanzamiento (ascendente).

La búsqueda anterior nos redirigirá a una página con el listado de películas filtrado. En esta página apuntaremos la URL para indicársela a Scrapy. En nuestro caso, la URL tendrá un aspecto similar a este Enlace: https://www.imdb.com/search/title/?title_type=feature&release_date=2020-01-01,2020-12-31&runtime=1,&sort=release_date,asc&count=250.

Una vez tengamos la URL desde la que empezaremos a recopilar información sobre las películas, nos dirigiremos a la raíz del proyecto de Scrapy y ejecutaremos el siguiente comando para generar el código base de una nueva *araña* de Scrapy que se ejecutará automáticamente para extraer la información de dicha URL:

```
$ cd caso_practico
$ poetry run scrapy genspider "movies-released-in-2020"
www.imdb.com/search/title/?title_type=feature&release_date=2020-01-01,2020-
12-31&runtime=1,&sort=release_date,asc&count=250"
```

Tras ejecutar este comando veremos como se ha creado un nuevo archivo *movies_released_in_2020.py* en el directorio *imdb/spiders/*. Este archivo contendrá todo el código que utilizaremos para extraer la información de las películas desde la URL indicada. Lo primero que haremos será abrirlo con VS Code y ajustaremos la variable *allowed_domains* para que únicamente almacene el nombre de dominio del sitio web, y la variable *start_urls* para que cambiar el protocolo de *http* a *https*. Tras esto, el código debería tener el siguiente aspecto:

```
1 class MoviesReleasedIn2020Spider(scrapy.Spider):
2     name = 'movies-released-in-2020'
3     allowed_domains = ['www.imdb.com']
4     start_urls = ['https://www.imdb.com/search/title/?title_type=feature&release_date
5                   =2021-01-01,2021-12-31&runtime=1,&sort=release_date,asc&count=250/']
6
7     def parse(self, response):
8         pass
```

Cuando se ejecute la araña, ésta descargará la página web que hayamos indicado en la variable *start_urls* y se ejecutará el método *parse(...)* con el documento HTML en el parámetro *response*. El documento HTML devuelto en este parámetro *response* será el que exploraremos mediante XPath utilizando el método *xpath()* o mediante selectores CSS utilizando el método *css()*. En nuestro caso, optaremos por la segunda opción por simplicidad.

Antes de continuar, necesitaremos volver a la búsqueda que acabamos de realizar para examinar los datos que recopilaremos de cada una de las películas. En nuestro caso hemos optado por extraer la información del título de la película, año de lanzamiento, duración, género(s), sinopsis, directores, actores y puntuación (ver Figura 3.75).

Para acceder a los campos de cada película deberemos poder referenciar cada película de manera única mediante una ruta XPath o un selector CSS. Para ello, abriremos las herramientas de desarrollador de nuestro navegador y exploraremos el árbol de elementos hasta identificar el elemento padre que contenga la informa-

1. **The Forgotten** (I) (2020)

58 min | Drama

[Rate this](#)

[Add a Plot](#)

Director: Lyuben Kanev | Stars: Shukraidin Asenov, Iliya Ilchev, Muhamed Kyamil, Marselo Martinov

2. **The Purser** (2020)

91 min | Fantasy

[Rate this](#)

An accountant takes LSD and disappears with incriminating evidence against his mysterious employers. The Purser is tasked with finding him and destroying the evidence.

Director: Jacques Aldridge | Stars: Jacques Aldridge, Forest Baker, Amy Marie Bowersox, Dawne Ellison

3. **Ghost Stories** (2020)

144 min | Horror, Thriller

[Rate this](#)

The winning team of LUST STORIES unite to tell some spine - chilling tales. Ghost Stories is an upcoming 2020 Indian anthology horror film, consisting of four short film segments.

Directors: Zoya Akhtar, Dibakar Banerjee, Karan Johar, Anurag Kashyap | Stars: Janhvi Kapoor, Sanya Malhotra, Amrita Subhash, Vijay Varma

Votes: 4.949

Figura 3.75: Datos a extraer de las películas para construir nuestro *dataset*

ción de cada película. En la Figura 3.76 podemos ver cómo este elemento es aquel que contiene la clase *lister-item* y es hijo de un elemento que contiene la clase *lister-list*. Conociendo esa relación, sabemos que la información de cada película irá contenida dentro del primer elemento mencionado, concretando un poco más, dentro de un elemento hijo que contiene una clase *lister-item-content*, por lo que podríamos iterar sobre cada uno de estos elementos para recopilar la información de cada película:

```

1 def parse(self, response):
2     movies_el = response.css('div.lister-item > .lister-item-content')
3
4     for movie_el in movies_el:
5         title = movie_el.css('.lister-item-header > a::text').get()
6         ...

```

En la línea ② lo que estamos haciendo es construir un selector CSS que nos devuelva la lista de elementos que tienen una clase *lister-item-content* y que a su vez tienen un parente que es un elemento de tipo *div* que contiene una clase *lister-item*. De esta forma, en la variable *movies_el* tendremos un documento HTML que

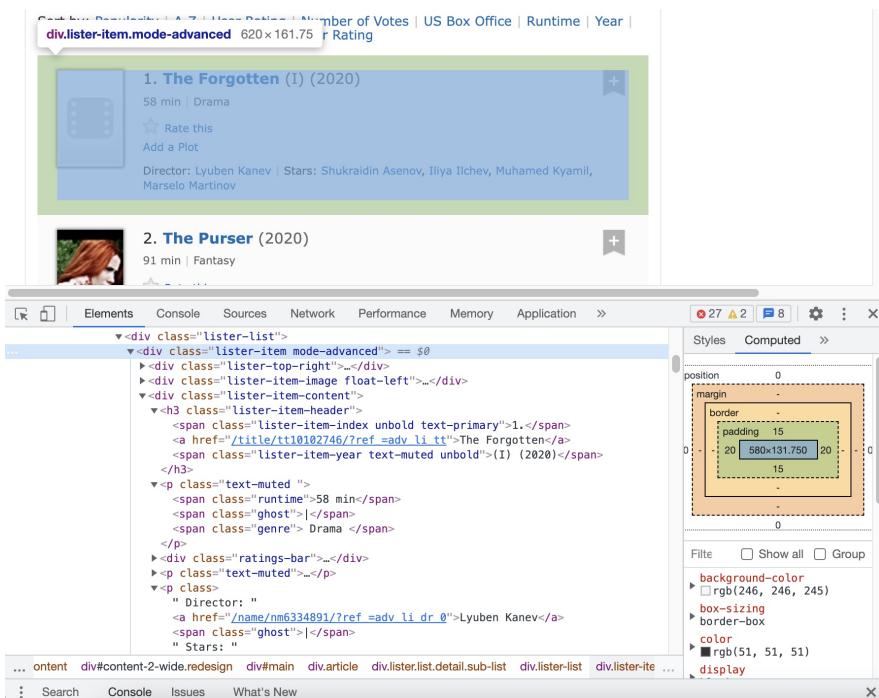


Figura 3.76: Inspeccionando el documento HTML para encontrar relaciones entre los elementos

contiene la lista de todas las películas de la página. Después, podemos iterar sobre esta lista (⊕4) para extraer cada uno de los campos que hemos identificado (⊕5). Utilizando el especificador `::text` podremos recuperar el texto contenido en un elemento HTML, y mediante el método `get()` devolverlo como una cadena de texto.

Para obtener los selectores concretos de cada campo que queramos extraer de las películas, tendremos que inspeccionar el árbol del documento HTML desde las herramientas de desarrollador e ir construyendo manualmente el selector. Si pulsamos la combinación de teclas `Ctrl+F` en el inspector de elementos de las herramientas de desarrollador del navegador, podremos escribir la ruta XPath o el selector CSS para comprobar visualmente si lo estamos haciendo correctamente (ver Figura 3.77).

En cualquier caso, una vez hayamos obtenido los campos que queremos, tendremos que indicarle a nuestra araña que los devuelva para que podamos procesarlos posteriormente. Además, el motor de búsqueda de IMDB únicamente nos ha proporcionado las 250 primeras películas, por lo que deberemos avanzar a la siguiente página una vez hayamos terminado de extraer la información de la página actual:

```
1 def parse(self, response):
2     movies_el = response.css('div.lister-item > .lister-item-content')
3
4     for movie_el in movies_el:
```

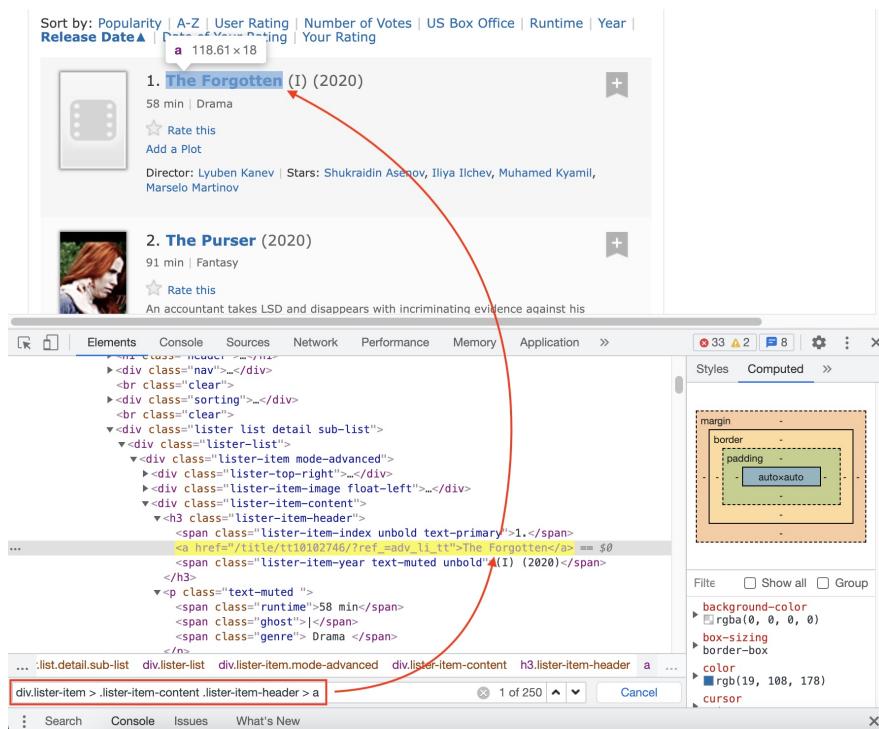


Figura 3.77: Comprobando un selector CSS desde las herramientas de desarrollador utilizando el propio navegador web

```

5      ...
6      yield {
7          'title': title,
8          'year': year,
9          'duration': duration,
10         'genres': genres,
11         'rating': rating,
12         'synopsis': synopsis,
13         'directors': directors,
14         'stars': stars
15     }
16
17     next_page = response.css('.next-page')
18     if next_page:
19         yield response.follow(next_page[0])

```

En las líneas ⑥-15 se construye un diccionario que será devuelto cada vez que se ejecute la araña, es decir, cada vez que se realice una iteración sobre el generador. Una vez el generador esté vacío y ya no queden más iteraciones por realizar, se avanzará de página (⑯-19), en caso de que haya alguna más. En caso contrario, la ejecución finalizará.

Una vez implementada nuestra araña, podremos ejecutarla mediante el comando:

```
$ poetry run scrapy crawl "movies-released-in-2020"
```

La ejecución tardará unos minutos, hasta que haya obtenido la información de las películas de todas las páginas de la búsqueda realizada. Durante el proceso, podremos ver un registro de los datos recuperados en la terminal, y finalmente unas estadísticas adicionales sobre la recopilación de datos.

Almacenamiento de los datos

Los datos que hemos recopilado han sido almacenados en memoria y después eliminados una vez ha finalizado la ejecución de la araña. Podemos indicarle un parámetro adicional al comando anterior para almacenar los datos en un archivo en formato JSON:

```
$ poetry run scrapy crawl "movies-released-in-2020" -o ./dataset.json
```

Este archivo JSON contendrá la información de todas las películas que haya recopilado la araña:

```
1 [  
2     {"title": "The Forgotten", "year": "2020", "duration": "58", "genres": "Drama", "  
3         "rating": null, "synopsis": "", "directors": "Lyuben Kanev", "stars": "  
4             Shukraidin Asenov,Iliya Ilchev,Muhamed Kyamil,Marselo Martinov"},  
5     {"title": "The Purser", "year": "2020", "duration": "91", "genres": "Fantasy", "  
6         "rating": null, "synopsis": "An accountant takes LSD and disappears with  
7             incriminating evidence against his mysterious employers. The Purser is tasked  
8                 with finding him and destroying the evidence.", "directors": "Jacques Aldridge"  
9                 , "stars": "Jacques Aldridge,Forest Baker,Amy Marie Bowersox,Dawne Ellison"},  
10    ...  
11 ]
```

Conversión de los datos

Una vez tengamos los datos almacenados, podremos cargarlos en nuestro cuaderno Jupyter y comenzar con el proceso de análisis exploratorio. Antes de eso, conviene convertir nuestro archivo de datos a un formato de *dataset* más estándar. Para ello, vamos a utilizar el módulo Pandas para cargar nuestro archivo JSON, comprobar que los datos son correctos y transformar nuestro *dataset* a formato CSV. En la Figura 3.78 puede verse este proceso realizado sobre el cuaderno Jupyter.



Es posible omitir este paso si generamos la salida de nuestra araña en formato CSV. Para ello, simplemente hay que cambiar la extensión del archivo de salida de `.json` a `.csv`.

```
[57] import pandas as pd
[57] ✓ 0.2s Python

df = pd.read_json('dataset.json')
print(len(df))
df.head()

[58] ✓ 0.4s Python
...
750

</>   title    year  duration   genres  rating   synopsis   directors   stars
0   The Forgotten  2020.0      58   Drama   NaN  An accountant takes LSD and disappears with in...  Lyuben Kanev  Shukraidin Asenov,Iliya Ilchev,Muhammed Kyamil,...  Jacques Aldridge,Forest Baker,Amy Marie Bowers...
1   The Purser  2020.0      91   Fantasy   NaN  The winning team of LUST STORIES unite to tell...  Jacques Aldridge  Janhvi Kapoor,Surekha Sikri,Amruta Subhash,Vij...
2   Ghost Stories  2020.0     144   Horror, Thriller  4.3  Pierfrancesco "Checco" Zalone, a failed entrep...  Zoya Akhtar,Dibakar Banerjee,Karan Johar,Anura...  Checco Zalone  Zalone,Souleymane Sylla,Manda Touré,Nas...
3   Tolo Tolo  2020.0      90   Comedy, Family  5.8  A group of siblings that share nothing but a f...
4   Baba Parasi  2020.0     116   Comedy  4.5  Selçuk Aydemir  Ahmet Kural,Murat Cemcir,Devrim Yakut,Rasim Öz...

df.to_csv('dataset.csv')
[59] ✓ 0.2s Python
```

Figura 3.78: Convirtiendo nuestro *dataset* a formato CSV

Inspección de los datos

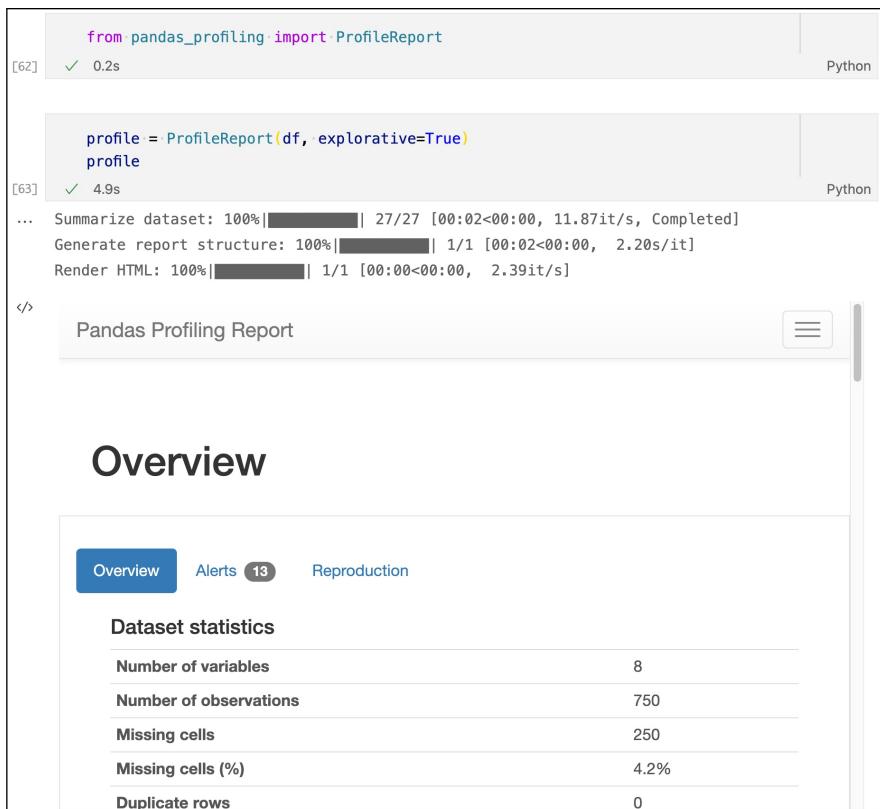
El último paso para dar por finalizado el proceso de recopilación de datos para construir nuestro *dataset* sería opcional, aunque conviene realizarlo para verificar que los datos que hemos obtenido son lo que esperamos.

Este proceso podemos realizarlo únicamente con Pandas, explorando los datos, sus tipos y algunos estadísticos de utilidad. Sin embargo, vamos a ver cómo hacerlo de manera sencilla utilizando para ello la biblioteca *pandas-profiling*⁵⁵. Esta biblioteca nos generará un informe en un documento HTML sobre nuestro *dataset* con información de utilidad.

Para utilizarlo, la instalaremos desde Poetry:

```
$ poetry add pandas-profiling[notebook]
```

⁵⁵ Enlace: <https://github.com/pandas-profiling/pandas-profiling>



```

from pandas_profiling import ProfileReport
[62]   ✓ 0.2s           Python

profile = ProfileReport(df, explorative=True)
profile
[63]   ✓ 4.9s           Python

... Summarize dataset: 100%|██████████| 27/27 [00:02<00:00, 11.87it/s, Completed]
Generate report structure: 100%|██████████| 1/1 [00:02<00:00,  2.20s/it]
Render HTML: 100%|██████████| 1/1 [00:00<00:00,  2.39it/s]

</> Pandas Profiling Report
  
```

The screenshot shows a Jupyter Notebook cell with two lines of Python code. The first line imports the ProfileReport class from pandas_profiling. The second line creates a ProfileReport object for a DataFrame named df, setting the explorative parameter to True. The cell is marked with a green checkmark and a duration of 0.2s. Below the cell, the output shows the progress of generating a report: summarizing the dataset (100% complete), generating the report structure (100% complete), and rendering the HTML (100% complete). The final output is a "Pandas Profiling Report" which is partially visible.

Overview

[Overview](#) [Alerts 13](#) [Reproduction](#)

Dataset statistics

Number of variables	8
Number of observations	750
Missing cells	250
Missing cells (%)	4.2%
Duplicate rows	0

Figura 3.79: Generación del informe del *dataset* desde el propio cuaderno Jupyter

Y la utilizaremos desde nuestro cuaderno Jupyter indicando el *dataframe* cargado mediante Pandas. Al ejecutarlo desde el cuaderno Jupyter, veremos como el informe será incrustado directamente después de la celda ejecutada (ver Figura 3.79).



Es posible exportar el informe a un archivo HTML utilizando el método `to_profile('ruta/al/archivo.html')`.

Desde este informe podremos analizar nuestro *dataset* para detectar valores que faltan, examinar las variables y las interacciones entre ellas, y detectar correlaciones, entre otras recomendaciones adicionales que nos proporciona el informe.

3.5. Manipulación de datos

Hasta ahora se han presentado recomendaciones y consideraciones a tener en cuenta a la hora de construir nuestro *dataset* para asegurar la calidad de los datos con los que trabajaremos para entrenar nuestros modelos de aprendizaje automático. Sin embargo, una vez hemos recopilado los datos con los que trabajaremos, necesitaremos transformarlos de algún modo para que puedan ser utilizados por los algoritmos de aprendizaje automático.

La **ingeniería de propiedades** (*feature engineering*) es el proceso de identificar qué propiedades pueden ser útiles para el entrenamiento de un modelo para después crearlas mediante la transformación de los datos en bruto recopilados previamente. En este tema se detalla este proceso, enfocándose en cuándo y cómo transformar los datos de tipo numéricos y categóricos, y en las ventajas y desventajas de los diferentes enfoques.

3.5.1. Visión general

Existen dos razones principales por las que realizamos la transformación de propiedades:

- **Transformaciones obligatorias** para hacer que los datos sean compatibles, por ejemplo, a la hora de convertir propiedades no numéricas en numéricas o para redimensionar las entradas de datos para que tengan un tamaño fijo. En el primer caso se hace, por ejemplo, porque no es posible multiplicar una matriz numérica por una cadena, por lo que habrá que convertir la cadena en algún tipo de representación numérica. En el segundo caso se hace, por ejemplo, cuando tenemos que suministrar los datos de entrada a una red neuronal que tenga un número fijo de nodos de entrada, como ocurre en el caso de los modelos de imagen, que necesitan redimensionar todas las imágenes al mismo tamaño.
- **Transformaciones opcionales (de calidad)** que puedan ayudar al modelo a que funcione mejor, por ejemplo, mediante la conversión de propiedades de texto a minúsculas, o la normalización de propiedades numéricas, entre otras.

Sobre los casos anteriores, las transformaciones de calidad no serían obligatorias, ya que nuestros modelos podrían seguir funcionando sin ellas, pero es posible que aplicarlas haga que nuestros modelos nos proporcionen mejores resultados.

Existen dos momentos principales en los que aplicar estas transformaciones: i) cuando se realiza la recopilación de los datos (antes del entrenamiento), o ii) cuando se ejecuta el modelo de aprendizaje automático. Ambas situaciones presentan ciertas ventajas y desventajas.

En el caso de la **transformación durante la recopilación de los datos** las transformaciones de los datos se realizan antes de proceder al entrenamiento del modelo; las transformaciones se realizan una única vez de manera *offline* y además es posible utilizar el *dataset* completo para determinar las transformaciones a aplicar. Sin embargo, las transformaciones que apliquemos en esta fase tendrán que poder ser reproducidas en el momento de utilizar el modelo. Además, cualquier cambio realizado sobre las transformaciones requerirá volver a entrenar el modelo, aumentando los tiempos de desarrollo.

En este caso, debemos tener cuidado con las desviaciones que se puedan producir en los datos tras aplicar las transformaciones. La aparición de desviaciones (o sesgo) en los datos suele ser más frecuentes en los casos donde el modelo se utiliza de forma *online*. Durante la utilización *offline*, se puede reutilizar el código que genera los datos de entrenamiento, pero durante la utilización *online* el código que crea el conjunto de datos y el código utilizado para manejar el tráfico en vivo son casi necesariamente diferentes, lo que facilita la introducción de desviaciones.

Por otra parte, en el caso de la **transformación durante la ejecución del modelo**, las transformaciones que se aplican a los datos forman parte del código del modelo. El modelo recoge los datos sin transformar como entrada y los transforma en el momento de realizar las predicciones. Esto facilita las futuras iteraciones, de tal forma que aunque se cambien las transformaciones que se aplican, todavía es posible utilizar los mismos datos de entrada. Además, de esta forma se garantiza que se apliquen las mismas transformaciones durante el entrenamiento y la realización de predicciones. Por otra parte, las transformaciones que sean más costosas de realizar podrían aumentar el tiempo de ejecución del modelo. Además, las transformaciones tendrían que aplicarse dentro del contexto de los datos de entrada en el momento de la predicción (también llamadas *transformaciones por lotes*).

Las transformaciones por lotes pueden tener cierto impacto en las predicciones. Supongamos que queremos normalizar una propiedad por su valor medio, es decir, queremos cambiar los valores de la propiedad para que tengan una media de 0 y una desviación estándar de 1. Al realizar la transformación durante la ejecución del modelo, esta normalización sólo tendrá acceso al conjunto de datos recibido en el momento de la predicción (lote), no al conjunto de datos completo. En este caso, podríamos normalizar por el valor medio del lote (poco recomendable si los lotes son muy variados), o precalcular la media y fijarla como una constante en el modelo.



Cuando hablamos de **normalizar**, nos referimos al proceso de convertir un rango real de valores en un rango estándar, normalmente de -1 a +1 o de 0 a 1. Por ejemplo, mediante la expresión $(x - \min(x))/(max(x) - \min(x))$ podríamos normalizar, por ejemplo, una propiedad que puede tomar valores entre 300 y 3000, para un valor de $x = 450$ a un rango de 0 a 1 (0,06).

Antes de empezar con el proceso de transformación de los datos, tendremos que explorar el *dataset* para analizarlo y limpiarlo aplicando alguna de las técnicas típicas para ello como, por ejemplo, revisar algunas filas del *dataset*, aplicar algunos estadísticos básicos, corregir valores numéricos que puedan faltar o eliminar filas a las que les falten valores categóricos relevantes, entre otros.

También resulta recomendable visualizar los datos frecuentemente. Los gráficos pueden ayudarnos a encontrar anomalías o patrones que no podamos identificar a partir de los resultados estadísticos numéricos. Por lo tanto, antes de adentrarnos en el análisis, deberíamos observar los datos gráficamente, ya sea mediante gráficos de dispersión o histogramas, y esto deberíamos hacerlo a lo largo del ciclo de vida del modelo. Las visualizaciones nos ayudarán a comprobar continuamente los supuestos y a ver los efectos de cualquier cambio importante.

Vamos a ver a continuación las posibles transformaciones que podemos realizar sobre los datos, dependiendo de si se tratan de datos numéricos o datos categóricos.

3.5.2. Transformación de datos numéricos

A la hora de tratar con datos numéricos es posible que tengamos que aplicar alguna de las siguientes transformaciones:

- **Normalización:** transformar datos numéricos a la misma escala de otros datos numéricos.
- **Clasificación:** transformar datos numéricos (normalmente variables continuas) en datos categóricos.

Normalización

El objetivo de la normalización es transformar las propiedades para que estén en una escala similar, lo cual mejora el rendimiento y la estabilidad de la fase de entrenamiento del modelo.

En este contexto, normalmente se suelen distinguir entre cuatro técnicas de normalizado que podemos aplicar: i) escalado a un rango, ii) recorte, iii) escalado logarítmico, y iv) escalado z-score. En la Figura 3.80 se muestra el aspecto que tendrían los datos tras aplicar cada una de las técnicas mencionadas a la distribución de la propiedad de más a la izquierda (precio). Los gráficos están basados en el *dataset* del anuario *Ward's Automotive* de 1985⁵⁶.

Normalizar mediante **escalado a un rango** significa convertir los valores de propiedades de su rango original (por ejemplo, 200 a 600) a un rango estándar (normalmente 0 a 1, o -1 a +1). Para ello, podemos utilizar la siguiente fórmula para escalar a un rango:

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)}$$

⁵⁶ Enlace: <https://archive.ics.uci.edu/ml/datasets/automobile>



Figura 3.80: Resumen de las técnicas de normalización

Esta técnica de escalado es una buena elección cuando se dan las siguientes condiciones:

1. Los límites inferiores y superiores de la propiedad a escalar son conocidos.
2. Existen pocos (o ningunos) valores atípicos.
3. Los datos se encuentran uniformemente distribuidos a lo largo de todo el rango (aproximadamente).

Por ejemplo, en el caso de la edad de las personas, la mayoría de edades estarían comprendidas entre 0 y 90 años, y cada subrango de este intervalo contiene una gran cantidad de individuos. Por otra parte, un caso donde no se debería escalar a un rango sería en el patrimonio neto de la población, ya que existen pocas personas que tengan un patrimonio alto; el límite superior de la escala lineal del patrimonio sería muy alto, mientras que la mayoría de la población se vería comprimida en una pequeña parte de la escala.

En el caso de la técnica del **recorte**, sería recomendable aplicarla si el *dataset* contiene valores atípicos extremos. La fórmula simplemente atiende a una expresión del tipo condicional: si $x > max$, entonces $x' = max$; si $x < min$, entonces $x' = min$. Aplicando esta técnica, limitaremos los valores por encima o por debajo de un determinado valor que fijemos. Por ejemplo, se podrían recortar todos los valores de temperatura superiores a 40 para que sean limitados a 40. Otra estrategia de recorte podría ser la de limitar en un número constante de veces la desviación típica (límite inferior negativo y superior positivo).

La técnica de **escalado logarítmico** nos permite calcular el logaritmo de los valores ($x' = \log(x)$) para comprimirlos de su rango original a un rango más reducido. Esta técnica resulta útil cuando existe una mayor concentración de valores de un tipo que de otro, lo que se conoce como una distribución de *ley potencial*. Por ejemplo, el caso del patrimonio neto mencionado anteriormente sí que sería un buen candidato para que apliquemos esta técnica de escalado. En la Figura 3.81 se puede ver cómo solamente dos individuos tienen un patrimonio neto excesivamente elevado, mientras que el resto tienen un patrimonio muy reducido. La escala logarítmica cambia la distribución, lo que ayuda a mejorar el rendimiento del modelo lineal.

Por último, la técnica del **escalado z-score** nos permite escalar con respecto del número de desviaciones típicas que se alejan de la media. Se calcula mediante la fórmula:



Figura 3.81: Comparación de una visualización en escala lineal y logarítmica

$$x' = \frac{x - \mu}{\sigma}$$

Siendo μ la media y σ la desviación típica. Se utiliza para garantizar que las distribuciones de las propiedades tengan una media de 0 y una desviación típica de 1. Es útil cuando hay algunos valores atípicos, pero no tan extremos como para aplicar la técnica del recorte.

Clasificación

La clasificación o *bucketing* es una técnica que se utiliza para convertir propiedades (normalmente que tienen valores continuos) en otras propiedades binarias denominadas categorías, normalmente basadas en un rango de valores.

Por ejemplo, en lugar de representar la temperatura como una propiedad de valores continuos, podríamos establecer un número de rangos discretos (categorías) para representarlas. Así, dada una serie de temperaturas con una precisión de una décima, podríamos definir tres categorías para representarlas: de 0,0 a 15,0 grados, de 15,1 a 30,0 grados, y de 30,1 a 45,0 grados.

Volvamos a revisar el ejemplo del *dataset* de los precios de los coches. Utilizando una categoría por propiedad, el modelo utilizará la misma capacidad para el rango de precio superior a los 40000 que para todos los ejemplos del rango 10000-20000. Obviamente, tras analizar el gráfico podemos ver cómo los datos se concentran aproximadamente en este último intervalo, por lo que utilizar una categoría del mismo tamaño para ambos rangos no sería recomendable, ya que en este caso no se capturaría la distribución correctamente (ver Figura 3.82, izquierda). La solución recae en crear categorías que contengan el mismo número (aproximadamente) de muestras, lo que se conoce como técnica de **clasificación en cuantiles**. Por ejemplo, en la Figura 3.82 (derecha) se muestra una clasificación del precio de los coches en cuantiles. Para obtener el mismo número de muestras en cada categoría, alguna de las categorías abarcan un intervalo de precios más reducido, mientras que otros abarcan un intervalo mayor.

Así, la clasificación en intervalos equivalentes es un método sencillo que funciona en la mayoría de distribuciones de datos. Para datos que presenten desviaciones, resulta más útil aplicar la clasificación en cuantiles.

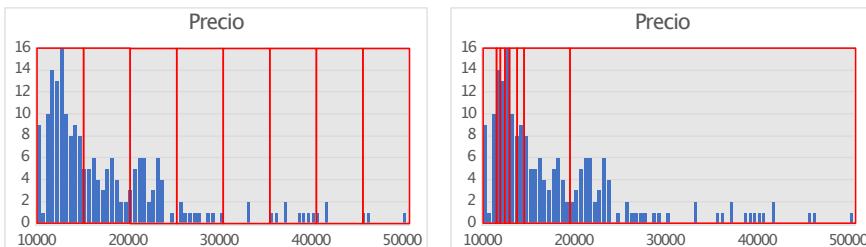


Figura 3.82: Clasificación de propiedades en intervalos equivalentes (izquierda) y en cuantiles (derecha)

3.5.3. Transformación de datos categóricos

Algunas de las propiedades pueden ser valores discretos que no forman parte de una secuencia de valores ordenada. Algunos ejemplos serían las razas de perros, las palabras o los códigos postales. Estas propiedades se conocen como categóricas y cada valor se denomina categoría. Estas categorías pueden representar valores categóricos como cadenas de texto o incluso números, sin que se puedan realizar operaciones aritméticas sobre ellos. De hecho, esto último suele ocurrir a menudo. Por ejemplo, consideremos una propiedad que represente el código postal en la que los valores son números enteros. Si representáramos erróneamente esta propiedad de forma numérica, estaríamos indicando al modelo que encuentre una relación numérica entre los diferentes códigos postales (por ejemplo, que un código postal es el doble o la mitad que otro).

Si el número de categorías de una propiedad es pequeño, como el día de la semana o los meses del año, es posible definir una propiedad única para cada categoría. Así, un modelo sería entonces capaz de asignar distintos pesos para cada nueva propiedad. Por ejemplo, el modelo podría aprender que en los meses de junio hay mayor ocupación hotelera que en los meses de febrero.

Una vez se han identificado las propiedades únicas, estas son indexadas para poder referenciarlas posteriormente. En la Figura 3.83 se muestra el proceso de división de la propiedad que representa los meses del año en propiedades únicas que representan cada uno de los meses del año.

Este proceso de indexación donde cada valor representa una única propiedad se conoce como *vocabulario*. El modelo recuperará este índice a partir de su valor categórico y asignará un 1.0 a la posición correspondiente en el vector de propiedades y un 0.0 para el resto de posiciones. Por ejemplo, en la Figura 3.84 se muestra este proceso completo donde, a partir del vector de categorías recuperado del *dataset* se recupera el índice de cada categoría y aplicando la técnica de codificación *one-hot* se codifica un valor de 1.0 en aquellas posiciones del vector de propiedades donde se ubiquen los índices de las categorías.

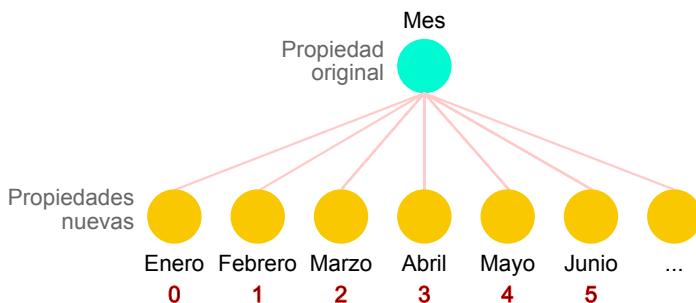


Figura 3.83: División e indexación de una propiedad categórica en otras nuevas propiedades

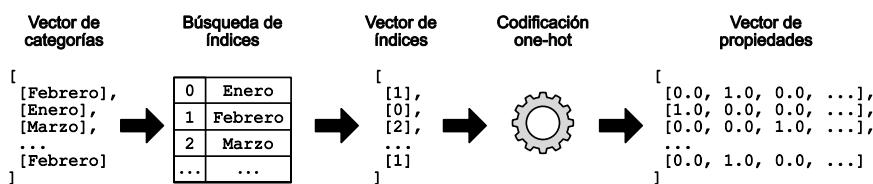


Figura 3.84: Proceso de creación del vector de propiedades a partir de propiedades categóricas aplicando la codificación one-hot

Valores atípicos

Al igual que las propiedades numéricas podían tener valores atípicos, las propiedades categóricas también los pueden presentar. Por ejemplo, consideremos un *dataset* que contenga descripciones de coches. Una de las propiedades de este conjunto de datos podría ser el color del coche. Supongamos que los colores comunes de los coches (negro, blanco, gris, etc.) están bien representados en este conjunto de datos y que hemos convertido cada uno de ellos en una categoría para que el modelo aprenda cómo afectan estos colores a la variable objetivo. Sin embargo, supongamos que este conjunto de datos contiene un pequeño número de coches con colores poco comunes (malva, púrpura o aguacate). En lugar de asignar cada uno de estos colores a una categoría propia, lo que haremos será asignarlos a una nueva categoría que se encargará de agrupar todas estas categorías *atípicas* llamada **palabras fuera de vocabulario** (OOV, por sus siglas en inglés). Al utilizar esta nueva categoría, el sistema no malgastará tiempo considerando cada una de estas categorías durante el entrenamiento.

3.5.4. Caso práctico: entrenamiento de modelo de regresión lineal

En este caso práctico se plantea un ejemplo de entrenamiento de un **modelo de regresión lineal**. El problema que se plantea, aunque básico, permite aplicar algunas de las técnicas de transformación de datos vistas hasta ahora para mejorar el proceso de entrenamiento. El caso práctico partirá del *dataset* creado por Jeffrey C. Schlimmer y publicado en el repositorio de UCI⁵⁷ (*imports-85.data*).

El *dataset* contiene información sobre 205 automóviles y sus precios, aunque no incluye los nombres de las columnas; esta información se encuentra en un archivo adicional disponible en el sitio web (*imports-85.names*). Concretamente, el *dataset* contiene i) información de un automóvil en términos de diversas características, ii) su nivel de riesgo en relación con el seguro, y iii) el pago medio relativo de los siniestros ocurridos por año de seguro. Finalmente, cada fila del *dataset* contiene información sobre el precio por el que se vendió dicho vehículo. Esta será nuestra variable objetivo.

Inicialización del entorno

En primer lugar procederemos inicializando el entorno de desarrollo, utilizando *poetry* como se ha ido viendo hasta ahora:

```
$ poetry new caso-practico  
$ cd caso-practico  
$ poetry add pandas scikit-learn seaborn matplotlib ipykernel
```

En esta ocasión hemos añadido dos nuevas bibliotecas a nuestro proyecto que utilizaremos para crear visualizaciones de los datos (*seaborn* y *matplotlib*).

Una vez inicializado el entorno, crearemos un directorio *notebooks* y, dentro de este nuevo directorio, un nuevo cuaderno Jupyter donde realizaremos el desarrollo del caso práctico:

```
$ mkdir notebooks  
$ touch notebooks/CasoPractico.ipynb
```

Una vez creado nuestro cuaderno, lo abriremos con Visual Studio Code y configuraremos el entorno de ejecución para que utilice nuestro entorno virtual que acabamos de crear con Poetry.

⁵⁷ Enlace: <https://archive.ics.uci.edu/ml/datasets/automobile>

```

import pandas as pd
import numpy as np
[220]   ✓ 0.1s                                     Python

# Añadimos los nombres a las columnas manualmente, ya que el dataset no los incluye
feature_names = ['symboling', 'normalized-losses', 'make', 'fuel-type',
..... 'aspiration', 'num-doors', 'body-style', 'drive-wheels',
..... 'engine-location', 'wheel-base', 'length', 'width', 'height', 'weight',
..... 'engine-type', 'num-cylinders', 'engine-size', 'fuel-system', 'bore',
..... 'stroke', 'compression-ratio', 'horsepower', 'peak-rpm', 'city-mpg',
..... 'highway-mpg', 'price']

df = pd.read_csv('datasets/imports-85.data', names=feature_names, na_values='?')
print(len(df))
df.head(5)
[222]   ✓ 0.4s                                     Python
... 205

</>
  symboling normalized-losses make fuel-type aspiration num-doors body-style drive-wheels engine-location wheel-base ...
0         3           NaN  alfa-romero    gas        std      two convertible       rwd      front     88.6 ...
1         3           NaN  alfa-romero    gas        std      two convertible       rwd      front     88.6 ...
2         1           NaN  alfa-romero    gas        std      two hatchback        rwd      front     94.5 ...
3         2          164.0  audi       gas        std      four sedan            fwd      front     99.8 ...
4         2          164.0  audi       gas        std      four sedan           4wd      front     99.4 ...

5 rows × 26 columns

```

Figura 3.85: Importación del *dataset* utilizando Pandas

Importación de los datos

Una vez tengamos abierto nuestro cuaderno Jupyter, vamos a comenzar cargando el *dataset* que utilizaremos durante el caso práctico. Para ello, lo descargaremos desde Enlace: <https://archive.ics.uci.edu/ml/machine-learning-databases/autos/imports-85.data> y lo guardaremos en un directorio *datasets*/ de la raíz del proyecto.

Después de guardar el *dataset*, lo importaremos utilizando el método `read_csv()` de Pandas. En este caso, el *dataset* no contiene una fila que indique los nombres de las columnas, por lo que tendremos que indicárselos manualmente al realizar la importación del *dataset*, utilizando el parámetro `names`. Del mismo modo, utilizaremos también el parámetro `na_values` para indicar que los valores encontrados como «?» al procesar el *dataset* se sustituyan por valores numéricos «NaN». Esto nos evitirá tener que corregir los tipos de datos de las propiedades numéricas posteriormente. En la Figura 3.85 se muestra el proceso de importación completo.



El método `read_csv()` de Pandas es capaz de cargar un archivo de datos directamente desde una URL, por lo que realmente no es necesario descargar el *dataset* manualmente.

```

>>> def print_cols_with_missing_values(df):
...     cols_with_missing = df.isnull().sum()
...     print(cols_with_missing[cols_with_missing > 0])

print_cols_with_missing_values(df)
[164]:   ✓ 0.2s
...    normalized-losses      41
num-doors          2
bore              4
stroke             4
horsepower        2
peak-rpm           2
price              4
dtype: int64

df['normalized-losses'].replace(np.nan, df['normalized-losses'].mean(), inplace=True)
df['bore'].replace(np.nan, df['bore'].mean(), inplace=True)
df['stroke'].replace(np.nan, df['stroke'].mean(), inplace=True)
df['horsepower'].replace(np.nan, df['horsepower'].mean(), inplace=True)
df['peak-rpm'].replace(np.nan, df['peak-rpm'].mean(), inplace=True)
[165]:   ✓ 0.2s
[166]:   num_doors_mode = df['num-doors'].value_counts().idxmax()
        print(num_doors_mode)
        df['num-doors'].replace(np.nan, num_doors_mode, inplace=True)
[166]:   ✓ 0.1s
...    four

df.dropna(subset=['price'], inplace=True)
df.reset_index(drop=True, inplace=True)
[167]:   ✓ 0.1s
[168]:   print_cols_with_missing_values(df)
[168]:   ✓ 0.1s
...    Series([], dtype: int64)

```

Figura 3.86: Preparación inicial de los datos mediante el tratamiento de los valores nulos

Preparación de los datos

Una vez importado nuestro *dataset*, empezaremos identificando y corrigiendo aquellos valores que falten en las filas. Para ello, primero identificaremos qué propiedades son las que presentan dichos valores utilizando el método `isnull()` del *dataframe* para filtrar por aquellas columnas que tengan al menos una celda sin valor. Despues, tendremos que decidir qué hacer con los valores de las columnas identificadas. En nuestro caso, vamos a indicar que se sustituyan por un valor medio del resto de valores de la columna. Por otra parte, para el caso concreto de la propiedad «*num-doors*», lo que haremos será asignar el valor más frecuente (la moda). Por ultimo, la columna «*price*» tambien incluye algunos valores nulos. En este caso, y dado que se trata de la variable objetivo, no hay mucho que podamos hacer, por lo que simplemente eliminaremos aquellas filas afectadas (ver Figura 3.86).

```

1 numeric_feature_names = ['symboling', 'wheel-base', 'normalized-losses',
2     'length', 'width', 'height', 'weight', 'engine-size',
3     'bore', 'stroke', 'compression-ratio', 'horsepower',
4     'peak-rpm', 'city-mpg', 'highway-mpg']
5 print(f'Propiedades numéricas: {numeric_feature_names} ({len(numeric_feature_names)})')
6 categorical_feature_names = ['make', 'fuel-type', 'aspiration', 'num-doors',
7     'body-style', 'drive-wheels', 'engine-location',
8     'engine-type', 'num-cylinders', 'fuel-system']
9 print(f'Propiedades categóricas: {categorical_feature_names} ({len(categorical_feature_names)})')
10
11 all_feature_names = numeric_feature_names + categorical_feature_names
12 target = 'price'
13
14 [169] ✓ 0.1s
15 ... Propiedades numéricas: ['symboling', 'wheel-base', 'normalized-losses', 'length', 'width',
16     'height', 'weight', 'engine-size', 'bore', 'stroke', 'compression-ratio', 'horsepower', 'peak-
17     rpm', 'city-mpg', 'highway-mpg'] (15)
18 Propiedades categóricas: ['make', 'fuel-type', 'aspiration', 'num-doors', 'body-style', 'drive-
19     wheels', 'engine-location', 'engine-type', 'num-cylinders', 'fuel-system'] (10)

```

Figura 3.87: Agrupación de las propiedades categóricas y numéricas en distintas variables

Normalización de los datos

Llegados a este punto, vamos a proceder al proceso de análisis de los datos y, si es necesario, su normalización. Antes de nada, lo primero que haremos será agrupar las propiedades categóricas y numéricas en dos variables distintas que nos permitan trabajar de manera más cómoda con los datos (ver Figura 3.87).

Después, comenzaremos nuestro análisis examinando los histogramas de las distintas propiedades numéricas para tener una visión general de las distribuciones. Para ello, ejecutaremos la siguiente expresión en nuestro cuaderno:

```

1 import matplotlib.pyplot as plt
2
3 df[numeric_feature_names].hist(ax=plt.figure(figsize=(15,15)).gca())

```

Lo cual nos mostrará una cuadrícula con los histogramas de todas las propiedades numéricas (ver Figura 3.88).

De estos histogramas podemos identificar varias desviaciones positivas (p. ej., *normalized-losses* o *engine-size*) ocasionadas por algunos valores atípicos. Para mejorar las distribuciones, vamos a proceder a aplicar algunas transformaciones a aquellas propiedades que presentan valores atípicos extremos.

En primer lugar, tendremos qué identificar cuáles son esos valores atípicos extremos. Para ello, podemos utilizar los diagramas de cajas (*boxplots*) para visualizarlos y confirmar que existen. Para dibujarlos, podemos hacerlo fácilmente utilizando la biblioteca *seaborn* que añadimos al principio:

```

1 import seaborn as sns
2
3 def draw_hist_boxplot(df, feature_name):
4     sns.boxplot(x=feature_name, data=df)

```

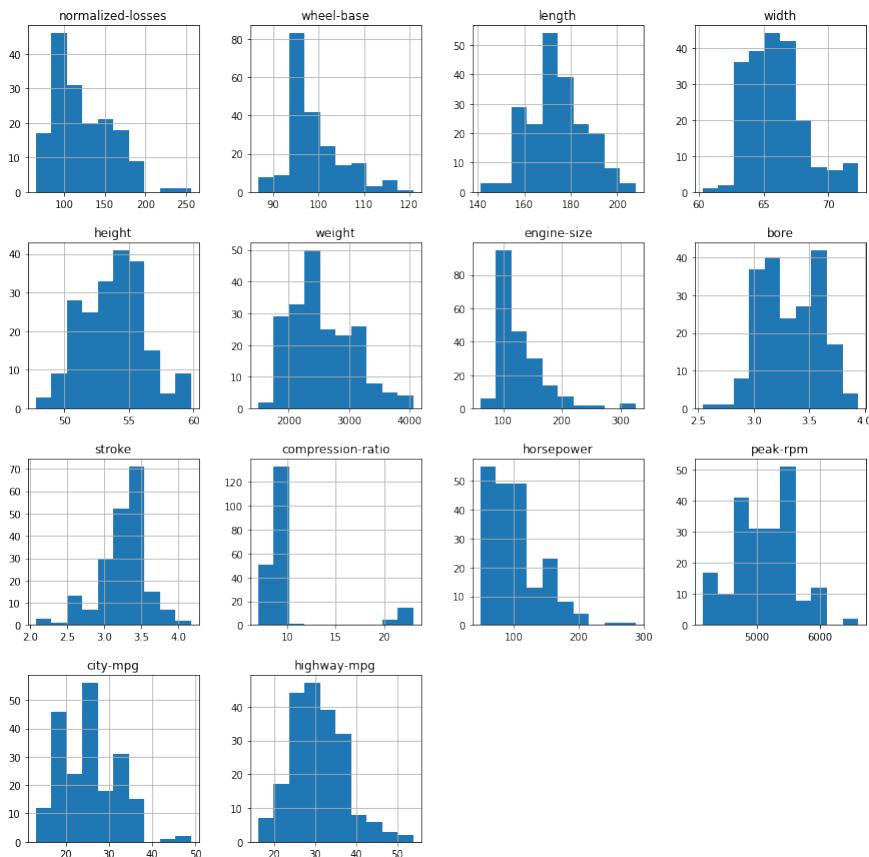


Figura 3.88: Histogramas de las propiedades numéricas

```
5     sns.stripplot(x=feature_name, data=df, color="#474646")
```

De esta forma, definimos una nueva función a la que le indicaremos el *dataframe* y el nombre de la propiedad que queremos dibujar. En la línea **④** dibujaremos el diagrama de cajas, mientras que en la línea **⑤** añadiremos sobre el diagrama todos los puntos de datos que tiene la propiedad. Por ejemplo, si la utilizamos sobre la propiedad *city-mpg* podremos ver cómo hay algunos (pocos) valores atípicos. Como son pocos, simplemente los eliminaremos del conjunto de datos y volveremos a visualizar el diagrama de cajas para comprobar que ya no están (ver Figura 3.89).

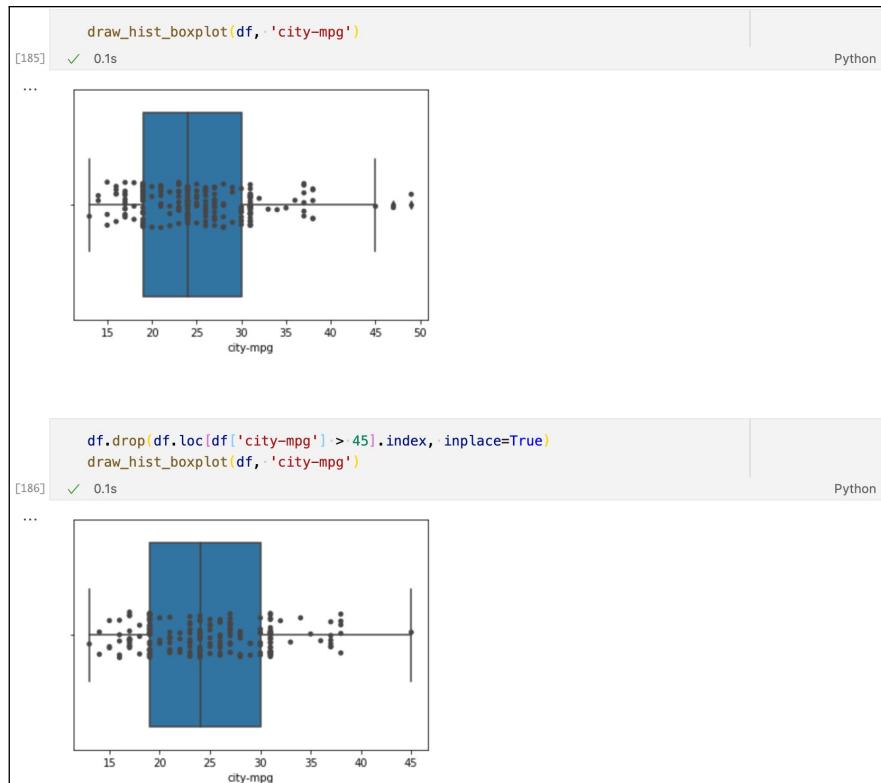


Figura 3.89: Diagrama de cajas de la propiedad *city-mpg* antes y después de eliminar los valores atípicos

En caso de que haya demasiados valores atípicos, puede ser contraproducente eliminarlos simplemente, sobre todo cuando no disponemos de demasiados datos. En este caso, lo que podemos hacer es reemplazar su valor por la *mediana*, de tal forma que esa propiedad no afecte al resto de valores pero manteniendo el resto de propiedades de la fila. Para ello, podemos definir una función que se encargue de realizar esta operación automáticamente en función de los cuartiles del diagrama de cajas:

```

1 def replace_outliers_with_median(df, feature_name):
2     Q1 = df[feature_name].quantile(0.25)
3     median = df[feature_name].quantile(0.5)
4     Q3 = df[feature_name].quantile(0.75)
5     IQR = Q3 - Q1
6     lower_whisker = Q1 - 1.5 * IQR
7     upper_whisker = Q3 + 1.5 * IQR
8     df[feature_name] = np.where((df[feature_name] < lower_whisker) |
9                                 (df[feature_name] > upper_whisker),
10                                median,
11                                df[feature_name])
  
```

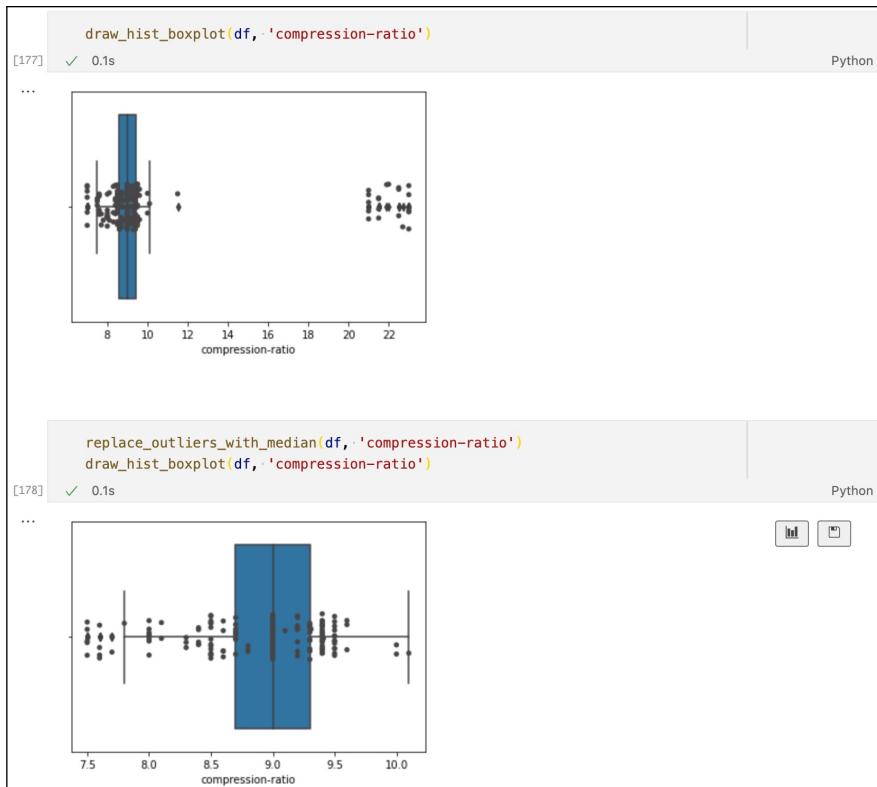


Figura 3.90: Diagrama de cajas de la propiedad *compression-ratio* antes y después de sustituir los valores atípicos por la mediana

Las líneas Θ 6,7 definen los extremos inferiores y superiores, respectivamente, del diagrama de cajas. Estos valores se utilizaran para identificar los valores atípicos automáticamente. De esta forma, cualquier valor que aparezca por debajo del límite inferior o por encima del límite superior, será reemplazado por la mediana.

Podemos aplicarlo, por ejemplo, para la propiedad *compression-ratio*, la cual presenta múltiples valores atípicos extremos. Tras ejecutar la función sobre ella, podremos ver como la distribución ha mejorado (ver Figura 3.90).



Podemos ejecutar el método `skew()` sobre la serie de un *dataframe* para cuantificar la desviación de la distribución y su signo. Si lo ejecutamos antes y después de corregir los valores atípicos, podremos ver cómo el valor de desviación se va reduciendo ligeramente.

Este proceso deberíamos aplicarlo a todas aquellas propiedades que presenten valores atípicos.

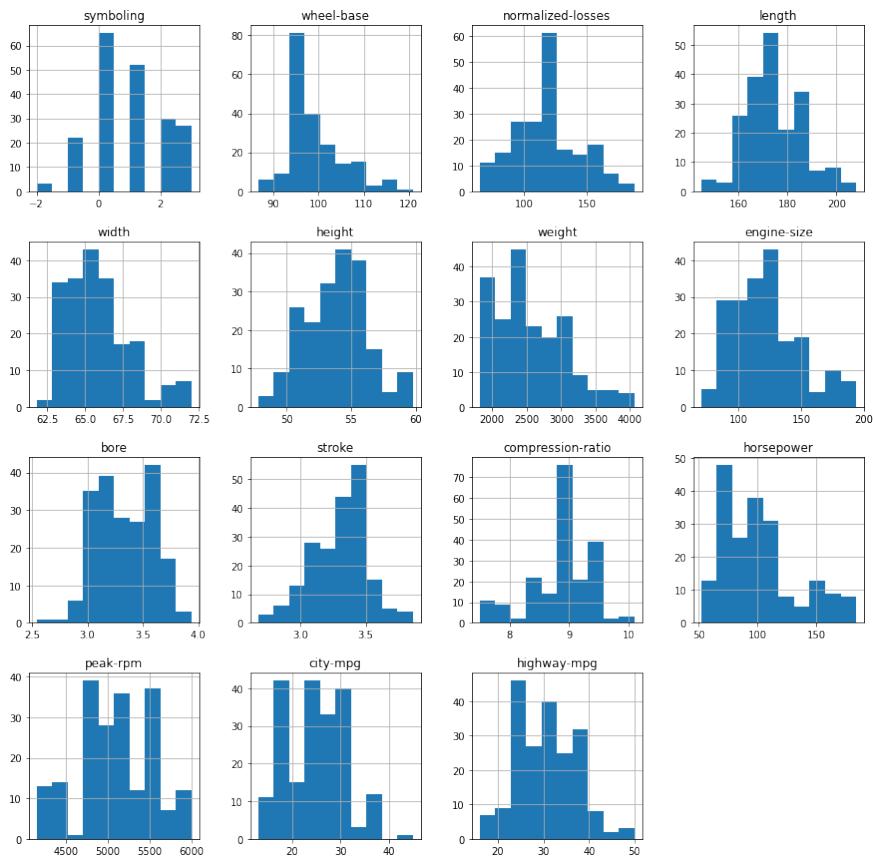


Figura 3.91: Histogramas de las propiedades numéricas tras corregir los valores atípicos

Una vez hayamos finalizado, podemos volver a visualizar los histogramas para ver que aspecto tienen las nuevas distribuciones (ver Figura 3.91).

Puede observarse como las desviaciones se han reducido ligeramente y ya no existen valores atípicos extremos. Para los valores atípicos que nos queden, podemos aplicar la técnica de normalizado mediante *escalado z-score*. Para ello, podemos utilizar la clase `StandardScaler` del módulo `sklearn.preprocessing` y visualizar de nuevo los histogramas para revisar las nuevas distribuciones (ver Figura 3.92):

```

1 from sklearn.preprocessing import StandardScaler
2
3 scaler = StandardScaler()
4 df[numerical_feature_names] = scaler.fit_transform(df[numerical_feature_names])
5 df[numerical_feature_names].hist(ax=plt.figure(figsize=(15,15)).gca())

```

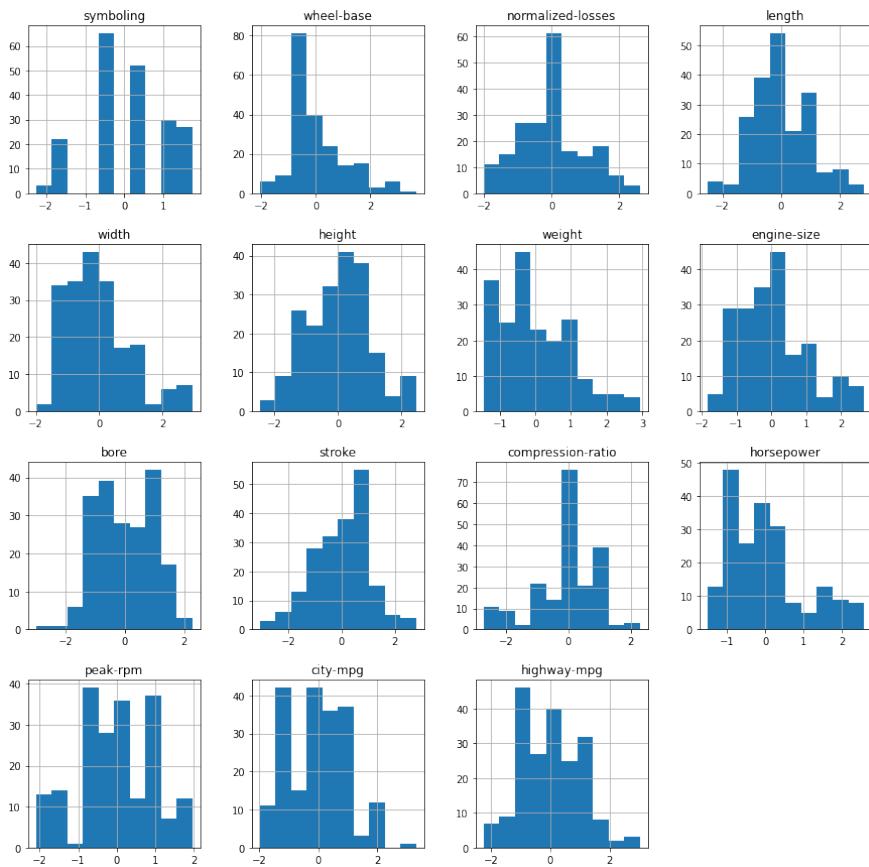


Figura 3.92: Histogramas de las propiedades numéricas tras aplicar la técnica de *escalado z-score*

En este caso podremos ver como la mayoría de distribuciones concentran la mayoría de sus valores en los intervalos [-2, 2], reduciendo así la cantidad de valores atípicos que teníamos.

Importancia de propiedades y división del conjunto de datos

Vamos a analizar ahora el impacto de nuestras propiedades sobre la variable objetivo. Para ello, vamos a empezar por las propiedades numéricas. Podemos realizar este análisis inicial mediante un mapa de calor que muestre las correlaciones que existen entre todas las variables, utilizando el método `heatmap()` del módulo `seaborn` e indicando que utilice el estadístico de correlación de Pearson para ello:

```

1 import seaborn as sns
2
3 plt.figure(figsize=(15,10))
4 sns.heatmap(df.corr(), cmap='BrBG', annot=True)

```

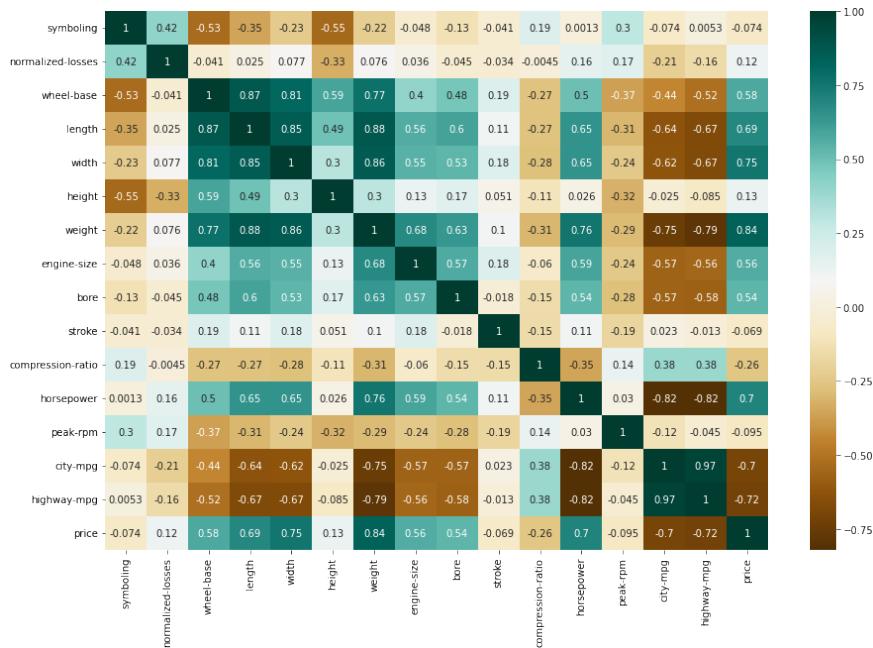


Figura 3.93: Mapa de calor de correlaciones entre las distintas propiedades numéricas

En la línea **④** se indica la técnica de correlación a aplicar, el esquema de colores para el mapa de calor (`cmap='BrBG'`) y que incluya los valores numéricos sobre el mapa (`annot=True`). En la Figura 3.93 se muestra el resultado, donde podemos ver cómo la variable objetivo presenta una fuerte correlación positiva con las propiedades *weight*, *width* y *horsepower*, entre otras.

Podríamos utilizar este análisis para eliminar algunas propiedades que no tengan ninguna correlación con la variable objetivo. En nuestro caso, eliminaremos la propiedad *symboling*, ya que además se trata realmente de una variable categórica que se está utilizando como una variable numérica:

```
1 df.drop(['symboling'], axis=1, inplace=True)
```

Sobre las variables categóricas, podríamos visualizar sus diagramas de cajas para analizar sus distribuciones. Por ejemplo, si visualizamos la propiedad *drive-wheels*, veremos que los vehículos de tracción trasera suelen ser más caros que los que tienen otro tipo de tracción, por lo que podría ser interesante utilizar esta propiedad en el proceso de entrenamiento (ver Figura 3.94).

En cualquier, utilizaremos todas las propiedades categóricas en primer lugar, y si la precisión del modelo no es la adecuada, probaremos a entrenar el modelo con otros conjuntos de propiedades para ver su rendimiento.

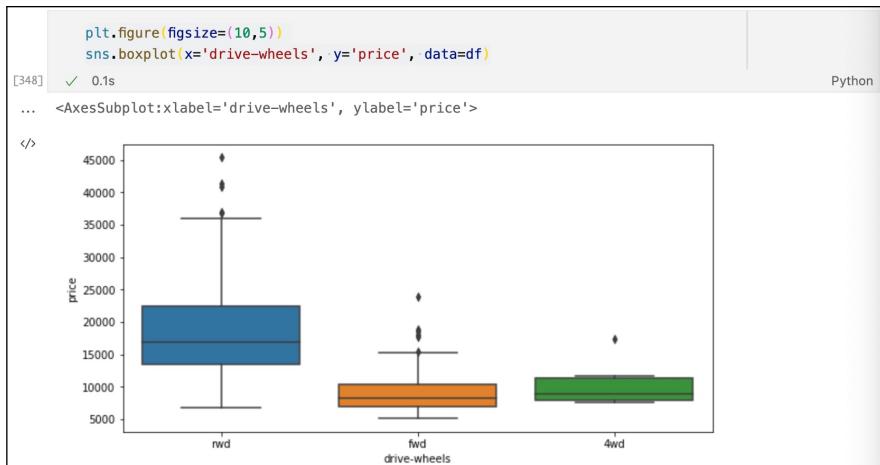


Figura 3.94: Diagrama de cajas de la propiedad *drive-wheels* en relación con el precio del vehículo

Una vez se hayan analizado las distintas propiedades y se hayan tomado las decisiones convenientes, fijaremos las propiedades con las que entrenaremos el modelo y procederemos a la división del conjunto de datos en los conjuntos de entrenamiento, validación y pruebas:

```

1 from sklearn.model_selection import train_test_split
2
3 numeric_feature_names = ['normalized-losses', 'wheel-base',
4     'length', 'width', 'height', 'weight', 'engine-size',
5     'bore', 'stroke', 'compression-ratio', 'horsepower',
6     'peak-rpm', 'city-mpg', 'highway-mpg']
7 categorical_feature_names = ['make', 'fuel-type', 'aspiration', 'num-doors',
8     'body-style', 'drive-wheels', 'engine-location',
9     'engine-type', 'num-cylinders', 'fuel-system']
10 all_feature_names = numeric_feature_names + categorical_feature_names
11
12 X_train_full, X_test = train_test_split(df[all_feature_names + [target]], test_size=0.2,
13                                         random_state=1)
14 X_train, X_val = train_test_split(X_train_full, test_size=0.33, random_state=1)
15 y_train = X_train[target].values
16 y_val = X_val[target].values
17 y_test = X_test[target].values
18
19 X_train.drop([target], axis=1, inplace=True)
20 X_val.drop([target], axis=1, inplace=True)
21 X_test.drop([target], axis=1, inplace=True)

```

```
from sklearn.linear_model import LinearRegression
model = LinearRegression()
model.fit(X_train, y_train)
[437] ✓ 0.2s
...
LinearRegression()

model.score(X_train, y_train)
[438] ✓ 0.2s
...
0.9837154885225918

X_val = dv.transform(X_val[all_feature_names].to_dict(orient='records'))
model.score(X_val, y_val)
[439] ✓ 0.2s
...
0.7462595560881797
```

Figura 3.95: Entrenamiento del modelo y coeficientes de determinación R^2

Ingeniería de propiedades

Una vez realizada la división del conjunto de datos, procederemos a codificar el conjunto de propiedades en un vector de propiedades. Utilizaremos la clase `DictVectorizer` para realizar la codificación *one-hot* de las variables categóricas:

```
1 from sklearn.feature_extraction import DictVectorizer
2
3 train_dict = X_train[all_feature_names].to_dict(orient='records')
4 dict(sorted(train_dict[0].items()))
5
6 dv = DictVectorizer(sparse=False)
7 dv.fit(train_dict)
8
9 X_train = dv.transform(train_dict)
```

Entrenamiento del modelo

Finalmente, podremos proceder a la fase de entrenamiento del modelo. En esta ocasión, utilizaremos la clase `LinearRegression` del módulo `sklearn.linear_model`, la cual nos permitirá entrenar un modelo de regresión lineal utilizando las propiedades de entrenamiento para predecir el valor de la variable objetivo (ver Figura 3.95).

En nuestro caso, hemos obtenido una precisión (utilizando el estadístico R^2) inferior para los datos de validación que para los datos de entrenamiento. Sin embargo, para el conjunto de datos de prueba, la precisión es superior al del conjunto de datos de validación (ver Figura 3.96). Esto puede ser debido a la división realizada del conjunto de datos, principalmente al reducido tamaño del *dataset*.

```

X_test = dv.transform(X_test[all_feature_names].to_dict(orient='records'))
y_pred = model.predict(X_test)
model.score(X_test, y_test)

[493] ✓ 0.2s Python
... 0.9177613476451484

import matplotlib.pyplot as plt
import numpy as np

ind = np.arange(15)
width = 0.35

plt.figure(figsize=(9,4))
plt.bar(ind, y_test[:15], width, label='Precio de venta real')
plt.bar(ind+width, y_pred[:15], width, label='Precio de venta predicho')

plt.xticks(ind+width/2, np.arange(1, 16))
plt.legend(loc='best')

plt.xlabel('Coches')
plt.ylabel('Precio de venta')
plt.title('Precio de venta - Real vs. Predicho')

plt.show()

[495] ✓ 0.2s Python
...

```

Figura 3.96: Predicciones del modelo realizadas para las 15 primeras muestras de los datos de prueba

Coches	Precio de venta real	Precio de venta predicho
1	14000	17000
2	6000	8000
3	8000	10000
4	8000	9000
5	6000	7000
6	10000	8000
7	9000	10000
8	17000	19000
9	5000	8000
10	9000	6000
11	16000	18000
12	6000	7000
13	8000	20000
14	10000	11000
15	7000	5000

Figura 3.96: Predicciones del modelo realizadas para las 15 primeras muestras de los datos de prueba

3.6. Evaluación de Resultados

3.6.1. Introducción

La evaluación de un modelo de aprendizaje en Inteligencia Artificial forma parte de su ciclo de vida. Gracias a la evaluación podemos cuantificar la calidad del modelo y su capacidad de predicción. En la mayoría de las ocasiones existe una realimentación entre la fase de evaluación y la de diseño, tal que una evaluación con resultados insatisfactorios da pie a modificaciones sobre el diseño para mejorar la calidad.

Los métodos de evaluación dependen del tipo de modelo; de esta forma, los modelos de aprendizaje automático como, por ejemplo, métodos de regresión lineal y métodos de *clustering* emplean unos métodos de evaluación distintos a los empleados en modelos de aprendizaje supervisado, principalmente métodos de clasificación a partir de un conjunto de ejemplos etiquetados previamente.

La biblioteca **Scikit-learn** es una biblioteca, libre, para el lenguaje de programación Python e incluye un amplio repertorio de herramientas con métodos de *Machine Learning* e Inteligencia Artificial. Dicha biblioteca facilita la evaluación de modelos en todas sus variantes y será la que empleemos en las próximas secciones para aportar un enfoque práctico.

En la documentación oficial de Scikit-learn y, en particular, en el apartado de modelos de evaluación⁵⁸ se ofrece una descripción detallada de las diferentes métricas soportadas para cuantificar la calidad de las predicciones de un modelo. Dichas métricas están especialmente divididas en tres categorías:

- Métricas de clasificación
- Métricas de *clustering*
- Métricas de regresión

En concreto las métricas soportadas por Scikit-learn en cada una de las categorías anteriores son las siguientes:

Clasificación

‘accuracy’	metrics.accuracy_score
‘balanced_accuracy’	metrics.balanced_accuracy_score
‘top_k_accuracy’	metrics.top_k_accuracy_score
‘average_precision’	metrics.average_precision_score
‘neg_brier_score’	metrics.brier_score_loss
‘f1’	metrics.f1_score
‘f1_micro’	metrics.f1_score
‘f1_macro’	metrics.f1_score
‘f1_weighted’	metrics.f1_score
‘f1_samples’	metrics.f1_score
‘neg_log_loss’	metrics.log_loss
‘precision’ etc.	metrics.precision_score
‘recall’ etc.	metrics.recall_score
‘jaccard’ etc.	metrics.jaccard_score
‘roc_auc’	metrics.roc_auc_score
‘roc_auc_ovr’	metrics.roc_auc_score
‘roc_auc_ovo’	metrics.roc_auc_score
‘roc_auc_ovr_weighted’	metrics.roc_auc_score
‘roc_auc_ovo_weighted’	metrics.roc_auc_score

⁵⁸ Enlace: https://scikit-learn.org/stable/modules/model_evaluation.html

Con el listado anterior, el lector puede hacerse una idea general de un conjunto representativo de métricas para la evaluación de modelos según su tipo. En las próximas secciones se profundizará en las métricas pertenecientes a la primera categoría: métricas para la evaluación de modelos de clasificación en aprendizaje supervisado.

Estos modelos parten de un conjunto de muestras (Dataset), siendo cada muestra una tupla formada por un conjunto de valores correspondientes a las variables de entrada, y una salida o clase a la que pertenece la muestra. Es decir, el modelo aprende a partir de un conjunto de ejemplos catalogados correctamente a priori. Por ejemplo, un Dataset muy popular con que el que los diseñadores prueban sus modelos es Iris, disponible en la mayor parte de plataformas de IA (Scikit learn, no es una excepción) y contiene 50 muestras de cada una de tres especies (150 ejemplos en total) de la flor Iris (iris setosa, iris virginica, iris versicolor). En este conjunto de datos las variables de entrada son: longitud del sépalo, anchura del sépalo, longitud del pétalo y anchura del pétalo. Por tanto, cada tupla está formado por cuatro valores de entrada, uno para cada variable de las mencionadas anteriormente, y la clase a la que pertenece (setosa, virginica, versicolor). A partir de los 150 ejemplos del Dataset, el modelo aplicado debe ser capaz de clasificar una nueva flor a partir de los cuatro valores de entrada. Precisamente, los métodos de evaluación nos ayudarán a medir la calidad de esa predicción.

3.6.2. Validación cruzada

A la hora de validar un modelo es muy habitual emplear una validación cruzada y, sobre ésta, aplicar las diferentes métricas que veremos en las próximas secciones.

En primer lugar, será necesario dividir en conjunto de datos inicial o Dataset en dos subconjuntos: conjunto de datos de entrenamiento (aprendizaje) y conjunto de datos para las pruebas (clasificación). En cuanto al porcentaje de ejemplos que corresponde a cada uno del conjunto inicial hay diversidad de opiniones, pero en la mayoría de los casos, los autores optan por un 70 % de ejemplos para el entrenamiento y un 30 % para las pruebas, o 80 %-20 %.

Los algoritmos de aprendizaje aplicados sobre el conjunto de entrenamiento generan como salida un modelo (1^a fase) con capacidad para clasificar los ejemplos del conjunto de pruebas (2^a fase). La evaluación del modelo se lleva a cabo en esta segunda fase, con la aplicación de métricas que determinan la exactitud, precisión o sensibilidad entre otros factores.

La proporción de ejemplos para el entrenamiento y las pruebas es importante, pero también lo es cuáles de ellos son seleccionados. Para evaluar correctamente un modelo es necesario un banco de pruebas con varias iteraciones. En cada una de esas iteraciones se hace una selección diferente de ejemplos para los conjuntos de entrenamiento y pruebas, y se aplican las métricas para obtener los marcadores. Los marcadores finales pueden obtenerse mediante la media aritmética de los obtenidos en cada una de las iteraciones.

La validación cruzada se preocupa por la selección de ejemplos en cada una de estas iteraciones, y hay diversas variantes. Una de ellas, de las más sencillas, es elegir al azar en cada iteración los ejemplos que forman parte de los conjuntos de entrenamiento y prueba. Sin embargo, en otras ocasiones se opta por la no repetición entre iteraciones. Para ello, se trocea el conjunto de datos en tantas partes como iteraciones o pruebas se pretendan realizar y, en cada iteración, se elige un fragmento diferente para el conjunto de pruebas. La Fig.3.97 muestra gráficamente un ejemplo de validación cruzada, en donde el conjunto de ejemplos empleado para la clasificación varía en cada iteración y sin repetición.

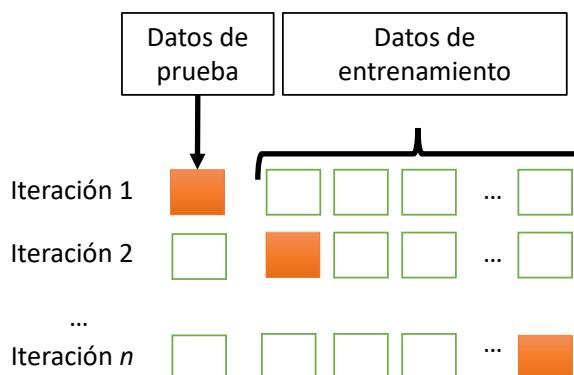


Figura 3.97: Validación cruzada

Scikit-learn también incluye herramientas para dividir el conjunto de datos mediante la técnica de validación cruzada. Veamos un ejemplo haciendo uso del conjunto de datos *Iris*.

