

UNIVERSIDAD POLITÉCNICA DE MADRID  
ESCUELA TÉCNICA SUPERIOR DE INGENIEROS DE  
TELECOMUNICACIÓN

UNIVERSIDAD COMPLUTENSE DE MADRID  
FACULTAD DE CIENCIAS MATEMÁTICAS

MÁSTER EN TRATAMIENTO ESTADÍSTICO COMPUTACIONAL DE  
LA INFORMACIÓN



### TRABAJO DE FIN DE MÁSTER

MLOps y cómo industrializar el uso de Machine Learning: Caso de uso para anticipar el *malware* en sistemas Windows

Gema Correa Fernández

Directora: M. Mar Fenoy Muñoz

Madrid, 2020



## Resumen

Este trabajo tiene como objetivo el desarrollo del enfoque actual de MLOps desde la perspectiva de las empresas en España, y en menor medida en el campo académico. En particular, se abordan distintos aspectos metodológicos y se proponen varias alternativas de arquitectura para llevar a cabo un MLOps de éxito. En términos generales, MLOps se basa en los principios y las prácticas de DevOps para aumentar la eficacia de los flujos de trabajo en proyectos de Machine Learning. En el trabajo se presenta MLOps en un caso real de empresa donde el objetivo de negocio es detectar (predecir) la ocurrencia de un evento - como puede ser el fraude de transacciones *online* o el *malware* informático. A modo de ejemplo se hace uso del dataset *Microsoft Malware Prediction* de Kaggle para predecir si un sistema Windows se va a infectar de *malware* informático, donde se desarrolla cada una de las etapas del proceso de modelización: creación de variables para el modelo, métricas de diagnosis y selección de los modelos de clasificación, para poner finalmente el modelo en producción. Gracias a MLflow, plataforma que permite gestionar todo el ciclo de vida de Machine Learning, y la aplicación web creada, se realiza un consumo de los modelos desplegados. Por el enfoque que se realiza, este trabajo puede resultar de utilidad para aquellas personas que se quieran adentrar en este tema.

**Palabras-clave:** MLOps, MLflow, modelo en producción, despliegue, API, aplicación web, *malware*.

## Abstract

The objective of this project is develop the current approach to MLOps from the perspective of companies in Spain and to a lesser extent in the academic area. In particular, different methodological aspects are addressed and various architectural alternatives are proposed to carry out a successful MLOps. Generally speaking, MLOps builds on DevOps principles and practices to increase the efficiency of workflows in Machine Learning projects. The work presents MLOps in a real case of a company where the business objective is to detect (predict) the occurrence of an event - such as online transaction fraud or computer malware. As an example, the Kaggle dataset (*Microsoft Malware Prediction*) is used to predict if a Windows system is going to be infected by computer malware, where each stage of the modeling process is developed, including creation of variables for the model, diagnostic metrics and model selection classification, to finally put the model into production. Due to MLflow, a platform that allows to manage the entire life cycle of Machine Learning and the Web application created, a consumption of the deployed models is carried out. The approach that is made on this work can be useful for those people who are interested learning of this topic.

**Key-words:** MLOps, MLflow, model in production, deployment, API, web application, malware.



---

Yo, **Gema Correa Fernández**, alumna del Máster en Tratamiento Estadístico Computacional de la Información de la Universidad Politécnica y Complutense de Madrid, con correo [gema.correa.fernandez@alumnos.upm.es](mailto:gema.correa.fernandez@alumnos.upm.es) y [gecorrea@ucm.es](mailto:gecorrea@ucm.es) autorizo la ubicación de la siguiente copia de mi Trabajo Fin de Máster en la biblioteca del centro para que pueda ser consultada por las personas que lo deseen.

Asimismo, el código fuente del proyecto y esta documentación pueden consultarse en la dirección [https://github.com/Gecofer/TFM\\_1920](https://github.com/Gecofer/TFM_1920), para que aquellos que lo deseen puedan conocer el proyecto.



Fdo: Gema Correa Fernández

Madrid a 16 de junio de 2020



# Agradecimientos

A mi familia, amigos y profesores. Gracias a vosotros he crecido tanto en lo personal como en lo profesional. A las tres ciudades donde he vivido, Granada, Valencia y Madrid. Pero sobretodo quiero agradecer a mi apoyo incondicional durante casi 17 años.



# Índice general

<b>1. Introducción</b>	<b>1</b>
1.1. Introducción . . . . .	1
1.2. Definición del problema . . . . .	3
1.3. Motivación . . . . .	3
1.4. Premisas e hipótesis . . . . .	4
1.5. Objetivos . . . . .	4
1.5.1. Objetivo general . . . . .	4
1.5.2. Objetivos específicos . . . . .	5
1.6. Estructura de la monografía . . . . .	5
<b>2. MLOps</b>	<b>7</b>
2.1. MLOps . . . . .	7
2.1.1. Conceptos generales . . . . .	7
2.1.2. Machine Learning . . . . .	8
2.1.3. DevOps . . . . .	10
2.2. ¿En qué se diferencia MLOps de DevOps? . . . . .	10
2.3. Beneficios de usar MLOps . . . . .	11
2.4. Fases de MLOps . . . . .	11
2.4.1. Fase 1: Entrenamiento del modelo . . . . .	12
2.4.2. Fase 2: Empaquetar e implementar modelos . . . . .	12
2.4.3. Fase 3: Automatizar flujos de trabajo, supervisar y administrar .	14
2.4.4. Fase 4: Aplicar gobernanza y control . . . . .	15
2.5. Conclusiones . . . . .	15
<b>3. Implementación de MLOps</b>	<b>17</b>
3.1. Introducción . . . . .	17
3.2. Ciclo de vida del proyecto en la parte de ML . . . . .	17
3.2.1. Fase I: <i>Business Understanding</i> . . . . .	18
3.2.2. Fase II: <i>Data Understanding</i> . . . . .	18
3.2.3. Fase III: <i>Data Preparation</i> . . . . .	19
3.2.4. Fase IV: <i>Modeling</i> . . . . .	19
3.2.5. Fase V: <i>Evaluation</i> . . . . .	22
3.2.6. Fase VI: <i>Deployment</i> . . . . .	24
3.3. Puesta en producción del modelo . . . . .	24
3.3.1. Caso de uso en local . . . . .	24
3.3.2. Caso de uso en el cloud . . . . .	28
3.4. Funcionamiento de la aplicación web . . . . .	31
3.5. Conclusiones . . . . .	34

