



UNIVERSITAT POLITÈCNICA DE VALÈNCIA
ESCUELA TÉCNICA SUPERIOR
DE INGENIERÍA INFORMÁTICA

MÁSTER EN BIG DATA ANALYTICS

TRABAJO FIN DE MÁSTER

**Plataforma para entrenamiento reactivo de
modelos en la nube**

Fco. Javier Fernández-Bravo Peñuela

Julio, 2020



UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA

 etsinf

Escola Tècnica
Superior d'Enginyeria
Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Plataforma para entrenamiento reactivo de modelos en la nube

Trabajo Fin de Máster
Máster en Big Data Analytics

Autor: Fco. Javier Fernández-Bravo Peñuela
Tutor: Dr. Germán Moltó Martínez

Julio, 2020

Resumen

Una de las aplicaciones más comunes de las técnicas de aprendizaje automático es la generación de modelos que permiten emitir predicciones sobre tareas de clasificación o regresión. La construcción de estos modelos se lleva a cabo mediante algoritmos que realizan un entrenamiento sobre grandes volúmenes de datos, representativos del universo de discurso a inferir. Cada una de estas técnicas, a su vez, suele poseer una serie de parámetros de configuración cuyos valores también determinan y condicionan el funcionamiento del algoritmo, siendo perentorio hallar la combinación óptima que lleve a producir el mejor modelo posible sobre el conjunto de datos proporcionado. Debido al elevado volumen de datos necesario para construir un modelo capaz de efectuar predicciones precisas en un escenario de cierta complejidad, unido al proceso de prueba y error necesario para identificar el algoritmo idóneo y la configuración de hiperparámetros óptima, el entrenamiento de estos modelos se constituye como un proceso que requiere una gran cantidad de recursos y tiempo de cómputo.

Tradicionalmente, la ejecución de procesos con grandes necesidades de recursos computacionales se ha llevado a cabo en súper-computadores que, por los costes de su adquisición y mantenimiento, sólo estaban a disposición de las instituciones gubernamentales, militares y académicas. En este contexto, las técnicas de computación distribuida posibilitan la división de los procesos de cómputo en unidades más reducidas y su reparto entre diferentes computadores de menor potencia interconectados en red, lo que permite la ejecución de estos procesos de cómputo intensivo empleando equipos asequibles. El advenimiento de las tecnologías en la nube, y particularmente la irrupción de los proveedores públicos de *cloud*, ha traído consigo la capacidad de ejecutar procesos de cómputo sin necesidad de realizar una inversión previa en infraestructura, mediante un modelo de pago por despliegue o de pago por uso sobre los recursos hardware aprovisionados.

La propuesta del presente Trabajo Final de Máster es el diseño e implementación de una plataforma gestionada en la nube y dirigida por eventos para el entrenamiento bajo demanda de modelos de aprendizaje automático. Su principal finalidad es automatizar la selección de la técnica de aprendizaje y la configuración de hiperparámetros óptimas mediante la ejecución paralela de diferentes combinaciones de las mismas, construyendo el modelo más preciso para la resolución de un problema. Para ello, se sirve de implementaciones distribuidas y escalables de los algoritmos de aprendizaje automático y de la disponibilidad de recursos por parte del proveedor *cloud*. La adopción de esta plataforma permite obtener dos importantes ventajas: la reducción del tiempo total de entrenamiento y una gestión inteligente de los recursos aprovisionados, adquiriéndolos y liberándolos dinámicamente en función de la carga de trabajo del sistema y pagando sólo por su tiempo de uso efectivo.

Palabras clave: aprendizaje automático, arquitectura *cloud*, computación distribuida.

Abstract

One of the most common applications of machine learning techniques is the creation of models intended to yield predictions on classification or regression tasks. The crafting of these models is carried out by means of algorithms that perform a training process on large volumes of data, which are representative of the domain of discourse to be inferred. In addition, every one of these techniques is likely to hold a series of configuration parameters whose values also determine the performance of the algorithm, making it necessary to find the optimal combination of them which makes for the best possible model on the dataset supplied. Due to the huge volume of data needed to build a model capable of yielding accurate predictions on a rather complex scenery, along to the trial and error process required for identifying the appropriate algorithm and the optimal hyperparameter configuration, these models' training appoints itself as a process which demands large amounts of time and computational resources.

Traditionally, the execution of highly resource-consuming processes has been carried out on supercomputers which, due to their acquisition and maintenance expenses, were only accessible to governmental, military, and academic institutions. In such a context, distributed computing techniques enable the division of computational processes into smaller units to be shared out between a set of interconnected lesser powered computers, making it feasible for affordable equipment to execute such intensive computing processes. The coming of cloud technologies, and specifically the irruption of public cloud providers, brought with it the ability to execute computational processes without any need for a prior investment in infrastructure, via an either pay-per-deployment or pay-per-use model over the hardware resources having been provisioned.

The present Master Thesis' proposal is the design and implementation of a cloud-managed and event-driven platform for training machine learning models on demand. It aims at automating the selection of the optimal learning technique and hyperparameter configuration through the parallel execution of multiple combinations of them, building the most accurate model for solving a given problem. Such a goal is fulfilled by making the most of the existence of distributed and scalable implementations of machine learning algorithms and the availability of resources from the cloud provider. Thus, the adoption of the platform being proposed provides its users with two prominent advantages: the reduction on the training global time and the intelligent management of provisioned resources, dynamically acquiring and releasing them based on the system's current workload and just paying for their actual usage period.

Keywords: machine learning, cloud architecture, distributed computing.

Tabla de contenidos

Resumen	I
Abstract	III
Tabla de contenidos	V
1. Introducción	1
1.1. Aproximación a los procesos de cómputo intensivo	2
1.2. Propuesta de trabajo	4
1.3. Estructura del documento	5
2. Antecedentes	7
2.1. Plataformas de aprendizaje automático asistido en la nube	8
2.1.1. Amazon SageMaker	8
2.1.2. Google Cloud AI Platform	12
2.1.3. Amazon Rekognition	16
2.2. Bibliotecas de aprendizaje automático distribuido	18
2.2.1. Spark MLlib	18
2.2.2. Horovod	20
2.3. Herramientas para soporte de procesos de entrenamiento	23
2.3.1. MLflow	23
2.3.2. Hyperopt	24
3. Objetivos y requisitos	25
3.1. Especificación de requisitos	25
3.1.1. Requisitos funcionales	25
3.1.2. Requisitos no funcionales	27

4. Diseño de alto nivel	29
4.1. Componentes del sistema	30
4.1.1. Recepción de tareas	30
4.1.2. Ejecución de tareas	31
4.1.3. Envío de informes	39
4.1.4. Clúster Spark	39
4.1.5. Clúster Horovod	40
4.2. Ajuste automatizado de hiperparámetros	41
4.2.1. Exploración en malla (<i>grid</i>)	42
4.2.2. Exploración aleatoria	43
4.2.3. Optimización bayesiana	45
4.3. Algoritmos de aprendizaje automático	49
4.3.1. Clasificación binomial	50
4.3.2. Clasificación multinomial	53
4.3.3. Regresión	56
4.3.4. Agrupamiento (<i>clustering</i>)	60
4.3.5. Redes neuronales y aprendizaje profundo (<i>deep learning</i>)	63
4.3.6. Miscelánea	64
4.4. Formato y esquema de las peticiones	66
5. Arquitectura en la nube	69
5.1. Arquitectura de despliegue sobre Amazon Web Services	70
5.1.1. Recepción de tareas	71
5.1.2. Ejecución de tareas	74
5.1.3. Envío de informes	79
5.2. Clúster Spark	82
5.2.1. Orquestación de clústeres efímeros	82
5.2.2. Ciclo de vida y ejecución de tareas	84
5.2.3. Opciones tecnológicas contempladas	85
5.3. Clúster Horovod	88
5.3.1. Orquestación de clústeres efímeros	88
5.3.2. Ciclo de vida y ejecución de tareas	89
5.3.3. Pila tecnológica adoptada	89

6. Despliegue y ejecución	93
6.1. Procedimiento de despliegue	93
6.2. Ejecución de ejemplo	102
7. Conclusiones y propuestas	111
7.1. Síntesis del trabajo realizado	111
7.2. Propuestas y líneas de trabajo futuro	113
7.3. Conclusión personal	117
A. Esquema JSON del formato de petición	121
Referencias bibliográficas	125

Capítulo 1

Introducción

UNA de las aplicaciones más comunes de las técnicas de aprendizaje automático o *Machine Learning* es la construcción de modelos que permiten realizar predicciones sobre tareas de clasificación o regresión. La construcción de estos modelos se lleva a cabo mediante algoritmos que realizan un entrenamiento sobre grandes volúmenes de datos, representativos del universo de discurso a inferir [HFR04]. Debido al elevado volumen de datos necesario para construir un modelo capaz de efectuar predicciones precisas en un escenario de cierta complejidad, el entrenamiento de estos modelos se constituye como un proceso que requiere una gran cantidad de recursos y tiempo de cómputo.

En numerosas ocasiones la construcción del modelo que mejor logra representar un fenómeno es una tarea de prueba y error, siendo necesario probar con diferentes técnicas o variantes de las mismas hasta identificar la que ofrece una mejor precisión, equilibrada con un rendimiento considerado aceptable. Dentro de cada una de estas técnicas, a su vez, suele existir una serie de parámetros de configuración que también determinan y condicionan el funcionamiento del algoritmo, siendo necesario hallar la combinación de hiperparámetros óptima que lleve a producir el mejor modelo posible sobre el conjunto de datos de entrenamiento proporcionado [Fla12]. Si el entrenamiento que lleva a construir un modelo, con una técnica y una configuración determinados, ya es un proceso costoso de por sí, estas circunstancias llevan a multiplicar el coste de la obtención del modelo más apropiado, en base a la variedad de técnicas y configuraciones de hiperparámetros a tener en cuenta.

El presente trabajo versa sobre el desarrollo de Zygarde, una plataforma para el entrenamiento y optimización de modelos de aprendizaje automático, construida para su despliegue y ejecución sobre la infraestructura de un proveedor de *cloud* público. Su objetivo es hallar el modelo óptimo en base a un conjunto de datos de entrada proporcionado por el usuario, minimizando el tiempo necesario para la identificación y construcción de este modelo, a la par que realizando un uso eficiente de los recursos empleados.

A lo largo de este capítulo se argumenta la catalogación de los procesos de aprendizaje automático como procesos de cómputo intensivo, a partir de lo que se lleva a cabo una introducción a la problemática inherente a este tipo de procesos y diversas aproximaciones a su resolución, organizadas desde una perspectiva cronológica. Entre éstas se encuentran

la computación paralela y la computación distribuida, la última de las cuales evoluciona al siguiente nivel que supone la transición a la computación en la nube. Seguidamente, y partiendo de las bases expuestas, se define la propuesta de trabajo sobre la que se erige el presente documento. Finalmente, se presenta la estructura que sigue el documento, incluyendo un breve resumen del contenido de cada uno de los capítulos que lo componen y el hilo conductor en torno al cual se cohesionan.

1.1 Aproximación a los procesos de cómputo intensivo

Tradicionalmente, la ejecución de procesos con grandes necesidades de recursos computacionales se ha llevado a cabo en súper-computadores, máquinas con prestaciones suficientes capaces de soportar la ejecución de esos procesos en un tiempo o con un rendimiento aceptables. Así, durante años, a medida que surgía la necesidad de ejecutar procesos de cómputo más costosos, se han ido construyendo máquinas cada vez con mayores prestaciones, integradas por componentes más potentes. Las limitaciones físicas sobre la capacidad de integración de transistores en un único chip, entre otros problemas, llevaron a emprender una transición a arquitecturas multinúcleo o multiprocesador, en las que las labores de cómputo se llevan a cabo en paralelo de forma colaborativa en varias unidades computacionales (CPUs), cuyo coste es menor que el de una sola unidad computacional de gran potencia, pero que comparten el resto de los recursos del computador (memoria, disco, buses de interconexión...) [OPA05]. Este enfoque se conoce como computación paralela con memoria compartida. Frente a él, se postula la denominada computación paralela con memoria distribuida, en la que la computación se reparte entre computadores independientes conectados en red entre sí, ya sea a través de Internet o en clústeres interconectados mediante redes con una alta tasa de transferencia de datos.

La ejecución de procesos de cómputo costosos tradicionalmente requería emplear equipos que, por los costes de su adquisición y mantenimiento, sólo estaban a disposición de las instituciones gubernamentales, militares y académicas. Una de las principales ventajas de la computación distribuida es que la posibilidad de dividir los procesos de cómputo en unidades más reducidas y repartirlas entre diferentes computadores interconectados posibilita la ejecución de estos procesos de cómputo empleando equipos asequibles [AGMV08], incluso que comunidades de usuarios puedan unir sus recursos computacionales para colaborar en la resolución de una tarea que no podrían resolver por separado.

Sin embargo, la ejecución de un proceso sobre una red o clúster de computadores no es trivial, puesto que los algoritmos empleados deben haber sido específicamente diseñados o adaptados para operar en estas condiciones maximizando su eficiencia [GKGK03]. Un factor límite en estos algoritmos distribuidos es su escalabilidad, que se define como la capacidad para mejorar su rendimiento a medida que se añaden más nodos. Las posibilidades que ofrece la computación distribuida para facilitar la ejecución de procesos costosos han propiciado

la implementación de versiones de algoritmos de aprendizaje automático preparadas y optimizadas para su ejecución distribuida, basadas en modelos de programación divisibles y escalables como MapReduce [DG04][GLNS⁺05], con el objetivo primordial de reducir el tiempo necesario para el entrenamiento de los modelos. [GFD06]

Por otra parte, el advenimiento de las tecnologías en la nube, y particularmente la irrupción en la última década de los proveedores públicos de *cloud* [FGJ⁺09], ha traído consigo la capacidad para ejecutar procesos de cómputo sin necesidad de realizar una inversión previa en infraestructura, mediante un modelo de pago por despliegue. De esta forma, se pueden alquilar unos determinados recursos hardware ubicados en los centros de datos del proveedor, pagando sólo por el tiempo que éstos están en uso y pudiendo en cualquier momento adquirir o liberar recursos de cómputo a medida que sea necesario (IaaS, *Infrastructure as a Service*). Pese a que inicialmente los servicios ofrecidos se limitaban a instancias de cómputo individuales (máquinas virtuales) dentro de un catálogo en base a sus prestaciones y precio por unidad de tiempo facturable, las oportunidades derivadas de sus numerosas aplicaciones han llevado a que los principales proveedores de *cloud* público desarrollen soluciones para el despliegue automatizado y gestionado de clústeres de tecnologías como Hadoop¹ y Spark², derivando en plataformas donde los clientes pueden ejecutar sus propios procesos o servicios (PaaS, *Platform as a Service*).

La tendencia actual es la de facilitar a los desarrolladores de software el despliegue de sus aplicaciones y servicios en la nube con garantías de alta disponibilidad, proporcionándoles una abstracción sobre la infraestructura (física y virtualizada) de despliegue. Por supuesto, esto implica que el diseño de las aplicaciones debe estar orientado a su despliegue sobre este tipo de plataformas, siguiendo modelos de arquitectura como las basadas en microservicios. El siguiente paso en esta evolución por parte de los proveedores públicos de *cloud* han sido las arquitecturas *serverless* [JSSS⁺19], que operan bajo un modelo reactivo dirigido por eventos. En ellas, el desarrollador especifica funcionalidad software que se ejecuta ante la ocurrencia de un evento, pudiendo a su vez realizar acciones que impliquen la activación de otros eventos. La ejecución de este software se realiza sobre un entorno autocontenido de prestaciones configurables, obviando cualquier noción sobre la máquinas o grupo de máquinas sobre las que se lleva a cabo cada ejecución de esta funcionalidad, siendo todo ello gestionado por el proveedor *cloud* en base a un acuerdo de nivel de servicio (SLA, *Service Level Agreement*). Las principales ventajas de este planteamiento, además de la delegación de la gestión de la infraestructura, son la construcción de aplicaciones distribuidas basadas en un *workflow* escalable dirigido por eventos, así como la transición de un modelo de pago por despliegue a un modelo de pago por uso real, en el que la facturación se produce en base al tiempo de consumo efectivo de los recursos dinámicamente aprovisionados.

¹Apache Hadoop, <https://hadoop.apache.org>

²Apache Spark™ - Unified Analytics Engine for Big Data, <https://spark.apache.org>

1.2 Propuesta de trabajo

En base a toda la información expuesta hasta el momento, la propuesta del Trabajo Final de Máster es el diseño e implementación de Zygarde, una plataforma gestionada en la nube y dirigida por eventos para el entrenamiento bajo demanda de modelos de aprendizaje automático. Su principal finalidad es automatizar la selección de la técnica de aprendizaje y la configuración de hiperparámetros óptimas mediante la ejecución paralela de diferentes combinaciones de las mismas, construyendo el modelo más preciso para la resolución de un problema que requiera la aplicación de mecanismos de aprendizaje automático. Zygarde aprovecha la disponibilidad de recursos por parte del proveedor *cloud* elegido para emplear implementaciones distribuidas de los algoritmos de aprendizaje automático sobre un número (limitado aunque potencialmente elevado) de nodos, reduciendo el coste frente al uso de nodos de mayores prestaciones y a la vez aumentando el rendimiento al dividir el problema del proceso de entrenamiento en unidades de menor magnitud que pueden computarse concurrentemente. Igualmente, la ejecución paralela del proceso de entrenamiento para diferentes técnicas y combinaciones paramétricas permite completar este proceso en un tiempo ostensiblemente menor al que supondría el entrenamiento secuencial de cada uno de estos posibles modelos para poder proceder a su evaluación y comparativa. Zygarde almacenará tanto los modelos generados como la combinación de técnica y parámetros que ha dado lugar a cada uno de ellos, permitiendo asimismo la realización de predicciones (ejecución de la tarea de clasificación o regresión a partir de nuevos datos de entrada) sobre el modelo en la nube o entrenar el modelo específico con un conjunto de datos mayor.

Las principales ventajas que emanan del uso de la plataforma propuesta son dos, que llevan respectivamente un alto rendimiento en la resolución del problema y un uso eficiente de los recursos de cómputo dirigido a minimizar su coste económico. En primer lugar, el hecho de que el proceso de obtención del modelo óptimo se lleve a cabo mediante su división y distribución en dos niveles, teniendo en cuenta tanto el entrenamiento simultáneo con diferentes técnicas y configuraciones de hiperparámetros como la ejecución distribuida de cada entrenamiento individual, supone una reducción muy importante en el tiempo necesario para obtener dicho modelo, en comparación con la prueba secuencial de diferentes combinaciones sin aplicar técnicas de paralelismo. Por otra parte, la división y distribución el problema permiten su resolución empleando máquinas de menores prestaciones, reduciendo el coste frente al aprovisionamiento de una máquina con prestaciones suficientes para resolver el problema de forma centralizada en un tiempo aceptable. Además, el carácter reactivo de Zygarde según el paradigma *serverless* contribuye a optimizar la utilización de los recursos del proveedor *cloud*, logrando un alto nivel de abstracción sobre la gestión de la infraestructura y reduciendo los costes económicos derivados de su aprovisionamiento al pagar sólo por su tiempo de uso efectivo.

1.3 Estructura del documento

El presente trabajo, tal como se ha descrito, trata el diseño y la construcción de una plataforma para el entrenamiento y optimización de procesos de aprendizaje automático denominada Zygarde, concebida para su despliegue y ejecución sobre la infraestructura de un proveedor de *cloud* público. Este documento sigue la siguiente estructura:

- 1. Introducción.** El capítulo actual ha ofrecido un estudio exploratorio general sobre las tareas de cómputo intensivo, entre las que se encuentran los procesos de aprendizaje automático, junto a diversas estrategias para abordar su resolución, desde las arquitecturas multiprocesador a la computación distribuida dirigida por eventos en la nube. En base a la información ofrecida, se ha presentado la propuesta del presente Trabajo Fin de Máster: Zygarde, una plataforma para entrenamiento reactivo de modelos en la nube.
- 2. Antecedentes.** En este capítulo se realiza un recorrido sobre las tecnologías ya existentes dirigidas a la ejecución de tareas de aprendizaje automático de forma distribuida, estudiando qué tipo de procesos y algoritmos engloban. También se estudian las principales plataformas ofrecidas por los proveedores de *cloud* público para facilitar la realización de tareas de aprendizaje automático en su infraestructura, señalando sus principales funcionalidades, virtudes y carencias, en base a las que refinar la especificación de requisitos de Zygarde.
- 3. Objetivos y requisitos.** En este capítulo, partiendo de la información expuesta respecto a la propuesta de trabajo y las tecnologías y plataformas ya existentes en este ámbito, se detalla la lista de objetivos a cubrir por el diseño de Zygarde. Esta lista de objetivos se traduce en la especificación formal de requisitos, tanto funcionales como no funcionales, que la implementación resultante ha de cumplir.
- 4. Diseño de alto nivel.** En este capítulo se describe el diseño de Zygarde atendiendo a una descomposición modular basada en componentes en cuyo análisis se emplea un enfoque descendente en lo que se refiere al nivel de abstracción. A lo largo de todo el capítulo se procede con especial énfasis en relación al modelo asíncrono de interacción entre componentes, dirigido por eventos. Gracias al desacoplamiento que proporciona este modelo, el flujo de ejecución sobre el sistema se estructura en un *workflow* o secuencia de tareas sobre las que pueden aplicarse técnicas de escalado independientes.

- 5. Arquitectura en la nube.** Una vez presentado el diseño de alto nivel de la plataforma, en este capítulo se expone la arquitectura del sistema, planteada desde su concepción para su despliegue sobre la infraestructura y los servicios de un proveedor *cloud* de elección, optando en este caso por Amazon Web Services³. Se presta especial atención a los procedimientos de aprovisionamiento y liberación dinámicos de recursos ante la entrada de peticiones en el sistema y el volumen de carga de trabajo que puedan suponer. Asimismo, se justifica la elección entre el uso de servicios específicos del proveedor *cloud* (PaaS) o la orquestación propia del despliegue sobre instancias de cómputo (IaaS), buscando equilibrar la relación entre complejidad, coste económico y portabilidad.
- 6. Despliegue y ejecución.** En este capítulo se muestra el procedimiento de despliegue de los elementos estáticos de la plataforma sobre la infraestructura del proveedor *cloud* y se estudia la ejecución originada en respuesta a la recepción de una petición de entrenamiento sobre un determinado conjunto de datos. Se aportan evidencias de la ejecución de los diferentes componentes, categorizados como servicios, y del tránsito de los datos a través de ellos, junto con los resultados obtenidos. De esta forma se contribuye a verificar el correcto funcionamiento del sistema y a mostrar en acción, tanto a nivel específico como del sistema en su conjunto, cada uno de los servicios que conforman la arquitectura.
- 7. Conclusiones y propuestas.** Finalmente, se elabora una síntesis detallada de las conclusiones que se extraen tras la materialización del trabajo expuesto y sus principales aspectos a destacar. Tras confirmar el cumplimiento de las metas alcanzadas en el contexto del alcance inicialmente propuesto, se enumeran posibles líneas de trabajo futuro a partir de la labor realizada, así como elementos que hayan quedado fuera del ámbito de este trabajo y cuya realización pudiera resultar útil e interesante.

³AWS | Cloud Computing - Servicios de informática en la nube, <https://aws.amazon.com/es/>

Capítulo 2

Antecedentes

EN este capítulo se realiza un recorrido sobre las principales plataformas ofrecidas por los proveedores de *cloud* público para facilitar la realización de tareas de aprendizaje automático en su infraestructura. El estudio de sus principales funcionalidades y aspectos positivos contribuirá a la definición de los objetivos y requisitos, tanto funcionales como no funcionales, a cubrir por Zygarde, tal como se especifican en el Capítulo 3. El análisis de los principios de diseño de estas plataformas, identificando sus puntos fuertes así como posibles carencias, también servirá como una referencia a la hora de estructurar determinados elementos de la arquitectura de Zygarde, ayudando a plantear soluciones a problemas específicos que puedan surgir durante el proceso de diseño y desarrollo.

También se estudian algunas de las tecnologías ya existentes dirigidas a la ejecución de tareas de aprendizaje automático de forma distribuida, enumerando qué tipos de procesos y algoritmos engloban. Con la finalidad de que Zygarde pueda dar soporte a la ejecución distribuida de una gran variedad de algoritmos y procesos de aprendizaje automático, estas herramientas podrán integrarse en Zygarde. El uso de implementaciones ya probadas y optimizadas de versiones distribuidas de algoritmos de aprendizaje automático permitirá acortar y simplificar el proceso de desarrollo de la plataforma frente a la implementación por medios propios de estos algoritmos, una tarea en absoluto trivial y sí muy propensa a error.

De forma complementaria, se ofrece una perspectiva sobre algunas herramientas que dan soporte a los procesos de aprendizaje automático a lo largo de sus diferentes fases y, en concreto, durante la ejecución de baterías de entrenamientos con diferentes característica y configuraciones de parámetros, con el fin de determinar el modelo que mejores resultados alcanza en cuanto a precisión y rendimiento. Estas herramientas también podrán integrarse en Zygarde, organizando y sistematizando la gestión de los procesos de entrenamiento, especialmente cuando varios de éstos se realicen de forma paralela (y, a su vez e internamente, distribuida), además de facilitando la importación, exportación y despliegue de modelos ya construidos.

2.1 Plataformas de aprendizaje automático asistido en la nube

Tal como se enuncia en la descripción del capítulo, esta primera sección se dirige al estudio de algunas de las principales plataformas ofrecidas por los proveedores de *cloud* público para facilitar la realización de tareas de aprendizaje automático sobre su infraestructura. En particular se estudian las plataformas que ofrecen Amazon Web Services y Google Cloud, **Amazon SageMaker** y **Google Cloud AI Platform**, respectivamente. Complementariamente, también se somete a escrutinio la plataforma **Amazon Rekognition**, un servicio en la nube para reconocimiento visual que constituye un caso de éxito de un SaaS cuya plataforma subyacente realiza procesos de aprendizaje automático, sin que el usuario del servicio tenga por qué tener nociones de aprendizaje automático ni de la infraestructura de despliegue y ejecución de sus peticiones.

El resultado de este análisis contribuye tanto a la especificación de la lista de objetivos y requisitos de Zygarde como a plantear determinados elementos de su arquitectura y el diseño de sus componentes.

2.1.1 Amazon SageMaker

Amazon SageMaker¹ es un servicio completamente gestionado, dirigido a desarrolladores y científicos de datos, que permite construir, entrenar y desplegar con rapidez modelos de aprendizaje automático. El desarrollo tradicional de estos modelos es una labor iterativa que resulta costosa y compleja, algo que se acentúa ante la falta de oferta de herramientas que actúen sobre el flujo completo de los procesos de aprendizaje automático, lo que provoca que lo más común sea tratar de cubrir este ciclo integrando herramientas de diversa índole y propósito, una tarea propensa a error y que requiere una gran cantidad de tiempo y esfuerzo. SageMaker busca paliar estas carencias proporcionando todos los componentes necesarios para coordinar el ciclo completo del desarrollo de un proceso de aprendizaje automático, integrados en una solución autocontenido que opera sobre la propia infraestructura de Amazon Web Services, de tal forma que sea posible construir modelos de alta calidad y llevarlos a producción con mucho menos esfuerzo y asumiendo un menor coste.

La Tabla 2.1 muestra las herramientas que Amazon SageMaker proporciona para dar soporte al ciclo completo de desarrollo de modelos de aprendizaje automático, señalando en qué fase se engloba cada herramienta y proporcionando una breve descripción de su cometido. Seguidamente, se detallan, de entre las herramientas expuestas, aquellas que se considera que resultan más interesantes desde la perspectiva de la propuesta de trabajo de Zygarde.

¹Amazon SageMaker, <https://aws.amazon.com/sagemaker/>

Preparación de datos	Construcción	Entrenamiento y ajuste	Optimización y despliegue
Amazon SageMaker Ground Truth Construcción y gestión de conjuntos de datos de entrenamiento			Amazon SageMaker Studio Entorno de desarrollo integrado (IDE, <i>Integrated Development Environment</i>) para procesos de aprendizaje automático
Amazon SageMaker Autopilot Construcción y entrenamiento automático de modelos		Amazon SageMaker Model Monitor Detección automática de desvíos entre modelo y datos	
Amazon SageMaker Notebooks <i>Notebooks</i> con ejecución sobre infraestructura elástica	Amazon SageMaker Experiments Captura, organización y búsqueda de cada ejecución realizada	Amazon SageMaker Neo Optimizador para el despliegue sobre plataformas específicas	
AWS MarketPlace Algoritmos y modelos preconstruidos	Amazon SageMaker Debugger Depura y traza ejecuciones de entrenamiento	Amazon Augmented AI Incorpora validación humana a las predicciones de los modelos	
	Automatic Model Tuning Optimización de hiperparámetros		

Tabla 2.1: Herramientas proporcionadas por Amazon SageMaker, organizadas según la fase del ciclo de desarrollo a la que prestan soporte.

PREPARACIÓN DE DATOS

Amazon SageMaker Ground Truth² constituye una herramienta de apoyo en la creación de corpus y conjuntos de datos, una de las bases a la hora de realizar entrenamientos a partir de los que obtener modelos de alta calidad. Esta herramienta lleva a cabo su labor etiquetando registros mediante el criterio humano gracias al servicio **Amazon Mechanical Turk³**, lo que supone un medio “escalable” que permite obtener un conjunto de datos del volumen que sea requerido, con relativa rapidez y liberando al usuario de asumir y gestionar esa labor.

CONSTRUCCIÓN, ENTRENAMIENTO Y AJUSTE

Amazon SageMaker Autopilot⁴ consiste en una herramienta que automatiza el ciclo de construcción, entrenamiento, ajuste y optimización de modelos; encargándose de inspeccionar los datos, procesarlos, extraer características relevantes, escoger los algoritmos más apropiados, entrenar y ajustar varios modelos, medir su desempeño y clasificarlos.

Generalmente, los procesos de construcción de modelos de aprendizaje automático se han abordado de dos formas completamente diferentes. La primera consiste en seleccionar manualmente las características a conservar del conjunto de datos, seleccionar el algoritmo y optimizar los parámetros del entrenamiento, todo con el fin de tener un control completo sobre el diseño del modelo y entender todos los pasos y decisiones responsables de su confección. En el otro extremo, el segundo método consiste en utilizar enfoques automatizados, de tal forma que todas las decisiones que conducen a la construcción del modelo se lleven a cabo de forma autogestionada, pero sin proporcionar apenas información sobre los elementos y decisiones que llevan a tal creación. Un modelo generado mediante técnicas de automatización y búsqueda puede ofrecer un rendimiento que un operador humano, mediante un proceso manual de prueba y error, sólo podría alcanzar con un consumo mucho mayor de tiempo y recursos. Sin embargo, la opacidad de las herramientas disponibles con este fin produce que los usuarios experimentados no terminen de fiarse de ellas, al no poder obtener información sobre qué decisiones se han tomado en la creación del modelo, entender cómo estas decisiones han devenido en la obtención de un mejor o peor rendimiento sobre el conjunto de datos disponible, o ser capaz de recrear el modelo de forma manual mediante la reproducción de los pasos seguidos.

En este contexto, Amazon SageMaker Autopilot incorpora todas las ventajas de un enfoque automatizado, al operar como una plataforma autogestionada, pero elimina los inconvenientes descritos, ya que ofrece al usuario visibilidad y control directos sobre los pasos que lleva a cabo de tal forma que éste pueda tener constancia de las acciones que se llevan a cabo en cada etapa.

²Amazon SageMaker Ground Truth, <https://aws.amazon.com/sagemaker/groundtruth/>

³Amazon Mechanical Turk, <https://www.mturk.com>

⁴Amazon SageMaker Autopilot, <https://aws.amazon.com/sagemaker/autopilot/>



El usuario puede escoger de entre los cincuenta mejores modelos generados aquel que más se ajusta a su caso de uso, permitiéndole, por ejemplo, elegir cómo balancear el equilibrio entre el rendimiento y la precisión del modelo a la hora de realizar predicciones. Esta plataforma permite que usuarios con muy distintos trasfondos puedan hacer uso de técnicas de aprendizaje automático según sus necesidades. Por una parte, el hecho de que esta plataforma se halle integrada en el **IDE Amazon SageMaker Studio** posibilita que personas sin experiencia en procesos de aprendizaje automático puedan generar modelos con facilidad, únicamente proporcionando un conjunto de datos en forma tabular y seleccionando la columna objetivo de la predicción. En el otro extremo, científicos de datos con experiencia también pueden llegar a servirse de ella para automatizar la construcción de un modelo de base a partir del que sus equipos especializados puedan progresar, evolucionando y mejorando la calidad del modelo mediante técnicas avanzadas derivadas del juicio experto.

Amazon SageMaker Experiments⁵ organiza, registra y evalúa las ejecuciones de procesos de entrenamiento y sus resultados. Puesto que la construcción de un modelo que obtenga el rendimiento deseable suele implicar realizar cientos de pruebas en las que, de forma iterativa, se emplean diferentes algoritmos, conjuntos de datos de entrenamiento, configuraciones de la plataforma y valores de parámetros, entre otros, es deseable contar con una herramienta que mantenga un registro de estas ejecuciones. Amazon SageMaker Experiments lleva a cabo esta labor de forma automática, registrando los parámetros de entrada, configuración y resultados de cada entrenamiento y almacenándolos catalogados como “experimentos”. Al igual que Amazon SageMaker Autopilot, esta herramienta se encuentra integrada en el IDE Amazon SageMaker Studio, desde donde el usuario puede examinar los experimentos actualmente activos, buscar experimentos previos en base a sus características, consultar el histórico de los experimentos realizados en sesiones anteriores junto con sus resultados y comparar todos ellos de forma visual.

OPTIMIZACIÓN Y DESPLIEGUE

Además, la plataforma Amazon SageMaker, a través de su IDE, proporciona facilidades para realizar el despliegue sobre su infraestructura de los modelos creados y poder comenzar a realizar predicciones en tiempo real o sobre lotes de dato. El uso de la propia infraestructura de Amazon Web Services permite gozar de alta disponibilidad debido al uso de técnicas de replicación con autoescalado y al despliegue sobre múltiples zonas de disponibilidad, consiguiendo redundancia y tolerancia a fallos geográfica. Puesto que al solicitar el despliegue es posible indicar la plataforma hardware sobre la que se desea que se realice (procesadores Intel o ARM, o GPUs NVIDIA) el uso de la herramienta **Amazon SageMaker Neo**⁶ permite

⁵Amazon SageMaker Experiments,

<https://aws.amazon.com/about-aws/whats-new/2019/12/introducing-amazon-sagemaker-experiments-organize-track-and-compare-your-machine-learning-training-experiments-on-amazon-sagemaker/>

⁶Amazon SageMaker Neo, <https://aws.amazon.com/sagemaker/neo/>

optimizar el modelo para la plataforma hardware de destino. Gracias a esta optimización, el modelo adaptado obtiene una precisión idéntica al modelo original en términos de resultados pero un rendimiento hasta dos veces superior al que se obtendría desplegando el modelo genérico, es decir, no optimizado para la plataforma hardware de despliegue.

Ante una orden de despliegue de un modelo específico, Amazon SageMaker se encarga de iniciar las instancias de cómputo necesarias, desplegar el modelo y habilitar un *endpoint* securizado al que las aplicaciones pueden conectarse para realizar predicciones sobre el modelo. Gracias a esto, la única integración que deben realizar las aplicaciones es la conexión al *endpoint* expuesto, sabiendo que, pese a que el modelo desplegado pueda cambiar, no es necesario realizar ninguna modificación sobre las propias aplicaciones que operan con él.

2.1.2 Google Cloud AI Platform

Google Cloud AI Platform⁷, antes denominado Google Cloud ML Engine, es una plataforma gestionada para facilitar el desarrollo de proyectos de aprendizaje automático en todas sus fases, desde su concepción hasta el despliegue, tratando de dotar de rapidez al proceso de desarrollo y minimizando los costes asociados. La plataforma se compone de un conjunto de herramientas que se ejecutan en la infraestructura de Google Cloud, cubriendo diferentes fases del ciclo de desarrollo.

La Tabla 2.2 muestra estas herramientas, señalando en qué fase se engloba cada una y proporcionando una breve descripción de su cometido. Seguidamente se detallan, de entre las herramientas expuestas, aquellas que se considera que resultan más interesantes desde la perspectiva de la propuesta de trabajo de Zygard.

CONSTRUCCIÓN, ENTRENAMIENTO Y AJUSTE

AI Platform Training⁸ permite utilizar algoritmos predefinidos directamente sobre un conjunto de datos para realizar un entrenamiento o bien implementar manualmente una aplicación que se encargue de ello. Un aspecto interesante de esta herramienta es que, sobre determinados algoritmos tales como la redes neuronales, permite realizar un entrenamiento distribuido con el fin de aplicar técnicas de paralelismo y mejorar el rendimiento, ofreciendo tres estrategias entre las que el usuario puede elegir la que resulte más adecuada:

- Entrenamiento con paralelismo de datos y actualización síncronas.
- Entrenamiento con paralelismo de datos y actualizaciones asíncronas.
- Entrenamiento paralelo de modelos.

⁷Google Cloud AI Platform, <https://cloud.google.com/ai-platform/>

⁸AI Platform Training, <https://cloud.google.com/ml-engine/docs/training/>

			
Preparación de datos	Construcción	Entrenamiento y ajuste	Optimización y despliegue
Big Query	Deep Learning VM Image	AI Platform Training	AI Platform Prediction
Data Labeling Service	AI Platform Notebooks	TFX Tools	
Transfer Service	Kubeflow Pipelines		Kubeflow
Cloud Storage			
Cloud Dataprep			
Cloud Dataflow			
Cloud Dataproc			

Tabla 2.2: Herramientas proporcionadas por Google AI Platform, organizadas según la fase del ciclo de desarrollo a la que prestan soporte.

En el entrenamiento paralelo de modelos, la aproximación es sencilla: varios modelos se entrena en diferentes nodos de forma simultánea, sin necesidad de aplicar división y distribución de datos ni de que dichos nodos interactúen entre sí durante la ejecución del algoritmo de entrenamiento. En las dos variantes con paralelismo de datos, el mismo algoritmo con la misma configuración se ejecuta sobre todos los nodos, a partir de lo que cada nodo entrena sobre una sección del conjunto de datos, calculando vectores de gradientes parciales. Estos vectores de gradientes se concentran en el nodo *maestro* mediante operación de agregación y se suman, actualizando los parámetros del modelo global. La forma en la que el nodo maestro recibe los resultados parciales de los demás nodos y aplica la operación de agregación para conformar su estado actualizado se determina en función de la elección entre la variante con actualizaciones síncronas o asíncronas.

El conjunto de herramientas de construcción y entrenamiento de modelos de la plataforma también dispone de una utilidad para el ajuste de hiperparámetros⁹. Esta utilidad se sirve de **Google Vizier** [GSM⁺17], la herramienta utilizada por Google para la optimización de sus procesos, con el fin de determinar la combinación óptima de valores de los hiperparámetros.

⁹Overview of hyperparameter tuning,

<https://cloud.google.com/ml-engine/docs/hyperparameter-tuning-overview>

El uso de esta herramienta proporciona una flexibilidad notable en cuanto a la forma en la que se lleva a cabo el proceso de búsqueda, pudiendo especificar la métrica a aplicar (es decir, cuál es el valor escalar que, mediante la variación de los parámetros de entrada, se desea maximizar o minimizar) y el número máximo de pruebas a realizar, lo que permite alcanzar un equilibrio entre la precisión del modelo obtenido y el coste en términos temporales (y, al tratarse de una plataforma en la nube, también económicos y computacionales) en los que incurre el proceso de optimización. Por lo general, se recomienda que el número máximo de pruebas a realizar no sea menor que diez veces el número de hiperparámetros de entrada empleados.

Puesto que las pruebas sobre diferentes combinaciones de valores de los hiperparámetros se pueden realizar en paralelo, también se permite utilizar múltiples instancias de cómputo para aprovechar este potencial y reducir el tiempo empleado en el proceso de optimización de hiperparámetros. En este caso, se puede indicar un factor de paralelismo máximo con el que limitar los recursos en uso de forma simultánea, alcanzando el equilibrio que se desee entre el coste temporal del proceso de optimización y el coste económico derivado de la cantidad de recursos hardware aprovisionados.

Google Vizier por defecto utiliza un algoritmo de optimización bayesiana en la búsqueda sobre los parámetros de entrada que permite hallar la combinación de sus valores que obtiene el mejor resultado sobre la métrica seleccionada. Además de este algoritmo, también es posible indicar otros, tal como una búsqueda en *grid*, un método de búsqueda exhaustiva que mediante fuerza bruta comprueba todas las posibles combinaciones en un espacio n -dimensional, considerando cada parámetro de entrada como un eje con un dominio de valores asociado. También es posible implementar algoritmos de optimización o búsqueda de carácter específico e integrarlos en la herramienta mediante el cumplimiento de una interfaz de operaciones. La decisión de qué algoritmo emplear es delicada y debe tenerse en cuenta su influencia sobre otros factores como la paralelización de pruebas. Por ejemplo, la búsqueda exhaustiva es inherentemente paralelizable pero requiere probar todas las combinaciones posibles. En el caso de la optimización bayesiana, mientras que ésta requiere un menor número de pruebas que la búsqueda en *grid*, los niveles de paralelización de los que puede beneficiarse se encuentran limitados, puesto que los resultados de las pruebas previas se emplean para determinar los valores a utilizar en las pruebas subsiguientes, estableciendo dependencias y sincronización entre ellas. Por lo general, y salvo que se posean motivaciones específicas para utilizar un determinado algoritmo de búsqueda, se aconseja optar por la optimización bayesiana, que se postula como la más adecuada en la mayor parte de los escenarios de uso.