Carga del conjunto de datos:

```
>>> import numpy as np
>>> from sklearn.model_selection import train_test_split
>>> from sklearn import datasets
>>> from sklearn import svm

>>> X, y = datasets.load_iris(return_X_y=True)
>>> X.shape, y.shape
((150, 4), (150,))
```

Para realizar una separación puntual del conjunto de datos en los conjuntos de entrenamiento (70 %) y pruebas (30 %), es tan sencillo como emplear la función *train test split*:

```
>>> X_train, X_test, y_train, y_test = train_test_split(  
... X, y, test_size=0.3, random_state=0)
```

El lector podría emplear una estructura iterativa (bucle) para repetir el proceso anterior un número concreto de iteraciones. En tal caso, la elección de ejemplos en cada conjunto sería al azar con riesgo de no cubrir en las pruebas algunas zonas del conjunto original.

Para realizar una validación cruzada sin repetición, podemos elegir un método de aprendizaje (por ejemplo, SVC) y la función *cross_val_score* en la que podremos indicar como argumento el número de iteraciones:

```
>>> from sklearn.model_selection import cross_val_score
>>> clf = svm.SVC(kernel='linear', C=1, random_state=42)
>>> scores = cross_val_score(clf, X, y, cv=5)
>>> scores
array([0.96..., 1. , 0.96..., 0.96..., 1. ])
```

Tal como puede observarse, se obtiene como salida un *array* con el grado de exactitud del modelo en cada iteración. En particular, la validación anterior se ejecutó con cinco iteraciones obteniendo un 96 % en la primera, 100 % en la segunda, 96 % en la tercera, 96 % en la cuarta y, finalmente, un 100 % en la última, lo cual es un indicador de la buena calidad del modelo para ese conjunto de datos en concreto.

En las siguientes secciones veremos con detalle las métricas a emplear sobre los conjuntos de pruebas, una vez aprendido el modelo con capacidad para clasificar muestras de entrada. Recuerden que, para una evaluación adecuada, estas métricas deberán aplicarse en diversas iteraciones sobre el conjunto de datos original.

3.6.3. Evaluación de modelos de clasificación

Matriz de confusión

La matriz de confusión no puede ser considerada una métrica como tal, pero es una herramienta fundamental para evaluar y optimizar los modelos de clasificación, ya que ayuda a profundizar en el tipo de error que el modelo está cometiendo y a comprender otras métricas que se emplean.

Una matriz de confusión se representa mediante una tabla en la que las columnas corresponden a los valores reales y las filas a los valores predichos. En el caso particular de un modelo de clasificación binaria, la Tabla 3.1 se confeccionaría tal como se muestra a continuación:

		Valores Reales	
Valores Predichos	Positivo (A)	Positivo (A)	Negativo (B)
		TP	FP
	Positivo (B)	FN	TN

Tabla 3.1: Matriz de Confusión para modelos de clasificación binaria

Supongamos un modelo de clasificación binaria, es decir, aquel en el que tan solo se pueden asignar dos clases diferentes (A y B) para cada ejemplo de entrada. Supongamos también un modelo cuyo objetivo es la clasificación de un tipo de cáncer como maligno (A) o Benigno (B). En función del valor real y el valor predicho por el modelo, podemos encontrarnos cuatro situaciones posibles (ver Tabla 3.1):

- **Verdadero Positivo** (True Positive, TP): el cáncer realmente es maligno (clase A) y ha sido clasificado como tal.
- **Falso Positivo** (False Positive, FP): el cáncer realmente es benigno (B) y el modelo lo ha clasificado como maligno (A).
- **Falso Negativo** (False Negative, FN): el cáncer realmente es maligno (A) y el modelo lo clasifica como benigno (B).
- **Verdadero Negativo** (True Negative, TN): el cáncer es benigno (B) y el modelo lo clasifica como benigno (B).

Por tanto, los Verdaderos (positivos y negativos) representan casos que han sido correctamente clasificados, mientras que los Falsos (positivos y negativos) representan errores de clasificación. De esta forma, nos interesa que la cantidad de ejemplos contabilizados en la diagonal principal (marcada en color verde en la Tabla 3.2) sea mucho mayor a los contabilizados en diagonal secundaria (marcados en color rojo en la Tabla 3.2)), lo que querrá decir que nuestro modelo se aproxima a un diagnóstico fiable.

		Valores Reales	
		Positivo (A)	Negativo (B)
Valores Predichos	Positivo (A)	TP	FP
	Positivo (B)	FN	TN

Tabla 3.2: Matriz de Confusión aciertos y errores de clasificación

En función de los valores obtenidos, se pueden realizar modificaciones sobre el modelo con la intención de reducir el número de falsos positivos o falsos negativos (o incluso ambos). Dependiendo del contexto en el que esté enmarcado el modelo, será más crítico reducir uno u otro. Por ejemplo, en el contexto médico que planteábamos anteriormente, los casos especialmente críticos corresponderían con los Falsos Negativos (FN), es decir, casos de tumores malignos diagnosticados como benignos, lo cual repercutiría gravemente en la salud del paciente. Por otro lado, en un contexto jurídico en el que un sospechoso puede ser clasificado como culpable (A) o inocente (B), los casos más críticos los representan los Falsos Positivos (FP), es decir, una persona inocente que ha sido catalogada como culpable, con el riesgo de terminar en prisión de forma injusta.

El siguiente fragmento de código muestra un ejemplo escrito en Python y la librería *Scikit-learn* en donde se define un vector básico de clases con valores reales y otro con los valores predichos. Ambos vectores son empleados para calcular la matriz de confusión y dibujarla posteriormente.

Listado 3.1: Cálculo de la matriz de confusión y representación

```
1 from sklearn.metrics import confusion_matrix
2 import matplotlib.pyplot as plt
3 import seaborn as sns
4
5 y_real = [1, 1, 1, 1, 1, 0, 0, 0, 0, 0]
6 y_pred = [1, 1, 1, 1, 0, 1, 1, 0, 0, 0]
7
8 cm = confusion_matrix(y_real, y_pred)
9
10 print(cm)
11
12 #Creación de Figura
13
14 ax= plt.subplot()
15 sns.heatmap(cm, annot=True, fmt=g, ax=ax, cmap=Greens);
16 #annot=True to annotate cells, fmt=g to disable scientific notation
17
18 # labels, title and ticks
19 ax.set_xlabel(Valores Predichos);ax.set_ylabel(Valores Reales);
20 ax.set_title(Confusion Matrix);
21 ax.xaxis.set_ticklabels([1, 0]); ax.yaxis.set_ticklabels([1, 0]);
22 plt.show()
```

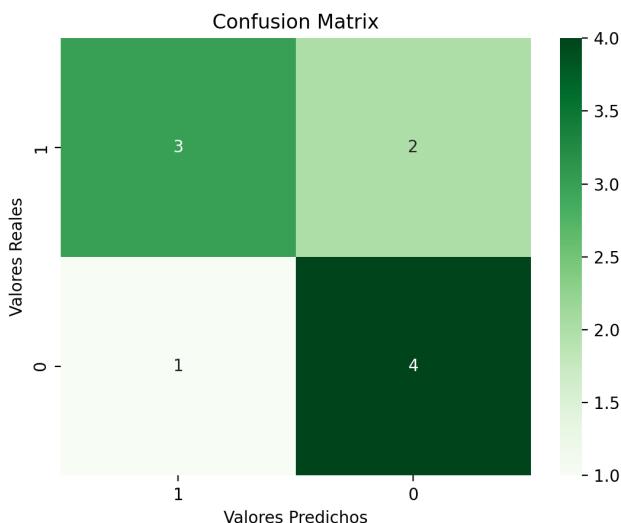


Figura 3.98: Imagen resultante que representa la matriz de confusión calculada en el fragmento de código 3.1

Por otro lado, para aquellos modelos de clasificación no binaria, en los que un ejemplo de entrada puede ser clasificado en n clases diferentes, también es posible la representación de un matriz de confusión. En este caso, las columnas representan el valor verdadero o la clase correcta a la que corresponde el ejemplo, y las filas las clases predichas. La Tabla 3.3 muestra una representación general de este tipo de matrices.

		Clases verdaderas			
		A	B	...	X
Clases Predichas	A				
	B				
	...				
	X				

Tabla 3.3: Matriz de Confusión para modelos multiclas

La diagonal principal representa los casos que han sido correctamente clasificados, es decir, la clase predicha corresponde con la clase real del ejemplo de entrada. El resto de las celdas representan casos de errores en la clasificación. Un fragmento de código en el que se lleva acabo una clasificación de estas características es el que se muestra a continuación:

Listado 3.2: Clasificación multiclas y cálculo de la matriz de confusión

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 from sklearn import svm, datasets
5 from sklearn.model_selection import train_test_split
6 from sklearn.metrics import ConfusionMatrixDisplay
7
8 #Importación de Iris Dataset
9 iris = datasets.load_iris()
10 #Conjunto de ejemplos
11 X = iris.data
12 #Clases
13 y = iris.target
14 #nombre de las clases
15 class_names = iris.target_names
16
17 # División del DataSet en dos conjuntos: entrenamiento y test
18 X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
19
20 # Usamos como clasificador Support Vector Classification
21 classifier = svm.SVC(kernel='linear', C=0.01).fit(X_train, y_train)
22 np.set_printoptions(precision=2)
23
24 # Dos gráficas, datos sin normalizar y normalizados.
25 titles_options = [
26     (Confusion matrix, without normalization, None),
27     (Normalized confusion matrix, True),
28 ]

```

```
29 for title, normalize in titles_options:
30     disp = ConfusionMatrixDisplay.from_estimator(
31         classifier,
32         X_test,
33         y_test,
34         display_labels=class_names,
35         cmap=plt.cm.Blues,
36         normalize=normalize,
37     )
38     disp.ax_.set_title(title)
39
40     print(title)
41     print(disp.confusion_matrix)
42
43 plt.show()
```

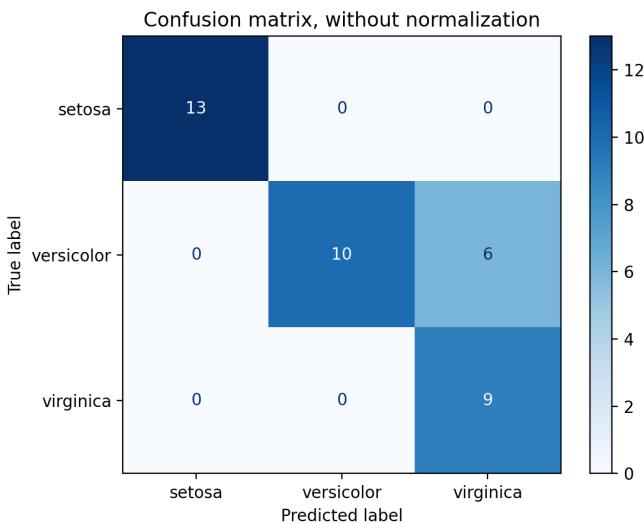


Figura 3.99: Matriz de Confusión calculada para Iris DataSet sin normalizar

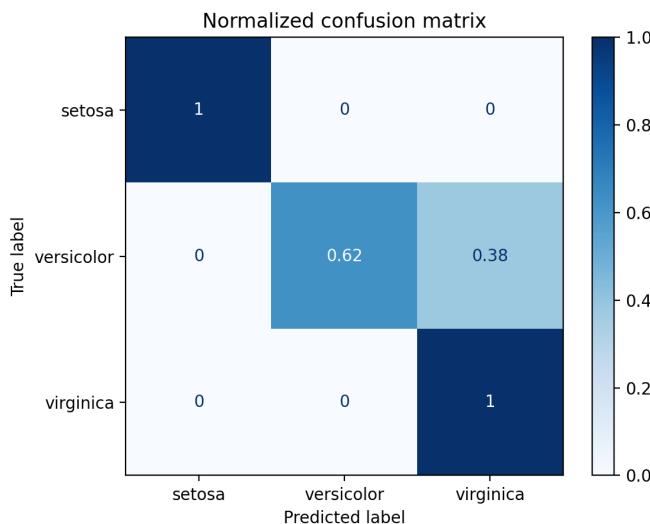


Figura 3.100: Matriz de Confusión calculada para Iris DataSet normalizada

Tras la ejecución del código anterior, se obtienen dos gráficas en las que se aprecia cómo se ha clasificado cada muestra:

Exactitud (accuracy)

La exactitud es el ratio entre las predicciones correctas: suma de verdaderos (positivos y negativos) y las predicciones totales. Dicho de otra forma, de todos los ejemplos ¿cuántos se predijeron correctamente? En el caso de una clasificación binomial el cálculo es tan sencillo como:

$$\frac{VP + VN}{TOTAL}$$

En el caso planteado en la Tabla 3.4 el cálculo de la exactitud del modelo, sería el siguiente:

		Valores Reales	
		Positivo (A)	Negativo (B)
Valores Predichos	Positivo (A)	8	2
	Positivo (B)	1	9

Tabla 3.4: Ejemplo de matriz de confusión para la aplicación de métricas

$$Exactitud = \frac{8 + 9}{8 + 9 + 2 + 1} = 0,85$$

De forma general, la exactitud de un modelo multiclasse podría representarse de la siguiente forma:

$$\text{accuracy_score}(y, \hat{y}) = \frac{1}{n} \sum_{i=0}^{n-1} (\hat{y} = \hat{y}_i)$$

Donde n representa el número de muestras (o ejemplos) totales, y la clase real es \hat{y} la clase predicha. Veamos cómo representar el ejemplo anterior en Python y realizar el cálculo de la exactitud en mediante *sklearn*:

Listado 3.3: Cálculo de la exactitud con *sklearn.metrics*

```
1 from sklearn.metrics import accuracy_score
2 y_pred =
3 ['A', 'A', 'A', 'A', 'A', 'A', 'B', 'A', 'A', 'B', 'B', 'B', 'B', 'B', 'B', 'B']
4 y_true =
5 ['A', 'A', 'A', 'A', 'A', 'A', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B']
6 print(accuracy_score(y_true, y_pred))
```

Precisión

La precisión determina el grado de acierto en la clase relevante. Suponiendo que la clase relevante sea A (tumor maligno en los ejemplos planteados anteriormente), la precisión sería la proporción entre los verdaderos positivos (TP) y el total de casos positivos (primera fila de la tabla). Con ello podemos saber cuántas de las muestras que fueron catalogadas como casos positivos eran realmente positivas:

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

El cálculo para el caso planteado en Tabla 3.4 sería el siguiente:

$$\text{accuracy} = \frac{8}{8 + 2} = 0,8$$

Listado 3.4: Cálculo de la precisión con *sklearn.metrics*

```
1 from sklearn.metrics import precision_score
2 y_pred =
3 ['A', 'A', 'A', 'A', 'A', 'A', 'B', 'A', 'A', 'B', 'B', 'B', 'B', 'B', 'B', 'B']
4 y_true =
5 ['A', 'A', 'A', 'A', 'A', 'A', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B']
6 print(precision_score(y_true, y_pred, labels=['A', 'B'],
7 pos_label='A', average="binary"))
```

La precisión también podría calcularse para la clase secundaria mediante la relación entre verdaderos negativos (VN) y la suma de VN + FN:

$$\frac{VN}{VN + FN}$$

La interpretación del dato en este caso es similar al anterior: cuántas de las muestras negativas reales han sido realmente detectadas. En el ejemplo de la clasificación de tumores, este valor serviría para conocer la proporción de tumores benignos detectados.

Exhaustividad (Recall)

Métrica también conocida como sensibilidad (sensitivity) o Ratio de Verdaderos Positivos (TPR, True Positive Rate), es un ratio entre los verdaderos positivos y la suma de los verdaderos positivos y los falsos negativos. El valor obtenido nos ofrece una idea clara sobre cuántos de los casos catalogados como positivos fueron predichos correctamente. De vuelta al ejemplo anterior, la exhaustividad determinaría cuántos de los tumores clasificados como malignos, son realmente malignos. ¿Cuál sería el número real de tumores malignos? La suma de TP (tumores malignos detectados correctamente) y los FN (tumores malignos no detectados).

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

El cálculo para el caso planteado en Tabla 3.4 sería para el siguiente:

$$Precision = \frac{8}{8+1} = 0,88$$

Listado 3.5: Cálculo de exhaustividad con sklearn metrics

```

1 from sklearn.metrics import recall_score
2 y_pred =
3 ['A','A','A','A','A','A','A','B','A','A','B','B','B','B','B','B']
4 y_true =
5 ['A','A','A','A','A','A','A','B','B','B','B','B','B','B','B','B']
6 print(recall_score(y_true, y_pred,labels=['A', 'B'], pos_label='A', average="binary"))

```

Pérdida Logarítmica (Log Loss)

La pérdida logarítmica es una puntuación única que representa la ventaja del clasificador sobre una predicción aleatoria. La pérdida logarítmica mide la incertidumbre del modelo comparando las probabilidades de sus resultados con los valores conocidos (verdad de base). El objetivo es minimizar la pérdida logarítmica para el modelo en su conjunto, cuanto menor sea el valor, menor será la incertidumbre y mejor será la capacidad del modelo para clasificar correctamente las muestras de entrada.

El cálculo de la pérdida logarítmica sería el siguiente:

$$\text{LogLoss} = -\frac{1}{n} \sum_{i=1}^n [y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)]$$

Donde n representa el número de muestras (o ejemplos) totales, y la clase real e \hat{y} la clase predicha. En Python y la librería sklearn el cálculo sería tan sencillo como se muestra a continuación:

Listado 3.6: Cálculo de pérdida logarítmica con sklearn metrics

```
1 from sklearn.metrics import log_loss
2 print(log_loss(y_true, y_pred))
```

Es importante recalcar que la función `log_loss` trabaja únicamente con clases representadas numéricamente. Por ello, en el caso del ejemplo anterior cuyas clases se representan mediante las letras A y B, sería necesario una conversión numérica previa.

Valor-F o Media-F (F1-Score)

El Valor-F es la media ponderada de la precisión y la exhaustividad (*recall*). Dicho valor varía entre 0 y 1, siendo 1 el valor ideal y 0 el caso opuesto. El Valor-F puede interpretarse como una media armónica de la precisión y la recuperación. La contribución relativa de la precisión y la recuperación al Valor-F son iguales.

$$F-score = \frac{2}{\frac{1}{\text{Precisión}} + \frac{1}{\text{Exhaustividad}}} = \frac{TP}{TP + \frac{FP+FN}{2}}$$

$$\text{F-score} = 2 * (\text{precision} * \text{recall}) / (\text{precision} + \text{recall})$$

El siguiente fragmento de código muestra como hacer uso de esta métrica en Python y sklearn.

Listado 3.7: Cálculo del Valor-F en sklearn metrics

```
1 from sklearn.metrics import f1_score
2 y_pred =
3 ['A','A','A','A','A','A','A','B','A','A','B','B','B','B','B','B','B','B','B']
```

```

4 y_true =
5 ['A', 'A', 'A', 'A', 'A', 'A', 'A', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B']
6 print(f1_score(y_true, y_pred, average='weighted'))

```

Entre los atributos de la función `f1_score` merece especial mención `average`, cuyo valor repercute directamente en el cálculo del Valor-F. Veamos las alternativas que ofrece dicho atributo:

- *None*: se devuelven las puntuaciones de cada clase. En caso contrario, el tipo de promedio viene determinado por alguna de las siguientes opciones:
 - *binary*: sólo informa de los resultados de la clase especificada por el atributo `pos_label`. Esto es aplicable sólo si los objetivos (`y_true,pred`) son binarios.
 - *micro*: calcula las métricas globalmente contando el total de verdaderos positivos, falsos negativos y falsos positivos.
 - *macro*: Calcula las métricas de cada etiqueta y encuentra su media no ponderada. Esto no tiene en cuenta el desequilibrio de las etiquetas.
 - *weighted*: calcula las métricas para cada etiqueta y encuentra su media ponderada por soporte (el número de instancias verdaderas para cada etiqueta). Esto altera la "macro" para tener en cuenta el desequilibrio de etiquetas; puede dar lugar a una puntuación F que no esté entre la precisión y la recuperación.
 - *samples*: calcula las métricas para cada instancia, y encuentra su promedio (sólo tiene sentido para la clasificación multietiqueta donde esto difiere de `accuracy_score`).

Curvas ROC

Una curva ROC (curva de característica operativa del receptor) es un gráfico que muestra el rendimiento de un modelo en base a la tasa de verdaderos positivos, y la tasa de falsos positivos. La forma de calcular dichas tasas es la siguiente:

- Tasa de verdaderos positivos (TPR) = $VP / VP + FN$ (misma formula de la Exhaustividad)
- Tasa de falsos positivos (FPR) = $FP / FP + VN$

No solo es un tipo de gráfico con el que se puede apreciar a simple vista el rendimiento de un modelo, sino también la comparación entre modelos. Por ejemplo, modelos de aprendizaje y clasificación diferentes aplicados sobre un mismo *Dataset*, como muestra la Figura 3.101

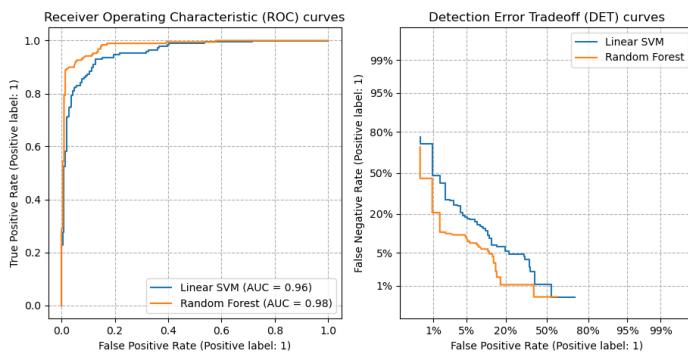


Figura 3.101: Ejemplo de curva Roc y comparación de clasificadores

Cuanto más cerca esté la curva al área de verdaderos positivos (cuanto más alta sea) mejor clasificará el modelo los datos de entrada.

AUC: área bajo la curva ROC

El área bajo la curva ROC toma un valor entre 0 y 1, en donde 1 representa una clasificación perfecta por parte del modelo. En tal caso, hay que prestar especial atención a que dicho resultado no sea consecuencia de un sobreajuste del modelo (*overfitting*). El sobreajuste de un modelo se produce cuando éste se ajusta casi a la perfección a los ejemplos del conjunto de entrenamiento (tiene capacidad para clasificarlos correctamente), pero no tiene capacidad para clasificar la mayoría de nuevos ejemplos (de los que no ha aprendido). En otras palabras, la capacidad de generalización del modelo es baja.

A medida que la curva ROC se acerque al área de verdaderos positivos, el área bajo la curva aumentará. El valor AUC de área representa la probabilidad de que el modelo clasifique correctamente un modelo de entrada. Por ejemplo, si AUC = 0.7, significa que el modelo ofrece un 70 % de probabilidad de clasificar correctamente una muestra de entrada.

Informe global con sklearn

Mediante la métrica *classification_report* de *sklearn* existe la posibilidad de elaborar un informe con información relevante, sin la necesidad de cálculo de métricas independientes. Con este informe podríamos obtener una primera idea general de la efectividad de nuestro modelo.

Listado 3.8: Elaboración de informe de clasificación con sklearn metrics

```
1 from sklearn.metrics import classification_report
2 y_pred =
```

```

3 ['A','A','A','A','A','A','B','A','A','B','B','B','B','B','B','B']
4 y_true =
5 ['A','A','A','A','A','A','B','B','B','B','B','B','B','B','B','B']
6
7 print(classification_report(y_true, y_pred, labels=['A', 'B']))

```

La salida en este caso sería la siguiente:

	precision	recall	f1-score	support
A	0.80	0.89	0.84	9
B	0.90	0.82	0.86	11
accuracy			0.85	20
macro avg	0.85	0.85	0.85	20
weighted avg	0.86	0.85	0.85	20

Los promedios indicados incluyen la macro media (que promedia la media no ponderada por etiqueta), la media ponderada (que promedia la media ponderada por soporte por etiqueta) y la media de la muestra (sólo para la clasificación multietiqueta). El micromedio (que promedia el total de verdaderos positivos, falsos negativos y falsos positivos) sólo se muestra para la clasificación multietiqueta o multiclase con un subconjunto de clases, porque de lo contrario corresponde a la exactitud y sería la misma para todas las métricas. Por último, *support* representa el número de ocurrencias de cada clase en *y_true*.

3.6.4. Ejemplo de análisis completo: clasificación de cáncer de mama como maligno o benigno

El *Dataset breast_cancer* es otro de los conjuntos de datos disponibles libremente y es de uso extendido en la comunidad científica. Se trata de un conjunto formado por 569 muestras, una dimensionalidad de 30 (variables de entrada) y dos clases como salida: maligno y benigno.

En primer lugar, cargaremos el conjunto de datos para poder trabajar con él: aprendizaje, clasificación y evaluación:

Listado 3.9: Carga del data set para diagnóstico del cáncer de mama

```

1 import numpy as np
2 import pandas as pd
3 from pandas import DataFrame
4 from sklearn.datasets import load_breast_cancer
5 from sklearn import svm
6 from sklearn.model_selection import train_test_split
7 from sklearn.metrics import confusion_matrix
8 import matplotlib.pyplot as plt

```

```

9 from sklearn.metrics import ConfusionMatrixDisplay, plot_roc_curve
10 from sklearn.metrics import accuracy_score, precision_score,
11 recall_score, f1_score, log_loss
12 import seaborn as sns
13
14 # 1. Cargamos los datos
15 data = load_breast_cancer()

```

Organizamos los datos en un DataFrame, y nos servirá para mostrarlos de una forma más interpretable por pantalla:

Listado 3.10: Organización de datos en un DataFrame.

```

1 # Propiedades o características de cada muestra estructuradas en columnas
2 df: DataFrame = pd.DataFrame(np.c_[data['data'], data['target']],
3                               columns=np.append(data['feature_names'], ['target']))
4
5 print(df.head())
6 print(df.describe())

```

Un ejemplo de salida de las líneas de código anteriores sería el siguiente:

	mean radius	mean texture	...	worst fractal dimension	target
0	17.99	10.38	...	0.11890	0.0
1	20.57	17.77	...	0.08902	0.0
2	19.69	21.25	...	0.08758	0.0
3	11.42	20.38	...	0.17300	0.0
4	20.29	14.34	...	0.07678	0.0

[5 rows x 31 columns]

	mean radius	mean texture	...	worst fractal dimension	target
count	569.000000	569.000000	...	569.000000	569.000000
mean	14.127292	19.289649	...	0.083946	0.627417
std	3.524049	4.301036	...	0.018061	0.483918
min	6.981000	9.710000	...	0.055040	0.000000
25%	11.700000	16.170000	...	0.071460	0.000000
50%	13.370000	18.840000	...	0.080040	1.000000
75%	15.780000	21.800000	...	0.092080	1.000000
max	28.110000	39.280000	...	0.207500	1.000000

[8 rows x 31 columns]

Figura 3.102: Datos organizados en Data Frame

Antes de la aplicación de algoritmos, puede resultar de gran utilidad, para la interpretación de datos y una evaluación posterior, representar gráficamente los ejemplos de acuerdo con el valor de diversas variables, así como la matriz de correlación entre variables:

Listado 3.11: Representación de muestras.

```

1 # Representación de las muestras por clase, de acuerdo a las variables
2 seleccionadas (las 4 primeras)
3 sns.pairplot(df, hue='target', vars=['mean radius', 'mean texture',
4 'mean perimeter', 'mean area'])
5 # mapa de correlación entre variables
6 plt.figure(figsize=(20, 12))
7 sns.heatmap(df.corr(), annot=True)

```

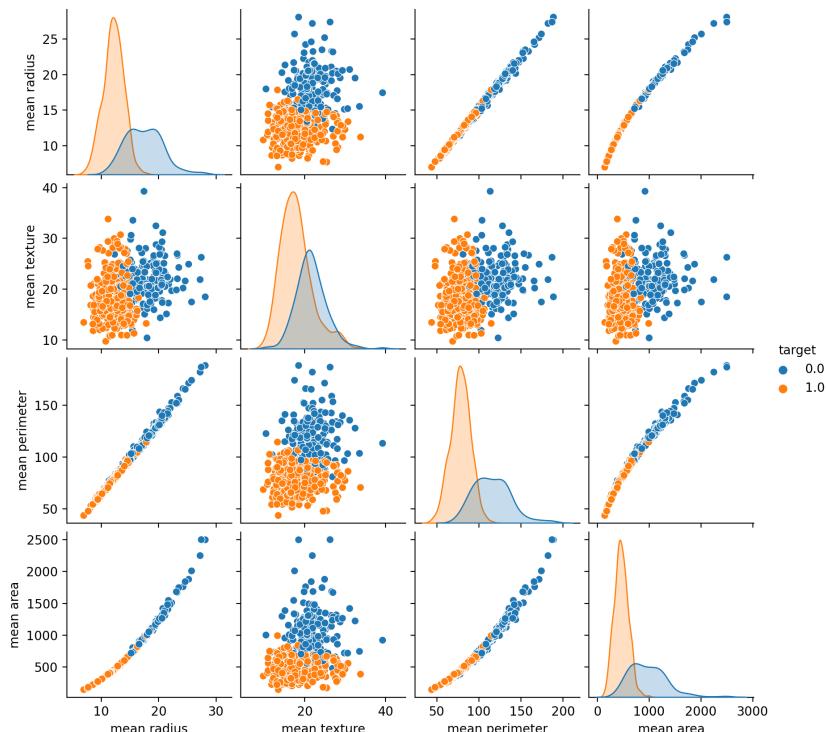


Figura 3.103: Distribución de ejemplos de acuerdo al valor de variables tomadas por pares

La Figura 3.103 corresponde a una gráfica bidimensional donde se aprecia la distribución de muestras (casos de cáncer) y su distribución de acuerdo con el valor de dos variables. En este caso particular se han tomado las cuatro primeras variables de las treinta que presenta el conjunto de datos. Fíjese también como las muestras pertenecientes a la clase “Maligno” (1.0) están coloreadas de naranja, mientras que las que pertenecen a la clase “Benigno” (0.0) lo están de color azul. En la representación, a simple vista, se aprecia que, en general, hay una diferencia clara entre las muestras de ambas clases en base al valor de las variables elegidas, a excepción de la frontera entre ambas donde puede haber un mayor número de casos conflictivos.

vos que den lugar a errores puntuales en el proceso de clasificación. En general, los métodos de aprendizaje y clasificación empleados para este Dataset deberían generar un modelo que arrojase resultados por encima del 90 % dada una separación tan clara.

Por otro lado, la Figura 3.104 muestra la matriz de correlación entre todas las variables. La correlación es un valor indicativo de la asociación entre dos variables. Dos variables están asociadas cuando el valor de una nos da información acerca de la otra. El valor de correlación o magnitud está comprendido entre -1 y 1; cuando el valor de correlación entre dos variables es positivo, se denomina correlación positiva y las dos variables se correlacionan en el mismo sentido o sentido directo: si el valor de una variable crece o decrece, la otra variable lo hace con el mismo comportamiento. En cambio, cuando el valor de correlación es negativo, hablamos de una correlación negativa en donde las variables se correlacionan en sentido inverso: cuando el valor de una variable crece el de la otra decrece, y viceversa. Si la magnitud de correlación es 1 hablamos de correlación positiva perfecta y supone una determinación absoluta entre las dos variables (en sentido directo). En el caso de que el valor sea -1 hablamos de correlación negativa perfecta (en sentido inverso). Finalmente, si el valor es 0, se dice que las variables están incorrelacionadas, es decir, no se puede establecer ningún sentido en la covariación.

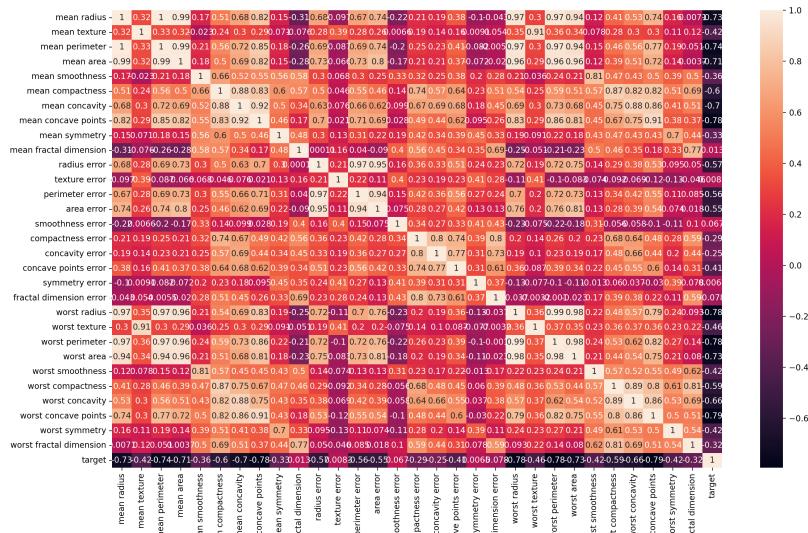


Figura 3.104: Matriz de correlación entre variables

En la Figura 3.105 se muestra el grado de dispersión de las muestras en base al valor de dos variables y la magnitud de su correlación. Cuanto mayor sea la magnitud, menor es la dispersión y más fuerte será la tendencia de las variables. Comprendido esto, vuelva a observar la Figura 3.103 y note la fuerte correlación entre alguna de las variables.

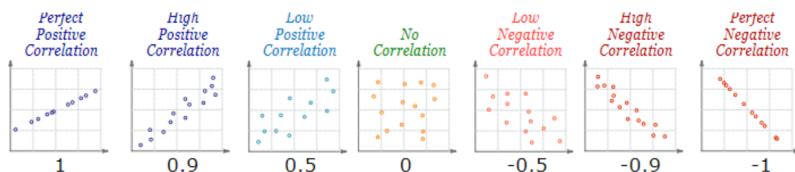


Figura 3.105: Diferentes valores del coeficiente de correlación y su correspondiente grado de dispersión

Listado 3.12: Representación gráfica de la cantidad de elementos por clase.

```

1 # Cantidad de elementos de cada Clase
2 print(df['target'].value_counts())
3 # Representación gráfica de la cantidad de elementos de cada clase
4 plt.figure(figsize=(8, 6))
5 sns.countplot(df['target'])
6 plt.xlabel("Diagnóstico")
7 plt.title('Cantidad de muestras por clase')

```

La representación gráfica del número de muestras por clase también nos ayuda a tener una mayor comprensión del conjunto de datos con el que estamos trabajando.

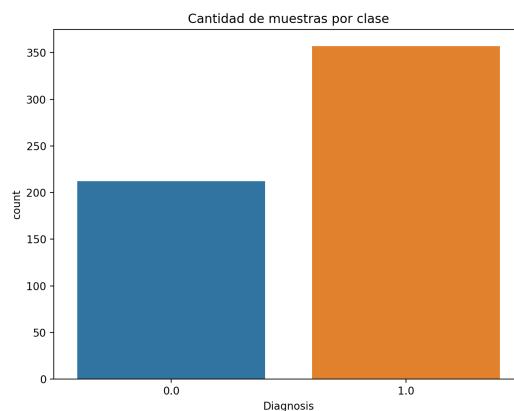


Figura 3.106: Cantidad de muestras por clase

Posterior a esta fase inicial, de preparación, cargamos el conjunto de datos en variables que posteriormente podamos manipular, y que son fundamentales para generar el modelo. Además, dividimos el conjunto inicial de entrada en dos subconjuntos: entrenamiento (70 % de muestras) y prueba (30 %).

Listado 3.13: Carga de ejemplos y división en subgrupos de entrenamiento y prueba.

```
1 # 2. Guardamos en X las variables de entrada (propiedades que definen
2 cada tipo de cáncer) y en y las clases
3 # Maligno y Benigno.
4 X, y = load_breast_cancer(return_X_y=True, as_frame=True)
5 print("CLASES: " + str(data.target_names))
6 # Creamos conjuntos de aprendizaje y de test, dejando un 30muestras para el aprendizaje
7 X_train, X_test, y_train, y_test = train_test_split(X,
8 y,
9 test_size=0.30,
10 random_state=1)
```

Elegimos uno de los clasificadores disponibles en Scikit-learn (en este caso, hemos optado por SVC (Support Vector Classification), generamos el modelo y hacemos la predicción con el conjunto de prueba (clasificación de muestras reservadas para las pruebas)

Listado 3.14: Aplicación de clasificador y generación del modelo.

```
1 # 3. Entrenamiento y clasificación
2 classifier = svm.SVC(kernel="linear", C=0.01).fit(X_train, y_train)
3 model = classifier.fit(X_train, y_train)
4 y_pred = model.predict(X_test)
```

Generamos la matriz de confusión a partir de las predicciones, la imprimimos por el terminal y generamos una gráfica que la represente.

Listado 3.15: Creación y representación de la matriz de confusión.

```
1 # 4 Matriz de Confusión
2 cm_data = confusion_matrix(y_test, y_pred, labels=np.unique(y_test))
3 print("Matriz de confusión")
4 print(cm_data)
5
6 # 5. Gráfica Matriz de Confusión
7 class_names = data.target_names
8 title = "Matriz de Confusión"
9 disp = ConfusionMatrixDisplay.from_estimator(
10     classifier,
11     X_test,
12     y_test,
13     display_labels=class_names,
14     cmap=plt.cm.Blues
15 )
16 disp.ax_.set_title(title)
17 plt.show()
```

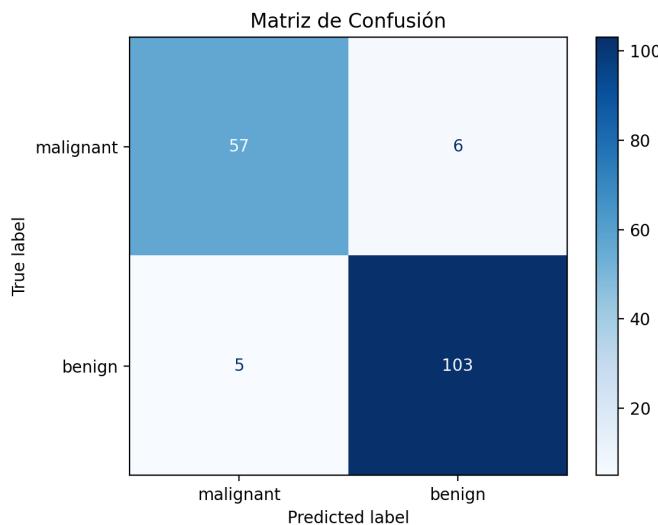


Figura 3.107: Matriz de confusión en base a los resultados obtenidos tras el proceso de clasificación

A continuación, se aplican las métricas para realizar la evaluación del modelo.

Listado 3.16: Métricas y Evaluación.

```
1 # 6 Cálculo de métricas
2 print("Exactitud: " + str(accuracy_score(y_test, y_pred)))
3 print("Precision: " + str(precision_score(y_test, y_pred, labels=[1,
4 0], pos_label=1, average="binary")))
5 print("Recall: " + str(recall_score(y_test, y_pred, labels=[1, 0],
6 pos_label=1, average="binary")))
7 print("Perdida Logarítmica: " + str(log_loss(y_test, y_pred)))
8 print("Valor-F: " + str(f1_score(y_test, y_pred, average='weighted')))
```

Obteniendo los siguientes resultados para una sola iteración:

- Exactitud: 0.935672514619883
- Precision: 0.944954128440367
- Recall: 0.9537037037037037
- Perdida Logarítmica: 2.221820689640929
- Valor-F: 0.9355634246907591

En los valores anteriores se puede apreciar como existe un equilibrio entre los valores de Exactitud y Precisión. Para cualquier modelo es fundamental siempre encontrar ese equilibrio. En el caso de que no exista, se pueden modificar los valores umbrales en el clasificador empleado y que determinan la frontera entre clases, provocando así la variación de VP, FP y FN y, en consecuencia, la Exactitud y Precisión. Finalmente, representamos la curva ROC y calculamos el área AUC para estudiar el rendimiento del modelo:

Listado 3.17: Cálculo y representación de curva ROC y área AUC.

```
1 #Curva ROC y área AUC
2 plot_roc_curve(model, X_test, y_test)
3 plt.show()
```

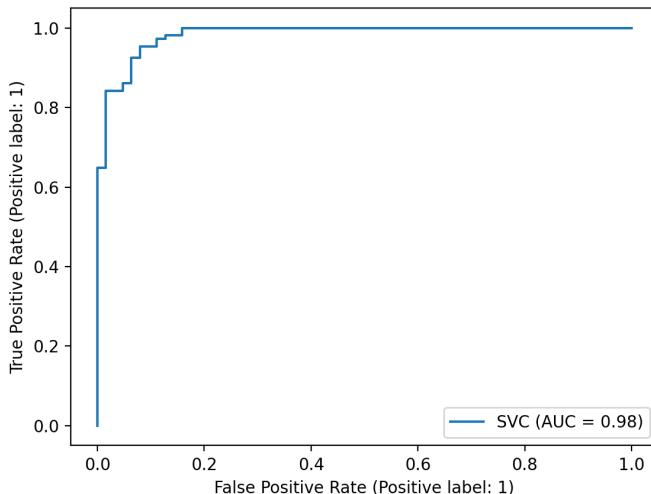


Figura 3.108: Curva ROC y cálculo de área AUC

En la gráfica 3.108 se puede apreciar como la curva se acerca rápidamente al área de verdaderos positivos y el área AUC = 0.98 (98 % de probabilidad de que un ejemplo sea correctamente clasificado), ambos indicativos del buen funcionamiento del clasificador elegido para el Dataset utilizado como ejemplo. Tal como se mencionó anteriormente, para una correcta evaluación del modelo se recomienda un mayor número de iteraciones cubriendo en la mayor medida de lo posible el espacio de muestras. Para ello se puede recurrir a una estructura iterativa (bucle) que permita la repetición de los últimos pasos indicados: división aleatoria del conjunto inicial de ejemplos y predicción, con posterior aplicación de métricas. O, también, emplear la validación cruzada que proporciona el paquete sklearn y que se describió en secciones anteriores.

3.7. Modelado de redes neuronales

El tema que se aborda en esta sección trata sobre las redes neuronales artificiales, un enfoque que recoge algunas ideas del sistema neuronal biológico para modelar programas con comportamiento inteligente. Con el objetivo de entender precisamente el funcionamiento de una red neuronal, se realiza primero un recorrido sobre las distintas unidades que las componen. Después, se discuten los detalles técnicos esenciales de su estructura y se presenta una visión intuitiva de su proceso de aprendizaje. Para poner en práctica lo estudiado en esta sección, se incluye al final un ejemplo relativamente simple para mostrar al lector la construcción de un clasificador mediante una red neuronal codificada desde cero.

3.7.1. Fundamentos básicos

Las redes neuronales, independientemente de que su origen sea biológico o artificial, se constituyen como un sistema compuesto de neuronas conectadas entre sí que reciben señales y las transmiten entre ellas. Las neuronas son la unidad básica de este esquema, constituidas por un cuerpo celular y ramificaciones. Solo como curiosidad, esta unidad simple de procesamiento se encuentra la mayor parte del tiempo a la espera de señales procedentes de las ramificaciones que conectan a una con otra neurona.

En esencia, la neurona está formada por un cuerpo o **soma** donde se encuentra el **núcleo**. Las ramificaciones que sirven de entrada de información a las neuronas se denominan **dendritas**. Por el contrario, la señal saliente de la red neuronal proviene del **axón**, una ramificación más alargada que la dendrita. Las conexiones entre neuronas es un mecanismo que se denomina **sinapsis** y se realiza uniendo el axón a una o varias dendritas.

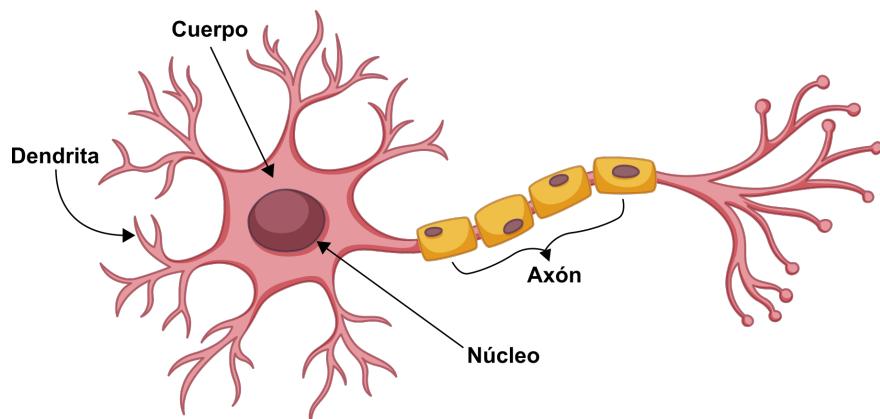


Figura 3.109: Partes de una neurona biológica. Imagen: Freepick.com.

Las neuronas emiten señales que se propagan de una a otra mediante una compleja reacción electroquímica, las cuales controlan la actividad del cerebro a corto plazo. Por el contrario, estas señales también habilitan la conectividad de las neuronas, a largo plazo. Estos mecanismos muy elaborados son la base del aprendizaje. La mayor parte del procesamiento de la información tiene lugar en la **corteza cerebral**, la capa externa del cerebro.

Una red neuronal artificial no busca imitar el comportamiento de su equivalente biológico. Al contrario, se ignoran la mayoría de mecanismos internos de las neuronas porque replicar su comportamiento no es en absoluto necesario para satisfacer el objetivo deseado. A modo de ejemplo, cualquier artefacto capaz de navegar, ya sea bajo o sobre el agua, no está construido siguiendo fielmente la estructura de una pez, es decir, aletas y branquias. Algo similar ocurre con las redes neuronales artificiales, dado que el objetivo no es simular el sistema biológico, sino imitar, de algún modo, el procesamiento del cerebro humano para añadir comportamiento inteligente a sistemas con cerebros artificiales.

Obviamente, los cerebros biológicos y los ordenadores presentan propiedades totalmente diferentes. Por un lado, una red neuronal, cuyo origen no sea artificial, es capaz de procesar, y simultáneamente, información en cada una de las unidades básicas de su estructura, es decir, las neuronas. Por el contrario, los computadores disponen de dos componentes para replicar este comportamiento, haciendo uso de un cerebro artificial, denominado CPU que procesa información, y de una memoria que guarda dicho resultado. Por lo tanto, la gran diferencia existente es que las neuronas biológicas son capaces de procesar información por su cuenta. En otras palabras, las neuronas tratan grandes cantidades de información paralelamente, mientras que una CPU realiza este cometido de uno en uno.

No obstante, inspirado en el **procesamiento paralelo**, las siguientes secciones presentan la arquitectura de una red neuronal artificial capaz de procesar simultáneamente numerosos datos, replicando a alto nivel el comportamiento de una red neuronal biológica.

3.7.2. Estructura de una red neuronal

Una red neuronal artificial, al igual que una red biológica real, está compuesta de nodos o unidades denominadas **neuronas artificiales**. Cada una está conectada por un **enlace directo** al cual se le asigna un **peso** numérico que determina la influencia de la conexión sináptica entre las neuronas. Estos pesos se utilizan como multiplicadores de las entradas de la red neuronal, resultando en una suma ponderada del producto de los pesos y entradas. Por ejemplo, si contamos con una red neuronal compuesta de 5 entradas x_1, x_2, x_3, x_4, x_5 , también disponemos de 5 pesos asociados a cada entrada w_1, w_2, w_3, w_4, w_5 . Además, se suele añadir un término extra denominado sesgo o «bias», una medida que indica lo fácil que es disparar la salida de la neurona. Como resultado, la salida en este paso consiste realizando la siguiente suma ponderada: $x_0 \times w_0 + x_1 \times w_1 + x_2 \times w_2 + \dots + x_5 \times w_5$. Una vez computado el resultado, la neurona efectúa una operación más, aplicando una **función de activación** cuyo término es prestado de la neurociencia, la cual

determine la salida de la neurona. En otras palabras, una predicción o decisión a partir de las entradas proporcionadas a la red. Para dotar de inteligencia a esta estructura, utilizando el símil de la red neuronal biológica, el aprendizaje resulta de la modificación de la influencia de las conexiones sinápticas entre neuronas. Esto se consigue visualizando la red neuronal artificial como un sistema dinámico que modifica los valores de sus pesos. De un modo simplificado, este es el funcionamiento de una red neuronal artificial. Veamos a continuación los detalles técnicos de su estructura.

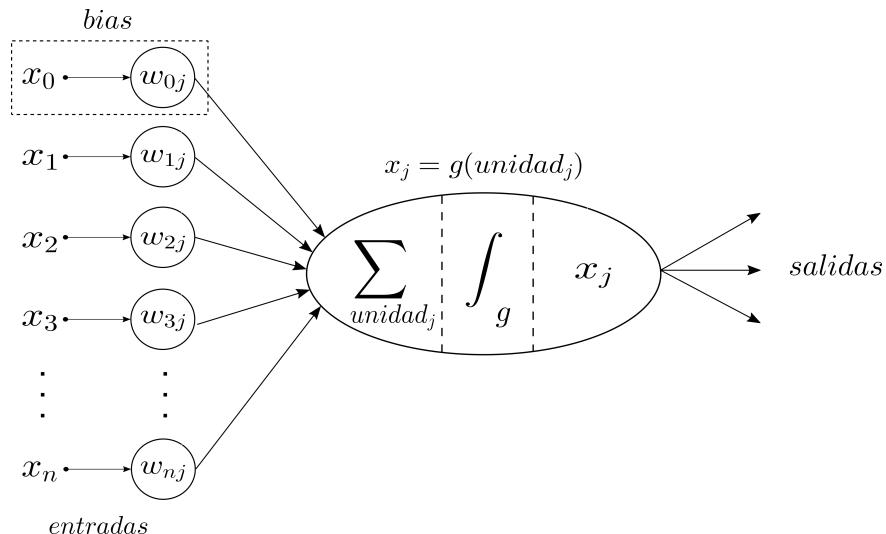


Figura 3.110: Unidad simple de procesamiento de una red neuronal.

La figura 3.110 muestra el esquema general de un elemento mínimo de procesamiento en una red neuronal. En esencia, una red cuenta con numerosas entradas (neuronas) y, en base a su simplicidad o complejidad, una o varias salidas. La salida x_j representa la activación de la entrada x_i y el peso w_{ij} en el enlace desde la unidad i hasta la unidad j . El valor de la red a la unidad j se puede definir como:

$$unidad_j = \sum_{i=0}^n x_i w_{ij} \quad (3.3)$$

Nótese que el número de interconexiones en una red determina indudablemente la velocidad en la cual se computa esta operación.



Perceptrón. Fue introducido por el psicólogo Frank Rosenblatt en 1957 y se considera «la madre de todas las redes neuronales artificiales». Este modelo fue inspirado por los trabajos de Warren McCulloch y Walter Pitts. El perceptrón toma varias entradas y produce una salida binaria.

Una vez resuelto el valor de la combinación entre los valores de la entrada y los pesos, se le aplica una función de activación g para resolver la salida, es decir, la predicción o decisión como consecuencia de las entradas introducidas en la red. Esta función se denota como:

$$x_j = g(\text{unidad}_j) = g\left(\sum_{i=0}^n x_i w_{ij}\right) \quad (3.4)$$

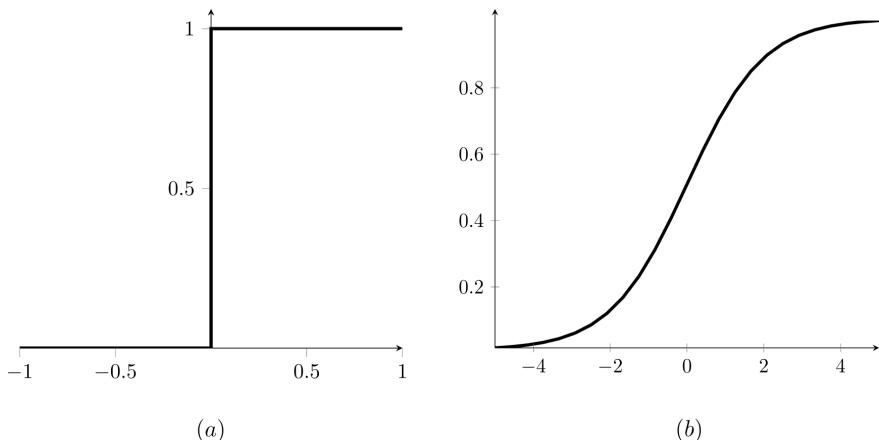


Figura 3.111: (a) Función escalón con salida 1/0. (b) Función sigmoide, cuya salida resulta en un valor real entre 0 y 1.

La función de activación g trata de imitar el mecanismo de los impulsos eléctricos en la comunicación entre dos o más neuronas. Este comportamiento se intenta replicar, de algún modo, con un umbral mediante la **función escalón** (véase figura 3.111 (a)) o con una versión suavizada usando la **función sigmoide** (véase figura 3.111 (b)). En ocasiones poco frecuentes se utiliza también la función identidad, cuyo objetivo es no hacer nada y producir la combinación lineal. Es por ello que rara vez se utiliza esta función en redes neuronales, dado que su comportamiento no conduce a nada nuevo.



Funciones de activación. Existe una variedad amplia de funciones que se utilizan en el contexto de las redes neuronales para activar sus unidades de procesamiento. Entre ellas destaca la función tangente hiperbólica, unidad lineal rectificada (ReLU, siglas por el nombre *rectified linear unit*, en inglés) o softmax, entre otras. Aunque las neuronas biológicas parecen implementar la función sigmoide, la función de activación ReLU resulta ser, en general, mejor en las redes neuronales artificiales. Debido a esto, se podría decir, por tanto, que la analogía biológica es errónea.

La función escalón (véase 3.5) produce una salida de 1 cuando el resultado de la combinación de entradas y pesos es mayor de 0. De lo contrario, su salida será 0. Como es de esperar, una salida con valor 1 significa que la unidad de salida se activa.

$$\theta(x) = \begin{cases} 1 & \text{si } x \geq 0 \\ 0 & \text{si } x < 0 \end{cases} \quad (3.5)$$

Por el contrario, la función sigmoide (véase 3.6) es una modificación suavizada de la función escalón que produce salidas entre 0 y 1. El resultado de las salidas se puede interpretar como probabilidades, lo cual es útil para predecir cómo de probable es que ocurra algo, en lugar de utilizar valores numéricos enteros. El uso de esta función no lineal se emplea para mejorar el aprendizaje de los pesos y el sesgo de la red. Al realizar pequeñas modificaciones en ambos elementos se produce un ligero cambio en la salida, resolviendo los valores adecuados para una tarea concreta, a diferencia de valores enteros que pueden cambiar el comportamiento del resto de la red.

$$\sigma(x) = \frac{1}{1 + e^{-x}} \quad (3.6)$$

No obstante, es imprescindible destacar que la convención a usar (salidas con números enteros o reales) viene determinada por el tipo de problema a resolver. En otras palabras, no existe una función que sea la «panacea» ante la resolución de un problema cualquiera.



Sesgos en una red neuronal. Como analogía, los sesgos o «bias» son como el desplazamiento en la regresión lineal que viene dada por $y = ax + b$, donde a es el coeficiente de la x (pendiente) y b es el término independiente, es decir, la ordenada en el origen. Básicamente, su uso sirve para desplazar la entrada procedente de una capa a otra en un valor constante, lo que puede ser fundamental para el éxito del aprendizaje. Estas característica no es en absoluto obligatoria en las redes neuronales, pero sí suele ser útil para el rendimiento del modelo.

Hasta ahora, una red neuronal artificial consta de entradas, pesos, funciones de activación y salidas, pero aún faltan por ver algunas partes todavía muy importantes en estos modelos computacionales complejos. Las entradas se organizan formando una columna distribuida en la parte más hacia la izquierda de la red neuronal; el nombre que se le da es el de **capa de entrada**. Por el contrario, la salida se encuentra en la parte más hacia la derecha de la red neuronal, atribuyéndole el nombre de **capa de salida**. Aunque aún no se ha discutido una red con más de dos capas, lo normal es diseñar redes con más de dos niveles, dado que las primeras, consideradas como perceptrones, solo resuelven problemas lineales. Los niveles intermedios de la red se conocen como **capas ocultas** que utilizan la salida de otras neuronas como entrada y cuya salida es utilizada como entrada para otras capas de neuronas.

Es importante destacar que las capas son independientes de una a otra, es decir, una capa específica puede tener un número arbitrario de nodos. Por ejemplo, en una tarea de reconocimiento de imágenes, las entradas contienen el valor de los píxeles de la imagen proporcionada. Esta información se envía a la siguiente capa, produciendo un resultado que es recogido por otra capa, y así sucesivamente. Finalmente, la capa de salida produce un resultado de toda la red.

3.7.3. Topologías de redes neuronales

La topología de una red neuronal especifica la disposición de sus nodos (neuronas) y líneas de conexión (sinapsis), formando capas o agrupaciones de neuronas. En esencia, este modelo computacional puede considerarse como un **grafo dirigido**, cuyos vértices representan los nodos o unidades de procesamiento y, a las aristas, se le asigna un número llamado peso o conexión sináptica. Fundamentalmente, las redes neuronales se clasifican en base a dos esquemas: **monocapa** y **multicapa**. La primera, como se ha discutido anteriormente, es la estructura más sencilla, ya que se tiene una capa de neuronas que proyectan las entradas a una de salida donde se realizan los diferentes cálculos. Por el contrario, una red multicapa, como su nombre bien indica, es una red compuesta de muchas capas que pueden estar parcial o completamente conectadas. Atendiendo al flujo de datos de una red neuronal, existen una multitud de tipos de estructuras que otorgan diferentes características de cómputo [RN20], aunque en esta sección se discuten solo dos por ser las más empleadas en la práctica.

El primer tipo se conoce como redes de **alimentación progresivas**. La principal característica de estas redes es que sus conexiones son unidireccionales y no forman ciclos, es decir, su topología es igual a la de un **grafo acíclico dirigido**. Por lo tanto, un nodo o neurona recibe como entrada un valor de la capa anterior y su resultado se envía a la capa siguiente; los cálculos se realizan de manera uniforme desde las unidades de entradas hasta las unidades de salida, puesto que no existen bucles.

Por otro lado, al segundo tipo se le atribuye el nombre de **redes recurrentes**, ya que toman información de entradas anteriores para influir su entrada y salida actual. Mientras las entradas y salidas de las redes neuronales progresivas son independientes unas de otras, la salida de las redes neuronales recurrentes dependen de los elementos anteriores dentro de la secuencia. Otra característica distintiva es que las redes recurrentes comparten parámetros en cada capa de la red. Es decir, comparten el mismo peso dentro de cada capa de la red, a diferencia de las redes de alimentación progresivas que mantienen pesos diferentes en cada nodo. Una red recurrente forma una **sistema dinámico**, puesto que algunas de las capas están conectadas hacia atrás con las capas anteriores, formando bucles de retroalimentación donde algunas neuronas procesan los datos varias veces antes de producir una salida. Se puede decir, por tanto, que una red neuronal recurrente posee **memoria a corto plazo**. Este hecho permite que esta arquitectura simule mejor el comportamiento del cerebro al mantener un estado interno. Sin embargo, una red neuronal recurrente, por lo general, es más difícil de entender e incluso de entrenar que las redes de alimentación progresivas.

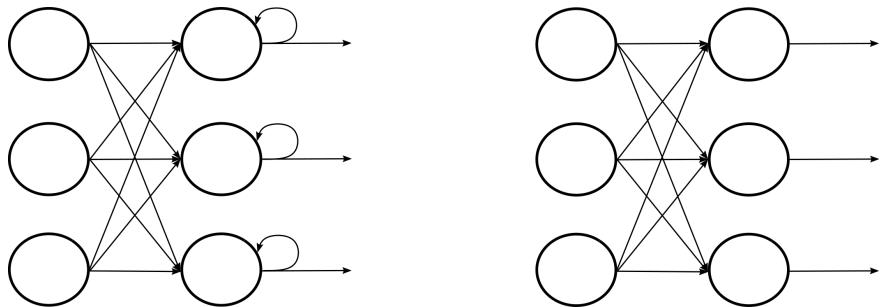


Figura 3.112: A la izquierda, una red neuronal recurrente. A la derecha, una red neuronal de alimentación progresiva.

Esta sección titulada Modelado de redes neuronales se centra en la arquitectura de red de alimentación progresiva, principalmente para limitar el alcance de este bloque. Además, su funcionamiento es mucho más simple y ayuda mejor a entender el comportamiento general de una red neuronal. No obstante, se recomienda la lectura [Ger19] para profundizar tanto en los fundamentos como en el comportamiento de las redes neuronales recurrentes.

3.7.4. Proceso de aprendizaje de una red neuronal

Diseño de una red neuronal

Antes de entrar en detalle en cómo aprende una red neuronal, es preciso primero discutir cuáles son las cuestiones importantes a resolver a la hora de diseñar su estructura para un problema determinado.

El diseño de las **entradas** y **salidas** de una red es, por lo general, directo, dado que, en la mayoría de los casos, estos elementos se conocen. Específicamente, el número de neuronas que componen la capa de entrada es igual a la cantidad de características en los datos. Por ejemplo, en el caso de diseñar una red neuronal para determinar si una imagen con texto escrito a mano es un número del 1 al 9, la forma más natural de resolver el problema es codificar las intensidades de los píxeles de la imagen en neuronas. Por lo tanto, si la imagen proporcionada es de 64×64 píxeles en una escala de grises, cuyas intensidades se encuentran entre 0 y 1, el número de entradas es de $4096 = 64 \times 64$. El número de salidas, por el contrario, depende del modelo de configuración elegido. En el caso del ejemplo planteado, la salida consistirá en 9 neuronas que atienden al número de dígitos a detectar.

No obstante, establecer el número de niveles intermedios y neuronas de la red no es para nada una tarea sencilla. De hecho, no existe un estándar o regla general que determine una configuración para un problema en particular; esta acción se considera un arte en el contexto de las redes neuronales. En cuanto al **número de capas ocultas**, y según varios teoremas de aproximación universal [Cyb89, HSW89, Bar93], una con bastantes entradas es normalmente suficiente

para aproximarse a cualquier función y, en definitiva, resolver problemas complejos. Obviamente, este regla sirve cuando no se requiere de **aprendizaje profundo**⁵⁹, donde un número mayor o igual a dos capas es lo habitual. Por otro lado, el **número de neuronas en cada capa** es otro de los aspectos que debe considerarse cuidadosamente. El uso de pocas neuronas en las capas ocultas puede dar lugar a lo que se denomina **subajuste** o *underfitting* (en inglés). Esto se produce cuando hay pocas neuronas en las capas ocultas para detectar adecuadamente las señales en un conjunto de datos complejo. Por el contrario, un uso excesivo de neuronas puede resultar en diversos problemas. Uno de ellos está relacionado con el **sobreajuste** u *overfitting* (en inglés), el cual ocurre cuando existe más capacidad de procesamiento que la cantidad de información a procesar. Esto, además, puede aumentar los tiempos de entrenamiento de la red, imposibilitando así la creación del modelo. Por suerte, existen diferentes reglas que ayudan a estimar el número de neuronas que podrían ser suficientes en las capas ocultas. En la mayoría de casos, el número de neuronas ocultas debe oscilar entre el tamaño de la capa de entrada y el tamaño de la capa de salida, aunque también es muy común utilizar el teorema de Kolmogorov [Kü92]: «El número de neuronas ocultas debe ser inferior al doble del tamaño de la capa de entrada». Existen también otros métodos como las **técnicas de regularización**, aunque no son estudiadas en este bloque para limitar su alcance. En la siguiente sección se tratan a alto nivel las más populares.



Aspectos relacionados con el número de neuronas. Por lo general, la precisión de la red aumenta conforme incrementa el número de neuronas. Sin embargo, a costa de esta ventaja, se pierde generalidad.

Obviamente, es necesario llegar a un compromiso en el número de neuronas en las capas ocultas de la red para mantener un equilibrio entre **precisión y generalidad**. El primero término se refiere al menor error posible que produce la salida de una red neuronal ante el conjunto de casos de entrada, mientras que el segundo se atribuye al número de casos posibles que puede abarcar una red.

También es interesante elegir adecuadamente la **función de activación** en cada capa de la red neuronal. En la mayoría de casos, la función ReLU (o alguna de sus variantes, como la función ELU) actúa bien en las capas intermedias (ocultas). Es un poco más rápida que otras funciones de activación, como son la función tangente hiperbólica o sigmoide. Para la capa de salida, la función de activación softmax suele ser una buena opción en tareas de clasificación (cuando las clases son mutuamente excluyentes). Para tareas de regresión, lo normal es utilizar la función identidad.

⁵⁹Este término es un subconjunto del aprendizaje automático, cuyo nombre se atribuye por el hecho de utilizar redes neuronales con varias capas en los niveles intermedios, particularmente cuando el número es mayor o igual a dos.

Finalmente, la **muestra de entrenamiento** o ejemplo es otra de las cuestiones importantes a tener en cuenta a la hora de diseñar una red neuronal artificial. Principalmente porque el comportamiento inteligente se consigue, por lo general, mediante el aprendizaje por ejemplos. Como símil, imagina este proceso igual al aprendizaje de un niño pequeño, pues es uno de los padres quién proporciona respuestas a las preguntas de este. Una red neuronal actúa de manera similar, haciendo preguntas sobre un tema muy concreto cuyas respuestas se proporcionan, de manera general, mediante un conjunto de datos. Este contiene las potenciales entradas que el modelo recibirá y la salida que se debería de producir como consecuencia de dichas entradas. Por ello, el **volumen** y **calidad** del conjunto de entrenamiento es muy importante en el diseño del modelo, ya que estas propiedades son decisivas para obtener una mayor **precisión** en las predicciones.

Mecanismos de aprendizaje

Como se ha discutido anteriormente, el aprendizaje de una red neuronal generalmente se realiza en base a un conjunto de ejemplos, aunque no siempre es realmente así. Este tipo de aprendizaje se denomina **aprendizaje supervisado**, el cual ya fue introducido en una de las secciones anteriores.



¿Cómo aprende una red neuronal? La forma en la que una red neuronal artificial puede aprender se clasifica fundamentalmente en tres categorías: aprendizaje supervisado, aprendizaje no supervisado y aprendizaje por refuerzo. Se invita al lector a profundizar en los mecanismos utilizados para los modos de aprendizaje no supervisado y por refuerzo, ya que esta sección dedica su contenido al primero por considerarse el método más común en este contexto.

Durante el entrenamiento de una red neuronal artificial bajo un aprendizaje supervisado, el proceso consiste en introducir a la red valores a sus entradas usando un conjunto de ejemplos, las cuales desencadenarán la generación de una o múltiples salidas, dependiendo particularmente de la estructura de la red. Esta salida se compara con el resultado deseado en base a las entradas proporcionadas del conjunto de ejemplo. La diferencia resulta ser el error que se genera a partir de la configuración de la red para realizar una predicción sobre unos valores de entrada. Sobre la base de este error, los pesos de la red se ajustan hasta que la salida real coincide, aunque no necesariamente igual, con la salida deseada.

El algoritmo base para ajustar los pesos de las neuronas fue introducido por el psicólogo Frank Rosenblatt en 1957, inspirado en la regla de **aprendizaje Hebb** por su autor Donald Hebb: «las neuronas que se activan simultáneamente refuerzan la sinapsis». En resumen, este método consiste en reforzar los pesos de las neuronas por un valor constante cuando las entradas y salidas se activan. En caso contrario, los pesos se quedan tal cual están. Se observa que el error, en este método, no es tenido en cuenta.

Cuando existe más de un nivel en la red, el método anterior resulta ineficiente y, por lo tanto, es necesario utilizar otro, como por ejemplo el **algoritmo de retropropagación** o *backpropagation* (en inglés). El algoritmo consta de dos pasos: hacia adelante y hacia atrás. La red, como ya se discutió anteriormente, se alimenta con el conjunto de datos diseñado para el entrenamiento, los cuales sirven para calcular la salida de la red (paso hacia adelante). Después, se computa su error, comparando la salida generada con la deseada. Más tarde, se procede a determinar, mediante un paso hacia atrás, cuánto ha contribuido cada neurona de la última capa al error de cada neurona de salida. Por tanto, se mide el error que procede de cada neurona de la capa anterior, y así sucesivamente hasta que el algoritmo llega a la capa de entrada. A continuación se proporciona en pseudocódigo una posible implementación del algoritmo *backpropagation* [RN20]:

INPUT α , una constante de velocidad de aprendizaje; *ejemplos*, un conjunto que contiene muestras e incluye un vector de entrada x y un vector de salida y ; *red*, una red compuesta de L capas, pesos $w_{i,j}$ y una función de activación g .

VARIABLES Δ , un vector de errores, indexado por los nodos de la red

REPEAT

```

FOR EACH peso  $w_{i,j}$  in red DO
     $w_{i,j} \leftarrow$  valor aleatorio pequeño

    /* Vuelta hacia adelante */

    FOR EACH ejemplo (in, out) in ejemplos DO
        FOR EACH nodo  $i$  in  $l = 1$  DO
             $x_i \leftarrow in_i$ 
        FOR EACH  $l = 2$  to  $L$  DO
            FOR EACH nodo  $j$  in  $l$  DO
                 $unidad_j \leftarrow \sum_{i=0} x_i w_{ij}$ 
                 $x_j \leftarrow g(unidad_j)$ 
            /* Vuelta hacia atrás */

            FOR EACH nodo  $j$  in  $l = L$  DO
                 $\Delta[j] \leftarrow g'(unidad_j) \times (out_j - x_j)$ 
            FOR EACH  $l = L - 1$  to  $1$  DO
                FOR EACH node  $i$  in  $l$  DO
                     $\Delta[i] \leftarrow g'(unidad_i) \sum_{j=0} w_{i,j} \Delta[j]$ 
                FOR EACH peso  $w_{i,j}$  in red DO
                     $w_{i,j} \leftarrow w_{i,j} + \alpha \times x_i \times \Delta[j]$ 

    UNTIL criterio de parada

RETURN red

```

En resumen, el algoritmo de retropropagación decide cuánto actualizar cada peso de la red tras comparar, para un ejemplo concreto, la salida obtenida con la salida deseada. En la vuelta hacia atrás, el algoritmo primero calcula el error generado en la capa de salida. Para ello, se calcula la derivada parcial del error con respecto al coste, es decir, el valor de la neurona de salida: $\Delta[j] \leftarrow g'(unidad_j) \times (out_j - x_j)$ (unión entre la última capa oculta y capa de salida). Como nota aclaratoria, g' representa la derivada de la función de activación g . Una vez conocido el error en la salida, queda por resolver el error en las capas sucesivas, es decir, capas anteriores a la salida. Aquí tiene lugar la **propagación posterior**, donde el error anteriormente calculado juega un papel importante. El nodo oculto anterior a la salida es responsable de una parte del error producido Δ en cada uno de los nodos de salida con los que conecta. En otras palabras, los nodos de la capa oculta tienen parte de culpa en el error producido en las salidas. Por lo tanto, los errores de la salida son divididos de acuerdo a la intensidad de las conexiones entre los nodos ocultos y las salidas, propagando el error hacia atrás para calcular el error del nodo oculto: $\Delta[i] \leftarrow g'(unidad_i) \sum_{j=0} w_{i,j} \Delta[j]$. Finalmente, los pesos de la red se calculan en base a cuatro valores: el peso inicial, una constante de velocidad de aprendizaje, el nodo de entrada y el error del nodo en el extremo de la conexión del peso. Así es la regla para actualizar los pesos de la red: $w_{i,j} \leftarrow w_{i,j} + \alpha \times x_i \times \Delta[j]$



Actualización de pesos. La idea que reside bajo este cálculo consiste en actualizar los pesos proporcionalmente en base a la diferencia entre el valor esperado (salida) y el real (valor obtenido), y proporcionalmente al valor de la entrada.

3.7.5. Caso práctico: implementación de una red neuronal para clasificar flores

Una vez introducido los elementos esenciales de una red neuronal y cómo esta funciona, veamos un ejemplo relativamente simple de un **clasificador** para distinguir **flores** de tres clases. Para ello se ha elegido el conjunto de datos *Iris* introducido por Donald Fisher en su artículo *The use of multiple measurements in taxonomic problems* [Fis36] de 1936. Este conjunto es popularmente conocido en el contexto de la ciencia de datos por utilizarse con alta frecuencia como ejemplo en el reconocimiento de patrones. Contiene 50 muestras de cada una de las tres especies Iris (Iris setosa, Iris virgínica e Iris versicolor), de las cuales se midieron cuatro rasgos fundamentales: el largo y ancho del sépalo y pétalo, en centímetros.



Figura 3.113: Variedad de clases de la flor iris.

En primer lugar, crearemos una **red neuronal desde cero**, y de manera incremental, con objeto de interiorizar los conceptos discutidos en este bloque. Más tarde, la red se entrenará para, dada una entrada, proporcionar como salida una especie de Iris. Es decir, unos valores de entrada del largo y ancho del sépalo y pétalo producirán un salida que corresponderá a Iris setosa, virgínica o versicolor. Se realizará además una pequeña manipulación en los datos para conseguir que la red efectúe predicciones lo mas precisas posibles.

Código de la red neuronal

El listado 3.18 muestra una posible implementación de la unidad base de una red neuronal, es decir, una neurona. Como se puede observar, esta abstracción proporciona una representación a alto nivel con tres elementos bien diferenciados: un listado de pesos, que serán números reales; un listado de entradas, que serán nodos de la red; y el valor de dicho nodo. Se podría añadir como cuarto elemento la función de activación del nodo. Sin embargo, este detalle se ha obviado en este nivel para simplificar la implementación.

Listado 3.18: Implementación de la unidad mínima de una red neuronal

```

1 #!/usr/bin/python3
2
3 class NetNode(object):
4     def __init__(self):
5         self.inputs = []
6         self.weights = []
7         self.value = None

```

En esencia, la implementación planteada está diseñada para crear una red que simule un grafo dirigido, dado que una neurona, en una red neuronal artificial, está unida a una serie de entradas cuya conexión contiene un peso. El valor del nodo (o neurona) es el resultado de aplicar una función de activación a la suma ponderada de pesos y entradas. En el listado 3.19 se refleja la construcción del grafo bajo la clase `Network`, que alojará los detalles técnicos de la red. De esta implementación destacan los siguientes puntos:

- Una red neuronal mediante esta implementación se inicializa proporcionando una lista con el número de neuronas de cada capa. Por ejemplo, una red con tres capas, en la primera dos neuronas, en la oculta tres y una en la de salida, se representa como [2, 3, 1].
- En la línea $\Theta 5$ se crea la red, la cual se representa como una lista de listas. Cada elemento actúa como una capa que contiene un conjunto de nodos.
- En las líneas $\Theta 7-12$ se crean las conexiones de la red o grafo. Concretamente, el bucle comienza iterando desde la segunda capa de la red con el objetivo de agregar en sus nodos las entradas de la capa anterior. Junto a esta operación se inicializan los pesos de cada nodo al valor cero.

Listado 3.19: Inicialización de la red neuronal

```

1 class Network(object):
2     def __init__(self, layers):
3         self.net = [[NetNode() for _ in range(size)] for size in layers]
4
5         sizes = len(layers)
6         for layer in range(1, sizes):
7             for node in self.net[layer]:
8                 for unit in self.net[layer - 1]:
9                     node.inputs.append(unit)
10                    node.weights.append(0)

```

Se ha optado por añadir las funciones de activación pertinentes (véase listado 3.20) al nivel de la clase `Network` para facilitar la construcción del código. A modo aclaratorio, estas funciones se utilizarán en cada capa de la red. El principal objetivo de este ejemplo es que la red aprenda para clasificar correctamente, no realizar una implementación exhaustiva de una red neuronal. No obstante, la mayoría de módulos predefinidos ofrecen, por el contrario, una mayor granularidad, permitiendo añadir una función diferente en cada nivel de la red.

Listado 3.20: Implementación de la función ReLU y su derivada

```

1 def relu(self, z):
2     return max(0, z)
3
4 def relu_prime(self, z):
5     return 1 if z > 0 else 0

```

Una vez implementados todos los elementos de la red neuronal, veamos cómo es el código encargado de alimentar la red y producir salidas. El listado 3.21 refleja una función que, en resumen, recibe un ejemplo y lo envía a la red para generar una predicción. Más detalladamente, los valores del ejemplo se envían primero a la capa de entrada, los cuales pasan más tarde por el resto de niveles de la red. Cada nodo calcula el producto escalar de sus entradas y sus pesos, cuyo valor es aplicado a un función que determina si la neurona se activa o no. De ser así, el valor pasa a la

siguiente capa, y así sucesivamente hasta llegar al último nivel. La predicción final es el nodo de la capa de salida con el valor máximo. Obviamente, en este punto la red arrojará un resultado incorrecto, ya que todos sus pesos contienen los mismos valores desde el momento de su inicialización, es decir, cero.

Listado 3.21: Implementación para realizar predicciones con una red neuronal

```
1 import numpy as np
2
3 def predict(self, input_data):
4     inputs = self.net[0]
5
6     # Initialize inputs
7     for value, node in zip(input_data, inputs):
8         node.value = value
9
10    # Forward step
11    for layer in self.net[1:]:
12        for node in layer:
13            in_val = [n.value for n in node.inputs]
14            unit_value = np.dot(in_val, node.weights)
15            node.value = self.relu(unit_value)
16
17    outputs = self.net[-1]
18    return outputs.index(max(outputs, key=lambda node: node.value))
```

El problema ahora a resolver es cómo entrenar la red neuronal para actualizar sus pesos y producir una salida de acuerdo a los valores de entrada proporcionados. En otras palabras, cuáles deberían ser los pesos para que, dado unos valores del sépalo y pétalo, la red neuronal determine que esa información corresponde a Iris setosa, virgínica, o versicolor. Para cumplir esto, es necesario utilizar el algoritmo *backpropagation*. En el listado 3.22 se incluye una posible implementación. En resumen, se realiza una clasificación, se comprueba si es correcta o no y se calcula cómo de lejos ha sido el fallo. Después, el error se propaga hacia atrás en la red, ajustando los pesos de los nodos en consecuencia. El algoritmo se ejecuta en el conjunto de datos de entrenamiento durante un número determinado de veces, cuyo término corresponde a *epochs*.

Los puntos más importantes del algoritmo se describen a continuación:

- El algoritmo recibe como entrada una constante de velocidad de aprendizaje (*eta*), un conjunto de datos de entrenamiento (*examples*) y el número de veces que se itera sobre estos (*epoch*s).
- En las líneas **⑨-12** se inicializan los pesos de la red neuronal siguiendo una distribución uniforme $[0, 1]$. El criterio de aleatoriedad es realmente importante, pues su elección tiene un impacto considerable en el aprendizaje de la red. Existen varias técnicas, las cuales se discutirán en la siguiente sección.
- La variable local *delta* (línea **②8**) es un vector de errores constituido por una lista de listas. Cada elemento representa una capa, la cual contiene el error de cada uno de sus nodos.

- En la línea Θ 31 se calcula el error producido en la salida, utilizando el valor esperado (salida del conjunto de entrenamiento) y el valor real (salida de la red neuronal).
- Las líneas Θ 34-48 reflejan el comportamiento de propagar el error desde la capa de salida a la capa de entrada.
- La última parte del algoritmo se ocupa de actualizar los pesos de cada capa de la red en base a los errores de sus nodos (líneas Θ 51-57). Concretamente, esta operación, en la línea Θ 57, combina una constante de velocidad de aprendizaje con el error del nodo actual y el valor de los nodos de la capa anterior. A esto se le suma los pesos iniciales entre el nodo actual y los de la capa anterior. Y así sucesivamente hasta llegar a la capa de salida. Como nota aclaratoria, la utilización de la constante η sirve para incrementar la velocidad de aprendizaje.

Listado 3.22: Implementación del algoritmo *backpropagation*

```
1 import numpy as np
2
3 def backpropagation(self, eta, examples, epochs):
4     inputs = self.net[0]
5     outputs = self.net[-1]
6     layer_size = len(self.net)
7
8     # Initialize weights
9     for layer in self.net[1:]:
10         for node in layer:
11             node.weights = [np.random.uniform()
12                             for _ in range(len(node.weights))]
13
14     for epoch in range(epochs):
15         for x_train, y_train in examples:
16             # Initialize inputs
17             for value, node in zip(x_train, inputs):
18                 node.value = value
19
20             # Forward step
21             for layer in self.net[1:]:
22                 for node in layer:
23                     in_val = [n.value for n in node.inputs]
24                     unit_value = np.dot(in_val, node.weights)
25                     node.value = node.activation(unit_value)
26
27             # Initialize delta
28             delta = [[] for _ in range(layer_size)]
29
30             # Error through cost function
31             err = [y_train[i] -
32                   outputs[i].value for i in range(len(outputs))]
33
34             delta[-1] = [self.relu_prime(outputs[i].value) * err[i]
35                         for i in range(len(outputs))]
36
37             # Backward step
```

```

38         hidden_layers = layer_size - 2
39         for i in range(hidden_layers, 0, -1):
40             layer = self.net[i]
41             n_layers = len(layer)
42
43             # Weights from the last layer
44             w = [[node.weights[] for node in self.net[i + 1]]
45                  for l in range(n_layers)]
46
47             delta[i] = [self.relu_prime(
48                 layer[j].value) * np.dot(w[j], delta[i + 1]) for j in range(n_layers)
49                 ]
50
51             # Update weights
52             for i in range(1, layer_size):
53                 layer = self.net[i]
54                 in_val = [node.value for node in self.net[i - 1]]
55                 n_layers = len(layer)
56                 for j in range(n_layers):
57                     layer[j].weights = np.add(
58                         layer[j].weights, np.multiply(eta * delta[i][j], in_val))
59
60             print(
61                 f"epoch {epoch}/{epochs} | total error={np.sum(err)/len(examples)}")

```

Finalmente, se ha incluido una función a la clase `Network` para calcular la precisión de la red neuronal (véase listado 3.23). En resumen, la función recibe un conjunto de ejemplos, distintos a los que se han usado para entrenar la red, y retorna un valor entre 0 y 1, el cual determina cómo de precisas son las predicciones que el modelo realiza. Para ello, se itera sobre el conjunto de ejemplos y se calcula la predicción para cada muestra, incrementando un contador en base a los resultados positivos. Al final, la precisión viene determinada por el número de aciertos frente al conjunto total.

Listado 3.23: Código para calcular la precisión de la red neuronal

```

1 def accuracy(self, examples):
2     correct = 0
3
4     for x_test, y_test in examples:
5         prediction = self.predict(x_test)
6
7         if (y_test[prediction] == 1):
8             correct += 1
9
10    return correct / len(examples)

```

Código de entrenamiento de la red

Terminada la implementación de los detalles técnicos de la red neuronal, pasemos a clasificar flores Iris. En general, antes de introducir los datos a la red, se suele realizar un ajuste o manipulación para preparar el conjunto acorde al problema a resolver. El listado 3.24 refleja esta consideración, además de calcular la precisión de la red. Los siguientes puntos tratan de explicar, de manera sintetizada, los detalles del código.

- Las líneas $\Theta 3\text{-}7$ reflejan los paquetes importados para manipular el conjunto de datos. En resumen, las bibliotecas `sklearn` y `keras` sirven para el tratamiento de los datos, y `network` para la creación de la red neuronal.
- `Sklearn` incorpora unos pequeños conjuntos de datos estándar que no requieren la descarga de ningún archivo desde algún sitio web externo. Gracias a esta funcionalidad, la línea $\Theta 9$ refleja cómo cargar el conjunto Iris.
- Los datos de entrada del conjunto contiene rangos de valores diferentes. Por lo tanto, para mantener una buena precisión, una regla muy utilizada es normalizar los datos en un rango de 0-1. Esto se realiza en la línea $\Theta 11$.
- En los procesos de entrenamiento es muy común dividir el conjunto en dos muestras: una para alimentar la red y otra para pruebas. En general, la regla que se utiliza es reservar un 80 % del conjunto para entrenar el modelo y el 20 % restante para las pruebas. Además, los datos se mezclan para mantener aleatoriedad en diferentes ejecuciones. En las líneas $\Theta 13\text{-}14$ se refleja esta consideración.
- El conjunto de datos contiene tres salidas (0, 1, 2) que se corresponden a los valores categóricos ‘Setosa’, ‘Versicolor’ y ‘Virginica’. Dado que se ha optado por normalizar las entradas del conjunto, es recomendable seguir la misma regla para las salidas. En este caso no se normalizan, sino que se codifican. Para ello se ha utilizado la técnica *one-hot* que convierte los valores en un vector binario (un 1 en el índice de la clase y el resto de valores a 0). Para este ejemplo, si el valor de la clase es 2, la matriz será [0, 0, 1].
- En las líneas $\Theta 19\text{-}21$ se crea el conjunto de ejemplos que recibirá el algoritmo *backpropagation* para entrenar la red.
- En la línea $\Theta 23$ se crea una red neuronal con tres capas, 4 neuronas en la entrada, 7 en la capa oculta y 3 en la salida. En la línea $\Theta 24$ se entrena la red mediante el algoritmo de retropropagación, el cual recibe como parámetros una constante de aprendizaje definida con el valor 0,1, un conjunto de entrenamiento y el número de veces que se iteran las muestras. Después, a modo de prueba, se realiza una predicción con el primero elemento del conjunto.
- Finalmente, en las líneas $\Theta 27\text{-}29$ se construye el conjunto de ejemplos, el cual es utilizado para calcular la precisión del modelo entrenado, reflejado en la línea $\Theta 31$.

Listado 3.24: train.py; código relativo a la manipulación de los datos para entrenar la red

```
1 #!/usr/bin/python3
2
3 from sklearn import datasets
4 from sklearn.preprocessing import normalize
5 from sklearn.model_selection import train_test_split
6 from keras.utils import np_utils
7 from network import Network
8
9 iris_X, iris_y = datasets.load_iris(return_X_y=True)
10
11 iris_X_normalized = normalize(iris_X, axis=0)
12
13 X_train, X_test, y_train, y_test = train_test_split(
14     iris_X_normalized, iris_y, test_size=0.2, shuffle=True)
15
16 y_train = np_utils.to_categorical(y_train, num_classes=3)
17 y_test = np_utils.to_categorical(y_test, num_classes=3)
18
19 examples = []
20 for i in range(len(X_train)):
21     examples.append([X_train[i], y_train[i]])
22
23 net = Network([4, 7, 3])
24 net.backpropagation(0.1, examples, 500)
25 prediction = net.predict(X_test[0])
26
27 examples = []
28 for i in range(len(X_test)):
29     examples.append([X_test[i], y_test[i]])
30
31 accuracy = net.accuracy(examples)
32 print(f"Accuracy: {accuracy}")
33
34 print(f"Desired output: {y_test[0]}")
35 print(f"Index of output: {prediction}")
```

La siguiente salida muestra un ejemplo de ejecución del programa *train.py*. Como se puede apreciar, la salida refleja la precisión del modelo y el ejemplo de una predicción. El valor de salida real del modelo es 1, lo que significa que el elemento clasificado es la segunda clase, es decir, *Versicolor*. A modo recordatorio, la salida del modelo es el nodo de la capa de salida con el máximo valor. En este caso, 1 representa la segunda posición del vector de nodos de la capa de salida, el cual equivale al vector de salida deseado del conjunto de prueba.