<b>4. Conclusiones y Trabajo Futuro</b>	<b>35</b>
4.1. Conclusiones . . . . .	35
4.2. Trabajo futuro . . . . .	37
<b>A. Planificación del TFM</b>	<b>39</b>
<b>B. Herramientas</b>	<b>41</b>
<b>C. Análisis descriptivo de los datos</b>	<b>43</b>
<b>D. Preprocesamiento de los datos</b>	<b>45</b>
<b>E. AutoML</b>	<b>51</b>
<b>Bibliografía</b>	<b>55</b>

# Índice de figuras

1.1. MLOps - Microsoft Azure [19] . . . . .	1
1.2. Ejemplos de Machine Learning Automatizado . . . . .	2
2.1. Composición de MLOps [15, 14] . . . . .	8
2.2. Proceso general ML . . . . .	8
2.3. Fases del proceso de CRISP-DM [3] . . . . .	9
2.4. Proceso DevOps . . . . .	10
2.5. DevOps vs. ML [16] . . . . .	10
2.6. Fases de MLOps [5] . . . . .	11
2.7. Fase 1 de MLOps [15] . . . . .	12
2.8. Fase 2 de MLOps [15] . . . . .	12
2.9. Elementos de MLflow [14] . . . . .	13
2.10. MLflow <i>Tracking</i> [14] . . . . .	13
2.11. MLflow <i>Projects</i> [14] . . . . .	13
2.12. MLflow <i>Models</i> [14] . . . . .	13
2.13. Fase 3 de MLOps [15] . . . . .	15
2.14. Fase 4 de MLOps [15] . . . . .	15
3.1. Entropía . . . . .	21
3.2. Métricas del algoritmo Random Forest . . . . .	22
3.3. Métricas del algoritmo Regresión Logística . . . . .	22
3.4. Métricas del algoritmo Gradient Boosting . . . . .	22
3.5. Variables más importantes para Random Forest . . . . .	23
3.6. Variables más importantes para Gradient Boosting . . . . .	23
3.7. Variables más importantes para Regresión Logística . . . . .	23
3.8. Esquema de la arquitectura de MLOps en local . . . . .	24
3.9. Página de inicio de MLflow . . . . .	25
3.10. Página seleccionando las ejecuciones para hacer la comparación . . . . .	26
3.11. Página con la comparativa de las tres ejecuciones seleccionadas . . . . .	26
3.12. Página con información detallada de la ejecución de un modelo . . . . .	26
3.13. Página de información con el fichero YAML del modelo . . . . .	27
3.14. Esquema de la arquitectura de MLOps en el cloud . . . . .	28
3.15. Crear la instancia del modelo en MLflow . . . . .	28
3.16. Integración de MLflow en Azure Databricks . . . . .	29
3.17. Modelo registrado en Azure ML Workspace . . . . .	30
3.18. Implementación del modelo . . . . .	30
3.19. Consulta al modelo a través de Postman . . . . .	31
3.20. Página de inicio de la aplicación . . . . .	32

3.21. Página para realizar la petición POST a la API del modelo de Regresión Logística (rellenar formulario) . . . . .	33
3.22. Página que devuelve la probabilidad de que un sistema Windows se infecte haciendo uso del modelo de Regresión Logística . . . . .	33
3.23. Página que devuelve la probabilidad de que un sistema Windows se infecte haciendo uso del modelo de Random Forest (para ello se ha completado el formulario que realiza la petición POST a la API del modelo) . . . . .	33
3.24. Página que devuelve la probabilidad de que un sistema Windows se infecte haciendo uso del modelo de Gradient Boosting . . . . .	34
3.25. Salida en terminal de las peticiones POST realizadas . . . . .	34
 B.1. Especificaciones de MacOS . . . . .	41
 C.1. Gráfico con la distribución de la variable <i>target HasDetections</i> para el conjunto de datos de aprendizaje . . . . .	43
C.2. Tabla con las variables que tienen más de 60 % de <i>missings</i> . . . . .	44
C.3. Tabla con variables más asimétricas . . . . .	44
 D.1. Información de la variable <i>OsBuildLab</i> . . . . .	45
D.2. Información de la variable <i>Census_InternalBatteryType</i> . . . . .	46
D.3. Información de la variable <i>Census_OsBuildNumber</i> . . . . .	46
D.4. Variable <i>SmartScreen</i> sin tratar mayúsculas . . . . .	48
D.5. Variable <i>SmartScreen</i> tratando mayúsculas . . . . .	48
D.6. Errores ortográficos en la variable <i>SmartScreen</i> . . . . .	48
D.7. Variable <i>Census_PrimaryDiskTypeName</i> sin y con la unificación de <i>unspecified</i> con <i>unknown</i> . . . . .	48
D.8. Ejemplo gráfico con la idea de Label Encoding [7] . . . . .	49
D.9. Ejemplo gráfico con la idea de One Hot Encoding [8] . . . . .	49
D.10. Correlación entre <i>OsVer</i> y <i>Platform</i> . . . . .	49
 E.1. Paso 1 en AutoML - Cargar los datos . . . . .	51
E.2. Paso 2 en AutoML - Definir tipo y datos a usar . . . . .	52
E.3. Paso 3 en AutoML - Indicar la variable objetivo a predecir . . . . .	52
E.4. Paso 4 en AutoML - Crear ejecución con ML automatizado . . . . .	53
E.5. Paso 5 en AutoML - Ejecutar ejecución del ML automatizado . . . . .	53

# Glosario

**Análisis de Componentes Principales (PCA).** Técnica matemática utilizada para reducir la dimensionalidad de un conjunto de datos hallando las causas de variabilidad del conjunto y ordenándolos por importancia.

**API REST.** Capa de abstracción para que dos sistemas se comuniquen. En el ámbito web, una API es un servicio *backend* que se utiliza para conectar dos aplicaciones.