Finalmente, Google Cloud AI Platform pone a disposición de los usuarios máquinas con GPUs que permiten acelerar el cómputo de determinados tipos de operaciones, reduciendo el tiempo de entrenamiento. Sin embargo, no proporcionan ninguna optimización ni interfaz

específica para la operación con GPUs. Por este motivo, establecen que es responsabilidad del desarrollador, en caso de optar por utilizar GPUs, asegurarse de emplear una biblioteca que, como TensorFlow, esté específicamente preparada para servirse de este hardware y mejorar su rendimiento.

DESPLIEGUE

AI Platform Prediction¹⁰ permite realizar inferencia sobre modelos almacenados y desplegados en la plataforma en la nube, obteniendo predicciones para nuevos datos de entrada. Dicho despliegue es también responsabilidad del servicio de predicciones, que se encarga de gestionar la infraestructura necesaria para garantizar la disponibilidad del modelo desplegado. Al solicitar el proceso de despliegue, el usuario puede especificar límites sobre la cantidad de recursos hardware a dedicar, así como definir la política de escalado a aplicar. Ésta puede variar desde la utilización de un número fijo de instancias de cómputo hasta la aplicación de un modelo elástico para la adquisición y liberación elástica de recursos, de tal forma que se ofrezca un tiempo de respuesta consistente y aceptable incluso ante un volumen de carga variable, en términos del número de peticiones de predicción recibidas.

El servicio de predicciones no sólo se puede emplear para exponer un modelo de cara a su uso en un entorno de producción, sino también para realizar baterías de pruebas en un entorno de preproducción, desde pruebas de precisión hasta pruebas de estrés, verificando que el modelo desplegado desempeña su labor correctamente y, en caso contrario, determinando qué es necesario corregir o ajustar antes de proceder al despliegue sobre el entorno de producción real.

Kubeflow Pipelines^{11,12} es una herramienta que permite diseñar y manipular secuencias de componentes de aprendizaje automático a través de los que definir procesos de aprendizaje que, representados en la forma de un grafo acíclico dirigido, constan de varias etapas ordenadas las cuales realizan sucesivas transformaciones sobre los datos de origen (preprocesado, transformación, entrenamiento de un modelo, etc). Esta herramienta permite adoptar un enfoque orientado a componentes, donde cada componente se presenta como una pieza de software encapsulada en un contenedor Docker y resulta sencillo modificar el *pipeline* y sustituir unos componentes por otros. El SDK que ofrece la herramienta permite a los desarrolladores poseer un control completo sobre todas y cada una de las acciones que se realizan en cada fase del proceso de aprendizaje, así como alterar la estructura de procesos a voluntad.

¹⁰Prediction overview | AI Platform Prediction,
<https://cloud.google.com/ml-engine/docs/prediction-overview>

¹¹Overview of Kubeflow Pipelines | Kubeflow,
<https://www.kubeflow.org/docs/pipelines/overview/pipelines-overview/>

¹²Understanding Kubeflow pipelines and components,
<https://cloud.google.com/ai-hub/docs/kubeflow-pipeline>



La interfaz gráfica de Kubeflow Pipelines permite gestionar experimentos, ejecuciones y trabajos planificados sobre los *pipelines* construidos. Puesto que el *pipeline* se encuentra compuesto por unidades de software encapsuladas en contenedores, se facilita su gestión, despliegue y distribución. Además, se puede llegar al punto de crear un repositorio de componentes (privado para una empresa o abierto para la comunidad) que facilite la reutilización de componentes ya existentes en escenarios similares.

El despliegue de un *pipeline* tiene lugar sobre un clúster de Kubernetes, la plataforma de orquestación de contenedores de Google que, debido a su amplia aceptación y soporte por parte de la comunidad, se ha convertido en la tecnología *de facto* en este ámbito. Gracias a ello, los responsables del despliegue pueden tratar con una tecnología ya conocida en lugar de tener que familiarizarse con una nueva. Además, una vez construido el *pipeline*, existe la posibilidad de realizar la ejecución de los procesos de entrenamiento y predicción sobre un clúster Kubernetes desplegado en unas instalaciones ajenas al proveedor *cloud*, obteniendo independencia frente al proveedor de infraestructura.

2.1.3 Amazon Rekognition

Amazon Rekognition¹³ es un servicio en la nube que permite integrar análisis de imágenes y vídeos en los programas de los usuarios a través de su API. Sus aplicaciones más comunes son el reconocimiento de objetos, personas, texto, escenas y actividades, así como la dilucidación de si una imagen posee contenido inapropiado. Además, incorpora funcionalidades para análisis y reconocimiento facial de alta precisión, permitiendo llevar a cabo tareas orientadas a la seguridad como verificación de identidad o identificación de individuos.

La simplicidad de su uso proviene del hecho de que, aunque emplea técnicas de aprendizaje automático sobre modelos ya entrenados que se refinan sucesivamente, toda esta complejidad queda oculta al usuario del servicio, que no tiene por qué poseer conocimientos de aprendizaje automático y puede explotar el potencial del servicio únicamente subiendo imágenes o vídeos a S3 (el servicio de almacenamiento en la nube de Amazon Web Services) y solicitando su procesamiento a través de la interfaz de operaciones de Rekognition. Al constituirse como un SaaS desplegado en la infraestructura en la nube de Amazon, ofrece disponibilidad y escalabilidad máximas abstrandiendo al usuario completamente de los recursos hardware empleados para el procesamiento de sus solicitudes y de la naturaleza del software que se ejecuta en éstos. La interoperabilidad con Amazon S3 permite integrarlo en aplicaciones dirigidas por eventos, en los que la simple acción de almacenar una imagen o un vídeo en un *bucket* de S3 por parte de una aplicación puede iniciar un proceso de análisis o reconocimiento visual, que automáticamente al completarse se concatene con la ejecución de una serie de acciones.

¹³Amazon Rekognition, <https://aws.amazon.com/rekognition/>

Las principales características de este servicio, tal como se presentan en su portal web, son las siguientes:

- Integración sencilla: Amazon Rekognition simplifica la tarea de añadir capacidades de análisis visual a una aplicación, gracias a contar con interfaces de aplicación sencillas y no requerir conocimientos de aprendizaje automático.
- Análisis de lotes y en tiempo real: Es posible ejecutar análisis en tiempo real de vídeos suministrados desde Amazon Kinesis Video Streams, o analizar series de imágenes conforme son depositadas en Amazon S3. En el caso de procesos de gran duración o volumen, es posible utilizar AWS Batch para automatizar el análisis de miles imágenes o vídeos.
- Aprendizaje continuo: El servicio se halla entrenando sobre nuevos datos continuamente, mejorando así su capacidad para reconocer y discriminar objetos, composiciones y actividades.
- Bajo coste: El modelo de coste de Amazon Rekognition viene dictado por el número de imágenes o minutos de vídeo analizados, así como los datos de rostros almacenados para ser específicamente identificados mediante reconocimiento facial de individuos.
- Completamente gestionado: Amazon Rekognition ofrece tiempos de respuesta constantes sin importar el volumen de solicitudes realizado o la carga que esté soportando el sistema, incluso si ésta es de decenas de millones de peticiones.

Aunque Amazon Rekognition no es una plataforma para procesos de aprendizaje automático, como sí lo son Amazon SageMaker y Google Cloud AI Platform, sino un servicio completo con una utilidad específica, se incluye en esta sección porque se trata de un ejemplo de éxito del uso de técnicas de aprendizaje automático en la nube. Supone un caso de uso apropiado, con características muy interesantes, del que extraer conceptos y elementos de diseño que pueden resultar de utilidad a la hora de plantear la arquitectura e interfaz pública de operaciones de Zygarde.



2.2 Bibliotecas de aprendizaje automático distribuido

Esta segunda sección se dedica al análisis de algunas de las tecnologías ya existentes dirigidas a la ejecución distribuida de tareas de aprendizaje automático, proporcionando implementaciones de algoritmos que se ejecutan sobre una infraestructura distribuida. En concreto, se someten a escrutinio dos bibliotecas. La primera de ellas es **Spark MLlib**, que se caracteriza por su velocidad al operar minimizando los accesos a memoria secundaria y abarca algoritmos de clasificación, regresión y agrupamiento, entre otras muchas utilidades. En segundo lugar se describe **Horovod**, dirigida específicamente a procesos de aprendizaje profundo sobre TensorFlow, y que como objetivo primordial busca optimizar el aprovechamiento de los recursos hardware disponibles.

Tal como se refiere en la introducción del presente capítulo, la integración de estas bibliotecas en Zygardel elimina la necesidad de desarrollar implementaciones propiamente distribuidas de los algoritmos más típicos de aprendizaje automático, una tarea sumamente compleja. Por el contrario, la adopción de implementaciones ya probadas y optimizadas de estos algoritmos permitirá simplificar el proceso de desarrollo de la plataforma, de tal forma que los esfuerzos dedicados puedan enfocarse en dar solución a otras potenciales problemáticas.

2.2.1 Spark MLlib

Spark MLlib¹⁴ es la biblioteca de aprendizaje automático de Apache Spark, concebida con el objetivo de facilitar la utilización de técnicas de aprendizaje automático de forma distribuida y escalable. Al estar construida sobre Spark, aprovecha sus características relativas a la ejecución distribuida de procesos que operan sobre un gran volumen de datos, trabajando en memoria principal y evitando así el incremento de la latencia derivado del tráfico constante de datos entre el disco y niveles superiores de la jerarquía de memoria. En particular, el modelo de computación de Spark se adecúa especialmente bien a la naturaleza iterativa de muchos algoritmos de aprendizaje automático, especialmente aquellos que deben ejecutarse repetidamente hasta alcanzar un determinado umbral de convergencia, tal como se puede observar en el gráfico de la Figura 2.1, que corresponde a una comparativa de los tiempos de resolución de un problema de regresión logística tanto sobre Hadoop como sobre Spark. Al ejecutarse sobre un clúster Spark, esta biblioteca permite trabajar con conjuntos de datos que se encuentren en cualquiera de los orígenes y fuentes de datos que Spark admite.

Las operaciones que Spark MLlib proporciona se encuentran optimizadas para su ejecución distribuida de forma escalable y eficiente sobre un clúster de Spark, al estar implementadas haciendo uso de modelos de programación que, con la ayuda del planificador de tareas YARN, permiten distribuir los datos y procesarlos concurrentemente con un rendimiento muy superior al de otros modelos como el MapReduce tradicional de Hadoop.

¹⁴MLlib | Apache Spark, <https://spark.apache.org/mllib/>



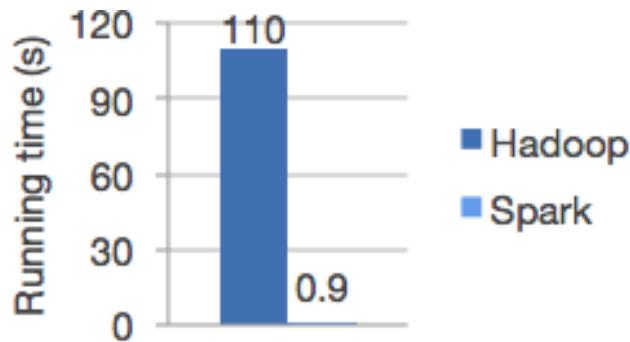


Figura 2.1: Comparativa de tiempos de ejecución en regresión logística sobre Hadoop y Spark¹⁵

A continuación se enumeran las principales herramientas, funcionalidades y operaciones que Spark MLlib provee, categorizándolas según su ámbito:

■ **Algoritmos de aprendizaje automático:**

- Clasificación: regresión logística, *naive Bayes*...
- Regresión: regresión generalizada lineal, regresión de supervivencia...
- Árboles de decisión, bosques aleatorios (*random forest*), árboles de gradiente
- Recomendación: Mínimos cuadrados alternativos (ALS, *Alternating Least Squares*)
- Clustering: K-Medias, mezclas gaussianas (GMMs, *Gaussian Mixtures*)...
- Modelado de tendencias: Asignación Latente de Dirichlet (LDA, *Latent Dirichlet Allocation*)
- Reglas de asociación, conjuntos frecuentes, minado de patrones secuenciales

■ **Flujo de procesos de aprendizaje automático:**

- Transformación de características: Normalización, *hashing*...
- Construcción de pipelines de algoritmos
- Evaluación de modelos y ajuste de hiperparámetros
- Persistencia de elementos de aprendizaje automático: Almacenamiento y recuperación de modelos y *pipelines* de procesos

■ **Utilidades adicionales:**

- Álgebra lineal distribuida: Descomposición en valores singulares (SVD, *Singular Value Decomposition*), análisis de componentes principales (PCA, *Principal Component Analysis*)...
- Estadística: Resumen y agregación de resultados, comprobación de hipótesis...

¹⁵Apache Spark™ - Unified Analytics Engine for Big Data, <https://spark.apache.org>

Finalmente, uno más de sus rasgos reseñables es que, aunque Spark se encuentra escrito en Java, la interfaz de Spark MLlib para Python hace uso intensivo de NumPy¹⁶ con el fin de que el paso de datos entre ambos lenguajes pueda interoperar de la forma más eficiente posible con vectores y matrices de características [Van16]. Este aspecto contribuye a facilitar el uso de Spark MLlib desde Python, actualmente el lenguaje de programación más utilizado¹⁷ en la ejecución de procesos de aprendizaje automático.

2.2.2 Horovod

Horovod^{18,19} es un *framework* de entrenamiento distribuido destinado a programas que realizan procesos de aprendizaje profundo o *deep learning* mediante TensorFlow. Esta herramienta, desarrollada por Uber, fue concebida con el objetivo de facilitar y acelerar la ejecución de procesos de aprendizaje profundo mediante la distribución de datos y cómputo. [SDB18]

Por lo general, los procesos de entrenamiento en técnicas de aprendizaje profundo se llevan a cabo en GPUs. Si bien el uso paralelo de múltiples GPUs puede acelerar en gran medida estos procesos en comparación al uso de una sola unidad de cómputo, esta aproximación supone dos problemas a tratar. Por una parte, la biblioteca de entrenamiento debe soportar comunicación entre GPUs, lo que, en caso de implementarse mediante técnicas que limiten la escalabilidad, puede implicar incurrir en una sobrecarga que penalice el rendimiento. Por otra parte, en función de la API ofrecida por la biblioteca de entrenamiento, las modificaciones a acometer sobre el código fuente pueden llegar a ser considerables.

Si bien existe una versión propiamente distribuida de TensorFlow, el proceso de adaptar un programa que hace uso de TensorFlow clásico para hacer que sea distribuido no es sencillo y se encuentra en gran parte infradocumentado. Además, su escalabilidad se aleja mucho de ser la ideal, como se puede apreciar en la Figura 2.2, algo especialmente llamativo sobre todo en procesos de cómputo como los basados en redes neuronales multicapa, que por su naturaleza y capacidad de parallelización son susceptibles de obtener grandes beneficios de la adopción de un enfoque distribuido. Esta inadecuada resolución de los dos problemas antes mencionados lleva con frecuencia a que los desarrolladores eviten utilizar la versión distribuida de TensorFlow, ciñéndose a la versión clásica sobre una sola GPU y renunciando al mayor rendimiento que potencialmente podría alcanzarse con un enfoque distribuido.

¹⁶NumPy, <https://numpy.org>

¹⁷The State of the Octoverse: machine learning - The GitHub Blog,
<https://github.blog/2019-01-24-the-state-of-the-octoverse-machine-learning/>

¹⁸Horovod: Distributed training framework for TensorFlow, Keras, PyTorch, and Apache MXNet,
<https://github.com/horovod/horovod>

¹⁹Meet Horovod: Uber's Open Source Distributed Deep Learning Framework for TensorFlow,
<https://eng.uber.com/horovod/>

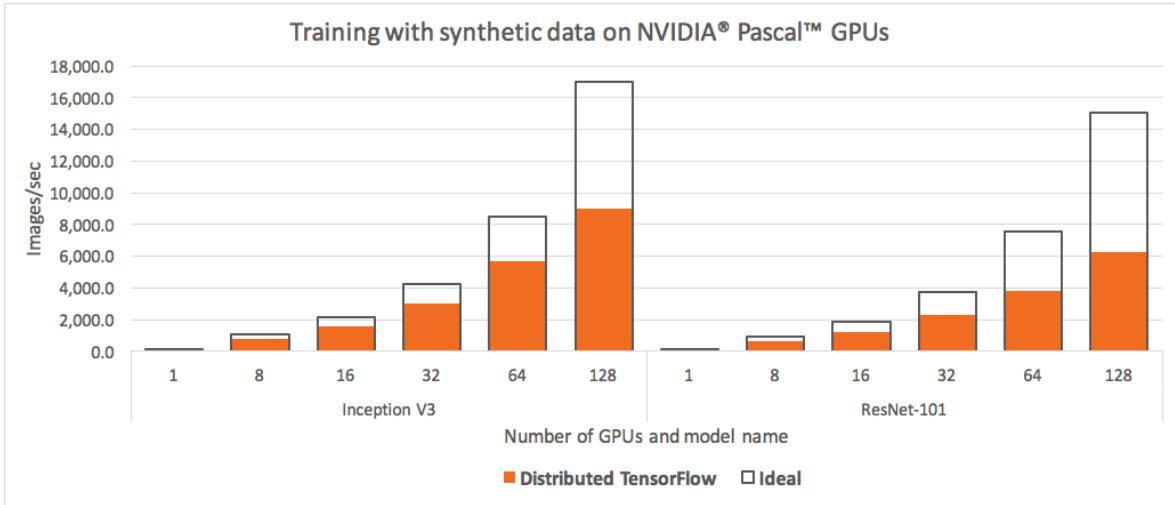


Figura 2.2: Comparativa de la tasa de procesado e imágenes por segundo utilizando TensorFlow distribuido respecto a la escalabilidad ideal teórica (calculada multiplicando la tasa obtenido con una sola GPU por el número de GPUs). La versión distribuida de TensorFlow muestra una mala escalabilidad que redunda en un desaprovechamiento de los recursos hardware disponibles. [SDB18]

La presunción de que obtener un alto nivel de escalabilidad es factible en este escenario, la ejecución distribuida de procesos de aprendizaje profundo sobre GPUs, proviene de un artículo publicado por Facebook [GDG⁺17], en el que se combinan, con óptimos resultados, principios de paralelismo de datos con una técnica innovadora para el ajuste de la tasa de aprendizaje. Así, los desarrolladores de Horovod han tratado de dotar a su plataforma de un nivel de escalabilidad que permita conseguir el máximo aprovechamiento posible de las prestaciones del hardware disponible, simplificando asimismo las modificaciones a realizar sobre el código fuente para obtener una versión distribuida de un programa que haga uso de TensorFlow.

El diseño en el que Horovod se fundamenta para alcanzar dicha escalabilidad se basa en un algoritmo publicado por Baidu denominado *ring-allreduce*²⁰, que a su vez se inspira en otro algoritmo propuesto con anterioridad [PY09]. El algoritmo *ring-allreduce* emplea una estrategia de cálculo y difusión de las medias de los gradientes obtenidos en los nodos, lo que permite eliminar el principal cuello de botella relativo a las interacciones entre unidades de cómputo existente en la implementación distribuida propia de TensorFlow. Así, busca minimizar las comunicaciones e interacciones necesarias entre los nodos y optimizar el reparto de datos, haciendo un uso lo más óptimo posible de los recursos hardware disponibles en la red de cómputo.

²⁰Bringing HPC Techniques to Deep Learning - Andrew Gibiansky, <https://andrew.gibiansky.com/blog/machine-learning/baidu-allreduce/>

En la implementación realizada por los desarrolladores de Horovod, la coordinación y comunicación entre los nodos en los que se lanzan los procesos individuales de TensorFlow se lleva a cabo utilizando Open MPI²¹, una implementación del estándar MPI (*Message Passing Interface*)²². Además, la comunicación entre GPUs se realiza mediante NCCL²³, la biblioteca de comunicaciones colectivas de NVIDIA, compatible con MPI y las operaciones de difusión y agregación de datos que proporciona, consiguiendo así una solución optimizada para la ejecución de procesos distribuidos sobre múltiples GPUs contenidas en equipos diferentes conectados en red.

Todas estas decisiones de diseño llevan, en el caso de Horovod, a obtener una mejora de rendimiento frente a la versión tradicional de TensorFlow distribuido que puede visualizarse en el gráfico de la Figura 2.3.

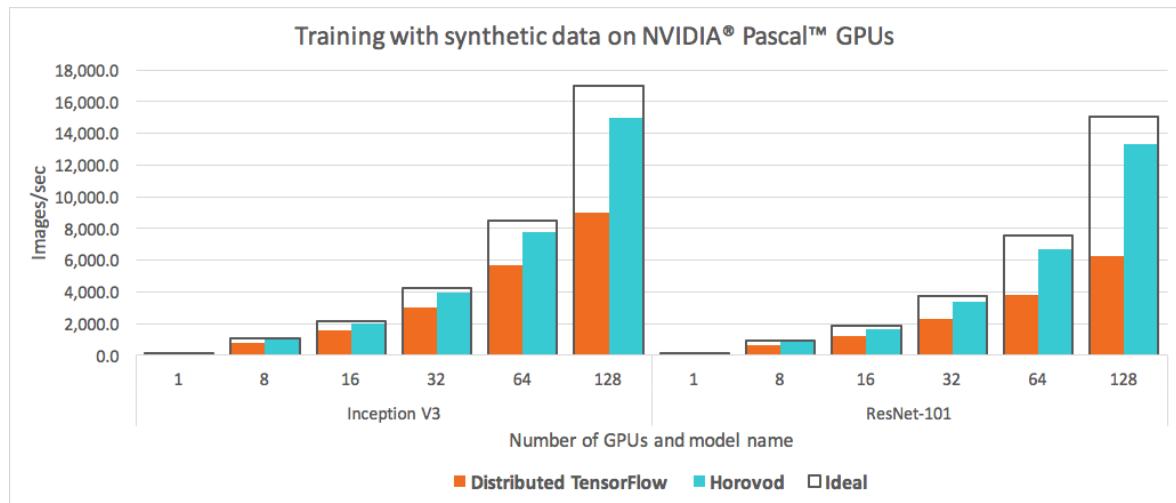


Figura 2.3: Comparativa de la tasa de procesado de imágenes por segundo utilizando la versión distribuida propia de TensorFlow y Horovod, ejecutando un proceso de entrenamiento distribuido sobre diferentes números de GPUs NVIDIA Pascal, con el fin de comprobar la escalabilidad alcanzada frente a la escalabilidad teórica óptima respecto al aprovechamiento de los recursos hardware disponibles. [SDB18]

²¹Open MPI: Open Source High Performance Computing, <https://www.open-mpi.org>

²²MPI Forum, <https://www.mpi-forum.org>

²³NVIDIA Collective Communications Library (NCCL), <https://developer.nvidia.com/nccl>

2.3 Herramientas para soporte de procesos de entrenamiento

Como cierre al capítulo y complemento de las plataformas y bibliotecas vistas hasta el momento a lo largo de las secciones previas, en esta última sección se muestran dos herramientas que dan soporte a los procesos de aprendizaje automático a lo largo de sus diferentes fases. Estas herramientas son de utilidad sobre todo en lo que se refiere a la gestión de los modelos construidos mediante procesos de entrenamiento con diferentes algoritmos y combinaciones de parámetros, a partir de los que seleccionar el modelo óptimo.

La primera de las herramientas sometidas a estudio es **MLflow**, una plataforma orientada a la gestión del ciclo de vida de proyectos de aprendizaje automático en sus distintas fases, la cual se divide en tres herramientas que abarcan diferentes fases de este ciclo. Seguidamente, se procede a comentar **Hyperopt**, una herramienta que almacena los resultados de los experimentos y ejecuciones de entrenamiento, realizando una búsqueda inteligente sobre el espacio de valores de los hiperparámetros para hallar la distribución óptima sin necesidad de probar todas las combinaciones posibles.

2.3.1 MLflow

MLflow²⁴ es una plataforma de código abierto desarrollada por Databricks para la gestión del ciclo de vida de proyectos de aprendizaje automático en sus distintas fases. Comprende tres herramientas:

- **MLflow Tracking:** Persiste el código, los datos, la configuración y los resultados de las ejecuciones y pruebas realizadas, facilitando y automatizando entre otros el proceso de determinar la combinación óptima de los valores de los hiperparámetros. Se presenta como un *wrapper* para múltiples bibliotecas de aprendizaje automático, entre ellas Spark MLlib, que persiste y organiza por sí mismo los resultados de las ejecuciones realizadas, liberando al desarrollador de la tarea de comprobar el resultado de ejecución realizada y compararlo con el histórico de ejecuciones para determinar la combinación hiperparamétrica óptima.
- **MLflow Projects:** Formato de empaquetado de *pipelines* y *workflows* de ejecución para posibilitar su reproducibilidad en diferentes plataformas, así como automatizar la repetición de la ejecución de un proceso sobre una configuración o serie de configuraciones.

²⁴MLflow - A platform for the machine learning lifecycle, <https://mlflow.org>

- **MLflow Models:** Formato de empaquetado de modelos de aprendizaje automático, compatible con diversas bibliotecas de aprendizaje con amplia adopción, tanto centralizadas como distribuidas. Gracias al empaquetado y almacenamiento de estos modelos en un formato específico, permite desplegar los modelos ya almacenados sobre diferentes plataformas y herramientas, alojadas físicamente tanto *on-premises* como en la nube.

2.3.2 Hyperopt

Hyperopt²⁵ es una biblioteca de código abierto para Python destinada a determinar la combinación óptima de valores de los parámetros de configuración que obtiene un mejor resultado y minimiza el tiempo de convergencia, maximizando por tanto el rendimiento. En lugar de llevar a cabo un procedimiento de fuerza bruta para hallar la mejor combinación, esta herramienta emplea optimización bayesiana para realizar una búsqueda inteligente sobre el espacio de dominio de combinaciones paramétricas, evitando de esta forma la necesidad de probar todas las posibles combinaciones de valores.

²⁵Hyperopt: Distributed Hyperparameter optimization, <https://github.com/hyperopt/hyperopt>

Objetivos y requisitos

EN este capítulo, partiendo de la información expuesta en los Capítulos 1 y 2 respecto a la propuesta de trabajo y las tecnologías, herramientas y plataformas ya existentes en ámbitos relacionados con su temática, se proporciona la lista de objetivos que se proponen como metas del presente trabajo. Estos objetivos se traducen en la especificación formal de requisitos, tanto funcionales como no funcionales, que el diseño de Zygarde y la implementación resultante han de cumplir. En aquellos elementos en los que resulta necesario un mayor nivel de detalle, cada uno de estos requisitos se encuentra desglosado en varios puntos concretos, detallando qué se pretende obtener y las limitaciones y condicionantes a considerar para la resolución del problema.

3.1 Especificación de requisitos

La meta del presente Trabajo Fin de Máster es el desarrollo de la plataforma presentada en la Sección 1.2. Partiendo de la descripción de esta meta, se listan los siguientes requisitos, que deben alcanzarse a lo largo del proceso de desarrollo del sistema y con los que la plataforma fruto de dicho desarrollo debe cumplir.

3.1.1 Requisitos funcionales

- La plataforma, ante la petición de un usuario, ejecutará una tarea de entrenamiento en el contexto de un proceso de aprendizaje automático.
- En la petición de entrenamiento, el usuario indicará la configuración relativa a la tarea de entrenamiento a realizar, incluyendo la ubicación del *dataset*.
 - Idóneamente, el *dataset* de entrenamiento se encontrará en el sistema de almacenamiento de archivos del proveedor de infraestructura en la nube, con el fin de poder acceder a él de forma eficiente y minimizando costes derivados de la transferencia de datos a través de la red.
 - El *dataset* podrá encontrarse en diferentes formatos, tanto de texto (ej., CSV) como binarios (ej., Parquet, Avro), correspondiendo en todo caso a datos organizados en una estructura tabular válida, sin deficiencias de contenido ni formato.



- La configuración de la tarea de entrenamiento se especificará en formato JSON conforme a un esquema por definir.
- La petición de entrenamiento podrá realizarse desde diferentes puntos de acceso de la plataforma, que puedan servir a casos de uso variados y clientes de diferente naturaleza. En concreto, se proponen los siguientes:
 - Una API REST.
 - Una cola de mensajería direccional a través de una URL.
 - El depósito de un archivo con la configuración de la tarea de entrenamiento en una determinada ubicación del sistema de almacenamiento de archivos del proveedor de infraestructura en la nube.
- En la configuración de la tarea de entrenamiento, se indicarán los algoritmos y métodos con los que se desea que se realice el entrenamiento.
- Los usuarios especificarán en qué dominio se engloba su tarea, diferenciando entre clasificación, regresión, agrupación (*clustering*) y aprendizaje profundo.
 - Para cada uno de estos dominios de tareas se ofrecerá un conjunto de algoritmos, sobre los que el usuario podrá escoger al especificar la configuración del entrenamiento.
- En caso de aquellos algoritmos que tengan hiperparámetros asociados, en la configuración de entrenamiento también se indicarán los valores a probar para cada uno de ellos, con el fin de determinar la combinación que ofrece mejores resultados para ese algoritmo.
- En la configuración mencionada también podrán especificarse restricciones referentes a la infraestructura a emplear durante el entrenamiento, tales como límites sobre el número de máquinas a utilizar de forma simultánea, sus prestaciones (tipo de instancias de cómputo) y la disponibilidad de hardware adicional, como GPUs.
- En base a los algoritmos indicados y los dominios de valores de hiperparámetros proporcionados, la plataforma determinará el algoritmo y la combinación de hiperparámetros que permiten construir el modelo que mejores resultados alcanza en su evaluación.
 - La métrica utilizada en la evaluación de modelos sobre problemas de clasificación multinomial será el Valor-F, que pondera mediante una media armónica tanto la precisión como la exhaustividad, permitiendo tener constancia de ambos factores en un único valor. [Chi92]
 - En el caso de la evaluación de modelos sobre problemas de clasificación binomial, la métrica utilizada será el coeficiente de correlación de Matthews (MCC, *Matthews Correlation Coefficient*) [Mat75], considerado como una mejor métrica en este escenario [CJ20].



- Al finalizar la tarea de entrenamiento, la plataforma almacenará los modelos construidos en el sistema de almacenamiento de archivos en la nube, ordenándolos de forma decreciente de acuerdo a los resultados ofrecidos de acuerdo a la métrica de evaluación. Cada modelo irá acompañado de una descripción del algoritmo y la combinación de hiperparámetros que han dado lugar a él, junto con los resultados alcanzados en su evaluación y otros datos relevantes que faciliten su inteligibilidad, comprensión y reproducibilidad.
 - Los modelos se encontrarán en un formato importable por el *framework* utilizado en el proceso de entrenamiento en función de la naturaleza de la tarea, contemplándose SparkML y TensorFlow, éste último en problemas de aprendizaje profundo.
- Igualmente, la finalización de la tarea de entrenamiento se notificará al usuario mediante el envío de un correo electrónico, indicándole la URL en la que puede encontrar los modelos construidos.

3.1.2 Requisitos no funcionales

- La plataforma ejecutará de forma paralela procesos de entrenamiento con diferentes algoritmos, de los especificados en la configuración de la tarea, mediante el aprovisionamiento de múltiples instancias de cómputo del proveedor de infraestructura en la nube.
- La plataforma ejecutará de forma paralela procesos de entrenamiento con diferentes combinaciones de los valores de hiperparámetros de un mismo algoritmo, mediante el aprovisionamiento de múltiples instancias de cómputo del proveedor de infraestructura en la nube.
 - Se empleará una estrategia de búsqueda inteligente sobre el dominio de valores de los hiperparámetros. La adopción de esta heurística perseguirá minimizar el número de ejecuciones de procesos de entrenamiento necesario para identificar la combinación de valores óptima, aquella cuyo correspondiente modelo maximiza el resultado alcanzado sobre la métrica de evaluación.
- La plataforma ejecutará cada proceso de entrenamiento, con un determinado algoritmo y una combinación específica de valores de sus hiperparámetros, de forma distribuida sobre múltiples instancias de cómputo.
 - Esta ejecución distribuida se llevará a cabo mediante versiones distribuidas de los algoritmos de aprendizaje automático, optimizadas para favorecer su escalabilidad y reducir el tiempo de entrenamiento.



- Se llevará a cabo un estudio que determine en qué casos merece la pena utilizar los servicios propios del proveedor *cloud* para la ejecución de procesos sobre su infraestructura (PaaS), aprovechando la abstracción que ofrece sobre los recursos aprovisionados. Como alternativa a estos servicios, se estudiará en qué casos es preferible realizar una gestión manual de los recursos operando directamente sobre las máquinas virtuales que constituyen las unidades de instancias computacionales, reduciendo costes frente a la utilización de los servicios de alto nivel ofrecidos por el proveedor.
 - Los puntos restantes se refieren principalmente a este segundo caso, en el que se implementan manualmente las políticas para la adquisición y la liberación de recursos computacionales en lugar de recurrir a servicios gestionados que ya proporcionan esta funcionalidad.
- La plataforma aprovisionará y liberará dinámicamente los recursos de cómputo necesarios para la ejecución distribuida de los procesos de entrenamiento.
- Se realizará una administración eficiente en la adquisición y utilización de los recursos de cómputo, ajustando elásticamente el grado de escalado horizontal en función de la carga computacional de los procesos en ejecución.
- A lo largo de la ejecución completa de una tarea se respetarán los límites establecidos en la configuración de entrenamiento respecto a las instancias computacionales a emplear en dicha tarea, tanto en número máximo como en prestaciones.

Capítulo 4

Diseño de alto nivel

En este capítulo se describe el diseño de Zygarde atendiendo a una descomposición modular basada en componentes, cada uno de los cuales se analiza empleando un enfoque descendente en lo que se refiere al nivel de abstracción. Se incide con particular nivel de detalle en el motor de ejecución de tareas, el módulo que opera como orquestador de la ejecución de los algoritmos de aprendizaje automático y actúa como núcleo de la lógica del sistema.

A lo largo de todo el capítulo se procede con especial énfasis en relación al modelo asíncrono de interacción entre los componentes, dirigido por eventos. El desacoplamiento que proporciona este modelo [Luc02] permite que el sistema se estructure como un *workflow* o secuencia de tareas que pueden constituirse como unidades de despliegue independientes, aplicando a su vez técnicas de escalado diferenciadas sobre cada una de ellas. La aplicación de este enfoque se amplía en el Capítulo 5, en el que se describe la arquitectura de despliegue del sistema en la nube, sobre la infraestructura de un proveedor de *cloud* público como Amazon Web Services.

Además del estudio de los componentes que conforman el sistema, en este capítulo se exponen las diferentes heurísticas que pueden seguirse para seleccionar las tareas a ejecutar con el fin de hallar la distribución de parámetros óptima, desde una exploración completa sobre un espacio de múltiples dimensiones a la búsqueda de mínimos en un función confeccionada mediante distribuciones de probabilidad. Tras ello, se presenta una síntesis de los algoritmos de aprendizaje automático que Zygarde pone a disposición de los usuarios, clasificados según la familia a la que pertenecen. Finalmente, se muestra el esquema que deben seguir las solicitudes dirigidas al sistema, en formato JSON y especificado de acuerdo al modelo JSON Schema¹.

¹JSON Schema, <https://json-schema.org>



4.1 Componentes del sistema

Siguiendo una aproximación estructurada mediante componentes de despliegue, el sistema se constituye como un conjunto de unidades que se comunican entre sí a través de la red. La interacción entre componentes se realiza de forma asíncrona cuando es posible, evitando el bloqueo de los interlocutores y propiciando así un desacoplamiento entre los componentes que favorece su escalabilidad potencial [Hil90]. La Figura 4.1 muestra los componentes que conforman el sistema, los cuales se detallan en las siguientes subsecciones. Mientras que este capítulo se centra en la lógica y en la funcionalidad de cada componente, esta distribución de elementos de despliegue se muestra de nuevo en la Sección 5.1, trasladándola al escenario de la arquitectura de despliegue en la nube, donde se describe en detalle cada flujo de interacción y cómo cada componente se materializa sobre la infraestructura y los servicios del proveedor *cloud*.

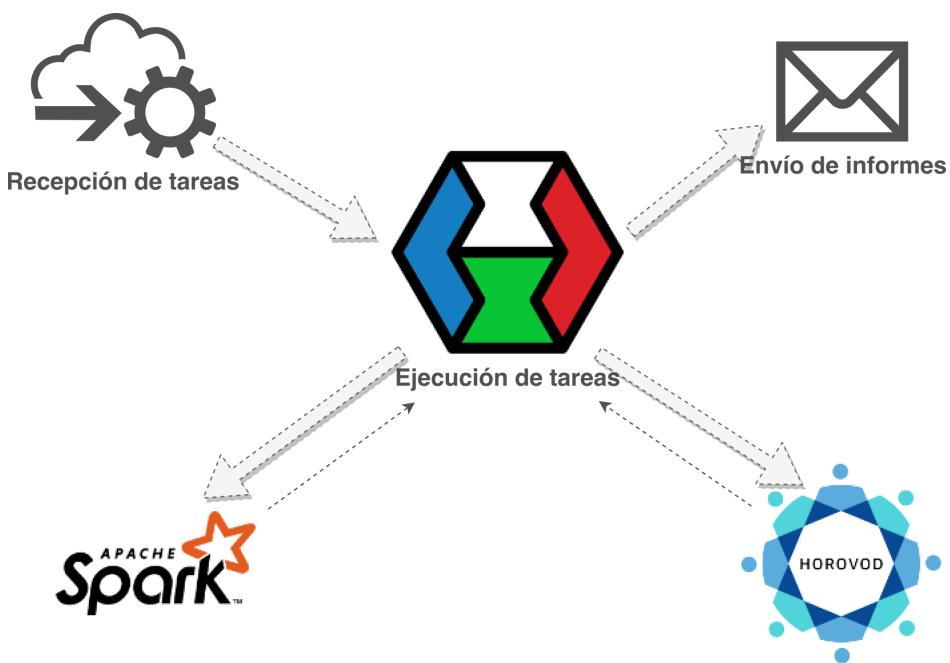


Figura 4.1: Componentes del sistema

4.1.1 Recepción de tareas

El componente de recepción de tareas actúa como un centralizador para la recepción de peticiones de ejecución, referentes a tareas de entrenamiento y creación de modelos mediante algoritmos de aprendizaje automático, por parte de los usuarios. Estas peticiones pueden provenir de varias fuentes, tal como detalla la especificación de requisitos funcionales en la Subsección 3.1.1, en concreto: a través de un *endpoint* de una API REST, como un mensaje dirigido a una cola de mensajería direccional y mediante el depósito de la petición en el sistema de almacenamiento de archivos del proveedor *cloud*.

El componente se encarga de gestionar la recepción de estas peticiones a través de cualquiera de estas tres fuentes, comprobar que la petición es sintácticamente válida de acuerdo a la especificación de su formato, realizar cualquier transformación que pueda resultar necesaria y poner la petición ya validada y procesada a disposición del motor de ejecución de tareas.

La forma en la que la petición se dirige al componente de ejecución de tareas es mediante su publicación en una cola escalable de mensajería. Este modelo de comunicación [EFGK03] permite desacoplar la recepción de peticiones de su procesamiento por parte del motor de ejecución de tareas, lo que supone evitar los potenciales bloqueos entre componentes que devendrían de un modelo de interacción síncrono. Además, en el contexto de un sistema en la nube, esta aproximación posibilita escalar horizontalmente los elementos de despliegue de forma selectiva, aliviando cuellos de botella y maximizando la productividad en términos de la tasa de procesamiento de peticiones por unidad de tiempo.

El componente de recepción de tareas se ha implementado según el paradigma *serverless*, mediante una serie de funciones Lambda², las cuales se analizan individualmente en la Subsección 5.1.1. Tal como se comenta en la Sección 1.1, la adopción del paradigma *serverless* abre la vía a transicionar a un modelo de computación dirigida por eventos. En el contexto de una aplicación distribuida y escalable en la nube, este modelo permite ofrecer garantías de alta disponibilidad a la par que minimiza el coste y la complejidad del despliegue (siempre, por supuesto, que todo ello venga acompañado de una adecuada implementación).

4.1.2 Ejecución de tareas

El motor de ejecución de tareas es el componente más complejo de implementación propia del sistema y actúa como orquestador de la ejecución de las peticiones de entrenamiento recibidas. Específicamente, las responsabilidades que abarca son las siguientes: la transformación de las peticiones de los usuarios en ejecuciones específicas de algoritmos de aprendizaje automático, la monitorización de dichas ejecuciones sobre el clúster de cómputo intensivo correspondiente, la recolección de los resultados y los modelos generados, su ordenación mediante la elaboración de una clasificación basada en una métrica de su calidad y la elaboración del informe que será remitido al usuario mediante el componente de envío de informes.

En lugar de implementar el componente como un servidor al que el componente de recepción de tareas envía directamente las peticiones de entrenamiento, la inclusión de una cola de mensajería como medio de comunicación entre ambos permite escalar horizontalmente este componente sin necesidad de balanceadores de carga que repartan las peticiones entre las réplicas. En su lugar, son las propias réplicas del componente las que extraen las peticiones

²AWS Lambda - Serverless Compute - Amazon Web Services, <https://aws.amazon.com/lambda/>

de la cola cuando se hallan en disposición de procesar nuevas solicitudes, de acuerdo a un modelo productor-consumidor. Así, se evita que una réplica del componente reciba un volumen de peticiones mayor que el que puede procesar o que exista un desequilibrio entre el volumen de carga asignado a cada réplica, puesto que son las propias réplicas las que solicitan la adjudicación de nuevas tareas. Para ello, la cola de mensajería utilizada ha de proporcionar garantías de exclusión mutua y ventanas temporizadas de visibilidad de los mensajes, que eviten efectos indeseados derivados del acceso concurrente a la cola por parte de varias réplicas del componente.