```
$ Python3 train.py
```

```
Accuracy: 0.9666666666666667
Desired output: [0. 1. 0.]
Index of output: 1
```



Exploración del conjunto de datos. Se recomienda al lector explorar el conjunto de datos Iris obviando el parámetro `return_X_y=True` en `dataset.load_iris`. Así, la función retorna una estructura de datos en formato *pandas*, lo cual permite estudiar el conjunto con las funciones `describe` y `head`. En la dirección Enlace: https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_iris.html se encuentra información detallada sobre los atributos del conjunto retornados por la función.

3.7.6. Aplicaciones generales

Las redes neuronales artificiales siguen siendo, a fecha de la redacción de este capítulo, una de las tecnologías con mayor popularidad en el campo de la IA, particularmente por la habilidad de reproducir y modelar procesos no lineales. En general, este modelo computacional es fundamentalmente útil para el aprendizaje de patrones, clasificación y predicción. Este enfoque juega un papel muy importante principalmente motivado por las siguientes razones:

- Encuentran soluciones para problemas cuya complejidad algorítmica es costosa, o incluso no existe aún un método para tal cometido.
- Aprenden a base de ejemplos. Por lo tanto, no es necesario programarlas en gran medida.
- Mantienen una precisión y velocidad significativamente más rápida comparadas con los enfoques convencionales, principalmente cuando se aprovecha el procesamiento paralelo.

Este modelo matemático se constituye como un enfoque interdisciplinario y está constantemente en evolución. Prueba de ello son la creación de modelos impresionantes de la mano de DeepMind⁶⁰ u OpenAI⁶¹ con WaveNet [O⁺16], GPT-3 [B⁺20], DALL-E [R⁺21] o Codex [C⁺21], entre muchos otros. A la vista está que son muchas las técnicas que surgen y poco a poco se adoptan para hacer frente a retos de visión por computador, el reconocimiento del habla y de los patrones, o la alineación y la detección de rostros. Esta sección resume una serie de áreas de aplicación donde las redes neuronales juegan un papel muy importante.

- Reconocimiento de voz.
- Procesamiento del lenguaje natural.
- Reconocimiento facial.
- Reconocimiento de patrones.
- Diagnóstico de enfermedades.

⁶⁰ Enlace: <https://deepmind.com/>

⁶¹ Enlace: <https://openai.com/>

- Compresión de datos.
- IA en videojuegos.
- Y un largo etc.

3.7.7. Resumen

En esta sección se ha profundizado sobre los aspectos fundamentales de las redes neuronales. A continuación se remarcán los puntos más importantes.

- Las **redes neuronales artificiales** recogen ideas del sistema neuronal biológico. Concretamente, las **neuronas artificiales** son la unidad mínima en el modelo, las cuales mantienen conexiones entre sí como su análoga biológica, con la diferencia que cada enlace contiene un **peso numérico**. La **activación** de la conexión se replica con una **función matemática**, la cual le asigna un valor a la neurona final en la conexión. Este valor se propaga hacia adelante, influyendo en las siguientes neuronas hasta alcanzar el último nivel de la red, es decir, la neurona de salida.
- El modelo más simple de red neuronal se conoce como **perceptrón**, el cual está compuesto de varias entradas y una salida.
- Las redes neuronales artificiales se clasifican en base a dos tipos bien diferenciados: redes **monocapa** o **multicapas**. En el segundo tipo, las neuronas concentradas en los niveles intermedios de la red, es decir, entre las entradas y salidas, se conocen como **capas ocultas**.
- Existe multitud de tipos de estructuras de red y cada una de ellas proporcionan diferentes características de cómputo. Entre las más usadas y conocidas se encuentran las redes de **alimentación progresivas** y **recurrentes**.
- El **algoritmo de retropropagación** o *backpropagation* permite entrenar una red neuronal, actualizando sus pesos en base al error de sus salidas.
- **Cuatro consideraciones** se deben tener en cuenta durante el diseño de una red neuronal: número de entradas, número de capas, número de neuronas en las capas ocultas y la muestra de entrenamiento.
- Esta sección termina con un **caso práctico** para poner en práctico los conceptos discutidos. Para ello, se ha creado una red neuronal desde cero, la cual se ha utilizado para **clasificar flores Iris**.

3.8. Modelado avanzado de redes neuronales

Esta sección se constituye como una continuación al bloque anterior titulado «Modelado de redes neuronales», el cual trata de profundizar sobre su creación con la ayuda de módulos ya previamente definidos. Su uso da pie a tratar algunas técnicas para mejorar el proceso de aprendizaje, las cuales serán discutidas a alto nivel. En esencia, el objetivo de esta sección es ofrecer al lector las herramientas necesarias para construir eficientemente redes neuronales, dejando a un lado los detalles avanzados de estas. Por último, se incluye un caso práctico, el cual muestra la resolución de un problema utilizando una red neuronal creada con un módulo predefinido.

3.8.1. Técnicas avanzadas de aprendizaje

El diseño de una red neuronal, como ya se discutió en la sección anterior, implica elegir adecuadamente diversos elementos que entre sí conformen un modelo que permita resolver un determinado problema. A modo de recordatorio, el número de neuronas en la capa entrada y salida, así como el número de capas intermedias y unidades en ellas se considera un punto fundamental respecto a la efectividad de la red. También la calidad del conjunto de datos o la función de activación que se aplica en cada nivel de la red. Sin embargo, existen más cuestiones que tienen un peso importante después del diseño de su arquitectura y que afectan considerablemente a su aprendizaje. Por ello, las siguientes subsecciones recopilan una serie de métodos, los cuales suelen aplicarse a estos modelos para mejorar problemas producidos en la fase de aprendizaje.

Inicialización de pesos

El algoritmo de retropropagación requiere un punto de partida en el espacio de posibles valores de los pesos. La idea de utilizar un enfoque no determinista frente a uno determinista se resume en el rendimiento del algoritmo, fundamentalmente porque la segunda opción resulta ineficiente para problemas complejos.

Por lo general, los pesos no deben inicializarse al valor 0, ya que esto impide que el algoritmo de ajuste busque de forma efectiva los valores correctos para realizar adecuadas predicciones.

Históricamente, la inicialización de pesos se ha venido definiendo mediante heurísticas simples, como por ejemplo:

- Pequeños valores aleatorios entre [0, 1]
- Pequeños valores aleatorios entre [-1, 1]
- Pequeños valores aleatorios entre [-0.1, 0.1]
- Otras similares.

En general, la inicialización de los pesos se realiza siguiendo una distribución gaussiana o uniforme. Sin embargo, la escala de la distribución inicial tiene un gran efecto tanto en el resultado del procedimiento de optimización como en la capacidad de generalización de la red. En definitiva, la inicialización de los pesos afecta considerablemente en el proceso de aprendizaje de la red. Estas técnicas continúan funcionando en general. Sin embargo, se han desarrollado nuevos enfoques que se han convertido en la norma de facto, dado que pueden dar lugar a un entrenamiento del modelo muchos más optimizado.

En esencia, estas nuevas heurísticas se clasifican en función del tipo de función de activación utilizada, principalmente destinadas a la función sigmoide, tangente hiperbólica y ReLU.

El método utilizado para inicializar los pesos de una red con funciones de activación sigmoide o tangente hiperbólica se denomina **inicialización de Glorot** o **inicialización de Xavier** [GB10], nombres por su autor Xavier Glorot. Los pesos de las conexiones deben inicializarse aleatoriamente como se describe en 3.7 y 3.8, donde n_{org} y n_{dest} son el número de conexiones de origen y destino para la capa cuyas conexiones están siendo actualizadas. Existen dos alternativas:

Con una distribución normal con media igual a 0 y una desviación estándar:

$$\sigma = \sqrt{\frac{2}{n_{org} + n_{dest}}} \quad (3.7)$$

O una distribución uniforme entre $-r$ y $+r$, siendo r:

$$r = \sqrt{\frac{6}{n_{org} + n_{dest}}} \quad (3.8)$$

Por otro lado, la estrategia de inicialización para la función de activación ReLU (y sus variantes, incluida la activación ELU) se denomina **inicialización He** [HZRS15], por el autor de este método Kaiming He. Para esta opción existen también dos alternativas:

Con una distribución normal con media igual a 0 y una desviación estándar:

$$\sigma = \sqrt{\frac{2}{n_{org}}} \quad (3.9)$$

O una distribución uniforme entre $-r$ y $+r$, siendo r:

$$r = \sqrt{\frac{6}{n_{org}}} \quad (3.10)$$



Inicialización de Xavier. Nótese que cuando el número de conexiones de entrada es igual al número de conexiones de salida, las ecuaciones se convierten más simples. Es decir, 3.7 cambiaría a $\sigma = \frac{1}{\sqrt{n_{org}}}$ y 3.8 sería $r = \frac{\sqrt{3}}{\sqrt{n_{org}}}$.

Función de pérdida

Este elemento en la arquitectura de la red neuronal corresponde al cálculo que se realiza para obtener el error durante el entrenamiento. Esto se discutió vagamente en el bloque anterior para actualizar los pesos de la red con el algoritmo de retropropagación.

La elección correcta de la función de pérdida es extremadamente importante, pues es la clave para conseguir una red que realice predicciones efectivas. En esencia, se evalúan las desviaciones entre las predicciones realizadas por la red y los valores reales de las observaciones durante el aprendizaje.

Afortunadamente, cuando se trata de problemas comunes como la clasificación o la regresión, hay pautas sencillas que se pueden seguir para elegir la pérdida correcta.

En clasificación binaria se utiliza **entropía cruzada binaria** o *binary cross entropy* (en inglés). Esta función de pérdida, definida en 3.11, se utiliza a menudo para una red con una salida que predice dos clases (normalmente una pertenencia de clase positiva para la salida 1 y negativa para la 0). En su definición, y' es el valor deseado e y es el valor real de salida de la red.

$$\mathcal{L}_{BCE} = -(y' \ln(y) + (1 - y') \ln(1 - y)) \quad (3.11)$$

Por el contrario, si el número de clases es mayor de 2 (es decir, clasificación multiclas), se utiliza la función de pérdida **entropía cruzada categórica** o *cross entropy* (en inglés). En esencia, calcula la pérdida para cada clase de la observación y se suma el resultado. En su definición (véase 3.12) i corresponde a la salida actual del total para un ejemplo de entrada.

$$\mathcal{L}_{CE} = - \sum_i y'_i \log(y_i) \quad (3.12)$$

Por otro lado, la función de **error cuadrático medio** o *mean squared root* (en inglés) se utiliza para problemas de regresión. En su definición (véase 3.13) n corresponde al número de neuronas de salida e i la salida actual total para un ejemplo de entrada.

$$\mathcal{L}_{MSE} = \frac{1}{n} \sum_i (y'_i - y_i)^2 \quad (3.13)$$



Aplicación de la función de pérdida. Es importante destacar que la función de pérdida no se utiliza tal cual es su definición, sino que se calcula su derivada en el paso hacia atrás del algoritmo de retropropagación para ajustar los pesos de la red.

Optimizadores

Como bien sabemos, entrenar una red neuronal no es para nada una tarea sencilla. De hecho, tanto en este bloque como en el anterior, se han visto estrategias para mejorar esta tarea, como elegir adecuadamente qué función de activación utilizar o aplicar una buena estrategia de inicialización de pesos. Incluso es a menudo normal utilizar la estructura de una red neuronal ya existente para cumplir una tarea similar.

Alternativamente a estos enfoques existen algoritmos, como los optimizadores, que tratan de mejorar el aprendizaje total de la red ajustando sus parámetros. En esencia, un optimizador, en el contexto de las redes neuronales, determina cómo se actualizará el grafo según la función de pérdida. En matemáticas, esto es un proceso usado para resolver un problema de optimización al minimizar o maximizar una función. En este caso, se trata de minimizar la función de pérdida, actualizando atributos como los pesos o la tasa de velocidad de aprendizaje.

Intuitivamente, este proceso puede representarse como un excursionista que intenta bajar una montaña con los ojos vendados. Es imposible saber en qué dirección debe ir, pero hay una cosa que sí conoce: cuando está bajando (progresando) y cuando está subiendo (perdiendo progreso). Al final, si sigue dando pasos que la lleven hacia abajo, llegará a la base.

Del mismo modo es imposible saber cuál debe ser el peso de nuestro modelo desde el principio. Sin embargo, usando un proceso similar al de prueba y error, los pesos acabarán teniendo un valor que produzca un error mínimo en la red (si el excursionista está descendiendo puede que acabe llegando a la base).

El **descenso del gradiente** es el algoritmo de optimización más popular y a la vez más básico que existe. En la figura 3.114 se muestra una representación intuitiva de su funcionamiento. Internamente, este optimizador es utilizado por el algoritmo de retropropagación, el cual se implementó en el bloque anterior. El descenso del gradiente se encarga de calcular en qué sentido deben modificarse los pesos para que la función de coste alcance un mínimo. A través de la propagación hacia atrás, la pérdida se transfiere de una capa a otra y los pesos del modelo se modifican. En este caso, la tasa de aprendizaje tiene una importancia considerable, ya que su valor determina cómo de rápido o lento se alcanzan los pesos óptimos. Más adelante discutiremos cuál es el valor adecuado para la tasa de aprendizaje, así como los problemas que puede ocasionar un valor inapropiado.

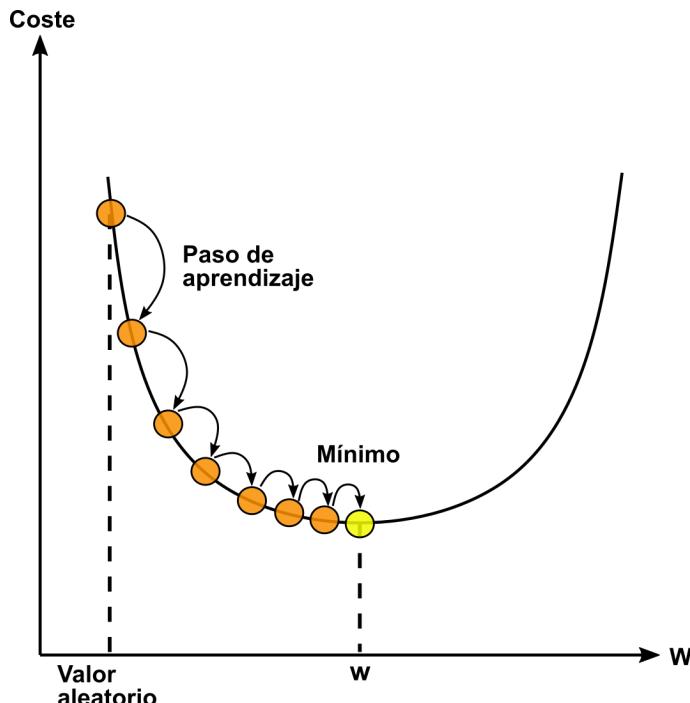


Figura 3.114: Representación intuitiva del algoritmo del descenso del gradiente.

Existen variantes del algoritmo anterior, como el **descenso del gradiente estocástico**, que agiliza la actualización de pesos al realizarse sobre cada ejemplo de entrenamiento, y no sobre el conjunto completo. En relación a este último algoritmo existen extensiones que se basan en tener una **tasa de aprendizaje adaptativa** para cada uno de los pesos. Es decir, su valor será modificado automáticamente durante el entrenamiento hasta alcanzar el valor óptimo que ajuste adecuadamente los pesos de la red. Entre ellos destacan **RMSProp**, **Adagrad** y **Adam**. En la práctica, Adam suele ser el algoritmo de optimización recomendado por los resultados que aporta. Sin embargo, este algoritmo, y los demás, contienen otros hiperparámetros diferentes de la tasa de aprendizaje que podrían ser necesarios ajustar. Profundizar en estos algoritmos está fuera del alcance de este bloque, por lo que se invita al lector a estudiarlos y conocer los detalles técnicos de su implementación.

Constante de velocidad de aprendizaje

La constante de velocidad de aprendizaje es una cantidad que determina cuánto actualizar los pesos durante el entrenamiento. Específicamente, la tasa de aprendizaje es un hiperparámetro configurable que es usado durante el aprendizaje, cuyo valor a menudo se define entre 0 y 1. Este elemento se suele representar mediante la letra griega minúscula η .

El valor que se establece a esta constante antes del aprendizaje es esencial, pues un valor alto puede producir que el error aumente, en lugar de disminuirlo. Por el contrario, cuando la tasa de aprendizaje es demasiado pequeña, el entrenamiento no sólo es más lento, sino que puede quedar permanentemente atascado con un error alto. Por lo tanto, es recomendable configurar la tasa de aprendizaje evitando los extremos, e ideal encontrar un valor que sea lo suficientemente bueno para aproximar los pesos al problema. A modo de curiosidad, algunos autores consideran este hiperparámetro como el más importante, recomendando solo su ajuste ante otros en caso de escasez de tiempo para entrenar la red [GBc16].

Un valor tradicional por defecto que es muy común utilizar para la tasa de aprendizaje es 0,1 o 0,01, el cual suele ser un buen punto de partida en el aprendizaje de la red. Esto puede ser a menudo óptimo, pero suele resultar más efectivo empezar con un valor alto y reducir este a medida que avanza el proceso de entrenamiento. En este sentido, existen diferentes estrategias, denominadas **esquemas de tasa de aprendizaje**, las cuales permiten reducir su valor durante el entrenamiento. Este esquema opera de forma independiente a los enfoques basados en tasas de aprendizaje adaptativas, como RMSProp, Adagrad o Adam. Se recomienda utilizar el descenso del gradiente estocástico cuando se usa un esquema de tasa de aprendizaje. A continuación se describen brevemente algunas de las más comunes en la práctica.

- **Disminución constante a trozos.** La idea es establecer valores de tasas de aprendizaje para intervalos de iteraciones (*epochs*) durante el entrenamiento. Por ejemplo, se podría inicializar $\eta = 0,1$ para las primeras 50 iteraciones. En el intervalo de iteración 51 a 100, η podría pasar a valer 0,001. Esta solución suele funcionar bien, aunque a menudo es necesario experimentar para averiguar los valores adecuados y cuándo utilizarlos.
- **Disminución exponencial.** En este caso se disminuye la tasa de aprendizaje a medida que el entrenamiento progresá. Su cálculo se realiza de la siguiente manera: $\eta = \eta_0 \lambda^{t/r}$, donde λ es una tasa de caída, r denota cada cuánto cae la tasa y t el índice actual de iteración durante el entrenamiento. Por ejemplo, $\eta = 0,1 \times 0,5^{t/100}$ significa que la tasa de aprendizaje disminuirá con un factor de 0,5 cada 100 iteraciones.
- **Disminución inversa del tiempo.** Esta estrategia opera de manera similar al enfoque previo, con la diferencia que se modifica η en función de la tasa de aprendizaje de la iteración anterior. Su cálculo se realiza de la siguiente manera: $\eta = \eta_0 / (1 + \lambda \times t/r)$.

Técnicas de regularización

Hasta ahora se han discutido técnicas para favorecer que el modelo, aún no entrenado, consiga ajustar adecuadamente sus parámetros en base al problema a resolver. Si bien es cierto que lo más importante es que la red aprenda, en ocasiones surge el problema de que el modelo entrenado clasifica realmente bien solo para el conjunto de entrenamiento, obteniendo resultados totalmente contrarios a si las predicciones se efectúan sobre otro distinto. Este aspecto se considera *overfitting*, concepto que ya fue introducido en el bloque anterior.

Fundamentalmente, el objetivo de la IA, y más concretamente del aprendizaje automático, es generalizar bien desde los datos de entrenamiento a cualquier otro dato del dominio del problema. En esencia, los modelos se diseñan para que se realicen predicciones en un futuro sobre datos que nunca el modelo ha visto. En este sentido, existen técnicas, métodos o heurísticas que tratan de conseguir redes totalmente generales y así obtener buenos resultados ante datos de entrada para los cuales la red ha sido previamente entrenada. A continuación se discuten brevemente algunas de las técnicas de regularización más populares.

- **Early Stopping.** En castellano se conoce como «parada temprana». Esta técnica de regularización interrumpe el entrenamiento cuando su rendimiento en el conjunto de validación comienza a caer. En esencia, la idea es evaluar el modelo en un conjunto de validación a intervalos regulares (por ejemplo, cada 50 *epochs*), y guardar una instantánea, potencialmente «ganadora», si la anterior es peor. Se cuenta el número de pasos desde que se guardó la última instantánea y se interrumpe el proceso cuando el error de validación aumenta de forma sostenida o no existe mejora tras un número específico de épocas. Finalmente, se restaura la última instantánea, que es potencialmente la «ganadora». En general, *early stopping* resulta funcionar bien en la práctica, aunque es recomendable combinarla con otra técnica de regularización para conseguir un mejor resultado.
- **Dropout.** En castellano se suele utilizar el término de «abandono». El procedimiento es el siguiente: en cada paso del entrenamiento, cada entrada (incluyendo entradas, pero excluyendo salidas) tiene una probabilidad p de ser temporalmente descartada o ignorada. Después del entrenamiento, las neuronas ignoradas son de nuevo activadas, por lo que es necesario realizar una compensación que recalcule en la inferencia la salida, pues habrá más activaciones contribuyendo a este valor. Un ejemplo de dicha compensación podría ser multiplicar todos los parámetros por la probabilidad contraria a la de descarte.
- **L1 & L2.** Son técnicas que consisten en reducir las conexiones de los peso de la red para que sean pequeños. Para ello se introduce un término de penalización en la función de coste original \mathcal{L} , añadiendo a su valor la suma de los cuadrados de los pesos (técnica L1) o el valor absoluto de la suma (técnica L2). No obstante, es posible que el término de penalización sea alto, hasta el

punto que la red minimizaría la función de coste, estableciendo los pesos a valores muy cercanos a 0, lo cual no es conveniente. Es por ello que se utiliza una constante l pequeña, cuyo valor suele ser 0.1, 0.01, etc., que multiplica a la suma.

- **Data Augmentation.** Consiste en generar muestras ligeramente nuevas en el conjunto de entrenamiento al aplicar transformaciones sobre las entradas originales. Por ejemplo, si se quiere clasificar un tipo concreto de árbol, se pueden realizar transformaciones al conjunto de entrenamiento como rotaciones, redimensionamientos, ajustes de brillo, deformaciones, etc. En definitiva, modificaciones en las imágenes que resultan en nuevas muestras y que incrementan el *dataset* de entrenamiento. Esta técnica se utiliza a menudo en el campo de la visión por computador, dado que los resultados obtenidos son sorprendentes.



Una imagen vale más que mil palabras. Una visualización clara y precisa de un concepto abstracto puede llegar a ser una herramienta totalmente poderosa. Atendiendo a esta consideración, se recomienda al lector visitar la herramienta Playground en Enlace: <https://playground.tensorflow.org/> para jugar con redes neuronales (añadiendo/quitando neuronas o modificando sus parámetros) y aprender así sus conceptos básicos.

3.8.2. Módulos predefinidos

Como ya se observó en el bloque anterior, la modelización de la inteligencia bajo una red neuronal no es para nada una tarea sencilla. Es decir, su diseño, en términos de codificación, no entraña una complejidad excesiva, pero sí los algoritmos y operaciones que generan un modelo inteligente a partir de un conjunto de datos. Además, existe la posibilidad que para un problema concreto el desarrollo no sea escalable y una modificación del código sea necesaria. Ante esta problemática, y la popularidad de este enfoque ante la resolución de problemas, surgen paquetes, con módulos predefinidos, que facilitan al desarrollador la creación de redes neuronales.

En anteriores secciones se han discutido brevemente algunos módulos utilizados hoy en día en el contexto de las redes neuronales. Con el objetivo de ofrecer una mayor panorámica al lector, los siguientes puntos presentan un conjunto de bibliotecas representativas de código libre para modelar redes neuronales en Python, elegidas en base a su popularidad en la plataforma GitHub. Se han obviado aquellas que parecen estar desactualizadas, principalmente por su inactividad en el repositorio.

- **Tensorflow**⁶²
- **PyTorch**⁶³
- **Sklearn**⁶⁴
- **Fast.ai**⁶⁵
- **Apache MXNet**⁶⁶
- **Aesara**⁶⁷

Aunque no se ha incluido en la lista anterior, es importante destacar *Keras*⁶⁸, una API construida sobre *Tensorflow* para facilitar a los desarrolladores la construcción de redes neuronales. *Keras*, en resumen, destaca por ser simple, es decir, por reducir la carga cognitiva; flexible, por habilitar una experimentación rápida; y potente, por proporcionar alta escalabilidad y rendimiento. En la siguiente sección se discute un caso práctico, utilizando *Tensorflow* como base para la ejecución de una red neuronal construida con la API de *Keras*.

3.8.3. Caso práctico: entrenando una red neuronal con *Tensorflow*

En esta sección se propone la resolución de un problema de clasificación utilizando la biblioteca de aprendizaje automático *Tensorflow*. En resumen, se implementa un modelo de red neuronal para clasificar imágenes de ropa. Este caso práctico consta de cuatro fases: el primer paso consiste en la preparación del conjunto de datos, el segundo en la construcción de la red neuronal y el tercero en el entrenamiento del modelo. La última fase del caso práctico simula un entorno de producción accediendo al modelo mediante un servidor en local.

Para el problema de clasificación se ha utilizado el conjunto de datos *fashion mnist*⁶⁹ proporcionado por Zalando, el cual está formado por artículos de ropa en formato imagen. En la figura 3.115 se muestra un ejemplo sobre cómo se presentan los datos. Nótese que cada clase del *dataset* se agrupa en tres filas en la imagen.

Preparación de los datos

Antes de preparar los datos para la resolución del problema es esencial explorar el conjunto y entender cómo están organizados y estructurados los datos. No obstante, primero veamos cómo importar el conjunto *fashion mnist*.

⁶² Enlace: <https://github.com/tensorflow/tensorflow>

⁶³ Enlace: <https://github.com/pytorch/pytorch>

⁶⁴ Enlace: <https://github.com/scikit-learn/scikit-learn>

⁶⁵ Enlace: <https://github.com/fastai/fastai>

⁶⁶ Enlace: <https://github.com/apache/incubator-mxnet>

⁶⁷ Enlace: <https://github.com/aesara-devs/aesara>

⁶⁸ Enlace: <https://keras.io/>

⁶⁹ Enlace: <https://github.com/zalandoresearch/fashion-mnist>

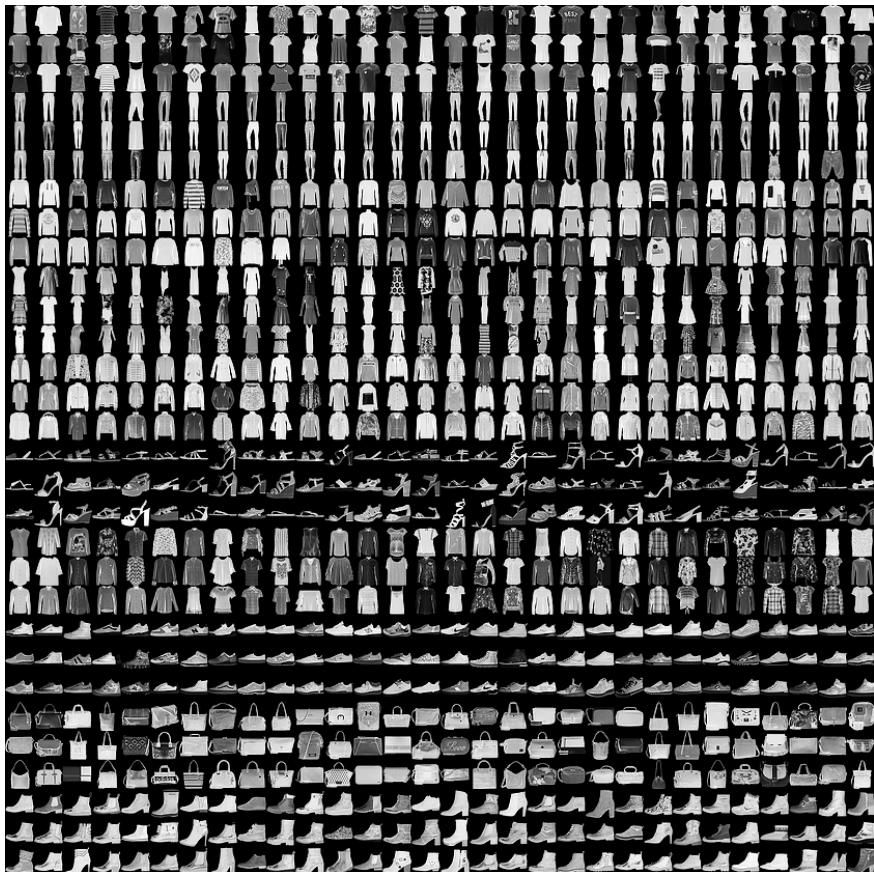


Figura 3.115: Artículos individuales de ropa del conjunto de datos de Zalando. Imagen: Recuperada de
🔗 Enlace: <https://github.com/zalandoresearch/fashion-mnist>.

Similar al caso práctico del bloque anterior, el conjunto de datos a utilizar se puede cargar desde *Tensorflow* sin necesidad de descargar el archivo de algún sitio web. Esta consideración, junto a los paquetes a importar, se reflejan en el listado 3.25. Como se puede apreciar, la función que recupera el conjunto *fashion mnist* retorna cuatro vectores:

- `train_images` es un vector con imágenes de ropa que forma parte del conjunto de entrenamiento.
- `train_labels` es un vector que contiene las etiquetas asociadas a las imágenes del vector anterior, el cual también forma parte del conjunto de entrenamiento.
- `test_images`. Similar a `train_images` con la diferencia que este vector forma parte del conjunto de pruebas.

- `test_labels`. Similar a `test_labels` con la diferencia que este vector forma parte del conjunto de pruebas.

Listado 3.25: Importación de paquetes y carga del dataset

```
1 #!/usr/bin/python3
2
3 import tensorflow as tf
4 from tensorflow import keras
5
6 import numpy as np
7 import matplotlib.pyplot as plt
8
9 (train_images, train_labels), (test_images,
10                                     test_labels) = keras.datasets.fashion_mnist.load_data()
```

Una vez importado el *dataset* es preciso realizar una exploración para entender cómo están organizados los datos (véase el listado 3.26). Concretamente, el conjunto de entrenamiento de imágenes consta de 60.000 muestras de 28x28 píxeles. Igualmente, el vector de etiquetas asociado al conjunto de entrenamiento también está formado por 60.000 valores. Por otro lado, el conjunto de pruebas de imágenes contiene 10.000 muestras con el mismo formato que el conjunto de entrenamiento. En cuanto al tamaño del vector de etiquetas, su longitud es también de 10.000. Adicionalmente, se muestran los valores únicos del vector de etiquetas de prueba, cuyas clases se encuentran asociadas en la tabla 3.5.

Listado 3.26: Exploración básica del conjunto de datos

```
1 # Output: (60000, 28, 28)
2 print(train_images.shape)
3
4 # Output: 60000
5 print(len(train_labels))
6
7 # Output: (10000, 28, 28)
8 print(test_images.shape)
9
10 # Output: 1000
11 print(len(test_labels))
12
13 # Output: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9], dtype=uint8)
14 print(np.unique(train_labels))
```

Los valores de los píxeles de cada imagen en el conjunto de datos son enteros sin signo en el rango [0-255] para el color blanco y negro. Existen diferentes alternativas para escalar los valores de los píxeles. Un buen punto de partida es normalizar estos valores en el rango [0, 1] con el objetivo de trabajar con valores pequeños. Esto implica dividir los valores de los píxeles por el valor máximo, es decir, 255. Esta consideración se representa en el listado 3.27. Alternativamente, podría utilizarse la técnica *one-hot-encoding*, la cual fue discutida en el bloque anterior.

Etiqueta	Clase
0	Camiseta
1	Pantalones
2	Jersey
3	Vestido
4	Abrigo
5	Sandalias
6	Camisa
7	Zapatillas
8	Bolsa
9	Botines

Tabla 3.5: Conjunto de clases del *dataset* de Zalando**Listado 3.27: Pre-procesamiento del conjunto de datos**

```

1 plt.figure()
2 plt.imshow(train_images[0])
3 plt.colorbar()
4 plt.grid(False)
5 plt.show()
6
7 train_images = train_images / 255.0
8 test_images = test_images / 255.0

```

Fase de construcción de la red neuronal

En el listado 3.28 se ofrece una versión del modelo utilizado para resolver el problema. Como se puede observar, el código es suficientemente explicativo para entender la estructura de la red. No obstante, se añaden a continuación una serie de aclaraciones del código, así como puntos donde se explican las decisiones de diseño tomadas respecto a la construcción de la red.

- La primera línea del listado utiliza la función `Sequential` de Keras para agrupar una pila lineal de capas. Como se puede apreciar, esta función recibe una lista que contiene la configuración de cada capa de la red. Alternativamente, se podría dejar el constructor en `Sequential` vacío para instanciar el modelo y usar `model.add` para añadir capas.
- El tamaño de la capa de entrada está determinado por el formato de las imágenes de la muestra. Como las imágenes son de 28x28 píxeles, cuyas intensidades se encuentran entre 0 y 1, el número de entradas es de $784 = 28 \times 28$. En este nivel se utiliza la función de activación `ReLU` y los pesos se inicializan mediante el esquema He, ambas buenas prácticas.

- Con la función `Flatten` se transforma el formato de las imágenes de un vector bidimensional (28x28 píxeles) a un vector unidimensional ($784=28 \times 28$ píxeles).
- Por otro lado, en los niveles intermedios se crea una capa oculta con 512 nodos para interpretar las características. Esta capa, del mismo modo que la anterior, utiliza la función de activación `ReLU` y el esquema de inicialización de pesos He.
- Dado que el problema es una clasificación multiclase, sabemos que necesitaremos una capa de salida con 10 nodos para asignar una imagen a uno de los diferentes tipos de artículos de ropa del conjunto de datos. Esto también requerirá el uso de una función de activación softmax. En términos generales, softmax es una buena opción para tareas de clasificación.
- La función `Dense` representa capas densamente conectadas o completamente conectadas.
- Por último, se utiliza una configuración normalmente usada en la práctica. Por un lado, se utiliza Adam como optimizador para ajustar los pesos y la tasa de aprendizaje. Por otro lado, se hace uso de entropía cruzada categórica como función de pérdida, principalmente porque el problema a resolver contiene más de dos clases. Nótese que la función en este caso tiene el nombre de *sparse categorical cross entropy*, cuya definición es la misma que en 3.12, pero con la diferencia que esta función opera con clases asociadas a enteros, en lugar de un vector de 1s y 0s como en *one-hot-encoding*. Por último, este ejemplo usa como métrica *accuracy* para controlar las imágenes que son correctamente clasificadas.

Listado 3.28: Construcción de la estructura de la red neuronal

```
1 model = keras.Sequential([
2     keras.layers.Flatten(input_shape=(28, 28)),
3     keras.layers.Dense(512, activation='relu', kernel_initializer='he_uniform'),
4     keras.layers.Dense(10, activation='softmax')
5 ])
6 model.summary()
7
8 model.compile(optimizer='adam',
9                 loss='sparse_categorical_crossentropy',
10                metrics=['accuracy'])
```

Fase de entrenamiento

A diferencia de los casos práctico anteriores, este ejemplo no necesita dividir el *dataset* en entrenamiento y pruebas, dado que el propio conjunto de datos proporciona dos subconjuntos específicamente para ello.

El entrenamiento se refleja en el listado 3.29 mediante la función `fit`, la cual trata de ajustar el modelo al conjunto de datos de entrenamiento. Más concretamente, la red neuronal aprenderá a partir de subconjuntos de 64 ejemplos en 10 iteraciones o *epochs*. Por otro lado, el conjunto de pruebas se utiliza para evaluar el modelo (línea $\Theta 3$) tras el entrenamiento, cuyo objetivo es poder estimar el rendimiento del modelo. Así, podemos establecer una comparación entre el error y precisión frente a los conjuntos de prueba y entrenamiento, lo que permite determinar cómo de bueno o malo es nuestro modelo.

Listado 3.29: Entrenamiento del modelo

```
1 model.fit(train_images, train_labels, batch_size=64, epochs=10)
2
3 test_loss, test_acc = model.evaluate(test_images, test_labels, verbose=0)
4
5 print('\nTest loss:', test_loss)
6 print('\nTest accuracy:', test_acc)
```

Al ejecutar el trozo de código anterior, el modelo aprende a medida que el error decremente durante el entrenamiento. Al final, la red alcanza una precisión de 0.91 (o 91 %) sobre el conjunto de entrenamiento y 0.88 (o 88 %) sobre el conjunto de pruebas. Como se puede observar, este último valor es un poco menor que la precisión sobre el conjunto de entrenamiento. Esta diferencia se debe a *overfitting* (sobreajuste). Esto sucede cuando un modelo de aprendizaje automático tiene un rendimiento peor sobre un set de datos nuevo, el cual nunca antes ha visto comparado con el de entrenamiento.



Mejorando la red neuronal. Aunque a priori la precisión del modelo puede parecer buena, se ha detectado que existe un ligero sobreajuste en la red. ¿Cómo sería posible mejorar su arquitectura? Podríamos pensar que la mejor opción sería añadir más capas o más neuronas, pero esto resulta erróneo. El problema a resolver consiste en una clasificación de imágenes en blanco y negro. Este tipo de problemas suele resolverse mediante **redes convolucionales**, cuyas capas corresponden a filtros que son porciones pequeñas de la imagen original. Estos filtros se adaptan a diferentes lugares de la imagen en un proceso que corresponde a la operación matemática de convolución, de ahí el término de la red. No obstante, este enfoque está fuera del alcance de este bloque. A modo de referencia, se invita al lector a consultar [Ger19] para profundizar en este tipo de redes con *Tensorflow*.

Uso de la red neuronal

Una vez completado el entrenamiento del modelo es preciso probar cómo de correctas son sus predicciones contra un nuevo conjunto de datos, por ejemplo, el destinado a pruebas. El listado 3.30 refleja esta consideración, realizando una predicción completa para cada imagen del *dataset*. Concretamente, en la línea $\Theta 1$ se

obtienen las predicciones, es decir, una lista de listas cuyos elementos representan un nivel de confianza sobre cada clase. Por lo tanto, a la hora de comprobar una predicción es necesario recuperar el elemento más grande de la lista (véase línea ④). Por último, y para corroborar la predicción, se utiliza el conjunto de etiquetas, cuyos elementos corresponden a la clase asociada con los datos de entrada del conjunto de pruebas. En este ejemplo, el modelo acierta en la segunda predicción.

Listado 3.30: Predicción del modelo

```
1 predictions = model.predict(test_images)
2
3 # Output: 2
4 np.argmax(predictions[1])
5
6 # Output: 2
7 test_labels[1]
```

Despliegue del modelo

Keras ofrece la posibilidad en su API de persistir el modelo en disco. Esta opción permite volver a crear el modelo a partir de un conjunto de archivos alojados en un directorio sin necesidad de tener acceso al código que lo creó.

El directorio aloja en archivos la siguiente información del modelo:

- Arquitectura.
- Pesos, los cuales se ajustaron durante el entrenamiento.
- Configuración (solo si la función `compile` se utilizó).
- Optimizador y estado, solo si existe (esto permite reiniciar el entrenamiento donde se dejó).

El listado 3.31 refleja el código encargado de alojar toda la configuración del modelo en disco. En esencia, la función `save` lo que hace es guardar el modelo por completo en un directorio, cuyo nombre es pasado como parámetro a la función.

Listado 3.31: Serialización del modelo

```
1 model.save('model/fashion_mnist')
```

El hecho de persistir un modelo habilita consumir IA en servicios web, alojando un modelo de aprendizaje automático en un servidor desplegado en la nube. Este ejemplo simula esta consideración, desarrollando simplemente un servidor en *Flask* que recibe una imagen y retorna como resultado la clase a la que pertenece la prenda de ropa.

Podríamos pensar que el modelo previamente entrenado es lo suficientemente robusto para poder realizar predicciones con cualquier imagen descargada en la web. Sin embargo, esto no es del todo así. Cada imagen del conjunto *fashion mnist* ha sido convertida a escala de grises. También se han segmentado, es decir, todos los píxeles del fondo son negros, y aquellos en primer plano tienen una intensidad gris. Aparte, se han redimensionado a 28x28 píxeles. Esto quiere decir que si se quisiera realizar predicciones sobre imágenes de moda y ropa reales habría que preprocessar los datos de la misma manera que el conjunto de datos, algo que queda fuera del alcance de este ejemplo.

En vista a la problemática discutida, este ejemplo hace uso de un repositorio⁷⁰ de GitHub, el cual contiene un archivo .zip con las imágenes del conjunto de datos *fashion mnist* en formato .png. Recupera algunas imágenes del *dataset* de pruebas y guárdalas dentro del directorio donde este ejemplo está siendo desarrollado.

En este punto, ya es posible introducir la implementación relativa al servidor, cuyo código se muestra en el listado 3.32. En esencia, este programa define un *endpoint* que contiene la lógica necesaria para clasificar artículos de ropa dada una imagen. Los siguientes puntos destacan aquellas partes del código más notables.

- En la línea **⑩ 11** se carga el modelo previamente entrenado que se encuentra en la ruta que se proporciona por parámetro. `new_model` consiste en una copia del modelo antes de crear una copia de su estado.
- La función `reshape_image` en la línea **⑩ 13** carga una imagen con el formato que espera el modelo, es decir, un vector unidimensional cuyos valores están en una escala [0, 1]. El parámetro `key` es el nombre de la imagen contenido, en este caso, en el directorio `img`.
- En la línea **⑩ 20** se define el *endpoint* que recibe una imagen de un artículo de ropa y retorna una predicción, es decir, un valor entre 0 a 9 que corresponde con el tipo de prenda.
- En la línea **⑩ 26** se carga en memoria la imagen que espera recibir como entrada el modelo en la línea **⑩ 27**. Por último, se pinta por consola la clase de prenda de ropa, buscando en la lista de predicciones el elemento con mayor valor.

Listado 3.32: Servidor en Flask que predice a qué clase corresponde una imagen

```

1 #!/usr/bin/python3
2
3 import cv2
4 import numpy as np
5 import tensorflow as tf
6 from flask import Flask, request, Response, abort
7
8 application = Flask(__name__)
9

```

⁷⁰ Enlace: https://github.com/DeepLenin/fashion-mnist_png

```
10 # Load fashion mnist model
11 new_model = tf.keras.models.load_model('model/fashion_mnist')
12
13 def reshape_img(key):
14     img = cv2.imread(f'img/{key}', 0)
15     img = img / 255.0
16     img = (np.expand_dims(img, 0))
17     return img
18
19 # /predict endpoint
20 @application.route('/predict', methods=['POST'])
21 def predict():
22     key = request.get_json()['key']
23     if key is None:
24         abort(400)
25
26     img = reshape_img(key)
27     prediction = new_model.predict(img)
28     print(prediction.argmax())
29     return Response(status=200)
30
31 # Run the app
32 if __name__ == "__main__":
33     application.debug = True
34     application.run() # Running on http://127.0.0.1:5000/
```

Una vez el servidor se encuentra en ejecución, una forma sencilla de activar el *endpoint* es simulando la creación de una petición con el comando `curl`. La respuesta como consecuencia del mensaje será un número entre 0 y 9, es decir, la clase correspondiente al artículo de ropa pasado como imagen en la petición.

```
$ curl localhost:5000/predict
-H 'Content-Type: application/json'
-X POST -d '{
    "key": "<image-name>"
}'
```

3.8.4. Resumen

En este bloque se ha discutido una amplia gama de técnicas para mejorar el rendimiento y aprendizaje de una red neuronal. A continuación, se resumen los aspectos y cuestiones más importantes, así como una serie de recomendaciones que suelen aplicarse en la mayoría de configuraciones de redes neuronales.

- El criterio para realizar la **inicialización de pesos** de una red afecta considerablemente a su aprendizaje. Existen dos heurísticas en base a la función de activación aplicada en las capas: inicialización de Xavier y He. La segunda opción suele usarse por defecto, dado que la función de activación que normalmente se aplica en las capas intermedias es ReLU o ELU.

- Existe una variedad importante de **funciones de pérdida**, aunque destacan principalmente tres: entropía cruzada binaria para problemas de clasificación de una o dos clases; entropía cruzada categórica para problemas multiclas; y error cuadrático medio para problemas de regresión.
- Una red neuronal es precisa cuando los valores de sus pesos hacen que la función de pérdida converja. Esto se consigue mediante algoritmos de **optimización**. Un ejemplo es el descenso del gradiente, el algoritmo más popular, aunque también más básico. Por otro lado, destacan también otros algoritmos no solo por actualizar los pesos, sino por adaptar dinámicamente una **tasa de aprendizaje**. Los más populares son RMSProp, Adagrad y Adam. En la práctica, Adam suele usarse como algoritmo de optimización por defecto debido a los buenos resultados que arroja.
- Una variante del algoritmo de optimización del descenso del gradiente es el descenso del gradiente estocástico. Esta configuración requiere un **esquema** para modificar, a lo largo del entrenamiento, la constante o tasa de aprendizaje dado un valor inicial. Destacan varias heurísticas: disminución constante a trozos, disminución exponencial y disminución inversa del tiempo. El esquema de disminución exponencial suele funcionar adecuadamente en casos comunes: si la convergencia es demasiado lenta, o por el contrario la convergencia es rápida, pero la precisión de la red no es óptima.
- Cuando la red realiza predicciones distintas con conjuntos contrarios al de entrenamiento se utilizan **técnicas de regularización** para reducir el sobreajuste. Entre ellas destacan: *early stopping*, *dropout*, l1 & l2, y *data augmentation*. Por lo general, la técnica con mejores resultados en configuraciones por defecto es *dropout*.
- La implementación de una red neuronal desde cero, por lo general, son imprecisas y poco escalables ante cambios en los casos de uso. Como consecuencia, existen **módulos predefinidos** que permiten crear, entrenar, y hasta desplegar, modelos de redes neuronales de un modo sencillo y rápido. Además, estos módulos están aprobados por la comunidad, por lo que son totalmente precisos y efectivos.
- Este bloque termina con un **caso práctico** que ejemplifica la utilización de las heurísticas discutidas con el módulo *Tensorflow*.

3.9. Herramientas de generación de código para crear IA

Como colofón final a este capítulo titulado «Entornos de modelado de IA», este bloque introduce herramientas capaces de generar internamente código a través de una serie de configuraciones para crear software con comportamiento inteligente. Obviamente, esta tecnología aún no está del todo madura, aunque tiene un objetivo totalmente claro: **independencia tecnológica**. Aunque esta tendencia se sigue en diversos ámbitos, también se adopta en el campo de la IA. A raíz de esto, cada vez son más las herramientas para modelar IA y así **democratizar** su uso, es decir, poner la IA al acceso de cualquiera.

Antes de recopilar algunas de las herramientas de generación de IA sin código, es preciso primero poner en contexto la importancia del desarrollo de componentes inteligentes mediante programación sin código.

3.9.1. Visión general

El código es la columna vertebral en la mayoría de programas y aplicaciones software. En esencia, cada línea de código sirve como instrucción, un mecanismo lógico, paso a paso, para que los ordenadores, servidores y otras máquinas realicen una acción.

La programación es una tarea completamente compleja en gran parte por la abstracción que se requiere para modelar aspectos reales y plasmarlos en líneas de código. En los comienzos de la programación se utilizó el lenguaje ensamblador, que es lo más parecido a las instrucciones del código máquina (es decir, 0 y 1), y evolucionó a lenguajes de alto nivel, como Java, Python, C y JavaScript, entre otros. Esta evolución ha consistido en añadir capas de abstracción para ocultar las complejidades que se esconden tras el código máquina, facilitando la programación a los desarrolladores de software.

Aún con la existencia de estos avances, la programación sigue siendo una herramienta poco accesible, que se une a la dificultad de crear software inteligente, principalmente por la heterogeneidad de conocimientos necesarios, así como la tecnología subyacente.

Relativamente reciente existe un concepto denominado **programación sin código** que permite la construcción de software sin necesidad de escribir una sola instrucción en lenguaje de bajo o alto nivel. En esencia, este enfoque puede considerarse como una forma de **programación visual**. En lugar de entornos de desarrollo basados en texto, los usuarios manipulan los elementos del código mediante interfaces de usuario. Un ejemplo popular es el lenguaje de programación Scratch del MIT Media Lab, que utiliza bloques de programación gráfica para enseñar a niños y adultos a programar.

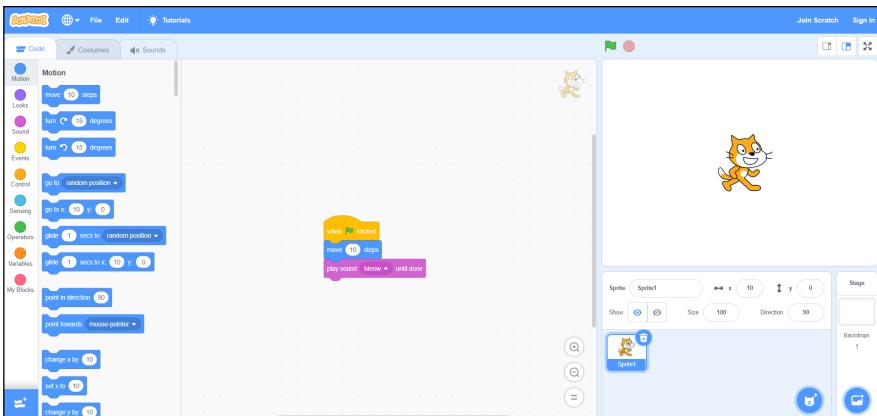


Figura 3.116: Interfaz de la herramienta Scratch.

Este enfoque en poco tiempo se ha extendido al campo de la IA, apostando por esta tecnología los principales gigantes tecnológicos, como Microsoft con Azure Machine Learning Studio. Este tipo de implementaciones reduce la complejidad de crear IA, aunque sigue siendo necesario poseer conocimientos profundos en la materia.

En poco tiempo ha surgido una vertiente con la misma idea, aunque su principal característica es que pone accesible la IA a casi a cualquier persona. Esta aproximación requiere de conocimientos mínimos sobre aprendizaje automático, ofreciendo la posibilidad de entrenar modelos mediante un **proceso guiado**. Sin embargo, esto no quiere decir que expertos no sean usuarios objetivos de esta tendencia, ya que un conocimiento profundo proporciona un mayor control en la resolución del problema.

3.9.2. Beneficios del enfoque “IA sin código”

En anteriores secciones se discutió que la computación en la nube es un catalizador eficaz para la IA y que esta tecnología impulsa su democratización. En esencia esto es cierto, puesto que se pone al alcance de cualquiera recursos y algoritmos de una sofisticación técnica incuestionable. No obstante, la IA sigue siendo solo accesible por individuos con conocimientos profundos en desarrollo software y aprendizaje automático.

Este nivel de democratización parece incrementar con el enfoque “sin código”, dado que habilita la implementación e implantación de la IA en su totalidad. Este tipo de herramientas son las que más se acercan al ideal de **personas sin conocimientos técnicos avanzados**.

Las herramientas de generación de comportamiento inteligente sin código aprovechan tres componentes: tiempo, valor y conocimiento. Así, tratan de conseguir que la IA sea **menos intimidante y más comprensible** para las personas sin conocimientos técnicos, o que carecen de tiempo o recursos para crear estos sistemas desde cero.

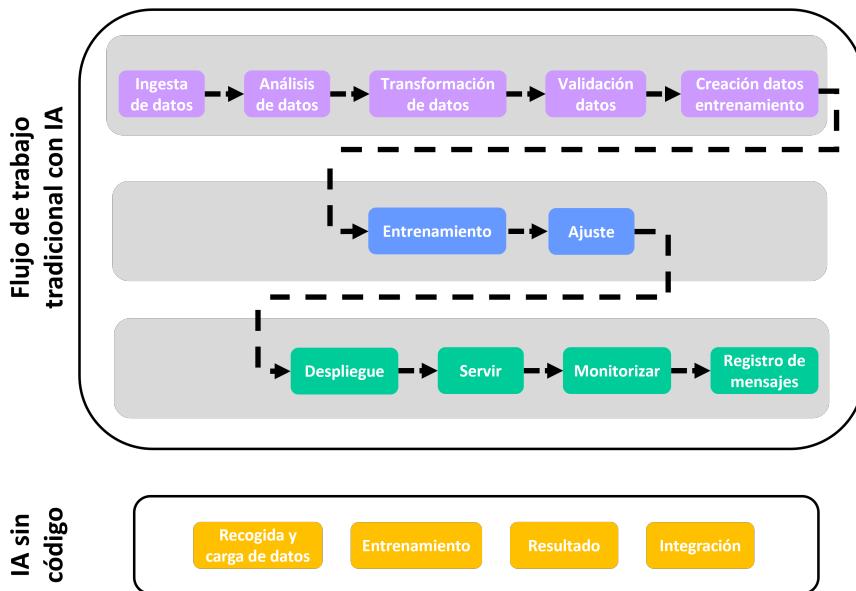


Figura 3.117: Flujo de trabajo tradicional en IA VS desarrollo de IA sin código.

Además de esto, la IA sin código tiene algunas ventajas adicionales:

- **Accesibilidad.** La IA sin código permite a las organizaciones ser más competitivas al tener acceso, de un modo relativamente sencillo, a una tecnología hasta ahora solo utilizada por gigantes tecnológicos. Además, la inversión se reduce considerablemente y se mitigan los obstáculos para la adopción de la IA en las pequeñas y medianas empresas.
- **Usabilidad.** La generación de IA sin código permite, de forma sencilla, encontrar una solución a un problema. Estas herramientas se han creado pensando en usuarios sin conocimientos previos en desarrollo software o IA, aunque siempre resulta importante contar, al menos, con una idea previa.
- **Rapidez.** La IA sin código permite a los usuarios iterar rápidamente a través de toda la cadena de valor del aprendizaje automático. Esto permite experimentar de forma más rápida y ver así lo que se puede hacer utilizando los datos. En definitiva, un ahorro en tiempo que es valioso para cualquier organización.

- **Calidad.** Las herramientas sin código están pensadas para reducir al máximo el número de pasos que se requieren para completar una tarea. Esto supone que el producto prime por un proceso automático, más que por uno manual. En general, la herramienta puede generar un resultado especialmente poco óptimo, cuyo solución normalmente no es directa, ya que en la mayoría de ocasiones no es posible acceder internamente al proceso y modificarlo. Para mitigar este riesgo, algunas herramientas incorporan la opción de revisión y piden al usuario un trato manual. Esta combinación reduce el error al mantener un punto de verificación por parte del usuario.
- **Escalabilidad.** Esta característica es inherente a la IA, dado que una tarea no se diseña para uno o cien usuarios, sino que un modelo proporciona información en base a los ejemplos que ha recibido para actuar en consecuencia. En definitiva, los límites los pone el usuario, no la IA ni la herramienta.

3.9.3. Herramientas de generación de IA “sin código”

En este apartado se muestran un conjunto de herramientas que automáticamente generan código para crear software con comportamiento inteligente. Resulta preciso destacar que el objetivo no reside en mostrar el estado del arte o profundizar en la tecnología. Al contrario, se pretende reflejar el estado actual de la tecnología con algunas de las herramientas más relevantes e innovadoras.

- **Datature**⁷¹. Es una aplicación web destinada a la creación de modelos de visión por computador. No requiere código aunque sí conocimientos mínimos en IA. Permite gestionar conjuntos de datos, etiquetar, entrenar y desplegar modelos en la nube.
- **MakeML**⁷². Es una aplicación exclusivamente para el sistema operativo Mac OS. MakeML se caracteriza por ser fácil de usar para los desarrolladores de iOS que quieran utilizar aprendizaje automático en sus aplicaciones. No se necesitan conocimientos en Python ni experiencia en IA. MakeML cuenta actualmente con dos modelos de entrenamiento: Detección de Objetos y segmentación de modelos.
- **Teachable Machine**⁷³. Es una plataforma de IA de Google que no requiere conocimientos de programación. Permite entrenar rápidamente modelos para reconocer imágenes, sonidos y poses directamente desde el navegador. Además, Teachable Machine es capaz de exportar el modelo con formato *Tensorflow* u otros diferentes para usarlos en otras plataformas, como Coral, Arduino y muchas más.

⁷¹ Enlace: <https://datature.io/>

⁷² Enlace: <https://makeml.app/>

⁷³ Enlace: <https://teachablemachine.withgoogle.com/>

- **Lobe⁷⁴**. Es una herramienta para MacOS y Windows, gratuita, capaz de entrenar modelos de *deep learning* que puedan ser fácilmente integrables en una aplicación. Lobe requiere de una muestra de ejemplos de lo que quieres que aprenda, y automáticamente entrena un modelo de aprendizaje automático personalizado. No se requiere código ni experiencia previa en IA. Todavía están en fase beta y ha sido recientemente comparada por Microsoft.

3.9.4. Ejemplo: detección de rostros con mascarillas usando Lobe

En esta sección se plantea el diseño y desarrollo de un modelo capaz de identificar si una persona lleva o no puesta una de las mascarillas utilizadas para combatir la pandemia de la COVID-19. En este ejemplo se utiliza la herramienta Lobe con objeto de mostrar cómo es sencillo crear comportamiento inteligente sin escribir una sola línea de código. El ejemplo se aborda desde dos enfoques: (i) a partir de un conjunto de datos y (ii) mediante un entrenamiento en tiempo real. Finalmente, el modelo se exporta como un servidor REST, es decir, un modelo en *Tensorflow* alojado en un servidor *Flask*.

Configuración

Antes de comenzar con el ejemplo es preciso instalar y configurar la aplicación de Lobe en nuestro equipo, ya que la herramienta aún no tiene un acceso vía web. En  Enlace: <https://www.lobel.ai/> se encuentra la página oficial desde donde proceder a su descarga. El proceso de instalación es el mismo que el de cualquier otra aplicación. Tras ello, su aspecto en Windows debería ser igual al de la figura 3.118.

Una vez dentro de la herramienta, crea un nuevo proyecto y añade un nombre representativo, por ejemplo, *Face Mask*.

Entrenamiento del modelo a partir de un *dataset*

Lobe proporciona tres métodos para construir un conjunto de datos con el que entrenar el modelo de IA: (i) importar una serie de imágenes desde una ubicación en local, (ii) desde un directorio estructurado o (iii) incluso desde la propia cámara del PC. En este caso elegiremos la segunda opción (véase figura 3.119), utilizando el conjunto de datos de  Enlace: <https://www.kaggle.com/dhruvmak/face-mask-detection/>.

La opción elegida significa que Lobe espera un directorio cuyas subcarpetas dividen el conjunto de datos en diferentes clases. Por ejemplo, el *dataset* utilizado contiene imágenes de personas con y sin máscaras, las cuales están correctamente agrupadas en dos carpetas nombradas como *with mask* y *without mask*.

⁷⁴ Enlace: <https://www.lobel.ai/>

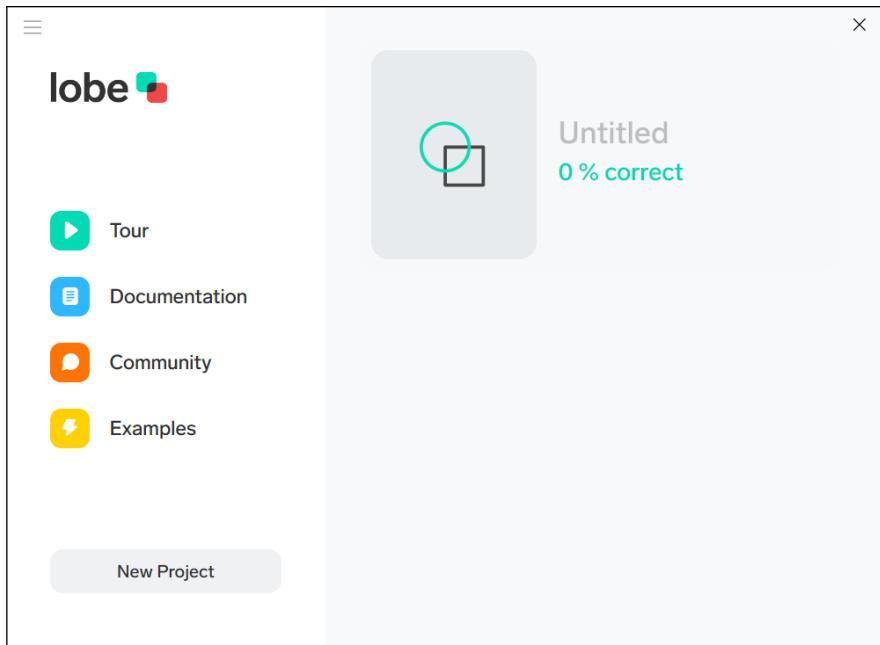


Figura 3.118: Pantalla de inicio de Lobe en Windows.

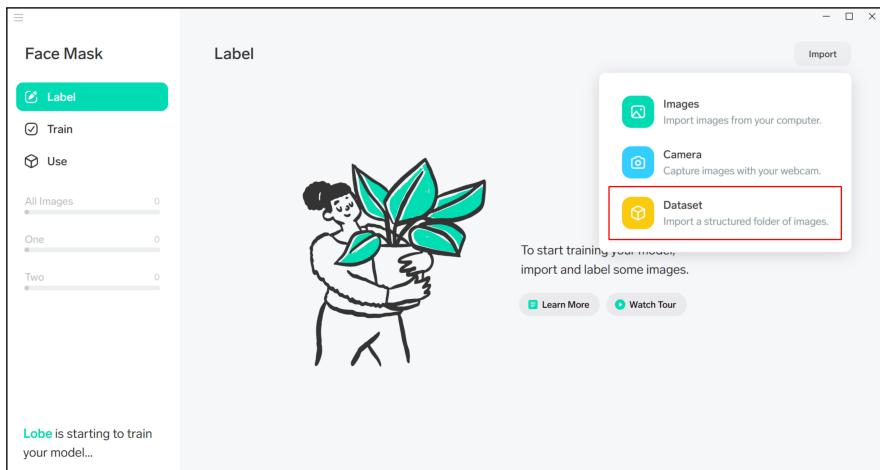


Figura 3.119: Importación del conjunto de datos.

Una vez importado el conjunto de datos, Lobe comienza a entrenar el modelo automáticamente, utilizando un 80 % para el aprendizaje del modelo y un 20 % restante para pruebas.

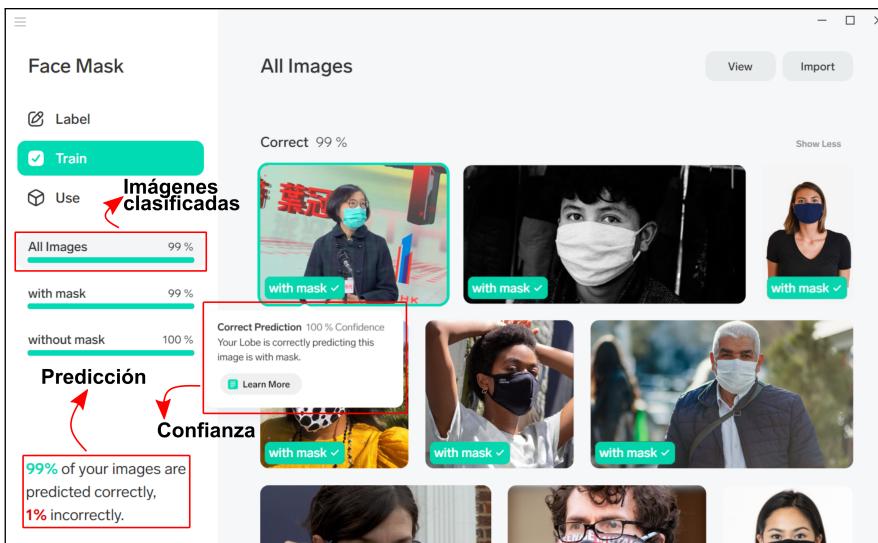


Figura 3.120: Información sobre el entrenamiento del modelo.

Variedad en los datos. Aunque Lobe funciona perfectamente bien, los datos deben ser lo suficientemente heterogéneos para que el modelo realice predicciones correctas. Es decir, Lobe requiere de imágenes desde distintos ángulos, orientaciones, fondos y condiciones de luz, entre otros, para evitar, por ejemplo, *overfitting*.

Lobe entrena el modelo en segundo plano y avisa con un sonido cuando el proceso termina. Para este conjunto de datos, que contiene 440 imágenes (220 con mascarilla y otras 220 sin mascarilla), el entrenamiento tarda alrededor de 2 minutos, aunque esto también varía en función del hardware del equipo. En la pestaña de Entrenamiento o *Train* se muestran las imágenes etiquetadas por Lobe, indicando con verde si la predicción ha sido correcta o rojo en caso contrario. Además, es posible conocer tanto la precisión total del modelo como la confianza sobre una imagen. Estas consideraciones se reflejan en la figura 3.120.

Existen dos opciones desde la pestaña Uso o *Use* para probar el modelo: (i) importando una imagen o (ii) capturándolas desde la propia cámara del PC. Cual sea la opción que elijamos, la herramienta utilizará el modelo entrenado y observaremos cómo de bien o mal realiza predicciones. En la figura 3.121 se muestra el resultado que el modelo arroja tras importar una imagen de una persona sin mascarilla.

Por otro lado, si intentamos confundir al modelo con la imagen de una persona tapándose la boca y nariz (véase figura 3.122), se observa que la predicción es incorrecta, puesto que el modelo se confunde cuando la boca y nariz de la persona en la imagen no aparece. Obviamente, el resultado es de esperar, dado que si ins-

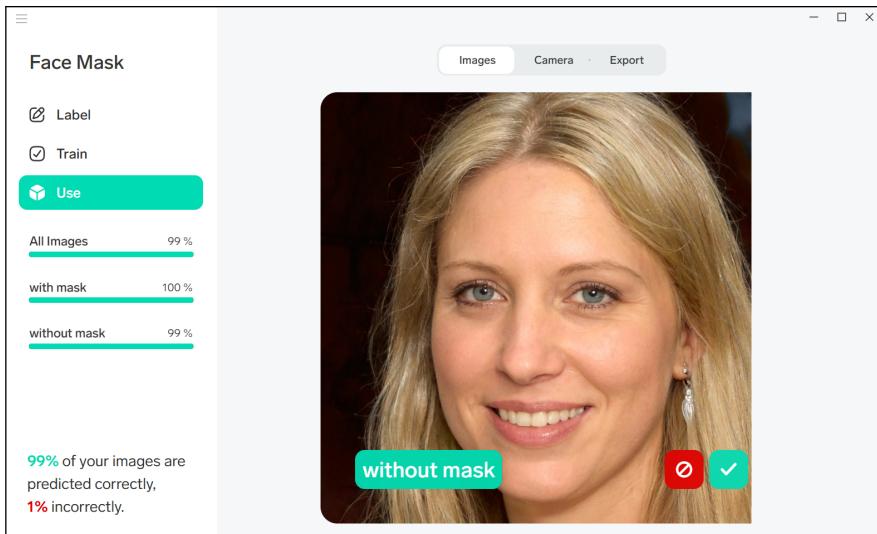


Figura 3.121: Predicción del modelo. La imagen utilizada ha sido recuperada de Enlace: <https://thispersondoesnotexist.com/>

peccionamos las imágenes etiquetadas como “without mask” observamos que, en general, todas ellas siguen el mismo patrón, es decir, personas mirando de frente a la cámara. Por lo tanto, se considera que el modelo está sobreajustado (overfitting), ya que actúa de forma incorrecta ante datos diferentes a los de entrenamiento.

Específicamente, el modelo necesita aprender de un conjunto cuyos datos sean lo más variados posibles. En este caso, imágenes de personas que claramente identifiquen si llevan o no mascarilla. El error anterior se puede solventar fácilmente desde la interfaz de Lobe. En la figura 3.122 se muestra que es posible modificar la predicción ofreciendo *feedback*, es decir, haciendo clic con el botón izquierdo en la marca de verificación roja. Esto añade la imagen a su conjunto de datos correspondiente (*without mask*). Después, Lobe entrena automáticamente el modelo con esta nueva imagen. Como resultado, el modelo obtiene una precisión del 100 %.

Entrenamiento del modelo en tiempo real

En la mayoría de casos, y siempre que el problema no sea excesivamente específico, existe un *dataset* que permite entrenar un modelo. En caso contrario, los datos se tienen que recopilar o crear, cuya tarea no es en absoluto rápida y sencilla. Lobe ofrece una opción que puede ser útil en casos de uso un tanto “caseros” utilizando la cámara del PC. Para el ejemplo de esta sección la opción de Lobe puede servir para mostrar cómo crear un *dataset* desde cero.

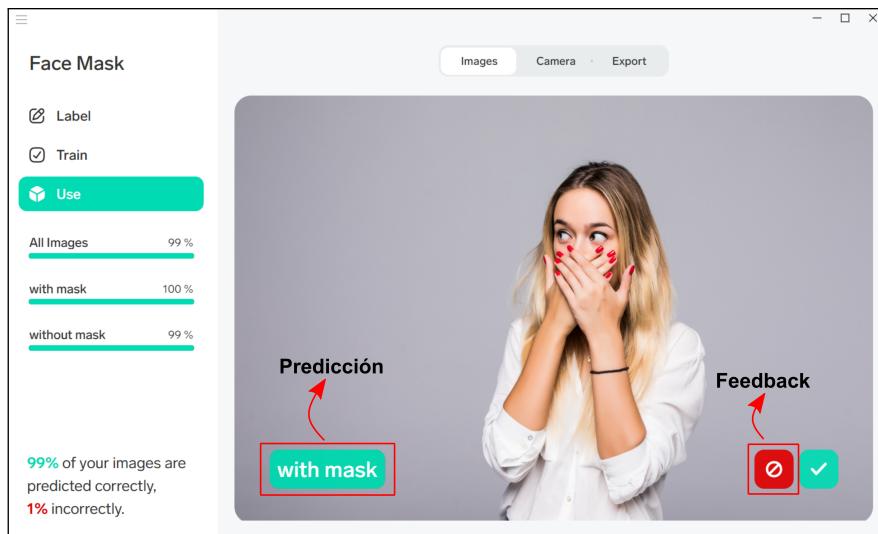


Figura 3.122: Predicción errónea del modelo. Imagen de persona: Freepick.com.

En primer lugar, será conveniente crear otro proyecto y no perder el trabajo del ejemplo anterior. A diferencia de la opción seleccionada en la figura 3.119, escogemos *Camera* para capturar nosotros mismos las imágenes que utilizaremos durante el entrenamiento.

La figura 3.123 muestra los elementos con los que crear los datos. Lobe sugiere que al menos se creen 5 imágenes por clase, aunque es un número muy reducido para conseguir un modelo relativamente eficaz. Con 50 imágenes por clase puede ser suficiente, haciendo un total de 100 datos para el entrenamiento (una clase para imágenes con mascarilla y otra sin mascarilla).

Se recomienda que los datos sean totalmente distintos entre sí. Es decir, las imágenes deberían captar diferentes ángulos, perspectivas, fondos, rostros de personas, etc. En este ejemplo, algunas de estas recomendaciones no son viables, como utilizar diferentes rostros de personas, complicando la precisión del modelo. Sin embargo, sí es posible capturar nuestro rostro de perfil o de frente, así como diferentes perspectivas, tratando de cubrir el mayor número de ángulos. El objetivo es que el modelo funcione siempre que la cámara capture nuestra cara.

Si al probar el modelo el comportamiento todavía no es el deseado, es preciso proporcionar *feedback* etiquetando adecuadamente la imagen. Además, resulta importante explotar sus potenciales debilidades, por ejemplo, tapándonos la boca y nariz con la mano. La idea consiste en “engaños” al modelo y observar qué tal se comporta. Si el resultado no es de nuevo el esperado, sería conveniente añadir la imagen a su conjunto correspondiente.

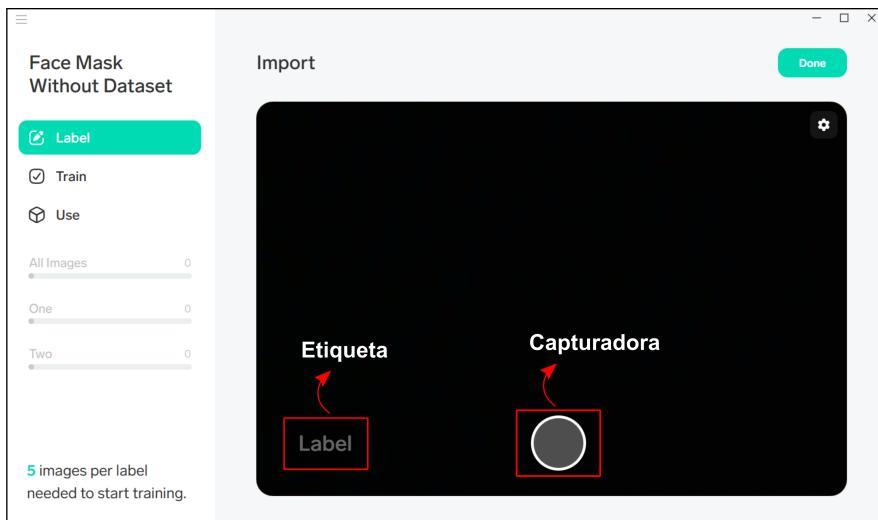


Figura 3.123: Creación y clasificación de datos.

Despliegue del modelo

Con objeto de ver un caso de uso real con el modelo generado, imagina un sistema de cámaras cuyo tarea es identificar personas sin mascarilla en un recinto cerrado, como un centro comercial, restaurante o discoteca, entre otros. Un servidor procesa las imágenes de las cámaras y avisa a los responsables del lugar si alguna persona no lleva mascarilla para evitar contagios entre los asistentes.

Possiblemente el modelo generado a partir del conjunto de datos sea el que mejor precisión tenga. Este ejemplo exporta el modelo en formato *Tensorflow* para alojarlo en un servidor *Flask* (véase figura 3.124). No obstante, Lobe ofrece más posibilidades, permitiendo generar el modelo para consumirlo en una aplicación móvil, web o incluso en aplicaciones universales con ONNX.

La opción de exportar el modelo con formato *Tensorflow* en Python genera la misma salida que el comando `save` de *Keras*, cuya estructura de directorios y archivos se plasma en la siguiente salida.