**Cloud.** Término genérico que describe el suministro de soluciones *hardware* o *software* a través de Internet, donde los usuarios pueden contratar la potencia de procesamiento, el espacio de almacenamiento o los entornos de *software* que necesiten.

**Correlación.** Técnica estadística que indica si dos variables están relacionadas o no.

**Data Mining.** Técnica que consiste en extraer información de un conjunto de datos y transformarla en una estructura comprensible, esto es, en información útil y accesible para que pueda ser usada posteriormente.

**Data Science.** Ciencia centrada en el estudio de los datos que se encarga en extraer información de grandes cantidades de datos.

**DevOps.** Metodología de trabajo basada en el desarrollo de código que usa nuevas herramientas y prácticas para reducir la tradicional distancia entre técnicos de programación y de sistemas, permitiendo mayor colaboración, agilidad y productividad.

**Framework.** Esquema para el desarrollo y/o la implementación de una aplicación.

**Gradient Boosting.** Técnica de aprendizaje automático que conduce a modelos de predicción que combinan conjuntos de modelos de predicción más débiles, normalmente árboles de decisión. Se basan en ajustar modelos sucesivamente de manera que cada uno va dando mayor peso a las observaciones peor clasificadas.

**Imputación de datos.** Técnica que sustituye los datos *missings* por estimaciones.

**Industrialización de modelos.** Convertir los modelos de aprendizaje automático e Inteligencia Artificial en herramientas productivas y no sólo en temas de investigación.

**Information Technology (IT).** Uso de cualquier computadora, almacenamiento, redes y otros dispositivos físicos, infraestructura y procesos para crear, procesar, almacenar, proteger e intercambiar todas las formas de datos electrónicos.

**Inteligencia Artificial (IA).** Cualquier tecnología que permite que un sistema demuestre inteligencia similar a la humana.

**Kaggle.** Plataforma online para realizar competiciones de *Data Mining*, con un repositorio online para que las empresas publiquen sus datos y realizar un concurso abierto para que los expertos en *Data Mining* de todo el mundo descarguen esos datos y propongan soluciones a los problemas de la empresa en cuestión.

**Label Encoding.** Enfoque que identifica los distintos valores existentes en las variables y sustituye cada uno de ellos por un número.

**Machine Learning (ML).** Rama de la Inteligencia Artificial que cubre la parte estadística y resuelve problemas observando cientos de datos (aprendizaje automático).

**Machine Learning Automatizado (MLA).** Herramienta que permite preparar modelos de aprendizaje automático personalizados y de alta calidad, sin apenas esfuerzo y sin necesidad de tener conocimientos avanzados en la materia.

**Malware.** Programa que se crea con la intención de dañar dispositivos, robar datos y, en general, causar problemas. Los virus, los troyanos, el *spyware* y el *ransomware* son algunos de los distintos tipos de malware.

**Microsoft Azure.** Servicio de computación en la nube creado por Microsoft para construir, probar, desplegar y administrar aplicaciones y servicios mediante el uso de sus centros de datos.

**Datos missing.** Datos que no se almacenan para una variable en la observación de interés.

**MLOps.** Técnica recientemente emergente que incluye una combinación de personas, procesos y tecnología que brindan una ventaja para optimizar e implementar modelos de aprendizaje automático de manera rápida y segura.

**One Hot Encoding.** Enfoque que crea una columna para cada valor distinto que existe en la variable que se está codificando y, para cada registro, marca con un 1 la columna a la que pertenezca dicho registro y deja las demás a 0.

**Partición de datos.** Técnica que permite dividir el dataset en subdatasets con el objetivo de hacer uso de distintos datos en el entrenamiento, validación y prueba.

**Producción de modelos.** Es el último paso del proceso de *Data Mining* y consiste en implementar los modelos que funcionan mejor en un entorno de producción.

**Random Forest.** Técnica de aprendizaje automático que nace como mejora sustancial de los árboles simples.

**Regresión Logística.** Técnica analítica que permite relacionar funcionalmente una variable dicotómica con un conjunto de variables independientes. Es una extensión de los modelos de regresión lineal, con la particularidad de que el dominio de salida de la función está acotado al intervalo [0,1].

**Variable dummy.** La categorización *dummy* consiste en la generación de variables dicotómicas para las distintas categorías de la variable.

# Capítulo 1

## Introducción

**RESUMEN:** Este capítulo define los límites del proyecto: problema a resolver, motivos de su elección, objetivos previstos y estructura. Teniendo como propósito poner de manifiesto la necesidad de poner en producción aplicaciones de Machine Learning (ML) en diversos entornos.

### 1.1. Introducción

El trabajo que se presenta a continuación es un estudio y experimentación de porqué es necesaria la puesta en marcha de algoritmos de ML e Inteligencia Artificial (IA) en un entorno corporativo, conceptos que deben ser introducidos en el ámbito académico. Es un hecho, que las nuevas tecnologías han permitido incorporar aplicaciones de ML en las empresas, en donde el número de personas que hacen uso de estas técnicas y los modelos disponibles en instituciones, como los bancos, crecen de manera exponencial. Sin embargo, la puesta en producción de modelos ML es una preocupación y un desafío en sí mismo. La dificultad de integrar ML en una aplicación no reside en la tecnología, las matemáticas, la ciencia o los algoritmos. El desafío se encuentra en implementar el modelo en un entorno de producción y mantenerlo operativo y soportable. Para que este proceso se lleve a cabo, es necesaria la colaboración de diversos equipos (figura 1.1). Por eso, es de suma importancia introducir todas estas ideas en el campo académico, con el fin de formar a profesionales competentes.



Figura 1.1: MLOps - Microsoft Azure [19]

Los equipos de desarrollo de software saben cómo integrar aplicaciones (empresariales) y servicios en la nube. Los equipos de ML saben cómo desarrollar modelos para transformar un negocio. La unión de ambas partes, permite incorporar las buenas prácticas desarrolladas por los ingenieros de desarrollo de software en la puesta en producción de modelos de ML/IA [12]. Esta unión se conoce como MLOps (*Machine Learning and Operations*) o DevOps para Machine Learning, y consiste en permitir a los equipos de ciencia de datos e IT (*Information Technology*) colaborar, aumentar y mejorar el ritmo de desarrollo e implementación de modelos ML/IA a través del monitoreo, supervisión, validación y gobernanza de los modelos de aprendizaje automático [19]. En términos generales, MLOps se basa en los principios y las prácticas de DevOps para aumentar la eficacia de los flujos de trabajo en los proyectos de Machine Learning.