El componente se plantea como un consumidor de peticiones multi-hilo, capaz de procesar múltiples tareas en paralelo. Tal como se verá más delante, opera con un elevado grado de concurrencia gestionada jerárquicamente en un árbol de tres niveles (más el hilo principal de ejecución), lo que obedece a una motivación y responde a una justificación. La motivación es, mediante la coordinación de la ejecución concurrente de múltiples configuraciones paramétricas de los algoritmos de aprendizaje automático, reducir el tiempo necesario para completar el proceso de entrenamiento y determinar cuál es la configuración cuyo modelo asociado ofrece los mejores resultados. La justificación es que, mientras que la labor de cómputo intensivo se lleva a cabo en los clústeres aprovisionados con tal fin, la labor de coordinación que realiza el componente implica un consumo mucho más asequible de recursos computacionales. Esto facilita el procesamiento simultáneo de múltiples peticiones en una única unidad de despliegue, así como la coordinación de las ejecuciones concurrentes de algoritmos de aprendizaje automático correspondientes a cada petición. No obstante, y tal como se trata detalladamente en la Subsección 5.1.2, el despliegue del componente se emplaza de forma replicada en un conjunto de instancias de cómputo que se reparten las peticiones a procesar mediante su extracción de la cola de mensajería, evitando así que el componente constituya un punto único de fallo o que una sola réplica haya de asumir todo el trabajo y se convierta en un potencial cuello de botella. Junto con la explicación del modelo de despliegue también se argumentan las razones que llevan a que este componente se haya implementado como un proceso de larga duración en lugar de como una función Lambda, rompiendo la hegemonía acorde con el paradigma *serverless* que se ha adoptado en los restantes componentes de implementación propia.

En las fases iniciales del análisis se planteó la posibilidad de realizar la implementación del componente en Python, dado que es el lenguaje en el que pueden encontrarse con mayor facilidad implementaciones de algoritmos de aprendizaje automático, gracias a la adopción de bibliotecas como NumPy (que, a su vez, emplea bibliotecas algebraicas como LAPACK³ y *kernels* de cálculo numérico como BLAS⁴, implementados y optimizados en C y Fortran, con el fin de maximizar su eficiencia). Esta opción fue descartada puesto que el intérprete de

³Linear Algegra PACKage, <http://www.netlib.org/lapack/>

⁴BLAS (Basic Linear Algebra Subprograms), <http://www.netlib.orgblas/>

Python inhibe la ejecución concurrente de varios hilos, debido al GIL⁵ (*Global Interpreter Lock*), un semáforo que mantiene la exclusión mutua en el acceso a los objetos del intérprete. Aunque el lenguaje permite lanzar varios hilos de ejecución, sólo uno se encuentra activo en un determinado momento. Esto impide aprovechar de esta forma la mejora del rendimiento que pudiera provenir del uso de arquitecturas hardware con múltiples procesadores. En el caso de NumPy⁶, las rutinas matemáticas que incluye se encuentran implementadas en lenguajes de menor nivel y precompiladas, por lo que su ejecución no se lleva a cabo en el contexto del intérprete de Python, circunvalando esta desventaja.

En el escenario del motor de ejecución de tareas de Zygarde, con el objetivo de dotar de forma sencilla al componente de un paralelismo real que puede gestionarse de forma sencilla a un nivel avanzado, la implementación se ha realizado en Java, aprovechando las APIs introducidas en Java 7 que proporcionan esta finalidad⁷. Además, tal como se expone en la Sección 4.3, en la mayor parte de los casos Zygarde utiliza las implementaciones de algoritmos de aprendizaje automático que proporciona Spark MLlib. Puesto que Spark está implementado en Java, la utilización de este lenguaje para la implementación del componente facilita la interacción directa con el clúster sin necesidad de utilizar *wrappers* como PySpark, además de la conveniencia ya expuesta en relación a la gestión de la concurrencia.

El comportamiento que implementa y exhibe el componente, ordenado secuencialmente, se describe a continuación:

1. Se extrae de la cola un mensaje pendiente de procesamiento, el cual contiene una petición de entrenamiento en formato JSON. Se crea un nuevo hilo de ejecución, responsable de procesar la petición, pasándole el mensaje como parámetro en su inicialización (primer nivel de gestión concurrente). El hilo principal de ejecución sigue esperando la llegada de mensajes a la cola, sin bloquearse durante el procesamiento de la petición recibida.
2. El documento JSON se mapea a una estructura de objetos. La petición de entrenamiento, entre otros datos, habrá especificado uno o varios algoritmos de aprendizaje automático, cada uno asociado a una serie de hiperparámetros, proporcionando una serie de valores a probar para cada parámetro. Desde el hilo de ejecución actual y para cada algoritmo especificado en la petición se crea un nuevo hilo de ejecución, que se inicializa con el algoritmo y toda la distribución de hiperparámetros propuesta para el mismo (segundo nivel de gestión concurrente). El hilo actual, responsable del procesamiento de la petición, se bloquea hasta que todos sus descendientes (los hilos responsables de cada algoritmo) se hayan detenido.

⁵GlobalInterpreterLock - Python Wiki, <https://wiki.python.org/moin/GlobalInterpreterLock>

⁶Parallel Programming with NumPy and SciPy, <https://www.scipy.org/topical-software.html>

⁷Java SE 7 Concurrency-Related APIs & Developer Guides,

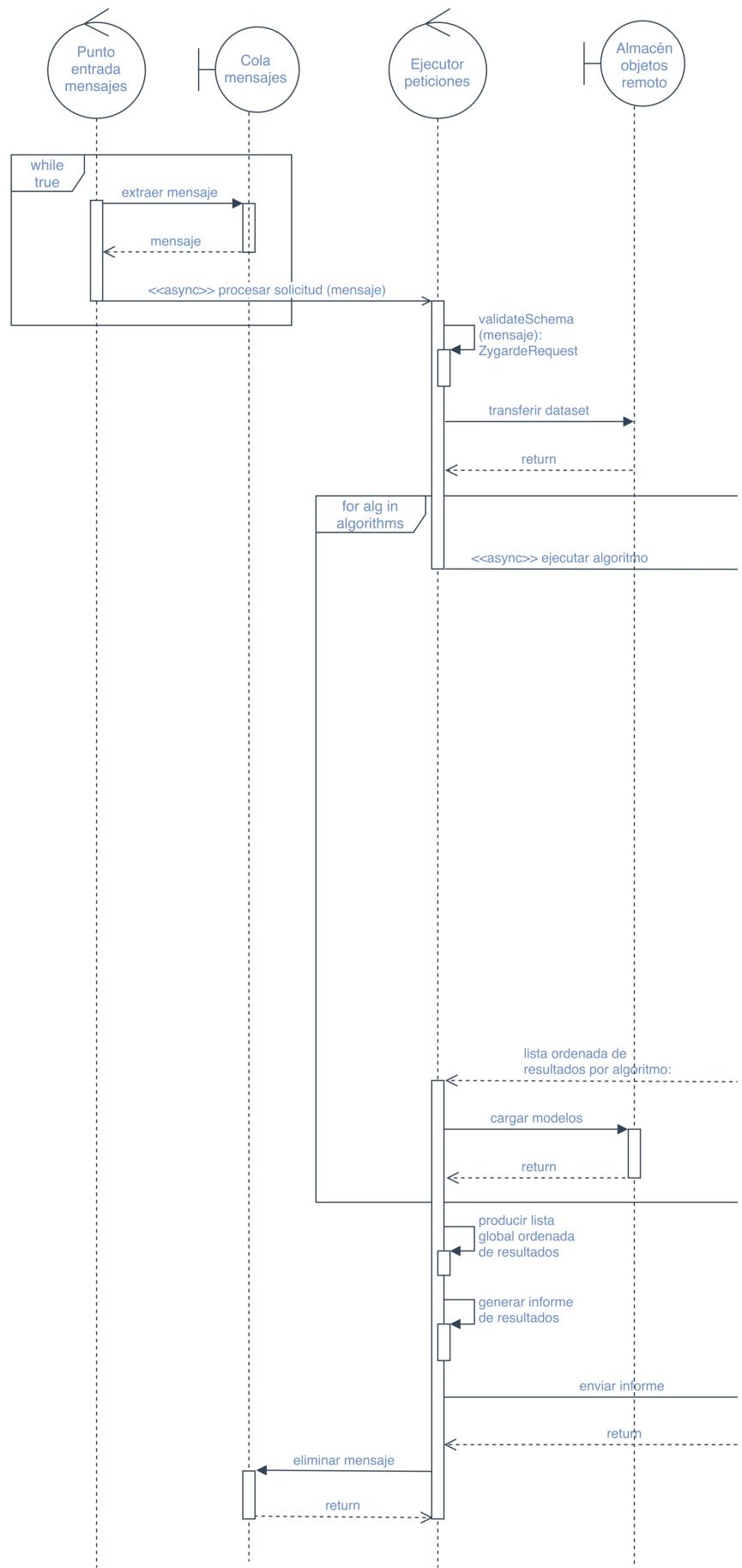
<https://docs.oracle.com/javase/7/docs/technotes/guides/concurrency/index.html>



3. El hilo responsable de un determinado algoritmo, en el contexto de la petición en la que se engloba, se basa en los hiperparámetros proporcionados y en los valores propuestos sobre cada uno para elaborar el conjunto de tareas de entrenamiento a ejecutar para el algoritmo en cuestión. La especificación de cada tarea consiste en un identificador único en formato UUID4, el nombre canónico del algoritmo, los parámetros a asignar en su construcción junto con sus valores (una combinación de los valores propuestos para cada uno), y la ubicación del *dataset*. La estrategia utilizada para seleccionar las combinaciones de valores de hiperparámetros que se ejecutarán se especifica también en la petición de entrenamiento y puede basarse en tres heurísticas, que se estudian detenidamente en la Sección 4.2: fuerza bruta, búsqueda aleatoria y optimización bayesiana.
4. Desde el hilo de ejecución actual se lanza un nuevo hilo de ejecución para cada tarea de entrenamiento (tercer nivel de gestión concurrente), entendida tal como se ha establecido en el punto anterior. El hilo de ejecución actual se bloquea hasta que todos sus descendientes se hayan detenido.
5. El hilo correspondiente a cada tarea de entrenamiento se conecta al clúster de cómputo intensivo apropiado para el algoritmo y se ordena la ejecución. Para favorecer la interoperabilidad con el *framework* en el que se ejecuta la propia tarea se ha especificado una interfaz de alto nivel agnóstica al algoritmo. Se proporcionan implementaciones de esta interfaz tanto para Spark MLlib como para TensorFlow, seleccionando dinámicamente la que corresponda en función del dominio del algoritmo a ejecutar, abstrayendo al nivel superior de las particularidades de la tecnología subyacente. Bajo esta interfaz, para cada algoritmo específico se ha implementado una clase que invoca directamente su ejecución, encargándose de la inicialización del objeto del algoritmo con los valores correspondientes asociados a sus parámetros, la recuperación del *dataset*, su particionado, la persistencia del modelo generado y la evaluación del modelo. Todas las clases que actúan como *wrappers* de los algoritmos ofrecen una interfaz de operaciones común, proporcionando un nivel de abstracción que permite a los niveles superiores operar con homogeneidad independientemente del algoritmo específico a ejecutar, su dominio y el *framework* sobre el que la ejecución tiene lugar.
6. El hilo permanece bloqueado hasta que finaliza la ejecución del algoritmo. El *wrapper* almacena el modelo generado en el sistema de archivos de la máquina, utilizando el identificador de la tarea (local a la ejecución del algoritmo con esa determinada combinación de valores de los hiperparámetros) y devuelve la medida de calidad obtenida en la evaluación. De vuelta en el nivel superior de ejecución, se conforma un objeto que incluye el identificador de la tarea específica, los datos de la ejecución (algoritmo y combinación de valores de hiperparámetros) y la medida de calidad obtenida. El hilo de ejecución finaliza devolviendo este objeto como resultado a su antecesor.

7. De nuevo en el segundo nivel de gestión concurrente, y una vez que todos sus descendientes han finalizado su ejecución, el hilo que se encontraba detenido retoma su actividad, recopilando los resultados devueltos por los hilos que había creado y ordenándolos de forma descendente de acuerdo con su calidad. Muestra un mensaje informativo de las ejecuciones que se han realizado y de cuál es la combinación de valores de los hiperparámetros que ha alcanzado un mejor resultado para este algoritmo (útil a efectos de depuración). Finalmente, se devuelve la lista ordenada de resultados al nivel superior y se concluye la ejecución de este hilo.
8. Nuevamente en el primer nivel de gestión concurrente, y una vez que todos sus descendientes han finalizado su ejecución, se obtienen las listas de resultados devueltos por los sucesores del hilo encargado de gestionar la petición del usuario, que contienen los resultados obtenidos por cada ejecución realizada en el contexto de cada algoritmo propuesto. Todas estas listas se mezclan en una sola que se ordena de forma descendente de acuerdo a la calidad de los modelos. Se muestra un mensaje indicando cuál ha sido la combinación de algoritmo y valores de hiperparámetros que ha logrado un mejor resultado en base a su métrica de evaluación y cuál ha sido el valor que ha alcanzado.
9. Todos los modelos generados, que habían sido almacenados en el sistema de archivos, se cargan en una ubicación accesible por el usuario (en este caso, el sistema de archivos en la nube del proveedor *cloud*). Se compone un informe con los datos de la lista de ejecuciones, ordenada cualitativamente, donde para cada elemento se hace constar el algoritmo utilizado, los valores de los hiperparámetros empleados, el resultado obtenido y la ubicación remota en la que el usuario puede localizar el modelo para su descarga. En este informe se muestran en un lugar preeminente los datos de la ejecución que ha logrado el mejor resultado. Dicho informe se pone a disposición del componente de envío de informes, mediante el que será remitido al usuario. Finalmente, el mensaje correspondiente a la petición se marca como procesado con éxito en la cola de mensajería (evitando que eventualmente volviera a encontrarse disponible para su admisión) y la ejecución del hilo actual finaliza, dando por concluido el procesamiento de la petición de entrenamiento emitida por el usuario.

La anterior enumeración ofrece una descripción textual del comportamiento del componente, complementándose con los dos siguientes diagramas que representan sendas perspectivas gráficas del sistema. En primer lugar, el diagrama de secuencia de la Figura 4.2 permite visualizar detalladamente las interacciones entre el componente y los clústeres de cómputo intensivo, el flujo de ejecución del propio componente y la creación y terminación de hilos de ejecución concurrente. En segundo lugar, el diagrama de flujo de datos de la Figura 4.3 muestra la transmisión ordenada de información entre los componentes, proporcionando un mayor nivel de abstracción que el diagrama de secuencia.



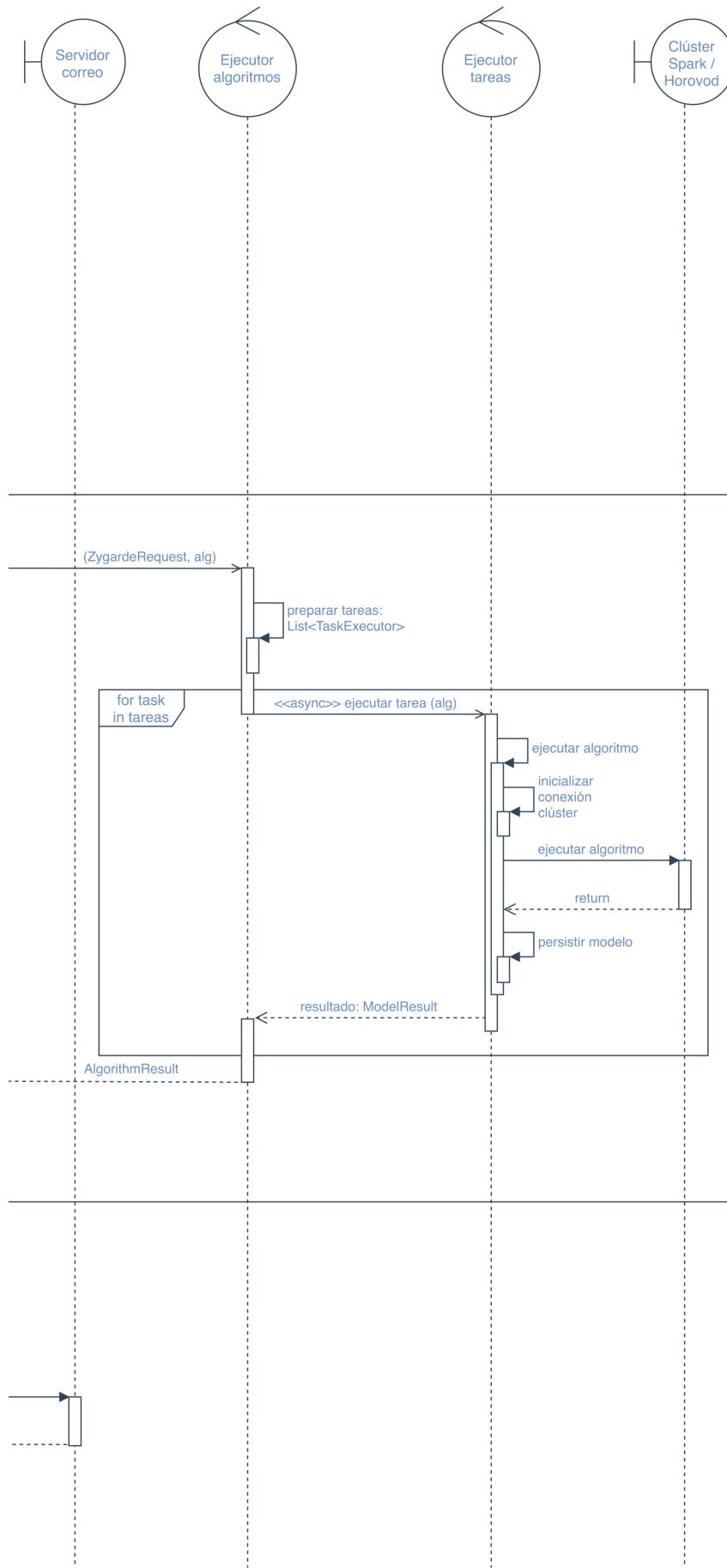


Figura 4.2: Diagrama de secuencia

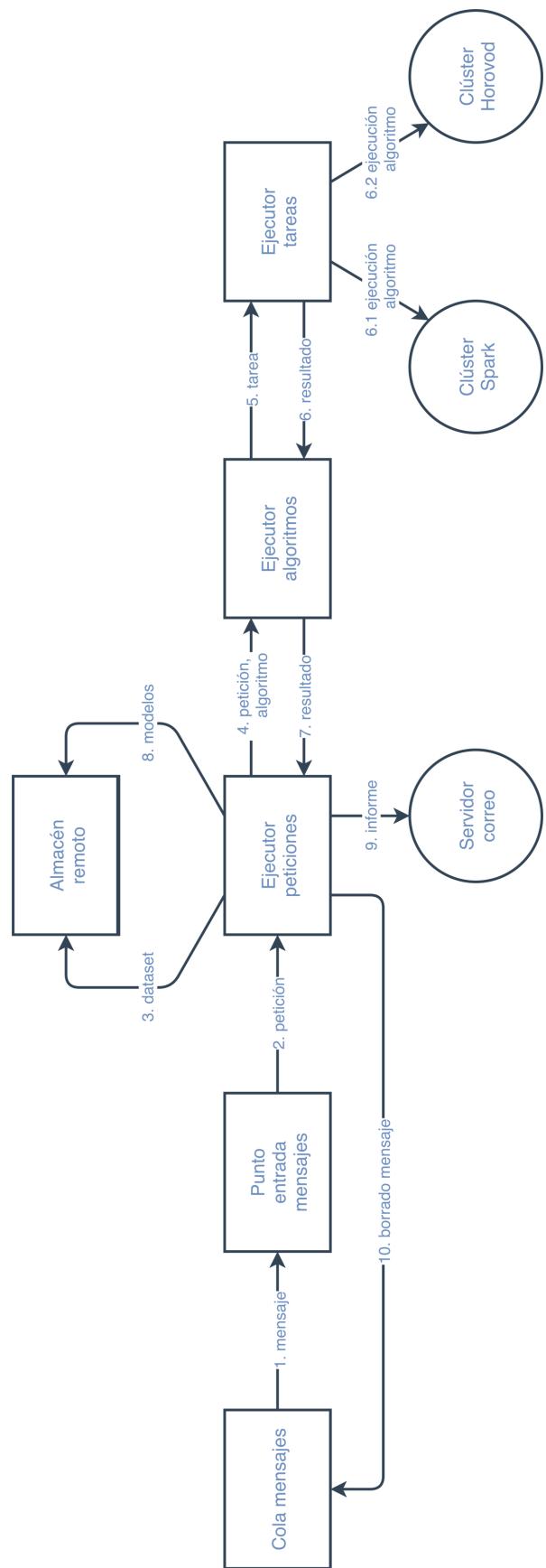


Figura 4.3: Diagrama de flujo de datos

4.1.3 Envío de informes

El componente de envío de informes se encarga de remitir a los usuarios el resultado de la ejecución de su petición de entrenamiento, una vez que el procesamiento de dicha petición ha concluido. En el momento en el que el componente de ejecución de tareas ha creado el informe en el que se detallan los resultados de la ejecución, tal como se describe en la Subsección 4.1.2, se pone este informe a disposición del componente de envío de informes, junto con la dirección de correo electrónico que el usuario indicó en la petición de entrenamiento. El componente de envío de informes compone un mensaje de correo electrónico con el envío del informe y lo envía a la dirección de correo del usuario.

El propósito de separar este módulo del componente de ejecución de tareas es doble. Por una parte, se pretende actuar de acuerdo con el principio de separación de responsabilidades, evitando que el componente de ejecución de tareas, que ya de por sí es el módulo más complejo del sistema, asuma funcionalidades que no le corresponden intrínsecamente y pueden extraerse y encapsularse en otros componentes. Por otra parte, se abstrae al componente de ejecución de tareas de cómo se realiza el envío de los informes a los usuarios. En concreto, en una instalación *on-premises* lo más habitual sería contar con un servidor SMTP [Kle08], como aiosmtpd⁸, a través del que se realizará el envío de los correos. En este caso, al plantear un sistema desplegado sobre la infraestructura de un proveedor de *cloud* público, se ha optado por utilizar el servicio de envío de correos que ofrece dicho proveedor.

Al igual que el componente de recepción de tareas, este componente se ha implementado según el paradigma *serverless* mediante una función Lambda que utiliza el SDK del servicio de envío de correos del proveedor *cloud*. Los detalles de esta aproximación, la forma en la que el componente recibe los informes emitidos por el motor de ejecución de tareas y la decisión de emplear el servicio de envío de correo del proveedor *cloud* en lugar de un servidor SMTP dedicado se tratan en la Subsección 5.1.3.

4.1.4 Clúster Spark

Cuando desde el motor de ejecución de tareas se lanza un hilo encargado de comandar la ejecución de un algoritmo de aprendizaje automático, éste delega la ejecución y monitorización de la tarea en un clúster de cómputo intensivo. Tal como se ha descrito en la Subsección 4.1.2, el hilo se conecta al clúster a través de una estructura de interfaces de operaciones, dispuesta en varios niveles, que le proporcionan una abstracción sobre la infraestructura y el *framework* específico sobre los que se ejecuta el algoritmo.

En función del dominio en el que se engloba el algoritmo, su ejecución se delega en un clúster de una determinada tecnología. En el caso de aquellos algoritmos en los que se adoptan las implementaciones de algoritmos de aprendizaje automático distribuidos ofrecidas por

⁸aiosmtpd - An asyncio based SMTP server, <https://aiosmtpd.readthedocs.io/en/latest/README.html>

Spark MLlib, la ejecución se realiza sobre un clúster de Spark. Un clúster de esta tecnología se despliega y configura de forma automatizada sobre la infraestructura del proveedor *cloud* para atender la petición del usuario, teniendo en cuenta los datos especificados en el documento de la petición respecto a las características de la infraestructura dinámicamente aprovisionada, tal como se trata en la Sección 4.4.

Una vez que el clúster se ha iniciado con éxito y se encuentra operativo y listo para admitir carga de trabajo, el motor de ejecución de tareas podrá conectarse al nodo maestro y solicitar la ejecución de las tareas que devengan de la petición del usuario. El procedimiento de despliegue y configuración automatizados del clúster Spark y los detalles de cómo tiene lugar la conexión y comunicación entre el motor de ejecución de tareas y el clúster se tratan pormenorizadamente en el Capítulo 5, concretamente en la Sección 5.2.

4.1.5 Clúster Horovod

A la hora de ejecutar un algoritmo de aprendizaje automático, el motor de ejecución de tareas deja esta labor en manos de clúster de cómputo intensivo, siguiendo los mecanismos expuestos en la Subsección 4.1.4. En función del dominio en el que se engloba el algoritmo, su ejecución se delega en un clúster de una determinada tecnología, proporcionando una abstracción sobre el *framework* específico y la infraestructura subyacente mediante una serie de interfaces de operaciones estructurada en varios niveles, tal como se describe en la Subsección 4.1.2.

En el caso del dominio de redes neuronales y aprendizaje profundo, los algoritmos ofrecidos adoptan las implementaciones ofrecidas por TensorFlow, utilizando Keras⁹ y delegando su ejecución en un clúster sobre el que se instala Horovod, como implementación eficiente de TensorFlow distribuido.

Al igual que ocurre con Spark, en este escenario un clúster con Horovod se despliega y configura de forma automatizada sobre la infraestructura del proveedor *cloud*, teniendo en cuenta los datos especificados en la petición del usuario respecto a las características de la infraestructura dinámicamente aprovisionada. El procedimiento automatizado de despliegue y la configuración del clúster Horovod, así como las particularidades de la conexión con el clúster y la interacción entre éste último y el modelo de ejecución de tareas, se tratan en detalle en el Capítulo 5, concretamente en la Sección 5.3.

⁹Keras: the Python deep learning API como interfaz de alto nivel, <https://keras.io>

4.2 Ajuste automatizado de hiperparámetros

Una de las principales propuestas de Zygard es no limitarse a evaluar el funcionamiento de diferentes algoritmos sobre un *dataset*, sino también poder probar diferentes valores para los hiperparámetros de cada algoritmo. Los hiperparámetros son características de los algoritmos de aprendizaje automático que se configuran con anterioridad a su ejecución. Es decir, no se aprenden durante el entrenamiento del modelo, sino que se establecen previamente. Un ejemplo se puede encontrar en el contexto de las redes neuronales artificiales, donde la estructura de la red y la función de activación se fijan de antemano, mientras que los pesos que cada perceptrón otorga a cada una de sus entradas se determina como consecuencia del proceso de entrenamiento. Otro ejemplo se presenta en los algoritmos de agrupamiento en los que se indica el número de clústeres antes de que el entrenamiento tenga lugar.

Con frecuencia, el ajuste de estos hiperparámetros dependientes del algoritmo específico es un factor crítico en los resultados que un algoritmo es capaz de alcanzar sobre un determinado conjunto de datos; de ahí la importancia de detectar cuál es la configuración de los hiperparámetros que maximiza la calidad de los resultados obtenidos. Normalmente la selección de los valores adecuados para los hiperparámetros es una tarea de prueba y error, cuyos resultados son muy dependientes de los propios datos y que suele requerir un juicio experto o conocimiento específico del funcionamiento del algoritmo para acotar los valores que merece la pena probar.

Este procedimiento, determinar cuáles son los valores de los hiperparámetros que ofrecen mejores resultados para un determinado algoritmo y conjunto de datos, se denomina *optimización de hiperparámetros*. Formalmente, la optimización de hiperparámetros puede representarse de la siguiente forma:

$$x^* = \arg \min_{x \in X} f(x)$$

En esta ecuación, $f(x)$ representa el valor objetivo a minimizar en la evaluación sobre el conjunto de datos de validación, que es una tasa de error (como, por ejemplo, el error cuadrático medio). En caso de utilizar como métrica una medida que no vaya expresada en términos del error sino de la calidad o acierto del modelo, puede operarse con $-f(x)$, lo que equivaldría a identificar los máximos de la función $f(x)$. Por su parte, x^* es la combinación de valores de los hiperparámetros que minimiza el valor de la función, en la que x puede tomar cualquier valor en el dominio X , que corresponde a los valores admitidos para dichos hiperparámetros. Aunque existen métricas que tienen en cuenta el tiempo necesario para el entrenamiento del modelo, con el fin de hallar un equilibrio entre la duración del entrenamiento y la calidad del modelo obtenido, en esta sección el proceso de optimización descrito se aplica exclusivamente en términos de los resultados alcanzados sobre el conjunto de datos de validación.



La automatización de esta labor puede ahorrar un gran volumen de tiempo y esfuerzo a los científicos de datos, máxime cuando la transición a la computación distribuida en la nube permite entrenar múltiples modelos simultáneamente, con diferentes algoritmos y configuraciones de hiperparámetros. Aunque siempre es deseable emplear un criterio de juicio experto para restringir los valores a probar, la automatización y la paralelización del procedimiento permiten que usuarios sin un conocimiento avanzado de los parámetros de configuración de cada algoritmo también puedan beneficiarse de esta técnica y así optimizar sus modelos aprendizaje automático.

No obstante, la optimización de hiperparámetros sufre de un problema que persiste incluso tras su automatización: el elevado coste de evaluar la función objetivo, es decir, calcular el valor de $f(x)$ para cada configuración x de hiperparámetros. Cada evaluación de la función objetivo equivale a completar un ciclo de entrenamiento con el que generar un modelo, realizar una serie de predicción y compararlas con los datos de evaluación para calcular una métrica del error la calidad del modelo. Este problema se magnifica al tener en cuenta que X corresponde a un espacio multidimensional donde el número de dimensiones equivale al número de hiperparámetros y donde el dominio de cada una de ellas corresponde al conjunto de los valores posibles para ese hiperparámetro. Por este motivo, conviene contar con técnicas que, mediante una exploración inteligente del espacio de búsqueda, sean capaces de identificar el punto x^* del espacio que minimiza el valor de la función objetivo, o al menos de aproximarse a él, reduciendo el número de evaluaciones que se realizan sobre la función objetivo.

Atendiendo a todos estos factores, la optimización de hiperparámetros de Zygarde implementa tres métodos de exploración del espacio de búsqueda, seleccionables por parte del usuario en el documento de petición, tal como figura en la Sección 4.4. En las siguientes subsecciones se describen estos tres métodos, ordenados desde el más rudimentario al más sofisticado, detallando los pros y contras de cada uno así como los escenarios en los que es preferible emplear uno u otro.

4.2.1 Exploración en malla (*grid*)

La exploración en malla o en *grid* consiste en calcular todas las posibles combinaciones de valores de los hiperparámetros, es decir, determinar todos los puntos del espacio multidimensional definido por los diferentes dominios y ejecutar un proceso de entrenamiento y evaluación para cada uno de ellos, comparando los resultados obtenidos. En este contexto, Zygarde sólo admite la definición de valores discretos en el espacio de los hiperparámetros de la petición de entrenamiento, lo que se justifica en la Sección 4.4.

Esta aproximación tiene dos ventajas principales. En primer lugar, al tratarse de un método de búsqueda exhaustiva existe la certeza de que se hallará la combinación de hiperparámetros óptima en el contexto de los valores definidos. En segundo lugar, no existe una relación de precedencia entre las ejecuciones, lo que permite una ejecución simultánea en términos de lo que se conoce como un problema “embarazosamente paralelo”. Esto último permite reducir el tiempo necesario para explorar el estado de búsqueda y determinar cuál es la combinación de valores de los hiperparámetros que minimiza el valor de la función objetivo.

Por otra parte, este método ostenta una gran desventaja. Justamente al tratarse de un método de búsqueda exhaustiva, realiza un proceso de entrenamiento y evaluación para cada posible punto del espacio, lo que puede resultar en un número ingente de evaluaciones de la función objetivo (ejecuciones de algoritmos de aprendizaje automático), con el consumo de recursos computacionales que ello conlleva. Aunque Zygard hace uso de la infraestructura de un proveedor de *cloud* público como Amazon Web Services, en el que los recursos son aparentemente ilimitados, no hay que olvidar el coste económico asociado a la utilización de estos recursos. Esto es especialmente significativo cuando Zygard, dentro de los límites que se hayan establecido en la petición de entrenamiento y en la configuración de la propia plataforma, procede a desplegar un clúster de cómputo intensivo con un gran número de nodos para soportar la ejecución simultánea de múltiples tareas, como puede ocurrir cuando se emplea este método de búsqueda.

Por los motivos expuestos, la utilización de este método de búsqueda para la exploración del espacio de los hiperparámetros y su optimización sólo se recomienda cuando el número de hiperparámetros es reducido y/o lo es el número de valores especificados para cada uno de ellos. Esto suele corresponder a una situación en la que, mediante juicio experto o un proceso de entrenamiento previo, se ha determinado cuáles son los valores de cada hiperparámetro que ofrecen mejores resultados, acotándolos a unas pocas opciones, y se desea realizar una búsqueda exhaustiva sobre estos valores específicos.

4.2.2 Exploración aleatoria

La exploración aleatoria consiste en escoger un número acotado de puntos del espacio multidimensional definido por los dominios de los hiperparámetros y realizar una evaluación de la función objetivo para cada uno de ellos. Su propósito es evitar el ingente número de ejecuciones que tienen lugar en la búsqueda en malla, limitándolo a una cantidad razonable y factible de ejecuciones. En la práctica, Zygard implementa este método invocando a la función del procedimiento de búsqueda en malla que construye todas las posibles combinaciones de valores de los hiperparámetros, y seguidamente seleccionando aleatoriamente un número acotado de elementos del conjunto obtenido.

Este método mantiene la ventaja de la búsqueda en malla respecto a la ausencia de una precedencia entre las ejecuciones, lo que permite múltiples evaluaciones simultáneas de la función objetivo. Por otra parte, a cambio de eliminar la desventaja derivada del elevado número de ejecuciones que podía tener lugar en la búsqueda en malla, pierde la exhaustividad en cuanto a la exploración del espacio. Esto provoca que los mínimos de la función objetivo (o puntos cercanos a ellos) puedan obviarse al no llegar a evaluarse, con lo que la búsqueda devolvería un resultado sub-óptimo en el contexto del espacio de valores proporcionado por el usuario para cada uno de los hiperparámetros. Sin embargo, la selección aleatoria de un número suficiente de muestras conforme a una distribución uniforme permite obtener mejores resultados de lo que cabría esperar en un primer momento, llegando a alcanzar resultados equiparables a los de la búsqueda en malla, pero con mayor eficiencia en el uso de los recursos computacionales [BB12]. Esto es particularmente cierto cuando se opera con variables numéricas o intervalos de variables continuas (si se permite), en cuyo caso este método posibilita explorar un espacio más extenso, puesto que operar de forma selectiva permite ampliar los límites establecidos sobre cada dimensión y abarcar un mayor espectro, mientras que el consumo de recursos computacionales sigue dependiendo directamente del número de muestras a seleccionar.

Incluso teniendo en cuenta que, tal como se ha comentado, este método puede devolver resultados sub-óptimos al no realizar una exploración exhaustiva del espacio de búsqueda, es especialmente útil para identificar regiones de este espacio en la que se alcanzan resultados prometedores y pueden delatar la presencia de un mínimo local o global en sus cercanías. Así, una de sus principales aplicaciones es realizar un primer reconocimiento del espacio definido por el dominio de los hiperparámetros cuando se carece de información previa o conocimiento específico del problema, identificando y acotando regiones en las que posteriormente realizar una búsqueda acotada más minuciosa.

El método de búsqueda aleatoria, como ya se ha expuesto, ofrece un buen equilibrio entre la magnitud del espacio de búsqueda explorado, los recursos computacionales empleados y, siempre que se seleccione un número suficiente de muestras, los resultados obtenidos. A esto se une su simplicidad en la selección de los puntos del espacio a evaluar y la asequibilidad de su implementación. Su mayor inconveniente proviene del hecho de que se trata de un método de búsqueda no informada, que no utiliza los resultados de anteriores evaluaciones de la función objetivo para determinar qué regiones del espacio puede resultar más provechoso explorar a continuación. Esto provoca que, al igual que la búsqueda en malla, pueda invertir recursos en explorar regiones en las que no existe un mínimo de la función objetivo ni señales de que pudiera haberlo. Este enfoque “ciego” supone un gasto potencialmente inútil de recursos computacionales que podrían invertirse en la exploración de regiones más prometedoras del espacio definido.

4.2.3 Optimización bayesiana

Con el objetivo de paliar las desventajas identificadas en los dos métodos anteriores, y que se describen brevemente en la sección anterior, Zygarde implementa un tercer método para el ajuste de los hiperparámetros, basado en optimización bayesiana. De forma resumida, la optimización bayesiana de hiperparámetros consiste en construir un modelo de probabilidad de la función objetivo y utilizarlo para seleccionar la configuración más prometedora de valores de los hiperparámetros, la cual será sometida a evaluación sobre la función objetivo real. [DMC16] [SSW⁺¹⁵]

Los enfoques basados en probabilidad bayesiana, a diferencia de las búsquedas en malla y aleatoria, mantienen un historial de los resultados de las evaluaciones realizadas con anterioridad sobre la función objetivo. Este histórico se utiliza para construir un modelo probabilístico que establece una correspondencia entre los valores de entrada (los hiperparámetros) y la probabilidad de una puntuación sobre la función objetivo (en base a la métrica adoptada, generalmente una tasa de error que se busca minimizar):

$$P(\text{puntuación} | \text{hiperparámetros})$$

A este modelo probabilístico $p(y|x)$ se le denomina “sustituto” (*surrogate*) de la función objetivo. Su utilización se debe a que es mucho más fácil optimizar el sustituto que la función objetivo real, debido al elevado coste que tiene la ejecución de evaluaciones de la función objetivo. Por este motivo, se pretende construir un modelo probabilístico que, a lo largo de las sucesivas iteraciones, se llegue a asemejar a la función objetivo. El flujo de ejecución que sigue este método en líneas generales es el siguiente:

1. Se construye un modelo de probabilidad que actuará como sustituto de la función objetivo.
2. Se identifica la configuración de hiperparámetros que obtiene un mejor resultado sobre el modelo sustituto.
3. Se realiza una evaluación de esta configuración de hiperparámetros sobre la función objetivo.
4. Se actualiza el modelo sustituto con los nuevos resultados.
5. Se repite la secuencia de pasos 2-4 hasta que se cumpla un criterio de parada, normalmente determinado por un número máximo de iteraciones o la expiración de un temporizador.

Cada evaluación de la función objetivo permite obtener algo más de información sobre ella. Así, la actualización del modelo sustituto en cada iteración en base al histórico completo de resultados busca ajustar el sustituto para que progresivamente llegue a describir un comportamiento semejante al de la función objetivo real. [FSK08]

Los métodos de búsqueda presentados en las subsecciones anteriores determinan las configuraciones de hiperparámetros a evaluar de forma instantáneamente, sin seguir una heurística compleja ni basarse en los resultados de evaluaciones anteriores (motivo por el cual en ellos no existe una relación de precedencia entre dichas evaluaciones, permitiendo su ejecución en paralelo). Por el contrario, los métodos de optimización bayesiana realizan una búsqueda *informada*, en la que se opta por invertir algo más de tiempo en elegir la siguiente configuración de hiperparámetros a evaluar pero a la par se persigue reducir el número de evaluaciones a realizar y que éstas sean significativas, permitiendo inferir el comportamiento de la función objetivo e identificar sus mínimos con el menor número de evaluaciones posible. Por supuesto, el procedimiento de selección de los valores de los hiperparámetros es mucho más complejo que en los casos anteriores y también su cálculo requiere un tiempo mayor, pero dicho lapso es insignificante en comparación con el tiempo de entrenamiento necesario para realizar una evaluación de la función objetivo [BYC13]. Por lo tanto, la reducción del número de evaluaciones que este método proporciona frente a los dos métodos de búsqueda no informada descritos en las subsecciones anteriores permite identificar los mínimos de la función objetivo con un menor consumo de recursos computacionales.

Aunque el método de búsqueda aleatoria realiza un menor número de evaluaciones que la búsqueda en malla, sigue requiriendo un número considerable de muestras, distribuidas uniformemente por el espacio a cubrir, superior al que necesitan los métodos de optimización bayesiana para hallar un resultado equivalente. No obstante, en comparación con los dos métodos de búsqueda anteriores, el refinamiento sucesivo del modelo sustituto sí impone una relación de precedencia entre las evaluaciones de la función objetivo que limita su ejecución simultánea. Aunque es cierto que podría realizarse una implementación de un método de optimización bayesiana que ejecutase concurrentemente varias instancias de los pasos 2 y 3 a partir de las configuraciones más prometedoras, actualizando un mismo modelo con sus resultados, el grado de simultaneidad alcanzable seguiría siendo inferior al de los otros dos métodos de búsqueda. Por otra parte, la disminución en el consumo de recursos que deviene de la diferencia en el número de evaluaciones es independiente del factor de concurrencia, haciendo que sea aconsejable optar por este método de búsqueda para la optimización de hiperparámetros en el caso general, siempre que no se posean motivos específicos para emplear una de las otras dos opciones propuestas.