```
$ tree -L 1
.
└── example
    ├── variables
    └── labels.txt
    └── saved_model.pb
    └── signature.json

2 directories, 3 files
```

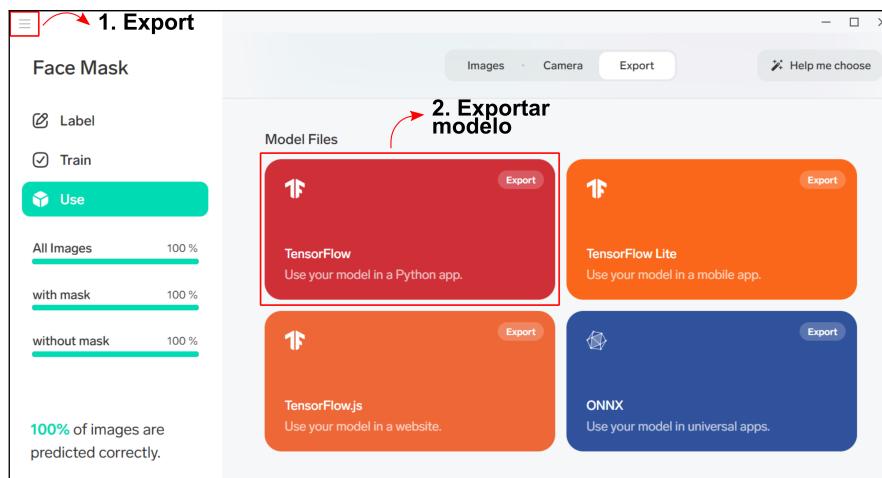


Figura 3.124: Exportación del modelo.

Lobe cuenta con una plantilla que facilita y agiliza enormemente la creación de una API REST. Esta opción se ha elegido por razones obvias del ejemplo, aunque existen más plantillas para diferentes casos de uso, por ejemplo, aplicaciones Android, iOS o web. El siguiente comando permite clonar un proyecto con los archivos necesarios para crear un servidor en *Flask*.

```
$ git clone https://github.com/lobe/flask-server.git
```

El proyecto contiene la siguiente estructura de carpetas y archivos:

```
$ tree -L 1
.
└── LICENSE
└── README.md
└── app.py
└── assets
└── model
└── requirements.txt
└── swagger
└── testing.py
└── tf_model_helper.py

3 directories, 6 files
```

El proyecto clonado está principalmente diseñado para usar el modelo exportado con Lobe. Simplemente es necesario mover los archivos `saved_model.pb` y `signature.json`, junto a la carpeta `variables` al directorio `/model` del proyecto *Flask*.

El servidor requiere como petición una imagen en base 64 y genera como resultado un array de predicciones. El código del servidor se define en `app.py` y el código para utilizar el modelo, incluyendo el preprocesamiento de la imagen y el formato de salida para una predicción, se encuentra en `tf_model_helper.py`.

```
{  
    "image": "<base64 image>"  
}
```

El listado 3.33 muestra un fragmento de código del archivo `testing.py` que realiza una petición POST a la URL `http://localhost:5000/predict`. En la línea $\Theta 5$ se añade la ruta de la imagen, la cual se codifica después en base 64 en la línea $\Theta 9$. Esta consideración es importante para asegurar que el dato permanece intacto, sin modificación, durante el envío.

Listado 3.33: Fragmento de código del archivo `testing.py`

```
1 import base64  
2 import requests  
3  
4 # Save string of image file path below  
5 img_filepath = "<path/to/image.jpg>"  
6  
7 # Create base64 encoded string  
8 with open(img_filepath, "rb") as f:  
9     image_string = base64.b64encode(f.read()).decode("utf-8")  
10  
11 # Rest of code...
```

La siguiente salida muestra el resultado como consecuencia de utilizar la imagen en la predicción de la figura 3.121.

```
// Terminal 1  
$ python3 app.py  
  
// Terminal 2  
$ python3 testing.py  
  
predicted label: without mask  
confidence: 1.0
```

3.9.5. Oportunidades de la generación de “IA sin código”

Actualmente el software inteligente es uno de los productos intangibles que más resultados financieros positivos produce en la industria. Esto conlleva una demanda creciente de personal cualificado con un perfil técnico que cada vez es más difícil de encontrar, ya que este campo se encuentra en evolución constante. La tendencia sin código es una de las soluciones a este problema de **oferta y demanda**. Hay una clara necesidad en el mercado por personas capaces de crear software inteligente,

dado que el código subyacente es muy valioso. Sin embargo, la programación, y concretamente en IA, es una habilidad compleja que requiere de años de aprendizaje. Las herramientas sin código, capaces de crear comportamientos inteligentes, están surgiendo porque reducen, por un lado, la complejidad asociada a la tarea de programar y, por otro, el hecho de resolver problemas de IA.

Además de una mínima curva de aprendizaje, las herramientas capaces de generar IA sin código permiten un desarrollo más rápido de las aplicaciones, lo que podría suponer una **reducción de costes** para las organizaciones. También habilitan una **innovación rápida** que se traduce en la materialización de la visión o idea de un producto. Incluso este enfoque puede servir como punto inicial en la fase de implementación, partiendo de un software cuyo propósito cumple con el objetivo establecido por la organización para resolver un problema concreto.

Quizá la ventaja más importante de la generación de IA sin código es hacer más **accesible el desarrollo de software inteligente**. En esencia, este tipo de herramientas potencian la democratización de la IA, es decir, consiguen que la tecnología sea lo menos intimidante y más accesible posible para personas sin conocimientos técnicos. La idea es que cada vez más personas sean capaces de crear IA, ya que esta tecnología es una parte central en nuestras vidas.

3.9.6. Resumen

En este apartado se resumen los puntos claves discutidos en este bloque titulado Herramientas de generación de código para crear IA.

- El enfoque **programación sin código** en el campo de la IA pretende poner al alcance de cualquiera, con o sin conocimientos técnicos, los medios para crear software con comportamiento inteligente. De momento el objetivo inicial es demasiado idílico, por lo que es necesario contar, al menos, con conocimientos mínimos en aprendizaje automático.
- Al igual que la computación en la nube, la programación sin código presenta varios **beneficios** al desarrollo software y, en concreto, a la IA: esta tecnología es *accesible* tanto para individuos como empresas; la consideración anterior se debe a que las herramientas de generación de IA sin código son *usables*; la IA sin código permite experimentar *rápidamente*; este enfoque ofrece *calidad*, en el sentido de que asegura resultados de una sofisticación considerable. De lo contrario, permite ajustes manuales para solventar pequeños errores en el entrenamiento; todas las soluciones son totalmente *escalables*.
- Las **herramientas de IA sin código** son soluciones que ofrecen un **enfoque de todo integrado**. Esto significa que se cubren todas las fases de un desarrollo software en IA, desde la generación del modelo hasta la puesta en producción con una solución en la nube.

- Esta sección ofrece un **ejemplo** sobre cómo implementar un modelo de IA sin escribir una línea de código. Además, se muestra cómo alojar, casi de forma automática, el modelo en un servidor.
- El enfoque IA sin código ofrece una serie de **oportunidades** en diversos ámbitos. Por un lado, se espera reducir la gran demanda que existen en personas especializadas en IA. Este enfoque también pretende reducir costes, principalmente a las pequeñas y medianas empresas. Por último, y especialmente el punto más importante, se espera que la IA esté al alcance de más personas u organizaciones, y no solo por los grandes gigantes tecnológicos como ocurre hoy en día.

Bloque 4. Convergencia Tecnológica

4

Capítulo

Convergencia Tecnológica

Carlos González Morcillo

En la última década se han alcanzado las condiciones de innovación disruptiva que permiten la eclosión de la convergencia tecnológica. Esta superposición de tecnologías de crecimiento acelerado requiere nuevos enfoques de integración y conexión de servicios. En este capítulo se analizan las características generales de este tipo de servicios realizando ejemplos específicos de integración empleando el motor de Microsoft Power Automate. La última parte del capítulo se destinará al análisis de algunos avances directamente relacionados con la convergencia tecnológica y la Inteligencia Artificial: Blockchain, IoT y la conexión con entornos audiovisuales e IA.

4.1. Introducción a la convergencia tecnológica

Para analizar correctamente la convergencia tecnológica, es importante comenzar con la idea de que existe un conjunto de tecnologías que se *aceleran exponencialmente*. Esto tiene su impacto tanto en el aumento de las prestaciones como en la bajada de precio a lo largo de tiempo. Un ejemplo clásico es el de la Ley de Moore, enunciada en los años 60 por Gordon Moore, uno de los socios fundadores de Intel. Predijo que cada dos años los ordenadores serían el doble de potentes manteniendo su precio. Aunque en su predicción original Moore habló de un escenario de corto plazo, esta ley sigue siendo válida aún hoy en día¹.

¹Aunque hay ciertas noticias que alertan del fin de la Ley de Moore alrededor del 2025 por limitaciones físicas en la tecnología de fabricación de los propios chips, varios autores [Wal16], [McB19] apuntan de un modo muy optimista por nuevos enfoques basados en computación cuántica, técnicas de inteligencia artificial y nuevas arquitecturas de chip.



Implicaciones de la Ley de Moore. Esta ley explica por qué el móvil que tienes es mil veces más pequeño, mil veces más barato y un millón de veces más potente que un supercomputador de los años 70.

Los circuitos integrados no es la única tecnología que tiene este tipo de crecimiento. En 2004 el director de ingeniería de Google, Ray Kurzweil, enunció la que se conoce como "*Ley del rendimiento acelerado*" [Kur04] por la que cualquier tecnología que se vuelve digital empieza a acelerarse de forma exponencial. Además, esta aceleración genera una realimentación positiva sobre otras tecnologías que aceleran aún más la propia aceleración².

En los últimos años se está haciendo más notable el efecto de la convergencia tecnológica. Los avances tecnológicos están *solapándose* con efectos potenciadores en otras disciplinas. Por ejemplo el desarrollo de nuevas vacunas se está acelerando por los avances en biotecnología junto con nuevos paradigmas en inteligencia artificial, big data y nanotecnología entre otros.

¿Por qué ahora es el momento de hablar de *convergencia tecnológica*? La revolución de la IA y el Big Data nos permite procesar una cantidad de datos enorme, permitiendo inferir resultados en fragmentos de segundo. Esto tiene aplicación en multitud de ámbitos industriales donde se recopilan datos de diversos tipos de sensores centralizados o distribuidos (por ejemplo mediante IoT) y se procesan para dar resultados en tiempo real que optimicen procesos y servicios. El precio y la rapidez de servicio son dos factores fundamentales en muchos ámbitos de consumo, con una conexión directa con la percepción de calidad. En estas dos dimensiones la convergencia tecnológica y la aplicación de IA juega un papel determinante.



Ley de rendimiento acelerado. Según la ley de Kurzweil en este siglo vamos a vivir el equivalente a unos 20.000 años de cambios tecnológicos. Será el equivalente a condensar todos los avances desde el descubrimiento de la escritura a la aparición de Internet en 30 años.

Por desgracia, muchas de las empresas y organizaciones consolidadas tendrán problemas para seguir el ritmo de esta convergencia acelerada. Según Foster [FK11] el 40 % de las empresas que aparecen en el famoso ranking Fortune 500 desaparecerán en 10 años reemplazadas por empresas emergentes que aún no existen.

4.2. Visión general

La ley de rendimiento acelerado tiene un fuerte impacto en la convergencia tecnológica y en la evolución en la vida de las personas. La vida del tatarabuelo de tu tatarabuelo era más o menos igual a la vida de su tataranieto. Sin embargo, ahora la tecnología tiene un fortísimo impacto en el mundo global y exponencial en el que

²Entre estas innovaciones podemos destacar la propia inteligencia artificial, la computación cuántica, la nanotecnología, el IoT, la realidad aumentada y blockchain entre otras.

vivimos. La organización y costumbres de la vida de las personas puede cambiar en cuestión de meses. Desde la aparición en 2007 del iPhone el número de dispositivos móviles a internet ha crecido exponencialmente (ver figura 4.1). En 2012, el número de smartphones y tablets con conexión a internet superaba al número de ordenadores (entre escritorio y portátiles juntos). En pocos años, los hábitos de consumo y publicación de información en redes sociales por parte de la humanidad también lo ha hecho.

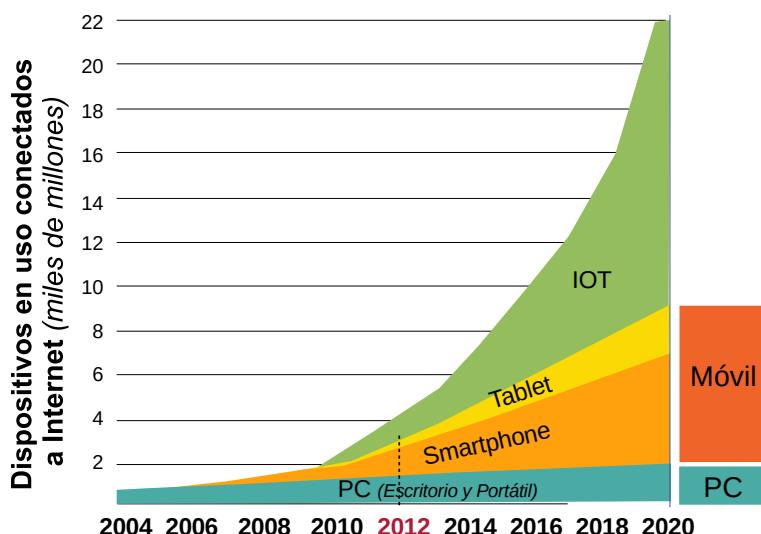


Figura 4.1: Evolución en el número de dispositivos conectados a internet. Fuente: Gartner IDC + Forbes (IOT). Diagrama de elaboración propia.

Diamandis y Kotler [DK20] identifican en su libro 9 tecnologías de crecimiento exponencial y una serie de agentes secundarios que están acelerando el ritmo del cambio en el mundo, así como el tamaño del impacto de esta convergencia tecnología. Además de la *Inteligencia Artificial* y el Blockchain (cuya implicación será estudiada en la sección ??), las otras 7 tecnologías amplificadoras son las siguientes:

Computación Cuántica. A diferencia de la computación clásica donde un bit representa el fragmento más pequeño de información binaria (0 o 1), un *cúbit* (o bit cuántico) pueden estar en múltiples estados al mismo tiempo mediante superposición³. La computación cuántica despliega todo su potencial a la hora de manipular gran cantidad de datos de una vez, mientras que la computación tradicional se muestra eficaz en procesamiento secuencia. En 2002 Geordie Rose, fundador de una de las primeras empresas de computación cuántica - DWave enunció la Ley de Rose: “El número de cúbits de un ordenador cuántico se duplica cada año”. En la página de Rigetti Computing www.rigetti.com se puede descargar el Toolkit

³La superposición se alcanza con temperaturas de trabajo muy bajas.

Forest para desarrolladores cuántico. Forest se construye sobre Quil, un lenguaje con instrucciones híbridas para algoritmos con parte clásica y parte cuántica. Además, ofrecen el computador de 32 cúbits de Rigetti donde ejecutar los programas desarrollados.

Redes - IoT. En la actualidad más de la mitad de los humanos de la tierra están conectados a internet con unos costes muy bajos. Con el despliegue de 5G se conseguirán tiempos de latencia y ancho de banda que permitirán la teleoperación desde cualquier lugar. Se tardará menos de 2 segundos en descargar una película completa en HD. La nueva red de satélites fabricados por Boeing O3B dará cobertura a todas las personas que hoy en día carecen de acceso de alta velocidad a Internet. La previsión es que para 2030 tengamos unos 500.000 millones de dispositivos conectados a la red (cada uno con decenas de sensores). Esta implantación de sensores conectados proporciona una ingente cantidad de datos sobre la que se pueden aplicar técnicas de procesamiento automático basadas en IA.



Evolución de coste en sensores. El primer GPS de consumo que llegó al mercado en 1981 costaba 110.000€ y pesaba 24 kilos. Treinta años después se había reducido a un chip de 1 cm de lado con un precio de 4€. El primer sensor de cámara digital de fotos (Kodak, 1976) tenía el tamaño de un microondas y costaba 9.000€. La cámara que viene con un *smartphone* actual es mil veces mejor en peso, coste y resolución.

Robótica. No cabe duda que los robots están entrando en nuestras casas y forman parte de nuestro día a día, realizando trabajos que son aburridos o peligrosos. Los desarrollos de Boston Dynamics sorprenden a la comunidad con máquinas adaptables a situaciones inesperadas, con movimientos rápidos y precisos. La robótica industrial ha evolucionado de máquinas que eran peligrosas y debían estar separadas de los operarios humanos al concepto de *cobot* (*robot colaborativo*) que trabajan de la mano de operarios humanos... Y han venido para quedarse. La empresa *Universal Robots* (www.universal-robots.com) alquila cobots por 600€ al mes en todo el mundo. Si lo deseas puedes comprar tu propio dispositivo por menos de 20.000€. En la convergencia con las tecnologías exponenciales anteriormente descritas, los *drones* suponen la conexión de la robótica con sensores conectados y el control de los dispositivos de vuelo y motores mediante métodos de inteligencia artificial.

Realidad Virtual y Realidad Aumentada. Aunque como muchas de las tecnologías que hemos visto anteriormente, la Realidad Virtual y la Realidad Aumentada no son desarrollos nuevos, pero la convergencia tecnológica ha conseguido que ahora hayan alcanzado un punto de eclosión. Para el 2025 la previsión es que el mercado del videojuego alcance 11.000 millones \$, seguido por las aplicaciones sanitarias con 5.000 millones \$ y el uso industrial con más de 4.500 millones \$. Dispositivos de reciente aparición en el mercado como las *Oculus Quest 2* de Face-

book, con una precisión de *tracking* libre menor de 1 milímetro y un precio menor de 400€ auguran una rápida adopción en hogares de todo el mundo. Por su parte, las HoloLens 2 de Microsoft se han convertido en el dispositivo más extendido en aplicaciones de Realidad Aumentada (Mixta) de ámbito profesional.

Impresión 3D. Desde la aparición de las primeras impresoras 3D en los años 80, la convergencia tecnológica ha permitido importantísimos avances en todos los aspectos. Actualmente podemos imprimir en cientos de materiales diferentes; desde materiales inorgánicos (todo tipo de metales, plástico, hormigón...) hasta compuestos orgánicos (células, cuero y alimentos). Actualmente se realiza impresión 3D de prácticamente todo, desde prótesis ortopédicas hasta piezas de motores a reacción e incluso bloques de pisos. La complejidad de las piezas impresas aumenta con tiempos de producción y costes cada vez menores. Existen decenas de sitios donde ofrecen servicios de impresión muy competitivos (como www.shapeways.com), y diversos fabricantes ofrecen impresoras 3D domésticas. La convergencia tecnológica con la informática ha permitido sacar nuevos productos de impresión, como la impresora que fabrica placas bases de *Nano Dimension*. La convergencia con la biotecnología ha permitido ya imprimir piezas de recambio para órganos humanos, como la impresión del primer vaso sanguíneo producido en San Diego en 2010.

Nanotecnología y ciencia de los materiales. A finales del siglo XIX, Thomas Edison tuvo que probar en más de un año cerca de 2.000 materiales antes de encontrar un material que le permitiera fabricar la primera bombilla eléctrica. Tuvieron que pasar varias décadas hasta que nuevas pruebas dieron con los filamentos de tungsteno, mucho más brillantes y resistentes que permitían fabricar bombillas más duraderas y brillantes. Actualmente las pruebas de materiales se realizan mediante simulaciones empleando ordenadores. Lo que hace años requería meses de trabajo hoy en día se realiza en horas. En 2011 en la Universidad de Carnegie Mellon se presentó la iniciativa del Genoma de los Materiales que emplea la IA para cartografiar los millones de combinaciones posibles de elementos y predecir las propiedades de los nuevos materiales que aún no existen. Estos avances permiten generar un mapa del mundo físico que facilita la generación de nuevos materiales a nivel de átomo. Este nivel de convergencia tecnológica ha permitido construir materiales más ligeros, resistentes y nuevos biomateriales que pueden implantarse en humanos sin rechazo.



Sin convergencia no es posible. Como explica el director de Applied Materials, O. Nalamasu, si se hubiera intentando construir una versión de un smartphone actual en los años 80, el dispositivo costaría unos 100 millones €, tendría la altura de un edificio de 5 plantas con el consumo energético de 20 viviendas familiares.

Biotecnología. La biotecnología se basa en la idea de transformar los elementos básicos de la vida (genes, proteínas y células) en herramientas manipulables. En el cuerpo humano tenemos unos 40.000 millones de células. Cada una de estas células tiene unas 6.400 letras codificadas en el ADN (la mitad de cada progenitor). Desde que finalizó el proyecto de secuenciación del Genoma Humano en 2001 y que costó

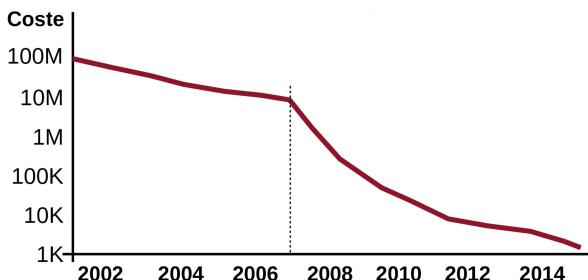


Figura 4.2: Evolución del coste de secuenciación del genoma desde 2002 en miles (K) y millones (M) de dólares. Fuente: Veritas Genetics 2015.

100 millones de dólares (ver Figura 4.2), los precios y plazos de secuenciación han caído. Actualmente secuenciar un genoma humano lleva unos pocos días y cuesta menos de 800€. Empresas como Illumina pometen hacer lo mismo en una hora y por menos de 100€. Secuencias el genoma de un modo rápido y barato va a permitir que técnicas de edición genética que permiten reparar el ADN que hay en el interior de las células o terapias basadas en células madre lleguen al mercado.

4.3. Ventajas de la convergencia tecnológica

En esta sección se identificarán algunas de las principales ventajas que presenta la convergencia tecnológica en el ámbito de la unificación de procesos, servicios, herramientas, métodos y sectores.

Como hemos visto en la sección anterior, gracias a la convergencia tecnológica es posible conseguir avances que de otro modo no serían posibles. Diamandis y Kotler [DK20] identifican una serie de ventajas que proporciona la convergencia:

- **Ahorro de tiempo.** El ahorro de tiempo puede verse como un motor de la innovación; realizar las tareas de un modo más rápido acelera el propio ritmo de progreso.
- **Disponibilidad de capital.** Nuevas posibilidades de financiación mediante *crowdfunding*, activos digitales que no podrá consumirse ni sustituirse (NFTs) e incremento de valor sin precedentes en empresas emergentes tecnológicas. Las 5 marcas más valiosas del mundo son los principales desarrolladores de IA: Google, Amazon, Facebook, Apple y Microsoft.
- **Nuevos modelos de negocio.** La convergencia tecnológica con la innovación disruptiva proporciona nuevos modelos de negocio. Hasta el momento de la convergencia tecnológica, los modelos de negocio habían cambiado lentamente a través de modelos de innovación sostenida (mejorar un producto o servicio a partir de pequeños pasos, muy costosos y constantes).

- **Desmonetización.** Al digitalizarse los productos o los servicios el dinero desaparece (casi totalmente) de la ecuación. Hace años relevar un carrete fotográfico era caro. Cuando la fotografía se hizo digital, los costes desaparecieron.
- **Desmaterialización.** La Wikipedia desmaterializó las enciclopedias. Prime-ro iTunes y posteriormente Spotify desmaterializaron las tiendas de discos. Esto es lo que ocurre cuando los productos entran en contacto con tecnologías aceleradas.



Decisiones estratégicas. Una de las principales ventajas de la convergencia tecnológica está directamente relacionada con la mejora en la capacidad de toma de decisiones estratégicas en un negocio conectado. La disponibilidad de gran cantidad de datos y el análisis inteligente de los mismos pueden suponer una enorme ventaja a la hora de tomar decisiones estratégicas.

4.4. Plataformas para la conexión tecnológica

A continuación se comentarán algunas de las plataformas y sistemas más extendidos que facilitan la conexión tecnológica. En la sección 4.4.1 secciones se analizarán las características principales de las plataformas elegidas para realizar los ejemplos de este capítulo.

El término *Glue Code* se utiliza para definir el código empleado para conectar componentes de software que inicialmente no son compatibles. Habitualmente el *glue code* no se escribe en el mismo lenguaje en el que están definidas las aplicaciones a conectar, sino que suelen emplearse lenguajes interpretados de alto nivel⁴ que permiten el desarrollo rápido (y normalmente con cierto impacto en el rendimiento y en la eficiencia).

Este tipo de código es habitual cuando es necesario integrar diferentes servicios en plataformas heterogéneas (por ejemplo Amazon y Google), de modo que el *glue code* sirve para conseguir la convergencia tecnológica entre los flujos de trabajo de estos servidores que, de otro modo, serían incompatibles. Para evitar que este código se transforme en el temido *spaghetti code* cuando aumenta su complejidad, es necesario utilizar frameworks que faciliten el desarrollo y garanticen la robustez y mantenibilidad de la solución.

⁴Algunos ejemplos de lenguajes utilizados para *glue code* son JavaScript, Perl, PHP, Python, Ruby, VBScript, Bash Script y PowerShell entre otros.

Una tendencia muy extendida actualmente es el uso de tecnologías que facilitan la preparación y combinación de datos (incluyendo transformación, normalización y tratamiento de los datos) en un entorno sin servidores explícitos en la nube. Así, las plataformas Cloud facilitan la convergencia tecnológica escalando automáticamente los recursos necesarios para la ejecución de los trabajos de integración de datos. Algunas de las plataformas en la nube más extendidas para facilitar la convergencia tecnológica son las siguientes:

- **AWS Glue.** Servicio de integración de datos en la nube de Amazon. Incorpora mecanismos avanzados como el descubrimiento automático de esquemas de datos, definición de flujos de datos en *streaming* (en tránsito), definición visual de nodos de tratamiento (mediante *AWS Glue Studio*) y replicación de datos en varios almacenes de datos con vistas materializadas (soporte mediante *AWS Glue Elastic View*). Puede verse como una plataforma en la nube ETL (*Extract, Transform, Load*) que da soporte para mover los datos desde diversas fuentes a un *data warehouse* específico. En el caso de AWS Glue existe una conexión directa con el servicio específico de almacen de datos de AWS: *Redshift*.
- **Integrate.io.** Plataforma que define un flujo de trabajo ETL en la nube. Al igual que AWS Glue dispone de servicios avanzados de replicación de datos en tiempo real. Proporciona flujo de trabajo visual y soporta integración en los almacenes de datos más extendidos: *AWS Redshift* y *Snowflake*.
- **Microsoft Power Automate.** Motor de integración de aplicaciones y de automatización en tareas de conversión y tratamiento de datos que forma parte de la suite Power Platform de Microsoft (ver sección 4.4.1). Será estudiado en detalle a lo largo del capítulo, en la sección 4.5 y siguientes.
- **Boomi.** Comenzó a ofrecer sus servicios en 2007 siendo la primera plataforma en ofrecer servicios iPaaS (*integration Platform as a Service*). Fue la primera plataforma que definía un sistema de nodos visuales que permite construir procesos de integración mediante *drag&drop*. Recientemente la compañía ha sido adquirida por *Dell* y la plataforma forma parte de la nube *AtomSphere*.
- **Zapier.** Plataforma de automatización de tareas de integración de aplicaciones web. Puede entenderse como un traductor de APIs web que permite la definición de *Zaps*: flujos de conexión entre aplicaciones que definen acciones que son disparadas a partir de ciertos eventos (*triggers*).

Herramientas *Cloud* de Convergencia Tecnológica

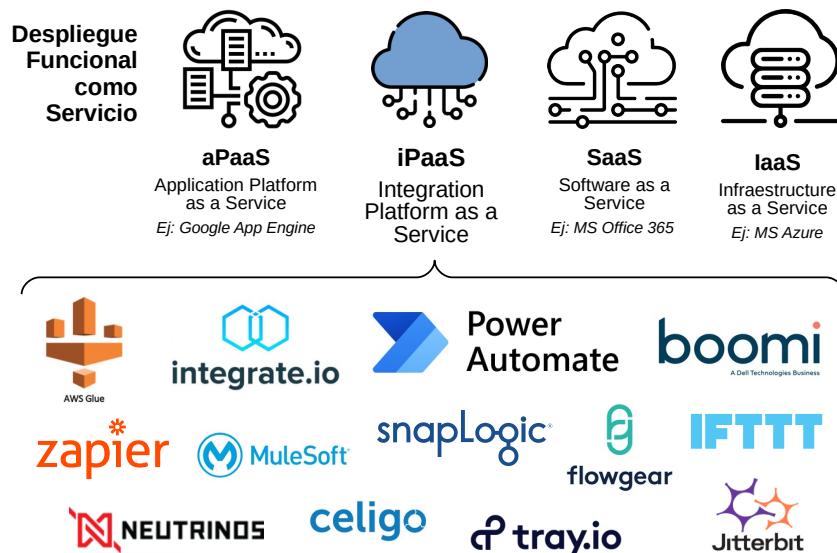


Figura 4.3: Desde el punto de vista de las plataformas de integración *Cloud* que podrían ser catalogadas como *iPaaS*, existe una amplia oferta de alternativas tecnológicas.

Existen multitud de proveedores que ofrecen servicios cloud para la integración de plataformas existentes y tratamiento de datos, como *Mulesoft*, *SnapLogic*, *Celigo*, *Jitterbit*, *Flowgear* o *Neutrinos*⁵ dentro del ámbito profesional o *IFTTT* y *Tray.io* en el ámbito de usuario final entre otras. Dependiendo de las necesidades específicas de cada proyecto, puede ser conveniente analizar tanto las prestaciones como los costes asociados a cada proveedor. De igual modo también es posible combinar las soluciones proporcionadas por diferentes plataformas de iPaaS.

4.4.1. Power Platform

Una de las plataformas más populares de sistemas cloud para la conexión tecnológica es **Microsoft Power Platform**. En el ecosistema de *Power Platform* se cuenta con varios módulos y componentes que proporcionan servicios en la nube para crear e integrar aplicaciones empresariales existentes de forma rápida y eficiente (ver Figura 4.4). Algunos de los módulos principales de la plataforma son los siguientes:

⁵La única alternativa estudiada que no utiliza bibliotecas de código propietario/privativo en su solución.

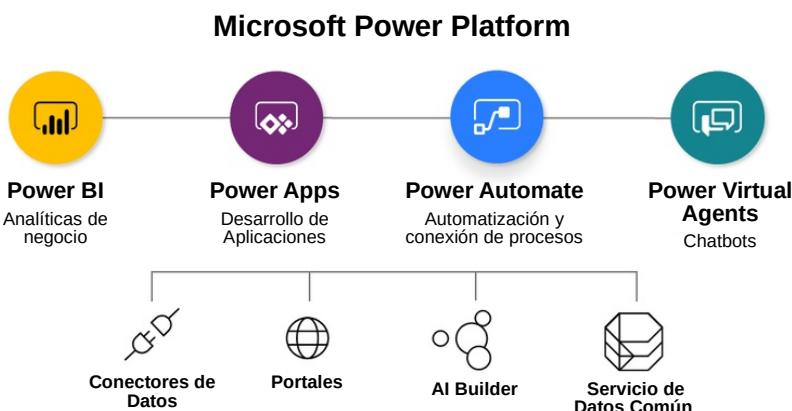


Figura 4.4: Principales módulos de la *Microsoft Power Platform*.

- **Power Apps.** Define un conjunto de componentes y plantillas de alto nivel para crear aplicaciones empresariales para PC y dispositivos móviles, así como portales web de acceso público utilizando el enfoque de desarrollo casi sin código.
- **Power Automate.** Permite la integración y conexión de aplicaciones existentes, así como la automatización de tareas de conversión y tratamiento de datos. Este motor de conexión de flujos se estudiará en detalle en la sección 4.5 y siguientes.
- **Power BI.** Herramienta de análisis y generación de informes empresariales muy consolidada y flexible. Ha sido estudiada en el capítulo correspondiente del módulo de *Sistemas de Big Data* de este curso.
- **Power Virtual Agents.** Herramienta que permite crear chatbots conversacionales que respondan preguntas de clientes o visitantes de una web. Construye bots que pueden desplegarse como componentes web, aplicaciones móviles o integrarse en diversas aplicaciones como Teams, Skype, Cortana, Facebook, Slack y Telegram entre otras.
- **AI Builder.** Proporciona un conjunto de modelos de IA precompilados que facilitan la realización de predicciones, detección de objetos, clasificación y análisis del lenguaje natural entre otras. Estos modelos se pueden incorporar casi sin código mediante conectores integrados en la Power Platform.

La *Power Platform* define dos nuevos conceptos que deben entenderse a la hora de integrar correctamente servicios y aplicaciones: el **Modelo de Datos Común (CDM)** (*Common Data Model*) y el **Servicio de Datos Común (CDS)** (*Common Data Service*).

El CDM está formado por un conjunto de esquemas (de datos) y un sistema de metadatos. El modelo define un conjunto pequeño de entidades principales que no están relacionadas con ningún ámbito de trabajo, además de un conjunto extenso de entidades que sí dependen de un dominio de aplicación concreto (banca, salud, educación, automoción, media y ONGs entre otros).



Modelo de Datos Común (CDM). El objetivo principal del CDM es definir una plataforma común para la integración de datos y el desarrollo de aplicaciones. Fue inicialmente presentado por el consorcio de Microsoft, Adobe y SAP como parte de la *Open Data Initiative*.

El CDS (ver Figura 4.4) se encarga de establecer la base sobre la que se construyen las aplicaciones basadas en modelos. Este tipo de aplicaciones se definen con dos componentes: un identificador (nombre y dirección), y un *Site Map* (esquema de navegación). En la construcción de estas aplicaciones basadas en modelos se establecen diversos modelos de automatización y conexión de componentes.



Servicio de Datos Común (CDS). El objetivo principal del CDS es dar la implementación necesaria del CDM para almacenar los datos para las aplicaciones.

4.5. Análisis de Power Automate

Power Automate (en versiones anteriores llamado *Flow*) es el motor para construir flujos de automatización dentro de Power Platform⁶. El uso de este motor de flujos facilita la conexión entre soluciones tecnológicas existentes sin la necesidad de disponer de un enfoque de desarrollo centralizado o de disponer de muchos medios para el desarrollo de soluciones específicas de integración.

Un flujo de *Power Automate* consiste en un desencadenador (disparador) y una lógica de la aplicación. Esta lógica se define mediante elementos de control de flujo (condiciones y bucles) y operaciones que se implementan con conectores.

En las siguientes subsecciones estudiaremos los principales elementos que conforman este módulo de la *Power Platform* a partir de un conjunto de ejemplos.

⁶La tecnología subyacente es la de *Azure Logic Apps*. El uso de conectores y el diseño gráfico de sus flujos son muy parecidos.

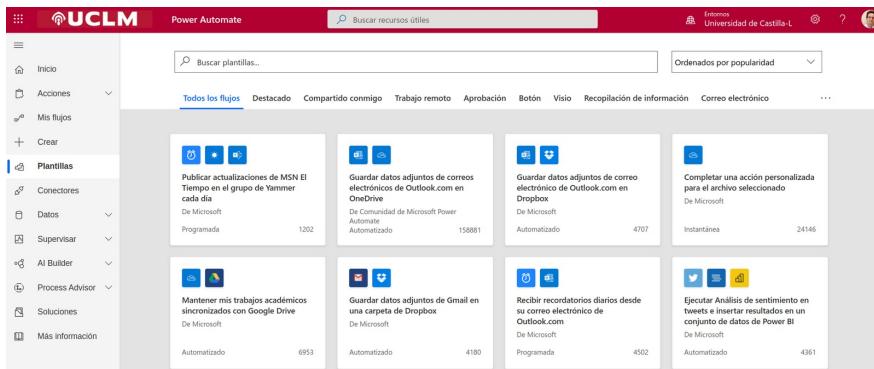


Figura 4.5: Interfaz que muestra la selección de plantillas predefinidas en Power Automate.

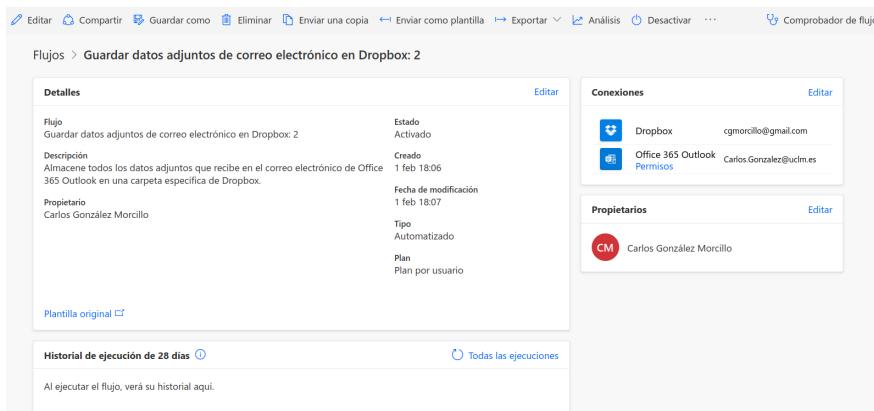


Figura 4.6: Resumen del flujo creado. En la zona superior de la ventana se encuentran las opciones principales de administración del flujo.

4.5.1. Ejemplo 1. Hola Mundo y uso de plantillas

En este primer ejemplo crearemos un flujo basado en una de las múltiples plantillas que hay disponibles en la plataforma. Para ello, en el menú principal de la herramienta (ver Figura 4.5) pincharemos en el botón **[Plantillas]**, y localizaremos la plantilla titulada «*Guardar datos adjuntos de correo electrónico en Dropbox*». Puede ser de utilidad emplear el buscador de plantillas de la zona superior de la pantalla.

Para utilizar la plantilla tendremos que conectar con las cuentas específicas de los servicios que emplea. Antes de definir los detalles de los nodos asociados al flujo será necesario conectar con la cuenta de Dropbox y de Outlook. Cuando las dos cuentas estén correctamente conectadas, pulsaremos el botón **[Crear flujo]** en la parte inferior de la pantalla.

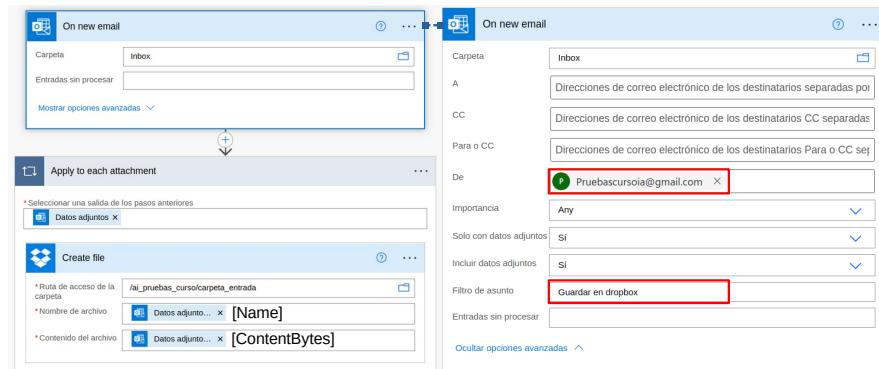


Figura 4.7: Definición del flujo que guarda los ficheros adjuntos recibidos en una cuenta de Outlook 365 en una carpeta de Dropbox, indicando ciertas propiedades de filtrado (nombre del remitente y asunto).

A continuación aparecerá la pantalla de la Figura 4.6, con las principales opciones de administración del flujo, y un resumen (en el panel de *Detalles*) del mismo, estado, fecha de creación, etc. El botón de **Editar** situado en la parte superior izquierda de la pantalla permite editar los detalles de uso del flujo. También son de uso básico los botones de **Desactivar** que permite desactivar el flujo sin necesidad de eliminarlo y **Análisis** que muestra el histórico del flujo de ejecución del flujo en un periodo de tiempo.

Cuando accedemos a la edición del flujo es posible realizar una configuración más precisa del nodo desencadenador y de las acciones a realizar. En el caso de este primer ejemplo, se han definido dos propiedades de filtrado en el nodo desencadenador; el email del remitente y el asunto. Para acceder a esta configuración es necesario pinchar en la etiqueta de *Mostrar propiedades avanzadas* del nodo (ver Figura 4.7).

El nodo de acción de Crear Archivo en Dropbox utiliza dos propiedades de contenido dinámico. Si pinchamos sobre las entradas de texto de *Nombre de archivo* y *Contenido del archivo* de este nodo aparecerá una ventana emergente que permite seleccionar el contenido dinámico de los nodos que se han utilizado anteriormente. En el caso de este ejemplo se han utilizado los campos *Datos adjuntos Nombre* y *Datos adjunto Contenido* del nodo desencadenador de Outlook 365.

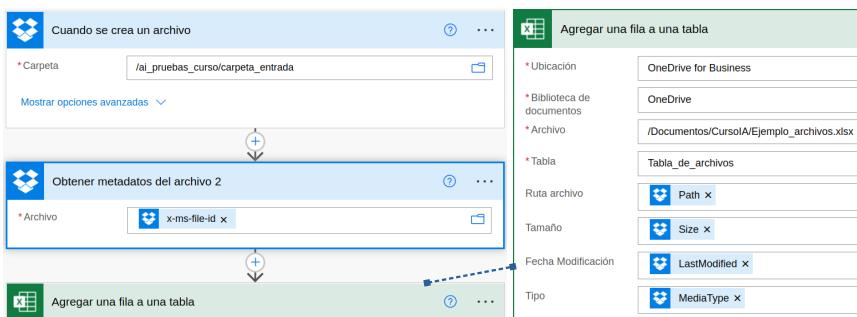


Figura 4.8: Definición del flujo automatizado que agrega una fila a una tabla de Excel disponible en OneDrive con información de un nuevo archivo añadido a una carpeta de Dropbox.

4.5.2. Ejemplo 2. Creación de flujo automatizado (Creación de Archivo a Excel)

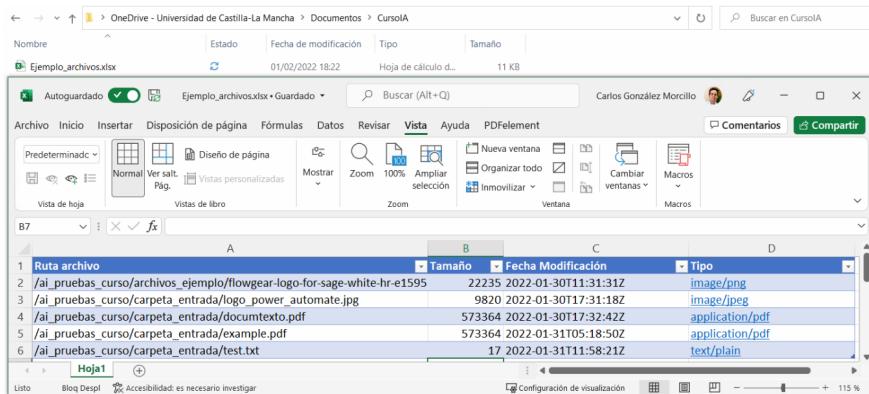
En este segundo ejemplo se creará un flujo automatizado manualmente. Para ello, en la pantalla principal pincharemos en el botón **[Crear]** y a continuación elegiremos *Flujo de nube automatizado*. Elegiremos como nombre del flujo cualquiera que sea representativo para la tarea (por ejemplo *Registro de archivos en Dropbox a Excel*), y como desencadenador de flujo⁷ *Cuando se crea un archivo* de Dropbox. Hecho esto, pincharemos en el botón **[crear]**.

Los dos primeros del nodo son muy sencillos. Tras detectar la creación del archivo en un directorio determinado, añadiremos un nodo que obtiene los metadatos del archivo (ver Figura 4.8). Este nodo utiliza como entrada la identificación dinámica del archivo del nodo anterior. El campo *x-ms-file-id* proviene del nodo desencadenador e identifica cada archivo de Dropbox de un modo único. Estos campos de metainformación son los que se utilizan posteriormente en el nodo que agrega una fila a una tabla en Excel.

El archivo de Excel está disponible en la nube corporativa de OneDrive, de modo que la actualización de los datos se realiza directamente en la nube y se propaga a todos los entornos que lo estén usando en modo conectado.

Los campos se han definido creando directamente la tabla desde Excel (en el menú Insertar/Tablas/Tabla). Como se muestra en la figura 4.9, aunque la definición de los datos es muy sencilla, el proceso es fácilmente exportable a otros dominios de aplicación.

⁷Puedes escribir "Dropbox.en" la caja de texto para filtrar entre las decenas de desencadenadores disponibles.



	A	B	C	D
1	Ruta archivo	Tamaño	Fecha Modificación	Tipo
2	/ai_pruebas_curso/archivos_ejemplo/flowgear-logo-for-sage-white-hr-e1595	22235	2022-01-30T11:31:31Z	image/png
3	/ai_pruebas_curso/carpetas_entrada/logo_power_automate.jpg	9820	2022-01-30T17:31:18Z	image/jpeg
4	/ai_pruebas_curso/carpetas_entrada/documento.pdf	573364	2022-01-30T17:32:42Z	application/pdf
5	/ai_pruebas_curso/carpetas_entrada/example.pdf	573364	2022-01-31T05:18:50Z	application/pdf
6	/ai_pruebas_curso/carpetas_entrada/test.txt	17	2022-01-31T11:58:21Z	text/plain

Figura 4.9: Definición una tabla de 4 columnas en Excel y compartición del archivo Excel en OneDrive.

4.5.3. Ejemplo 3. Creación de flujo programado y utilización de variables de flujo

Aunque los límites de ejecución de Power Automate son bastante altos⁸, en muchas ocasiones no es necesario disparar flujos continuamente. En muchas ocasiones será mejor tener flujos de ejecución programados que obtengan resultados y los almacenen periódicamente. En este ejemplo enviaremos un resumen del número de archivos y tamaño total de un directorio de Dropbox a una dirección de correo. Aunque el ejemplo utiliza la acción de enviar un email, podría tener más sentido almacenar los resultados en una base de datos (o en una sencilla tabla de Excel como en el ejemplo anterior).

La Figura 4.10 muestra la descripción del flujo completo. Los nodos de color violeta trabajan con variables. Al *añadir un nuevo paso* en el listado de nodos de acción podemos buscar por el literal "*Variable*", y aparecerán todos los nodos específicos para la creación de variables *Iniciar variable* o, en general, para trabajar con esos valores.

Estas variables pueden usarse como cualquier otro contenido dinámico dentro del resto de nodos. Por ejemplo, en el envío del correo electrónico cada hora se emplea el valor de las variables para mostrar el resumen del número de archivos y el tamaño de los mismos.

⁸Consultar detalles en <https://docs.microsoft.com/es-es/power-automate/limits-and-config>

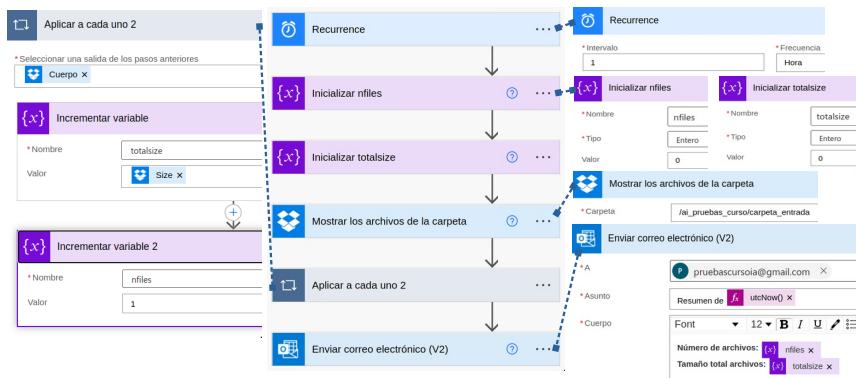


Figura 4.10: Flujo automatizado que envía un resumen del número de archivos y tamaño total de un directorio de Dropbox a una dirección de correo cada hora. El flujo principal se define en la columna central de la figura. La columna de la izquierda contiene las acciones que están incluidas en el nodo de "Aplicar a cada uno". Los nodos de color violeta utilizan variables definidas en el ámbito del flujo.

4.5.4. Ejemplo 4. Creación de flujo de escritorio

El cliente de escritorio de Power Automate permite automatizar multitud de procesos repetitivos de escritorio. En estos primeros ejemplos utilizaremos la funcionalidad básica de este cliente con funcionalidad encapsulada para la ejecución de acciones sobre archivos existentes. En ejemplos posteriores veremos cómo diseñar flujos de escritorio usando acciones de *Drag&Drop* o grabando directamente flujos que pueden ser reproducidos después.

Para definir un flujo de escritorio, pincharemos en **Crear** y elegiremos *Flujo de escritorio*. Si no hemos instalado en el ordenador el cliente para definir flujos de escritorio podremos descargarlo e instalarlo. Una vez que iniciamos sesión, es posible añadir acciones directamente del panel situado en la izquierda (ver Figura 4.11). En este primer ejemplo utilizaremos únicamente una acción de extracción de texto de un archivo PDF, que estará localizado en una carpeta local sincronizada en Dropbox.

En la zona de la derecha de la ventana, en la sección *Variables* es posible definir variables de entrada y salida para el flujo. En este ejemplo definiremos tres variables de entrada *filename* (que contendrá el nombre del archivo a procesar), *frompage* (página inicial) y *topage* (página final). Se definirá una variable de salida *outputtext* que contendrá el texto extraído del PDF en el rango de páginas indicados en las variables de entrada.

La ejecución del script de escritorio anterior se realiza a partir del ejemplo que hemos definido en un ejemplo anterior (ver sección 4.5.1). Cuando se recibe el email se añade el archivo a Dropbox y posteriormente se ejecuta el flujo de escritorio (ver Figura 4.12).

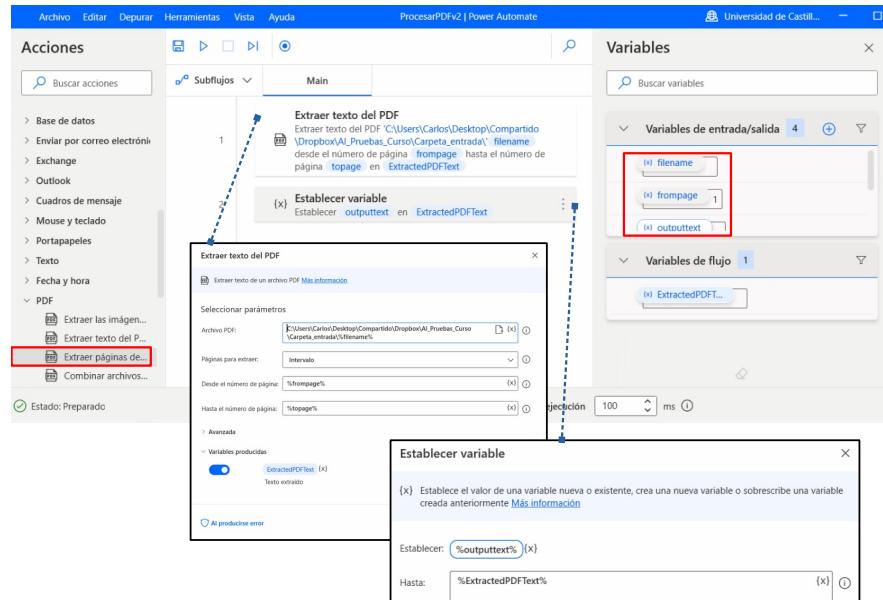


Figura 4.11: Definición del flujo de escritorio en Power Automate para escritorio. El flujo utiliza la funcionalidad del bloque de procesamiento de PDFs y define tres variables de entrada y una de salida.

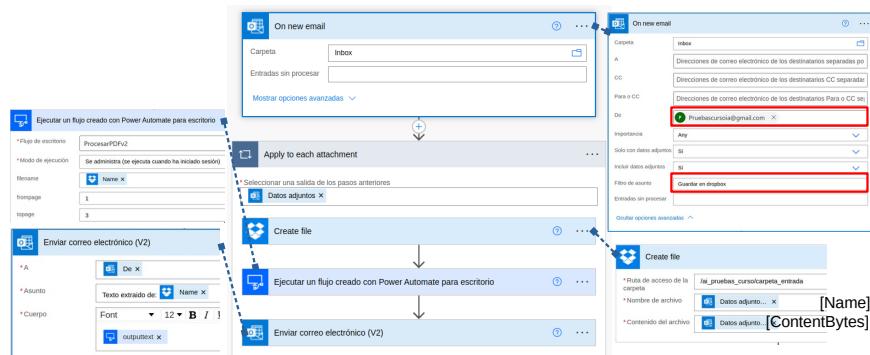


Figura 4.12: Ampliación del ejemplo estudiado en la sección 4.5.1. Tras añadir cada archivo a Dropbox se ejecuta el flujo de escritorio y se envía el resultado del procesamiento por email al remitente del correo original.

Cabe destacar que el *Modo de ejecución* del flujo de escritorio puede configurarse como *Se administra* (*se ejecuta cuando ha iniciado sesión*) o como *Desatendido* (*se ejecuta en una máquina que ha cerrado sesión*). En las pruebas será normal trabajar en modo "*Se administra*", pero cuando se ejecute el flujo en producción, lo normal será en modo *Desatendido*. En el nodo de envío del email (ver Figura 4.12) se ha incluido como cuerpo del mensaje directamente el campo de salida de texto del nodo de escritorio.

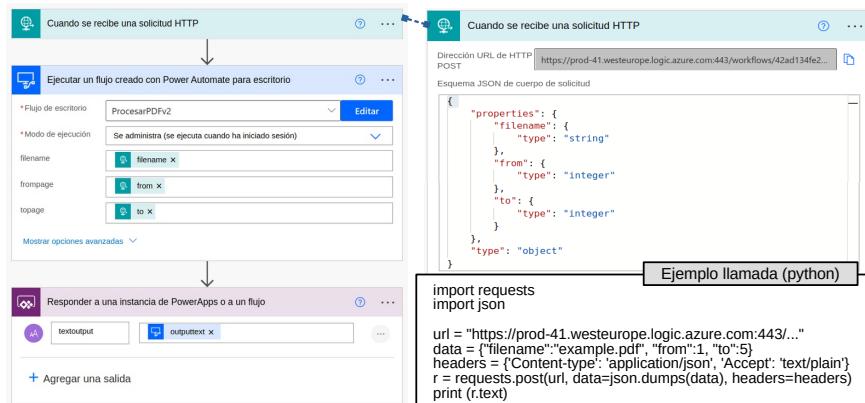


Figura 4.13: Flujo en la nube automatizado que recibe llamadas basadas en un esquema Json. La llamada desencadena instantáneamente el conjunto de peticiones al flujo de escritorio definido en el ejemplo de la sección 4.5.4.

4.5.5. Ejemplo de llamado mediante API REST

En este ejemplo utilizaremos una interfaz REST (*REpresentational State Transfer*) sobre HTTP. Enviaremos una petición Json sobre una interfaz REST que expone el servidor (en este caso, gestionada por Power Automate). Los recursos en REST se manipulan a partir de una URI (identificador universales de recursos). Esta URI estará disponible cuando guardemos el flujo.

Las peticiones a la API están asociadas a uno de las *acciones* de HTTP: GET para obtener un recurso, POST para escribir un recurso, PUT para modificar un recurso y DELETE para borrarlo. En este ejemplo no vamos a ser especialmente formales con el uso de estos verbos y dejaremos la acción por defecto.

La Figura 4.13 muestra los nodos que se han utilizado a la hora de definir el ejemplo. Es necesario especificar el esquema de Json que va a aceptar las peticiones HTTP. Este esquema se puede validar con cualquiera de los múltiples servicios online que hay disponibles⁹. Una vez guardado el flujo, estará disponible la URL de HTTP que tendrá que incluirse en el script de ejemplo en Python que se ha desarrollado para comprobar que el funcionamiento es correcto.

El nodo de *Responder a un flujo* permite devolver el valor de la variable del script de escritorio codificada en una cadena Json (empaquetada como contenido del atributo *textoutput*).

⁹Personalmente he utilizado <https://www.jsonschemavalidator.net/>

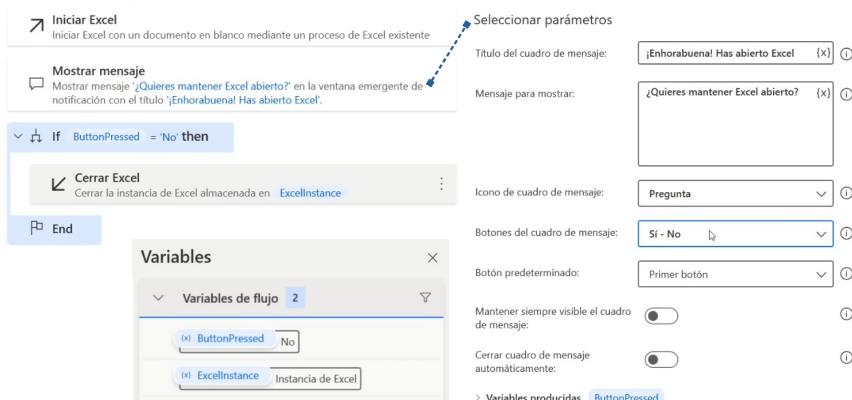


Figura 4.14: Definición de un cuadro de diálogo con dos opciones de respuesta y uso de una variable de flujo sobre un condicional simple.

4.5.6. Ejemplo 5. Uso de condicionales

En este sencillo ejemplo de utilizará un elemento básico de interacción con el usuario: un cuadro de diálogo con varias respuestas posibles (ver Figura 4.14). En el flujo se añade un elemento del grupo *Cuadros de mensaje*, que permite especificar varias alternativas de respuesta.

El valor de la salida se recoge en la variable de flujo *ButtonPressed*. El inspector del valor de la variable en tiempo de ejecución nos permite saber la etiqueta por la que deberá compararse en el condicional.



Sobre la traducción de la salida de ButtonPressed. Hay que prestar atención a este aspecto, ya que los valores devueltos están en inglés y la traducción que se ha realizado desde el castellano no es en todos los casos la que podría esperarse.

4.5.7. Ejemplo 6. Captura de eventos de escritorio

En la versión de escritorio de Power Automate existe un componente que permite automatizar tareas de escritorio de un modo muy práctico, simplemente realizando las acciones, grabándolas y editándolas posteriormente.

En estos casos la grabadora sirve para facilitar el desarrollo de flujos realizando primero las tareas manualmente y, posteriormente, Power Automate generará acciones para replicar el comportamiento.



Figura 4.15: Acceso a la grabadora en la ventana de edición del flujo.

La grabadora se puede utilizar para automatizar tanto aplicaciones de escritorio de Windows como aplicaciones que se accedan desde un navegador web. La grabadora realiza la automatización del interfaz de usuario generando eventos de teclado y ratón que enviará a las aplicaciones de escritorio (o al navegador web en su caso). Aunque en este ejemplo utilizaremos una aplicación de escritorio, los conceptos son similares a la hora de trabajar con cualquiera de los navegadores disponibles en Windows.



Sobre el uso de navegadores... Aunque la grabadora de Power Automate Desktop es compatible con cualquier navegador web (Chrome, Firefox, Edge y Explorer) en las pruebas realizadas se ha detectado que el comportamiento es mejor si se utiliza el navegador recomendado por Microsoft Windows: Edge.

La grabadora está accesible en el menú principal de la ventana de edición del flujo Herramientas >Grabadora o en el botón directamente accesible en la zona superior de la misma ventana (ver Figura 4.15).

Cuando accedemos a la grabadora, aparecerá una nueva ventana como se muestra en la Figura 4.16. Esta ventana cuenta con un botón en la esquina superior izquierda Grabar que permite iniciar el proceso de grabación. Es posible pausar la grabación pinchando de nuevo sobre el mismo botón (que tendrá la etiqueta Pausa). Cuando la herramienta está en modo grabación remarcará en color rojo el elemento del interfaz que va a ser capturado (ver imagen central en la Figura 4.16, con el botón 3 de la capturadora capturado).

En este ejemplo capturaremos los botones correspondientes a los números y a los operadores +, - e =, que utilizaremos posteriormente a la hora de automatizar un flujo.

Los elementos capturados del interfaz pueden verse recogidos en la pestaña de *Elementos de Interfaz de usuario*, accesible en el ícono señalado en la Figura 4.16 (ver parte derecha). Este listado permite identificar rápidamente los elementos de interfaz grabados tanto por su nombre como por la imagen asociada al mismo.

La captura de elementos de interfaz puede realizarse igualmente desde cualquier elemento de flujo que necesite referenciarlos. Por ejemplo, en el componente de *clic en el elemento de interfaz de usuario de la ventana* (dentro del grupo de acciones de *Automatización de interfaz de usuario*) también es posible agregar nuevos elementos de interfaz de usuario.

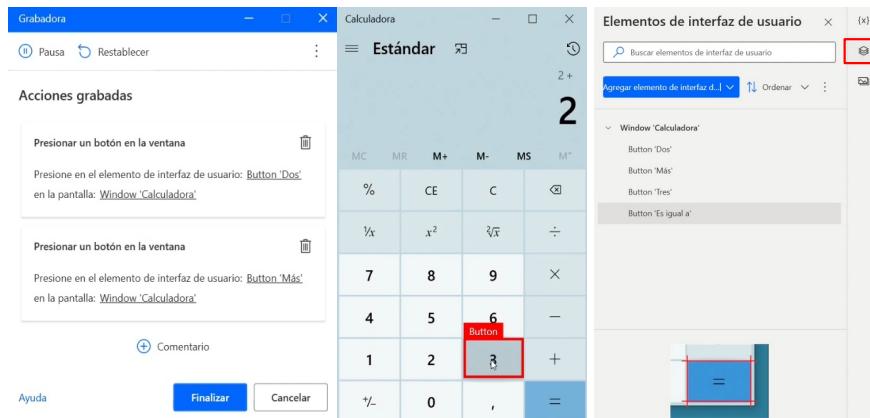


Figura 4.16: Utilización de la grabadora de elementos de interfaz (izquierda) sobre la calculadora estándar de Windows 10. A la derecha se muestra el panel de elementos de interfaz de usuario capturados disponible en la ventana principal de Power Automate Desktop.

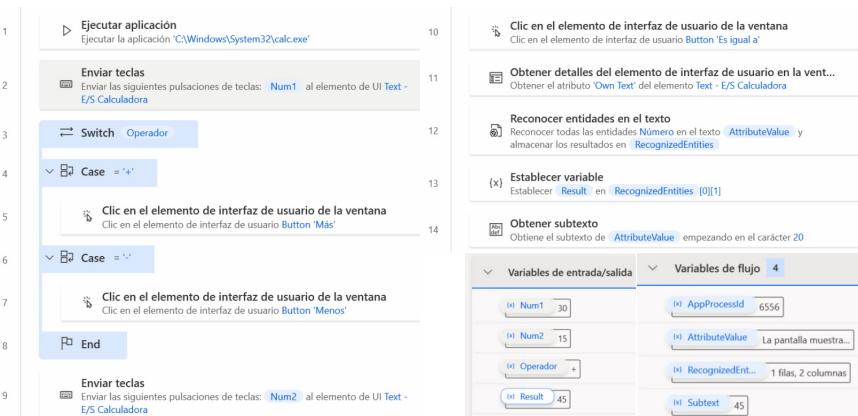


Figura 4.17: Definición del flujo completo del ejemplo con 14 pasos de ejecución. En la parte inferior derecha se recogen las 4 variables de E/S y las 4 variables de flujo utilizadas. El paso 14 del flujo no es necesario realizarlo, es una versión más directa y sencilla que el paso 12 de reconocimiento de entidades.

Cuando se selecciona en el desplegable los elementos disponibles, es posible pinchar en el botón **Agregar elementos** y dinámicamente pueden capturarse los elementos que sean necesarios. En este caso, será necesario mantener pulsada la tecla **Control** mientras pulsamos el botón izquierdo del ratón para capturar los elementos del interfaz que sean necesarios.

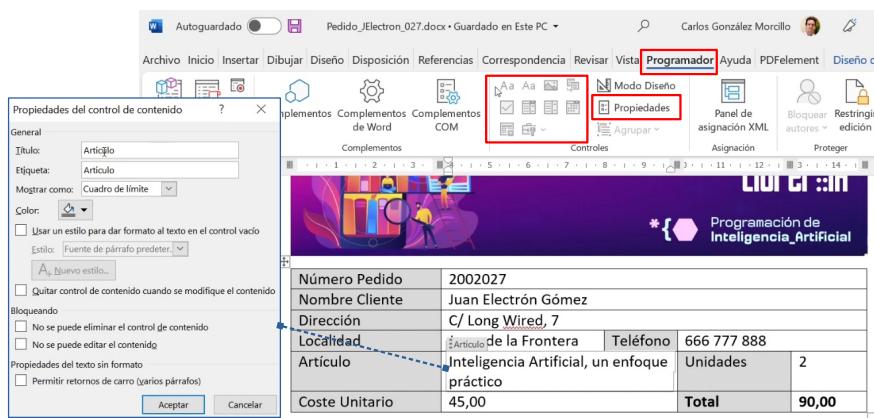


Figura 4.18: Grupo de controles en la pestaña del *Programador* en Microsoft Word para añadir elementos de control de contenido. En esta imagen se muestra el control de cuadro de texto para el *Artículo*.

La Figura 4.17 muestra el ejemplo final de flujo definido empleando 13 pasos. En este ejemplo se utilizan dos variables de entrada (llamadas *Num1* y *Num2*) que son enviadas al componente de entrada de texto de la calculadora de Windows (en la figura se ha llamado *Text E/S Calculadora*), en las líneas 2 y 9. A continuación en el bloque del switch (pasos del 3 al 8) se hace click en los botones + o - según se indique en la variable de entrada al flujo llamada *Operador*¹⁰.

Por último, se recupera el resultado del texto (en la línea 11), que se reconoce llamando al método de *Reconocer entidades en el texto* en la línea 12. Este resultado es finalmente copiado (línea 13) en la variable de salida *Result*.

4.5.8. Ejemplo 7. Procesamiento por lotes

En algunos tipos de aplicaciones de Inteligencia Artificial y Big Data es necesario procesar grandes cantidades de datos tanto estructurados como no estructurados. Existen diversos tipos de arquitecturas que se utilizan para procesar y cargar los datos que posteriormente serán utilizados en las fases de análisis y toma de decisiones.

En las arquitecturas por lotes (o batch) frente a las arquitecturas de trabajo en tiempo real los datos se procesan en intervalos de tiempo específicos, pero con una latencia considerable. Los procesos por lotes suelen comprobar la existencia de datos nuevos periódicamente. Este tipo de procesamiento por lotes es adecuado para manejar tanto grandes volúmenes de datos como aquellos que provienen de sistemas heredados o cerrados donde no es posible gestionar los datos en tiempo real. Con este enfoque se procesan los datos, cargándolos en un *data lake* para su posterior procesamiento.

¹⁰Queda como ejercicio propuesto al lector añadir el soporte para el resto de operadores que quieran considerarse.

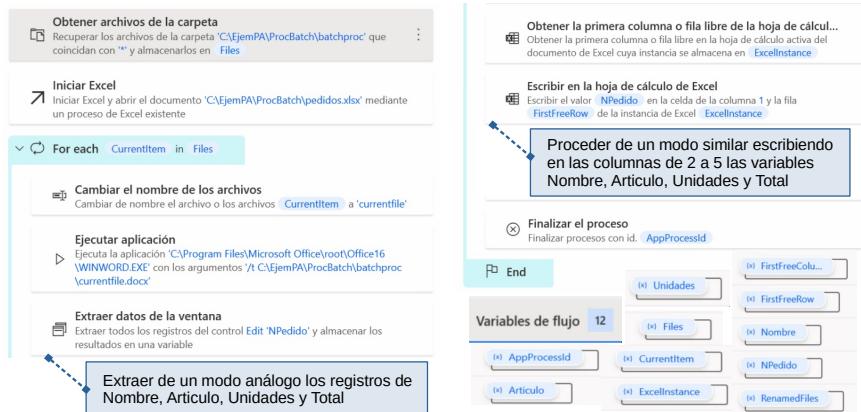


Figura 4.19: Descripción del flujo completo del ejemplo. Se omiten la descripción de pasos similares en la captura y asignación de las variables indicadas en la figura.

En este ejemplo crearemos un sencillo proceso de procesamiento de datos por lotes empleando *Excel* y *Microsoft Word*, en conjunción con las herramientas de captura de eventos que se ha utilizado en el ejemplo anterior. La grabadora de *Power Automate* requiere que los elementos de interfaz tengan un identificador único dentro de la aplicación. De este modo, si los datos provienen de herramientas no estructuradas, requieren que se ayude a la identificación de los campos del formulario.

En este ejemplo se extraerán datos de unos formularios de pedidos guardados en Word que incorporarán controles de contenido en Word. Cada campo del formulario tendrá un identificador único dentro del propio documento, creado desde el menú de la vista de *Programador* en Word (ver Figura 4.18). En este ejemplo todas las entradas se han creado como entradas de *Texto sin formato*, asociando una etiqueta única para cada entrada.

En la Figura 4.19 se muestra el flujo definido para este ejemplo. En el directorio de *batchproc* se sitúan los archivos *.docx* a procesar. Cada archivo contendrá una tabla con los datos a capturar. En el primer paso del flujo se obtiene sobre la variable de flujo *files* la lista de archivos existentes en esa carpeta, y posteriormente se inicia *Excel* abriendo el archivo con la tabla vacía sobre la que se recogerán los datos (ver Figura 4.20). En el bucle principal del flujo se recuperarán los datos de un documento de Word, almacenando los valores del formulario en variables de flujo.

El primer paso que haremos en el interior del bucle de procesamiento es renombrar el archivo actual con un nombre fijo (en este caso, "*currentfile.docx*"). A continuación ejecutaremos Word pasándole siempre el mismo archivo, de modo que la ventana de Word será, a efectos del sistema operativo, siempre la misma. Al acabar el procesamiento de cada archivo cerraremos la instancia de Word de modo que no será posible tener dos instancias de Word con el mismo archivo abierto.



Procesamiento de ventanas. Es importante que el nombre de la ventana a capturar sea siempre el mismo a la hora de utilizar los eventos de escritorio en Power Automate. Esto podemos conseguirlo renombrando el archivo de trabajo actual, de modo que la instancia de Word siempre esté trabajando con el mismo archivo en el interior del bucle.

La inserción de los datos capturados en la hoja de Excel se realiza empleando un módulo llamado *Obtener la primera columna o fila libre de la hoja de cálculo*. Este módulo genera dos variables de flujo: *FirstFreeRow* y *FirstFreeColumn*. La que nos interesa en este ejemplo es la que nos devuelve la primera fila libre. Empleando esta variable de flujo insertaremos en cada fila los valores que han sido capturados anteriormente a partir del documento de Word (ver Figura 4.19).

A	B	C	D	E
1	NumPedido	Nombre	Artículo	Cantidad
2	20022031	Armando Casas Seguras	Advanced Deep Learning	1 115,00
3	2002027	Juan Electrón Gómez	Inteligencia Artificial, un enfoque práctico	2 90,00
4	2021025	Azucena Margarita de la Flor	Lo inevitable	1 68,00
5	2022028	Ramón Agullo López	¿La singularidad está cerca?	3 75,00

Figura 4.20: Resultado final de la tabla Excel tras el procesamiento en lotes de los 4 archivos en formato Word.

4.5.9. Ejemplo 8. Operaciones cognitivas

En el bloque de *Operaciones Cognitivas con Microsoft de Power Automate Desktop* es posible emplear diversos nodos de procesamiento que son proporcionados por los servicios cognitivos de Azure. A continuación veremos cómo obtener las claves asociadas a la cuenta en Azure, así como algunos ejemplos sencillos de utilización.

Uso de cuenta de servicios cognitivos de Azure

En esta sección utilizaremos algunas llamadas a ejemplos específicos de los servicios cognitivos de Azure.

Los servicios cognitivos de Azure (*Azure Cognitive Services*) permiten añadir, mediante una llamada a una API específica, diversas capas de procesamiento basadas en técnicas de IA. Los servicios se dividen en 4 áreas de procesamiento: Voz, Lenguaje, Visión y Decisión. Aunque el servicio tiene un coste considerable, es posible utilizar una prueba gratuita para valorar el potencial de las soluciones.¹¹

Una vez tenemos la cuenta de Azure creada, debemos solicitar una cuenta en los *Cognitive Services* (ver Figura 4.21). En el caso de este ejemplo, la cuenta asociada se llama CGM.

¹¹Es posible solicitar una prueba gratuita de los servicios cognitivos de Azure en la siguiente URL: <https://azure.microsoft.com/es-es/free/cognitive-services/>

A continuación será necesario ir a las opciones del panel de Administración de recursos para obtener una clave para acceder a la API de los servicios. Esta clave permitirá realizar las llamadas a los servicios cloud proporcionados en la plataforma.

The screenshot shows the Microsoft Azure portal interface. At the top, there's a navigation bar with 'Microsoft Azure' and 'Actualización'. A search bar says 'Buscar recursos, servicios y documentos (G+.)'. On the right of the bar are icons for help, refresh, and user information ('Carlos.Gonzalez@uclm.es UNIVERSIDAD DE CASTILLA-LA...'). Below the bar, the main content area has a title 'Servicios de Azure' and a row of service icons: 'Crear un recurso', 'Cognitive Services', 'Azure Virtual Desktop', 'Máquinas virtuales', 'Centro de inicio rápido', 'App Services', 'Cuentas de almacenamiento', 'SQL Database', 'Azure Cosmos DB', and 'Todos los servicios'. Underneath this is a section titled 'Recursos recientes' with a table:

Nombre	Tipo	Última consulta
CGM	Cuenta de varios servicios de Cognitive Services	hace unos segundos
Azure subscription 1	Suscripción	hace 29 minutos
Prueba	Grupo de recursos	hace 31 minutos

At the bottom of the page, there are navigation links: 'Ver todo', 'Navegar', 'Suscripciones', 'Grupos de recursos', 'Todos los recursos', and 'Panel'.

Figura 4.21: Panel de administración de recursos de Azure. La instancia de servicios cognitivos en estos ejemplos se ha creado con nombre «cgm».

En una puesta en explotación de los servicios, se recomienda regenerar la clave periódicamente. Si la regeneración se realiza directamente en código, se puede emplear el par de claves de la plataforma. Mientras se sigue accediendo al servicio con la segunda clave, se puede regenerar la primera (ver Figura 4.22).

Mediante el botón de color azul situado a la derecha de cada una de las claves (ver Figura 4.22) es posible copiar en el portapapeles la referencia a la misma. Esta clave será la que necesitaremos para utilizar los servicios desde Automate. En la parte inferior de la ventana puede verse la URL (extremo) asociada para la conexión externa.

Ejemplo Computer Vision: Describir Imagen

En la Figura 4.23 se muestran los parámetros que son necesarios especificar para utilizar este tipo de nodo disponible dentro del grupo de *Computer Vision*. El parámetro de Archivo de imagen puede ser tanto un archivo disponible localmente como el contenido de cualquier variable resultado de otros nodos de procesamiento.

La salida del nodo es de tipo JSON. El algoritmo devuelve varias descripciones según diferentes características visuales. Internamente el nodo realiza llamadas a la API *Analyze Image*. Cada descripción tiene una puntuación de confianza asociada entre 0 y 1. El resultado es una lista de descripciones ordenadas de mayor a menor confianza.

The screenshot shows the Microsoft Azure Cognitive Services Management Portal. The top navigation bar includes 'Microsoft Azure', 'Actualización', 'Buscar recursos, servicios y documentos (G+)', and several icons. The left sidebar shows a tree view of resources under 'CGM', with 'Claves y punto de conexión' selected. The main content area displays information about keys and endpoints. A note says: 'Estas claves se usan para obtener acceso a la API de Cognitive Services. No comparta sus claves; almacénelas de forma segura, por ejemplo, con Azure Key Vault. También se recomienda regenerar estas claves periódicamente. Solo se necesita una clave para realizar una llamada a la API. Al volver a generar la primera clave, puede usar la segunda clave para continuar accediendo al servicio.' Below this is a 'Mostrar claves' button, followed by fields for 'Clave 1' and 'Clave 2' (both showing masked values), 'Ubicación o región' (set to 'northeurope'), and 'Extremo' (set to 'https://cgm.cognitiveservices.azure.com/').

Figura 4.22: Panel de administración de claves y punto de conexión de los servicios cognitivos.

The screenshot shows the 'Seleccionar parámetros' (Select parameters) section for the Azure Cognitive Services Text Analytics API. It includes fields for 'Ubicación del servidor' (set to 'Norte de Europa'), 'Clave de suscripción' (a placeholder field), 'Proporcionar imagen' (set to 'Desde archivo'), and 'Archivo de imagen' (set to 'C:\EjemPA\TestCognitivo\testimage.jpg'). To the right, there is a thumbnail image of a brown bird standing on a rock. Below the image is a table titled 'Nombre' (Name) and 'Valor' (Value) showing the analysis results:

Nombre	Valor
description	{ 'tags': [bird, animal, outdoor, water, brown, stand...]
requestId	492dd708-5e65-4520-874a-20ea3d795b69
metadata	{ 'height': 1320, 'width': 1920, 'format': Jpg }

At the bottom, there are links for 'Avanzada' and 'Variables producidas' (produced variables), with 'JSONResponse' and 'StatusCode' selected. A dashed blue arrow points from the 'Variables producidas' section to the table.

Figura 4.23: Resultado del ejemplo de descripción de una imagen en Azure Cognitive Services.

Ejemplo Text Analytics: Frases Clave

Este es un ejemplo de uso de las capacidades de comprensión del lenguaje natural para extraer información de texto sin estructura. En este ejemplo (y en los dos siguientes) se emplean llamadas específicas del grupo de análisis de texto de Azure.

La extracción de frases clave identifica rápidamente los conceptos principales del texto, proporcionando una lista de descriptores que resumen el texto en esencias. Por ejemplo, en el texto «*La comida fue deliciosa y el personal maravilloso.*», la extracción de frases clave devolverá como temas: «*comida*» y «*personal maravilloso*».

En la Figura 4.24 se muestra el resultado en JSON de las primeras 7 palabras clave identificadas para un fragmento de texto sobre IA.

La inteligencia artificial (IA) es, en informática, la inteligencia expresada por máquinas, sus procesadores y sus softwares, que serían los análogos al cuerpo, el cerebro y la mente, respectivamente, a diferencia de la inteligencia natural demostrada por humanos y ciertos animales con cerebros complejos. Se considera que el origen de la IA se remonta a los intentos del hombre desde la antigüedad por incrementar sus potencialidades físicas e intelectuales, creando artefactos con automatismos y simulando la forma y las habilidades de los seres humanos. [...]

← Valor variable	
JSONResponse ['documents'][0]['keyPhrases'] (Lista de Valores)	
#	Elemento
0	término inteligencia artificial
1	inteligencia expresada
2	inteligencia natural demostrada
3	ejemplo
4	seres humanos
5	máquinas
6	adaptación flexible
7	agente flexible

Figura 4.24: Ejemplo de análisis de frases clave sobre un fragmento de texto (no mostrado completo en la figura) de Wikipedia.

Ejemplo Text Analytics: Detectar Idioma

La característica de "Detección de idioma" puede detectar el idioma en el que está escrito un documento y devuelve códigos de idioma para diversos idiomas, variantes, dialectos y algunos idiomas regionales. Los parámetros de código de idioma devueltos se ajustan al estándar *BCP-47* y cumplen con los identificadores *ISO-639-1*.¹²

Como puede verse en la Figura 4.25, la variable de salida es un JSON donde, además del código ISO asociado al idioma se devuelve un valor de certeza (entre 0 y 1) con la confianza en la detección realizada.

¹²Puede consultarse la referencia completa de idiomas soportados en: <https://docs.microsoft.com/es-es/azure/cognitive-services/language-service/language-detection/language-support>

Detectar idioma

Invoca el servicio cognitivo de Microsoft para detectar idioma de Text Analytics

Seleccionar parámetros

Ubicación del servidor: Norte de Europa

Clave de suscripción: [REDACTED]

Texto: Esto es una prueba para comprobar qué tal detecta el idioma el servicio de operaciones cognitivas de Microsoft.

Avanzada

Variables producidas: JSONResponse StatusCode

← Valor variable

JSONResponse ['documents'][0]

Nombre	Valor
name	Spanish
iso6391Name	es
score	1

Figura 4.25: Ejemplo de detección de idioma y salida JSON producida.

Ejemplo Text Analytics: Sentimiento

Gracias al uso de este bloque de características es posible automatizar ciertos procesos de negocio interesantes, como averiguar qué piensa la gente de un tema mediante la minería de texto. Clasifica las opiniones como "negativo", "neutral" o "positivo" basadas en la mayor puntuación de confianza que ha encontrado el servicio en el nivel del texto completo. El valor de 0 sería totalmente negativo, 1 implica totalmente positivo y valores cercanos a 0.5 serán neutros.

Seleccionar parámetros

Ubicación del servidor: Norte de Europa

Clave de suscripción: [REDACTED]

Texto: Me dirijo a usted para manifestarle mi agradecimiento por el tiempo y la atención que me brindara durante el día de hoy en la entrevista por la posición de Gerente de Logística y Distribución para Ruidos S.A. de Argentina, posición y corporación en las que poseo sumo interés.

Idioma: ES

Nombre	Valor
documents	[{"id": "text_1", "score": 0.642937898635864}]
errors	[]

JSONResponse (Objeto personalizado)

Figura 4.26: Ejemplo de resultado JSON a partir del análisis del sentimiento de un texto.

4.6. AI Builder

AI Builder es uno de los últimos módulos que han sido incorporados a la familia de productos de *Power Platform*. Proporciona modelos de IA diseñados para optimizar procesos empresariales, de modo que se integren diversas tecnologías accesibles tanto en Power Automate como en Power Apps.

El objetivo de la tecnología de *AI Builder* es permitir la creación de componentes de IA utilizando una interfaz gráfica integrada en *Power Automate* y sin código. De este modo se facilita el acceso a necesidades generales de uso de IA a usuarios sin conocimientos de programación específicos.

Existen dos modos de trabajo para crear una solución de IA: emplear algunos de los modelos de IA predefinidos existentes o crear un modelo personalizado adaptado a las necesidades concretas de un dominio de aplicación. Además, es posible publicar el modelo creado para que pueda integrarse con el resto de elementos de la Power Platform; en concreto veremos cómo utilizarlos dentro de los flujos de *Power Automate*.

Actualmente están disponibles **17 modelos de IA**. Se han utilizado los siguientes iconos en el listado: Modelo Predefinido, Modelo Personalizado, Documental, Imágenes, Datos Estructurados, Texto.

- **Extraer información de facturas** Módulo de procesamiento de facturas. Se encarga de extraer los datos clave de cada factura (identificador de factura, datos del cliente, total y subtotal, etc...) para ayudar a automatizar su procesamiento.
- **Extraer texto en fotos y documentos PDF** Modelo precompilado de reconocimiento de texto basado en reconocimiento de caracteres óptico (OCR) en texto impreso y manuscrito a partir de diversos formatos de archivo (JPG, PNG y PDF entre otros). Este modelo se utiliza en el ejemplo de la sección 4.6.1.
- **Extraer información de recibos** Modelo precompilado específico para el procesamiento de recibos (*tickets*). Es capaz de extraer diversa información de interés: comerciante, dirección, lista de artículo, total, etc.
- **Extraer información de documentos de identidad** Modelo precompilado para extraer información de pasaportes y carnés de conducir. Desafortunadamente en la actualidad solo admite el formato de los EEUU.
- **Extraer información de tarjetas de visita** Modelo precompilado que extrae los datos principales de tarjetas de visita a partir de una imagen (nombre, puesto, dirección postal, email, etc).
- **Extraer información personalizada de documentos** Este modelo personalizado permite extraer diversa información de documentos: texto, tablas, números, casillas (booleanas) y texto escrito a mano entre otros. Este modelo se utiliza en el ejemplo de la sección 4.6.2.

- **Detectar opiniones positivas, negativas o neutras en texto ↗ A** Este modelo precompilado de análisis de sentimiento detecta opiniones positivas, negativas o neutras en el texto de entrada. Este análisis de sentimiento se realiza mediante la tecnología de *Text Analytics* de *Azure Cognitive Services*. Es posible utilizar diferentes idiomas y regiones. Este modelo puede ser de utilidad para analizar opiniones en redes sociales, opiniones sobre productos en web o ayuda al procesamiento automático de email.
- **Clasificar comentarios de clientes en categorías predefinidas ↗ A** Modelo precompilado de clasificación de categorías que puede clasificar texto (de máximo 5.000 caracteres) con los comentarios de los clientes en 6 categorías predefinidas: Problemas, Precio, Servicio al cliente, Documentación, Facturación y Personal.
- **Extraer elementos clave del texto y clasificarlos ↗ A** Modelo precompilado que reconoce datos del texto de interés. Tras la identificación de los elementos clave, posteriormente los clasifica en 25 categorías predefinidas. En la sección 4.6.1 se utiliza este modelo.
- **Extraer las palabras y frases más relevantes del texto ↗ A** Modelo precompilado que identifica los puntos principales (mediante conjuntos de palabras clave) en un texto. El texto está limitado a 5.120 caracteres.
- **Detectar el idioma de un texto ↗ A** Modelo precompilado para identificar el idioma predominante de un documento de texto. El modelo analiza el texto y devuelve el idioma detectado y una puntuación numérica (factor de certeza) desde 0 a 1, donde el 1 indica mayor confianza en la detección. El texto está limitado a 5.120 caracteres.
- **Detectar y traducir en 90 idiomas diferentes ↗ A** Modelo precompilado que detecta y traduce el texto de entrada. El texto está limitado a 5.120 caracteres.
- **Clasificar textos en categorías personalizadas ➤ A** Modelo personalizado para utilizar un modelo de aprendizaje máquina para clasificar en categorías los datos de texto. Estos modelos se entrenan con tus propios datos de texto y categorías personalizadas, de modo que se puedan adaptar a las necesidades de cada organización. La clasificación en categorías es uno de los requisitos habituales en aplicaciones de procesamiento del lenguaje natural, con aplicaciones directas en detección de *Spam*, enrutado automático de peticiones y análisis de opiniones.
- **Extraer entidades personalizadas del texto ➤ A** El modelo identifica elementos clave del texto y los clasifica en categorías predefinidas. Con esta funcionalidad se pueden transformar datos no estructurados en datos estructurados. Es preciso proporcionar al menos 10 ejemplos de datos del texto.

- **Predecir resultados a partir de datos históricos**  Este modelo analiza patrones en datos históricos asociándolos a los resultados. El modelo permite resolver cuestiones de tres tipos: (1) Predicciones binarias, con respuesta sí/no. Ejemplo: ¿Es un buen candidato este cliente para extenderle la licencia demo? (2) Predicción de resultado múltiple. Ejemplo: ¿Qué productos le interesarían a un cliente (3) Predicción numérica. Ejemplo: ¿En cuántos días llegará el envío del proveedor?
- **Detectar objetos personalizados en imágenes**  Modelo personalizado de detección de objetos basado en imágenes. Las imágenes que se utilizan en el entrenamiento deben estar en formato JPG, PNG o BMP, con un tamaño máximo de 6MB y una resolución mínima de 256 píxeles en cada dimensión. Este modelo se utiliza en el ejemplo de la sección 4.6.3.
- **Clasificar imágenes en función del contenido**  Este modelo utiliza la herramienta de escritorio gratuita Lobe para construir el modelo de aprendizaje máquina. Una vez entrenado, el modelo se sube a *AI Builder* para el uso en *Power Automate* y en *Power Apps*. Existen diversos ámbitos de aplicación, como identificar errores en cadenas de fabricación, enviar una alerta cuando una estantería esté casi vacía, notificar contenido subido no apropiado, etc.

En términos generales, los pasos para añadir un flujo de *AI Builder* son los siguientes:

1. **Seleccionar un tipo de modelo de IA.** En la sección anterior se enumeran todos los disponibles actualmente en la plataforma.
2. **Conectar los datos.** Esta conexión puede realizarse tanto con flujos de datos de Power Automate como con Microsoft Dataverse.
3. **Adaptar el modelo de IA.** Dependiendo del tipo de modelo, en muchos casos es posible ajustar parámetros de trabajo para mejorar el resultado obtenido.
4. **Entrenar el modelo de IA.** En este proceso automático se enseña al modelo de IA cómo resolver el problema de su negocio basándose en los datos que se han proporcionado y en la propia personalización realizada en la etapa anterior.
5. **Utilizar el modelo de IA.** Una vez realizados los pasos anteriores el modelo está listo para crear soluciones que resuelven los problemas específicos.

4.6.1. Ejemplo 1. Uso de modelos predefinidos

En este ejemplo utilizaremos algunos de los modelos predefinidos de *AI Builder*. Gracias a estos modelos los flujos pueden añadir funcionalidades transversales de IA sin tener que recopilar datos y entrenar los modelos.

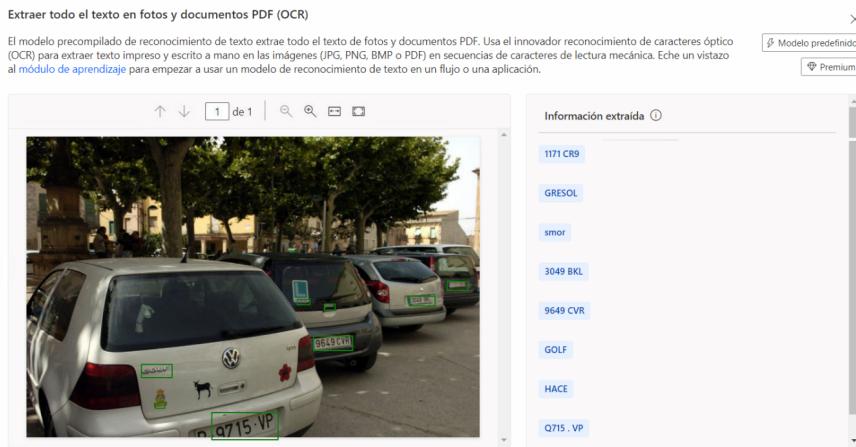


Figura 4.27: Utilización del modelo precompilado de detección de texto basado en OCR. El sistema detecta correctamente texto en situaciones muy complicadas.

Extraer texto en fotos y documentos PDF

El modelo de extracción de texto basado en OCR de AI Builder funciona con un altísimo rendimiento en diversas configuraciones. Como se puede ver en el ejemplo de la Figura 4.27, la detección es muy precisa, incluso en situaciones con texto inclinado, baja resolución (como la matrícula de la derecha) o incluso con texto manuscrito.

Extraer elementos clave del texto y clasificarlos

En el ejemplo de la Figura 4.28 el modelo precompilado detecta algunos de los principales tokens del texto no estructurado, obteniendo elementos que pueden ser correctamente clasificados para su posterior análisis.

Utilización de modelos predefinidos en Power Automate

El uso de modelos predefinidos de Power Automate es similar a la de cualquier otro componente de la plataforma. En la categoría de *AI Builder* (ver Figura 4.29) se encuentran disponibles todos los modelos para su uso directo¹³.

¹³Hay que recordar que estos modelos obedecen a la suscripción *Premium* y tienen un coste asociado al uso del servicio.

Extraer elementos clave del texto y clasificarlos en categorías predefinidas

El modelo precompilado de extracción de entidades es un modelo listo para usar que identifica los elementos clave del texto y luego los clasifica en categorías predefinidas como Edad, Ciudad, Fecha y hora, Organización, Nombre de la persona y más...



The screenshot shows the AI Builder interface for entity extraction. On the left, there's a text input area with the placeholder "Escriba su propio texto" containing a sample email message. Below it are two buttons: "Analizar texto" and "Ejemplo aleatorio". To the right, a table titled "Entidades" lists extracted entities like "Gómez" (PersonName), "6%" (Percentage), "Toledo" (City), "año 2021." (DateTime), "www.errealcubo.es" (URL), and "Ribogerto Robles Ruperto" (PersonName). A progress bar at the bottom indicates 394/5000 words analyzed.

[Ver documentación](#) [Usar en un flujo](#) [Usar en una aplicación](#)

Figura 4.28: Utilización del modelo de extracción de elementos clave del texto con su clasificación.

This screenshot shows the Power Platform AI Builder configuration interface. At the top, there are tabs for "Todo", "Integrado", "Estándar", "Premium", "Personalizar", and "Mi portapapeles". Below, under "Desencadenadores", there are four actions: "Condición Control" (Control Condition), "Analizar opinión positiva o negativa en el texto" (AI Builder), "Clasifica el texto en categorías con el modelo estándar (vista previa)" (AI Builder), and "Clasificar el texto en categorías con uno de sus modelos personalizados" (AI Builder). To the right, three specific actions are shown in sequence: "Cuando llega un nuevo correo electrónico (V3)", "Traducir texto en otro idioma", and "Enviar correo electrónico (V2)". Each action has its own configuration screen with various fields and options.

Figura 4.29: La utilización de los modelos preconfigurados del AI Builder es directa, las variables de salida se insertan directamente en el resto de controles de flujo de la plataforma.

4.6.2. Ejemplo 2. Extracción de información de documentos

En este ejemplo entrenaremos un modelo personalizado adaptado a las necesidades concretas de una empresa ficticia. En este caso, utilizaremos unos documentos en PDF de Contoso, la empresa que utiliza Microsoft para los ejemplos de Power Platform.

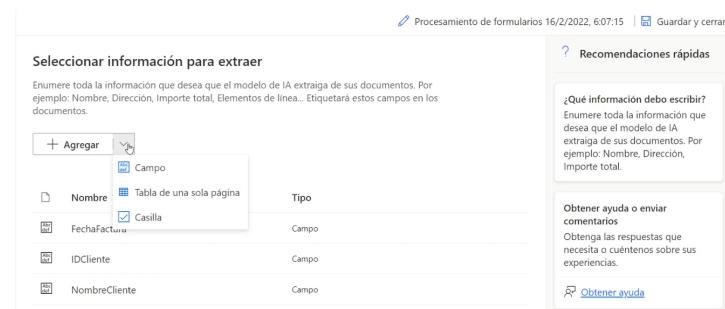


Figura 4.30: Agregación de los diferentes campos, tablas y casillas que serán reconocidas en el documento.

En cierto modo, este ejemplo muestra un ejemplo de un dominio de aplicación más cercano a la problemática que se definió en el ejemplo de la sección 4.5.8. En este caso, los PDFs se entrada son documentos no estructurados (en el sentido estricto del término), ya que las tablas no cuentan con ningún tipo de identificador único para cada campo.

Una vez subidos los 5 documentos mínimos que hacen falta para comenzar el procesamiento, el primer paso consiste en definir los diferentes elementos a extraer de cada documento. La Figura 4.30 muestra los tres tipos de datos que se pueden extraer: *campos de texto*, *tablas* (de una página) o valores booleanos (llamados *casillas*).

Fechalimp...	Formacion	C	D	Subtotal
10/12/2020	Leadership Training	\$675		
10/13/2020	Leadership Training	\$675		
10/14/2020	Leadership Training	\$675		

Figura 4.31: Segmentación de la tabla (de una sola página) que será identificada en los documentos.

En este ejemplo extraeremos solo algunos de los campos disponibles en los documentos originales; algunos campos de texto (como el identificador de la factura, el nombre del cliente y el total de la misma), tres valores booleanos (correspondientes al tipo de trabajo realizado) y tres columnas de la tabla principal (con la Fecha de cada evento, el concepto principal de cada fila y el subtotal de la misma).

La Figura 4.31 muestra el proceso de segmentación de la tabla en el proceso de etiquetado de documentos. En el proceso de entrenamiento se definen las filas y columnas de cada documento, etiquetando en la cabecera los campos que se quieren extraer. En la zona derecha de la ventana se muestra la previsualización de cómo quedaría la tabla una vez reconocida. Las columnas marcadas como C y D no serán recuperadas en la fase de procesamiento. De igual modo, la fila de la cabecera no será reconocida (se ha ignorado en la fase de entrenamiento en los 5 documentos de muestra).

Por último, una vez que se ha entrenado el modelo se puede probar el rendimiento del mismo. La figura 4.32 muestra el resultado de ejecutar la prueba con un documento diferente a los utilizados en el entrenamiento. Como se muestra en la figura, en todos los casos la confianza en la detección es del 99 %. En realidad, la identificación de los documentos es muy buena porque los datos obedecen a la misma plantilla de generación. Aún así, la combinación con los módulos de OCR en la plataforma hacen que, incluso trabajando con documentos escaneados, el resultado sea muy efectivo.

The screenshot displays the AI Builder interface for document processing. At the top, it shows the company name "Contoso, Ltd." and the document type "Project Statement". Below this, there's a "Sold To" section with "Woodgrove Bank" and its address "12 Sycamore Lane, Ash, NC, 28137". A green box highlights the "Woodgrove Bank" text. Next is a "Puntuación de confianza" (Confidence Score) of "99 %". A green box highlights the confidence score. To the right, there's a "Receipt Statement" section with a date "8/22/2020" and receipt number "No. 1357". A green box highlights the date. Below these sections is a table titled "Formacion" (Training) with three rows: "Speaker Training" (9/3/2020), "Leadership Training" (9/4/2020), and "Compliance Training" (9/5/2020). The "Subtotal" for each row is \$3,150, \$4,050, and \$1,350 respectively. A green box highlights the "Formacion" title. At the bottom left, there's a table titled "Training Date" with three rows: "9/3/2020", "9/4/2020", and "9/5/2020". The "Description" column lists "Speaker Training", "Leadership Training", and "Compliance Training". The "Price" column shows "\$5,500" for Speaker Training and "10%" for Leadership and Compliance training. The "Discount" column shows "10%" for Leadership and Compliance training. The "Line Total" column shows "\$3,150", "\$4,050", and "\$1,350" respectively. A green box highlights the "Formacion" title in this table. A callout box at the bottom center says "Haga clic para ver la tabla detectada" (Click to view the detected table).

Figura 4.32: Resultado de la identificación de uno de los documentos de prueba.



Figura 4.33: Izquierda: Selección de los objetos para que los detecte el modelo. Centro y derecha: Proceso de etiquetado de imágenes.

4.6.3. Ejemplo 3. Detección de objetos

La detección de objetos puede ayudar a agilizar diversos procesos de negocio. En este ejemplo forzaremos el uso con una tipología de imágenes que no es la que habitualmente se utiliza en este ámbito de aplicación.

Como indican en la documentación del módulo, es importante cargar suficientes imágenes para entrenar el modelo de IA. Al menos se requieren 15 imágenes por objeto; con menor número de imágenes hay mucho riesgo de que el modelo solo aprenda ruido. A mayor número de imágenes pertinentes el modelo funcionará mejor.

Por otro lado, los datos deben estar equilibrados. Es decir, el número de imágenes asociadas a cada tipo de objeto debe ser similar. Si se tienen 1000 imágenes de un objeto y solo 100 de otro, el conjunto de datos de entrenamiento no estará equilibrado¹⁴.

La Figura 4.33 muestra dos de los pasos de entrenamiento del modelo. A la izquierda se definen las categorías de objetos que serán detectadas por el modelo. A la derecha se muestra el paso principal de entrenamiento del modelo con el etiquetado manual de las imágenes. En la zona derecha del interfaz se muestra el recuento de cada categoría de objeto reconocido.

Una vez entrenado el modelo, es posible realizar una prueba manual y, si es satisfactoria, proceder a su publicación para ser utilizado en los flujos de Power Automate.

¹⁴En este caso, es posible que el modelo sea mejor reconociendo uno de los objetos

4.7. Sistemas Blockchain

Para explicar los sistemas basados en Blockchain es necesario primero introducir el concepto de los sistemas descentralizados frente a los sistemas centralizados.

En los **sistemas centralizados**, el servicio se construye a partir de un conjunto de nodos centrales que gestionan el procesamiento de la red. Esto facilita mucho la gestión del entorno de explotación, pero también tiene muchos aspectos negativos. Si hay algún fallo en los servidores centrales, el sistema deja de funcionar.

Por su parte, los **sistemas descentralizados** se basan en nodos donde cada uno de los equipos gestionan tanto los datos como el procesamiento, coordinándose de forma colectiva para la toma de decisiones. De este modo, si alguno de los nodos deja de funcionar, el resto de los nodos puede seguir operando sin ningún problema.

Sobre esta idea de descentralización se basan multitud de tecnologías actuales: criptomonedas, NTFs, contratos inteligentes e inteligencia artificial descentralizada entre otros.

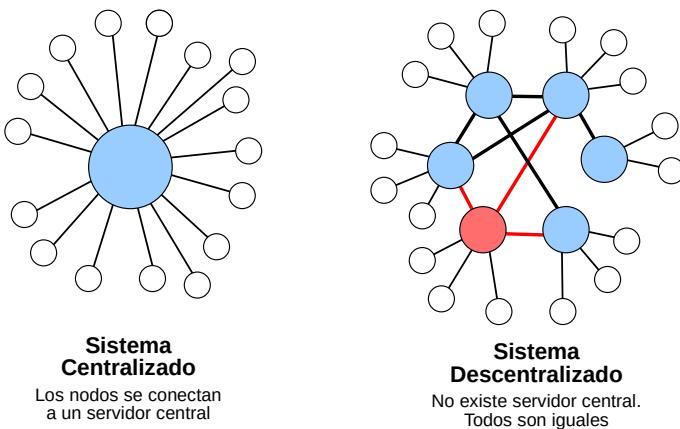


Figura 4.34: Sistemas centralizados Vs. Sistemas descentralizados. En caso de fallo de algún nodo de servicio (representado en rojo, en la figura de la derecha), los sistemas descentralizados siguen operando sin problemas.

4.7.1. Dificultades de los sistemas descentralizados

Aunque las ventajas de este enfoque descentralizado son claras, las dificultades para llevarlo a cabo también son igualmente relevantes. Hay que pensar en este tipo de redes como aquellas en las que todos los nodos se comportan *como iguales*. No hay la figura de ningún nodo que actúe de “*coordinador*” de la red.



No te fíes ni de tu sombra. Imaginemos por ejemplo que en un grupo de amigos queremos decidir qué película vamos a ver al cine. Lo más sencillo será establecer un sistema de votación, de modo que la película más votada será la que veremos todo el grupo. En un sistema descentralizado el método de votación tendrá que diseñarse de modo que nadie atesore todos los votos y tenga la capacidad de realizar el conteo final de votos. Si se hiciera así, tendríamos un sistema centralizado: ¿qué ocurriría si ese "amigo" quisiera salirse con la suya y hacer trampas en el proceso para ver la película *romanticona* que sólo le apetece ver a él?

Una de las principales ventajas que se persiguen con los sistemas descentralizados es que no es necesario confiar en nadie, el propio diseño de la red y el método de operar hace que el sistema funcione de por sí.

Este tipo de redes donde todos actúan y sirven como iguales se denominan redes *Peer-to-peer* (o P2P). Una red P2P muy extendida actualmente es BitTorrent. Cuando alguien descarga un archivo de esta red, el archivo no está alojado completamente en los servidores centrales de la red: hay un conjunto de usuarios que tienen fragmentos del archivo. Todos los nodos de la red comparten y descargan trozos de ese archivo, trabajando *como iguales*. Un archivo .torrent contiene la localización de las diferentes piezas que forman el archivo de destino. Estos pequeños fragmentos (normalmente de 256KB) se localizan en un número de máquinas diferentes, y se descargan paralelamente. Cuando se han descargado todos los fragmentos se puede ensamblar el archivo final, que ya será utilizable.

Basándose en esta idea de P2P, en 2009 se publicó el paper titulado "*Bitcoin: A peer-to-peer electronic cash system*", firmado por el pseudónimo de Satoshi Nakamoto. En este paper, que todavía no se sabe quién (o quienes) escribieron, el autor proponía una solución para el intercambio de dinero entre personas sin ninguna institución central que controle las transacciones.

De forma similar a como en BitTorrent los usuarios de la red comparten *trozos* de los archivos. En BitTorrent estos *fragmentos* se corresponden con el contenido de los archivos a descargar. En Bitcoin los usuarios comparten *trozos* de las transacciones económicas que se están realizando en la red. En realidad es un registro de movimientos de dinero que se realizan entre los usuarios de la red. La moneda no es más que ese enorme registro de transacciones. Por ejemplo, si el Sr. Satoshi me compra un chalet que vendo en la playa por 50 bitcoins, esa transacción (*Satoshi → Carlos [50BTC]*) primero se comprobaría que puede ser realizada (es decir, si el Sr. Satoshi dispone de esos 50 BTCs) y después se registraría, siendo propagada a todos los nodos de la red que mantienen el histórico de transacciones.

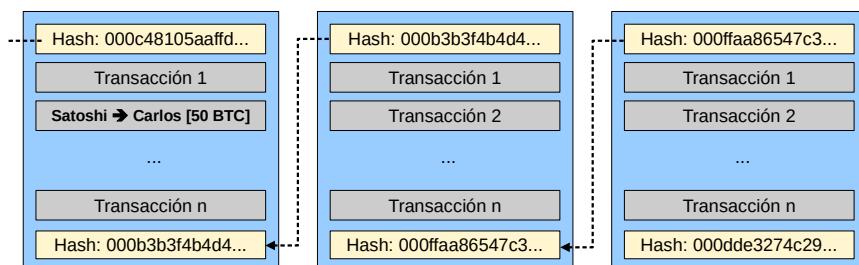


Figura 4.35: El concepto de bloque como conjunto de transacciones agrupadas con un enlace al anterior bloque que fue incluido en la cadena.

4.7.2. El concepto de bloque

El problema viene cuando se tienen que registrar todas estas transacciones en el mismo orden. Es muy fácil que, con tal cantidad de máquinas interviniendo a la vez, todo se desincronice, ya que cada máquina tiene su propio reloj interno. ¿Se pueden sincronizar relojes de modo que todos tengan exactamente la misma hora? Es muy complicado sin tener que depender de nodos centrales en la red. En el enfoque de *blockchain* queremos evitar cualquier nodo central de la red, por lo que un servidor de tiempo central no es posible en este enfoque. Una caída de este servidor de tiempo central y la red dejaría de operar de un modo correcto. Tampoco es posible que los nodos se comuniquen el tiempo entre ellos, porque el hecho de tener que hacerlo por Internet y de un modo absolutamente distribuido entre todos, hace que la sincronización perfecta no sea viable.

Una posible idea es agrupar un conjunto de transacciones que ocurren en intervalos de tiempo cercanos en un bloque. El trabajo de cada nodo será estar escuchando todas las transacciones, verificando que son correctas (que el que realiza la transacción tiene fondos suficientes para realizarlos¹⁵) e ir agrupándolas en un bloque.



El trabajo de un nodo básicamente es escuchar transacciones, verificarlas y crear un paquete de transacciones. Este paquete se denomina **Bloque**. Cuando el nodo lo tiene listo, lo compartirá con el resto de miembros de la red. De este modo todos los nodos tendrán el mismo registro de transacciones.

El problema viene en que otros nodos estarán haciendo justo el mismo trabajo, y tendrán también bloques con transacciones verificadas. Estos nodos también querrán compartir su registro de transacciones en un bloque con el resto de nodos de la red. ¿Cómo se llega entonces a un consenso de qué bloque aceptar?. ¿Cuál de ellos se adopta como el *oficial*?

¹⁵Este trabajo en realidad es sencillo, basta con analizar el histórico de transacciones realizadas y validar que esta se puede llevar a cabo

Podemos pensar que una votación podría ser útil en este caso. Si cada ordenador votara eligiendo un bloque, resolveríamos el problema. Sin embargo, esta aproximación no es válida. Es muy sencillo para una máquina hacerse pasar por un alto número de ordenadores, generando IPs virtuales por ejemplo. Un ataque de este tipo permitiría generar miles de votos sin esfuerzo, corrompiendo así la integridad del sistema de voto.

4.7.3. Qué es el *Proof of Work* y ajustes

¿Cómo se resuelve en Blockchain entonces? Cada nodo tiene que resolver un problema criptográfico *complejo* de modo que para votar tiene que certificar que es una máquina de verdad (y que no se está haciendo pasar por múltiples máquinas). Esta votación basada en trabajo (ganarte la participación garantizando que eres una persona y no varias) es lo que se conoce como “Prueba de trabajo” (*Proof of Work*).



La prueba de trabajo en Blockchain se basa en el algoritmo SHA-256 que utiliza funciones *hash* criptográficas. Una función *hash* es un algoritmo que transforma un conjunto de datos diversos (en este caso un bloque de transacciones), en un único valor de longitud fija: el "hash". En el caso de SHA-256, la longitud de la cadena es de 256 bits. Este hash es una cadena única para esa entrada. Cualquier modificación mínima; un único bit en la entrada, genera un hash totalmente distinto. El valor hash calculado es único y es irreversible, y puede ser utilizado para verificar la integridad de copias de un dato original.

En Blockchain se utiliza este algoritmo de modo que la prueba de trabajo es la siguiente: en cada bloque hay un espacio reservado para incluir un número. Una vez construido el bloque con las transacciones, se posible cambiar este número mágico de modo que el resultado del Hash SHA-256 dé como resultado un valor hash con un determinado número de ceros al principio. El *enigma criptográfico* se reduce entonces a conseguir averiguar con qué número mágico incluido en el bloque conseguimos como salida un valor hash con un número determinado de ceros. La única forma de resolverlo es ir probando números hasta que alguno dé la solución correcta.

Dificultad de conseguir premio

La dificultad representa lo difícil que es encontrar el hash necesario para minar un nuevo bloque en la blockchain asociada a la prueba de trabajo. La dificultad representa el número de combinaciones diferentes que pueden darse para que los mineros adivinen el hash. A mayor dificultad, mayor será el trabajo que tendrán que hacerlo para crear un nuevo bloque. En el caso de bitcoin la dificultad se ajusta cada 2 semanas para que los nuevos bloques se añadan, de media, cada 10 minutos. Si más mineros se unen a la red Bitcoin, contribuyendo así con más poder de cómputo para el hash, la dificultad aumentará y se ajustará para que los mineros descubran el hash en aproximadamente cada 10 minutos.

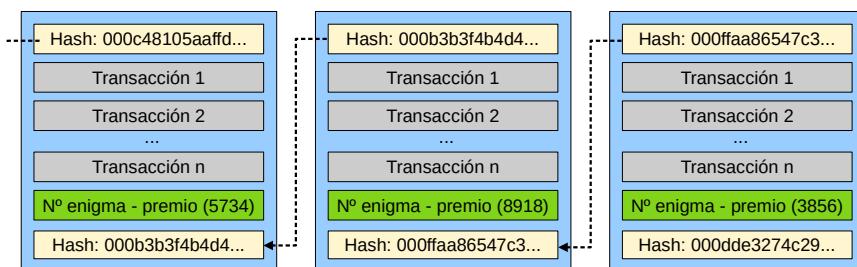


Figura 4.36: El concepto de bloque actualizado, incluyendo el número necesario para resolver el enigma criptográfico de modo que la nueva dirección de bloque tenga el número de ceros necesario al inicio.

La dificultad se ajusta cambiando el número de ceros que deben generarse consecutivamente al inicio del hash. A mayor número de ceros, mayor dificultad para que *toque la lotería*.

En <https://www.blockchain.com/explorer> se puede ver la cadena de bloques de Blockchain, que es totalmente pública. En el momento de escritura de este documento, el último bloque publicado es el 727280, con el siguiente Hash compuesto por 19 ceros al inicio de la cadena.

000000000000000000003cf34a68fa9d3b4bec76e9a555ba292a44495ce5ab4a0

Nótese la diferencia con el primer bloque (minado el 9 de enero de 2009) con el siguiente Hash compuesto por 8 ceros al inicio.

00000000839a8e6886ab5951d76f411475428afc90947ee320161bbf18eb6048

Esta prueba de trabajo se utiliza para elegir quién es el ganador. El primero que encuentra el bloque puede añadirlo al registro general: el nodo habrá **minado** un bloque.

El modelo de consenso de Nakamoto

Lo que ocurre es que puede ocurrir que a la vez, en el mismo instante de tiempo, haya más de un minero que encuentre la secuencia correcta del hash y quiera añadirlo al registro general. Esto implicaría que hay dos versiones diferentes del registro con bloques que son correctos (verificados y validados), pero que son diferentes. ¿Qué hacer entonces?

No se pueden admitir los dos, porque una misma transacción podría quedar registrada dos veces (esto equivaldría en el ejemplo anterior a que se ha pagado dos veces por el chalet en la playa). En el paper original de Bitcoin se desempata de la siguiente forma. Se mantienen las dos versiones “*provisionalmente*” y se sigue trabajando en ambas, hasta que se rompa el empate porque una de las dos cadenas

se hará más larga. En un momento, en una de las cadenas se incluirá un nuevo bloque y el algoritmo indica que nos quedamos con la cadena más larga, y pasará a ser la cadena oficial. La cadena más corta se quedará cortada en ese punto y los bloques que no se usen se quedarán huérfanos.



El modelo de consenso de Bitcoin, llamado **Consenso Nakamoto**, utiliza bloques ordenados secuencialmente ponderado en importancia (según la prueba de trabajo) para determinar la cadena más larga que define el estado actual.

El minero que consigue incluir un bloque que se añade a la cadena oficial recibe como recompensa una transacción a él como forma de pago. El minero que añade el bloque incluye, en el propio bloque que añade, esta transacción como forma de pago, ganando bitcoins como pago por el proceso de minado. Esta dinámica de pago es la principal responsable de que existan estas granjas dedicadas al minado de bloques.

En el momento de la escritura de este documento, la recompensa por minado de un bloque en Bitcoin es de 6,25BTC, una cantidad nada desdoblable teniendo en cuenta que el cambio de BTC está a más de 35.000€. A esta cantidad fija se le suman las comisiones por cada una de las transacciones. Un negocio interesante en definitiva.

4.7.4. Seguridad en Blockchain

Gracias al uso de SHA-256, si se cambia un solo bit del bloque de entrada, se modifica su resultado. Esto es de especial utilidad, ya que si alguien intenta cambiar algo en el bloque, su Hash cambiará y no será válido. Además, el diseño de este bloque incluye como parte de su contenido, además de las transacciones y el número de verificación (que permite generar los ceros necesarios en la hash de salida), también incluye el identificador del bloque anterior, de modo que cada bloque contiene un “*puntero*” al previo, al último bloque que fue aceptado; encadenado así cada bloque con el anterior, construyendo de este modo una Cadena de Bloques (Blockchain). La cadena de bloques es una estructura de datos que encadena cada bloque con su anterior por su Hash. Por el propio diseño de la red es inmutable. Si quieras modificar un bloque, estarás alterando su hash y, en cadena, la de todos los bloques que vayan a continuación. Si quisieras hacerlo, tendrías que volver a generar bloques válidos y competir con la cadena oficial (hasta ahora más larga), que seguirá creciendo. Y la condición de aceptación según el algoritmo es quedarse siempre con la cadena más larga. Esto solo lo podrá conseguir el atacante si tuviera más potencia de cálculo que el resto de integrantes en la red; a esta condición se le conoce como ataque del 51 %. En este punto el atacante tendría más poder computacional que el resto de nodos de la red y podrían comenzar a imponer sus modificaciones.

A la escala de redes que se manejan hoy en día en Bitcoin (con cerca de medio millón de nodos) o Ethereum es absolutamente improbable que ocurra. Si la red es suficientemente grande tenemos garantizada la seguridad e inmutabilidad, sin necesidad de que organismos centrales velen por esta garantía.

4.7.5. ¡Blockchain es una estructura de datos descentralizada!

El blockchain no es más que una estructura de datos y un modo de almacenamiento de esta estructura de datos descentralizado, de modo que se almacena en los nodos de todos los participantes de la red. Es un soporte de información que permite agregar datos. Esta cadena de bloques cumple una serie de propiedades garantizadas muy interesantes. Esta garantía viene dada matemáticamente por el propio diseño de la estructura de datos, no es una “confianza” que el usuario deba tener en un tercero. Estas propiedades son:

- **Pública.** Cualquier persona del mundo puede *fiscalizar* la lista de transacciones que han ocurrido en el tiempo. Que sea pública no implica que no pueda ser confidencial. Por ejemplo, en Bitcoin aunque la información es pública ofrece protección criptográfica para ocultar la identidad de quién hace las transacciones (no es posible saber qué persona está enviando dinero a quién).
- **Immutable.** Los datos no se pueden cambiar. No es posible que un dato que haya sido incluido en una cadena de bloques pueda ser modificada en el tiempo por ningún otro usuario. Cuando algo está en la blockchain, es definitivo. No hay comando *deshacer*, el `Control Z` no funcionará.
- **Validada.** La nueva información que se va agregando está validada, usuarios maliciosos no pueden alterar el uso añadiendo transacciones erróneas. Esta validación dependerá del uso concreto que se haga de la cadena de bloques. Por ejemplo, en Bitcoin si un usuario quiere gastar 2 bitcoins, la validación consiste en comprobar que ese usuario dispone de ese dinero antes de validar la operación.

Blockchain puede usarse con otras tecnologías, no tiene por qué ser algo totalmente independiente. Es habitual combinarlo con otras tecnologías *frontend* y *backend*. Por otro lado, cada usuario tiene asociada una identidad digital. Es mejor que no pierdas la clave privada, pues perderás todo lo que tengas en Blockchain. Como principal beneficio asociado al uso de esta tecnología, cabría destacar los derivados de la descentralización. No es necesario confiar en los nodos de la red, ni hay que preocuparse por un punto de fallo principal.

Aunque las ventajas son suficientes, también existen algunas dificultades asociadas al uso de Blockchain, y que deben estudiarse antes de decidir utilizar esta tecnología. Algunas de las más relevantes son:

- **Es costoso.** Si tratas de usar Blockchain donde no es necesario, estarás incurriendo en un coste importante.

- **No permite datos privados.** Sí permite privacidad, pero no datos privados (las cadenas de bloques son públicas). Esto puede solventarse añadiendo niveles adicionales de encriptación sobre los datos públicos.
- **No es centralizado.** Si se requiere centralización las cadenas de bloques no encajan en el modelo de desarrollo.
- **Tamaño de los datos.** Almacenar archivos de gran tamaño no es recomendable porque más computación significa más energía y más dinero. Es posible no obstante sacar los datos a almacenar en un sistema de almacenamiento externo con referencias únicas a los mismos que garanticen que no han sido modificados.

Para finalizar esta sección, a continuación analizaremos algunos de los usos y casos de éxito más relevantes de esta tecnología. **Bitcoin** es la criptomoneda por excelencia. Como hemos estudiado desde el inicio del capítulo, es el fundamento de Blockchain. Pero blockchain es una tecnología que se puede aplicar a muchos más entornos, como veremos a continuación.

Contratos inteligentes. En lugar de almacenar transacciones monetarias, en Blockchain se almacenan *scripts* (piezas de código) que se van a ejecutar si se cumplen una serie de condiciones. En el caso de que se cumplan las condiciones se ejecuta el contrato. Aquí el contrato está público, disponible dentro de los nodos de la Blockchain y se ejecutará automáticamente si las condiciones se cumplen.

NFTs (Non Fungible Tokens). Este término se ha puesto totalmente de moda en los últimos meses. La cadena de bloques certifica la propiedad de ciertos activos digitales. Los artistas digitales ponen a disposición obras y los interesados podrán adquirir su propiedad.

4.8. Introducción a Ethereum

Ethereum se denomina con frecuencia como *La computadora global*. Desde el punto de vista puramente informático puede describirse como una máquina de estado determinista prácticamente ilimitada [AW18]. Esta máquina consta de un estado accesible globalmente, y una máquina virtual que aplica cambios a ese estado.

En una descripción funcional más práctica se puede describir *Ethereum* como una infraestructura descentralizada de código abierto que ejecuta programas, que se denominan contratos inteligentes (*Smart Contracts*). Los contratos inteligentes son códigos que se escriben utilizando la *Máquina Virtual de Ethereum (EVM)*, que se encarga tanto de automatizar como de ejecutar estos acuerdos en un libro de contabilidad inmutable.



Ethereum es el Bitcoin del código. Es una cadena de bloques de desarrollador@s, construida por programador@s para programador@s. Ethereum utiliza una tecnología similar a la de Bitcoin, y cuenta con su propia moneda llamada Éter (*Ether* en inglés, con acrónimo ETH). No hay mucha diferencia entre Bitcoin y Ethereum salvo por los contratos inteligentes. Ethereum aplica blockchain para los contratos inteligentes y almacena la lógica (código) de un modo inmutable.

Al igual que ocurre con Bitcoin, el Éter se divide a su vez en unidades más pequeñas: Wei, GWei y Finney. La unidad más pequeña es el Wei (puede considerarse calderilla digital): 1 ETH equivale a 10^{18} Weis. Como quizás imagines, 1 GWei es equivalente a 10^9 Weis, por lo que 1 ETH es equivalente a 10^9 GWei. Por último, 1 ETH equivale a 1000 Finneys (1 Finney es unidad de miliéter).



Curiosidad: El nombre del miliéter (Finney) proviene de **Hal Finney**, maestro del ciberpunk y el primer humano en recibir un Bitcoin de Satoshi.

Como hemos visto anteriormente, Bitcoin fue el primer blockchain, pero Bitcoin estaba pensado como dinero digital. Como la especificación y el diseño de Bitcoin estaba en abierto, en 2013 el programador Vitalik Buterin decidió empezar a desarrollar Ethereum sobre la blockchain de Bitcoin para tener una plataforma de desarrollo de aplicaciones descentralizadas y colaborativas¹⁶. Construir directamente sobre el blockchain de Bitcoin implicaba aceptar muchas restricciones que intencionalmente se habían puesto en esa red: un conjunto muy limitado de transacciones, tipos de datos y tamaños de almacenamiento muy restrictivos. A la especificación inicial del proyecto se unieron otros entusiastas (como Gavin Wood), que definieron la cadena de bloques de propósito general que permitiera al programador abstraerse de todos los detalles subyacentes de redes P2P, algoritmos de consenso, etc. En Julio de 2015 se lanzó la primera versión de Ethereum y fue minado el primer bloque. La *computadora global* comenzó a funcionar¹⁷.

	Bitcoin	Ethereum
Uso	Se utiliza para pagos	Se utiliza para código / lógica
Definición	Moneda digital	Plataforma de contratos inteligentes
Tiempo procesar bloques	10 minutos	17 segundos

Tabla 4.1: Diferencias entre Bitcoin y Ethereum

¹⁶En la actualidad, Ethereum cuenta con su propia blockchain que elimina algunas limitaciones de la Blockchain original de Bitcoin

¹⁷En el artículo titulado „A Prehistory of the Ethereum Protocol“, el propio Vitalik explica muchos detalles del inicio de la red <https://vitalik.ca/general/2017/09/14/prehistory.html>

A diferencia de Bitcoin que cuenta con un lenguaje de script extremadamente limitado (básicamente evaluación de condiciones booleanas sobre el gasto asociado a la moneda), en *Ethereum* el propósito principal es el de crear una cadena de bloques programable de propósito general que ejecuta una máquina virtual capaz de desplegar código de cualquier nivel de complejidad, definiendo un lenguaje *Turing completo*.

4.8.1. Ethereum: Un blockchain de propósito general

A diferencia de Bitcoin que solo rastrea el estado de la titularidad de la moneda, en Ethereum se rastrean también las transiciones de estado de un almacén de datos de propósito general. Este almacén puede contener cualquier valor arbitrario que pueda expresarse con una tupla *clave-valor*.

Ethereum tiene una memoria que almacena tanto código como datos, empleando blockchain para rastrear cómo cambia la memoria a lo largo del tiempo. Existen dos diferencias importantes con las computadoras *tradicionales*: (1) los cambios de estado en Ethereum siguen las reglas del consenso y (2) el estado se distribuye globalmente entre todas las máquinas que forman parte de la red.

Las transiciones de estado en Ethereum se procesan en la Máquina Virtual de Ethereum (EVM). Los programas de la EVM se denominan *Contratos Inteligentes* y se escriben en lenguajes de alto nivel (como Solidity), y se compilan posteriormente a código binario para su ejecución en la EVM.



El estado de Ethereum se almacena en cada nodo como una base de datos que contiene las transacciones y el estado del sistema. Toda esta información se serializa en una estructura de datos hash del tipo *Árbol de Merkle-Patricia*.

Ethereum actualmente utiliza el mismo modelo de consenso que Bitcoin (consenso Nakamoto). Sin embargo, ya se han anunciado planes para migrar a un sistema de votación ponderada de pruebas colaterales (llamado Prueba de Participación - *Proof of Stake*). Con la **Prueba de Participación** los mineros comprometen una participación de moneda digital antes de poder validar las transacciones. De este modo, la capacidad de un nodo minero para validar bloques depende del número de monedas que haya puesto en juego y del tiempo que lleve validando transacciones. Cuantas más monedas posean, más poder tendrán para minar. El minero elegido para cada transacción se elige aleatoriamente mediante un algoritmo ponderado que tiene en cuenta la potencia relativa de los mineros. Mediante este enfoque de pruebas de participación se mitiga el impacto medioambiental que tiene la prueba de trabajo del algoritmo original.

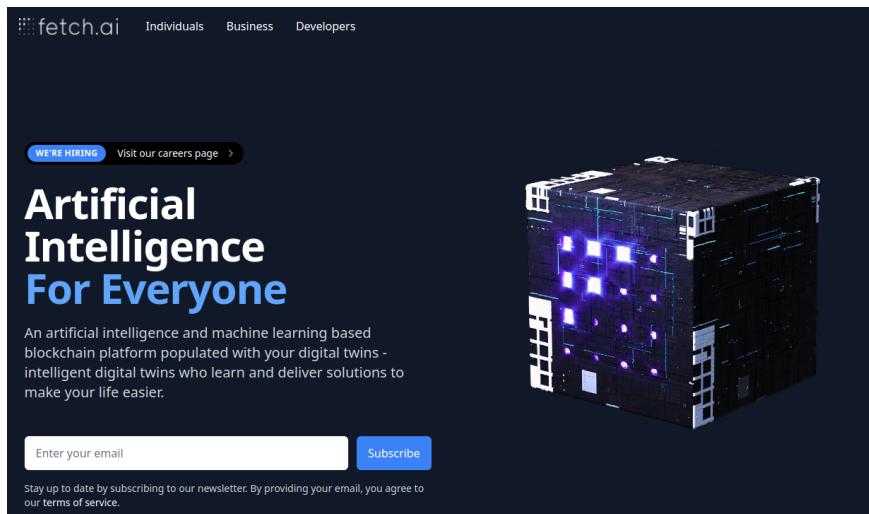


Figura 4.37: Página web del proyecto Fetch.ai, centrada en crear una plataforma de machine learning sobre blockchain.

Algunos proyectos basados en Ethereum

A continuación se enumeran algunos proyectos que se basan en *Ethereum*. Se recomienda visitar la web oficial de cada proyecto para tener una visión más detallada de las características de cada plataforma.

- **Gnosis.** Según palabras de sus creadores, esta plataforma permite crear, comerciar y mantener de forma segura activos digitales en Ethereum. Básicamente hace predicciones de comercio, de modo que las predicciones correctas reciben los tokens que estaban en juego. Web: <https://gnosis.pm/>.
- **Golem.** La plataforma Golem funciona como un supercomputador descentralizado. Hace cálculos fuera de Blockchain, verificando los resultados online. Esto permite a los usuarios alquilar (si te sobra en tu ordenador) o contratar potencia de cálculo. Web: <https://golem.network/>.
- **Fetch.ai.** Plataforma de aprendizaje automático descentralizada creada en 2021 con el objetivo de convertirse en el ecosistema de soluciones IA sobre blockchain de referencia a nivel mundial. Los agentes de Fetch.AI son capaces de tomar decisiones en nombre de las partes interesadas (como individuos, empresas privadas o gobiernos), integrándose fácilmente en sistemas existentes. Se basa en una nueva criptomoneda: el token FET, que permite el pago de comisiones y proteger la red contra el mal uso de los recursos que están disponible en la misma. Web: <https://fetch.ai/>

4.8.2. Aplicaciones Descentralizadas (DApps)

Originalmente Ethereum se concibió como una cadena de bloques programable de propósito general. En los últimos años la visión del proyecto ha crecido para convertirse en una plataforma de desarrollo de DApps, mucho más ambiciosas que los contratos inteligentes.



Una **Aplicación Descentralizada (DApp)** es una aplicación web que se desarrolla sobre infraestructuras de servicios de código abierto descentralizados y de redes P2P. Puede verse como un contrato inteligente (en *Blockchain*) + interfaz web (*frontend*).

Las DApps se configuran así como una de las tecnologías naturales de evolución de la World Wide Web, introduciendo descentralización basada en protocolos P2P. Un ejemplo de tecnología ya disponible es la biblioteca de JavaScript `web3.js`, la cual permite enlazar aplicaciones JavaScript que se ejecutan en el navegador con la *Blockchain* de Ethereum. Esta biblioteca incluye una red de almacenamiento P2P llamada *Swarm*, así como el servicio de mensajería P2P llamado *Whisper*. Estos tres paquetes permiten a cualquier desarrollador construir DApps en la web. En el caso de acceso a archivos hipermedia descentralizados, merece la pena estudiar IPFS, un protocolo P2P para hipermedia que se ha convertido en el estándar.

Muévete rápido y rompe cosas

A diferencia de otros proyectos más conservadores, el *mantra* de Ethereum se caracteriza por evolucionar rápidamente la plataforma incorporando novedades a expensas de no conseguir compatibilidad con versiones anteriores. Esto implica que el código debe reconstruirse en el caso de que no sea compatible con versiones posteriores de la plataforma¹⁸. La autonomía y nivel de estabilidad que se requiere para escenarios complejos de descentralización se conseguirá con seguridad en los próximos años. Hasta que los interfaces sean estables, la innovación será el impulsor de la plataforma.

4.8.3. Desarrollando en Solidity

Ethereum permite utilizar varios lenguajes de alto nivel que pueden utilizarse para escribir contratos y producir código binario compatible con la EVM. De entre todos, el más utilizado con diferencia es Solidity.

Solidity es un lenguaje de alto nivel orientado a contratos. Tiene una sintaxis similar a JavaScript. Está tipado de manera estática y permite trabajar con herencia y bibliotecas externas.

¹⁸Esta actualización no implica solo *actualizar* los contratos inteligentes, sino que puede ser algo más doloroso: desplegar nuevos contratos, migrar usuarios, etc.

Los contratos de Ethereum son programas que se ejecutan dentro de la máquina virtual EVM. Se crean mediante una transacción especial que envía su código binario para que se registre en la cadena de bloques. Cuando están en la cadena de bloques, tendrán asociada una dirección de Ethereum (igual que las carteras). Cada vez que alguien envía una transacción a la dirección del contrato, se ejecuta en la EVM, con la transacción como entrada. Las transacciones que se envían a las direcciones de un contrato pueden contener moneda (Éter), datos o ambos. Si contiene Éter, se depositará asociado al saldo del contrato. Si contiene datos, es posible especificar el nombre de una función y pasar argumentos a la misma.

El gas es una forma de combustible (virtual) que emplea la EVM para contabilizar el consumo de recursos informáticos asociados a cada contrato. El límite de gas indica la cantidad máxima de gas que puede consumir una transacción o un bloque.

Una vez desarrollado el contrato en Solidity, el siguiente paso es compilarlo. El compilador de Solidity puede utilizarse en multitud de IDEs, cuenta con un compilador independiente y actualmente se integra sin problema en los principales entornos de desarrollo multiplataforma. La forma más rápida de comenzar es directamente en el IDE de Remix, que solo requiere un navegador web.

En la URL <https://remix.ethereum.org/> se accede al Remix IDE oficial de Ethereum. Cuando se abre Remix por primera vez, se cargan varios contratos de prueba en la carpeta *Contracts*. En esta introducción no vamos a usarlos, y directamente podemos obviarlos.



Chrome y Remix. El navegador recomendado para ejecutar Remix es Google Chrome. En el momento de la escritura de este documento se han probado otros navegadores con resultados incorrectos. Los ejemplos de esta sección se han ejecutado en la versión 97.0 oficial.

El interfaz que presenta Remix puede verse en la Figura 4.38. En la zona izquierda de la pantalla se muestran las opciones del menú principal, donde se puede acceder al explorador de archivos, las diferentes versiones del compilador de Solidity y versiones de la EVM, así como opciones directas de despliegue del contrato inteligente en diferentes redes y, por último, conexión con el depurador. A continuación a la derecha se encuentra la zona de pantalla reservada para el menú secundario, que depende de la sección del menú principal que se ha elegido. En la Figura 4.38 se muestra el menú secundario asociado al explorador de archivos, donde se puede definir el espacio de trabajo.

La zona principal de la pantalla se reserva al editor de código que tiene soporte de remarcado de sintaxis en Solidity. Por último, la zona inferior de la pantalla se reserva para la consola, donde se mostrarán los resultados de las operaciones realizadas sobre los bloques y, en el caso de conectar con alguna red pública, nos añadirá los enlaces a los mismos para que puedan ser inspeccionados con *Etherscan*.

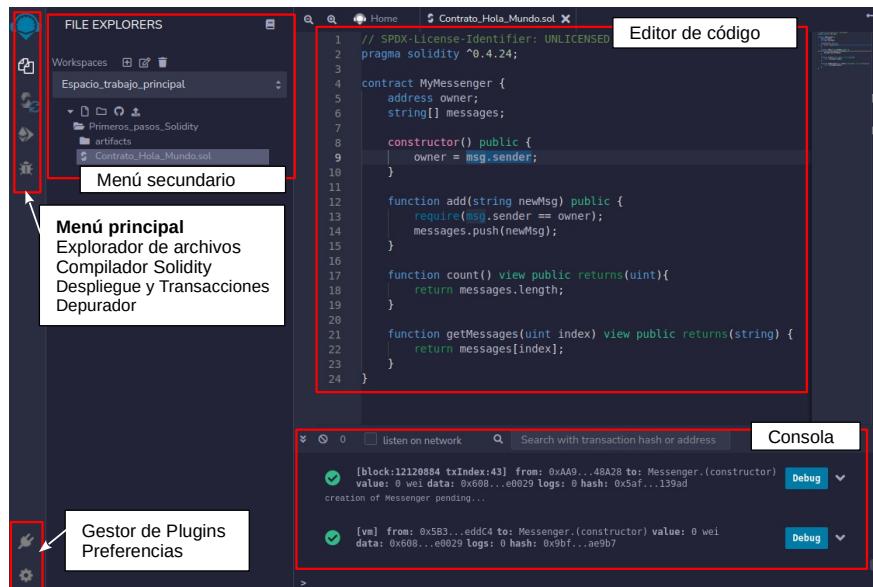


Figura 4.38: Descripción general del interfaz de Remix.

¡Hola Contrato! en Solidity

En esta sección analizaremos un contrato muy sencillo en Solidity¹⁹ (ver Listado 4.1). La primera línea indica el tipo de licencia del código empleando el identificador corto de SPDX. Dependiendo de la versión del compilador que utilicemos, si no se incluye esta primera línea de comentario, podemos obtener un *Warning* en el proyecto.



La especificación de SPDX (*Software Package Data Exchange*, un estándar internacional ISO/IEC 5962:2021) contiene una lista de las licencias más habituales utilizadas en el software y contenidos libres y abiertos. Este listado, disponible en <https://spdx.org/licenses/>, incluye un identificador corto estandarizado, el nombre completo y una URL permanente para cada licencia.

En la línea ② se utiliza la directiva para el compilador mediante la palabra reservada `pragma`, que le indican cómo operar con el código fuente. En este caso se indica que el código fuente que viene a continuación está escrito en Solidity, en concreto con la versión 0.4.0 o cualquiera que sea totalmente compatible con esta (cualquiera que sea anterior a la 0.5). Esto permite garantizar que el contrato no se

¹⁹La documentación completa de la última versión de Solidity con la referencia del lenguaje puede consultarse en <https://solidity-es.readthedocs.io/es/latest/index.html>.

va a comportar de un modo diferente con versiones posteriores del compilador. De hecho, si se intenta compilar con una versión posterior a la 0.5, veremos que faltan ciertas palabras reservadas que se incorporaron posteriormente al estándar. Todo contrato debe comenzar indicando la versión del compilador que se utilizará.

También es posible especificar un rango de versiones que estamos soportando. Por ejemplo, se podrían utilizar operadores estándar de comparación para indicar rangos en los números de versión como `pragma solidity >=0.4.0 <0.6.0;`

En el entorno de Remix, cuando el IDE detecte la versión del compilador que se ha indicado en la directiva *pragma*, elegirá la última versión disponible que sea totalmente compatible con el rango de versiones elegidas. Por ejemplo, si la directiva *pragma* es la que aparece en el listado 4.1, Remix elegirá la versión 0.4.26, que es la última compatible antes de la 0.5.0.

El código de Solidity encapsula la funcionalidad en *contratos*. Un contrato no es más que un conjunto de funciones y datos (que definen el estado). Cada contrato reside en una dirección específica en la blockchain de Ethereum. Un contrato es el bloque más básico de lo que podemos definir en Ethereum. Mediante la palabra clave *contract* podemos definir un contrato. En el ejemplo del listado 4.1, el contrato se ha llamado *Messenger* y se define entre las líneas **④-24**. Si aparece, solo se permite un constructor por contrato (la sobrecarga no está soportada).

Listado 4.1: Ejemplo de contrato sencillo en Solidity