Actualmente, cada vez son más las organizaciones que están desarrollando modelos de ML para su uso interno. La incorporación de equipos de ciencias de datos combinada con el avance tecnológico ha provocado una explosión de modelos disponibles. Sin embargo, muchas organizaciones no cuentan con un marco de trabajo (*framework*) estable que permita contar con un inventario de modelos, realizar un seguimiento automático de su calidad predictiva, asegurar su robustez y decidir si requiere ser reentrenado. Según Gartner en [6], “muchas empresas desarrollan modelos de ML, pero solo el 47 % se ponen en producción. Y aún así, el 88 % de las iniciativas de IA tienen dificultades para pasar la etapa de producción”. De manera que está en nuestra mano, incorporar esta nueva forma de trabajar al entorno en el que nos movemos.

Otra de las grandes ventajas que conlleva este avance tecnológico, es la aparición de herramientas que permitan preparar modelos de aprendizaje automático personalizados y de alta calidad, sin apenas esfuerzo y sin necesidad de tener conocimientos avanzados en la materia. Un ejemplo de ello, es AutoML de Microsoft [18], sin embargo, existen otras, tanto libres como no libres, las cuales podemos observar en la figura 1.2.

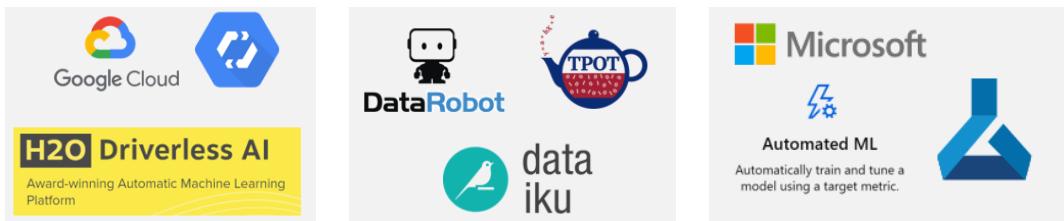


Figura 1.2: Ejemplos de Machine Learning Automatizado

Dichas herramientas permiten disponer de miles de modelos, sin apenas la intervención de un analista. A pesar de que representan un cambio fundamental en la forma en que las organizaciones adoptan la ciencia de datos, no se puede olvidar la importancia que tiene el análisis y tratamiento de los datos por parte de un profesional (científico de datos) qué entienda el problema de negocio, garantice la calidad de los datos utilizados, interprete las salidas de los algoritmos/herramientas y sobre todo verifique que los resultados son coherentes, proporcionando el valor esperado y, por lo tanto, puestos a disposición de toda la organización (es decir, en producción). Además, el

Machine Learning Automatizado (MLA) no garantiza encontrar el algoritmo óptimo<sup>1</sup>, ya que por defecto está creado para ejecutar una serie de algoritmos de acuerdo a unos parámetros establecidos.

## 1.2. Definición del problema

Las operaciones de aprendizaje automático (MLOps) se refieren al uso de modelos de aprendizaje automático por parte de los equipos de desarrollo/operaciones (DevOps). Esto permite, introducir un conjunto de mejoras prácticas a la hora de desarrollar y poner en producción modelos de ML. Dichas mejoras serán detalladas en el capítulo 2 y empleadas en el ejemplo práctico descrito en el capítulo 3. En consecuencia, debemos saber que todo proceso de MLOps no es posible sin unos datos de calidad. Partiendo de estas ideas, vamos a definir el problema que abarcaremos a lo largo de todo el trabajo.

El trabajo tiene como objetivo poner en producción modelos de ML/IA, abordando aspectos metodológicos y proponiendo alternativas de arquitectura (herramientas y configuración del sistema) para llevar a cabo un MLOps de éxito, tanto en local como en el *cloud*. Para ello, será necesario mostrar cómo se desarrolla cada una de las etapas del proceso de modelización y la introducción de MLflow (<https://www.mlflow.org>), plataforma que nos permite gestionar todo el ciclo de vida de Machine Learning. A modo de ejemplo, se hace uso del conjunto de datos de Kaggle, *Microsoft Malware Prediction*, dataset compuesto por 9 millones de datos<sup>2</sup> donde se quiere detectar (predecir) si un sistema Windows se ha infectado de malware en función de las diferentes propiedades de dicho dispositivo, a partir de la información recopilada por la solución de protección de Microsoft, Windows Defender. Gracias a este conjunto de datos podremos implementar y explicar el enfoque actual de MLOps.

## 1.3. Motivación

*¿Qué motivación y razones me llevan a comenzar un proyecto de tales características?* Como Ingeniera Informática especializada en Computación y Sistemas Inteligentes por la UGR, y actualmente estudiante del Máster de Tratamiento Estadístico Computacional de la Información impartido conjuntamente por las Universidades Complutense y Politécnica de Madrid, he adquirido a lo largo de mis años de formación tanto académica como profesional, diversos conocimientos y usos que se le puede dar a la tecnología. En mi último año, donde me he especializado en áreas como Big Data y/o Data Science, me he planteado cuestiones acerca de la posibilidad de combinar la idea de ML con la idea que reside por detrás de la puesta en producción de un proyecto *software*, es decir, dotar de más funcionalidad a la creación de modelos ML/IA. En concreto, la elección de este tema surge de intentar unir dos propósitos tanto personales como profesionales:

1. Combinar dos de los ámbitos que más me interesan: Big Data, ML y *Data Science* con mi formación en informática.

---

<sup>1</sup>Este hecho ha podido ser contrastado con las herramientas H2O en R y AutoML de Microsoft.

<sup>2</sup>Se va hacer uso de unos datos de gran tamaño, es decir, lo que muchas empresas considerarían como Big Data para obtener un funcionamiento real de un caso real.

2. Realizar una aplicación técnica enfocada al mundo laboral más allá del ámbito pedagógico, haciendo uso de conocimientos adquiridos.