Éstos son los principios teóricos, pero para poder trasladarlos a la práctica hay determinados elementos dependientes del escenario particular y de otras decisiones de diseño que deben concretarse. En este contexto resulta de especial utilidad plantear el problema en términos de los denominados “métodos de optimización secuencial basada en modelos” (SMBO, *Sequential Model-Based Optimization methods*) [HHLB11], una formalización de las técnicas de optimización bayesiana cuando se emplean en un caso como el presente: aplicando razonamiento bayesiano de forma iterativa para determinar la siguiente evaluación

a realizar y a continuación utilizando los resultados para actualizar un modelo probabilístico sustituto. A la hora de plantear un método que se englobe en esta categoría deben definirse cinco elementos:

1. El dominio de valores sobre el que realizar la búsqueda. En este caso, el dominio definido por el usuario en la petición de entrenamiento para cada uno de los hiperparámetros especificados, que constituyen las múltiples dimensiones del espacio de búsqueda.
2. Una función objetivo que reciba las entradas y como salida proporcione una puntuación, la cual se desee minimizar o maximizar. En este caso, una evaluación de la función objetivo corresponde a la ejecución de un proceso de entrenamiento, con el algoritmo y el *dataset* especificados y la configuración de hiperparámetros seleccionada. El resultado del entrenamiento (y la consiguiente comprobación de las predicciones realizadas sobre los datos de evaluación) devuelve un número real, acorde a la métrica utilizada por el algoritmo, y que constituye una puntuación a minimizar en caso de tratarse de una tasa de error o a maximizar en caso de tratarse de una métrica cualitativa.
3. El modelo sustituto de la función objetivo real. Es la representación probabilística de la función objetivo a partir de evaluaciones previas. También se denomina “superficie de respuesta”, debido a que es el mapeo entre un punto en un espacio de elevada dimensionalidad y la probabilidad de una puntuación en la función objetivo.
4. Un criterio, denominado función de selección, para determinar qué valores (configuración de los hiperparámetros) elegir del modelo sustituto para su evaluación sobre la función objetivo.
5. Un historial de pares [*puntuación, valores de entrada*] utilizado por el algoritmo para actualizar el modelo sustituto en cada iteración.

Si bien los dos primeros puntos ya han sido suficientemente definidos y el quinto no entraña una especial dificultad, el tercer y el cuarto punto aún deben definirse para su adaptación al presente escenario. La construcción del modelo sustituto puede realizarse mediante varias técnicas y algoritmos, siendo los más comunes según [BBBK11] los procesos gaussianos, la regresión mediante bosques aleatorios y los estimadores de árboles de Parzen (*TPE, Tree Parzen Estimators*).

En lo que respecta a la función que implementa el criterio de selección, la opción más utilizada, también según [BBBK11], es la mejora esperada o *Expected Improvement* [WHD18]:

$$EY_{y^*}(x) = \int_{-\infty}^{y^*} (y^* - y) \ p(y|x) \ dy$$

En esta función, y^* es un umbral de la función objetivo, x son los valores propuestos para los hiperparámetros, y es el valor real devuelto por la función objetivo al utilizar x como valor de entrada y $p(y|x)$ es el modelo probabilístico sustituto que expresa la probabilidad de y dado x . Maximizar la mejora esperada respecto a x equivale a localizar la mejor configuración de hiperparámetros en la función $p(y|x)$ del modelo sustituto, aquella que se espera que alcance un mejor resultado al evaluarse sobre la función objetivo. Si $p(y|x)$ es cero en todo el espacio en el que se cumple $y < y^*$, entonces no se espera que la configuración x de hiperparámetros proporcione ninguna mejora. Si el resultado de la integral es positivo, significa que se espera que la configuración x de hiperparámetros obtenga un resultado superior al valor del umbral y^* .

Puesto que la implementación de estos elementos es de una considerable complejidad, existen herramientas y bibliotecas que proporcionan implementaciones de métodos de optimización secuencial basada en modelos, utilizando diferentes algoritmos para la construcción del modelo probabilístico sustituto. Actualmente la más completa y que posee un mayor grado de madurez es Hyperopt¹⁰, la cual realiza la construcción del modelo sustituto mediante TPEs (estimadores de árboles de Parzen). Otras opciones son Spearmint¹¹ y MOE¹², que emplean procesos gaussianos, y SMAC¹³, que emplea regresión mediante bosques aleatorios. Puede encontrarse más información sobre estos métodos en [DMC16] y [BBBK11], mientras que [BYC13] profundiza en la aplicación específica de los TPEs. La regresión mediante bosques aleatorios o *random forests* se describe en uno de los apartados de la Subsección 4.3.3, al ser uno de los algoritmos de aprendizaje automático que ofrece Zygard.

En un primer momento se planteó que Zygard utilizase Hyperopt, al ser la herramienta más completa y con un mayor nivel de madurez y soporte, pero finalmente se decidió utilizar SMAC. Esta elección se debe a que, mientras que las tres primeras tecnologías citadas se encuentran implementadas en Python, SMAC se encuentra implementado en Java. Ya que el motor de ejecución de tareas, descrito en la Subsección 4.1.2 y desde el que se coordina la optimización de hiperparámetros, también se encuentra implementado en Java, esto permite utilizar la tecnología no como una herramienta externa que deba ser invocada en un nuevo proceso, sino como una dependencia adicional a la que puede accederse como a una biblioteca, facilitando la integración directa con sus clases.

Desafortunadamente, SMAC no se distribuye como una biblioteca, sino como una herramienta con entidad propia. Debido a este inconveniente, para integrar SMAC en Zygard se ha inspeccionado el código fuente de los niveles superiores de su estructura para comprender su funcionamiento mediante ingeniería inversa e implementar una interfaz para operar

¹⁰hyperopt: Distributed Asynchronous Hyperparameter Optimization in Python,

<https://github.com/hyperopt/hyperopt>

¹¹Spearmint: a package to perform Bayesian optimization, <https://github.com/JasperSnoek/spearmint>

¹²MOE: A global, black box optimization engine for real world metric optimization,

<https://github.com/Yelp/MOE>

¹³SMAC: Sequential Model-based Algorithm Configuration, <http://cs.ubc.ca/labs/beta/Projects/SMAC/>

fácilmente con la estructura de clases de SMAC desde el código propio del motor de ejecución de tareas de Zygarde. Además, ha sido necesario trazar e identificar las dependencias propias de SMAC para incluirlas en el proyecto Maven del motor de ejecución de tareas y reempaquetar el código fuente de SMAC para incluirlo también como una dependencia, todo ello de acuerdo con la licencia de SMAC para fines académicos (AGPLv3).

En cuanto a la estructura del código de Zygarde desarrollado para esta integración, en todo momento se hace uso de las técnicas características del paradigma de programación orientada a objetos, como encapsulamiento, herencia y polimorfismo, para proporcionar una interfaz común que permite a los niveles superiores operar homogéneamente con independencia del método de optimización de hiperparámetros utilizado. Aunque la implementación de esta interfaz en el caso de la optimización bayesiana es más compleja que en los otros dos métodos de optimización disponibles, al incluir la integración con SMAC y la lógica de transformación de objetos entre las jerarquías de clases de SMAC y Zygarde, la especificación de las operaciones públicas ofrecidas es la misma, garantizando la abstracción en la separación de niveles. Esta estructura responde a la voluntad de realizar un diseño que se adhiere a los principios SOLID [Mar00] y en este caso específico el cumplimiento del principio de inversión de dependencias.

4.3 Algoritmos de aprendizaje automático

Zygarde pone a disposición de los usuarios un abanico de algoritmos de aprendizaje automático, pertenecientes a varios dominios, que se pueden seleccionar en la petición de entrenamiento para su evaluación por parte de la plataforma y así determinar cuál alcanza mejores resultados. Mientras que la sección anterior trató los aspectos referentes a la selección de la combinación de hiperparámetros óptima, con independencia del algoritmo empleado, la presente sección realiza una enumeración de cada uno de los algoritmos que proporciona Zygarde, agrupados por dominio. Tal como se expuso en la Sección 1.2, no es la intención de este trabajo realizar implementaciones propias de los algoritmos de aprendizaje automático, sino aprovechar las implementaciones distribuidas existentes de estos algoritmos, ya probadas y optimizadas, e incluirlas en la plataforma.

En todos los dominios presentados en las siguientes subsecciones, las implementaciones de los algoritmos corresponden a las ofrecidas por Spark MLlib, salvo en el dominio relativo a redes neuronales, en el que se hace uso de Horovod (como variante de TensorFlow distribuido). Aunque es cierto que Spark MLlib también dispone de un clasificador basado en redes neuronales, se ha optado por adoptar Horovod en este caso para así mostrar cómo Zygarde es capaz de operar con diferentes *frameworks* que se ejecutan sobre clústeres de diferentes tecnologías. El diseño del motor de ejecución de tareas, presentado en la Subsección 4.1.2, no sólo permite que los usuarios emitan sus peticiones sin necesidad de conocer

la implementación específica de los algoritmos, sino que los sucesivos niveles en los que se estructura la jerarquía de clases, de acuerdo con los principios de la programación orientada a objetos, proporcionan al propio desarrollador una abstracción sobre las tecnologías utilizadas y las particularidades de cada una.

La exportación de los modelos generados, que podrán importarse de nuevo desde los programas del usuario para su reutilización, se realiza en el formato PMML (*Predictive Model Markup Language*)¹⁴ cuando es posible. La adopción de este formato, estandarizado y bien definido, persigue la interoperabilidad de los modelos generados entre plataformas con independencia del *framework* utilizado. En aquellos casos en los que el algoritmo empleado carece de soporte por parte de la especificación de PMML, la exportación se realiza en el formato de persistencia propio de Spark MLlib. En cualquier caso, el usuario siempre tiene la opción de utilizar Zygade para determinar a cuál es la combinación de algoritmo e hiperparámetros que ofrece un mejor resultado y posteriormente ejecutar una tarea de entrenamiento en su propio entorno en base a esta configuración, para así generar un modelo cuya compatibilidad con los *frameworks* y programas del usuario esté garantizada.

Para cada dominio, que abarca varios algoritmos, se proporciona una breve descripción y se especifica la métrica utilizada para evaluar la ejecución de los algoritmos del dominio. Esto es particularmente relevante dado que al comparar el desempeño de diferentes algoritmos, todos ellos pertenecientes al mismo dominio, determinado por la naturaleza del problema, éstos deben evaluarse mediante la misma métrica. Para cada uno de los algoritmos, englobado en el dominio correspondiente, se proporciona una breve descripción junto con la especificación de los parámetros de entrada que dicho algoritmo admite, detallando su significado, su tipo de datos y, si procede, el conjunto de los valores permitidos. Igualmente, para cada algoritmo se indica en qué formato se realiza la exportación del modelo generado.

4.3.1 Clasificación binomial

Las tareas de clasificación son, por lo general, las más comunes en aprendizaje automático. Un clasificador implementa un mapeo $\hat{c} : \mathcal{X} \rightarrow \mathcal{C}$, donde $\mathcal{C} = \{C_1, C_2, \dots, C_k\}$ es un conjunto finito (y generalmente reducido) de clases identificadas mediante etiquetas. El conjunto de datos utilizado para entrenar un clasificador tiene la forma $(x, c(x))$, donde $x \in \mathcal{X}$ son los valores de los datos de entrada (características) y $c(x)$ es el valor correspondiente a dicha entrada. El entrenamiento de un clasificador implica la construcción de una función \hat{c} que coincida lo máximo posible con c , no tan solo sobre los datos de entrenamiento, sino idealmente sobre el espacio muestral al completo.

Si los datos deben discriminarse únicamente entre dos clases diferentes, este tipo de tarea de clasificación recibe el nombre de clasificación binomial.

¹⁴PMML 4.3 - General Structure, <http://dmg.org/pmm/v4-3/GeneralStructure.html>

La calidad de los modelos generados por los algoritmos de este dominio se calcula mediante el Coeficiente de Correlación de Matthews (MCC, *Matthews Correlation Coefficient*), particularmente apropiado para la evaluación de clasificaciones binomiales [CJ20]. Este coeficiente, similar al coeficiente phi (Φ) de Pearson, puede calcularse directamente a partir de la matriz de confusión mediante la siguiente fórmula, la cual se expresa en términos de los verdaderos positivos (TP), falsos positivos (FP), verdaderos negativos (TN) y falsos negativos (FN):

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}$$

A continuación se estudian los algoritmos proporcionados por Zygarde pertenecientes al dominio de la clasificación binomial.

Máquinas de vectores de soporte (SVMs)

Las máquinas de vectores de soporte (SVMs, *Support Vector Machines*) son un método lineal que construye un hiperplano o un conjunto de hiperplanos en un espacio de elevada dimensionalidad, empleadas principalmente para resolver tareas de clasificación. Con el fin de lograr una discriminación lo más clara posible entre las clases, se busca el hiperplano que maximiza la distancia a los puntos más cercanos que representan los datos de entrenamiento de cualquier clase (el denominado margen funcional), dado que, por lo general, cuanto mayor es este margen menor es el error del clasificador.

Muchos métodos dirigidos a la resolución de problemas mediante la aplicación de técnicas de aprendizaje automático pueden formularse como un problema de optimización convexo: la tarea de hallar un minimizador de una función convexa f que depende de un vector variable \mathbf{w} , con d elementos. Esta distribución puede expresarse como un problema de optimización $\min_{\mathbf{w} \in \mathbb{R}^d} f(\mathbf{w})$, donde la función objetivo toma la siguiente forma:

$$f(\mathbf{w}) := \lambda R(\mathbf{w}) + \frac{1}{n} \sum_{i=1}^n L(\mathbf{w}; \mathbf{x}_i, y_i)$$

En este escenario, los vectores $x_i \in \mathbb{R}^d$ son las instancias o registros de los datos de entrenamiento, mientras que $y_i \in \mathbb{Z}^+$ son los índices asignados a los valores posibles de la variable de salida. En el caso de los métodos lineales, $L(\mathbf{w}; \mathbf{x}, y)$ puede expresarse como una función de $\mathbf{w}^T \mathbf{x}$ e y .

La función objetivo f consta de dos partes: el parámetro regularizador que controla la complejidad del modelo y la función de pérdida que mide el error del modelo sobre los datos de entrenamiento. El parámetro regularizador, cuyo valor es fijo, denotándose como $\lambda \geq 0$, define el equilibrio entre la minimización de la función de pérdida (la reducción del error de entrenamiento) y la minimización de la complejidad del modelo (dado que un modelo

excesivamente complejo puede acarrear problemas como el sobreajuste sobre las muestras de los datos de entrenamiento).

En el caso de las máquinas de vectores de soporte, la función de pérdida se define en términos de la pérdida de articulación:

$$L(\mathbf{w}; \mathbf{x}, y) := \max\{0, 1 - y\mathbf{w}^T \mathbf{x}\}$$

Dada una entrada \mathbf{x} de los datos de entrenamiento, el modelo realiza predicciones a partir del valor de $\mathbf{w}^T \mathbf{x}$. Por defecto, el resultado es positivo si $\mathbf{w}^T \mathbf{x} \geq 0$, y negativo en caso contrario.

Los modelos generados mediante este método se exportan al formato PMML, puesto que es uno de los algoritmos para los que la especificación de este formato ofrece soporte.

Regresión logística binomial

La regresión logística es otro método popularmente utilizado para predecir el valor de una variable categórica. Constituyéndose como un caso específico de los modelos lineales generalizados (GLMs, los cuales se estudian en un apartado posterior), la regresión logística puede adaptarse tanto para la predicción de clases con sólo dos valores o con múltiples valores posibles.

Al igual que las máquinas de vectores de soporte, que también son un método lineal, se constituyen como un problema de optimización en el que la función objetivo toma la forma

$$f(\mathbf{w}) = \lambda R(\mathbf{w}) + \frac{1}{n} \sum_{i=1}^n L(\mathbf{w}, \mathbf{x}_i, y_i)$$

con la salvedad de que en este caso la función de pérdida viene dada en términos de la pérdida logística:

$$L(\mathbf{w}; \mathbf{x}, y) = \log(1 + \exp(-y\mathbf{w}^T \mathbf{x}))$$

En el escenario de la regresión logística aplicada a problemas de clasificación binaria, como es el caso, el modelo realiza predicciones a partir de una nueva entrada de datos \mathbf{x} mediante la función logística

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

donde $z = \mathbf{w}^T \mathbf{x}$. Por defecto, si $f(\mathbf{w}^T \mathbf{x}) > 0,5$, la salida es positiva, siendo negativa en caso contrario. No obstante, y a diferencia de las máquinas de vectores de soporte, el valor de salida del modelo de regresión logística $f(z)$ tiene una interpretación probabilística (por ejemplo, la probabilidad de que \mathbf{x} sea positivo).



Los modelos generados mediante este método se exportan al formato PMML, puesto que es uno de los algoritmos para los que la especificación de este formato ofrece soporte.

4.3.2 Clasificación multinomial

Las tareas de clasificación multinomial son similares a las tareas de clasificación binomial ya vistas, con la salvedad de que el valor a predecir sobre la variable de salida no sólo oscila entre dos clases antagónicas, sino que debe elegirse de entre un número K mayor de clases. Algunos de los algoritmos ya estudiados sobre el escenario de la clasificación binomial pueden adaptarse para su utilización en tareas de clasificación multinomial, tal como se describe en los siguientes apartados.

La calidad de los modelos generados por los algoritmos de este dominio se calcula mediante el Valor-F [Chi92], que pondera tanto la precisión como la exhaustividad (*recall*) mediante una media armónica, permitiendo tener constancia de ambos factores en un único valor. Estos dos valores se definen como sigue, en términos de positivos y negativos, verdaderos y falsos:

$$precision = \frac{TP}{TP + FP}$$

$$recall = \frac{TP}{TP + FN}$$

A partir de estos valores, la métrica del Valor-F puede calcularse de la siguiente forma:

$$F_1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

A continuación se estudian los algoritmos proporcionados por Zygarde pertenecientes al dominio de la clasificación multinomial.

Naive Bayes

Naive Bayes es un algoritmo de clasificación multinomial que opera bajo la premisa de la independencia entre cada par de variables. Partiendo de la certeza de dicha premisa (que se tendría por cierta, en caso de haber optado por este algoritmo), el funcionamiento del algoritmo es sencillo y puede entrar un modelo de forma muy eficiente. Con una sola pasada sobre los datos de entrenamiento calcula la distribución de probabilidad condicionada de cada etiqueta perteneciente a los valores admitidos en cada característica (contando con variables categóricas o discretas). Seguidamente, aplica el teorema de Bayes para calcular la distribución de probabilidad condicionada de la variable de salida a partir de una observación y utiliza esta distribución de probabilidad para realizar una predicción.



La aplicación más común de los modelos generados mediante el método de *naive Bayes* es la clasificación de documentos, en cuyo caso cada observación (o entrada de datos) es un documento y cada característica es un término cuyo valor es la frecuencia del término (en el método de *naive Bayes* multinomial) o un valor booleano indicando si el término aparece en el documento (en el método de *naive Bayes* de Bernoulli). Dado el gran número de características presentes en el espacio de entrada de este tipo de aplicaciones, los vectores de características que componen el conjunto de datos de entrenamiento suelen expresarse en la forma de matrices dispersas, reduciendo el espacio necesario para su almacenamiento y favoreciendo la eficiencia en su procesamiento.

Los modelos generados mediante este método se exportan al formato de persistencia propio de Spark MLlib, ya que este algoritmo no se encuentra soportado por la especificación del formato PMML.

Regresión logística multinomial

El método de regresión logística binomial ya estudiado puede generalizarse al caso de la regresión logística multinomial, aplicable al entrenamiento dirigido a problemas de clasificación en los que la variable a predecir puede pertenecer a más de dos clases. Asumiendo que la predicción debe realizarse eligiendo entre K clases diferentes, este algoritmo opera eligiendo una de ellas como pivote y elaborando $K - 1$ modelos de regresión logística binomial que cubren la clase pivote y cada una de las clases restantes por separado. A partir de los $K - 1$ modelos de regresión logística binomial generados, la clase con mayor probabilidad será designada como la clase predicha (pudiendo también resultar elegida la clase que actúa como pivote).

Los modelos generados mediante este método se exportan al formato de persistencia propio de Spark MLlib, ya que este algoritmo no se encuentra soportado por la especificación del formato PMML.

Árboles de decisión

Los árboles de decisión y sus “ensembles” son uno de los métodos de aprendizaje automático más populares, pudiendo utilizarse en tareas tanto de clasificación como de regresión. Su uso se encuentra ampliamente extendido dado que son fáciles de interpretar, pueden operar con características categóricas, se pueden adaptar con facilidad para aplicarlos en un escenario con múltiples clases, no requieren el escalado de las características y son capaces de capturar las interacciones y relaciones entre características.

La construcción de un árbol de decisión se sustenta sobre un algoritmo iterativo que sucesivamente determina cuál es la característica sobre la que efectuar la siguiente escisión. Por lo general se emplean árboles binarios, que exigen que las características de entrada del modelo sean variables booleanas o pueda aplicarse una transformación sobre ellas que

permite tratarlas como tales (mediante la agrupación de los valores posibles en el caso de una variable discreta o estableciendo un umbral o rango en el caso de una variable continua). La selección de la característica a escindir en la siguiente iteración se realiza en términos de la ganancia de información que se puede obtener en esa escisión, tratando de maximizar la discriminación que se produce sobre la variable de salida.

A continuación se presenta un escenario de ejemplo simplificado, extraído de [Fla12], en el que se asume que se opera con características booleanas. En este ejemplo se aborda la división de D en D_1 y D_2 . Igualmente, se asume que se opera con dos clases, denotando con D^+ y D^- los positivos y negativos en D . A la hora de identificar la mejor escisión posible en términos de positivos y negativos sobre el espacio de muestras, el caso en el que $D_1^+ = D^+$ y $D_1^- = D^-$ es el mejor caso posible y en el que se dice que los nodos hijos resultantes son "puros". Contando con un conjunto de muestras con n^+ positivos y n^- negativos, la impureza resultante de la escisión puede definirse a partir de la proporción $\dot{p} = n^+/(n^+ + n^-)$, denominada la probabilidad empírica de la clase positiva. Algunas de las medidas utilizadas para el cálculo de la impureza son la clase minoritaria o tasa de error ($\min(\dot{p}, 1 - \dot{p})$), el índice de Gini ($2\dot{p}(1 - \dot{p})$) y la entropía ($-\dot{p} \log_2(\dot{p}) - (1 - \dot{p}) \log_2(1 - \dot{p})$). A través de las sucesivas iteraciones se recorren los nodos hoja del árbol, realizándose una escisión sobre aquel nodo que minimiza la impureza calculada, buscando así maximizar la pureza de los nodos hijos resultantes y, por lo tanto, la ganancia de información resultante de la escisión.

Los modelos generados mediante este método se exportan al formato de persistencia propio de Spark MLlib, ya que este algoritmo no se encuentra soportado por la especificación del formato PMML.

Bosques aleatorios (*random forests*)

Un método particularmente efectivo de “ensemble” es el denominado *bagging* (que proviene de la contracción de “*bootstrap aggregating*”), el cual crea diferentes modelos a partir de muestras aleatorias del conjunto de datos original. Dado que la forma en la que se seleccionan los datos que conforman las muestras permite que éstas contengan datos duplicados, algunas entradas de datos podrían encontrarse ausentes incluso si el tamaño de la muestra fuera equivalente al del propio conjunto de datos original. En este caso se persigue exactamente ese fenómeno, provocar que las diferencias entre las muestras originen diversidad entre los modelos que conformarán el ensemble.

Este método de *bagging* resulta especialmente útil en combinación con modelos basados en árboles, que muestran una considerable sensibilidad a variaciones, que muestran una considerable sensibilidad a variaciones en los datos de entrenamiento. Al aplicarse sobre modelos basados en árboles, este método se emplea con la intención de construir diferentes árboles a partir de diferentes subconjuntos de las características proporcionadas, un proceso

conocido como “muestreo de sub-espacios”. Esto contribuye a aumentar la diversidad del ensemble aún en mayor medida sobre el *bagging* original y cuenta con la ventaja adicional de reducir el tiempo necesario para el entrenamiento de cada árbol, al limitar el número de características de entrada en cada modelo. El ensemble resultante de este proceso se conoce como “bosque aleatorio” o “*random forest*”.

Los modelos generados mediante este método se exportan al formato de persistencia propio de Spark MLlib, ya que este algoritmo no se encuentra soportado por la especificación del formato PMML.

GBT (*Gradient-Boosted Trees*)

Los GBTs (*Gradient-Boosted Trees*) son, al igual que *random forests*, “ensembles” de árboles de decisión, pero con un proceso de entrenamiento diferente al de éste último. Mientras que el método de *random forests* puede entrenar a varios árboles simultáneamente, al no existir una relación de precedencia entre ellos, el entrenamiento con GBTs se realiza de forma iterativa, entrenando un único árbol cada vez, cuyos resultados tendrán una influencia directa sobre la siguiente iteración. De esta forma, los GBTs entranan árboles de decisión de forma iterativa con el objetivo de minimizar una función de pérdida.

En cada iteración, el algoritmo utiliza el estado actual del árbol para predecir el valor de cada elemento del conjunto de datos de entrenamiento y evalúa los resultados de la predicción. El conjunto de datos de entrenamiento es re-etiquetado para dar más peso a entradas de datos en las que la predicción no ha sido precisa. Gracias a esta potenciación, se pretende que el árbol construido en la siguiente iteración sea capaz de corregir el déficit previamente existente y devuelva un resultado más acertado sobre los datos en cuya predicción había errado anteriormente. Las entradas a potenciar del conjunto de datos de entrenamiento se determinan en base a la función de pérdida anteriormente mencionada. Mientras que en tareas de clasificación suele utilizarse la pérdida en escala logarítmica, en tareas de regresión suele emplearse el error cuadrático o en ocasiones el error absoluto, más robusto ante la presencia de datos anómalos (*outliers*).

Al igual que ocurre con los otros dos métodos basados en árboles de decisión, los modelos generados mediante este método se exportan al formato de persistencia propio de Spark MLlib, ya que este algoritmo no se encuentra soportado por la especificación del formato PMML.

4.3.3 Regresión

Los métodos de regresión son técnicas de aprendizaje supervisado que tratan de realizar una inferencia sobre el valor de una variable continua perteneciente al dominio de los números reales. Por lo tanto, la labor de un modelo capaz de realizar una regresión, también denominado un estimador de funciones, es ser capaz de llevar a cabo un mapeo $\hat{f} : \mathcal{X} \rightarrow \mathbb{R}$

a partir de un conjunto de ejemplos en la forma $(x_i, f(x_i))$, de tal forma que \hat{f} y f coincidan lo máximo posible.

Puesto que el valor a estimar es un número real, la forma de evaluar el desempeño del modelo es teniendo en cuenta la diferencia en valor absoluto entre el valor predicho y el valor real a predecir. Por este motivo, los modelos generados mediante los algoritmos regresión proporcionados por Zygarde se evalúan mediante la raíz del error cuadrático medio (RMSE, *Root Mean Square Error*, también conocido como RMSD, *Root Mean Square Deviation*), una medida frecuentemente usada en este tipo de estimaciones:

$$RMSE = \sqrt{\frac{1}{n} \sum_{i=1}^n (\hat{y}_i - y_i)^2}$$

A continuación se estudian los algoritmos proporcionados por Zygarde pertenecientes al dominio de la regresión.

Regresión lineal

Los modelos lineales son utilizados con frecuencia en tareas predictivas, como clasificación, estimación de probabilidades y regresión. En particular, la regresión lineal es un problema conocido y estudiado en profundidad, existiendo diferentes métodos para su resolución. La implementación utilizada por Zygarde se basa en el método de mínimos cuadrados, desarrollado por Gauss a finales del siglo XVIII. Denominando “residuos” a la diferencia entre el valor real de la función y el estimado, tal que $\epsilon_i = f(x_i) - \hat{f}(x_i)$, este método trata de hallar la función \hat{f} que minimiza el valor de $\sum_{i=1}^n \epsilon_i^2$.

Los modelos generados mediante este método se exportan al formato PMML, puesto que es uno de los algoritmos para los que la especificación de este formato ofrece soporte.

Modelo lineal generalizado (GLM)

En contraste con la regresión lineal, donde se asume que la variable de salida Y_i sigue una distribución gaussiana, los modelos lineales generalizados (GLMs, *Generalized Linear Models*) son especificaciones de modelos lineales en los que la variable de respuesta sigue una distribución perteneciente a una familia exponencial natural¹⁵, como es el caso de la distribución de Poisson.

La forma de una distribución perteneciente a una familia exponencial natural puede representarse de la siguiente forma:

$$f_Y(y|\theta, \tau) = h(y, \tau) \exp\left(\frac{\theta \cdot y - A(\theta)}{d(\tau)}\right)$$

¹⁵Natural exponential family - Wikipedia, https://en.wikipedia.org/wiki/Natural_exponential_family

donde θ es el parámetro de interés y τ es un parámetro de dispersión. En un GLM se asume que la variable de respuesta Y_i es acorde a una distribución de una familia exponencial natural, de tal forma que:

$$Y_i \sim f(\cdot | \theta_i, \tau)$$

donde, a su vez, el parámetro de interés θ_i y el valor esperado de la variable de respuesta μ_i se encuentran relacionados:

$$\mu_i = A'(\theta_i)$$

En este punto, $A'(\theta_i)$ se define en base a la forma de la distribución seleccionada. Los GLMs también permiten la especificación de una función de enlace, que determina la relación entre el valor esperado de la variable de respuesta μ_i y el denominado *predictor lineal* η_i :

$$g(\mu_i) = \eta_i = \vec{x}_i^T \cdot \vec{\beta}$$

Por lo general, la función de enlace se escoge de tal forma que $A' = g^{-1}$, lo que contribuye a simplificar la relación entre el parámetro de interés θ y el predictor lineal η . En este caso, se dice que la función de enlace $g(\mu)$ es la función de enlace “canónica”:

$$\theta_i = A'^{-1}(\mu_i) = g(g^{-1}(\eta_i)) = \eta_i$$

En última instancia, un GLM localiza los coeficientes de regresión $\vec{\beta}$ que maximizan la función de verosimilitud:

$$\max_{\vec{\beta}} \mathcal{L}(\vec{\theta} | \vec{y}, X) = \prod_{i=1}^N h(y_i, \tau) \exp\left(\frac{y_i \theta_i - A(\theta_i)}{d(\tau)}\right)$$

donde el parámetro de interés θ_i se relaciona con los coeficientes de regresión $\vec{\beta}$ de la siguiente forma:

$$\theta_i = A'^{-1}(g^{-1}(\vec{x}_i \cdot \vec{\beta}))$$

Los modelos generados mediante este método se exportan al formato de persistencia propio de Spark MLlib, ya que este algoritmo no se encuentra soportado por la especificación del formato PMML.

Árboles de decisión

Partiendo del caso ya estudiado del uso de árboles de decisión en problemas de clasificación, los árboles de decisión también pueden adaptarse de forma sencilla para su utilización en tareas de regresión. En este caso, la varianza de un conjunto Y de valores objetivo puede definirse como la media de la distancia cuadrática respecto a la media:

$$\text{Var}(Y) = \frac{1}{|Y|} \sum_{y \in Y} (y - \bar{y})^2$$

donde $\bar{y} = \frac{1}{|Y|} \sum_{y \in Y} y$ es la media de los valores objetivo en Y . Si se partitiona el conjunto de valores objetivo de Y en una serie de conjuntos mutuamente excluyentes $\{Y_1, \dots, Y_l\}$, la media ponderada de la varianza pasa a ser:

$$\text{Var}(\{Y_1, \dots, Y_l\}) = \sum_{j=1}^l \frac{|Y_j|}{|Y|} \text{Var}(Y_j) = \sum_{j=1}^l \frac{|Y_j|}{|Y|} \left(\frac{1}{|Y_j|} \sum_{y \in Y_j} y^2 - \bar{y}_j^2 \right) = \frac{1}{|Y|} \sum_{y \in Y} y^2 - \sum_{j=1}^l \frac{|Y_j|}{|Y|} \bar{y}_j^2$$

Por lo tanto, para utilizar árboles de decisión en una tarea de regresión bastaría con sustituir la medida de impureza utilizada en la tarea de clasificación por la varianza aquí mostrada. Puesto que $\frac{1}{|Y|} \sum_{y \in Y} y^2$ es constante en términos del conjunto de valores Y , minimizar la varianza sobre todas las posibles escisiones de un determinado nodo padre es equivalente a maximizar el promedio ponderado de las distancias al cuadrado respecto a la media de los nodos hijos.

Al igual que ocurre con el algoritmo de árboles de decisión utilizado para tareas de clasificación, los modelos generados mediante este método se exportan al formato de persistencia propio de Spark MLlib, ya que este algoritmo no se encuentra soportado por la especificación del formato PMML.

Bosques aleatorios (*random forests*)

La aplicación del método de “ensemble” basado en bosques aleatorios, o *random forests*, en tareas de regresión es similar al procedimiento expuesto en la descripción de este método sobre tareas de clasificación. Se opera de idéntica forma en lo que se refiere a la construcción de diferentes árboles, mediante la selección de muestras del conjunto original de datos de entrenamiento y subconjuntos de las características de entrada. La transformación y adaptación necesarias para aplicar este método en tareas de regresión tiene lugar a nivel de los árboles de decisión individuales y es la que se describe en el apartado anterior, utilizando la varianza como medida de impureza para determinar la ganancia de información que, en términos probabilísticos, puede obtenerse al realizar una escisión sobre cada nodo hoja del árbol.

Al igual que ocurre con el algoritmo de árboles de decisión (y, por ende, con el de bosques aleatorios) utilizado para tareas de clasificación, los modelos generados mediante este método se exportan al formato de persistencia propio de Spark MLlib, ya que este algoritmo no se encuentra soportado por la especificación del formato PMML.

GBT (*Gradient-Boosted Tree*)

La aplicación del método de *Gradient-Boosted Trees* en tareas de regresión es similar al procedimiento expuesto en la descripción de este método sobre tareas de clasificación. Al igual que en el método de bosques aleatorios del apartado anterior, la adaptación necesaria para aplicar este método en tareas de regresión tiene lugar a nivel de los árboles de decisión individuales y consiste en utilizar la varianza como media de impureza a partir de la que determinar la ganancia de información que, potencialmente, puede obtenerse al realizar una escisión sobre cada nodo hoja del árbol.

Al igual que ocurre en el caso anterior, los modelos generados mediante este método se exportan al formato de persistencia propio de Spark MLlib, ya que los algoritmos basados en árboles de decisión no se encuentran soportados por la especificación del formato PMML.

4.3.4 Agrupamiento (*clustering*)

El agrupamiento, o *clustering*, es un problema de aprendizaje no supervisado en el que se busca agrupar entidades en subconjuntos denominados clústeres, encontrándose las entidades de un mismo clúster relacionadas entre sí por medio de alguna noción o medida de similitud. Las técnicas de agrupación con frecuencia se emplean en análisis exploratorio y/o como componentes de un *pipeline* jerárquico de aprendizaje supervisado (en los que diferentes clasificadores o modelos de regresión se entrena en cada clúster).

Utilizando la terminología apropiada, un agrupador puede verse como un clasificador que implementa el mapeo $\hat{q} : \mathcal{X} \rightarrow \mathcal{C}$, donde $\mathcal{C} = \{C_1, C_2, \dots, C_k\}$ corresponden a las etiquetas de los clústeres en torno a los que se realiza la agrupación, determinando en qué clúster se engloba cada entrada de datos.

Dado que los métodos de agrupación se tratan en el contexto del aprendizaje no supervisado, no hay un conjunto de etiquetas o valores correctos con los que realizar una comparación para evaluar la calidad de las predicciones realizadas por el modelo. En este caso se ha optado por evaluar la calidad del modelo mediante el cálculo de “siluetas”.

Para cada entrada o punto de datos x_i , $d(x_i, D_j)$ denota la distancia media de x_i a los puntos de datos en el clúster D_j , y $j(i)$ denota el índice del clúster al que x_i pertenece. También se asume que $a(x_i) = d(x_i, D_{j(i)})$ es la distancia media de x_i a los puntos $D_{j(i)}$ de su propio clúster, y que $b(x_i) = \min_{k \neq j(i)} d(x_i, D_k)$ es la distancia media a los puntos

del clúster vecino (el más próximo). En estos términos, se podría esperar que $a(x_i)$ fuera considerablemente menor que $b(x_i)$, aunque no existen garantías al respecto. Por lo tanto, se puede tomar la diferencia $b(x_i) - a(x_i)$ como una noción de cómo de “bien agrupado” está x_i , y dividirlo entre $b(x_i)$ para obtener un número cuyo valor oscile en torno a 1.

No obstante, puede darse el caso en el que $a(x_i) > b(x_i)$, en cuyo caso la diferencia $b(x_i) - a(x_i)$ es negativa. Esto describe una situación en la que, de media, los miembros del clúster vecino están más cerca de x_i que los restantes miembros de su propio clúster. En este caso, y con el fin de obtener un valor normalizado, se realiza una división entre $a(x_i)$, lo que lleva a la siguiente definición:

$$s(x_i) = \frac{b(x_i) - a(x_i)}{\max(a(x_i), b(x_i))}$$

Una *silueta* calcula $s(x)$ para cada entrada de datos, en el clúster en el que haya sido agrupada. Mientras que la media de los $s(i)$ pertenecientes a cada clúster opera como una medida de la dispersión de los puntos de datos que conforman el clúster, la media de $s(i)$ sobre todos los puntos que conforman el *dataset* permite contar con una medida que proporciona una referencia numérica de calidad del agrupamiento que se ha realizado y que puede compararse con la obtenida por otros modelos de agrupamiento para determinar cuál ha obtenido un mejor resultado.

Para determinar la distancia entre los puntos de datos pueden utilizarse varias métricas. En este caso se ha optado por la distancia euclídea elevada al cuadrado, que es la métrica más frecuentemente seleccionada en los algoritmos de agrupamiento basados en distancias.

A continuación se estudian los algoritmos proporcionados por Zygarde pertenecientes al dominio de agrupamiento o *clustering*.

K-Means

K-Means es uno de los algoritmos de agrupamiento cuyo uso se encuentra más ampliamente extendido para agrupar entradas o puntos de datos en un número K predefinido de clústeres. El algoritmo itera de forma sucesiva, decidiendo cómo particionar los puntos de datos en base al centroide más cercano y recalculando los centroides para la siguiente iteración en base al nuevo estado de las particiones. Llega un punto en el que el algoritmo converge, aunque es difícil determinar si el punto de convergencia alcanzado es un mínimo local o global. El método de inicialización de los centroides suele ser significativo en este aspecto, motivo por el cual es recomendable ejecutar el algoritmo varias veces con diferentes métodos de inicialización sobre el conjunto de datos y seleccionar la solución con menor dispersión en el contexto de cada clúster (o, directamente, aquella cuyo coeficiente de silueta alcanza un mejor resultado).

Los modelos generados mediante este método se exportan al formato PMML, puesto que es uno de los algoritmos para los que la especificación de este formato ofrece soporte.

Modelos de mixturas gaussianas (GMMs)

Un modelo de mixturas gaussianas (GMM, *Gaussian Mixture Model*) representa una distribución compuesta en la que los puntos de datos se generan a partir de K sub-distribuciones gaussianas, cada una con su propia media μ_j y matriz de covarianza Σ_j , y donde la proporción de puntos procedente de cada sub-distribución se rige por un $\tau = (\tau_1, \dots, \tau_K)$ prefijado. Si cada punto de datos en una muestra se encontrase etiquetado con el índice de la sub-distribución de la que procede, se trataría de un problema típico de clasificación, el cual podría resolverse con facilidad estimando cada μ_j y Σ_j por separado a partir de los puntos de datos pertenecientes a la clase j . No obstante, dado que en este caso se está tratando con un problema de aprendizaje no supervisado en el que no se dispone de las etiquetas de las clases, el problema se presenta como una tarea de agrupamiento predictivo en la que debe obtenerse el clúster en el que se engloba cada punto de datos, mediante un proceso de reconstrucción a partir de los valores de las características observadas.

Una forma de crear el modelo es disponer cada entrada o punto de datos \mathbf{x}_i en la forma de un vector de valores booleanos $\mathbf{z}_i = (z_{i1}, \dots, z_{iK})$ de tal forma que exactamente un bit z_{ij} tenga el valor 1 y el resto tengan el valor 0, señalando que el i -ésimo punto de datos procede de la j -ésima sub-distribución normal. Partiendo de esta notación, puede representarse la expresión general para un GMM:

$$P(\mathbf{x}_i, \mathbf{z}_i | \theta) = \sum_{j=1}^K z_{ij} \tau_j \frac{1}{(2\pi)^{d/2} \sqrt{|\Sigma_j|}} \exp \left(-\frac{1}{2} (\mathbf{x}_i - \mu_j)^T \Sigma_j^{-1} (\mathbf{x}_i - \mu_j) \right)$$

En dicha expresión, θ recoge todos los parámetros $\tau, \mu_1, \dots, \mu_k$ y $\Sigma_1, \dots, \Sigma_K$.

Interpretando la expresión como un modelo generativo, en primer lugar se selecciona una de las sub-distribuciones utilizando el valor τ y a continuación se invoca a la distribución gaussiana elegida utilizando las variables z_{ij} , que actúan como indicadores.

La estimación de los parámetros de un modelo de mixturas gaussianas a partir de un conjunto de datos es una aplicación común de los algoritmos esperanza-maximización (o algoritmos EM), métodos iterativos utilizados para encontrar estimadores de máxima verosimilitud de parámetros en modelos probabilísticos que dependen de variables no observables.

Los modelos generados mediante este método se exportan al formato de persistencia propio de Spark MLlib, ya que este algoritmo no se encuentra soportado por la especificación del formato PMML.

4.3.5 Redes neuronales y aprendizaje profundo (*deep learning*)

Las redes neuronales artificiales son sistemas inspirados en las redes neuronales biológicas y se caracterizan por sus múltiples aplicaciones en el terreno del aprendizaje automático, desde reconocimiento de imágenes hasta procesamiento del lenguaje natural.