```
1 // SPDX-License-Identifier: GPL-3.0-or-later
2 pragma solidity ^0.4.0;
3
4 contract Messenger {
5     address owner;
6     string[] messages;
7
8     constructor() public {
9         owner = msg.sender;
10    }
11
12    function add(string newMsg) public {
13        require(msg.sender == owner);
14        messages.push(newMsg);
15    }
16
17    function count() view public returns(uint){
18        return messages.length;
19    }
20
21    function getMessages(uint index) view public returns(string) {
22        return messages[index];
23    }
24 }
```

En el cuerpo del contrato tendremos primero que definir las **variables de estado**. Cada variable necesita que se defina el tipo de un modo estático. En la línea **④5** de declara la variable llamada `owner` que es de tipo `address` (un tipo especial de datos para almacenar direcciones de Ethereum, que no es más que un valor de 20 bytes). Por su parte, la línea **④6** declara un array dinámico de strings llamado `messages` que almacenará una lista de mensajes.



Privado por defecto. Estas variables de estado son por defecto privadas (salvo que se indique después del tipo la palabra reservada `public`). Si la variable de estado se hace pública podrá ser accesible públicamente. Sin esta palabra reservada otros contratos no pueden acceder a esta variable.



Sobre la visibilidad en Solidity. Como se indica en la documentación del lenguaje: todo lo que está definido dentro de un contrato es *visible* para todo el mundo (observadores externos). Emplear el modificador `private` solo impide que otros contratos puedan modificar la información, pero la información en sí siempre será visible para todo el mundo fuera de la blockchain.

A diferencia de otros lenguajes de programación, en Solidity el constructor es una función opcional (puede haber contratos sin constructor). Si no se especifica constructor, internamente el compilador creará uno por defecto. En el constructor se inicializan las variables de estado del contrato. En la línea **④8** se utiliza la palabra clave `constructor` para definir esta función especial. El constructor debe ser público de modo que pueda ser invocado por cualquiera. En nuestro ejemplo solo es necesario inicializar la variable del propietario (línea **④9**).

En Solidity existen variables especiales de ámbito global que siempre están disponibles y que proporcionan información sobre la blockchain. En la línea **④9** se está utilizando `msg.sender` que devuelve la dirección del remitente de la llamada actual²⁰.

En el contrato de este ejemplo se han definido tres funciones: `add` (líneas **④12-15**), `count` (líneas **④17-19**) y `getMessages` (líneas **④21-23**). En el caso de este ejemplo, todas son públicas, y solo la primera modifica los datos (las funciones `count` y `getMessages` utilizan el modificador `view` para indicar que son de *solo lectura*; no van a modificar el estado. El modificador `view` en este caso hace que la ejecución de la función no tenga un coste de gas al ejecutarse externamente.

²⁰Puedes consultar el listado completo de variables y funciones especiales (*Bloque y propiedades de transacciones*) en: <https://solidity-es.readthedocs.io/es/latest/units-and-global-variables.html>

Una función puede especificarse como *external*, *public*, *internal* o *private*. Las funciones *external* pueden llamarse desde otros contratos (forman parte del interfaz del contrato) y no pueden llamarse internamente. Las funciones *public* también forman parte del interfaz del contrato y pueden llamarse internamente o mediante mensajes. Las *internal* y *private* solo pueden llamarse desde el contrato; las *internal* permiten llamarse desde contratos derivados del actual, pero las *private* no permiten acceder a contratos derivados del actual.

Las funciones pueden, opcionalmente, devolver uno o más resultados mediante parámetros de salida. Es necesario indicar los tipos de los datos devueltos en el prototipo de la función indicando en el interior de la definición de returns (ver líneas $\Theta 17$ y $\Theta 21$). Es posible además, utilizar nombres para los parámetros devueltos. Por ejemplo, en el listado 4.2 se la función devuelve dos valores empleando nombres simbólicos (*sum* y *dif*).

Listado 4.2: Retorno de múltiples valores con nombres de parámetros

```

1 pragma solidity ^0.5.0;
2 contract Test {
3     function testResult() public view returns(uint sum, uint dif){
4         uint a = 10; // Variable local
5         uint b = 5; // Variable local
6         sum = a + b; // Mismo nombre que parámetro returns
7         dif = a - b; // Mismo nombre que parámetro returns
8     }
9 }
```

Continuando con el ejemplo del listado 4.1, la llamada de la línea $\Theta 13$ hace uso de la función *require*. La función *require* debe utilizarse para asegurar que se cumplen ciertas condiciones, como entradas, variables de estado del contrato, o para validar los valores de retorno de las llamadas a contratos externos. Debe utilizarse al inicio de la función para validar permisos, valores que proporciona el usuario (por ejemplo `require(input>10);`), validar condiciones de estado antes de ejecutar algo (por ejemplo `require(balance[msg.sender]>=amount);`), etc.

Para terminar de analizar el ejemplo, las llamadas realizadas sobre el array dinámico de mensajes (*messages*) que se llaman en las líneas $\Theta 14$, $\Theta 18$ y $\Theta 22$ son autoexplicativas, añaden un nuevo elemento al array de almacenamiento dinámico mediante *push* (línea $\Theta 14$), obtienen el número de elementos del array (línea $\Theta 18$) o acceden al contenido de una posición del array (línea $\Theta 22$).

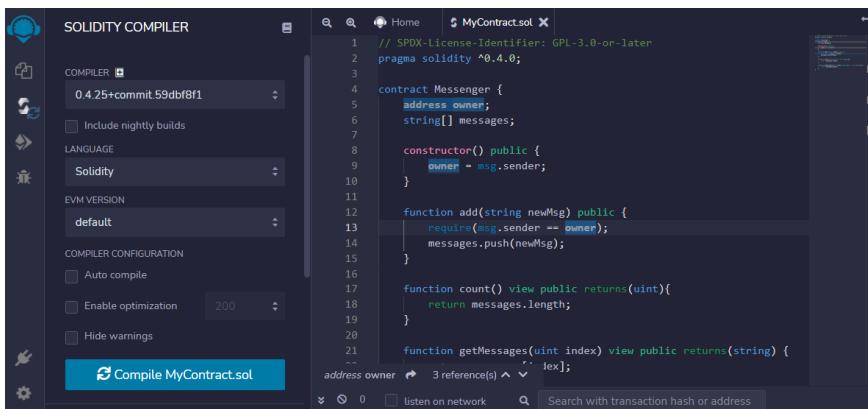
Compilación del ejemplo

Una vez introducido el código fuente del ejemplo en Remix, procederemos a su compilación. Para ello, accederemos al menú de compilación de Solidity (segundo ícono del menú principal, como se muestra en la Figura 4.39). Como vemos, la versión del compilador se ajusta automáticamente según la directiva *pragma* que indicamos la inicio del código fuente (en este caso, la 0.45).



Ojo con la versión del compilador: se pueden desplegar contratos con versiones del compilador posteriores a la 0.4.12. No deberías usar versiones del compilador anteriores porque alguna funcionalidad y plugins no se ejecutarán correctamente.

Pinchando en el botón azul de *Compile MyContract.sol* se compilará el ejemplo. Si no hay ningún error, aparecerá el tick verde al lado del ícono del menú principal. En otro caso, en la zona del submenú de compilación aparecerá una caja en rojo con el error encontrado, y un ícono de exclamación en la línea de código fuente afectada.