La idea de realizar un trabajo de esta envergadura se manifestó antes de su propia elección, puesto que durante mi formación universitaria siempre he querido realizar un proyecto de tales características, relacionado con el mundo que engloba Big Data, ML o IA. Gracias a mi experiencia laboral y mis estudios, me he beneficiado enormemente de la disponibilidad de acceder a los recursos de Microsoft Azure, lo cual ha sido de utilidad para el aprendizaje, desarrollo e implantación de la aplicación de MLOps. Debido a que en Microsoft Azure es posible poner en producción un modelo ML, gracias a la integración de la herramienta MLflow en Azure Databricks [31]. Lo que ha permitido, realizar comparaciones y ver como se trabaja el enfoque de MLOps tanto de manera local como en el *cloud* con Microsoft Azure. Siendo éste, un proyecto que enriquezca tanto a una empresa como a una Universidad, además de a mi formación. En los sucesivos capítulos, iremos detallando cada uno de los conceptos que aquí mencionamos.

## 1.4. Premisas e hipótesis

La realización de este trabajo ha partido de varios supuestos básicos: el primero de ellos parte de la posibilidad de combinar las fases del proceso de Machine Learning con las fases de un proyecto de desarrollo de IT, haciendo uso del lenguaje de programación Python y distintos *frameworks*. Asumiendo de antemano, que Python permitirá realizar las fases del proceso de ML (preprocesamiento, creación de modelos) y la implementación de la aplicación web. El segundo supuesto es un hecho, y consiste en la utilidad de poner en producción modelos en inmesidad de aplicaciones. El tercer supuesto parte de implementar la solución de MLOps tanto en local como en *cloud*, en donde, la curva de aprendizaje será distinta en función del entorno escogido. Por último, MLOps es una práctica relativamente reciente pero que merece la máxima difusión, siendo desconocida para gran parte de los profesionales dedicados en estas áreas.

## 1.5. Objetivos

Este apartado recoge los objetivos iniciales marcados para la realización del proyecto, especificando los propósitos que se esperan conseguir del mismo. A continuación, se detallan tanto los objetivos generales como específicos para el TFM.

### 1.5.1. Objetivo general

En este trabajo se **desarrolla el enfoque actual de MLOps, desde la perspectiva de las empresas en España y su posible introducción al campo académico**. Además, se abordan distintos aspectos metodológicos, donde se proponen varias alternativas de arquitectura (herramientas y configuración del sistema) para llevar a cabo un MLOps de éxito. Asimismo, se presenta MLOps en casos reales de empresas donde el objetivo de negocio es detectar (predecir) la ocurrencia de un evento como es la detección de *malware* informático. Para ello, será necesario mostrar cómo se desarrolla cada una de las etapas del proceso de modelización, incluyendo las opciones de creación de variables para el modelo, métricas de diagnosis y selección de modelos de

clasificación. Además, de introducir MLflow, plataforma que permitirá gestionar todo el ciclo de vida de los modelos de aprendizaje automático.

### 1.5.2. Objetivos específicos

En el subapartado anterior, se menciona el objetivo general que vamos a lograr con dicho trabajo. En este subapartado, detallaremos los objetivos más específicos del proyecto, en donde al final del documento comprobaremos que se han cumplido:

1. Presentar una **propuesta de industrialización de modelos de Machine Learning** conocida como MLOps.
2. Entender cómo se utiliza la **ciencia de datos en un entorno empresarial y académico**.
3. Conocer las diferentes alternativas que existen en el mercado para la **adopción de la ciencia de datos con tecnologías cloud**.
4. Desarrollar un caso de uso de **clasiﬁcación predictiva con algoritmos de Machine Learning**.
5. Conocer las capacidades que ofrecen diversas herramientas **para la implementación de modelos ML/IA**.
6. Conocer y mostrar **alternativas de arquitectura para llevar a cabo el despliegue o puesta en producción del servicio web (modelos de Machine Learning)** utilizando la imagen del contenedor del modelo.
7. **Aportaciones a la comunidad o al lector**, llevando a cabo una aplicación para implementar el proceso de MLOps.
8. **Adquisición de nuevos conocimientos** en el ecosistema de Microsoft Azure o de herramientas como Python, Spark, Django, Flask o MLflow. Además, de aprendizaje en el análisis de datos de gran dimensionalidad, creación e implementación de modelos, creación de una API REST conectada al modelo y despliegue del mismo.

## 1.6. Estructura de la monografía

En este primer capítulo, se ha desarrollado una pequeña introducción al contexto que desarrolla este trabajo, puntuizando en los motivos de la elección del mismo. Además, previamente se han tratado algunos de los términos esenciales en un glosario, necesarios para comprender el desarrollo del proyecto. A continuación, se detallan de forma resumida los contenidos del resto de capítulos de este documento:

- En el *capítulo 2 (MLOps)*, se contextualiza el trabajo, presentando que es MLOps, porqué nace, de dónde viene, cómo se construye y qué herramientas permiten llevárselo a cabo, entre otros. Además, veremos brevemente el estado del arte sobre ciertos aspectos relacionados con el proyecto.

- En el *capítulo 3 (Implementación de MLOps)*, se realiza una demo (aplicación) de cómo sería la construcción de la arquitectura MLOps o puesta en producción de algoritmos Machine Learning a través de diferentes casos de uso. Realizando énfasis en el proceso llevado a cabo por el *Data Scientist*.
- En el *capítulo 4 (Conclusiones y Trabajo Futuro)*, se evalúan las propuestas realizadas en el capítulo 3, recopilando tanto lo que se ha hecho a lo largo del desarrollo de este trabajo, como las conclusiones y resultados finales obtenidos de esta experiencia. Se comentan además, una serie de avances para aplicar en el futuro y seguir desarrollando esta aplicación.
- En el *apéndice A (Planificación del TFM)*, se presenta la planificación realizada para el desarrollo del Trabajo de Fin de Máster.
- En el *apéndice B (Herramientas)*, se mencionan las herramientas usadas en el proyecto.
- En el *apéndice C (Análisis descriptivo de los datos)*, se detalla el análisis descriptivo realizado a los datos con el fin de extraer información y conocimiento.
- En el *apéndice D (Preprocesamiento de los datos)*, se citan las técnicas aplicadas en el preprocesamiento de los datos para la obtención de unos datos de calidad.
- En el *apéndice E (AutoML)*, se nombra el proceso a seguir para hacer uso de técnicas de Machine Learning automatizadas.
- Por último, se incluye la **Bibliografía** con los enlaces consultados y usados para la realización del trabajo.
- **En el enlace [https://github.com/Gecofer/TFM\\_1920](https://github.com/Gecofer/TFM_1920) se encuentra el código implementado y documentación adicional del proyecto.**

# Capítulo 2

## MLOps

**RESUMEN:** Este capítulo presenta de forma teórica el enfoque de MLOps. La información explicada a lo largo de este capítulo sirve como base para el desarrollo del ejemplo práctico de MLOps en el capítulo 3. De esta manera, dividiremos el capítulo en dos grandes apartados: el primero de ellos se centra en el proceso de ML y, en el segundo, se mencionan las técnicas y mejoras prácticas integradas en MLOps, gracias al enfoque de DevOps.