Una red neuronal artificial se compone de nodos interconectados y denominados perceptrones. Cada conexión, análoga a las sinapsis que unen y comunican las neuronas en un cerebro biológico, permite la difusión de datos entre los perceptrones. En el caso más típico, un perceptrón cuenta con varias conexiones de entrada y una de salida, transmitiendo números reales a través de ellas. En este escenario, el valor de salida de un perceptrón se calcula por medio de una función no lineal de la suma del valor de sus entradas, mientras que el grado de participación de cada conexión de entrada en el cálculo del valor de salida se regula mediante una ponderación que se reajusta continuamente conforme el proceso de aprendizaje tiene lugar. Por lo general, los perceptrones cuentan con una función de propagación que determina el valor de las entradas del perceptrón a partir del valor de salida de sus predecesores, un umbral basado en una función de activación, que determina cuándo el perceptrón emite un valor de salida y una función de salida que genera el valor a emitir.

En el caso más común, los perceptrones se organizan en capas. Las señales viajan desde la primera capa (la capa de entrada) a la última (la capa de salida), normalmente tras atravesar múltiples capas intermedias de perceptrones, cada una de las cuales puede aplicar diferentes transformaciones sobre los valores de sus señales de entrada.

Existen varios tipos de redes neuronales artificiales. El más básico es el de las redes neuronales prealimentadas, donde los perceptrones se encuentran claramente alineados en capas y las conexiones entre perceptrones en ningún caso forman un ciclo. Otros tipos son las redes neuronales convolucionales (muy efectivas en la clasificación y segmentación de imágenes, así como en tareas de visión artificial), las redes neuronales recurrentes (que integran bucles de realimentación y se emplean en la predicción de secuencias de datos o series temporales) o las redes generativas antagónicas (utilizadas en aprendizaje no supervisado e implementadas mediante un sistema de dos redes neuronales que compiten entre sí).

Clasificación mediante red neuronal artificial

Un clasificador mediante perceptrón multicapa (MLPC, *MultiLayer Perceptron Classifier*) es un método de clasificación basado en una red neuronal prealimentada; es decir, en la que no hay ciclos y la información se mueve en una sola dirección desde la capa de entrada hasta la capa de salida, atravesando la totalidad de las posibles capas ocultas. Este tipo de clasificador basado en redes neuronales se compone de varias capas de nodos en las que cada capa está conectada de forma completa a la siguiente capa de la red (es decir, cada nodo posee enlaces a todos y cada uno de los nodos de la siguiente capa). Los nodos de la capa de entrada introducen en la red los datos de entrenamiento y los nodos de las capas restantes

mapean sus entradas a valores de salida mediante una combinación lineal de los valores de los valores de entrada, el peso w que se le concede a cada conexión de entrada, un bias b y la aplicación de una función de activación. De esta forma, la función representada por un MLP con $K + 1$ capas puede representarse en forma matricial como sigue:

$$y(\mathbf{x}) = f_K(\dots f_2(f_1(\mathbf{w}_1^T \mathbf{x} + b_1) + b_2) \dots + b_K)$$

Los nodos que se encuentran en las capas intermedias por lo general utilizan una función sigmoide (logística) para generar sus variables de salida:

$$f(z_i) = \frac{1}{1 + e^{-z_i}}$$

Por su parte, los nodos de la capa de salida suelen generar sus valores de salida mediante la función denominada *softmax*. En este escenario de clasificación multinomial, el número de nodos N en la capa de salida corresponde con el número de clases.

$$f(z_i) = \frac{e^{z_i}}{\sum_{k=1}^N e^{z_k}}$$

Los MLP emplean retropropagación para ajustar los pesos de las conexiones de entrada en los perceptrones de las sucesivas capas y así aprender el modelo.

En el escenario que se está tratando, este tipo de red neuronal se emplea en el contexto de una tarea de clasificación multinomial. Por este motivo, el desempeño de los diferentes modelos generados se evalúa utilizando como métrica el Valor-F, como en el caso de los métodos de clasificación multinomial anteriormente estudiados.

Los modelos generados mediante este método se exportan al formato PMML, puesto que es uno de los algoritmos para los que la especificación de este formato ofrece soporte.

4.3.6 Miscelánea

En este apartado se incluyen funciones y métodos que no se ajustan a ninguna de las categorías anteriores o no son algoritmos de aprendizaje automático, pero que aún así se mantienen dentro de la oferta algorítmica de Zygarde por diversas razones.

Función de Branin-Hoo

La función de Branin-Hoo [Bra72] es una función tridimensional utilizada en problemas de optimización. Esta función cuenta con tres mínimos globales y su apariencia es la que se puede observar en la Figura 4.4, mientras que su formulación responde a la siguiente ecuación:



$$f(\mathbf{x}) = a(x_2 - bx_1^2 + cx_1 - r)^2 + s(1 - t) \cos(x_1) + s$$

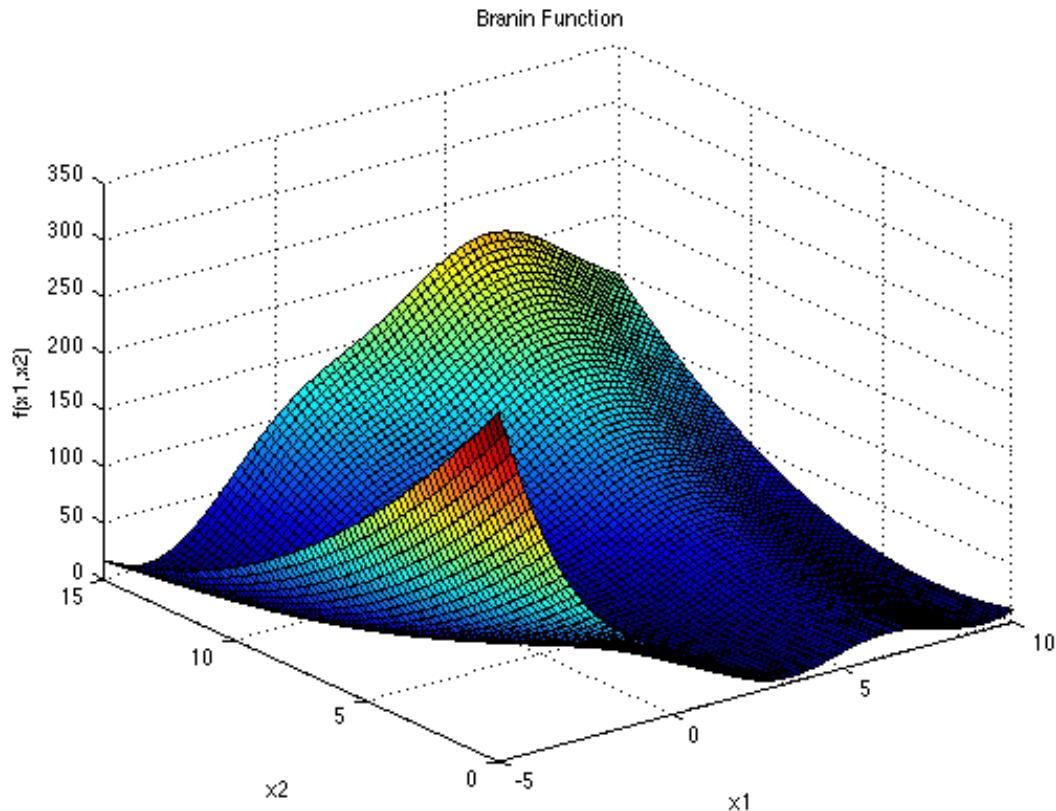


Figura 4.4: Función de Branin-Hoo¹⁶

Esta función se ha utilizado activamente como función objetivo durante las pruebas del ajuste automatizado de hiperparámetros, descrito en la Sección 4.2, en sus tres modalidades. Su objetivo ha sido verificar el correcto funcionamiento de los mecanismos referidos, comprobando que, mediante combinaciones de valores de las variables de entrada x_1 y x_2 , se lograba identificar los mínimos de la función. Es importante tener en cuenta que, tal como se refirió en la Subsección 4.2.3, el ajuste de los hiperparámetros mediante la comparación del desempeño de diferentes modelos de aprendizaje automático no deja de ser un problema de optimización, referente a un valor de salida correspondiente a una función a maximizar o minimizar [Dix78]. En este contexto, esta función ha permitido probar y ajustar la técnica de optimización bayesiana utilizada, así como verificar la correcta ordenación de los resultados en las tres técnicas. Puesto que el cómputo de la función es inmediato y puede calcularse en el propio entorno de desarrollo, su utilización ha evitado tener que realizar varias ejecuciones de algoritmos reales de aprendizaje automático, mucho más pesados en términos

¹⁶Branin Function, <https://sfu.ca/~ssurjano/branin.html>

computacionales y temporales, en cada prueba. Esto ha contribuido en gran medida a agilizar la implementación y depuración de las técnicas de ajuste de hiperparámetros. Por estos motivos “históricos” y para contribuir a facilitar las pruebas de futuros desarrollos igual que lo ha hecho hasta el momento presente, se mantiene esta función entre los algoritmos cuya implementación incorpora Zygarde.

4.4 Formato y esquema de las peticiones

Las peticiones de entrenamiento de Zygarde, independientemente del canal del que provengan, consisten en un documento JSON que responde al esquema que puede encontrarse en el Apéndice A, especificado mediante JSON Schema. A continuación se procede a comentar la estructura que deben seguir los documentos con peticiones de entrenamiento, de acuerdo con los elementos presentes en la definición de dicho esquema.

En primer lugar se determina el dominio del problema, que acota los algoritmos disponibles para su resolución y la métrica utilizada, así como ciertos aspectos relativos al procesamiento del *dataset*,

En segundo lugar se puede ver cómo se permite indicar la ruta (remota) en la que se encuentra el *dataset*, admitiendo los protocolos HTTP y HTTPS así como el prefijo s3a, que referencia directamente a recursos almacenados en Amazon S3, el sistema de almacenamiento de objetos de AWS.

En tercer lugar se señalan los algoritmos que se desea ejecutar, y que deberán pertenecer al dominio especificado. Para cada algoritmo se proporciona un conjunto de hiperparámetros, asociando a cada uno de ellos los valores que se desean probar. Por simplicidad, Zygarde sólo admite la definición de valores categóricos, tanto numéricos como textuales. La posibilidad de definir el espacio de búsqueda mediante valores continuos, rangos o incluso distribuciones de probabilidad es una característica deseable que no se ha incluido en Zygarde, debido a que su implementación entraña una complejidad que habría excedido el alcance acotado para el trabajo, pero que se ha considerado y se hace constar como una de las líneas de trabajo futuro.

En cuarto lugar se pueden especificar características relativas a los recursos computacionales a aprovisionar, pudiendo especificar modelos de instancia de Amazon EC2¹⁷, definir límites inferiores y superiores sobre el número de instancias de cada tipo a aprovisionar y también indicar si dicho tipo de instancia debe incluir algún hardware de propósito específico, como una GPU. La definición de los recursos de cómputo se puede realizar tanto con el nivel de detalle ya descrito o simplemente estableciendo límites inferiores y superiores sobre el número de instancias de cómputo que conformarán el clúster sobre el que se ejecutarán

¹⁷Amazon EC2 Instance Types - Amazon Web Services, <https://aws.amazon.com/ec2/instance-types/>

las tareas, permitiendo que sea Zygarde quien determine el tipo de instancias de cómputo a utilizar. Dado que la inclusión de este elemento es opcional, se puede decidir no incluirlo en la petición, estableciendo así que la propia plataforma sea la que asuma la responsabilidad de determinar el número de instancias de cómputo que se desplegarán. Los aspectos relacionados con el aprovisionamiento y despliegue dinámico de recursos de cómputo en la infraestructura del proveedor *cloud* se tratan detalladamente en el Capítulo 5.

Finalmente y en quinto lugar se especifica la dirección de correo electrónico del usuario. Es a esta dirección a la que se enviará el informe con los resultados de las ejecuciones realizadas, clasificando cualitativamente las combinaciones de hiperparámetros de cada algoritmo e indicando las rutas desde las que el usuario puede descargar los modelos generados.

Capítulo 5

Arquitectura en la nube

UNA vez presentado el diseño de alto nivel de la plataforma, habiendo descrito su comportamiento, la interacción entre sus componentes y su oferta algorítmica, en este capítulo se expone la arquitectura del sistema. Dicha arquitectura se ha concebido desde su planteamiento inicial para adecuarse al despliegue sobre la infraestructura y los servicios de un proveedor de *cloud* público, habiendo optado en este caso por Amazon Web Services.

A lo largo del capítulo se vuelven a estudiar los componentes en los que se divide la implementación del sistema, ya presentados en la Sección 4.1, pero esta vez desde una perspectiva enfocada en su modelo de despliegue. En este contexto, se tratan aspectos tales como el entorno de ejecución de cada componente, su estrategia de replicación o las medidas adoptadas para proporcionar tolerancia a fallos. Así, se pretende complementar la información ofrecida en el Capítulo 4 y plasmar una visión completa de la plataforma, ya instaurada sobre su infraestructura de despliegue. Buscando destacar el papel que cada componente individual juega en la colectividad del sistema, de nuevo se enfatiza el modelo de comunicación entre componentes adoptado en cada situación. En el transcurso de esta argumentación se alude a los principios sobre los que se sustenta el planteamiento de la arquitectura, que son extrapolables al diseño de cualquier arquitectura de calidad dirigida a aplicaciones distribuidas que hayan de operar en un escenario similar.

Merecen especial atención los procedimientos de aprovisionamiento y liberación dinámicos de recursos ante la entrada de peticiones en el sistema, los cuales se ajustan en función de las peticiones en curso y del volumen de carga de trabajo que puedan suponer. Asimismo, se justifica la elección entre el uso de servicios específicos del proveedor *cloud* (*PaaS*, *Platform as a Service*) o la orquestación propia del despliegue directamente sobre instancias de cómputo en la forma de máquinas virtuales (*IaaS*, *Infrastructure as a Service*). Con ello, se pretende equilibrar la relación entre la complejidad de la solución, su coste económico y la portabilidad entre las plataformas de diferentes proveedores de *cloud* público.



5.1 Arquitectura de despliegue sobre Amazon Web Services

Los componentes del sistema, que se expusieron en el Capítulo 4 desde una perspectiva funcional, se presentan en esta sección desde la perspectiva de su despliegue sobre un proveedor de *cloud* público, que en este caso es Amazon Web Services. Mientras que la Figura 4.1 mostraba los componentes de una forma muy esquemática, únicamente reflejando los flujos de información de interacción entre ellos, la Figura 5.1 representa estos componentes ya en el contexto de una arquitectura de aplicación en la nube, utilizando la simbología apropiada en función de los servicios del proveedor *cloud* que se emplean en cada caso. Las particularidades del planteamiento y el despliegue de cada uno de estos componentes se desglosan a lo largo de las siguientes secciones, siguiendo un orden análogo al de la Sección 4.1, en la que se describió su propósito y su modelo de funcionamiento.

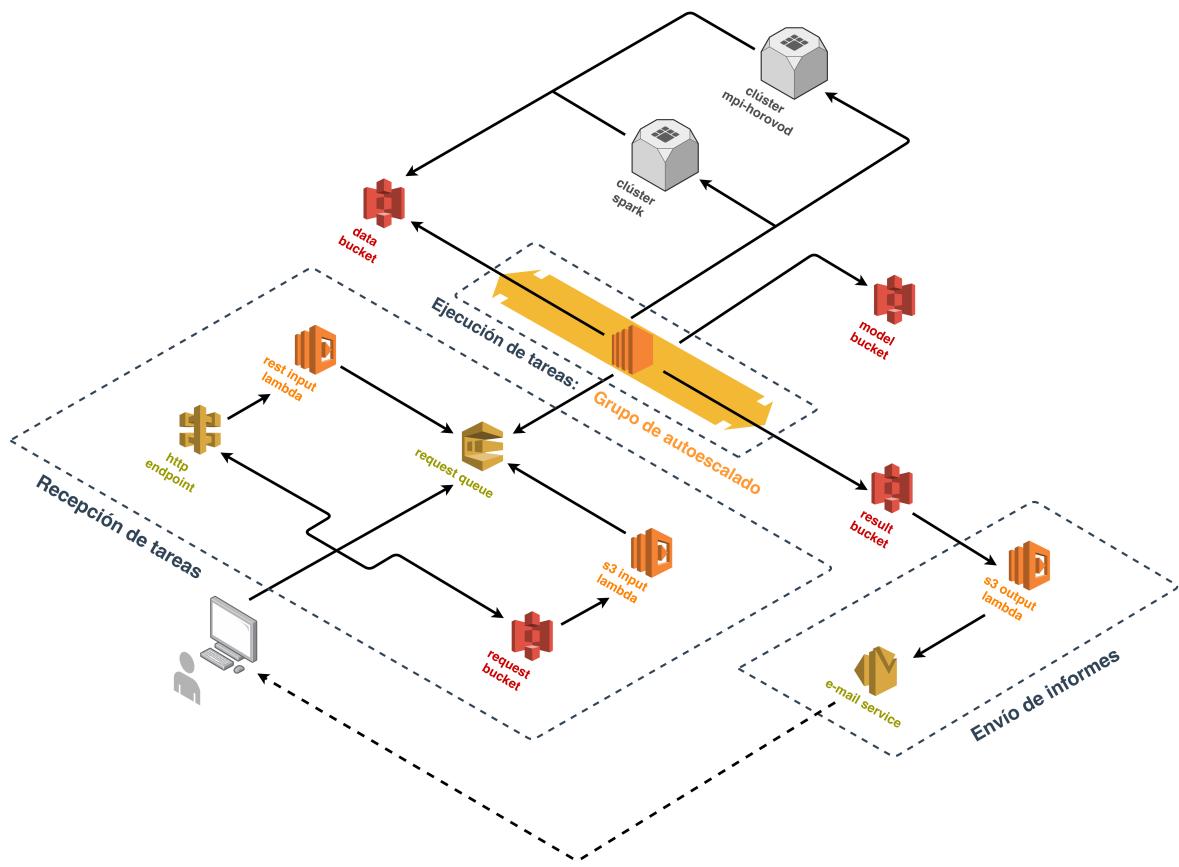


Figura 5.1: Arquitectura de despliegue sobre AWS

5.1.1 Recepción de tareas

El componente de recepción de tareas actúa como un centralizador para la recepción de peticiones de entrenamiento, tal como se describió en la Subsección 4.1.1 desde la perspectiva de su diseño funcional. De acuerdo con la especificación de requisitos funcionales de la Subsección 3.1.1, las peticiones de los usuarios pueden provenir de tres fuentes diferentes: como un mensaje dirigido a una cola de mensajería direccional, a través de un *endpoint* de una API REST y mediante el depósito de la petición en el sistema de almacenamiento de objetos del proveedor *cloud*. A continuación se detalla cómo cada uno de estos puntos de entrada de peticiones se ha construido sobre la infraestructura de Amazon Web Services. A lo largo de la explicación se referencia a la Figura 5.2, en la que se muestran gráficamente los elementos que conforman la arquitectura de despliegue del sistema.

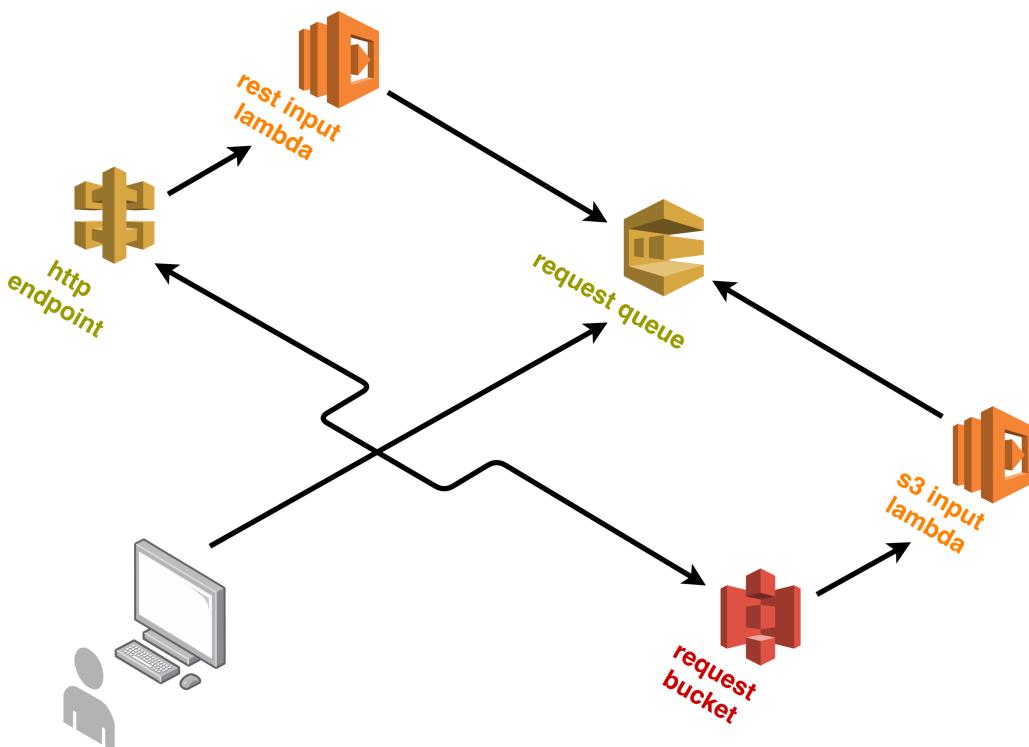


Figura 5.2: Arquitectura del componente de recepción de tareas

La recepción de peticiones directamente en una cola de mensajería se ha implementado mediante el servicio Amazon SQS¹ (*Simple Queue Service*). Este servicio proporciona colas de mensajes gestionadas por el proveedor *cloud*, ofreciendo características personalizables tales como ventanas temporales de visibilidad de los mensajes y la readmisión automática de un mensaje si su procesamiento no finaliza en un periodo estipulado por el administrador de la cola. Además, el servicio incluye dos tipos de colas: estándar y FIFO, en función de

¹Amazon Simple Queue Service (SQS), <https://aws.amazon.com/sqs/>

las garantías que requiera el usuario de acuerdo a la idempotencia en el procesamiento de los mensajes y a su ordenación, especialmente si hay relaciones de causalidad o temporales entre ellos. La utilización de colas de mensajes como medio de comunicación entre los componentes de un sistema distribuido resulta especialmente conveniente en aplicaciones *serverless* y orientadas a microservicios, puesto que permite desacoplar dichos componentes y escalarlos por separado. Esto conlleva dos ventajas esenciales. En primer lugar, posibilita maximizar la productividad global del sistema al aliviar potenciales potenciales cuellos de botella, a la par que en cada componente individual sólo se aplica el factor de escalado que resulte necesario en función de su carga de trabajo en cada momento y su criticidad, optimizando el uso de los recursos. En segundo lugar, este desacoplamiento y la asincronía en las comunicaciones que proporciona evitan que un fallo en un componente se propague en cascada a otros elementos del *workflow*, sin necesidad de implementar medidas específicas (como el patrón *Circuit Breaker*) para tolerar este tipo de fallos. [Nyg18]

Aunque es posible enviar directamente un mensaje a una cola de SQS a través de una petición HTTP dirigida a la URL de la cola, este mecanismo no se ha utilizado como tal en Zygarde, debido a que habría supuesto la inclusión de mensajes en la cola sin validar previamente su adecuación y conformidad respecto a la estructura definida para los mensajes de solicitud de entrenamiento. Además, para realizar esta operación es necesario incluir en los parámetros de la petición ciertas credenciales de acceso a la cola, las cuales no corresponde a los usuarios poseer. Por estos motivos, la inclusión directa de mensajes en la cola desde el exterior del sistema se ha restringido a la batería de programas creados para facilitar las pruebas del desarrollo, que sí tienen la potestad de enviar mensajes a la cola sin intermediarios, utilizando las credenciales necesarias. En este caso, el acceso a la cola de mensajes se realiza a través del SDK de Amazon SQS para Python, más cómodo que la petición manual de peticiones HTTP.

La inyección de mensajes en la cola a partir de una petición HTTP se ha implementado mediante la combinación del servicio Amazon API Gateway² y una función Lambda. Mediante API Gateway se ha expuesto una API REST con un *endpoint* que admite la recepción de una operación POST, en cuyo cuerpo debe incluirse la solicitud de entrenamiento en formato JSON, de acuerdo al esquema especificado en el Apéndice A. Cuando se recibe una petición en dicho *endpoint*, automáticamente se ejecuta una función Lambda a la que se le proporciona el cuerpo de la petición. Esta función comprueba que el contenido de la petición es un documento JSON válido y que su estructura concuerda con el esquema esperado. Si es así, procede a componer un mensaje con la solicitud de entrenamiento y a enviarlo a la cola de mensajes, utilizando para ello el SDK de Amazon SQS para Python. La combinación de estos servicios permite implementar el control y la verificación de los mensajes recibidos a través de una petición HTTP. Además, al operar de acuerdo al paradigma *serverless* se

²Amazon API Gateway, <https://aws.amazon.com/api-gateway/>

evita tener un servidor permanentemente dedicado a la recepción de peticiones, con lo que la facturación tiene lugar en base al número de peticiones recibidas y al tiempo total de ejecución de la función Lambda. Igualmente, se evita asumir ciertas problemáticas que pudieran derivarse de la gestión de una API REST accesible desde el exterior del sistema, tales como el control de la congestión o el soporte a CORS (*Cross-Origin Resource Sharing*), puesto que éstas se delegan en el servicio API Gateway.

La recepción de peticiones a través del depósito de un documento con la solicitud de entrenamiento en una ubicación específica del sistema de archivos del proveedor *cloud* se ha implementado de forma parecida a la recepción de peticiones HTTP. En este caso, se ha implementado una función Lambda que se ejecuta automáticamente cuando se deposita un documento en un *bucket* específico de Amazon S3. Esta función es análoga a la ya descrita: comprueba la estructura del documento JSON y verifica su correspondencia con el esquema esperado, tras lo que compone un mensaje con el contenido de la petición de entrenamiento y lo inserta en la cola de mensajería, utilizando el SDK de Amazon SQS para Python. Aunque presumiblemente existen menos escenarios en los que resulte preferible para los usuarios utilizar esta vía para el envío de peticiones respecto a la API REST, permite mostrar cómo el sistema admite múltiples puntos de entrada para las peticiones con independencia de la naturaleza de cada uno de ellos, puesto que todas las peticiones terminan confluyendo en la cola de mensajes. En este contexto, puede interpretarse este componente de recepción de peticiones como una abstracción de la gestión de múltiples productores en un problema productor-consumidor distribuido, en el que el motor de ejecución de tareas de entrenamiento actúa como consumidor.

Ya habiendo explicado el funcionamiento de este componente y los elementos en los que se divide su arquitectura de despliegue en la nube, esta subsección finaliza argumentando la adopción del servicio Amazon SQS frente a la utilización de otras tecnologías como Apache Kafka³ o RabbitMQ⁴. Estas tecnologías también proporcionan sistemas de colas de mensajes que permiten ofrecer garantías de alta disponibilidad. Sin embargo, su utilización habría implicado el despliegue de instancias de cómputo dedicadas a sostener el sistema de colas, con un número suficiente de réplicas como para proporcionar un nivel adecuado de tolerancia a fallos, lo que supone un coste económico a tener en cuenta. Además, el despliegue y la administración de estos sistemas de colas de mensajes entrañan una complejidad que habría sido necesario asumir. Por el contrario, Amazon SQS releva al administrador del sistema en la tarea de lidiar con la complejidad asociada al sistema de colas subyacente. Al tratarse de un servicio PaaS gestionado por el proveedor *cloud*, permite ofrecer garantías de alta disponibilidad conforme a un acuerdo de nivel de servicio sin necesidad de desplegar instancias

³Apache Kafka, <https://kafka.apache.org>

⁴RabbitMQ - Messaging that just works, <https://rabbitmq.com>

de cómputo ni interactuar directamente con la infraestructura. La facturación se realiza en términos del número de mensajes insertados en la cola, de acuerdo con un modelo de pago por uso y no un modelo de pago por despliegue, como habría tenido lugar en caso de instalar uno de los sistemas de colas mencionados sobre un conjunto de instancias de cómputo aprovisionadas explícitamente.

Desde el punto de vista de la portabilidad entre plataformas, los principales proveedores de *cloud* público proporcionan sus propios servicios gestionados de colas de mensajes, como Google Cloud Pub/Sub⁵, y Azure Queue Storage⁶, que proporcionan sus propios SDKs y son similares a Amazon SQS. Gracias a esta oferta de servicios, la migración entre plataformas (o la constitución de un entorno multi-nube) puede llevarse a cabo utilizando el servicio específico de cada plataforma y proporcionando diferentes *drivers* o implementaciones de la interfaz de operación con el servicio, sin necesidad de modificar el resto de los componentes del sistema.

En base a esta información, puede concluirse que el uso de sistemas de colas como Apache Kafka o RabbitMQ al diseñar una arquitectura de aplicación en la nube tiene sentido en dos situaciones: bien cuando se opera sobre una infraestructura *cloud on-premises* o con un proveedor *cloud* que no proporciona un servicio de colas de mensajería, o bien cuando el volumen de mensajes a asumir es tan desmesurado que económicamente se justifica la adopción de un modelo de pago por despliegue frente a un modelo de pago por uso. En el escenario de operación de Zygarde no se cumple ninguna de estas dos condiciones, mientras que puede beneficiarse de todas las ventajas que se han expuesto y que derivan de contar con un servicio gestionado por el proveedor *cloud*. Por estos motivos, y sumándose a la justificación que ya se ha presentado en relación al uso de colas de mensajes para la comunicación entre diferentes servicios del sistema, se decide que la arquitectura del componente de recepción de peticiones se erija en torno a una cola de mensajería sustentada en el servicio Amazon SQS.

5.1.2 Ejecución de tareas

El motor de ejecución de tareas actúa como orquestador del procesamiento de las solicitudes de entrenamiento, tal como se describió en la Subsección 4.1.2 desde la perspectiva de su diseño funcional. Tal como se detalla en dicha subsección, se trata del componente más complejo de implementación propia del sistema y las responsabilidades que abarca son la transformación de las peticiones de los usuarios en ejecuciones específicas de algoritmos de aprendizaje automático, la monitorización de dichas ejecuciones sobre el clúster de cómputo

⁵Cloud Pub/Sub | Global messaging and event ingestion made simple, <https://cloud.google.com/pubsub>

⁶Queue Storage | Durable queues for large-volume cloud services,
<https://azure.microsoft.com/en-us/services/storage/queues/>

intensivo correspondiente, la recolección de los resultados y los modelos generados, su ordenación mediante la elaboración de una clasificación basada en una métrica de su calidad y la confección del informe que será remitido al usuario mediante el componente de envío de informes. A continuación se explica cómo este componente se ha construido sobre la infraestructura de Amazon Web Services, a la par que se detalla cuáles han sido los mecanismos de comunicación utilizados en la interacción con los restantes componentes y servicios con los que se interconecta, justificando y argumentando las decisiones de diseño tomadas en cada situación. A lo largo de la explicación se referencia a la Figura 5.3, en la que se muestran gráficamente los elementos que conforman la arquitectura de despliegue del componente y los flujos de interacción entre ellos.

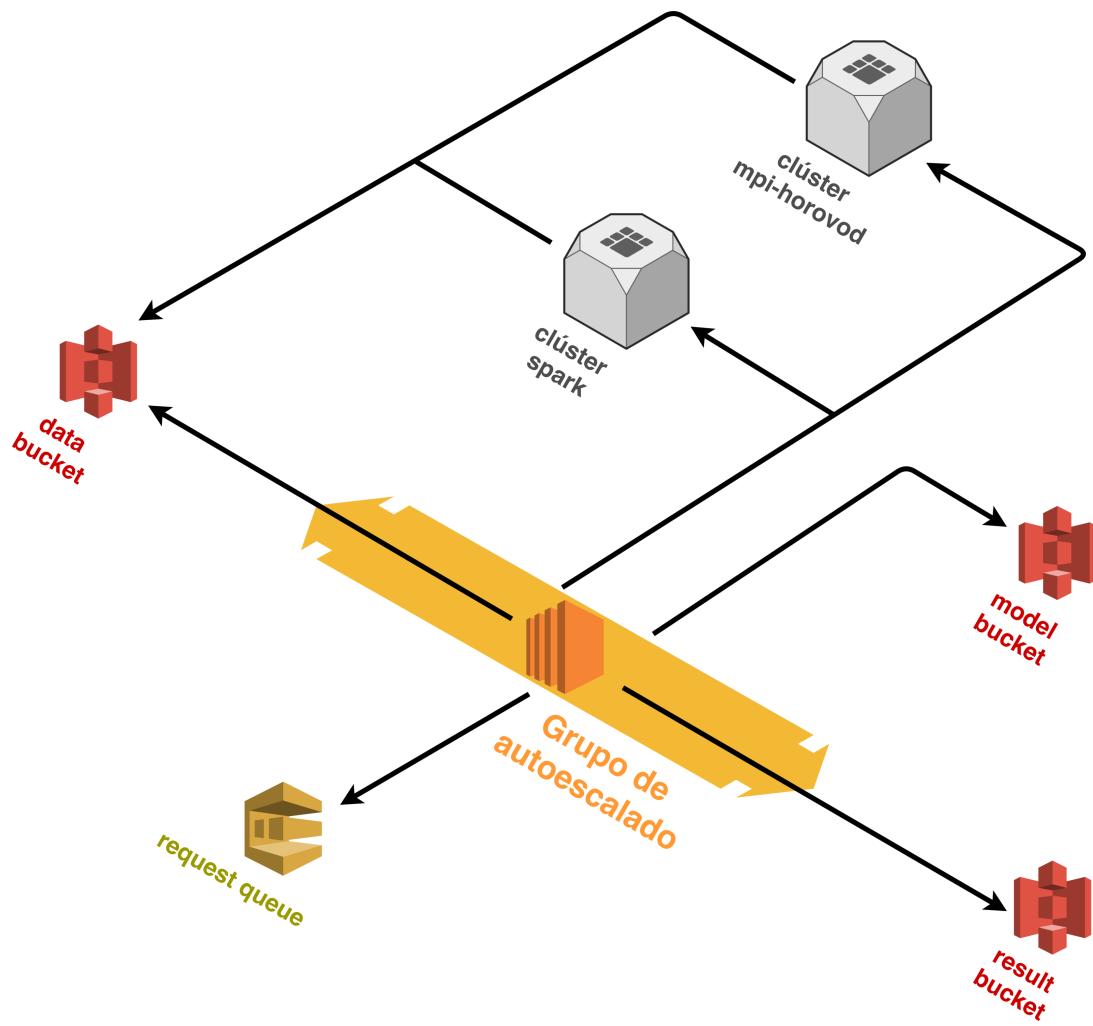


Figura 5.3: Arquitectura del motor de ejecución de tareas

A diferencia de los restantes componentes de implementación propia, el motor de ejecución de tareas no se estructura sobre una función Lambda o mediante la combinación de una función Lambda con otros servicios. Este componente se implementa como un proceso de larga duración y ejecución continua que extrae mensajes de la cola de Amazon SQS y los procesa de acuerdo al modelo de funcionamiento presentado en la Subsección 4.1.2. Dicho proceso se ejecuta sobre una máquina virtual aprovisionada mediante el servicio Amazon EC2, adoptando un modelo de despliegue tradicional frente al enfoque orientado al paradigma *serverless* descrito anteriormente. El principal motivo por el que el motor de ejecución de tareas se ha planteado de esta forma, requiriendo una unidad de despliegue permanentemente activa para poder operar, es una restricción existente en las funciones Lambda de Amazon Web Services que limita la duración de la ejecución de una función a quince minutos⁷. Si se considera la necesidad de contar con un proceso o hilo de ejecución que coordine todas las ejecuciones de algoritmos de aprendizaje automático que tienen lugar en el transcurso del procesamiento de una petición, el tiempo de ejecución del proceso muy probablemente supere este lapso, especialmente si se emplean técnicas de optimización bayesiana para el ajuste de los hiperparámetros, en cuyo caso las ejecuciones de algoritmos tienen lugar secuencialmente, no en paralelo. Pero incluso si el alcance de una hipotética función Lambda comprendiera únicamente la ejecución de un algoritmo de aprendizaje automático con una determinada configuración de hiperparámetros, el tiempo necesario para completar el entrenamiento puede superar fácilmente este umbral, imposibilitando el uso de una función Lambda con este fin. Por este motivo, en este contexto actualmente no resulta factible coordinar y monitorizar las ejecuciones de los algoritmos de aprendizaje automático, ya sea a nivel colectivo o individual, mediante funciones Lambda. Sin embargo, esta restricción no es inherente a la tecnología sobre la que se sustentan las funciones Lambda (contenedores de procesos), sino parece responder principalmente a un modelo de negocio, por lo que se espera que el servicio llegue a flexibilizar estas restricciones con el paso del tiempo. [JSSS^{+19]}

Al adoptar un modelo de despliegue tradicional, haciendo uso directamente de máquinas virtuales emplazadas sobre la infraestructura del proveedor *cloud*, deben gestionarse aspectos que antes permanecían ocultos, enmascarados por los servicios de alto nivel del proveedor *cloud*, tales como el aprovisionamiento de las máquinas sobre las que realizar el despliegue y los mecanismos con los que proporcionar tolerancia a fallos. En esta coyuntura, se ha optado por disponer varias réplicas del componente en un grupo de autoescalado, el cual hace uso de una base de reglas para adaptarse dinámicamente al consumo de recursos que estén asumiendo las instancias desplegadas, de acuerdo con la propiedad denominada “elasticidad”. El número de réplicas oscila entre los umbrales establecidos como mínimo y máximo en función de si es conveniente añadir más instancias de cómputo o suprimir alguna de las ya desplegadas. Igualmente, cada instancia de cómputo es monitorizada en todo momento

⁷AWS Lambda limits - AWS Lambda,
<https://docs.aws.amazon.com/lambda/latest/dg/gettingstarted-limits.html>

mediante el servicio Amazon CloudWatch⁸, lo que permite detectar si una máquina virtual ha dejado de responder, para en tal caso eliminarla y desplegar automáticamente otra que ocupe su lugar.

Puesto que el proceso de despliegue e inicialización de una máquina virtual no es inmediato, sino que se sitúa en el orden de varios minutos, debe tenerse en cuenta este lapso a la hora de establecer cómo se desplegarán nuevas instancias de cómputo. De igual forma, desde que una máquina virtual en ejecución falla hasta que este fallo es detectable (si llega a serlo) y se decide desplegar una nueva máquina virtual que la sustituya, también pueden pasar minutos. Para evitar que en este lapso no haya ninguna instancia de cómputo preparada para asumir las peticiones entrantes, se han tomado las siguientes medidas. En primer lugar, se ha establecido el umbral mínimo de escalado horizontal en dos réplicas, de forma que el fallo de una de ellas no condicione la disponibilidad del sistema, debiendo para ello producirse al menos dos fallos simultáneos. En segundo lugar, cuando una nueva máquina virtual se despliega, este despliegue no tiene lugar a partir de una imagen limpia del sistema operativo desde la que se instala todo el software necesario sobre la marcha, utilizando herramientas como Ansible o Puppet. Por el contrario, se ha generado una imagen canónica de máquina virtual o *Golden AMI (Amazon Machine Image)* que ya cuenta con todo el software y la configuración requerida, a partir de la que se inician las nuevas máquinas virtuales, reduciendo sustancialmente el tiempo de preparación en comparación con un escenario en el que la labor de instalación y configuración del software tuviese lugar durante el propio proceso de despliegue. Para facilitar el mantenimiento de estas AMIs, Amazon Web Services ofrece herramientas^{9,10} que ayudan a automatizar la gestión de su ciclo de vida aplicando procedimientos de integración continua.

Habiendo ya descrito el modelo de despliegue del componente, se aborda su interacción con otros componentes y elementos del sistema durante su funcionamiento. Las réplicas del motor de ejecución de tareas extraen mensajes de la cola de Amazon SQS, los procesan (llevando a cabo todo el procedimiento descrito en la Subsección 4.1.2) y, una vez completado este proceso, los marcan en la cola de mensajes para su eliminación. La operación de eliminación es necesaria debido a la política de reintentos que implementan las colas de mensajes de SQS. Cuando una réplica extrae un mensaje de la cola, este mensaje deja de estar disponible para su recuperación por parte de otras réplicas durante un periodo de tiempo; si pasado este periodo el mensaje no se ha marcado para ser eliminado, se asume que su procesamiento ha fallado y vuelve a estar disponible para ser recuperado de nuevo. Tanto el periodo tras el que el mensaje volverá a ser visible en la cola como el número máximo de reintentos

⁸Amazon CloudWatch - Observability of your AWS resources and applications on AWS and on-premises, <https://aws.amazon.com/cloudwatch/>

⁹EC2 Image Builder | Build and maintain secure images, <https://aws.amazon.com/image-builder/>

¹⁰Automate OS Image Build Pipelines with EC2 Image Builder, <https://aws.amazon.com/blogs/aws/automate-os-image-build-pipelines-with-ec2-image-builder/>

y el tiempo que un mensaje puede permanecer en la cola sin ser procesado son parámetros configurables desde la administración del servicio Amazon SQS. Esta política parametrizable de reintentos automáticos, junto con la estrategia de replicación y autoescalado ya presentada, permiten dotar de tolerancia a fallos al motor de ejecución de tareas, con el objetivo de que si una réplica del componente falla durante el procesamiento de uno o varios mensajes, las correspondientes solicitudes de los usuarios no se pierdan, sino que más adelante sean recuperadas de nuevo y procesadas por otras réplicas.

Tal como se enuncia en la subsección anterior, el uso de una cola de mensajes como medio de comunicación entre el componente de recepción de tareas y el motor de ejecución de las mismas permite que esta interacción se realice en términos de un modelo productor-consumidor con múltiples productores (cada uno de los posibles orígenes de las solicitudes de los usuarios) y múltiples consumidores (las réplicas del motor de ejecución de tareas). Este modelo de comunicación, inherentemente asíncrono, favorece el desacoplamiento de los agentes implicados, permitiendo gestionarlos y actuar sobre ellos de forma independiente. Su principal desventaja reside en la necesidad de disponer de mecanismos para el control de la concurrencia en el acceso a un recurso compartido como lo es la cola de mensajes (algo aún más complejo en el escenario de un sistema distribuido que en la comunicación entre procesos de un mismo sistema local). Afortunadamente, las tecnologías y servicios de colas de mensajes incluyen de serie estos mecanismos, convenientemente optimizados, por lo que tanto productores como consumidores puedan operar con la cola de mensajes sin coordinarse con los restantes agentes implicados ni tener siquiera constancia de su presencia.