```
// SPDX-License-Identifier: GPL-3.0-or-later
pragma solidity ^0.4.0;

contract Messenger {
    address owner;
    string[] messages;

    constructor() public {
        owner = msg.sender;
    }

    function add(string newMsg) public {
        require(msg.sender == owner);
        messages.push(newMsg);
    }

    function count() view public returns(uint){
        return messages.length;
    }

    function getMessage(uint index) view public returns(string) {
        address owner
        3 reference(s) ^ lex];
    }
}
```

Figura 4.39: Compilación del ejemplo en Remix.

Es posible activar la optimización del código en el checkbox "*Enable optimization*". El optimizador de código intenta reducir el tamaño del código (y su coste asociado de ejecución, de modo que necesite menos *gas* para desplegar el contrato, así como las llamadas externas). En versiones muy recientes de Solidity, es recomendable activar la optimización. El valor numérico asociado a la optimización indica el número de iteraciones de optimización a realizar (por defecto 200). Un número bajo del parámetro de optimización (por ejemplo, *optimización = 1*) hará hincapié en que el coste de despliegue del contrato sea más barato (aunque el coste de ejecución de las transacciones sea mayor). Sin embargo, números mayores intentará que el coste de las transacciones sea menor aunque el tamaño del contrato crezca.

Si la compilación ha sido correcta, podemos pasar al siguiente paso de desplegar el contrato.

Despliegue local

Para utilizar el módulo de despliegue local (ver Figura 4.40) es necesario haber compilado anteriormente el contrato. En el desplegable nombrado **CONTRACT** debe aparecer el nombre del contrato que hemos compilado anteriormente. En el caso de la Figura 4.40 el contrato es *Messenger*. Si no hay ningún contrato en ese selector, debes compilarlo siguiendo las instrucciones de la sección anterior.

El primer selector de este submenú permite elegir el entorno de despliegue. Existen tres opciones a la hora de desplegar el contrato:

- **Javascript VM:** Con dos versiones, London y Berlin que funcionan igual. Todas las transacciones se ejecutan en una máquina virtual de pruebas localmente en tu browser. No hay datos persistentes en ningún sitio; si recargas la página los datos locales se perderán. Este entorno es ideal para hacer las primeras pruebas y será el que utilicemos en esta sección.
- **Injected Web3:** En este caso Remix se conectará con algún proveedor Web3 sobre el que injectará las transacciones (por ejemplo, como veremos más adelante, con el uso de Metamask). Este entorno será el que utilizaremos para hacer pruebas de despliegue reales sobre la red de pruebas Ropsten en la subsección posterior de este documento.
- **Web3 Provider:** Remix se conectará con un nodo remoto proporcionando una determinada URL.

En el campo de **Account** tenemos la lista de cuentas (carteras con su correspondiente saldo) disponibles en el entorno. En el caso de pruebas en el entorno *Javascript VM* se crean 15 cuentas ficticias con 100 ETH cada una para facilitar la realización de pruebas.

El campo de **Gas Limit** indica la cantidad máxima de gas que se permitirá usar por todas las transacciones creadas en Remix. Es habitual dejar el valor por defecto.

El campo **Value** indica la cantidad de ETH (usando la fracción correspondiente Wei, GWei, etc...) que se enviará al contrato o a una función que requiera de pago. Este valor no es el valor de gas.

El botón naranja **Deploy** permite desplegar el contrato. Si el despliegue ha sido correcto, aparecerá una instancia del contrato desplegado en la parte inferior izquierda de la ventana. Pinchando sobre la flecha ↗ a la izquierda del nombre del contrato podemos acceder al interfaz de llamado de las funciones disponibles, con botones habilitados para cada una, así como entradas de datos si la función lo requiere.

Podemos probar a llamar a la función *add* y pasarle como parámetro (entre comillas) "*Hola contrato*". Después ejecutaremos la función *count* y veremos que ya hay un mensaje disponible. Si cambiamos en **Account** la cuenta con la que estamos trabajando y tratamos de ejecutar *add* veremos que salta una excepción, pues el

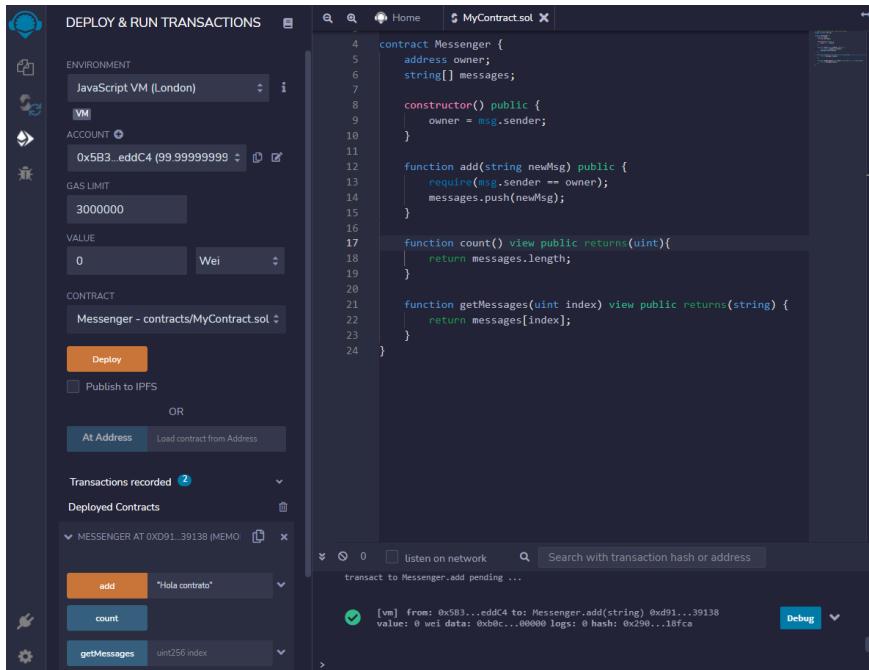


Figura 4.40: Opciones disponibles en Remix para desplegar el contrato y realizar transacciones.

usuario que ha desplegado el contrato no es el mismo que está intentando ejecutar esta función. Si comentamos el *require* del código fuente (y compilamos y volvemos a subir) veremos que ahora sí deja ejecutar la función *add* desde cualquier cuenta.

Creación de una cartera con Metamask

Metamask es la cartera por excelencia que se utiliza para almacenar los Éter y para gestionar la conexión con las diferentes redes disponibles en Ethereum, así como la conexión con IDEs como Remix. Está disponible como extensión de Google Chrome. Accediendo a <https://metamask.io/>, es posible instalar directamente la extensión en el navegador. Aunque existen otras alternativas igualmente adecuadas como Jaxx, MyEtherWallet (MEW) o Emerald, por simplicidad, seguridad y uso por parte de la comunidad de desarrolladores, en esta sección utilizaremos Metamask.

Una vez instalada, si ya dispones de una cartera en Metamask puedes *Importar la cartera* o si es la primera vez, pincharás en *Crear una cartera* (ver Figura 4.41). En el proceso de creación, Metamask te mostrará un conjunto de 12 palabras que definen la clave privada de esta cartera. Este conjunto de 12 palabras debe mante-

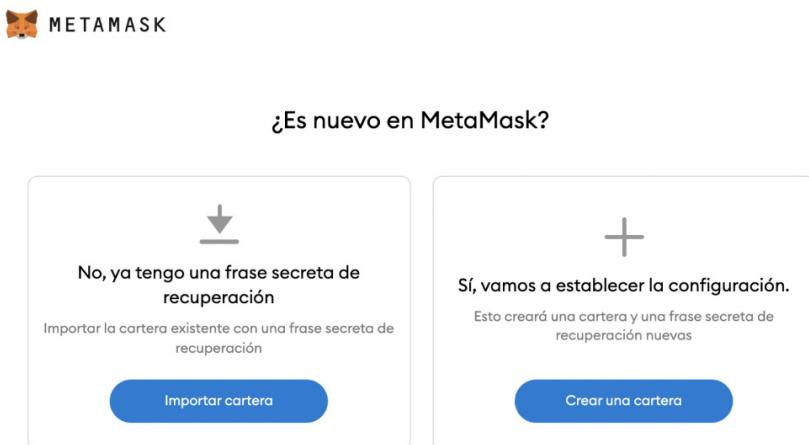


Figura 4.41: Opciones de creación de una nueva cartera o importación de cartera existente en Metamask.

nerse en secreto para que nadie pueda tener acceso a la cartera. Si solo vas a utilizar esta cartera para probar con Éter de las redes de prueba, no pasa nada (no tendrás valor), pero si en algún momento usas la cartera para algo más relevante, no querrás que nadie tenga acceso a tu cartera de Metamask.

Cuando Metamask se encuentra en ejecución, es posible abrir la ventana principal accediendo directamente al ícono de la extensión de Chrome, situado a la derecha de la barra de direcciones (ver ícono con la cabeza del zorro en la parte superior de la Figura 4.42).

Por defecto Metamask accederá a la red principal de Ethereum, donde el Éter tiene un valor real. No es recomendable hacer pruebas en la red principal de Ethereum (salvo que seas asquerosamente ric@), porque las transacciones deben pagarse con ETH real, que tiene un coste real y consecuencias reales para tu bolsillo. Para evitar estas consecuencias reales, la comunidad de Ethereum dispone de un conjunto de redes de pruebas públicas que utilizan exactamente la misma tecnología, pero donde el ETH no tiene un valor. Pinchando sobre el nombre de la red (situado en la parte superior derecha de la pantalla) se puede cambiar la red. La primera vez será necesario pinchar sobre *Show/hide test networks* y activar la opción *Show test networks* para que aparezca el listado de redes de prueba disponibles. En la actualidad existen 4 redes de prueba públicas: Ropsten, Kovan, Rinkeby y Goerli.



Aunque en las redes de prueba el ETH no tiene un valor real hay individuos que se dedican a atesorar ETH de prueba (*sin valor!*), por lo que el proceso de obtención de estos ETHs se ha limitado para evitar estos comportamientos.

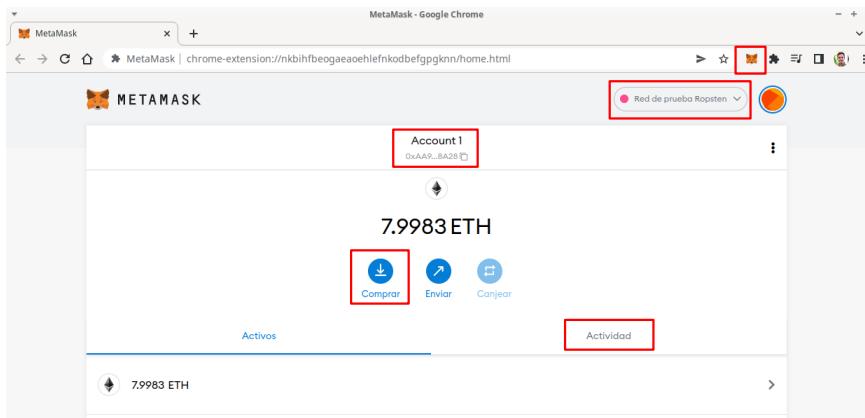


Figura 4.42: Aspecto de la ventana principal de Metamask con algunas de las opciones principales destacadas.

En estos ejemplos utilizaremos Ropsten porque el proceso de *compra* (gratuita) de ETHs en esta red es la más sencilla de todas²¹.

En la zona central del interfaz se muestran los ETHs que el usuario tiene disponibles en esa red, así como la dirección Ethereum de la cartera (en el campo Account). Para comprar ETH en la red de Ropsten, pincharemos en *Comprar* (ver Figura 4.42) y elegiremos *Obtener ether* dentro de la opción *Obtener ether de un faucet para Ropsten*. Un *faucet* no es más que un grifo que "gotea" pequeñas cantidades de criptomonedas. Estos grifos están limitados incluso en las redes de prueba. En el caso de Ropsten se abrirá una nueva ventana como se muestra en la Figura 4.43, donde habrá que pinchar en el botón verde *Request 1 ether from faucet*. En address podremos ver el monedero desde el que se realizará la transacción a nuestra cuenta. Una vez solicitado, el proceso requerirá de unos segundos para aprobarse la transacción y en muy pocos minutos aparecerán los Éter de prueba en tu cartera.

En la zona inferior de la misma ventana se puede devolver los Éter que no estés usando al grifo original (botones de color naranja). Cuando acabes de hacer pruebas no seas avaricioso y devuelve el capital para que otros puedan hacer pruebas.

Por último, en la pestaña de *Actividad* de la vista principal (ver Figura 4.42) es posible ver todas las transacciones realizadas sobre esta cuenta (envío de Éter y gasto asociado a las implementaciones de contratos entre otros).

²¹Para evitar los comportamientos avariciosos de usuarios que quieren atesorar ETHs de prueba, cada red impone sus mecanismos. Por ejemplo, en Rinkeby te exigen publicar un post en alguna red social con la dirección de Ethereum de tu cartera.

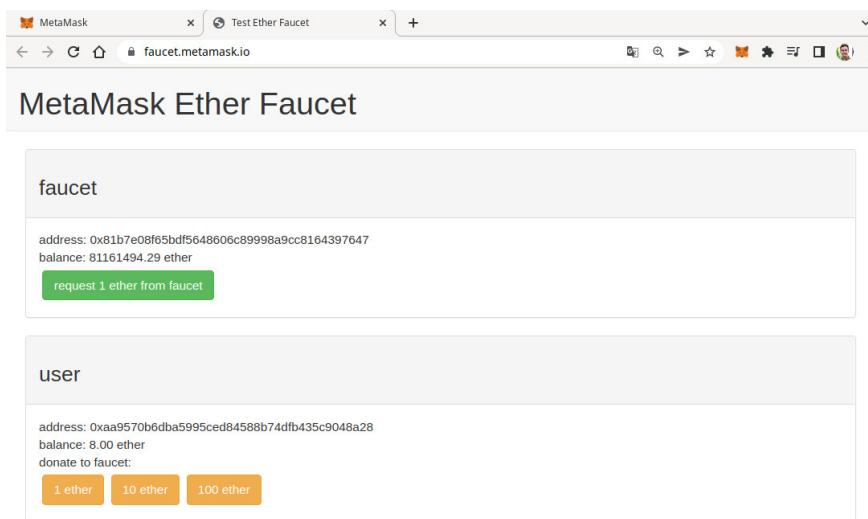


Figura 4.43: Grifo de la red Ropsten donde se puede solicitar Éter de prueba.

Despliegue en la red de pruebas Ropsten

Para desplegar en la red de pruebas Ropsten, tendremos que elegir como Environment *Injected Web3*. Si tenemos Metamask correctamente instalado, nos pedirá que hagamos login en nuestra cuenta y debajo del campo de *Environment* aparecerá la red a la que nos hemos conectado (en el caso de la Figura 4.44, la red Ropsten).

Una vez conectado a la red, con el contrato compilado pincharemos en el botón **Deploy**. En Metamask te aparecerá un mensaje que debes confirmar para pagar el coste de publicación del contrato. Hecho esto, deberás esperar unos segundos a que el bloque esté disponible.



Cuidado con los tiempos! Importante: cuando estamos conectados a una red real de Ethereum, aunque sea una red de pruebas sin *dinero* real, hay que tener en cuenta los tiempos de transacción. En los entornos locales de prueba, las transacciones son inmediatas. Ahora, cuando solicitamos desplegar un contrato hay que esperar unos segundos a que se procese el bloque y se pueda realizar la transacción. No pinches varias veces al botón *Deploy*, pues estarás solicitando que se desplieguen varias instancias del contrato.

Cuando el contrato esté disponible podrás llamar a las funciones correspondientes desde el propio interfaz de Remix, igual que hicimos en el ejemplo de despliegue local. Todas las transacciones con botón de color naranja tienen un coste, y tendrán que ser autorizadas en Metamask; estas transacciones que modifican la Blockchain requieren un tiempo de procesamiento.

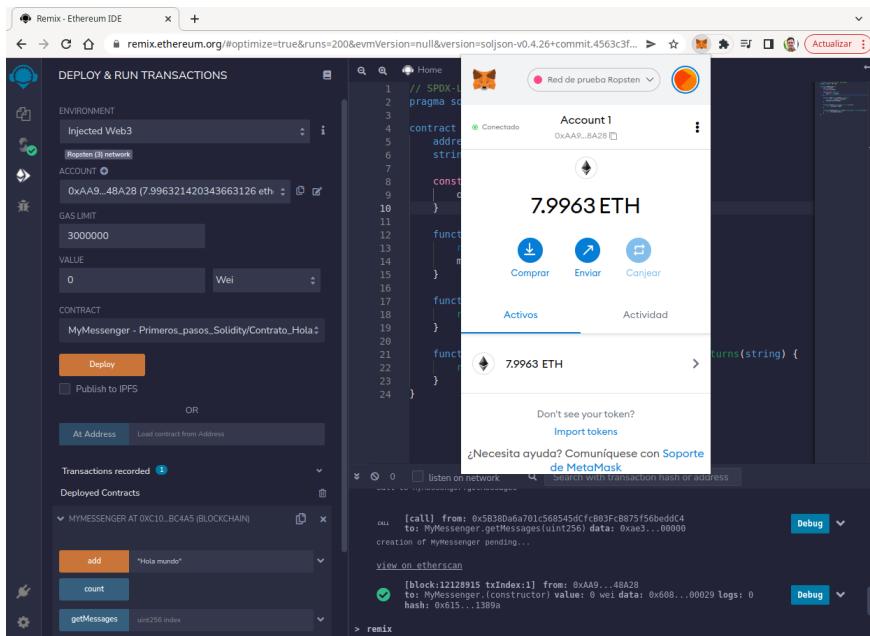


Figura 4.44: Configuración de la conexión a través de Metamask a la red de pruebas Ropsten.

En la siguiente sección veremos cómo inspeccionar los bloques disponibles en la red Ethereum.

Verificando despliegue en Etherscan

Etherscan es el explorador blockchain para la red Ethereum más utilizado en la actualidad. Es gratuito y se accede directamente desde el navegador. En Etherscan se pueden buscar transacciones, bloques, direcciones y otros datos de la blockchain. Como todas las interacciones en Ethereum son públicas podemos visualizar la información de cualquier contrato desplegado en Ethereum (no solo los nuestros). Etherscan ofrece la funcionalidad de un motor de búsqueda completo. Para usar Etherscan no es necesario un registro, pero si creas una cuenta de usuario podrás acceder a funcionalidad adicional como recibir notificaciones de transacciones entrantes (muy interesante si tienes desplegado un contrato real), o acceder a algunas herramientas de desarrollo específicas.

A continuación veremos cómo inspeccionar los contratos que hemos desplegado en el paso anterior. En la consola de Remix, cuando se aprueba una transacción, aparece un link titulado *view on etherscan*. Pinchando sobre el link, se abrirá una nueva ventana del navegador donde accederemos a esa dirección. Cualquier persona que acceda a esa dirección podrá ver toda la información del contrato y las transacciones realizadas. Prueba ahora si quieres a abrir la siguiente url:

<https://ropsten.etherscan.io/address/0xc108419274e461622d6bac12031d99e499ebc4a5>

Esa es la dirección en la Blockchain de esta red de pruebas del contrato que acabo de publicar, junto con las dos llamadas a la función *add*. Puedes ver una captura de pantalla en la Figura 4.45. En la parte inferior de la pantalla se pueden ver las 3 transacciones realizadas. Si se pincha sobre la primera de las llamadas a la función *add* (la que comienza por 0x38), accederemos a una nueva pantalla con el detalle de esta transacción (ver Figura 4.46).

The screenshot shows the Etherscan interface for the Ropsten Testnet Network. At the top, there's a search bar and navigation links for Home, Blockchain, Tokens, Msc, and Ropsten. Below the header, a card displays 'Contract Overview' for the address 0xc108419274e461622d6bac12031d99e499ebc4a5. It shows a balance of 0 Ether and provides 'More Info' such as My Name Tag (Not Available) and Contract Creator (0xa09570b6dba5995ced... at tx 0x35d614a8e42a50ed58). A 'Transactions' tab is selected, showing three transactions:

Txn Hash	Method	Block	Age	From	To	Value	Txn Fee
0x47ee4f47d2a528fad2b...	Add	12120960	2 mins ago	0xa09570b6dba5995ced...	0xc108419274e461622d...	0 Ether	0.0001289775
0x3836865fa949f1bbc05...	Add	12120960	2 mins ago	0xa09570b6dba5995ced...	0xc108419274e461622d...	0 Ether	0.0001716675
0x35d614a8e42a50ed58...	0x60806040	12120915	13 mins ago	0xa09570b6dba5995ced...	Contract Creation	0 Ether	0.000644265002

At the bottom, a note states: "⚠ A contract address hosts a smart contract, which is a set of code stored on the blockchain that runs when predetermined conditions are met. Learn more about addresses in our Knowledge Base." There are also download and export options.

Figura 4.45: Inspeccionando información sobre el contrato desplegado en Etherscan.

Como podemos ver, aunque los datos aparecen serializados como hash en el interior de los bloques, en Etherscan es posible decodificar los datos de entrada y obtener una representación limpia para humanos, como aparece en la tabla asociada a la llamada a la función. También se muestra el detalle del coste asociado a la transacción (Gas utilizado y coste de la operación). Cuidado que cuando trabajamos en la red real, estos costes hay que tenerlos muy en cuenta. El coste de la llamada anterior no es nada despreciable: El coste de la transacción que podemos inspeccionar en Etherscan es de 0.00017 ETH que al cambio actual serían 0,48€ en la red principal de Ethereum.

4.8.4. Siguientes pasos

Una vez finalizada esta introducción, el siguiente paso natural es profundizar en el uso de Solidity aprendiendo más características del lenguaje como el uso de estructuras de datos más completas (como los *mappings*), el modificador *payable* de una función, interacción entre smart contracts, y el uso de modificadores *internal* y *external* entre otros.

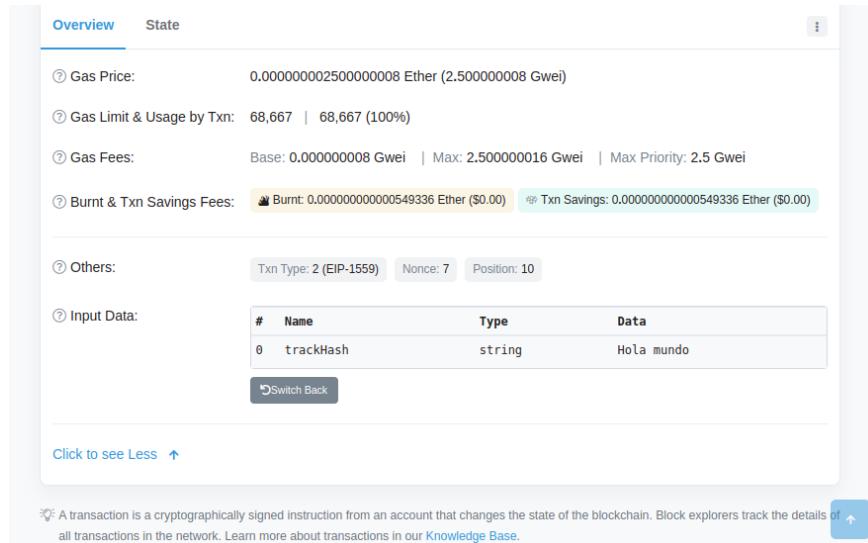


Figura 4.46: Detalle de la transacción relativa a la llamada a la función *add*.

También puede ser razonable estudiar el uso de alguna suite local de desarrollo de proyectos, en lugar de emplear Remix. En este caso, Truffle, Ganache y Drizzle puede ser una buena combinación²²:

- **Truffle.** Entorno de desarrollo con framework de pruebas basado en la EVM. Este entorno resulta de especial utilidad cuando se trabaja con proyectos de cierta complejidad y queremos disponer de un entorno integrado con control de versiones, trabajo colaborativo, etc.
- **Ganache.** Permite disponer de un blockchain personal de pruebas para el desarrollo de Ethereum. Está disponible tanto como aplicación de escritorio como herramienta de línea de comandos para Windows, GNU/Linux y Mac.
- **Drizzle.** Colección de bibliotecas de front-end específica para DApps. El núcleo de Drizzle se basa en un almacén Redux (contenedor del estado de aplicaciones JavaScript), por lo que el desarrollador tiene acceso a todas las herramientas que hay disponibles sobre esta plataforma.

²²Disponibles en <https://trufflesuite.com/>

4.9. Conexión entre tecnologías

En el inicio del capítulo estudiamos que las tecnologías exponenciales se amplifican de un modo mucho mayor cuando convergen. En esta convergencia es necesario la conexión de datos de tipos diversos: texto, imágenes y sonido. La IA se ha convertido posiblemente en la herramienta de colaboración más importante jamás creada. Además, proporciona las **interfaces de usuario naturales** más potentes que pueden existir para acceder a los servicios que proporcionan todas estas tecnologías exponenciales.

En los últimos años han ocurrido algunos hitos que ya aportaban alguna pista sobre los avances que están sucediendo en la actualidad.

En 2011, Watson de IBM ganó una partida de Jeopardy a los dos mejores campeones de todos los tiempos. Esto fue una demostración del poder del procesamiento del lenguaje natural, gestión del contexto y *deep learning*.

Desde entonces este tipo de sistemas han evolucionado mucho. Los asistentes virtuales como Siri y Google permiten resolver cualquier tipo de consulta e interacción en lenguaje natural. Actualmente, y como veremos en esta sección, plataformas como OpenAI con GPT-3 permite generar lenguaje natural y escribir textos con total coherencia semántica y de contexto.

Queda patente cómo los sistemas de *deep learning* permiten el aprendizaje no supervisado, con grandes progresos en este ámbito. En concreto, en reconocimiento e interpretación de imágenes los sistemas existentes ofrecen mejores resultados que las capacidades de los humanos. En 2016 por ejemplo el MIT Technology Review publicó la noticia de que "Una IA derrota a los mejores médicos en el diagnóstico de la retinopatía diabética basada en imagen".

El reconocimiento de imágenes se ha disparado en los últimos años. Facebook tienen miles de millones de imágenes en su plataforma. Estos datos se utilizan para el etiquetado automático, con resultados que hemos utilizado de un modo directo en sesiones anteriores del curso.

En la integración de los otros sentidos, la IA seguirá trabajando para lograr interacciones más naturales. El tacto, oído y olfato irán cobrando protagonismo en los próximos años. Por ejemplo, la integración con sistemas de Realidad Virtual está siendo ampliamente explorada en el nuevo ecosistema de Meta.

En última instancia, durante la revolución de la IA que está ocurriendo, la integración será completa en todo lo que nos rodea, combinando sensores y redes y haciendo que todos los sistemas sean inteligentes".

Para conseguir esta interacción completa será necesario lograr una conexión con todos los sectores económicos y sistemas informacionales (agentes conversacionales) y en todos los canales posibles (basados en texto, imágenes y sonidos). En esta sección analizaremos las herramientas tecnológicas disponibles para su consecución.

4.10. Agentes conversacionales

Una de las herramientas más extendidas actualmente como sistema de convergencia que utiliza conexión de datos heterogéneos son los agentes conversacionales. Un agente conversacional (o *chatbot*) es un programa basado en IA que puede entender el lenguaje natural y es capaz de comunicarse con los usuarios a través de sitios web, aplicaciones, plataformas sociales y sistemas de mensajería. Estos chatbots obtienen la información de sus respuestas de diversas fuentes de información tanto estructurada como no estructurada. Los usuarios pueden interactuar con estos bots, usando texto o voz, para obtener respuestas a las preguntas que planteen.



Interacción por voz. En el caso de interacción mediante voz, es necesario acoplar al bot un par de módulos: uno de reconocimiento del habla y un módulo de *text-to-speech*. Las soluciones que proporciona Google en su *Google Cloud - Compute Engine* (<https://cloud.google.com/>) son líderes en el mercado.

Según el informe de Accenture *Chatbots are here to stay: So what are you waiting for?*²³, casi el 60 % de los CIO y CTO encuestados en su informe aseguran que los chatbot pueden ofrecer un gran retorno de la inversión con un mínimo esfuerzo.

Los bots más extendidos son los de tipo informativo, que permiten obtener información útil dentro de un dominio de problema específico, ayudando a resolver consultas de los usuarios. En el caso de no poder resolver la incidencia, es habitual que se redirija la conversación a un operador humano que atienda las cuestiones planteadas.

4.10.1. Introducción a Power Virtual Agents

Power Virtual Agents es la herramienta existente en la plataforma Microsoft Power Platform que permite crear agentes conversacionales basados en un enfoque prácticamente sin código. Planteado como una solución SaaS (*Software como Servicio*), abstrae al programador de la infraestructura necesaria para su funcionamiento.

Las capacidades de procesamiento del lenguaje natural facilita tener interacciones de varios turnos de conversación que guía al usuario final rápidamente hacia la respuesta que necesitaban. Gracias a los conectores existentes con Power Automate, es posible integrar múltiples back-end que existan en la organización para responder a preguntas con datos dinámicos. Un ejemplo de esta funcionalidad será analizado en la sección 4.10.3.

²³Disponible en: <https://www.accenture.com/cl-es/insights/industry-x-0/conversational-bots>

Finalmente, el módulo de Analíticas permite supervisar y mejorar el rendimiento del bot. Es posible analizar en tiempo real los temas que están funcionando bien y aquellos que no están dando respuesta a las preguntas de los usuarios pudiendo realizar rápidamente los ajustes necesarios para mejorar el rendimiento de la aplicación.

4.10.2. Temas, Variables, Entidades y Control de Flujo

En Power Virtual Agents, un tema define un flujo de desarrollo de una conversación bot. En cada tema se utilizan estructuras de control de flujo de modo que la conversación no tiene que ser totalmente lineal. El uso de variables y entidades permite capturar datos de interés en tiempo real.

Cada tema debe tener varias **frases desencadenadoras**, que son frases o preguntas que es probable que escriba el usuario y que tienen que estar relacionadas directamente con el tema que estamos definiendo. La IA de Power Virtual Agents procesa el lenguaje natural de modo que buscará entre todos los temas existentes para devolver el nodo desencadenante más apropiado. No es necesario que el usuario utilice exactamente la misma frase desencadenadora; el módulo de comprensión del lenguaje natural de Virtual Agents se encarga de devolver la frase más cercana.

Si no encuentra ninguno que se ajuste a las frases desencadenadoras le indicará al usuario que especifique más detalladamente a qué se refiere. En el caso de que más de un tema se ajuste a la pregunta del usuario le indicará igualmente que a qué se está refiriendo. Por este motivo es importante describir claramente el título de cada tema (es posible que el usuario los vea en caso de necesitar desambiguar).

Un aspecto clave en el reconocimiento del lenguaje natural es la identificación de **entidades precompiladas**. Una entidad es una unidad de información que representa un cierto tipo de elemento estructurado, como un número de teléfono, un nombre de persona, un email, una dirección, etc.

En Virtual Agents, además de las entidades precompiladas que cubren los tipos de datos de uso común, también es posible entrenar al modelo del bot algún conocimiento específico del dominio. En este caso se pueden crear **entidades personalizadas** con ese conocimiento específico. Por ejemplo si se está desarrollando un agente para una tienda de bicicletas, el bot deberá conocer los diferentes tipos de bicicletas (carretera, montaña, paseo, plegables, etc...).

El valor capturado de estas entidades se almacenan en **variables**, que guardan las respuestas de los usuarios a preguntas del bot, y que pueden ser reutilizadas más adelante en la conversación. Como veremos en el ejemplo de la sección 4.10.3, es posible guardar el nombre de un estudiante en una variable de *nombre*, de modo que el bot podrá dirigirse a él posteriormente por su nombre de pila.

Las variables también pueden emplearse dentro de expresiones lógicas que modifiquen el **flujo** de la conversación. Como también veremos en el ejemplo de la sección 4.10.3, las variables pueden igualmente servir para enviar y recibir parámetros en llamadas a flujos de Power Automate.

4.10.3. Un ejemplo de conexión con Power Automate

En este ejemplo se definirá un tema relativo a la consulta de horarios. El flujo que se utilizará en este tema está descrito en la Figura 4.47. El primer nodo define las frases desencadenadoras. En este ejemplo solo se han utilizado 5, aunque la recomendación es añadir alguna más con mayor número de palabras clave.

En el nodo de la primera pregunta se utiliza la entidad precompilada que identificará el nombre de la persona y la guardará en la variable *nombre*. Esta variable se utilizará en sucesivos nodos para referirnos al usuario por su nombre de pila en las preguntas que se le realizarán.

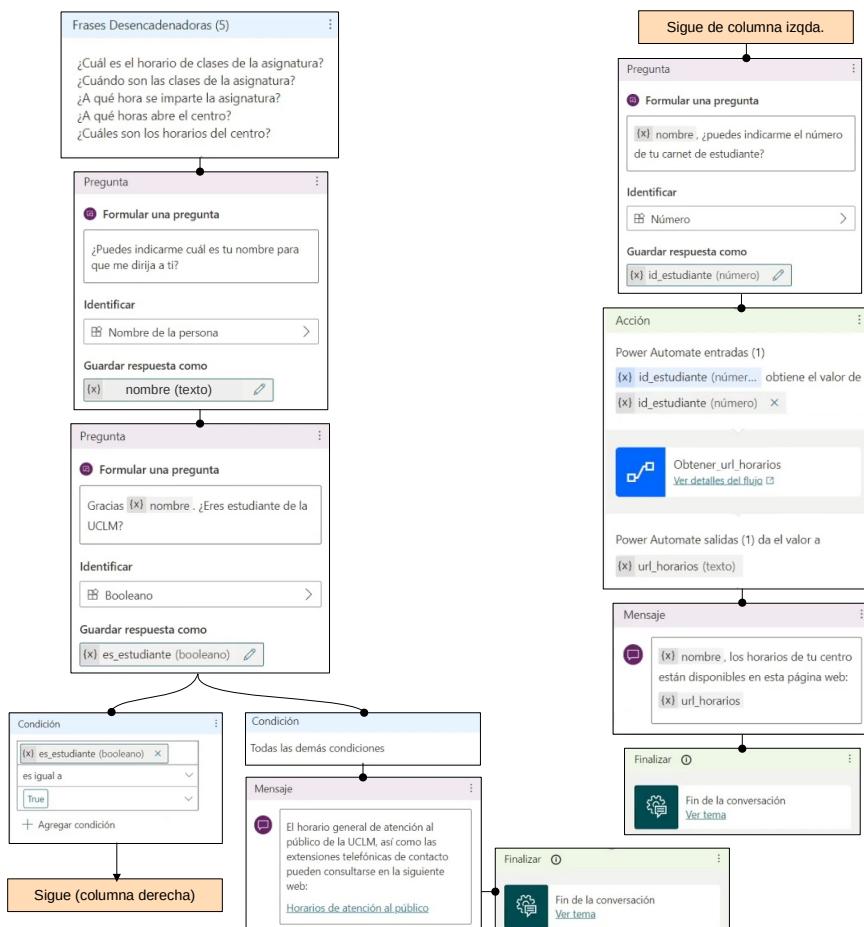


Figura 4.47: Nodos utilizados en *Power Virtual Agents* en la definición del tema para la definición de *Horarios*.

El siguiente nodo realiza una pregunta Booleana (obteniendo el valor de la respuesta sobre la variable *es_estudiante*). A continuación se genera una bifurcación dependiendo del valor de la respuesta. Si el alumno es estudiante en la UCLM, se continuará por el camino de la izquierda, y se le pedirá su identificador de estudiante. De nuevo se utilizará una entidad precompilada que recogerá la respuesta sobre la variable de tipo numérico en *id_estudiante*.

A continuación se llamará a un nodo de acción de *Power Automate*, pasándole como parámetro este identificador del estudiante. Las llamadas a flujos de *Power Automate* permiten pasar como parámetros de entrada y salida valores numéricos, strings y booleanos. La definición del flujo en *Power Automate* puede verse en la Figura 4.48.

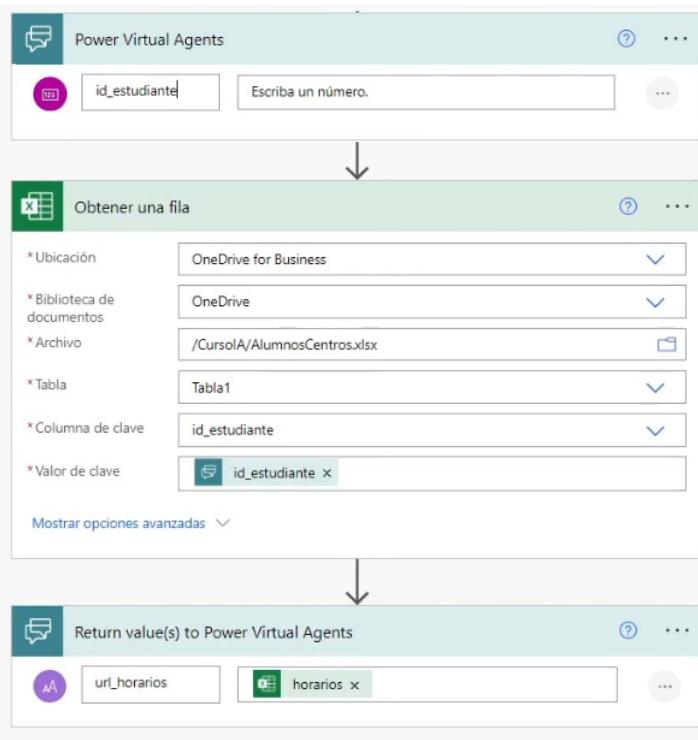


Figura 4.48: Conexión con flujo llamado *Obtener_url_horarios* en Power Automate para la obtención dinámica de una columna en una tabla de Excel almacenada en OneDrive.

Este flujo en *Power Automate* se encarga de obtener de un archivo de Excel (ver Figura 4.49) la columna específica sobre la URL donde se pueden consultar los horarios de cada centro. Este archivo Excel está almacenando en One Drive, de modo que la actualización de esta información podría realizarla cualquier usuario que tenga permisos de edición sobre este archivo.

Resulta obvio que esta estructura de datos para manejar la información sobre alumnos y centros no es la más correcta desde el punto de vista del diseño de una Base de Datos, pero por simplicidad del ejemplo, nos permitimos esta pequeña concesión. Como sabemos, en Power Automate es posible conectar con almacenes de datos bien estructurados y bases de datos completas, por lo que sin pérdida de generalidad, usar una pequeña tabla Excel para este experimento lo vemos adecuado.

El flujo se encarga de obtener la columna de la tabla llamada *horarios*, con los horarios de cada centro a partir del identificador del estudiante. Como vemos en la definición del flujo, se devuelve una variable que hemos llamado *url_horarios*, que será utilizada de nuevo en el flujo principal de Virtual Agents.

	A	B	C	D	E	F
1	id_estudiante	nombre	apellido	centro	urlweb	horarios
2	12345 Juan	Electrón	Escuela Superior de Informática		https://www.esi.ucm.es	https://esi.ucm.es/index.php/grado-en-ingineria-informatica/horarios-y-examenes/
3	12346 María	López	Escuela de Caminos		https://www.caminosciudadreal.ucm.es/	https://www.caminosciudadreal.ucm.es/
4	12347 Carlos	Rodríguez	Facultad de Letras		https://www.ucm.es/ciudad-real/letras	https://www.ucm.es/ciudad-real/letras
5	12348 Perico	Palotes	Facultad de Medicina		https://www.ucm.es/ciudad-real/medicina	https://www.ucm.es/ciudad-real/medicina
6	12349 Lucía	González	Escuela Superior de Informática		https://www.esi.ucm.es	https://esi.ucm.es/index.php/grado-en-ingineria-informatica/horarios-y-examenes/

Figura 4.49: Vista de la tabla almacenada en Excel que será filtrada por la columna de *id_estudiante*.

Para finalizar, el agente mostrará la información al usuario empleando la variable devuelta por el flujo de Automate. En el campo del mensaje se emplea la variable de salida *url_horarios* obtenida en el nodo anterior.

4.10.4. Publicación del Agente

Es posible publicar bots de Power Virtual Agents en diferentes plataformas o canales. Entre otras se incluyen webs, aplicaciones móviles y sistemas de mensajería como Microsoft Teams y Facebook Messenger. Tras publicar el bot se podrá conectar a la vez a todos los canales que sea necesario. Si se actualiza el bot será necesario publicarlo de nuevo desde la plataforma. Esta actualización se propagará en todos los canales donde esté conectado el bot.

Para publicar el bot, bastará con elegir la pestaña **[Publicar]** en el menú principal de la plataforma, y elegir de nuevo la opción de *Publicar* en el botón principal de la zona derecha. Hecho esto, en la pestaña de **[Canales]** del menú principal se podrá acceder a los diferentes sitios donde puede ser publicado el bot.

En la Figura 4.50 se muestra el resultado de conectar el bot en Teams. Para activar el uso del bot en esta plataforma basta con seleccionar Microsoft Teams en la pestaña de Canales, y pulsar el botón de *Activar Teams*.

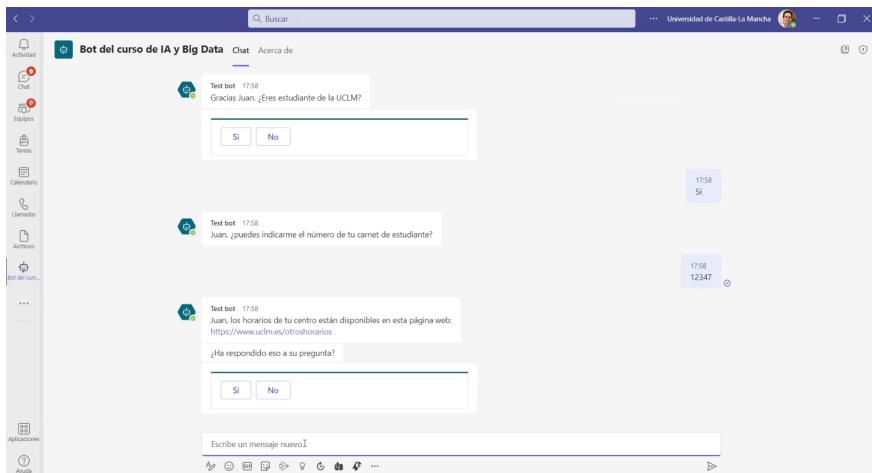


Figura 4.50: Además de su uso directamente en web, el agente conversacional puede publicarse fácilmente en otras plataformas, como Microsoft Teams.

4.11. OpenAI

Analizando las plataformas que procesen texto y que permitan su conexión en otros ámbitos (generación de imagen y sonido), es obligatorio hablar de OpenAI. La organización de OpenAI fue fundada en 2015 por Elon Musk. Es un laboratorio de investigación en IA. Definen su objetivo principal como el desarrollo de una IA *amigable* para el beneficio de la humanidad. En 2019, OpenAI recibió una inversión de mil millones de dólares de Microsoft.

Tal y como indican en su documentación oficial, OpenAI puede aplicarse a diversas tareas que impliquen comprender o generar lenguaje natural. Estas tareas pueden ser de generación de contenidos, búsqueda semántica, clasificación, etc. En versión beta privada también tienen el soporte para generar código. La API ofrece diversos modelos con diferentes niveles de rendimiento (y coste) que son adecuados para diferentes tareas. También es posible ajustar tus propios modelos personalizados.

En OpenAI las consultas se basan en realizar consultas basadas en terminaciones (**completions endpoints**). Estas terminaciones son el punto central de acceso a la API. El usuario introduce un texto como pregunta y el modelo generará una terminación de texto que intentará coincidir con el contexto o el patrón que le haya dado.

Por ejemplo, si le pides a OpenAI: "*Escribe un título gracioso para una novela de romántica*", la API te podrá devolver la terminación de "*La chica del metro que no se bajaba del tren*".



Programación basada en terminaciones. El diseño de la pregunta es la forma de *programar* el modelo. A diferencia de otros servicios de procesamiento del lenguaje natural más enfocados a una única tarea, en OpenAI es posible utilizar las terminaciones para realizar cualquier tipo de tarea: generar texto, resumir contenido, tener una conversación con un agente, escritura creativa, transferencia de estilo, traducción y un larguísimo etcétera.

OpenAI emplea un **conjunto de modelos** con diferentes capacidades y precios. Los modelos predefinidos de GPT-3 son *Davinci*, *Curie*, *Babbage* y *Ada*. Actualmente tienen en beta un modelo privado llamado *Codex* entrenado para lenguaje natural y código fuente.

Los modelos procesan el texto dividiéndolo en tokens. Los tokens pueden ser palabras o grupos de caracteres. Por ejemplo, las palabras "*language natural*" se corresponden con dos tokens, uno para "*language*" y otro para "*natural*". La palabra "*hamburger*" se divide en tres tokens: "*ham-bur-ger*". En castellano, las palabras "*lenguaje natural*" se dividen en 5 tokens: cuatro para lenguaje "*l-engu-aj-e*" y 1 para "*natural*". La Figura 4.51 muestra el uso de una herramienta disponible online para comprobar el número de tokens que ocupa una determinada cadena.

El número de tokens que se procesan en cada solicitud a la API depende de la longitud de las entradas y salidas. La combinación de la solicitud y la terminación generada no puede superar la longitud máxima del contexto del modelo (2048 tokens en la mayoría de los modelos).

Además del diseño de las terminaciones, hay una serie de parámetros que se pueden controlar los resultados que ofrece la plataforma. El más importante es la denominada **temperatura**. El modelo predice un texto que va a continuación del que le precede. El valor de temperatura se define entre 0 y 1 y permite controlar el grado de confianza que debe tener el modelo al hacer estas predicciones. Con temperaturas cercanas a 0 el modelo tendrá menos en cuenta el azar, y el texto generado será determinista. Si el valor de temperatura aumenta, los textos generados serán diferentes. En ciertos dominios de aplicación es habitual trabajar con valores de temperatura intermedios y, si un resultado no es satisfactorio, se puede volver a generar otro resultado.

4.11.1. Motores y modelos

La API de OpenAI utiliza diferentes modelos que presentan diferentes capacidades de procesamiento. Los motores dan acceso a un conjunto de modelos. Como indican en la documentación, también se pueden personalizar los modelos preentrenados para un caso de uso específico con un ajuste fino. Actualmente existen dos motores en la plataforma: GPT-3 y Codex. **GTP-3** es un motor que implementa un conjunto de modelos que pueden entender y generar lenguaje natural. **Codex** por su parte es un motor que proporciona modelos que pueden entender y generar tanto lenguaje natural como código. Incluye capacidades para la traducción de lenguaje natural a código.

The screenshot shows the OpenAI Tokenizer interface. At the top, there's a navigation bar with links for Overview, Documentation, Examples, and Account. Below the header, the title "Tokenizer" is displayed in a large, bold font. A descriptive text block explains that GPT models process text using tokens. Below this, a text input area contains the Spanish sentence: "Esto es una prueba del tokenizador de la familia de modelos de GPT que están disponibles en OpenAI". Underneath the input, two buttons are visible: "Clear" and "Show example". To the right of the input area, a table provides the token counts: "Tokens" (31) and "Characters" (98). Below the table, the input text is shown again, but each word is highlighted with a different color, illustrating how the tokenizer breaks down the sentence into tokens.

Figura 4.51: Mediante la herramienta del Tokenizer se puede comprobar fácilmente el número de tokens asociados a una consulta concreta en GPT-3. La longitud de los tokens difiere en cada idioma a utilizar.

GPT-3 proporciona 4 modelos diferentes que se ajustan a diferentes necesidades concretas. En general cada modelo puede realizar mejor las tareas de los modelos inferiores, pero a mayor coste computacional (que se traduce en coste económico). Los modelos inferiores no pueden realizar con calidad las tareas de modelos superiores. A modo orientativo, 1.000 tokens son unas 750 palabras en inglés y unas 500 palabras en castellano.

- **Davinci.** Es el modelo más potente de los cuatro, que puede realizar tareas más complejas. Es ideal para aplicaciones que requieran mejores resultados. Estas capacidades requieren más recursos computacionales, por lo que el coste del servicio es el mayor de los cuatro. Además, requiere más tiempo para dar un respuesta.

Aplicaciones: Procesamiento complejo, definición de causa y efecto, realización de resúmenes.

Características técnicas: Petición máxima de 4.096 tokens. Actualizado hasta Junio 2021. Coste de 0.06\$ por cada 1.000 tokens.

- **Curie.** Es un modelo muy potente y rápido. Davinci es mejor cuando requiere analizar textos complejos, pero Curie puede realizar igualmente muchas tareas complejas (aunque no tanto como Davinci) más rápidamente. Curie también es bastante bueno respondiendo a preguntas y como chatbot de propósito general.

Aplicaciones: Traducción de idiomas, clasificación compleja, análisis de sentimientos en texto, resúmenes.

Características técnicas: Petición máxima de 2.048 tokens. Actualizado hasta Octubre 2019. Coste de 0.006\$ por cada 1.000 tokens.

- **Babbage.** Modelo para realizar tareas más sencillas, como clasificación o búsqueda semántica.

Aplicaciones: Clasificación sencilla y búsqueda semántica.

Características técnicas: Petición máxima de 2.048 tokens. Actualizado hasta Octubre 2019. Coste de 0.0012\$ por cada 1.000 tokens.

- **Ada.** Es el modelo más rápido y permite realizar tareas como el análisis sintáctico de textos, la corrección de direcciones y ciertos tipos de tareas de clasificación básica.

Aplicaciones: Análisis de texto, clasificación simple, corrección de direcciones, palabras clave.

Características técnicas: Petición máxima de 2.048 tokens. Actualizado hasta Octubre 2019. Coste de 0.0008\$ por cada 1.000 tokens.

4.11.2. Ejemplo de uso en Python

A continuación describiremos un ejemplo de uso sencillo en Python. Para utilizar la plataforma tendremos que instalar primero el framework en Python. Para ello, bastará con ejecutar desde un terminal:

```
$ pip install openai
```

Una vez instalado, podremos desarrollar un sencillo "Hola Mundo", con el código que se muestra en el listado 4.3. Se puede llamar a la API a través de cualquier lenguaje que pueda realizar llamadas HTTP. En concreto, existen bibliotecas oficiales en Node.js y Python, aunque la comunidad de usuarios mantiene un alto número de interfaces a otros lenguajes.

La referencia completa de las llamadas y los parámetros pueden consultarse en la web de la plataforma²⁴

²⁴En concreto, la especificación de la API está en: <https://beta.openai.com/docs/api-reference/>

Listado 4.3: Hola Mundo en OpenAI