### 2.1. MLOps

A continuación, vamos a introducir la definición de MLOps, el contexto dónde nace, qué valor aporta a los procesos actuales de *Data Science*, cómo se integra en DevOps y/o las fases por las que está constituido (conocimientos adquiridos tanto de forma académica como laboral). Para finalmente realizar con éxito el despliegue y la puesta en producción de los modelos que explicaremos en el siguiente capítulo.

#### 2.1.1. Conceptos generales

Las tecnologías de ML e IA permiten ampliar las capacidades de las aplicaciones de *software* que usamos en nuestro día a día: asistentes digitales, reconocimiento facial, servicios bancarios o recomendaciones de productos [15]. De dicha dependencia, surge la necesidad de combinar las prácticas y herramientas usadas en los equipos de *Data Science* y *Operations* (figura 1.1). Esta unión posibilita que el enfoque de MLOps o DevOps para Machine Learning ayude a los equipos de ciencia de datos e IT a colaborar y aumentar el ritmo de desarrollo e implementación de modelos a través de la supervisión, validación y gobernanza de los modelos de aprendizaje automático, es decir, permite la colaboración entre diversos usuarios (*Data Scientists*, *Data Engineers*, *Business Analysts* e *ITOps*) en las operaciones de ML [19].

MLOps establece una cultura y un entorno donde las tecnologías de ML generan beneficios comerciales al optimizar el flujo de vida de ML para automatizar y escalar el rendimiento empresarial en la producción [15]. Todo este proceso se puede ver desde una perspectiva general conocida como **arquitectura MLOps** (figura 2.1), que permite combinar diversas técnicas. Antes de entrar en detalle con el enfoque de MLOps, veamos qué significan cada una de las disciplinas por las que está formado el término MLOps: Machine Learning y DevOps.

$$\text{MLOps} = \text{ML} + \text{DEV} + \text{OPS}$$



Figura 2.1: Composición de MLOps [15, 14]

### 2.1.2. Machine Learning

El primer término que vamos a tratar es **Machine Learning**. Es una disciplina científica del ámbito de la IA que crea sistemas que aprenden automáticamente mediante la identificación de patrones complejos observando cientos de datos, y permiten predecir comportamientos futuros [28]. Para llevar a cabo el ciclo de vida de un proyecto de ML, podemos hacer uso diversas metodologías: CRISP-DM (*Cross Industry Standard Process for Data Mining*) o SEMMA (*Sample, Explore, Modify, Model, and Assess*). A pesar de que entre ambas existe un paralelismo, en la literatura se contempla usar la **metodología CRISP-DM** debido a que se tiene en cuenta la aplicación del modelo al entorno de negocio [24], considerándose así más completa(figura 2.2).

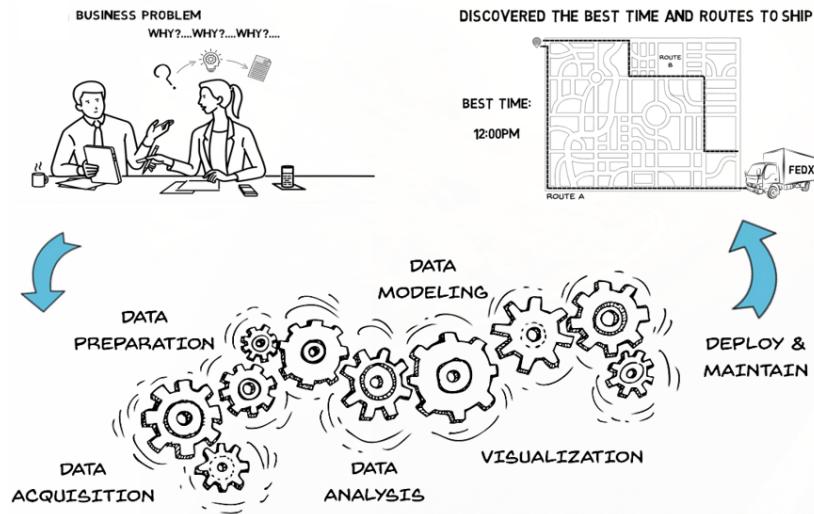


Figura 2.2: Proceso general ML

La metodología CRISP-DM describe los enfoques que utilizan los expertos para abordar los problemas de *Data Mining* de forma tradicional. Este proceso consta de 6 fases (figura 2.3), donde el resultado de cada fase determina qué fase o qué tarea de una fase hay que hacer después. Además, en este método se puede ir hacia adelante o hacia

atrás entre las distintas fases. Sin embargo, el círculo externo de la figura 2.3 simboliza la naturaleza cíclica de los proyectos de análisis de datos. Por otro lado, es importante destacar que el proyecto no se termina una vez entrenado o desplegado el modelo, sino que los expertos en *Data Mining* usan los resultados exportándolos a bases de datos o a cualquier otra aplicación [24, 3]. A continuación, vamos a describir brevemente cada una de las fases que componen la metodología CRISP-DM [24, 2]:

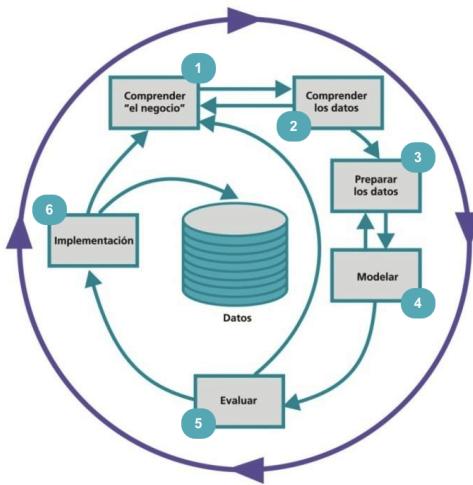


Figura 2.3: Fases del proceso de CRISP-DM [3]

1. **Fase I: Business Understanding.** En esta fase se definen las necesidades del cliente o comprensión del negocio con el fin de comprender los objetivos y requisitos del proyecto desde una perspectiva comercial, para convertir esa información en un problema de minería de datos.
2. **Fase II: Data Understanding.** En esta fase se estudian y comprenden los datos con el fin de familiarizarse con ellos y descubrir información interesante.
3. **Fase III: Data Preparation.** En este fase se analizan los datos y se realiza la selección de características para construir el conjunto final de datos (transformación y limpieza de datos), es decir, se obtienen los datos que se utilizarán en las herramientas de modelado (crear el modelo de datos para la fase de modelado).
4. **Fase IV: Modeling.** En esta fase se seleccionan y aplican técnicas de modelado hasta alcanzar los parámetros que devuelvan valores óptimos. La fase de modelado y la fase de evaluación están acopladas.
5. **Fase V: Evaluation.** En esta fase, se evalúan los modelos obtenidos en la fase anterior con el fin de obtener un modelo final que se adapte a los requisitos y objetivos expuestos en la fase I. Si el modelo no satisface las expectativas, se vuelve a la fase de modelado y se reconstruye el modelo cambiando los parámetros hasta que se consigan los valores óptimos (o adecuados).
6. **Fase VI: Deployment.** En esta fase se realiza el despliegue o puesta en producción del modelo final, donde se usan los resultados en bases de datos o en cualquier otra aplicación. Cómo por ejemplo, generando un informe con información detallada de la salida del modelo.

### 2.1.3. DevOps

El segundo término a estudiar es lo que se conoce como **DevOps**. DevOps es un conjunto de prácticas que impulsan una integración perfecta entre *development* (desarrollo) y *operations* (operaciones) (figura 2.4). Describe los enfoques para agilizar los procesos, en donde una idea pasa del desarrollo a la implementación en un entorno de producción que genera valor para el usuario. Esta práctica permite poner en producción nuevas versiones de *software*

de aplicación, permitiendo mayor colaboración, agilidad y productividad [23].

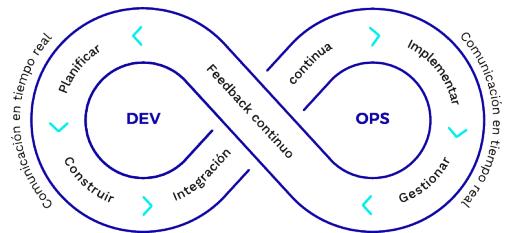


Figura 2.4: Proceso DevOps

## 2.2. ¿En qué se diferencia MLOps de DevOps?

Una vez que se han comprendido las disciplinas que conforman MLOps, se asimila que el proceso de ML forma parte en su totalidad del enfoque de MLOps. En términos generales, el proceso de creación de IA implica trabajar con muchos datos, limpiarlos, escribir y ejecutar experimentos, publicar modelos y finalmente recopilar datos del mundo real y mejorar los modelos. Entonces, *¿por qué es necesario usar MLOps y no ML? ¿cómo influye DevOps en MLOps?* Cómo hemos visto, DevOps es el estándar de desarrollo esencial para servicios en la nube que pone énfasis en el proceso, la automatización y la colaboración entre los equipos. Por tanto, los proyectos de IA tienen sus propias metodologías de desarrollo donde los equipos utilizan un enfoque exclusivo para proyectos de ciencia de datos con iteraciones pequeñas y frecuentes para refinar las características de los datos, el modelo y la pregunta analítica, habiendo poca interacción con un equipo de operaciones [5]. Y DevOps permite que esas metodologías se basen en principios y prácticas aprendidas de proyectos reales (figura 2.5).

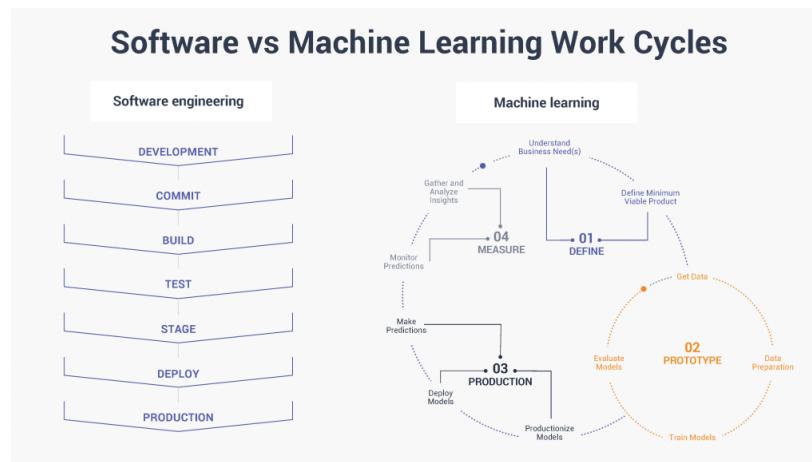


Figura 2.5: DevOps vs. ML [16]

De modo que MLOps surge de juntar dos metodologías independientes con un

objetivo común: **poner una aplicación de IA en producción**. Así, los proyectos de IA deben incorporar algunas de las prácticas operativas y de implementación que hacen que DevOps sea efectivo y, en cambio, los proyectos de DevOps deben adaptarse al proceso de desarrollo de ML/IA para automatizar el proceso de implementación y lanzamiento de los modelos de ML [5, 15].