Una vez que el componente ha recuperado de la cola de mensajes una solicitud de entrenamiento, se comprueba si la ruta del *dataset* que figura en ella corresponde a una ubicación de Amazon S3. Si no es así, se procede a descargar los datos y almacenarlos en un *bucket* de S3, utilizando clases orientadas a flujo de datos que permiten realizar la transferencia sin almacenar los *datasets* en el sistema de archivos local de la máquina virtual. El propósito de esta transferencia de datos es agilizar el proceso de entrenamiento, al reducir la latencia en el acceso a los datos desde los nodos en los que se ejecutan los algoritmos de aprendizaje automático, los cuales se despliegan sobre la infraestructura del proveedor *cloud*.

Seguidamente, el motor de ejecución de tareas procede a comandar el despliegue de un clúster de cómputo intensivo, cuya tecnología (Spark o Horovod) se determina en función del dominio del problema, tal como se indica en la Sección 4.3. Las prestaciones del clúster dependen de los límites y características especificados por el usuario en la petición de entrenamiento. Una vez que el clúster se halla listo y en disposición de recibir tareas, el conjunto de hilos desplegados por el componente solicita la ejecución de los algoritmos de aprendizaje automático con determinadas configuraciones de hiperparámetros, cuya selección se describe pormenorizadamente a lo largo del Capítulo 4. Una vez que la totalidad de las ejecuciones de algoritmos de aprendizaje automático asociadas a la petición del usuario ha finalizado,

el motor de ejecución de tareas ordena la detención y el desmantelamiento del clúster y prosigue su labor elaborando un informe en el que se incluye una clasificación ordenada obtenida de los resultados alcanzados. El procedimiento mediante el que se ordena el despliegue de un clúster de cómputo intensivo de una determinada tecnología sobre la infraestructura del proveedor *cloud*, se espera a que este clúster se encuentre preparado para ejecutar tareas, se obtiene la cadena de conexión al nodo maestro y en última instancia se ordena la detención del clúster, se trata con el adecuado nivel de detalle en las Secciones 5.2 y 5.3, puesto que se trata de un proceso complejo cuyo estudio merece abordarse individualizadamente.

Finalmente, una vez que la petición de entrenamiento de un usuario se ha procesado con éxito y se ha generado un informe con los resultados de las ejecuciones de algoritmos de aprendizaje automático realizadas, éste debe remitirse al usuario. Para ello, la interacción entre el motor de ejecución de tareas y el componente de envío de informes se realiza mediante el depósito de un documento, que contiene el informe con los resultados de las ejecuciones realizadas en el contexto de la petición del usuario, en un *bucket* de Amazon S3. La acción de almacenar un elemento en este *bucket* automáticamente genera un evento ante el que el componente de envío de informes reacciona recuperando el documento y realizando las acciones necesarias para remitírselo al usuario. Este modelo de comportamiento y la justificación del mecanismo elegido para la comunicación entre los dos componentes se trata en detalle en la Subsección 5.1.3, que describe la arquitectura de despliegue del componente de envío de informes.

5.1.3 Envío de informes

El componente de envío de informes se encarga de remitir a los usuarios el resultado de la ejecución de su petición de entrenamiento, una vez que el procesamiento de dicha petición ha concluido, tal como se describió en la Subsección 4.1.3 desde la perspectiva de su diseño funcional. Los informes se envían al usuario mediante un correo electrónico a la dirección indicada en la petición de entrenamiento. En estos informes figuran los algoritmos y combinaciones de valores de los hiperparámetros que se han probado, junto con el resultado obtenido para cada una de estas ejecuciones en términos de la métrica utilizada según el dominio del problema. Las ejecuciones aparecen ordenadas descendentemente en función de su resultado, permitiendo identificar rápidamente la combinación de algoritmo e hiperparámetros que ha alcanzado un mejor resultado. Para cada ejecución de un algoritmo con una determinada configuración de hiperparámetros se proporciona un enlace a la ubicación de Amazon S3 desde la que el usuario puede descargar el modelo generado, por si desea importarlo e integrarlo directamente en sus aplicaciones. En caso contrario, el usuario puede optar por identificar la configuración que mejor resultado ha logrado y utilizar esta información para construir el modelo por sus propios medios, con el *framework* y las herramientas de su preferencia. A continuación se detalla cómo este componente se ha construido sobre

la infraestructura y los servicios de Amazon Web Services. A lo largo de la explicación se referencia a la Figura 5.4, en la que se muestran gráficamente los elementos que conforman la arquitectura de despliegue del componente y los flujos de interacción entre ellos.

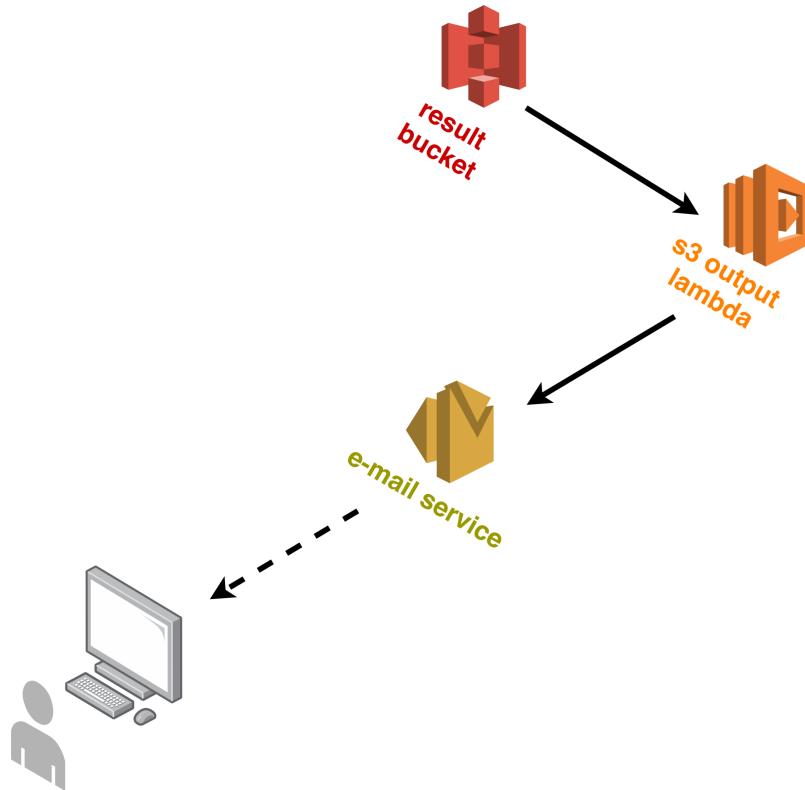


Figura 5.4: Arquitectura del componente de envío de informes

Cuando el motor de ejecución de tareas finaliza la coordinación y monitorización de una serie de ejecuciones de algoritmos de aprendizaje automático asociados a una misma petición de entrenamiento, construye una clasificación ordenada de las ejecuciones realizadas, a partir de la que compone el informe a remitir al usuario. Dicho informe se deposita en un *bucket* de Amazon S3 que no se encuentra públicamente accesible, haciendo figurar en el nombre del documento el identificador de la petición y en su ruta de acceso (en el contexto del sistema de archivos de S3) la dirección de correo del usuario. Al depositar el documento con el informe en ese *bucket* específico, automáticamente se ejecuta una función Lambda que recupera el contenido del informe y la dirección del correo del usuario, a partir de las que redacta un mensaje de correo electrónico que se envía al usuario a través del servicio Amazon SES¹¹, haciendo uso para ello del SDK de AWS para Python.

¹¹Amazon Simple Email Service (Amazon SES), <https://aws.amazon.com/ses/>

Mientras que los motivos para separar el componente de envío de informes respecto al motor de ejecución de tareas ya se expusieron en la Subsección 4.1.3, el hecho de que la comunicación entre estos componentes se realice mediante el depósito de un documento en Amazon S3 responde a la voluntad de operar de acuerdo a un modelo reactivo dirigido por eventos [Luc02] siempre que sea posible. En este caso, la presencia de una función Lambda que se encarga de recuperar el contenido del informe, construir un mensaje de correo electrónico a partir de él y remitírselo al usuario a través de un servicio de envío de correo evita tener que desplegar una instancia de cómputo para mantener esta funcionalidad. De esta forma, se opera de acuerdo a un modelo de pago por uso real, y la adopción del servicio FaaS (*Function as a Service*) del proveedor *cloud* permite abstraer cuestiones tales como el reintento de la ejecución de una función que hubiera fallado con anterioridad o la estrategia de replicación del componente.

El uso de Amazon SES, el servicio de envío de correo electrónico de AWS, en lugar de contar con un servidor SMTP dedicado, obedece a motivos similares a los esgrimidos en la Subsección 5.1.1 para justificar el uso de Amazon SQS frente a otras tecnologías de colas de mensajes. La adopción de este servicio permite liberar al administrador del sistema de las problemáticas derivadas de la gestión de un servidor de correo propio. Además, dicho servidor debería desplegarse sobre una instancia de cómputo previamente aprovisionada (y posiblemente replicada para proporcionar tolerancia a fallos), mientras que el servicio del proveedor *cloud* evita esta necesidad, transicionando una vez más de un modelo de pago por despliegue a un modelo de pago por uso real. Resulta oportuno mencionar que Amazon SES permite la operación con el servicio tanto mediante el SDK de AWS como a través de una interfaz SMTP. En caso de desarrollar la integración con el servicio sobre ésta última y que, por motivos administrativos o ligados al volumen de correos a enviar, en algún momento se decidiera pasar a utilizar un servidor de correo propio, la transición entre servicios sería inmediata, pues para ello bastaría con tan solo actualizar la dirección del servidor SMTP en la función Lambda.



5.2 Clúster Spark

Una de las dos plataformas utilizadas para la ejecución distribuida de algoritmos de aprendizaje automático es Spark. En concreto, se utiliza la biblioteca Spark MLlib en tareas de clasificación binomial y multinomial, regresión y *clustering*, como se trata en detalle en la Sección 4.3. Para proceder a la ejecución de un algoritmo de aprendizaje automático mediante esta tecnología, es necesario disponer de un clúster Spark operativo, a cuyo nodo maestro se conectará la aplicación que solicite la ejecución del algoritmo.

Tal como se apunta en las Subsecciones 4.1.2 y 5.1.2, desde la perspectiva del diseño funcional del sistema y de su arquitectura de despliegue en la nube, respectivamente, cuando el motor de ejecución de tareas procesa la petición de un usuario, comprueba el dominio al que pertenece el problema planteado en la petición para determinar la tecnología que se empleará en su resolución. En caso de que el dominio del problema corresponda a los algoritmos proporcionados por Spark MLlib, se procede a desplegar un clúster Spark sobre la infraestructura del proveedor *cloud*, sobre el que se realizarán múltiples ejecuciones del algoritmo o algoritmos de aprendizaje automático indicados en la petición, en base a la estrategia seleccionada para la exploración del espacio de los hiperparámetros. La topología del clúster y las características de las instancias de cómputo aprovisionadas se deciden dinámicamente en base a las restricciones expresadas por el usuario en la petición de entrenamiento.

En la presente sección se describe el proceso mediante el que se coordina el despliegue y la configuración automatizada de un clúster Spark sobre Amazon EC2, incluyendo la gestión del ciclo de vida completo del clúster. Asimismo, se describe cómo tiene lugar la interacción entre un conjunto de hilos de ejecución del motor de tareas de Zygarde y un clúster Spark ya desplegado. Al final de la explicación se hace referencia a las opciones tecnológicas cuya inclusión se ha sopesado para facilitar esta labor, junto con los motivos por los que cada una de ellas ha terminado siendo adoptada o descartada.

5.2.1 Orquestación de clústeres efímeros

El escenario de operación de Zygarde suma varias dificultades adicionales a la complejidad inherente al despliegue de un clúster Spark sobre Amazon Web Services o cualquier otro proveedor *cloud*. Éstas son consecuencia del hecho de que Zygarde no opere sobre un clúster desplegado manualmente con anterioridad y concebido para encontrarse en funcionamiento durante un periodo de tiempo indefinido hasta que los administradores del sistema decidan detenerlo. Por el contrario, la propuesta de Zygarde es la operación con clústeres efímeros, que se despliegan bajo demanda y se eliminan una vez han cumplido su propósito. Además, la topología de estos clústeres y las características de las máquinas sobre las que se emplazan no están predefinidas, sino que se determinan en el momento del despliegue en base a las restricciones especificadas por el usuario en la petición de entrenamiento. Otro inconveniente

proviene del hecho de que el ciclo de vida del clúster se controla desde el código fuente del motor de ejecución de tareas de Zygarde, lo que obliga a integrar su gestión y monitorización en el flujo de ejecución del componente.

Las decisiones de diseño que conducen a este planteamiento obedecen a un doble propósito. Por una parte, se aprovecha la disponibilidad de recursos computacionales en la nube para garantizar que cada petición se ejecutará en un clúster dedicado en exclusiva a ella, no teniendo que competir con otras peticionar en curso por el uso de recursos compartidos. Por otra parte, se busca evitar el innecesario desembolso económico que supondría mantener recursos computacionales que no se están utilizando (como un clúster permanentemente disponible, incluso cuando no hay carga de trabajo en el sistema), mediante la transición a un modelo de despliegue bajo demanda en el que los recursos sólo se materializan cuando efectivamente se requieren. Estas dos razones obran de acuerdo con los principios de diseño que se ha tratado de mantener en el planteamiento de la arquitectura del sistema y en las que se ha incidido a lo largo de la Sección 5.1: el sistema debe ser capaz de asumir y dar respuesta a las peticiones de los usuarios adaptándose de forma autónoma a las variaciones en el volumen de la carga de trabajo, a la par que realiza una gestión inteligente de los recursos empleados con la finalidad de reducir el coste económico que se deriva de ellos.

El planteamiento descrito proporciona importantes ventajas frente a la operación sobre un clúster desplegado manualmente con una configuración y topología predefinidas, pero éstas vienen acompañadas de un alto grado complejidad añadida, puesto que su aplicación introduce una serie de problemáticas adicionales cuya resolución no es trivial. Para facilitar su gestión, el despliegue de clústeres Spark sobre el servicio de máquinas virtuales de Amazon Web Services se ha implementado a través de Flintrock, una herramienta de línea de órdenes que automatiza el proceso de despliegue y configuración de un clúster Spark a partir de un archivo de configuración o manifiesto de despliegue. Aunque actualmente esta herramienta sólo soporta el despliegue de clústeres Spark sobre Amazon EC2, su arquitectura está planteada de forma que más adelante pueda pasar a proporcionar soporte a otros proveedores *cloud*, utilizando una misma interfaz de órdenes y seleccionando el *back-end* correspondiente a partir de la información reflejada en el manifiesto de despliegue. Los motivos que han llevado a la elección de esta tecnología en particular, en detrimento de alternativas propias de AWS como Amazon EMR, se exponen ampliamente en la Subsección 5.2.3.

La siguiente subsección profundiza en la aplicación de estas propuestas y presenta la forma en la que Zygarde, a través de su motor de ejecución de tareas, gestiona el ciclo de vida de un clúster Spark, desde su despliegue hasta su eliminación. Ligado a este control de los recursos desplegados, también se estudia cómo se produce la interacción entre los hilos responsables de la ejecución de algoritmos de aprendizaje automático y el clúster.

5.2.2 Ciclo de vida y ejecución de tareas

Tal como se apunta en el encabezado de la sección, cuando el motor de ejecución de tareas despliega un nuevo hilo de ejecución para hacerse cargo del procesamiento de la petición de entrenamiento de un usuario, comprueba cuál es el dominio del problema para determinar la tecnología del clúster de cómputo intensivo que debe desplegar. En aquellos casos en los que el dominio de la tarea se sitúa entre los que se delegan en los algoritmos proporcionados por Spark MLlib, se procede a desplegar un clúster Spark sobre Amazon EC2 valiéndose para ello de la herramienta Flintrock.

Puesto que el proceso de despliegue y configuración del clúster tarda unos minutos hasta que éste se encuentra preparado para recibir tareas, se solapa con la transferencia del *dataset* a Amazon S3 (en caso de que no se encontrara ya almacenado en este sistema de archivos remoto), con el objetivo de aprovechar el tiempo de espera realizando simultáneamente varias acciones que no tienen una relación de precedencia entre sí. Tal como se comenta en la Subsección 5.1.2, la transferencia de archivos entre un origen y un destino remotos se implementa utilizando clases orientadas a flujo de datos que permiten llevarla a cabo sin necesidad de almacenar el *dataset* en el sistema de archivos de la máquina virtual en la que se ejecuta la réplica del motor de ejecución de tareas, evitando así que el espacio disponible en disco pudiera llegar a resultar insuficiente. Para lograr este solapamiento, se despliegan dos hilos concurrentes, uno de los cuales se encarga de desplegar el clúster Spark y otro que comprueba si el *dataset* está ya almacenado en S3 y, en caso negativo, procede a realizar la transferencia de datos.

Para realizar el despliegue del clúster Spark, el hilo de ejecución encargado de llevar a cabo esta labor compone en primer lugar un archivo YAML con el manifiesto de despliegue en el formato esperado por Flintrock, incluyendo información relativa a la configuración de AWS, la versión del software a instalar, el número de nodos y el tipo de instancias de cómputo de Amazon EC2 a aprovisionar. Tanto la topología del clúster como el tipo de las instancias de cómputo se determinan a partir de las restricciones especificadas por el usuario en la petición de entrenamiento, en caso de que haya proporcionado dicha información. Puesto que Flintrock se distribuye como una aplicación de línea de órdenes (instalable a través de pip, el gestor de paquetes por defecto de Python), la llamada a la herramienta se produce mediante el lanzamiento de un nuevo proceso desde la JVM (*Java Virtual Machine*). Entre los parámetros con los que se lanza dicho proceso se incluyen la ruta al manifiesto de despliegue y el identificador del clúster, el cual se asigna a partir del identificador de la petición de entrenamiento, facilitando su seguimiento.

Ante esta llamada, Flintrock se encarga de desplegar las instancias de cómputo correspondientes sobre el espacio de AWS asociado a la cuenta de usuario que figura en el manifiesto de despliegue, instalar el software necesario, configurar las claves SSH para que los nodos puedan comunicarse entre sí y conformar el clúster de tal forma que los nodos esclavos se

unan al nodo maestro, con lo que el clúster Spark pasa a encontrarse operativo. El despliegue del clúster se realiza sobre un VPC (*Virtual Private Cloud*, similar a una subred) creado previamente con este fin específico, y sobre el que se ha aplicado una política de seguridad que asegura que sus nodos sólo sean accesibles desde el exterior por parte de miembros del VPC en el que se engloba el grupo de autoescalado del motor de ejecución de tareas, protegiendo así los clústeres Spark ante accesos externos no autorizados. Si todos estos pasos se han llevado a cabo correctamente, el proceso imprime el nombre DNS del nodo maestro del clúster y concluye su ejecución.

Tras comprobar que el código de salida de Flintrock indica una finalización correcta, se analiza la salida estándar del proceso y se captura el nombre DNS del nodos maestro, que se utiliza para componer la ruta que permite lanzar tareas al clúster Spark desde una aplicación. El hilo de ejecución finaliza devolviendo al nivel previo esta cadena de texto. Una vez que tanto el hilo de ejecución que se ha encargado de desplegar el clúster como el que ha llevado a cabo la transferencia del *dataset* a Amazon S3 (en caso de que fuera necesario) han finalizado, desde su antecesor común se lanza un nuevo hilo de ejecución para cada uno de los algoritmos indicados en la petición de entrenamiento, tal como se explica detalladamente en la Subsección 4.1.2, recibiendo cada uno de estos hilos la cadena de conexión al clúster Spark como parte de sus parámetros de inicialización. Una vez que la ejecución de todos los algoritmos (y sus respectivas configuraciones de hiperparámetros, de acuerdo con la estrategia de optimización seleccionada) ha concluido, se procede a detener el clúster y a liberar los recursos de cómputo aprovisionados, mediante una nueva llamada a Flintrock en la que, junto al comando correspondiente, se indica el identificador del clúster a eliminar. Con esto, se da por finalizado el ciclo de vida del clúster, que se ha desplegado y configurado de forma automatizada directamente sobre la infraestructura del proveedor *cloud* y que sólo ha permanecido activo durante el periodo en el que se han estado ejecutando tareas sobre él, facilitando su gestión y reduciendo a lo imprescindible el coste económico derivado.

5.2.3 Opciones tecnológicas contempladas

En el momento de plantear cómo llevar a cabo el despliegue y la configuración de un clúster Spark de forma automatizada, Amazon EMR¹² (*Elastic Map Reduce*) se postuló como la opción que en principio resultaba más adecuada. El servicio EMR de AWS se encarga del despliegue y la configuración de clústeres con tecnologías orientadas al procesamiento distribuido de volúmenes masivos de datos, entre ellas Spark. De hecho, las últimas versiones de Amazon EMR incluyen un *runtime* propio de Spark optimizado para ejecutarse sobre la infraestructura de EC2 el cual, según enuncian sus creadores¹³, puede alcanzar un rendimiento tres veces superior al de Apache Spark. Los principales proveedores de *cloud*

¹²Amazon EMR, <https://aws.amazon.com/emr/>

¹³Apache Spark on Amazon EMR, <https://aws.amazon.com/emr/features/spark/>

público proporcionan servicios análogos a Amazon EMR, como Google Cloud Dataproc y Azure HDInsight, con lo que la dificultad de una hipotética portabilidad entre plataformas en este caso se limitaría a sustituir el servicio utilizado por el del proveedor *cloud* correspondiente y adaptar la interacción con él.

Sin embargo, existe una restricción¹⁴ que impide utilizar Amazon EMR en conjunción con el modelo de interacción con el clúster Spark que adopta Zygarde, debido a que EMR no soporta el modo autónomo (*standalone*) de Spark, limitándose a soportar la instalación sobre YARN, el gestor de recursos de Hadoop. Incluso aunque se configuren todas las reglas de los grupos de seguridad (que actúan a modo de cortafuegos) para permitir el tráfico de red entre las réplicas del motor de ejecución de reglas y el clúster Spark, el envío de trabajos al clúster debe realizarse mediante el comando `spark-submit`, no siendo posible configurar la conexión a un clúster Spark sustentado por Amazon EMR mediante una instrucción como la siguiente:

```
SparkConf conf = new SparkConf()  
    .setMaster("spark://master_url:7077")  
    .setAppName("Word Count");
```

El motor de ejecución de tareas de Zygarde se conecta a los clústeres de cómputo intensivo y ordena la ejecución de algoritmos sobre ellos de forma programática, puesto que este modelo de interacción permite acceder directamente desde el código fuente a los objetos que albergan los *datasets*, los modelos construidos y los resultados alcanzados en la evaluación de cada algoritmo. Además, este método permite que la ejecución de trabajos sobre el clúster se lleve a cabo de forma síncrona y bloqueante, retomando el flujo de instrucciones del programa cuando la ejecución del algoritmo sobre el clúster haya finalizado, sin necesidad de monitorizar activamente el estado de la tarea ni realizar una recuperación manual de los resultados desde el sistema de archivos HDFS. Debe remarcarse que esta interacción síncrona tiene lugar en el contexto de un hilo de ejecución dedicado a la coordinación de esa tarea específica, en un escenario con un nivel de concurrencia potencialmente elevado, con lo que el bloqueo de hilos individuales en la interacción con el clúster o clústeres de cómputo intensivo no dificulta el avance global del sistema.

La limitación detectada impide la comunicación programática con el clúster utilizando la propia API de Spark, lo que obligó a elegir entre modificar el modelo de interacción con el clúster desde el motor de ejecución de tareas para poder hacer uso de Amazon EMR, investigar en busca de otra tecnología que sí fuera compatible con el modelo de interacción descrito o proceder a implementar por medios propios los mecanismos para automatizar el despliegue y la configuración de un clúster Spark sobre Amazon EC2. Modificar el modelo

¹⁴Submit Spark Jobs to a Remote Amazon EMR Cluster,
<https://aws.amazon.com/premiumsupport/knowledge-center/emr-submit-spark-job-remote-cluster>

de interacción planteado habría supuesto graves inconvenientes, obligando a rehacer gran parte de la implementación ya realizada (cuyas pruebas habían tenido lugar sobre un clúster Spark *on-premises* desplegado en modo autónomo en una red de área local). Por su parte, implementar los mecanismos para automatizar el despliegue del clúster habría excedido el alcance acotado para el presente trabajo y habría supuesto un retraso de semanas o meses sobre la planificación inicial, incluso contando con la ayuda de tecnologías como Vagrant y Ansible para facilitar el despliegue y la configuración de las máquinas virtuales que conformaran el clúster. En cambio, sí resultó fructífera la búsqueda de otras tecnologías que se postularan como un sustituto válido para Amazon EMR, identificando dos candidatos claros: spark-ec2¹⁵ y Flintrock¹⁶.

El desarrollo de spark-ec2 se encuentra descontinuado y apunta directamente a Flintrock como su sucesor, el cual aporta ventajas como la reducción del tiempo requerido para contar con un clúster operativo y funcionalidades complementarias como la posibilidad de especificar las características del clúster mediante archivos de configuración (los manifiestos de despliegue mencionados en las subsecciones previas), la adición o eliminación de nodos esclavos de un clúster ya en funcionamiento y la posibilidad de utilizar AMIs propias. Resulta particularmente conveniente la capacidad para dirigir el despliegue mediante archivos de configuración (confeccionados dinámicamente por el motor de ejecución de tareas al procesar las peticiones de los usuarios) y, muy especialmente, la reducción del tiempo necesario para completar el despliegue. En este aspecto, Flintrock incluso ofrece un desempeño muy superior al de Amazon EMR, puesto que el despliegue de un clúster con EMR oscila en torno a los quince minutos, mientras que Flintrock es capaz de erigir un clúster operativo con hasta cien nodos esclavos en menos de cinco minutos, según manifiesta su documentación.

Por estos motivos, se toma la decisión de que la orquestación del ciclo de vida de los clústeres Spark, incluyendo su despliegue y su configuración, se automatice haciendo uso de Flintrock. Esta herramienta es capaz de desplegar clústeres Spark en modo autónomo, dispone de una interfaz de línea de órdenes muy sencilla y potente en combinación con los manifiestos de despliegue y ofrece unos tiempos de respuesta que son notablemente inferiores incluso a los del servicio proporcionado por Amazon Web Services. Todo ello facilita la interacción con el motor de ejecución de tareas de Zygade y contribuye a reducir el tiempo requerido para atender las peticiones de los usuarios. En lo que se refiere a la portabilidad entre plataformas, Flintrock actualmente sólo soporta el despliegue de clústeres Spark sobre Amazon EC2, pero su arquitectura se encuentra planteada de forma que pueda operar con otros proveedores de servicios en la nube sin alterar su interfaz, por lo que se ha planteado que esta herramienta incluya soporte adicional a Google Cloud en futuras versiones, algo que también constituye un aliciente para optar por su adopción.

¹⁵spark-ec2: Scripts used to setup a Spark cluster on EC2, <https://github.com/amplab/spark-ec2>

¹⁶flintrock: A command-line tool for launching Spark clusters, <https://github.com/nchammas/flintrock>

5.3 Clúster Horovod

La plataforma restante utilizada para la ejecución distribuida de algoritmos de aprendizaje automático es Horovod, empleada en tareas relativas al dominio de redes neuronales y aprendizaje profundo. Aunque Horovod funciona de forma nativa sobre un clúster MPI, se ha aprovechado la capacidad que posee para ejecutar trabajos de MPI sobre un clúster Spark para así reutilizar los mecanismos de automatización del despliegue de un clúster Spark ya implantados, y que se han descrito en la Sección 5.2. En este caso, la transmisión de datos para el paso de mensajes con el que opera MPI se realiza valiéndose del establecimiento de conexiones SSH entre los nodos del clúster, que es el mismo mecanismo que utiliza Spark a bajo nivel para repartir e intercambiar datos entre los nodos, facilitando la superposición de ambas tecnologías.

El proceso para la orquestación del clúster es similar al expuesto en la Sección 5.2. Cuando el motor de ejecución de tareas procesa la petición de un usuario, comprueba el dominio al que pertenece el problema planteado en la petición. En caso de que el dominio del problema se halle entre los cubiertos por Horovod, se procede a desplegar un clúster Spark, utilizando Flintrock para iniciar las máquinas virtuales de los nodos a partir de una AMI personalizada de CentOS Linux que, además de Spark y Python, tiene instalados Horovod, Open MPI y las demás dependencias requeridas. Con la salvedad de la imagen de máquina virtual utilizada, el resto de los aspectos relativos al despliegue del clúster se gestionan de forma idéntica a la ya vista, confeccionando un manifiesto de despliegue de Flintrock al que se trasladan las restricciones expresadas por el usuario en la petición de entrenamiento para especificar la topología del clúster y las características de las instancias de cómputo aprovisionadas.

En la presente sección se describe el procedimiento mediante el que se coordina el despliegue y la configuración automatizada de un clúster de Horovod sobre Amazon EC2, controlando el ciclo de vida completo del clúster. Se sigue una estructura paralela a la de la Sección 5.2, aunque con una extensión considerablemente menor, enfatizando las diferencias entre los procedimientos seguidos y las tecnologías adoptadas, y evitando reincidir en los elementos y fases del proceso que son similares a los ya tratados.

5.3.1 Orquestación de clústeres efímeros

Al igual que ocurre en el caso anterior, la propuesta de Zygarde en lo que se refiere al uso de clústeres de cómputo intensivo de diversas tecnologías no se basa en operar sobre un clúster gestionado manualmente, desplegado con anterioridad y con un periodo de vida prolongado o quasi-permanente. Por el contrario, se propone la operación con clústeres efímeros, que se despliegan bajo demanda y se eliminan una vez han cumplido su propósito, y cuyas características y topología no están predefinidas, sino que se determinan dinámicamente en el momento de proceder a su despliegue.

5.3.2 Ciclo de vida y ejecución de tareas

El proceso de despliegue de un clúster es análogo al ya descrito en la sección anterior; comprobando el dominio del problema de la petición del usuario para determinar qué imagen de máquina virtual emplear y procediendo a desplegar el clúster valiéndose de la herramienta Flintrock. Igualmente, el proceso de despliegue y configuración del clúster se solapa con la posible transferencia del *dataset* a Amazon S3, aprovechando el tiempo de espera para realizar varias acciones simultáneamente.

Al utilizar Horovod sobre Spark, los trabajos se envían al clúster utilizando la cadena de conexión al nodo maestro del clúster Spark, por lo que el procedimiento que se sigue para formar la cadena de conexión e injectarla en los parámetros de inicialización de los hilos que se encargan de controlar la ejecución de los diferentes algoritmos es idéntica a la ya estudiada. Igualmente, una vez ha concluido la ejecución de todas las tareas de entrenamiento correspondientes a los algoritmos de la petición, el desmantelamiento del clúster se lleva a cabo mediante una llamada a Flintrock en la que se indica el identificador del clúster a eliminar.

5.3.3 Pila tecnológica adoptada

El despliegue de un clúster Spark para la ejecución de algoritmos sobre Horovod se beneficia de la investigación previa efectuada para el caso de los algoritmos que se ejecutan sobre Spark MLLib. En este escenario, se aprovecha el trabajo ya incurrido y se adopta Flintrock como una alternativa a Amazon EMR que además incurre en menores tiempos de despliegue y no se encuentra necesariamente ligada a un proveedor *cloud* específico.

Más allá de la utilización de Flintrock para facilitar la orquestación de los clústeres Spark, en este caso también procede hacer mención a la pila de tecnologías que se disponen para la ejecución distribuida de algoritmos sobre Horovod. Dicho compendio se muestra en la Figura 5.5 y se analiza ulteriormente, constituyendo el cierre de la presente sección.

El despliegue de los nodos que conforman el clúster se realiza directamente sobre máquinas virtuales del servicio Amazon EC2. Algunos servicios de Amazon Web Services no sólo facturan en base a la infraestructura reservada a nivel de IaaS, sino que cobran una tarifa adicional por el uso de los propios servicios a nivel de PaaS. Aunque actualmente éste no es el caso de Amazon EMR, en ocasiones puede resultar conveniente no hacer uso de los servicios de alto nivel del proveedor *cloud* cuando éstos implican un sobrecoste respecto a la infraestructura aprovisionada y se puede obtener un resultado similar o incluso mejor adoptando herramientas externas. Flintrock opera directamente sobre Amazon EC2, lo que permite tener un control preciso del coste económico asumido por el despliegue de la infraestructura y supone una razón adicional para utilizar esta herramienta, además de las ya presentadas en la Subsección 5.2.3.





Figura 5.5: Pila tecnológica para la ejecución distribuida de algoritmos con Horovod

Flintrock despliega un clúster Spark sobre la infraestructura de Amazon EC2, encargándose de todo el proceso de inicialización y configuración hasta proporcionar un clúster operativo y listo para recibir tareas. En este caso particular, como ya se ha comentado, se procede a desplegar un clúster Spark no porque el *framework* de aprendizaje automático utilizado en el nivel superior dependa directamente de él, sino por una cuestión de conveniencia. La capacidad de Horovod para ejecutarse sobre Spark permite que en ambos casos, tanto con Spark MLlib como con Horovod, sea posible hacer uso de un clúster con una misma tecnología de base, aprovechando el trabajo que ya se ha realizado. Esta unificación también reduce la complejidad de la solución final al no tener que implementar y mantener sendos mecanismos para la orquestación de clústeres de cómputo intensivo dirigidos a diferentes tecnologías, uno para Spark MLlib y otro para Horovod.

Horovod se ejecuta de forma nativa sobre MPI, aunque, como ya se ha comentado, dispone de un complemento que permite su instalación y despliegue sobre un clúster Spark en funcionamiento, accediendo al mismo a través del nodo maestro. Dada la conveniencia de esta solución, no ha sido necesario desplegar un clúster MPI como tal para hacer frente a la ejecución de algoritmos de Horovod, sino que se han aprovechado las capacidades ya adquiridas para el despliegue automatizado de clústeres Spark que se han presentado. No obstante, esto no quiere decir que, al operar sobre un clúster Spark, Horovod no haga uso de MPI. La distribución de Horovod actúa sobre Open MPI, una implementación de MPI, la cual se encuentra instalada en las AMIs a partir de las que se crean las máquinas virtuales. Al operar sobre Spark, Horovod hace que Open MPI realice la transmisión de datos entre los nodos del clúster aprovechando las conexiones SSH que Spark dispone entre los nodos. Así, se posibilita superponer Spark y Open MPI de forma sencilla y operar con Horovod sin necesidad de desplegar un clúster MPI explícitamente.

Gracias a que la disposición del clúster se ha resuelto en los niveles inferiores, desde el propio *framework* Horovod no es necesario llevar a cabo ninguna labor de configuración que exceda lo trivial para poder ejecutar algoritmos sobre él. Horovod y sus complementos se distribuyen como un conjunto de paquetes Python instalables vía pip, haciendo uso de la distribución de MPI y de los compiladores que se encuentren instalados en el sistema. Para utilizar Horovod sobre Spark basta con implementar la lógica del proceso de entrenamiento en el contexto de una función, empleando el *framework* de aprendizaje que se deseé de entre los soportados por Horovod, e invocar a `horovod.spark.run` proporcionando el objeto con la información de la conexión al clúster y dicha función para que ésta última se ejecute de forma distribuida a través de MPI y, a su vez, sobre Spark.

Los algoritmos de redes neuronales en general, y en particular aquellos dirigidos a resolver problemas de aprendizaje profundo, se ejecutan en Zygarde mediante TensorFlow. A diferencia de la versión original de TensorFlow distribuido, en la que para conseguir ejecutar de forma distribuida un programa desarrollado para TensorFlow monolítico es necesario



realizar modificaciones importantes y en ocasiones infradocumentadas sobre el código fuente, las modificaciones a realizar sobre dicho programa para ejecutarlo de forma distribuida sobre Horovod son menores y se encuentran bien identificadas¹⁷. A grandes rasgos, debe inicializarse Horovod mediante la instrucción `hvd.init()`, escalar la tasa de aprendizaje en base al número de nodos esclavos o *workers* (según la terminología asimilada) y encapsular el objeto optimizador, que contiene la configuración con la que entrenar el modelo, en un optimizador distribuido de Horovod. La importación del objeto `hvd`, que se utiliza tanto para inicializar el *runtime* de Horovod como para acceder a los optimizadores distribuidos, es dependiente del *framework* utilizado. Así, para utilizar TensorFlow sobre Horovod se recurre a la instrucción `import horovod.tensorflow as hvd`.

Keras es una API de redes neuronales y aprendizaje profundo para Python, que actúa como una interfaz de alto nivel para TensorFlow. Ya que Horovod también proporciona soporte a Keras¹⁸ (aplicando las mismas transformaciones sobre el código fuente que en el escenario de TensorFlow pero importando el objeto `hvd` mediante la instrucción `import horovod.tensorflow.keras as hvd`), se ha utilizado esta biblioteca debido a que simplifica la interoperación con TensorFlow.

Gracias a la combinación de todas las tecnologías citadas, superpuestas tal como muestra la Figura 5.5, se ha logrado ejecutar algoritmos de aprendizaje profundo en TensorFlow distribuido sobre un clúster Spark, desarrollando programar que poseen una estructura simple y mantienen la perspectiva de un alto nivel de abstracción, independiente y agnóstico de las tecnologías y la infraestructura subyacente.

¹⁷Horovod with TensorFlow, <https://horovod.readthedocs.io/en/stable/tensorflow.html>

¹⁸Horovod with Keras, <https://horovod.readthedocs.io/en/stable/keras.html>

Capítulo 6

Despliegue y ejecución

HABIENDO presentado ya tanto el diseño de alto nivel del sistema como su arquitectura de despliegue sobre la infraestructura de Amazon Web Services, en este capítulo se muestra el procedimiento mediante el que se despliegan aquellos elementos estáticos de la arquitectura. Complementariamente, se analiza la traza de la ejecución originada en respuesta a la recepción de una petición de entrenamiento de ejemplo sobre un determinado conjunto de datos. A lo largo de las sucesivas etapas del *workflow* se aportan evidencias del tránsito de los datos a través de los diferentes componentes, categorizados como servicios, junto con los resultados obtenidos tras su ejecución. De esta forma, se contribuye a verificar el correcto funcionamiento del sistema en su escenario de uso común y a mostrar en acción, tanto a título individual como desde la perspectiva de la arquitectura en su conjunto, cada una de las unidades funcionales que conforman el sistema.

6.1 Procedimiento de despliegue

El propósito de esta sección es reflejar visualmente cómo los componentes de la arquitectura se materializan sobre la infraestructura del proveedor *cloud*. Con este fin, se realiza un recorrido superficial sobre los elementos tratados en las Secciones 4.1 y 5.1, mostrando cómo cada uno de ellos se representa en el panel de control de Amazon Web Services. En aquellos casos en los que un servicio se ejecuta en respuesta a un evento emitido desde otro componente, se muestra cómo configura esta relación reactiva entre servicios. Además, en aquellos componentes que se despliegan sobre máquinas virtuales de Amazon EC2 se muestra el panel de control de este servicio IaaS, en el que se reflejan las instancias de cómputo automáticamente desplegadas en función del volumen de peticiones en curso en el sistema, sin requerir en ningún momento la intervención de un operador humano.

El elemento que actúa como centralizador de la entrada de peticiones en el sistema es la cola de mensajes del servicio Amazon SQS, cuyo panel de control se presenta en las Figuras 6.1 y 6.2. Ésta última muestra la configuración de la cola, en la que se permite ajustar valores como el límite de tiempo que los mensajes pueden permanecer en ella, el periodo tras el que se considera que el procesamiento de un mensaje ha fallado o el número máximo de reintentos de procesamiento de un mensaje.



Queues (1)					
<input type="text"/> Search queues					
Name	Type	Created	Messages available	Messages in flight	
ZygardeQueue	Standard	18/2/2020 21:05:19	0	0	0

Figura 6.1: Panel de control de Amazon SQS

Details Info		
Name ZygardeQueue	Type Standard	ARN arn:aws:sqs:eu-west-1:922213900123:ZygardeQueue
Encryption Disabled	URL https://sns.eu-west-1.amazonaws.com/922213900123/ZygardeQueue	Dead-letter queue Disabled
▼ Hide		
Created 18/2/2020 21:05:19	Maximum message size 256 KB	Last updated 27/4/2020 0:53:32
Message retention period 1 Day	Default visibility timeout 12 Hours	Messages available 0
Delivery delay 0 Seconds	Messages in flight (not available to other consumers) 0	Receive message wait time 20 Seconds
Messages delayed 0	Content-based deduplication -	

Figura 6.2: Configuración de la cola de mensajes

El punto de entrada más común para la recepción de peticiones de entrenamiento en el sistema es la API REST, que se expone mediante Amazon API Gateway. Las Figuras 6.3 y 6.4 muestran el panel de control de este servicio. Resulta especialmente interesante comprobar cómo la recepción de peticiones POST en el *endpoint* accesible está integrada en un flujo de transformaciones entre las que se encuentra la ejecución de la función Lambda *ZygardeRestInput*; dicha integración entre servicios se muestra en la Figura 6.5 desde la perspectiva de la propia función Lambda. Aunque es posible configurar directamente la cola de mensajes de Amazon SQS como destino de la petición recibida, en este caso la redirección se ha implementado en el código fuente de la función. Esto se debe a que la inserción en la cola de mensajes sólo se produce si el documento JSON contenido en el cuerpo de la petición POST recibida supera las validaciones sobre el esquema de peticiones de entrenamiento. En caso de realizar alguna transformación adicional sobre el mensaje recibido a través de la API REST, también podría implementarse en la lógica de esta función. La Figura 6.6 detalla las características configurables básicas de la función Lambda, entre las que figuran su entorno de ejecución, la cuota de prestaciones físicas que se le adjudicará y el tiempo máximo estipulado para cada ejecución de la función.