```

1 import os
2 import openai
3
4 openai.api_key = "TUCLAVEAQUI"
5
6 response = openai.Completion.create(
7     engine="text-davinci-002",
8     prompt="Traduce esto al 1. Ingles, 2. Frances y 3. Aleman:\n\nCuantos robles roeria un
9         roedor si los roedores royeran robles?\n\n1.",
10    temperature=0.3,
11    max_tokens=100,
12    top_p=1.0,
13    frequency_penalty=0.0,
14    presence_penalty=0.0
15 )
16 print (response)

```

Como se muestra en el listado anterior, en la ④ hay que especificar la clave de la API que permite utilizar los servicios de OpenAI. Esta clave puede obtenerse en la sección específica de la cuenta de usuario de la plataforma (ver Figura 4.52), y es secreta. No deberás compartirla con nadie.

SECRET KEY	CREATED	LAST USED
sk-...jH9F	2 abr 2022	2 abr 2022

[+ Create new secret key](#)

Default organization

If you belong to multiple organizations, this setting controls which organization is used by default when making requests with the API keys above.

Universidad de Castilla-La Mancha

Note: You can also specify which organization to use for each API request. See [Authentication](#) to learn more.

Figura 4.52: En la sección de API keys se puede obtener la clave necesaria para realizar las llamadas a OpenAI.

La llamada se realiza en las líneas ⑥-11 con el método `openai.Completion.create`. En esta llamada se indican una serie de parámetros que analizaremos a continuación.

- **engine.** En este parámetro es obligatorio y requiere indicar el ID del modelo a utilizar. Dependiendo del motor se accederán a las diferentes capacidades que se han comentado anteriormente. Los literales permitidos son `text-davinci-002`, `text-curie-001`, `text-babbage-001` y `text-ada-001`.
- **prompt.** Este es el parámetro principal de trabajo del sistema, contiene la tarea que se le solicita a GPT-3. Está codificado como un *string* o un *array* de *strings*.
- **temperature.** Este parámetro indica la temperatura de muestreo que se utilizará. Valores altos implica que el modelo dará resultados más aleatorios. En el caso de aplicaciones creativas, el valor debe ser alto. En el caso de aplicaciones con respuestas más controladas es mejor utilizar valores bajos.
- **max_tokens.** Número máximo de tokens a generar. El número de tokens de la solicitud más este parámetro (*max_tokens*) no puede exceder la longitud del contexto del modelo. Por ejemplo, en Davinci no puede superar los 4096 tokens.
- **top_p.** El muestreo de núcleo es una alternativa al muestreo con temperatura. En este tipo de muestreo el modelo considera solo los resultados de los tokens con un valor de *masa de probabilidad* especificado en *top_p*. Por ejemplo, si *top_p = 0.1* significa que solo se usarán los tokens cuya *masa de probabilidad* esté en el 10 % superior. Se recomienda en cualquier caso modificar o la temperatura o este valor *top_p*, pero no ambos a la vez.
- **frecuency_penalty.** Valor entre -2,0 y 2,0. Los valores positivos penalizan los nuevos tokens en función de su frecuencia en el texto hasta ese momento. Si el modelo repite líneas en el texto puede ser recomendable aumentar el valor para que disminuya la probabilidad de que ocurra esta repetición.
- **presence_penalty.** Valor entre -2,0 y 2,0. Los valores positivos penalizan los tokens nuevos en función de su aparición en el texto hasta el momento. Valores negativos aumentan la probabilidad de que el modelo hable de temas nuevos.

En la web de OpenAI existe una sección llamada *Playground* (ver Figura 4.53) donde el usuario puede realizar pruebas de terminaciones, solicitar la regeneración de otras versiones de texto, así como ver los efectos de modificar los parámetros de configuración del modo de trabajo del modelo. En la zona inferior derecha aparece el número de tokens que se están utilizando en el ejemplo (en el caso de la Figura 4.53 se están utilizando 175 tokens).

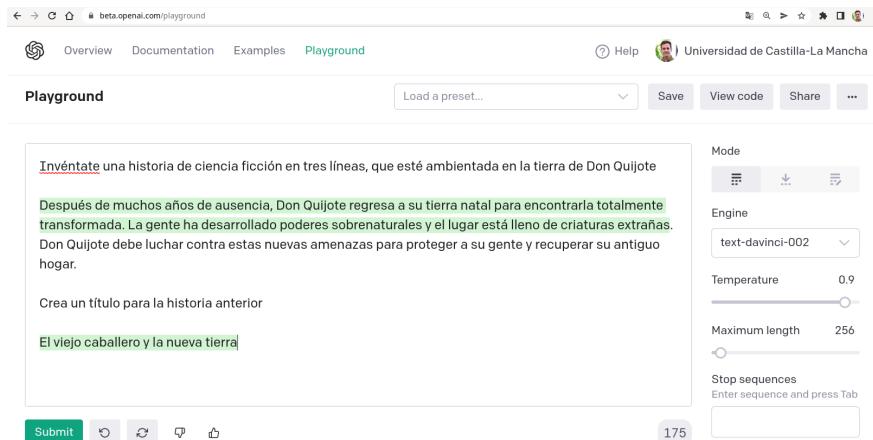


Figura 4.53: En la URL <https://beta.openai.com/playground> se accede a un entorno donde se pueden probar diferentes valores de los parámetros de configuración del motor (ver zona derecha de la ventana).

4.12. Sistemas IoT

El término IoT proviene del inglés *Internet of Things*. Podría definirse como la interconexión de dispositivos (*cosas*) a través de una red, normalmente Internet²⁵. En esta red los dispositivos pueden ser visibles e interaccionar entre ellos. En la figura 4.54 se muestra la evolución del número de *cosas* conectadas a internet en los últimos años. Estas *cosas* conectadas pueden ser dispositivos de tipos muy diversos: vestibles (calzado y ropa), frigoríficos, asistentes digitales, cerraduras e incluso cepillos de dientes. La idea subyacente es que las *cosas* puedan interaccionar sin necesidad de la intervención humana (interacción M2M - *machine to machine*). Estas interacciones generan gran cantidad de datos digitales que, posteriormente, pueden ser analizados, tratados y utilizados para la toma de decisiones automática. Cuando la IA se añade al IoT surge un nuevo término: AIoT, que genera datos sobre IoT y extrae conocimiento a través de IA. La IA por tanto añade en análisis de datos y la tomar decisiones sin la participación de los humanos.

Esta convergencia tecnológica tiene muchas áreas de aplicación. A continuación se enumeran algunas de las principales:

- **Aplicaciones industriales.** AIoT puede realizar mantenimientos predictivos de maquinaria, evitando el tiempo de inactividad y mejorando así la vida útil de la misma.
- **Comercio inteligente.** Identificación de compradores en la entrada de un comercio identificando sexo, edad, flujos de tráfico en la tienda, etc. Analiza los datos para predecir el comportamiento de consumo, ofreciendo promociones personalizadas en monitores y paneles situados en lugares de la tienda.

²⁵Aunque también se considera IoT la conexión de dispositivos en una red privada

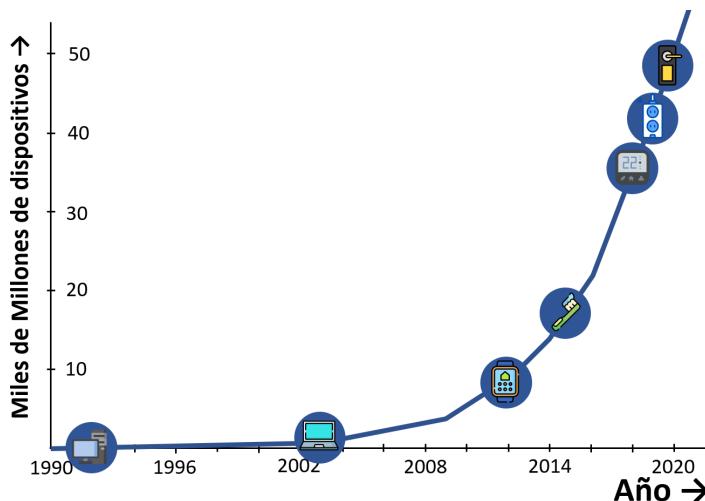


Figura 4.54: Evolución en el número de dispositivos IoT y algunos hitos tecnológicos asociados con su fecha de aparición en el mercado. Fuente: Elaboración propia a partir de datos de *sociedaddelainformación.es*

- **Aplicaciones domésticas.** Reducción de costes energéticos haciendo que las viviendas sean más eficientes energéticamente. Por ejemplo, se pueden tener sistemas de climatización inteligentes, bombillas inteligentes, sistemas de diagnóstico predictivos, etc.
- **Ciudades Inteligente.** En las ciudades inteligentes hay muchos usos de la AIoT, como por ejemplo la supervisión del tráfico drones. Los atascos pueden reducirse si se puede monitorizar, realizando ajustes en límites de velocidad y tiempos de semáforos en tiempo real.
- **Fabricación y logística.** Los robots en las fábricas emplean diversos sensores integrados que permite hacer el análisis del proceso de fabricación identificando posibles mejoras de un modo automático. Por otra parte, la AIoT se emplea también con vehículos autónomos con recopilación de datos sobre el entorno que facilita la toma de decisiones sobre la navegación en cada momento. En la actualidad muchos gigantes tecnológicos como Amazon, Google o FedEx ya cuentan con robots de entrega en el negocio minorista.

En relación con la sección 4.13, el artículo "*Role of Artificial Intelligence in the Internet of Things (IoT) cybersecurity*" resume los principales métodos, técnicas y trabajos de investigación disponibles en la actualidad para atacar los sistemas IoT²⁶.

²⁶Un artículo relevante en este sentido puede encontrarse en modalidad Open Access en el siguiente enlace: <https://link.springer.com/article/10.1007/s43926-020-00001-4>

4.13. Seguridad en la convergencia tecnológica

La Ley de Reglamento General de Protección de Datos (RGPD²⁷) establece, en su artículo 5, la seguridad como principio básico en el tratamiento de datos personales. Según el RGPD, la seguridad es un requisito obligatorio (no es una opción, es una necesidad), y no aplicar las medidas de seguridad adecuadas es ilegal. Los aspectos de seguridad no pueden quedar relegados únicamente a un posible impacto económico relativo a su incorrecta aplicación.

El artículo 32 exige medidas de seguridad que deben aplicarse en función de la probabilidad del riesgo y su gravedad. Los datos personales deben protegerse de forma progresiva. Cuanto mayores sean los riesgos, más estrictas serán las medidas.

Los sistemas de IA pueden no ser totalmente coherentes y completos, lo que significa que no se puede predecir durante la fase de diseño todos los posibles factores contextuales que pueden alterar su funcionamiento. Esto expone a las personas a los riesgos de resultados inesperados en la ejecución de sistemas de IA.

Así, en sistemas basados en IA, es muy importante introducir mecanismos de seguridad técnica como anonimización de datos (de modo que los datos personales no puedan consultarse de un modo directo) y cifrado.

Como se ha comentado anteriormente, la seguridad no debe aplicarse únicamente para evitar pérdidas, sino para crear valor. Los sistemas de IA generarán confianza atrayendo inversores y usuarios si no existen incidentes de seguridad.

La Agencia de Ciberseguridad de la Unión Europea, ENISA, ha creado un informe titulado *Retos de ciberseguridad de la AI: Panorama de las amenazas para la Inteligencia Artificial*²⁸. El contenido de esta sección está directamente extraído de los contenidos que se abordan en este informe.

El informe presenta las amenazas relativas a ciberseguridad en IA en tres líneas de trabajo principales:

- Alcance de la IA en el contexto de la ciberseguridad prestando atención al ciclo de vida de las aplicaciones de IA.
- Identificación de los activos de las aplicaciones de IA para determinar lo que debe protegerse y lo que podría ir mal en términos de seguridad.
- Creación de una taxonomía detallada de posibles amenazas. Esta taxonomía sirve como base para la identificación de posibles vulnerabilidades y escenarios de ataque.

A su vez en el documento se analizan las interrelaciones entre el ámbito de la IA y la ciberseguridad. En concreto, se plantean tres dimensiones esenciales:

²⁷<https://www.boe.es/doue/2016/119/L00001-00088.pdf>

²⁸El informe completo de ENISA está accesible y puede descargarse en la siguiente URL: <https://www.enisa.europa.eu/news/publications/artificial-intelligence-cybersecurity-challenges>

1. **Ciberseguridad para la IA.** Esta dimensión analiza la posible falta de solidez y vulnerabilidades de los modelos y algoritmos de IA. En esta dimensión tendrían cabida la inferencia y manipulación de modelos por parte de terceros, ataques contra sistemas físicos impulsados por la IA, manipulación de los datos, etc.
2. **La IA para apoyar la ciberseguridad.** Esta dimensión analiza el uso de IA como herramienta para crear añadir medidas de ciberseguridad desarrollando controles más eficaces. Algunos ejemplos de uso serían el uso de antivirus inteligentes, análisis forense inteligente, escaneo de correo electrónico, análisis automatizado de malware, etc.
3. **Uso malintencionado de IA.** Uso de la IA para crear ataques más sofisticados. Ejemplos: malware basado en IA, ingeniería social avanzada, cuentas de redes sociales falsas con uso de IA, modelos generativos para crear datos falsos, descifrado de contraseñas apoyado por IA, etc.

Queda fuera del alcance de esta sección analizar en detalle las consideraciones sobre seguridad en aplicaciones de IA. Se recomienda el estudio del informe de ENISA mencionando anteriormente. El informe profundiza en los aspectos esenciales en el desarrollo de sistemas seguros. En concreto, el capítulo 2 presenta un modelo de referencia general para el ciclo de vida de los sistemas de IA que identifica claramente los activos y procesos que intervienen. El capítulo 3 clasifica los activos del ecosistema de IA (teniendo en cuenta el ciclo de vida definido en el capítulo anterior) en 6 grupos: Datos, Modelo, Actores, Procesos, Entorno/Herramientas y Artefactos. El capítulo 4 presenta una taxonomía de amenazas y su asignación con los activos introducidos en el capítulo anterior. El capítulo final propone una serie de recomendaciones de alto nivel. Los anexos del documento incluyen unas tablas muy completas que incluyen la taxonomía de amenazas analizadas en detalle, así como la asignación de activos a las etapas del ciclo de vida del desarrollo de aplicaciones de IA.

5

Capítulo

Evaluación de modelos

David Vallejo Fernández, Cristian Gómez Portes

Este capítulo pone el foco en estrategias y modelos de negocio donde el uso de la IA juega un papel fundamental. Particularmente, se tratarán cuatro grandes cuestiones: estrategias corporativas, modelos de negocio, modelos de automatización, y gestión de activos y recursos.

El uso de IA a nivel corporativo ha llegado para quedarse y, antes o después, cualquier empresa, independientemente de su dominio de trabajo y de su capacidad, deberá valorar qué puede aportar la IA a su negocio para no quedarse atrás con respecto a sus competidores. En este sentido, el presente capítulo pretende aportar contenido que sirva para reflexionar sobre cómo plantear la visión estratégica de una empresa que vaya a hacer uso de IA. Este tema, por sí solo, podría servir como eje para escribir uno o varios libros, por lo que aquí se ofrece una visión general que el lector puede completar con la bibliografía recomendada.

5.1. Estrategias corporativas

5.1.1. Visión general

Cualquier líder o responsable de un negocio con cierta proyección no puede ignorar el **impacto potencial de utilizar IA**. Antes o después, e independientemente del tamaño de dicho negocio, será necesario evaluar qué valor añadido puede aportar la IA para ofrecer un mayor beneficio. Esto se traduce en llevar a cabo el diseño de una estrategia que identifique cuáles son las prioridades del negocio en base a los objetivos estratégicos de la empresa y en un contexto donde la IA desempeñe un rol fundamental.

Actualmente, ya existen numerosos ejemplos de empresas que usan la IA para dirigir sus avances en tres áreas principales [Mar20] (ver figura 5.1):

- Creación de **productos inteligentes**. La etiqueta *smart* se ha venido utilizando en los últimos años para denotar ese componente de inteligencia en una gran cantidad de dispositivos que forman parte de nuestro día a día. *Smartphone*, *smartband*, *smart TV* y *smartwatch* son algunos de los ejemplos más representativos de esta nueva generación de productos inteligentes. La noción de inteligencia está presente no solo en los asistentes virtuales que nos recomiendan regalos o nos sugieren información de interés, sino también en dispositivos como, por ejemplo, los frigoríficos inteligentes que son capaces de realizar la compra por nosotros. Así, los productos inteligentes diseñados para mejorar la calidad de vida de los usuarios finales aumentan el número de clientes satisfechos, lo cual implica un beneficio para los negocios en términos de fidelización y captación de clientes. En el corazón de los productos inteligentes se encuentran los datos.
- Ofrecimiento de **servicios inteligentes**. Uno de los servicios inteligentes por excelencia es Netflix™, el cual se basa en analizar y entender cómo los usuarios emplean la plataforma de *streaming*, basada en suscripción, para ofrecer recomendaciones personalizadas. Resulta importante recalcar la idea de que un servicio inteligente evoluciona, dinámicamente, en base a cómo sus clientes lo utilizan, lo cual marca una diferencia fundamental con el enfoque clásico de tratar de predecir, de antemano, cómo los clientes harán uso de un determinado servicio. El lector habrá identificado que, de nuevo, en el núcleo de un servicio inteligente se encuentran los datos.
- Desarrollo de **procesos de negocio inteligentes**. La unión de la proliferación de modelos de automatización y el avance de la robótica representan uno de los ejemplos más relevantes en el ámbito de los procesos inteligentes. Desde el punto de vista de las cadenas de fabricación automatizadas, y en el contexto de la industria manufacturera, existe una clara tendencia a la reducción de empleados humanos y al incremento de robots. Esto introduce mecanismos de interacción entre personas y máquinas que se discutirán más adelante. Esta forma de interacción, a nivel de operaciones, permite la generación de datos que pueden utilizarse para mejorar, de forma incremental, los procesos de negocio de una empresa.



Internet de las Cosas. Los productos inteligentes están directamente relacionados con el concepto Internet de las Cosas (*Internet of Things*), y es que, actualmente, vivimos rodeados de todo un ecosistema de redes de sensores dotados de inteligencia para facilitarnos las tareas en nuestro día a día. El abaratamiento del hardware unido a los últimos avances en IA han facilitado la eclosión de esta tecnología.

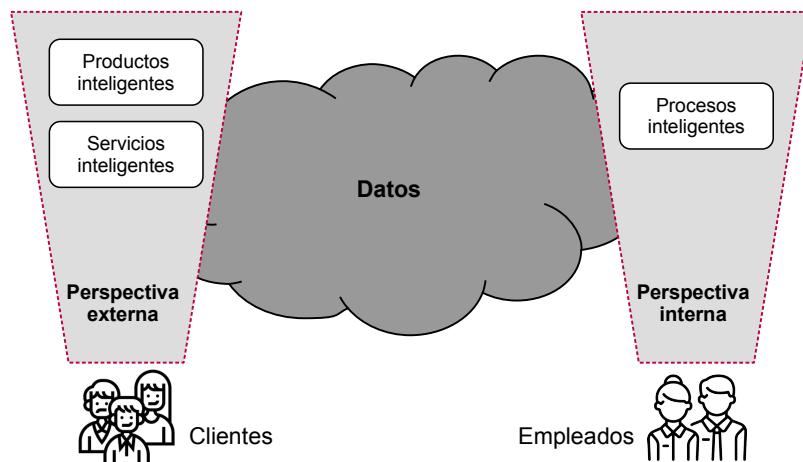


Figura 5.1: Representación gráfica de las perspectivas con las que se pueden entender los productos, servicios y procesos inteligentes. Los datos juegan un papel fundamental en todos ellos.

Los productos y los servicios inteligentes responden a una **perspectiva externa**, es decir, se diseñan desde la óptica del cliente con el objetivo de proporcionar un valor al usuario final. Este valor está relacionado con un abanico funcional que comprende no solo herramientas que facilitan la vida del cliente, sino también la forma de identificar y anticiparse a necesidades futuras. La IA ya es un componente más que relevante en este ámbito.

Los procesos de negocio inteligentes se relacionan con una **perspectiva interna**, en el contexto del nivel de operaciones de la propia empresa, y con el objetivo final de mejorar la eficiencia en los procesos internos que la gobiernan. Dependiendo del tipo de negocio, los procesos de negocio pueden afectar tanto a los empleados que conforman la empresa como a la cadena de producción física, si es esta existe. Si la perspectiva externa previamente introducida estaba centrada en los clientes, la perspectiva interna está centrada en los empleados (humanos o máquinas). Esto afecta tanto a las interacciones existentes entre dichos empleados, tratando de mejorar y aumentar su nivel de colaboración efectiva, como a la capacidad de atraer talento externo. Aunque pueda parecer que la IA tiene menos que aportar en esta perspectiva, la realidad es que todo lo que tiene que ver con gestión de recursos humanos es un área en la que la IA está demostrando ser particularmente útil.



Data-driven. Los datos juegan un papel fundamental en las tres áreas de productos inteligentes, servicios inteligentes y procesos de negocio inteligente. Su captación y análisis posibilita aplicar técnicas de IA orientadas a mejorarlos constantemente. Piense, por ejemplo, en la cantidad de información que se puede generar a partir de la forma de uso de un dispositivo por parte de miles de usuarios.

5.1.2. Productos inteligentes

Los dispositivos como los *smartphones* o los *smartwatches*, entre otros muchos que se podrían encuadrar en el ámbito de los productos inteligentes, comparten la capacidad de recoger, enviar y recibir datos. Es esta capacidad la que dirige las interacciones con el usuario final, y en el núcleo del proceso se encuentran, una vez más, los datos. Esta **conectividad existente entre el producto y el usuario** es la que sirve para aportar un valor añadido con respecto a un producto tradicional. Adicionalmente, y en función de las características y el ámbito de aplicación del producto inteligente, este también tendrá la capacidad de procesar datos localmente.

Piense, por ejemplo, en un monitor de actividad que se coloca en la pulsera para medir la actividad física y la condición de salud del usuario. Típicamente, este producto tendrá la capacidad de monitorizar el estado del usuario (recolección de datos) para, posteriormente enviar la información asociada a dicho estado a un servidor central a través del teléfono móvil al que se encuentra conectado vía *Bluetooth* (envío de datos) y, finalmente, obtener un *feedback* que aporte valor añadido al usuario (recepción de datos). Una situación particular podría ser la detección de un nivel de estrés elevado por parte del producto inteligente que tendría como respuesta una recomendación para que el usuario se relaje (a través de, por ejemplo, un temporizador de respiración para calmarse).



Más allá de los productos. Existen dos conceptos que están íntimamente relacionados con los productos inteligentes: i) el Internet de las Cosas (Internet of Things) y ii) los dispositivos wearables.

¿Productos inteligentes en tu negocio?

Aunque existen diversos beneficios que se pueden derivar como consecuencia de añadir una capa de inteligencia a tus productos, la **principal aportación** sería la capacidad de **entender cómo tus usuarios emplean realmente tus productos**. Este cómo se puede entender más allá del nivel operativo, es decir, de la forma de usar los productos, y abarca la comprensión de hábitos y preferencias de uso por parte de los clientes. Una vez más, los datos obtenidos por el producto inteligente se pueden utilizar para mejorar y dirigir la evolución de tus productos.

Por otro lado, la tendencia actual, que no parece que vaya a cambiar en los próximos años, refleja el hecho de que todo se está volviendo más inteligente. Esto implica que una empresa que basa su negocio en el diseño y comercialización de productos se puede quedar atrás si no contempla el **uso de IA en su estrategia**, al menos a medio y largo plazo.



Figura 5.2: Uno de los ámbitos donde más están proliferando los productos inteligentes es el relacionado con el cuidado y monitorización de la salud.

El lector puede pensar que existen ámbitos donde no existe la necesidad de añadir inteligencia a determinados productos. Esto puede responder a varios motivos, como la complejidad técnica derivada de incorporar la capa de IA, el potencial hecho de romper con un diseño tradicional, lo cual implica un nuevo diseño más disruptivo, o el encarecimiento del producto debido a la necesidad de añadir nuevo hardware y capacidades de comunicación. Sin embargo, existen soluciones a todos estos motivos y, probablemente, a cualquier reto que se plantee como consecuencia de añadir IA. La complejidad técnica se puede combatir con soluciones ya existentes de terceras partes; no es necesario partir de cero. Los diseños tradicionales perderán peso con respecto a otros más disruptivos, siempre y cuando la IA aporte un valor añadido sobre la solución tradicional. El encarecimiento de añadir nuevos sensores se irá mitigando conforme avance el tiempo (salvo que haya una crisis, temporal, de suministro de componentes).

Un ejemplo que da qué pensar es el de la industria relojera [Mar20]. Compare un reloj Rolex™ tradicional, por poner un ejemplo, con un *smartwatch* como el Apple Watch™. Ambos son dispositivos que el usuario lleva encima, normalmente, las 24 horas del día. Evidentemente, el *smartwatch* aporta mucha más funcionalidad que el reloj tradicional, pero nada puede compararse a un reloj analógico basado en componentes mecánicos y con un diseño espectacular... ¿O sí? ¿Piensa que Rolex está abocada a desaparecer si no introduce la IA en su estrategia de negocio? Ahora relacione la capacidad del *smartwatch* con la funcionalidad ofrecida por el producto inteligente previamente descrito, y que posibilita la detección de picos de estrés y las acciones que sugiere al usuario para reducirlo. Es poco probable que el Rolex tradicional lo haga. Ahora extrapole la potencial capacidad del reloj inteligente para realizar un electrocardiograma en tiempo real (el Apple Watch lo hace) y para anticiparse a un posible escenario en el que su usuario va a sufrir infarto de miocardio. El valor añadido que ofrece el reloj inteligente empieza a contrarrestar la belleza y elegancia del Rolex tradicional.



Figura 5.3: A la izquierda, un Rolex Datejust 41mm (2021). A la derecha un Apple Watch Series 7 (2021).

Esto no quiere decir que todos los productos tengan que ser inteligentes de hoy para mañana. Sin embargo, la tendencia en el diseño y creación de productos muestra una clara evolución hacia el uso de la IA, independientemente de la categoría en la que dichos productos se clasifiquen.

Clasificación de productos inteligentes

Es posible establecer diversas clasificaciones de productos inteligentes. A continuación, se resumen una de ellas basada en sectores relacionados con la localización física en la que se utilizan dichos productos [Mar20]¹:

- **Productos inteligentes en el ámbito doméstico.** Nuestras casas han ido integrando, cada vez más, diversos dispositivos orientados a facilitar la interacción con nuestro entorno directo. Algunos ejemplos representativos son los altavoces o asistentes inteligentes, como AlexaTM o Google AssistantTM, lavadoras inteligentes, capaces de adaptar el ciclo de lavado en función de la meteorología, frigoríficos inteligentes, capaces de monitorizar lo que tienen dentro y de realizar pedidos de forma automática, cepillos de dientes inteligentes, con funcionalidad para monitorizar y mejorar los hábitos de lavado, termostatos inteligentes, que van más allá de la simple programación de rutinas y adaptan la temperatura por iniciativa propia, luces inteligentes, capaces de comportarse de una forma u otra en función de la hora y la presencia de humanos, o dispositivos inteligentes de vigilancia, que ofrecen funcionalidad como la detección facial o la monitorización en base a patrones de comportamiento.
- **Productos inteligentes para transporte y movilidad.** La forma en la que nos movemos también se está viendo influenciada enormemente por el uso de IA. El ejemplo clásico es el del uso de coches inteligentes, capaces de conducir de manera autónoma. Sin embargo, también existen bicicletas inteligentes, capaces de adaptar el soporte artificial al ciclista en base a la distancia

¹ Consideraremos que el smartphone forma parte del propio usuario, ya que es muy poco común encontrar a alguien que salga de casa sin su teléfono inteligente.



Figura 5.4: Ejemplo de aplicación de las gafas de realidad mixta Microsoft HoloLens 2TM.

a un determinado destino o de emitir alertas ante situaciones potenciales de peligro, o incluso barcos autónomos capaces de operar sin soporte humano, especialmente en operaciones de carga, transporte y descarga de mercancías. Los robots y drones diseñados para entregar paquetes de mercancías son otro ejemplo que se podría encuadrar en la categoría de transporte.

- **Productos inteligentes en el ámbito industrial y manufacturero.** Los productos diseñados en este contexto tratan de aportar un valor añadido en el ámbito de las operaciones de una empresa, por lo que están relacionados con los procesos de negocio. Un ejemplo representativo serían las gafas inteligentes, como por ejemplo las Google GlassTM, creadas para ofrecer información en tiempo real y aumentar las capacidades de interacción de un usuario gracias a una conexión permanente a Internet (y a sus servicios). La idea de gafas inteligentes también está vinculada al soporte para la toma de decisiones, que típicamente permite a un operario inexperto llevar a cabo un procedimiento o una tarea de forma guiada y a través de información aumentada. En este contexto, las gafas de realidad aumentada o de realidad mixta, como las populares Microsoft HoloLens, cobran especial importancia (ver figura 5.4). Las industrias ganadera y agrícola también se están beneficiando de la penetración progresiva de la IA, gracias a la incorporación de capas inteligentes en la maquinaria empleada. Un ejemplo concreto es la integración de técnicas de visión por computador para distinguir entre una materia prima que se encuentra, o no, en buen estado (piense en el fruto de una cosecha recolectado de manera automática y un proceso que considera parámetros de calidad). Por otra parte, la existencia de robots capaces de aprender por sí mismos, mediante técnicas de aprendizaje automático, a realizar tareas de una manera más efectiva es otro ejemplo de lo que la IA puede aportar en este campo.

- **Productos inteligentes para el deporte.** Si bien los dispositivos wearables conforman el núcleo de lo que serían productos inteligentes en el ámbito deportivo, existe todo un ecosistema de aparatos que, gracias a la IA, aportan valor añadido a sus usuarios. Algunos de ellos se basan en la idea de monitorear, y detectar patrones, sesiones de entrenamiento en deportes como el baloncesto o el balonmano. El conocimiento generado se puede utilizar como realimentación para mejorar la técnica del jugador.
- **Productos inteligentes en el ámbito sanitario.** Probablemente, el sector de la salud sea uno de los que más se pueden beneficiar de la utilización de IA y, en consecuencia, esto puede contribuir a mejorar nuestra calidad de vida. Considere ejemplos como las lentes de contacto inteligentes, con la capacidad de focalizar de manera instantánea, el equipamiento de análisis de imagen médica, que podría detectar anomalías en menos tiempo y con menos exposición a radiación, o los sistemas de detección automática de caídas en casa. Finalmente, piense en productos que se anticipen a potenciales problemas de salud, en lugar de esperar a que algo ocurra y que se aplique un tratamiento. Es lo que se denomina medicina preventiva.

5.1.3. Servicios inteligentes

El concepto de inteligencia asociado a un servicio se puede relacionar principalmente con dos ideas: i) hacer uso de IA para mejorar el servicio ofrecido a los clientes finales o ii) poner el uso de IA en el centro del modelo de negocio, de forma que la IA se convierte en el servicio ofrecido.



IA y nuevos servicios. El hecho de utilizar IA podría abrir nuevas vías y oportunidades de negocio, lo cual representaría un nuevo uso de la IA en el contexto del diseño y ofrecimiento de servicios.

El poder de la personalización

Si tuviéramos que elegir una característica común a los modelos de negocio basados en servicios que hacen uso de IA, esta sería la capacidad de personalización (traducción del término en inglés *customisation*). En este sentido, la **IA resulta fundamental para aprender y perfilar** el tipo de servicio que todos y cada uno de tus clientes necesitan. Como se introdujo anteriormente, la posibilidad de capturar y analizar datos permite que las empresas alcancen un mayor nivel de entendimiento de cómo sus clientes utilizan sus servicios. Este nivel de entendimiento es posible gracias no solo a los procesos de seguimiento establecidos por las empresas para seguir el día a día de sus clientes, sino también a la inferencia de los motivos que permiten conocer por qué dichos servicios captan la atención continua de los clientes.

Comprados juntos habitualmente

Figura 5.5: Recomendación de Amazon basada en cómo los usuarios adquieren productos que están relacionados entre sí.

Y es que la personalización es un aspecto clave para ofrecer un servicio, y conforme pase el tiempo lo será aún más. Los clientes demandarán (de hecho, ya lo hacen) servicios personalizados, independientemente del dominio de aplicación de estos. Las empresas que ofrezcan servicios estarán obligadas inexorablemente a pivotar en torno al concepto de personalización. A modo de ejemplo, tenga en cuenta cómo algunos sectores tradicionales, como el de la banca, han comenzado este proceso de cambio empujado por nuevas empresas que fijan su modelo de negocio alrededor del término *fintech* (*financial technology*).

Introduzcamos ahora dos ejemplos representativos de servicios basados en la personalización: i) las recomendaciones personalizadas de AmazonTMy ii) las sugerencias de visualización de NetflixTM.

Amazon es un ejemplo claro de compañía que ha entendido el poder de la IA como eje fundamental de su modelo de negocio. Es evidente que no le faltan recursos para posicionarse como empresa de referencia, pero también resulta evidente que la dirección de la empresa ha sabido alinear en todo momento su estrategia corporativa con el uso de la IA. Dentro del amplio repertorio de aplicaciones de IA, como por ejemplo AlexaTM, destaca el **motor de recomendaciones** del gigante del comercio electrónico (ver figura 5.5). Y es que cuando compras un producto en Amazon, el sistema ya te está ofreciendo recomendaciones de compra que están relacionadas con tu última compra pero, al mismo tiempo, personalizadas en base a tu perfil. Este sistema hace uso de datos e información derivados de tu historial de búsqueda, del perfil de compradores similares a ti, y de aspectos demográficos, entre otras cuestiones.

Netflix es otro ejemplo claro de cómo la personalización en un servicio juega un papel principal. Una de las referencias de Netflix consiste en maximizar el tiempo que los usuarios emplean en su plataforma de *streaming* consumiendo contenidos, tanto en términos absolutos como en relación a otras plataformas de *streaming*. Para ello, el motor de recomendación analiza todas y cada una de las interacciones del usuario, y de usuarios parecidos, para sugerir cuál es la siguiente serie o película que deberías ver. Además de analizar esta información, que define la forma en la consumes contenido multimedia, Netflix considera toda **información de carácter individual** que pueda resultar útil para alimentar su motor de IA. El nivel de per-

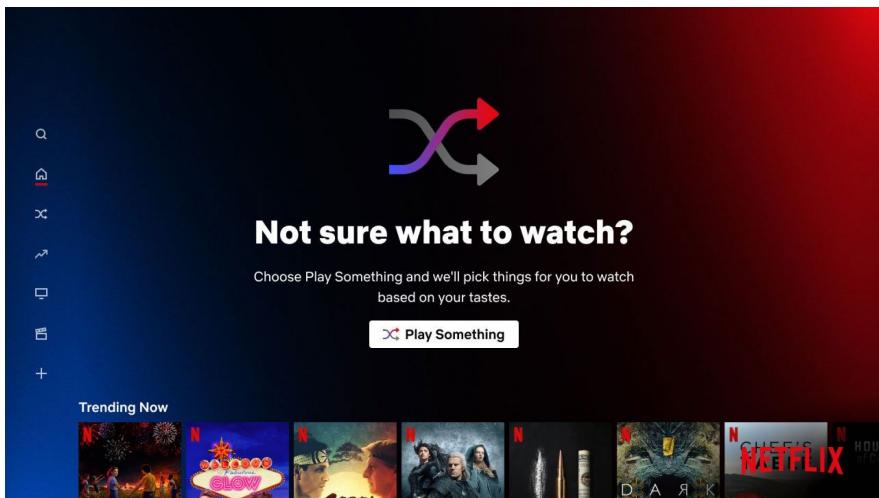


Figura 5.6: Recomendaciones de Netflix basadas en las preferencias de los usuarios.

sonalización es tan detallado, que las propias imágenes utilizadas para representar visualmente un contenido se adaptan a las preferencias del usuario. Por ejemplo, Netflix decidirá, para cada usuario, si en una película protagonizada por Al Pacino tiene más garantía de éxito presentar una foto de dicho actor o emplear un fotograma representativo de la película. Esto dependerá, entre otros aspectos, de si un usuario cualquiera dirige su historial de búsqueda por el nombre del actor y no tanto por el tipo de película en la que participa.

Clasificación de servicios inteligentes

A continuación se muestra una posible clasificación de servicios inteligentes. No se trata de un listado exhaustivo, pero sirve como referencia a la hora de entender cómo las empresas están aumentando la capacidad de sus servicios con el uso de IA [Mar20]:

- **Servicios financieros inteligentes.** La tecnología financiera, abreviada como *fintech*, es una industria emergente que hace uso de la tecnología, y particularmente de la IA, para mejorar la actividad en el ámbito financiero. Algunos ejemplos concretos son la banca móvil, la inversión o la gestión de créditos. Y es que, aunque el sector bancario es un gigante que se mueve muy despacio, en los últimos años se está generando todo un ecosistema de empresas que están empezando a competir en este sector. Parte del valor añadido se basa en ofrecer servicios personalizados a los usuarios finales. Un ejemplo recurrente es el análisis automático de los hábitos de gasto, que puede derivar en ofrecer recomendaciones para mejorar la salud financiera de los clientes

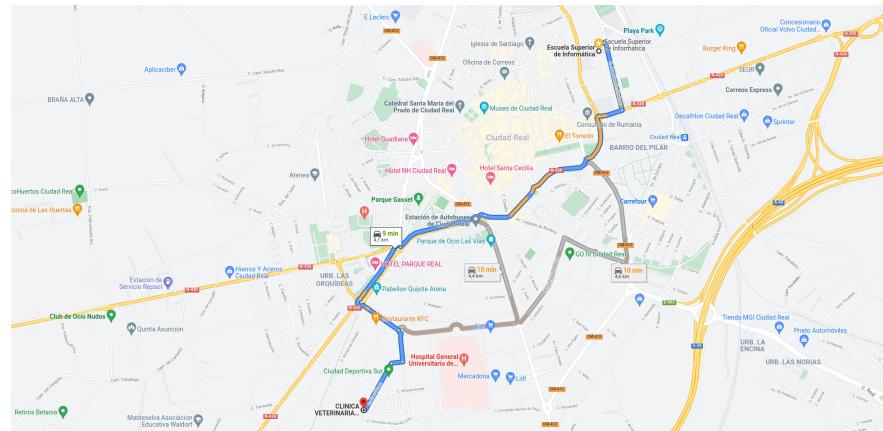


Figura 5.7: Análisis de Google Maps™ relativo a la congestión de tráfico a la hora de calcular una ruta.

e incluso en gestionar los ahorros, o el presupuesto, de manera automática. Otro ejemplo representativo en este campo es el de los *robo-advisors*, que automatizan los servicios de asesoramiento y gestión de carteras de inversión en función del perfil del usuario.

- **Transporte inteligente.** Si bien el transporte se puede entender desde el punto de vista de un producto (piense en su coche, si lo tiene), también se puede abordar desde la perspectiva de un servicio con el valor añadido de la IA. Un ejemplo sería el creciente modelo de negocio basado en alquiler o *renting* de vehículos, el cual, de nuevo, puede verse beneficiado de la capacidad de analizar datos para mejorar el servicio en base a los hábitos de uso de los clientes. Otro ejemplo de servicio inteligente en el ámbito del transporte es el relacionado con el análisis automático y en tiempo real de tráfico, que permite optimizar rutas de transporte en un sinfín de aplicaciones.
- **Seguros inteligentes.** Se prevé que las compañías de seguros hagan uso de IA para estrechar la relación existente con sus asegurados, más allá incluso de personalizar los servicios existentes o de ofrecer servicios personalizados. Actualmente, ya existen compañías de seguros de salud que basan su modelo de negocio en premiar a aquellos clientes que siguen hábitos de vida saludables. Este planteamiento no solo permite rebajar la cuota del seguro, sino que también reduce la posibilidad de que el cliente enferme, precisamente gracias al hecho de seguir dichos hábitos. El uso de IA y de dispositivos wearables, para medir el estado y la actividad física del cliente, conforman el núcleo de este tipo de propuestas.
- **Salud inteligente.** El uso de IA en el sector de la salud se está convirtiendo, poco a poco, en un catalizador para facilitar el acceso a una sanidad universal de calidad. Aunque pueda parecer evidente que aún queda un largo camino por recorrer, la realidad es que ya existen servicios inteligentes de diagnóstico que rivalizan, o incluso son más efectivos en ciertos ámbitos,

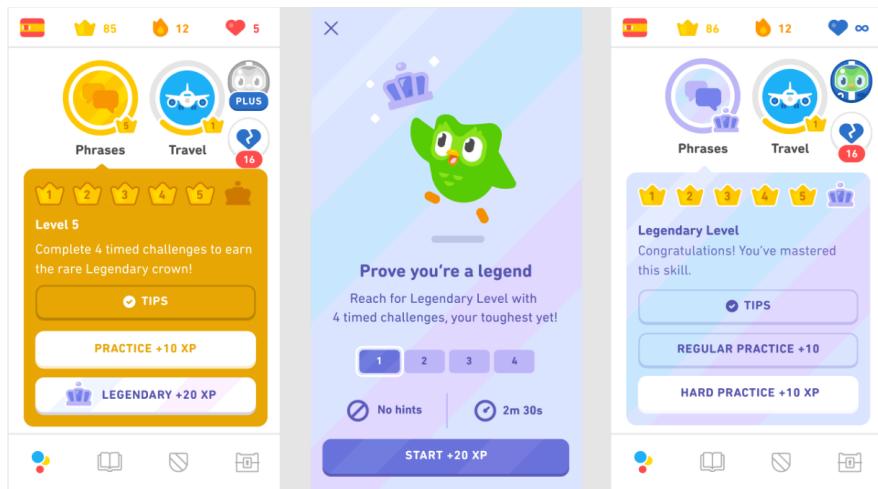


Figura 5.8: Sistema de niveles de Duolingo™ para el aprendizaje personalizado de idiomas.

con el diagnóstico realizado por un profesional humano. De nuevo, el análisis automático de grandes volúmenes de datos permite inferir conocimiento. Esto es especialmente relevante en regiones físicas donde existe una gran limitación de médicos; la tecnología y, particularmente, la IA pueden marcar la diferencia.

- **Educación inteligente.** La personalización en el proceso de aprendizaje aporta un valor añadido, considerando la importancia que tiene hoy en día la necesidad de estar constantemente aprendiendo (y desaprendiendo). En este ámbito existen innumerables ejemplos de cómo la IA puede contribuir, desde la integración de módulos de IA para analizar automáticamente el comportamiento y el rendimiento de cada estudiante, de manera individual, hasta la capacidad de establecer, automáticamente, el punto de partida de un estudiante a la hora de afrontar un curso o recurso de aprendizaje.
- **Citas inteligentes.** Las relaciones personales, y la forma en la que interactuamos, se han visto afectadas por cómo usamos la tecnología (o, en ocasiones, cómo la tecnología dirige la forma en la que interactuamos). En este sentido, el sector de los portales de citas también está apostando por el uso de la IA para mejorar las recomendaciones ofrecidas, entendiendo por recomendación el *matchmaking* generado por la plataforma con respecto a dos personas a la hora de formar una potencial pareja. Uno de los usos más recientes de la IA en este dominio es el de la detección automática de información incorrecta, o falsa, en los perfiles de sus usuarios. Para ello, estas plataformas analizan la forma en la que el usuario actualiza su perfil de usuario. Recuerde, de nuevo, que es posible capturar los datos que todo usuario ofrece, identificar cuándo lo hizo y realizar suposiciones de por qué lo hizo.

- **Moda inteligente.** Desde el punto de vista del cliente final, el proceso de compra de ropa puede ser una actividad recreativa relajante y placentera. Desafortunadamente, en la una parte significativa de las ocasiones, comprar ropa implica tiendas llenas de personas, largas esperas en la cola y probadores en los que no hay tiempo, ni espacio, para enriquecer la experiencia del usuario. Actualmente, existen empresas que ofrecen servicios que tienen como fin mitigar los aspectos negativos del proceso de compra, facilitando la vida a los usuarios. Piense, por ejemplo, en la opción de ofrecer, por parte del cliente, información como el tipo de ropa que le gusta, información sobre su talla y complejión, o incluso un modelo tridimensional de su pie. Esta información se podría utilizar para personalizar qué ropa o calzado se ofrece a cada usuario, de manera individualizada.

5.1.4. Procesos de negocio inteligentes

Previamente, se introdujo la opción de utilizar IA para **mejorar los procesos internos de una empresa**, planteando así una perspectiva interna que permite poner el foco en mejorar la operativa de la empresa. Evidentemente, esto tiene un impacto directo sobre los empleados, aunque también mantiene un impacto indirecto sobre los clientes finales.

El uso de IA sobre los procesos de negocio está estrechamente ligado al **concepto de automatización**. No obstante, es importante destacar que el hecho de automatizar una tarea o un proceso no implica, necesariamente, hacer uso de IA. En este sentido, las operaciones de negocio mejoradas o aumentadas por IA permiten tanto aumentar la capacidad de trabajo de los humanos como estructurar flujos de trabajo para optimizar la relación entre humano y máquina.

La adopción de IA para incorporar procesos de negocio inteligentes introduce ventajas como las que se exponen a continuación [Mar20]:

- Reducción de costes.
- Automatización e incremento del nivel de paralelismo de las actividades de negocio que conforman el núcleo operativo de una empresa.
- Optimización del tiempo de calidad dedicado por los trabajadores humanos a tareas en los que aporten un mayor valor a la empresa, gracias a la reducción de tareas puramente repetitivas.
- Mejora de la satisfacción del cliente final.
- Control sobre el aumento de ventas y su valor de retorno.

El impacto de la automatización robótica de procesos

En los últimos años, la *automatización robótica de procesos* o RPA (Robotic Process Automation) ha sido una de las principales soluciones diseñadas por las empresas para incrementar al rendimiento y reducir costes. Es importante destacar que el término *robótica* no implica, necesariamente, el uso de robots físicos.



Figura 5.9: Camión autonómico Volvo Autonomous Refuse Truck, ideado para automatizar la recogida de basura en contenedores, en pruebas.

Realmente se refiere al uso de ordenadores para automatizar, parcial o totalmente, los procesos internos de una empresa. Esta **automatización** suele estar centrada, en primera instancia, en **procesos repetitivos** y estructuras en los que los trabajadores humanos invierten su tiempo (ver ejemplo en la figura 5.9). La ventaja de la automatización resulta evidente en este contexto, debido a la naturaleza de estas tareas y a la capacidad de liberar al trabajador humano de trabajos repetitivos (que, potencialmente, son propensos a generar errores).

En este contexto de automatización de procesos, el término RAAS (Robot-Ass-A-Service) se ha hecho un hueco en el mercado [Mar20]. Este se refiere a la capacidad de contratar *bots* especializados en una determinada tarea, los cuales están comúnmente desplegados sobre una infraestructura *cloud*.

A continuación se listan algunos ejemplos en los que RPA aporta un valor añadido con respecto a otras soluciones más tradicionales [Mar20]:

- Comunicación automatizada entre sistemas digitales dentro del ámbito financiero para capturar datos y ejecutar transacciones bancarias.
- Centralitas y soporte telefónico a clientes.
- Digitalización de documentos físicos.
- Respuestas automatizadas en centros de ayuda técnica.
- Tratamiento de solicitudes de tarjetas de crédito.
- Soporte a recursos humanos para la contratación de personal.

5.1.5. Reflexión

En el capítulo 2, y particularmente en la sección 2.3, se introdujeron los fundamentos del término AIAAS (Artificial Intelligence as a Service), el cual enlaza con lo discutido en la presente sección desde el punto de vista de la definición de estrategias corporativas basadas en IA. Merece la pena reflexionar sobre la idea de ofrecer la IA como el servicio, en comparación de ofrecer un servicio basado en IA.

Las grandes tecnológicas han entendido perfectamente cómo el modelo de negocio de muchas empresas ha cambiado, y no han dudado en ofrecer su infraestructura tecnológica y de comunicaciones para proporcionar servicios transversales como los basados en chatbots, APIs para la computación cognitiva, *frameworks* de aprendizaje automático, e incluso modelos predefinidos para integrarlos en dichos *frameworks*. De nuevo, los datos juegan un papel fundamental en todo este proceso, y se plantean retos como la seguridad, la dependencia con terceras partes, e incluso la transparencia a la hora de conocer lo que ocurre realmente con ellos.

5.2. Modelos de negocio

5.2.1. Introducción

Después de introducir, en la sección 5.1, cómo la IA puede actuar de catalizador para definir la estrategia corporativa de una empresa, la presente sección pone el foco en la generación de modelos de negocio donde la IA juega un papel representativo. En este sentido, se volverá a incidir en cómo el **uso de IA** y de mecanismos basados en **automatización inteligente** está empezando a convertirse, desde el punto de vista práctico, en una obligación.

Posteriormente, en la sección 5.2.3, se abordará un caso de estudio en el que se discute un modelo de negocio basado en el uso de IA. Este caso de estudio está centrado en la creación de una herramienta tecnológica para la rehabilitación física de pacientes afectados por enfermedades neurológicas. Se pretende ilustrar al lector, de forma sintetizada y directa, con algunas de las cuestiones más significativas a la hora de plantear un reto tan ambicioso.

5.2.2. Reinvención del modelo de negocio

En el libro *Intelligent Automation* se incluye un análisis muy representativo del impacto que tanto la IA como la automatización inteligente han tenido en los últimos años [BBW21]. Este análisis reflexiona sobre el rendimiento económico de las empresas top del ranking *Fortune 500*, considerando datos del año 2020. En concreto, el elemento que se analiza es el beneficio por empleado, tal y como se muestra en la tabla 5.10.

Posición	Empresa	Beneficio	Empleados	Beneficio por empleado
1	Facebook	18.485 M\$	44.942	411.308 \$
2	Apple	55.256 M\$	137.000	403.328 \$
3	Alphabet	34.343 M\$	118.899	288.842 \$
4	Microsoft	39.240 M\$	144.000	272.500 \$
5	AbbView	7.882 M\$	30.000	262.733 \$

Figura 5.10: Lista de las 5 empresas del ranking *Fortune 500* que más beneficio generan por empleado. Las cantidades económicas están expresadas en dólares americanos. Fuente original: Tipalti, 2020. "Profit per Employee". <https://tipalti.com/profit-per-employee>

Los datos de esta tabla se pueden comparar con los reflejados en la tala 5.11, de nuevo expuestos en el libro *Intelligent Automation* y recogidos por los autores, para reflexionar sobre la correlación existente entre el beneficio por empleado y la antigüedad de la empresa. Como se puede apreciar, el beneficio por empleado desciende cuanto más antigua es la empresa.

Fundación	Empresa	Beneficio	Empleados	Beneficio por empleado
1911	IBM	9.431 M\$	383.800	24.573 \$
1940	McDonald's	6.025 M\$	205.000	29.392 \$
1968	Intel	21.048 M\$	110.800	189.964 \$
1975	Microsoft	39.240 M\$	144.000	272.500 \$
1976	Apple	55.256 M\$	137.000	403.328 \$
1998	Alphabet	34.343 M\$	118.899	288.842 \$
2004	Facebook	18.485 M\$	44.942	411.308 \$

Figura 5.11: Lista global de las empresas que más beneficio generan por empleado. Las cantidades económicas están expresadas en dólares americanos. Fuente: Intelligent Automation, 2021 [BBW21].

La principal conclusión que se puede obtener de este análisis es que el funcionamiento de las principales empresas, a nivel global, está cambiando, y este cambio encaja con un **modelo de negocio en el que cada vez intervienen menos empleados**, donde los procesos están cada vez más automatizados y el uso de IA cobra, cada vez más, un mayor protagonismo. Evidentemente, existen otros aspectos a tener en cuenta, como la potencial dificultad de las empresas más asentadas para adaptarse a la transformación digital. Cuando más joven es la empresa, menor es la inercia de la misma con respecto a procesos y estrategias heredadas de épocas donde la digitalización no tenía tanto protagonismo.

Es inevitable pensar que, en los próximos años, todas las empresas evolucionarán hacia un modelo digital donde la automatización y la IA tendrán cada vez más presencia. Además, la intervención humana irá perdiendo peso. Un ejemplo representativo de este cambio es Amazon, que actualmente emplea, aproximadamente, a 400.000 personas y a 200.000 robots. Esto último número asciende más rápido del que lo hace el relativo al personal humano.

En este sentido, los modelos de negocio han de adaptarse a este cambio para que las empresas que los definen sean competitivas. En el siguiente apartado se discute, de manera sintetizada, un caso práctico que trata de ilustrar este cambio en los modelos de negocio.

5.2.3. Caso práctico: rehabilitación remota de ejercicios físicos basada en IA

A continuación, se presenta un caso de estudio en el que se define un modelo de negocio que pivota en torno al **uso de IA para ofrecer una solución tecnológica que permita la supervisión remota de pacientes que requieren rehabilitación física**. Particularmente, se contempla un escenario en el que un terapeuta supervisa, simultáneamente, a varios pacientes afectados por alguna enfermedad neurológica que realizan ejercicios de rehabilitación física desde casa. Tanto la supervisión como la ejecución de ejercicios está guiada por un sistema software basado en IA y en visión por computador. Desde el punto de vista de la monetización, el cliente final es el terapeuta, y no el paciente.

En los siguientes apartados se expone el contexto en el que se encuadra este caso práctico, se detalla el modelo de negocio propuesto, y se discute la arquitectura hardware y software que soporta la funcionalidad disponible tanto para terapeuta como para paciente.

Contexto y problemática

La rehabilitación física es una actividad esencial en el proceso de recuperación de varias enfermedades, como las neurológicas, las lesiones físicas o la recuperación tras una intervención quirúrgica. Este caso práctico está contextualizado para el caso de las **enfermedades neurológicas** y, particularmente, para la rehabilitación de personas que han sufrido un ictus. Este se produce normalmente cuando una parte del cerebro se ve repentinamente privada del suministro de sangre. Los pacientes afectados por el ictus quedan con efectos incapacitantes, como la pérdida de fuerza, movilidad o sensibilidad en algunas partes de su cuerpo.

Por lo tanto, la rehabilitación física a largo plazo es necesaria para mejorar su calidad de vida y recuperar la movilidad. Además, el tiempo es un factor crucial, ya que cuanto antes se les trate, mayor será la posibilidad de que los pacientes recuperen algún grado de movilidad [MFP10]. El ictus es una de las principales causas de mortalidad y discapacidad en el mundo, por no hablar de los costes económicos del tratamiento y la recuperación tras el ictus que conlleva [JNR⁺19]. Sólo en la Unión Europea, se estima que el número de personas que sufren un ictus aumentará un 27 % entre 2017 y 2047 [WWE⁺20]. Además, se prevé incluso que este problema se agrave debido al aumento de la edad de las personas mayores, lo que provocará un impacto negativo en los próximos años.

Desafortunadamente, la **problemática** asociada a la rehabilitación física de este tipo de pacientes sigue representando un **reto global** [WWE⁺20] debido, entre otras cuestiones, a la necesidad de llevar a cabo una terapia continuada de rehabilitación durante un periodo de tiempo significativo para recuperar, en la medida de lo posible, la calidad de vida que el paciente gozaba antes de sufrir el ictus.

Por otra parte, en **términos económicos**, hay un gran coste en relación con enfermedades neurológicas. Según el informe anual del informe de 2018 del *National Health Service* británico, y tomando como referencia el Reino Unido, los servicios cuidados curativos y rehabilitación asumieron más de la mitad del gasto sanitario público en 2016, alcanzando los 58.010 millones de euros (57,0 % del gasto sanitario total). En España, como ejemplo particular, el coste medio de un paciente afectado por un ictus se estimó en 27.711€ al año [ASQM⁺17]. Más de dos tercios se deben a los costes sociales. Además, según el informe del Consejo Europeo del Cerebro, más de 179 millones de europeos viven con trastornos neurológicos [DO14]. De hecho, se calcula que 1 de cada 3 sufrirá algún trastorno neurológico o psiquiátrico a lo largo de su vida. Por desgracia, el impacto socio-económico es incluso mucho más acentuado en países de ingresos bajos y medios.

Por lo general, los pacientes con ictus no realizan la rehabilitación solos. En su lugar, cuentan con el apoyo de terapeutas en sesiones cara a cara de forma regular. Esto supone un problema para pacientes cuyo estado de salud les impide asistir a sesiones en el centro de rehabilitación, situación que se ve muy afectada por contextos en los que las restricciones de movilidad pueden impactar (como situaciones de pandemia). Por otra parte, las sesiones de rehabilitación que se ofrecen en los hospitales o centros de rehabilitación a veces no son suficientes para los pacientes, ya que son limitadas en duración. Además, existe una barrera relacionada con la motivación y el compromiso. Los ejercicios físicos tradicionales consisten en realizar ejecuciones repetitivas de movimientos correctos que tienden a ser monótonos y aburridos. Esto puede hacer que los pacientes pierdan la motivación y, en consecuencia, su compromiso con la terapia [JMMG10]. Por desgracia, esto puede afectar a la calidad de la terapia.

Por estos motivos, surge la **telerehabilitación**. Esta rama de la telemedicina permite realizar tratamientos de la fase aguda de la enfermedad sustituyendo las sesiones presenciales tradicionales con una rehabilitación a domicilio (ver figura 5.12). Existe un importante número de aplicaciones bajo este enfoque, generalmente aplicadas a la fisioterapia, donde se utilizan técnicas de realidad virtual (RV) para que el paciente imite los movimientos de un avatar virtual. En esencia, se trata de un enfoque relevante para la rehabilitación en casa. Sin embargo, hay que tener en cuenta que cuando la responsabilidad de realizar estos ejercicios se delega en el paciente, en casa y sin la supervisión directa del terapeuta, la precisión de los ejercicios que se ejecutan puede ser baja, lo que puede afectar a la calidad de la rehabilitación. En este sentido, es posible que no se realicen bien en casa, lo que puede llevar a situaciones no deseadas en las que los efectos de la rehabilitación pueden ser negativos.

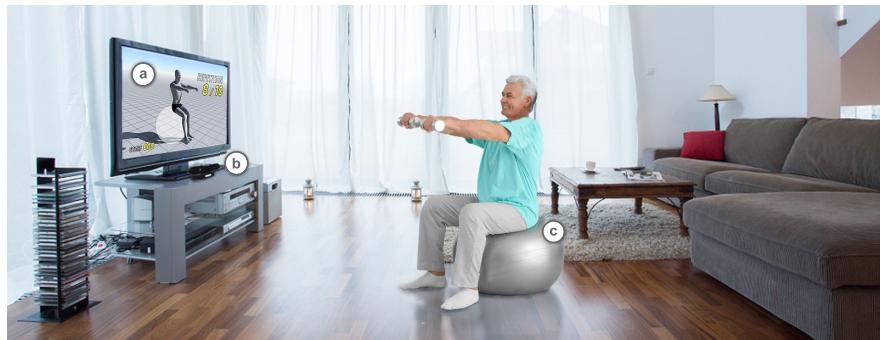


Figura 5.12: Representación gráfica del entorno donde el paciente realiza los ejercicios de rehabilitación en casa: (a) TV conectada al dispositivo local de procesamiento. (b) Dispositivo autónomo de seguimiento del esqueleto del paciente. (c) Paciente jugando a un exergame o juego terapéutico.

Modelo de negocio

A continuación, se presenta la descripción de lo que sería la **información fundamental para definir el modelo de negocio** previamente introducido, que gira en torno al **uso de IA para ofrecer una solución tecnológica** orientada a la supervisión remota de pacientes que requieren rehabilitación física. Dicha información, resumida, se estructura en los siguientes puntos:

- Descripción general de la tecnología.
- Análisis de mercado.
- Propiedad intelectual.
- Plan de negocio.
- Aspectos de igualdad, diversidad y ecología.

Descripción general de la tecnología

Se propone una solución tecnológica basada en la nube para la rehabilitación física de pacientes con ictus. La plataforma facilita la gestión clínica a través de la **monitorización activa y la analítica de datos** para el entrenamiento de tareas repetitivas del miembro superior en casa. La interfaz del clínico permite i) definir y asignar rutinas de rehabilitación, ii) monitorizar offline el progreso de los pacientes, e iii) interactuar con ellos a través de videollamada en directo, si es necesario. La interfaz del paciente muestra la terapia prescrita e incorpora elementos de tecnología de juegos para motivar y comprometer a los pacientes.

El paciente sólo necesita un ordenador portátil y un dispositivo que integra un **kit de desarrollo con sensores de inteligencia artificial avanzados** para llevar a cabo la rehabilitación en casa, y la plataforma le guía a través de sencillas instrucciones y pautas de seguridad. La plataforma sigue y mide con precisión el movimiento de las extremidades y proporciona al clínico una medida objetiva del

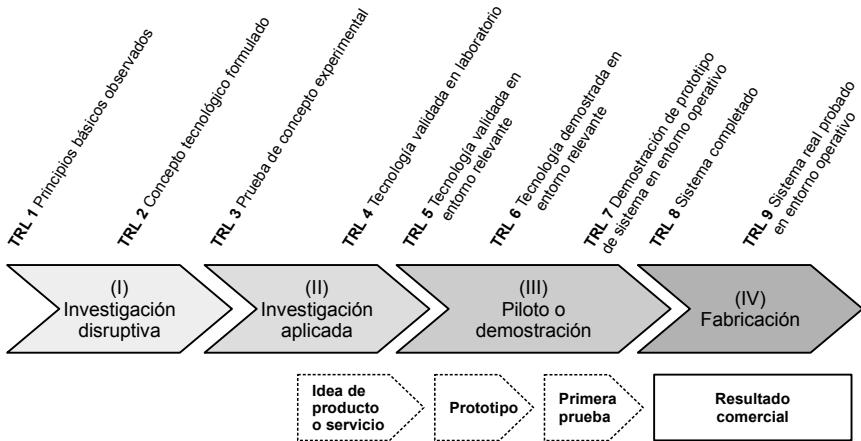


Figura 5.13: Descripción visual de los diferentes niveles de madurez tecnológica TRL (Technology Readiness Levels).

progreso. El clínico puede trabajar con uno o varios pacientes remotos simultáneamente con comunicación audiovisual en directo o sin conexión, en cuyo caso la evaluación del paciente se ve más tarde. El sistema garantiza que se reciba la cantidad de terapia recomendada con un tratamiento de fisioterapia *supervisado* más frecuente y con un coste reducido. Las rutinas de rehabilitación pueden ser personalizadas por el terapeuta.

En cuanto al estado actual de desarrollo, se desarrolló un prototipo² que fue evaluado por clínicos y pacientes, tanto en el Reino Unido como en España. Las pruebas realizadas con pacientes permitieron identificar problemas de usabilidad y accesibilidad. Asimismo, un análisis de mercado indicó las vías de comercialización previas a un ensayo clínico dentro del NHS (National Health Service (UK)). Así, el prototipo desarrollado puede considerarse una prueba de concepto temprana demostrada en el laboratorio (TRL (Technology Readiness Levels) 3). El siguiente ciclo de desarrollo está representado por la plataforma propuesta, que combina las mejoras identificadas con la experiencia adquirida y las características aportadas por el prototipo anterior, y tiene como objetivo el TRL 4/TRL 5 (ver figura 5.13).

Los resultados esperados como consecuencia de desplegar el modelo de negocio planteado son los siguientes: i) facilitar el acceso a la tecnología por parte de los pacientes con ictus, ii) incrementar la frecuencia de la fisioterapia *supervisada* por parte de los pacientes con ictus al realizar la rehabilitación en casa, iii) facilitar a los clínicos la supervisión a distancia de múltiples pacientes con ictus, iv) aumentar el tiempo de calidad dedicado por los clínicos a los pacientes con ictus, v) garantizar el compromiso de los sistemas nacionales de salud con la tecnología propuesta.

²<https://www.virtualphysioproject.com/>

Por último, se han identificado una serie de **beneficios** para sistemas nacionales de salud y el mercado en general: i) reducir los costes, ii) llegar a (más) pacientes, especialmente al salir del hospital, iii) aumentar el acceso a la tecnología en las zonas rurales, iv) proporcionar a los fisioterapeutas una tecnología de vanguardia, v) ofrecer una tecnología fácilmente escalable a otras afecciones neurológicas.

Análisis de mercado

En primera instancia, se han analizado decenas de competidores en el mercado, y una de las principales conclusiones obtenidas es que no es fácil tener acceso a los precios. Algunos de los competidores conocidos en estos mercados son, por ejemplo, RGS@home, Neofect, Kaia Health, EvolvRehab, CurvHealth, Rehametrics, Sword Health, Phio, BTS Telerehab, Reactive Rehab, Goodlife Technology, entre otros que han sido descubiertos *online*. Sin embargo, la mayoría de ellas ofrecen soluciones pensadas para centros hospitalarios o de rehabilitación, donde el terapeuta supervisa a los pacientes cara a cara.



Figura 5.14: Paciente haciendo uso de la plataforma EvolveRehab.

Desde el punto de vista de la tecnología empleada, gran parte de los sistemas estudiados utilizan, o han iniciado el proceso de migración, el sistema Azure Kinect DK™(ver figura 5.15). Este representa la punta de lanza de los dispositivos punteros en lo que se refiere a análisis de imagen y, particularmente, *tracking* o seguimiento de esqueletos.

El hecho de que la competencia ya exista es, en realidad, algo positivo, porque estos *pioneros* han abordado la peor parte: la introducción de un nuevo concepto a las masas, basado en conseguir que la gente lo acepte. Por lo tanto, esta barrera de entrada ya está siendo rota por otros competidores existentes. Al mismo tiempo, el hecho de que haya competidores supone una barrera de entrada evidente, porque no

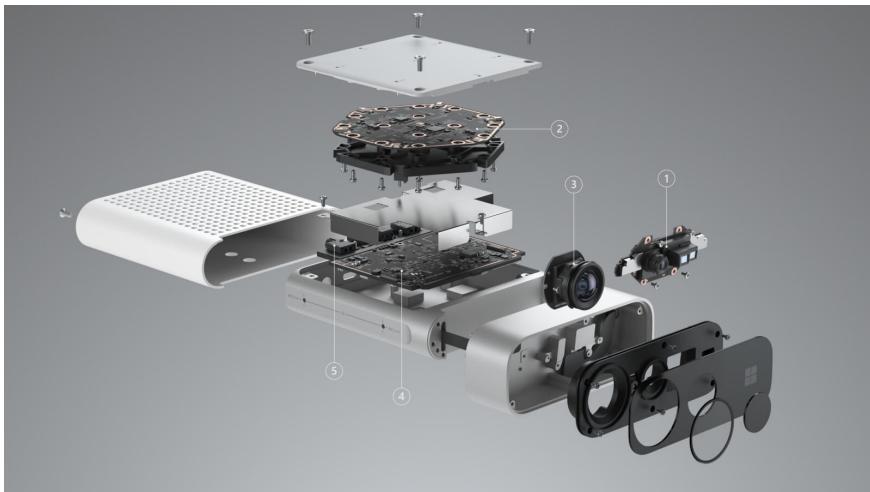


Figura 5.15: (1) Sensor de profundidad de 1 MP con opciones de campo de visión ancho y estrecho. (2) Matriz de 7 micrófonos para capturar sonidos y voz de campo lejano. (3) Cámara de video RGB de 12 megapíxeles para obtener una secuencia de colores adicional en línea con la secuencia de profundidad. (4) Acelerómetro y giroscopio para la orientación del sensor y el seguimiento espacial. (5) Conexiones de sincronización externas para proporcionar sincronización con otros dispositivos Kinect. Fuente: <https://azure.microsoft.com/es-es/services/kinect-dk>

solo es necesario evaluar los precios en función de las soluciones existentes, lo que ha resultado ser duro, sino que además se debe intentar tener algo *disruptivo*. En este contexto, los **puntos de venta únicos de la propuesta basada en el presente modelo de negocio** son los siguientes:

- **bajo coste**, ya que la solución diseñada no depende de dispositivos de hardware cuyo precio sea elevado, más allá del dispositivo de tracking Azure Kinect DK y un PC conectado a Internet,
- **accesibilidad**, ya que el software se basa principalmente en tecnología web y se despliega en la nube,
- **capacidad de personalización**, ya que el sistema facilita la gestión de los ejercicios y los planes de rehabilitación en función del estado y la evolución del paciente. También se pretende orientar esta característica para que la plataforma sea capaz de aprender y adaptar automáticamente la rutina de rehabilitación a cada paciente utilizando IA.

Otras barreras son el hecho de que los usuarios finales son en su mayoría personas mayores y, por lo tanto, supone un gran reto no sólo poder ofrecer una solución lo suficientemente fácil de entender y utilizar para las generaciones mayores.

Propiedad intelectual

Para este caso de estudio se debe considerar que el software no es patentable en Europa. Así, en esta primera aproximación, el trabajo estaría centrado en los aspectos tecnológicos y de validación del modelo de negocio. En particular, se prestaría especial atención a la evaluación preliminar del software, ya que el hardware necesario, como se comentará más adelante, no precisa una validación específica porque no se abordaría un proceso de creación del mismo. Por lo tanto, no se considerará la protección de la propiedad intelectual en relación a potenciales patentes.

Sin embargo, y como se ha mencionado anteriormente con respecto a la capacidad de personalización, consideramos la integración de la IA y, particularmente, de una versión preliminar de un algoritmo de aprendizaje automático que sentará las bases de este módulo funcional. Este será un resultado que podrá ser considerado para una futura protección de la propiedad intelectual.

En cuanto a la gestión de la propiedad intelectual, el sistema combina algoritmos de código abierto (bajo licencias que permiten comercializar los resultados) y propietarios. Se optaría por la denominada *publicación defensiva (defensive publishing)*. Así, se optaría por la generación y publicación de un *white paper*, con contenido técnico, que reflejaría los aspectos más representativos del software y, en concreto, del módulo de personalización individualizada de rutinas.

Plan de negocio

Con respecto a la **comercialización** y el plan de negocio, la primera etapa se podría basar en una **fase de pre-venta** en la que se define un precio bajo, o con descuento, para establecer una relación estrecha con un conjunto selecto de clientes. Esta estrategia estaría planteada con un doble objetivo: i) refinar y validar el prototipo software generado y ii) generar una sensación de exclusividad que facilitara el aterrizaje de los primeros clientes. Es importante destacar la necesidad de generar un flujo de caja en efectivo que permita financiar la mejora del primer prototipo software generado, particularmente todo aquello relacionado con la experiencia del usuario.

Para llegar a estos clientes iniciales, se realizaría una **inversión en redes sociales** y se contactaría directamente con terapeutas que trabajen en clínicas donde el componente tecnológico y de innovación sea representativo. Esta primera etapa del plan de negocio, estimada con una duración de 6 meses, sentaría las bases para lanzar el producto final y planificar diferentes estrategias de marketing, como anuncios en línea y la contratación de vendedores para llegar a los clientes directamente.

Con respecto a la **segmentación del mercado y estrategia de precios**, el mercado principal está claramente definido, ya que estaría centrado en clínicas de fisioterapia y los fisioterapeutas, en general. Los precios calcularían en base a los pocos competidores identificados en la fase de análisis de mercado, aunque no se ha podido identificar un precio concreto para una plataforma basada en servicios (SAAP (Service-As-A-Platform)). Esta situación complica la definición adecuada de un precio inicial.

Con respecto a **posibles barreras de adopción de la tecnología propuesta**, el principal reto es la fragmentación o dispersión de las entidades dedicadas a la rehabilitación física. En definitiva, hay que dirigirse a cada organización por separado. Por otro lado, se considera la situación en la que los fisioterapeutas son reacios a utilizar una nueva tecnología, debido a la falta de confianza en el uso de la misma y el miedo a que les sustituya o incluso sean prescindibles en su trabajo. Finalmente, también es importante considerar la potencial barrera tecnológica en la que los pacientes podrían reflejar que no tienen confianza en la tecnología propuesta.

En relación a la **capacidad de atraer clientes y generar confianza** con respecto a este primer prototipo software, se llevó a cabo una fase inicial de trabajo con médicos y supervivientes de accidentes cerebrovasculares. Particularmente, se organizaron eventos de consulta a los que asistieron neurólogos consultores del NHS en el Reino Unido, consultores en medicina de rehabilitación, fisioterapeutas y terapeutas ocupacionales. Los objetivos de estos eventos fueron comprender las necesidades clínicas, las vías de atención y las condiciones de interés. Los comentarios de los médicos fueron positivos. Se reclutó a supervivientes de accidentes cerebrovasculares para evaluar la versión preliminar del prototipo software actual desde el punto de vista de la usabilidad. Los comentarios recogidos de los supervivientes de accidentes cerebrovasculares se registraron mediante cuestionarios de salud estandarizados. Los comentarios fueron en general positivos, con sugerencias para mejorar la usabilidad.

Aspectos de igualdad, diversidad y ecología

El modelo de negocio planteado considera tres áreas de desigualdades sanitarias sobre las que incide la tecnología planteada: i) el estado de salud, por ejemplo, para mejorar la calidad de vida y los resultados con una terapia supervisada de forma más frecuente, ii) el acceso a la atención sanitaria, que permite a más pacientes acceder a la fisioterapia durante más tiempo, y iii) la calidad y la experiencia de la atención, mejorando la experiencia y la satisfacción del paciente con una interacción a medida entre el paciente y el sistema.

La tecnología puede contribuir directamente a **reducir las desigualdades en materia de salud** al permitir una rehabilitación de bajo coste en casa. Los estudios demuestran que los supervivientes de accidentes cerebrovasculares no reciben la cantidad recomendada de fisioterapia por razones como los costes y la accesibilidad, así como la propia escasez de fisioterapeutas, con carácter general. La tecnología planteada ofrece una opción de bajo coste para su uso en el entorno doméstico sin banda ancha de alta velocidad (descarga e instalación única de software).

Visión general de la arquitectura hardware/software

El modelo de negocio que previamente se ha introducido es el que se utilizará como referencia para discutir la arquitectura hardware y software propuesta para construir un potencial MVP (Minimum Viable Product).

En primer lugar, se abordará la dimensión hardware, que se puede apreciar de forma visual en la figura 5.16. A continuación se listan, y justifican brevemente, las **decisiones tomadas para diseñar el sistema hardware de rehabilitación remota**:

- Se ha optado por utilizar el dispositivo Azure Kinect DK, un dispositivo hardware que incorpora sensores avanzados para realizar el seguimiento del esqueleto del usuario y que está integrado con modelos de voz y visión artificial. Para utilizar el SDK de seguimiento de personas³, es necesario conectar el dispositivo a un equipo con un controlador gráfico Nvidia, de forma que se pueda hacer uso de hardware específico para llevar a cabo un *tracking* adecuado. Aunque existen otros dispositivos de *tracking* que no imponen este tipo de condicionantes, se establece como requisito no funcional la realización de un seguimiento de calidad, y para ello Azure Kinect DK representa la solución más avanzada.
- El diseño de caja negra, que incluye un botón para encender y apagar el equipo, junto con una pequeña pantalla de visualización en el panel frontal, tiene como objetivo reducir al máximo la complejidad a la hora de interactuar con el prototípico. La caja integra una salida de vídeo por HDMI para que sea posible conectar el equipo a un monitor externo (o a la televisión del salón).
- La caja negra también incorpora una clavija para conectar el equipo a la red eléctrica mediante un cable estándar. Esta conexión dota de alimentación al mini-PC que integra la tarjeta gráfica Nvidia, el cual, a su vez, sirve para alimentar al dispositivo de *tracking* mediante una conexión USB.

Por otra parte, y con respecto a la **dimensión software del prototipo** planteado en relación al modelo de negocio diseñado, la figura 5.17 muestra la **interfaz gráfica** generado con el motor de videojuegos multiplataforma Unity™. A continuación, se listan las principales decisiones tomadas a la hora de abordar dicho diseño:

- El avatar virtual sirve para que el usuario sepa cómo está moviendo su cuerpo, en todo momento, cuando efectúa la rehabilitación física. En la figura 5.17 se muestra cómo la parte relevante para realizar el ejercicio planteado es la mano, remarcada en rojo.
- El esquema de interacción es muy sencillo, y se basa en llevar a cabo colisiones de la parte del cuerpo marcada en rojo con las diferentes esferas virtuales que conforman la trayectoria de un ejercicio. Estas están numeradas para facilitar que el usuario sepa por dónde debe mover una determinada articulación. Se considera el concepto general de restricción, que permite asociar una articulación con otras, para que el usuario ejecute los ejercicios con la técnica adecuada. A modo de ejemplo, se considera que sería posible indicar que la cadera debe estar fija cuando se mueve el brazo.

³<https://docs.microsoft.com/es-es/azure/kinect-dk/body-sdk-download>

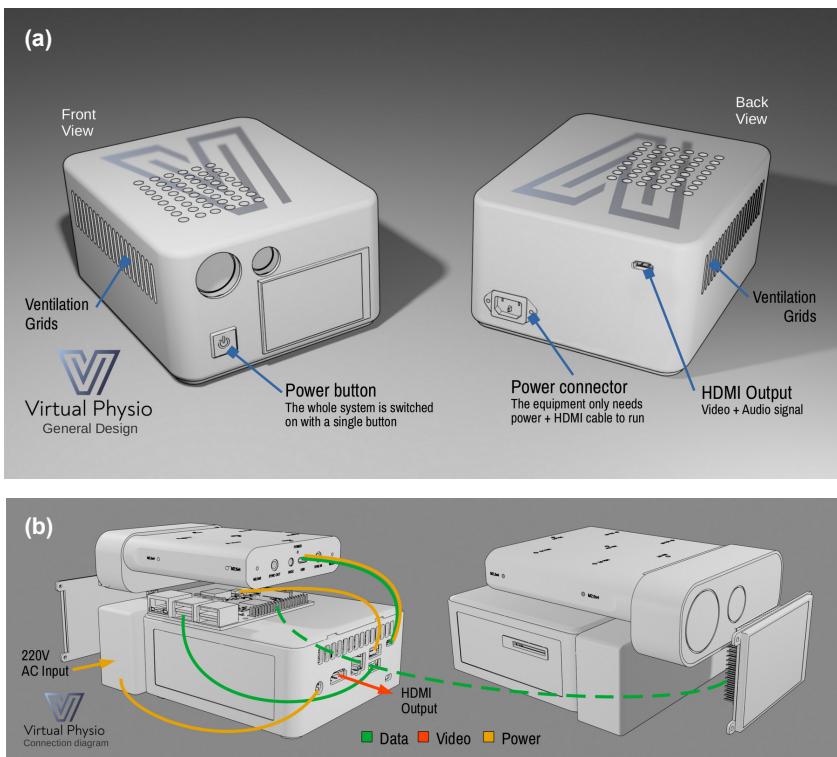


Figura 5.16: Prototipo hardware para la rehabilitación remota en casa. a) Diseño de una *caja negra* que integra los componentes físicos necesarios para desplegar el sistema software. b) Interior del prototipo, en el que se muestra el dispositivo de *tracking* Azure Kinect DK conectado a un mini-pc que ejecuta el software.

- Además de ofrecer una interfaz basada en mecanismos naturales de interacción, ya que el sistema detecta el movimiento del usuario sin la necesidad de sensores físicos, el prototipo posibilita la interacción mediante comandos de voz. De hecho, en la parte superior derecha de la interfaz aparecen algunos de los comandos más frecuentes, y que el sistema es capaz de reconocer cuando el usuario los emite. En la parte superior de la interfaz aparece un vómetro visual para ofrecer *feedback* al usuario cuando está hablando.
- El prototipo muestra información básica asociada a la ejecución de una rutina, como el ejercicio en curso, el número de repeticiones, la puntuación y el tiempo. Estos últimos se aprecian en la parte izquierda de la figura 5.17.

La figura 5.18 muestra, de manera resumida, la arquitectura interna del prototipo software para la rehabilitación remota desde una perspectiva de **flujo de información** entre módulos. Esencialmente, se distinguen tres grandes pasos:

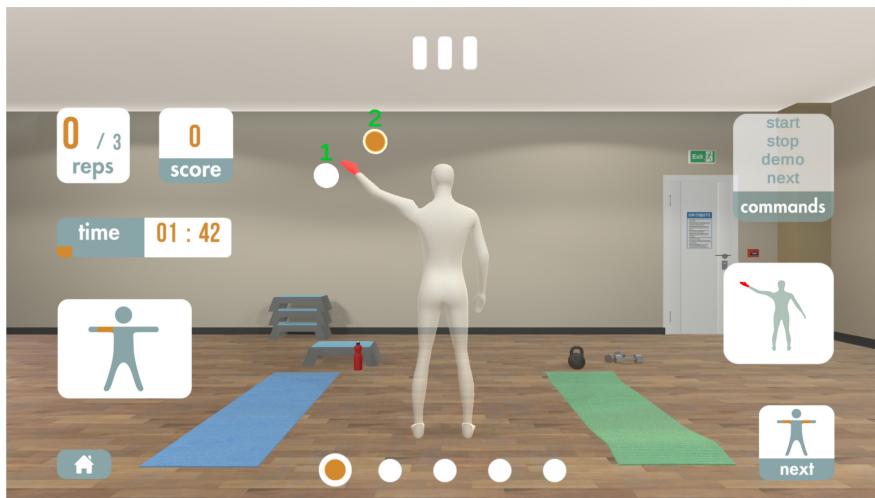


Figura 5.17: Interfaz gráfica del prototipo software para la rehabilitación remota en casa.

1. Captura de datos. El sistema obtiene y serializa los datos de *tracking* del esqueleto del usuario junto con los potenciales comandos de voz reconocidos. En realidad, los datos de *tracking* recogen las posiciones y orientaciones de las articulaciones del cuerpo del paciente en el espacio 3D.
2. Procesamiento de información. El sistema evalúa si el paciente está realizando correctamente el ejercicio de rehabilitación planteado, considerando el esquema de interacción previamente introducido y que se basa en la colisión de articulaciones con esferas virtuales. En este punto, el sistema gestiona aspectos de motivación mediante el uso de gamificación. Por ejemplo, el sistema generará un efecto visual y reproducirá una notificación sonora cuando el usuario ha sido capaz de completar una parte del ejercicio.
3. Visualización de información. El sistema ofrece *feedback* al usuario, de forma que este sepa cómo está realizando la rutina de rehabilitación asignada. Este *feedback* contiene aspectos como la puntuación, barras de progreso para ver la evolución temporal y desbloqueo de logros conforme el proceso avanza.

Desde el punto de vista de integración de IA, el sistema software permite habilitar un módulo para llevar a cabo tanto evaluaciones como clasificaciones automática de ejercicios físicos. Este enfoque dista del utilizado por defecto por el sistema, y que se ha simplificado de acuerdo a un mecanismo de interacción entre las articulaciones del paciente y la definición explícita de trayectorias virtuales que recrean ejercicios de rehabilitación.

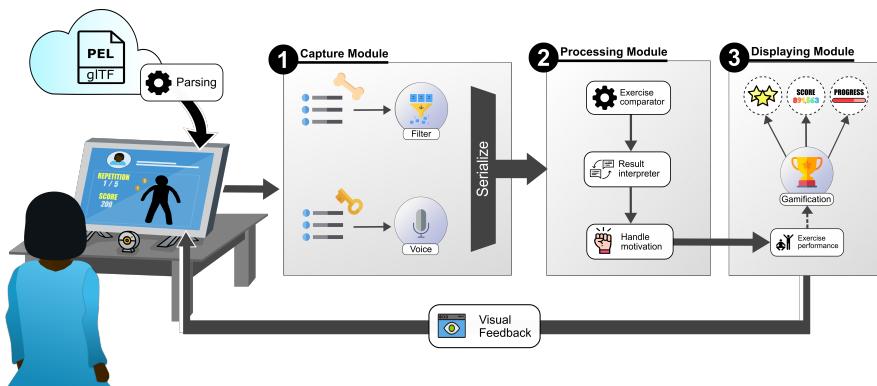


Figura 5.18: Principales módulos funcionales de la arquitectura interna del prototipo de rehabilitación remota, visualizados desde el punto de vista del flujo de información entre ellos.

Así, sería posible habilitar un modo de funcionamiento en el que **el sistema compara, automáticamente, el ejercicio del paciente**, en su conjunto, con la ejecución del mismo ejercicio realizado, de manera correcta, por un terapeuta. Para ello, el sistema implementa una variable del algoritmo DTW (Dynamic Time Warping), el cual posibilita la comparación de ejercicios que pueden variar en términos de velocidad. En otras palabras, si el paciente realiza correctamente un ejercicio de rehabilitación pero de forma más lenta en comparación con el *gold standard* (que es el ejercicio de referencia realizado por el terapeuta), el sistema será capaz de detectar que se trata del mismo movimiento, aislando la dimensión temporal. Este planteamiento resulta adecuado en el ámbito de la rehabilitación de pacientes afectados por accidentes cerebrovasculares, como ictus, especialmente en sus primeras etapas.

Particularmente, en el prototipo software se contempla una variante del algoritmo DTW, la versión FastDTW [SC07], que ofrece un orden temporal lineal de complejidad para analizar y evaluar automáticamente los ejercicios realizados por el paciente. El algoritmo proporciona la alineación óptima de dos secuencias temporales calculando una matriz de costes obtenida a partir de la diferencia entre dos índices de puntos de datos en las secuencias. El algoritmo elimina la dimensión temporal, proporcionando así resultados independientes de la diferencia de tiempo entre las dos secuencias. Esto lo hace especialmente útil para comparar series temporales, como el reconocimiento de voz y la sincronización de audio.

El movimiento vinculado a un ejercicio consiste en un conjunto de puntos marcados en el tiempo, uno por cada una de las articulaciones implicadas en el movimiento (ver la parte izquierda de la figura 5.19). Cada serie es una secuencia temporal de 3 tuplas $(x, y, z) \in R^3$, que indican la posición de la articulación asociada a la serie en un instante determinado, obtenida de la cámara del sensor de profundidad. Los valores de cada eje representan la posición de la articulación a lo largo del tiempo en ese eje y pueden verse como una trayectoria o curva. De este modo, el problema de la comparación de movimientos puede verse como un

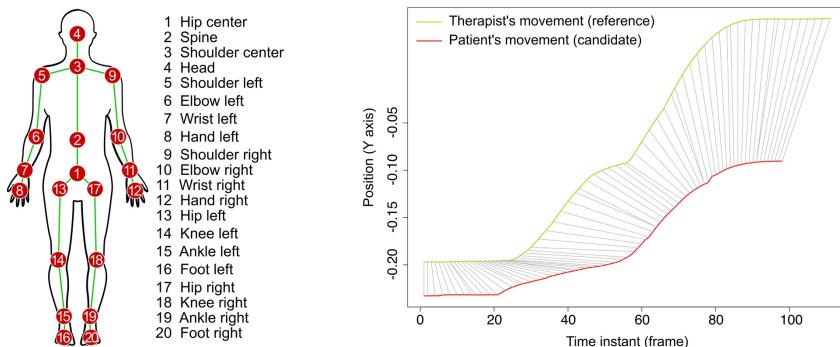


Figura 5.19: A la izquierda, el conjunto de articulaciones detectadas por la cámara de profundidad del dispositivo de tracking. A la derecha, representación visual de la comparación de una articulación a lo largo del eje Y.

problema de comparación de series temporales. El algoritmo DTW permite comparar dos series temporales midiendo la similitud entre dos secuencias temporales, que pueden variar en velocidad, es decir, independientemente de la diferencia de tiempo entre las dos secuencias.

La parte derecha de la figura 5.19 muestra dos curvas generadas a lo largo del eje Y para la articulación del codo derecho (punto 10 del esqueleto de la figura 5.19) durante un ejercicio en el que se levanta el brazo derecho. La curva discontinua corresponde al movimiento realizado por el terapeuta (ejercicio de referencia), mientras que la curva continua corresponde al del paciente (ejercicio a comparar). La alineación de ambas curvas obtenida tras la aplicación del algoritmo DTW (distancia = 5,2) está representada por los segmentos que unen las curvas. Este valor indica la distancia entre las dos curvas, por lo que cuanto más se acerque este valor a 0, mayor será la similitud entre los dos ejercicios, y menos diferencias significativas existirán entre las dos curvas. Este valor de distancia es calculado por el algoritmo utilizando como métrica la distancia euclídea entre las curvas.

La comparación, a nivel de ejercicio, se realiza comparando las curvas en los ejes X, Y y Z de los movimientos realizados por el paciente r y el movimiento del terapeuta m en cada uno de los puntos del esquema articular (puntos 1 a 20 de la figura 5.19) implicados en el movimiento, aplicando el algoritmo DTW a cada articulación independientemente. Para cada serie, es decir, para cada movimiento de una articulación i en el ejercicio se obtiene una distancia que considera las distancias en los ejes X, Y y Z de esa articulación; esta distancia se anota como d_{DTW_i} .

En el sistema planteado, la distancia total entre el ejercicio realizado por el paciente y el realizado por el terapeuta se calcula como la media aritmética de las distancias asociadas a las articulaciones que se utilizan en un ejercicio determinado.

Por otro lado, en el sistema propuesto, el paciente debe seleccionar el ejercicio a realizar, mediante comandos de voz o la interfaz de usuario. La interfaz de voz o táctil puede no ser adecuada para los pacientes con una discapacidad física o cognitiva más grave. Así, se requiere un mecanismo que elimine la necesidad de que el paciente seleccione y realice el ejercicio de manera explícita.

El algoritmo DTW estándar opera sobre series temporales finitas, es decir, sobre un ejercicio completado. El Open-End DTW (OE-DTW) de Tormene et al. [TGQS09], una variante de DTW, permite la **comparación sobre series temporales incompletas**. Proporciona el porcentaje de coincidencia entre dos curvas en cada instante de tiempo de la serie. Esta característica puede utilizarse para permitir que la comparación comience tan pronto como se detecte el movimiento y continúe, de modo que no sea necesario seleccionar manualmente un ejercicio. Además, el algoritmo OE-DTW puede utilizarse para identificar el ejercicio realizado por el paciente comparando la serie temporal generada con toda la referencia predefinida. A medida que el ejercicio progresá, se recogen más datos para establecer una comparación más informada.

En un funcionamiento normal, el paciente iniciaría el movimiento del ejercicio que desea realizar y el sistema detectaría el ejercicio que se está realizando. Para ello, el sistema compara periódicamente las posiciones de las articulaciones con las almacenadas para los ejercicios de referencia existentes. Cuando se encuentra un candidato óptimo, el ejercicio correspondiente a ese candidato se marca como definitivo y al completar el ejercicio se informa al paciente de su realización. Se considera que un candidato óptimo es el ejercicio de referencia que minimiza la distancia entre la trayectoria articular del paciente y la referencia, tal y como indica el algoritmo OE-DTW.

5.2.4. Caso práctico: MediaPipe Pose con Python

En caso práctico basado en IA para la rehabilitación remota existe una dependencia significativa en términos de hardware: el uso de Microsoft Azure Kinect DK como dispositivo de tracking que, a su vez, implica usar un equipo con una tarjeta gráfica dedicada con unas características especiales. Afortunadamente, existen **otras soluciones disponibles que permiten construir prototipos software con la misma base funcional sin sacrificar en exceso el rendimiento y la precisión**. Una de estas opciones es **MediaPipe**⁴.

MediaPipe se define como un sistema multi-plataforma que ofrece soluciones basadas en aprendizaje automático para procesar datos multimedia procedentes de fuentes de *streaming*. MediaPipe mantiene una licencia Apache 2.0, la cual facilita la utilización de esta herramienta tanto en proyectos open-source como en proyectos comerciales.

⁴<https://mediapipe.dev/>



Figura 5.20: 21 puntos clave de la mano capturados por MediaPipe. Fuente: <https://google.github.io/mediapipe/solutions/hands.html>

Una de las características más reseñables de MediaPipe es la **utilización de modelos de aprendizaje automático** para resolver, con solvencia, problemas como la detección de caras, la detección y el tracking de manos, la detección de poses, el tracking completo del esqueleto, la detección y el etiqueta de objetos, o incluso la generación de regiones de interés en el espacio 2D. Y todo ello tomando como entrada un flujo de vídeo compuesto por fotogramas con información en el espacio 2D. Esto es interesante porque MediaPipe es incluso capaz de inferir información en el espacio 3D.

Para instalar MediaPipe en entornos GNU/Linux solo es necesario ejecutar las instrucciones del siguiente listado. La principal dependencia software de MediaPipe es OpenCV, una de las bibliotecas de visión por computador más populares.

```
$ sudo apt install python3.8-venv
$ python3 -m venv mp_env && source mp_env/bin/activate

(mp_env)$ pip install mediapipe
(mp_env)$ python3 mediapipe_sample_hands.py
```

Por otro lado, el listado 5.1 muestra un ejemplo de uso del módulo de tracking de manos con MediaPipe, empleando para ello la API de Python. Más allá del código necesario para importar los paquetes que permiten su ejecución (líneas ①-5) y de la lógica para configurar, abrir y procesar un flujo de vídeo (líneas ⑧-24), la parte destacable de este fragmento de código es la que permite obtener, directamente a través de la API proporcionada, y renderizar la información del proceso de tracking realizado por MediaPipe, como se puede apreciar en las líneas ⑨ 29-36.

La estructura de datos importante es *results.hand_landmarks*, mostrada en la línea ⑨ y procesada en la cabecera del bucle de la línea ⑩. De acuerdo a la documentación de MediaPipe, se trata de una colección de las manos detectadas, en la que cada mano se representa como una lista de 21 puntos de referencia (ver figura 5.21) y cada punto de referencia se compone de las coordenadas *x*, *y*, *z* de la misma. *x* e *y* se normalizan con respecto al intervalo [0,0, 1,0] considerando la

anchura y la altura de la imagen, respectivamente. z representa la profundidad del punto de referencia, siendo la profundidad en la muñeca el origen, y cuanto menor sea el valor más cerca estará el punto de referencia de la cámara. La magnitud de z utiliza aproximadamente la misma escala que la de x.

Listado 5.1: ¡Hola, Mundo! en MediaPipe

```
1 import cv2
2 import mediapipe as mp
3 mp_drawing = mp.solutions.drawing_utils
4 mp_drawing_styles = mp.solutions.drawing_styles
5 mp_hands = mp.solutions.hands
6
7
8 cap = cv2.VideoCapture(0)
9 with mp_hands.Hands(
10     model_complexity=0,
11     min_detection_confidence=0.5,
12     min_tracking_confidence=0.5) as hands:
13     while cap.isOpened():
14         success, image = cap.read()
15         if not success:
16             print("Ignoring empty camera frame.")
17             # If loading a video, use 'break' instead of 'continue'.
18             continue
19
20         # To improve performance, optionally mark the image as not writeable to
21         # pass by reference.
22         image.flags.writeable = False
23         image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
24         results = hands.process(image)
25
26         # Draw the hand annotations on the image.
27         image.flags.writeable = True
28         image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)
29         if results.multi_hand_landmarks:
30             for hand_landmarks in results.multi_hand_landmarks:
31                 mp_drawing.draw_landmarks(
32                     image,
33                     hand_landmarks,
34                     mp_hands.HAND_CONNECTIONS,
35                     mp_drawing_styles.get_default_hand_landmarks_style(),
36                     mp_drawing_styles.get_default_hand_connections_style())
37
38         # Flip the image horizontally for a selfie-view display.
39         cv2.imshow('MediaPipe Hands', cv2.flip(image, 1))
40         if cv2.waitKey(5) & 0xFF == 27:
41             break
42 cap.release()
```

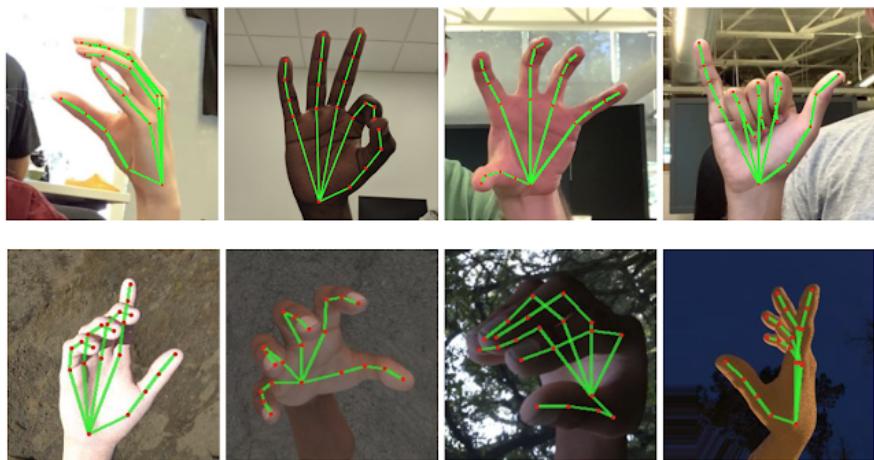


Figura 5.21: Arriba: información de entrada pasada a la red de tracking, incluyendo anotaciones. Abajo: imágenes de manos sintéticas renderizadas, incluyendo anotaciones. Fuente: <https://google.github.io/mediapipe/solutions/hands.html>

Integración de MediaPipe con PyGame

Python nos ofrece todo un catálogo de módulos para representar contenido visual. Uno de ellos es el popular PyGame⁵, un conjunto de módulos de Python diseñados para programar juegos. Pygame se apoya sobre la biblioteca SDL (Simple DirectMedia Layer)⁶, añadiendo funcionalidad sobre la misma, lo cual simplifica la creación de juegos y programas multimedia en el lenguaje Python.

Particularmente, se considera un ejemplo sencillo de juego en el que **el usuario debe interactuar con sus muñecas** para colisionar con las diferentes esferas que van cayendo desde la parte superior. La figura 5.22 muestra una captura de pantalla en la que se aprecia como la aplicación capture y renderiza la posición de las muñecas izquierda y derecha en el espacio 2D, mientras cae una esfera azul desde arriba.

La aplicación incrementará la puntuación y reproducirá un efecto de sonido cuando se produzca una colisión, mientras que decrementará el valor del número de fallos que son permisibles. Un fallo es una situación en la que una esfera virtual no ha sufrido colisión y desaparece por la parte inferior de la pantalla.

Tanto para las propias esferas como para los puntos de interés vinculados a las muñecas o *landmarks*, se definen sendas clases específicas en Python. Esta decisión se ha tomado para incrementar la escalabilidad del diseño y delegar la funcionalidad de actualización de estado y renderizado, en su caso. De hecho, tanto la esfera como los *landmarks* heredan de la clase *Sprite* de PyGame.

⁵<https://www.pygame.org/>

⁶<https://www.libsdl.org/>



Figura 5.22: Captura de pantalla del ejemplo de integración de MediaPipe y Pygame.

El listado 5.2 muestra una posible implementación de la clase *Sphere*. Note la función update() de la línea **②6**, que actualiza la posición en el eje Y de la bola considerando una velocidad establecida a nivel de configuración general.

Listado 5.2: Clase Sphere

```

1  class Sphere(Sprite):
2      """A class to represent a single sphere."""
3
4      def __init__(self, screen, filename):
5          """Initialize the sphere and set its starting position."""
6          super(Sphere, self).__init__()
7          self.screen = screen
8
9          # Load the sphere image, scale it and set its rect attribute.
10         self.image = pygame.image.load(filename)
11
12         xsize, ysize = self.image.get_size()
13         self.image = pygame.transform.scale(
14             self.image, (xsize * 0.3, ysize * 0.3))
15         self.rect = self.image.get_rect()
16
17         # Start each new sphere.
18         bound_xa, bound_xb = (self.image.get_width() // 2,
19                               self.screen.get_width() - (self.image.get_width() // 2))
20         self.rect.centerx = random.randint(bound_xa, bound_xb)
21         bound_ya, bound_yb = (-(self.image.get_height() // 2), 0)
22         self.rect.centery = random.randint(bound_ya, bound_yb)
23
24     def update(self, speed):
25         """Move the sphere down."""
26         self.rect.centery += speed

```

En términos de integración efectiva, el listado 5.3 refleja los diferentes pasos en el proceso de inicialización de los recursos empleados, tanto por PyGame como por MediaPipe. Destaca la inicialización de los denominados *landmarks*, en las líneas ⑩ 19-20, vinculados a las muñecas del usuario. Estos se gestionan en un contenedor *Group* del módulo sprite de PyGame para simplificar la gestión de colisiones múltiples entre objetos (ver línea ⑩ 21). El mismo enfoque se utiliza para manejar las colisiones de las esferas que se irán creando (ver línea ⑩ 24), de manera aleatoria, cuando la aplicación arranque.

Listado 5.3: Código de inicialización de PyGame y creación de esferas/landmarks

```
1 # Initialise screen.
2 pygame.init()
3 pygame.display.set_caption(title)
4 display = pygame.display.set_mode(screen_size)
5
6 # Initialise audios.
7 sfx_coin = pygame.mixer.Sound(SFX_COIN_FILENAME)
8 bg_music = pygame.mixer.Sound(MUSIC_FRANTIC_FILENAME)
9 bg_music.play(-1, 0, 3000)
10 bg_music.set_volume(0.2)
11
12 # Initialise fonts.
13 pygame.font.init()
14 SCORE_FONT = pygame.font.Font(BANGERS_FONT_FILENAME, 30)
15 MISTAKES_FONT = pygame.font.Font(BANGERS_FONT_FILENAME, 30)
16 GAME_OVER_FONT = pygame.font.Font(BANGERS_FONT_FILENAME, 50)
17
18 # Initialise landmarks.
19 wrist_left = Landmark(display, LANDMARK_FILENAME)
20 wrist_right = Landmark(display, LANDMARK_FILENAME)
21 landmarks = Group([wrist_left, wrist_right])
22
23 # Initialise spheres.
24 spheres = Group()
25
26 # Initialise clock
27 clock = pygame.time.Clock()
28
29 width, height = screen_size
30
31 score, mistakes = 0, 10
```

Por otro lado, el listado 5.4 pretende ilustrar los aspectos más representativos de la lógica general del mini-juego discutido. No se persigue, en este punto, ofrecer una discusión detallada de este fragmento de código, si no más bien de ofrecer un ejemplo de referencia e ilustrar sobre las facilidades que ofrece MediaPipe en combinación con PyGame.

Listado 5.4: Lógica general del juego