APIs (1)					
Name	Description	ID	Protocol	Endpoint type	Created
ZygardeRestApi	Zygarde REST API	rs4qb8njs8	REST	Regional	2020-03-01

Figura 6.3: Panel de control de Amazon API Gateway

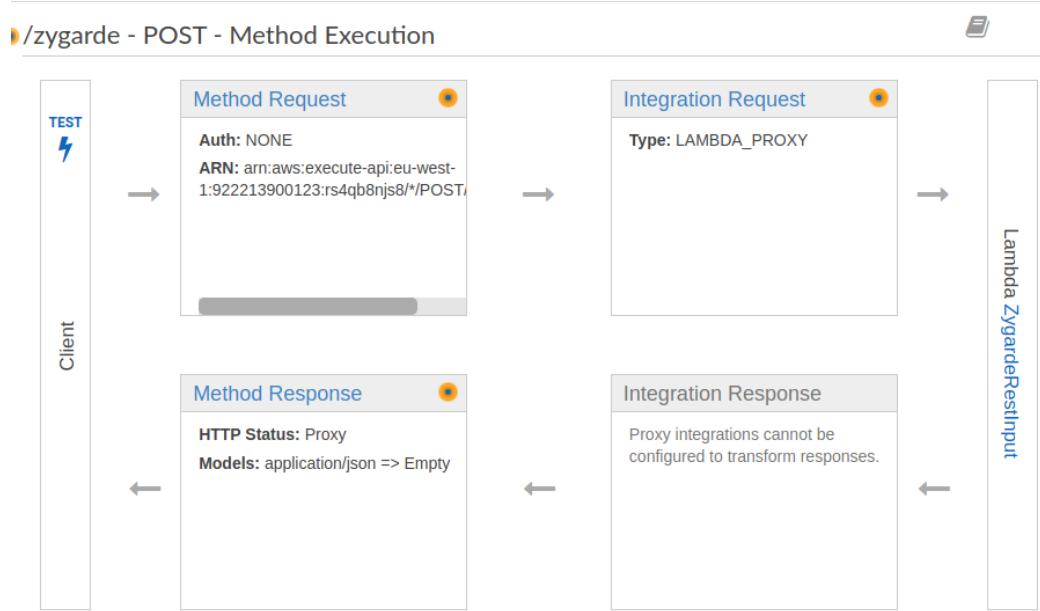


Figura 6.4: Secuencia de transformaciones de la petición POST

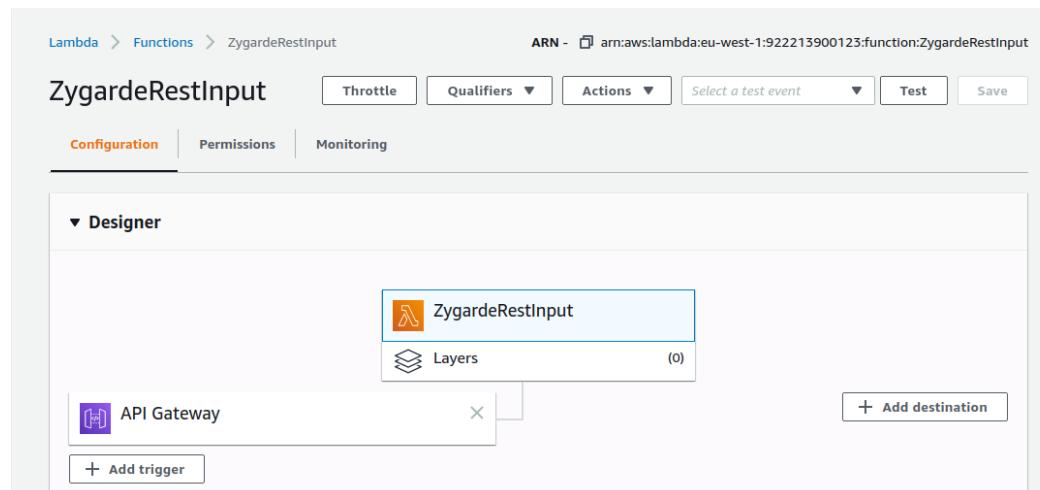


Figura 6.5: Integración de API Gateway como origen de la función Lambda

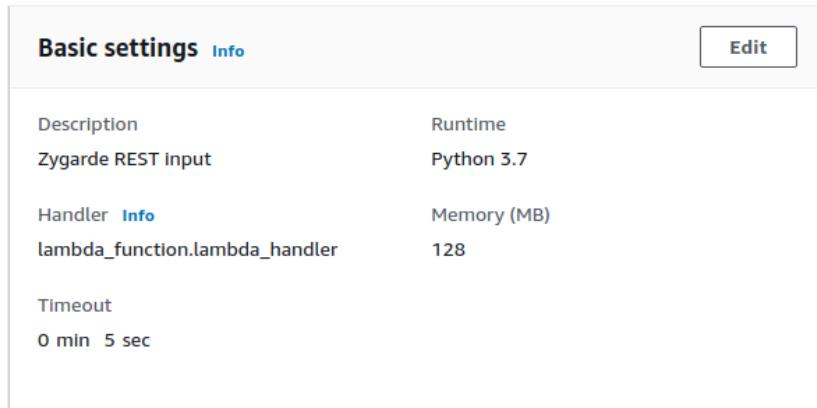


Figura 6.6: Configuración básica de función Lambda

Otra de las vías para enviar una petición de entrenamiento al sistema es mediante el depósito de un documento con dicha petición en un *bucket* de Amazon S3, ante lo que se ejecuta una función Lambda que, de forma similar a la ya vista, valida el formato de la petición y la inserta en la cola de mensajes. La integración entre una acción determinada sobre un *bucket* específico y la función Lambda se muestra en la Figura 6.7.

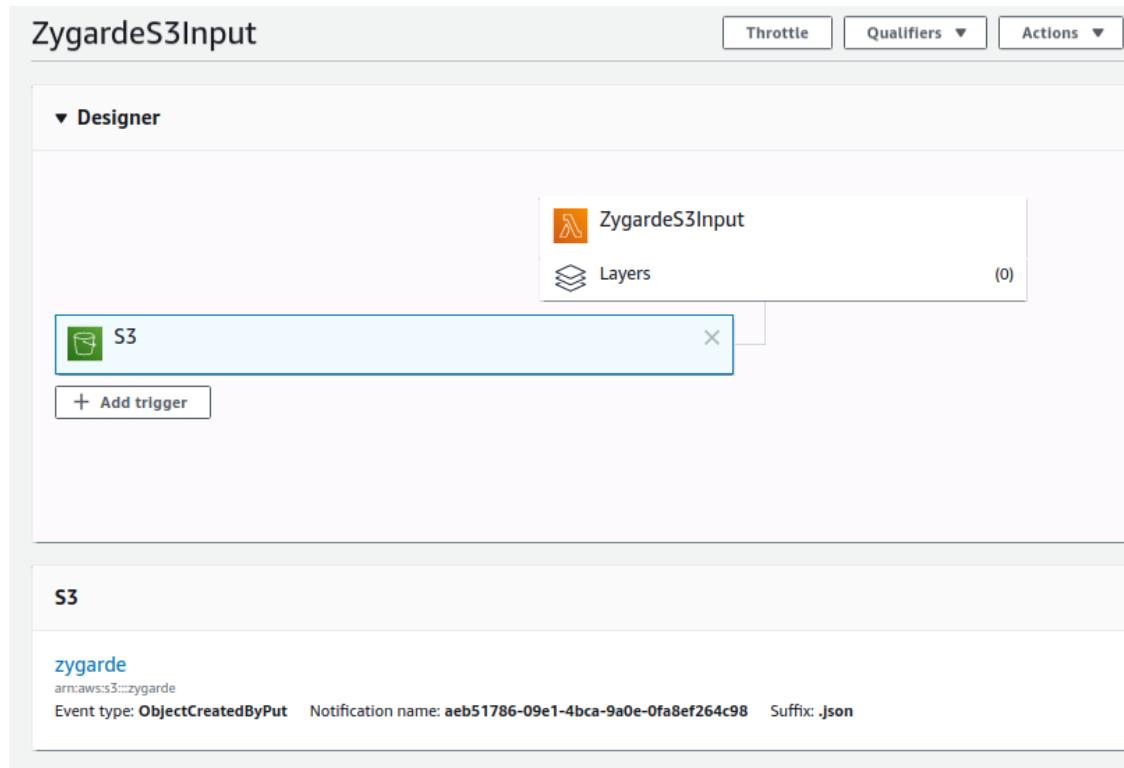


Figura 6.7: Acción sobre *bucket* de S3 como origen de ejecución de la función

El motor de ejecución de tareas extrae mensajes de la cola de Amazon SQS y los procesa. Este componente se despliega directamente sobre el servicio Amazon EC2, favoreciendo su disponibilidad a través de la replicación mediante un grupo de autoescalado. El hecho de que sean las propias réplicas las que extraigan los mensajes de la cola posibilita que cada una solicite nuevas tareas de acuerdo a su propio volumen de trabajo en cada momento y evita la necesidad de contar con un balanceador de carga. El grupo de autoescalado en torno al cual se despliegan las réplicas del motor de ejecución de tareas se define mediante una configuración de arranque, en la que se especifica el tipo de máquina a aprovisionar (con unas determinadas prestaciones según el modelo) y la imagen de disco virtual a utilizar, con el sistema operativo ya instalado. En la Figura 6.8 se muestran los detalles del grupo de autoescalado y su configuración de arranque asociada, a partir de la cual se inician las máquinas virtuales. La Figura 6.9 refleja las máquinas que se encuentran desplegadas en el contexto de este grupo de autoescalado, las cuales se han desplegado en diferentes zonas de disponibilidad de la región, proporcionando tolerancia a fallos al distribuir las réplicas entre centros de datos geográficamente separados.

Por otra parte, tanto los modelos construidos como los informes generados con los resultados del procesamiento de una petición se almacenan en *buckets* de Amazon S3. Al recibir una petición de entrenamiento, se comprueba si el *dataset* se encuentra alojado en S3 y, si no es así, se transfiere a este servicio, debido a que desde las máquinas que conforman los clústeres Spark se utiliza un protocolo de transmisión de datos propio de S3, gracias al que se consigue acceder al contenido con mayor eficiencia que si hubiera que recuperarlo de un servidor remoto de terceros. La Figura 6.10 muestra los *buckets* de Amazon S3 de los que se dispone, junto con sus correspondientes políticas de acceso.

EC2 > Auto Scaling groups > zygarde-auto-scaling-group

Details **Activity** **Automatic scaling** **Instance management** **Monitoring**

Group details

Desired capacity	3
Minimum capacity	3
Maximum capacity	3

Launch configuration

Launch configuration	zygarde-launch-configuration	AMI ID	ami-02cf1a9f247237e91
Instance type	t2.micro	Key pair name	aluccloud70-keypair
Storage (volumes)	/dev/sda1	View details in the launch configuration console	

Figura 6.8: Grupo de autoescalamiento y configuración de arranque

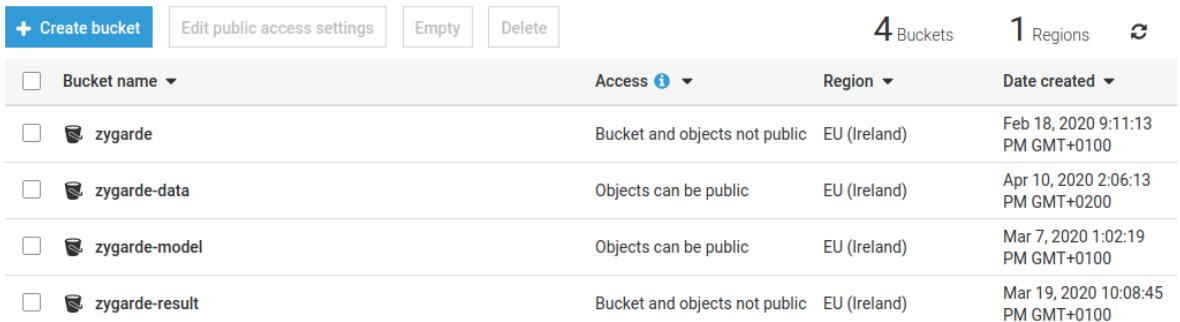
EC2 > Auto Scaling groups > zygarde-auto-scaling-group

Details **Activity** **Automatic scaling** **Instance management** **Monitoring** **Instance refresh**

Instances (3)

<input type="checkbox"/>	Instance ID	Lifecycle	Instance type	Weighted capacity	Launch template/configuration	Availability Zone	Health status
<input type="checkbox"/>	i-00d7a9a...	InService	t2.micro	-	zygarde-launch-configuration	eu-west-1a	Healthy
<input type="checkbox"/>	i-067d9a8e...	InService	t2.micro	-	zygarde-launch-configuration	eu-west-1c	Healthy
<input type="checkbox"/>	i-0a371b51...	InService	t2.micro	-	zygarde-launch-configuration	eu-west-1b	Healthy

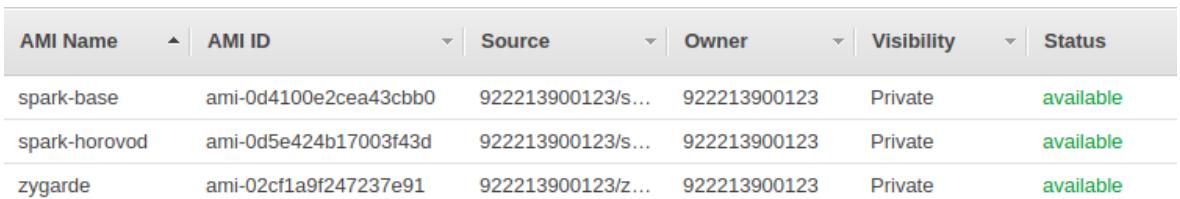
Figura 6.9: Réplicas desplegadas sobre el grupo de autoescalamiento



<input type="checkbox"/> Bucket name	Access	Region	Date created
<input type="checkbox"/> zygarde	Bucket and objects not public	EU (Ireland)	Feb 18, 2020 9:11:13 PM GMT+0100
<input type="checkbox"/> zygarde-data	Objects can be public	EU (Ireland)	Apr 10, 2020 2:06:13 PM GMT+0200
<input type="checkbox"/> zygarde-model	Objects can be public	EU (Ireland)	Mar 7, 2020 1:02:19 PM GMT+0100
<input type="checkbox"/> zygarde-result	Bucket and objects not public	EU (Ireland)	Mar 19, 2020 10:08:45 PM GMT+0100

Figura 6.10: Buckets de Amazon S3

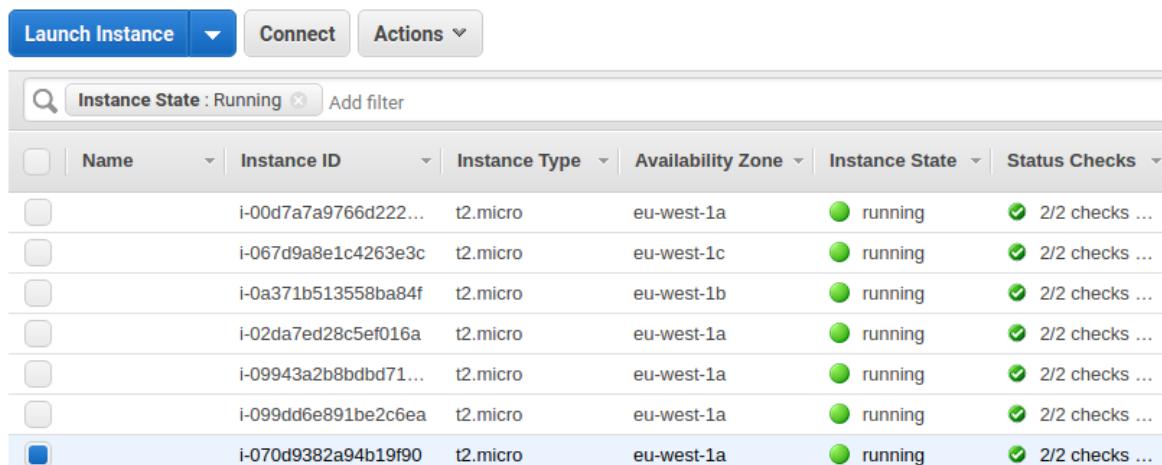
Al recibir una petición de entrenamiento, el motor de ejecución de tareas ordena el despliegue de un clúster en el que se delegarán los procesos de aprendizaje automático que permitan determinar cuál es la combinación de algoritmo e hiperparámetros capaz de obtener un modelo que optimice los resultados alcanzados sobre el conjunto de datos proporcionado. Como se expuso en las Secciones 5.2 y 5.3, actualmente Zygarde soporta la gestión de clústeres de dos tecnologías: Spark y Horovod, aunque la capacidad de Horovod para ejecutar MPI sobre Spark ha llevado a que en todos los casos se recurra a la herramienta Flintrock para automatizar el despliegue de un clúster Spark con la distribución de software necesaria. Puesto que Zygarde determina la tecnología a utilizar en el proceso de entrenamiento en base al dominio del problema expresado en la petición, se cuenta con imágenes de máquina virtual que incluyen la pila tecnológica requerida para cada caso, ya instalada y configurada. Estas imágenes se listan en la Figura 6.11, en la que también figura la AMI utilizada para iniciar las réplicas del motor de ejecución de tareas a partir de la configuración de arranque de su grupo de autoescalamiento. Una alternativa al uso de imágenes preconfiguradas sería partir de una imagen base del sistema operativo sobre la que emplear utilidades de *DevOps* como Ansible para instalar y configurar el software al lanzar la máquina virtual, lo que evitaría la tarea de mantener las imágenes actualizadas y parcheadas, pero también implicaría un tiempo de inicio considerablemente mayor, motivo por el que se ha descartado esta opción.



AMI Name	AMI ID	Source	Owner	Visibility	Status
spark-base	ami-0d4100e2cea43ccb0	922213900123/s...	922213900123	Private	available
spark-horovod	ami-0d5e424b17003f43d	922213900123/s...	922213900123	Private	available
zygarde	ami-02cf1a9f247237e91	922213900123/z...	922213900123	Private	available

Figura 6.11: Imágenes preconfiguradas de máquina virtual

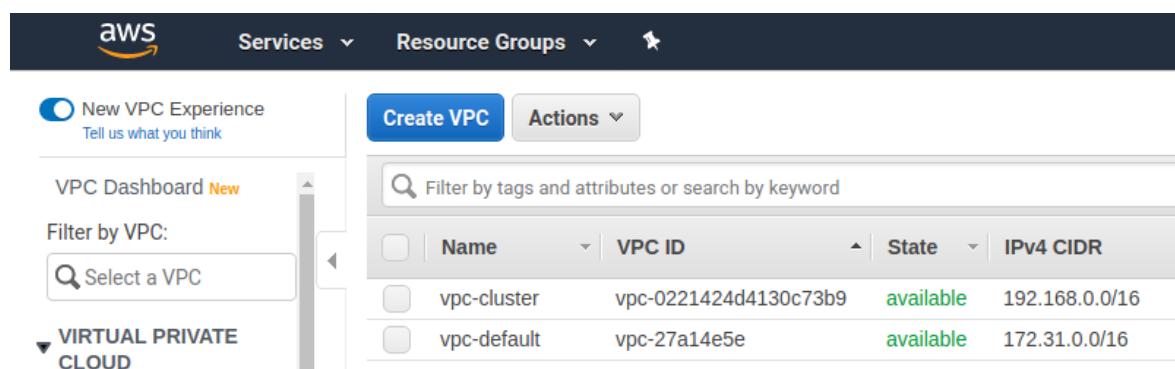
La Figura 6.12 ofrece una perspectiva del panel de Amazon EC2 en la que figuran las máquinas virtuales activas, entre las que se encuentran las tres réplicas del grupo de auto-escalado del motor de ejecución de tareas y un clúster Spark activo en el momento de la captura, el cual cuenta con un nodo maestro y tres esclavos. Los clústeres se despliegan en una subred diferente a la del motor de ejecución de tareas, lo que, en combinación con la aplicación de políticas de seguridad, permite refinar el control que se posee sobre el tráfico hacia y desde los nodos. En el caso concreto del clúster Spark, se ha establecido una política de seguridad sobre sus nodos que sólo permite que éstos sean objeto de acceso por parte de otros nodos de su propia subred o de la subred del motor de ejecución de tareas, lo que ayuda a protegerlos ante accesos indeseados. Las subredes habilitadas, denominadas VPCs (*Virtual Private Cloud*) en la terminología de AWS, se muestran en la Figura 6.13.



The screenshot shows the AWS EC2 Instances page. At the top, there are buttons for 'Launch Instance', 'Connect', and 'Actions'. Below that is a search bar with 'Instance State : Running' and an 'Add filter' button. A table lists seven instances:

	Name	Instance ID	Instance Type	Availability Zone	Instance State	Status Checks
<input type="checkbox"/>	i-00d7a7a9766d222...	t2.micro	eu-west-1a	running	2/2 checks ...	
<input type="checkbox"/>	i-067d9a8e1c4263e3c	t2.micro	eu-west-1c	running	2/2 checks ...	
<input type="checkbox"/>	i-0a371b513558ba84f	t2.micro	eu-west-1b	running	2/2 checks ...	
<input type="checkbox"/>	i-02da7ed28c5ef016a	t2.micro	eu-west-1a	running	2/2 checks ...	
<input type="checkbox"/>	i-09943a2b8bdbd71...	t2.micro	eu-west-1a	running	2/2 checks ...	
<input type="checkbox"/>	i-099dd6e891be2c6ea	t2.micro	eu-west-1a	running	2/2 checks ...	
<input checked="" type="checkbox"/>	i-070d9382a94b19f90	t2.micro	eu-west-1a	running	2/2 checks ...	

Figura 6.12: Máquinas virtuales desplegadas y activas sobre Amazon EC2



The screenshot shows the AWS VPC Dashboard. At the top, there is a navigation bar with the AWS logo, 'Services', 'Resource Groups', and a star icon. On the left, there is a sidebar with a 'New VPC Experience' link and a 'VPC Dashboard' link. The main area has a 'Create VPC' button and an 'Actions' dropdown. A search bar at the top right says 'Filter by tags and attributes or search by keyword'. A table lists two VPCs:

	Name	VPC ID	State	IPv4 CIDR
<input type="checkbox"/>	vpc-cluster	vpc-0221424d4130c73b9	available	192.168.0.0/16
<input type="checkbox"/>	vpc-default	vpc-27a14e5e	available	172.31.0.0/16

Figura 6.13: Subredes virtuales en la nube

Por último, una vez que el motor de ejecución de tareas finaliza el procesamiento de una petición de entrenamiento, envía al usuario un correo electrónico que contiene un informe con los resultados. Esta remisión se realiza mediante una función Lambda, que se activa cuando el motor de ejecución de tareas deposita el informe en un determinado *bucket* de Amazon S3 e invoca al servicio Amazon SES para enviar un mensaje a la dirección del usuario. La integración entre Amazon S3 y la función Lambda se realiza de forma idéntica a la ya vista en la recepción de peticiones, pero reaccionando a los eventos de creación de archivos de un *bucket* diferente. En este caso, el envío del correo también se realiza desde el código fuente de la función, pues Amazon SES no se encuentra entre los destinos cuya integración directa permite Amazon Lambda. La Figura 6.14 muestra la estructura de un evento recibido por la función al depositar un documento en el *bucket* de Amazon S3 vinculado a la misma. Entre los campos del documento JSON con la información del evento se encuentra la ruta del archivo almacenado, incluyendo la dirección de correo de destino. Al ejecutarse, la función procede a recuperar este archivo del *bucket* y utiliza su contenido para componer un mensaje de correo electrónico que será enviado al usuario.



Configure test event

A function can have up to 10 test events. The events are persisted so you can switch to another computer or web browser and test your function with the same events.

- Create new test event
- Edit saved test events

Saved test event

S3ObjectCreateEvent

```

1- []
2- "Records": [
3-   {
4-     "eventVersion": "2.0",
5-     "eventSource": "aws:s3",
6-     "awsRegion": "eu-west-1",
7-     "eventTime": "2020-03-01T03:48:32.000Z",
8-     "eventName": "ObjectCreated:Put",
9-     "userIdentity": {
10-       "principalId": "EXAMPLE"
11-     },
12-     "requestParameters": {
13-       "sourceIPAddress": "127.0.0.1"
14-     },
15-     "responseElements": {
16-       "x-amz-request-id": "EXAMPLE123456789",
17-       "x-amz-id-2": "EXAMPLE123/5678abcdefghijklmabaisawesome/mnopqrstuvwxyzABCDEFGH"
18-     },
19-     "s3": {
20-       "s3SchemaVersion": "1.0",
21-       "configurationId": "testConfigRule",
22-       "bucket": {
23-         "name": "zygarde-result",
24-         "ownerIdentity": {
25-           "principalId": "ZYGARDE"
26-         },
27-         "arn": "arn:aws:s3:::zygarde-result"
28-       },
29-       "object": {
30-         "key": "thanatos.dreamslayer@gmail.com/46988f20-6c6f-4101-aa35-16b53216983a.txt",
31-         "size": 1850,
32-         "eTag": "54b6d47910747a916f3ec89caa13fa3a",
33-         "sequencer": "0A1B2C3D4E5F678901"
34-       }
35-     }
36-   }
37- ]
38-
```

Delete **Cancel** **Format JSON** **Save**

Figura 6.14: Evento recibido por la función de envío de informes

6.2 Ejecución de ejemplo

En esta sección se analiza la ejecución que tiene lugar a partir de la recepción de una petición de entrenamiento, remitida por un usuario, sobre un conjunto de datos. El procesamiento de la petición recibida se desarrolla a lo largo de una serie de etapas estructuradas mediante un *workflow*. Para cada una de estas etapas se aportan evidencias gráficas de su ejecución y del tránsito de los datos a través de los componentes que la conforman, reflejando cómo el diseño descrito en el Capítulo 4 se ha materializado en un sistema funcional que cumple el propósito con el que fue concebido.

La petición de entrenamiento a partir de la que tiene lugar esta demostración se envía al sistema mediante una operación POST a través de la API REST expuesta por el servicio Amazon API Gateway. En el cuerpo de la petición se incluye el documento JSON con las características del entrenamiento a realizar, de acuerdo con el formato de esquema de petición presentado en la Sección 4.4 y el Apéndice A, tal como se muestra en la Figura 6.15. En este caso, para actuar sobre un ejemplo sencillo que se pueda trazar con facilidad, se solicita la ejecución de un único algoritmo (regresión lineal) proporcionando diferentes valores para tres de sus hiperparámetros y se establece que la exploración del espacio de los hiperparámetros se implemente mediante una estrategia de búsqueda aleatoria. Se ha proporcionado intencionadamente un *dataset* que no se encuentra alojado en Amazon S3, por lo que el motor de ejecución de tareas deberá encargarse, llegado el momento, de transferirlo a este servicio de almacenamiento de objetos para así optimizar el acceso a los datos durante el proceso de entrenamiento.

Tras el envío de la petición, se puede observar que el servicio devuelve un código HTTP 200 (OK), pero tarda algo más de medio segundo en proporcionar su respuesta, debido a que el procesamiento síncrono de la petición incluye la ejecución de la función Lambda en la que se valida su concordancia respecto al formato esperado y su inserción en la cola de mensajes de Amazon SQS. Esta latencia puede oscilar en función de si se han realizado ejecuciones anteriores de la función Lambda y el servicio es capaz de reutilizar un contenedor de procesos ya iniciado, o por el contrario debe crear un nuevo contenedor para ejecutar la función, lo que supone un incremento de unos cientos de milisegundos,¹ aunque en cualquier caso se trata de algo intrínseco a la gestión de los recursos y la infraestructura del propio servicio Amazon Lambda [GAMC19]. En aplicaciones en las que el tiempo de ejecución y respuesta de la función es un factor crítico, este efecto se puede mitigar a través de opciones como la “conurrencia aprovisionada”.² La Figura 6.16 muestra la recepción de un mensaje en la cola de Amazon SQS, al que se le ha asignado un identificador en formato UUIDv4 y cuyos metadatos se presentan en la Figura 6.17, desde un panel que también permite consultar el contenido del mensaje. El identificador asignado al mensaje recibido es relevante, puesto que Zygarde lo utiliza para marcar la petición y facilitar su seguimiento a lo largo del ciclo completo de procesamiento.

¹AWS Lambda Cold Start Language Comparisons, 2019 edition, 
<https://levelup.gitconnected.com/aws-lambda-cold-start-language-comparisons-2019-edition>

²New - Provisioned Concurrency for Lambda Functions | AWS News Blog,
<https://aws.amazon.com/blogs/aws/new-provisioned-concurrency-for-lambda-functions/>



POST https://rs4qb8njs8.execute-api.eu-west-1.amazonaws.com/dev/zygarde

```

1 { "domain": "regression",
2   "methods": [
3     {
4       "algorithm": "linear-regression",
5       "hyperparameters": [
6         {
7           "param": "maxIter",
8           "values": [ 10, 50, 100 ]
9         },
10        {
11          "param": "regParam",
12          "values": [ 0.0, 0.3 ]
13        },
14        {
15          "param": "elasticNetParam",
16          "values": [ 0.0, 0.8 ]
17        }
18      ]
19    },
20  ],
21  "dataset": {
22    "path": "https://raw.githubusercontent.com/apache/spark/master/data/mllib/sample_linear_regression_data.txt",
23    "format": "libsvm"
24  },
25  "parameterSearch": "random",
26}

```

Body Cookies Headers (7) Test Results Status: 200 OK Time: 563 ms

Pretty Raw Preview JSON 🔍

```

1 { "MD5OfMessageBody": "8cc0dca79533f41a72f96c9722b8177f",
2   "MessageId": "27ff9694-3366-48d6-903a-114de569c4fa",
3   "ResponseMetadata": {
4     "RequestId": "8bd03cad-326f-548b-a0fd-6b2ae76239ea",
5     "HTTPStatusCode": 200,
6     "HTTPHeaders": {
7       "x-amzn-requestid": "8bd03cad-326f-548b-a0fd-6b2ae76239ea",
8       "date": "Tue, 14 Jul 2020 15:44:13 GMT",
9       "content-type": "text/xml",
10      "content-length": "378"
11    }
12  }

```

Figura 6.15: Envío de petición de entrenamiento a través de la API REST

Receive messages [Info](#)

Messages available: 1 Polling duration: 30 Maximum message count: 10 Polling progress: 1 receives/second

[Edit poll settings](#) [Stop polling](#)

Messages (1)

ID	Sent	Size	Receive count
27ff9694-3366-48d6-903a-114de569c4fa	14/7/2020 17:44:13	644 bytes	1

[View](#)

Figura 6.16: Recepción de un mensaje en la cola de Amazon SQS

Message: 27ff9694-3366-48d6-903a-114de569c4fa			
Details	Body	Attributes	
ID 27ff9694-3366-48d6-903a-114de569c4fa	Size 644 bytes	MD5 of message body 8cc0dca79533f41a72f96c9722b8177f	Sender account ID AROA5NOB3L5NTODTHLY3:ZygardeRestInput
Sent 14/7/2020 17:44:13	First received 14/7/2020 17:44:42	Receive count 1	Message attributes count -
Message attributes size -	MD5 of message attributes -		

Figura 6.17: Metadatos y detalle del mensaje recibido

El mensaje depositado en la cola de Amazon SQS es recuperado por una de las réplicas del motor de ejecución de tareas, dando comienzo a su procesamiento propiamente dicho, cuya traza de ejecución se muestra en la Figura 6.18. El punto de inicio de la traza de ejecución se sitúa aproximadamente un minuto después de la inserción del mensaje en la cola, debido a que el motor de ejecución de tareas se encontraba detenido para poder capturar las imágenes de las Figuras 6.16 y 6.17, en las que el mensaje se encuentra en la cola de SQS pero aún no ha sido extraído. Aunque colateralmente, esta situación muestra cómo el sistema es capaz de proporcionar respuesta al usuario incluso aunque todas las réplicas del motor de ejecución de tareas hubieran fallado simultáneamente, en cuyo caso las peticiones pendientes quedarían almacenadas en la cola de mensajes y se procesarían cuando el servicio de monitorización del grupo de autoescalado detectase esta situación y sustituyera las máquinas caídas por nuevas réplicas.

```

System Log: i-0fe378c8f1cb3cf77

2020-07-14 17:45:31 INFO SqsEntryPoint:41 - Queue: ZygardeQueue
2020-07-14 17:45:31 INFO SqsEntryPoint:42 - Queue URL: https://sqs.eu-west-1.amazonaws.com/92213900123/ZygardeQueue
2020-07-14 17:45:31 INFO SqsEntryPoint:51 - Received message 27ff9694-3366-48d6-903a-114de569c4fa
2020-07-14 17:45:31 INFO JobRequestExecutor:67 - es.upv.mbd.tfm.zygarde.schema.ZygardeRequest@6009dd145
2020-07-14 17:45:32 INFO SparkClusterLauncher:126 - Launching Spark cluster (3 worker nodes)
2020-07-14 17:45:32 INFO JobRequestExecutor:161 - Transferred sample linear regression data.txt to zygarde-data S3 bucket
2020-07-14 17:48:34 INFO SparkClusterLauncher:231 - Spark cluster deployed. Master URL: spark://ec2-54-171-165-133.eu-west-1.compute.amazonaws.com:7077
2020-07-14 17:48:39 INFO ParameterizedAlgorithmExecutor:76 - spark.algo.algo.wrapper.zygarde.py --id 27ff9694-3366-48d6-903a-114de569c4fa --algorithm linear-regression
--dataset s3a://zygarde-data/27ff9694-3366-48d6-903a-114de569c4fa/sample_linear_regression_data.txt --dataFormat libsvr --elasticNetParam 0.8 --maxIter 10 --regParam 0.0
2020-07-14 17:48:39 INFO ParameterizedAlgorithmExecutor:76 - spark.algo.algo.wrapper.zygarde.py --id 27ff9694-3366-48d6-903a-114de569c4fa --algorithm linear-regression
--dataset s3a://zygarde-data/27ff9694-3366-48d6-903a-114de569c4fa/sample_linear_regression_data.txt --dataFormat libsvr --elasticNetParam 0.0 --maxIter 10 --regParam 0.0
2020-07-14 17:48:39 INFO ParameterizedAlgorithmExecutor:76 - spark.algo.algo.wrapper.zygarde.py --id 27ff9694-3366-48d6-903a-114de569c4fa --algorithm linear-regression
--dataset s3a://zygarde-data/27ff9694-3366-48d6-903a-114de569c4fa/sample_linear_regression_data.txt --dataFormat libsvr --elasticNetParam 0.8 --maxIter 100 --regParam 0.3
2020-07-14 17:48:39 INFO ParameterizedAlgorithmExecutor:76 - spark.algo.algo.wrapper.zygarde.py --id 27ff9694-3366-48d6-903a-114de569c4fa --algorithm linear-regression
--dataset s3a://zygarde-data/27ff9694-3366-48d6-903a-114de569c4fa/sample_linear_regression_data.txt --dataFormat libsvr --elasticNetParam 0.0 --maxIter 100 --regParam 0.3
2020-07-14 17:48:39 INFO ParameterizedAlgorithmExecutor:76 - spark.algo.algo.wrapper.zygarde.py --id 27ff9694-3366-48d6-903a-114de569c4fa --algorithm linear-regression
--dataset s3a://zygarde-data/27ff9694-3366-48d6-903a-114de569c4fa/sample_linear_regression_data.txt --dataFormat libsvr --elasticNetParam 0.0 --maxIter 50 --regParam 0.3
2020-07-14 17:49:07 INFO ParameterizedAlgorithmExecutor:53 - linear-regression: -10.9835
2020-07-14 17:49:07 INFO SqsEntryPoint:103 - linear-regression: -10.9835
2020-07-14 17:49:07 INFO SqsEntryPoint:103 - linear-regression: -10.9835
2020-07-14 17:49:07 INFO SqsEntryPoint:105 - Uploaded 27ff9694-3366-48d6-903a-114de569c4fa/-10.9835-linear-regression-report.txt to zygarde-model AWS S3 bucket
2020-07-14 17:49:23 INFO ParameterizedAlgorithmExecutor:53 - linear-regression: -9.4276
2020-07-14 17:49:23 INFO SqsEntryPoint:104 - linear-regression: -9.4276
2020-07-14 17:49:23 INFO SqsEntryPoint:104 - Uploaded 27ff9694-3366-48d6-903a-114de569c4fa/-9.4276-linear-regression-report.txt to zygarde-model AWS S3 bucket
2020-07-14 17:49:23 INFO SqsEntryPoint:105 - linear-regression: -9.4276
2020-07-14 17:49:23 INFO ParameterizedAlgorithmExecutor:53 - linear-regression: -10.5735
2020-07-14 17:49:39 INFO ParameterizedAlgorithmExecutor:53 - linear-regression: -10.5735
2020-07-14 17:49:39 INFO SqsEntryPoint:103 - linear-regression: -10.5735
2020-07-14 17:49:39 INFO SqsEntryPoint:105 - Uploaded 27ff9694-3366-48d6-903a-114de569c4fa/-10.5735-linear-regression-report.txt to zygarde-model AWS S3 bucket
2020-07-14 17:49:45 INFO ParameterizedAlgorithmExecutor:53 - linear-regression: -10.2140
2020-07-14 17:49:45 INFO SqsEntryPoint:103 - linear-regression: -10.2140
2020-07-14 17:49:45 INFO SqsEntryPoint:104 - Uploaded 27ff9694-3366-48d6-903a-114de569c4fa/-10.214-linear-regression-report.txt to zygarde-model AWS S3 bucket
2020-07-14 17:49:45 INFO SqsEntryPoint:105 - Uploaded 27ff9694-3366-48d6-903a-114de569c4fa/-10.214-linear-regression-report.txt to zygarde-model AWS S3 bucket
2020-07-14 17:49:55 INFO GridSearchMethodExecutor:51 - Algorithm: linear-regression Best precision: -9.4376
2020-07-14 17:49:55 INFO JobRequestExecutor:86 - Best precision: -9.4376
2020-07-14 17:49:55 INFO SqsEntryPoint:119 - Uploaded thanhtuan.dreamlayer@gmail.com/27ff9694-3366-48d6-903a-114de569c4fa.txt to zygarde-result S3 bucket
2020-07-14 17:49:59 INFO SqsEntryPoint:122 - Finished execution of job from message 27ff9694-3366-48d6-903a-114de569c4fa

```

Figura 6.18: Traza del motor de ejecución de tareas

La traza de ejecución de la Figura 6.18 muestra cómo el mensaje es recuperado de la cola de Amazon SQS y, tras comprobar sus características, se procede a ordenar el despliegue de un clúster Spark con un nodo maestro y tres esclavos de unas determinadas prestaciones. El proceso de despliegue y configuración del clúster, que tarda algo más de tres minutos, se realiza en paralelo a la transferencia del *dataset* a un *bucket* de Amazon S3. Gracias a esta transferencia, a las tareas individuales de entrenamiento se les indica que el *dataset* se encuentra en una ruta precedida por el prefijo s3a, permitiendo que el acceso a los datos se implemente de forma eficiente haciendo uso de un protocolo propio de S3. Las dos acciones referidas a su vez se solapan con el cálculo de las tareas de entrenamiento a ejecutar, de acuerdo con la estrategia de exploración aleatoria seleccionada. Una vez que el despliegue del clúster Spark se ha completado, las tareas de entrenamiento se lanzan al clúster para su ejecución, indicando la URL de acceso al nodo maestro, el identificador de la tarea, el algoritmo a utilizar, los valores de los hiperparámetros y la ubicación y el formato del *dataset*. La Figura 6.19 muestra la interfaz del nodo maestro del clúster Spark desplegado, en la que figuran los tres nodos esclavos vinculados al clúster y cuatro tareas ya ejecutadas, que corresponden a las cuatro combinaciones de valores de los hiperparámetros que se han seleccionado. Las políticas de seguridad instauradas sobre los nodos del clúster impiden acceder directamente a ellos desde el equipo de desarrollo, por lo que para realizar esta captura ha sido necesario conectarse vía SSH a una máquina en la subred del motor de ejecución de tareas e instalar un servidor proxy que redirigiera al exterior la conexión al puerto 8081 del nodo maestro del clúster, habilitando el acceso a través del navegador web de un equipo externo.



The screenshot shows the Spark Master web interface at <spark://ec2-54-171-165-133.eu-west-1.compute.amazonaws.com:7077>. It displays the following information:

- Cluster Statistics:**
 - URL: spark://ec2-54-171-165-133.eu-west-1.compute.amazonaws.com:7077
 - Alive Workers: 3
 - Cores in use: 3 Total, 0 Used
 - Memory in use: 6.0 GB Total, 0.0 B Used
 - Applications: 0 Running, 4 Completed
 - Drivers: 0 Running, 0 Completed
 - Status: ALIVE
- Workers (3)**: A table showing three workers with their addresses, states, cores, and memory usage.
- Running Applications (0)**: A table showing zero running applications.
- Completed Applications (4)**: A table showing four completed applications with their details like ID, name, cores, memory, submission time, user, state, and duration.