```
1 # Update landmark's position
2 wrist_left.rect.centerx = float(
3     results.pose_landmarks.landmark[mp_pose.PoseLandmark.LEFT_WRIST].x) * width
4 wrist_left.rect.centery = float(
5     results.pose_landmarks.landmark[mp_pose.PoseLandmark.LEFT_WRIST].y) * height
6 wrist_right.rect.centerx = float(
7     results.pose_landmarks.landmark[mp_pose.PoseLandmark.RIGHT_WRIST].x) * width
8 wrist_right.rect.centery = float(
9     results.pose_landmarks.landmark[mp_pose.PoseLandmark.RIGHT_WRIST].y) * height
10
11 pygame.surfarray.blit_array(display, frame.swapaxes(0, 1))
12
13 # Game finished when number of mistakes is reached
14 if mistakes <= 0:
15     game_over_text = GAME_OVER_FONT.render(
16         "Game Over", True, (0, 0, 0))
17     display.blit(game_over_text, game_over_text.get_rect(
18         center=(width // 2, height // 2)))
19     pygame.display.flip()
20
21 bg_music.fadeout(3000)
22
23 # No further processing is required.
24 continue
25
26 speed = configuration['game_dynamic']['speed']
27 spheres.update(speed)
28
29 # Generate spheres randomly.
30 if random.randrange(100) < 2:
31     spheres.add(Sphere(display, SPHERE_FILENAME))
32
33 for sphere in spheres.sprites():
34     # Kill spheres when they leave the screen.
35     if sphere.rect.top > display.get_height():
36         sphere.kill()
37
38     # Also, update mistakes.
39     mistakes -= 1
40
41 # This will find collisions between landmarks and spheres,
42 # killing the sprite of last group which collided.
43 hit_list = pygame.sprite.groupcollide(landmarks, spheres, False, True)
44 # Check the list of colliding sprites, and add one to the score for each one.
45 for _ in hit_list:
46     score += 1
47
48     # Play SFX coin when score is increased.
49     sfx_coin.play()
50
51 # Text rendering here...
52
53 spheres.draw(display)
54 landmarks.draw(display)
55
56 pygame.display.flip()
```

Los aspectos más destacables del listado 5.4 son los siguientes:

- Los landmarks definidos para este ejemplo actualizan su estado visual en función de los resultados de tracking de MediaPipe Pose. Esto se refleja en las líneas Θ 2-9.
- La condición de fin de juego está controlada en las líneas Θ 14-24, y consiste en comprobar si el valor de la variables *mistakes*, que representa el número de fallos que puede cometer el usuario, es igual a cero.
- La generación y la destrucción de esferas virtuales se maneja mediante la variable *spheres*, contenedor del tipo Group (previamente introducido como un elemento de PyGame que simplifica la gestión de colisiones). Así, en las líneas Θ 30-31 se crean, aleatoriamente, las esferas virtuales, mientras que las líneas Θ 33-39 son responsables de su destrucción en caso de que lleguen al borde inferior de la pantalla de juego (su posición en Y superar el alto de dicha pantalla).
- La detección de colisiones entre alguna de las muñecas y una o varias esferas virtuales se delega en la función *groupcollide()* del módulo *sprite* de PyGame. A través de una simple llamada en la línea Θ 43, es posible obtener una lista de potenciales colisiones y eliminar las colisiones asociadas al último grupo de sprites⁷. El último argumento de esta llamada, establecido a True, implica que el grupo de *sprites* asociado a *spheres* será eliminado.

5.3. Modelos de automatización

Los modelos de automatización existentes para los nuevos requerimientos industriales y de negocio se basan en la idea de construir una *fuerza de trabajo digital*, basada en software, que establezca sinergias entre los operadores humanos para incrementar el rendimiento del sistema, al mismo tiempo que se reducen costes y se aumenta el beneficio final generado.

En un extremo de este amplio abanico, es posible visualizar soluciones sencillas que automatizan tareas eminentemente repetitivas, que aportan poco valor y que son propensas a errores por parte de los operadores humanos. En el otro extremo, los **modelos de automatización se combinan con avanzadas técnicas de IA** para aumentar las capacidades de los operadores humanos hasta límites que hace unos años eran difícil de imaginar. Piense, por ejemplo, en la capacidad de un sistema para extraer patrones de comportamiento, en segundos, respecto a los usuarios de un determinado servicio, considerando millones de registros en una base de datos.

El uso de IA, combinado con la automatización de modelos de negocio, permite construir sistemas donde la interacción humana se minimiza para incrementar tanto la eficiencia como la efectividad, relegando a los operadores humanos a tareas donde aportan valor añadido con respecto a las máquinas.

⁷<https://www.pygame.org/docs/ref/sprite.html#pygame.sprite.groupcollide>

Así, esta sección introduce la **noción de automatización inteligente**, la cual se caracteriza a continuación. Posteriormente, se aborda el **estudio de un caso práctico**, desplegado en producción en un ámbito real, que muestra la potencia de la automatización inteligente en el contexto del reconocimiento automático de matrículas en entornos de tráfico urbano.

5.3.1. El concepto de *automatización inteligente*

La automatización inteligente (del término en inglés *intelligent automation*) se relaciona con una nueva **generación de software basado inherentemente en la automatización** [BBW21]. En esencia, combina métodos, técnicas y tecnologías para ejecutar procesos de negocio de manera automática en lugar de delegarlos en trabajadores humanos. Este nivel de automatización se alcanza imitando o simulando las capacidades de dichos trabajadores.

El objetivo final de la automatización inteligente es la de generar el mismo resultado mediante máquinas o programas software, desde el punto de vista del negocio, que se obtendría cuando un determinado proceso está siendo ejecutado por operadores humanos. Los principales beneficios de este modelo se resumen en el incremento de la velocidad de ejecución de los procesos, la reducción de costes, la mejora de la calidad de los procesos, el incremento de la robustez frente a posibles errores, y la optimización de los resultados de las decisiones vinculadas al negocio [BBW21]. A nivel global, la empresa se vería beneficiada de un mayor nivel de satisfacción por parte de sus clientes y de un incremento en los beneficios.

Particularmente, el concepto de automatización inteligente pone el foco en procesos desarrollados por operadores humanos en los que el **principal activo es el conocimiento**. Este activo está vinculado con industrias donde todo gira en torno a la información, como es el caso de la industria del sector servicios.

Piense, por ejemplo, en un servicio de *call center* en el que un grupo significativo de personas ofrecen un servicio de soporte vinculado a un determinado producto. La aplicación de un modelo de automatización inteligente implicaría modelar el proceso de negocio para generar una versión automatizable que redujera, o eliminaría totalmente, la dependencia con respecto a los operadores humanos. Los chatbots representan una tecnología que se puede emplear para automatizar dicho proceso.

Es posible identificar un conjunto de características clave en el contexto de automatización inteligente [BBW21]:

- Se trata de una tendencia **reciente**, acuñada por IEEE en 2017, que aúna la aplicación de una serie de tecnologías desarrolladas en la última década.
- Es un elemento **transversal**, ya que gran parte de la funcionalidad desarrollada se puede aplicar, horizontalmente, a un número significativo de industrias y negocios.
- Es **escalable**, en el sentido de que es posible instanciar más agentes virtuales cuando la carga de trabajo se ve incrementada, reutilizando el mismo código que fue desarrollado originalmente.

- Tiene una **alta disponibilidad**, ya que es posible ofrecer servicios bajo demanda, a cualquier hora del día, garantizando la continuidad y la sostenibilidad de los procesos de negocio automatizados.
- Es **fiable**, de forma que la salida esperada por un proceso de negocio automatizado, suponiendo una misma entrada de datos, se mantiene constante. Sin cambios en el entorno operativo, es posible asumir que el nivel de desarrollo del sistema no se verá alterado.
- Es un modelo **rentable**, ya que gran parte de las tecnologías vinculadas a la automatización inteligente están disponibles bajo un coste razonable, lo cual facilita los retornos de la inversión en no mucho tiempo.
- Es **accesible**, siempre y cuando la capa de presentación vinculada a las interfaces gráficas de usuario empleadas estén bien diseñadas. En este sentido, se persigue la facilidad de uso para que no sean necesarias habilidades concretas a la hora de manejar las tecnologías involucradas.

5.3.2. Caso práctico: reconocimiento automático de matrículas

Esta sección presenta un caso de estudio aplicado sobre un problema real en el municipio de Ciudad Real, donde se llevó a cabo un estudio del flujo de tráfico tomando como referencia un conjunto de puntos estratégicos distribuidos por el entorno urbano. La figura 5.23 muestra, de manera visual, el mapa asociado a uno de los experimentos llevados a cabo. Este caso de estudio fue abordado por un conjunto de investigadores de la Escuela Técnica Superior de Ingeniería de Caminos, Canales y Puertos y de la Escuela Superior de Informática, ambas de la Universidad de Castilla-La Mancha.

Los estudios preliminares sobre el flujo del tráfico se realizaron mediante un enfoque manual, de forma que primero se desplegaban una serie de cámaras que se usaban para grabar secuencias de vídeo en diversos puntos de la ciudad. Posteriormente, se efectuaba una anotación manual de las matrículas asociadas a los coches que se detectaban en las secuencias de vídeo. Dicha anotación era manual en el sentido de que existía un conjunto de personas que visualizaban los vídeos, de principio a fin, para escribir en una hoja de cálculo qué coche pasaba en qué instante de tiempo.

Como el lector puede imaginar, esta solución no es escalable, ya que es necesaria una reproducción integral de todos los fragmentos de vídeo y una generación manual de información para cada uno de ellos. Además, es propensa a errores, ya que son los operadores humanos los que deben realizarla.

A partir de esta problemática, se estableció un **modelo de automatización que permitiera la replicabilidad de experimentos de análisis de tráfico**, manteniendo como variables independientes el número y la localización de los nodos de análisis de tráfico. Este modelo de automatización gira en torno a la definición de una arquitectura para el despliegue de sistemas de bajo coste para el reconocimiento

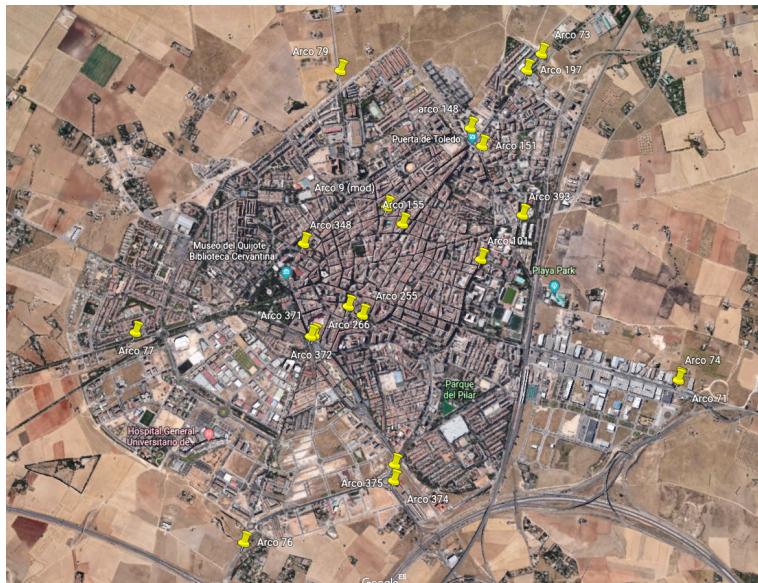


Figura 5.23: Imagen de satélite con los diferentes puntos estratégicos, resaltados en amarillo, utilizados para analizar el tráfico urbano de Ciudad Real.

automático de matrículas. Los sistemas desplegados se basan, a su vez, en la creación de un prototipo hardware/software de sensor de reconocimiento de matrículas, diseñado para integrarse en redes escalables de sensores guiadas por un modelo de localización capaz de establecer la mejor distribución de los mismos en un escenario de tráfico urbano.

Necesidades de automatización

Como se ha introducido previamente, los estudios preliminares se basaban en reclutar a un conjunto de personas que grabaran el tráfico en diversas localizaciones físicas de un entorno urbano. Las localizaciones se establecían a priori, en base a un modelo de análisis de tráfico urbano ya existente. Cada localización tenía asignada una persona, que se desplazaba antes del comienzo del experimento para grabar, cámara y trípode en mano, los vehículos que pasaban por dicha localización. La figura 5.24 muestra dos capturas en las que se aprecia este enfoque, de manera visual.

Tras la grabación distribuida del tráfico en los diferentes puntos físicos del municipio, existía un proceso de post-procesamiento que consistía en reproducir todos y cada uno de los vídeos grabados para anotar la matrícula de los vehículos detectados. Se trataba de un proceso puramente manual, en el que se utilizaba una hoja de cálculo para escribir los números de matrículas asociados a cada nodo durante el experimento.



Figura 5.24: Imágenes que representan el proceso de captura manual de vídeos para su posterior post-procesamiento y generación manual de matrículas de coches.

Como el lector puede imaginar, este enfoque es manejable cuando el número de nodos de la red de tráfico es bajo. Desafortunadamente, cuando el número de nodos comienza a aumentar, se hace necesario introducir un modelo que automatice tanto la grabación de los vídeos como la detección automática de las matrículas.

El proceso de reconocimiento automático de matrículas

El reconocimiento automático de matrículas, conocido por su término en inglés como ALPR (Automatic License Plate Recognition) y también como ANPR (Automatic Number Plate Recognition) tiene su fundamento en la **aplicación de técnicas de procesamiento de imágenes para identificar los vehículos por sus matrículas**, principalmente en tiempo real (para el control automático de las normas de tráfico). El proceso general de reconocimiento automático de matrículas puede resumirse en varias etapas bien definidas. Cada paso implica un **conjunto diferente de algoritmos** y consideraciones, los cuales se exponen a continuación:

- **Captura de la imagen del vehículo.** Este paso tiene un impacto crítico en los pasos posteriores, ya que el resultado final depende en gran medida de la calidad de las imágenes capturadas. La tarea de capturar correctamente imágenes de vehículos en movimiento en tiempo real es compleja y está sujeta a muchas variables del entorno, como la iluminación, la velocidad del vehículo y el ángulo de captura.
- **Detección de matrículas.** Este paso se centra en la detección de la zona en la que se espera que se encuentre una matrícula. Las imágenes se almacenan de forma digital utilizando matrices, en las que cada elemento representa la intensidad luminosa de un píxel individual. Diferentes técnicas y algoritmos ofrecen diferentes *definiciones* de una matrícula. Por ejemplo, en el caso de los algoritmos de detección de bordes, la definición de matrícula podría

entenderse como una *zona rectangular con una mayor densidad de bordes verticales y horizontales*. Esta elevada presencia de bordes suele estar causada por las placas de los bordes, así como por los límites entre los caracteres y el fondo de la placa.

- **Segmentación de los caracteres.** Una vez detectada la región de la placa, es necesario dividirla en trozos, cada uno de los cuales contiene un carácter diferente. Este es, junto con la fase de detección de la placa, uno de los pasos más importantes de un sistema ALPR, ya que todas las fases posteriores dependen de él. Otra similitud con el proceso de detección de matrículas es que existe una amplia gama de técnicas que van desde el análisis de la proyección horizontal de una matrícula, hasta enfoques más sofisticados, como el uso de redes neuronales.
- **Reconocimiento de caracteres.** El último paso del proceso ALPR consiste en reconocer cada uno de los caracteres que han sido previamente segmentados. En otras palabras, el objetivo de este paso consiste en identificar y convertir el texto de la imagen en texto editable. Existen varias técnicas para afrontar este reto, como redes neuronales artificiales, la coincidencia de plantillas o el reconocimiento óptico de caracteres. Dado que el reconocimiento de caracteres tiene lugar después de la segmentación, el sistema de reconocimiento debe hacer frente a los caracteres ambiguos, ruidosos o distorsionados obtenidos en la etapa anterior.

Un paso adicional que complementa los pasos anteriores es la **recuperación de errores** que puede producirse al reconocer los números de matrícula. Este problema también ha de considerarse cuando los datos de escaneo de matrículas se utilizan para estimar el flujo de tráfico.

Por otro lado, y al margen de la problemática inherente de diseñar y desplegar un sistema ALPR, el creciente desarrollo de estos sistemas se enfrenta a algunos retos adicionales, como por ejemplo el alto coste de los sensores hardware existentes para abordar el proceso previamente discutido (unos 20.000 dólares por cámara, a lo que hay que sumar el mantenimiento). En este sentido, los nuevos modelos de automatización, en este contexto, deben considerar el despliegue de este tipo de servicios a mayor escala para hacer frente a problemas de transporte como los planes de movilidad urbana.

Por último, es muy importante tener en cuenta los **derechos de privacidad de los usuarios** en el proceso en el que se recogen los datos de los vehículos, en función de las diferentes ubicaciones a lo largo de la red. Todo esto significa que, a la hora de diseñar un tipo de sensor que pueda ser implementado en una arquitectura que sirva para monitorizar la red de tráfico, los criterios de coste respecto a la fabricación, la instalación, la operatividad y la resiliencia, y el procesamiento de la información deben tenerse en cuenta.



Figura 5.25: Coche de policía que integra un sistema para reconocer matrículas de manera automática y en movimiento.

Arquitectura para el despliegue de sistemas de bajo coste para el reconocimiento automático de matrículas

La figura 5.26 muestra la arquitectura multicapa diseñada para desplegar redes de sensores de bajo coste para detección automática de matrículas. El uso de un enfoque multicapa garantiza la escalabilidad de la arquitectura, ya que es posible realizar modificaciones en cada una de las capas sin afectar al resto. En concreto, la arquitectura se compone de tres capas:

- **La capa perceptiva**, que integra los sensores autónomos responsables de la captura de imágenes. Cada uno de estos sensores integra un dispositivo de procesamiento de bajo consumo y un conjunto de dispositivos de bajo coste que realizan la captura de la imagen. La cámara utilizada permite diferentes configuraciones en función de las características del entorno urbano en el que se realiza el análisis del tráfico.
- **La capa de gestión inteligente**, que proporciona la funcionalidad necesaria para la definición y ejecución de los experimentos de análisis de tráfico. Esta capa integra los módulos funcionales responsables de la configuración de los experimentos, la detección automática de matrículas, a partir de las imágenes proporcionadas por los sensores de la capa perceptiva, y el almacenamiento permanente de información en la base de datos del sistema.
- **La capa de monitorización *online***, que permite la visualización, a través de un navegador web, de la evolución de un experimento a medida que se realiza. Gracias a esta capa, es posible consultar el estado de los diferentes sensores de la capa perceptiva, mediante interacciones a través de la capa de gestión inteligente.

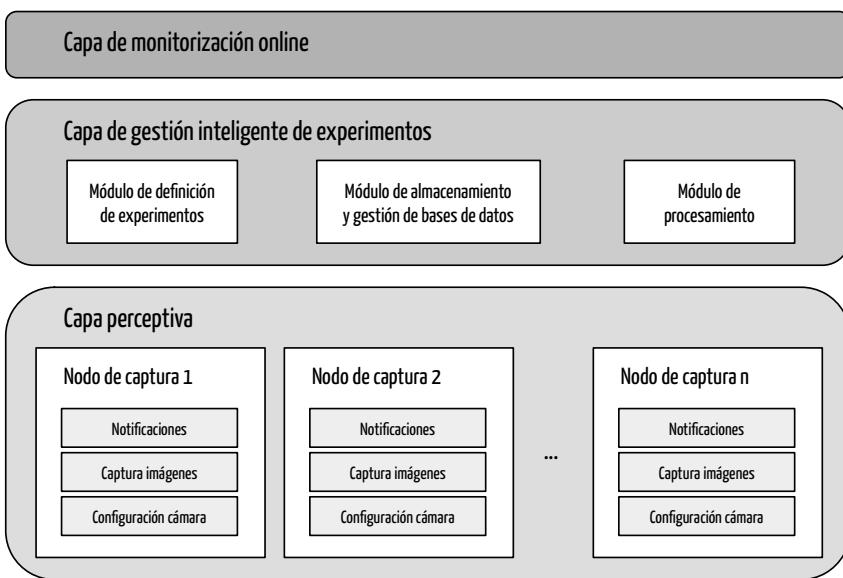


Figura 5.26: Arquitectura multicapa para el despliegue de redes de sensores de bajo coste para el reconocimiento automático de matrículas.

La **capa perceptiva** es la capa de más bajo nivel de la arquitectura propuesta. Contiene el conjunto de sensores básicos de procesamiento de bajo coste que se desplegarán en el entorno físico para realizar la captura de imágenes. En esta capa, estos sensores no conocen la existencia del resto de los sensores. Es decir, cada sensor es independiente de los demás y su responsabilidad se limita a tomar imágenes en un punto físico determinado, enviarlas a la capa superior y, periódicamente, notificar que están funcionando correctamente. Dado que el diseño del sensor representa un componente importante de la arquitectura arquitectura propuesta, más adelante se hace una descripción detallada de las principales características del prototipo hardware/software desarrollado.

La **capa de gestión inteligente** proporciona la funcionalidad necesaria para (i) facilitar el despliegue de redes de sensores de bajo coste y la ejecución de experimentos, (ii) procesar las imágenes captadas por los sensores de la capa perceptiva, realizando la detección automática de matrículas, y (iii) almacenar de forma persistente toda la información asociada a un experimento para su posterior análisis forense.

Esta capa de la arquitectura sigue un modelo PAAS (Platform-As-A-Service), es decir, utiliza la infraestructura desplegada en la nube que proporciona las necesidades computacionales del sistema de análisis de tráfico. Este enfoque permite ofrecer una solución escalable que responda a las demandas del sistema de detección automática de matrículas, evitando la complejidad que supondría desplegar servidores propios para dar soporte funcional.

En particular, se ha utilizado GAE (Google App Engine)⁸ como soporte funcional del servidor del sistema, utilizando el lenguaje Python para desarrollar los diferentes componentes del sistema y el framework de aplicaciones web Flask⁹ para gestionar las peticiones web. La recuperación de información respecto a los sensores de la capa perceptiva se materializa a través de peticiones web, de forma que éstas pueden solicitar la configuración inicial de un sensor, o enviar información, como los llamados paquetes de control, a medida que evoluciona el estado de un experimento de análisis de tráfico.

En este contexto, el **concepto de paquete de control** destaca como la unidad básica de información que debe manejar la capa de gestión inteligente. El paquete de control se compone de los siguientes campos:

- *ID del cliente*. ID único del sensor que envía el paquete dentro de la red de sensores.
- *Marca de tiempo*. Marca temporal asociada al momento en que el sensor capturó la imagen.
- *Latencia*. Retraso respecto al paquete de control enviado anteriormente. Su valor es 0.0 para el primer paquete de control.
- *Matrículas*. Lista de números de matrícula candidatos, junto con sus respectivos valores de confianza, detectados en la imagen capturada por el sensor.
- *Imagen*. Serialización binaria de la imagen capturada por el sensor.

Hay tres módulos diferentes contenidos en esta capa, los cuales se sintetizan a continuación:

- **Módulo de definición del experimento.** Este módulo se encarga de gestionar la información de alto nivel vinculada a un experimento de análisis de tráfico. Esta información incluye las horas de inicio/fin del experimento y la configuración de los parámetros que guían el funcionamiento de los sensores de la capa perceptiva. Esta configuración es recuperada por cada uno de estos sensores a través de una petición web cuando inician su actividad, de forma que es posible ajustarla sin modificar el estado de los sensores cada vez que sea necesario.
- **Módulo de procesamiento.** Este módulo proporciona la funcionalidad necesaria para realizar eficazmente la detección automática de matrículas. Así, la entrada de este módulo es un conjunto de imágenes, en las que potencialmente pueden aparecer vehículos, y la salida es el conjunto de matrículas detectadas, junto con el grado de confianza asociado a dichas detecciones. En la versión actual del sistema se utiliza la biblioteca comercial OpenALPR

⁸<https://cloud.google.com/appengine>

⁹<https://flask.palletsprojects.com/en/2.0.x/>

en versión web¹⁰. Este módulo se encarga de atender las peticiones de análisis de imágenes realizadas por los sensores de la capa perceptiva. Tanto las imágenes en sí como las detecciones de matrículas asociadas a ellas son reportadas al módulo de gestión y almacenamiento de la base de datos.

- **Módulo de gestión y almacenamiento de la base de datos.** Este módulo permite el almacenamiento permanente de toda la información asociada a un experimento de análisis de tráfico y detección automática de matrículas. A nivel funcional, este módulo ofrece un servicio de análisis forense de toda la información generada como resultado de la ejecución de un experimento.

Es importante destacar que el módulo de procesamiento ofrece dos modos de funcionamiento: i) *online* y ii) *offline*. En el modo online, el módulo de procesamiento realiza un análisis online de las imágenes obtenidas de la capa perceptiva, paralelizando las peticiones que recibe para proporcionar resultados en un tiempo adecuado. Por el contrario, el modo de funcionamiento offline está diseñado para analizar grandes conjuntos de imágenes asociados a la ejecución pasada de un experimento de análisis de tráfico.

Finalmente, el objetivo general de la **capa de monitorización online** es facilitar el seguimiento, en tiempo real, de la evolución de un experimento de análisis de tráfico. Para facilitar el uso del sistema y evitar la instalación de software por parte del usuario, la interacción a través de esta capa se realiza mediante un navegador web. Desde un punto de vista de alto nivel, la capa de monitorización online ofrece la siguiente funcionalidad:

- **Visión general del estado del sistema.** A través de una vista de cuadrícula, el usuario puede visualizar un subconjunto de sensores en tiempo real. Esta vista está diseñada para proporcionar una perspectiva visual de alto nivel de los sensores desplegados en un experimento de análisis de tráfico. Es posible configurar el número de componentes de la cuadrícula.
- **Análisis del estado de un sensor.** Esta vista permite conocer el estado, en tiempo real, de uno de los sensores previamente desplegados (ver figura 5.27). Además de visualizar la última imagen captada por el sensor, es posible obtener estadísticas globales de los datos obtenidos y de la información generada si se realiza un análisis online.

En ambos casos, la información representada en esta capa, a través de un navegador web, se obtiene realizando consultas a la capa del nivel inmediatamente inferior, es decir, la capa de gestión inteligente. Esta última, a su vez, obtendrá la información de la capa perceptiva, donde se encuentran los sensores desplegados en el escenario físico.

¹⁰<https://es.openalpr.com/>

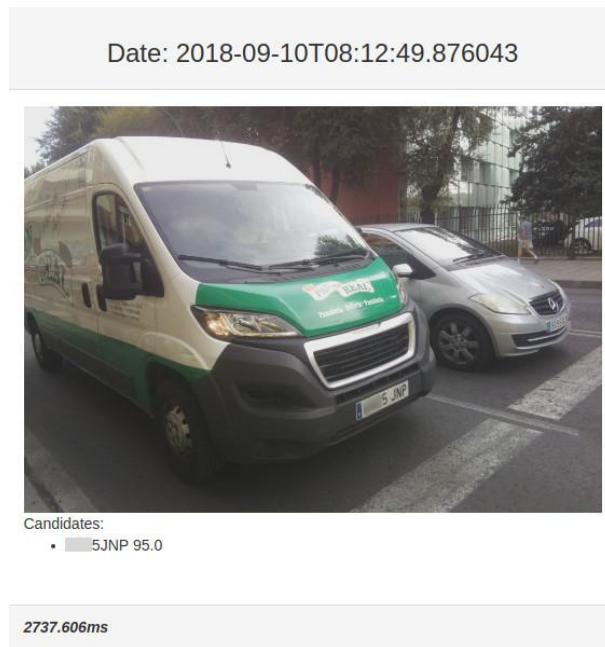


Figura 5.27: Visualización asociada a uno de los sensores de análisis de tráfico.

Prototipo hardware de bajo coste

El prototipo hardware diseñado plantear el modelo de automatización discutido en esta sección se muestra en la figura 5.28. Se puede afirmar que es un prototipo de bajo coste (alrededor de 60€) que está formado de los siguientes componentes:

- Raspberry Pi Zero W.
- Raspberry Pi Camera Module V2.1.
- Power bank PowerAdd Slim2 5000mAh.
- Caja de plástico fabricada con impresora 3D (PLA).
- MicroSD card U1 16GB.
- Memory stick 128MB.
- Trípode.

Raspberry Pi es un ordenador de placa única de bajo coste que ejecuta software de código abierto. Las múltiples versiones de la placa emplean un procesador Broadcom (arquitectura ARM) y un conector de cámara específico. Gracias al uso de este hardware, se pueden utilizar las versiones del **sistema operativo Raspberry Pi** (antes llamado Raspbian), derivado de la distribución GNU/Linux Debian. Así, es posible el desarrollo en varios lenguajes de programación de propósito general.

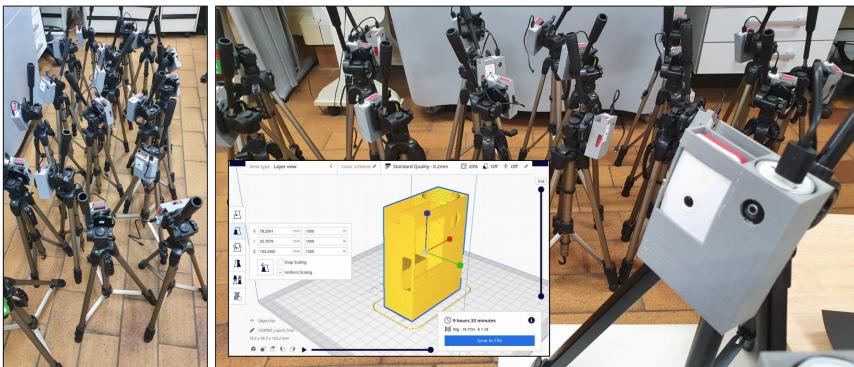


Figura 5.28: Múltiples nodos de procesamiento del prototipo del sensor diseñado.

Para el desarrollo del sensor anteriormente presentado, se ha utilizado la versión de la **placa denominada Pi Zero W**, que incorpora el microprocesador Broadcom BCM2835. Esta tiene un procesador de un solo núcleo que funciona a 1GHz, 512MB de RAM, una tarjeta gráfica VidoCore IV y una tarjeta MicroSD como dispositivo de almacenamiento. Basada en el modelo Pi Zero, esta versión ofrece conectividad Wi-Fi, que permite la monitorización en línea. En las pruebas realizadas, la conectividad con la nube se ha realizado mediante sensores de conexión 3G/4G, utilizando la red Wi-Fi institucional existente en la Universidad de Castilla-La Mancha siempre que ha sido posible.

La **Raspberry Pi Camera V2.1** de 8 megapíxeles cuenta con el sensor de imagen IMX219RQ de Sony, con una alta sensibilidad a las condiciones de iluminación exterior adversas, con reducción de ruido y manchas de patrón fijo. La conexión se realiza mediante el puerto de interfaz serie de la cámara directamente al bus CSI-2 a través de un cable plano de 15 pines. La cámara realiza, automáticamente, las calibraciones de nivel de negro, balance de blancos y filtro de banda, así como la detección automática de luminancia (para condiciones cambiantes) de 50 Hz en el hardware. En la configuración de cada sensor se puede especificar la resolución con la que se toma cada fotografía, hasta un máximo de 3280x2464 píxeles.

La versión actual del prototipo utiliza una **tarjeta microSD UHS Speed Class 1(U1)**, con una velocidad de escritura de 10MB/s necesaria para grabar fotografías de alta definición en intervalos cortos. Cada fotografía de 8 MP puede requerir unos 4 MB en formato JPG si se almacena a máxima resolución (dependiendo de la complejidad de la escena y de las condiciones de iluminación).

En las pruebas realizadas, cada sensor realizó las capturas con una resolución de 1024x720 píxeles. Cada imagen ocupaba una media de 412KB, tamaño que se redujo a 151KB tras el proceso de optimización con subregiones de captura. La frecuencia de captura se estableció en 1 imagen por segundo. Esto requiere un

almacenamiento en disco de 1,4GB por cada hora de captura sin optimización. Así, con más de 14GB disponibles en la tarjeta SD para datos, es **posible almacenar más de 8 horas de imágenes sin optimización**, y más de 24 horas definiendo subregiones de captura.

La unidad flash de 128 MB se utiliza para almacenar los **parámetros de configuración del sensor** de procesamiento, como el identificador único del sensor de procesamiento, la dirección del servidor web asociado a la capa de gestión del experimento inteligente y la configuración de la red. Para la integración de todos los componentes hardware del sistema, se ha realizado un prototipo básico mediante impresión 3D, con un tamaño de 103x78x35mm (59g de PLA de 1,75mm).

El uso de un procesador de propósito general, como el Broadcom BCM2835, facilita la creación rápida de prototipos, así como la integración de los módulos de software existentes. En particular, la integración de la funcionalidad ofrecida a la capa de gestión inteligente se realiza de forma sencilla gracias a este enfoque.

Por otra parte, el impacto de los costes de mantenimiento y la integración de nuevas funcionalidades se minimiza al utilizar un **enfoque basado en computación en la nube** en el que cada sensor se configura mediante parámetros específicos. Se especifica un identificador único y una dirección asociada al servidor en la nube. Desde el servidor, el sensor recibe un mensaje JSON con los parámetros a utilizar en cada experimento de análisis. Utilizando este paquete de configuración por sensor, es posible ajustar la configuración de captura específica de cada sensor en la red, en función de su posición, condiciones meteorológicas o nivel de iluminación en cada momento del día. Por ejemplo, un sensor que puede estar mejor posicionado para identificar matrículas podrá realizar capturas de menor resolución (ahormando costes de procesado) que un sensor que esté situado más lejos del tráfico. Incluso el mismo sensor puede necesitar realizar capturas de mayor resolución en situaciones meteorológicas adversas, como la lluvia o la niebla.

El **formato de los mensajes JSON** utilizado es el siguiente:

Listado 5.5: Formato del mensaje JSON usado por el prototipo

```
1  {
2      "begTime": "2020-06-10T09:00:00",
3      "endTime": "2020-06-10T11:00:00",
4      "resolution": "1024x720",
5      "mode": "manual",
6      "exposure_time": 1000,
7      "freq_capture": 1000,
8      "iso": 320,
9      "rectangle_p1": [
10          280,
11          262
12      ],
13      "rectangle_p2": [
14          1024,
15          574
16      ]
17 }
```

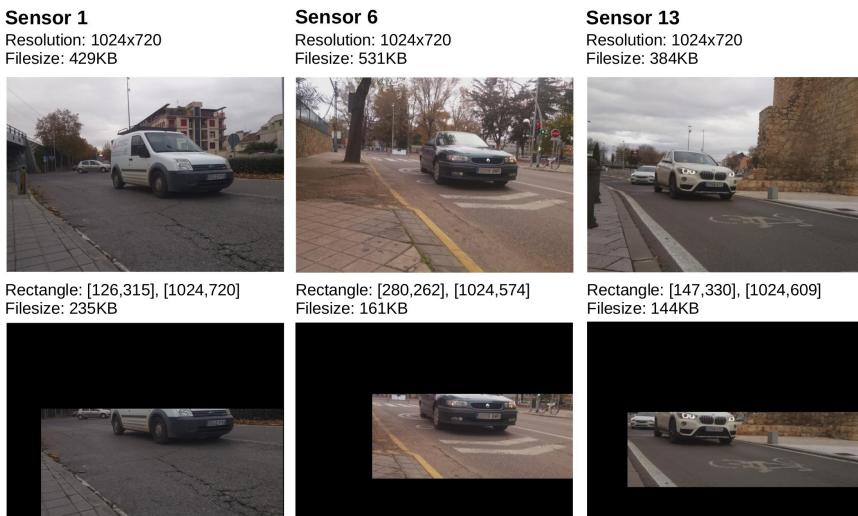


Figura 5.29: Definición de los parámetros de recortado en 3 nodos de procesamiento diferentes. Incluye un análisis comparativo del almacenamiento de cada frame.

Los campos *begTime* y *endTime* indican la fecha y hora de inicio y fin de la sesión de captura. *Resolution* indica la resolución de captura del sensor con valores soportados por el hardware hasta un máximo de 3280×2464 píxeles. Si el modo de campo (*mode*) está configurado como manual, es posible indicar la velocidad de obturación o el tiempo de exposición, que define la cantidad de luz que entra en el sensor de la cámara. El parámetro *exposure_time* define la fracción de segundo (en la forma 1/segundos de exposición) que se deja pasar la luz. El campo *freq_capture* indica el número de milisegundos que pasarán entre cada captura. El campo *iso* define la sensibilidad del sensor a la luz (valores bajos para capturas con buen nivel de luz). Finalmente, los campos que comienzan con la palabra clave *rectangle* permiten definir subregiones de captura dentro de una imagen. Las esquinas superior izquierda e inferior derecha definen el rectángulo de captura válido dentro de la imagen. El resto de los píxeles se eliminan de la imagen, lo que facilita la transmisión de la imagen a través de la red y evita los costes de almacenamiento y procesamiento en las regiones en las que nunca aparecerán los números de matrícula, tal y como se aprecia en la figura 5.29.

El uso de parámetros que se utilizan para definir subregiones en las **imágenes capturadas posibilita que su tamaño puede reducirse drásticamente**. Cualquier conexión 3G es más que suficiente para cubrir las necesidades de ancho de banda de cada sensor de procesamiento, sin pérdida de calidad de imagen. Incluso en condiciones de transmisión más adversas (como cobertura EDGE o GPRS con velocidades máximas entre 114 y 384 Kbps), el fotograma podría ser almacenado



Figura 5.30: Diferentes niveles de compresión JPG. Con valores inferiores a 60 (pérdida significativa de información de alta frecuencia), la calidad de la imagen compromete significativamente la tasa de éxito de los algoritmos de detección de matrículas.

utilizando un mayor nivel de compresión JPG sin pérdida significativa de calidad de imagen (hasta un nivel de 65 sería aceptable) y, por tanto, sin poner en riesgo la identificación de la matrícula. La figura 5.30 muestra diversos niveles de compresión aplicados, junto con el resultado obtenido.

Respecto al **procesamiento de información**, la arquitectura propuesta, y en particular la capa de gestión inteligente que se ha comentado anteriormente, mejora los costes de procesamiento, ofreciendo resultados que pueden ser en tiempo real o con ejecución programada offline. De esta forma, el uso de la plataforma en su conjunto puede ser incluso compartido entre diferentes conjuntos de sensores, evitando la complejidad innecesaria de la gestión local a nivel de cada sensor o grupo de sensores.

Desde el punto de vista del procesamiento de la información, es posible minimizar el tráfico de información entre el sistema de análisis de imágenes (en la nube) y los sensores de procesamiento. A modo de ejemplo, la figura 5.31 muestra cómo los sensores pueden realizar menos peticiones codificando varias capturas en una sola imagen.

La biblioteca OpenALPR

OpenALPR es una biblioteca software, escrita en C++ pero con bindings para C#, Java, Node.js y Python, que analiza imágenes y flujos de vídeo para identificar matrículas y vehículos. La **salida de datos** generada cuando se utiliza este software consiste en una **lista de características del vehículo** que incluye la representación textual de los caracteres de la matrícula detectados. Una de las principales ventajas existentes en la solución comercial es la capacidad de integrar la biblioteca con cualquier cámara IP. Los **principales casos de uso** que permite manejar esta herramienta son los siguientes:



Figura 5.31: Detección de vehículos en una matriz de imágenes de 3x3, que permite, mediante un único envío a la nube, procesar 9 segundos de análisis de tráfico de un sensor determinado. El sistema detecta tanto la matrícula como determinadas características del vehículo, asignando un valor de confianza a cada detección.

- Reconocer matrículas y vehículos a partir de secuencias de cámaras. Los resultados se pueden consultar, son indexables y es posible activar alertas. El repositorio de datos puede estar en la nube o almacenado completamente en la infraestructura del cliente.
- Reconocer matrículas y vehículos a partir de flujos de cámaras y enviar los resultados a la propia aplicación del cliente.
- Integrar el reconocimiento de matrículas y vehículos en la aplicación del cliente mediante código fuente (C/C++, C#, VB.NET, Java, Python, Node.js).

Actualmente, el reconocimiento automático de matrículas ofrecido por OpenALPR es capaz de identificar matrículas de más de 70 países y 500 regiones de todo el mundo. Por otro lado, las capacidades de reconocimiento de vehículos ofrecidas se basan en IA y técnicas de aprendizaje automático para identificar más de 2.000 características únicas de los vehículos. Además de reconocer la matrícula y la región, la biblioteca es capaz de inferir información como el tipo de vehículo, la marca, el modelo, el color, el año y el sentido de la marcha.

La **instalación de OpenALPR** es sencilla y está disponible para los sistemas operativos más populares¹¹, pero requiere una licencia (existen licencias de evaluación) para poder utilizarlo desde dispositivos propios. El listado 5.6 muestra el tipo de salida esperada cuando OpenALPR se utiliza desde un terminal. Como se puede apreciar, es posible obtener diferentes resultados, con valores de confianza asociados, cuando se analiza una imagen.

¹¹<https://github.com/openalpr/openalpr>

Listado 5.6: Ejemplo de uso de OpenALPR desde consola

```
1 user@linux:~/openalpr$ alpr ./samplecar.png
2
3 plate0: top 10 results -- Processing Time = 58.1879ms.
4     - PE3R2X    confidence: 88.9371
5     - PE32X     confidence: 78.1385
6     - PE3R2     confidence: 77.5444
7     - PE3R2Y    confidence: 76.1448
8     - P63R2X    confidence: 72.9016
9     - FE3R2X    confidence: 72.1147
10    - PE32      confidence: 66.7458
11    - PE32Y     confidence: 65.3462
12    - P632X     confidence: 62.1031
13    - P63R2     confidence: 61.5089
```

La integración con lenguajes como Python es simple, y se resume en utilizar la API proporcionada, como se recoge en el listado 5.7.

Listado 5.7: Ejemplo básico de uso de OpenALPR en Python

```
1 import json
2 from openalpr import Alpr
3
4 alpr = Alpr("us", "/etc/openalpr/openalpr.conf",
5             "/usr/share/openalpr/runtime_data")
6
7 if not alpr.is_loaded():
8     print("Error loading OpenALPR")
9     sys.exit(1)
10
11 results = alpr.recognize_file("/path/to/image.jpg")
12 print(json.dumps(results, indent=4))
13
14 alpr.unload()
```

Cabe destacar que la evolución de OpenALPR lo ha llevado a un modelo de negocio basado en SAAS (Software-As-A-Service), de forma que actualmente se suele utilizar desde la nube cuando se considera la solución comercial de esta biblioteca¹². Típicamente, se llevará a cabo una serie de peticiones al servicio en la nube, ya sea de manera online u offline, a la hora de reconocer matrículas.

En este sentido, el listado 5.8 muestra el prototipo de función que se podría utilizar para, delegando en el servicio en la nube de OpenALPR, obtener las matrículas asociadas a una imagen pasada como argumento. En esencia, y como se puede apreciar en las líneas ⑧-14, es necesario construir la petición web que se enviará al servicio. Posteriormente, en las líneas ⑯-19 se obtienen y devuelven los resultados obtenidos. Estos consisten en una lista de posibles matrículas detectadas, junto con sus respectivos niveles de confianza.

¹²<https://cloud.openalpr.com/cloudapi/>

Listado 5.8: Función en Python para solicitar el reconocimiento en la nube de la matrícula asociada a una imagen

```
1 def request_plate(self, image):
2     """ Requests a recognition to the indicated
3     service, and returns a tuple consisting of:
4     1) A list of strings with the recognized plate(s).
5     2) A list of floats with the corresponding confidence value(s).
6     3) The raw JSON response
7     """
8     payload, headers = multipart_encode([MultipartParam("image", image,
9                                         filename="image",
10                                        filetype="jpeg")])
11    r_petition = urlfetch.fetch(constants.CLOUD_SERVICE_URL,
12                                method = urlfetch.POST,
13                                payload = "".join(payload),
14                                headers = headers)
15    try:
16        json_response = json.loads(r_petition.content.decode())['results']
17        return ([result['plate'] for result in json_response],
18                [result['confidence'] for result in json_response],
19                json_response)
20    except KeyError:
21        return []
```

5.4. Gestión de recursos y activos

En el presente capítulo se han abordado, desde un punto de vista general, cuestiones relativas a estrategias y modelos de negocio donde el uso de la IA juega un papel fundamental. Esta última sección se centra en la gestión de recursos y activos, un aspecto que, por sí mismo, daría para un libro independiente. El enfoque elegido para introducirlo ha sido el de la **adopción de IA en la empresa**, desde una perspectiva transversal. Se discutirán aspectos como las necesidades humanas y culturales requeridas para afrontar los cambios que van a ocurrir en los próximos años y la proliferación de una inteligencia colaborativa entre humanos e IA.

El caso práctico que se incluye al final de esta sección se ha planteado considerando que **los datos representan uno de los activos más importantes de una empresa** que pretende utilizar la IA como piedra angular de su negocio. Este caso práctico introduce GAE (Google App Engine), una solución PAAS ofrecida por Google e integrada a la perfección en su ecosistema tecnológico. GAE proporciona diferentes módulos funcionales para gestionar datos de naturaleza heterogénea, al mismo tiempo que abstractea al desarrollador de la infraestructura necesaria para escalar aplicaciones desplegadas en la nube.

5.4.1. Adoptando IA en la empresa

La integración de IA en una empresa tiene un impacto directo en los recursos de la misma. Comúnmente, el **mayor exponente de estos recursos son los humanos**. Sin embargo, en gran parte de las ocasiones, la adopción de IA en la empresa pasa por la contratación de nuevo personal, experto en este campo, y no tanto por llevar a cabo una inversión para que el personal actual entienda esta tecnología de una mejor forma. El hecho de que una persona con experiencia en una empresa tenga la capacidad de aportar valor en la toma de decisiones cuando se introduce IA en una empresa puede marcar la diferencia a la hora de adoptarla. En otras palabras, los recursos humanos existentes en una empresa son candidatos ideales para identificar qué tareas se pueden automatizar o beneficiar del uso de la IA, de forma que su tiempo se pueda dedicar a aquellas tareas en las que los operadores humanos superan a las máquinas.



Figura 5.32: El comercio electrónico es uno de los dominios que mejor ha gestionado la adopción de IA, destacando principalmente el uso de sistemas de recomendación automática.

En este contexto, es posible plantear **tres preguntas significativas** cuyas respuestas sirvan para dirigir lo que los empleados de una empresa deberían conocer acerca del **uso de IA para facilitar su adopción** [DEMW19]:

- **¿Cómo funciona?** Aunque los miembros de un equipo de trabajo no tengan el *expertise* necesario para construir un módulo de IA o para tomar decisiones sobre qué solución es la mejor para afrontar un determinado problema o reto, sí que deberían tener las nociones básicas sobre cómo procesa información y cómo ofrece respuestas a potenciales preguntas formuladas. En el contexto del aprendizaje automático, una de las principales técnicas empleadas en la actualidad para adoptar IA, resulta especialmente relevante que un empleado sea capaz de distinguir entre cómo aprender un operador humano y cómo *aprende* la máquina cuando se afronta una determinada tarea. A modo de ejemplo, cuando un operador humano trata de detectar posibles patrones en relación a un conjunto de datos de entrada, probablemente tratará de generar

una simplificación de los mismos antes de entender cómo se están comportando. La máquina, por otro lado, usará todos y cada uno de esos datos para inferir patrones de comportamiento. De nuevo, los datos son fundamentales. Por ello, los operadores humanos deberían tener unas nociones básicas a la hora de representar e interpretar datos.

- **¿En qué es buena la IA?** Los operadores humanos deben ser capaces de identificar, e interiorizar, qué tareas son ideales para ser resueltas por una máquina y cuáles no, en el contexto de un determinado proceso de negocio. Tradicionalmente, las tareas que no se han resuelto de manera previa o aquellas para las cuales no se disponen de datos son candidatas a ser resueltas por el personal humano, en primera instancia. Por el contrario, las tareas repetitivas o aquellas en las que existe un conjunto de datos de cierta calidad son potenciales candidatas para ser resueltas, aplicando IA, por una máquina. Debido a que este mecanismos de identificación puede ser abstracto para personas que no estén familiarizadas con la IA, resulta recomendable mostrarle ejemplo concretos de herramientas que basan su funcionamiento en el uso de IA.
- **¿Qué no debería hacer la IA?** Existen tareas que, a priori y considerando la estrategia de la empresa, no deberían ser resueltas de manera artificial. Esta información también es relevante para que un operador humano entienda el alcance del uso de IA a la hora de adoptarla en una empresa. Por ejemplo, y a modo de ejemplo, una empresa podría considerar que un algoritmo no debería ser responsable de gestionar los conflictos entre los compañeros que conforman un equipo de trabajo. Esta última pregunta complementa a las otras dos para definir una referencia transversal en la que todo el equipo de trabajo esté preparado para afrontar la adopción efectiva de IA en su día a día.

5.4.2. Necesidades humanas y culturales

El impacto que la IA ha tenido y está teniendo afecta, culturalmente, al mundo empresarial. A principios de siglo, era difícil imaginar que en la década de 2020 iban a existir trabajos tan relevantes como el de *data scientist*, desarrollador de apps o director de datos, entre otros. **La explosión resultante de la capacidad de disponer, analizar y presentar datos**, junto con su íntima relación con la IA ha posibilitado la creación de puestos profesionales que muy pocos podían pronosticar.

Piense ahora, en lo que ocurrirá en la década de 2040. A priori, parece razonable pensar que, en el futuro, existirán trabajos y roles que hoy en día ni siquiera concebimos. Probablemente, la aparición de nuevos puestos de trabajo será incluso más acusada que la que hemos venido disfrutando en la última década. Esto supone una oportunidad laboral, pero también será necesario **cambiar la mentalidad cultural y profesional** de tanto individuos como empresas.

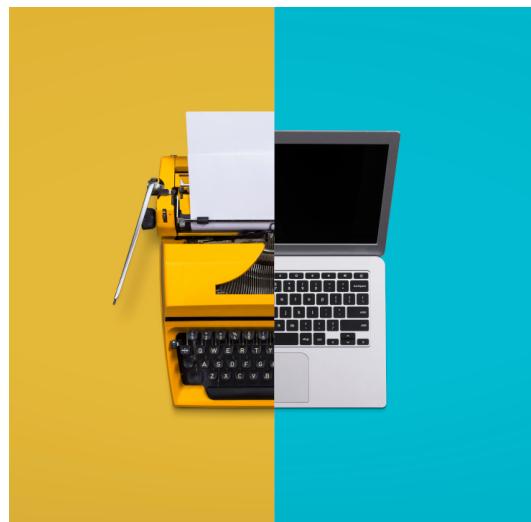


Figura 5.33: El cambio tecnológico de los últimos años ha dirigido, en gran medida, la aparición de trabajos que en su momento eran inimaginables.

En algunos dominios de aplicación, los más afectados por la adopción de IA y de la automatización inteligente, estos cambios ya se están produciendo y están afectando a la forma de pensar y trabajar de los empleados. Un ejemplo representativo es el de los radiólogos, médicos especializados en diagnosticar y tratar enfermedades empleando técnicas de imágenes médicas. Algunos casos concretos son la tomografía computada, los rayos X, o la resonancia magnética nuclear, entre otros. El análisis y la interpretación de estas imágenes representan tareas que un sistema basado en IA podría resolver de manera automática. En este sentido, la utilización de este tipo de sistemas y herramientas se está convirtiendo en una responsabilidad más del radiólogo.

El lector que reflexione sobre esta tendencia puede pensar que, eventualmente, la IA reemplazará a los operadores humanos en un número significativo de dominios y campos de trabajo. Sin embargo, y aunque esto no tiene que ser necesariamente así, sí que es muy probable que el uso de **máquinas y de procesos automáticos** tome el control de gran parte de las tareas que se presten a la automatización, como las **tareas mundanas y repetitivas**¹³.

Para el resto de casos, la **tendencia en los próximos años** se basará en una **cooperación humano-máquina** que se beneficie del valor añadido que aporta cada rol. Retomando el ejemplo del radiólogo y el diagnóstico automatizado a partir del análisis de imágenes médicas, probablemente el médico no necesitará pasar tanto

¹³<https://www2.deloitte.com/us/en/insights/focus/human-capital-trends/2019/impact-of-ai-turning-jobs-into-superjobs.html>



Figura 5.34: El análisis automático de imágenes médicas es uno de los campos en los que la IA está cada vez más presente.

tiempo analizando manualmente imágenes para establecer un diagnóstico, sino que supervisará el emitido por la máquina y podrá dedicar **más tiempo de calidad** a sus pacientes. Este enfoque basado en supervisión posibilita, a su vez, que el sistema basado en IA se realmente del *feedback* ofrecido por el experto humano.

Esta evolución nos llevará a **reinventar**, inevitablemente, la **mentalidad cultural** tanto de los individuos que representan a los empleados de una empresa como de la empresa entendida como organización integral. El cambio de mentalidad servirá (de hecho, ya lo está haciendo) para identificar las necesidades derivadas de este cambio progresivo.

Dicho cambio afectará significativamente a la experiencia de trabajo de los empleados de una empresa, la cual estará cada vez más influenciada por la dependencia tecnológica que la adopción transversal de IA causará. En este sentido, seremos testigos de una relación más fluida y cercana a las máquinas y a los procesos automatizados, dibujando un **escenario de co-working** entre ellos y los humanos. Recuerde que tanto las máquinas como los procesos son elementos diseñados para que el análisis interno de su funcionamiento esté constantemente expuesto a algoritmos que detecten posibles cuellos de botella e identifiquen mejoras de rendimiento. Esta exposición afectará igualmente a los operadores humanos.

En este sentido, uno de los **cambios** cuyo impacto será más relevante a **nivel ético, social y de privacidad** será el incremento en la **monitorización del personal humano** para, precisamente, incrementar su eficiencia e identificar más puntos de automatización o mejora de los procesos que realizan en su día a día. Así, y con carácter general, la IA irá cobrando un mayor protagonismo tanto desde el punto de vista de la automatización continua como desde la perspectiva de integración en el trabajo habitual de un operador humano. Eventualmente, una empresa guiada por

el uso de IA se beneficiará de la capacidad de las máquinas y procesos y de la dedicación de tiempo de calidad de los empleados humanos, vinculado especialmente a las tareas que la empresa acomete por primera vez y a las ventajas de ofrecer una *relación humana* con otras entidades del ecosistema empresarial.

Todo esto nos lleva a reflexionar sobre los cambios sustanciales que veremos en los próximos años acerca de la forma de trabajar. Será diferente a la actual, y los cambios llegarán pronto. En este sentido, en la actualidad ya estamos comprobando cómo los recursos dedicados al trabajo, y su gestión, se están viendo modificados [Mar20]:

- Los **recursos de trabajo** son cada vez más **descentralizados**. Para muchos tipos de trabajo, la localización física de los empleados es una variable que ya no tiene importancia. Por ejemplo, un desarrollador de software que trabaje para una empresa localizada en Madrid podría residir perfectamente en un pueblo de Castilla-La Mancha o en una ciudad del norte del país.
- Los **puestos de trabajo** son cada vez más **líquidos**. En otras palabras, los empleados de las empresas tienden a jugar diferentes roles en la organización, en lugar de desempeñar uno de forma fija.
- Los **empleados** están interiorizando la necesidad de convertirse en **aprendices permanentes**. Esto quiere decir que se están adaptando a la necesidad continua de reciclarse constantemente, debido al impacto que la tecnología tiene en sus lugares de trabajo.

Finalmente, la automatización inteligente permitirá que los trabajadores orienten su trabajo a tratar con cuestiones que son más creativas y orientadas a gestionar por operadores humanos. Esto conlleva la implicación de habilidades más vinculadas a las denominadas *soft skills*, como la resolución de problemas, la comunicación, la empatía o la capacidad de colaborar de manera efectiva. Aunque resulta evidente, merece la pena recordar en este punto que, con respecto a estas habilidades, los humanos somos mejores que las máquinas (al menos, de momento).

5.4.3. Inteligencia colaborativa entre humanos e IA

Nos encontramos en un momento en el que los sistemas y herramientas artificiales, ya estén fuertemente basadas o no en el uso de IA, son especialmente sensibles a la forma en la que interactuamos con ellas. Nunca antes hemos dispuesto de herramientas tan sensibles. Al mismo tiempo, estas herramientas nunca antes han sido entendidas como ahora por parte de operadores humanos. Esta sinergia se puede fortalecer para mejorar el rendimiento y el desempeño de las empresas considerando un horizonte temporal a medio y largo plazo. Una de las claves de este fortalecimiento pasa por focalizar un impacto tecnológico donde las **capacidades de las máquinas y las personas se complementen y aumenten entre sí**.

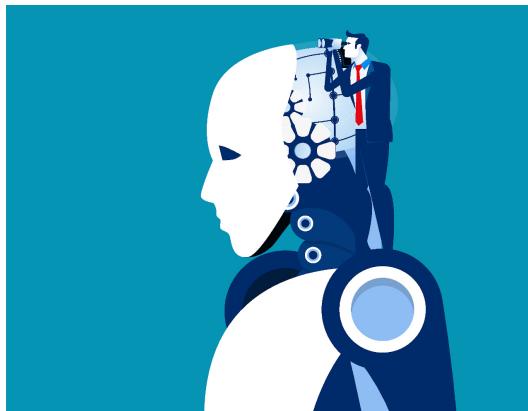


Figura 5.35: En términos de adopción efectiva de IA por parte de una empresa, el impacto a medio y largo plazo vendrá determinado por la capacidad de complementar y aumentar las habilidades de máquinas y humanos.

Si bien previamente se ha discutido el impacto que tiene la automatización inteligente, paulatina e implacable, de tareas que actualmente las realizan los empleados, diversos estudios coinciden en que la clave, a medio y largo plazo, reside en establecer una colaboración efectiva entre máquinas y humanos. La sustitución radical de estos últimos puede generar beneficios a corto plazo, pero el impacto más allá de esta primera etapa es algo que debe estudiarse con cautela. En este sentido, en el libro *Artificial Intelligence: The Insights You Need from Harvard Business Review* [DEM19] se proponen una serie de directrices para que las empresas se beneficien de esta inteligencia colaborativa.

Con respecto a un contexto donde los **humanos asisten a las máquinas**, los primeros deben desempeñar tres roles fundamentales:

- **Entrenamiento.** Se debe enseñar a los algoritmos basados en aprendizaje automático acerca de cómo llevar a cabo el trabajo para el cual están diseñados. En este sentido, resulta esencial utilizar grandes conjuntos de datos de entrenamiento. Además, a los sistemas artificiales debemos indicarles cómo interactuar de manera efectiva con los humanos. Los asistentes personales de Microsoft, Apple, Google y Amazon, entre otros, representan ejemplos concretos donde el entrenamiento ha jugado un papel fundamental.
- **Explicación.** Si bien un motor de IA puede obtener resultados más precisos y en menos tiempo que un humano para una determinada tarea, lo más probable es que no sea capaz de explicar, de manera adecuada y justificada, por qué ha llegado a dicho resultado o conclusión. Esto se conoce como el problema de la caja negra. El concepto *Explainable AI* ha surgido recientemente para dar respuesta a este reto.

- **Mantenimiento.** Además de preparar a las máquinas y a los procesos para trabajar y explicar los resultados alcanzados, existe la necesidad de un mantenimiento constante para garantizar que la IA continúe funcionando de manera correcta, considerando especialmente el potencial daño que pueda causar a terceros. Un ejemplo claro está representado por los sistemas de conducción autónoma.

Por otro lado, y en relación a un contexto donde las **máquinas asisten a las personas**, se consideran otros tres roles fundamentales:

- **Amplificación.** La IA tiene la capacidad de facilitarnos los procesos de análisis y toma de decisiones gracias a que nos puede ofrecer la información adecuada en el momento preciso. En determinados dominios, especialmente sensibles, la última palabra la debe tener un responsable humano, pero esta decisión puede venir sustentada por la información y el conocimiento aportado por la IA.
- **Interacción.** La colaboración persona-ordenador permite que las empresas se relacionen con sus clientes y sus empleados mediante mecanismos efectivos y modernos. De nuevo, piense en el ejemplo de los asistentes virtuales que facilitan tareas del día a día, desde la búsqueda de información contextualizada a la reserva automatizada de recursos.
- **Personificación.** En ámbitos donde el trabajo físico o mental representa un reto para el operador humano, la máquina puede aumentar las capacidades del operador gracias al uso de sensores avanzados o de una capacidad de procesamiento que en la práctica es ilimitada. Piense en los exoesqueletos, controlados remotamente, que algunas empresas utilizan para llevar a cabo tareas de alto riesgo para operadores humanos.

De nuevo, esta colaboración inteligente nos lleva a un escenario que previamente hemos introducido y discutido: la necesidad de **rediseñar la columna vertebral de operaciones** de una empresa para obtener el mayor **beneficio de una potencial adopción de IA** en sus modelos de negocio.

5.4.4. Caso práctico: Google App Engine

Visión general

GAE (Google App Engine) (o GAE) es una **solución PAAS (Platform-As-A-Service)**, integrada en GCP (Google Cloud Platform), que ofrece alojamiento para aplicaciones basadas en la web. Google lo anuncia como una "*plataforma totalmente gestionada y sin servidores*"¹⁴. Fue lanzado el 7 de abril de 2008, como una versión preliminar, y salió de la vista previa en septiembre de 2011. Desde entonces, se ha actualizado continuamente. GAE ofrece entornos de ejecución para

¹⁴<https://cloud.google.com/appengine/docs>

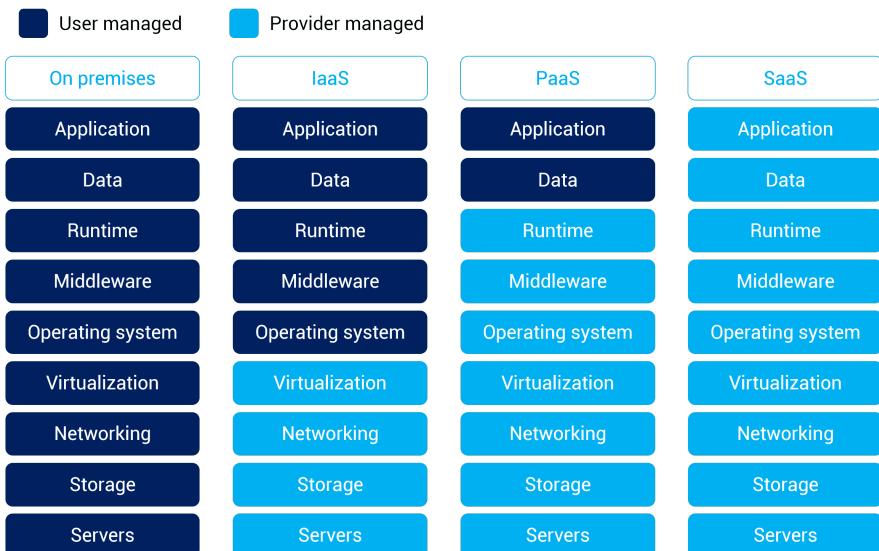


Figura 5.36: Representación visual de la identificación de responsabilidades entre el usuario y el proveedor de la nube en los modelos *On Premises*, IAAS, PAAS y SAAS. Fuente: <https://www.alibabacloud.com/knowledge/difference-between-iaas-paas-saas>

los lenguajes Go, PHP, Java, Python, Javascript, Ruby y el framework .NET. También es posible utilizar entornos de ejecución personalizados, incluyendo cualquier componente adicional que los desarrolladores puedan necesitar, como servidores de aplicaciones o intérpretes de lenguajes. Esta flexibilidad, y las opciones de escalado, monitorización y depuración, hacen de GAE un buen candidato para ejecutar una arquitectura de microservicios.

GAE ofrece **dos tipos de entornos** para ejecutar aplicaciones¹⁵: el entorno *standard* y el entorno *flexible*. Ambos entornos son complementarios y están pensados para diferentes casos de uso; sin embargo, dependiendo del entorno, se aplicarán diferentes características de GAE.

Una de las principales diferencias entre ellos es que el entorno *estándar* es más restrictivo en el sentido de que las aplicaciones se ejecutan en un *sandbox*, utilizando los entornos de ejecución proporcionados con versiones específicas de lenguajes y frameworks. El acceso al sistema operativo, tanto de la aplicación como del desarrollador, está restringido: por ejemplo, la aplicación no puede utilizar el sistema de archivos o los binarios personalizados, y el desarrollador no puede conectarse al entorno de ejecución con herramientas como SSH. La comunicación con otros servicios de GCP es posible con una API proporcionada.

Por otro lado, estas restricciones permiten un escalado mucho más rápido de la aplicación, ya que se utiliza un algoritmo de escalado diseñado a medida para el entorno estándar. Un caso de uso típico para el entorno estándar es una aplicación web sin estado (por ejemplo, una API REST) destinada a obtener tiempos de res-

¹⁵<https://cloud.google.com/appengine/docs/flexible/go/flexible-for-standard-users>

puesta bajos, y que no ejecuta ningún otro proceso en segundo plano. En cuanto a la asignación de recursos, las aplicaciones del entorno estándar pueden ejecutarse en diferentes clases de instancias¹⁶, cada una de las cuales comprende diferentes cantidades de CPU y memoria. Las clases de instancia pueden dividirse en instancias *backend* y *frontend*.

El **entorno flexible**, por su parte, nos permite ejecutar cualquier contenedor Docker personalizado¹⁷ proporcionado por el usuario. Estos contenedores pueden incluir cualquier intérprete de lenguaje y/o entorno de ejecución especificado por el desarrollador, incluyendo cualquier dependencia requerida y binario personalizado. También se permite la ejecución de múltiples procesos. A diferencia del entorno estándar, no es posible llamar a otros servicios de GCP utilizando la API proporcionada por la construcción; en su lugar, es necesario instalar las bibliotecas de cliente de Google Cloud como una dependencia en el contenedor proporcionado. Esto permite que la aplicación sea más portable entre otros servicios. Por último, es posible elegir diferentes tipos de instancias para el entorno flexible, cada una de ellas con diferente CPU y memoria. Esto es diferente al entorno estándar, que limita los recursos de CPU y memoria disponibles para las aplicaciones. Por otro lado, a la hora de escalar, el entorno flexible tiene más limitaciones: a diferencia del entorno estándar, que se escala en cuestión de segundos, las instancias del entorno flexible pueden tardar minutos en aprovisionarse. Esto hace que un mínimo de una instancia flexible deba estar en marcha en todo momento para servir el tráfico de forma ininterrumpida, lo que provoca costes adicionales en aplicaciones en las que el tráfico no es continuo.

Merece la pena mencionar que, para facilitar el desarrollo y las pruebas de las aplicaciones, se ofrece un *nivel gratuito*¹⁸. En el caso de GCP, este *nivel gratuito* se compone tanto de una oferta de crédito inicial de 300 dólares para las nuevas cuentas, que expira al cabo de un año, como de un programa *siempre gratuito* que proporciona acceso gratuito a algunos servicios de GCP. GAE está incluido en el programa *siempre gratis* sólo para el entorno estándar, ofreciendo 28 horas de instancia *frontend* gratis al día, y 9 horas de instancia *backend* gratis al día. El programa *siempre gratis* es, por tanto, otro punto a favor del entorno estándar a la hora de desarrollar y probar pequeñas aplicaciones sin estado.

La figura 5.37 muestra la arquitectura de refencia utilizada para desarrollar aplicaciones con GAE. En la parte de la izquierda se muestra cómo la infraestructura de Google sirve el contenido estático, mientras que en la parte superior se refleja el contenido dinámico, asociado a los módulos de almacenamiento de información (*memcache* y *datastore*). Una de las principales ventajas que ofrece GAE es la capacidad de integrarse con otros servicios ofrecidos por la infraestructura de Google, como todos aquellos asociados a la monitorización y la gestión de logs, representados en la parte inferior derecha de la imagen, pero también con otros que aportan valor añadido en función de los datos gestionados por la aplicación, como por ejemplo *BigQuery*.

¹⁶https://cloud.google.com/appengine/docs/standard#instance_classes

¹⁷<https://www.docker.com>

¹⁸<https://cloud.google.com/free/docs/gcp-free-tier>

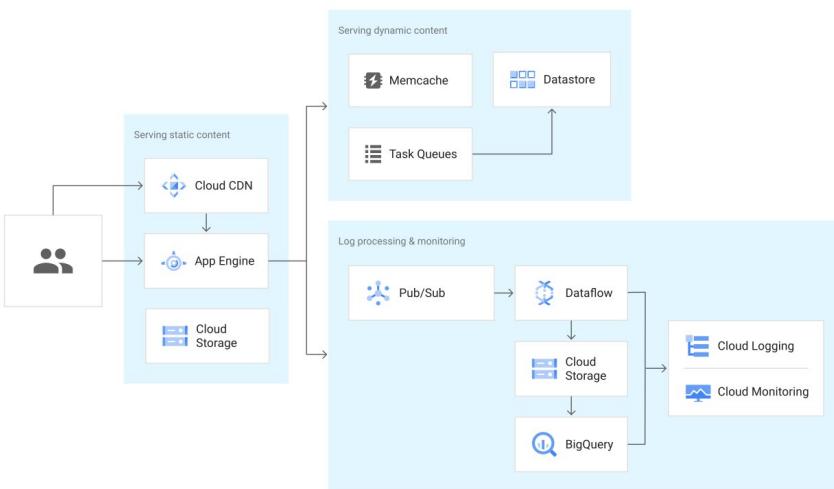


Figura 5.37: Arquitectura de referencia para crear aplicaciones con GAE. Fuente: documentación oficial de GAE.

Principales ventajas de la infraestructura cloud para la gestión de recursos y activos

Desde su creación, los servicios en la nube han sido ampliamente utilizados tanto por las organizaciones tanto para sostener su infraestructura informática como para gestionar sus activos y recursos de manera escalable. Las razones de su notable éxito son variadas; algunas de ellas se introducen a continuación.

- **Economía de escala.** Dado que los recursos informáticos y de almacenamiento se extraen de un conjunto compartido, es posible que el proveedor de la nube reduzca los gastos proporcionalmente al número de clientes existentes. El papel de las economías de escala en la computación en nube ha sido ampliamente discutido [JW11]. Así, y desde un punto de vista efectivo, el uso de la suma de recursos informáticos se optimiza con el modelo de la nube, con una mayor eficiencia en la utilización de la infraestructura compartida, traduciéndose en precios más bajos para el cliente final y en una mayor facilidad de gestión de activos y recursos.
- **Menor inversión inicial.** La computación en la nube permite a las personas y organizaciones pasar de un modelo basado en los gastos de capital (*Capex*) a un modelo basado en los gastos operativos (*Opex*), gracias al modelo de facturación de pago por uso. Por lo tanto, la inversión inicial en recursos informáticos para nuevos proyectos se reduce sustancialmente. Sin embargo, con algunos proveedores de nube, es posible acceder a tarifas con descuento a

cambio de un pago inicial. Esta opción se utiliza habitualmente en proyectos de producción con un número constante de solicitudes. Se ajusta a un modelo *Capex*, lo que puede ser una ventaja para las organizaciones en las que no se ha introducido el modelo *Opex*.

- **Escalabilidad geográfica.** Los principales proveedores de la nube tienen instalaciones en muchas regiones del mundo, lo que supone una ventaja a la hora de escalar una plataforma o aplicación a nivel mundial. Esto ofrece al cliente de la nube la posibilidad de llegar a los usuarios finales con una latencia más baja, proporcionándoles así una mejor experiencia. Para los desarrolladores de aplicaciones, el entorno en el que despliegan su código tiende a ser muy similar en todas las ubicaciones geográficas, lo que facilita el escalado global.
- **Gestión de servicios.** Los servicios en la nube permiten al cliente abstraerse, a voluntad, de las diferentes capas que se manejan. Esto lo libera de la carga de tener que considerar algunos aspectos de las operaciones TIC, como la seguridad de la red, las actualizaciones del sistema operativo, las copias de seguridad, la escalabilidad y la estabilidad, o el cumplimiento de las normas. Dependiendo del servicio, algunos de estos aspectos pueden estar expuestos al cliente en diferentes grados; sin embargo, los servicios gestionados ofrecen la posibilidad de gastar menos en el mantenimiento de la infraestructura y las operaciones de los sistemas. Así, es posible que equipos más pequeños puedan responsabilizarse de las partes más grandes de la infraestructura, lo que permite a las organizaciones moverse e innovar más rápidamente, y reducir drásticamente el tiempo de inactividad.
- **Despliegue ágil.** En un entorno de computación en nube, los nuevos recursos están disponibles rápidamente y se obtienen de un conjunto aparentemente ilimitado de potencia de cálculo y almacenamiento. Esta capacidad de utilizar nuevos recursos bajo demanda permite a las organizaciones reducir los tiempos de despliegue a minutos y, al mismo tiempo, no tener que predecir las necesidades de capacidad futuras.

¡Hola Mundo! con GAE (Google App Engine)

Este apartado muestra el clásico Hello, World! implementado en Python y soportado por la infraestructura de GAE. El código que se discute a continuación sirve tanto para un entorno de pruebas local como para un entorno remoto que se ejecute sobre GAE.



GAE y múltiples lenguajes de programación. Es posible utilizar diversos lenguajes para desarrollar aplicaciones con GAE: Node.js, Java, Ruby, C#, Go, Python o PHP.

Todo proyecto de GAE necesita un archivo, especificado en formato YAML, que tiene como objetivo la **configuración de una aplicación**. Típicamente, este archivo tendrá como nombre app.yaml y especificará el entorno de ejecución y el identificador de versión más reciente. Para la versión 3 de Python, es necesario que el fichero app.yaml contenga al menos una entrada (runtime: python39). El listado 5.9 muestra este fichero para el primer ejemplo discutido.

Listado 5.9: Fichero YAML con la configuración básica de la aplicación en GAE

```
1 runtime: python39
```

En este primer ejemplo, el código del servidor se preocupará únicamente de servir la cadena de texto *Hello, World!* a los clientes web que así lo soliciten. Para llevar a cabo esta tarea, es necesario incorporar en el proyecto algún framework web, como por ejemplo Flask o Django. En este ejemplo, se utilizará Flask. Además, es necesario asociar la ruta o URL con el código responsable de atender las peticiones web que se realizarán sobre dicha ruta. El listado 5.10 muestra una posible implementación del código del servidor de la aplicación *Hello, World!* Note cómo en las líneas ④-7 se define el manejador hello(), responsable de servir la cadena de texto. El código principal, mostrado en las líneas ⑪-15 simplemente representa la configuración del servidor cuando se despliegue localmente para propósitos de prueba.

Listado 5.10: Código del servidor en Python del ejemplo *Hello, World!* en GAE

```
1 from flask import Flask
2
3
4 app = Flask(__name__)
5
6
7 @app.route('/')
8 def hello():
9     return 'Hello World!'
10
11
12 if __name__ == '__main__':
13     # Basic config when running the app on a local server.
14     # When using GAE infrastructure, a webserver process will serve the app.
15     app.run(host='127.0.0.1', port=8080, debug=True)
```

Por otra parte, y aunque GAE está centrado en la parte del servidor de las aplicaciones desarrolladas, suele ser bastante común desarrollar código cliente para probar y depurar la funcionalidad, de manera previa al despliegue en producción del código servidor. El listado 5.11 muestra un ejemplo de código creado para probar, en un entorno local, el código del listado 5.10. En esta sencilla prueba, y como se aprecia en las líneas ⑨-10, se comprueba que el código devuelto por el servidor es 200 y que la cadena de texto devuelta por el servidor se corresponde con 'Hello World!'.

Listado 5.11: Código del cliente en Python del ejemplo *Hello, World!* en GAE

```
1 import main
2
3
4 def test_index():
5     main.app.testing = True
6     client = main.app.test_client()
7
8     r = client.get('/')
9     assert r.status_code == 200
10    assert 'Hello World' in r.data.decode('utf-8')
```

Ejemplo de BigQuery sobre GAE (Google App Engine)

Una de las principales ventajas de emplear GAE es la integración con otros servicios ofrecidos por Google. A continuación, se muestra un sencillo ejemplo de **integración con BigQuery**.

BigQuery es uno de los almacenes de datos de Google, concebido como un servicio de bajo coste, administrado desde la infraestructura de la empresa y que posibilita la **extracción de analíticas a partir de grandes fuentes de datos** (del orden de petabytes). BigQuery permite poner el foco en el análisis de datos, obviando la complejidad subyacente de manejar grandes volúmenes de información e integrando **capacidades de aprendizaje automático**.



Aprendizaje automático y modelado predictivo con BigQuery ML. Mediante BigQuery ML, es posible crear y usar modelos de aprendizaje automático a partir de datos estructurados o semiestructurados, empleando un lenguaje SQL sencillo y a través de un modelo ágil de desarrollo. Los modelos de BigQuery ML se pueden exportar a *Vertex AI* para predecir los datos online.

En la documentación oficial de Google, se plantean tres casos de uso representativos a la hora de utilizar BigQuery:

- Migración de almacenes de datos a BigQuery, pensando en incrementar los niveles de escalado de la aplicación desarrollada.
- Uso de una analítica predictiva, orientada a la detección de patrones en grandes conjuntos de datos.
- Integración de datos en BigQuery, considerando fuentes heterogéneas.

El ejemplo discutido en este apartado gira en torno a la definición de una consulta a un dataset público que contiene información sobre preguntas lanzadas en el portal StackOverflow¹⁹. Se pretende obtener, de forma ordenada de acuerdo al número de visualizaciones, aquellas preguntas que contienen la etiqueta 'google-bigquery'.

¹⁹<https://cloud.google.com/blog/topics/public-datasets>

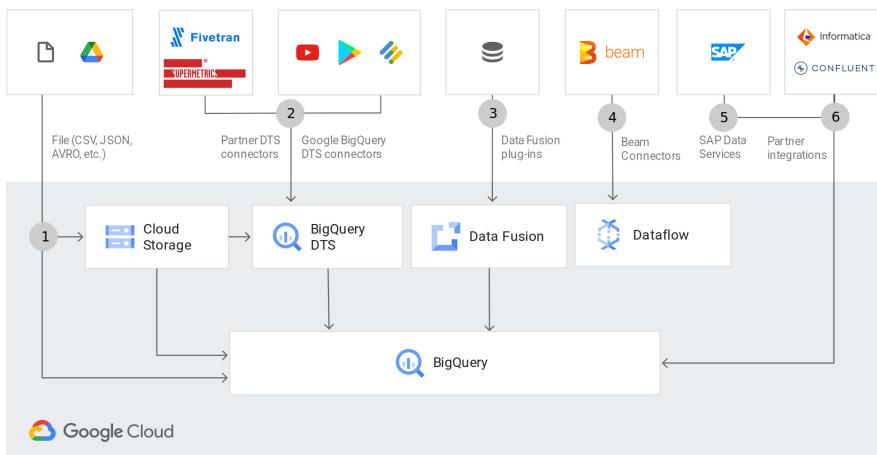


Figura 5.38: Enfoque de integración de datos de fuentes heterogéneas en BigQuery. Fuente: <https://cloud.google.com/bigquery>

Particularmente, el listado 5.13 muestra cómo se construye la consulta SQL en BigQuery cuando se conecta con el servidor desplegado sobre GAE. Esta consulta se limita a los 10 primeros resultados asociados a la etiqueta previamente mencionada. Posteriormente, se redirige al cliente a la vista de resultados.

Listado 5.12: Código principal del ejemplo de integración con BigQuery, donde se lanza una consulta compleja respecto a preguntas formuladas en el portal *StackOverflow*

```

1 import concurrent.futures
2
3 import flask
4 from google.cloud import bigquery
5
6 app = flask.Flask(__name__)
7 bqclient = bigquery.Client()
8
9 @app.route("/")
10 def main():
11     query_job = bqclient.query(
12         """
13             SELECT
14             CONCAT(
15                 'https://stackoverflow.com/questions/',
16                 CAST(id as STRING)) as url,
17             view_count
18             FROM `bigquery-public-data.stackoverflow.posts_questions`
19             WHERE tags like '%google-bigquery%'
20             ORDER BY view_count DESC
21             LIMIT 10
22         """
23     )
24
25     return flask.redirect(
26         flask.url_for(

```

```
27         "results",
28         project_id=query_job.project,
29         job_id=query_job.job_id,
30         location=query_job.location,
31     )
32 )
```

Los resultados obtenidos tras realizar la consulta se renderizan en base al código del listado 5.13. Será necesario ajustar cuestiones de autenticación para emplear la API de BigQuery²⁰.

Listado 5.13: Código del ejemplo de integración con BigQuery para renderizar resultados

```
1 @app.route("/results")
2 def results():
3     project_id = flask.request.args.get("project_id")
4     job_id = flask.request.args.get("job_id")
5     location = flask.request.args.get("location")
6
7     query_job = bigquery_client.get_job(
8         job_id,
9         project=project_id,
10        location=location,
11    )
12
13    try:
14        # Timeout to handle queries that take longer than expected.
15        results = query_job.result(timeout=30)
16    except concurrent.futures.TimeoutError:
17        return flask.render_template("timeout.html", job_id=query_job.job_id)
18
19    return flask.render_template("query_result.html", results=results)
```

²⁰<https://cloud.google.com/docs/authentication/getting-started>

Bibliografía

- [ASQM⁺17] Jose Alvarez-Sabín, Manuel Quintana, Jaime Masjuan, Juan Oliva-Moreno, Javier Mar, Nuria Gonzalez-Rojas, Virginia Becerra, Covadonga Torres, María Yebenes, et al. Economic impact of patients admitted to stroke units in spain. *The European Journal of Health Economics*, 18(4):449–458, 2017.
- [AW18] Andreas M Antonopoulos and Gavin Wood. *Mastering ethereum: building smart contracts and dapps*. O’reilly Media, 2018.
- [B⁺20] Tom B. Brown et al. Language models are few-shot learners. *CoRR*, abs/2005.14165, 2020.
- [BA04] K. Beck and C. Andres. *Extreme Programming Explained: Embrace Change, 2nd Edition*. Addison-Wesley Professional, 2004.
- [Bar93] A.R. Barron. Universal approximation bounds for superpositions of a sigmoidal function. *IEEE Transactions on Information theory*, 39(3):930–945, 1993.
- [BBW21] P. Bornet, I Barkin, and J. Wirtz. *Intelligent Automation*. Independently published, 2021.
- [Bec04] K. Beck. *Test Driven Development: By Example, 1st Edition*. Addison-Wesley Professional, 2004.
- [Bis06] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg, 2006.

- [Bra] T. Bray. On Semantics and Markup, Taxonomy of Markup. <https://www.tbray.org/ongoing/When/200x/2003/04/09/SemanticMarkup#p-1>. Accedido: 23/09/2021.
- [Bur19] A. Burkov. *The Hundred Page ML book*. Andriy Burkov, 2019.
- [C⁺21] M. Chen et al. Evaluating large language models trained on code. *CoRR*, abs/2107.03374, 2021.
- [CRD87] J.H. Coombs, A.H. Renear, and S.J. DeRose. Markup systems and the future of scholarly text processing. *Communications of the ACM*, 30(11):933–947, 1987.
- [Cyb89] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.
- [DEMW19] T.H. Davenport, Brynjolfsson E., A. McAfee, and H.J. Wilson. *Artificial Intelligence: The Insights You Need from Harvard Business Review (HBR Insights Series)*. Harvard Business Review Press, 2019.
- [DK20] Peter H Diamandis and Steven Kotler. *The future is faster than you think: How converging technologies are transforming business, industries, and our lives*. Simon & Schuster, 2020.
- [DO14] Monica DiLuca and Jes Olesen. The cost of brain diseases: a burden or a challenge? *Neuron*, 82(6):1205–1208, 2014.
- [EPM13] T. Erl, R. Puttini, and Z. Mahmood. *Cloud Computing: Concepts, Technology & Architecture*. Pearson, 2013.
- [Fis36] R.A. Fisher. The use of multiple measurements in taxonomic problems. *Annals of eugenics*, 7(2):179–188, 1936.
- [FK11] Richard Foster and Sarah Kaplan. *Creative Destruction: Why companies that are built to last underperform the market—And how to successfully transform them*. Currency, 2011.
- [GB10] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256. JMLR Workshop and Conference Proceedings, 2010.
- [GBc16] I. Goodfellow, Y. Bengio, and A. courville. *Deep Learning*. MIT Press, 2016. <http://www.deeplearningbook.org>.
- [Ger19] A. Geron. *Book Review: Hands-on Machine Learning with Scikit-Learn, Keras, and Tensorflow, 2nd edition*. O'Reilly Media, Inc, USA, 2019.

-
- [GHJV94] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, 1994.
- [Hau85] J. Haugeland. *Artificial Intelligence: The Very Idea*. Massachusetts Institute of Technology, USA, 1985.
- [HF10] J. Humbel and D. Farley. *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*. Addison-Wesley Professional, 2010.
- [HSW89] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.
- [HZRS15] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *CoRR*, abs/1502.01852, 2015.
- [JMMG10] Kirsten Jack, Sionnadh Mairi McLean, Jennifer Klaber Moffett, and Eric Gardiner. Barriers to treatment adherence in physiotherapy outpatient clinics: a systematic review. *Manual therapy*, 15(3):220–228, 2010.
- [JNR⁺19] Catherine Owens Johnson, Minh Nguyen, Gregory A Roth, Emma Nichols, Tahiya Alam, Degu Abate, Foad Abd-Allah, Ahmed Abdelalim, Haftom Niguse Abraha, Niveen ME Abu-Rmeileh, et al. Global, regional, and national burden of stroke, 1990–2016: a systematic analysis for the global burden of disease study 2016. *The Lancet Neurology*, 18(5):439–458, 2019.
- [JW11] Kevin L Jackson and Robert Williams. The economic benefit of cloud computing. *NJVC Executive*, 2011.
- [KM01] N.N. Karnik and J.M. Mendel. Centroid of a type-2 fuzzy set. *information SCIENCES*, 132(1-4):195–220, 2001.
- [Kri21] T.S. Kristensen. *Artificial Intelligence : Models, Algorithms and Applications*. Bentham Science Publishers, 2021.
- [Kur04] Ray Kurzweil. The law of accelerating returns. In *Alan Turing: Life and legacy of a great thinker*, pages 381–416. Springer, 2004.
- [K  92] V. K  rkov  . Kolmogorov's theorem and multilayer neural networks. *Neural Networks*, 5(3):501–506, 1992.
- [Mam74] E.H. Mamdani. Application of fuzzy algorithms for control of simple dynamic plant. In *Proceedings of the institution of electrical engineers*, volume 121, pages 1585–1588. IET, 1974.

- [Mar08] R.C. Martin. *Clean Code: A Handbook of Agile Software Craftsmanship*. Prentice-Hall, 2008.
- [Mar20] Bernard Marr. *The intelligent revolution. Transforming your business with AI*. 2020.
- [McB19] Stephen McBride. These 3 computing technologies will beat moore's law. *Forbes*, 23(04), 2019.
- [McC04] S. McConnel. *Code Complete: A Practical Handbook of Software Construction, 2dn Edition*. Microsoft Press, 2004.
- [MFP10] Michael McCue, Andrea Fairman, and Michael Pramuka. Enhancing quality of life through telerehabilitation. *Physical Medicine and Rehabilitation Clinics*, 21(1):195–205, 2010.
- [O⁺16] A. Oord et al. Wavenet: A generative model for raw audio. *CoRR*, abs/1609.03499, 2016.
- [Qui96] J.R. Quinlan. Learning decision tree classifiers. *ACM Computing Surveys (CSUR)*, 28(1):71–72, 1996.
- [R⁺21] A. Ramesh et al. Zero-shot text-to-image generation. *CoRR*, abs/2102.12092, 2021.
- [RN20] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Pearson, USA, 4 edition, 2020.
- [RN21] S. Russel and P. Norvig. *Artificial intelligence—a modern approach 4th Edition*. 2021.
- [S⁺14] D. Sculley et al. Machine learning: The high interest credit card of technical debt. *SE4ML: Software Engineering for Machine Learning (NIPS 2014 Workshop)*, pages 1–9, 2014.
- [SC07] S. Stan and P. Chan. Toward accurate dynamic time warping in linear time and space. *Intelligent Data Analysis*, 11(5):561–580, 2007.
- [Str13] B. Stroustrup. *The C++ Programming Language, 4th Edition*. Addison Wesley, 2013.
- [TGQS09] Paolo Tormene, Toni Giorgino, Silvana Quaglini, and Mario Stefanelli. Matching incomplete time series with dynamic time warping: an algorithm and an application to post-stroke rehabilitation. *Artificial intelligence in medicine*, 45(1):11–34, 2009.
- [VM15] D. Vallejo and C. Martín. *Desarrollo de Videojuegos. Un Enfoque Práctico. Módulo 1: Arquitectura del Motor*. Amazon CreateSpace, 2015.
- [Wal16] M Mitchell Waldrop. The chips are down for moore's law. *Nature News*, 530(7589):144, 2016.

-
- [WWE⁺20] Hatem A Wafa, Charles DA Wolfe, Eva Emmett, Gregory A Roth, Catherine O Johnson, and Yanzhong Wang. Burden of stroke in europe: thirty-year projections of incidence, prevalence, deaths, and disability-adjusted life years. *Stroke*, 51(8):2418–2427, 2020.
 - [Zad99] L.A. Zadeh. Fuzzy logic = computing with words. *Computing with Words in Information, Intelligent Systems* 1:3–23, 1999.