Figura 6.19: Interfaz web del nodo maestro del clúster Spark

Cuando una de las tareas de entrenamiento solicitadas al clúster finaliza, se muestran por pantalla sus características (algoritmo y valores de los hiperparámetros) y el resultado alcanzado, tras lo que se procede a almacenar en Amazon S3 el modelo generado y un informe con los resultados de la ejecución de la tarea. Resulta llamativo comprobar que el resultado obtenido por las tareas es un número negativo, si bien la explicación es sencilla: Zygarde compara los resultados tratando de maximizar el valor obtenido, con independencia del algoritmo o el dominio del problema. La métrica utilizada para determinar la calidad de los modelos generados por los algoritmos de regresión es la raíz del error cuadrático medio (RMSE), como se establece en la Subsección 4.3.3. Puesto que esta métrica opera sobre el error incurrido, un modelo será mejor cuanto menor sea el valor devuelto por la función de evaluación, motivo por el que se niega su valor para que Zygarde identifique como el mejor modelo aquel que minimiza el valor absoluto del error alcanzado. Una vez que todas las tareas de entrenamiento han finalizado, se ordena la detención y destrucción del clúster, se señala cuál es el algoritmo que mejor resultado ha alcanzado y se compone el informe con las tareas de entrenamiento ejecutadas, ordenadas en base a sus respectivos resultados.

Por último, se sube el informe a una ubicación específica de Amazon S3 y se elimina el mensaje de la cola de Amazon SQS, marcándolo como procesado con éxito. Las Figuras 6.20, 6.21 y 6.22 muestran los archivos almacenados a lo largo del procesamiento de la petición en diferentes *buckets* de Amazon S3. Estos *buckets* corresponden, respectivamente, al *dataset* (cuya transferencia y almacenamiento no se habría producido si ya se hubiera encontrado alojado en S3), los modelos e informes individuales generados y el informe final que aglutina todos los informes individuales ordenados según la calidad del modelo obtenido.

Amazon S3 > zygarde-data > 27ff9694-3366-48d6-903a-114de569c4fa

zygarde-data

Overview

 Type a prefix and press Enter to search. Press ESC to clear.

 Upload

 + Create folder

Download

Actions ▾

Name ▾

Last modified ▾

 sample_linear_regression_data.txt

Jul 14, 2020 5:48:35 PM GMT+0200

Figura 6.20: *Dataset* transferido a Amazon S3

Amazon S3 > zygarde-model > 27ff9694-3366-48d6-903a-114de569c4fa

zygarde-model

Overview

 Type a prefix and press Enter to search. Press ESC to clear.

 Upload

 + Create folder

Download

Actions ▾

Name ▾

Last modified ▾

 -10.2140-linear-regression-model.json

Jul 14, 2020 5:49:55 PM GMT+0200

 -10.2140-linear-regression-report.txt

Jul 14, 2020 5:49:54 PM GMT+0200

 -10.5735-linear-regression-model.json

Jul 14, 2020 5:49:39 PM GMT+0200

 -10.5735-linear-regression-report.txt

Jul 14, 2020 5:49:38 PM GMT+0200

 -10.9835-linear-regression-model.json

Jul 14, 2020 5:49:07 PM GMT+0200

 -10.9835-linear-regression-report.txt

Jul 14, 2020 5:49:06 PM GMT+0200

 -9.4376-linear-regression-model.json

Jul 14, 2020 5:49:23 PM GMT+0200

 -9.4376-linear-regression-report.txt

Jul 14, 2020 5:49:22 PM GMT+0200

Figura 6.21: Modelos generados e informes individuales

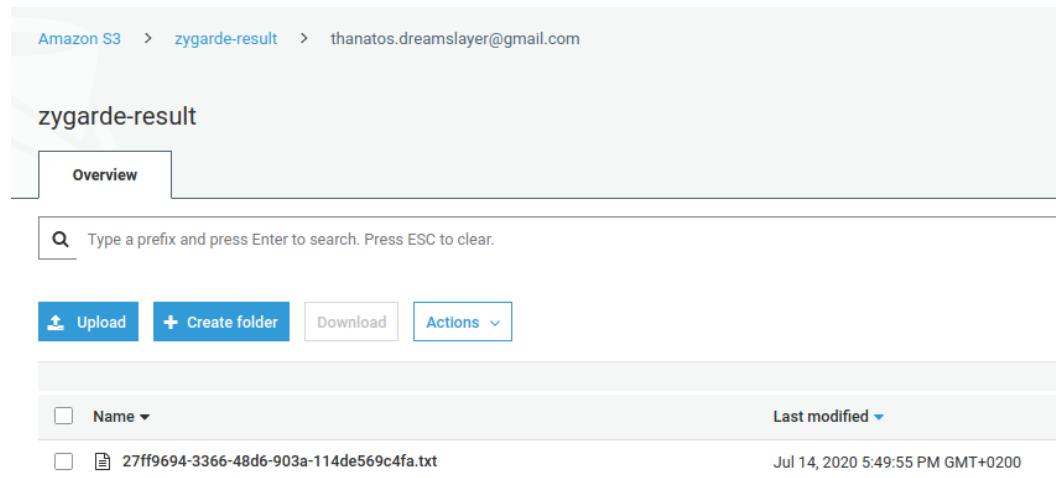


Figura 6.22: Informe final de resultados, almacenado en Amazon S3

La acción de depositar el informe final en el *bucket* `zygarde-result` de Amazon S3 desencadena la ejecución de una función Lambda que recupera el documento con el informe y utiliza sus contenidos para componer un correo electrónico, el cual se envía al usuario. La Figura 6.23 muestra el mensaje de correo recibido en el dispositivo del usuario, albergando los resultados de su petición de entrenamiento sobre Zygarde y las rutas desde las que puede recuperar los informes y modelos individuales, lo que supone la compleción del ciclo de interacción del usuario con la plataforma. Finalmente, y actuando como cierre y perspectiva complementaria de la sección, la Figura 6.24 ofrece un conjunto de gráficos que agregan diferentes aspectos de la ejecución de las funciones Lambda, reflejando tanto aquellas que han tenido lugar a lo largo del procesamiento de la petición de entrenamiento estudiada como otras ejecuciones realizadas con anterioridad.

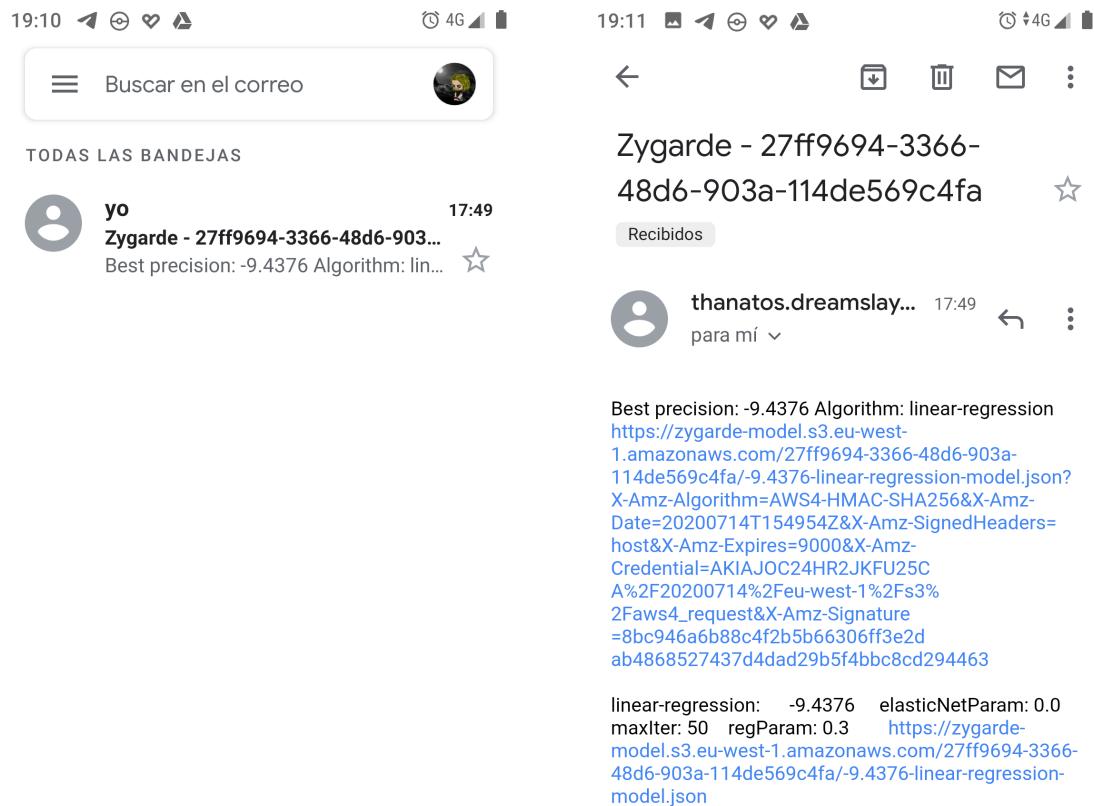


Figura 6.23: Correo electrónico con los resultados, recibido por el usuario

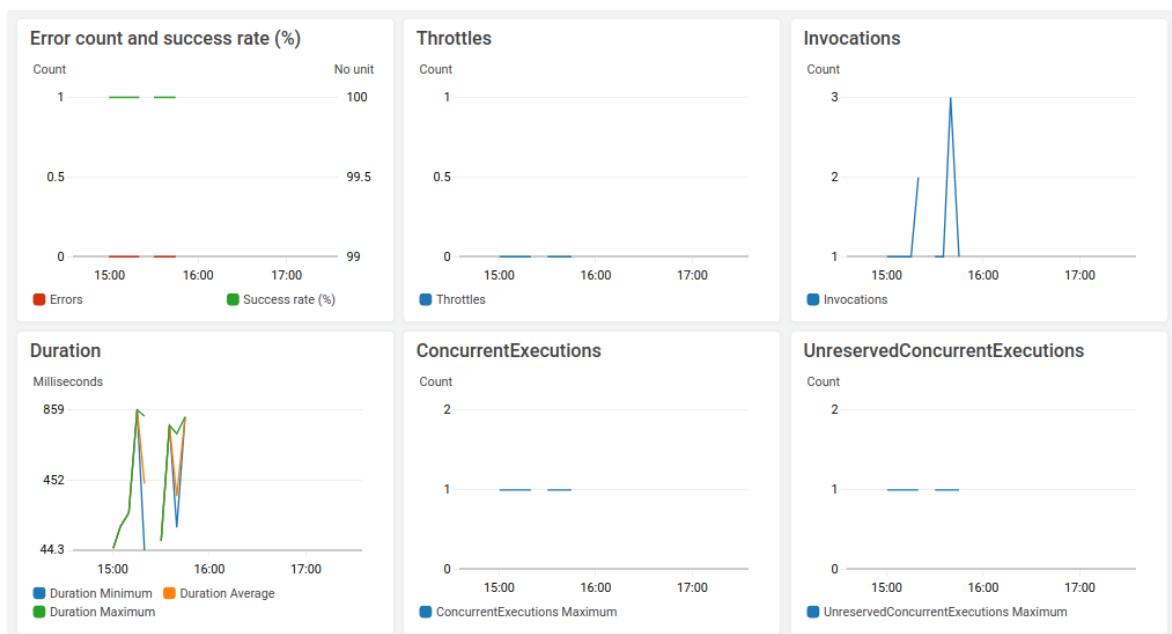


Figura 6.24: Ejecución agregada de las funciones Lambda

Conclusiones y propuestas

FINALMENTE, en este capítulo se elabora una síntesis del trabajo realizado y sus principales aspectos a destacar. Tras constatar el cumplimiento de las metas alcanzadas en el contexto del alcance inicialmente propuesto, se detallan algunas de las líneas de trabajo futuro que pueden resultar interesantes de cara a la mejora o ampliación del sistema desarrollado. Por último y ya como cierre al documento, se ofrece una apreciación personal sobre lo que simboliza el presente Trabajo Fin de Máster como punto final a la titulación del Máster en Big Data Analytics.

7.1 Síntesis del trabajo realizado

Una vez completado, el presente Trabajo Fin de Máster cumple con todos y cada uno de los objetivos que se especificaron en el Capítulo 3, tanto en lo que se refiere a requisitos funcionales como a requisitos no funcionales. Se ha construido una plataforma, gestionada en la nube, que es capaz de realizar procesos de entrenamiento en el contexto de tareas de aprendizaje automático. Los usuarios tienen la posibilidad de escoger entre una amplia gama de algoritmos pertenecientes a diferentes dominios, indicando las configuraciones de hiperparámetros a probar y pudiendo especificar la heurística que debe seguirse en la exploración del espacio de valores de los hiperparámetros. De esta forma, se consigue automatizar el proceso de búsqueda mediante el que identificar el algoritmo y la combinación de valores de los hiperparámetros del mismo con los que se obtiene el modelo más preciso, operando sobre el conjunto de datos especificado por el usuario y ciñéndose a las restricciones formuladas por éste en la petición de entrenamiento. En este aspecto, resulta particularmente relevante la implementación de un método de búsqueda inteligente basado en técnicas de optimización bayesiana. Este método de búsqueda permite aproximarse al modelo óptimo reduciendo el número de entrenamientos a realizar en comparación con los métodos de búsqueda no informada, particularmente frente a uno que implemente una exploración exhaustiva del espacio multidimensional definido por el dominio de valores de los hiperparámetros.



La plataforma desarrollada, denominada Zygarde, se ha desplegado en la nube sobre la infraestructura y los servicios del proveedor Amazon Web Services. La arquitectura del sistema en su conjunto se ha concebido y diseñado para asumir este modelo de despliegue, prestando especial atención a la división de la funcionalidad en servicios diferenciados y al modelo de interacción entre componentes, optando en este último particular por la adopción de un comportamiento reactivo basado en un modelo asíncrono de interacción dirigida por eventos. Este modelo permite alcanzar un alto grado de desacoplamiento e independencia entre los componentes, lo que posibilita modificarlos o reemplazarlos por completo de forma transparente a los demás, evitar la propagación de fallos en cascada e incluso aplicar políticas y factores de escalado horizontal individualizados en cada componente. El planteamiento seguido para fragmentar la arquitectura en servicios compuestos por múltiples unidades de despliegue ha permitido emplear técnicas de replicación y tolerancia a fallos que contribuyen a garantizar la estabilidad y la disponibilidad del sistema. La inclusión de mecanismos de elasticidad en la gestión del factor de escalado horizontal sobre los componentes ha permitido dimensionarlos reaccionando de acuerdo a su volumen de carga en cada momento, realizando una gestión eficiente de los recursos de cómputo y su aprovisionamiento. A la hora de elegir la tecnología sobre la que estructurar y desplegar cada componente se ha llevado a cabo un estudio de las opciones disponibles, escogiendo según resultara conveniente en cada caso entre los servicios de alto nivel del proveedor, la acción directa sobre máquinas virtuales y la combinación de ésta última con soluciones propias o de terceros, tomando en consideración elementos como el coste económico asociado y la portabilidad entre los principales proveedores de *cloud* público.

La oferta de recursos hardware por parte del proveedor *cloud* se ha aprovechado decre-tando el uso de implementaciones distribuidas de los algoritmos de aprendizaje automático ofrecidos, que se ejecutan sobre clústeres erigidos para atender los procesos de entrenamiento derivados de cada petición individual de un usuario. La adopción de dichas implementaciones distribuidas y escalables de los algoritmos permite reducir el tiempo total de entrenamiento y su ejecución sobre un conjunto de instancias de cómputo sin grandes prestaciones, más económicas que la adquisición o el alquiler de un único nodo con una elevada dotación de recursos físicos. Ante la recepción de una petición de entrenamiento por parte de un usuario, se procede a desplegar sobre la marcha un clúster cuya pila tecnológica se determina en base al dominio del problema a abordar, sin que afecte en modo alguno a los niveles del programa que no tienen una interacción directa con el clúster, y que se eliminará cuando haya finalizado su labor. Todo ello se orquesta en torno a una gestión elástica de los recursos aprovisionados, que se adquieren o liberan dinámicamente en función de la carga actual del sistema y las peticiones en curso, tanto en el caso de los propios clústeres de cómputo intensivo como en el resto de los componentes y las unidades de despliegue sobre las que se materializan.

El propósito de este trabajo es mostrar las ventajas alcanzables mediante la conjugación de las áreas de la computación distribuida y el aprendizaje automático, en concreto cómo la segunda puede beneficiarse de la adopción de los principios y técnicas de la primera. La aplicación de los procedimientos y estrategias de la computación distribuida, particularmente la computación en la nube, puede agilizar los procesos de entrenamiento relativos a tareas de aprendizaje automático, acelerando su ejecución, evitando realizar una inversión previa en infraestructura y tomando medidas para reducir el coste económico derivado del uso de los recursos de cómputo empleados. La utilización de técnicas de aprendizaje automático, construyendo modelos que permiten efectuar predicciones a partir de un entrenamiento previo sobre un conjunto de datos representativos del dominio del problema, resulta de gran utilidad en numerosos ámbitos, desde la elaboración de recomendaciones personalizadas para los usuarios de una web de compras al cálculo de series temporales en relación al número de usuarios simultáneos de una plataforma de *streaming*, pasando por la optimización de procesos industriales y el soporte a la toma de decisiones ante la información que arrojan los índices bursátiles. En todos estos escenarios es deseable completar con la mayor rapidez posible el proceso de entrenamiento que lleva a la obtención de un modelo. Sin embargo, en situaciones de emergencia sanitaria como la actual, en la que el cálculo de tendencias para predecir la expansión de un virus o la estimación de la efectividad de un compuesto basándose en datos previos pueden probarse cruciales para el bienestar de la población, es imprescindible que el lapso requerido para el entrenamiento de los modelos no constituya un cuello de botella en la investigación. Es aquí donde el empleo de las técnicas de computación distribuida en sistemas altamente escalables para el análisis y procesamiento masivo de datos supone marcar una diferencia trascendental respecto al tradicional enfoque monológico, acelerando estos procesos mediante los que proporcionar información útil y actualizada a los expertos de diversas áreas, que pueden servirse de ella para avanzar en sus propias líneas de trabajo y alcanzar resultados con la mayor brevedad.

7.2 Propuestas y líneas de trabajo futuro

Si bien el presente Trabajo Fin de Máster ha resultado en una plataforma completa que cumple con el propósito que motiva su creación y con todos los objetivos especificados, aún existen diferentes vías de evolución que pueden ampliar y mejorar el sistema desarrollado y que podrían servir como líneas de trabajo futuro. Estas líneas de acción, que no se han llevado a cabo por situarse fuera del alcance acotado o por restricciones temporales, se plantean como una extensión al trabajo realizado o como una forma de complementarlo y son las siguientes:

■ Flexibilidad en la especificación del dominio de los hiperparámetros

Como se indica en la Sección 4.4, el esquema de las peticiones de entrenamiento de Zygarde sólo admite la definición de valores categóricos, tanto numéricos como textuales, en la especificación de los valores a probar sobre los hiperparámetros de un determinado algoritmo. La capacidad de definir el espacio de búsqueda del dominio a explorar mediante valores continuos, rangos o incluso distribuciones de probabilidad es una característica cuya implementación sería deseable. En concreto, es una funcionalidad que sí incorpora la biblioteca Hyperopt, estudiada en la Subsección 2.3.2, y que resulta de gran utilidad para definir fácilmente dominios de cierta complejidad sobre el espacio de los hiperparámetros.

■ Detección y tratamiento de fallos en la gestión de los recursos aprovisionados

El modelo de procesamiento de peticiones de Zygarde, utilizando una cola de mensajes a la que acceden las réplicas del motor de ejecución de tareas, consigue que, ante la detención de alguna de dichas réplicas, las peticiones que dicha réplica estaba procesando vuelvan a insertarse en la cola transcurrido un periodo, con lo que no se perderán y tarde o temprano serán procesadas con éxito. Igualmente, la monitorización de las réplicas del motor de ejecución de tareas en combinación con el grupo de autoescalado en el que éstas se engloban provoca que, si alguna de las réplicas del componente deja de responder durante un lapso de tiempo, ésta sea eliminada y reemplazada mediante el despliegue de una nueva réplica. Sin embargo, hay dos situaciones derivadas de la gestión de los recursos que realiza el motor de ejecución de tareas de Zygarde que son potencialmente problemáticas y no se han abordado en el desarrollo, dado que entrañan una sustancial complejidad que habría alterado la planificación prevista.

La primera de estas situaciones tiene lugar cuando, habiendo desplegado réplicas adicionales del motor de ejecución de tareas para atender un incremento en la carga de trabajo, dicha carga posteriormente desciende y se procede a eliminar una de las réplicas desplegadas. En estas circunstancias, cualquiera de las réplicas, indistintamente de si está procesando peticiones o no, es candidata a ser eliminada sin previo aviso. Si la réplica elegida se encuentra procesando peticiones de los usuarios, cuya resolución puede prolongarse en el tiempo, su detención supondría la pérdida de todo el trabajo realizado por la réplica sobre las peticiones en curso. Esta eventualidad hace patente la necesidad de implementar un mecanismo para eliminar réplicas de forma ordenada, primero evitando que éstas asuman nuevas tareas y seguidamente aguardando a que finalice el procesamiento de todas las peticiones en curso sobre la réplica o a que expire un temporizador, con lo que la réplica podrá detenerse y eliminarse en condiciones de seguridad. Actualmente, para evitar que este efecto indeseado provoque que haya tareas cuyo procesamiento se interrumpa y deba reiniciarse por completo, se ha establecido una restricción de acuerdo a la que el grupo de autoescalado cuenta en todo

momento con tres réplicas activas, sin variar su cardinalidad, al menos hasta que haya podido implementarse el mecanismo escrito. El único caso en el que se interviene sobre las réplicas desplegadas no tiene lugar a raíz de una variación sobre el nivel de carga de las instancias del servicio, sino cuando se detecta que una réplica ha dejado de funcionar, momento en el que es suprimida y reemplazada por una nueva máquina virtual.

La segunda situación problemática está relacionada con la primera y es relativa a la gestión del ciclo de vida de los clústeres de cómputo intensivo. Ya que cada réplica del motor de ejecución de tareas es quien coordina el despliegue y eliminación de un clúster al comenzar y finalizar el procesamiento de una petición de entrenamiento, los recursos aprovisionados por una réplica que se detenga de forma inesperada seguirán desplegados indefinidamente, sin que nadie los controle ni se encargue de eliminarlos. La restricción paliativa adoptada en la primera situación problemática expuesta evita que esta segunda situación tenga lugar al decrementar el factor de escalado como consecuencia de una reducción de la carga de trabajo, pero este problema sigue produciéndose cuando una réplica se detiene debido a un fallo. Esta eventualidad hace necesario implementar un mecanismo que complemente al descrito en la primera situación problemática y que se sirva de Flintrock para controlar los recursos aprovisionados por el motor de ejecución de tareas, de tal forma que éstos sean eliminados automáticamente al desaparecer la propia réplica que solicitó el despliegue de dichos recursos.

■ **Implementación propia del método de optimización bayesiana**

El método de búsqueda sobre el espacio de valores de los hiperparámetros empleando técnicas de optimización bayesiana se ha implementado mediante la integración de SMAC en Zygarde, tal como se describió en la Subsección 4.2.3. Esta integración ha sido ciertamente complicada, pues para obtener el modelo de interacción deseado ha sido necesario realizar la integración a nivel del código fuente de SMAC, el cual ha habido que estudiar con anterioridad. Con la finalidad de eliminar esta dependencia conflictiva, se propone la implementación por medios propios de un método de optimización bayesiana que sustituya al actual, sin necesidad de incluir dependencias adicionales de terceros que obliguen a decompilar su código fuente o a tomar otras medidas poco elegantes. A la hora de plantear esta implementación, sería conveniente realizar un estudio sobre las características y las cualidades de los algoritmos comúnmente utilizados para la optimización de los valores de los hiperparámetros [DMC16], decidiendo cuál resulta más razonable adoptar. En este escenario, existe la posibilidad de que la regresión mediante *random forest* que implementa SMAC no sea la mejor opción y resulte preferible utilizar *Tree Parzen Estimators* [BBBK11], como ocurre en Hyperopt, aunque eso es algo que deberá concluirse a partir del citado estudio.

■ Transformaciones sobre los datos y *pipelines*

Una limitación en los procesos de aprendizaje automático que implementa Zygarde es la carencia de soporte a transformaciones sobre el conjunto de datos de entrenamiento para aplicar técnicas como la reducción de la dimensionalidad mediante el análisis de componentes principales. Igualmente, aunque Zygarde permite identificar el algoritmo que mejor resultado obtiene sobre el conjunto de datos, no permite definir o inferir secuencias de algoritmos, intercalados con las transformaciones mencionadas sobre los datos, constituyendo *pipelines* como los existentes en Google Cloud AI Platform y MLflow. Aunque no es algo trivial y sería necesario sopesar detenidamente cómo plantear la composición de estos *pipelines*, tanto desde la perspectiva de su definición en la petición de entrenamiento por parte del usuario, como su conformación dinámica como consecuencia del proceso de optimización del modelo generado, disponer de esta funcionalidad supondría un gran avance al incrementar la flexibilidad que ofrece la plataforma, revalorizando su potencial.

■ Definición de infraestructura como código

Los elementos de la arquitectura que constituyen unidades de despliegue estáticas sobre la infraestructura y los servicios del proveedor *cloud* (es decir, no creados y eliminados dinámicamente durante la operación del sistema, como ocurre con los clústeres Spark gestionados a través de Flintrock) se han creado manualmente mediante el panel de control de AWS, tal como se ha mostrado en la Sección 6.1. Aunque ésta es la vía más rápida para desplegar servicios durante las fases de prototipado y pruebas, no facilita sucesivas instalaciones de la plataforma, cuyas necesidades no se limitan a contar con el código fuente, sino también a ser capaces de reproducir el procedimiento completo de despliegue. Con el objetivo de plasmar este procedimiento de forma no ambigua, reproducible e incluso automatizable, se propone la definición de su infraestructura como código. Para este fin, Amazon Web Services cuenta con la herramienta CloudFormation¹, que proporciona un lenguaje para la definición de los elementos de su infraestructura y sus servicios, así como la expresión de las relaciones entre ellos, permitiendo el despliegue directo de la arquitectura de una aplicación distribuida a partir de un documento escrito en dicho lenguaje. Mientras que CloudFormation es un producto específico de Amazon Web Services, Terraform² es una alternativa capaz de operar sobre la infraestructura *cloud* de diferentes proveedores, incluso de realizar un despliegue de acuerdo a un modelo de *cloud* híbrido, ofreciendo un lenguaje de definición para sus archivos de configuración que es agnóstico a la tecnología subyacente. Gracias a ello, el procedimiento de despliegue se adapta por sí mismo a los servicios del proveedor o proveedores correspondientes, siempre que éstos estén soportados.

¹AWS CloudFormation - Infrastructure as Code & AWS Resource Provisioning,

<https://aws.amazon.com/cloudformation/>

²Terraform by HashiCorp, <https://terraform.io>

Otra característica interesante de Terraform es que separa las fases de planificación y ejecución del despliegue. En el momento de realizar el despliegue de una arquitectura de aplicación a partir de un archivo de definición, la herramienta elabora un grafo de las acciones a realizar, que puede ser revisado, modificado o acotado (estableciendo que sólo se lleven a cabo determinados subconjuntos de las acciones especificadas), otorgando a sus usuarios un alto grado de control y detalle sobre el proceso de despliegue a ejecutar antes de que éste se haga efectivo.

■ **Informe con los resultados del proceso de entrenamiento**

Puesto que la estética de los informes que se envían a los usuarios con el resultado de la ejecución de su petición de entrenamiento nunca fue una prioridad, éstos se han compuesto de una forma sumamente rudimentaria y su formato corresponde al de un archivo de texto plano. Un valor añadido en futuras iteraciones del desarrollo sería la utilización de bibliotecas como JasperReports³, que se sirven de una plantilla y un origen de datos suministrado por el programa para generar informes en múltiples formatos, ofreciendo una apariencia profesional y cuidada.

7.3 Conclusión personal

Durante la recta final del Máster Universitario en Computación Paralela y Distribuida del DSIC, hace ya algo más de dos años, era consciente de haber adquirido un conocimiento muy valioso sobre los procedimientos y las técnicas a emplear en la resolución de problemas que requieren una gran cantidad de recursos y tiempo de cómputo. Sin embargo, deseaba profundizar en mayor grado en las aplicaciones de este tipo de técnicas y consideré el análisis y procesamiento de volúmenes masivos de datos como el área más prometedora mediante cuyo aprendizaje complementar mi formación, por lo que procedí a inscribirme en la siguiente edición del Máster en Big Data Analytics.

En aquel momento estaba familiarizado con tecnologías como Hadoop y Spark, que implementan modelos de programación escalables como MapReduce, los cuales permiten procesar grandes volúmenes de datos de forma distribuida, pero apenas acertaba a vislumbrar la gran cantidad de cuestiones y problemáticas adicionales que es necesario resolver para poder tratar esos datos debidamente. Lo que más me sorprendió al dar comienzo al nuevo curso fue la multidisciplinariedad de este ámbito, en el que confluyen materias como estadística, procesamiento del lenguaje natural o visualización, por citar sólo unas pocas. De hecho, esta variedad de disciplinas, unido al diverso trasfondo de quienes hemos concurrido al máster, hace que, habiendo cursado las mismas asignaturas con los mismos contenidos, cada uno

³JasperReports® Library | JasperSoft Community,
<https://community.jaspersoft.com/project/jasperreports-library>

hayamos podido optar por una vertiente diferente en función de nuestras preferencias, desde un enfoque orientado a la ciencia de datos a otra perspectiva encaminada al diseño de arquitecturas software escalables y eficientes para extraer, almacenar, procesar y transmitir dichos datos. En mi caso particular, esta tendencia estaba clara debido a mi formación específica y experiencia profesional previas, pero desde el primer momento me llamó la atención la estrecha relación que existe entre el área de Big Data Analytics y el aprendizaje automático o *machine learning*. Siendo testigo de su evidente utilidad pero también del largo tiempo de ejecución en el que con frecuencia incurren los procesos de aprendizaje automático, empecé a investigar sobre la forma en la que se podrían aplicar técnicas de computación paralela o distribuida para reducir los tiempos de entrenamiento, descubriendo así tecnologías como Spark MLlib. Constatando la existencia del aprendizaje automático distribuido como un área ya existente y con entidad propia, sobre la que hasta la fecha se ha realizado un considerable avance pero en la que aún cabe un espacio de mejora aún mayor, decidí que mi Trabajo Fin de Máster fuera en esa dirección.

Combinando mi formación previa en computación distribuida en la nube y arquitectura software con la amplia variedad de conocimientos adquiridos a lo largo del Máster en Big Data Analytics, y con el inestimable apoyo del profesor Germán Moltó, he dado lugar al presente Trabajo Fin de Máster y a la plataforma desarrollada en su contexto. Considero que no se trata en absoluto de un mero trabajo académico sin mayor recorrido potencial ni aplicación práctica, sino de un sistema que sirve a un propósito específico y que puede resultar de gran utilidad para quienes requieran ejecutar procesos de aprendizaje automático sin necesidad de conocer la tecnología y la infraestructura subyacentes, y muy especialmente en la automatización de la selección de algoritmos e hiperparámetros que conduce a la optimización de los modelos generados. En la confección de este trabajo he tratado de ilustrar y materializar muchas de las conclusiones a las que, en materia de computación, he llegado a lo largo de la última década, mediante la lectura de numerosos artículos y libros y una cantidad aún mayor de cafés e instructivas conversaciones sobre el tema. Es posible que Zygarde no esté en situación de competir contra las plataformas de gigantes como Amazon, Google o Microsoft; pero es el producto de mi propio esfuerzo y a través de su creación he seguido aprendiendo, evolucionando y mejorando, algo que de otra forma puede que no hubiera logrado. Sólo por eso, ya merece la pena.

APÉNDICES

Apéndice A

Esquema JSON del formato de petición

```
1  {
2      "id": "https://github.com/systematic-chaos/zygarde",
3      "$schema": "https://json-schema.org/draft-04/schema#",
4      "type": "object",
5      "properties": {
6          "domain": {
7              "type": "string",
8              "enum": [
9                  "regression",
10                 "classification",
11                 "clustering",
12                 "deep-learning",
13                 "binomial-classification",
14                 "multinomial-classification",
15                 "misc-functions"
16             ]
17         },
18         "dataset": {
19             "type": "object",
20             "properties": {
21                 "path": {
22                     "type": "string"
23                 },
24                 "format": {
25                     "type": "string",
26                     "enum": ["libsvm"]
27                 }
28             },
29             "required": ["path"]
30         }
31     }
32 }
```



```

31     "methods": {
32         "type": "array",
33         "items": {
34             "type": "object",
35             "properties": {
36                 "algorithm": {
37                     "type": "string",
38                     "enum": [
39                         "linear-regression",
40                         "generalized-linear-regression",
41                         "random-forest-regression",
42                         "decision-tree-regression",
43                         "gradient-boosted-tree-regression",
44                         "linear-support-vector-machine",
45                         "binomial-logistic-regression",
46                         "naive-bayes",
47                         "random-forest-classification",
48                         "multinomial-logistic-regression",
49                         "decision-tree-classification",
50                         "gradient-boosted-tree-classification",
51                         "k-means",
52                         "gaussian-mixture-model",
53                         "multilayer-perceptron-classifier",
54                         "branin"
55                     ]
56                 },
57                 "hyperparameters": {
58                     "type": "array",
59                     "items": {
60                         "type": "object",
61                         "properties": {
62                             "param": {
63                                 "type": "string"
64                             },
65                             "values": {
66                                 "type": "array",
67                                 "items": {
68                                     "type": [ "number", "string" ]
69                                 }
70                             }
71                         }
72                     }
73                 }
74             }
75         }
76     }
77 }
```

```

72         "required": [
73             "param",
74             "values"
75         ]
76     }
77 }
78 },
79 "required": [
80     "algorithm"
81 ]
82 },
83 "uniqueItems": true
84 },
85 "parameterSearch": {
86     "type": "string",
87     "enum": [
88         "grid",
89         "random",
90         "bayesian-optimization"
91     ]
92 },
93 "computationalResources": {
94     "type": "object",
95     "properties": {
96         "instances": {
97             "type": "array",
98             "items": {
99                 "type": "object",
100                "properties": {
101                    "ec2Instance": {
102                        "type": "string",
103                        "pattern": "[A-Za-z][0-9][a-z]?(\\d{1,2}x)|x)?(nano|micro|small|medium|large|metal)$"
104                    }
105                },
106                "additionalHardware": {
107                    "type": "array",
108                    "items": {
109                        "type": "string",
110                        "enum": [ "GPU" ]
111                    },
112                    "uniqueItems": true

```

```
113         },
114         "minInstances": {
115             "type": "integer",
116             "minimum": 1,
117             "maximum": 32
118         },
119         "maxInstances": {
120             "type": "integer",
121             "minimum": 1,
122             "maximum": 32
123         }
124     }
125 }
126 },
127 "minTotalInstances": {
128     "type": "integer",
129     "minimum": 1,
130     "maximum": 32
131 },
132 "maxTotalInstances": {
133     "type": "integer",
134     "minimum": 1,
135     "maximum": 64
136 }
137 }
138 },
139 "emailAddress": {
140     "type": "string",
141     "format": "email"
142 }
143 },
144 "required": [
145     "domain",
146     "methods",
147     "emailAddress"
148 ]
149 }
```

Referencias bibliográficas

- [AGMV08] Francisco Almeida, Domingo Giménez, José Miguel Mantas, y Antonio M. Vidal. *Introducción a la programación paralela*. Thompson Paraninfo, 2008.
- [BB12] James Bergstra y Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of machine learning research*, 13(Feb):281–305, 2012.
- [BBBK11] James S Bergstra, Rémi Bardenet, Yoshua Bengio, y Balázs Kégl. Algorithms for hyper-parameter optimization. En *Advances in neural information processing systems*, páginas 2546–2554, 2011.
- [Bra72] Franklin H Branin. Widely convergent method for finding multiple solutions of simultaneous nonlinear equations. *IBM Journal of Research and Development*, 16(5):504–522, 1972.
- [BYC13] James Bergstra, Daniel Yamins, y David Daniel Cox. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. 2013.
- [Chi92] Nancy Chinchor. The statistical significance of the MUC-4 results. En *Proceedings of the 4th conference on Message understanding*, páginas 30–50. Association for Computational Linguistics, 1992.
- [CJ20] Davide Chicco y Giuseppe Jurman. The advantages of the Matthews correlation coefficient (MCC) over F1 score and accuracy in binary classification evaluation. *BMC genomics*, 21(1):6, 2020.
- [DG04] Jeffrey Dean y Sanjay Ghemawat. MapReduce: Simplified data processing on large clusters. 2004.
- [Dix78] Laurence Charles Ward Dixon. The global optimization problem. an introduction. *Towards global optimization*, 2:1–15, 1978.
- [DMC16] Ian Dewancker, Michael McCourt, y Scott Clark. Bayesian optimization for machine learning: A practical guidebook. *arXiv preprint arXiv:1612.04858*, 2016.

- [EFGK03] Patrick Th Eugster, Pascal A Felber, Rachid Guerraoui, y Anne-Marie Kermarrec. The many faces of publish/subscribe. *ACM computing surveys (CSUR)*, 35(2):114–131, 2003.
- [FGJ⁺09] Armando Fox, Rean Griffith, Anthony Joseph, Randy Katz, Andrew Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, y Ion Stoica. Above the clouds: A Berkeley view of cloud computing. *Dept. Electrical Eng. and Comput. Sciences, University of California, Berkeley, Rep. UCB/EECS*, 28(13):2009, 2009.
- [Fla12] Peter Flach. *Machine learning: the art and science of algorithms that make sense of data*. Cambridge University Press, 2012.
- [FSK08] Alexander Forrester, Andras Sobester, y Andy Keane. Engineering design via surrogate modelling: a practical guide, 2008.
- [GAMC19] Vicent Giménez-Alventosa, Germán Moltó, y Miguel Caballer. A framework and a performance assessment for serverless MapReduce on AWS Lambda. *Future Generation Computer Systems*, 97:259–274, 2019.
- [GDG⁺17] Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, y Kaiming He. Accurate, Large Minibatch SGD: Training ImageNet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017.
- [GFD06] Dan Gillick, Arlo Faria, y John DeNero. MapReduce: Distributed computing for machine learning. *Berkley, Dec*, 18, 2006.
- [GKGK03] Ananth Grama, Vipin Kumar, Anshul Gupta, y George Karypis. *Introduction to parallel computing*. Pearson Education, 2003.
- [GLNS⁺05] Jim Gray, David T Liu, Maria Nieto-Santisteban, Alex Szalay, David J DeWitt, y Gerd Heber. Scientific data management in the coming decade. *Acm Sigmod Record*, 34(4):34–41, 2005.
- [GSM⁺17] Daniel Golovin, Benjamin Solnik, Subhodeep Moitra, Greg Kochanski, John Karro, y D Sculley. Google Vizier: A service for black-box optimization. En *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*, páginas 1487–1495, 2017.
- [HFR04] José Hernández, Cèsar Ferri, y María José Ramírez. Introducción a la minería de datos. 2004.
- [HHLB11] Frank Hutter, Holger H Hoos, y Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. En *International conference on learning and intelligent optimization*, páginas 507–523. Springer, 2011.

- [Hil90] Mark D Hill. What is scalability? *ACM SIGARCH Computer Architecture News*, 18(4):18–21, 1990.
- [JSSS⁺19] Eric Jonas, Johann Schleier-Smith, Vikram Sreekanti, Chia-Che Tsai, Anurag Khandelwal, Qifan Pu, Vaishaal Shankar, Joao Carreira, Karl Krauth, Neeraja Yadwadkar, et al. Cloud Programming Simplified: A Berkeley view on serverless computing. *arXiv preprint arXiv:1902.03383*, 2019.
- [Kle08] J Klensin. RFC 5321-Simple Mail Transfer Protocol. *URL http://tools.ietf.org/html/rfc5321*, páginas 03–16, Oct 2008.
- [Luc02] David Luckham. *The power of events*, volume 204. Addison-Wesley Reading, 2002.
- [Mar00] Robert C Martin. Design principles and design patterns. *Object Mentor*, 1(34):597, 2000.
- [Mat75] Brian W Matthews. Comparison of the predicted and observed secondary structure of T4 phage lysozyme. *Biochimica et Biophysica Acta (BBA)-Protein Structure*, 405(2):442–451, 1975.
- [Nyg18] Michael T Nygard. *Release it! Design and deploy production-ready software*. Pragmatic Bookshelf, 2018.
- [OPA05] Julio Ortega, Alberto Prieto, y Mancia Anguita. *Arquitectura de computadores*. Thomson, 2005.
- [PY09] Pitch Patarasuk y Xin Yuan. Bandwidth optimal all-reduce algorithms for clusters of workstations. *Journal of Parallel and Distributed Computing*, 69(2):117–124, 2009.
- [SDB18] Alexander Sergeev y Mike Del Balso. Horovod: fast and easy distributed deep learning in TensorFlow. *arXiv preprint arXiv:1802.05799*, 2018.
- [SSW⁺15] Bobak Shahriari, Kevin Swersky, Ziyu Wang, Ryan P Adams, y Nando De Freitas. Taking the human out of the loop: A review of Bayesian optimization. *Proceedings of the IEEE*, 104(1):148–175, 2015.
- [Van16] Jake VanderPlas. *Python Data Science Handbook: Essential tools for working with data*. O'Reilly Media, Inc., 2016.
- [WHD18] James Wilson, Frank Hutter, y Marc Deisenroth. Maximizing acquisition functions for Bayesian optimization. En *Advances in Neural Information Processing Systems*, páginas 9884–9895, 2018.

Este documento fue editado y tipografiado con L^AT_EX empleando la clase **esi-tfm** (versión 0.20200716) que se puede encontrar en:

https://github.org/arco_group/esi-tfg