### 2.3. Beneficios de usar MLOps

A través de las prácticas y las herramientas, MLOps permite establecer una cultura y un entorno donde las tecnologías de ML generan beneficios comerciales, gracias a la colaboración y comunicación entre los científicos de datos (expertos en algoritmos, matemáticas, simulaciones, herramientas de desarrollo) y los profesionales de la tecnología de la información (expertos en despliegues de producción, actualizaciones, gestión de recursos y datos, seguridad), al tiempo que automatiza y produce algoritmos de aprendizaje automático. Por tanto, la necesidad de implementar nuevos modelos y algoritmos sin problemas y sin tiempo de inactividad, es un factor a tener en cuenta [30]. Asimismo, el reentrenamiento frecuente puede marcar la diferencia entre una predicción óptima que tenga en cuenta la historia reciente y una predicción subóptima. Por lo cual, algunas de las ventajas de usar MLOps son [15]:

- Creación de **modelos reproducibles**, permitiendo retrocesos (en caso de errores) y auditorías si se requiere seguimiento.
- MLOps hereda de DevOps conceptos como **validaciones** automatizadas, pruebas, creación de perfiles y gestión del entorno. Además, permite realizar comparaciones del rendimiento esperado y recopilar información para futuras mejoras mediante la **automatización y observación**.

### 2.4. Fases de MLOps

De los párrafos anteriores, se ha podido extraer la necesidad de disponer de un enfoque híbrido en la era moderna de la IA, como es MLOps. Básicamente, la contextualización de este enfoque viene determinada por la distinción de las fases que constituyen su arquitectura (figura 2.6).

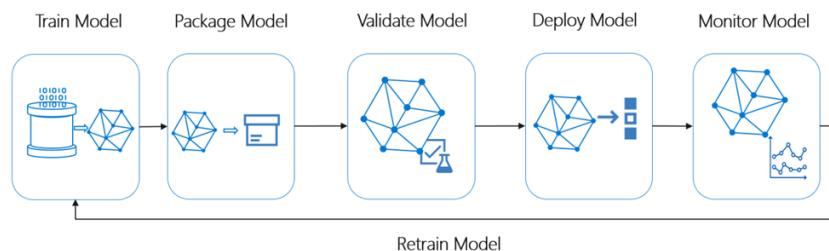


Figura 2.6: Fases de MLOps [5]

Entonces, *¿por qué usar MLOps?*. Para poder responder a esta pregunta veamos las fases que componen este conjunto de prácticas que proporcionan determinismo,

escalabilidad, agilidad y gobernanza en el desarrollo del modelo y la implementación del *pipeline*<sup>1</sup> [1]. Es importante destacar que dependiendo de la literatura podemos distinguir entre 4 ó 5 fases, puesto que la fase de validación del modelo suele estar incluida por defecto en la de empaquetar el modelo.

#### 2.4.1. Fase 1: Entrenamiento del modelo

Previo al entrenamiento del modelo se deberán de haber realizado las 3 primeras fases del proceso de ML, es decir, disponer de los datos tratados y “limpios”. Entonces, en esta fase se entrena el modelo, realizando el mismo proceso que en la fase IV del ML (figura 2.7). En este caso, los científicos de datos utilizan una variedad de lenguajes, *frameworks*, herramientas e IDE para desarrollar modelos utilizando una interfaz, un *framework* y el lenguaje que tenga más sentido para la tarea en cuestión. Cómo, por ejemplo, la creación de un modelo prototípico en un *notebook* de Jupyter, en donde dicho modelo se entrena haciendo uso de la biblioteca **scikit-learn**<sup>2</sup> [1] (en el apéndice B se mencionan las herramientas usadas y su porqué). Adicionalmente, se hará uso de un repositorio como Github<sup>3</sup> o Azure DevOps para tener un control de códigos [15] y disponer gracias a la integración de estas herramientas, tanto en local como en el *cloud*, de la orquestación de un *pipeline* que permita desarrollar todas las fases de MLOps con éxito. Asimismo, se dispondrá de un “lugar” para guardar los datos con los que entrenar y validar el modelo, cómo puede ser Azure Blob Storage en el *cloud*.

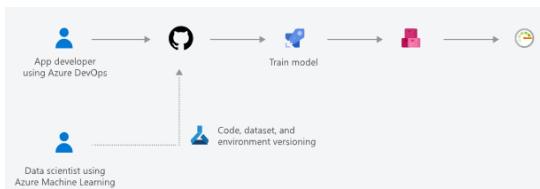


Figura 2.7: Fase 1 de MLOps [15]

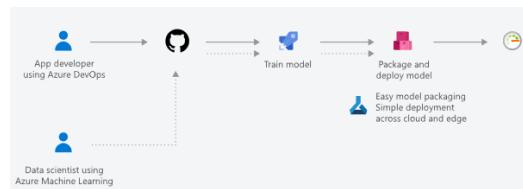


Figura 2.8: Fase 2 de MLOps [15]

#### 2.4.2. Fase 2: Empaquetar e implementar modelos

Después de haber seleccionado el modelo óptimo, en esta fase se empaqueta dicho modelo en una instancia, como puede ser un contenedor<sup>4</sup>, donde se llevará a cabo su implementación (figura 2.8). Sin embargo, tanto en la fase de entrenamiento como en ésta es necesario validar dicho modelo. En el primer caso, se validará para obtener el modelo más óptimo y en el segundo caso, para garantizar que los modelos funcionan como se espera una vez que se implementan tanto en local como en la nube.

Por tanto, una vez dispongamos del modelo a empaquetar, existen diversas herramientas con las que crear la instancia del modelo. En este caso, de entre todas las posibles, veremos como usar **MLflow**. La elección de MLflow se ha debido a su fácil

<sup>1</sup>Una tubería (*pipeline*) consiste en una cadena de procesos conectados de forma que la salida de cada elemento de la cadena es la entrada del próximo, permitiendo comunicación y sincronización.

<sup>2</sup>**Scikit-learn** es una biblioteca para aprendizaje automático de *software* libre para el lenguaje de programación Python (<https://scikit-learn.org/stable/>).

<sup>3</sup>En este caso, se usará Github al disponer de una buena integración en local y en el *cloud*.

<sup>4</sup>Los contenedores ofrecen un modo estándar de empaquetar el código, las configuraciones y las dependencias de la aplicación en un único objeto. Y son más livianos que las máquinas virtuales.

integración en local y en el *cloud*. Puesto que si hubiéramos desplegado sólo en el cloud, podríamos haber hecho uso de Azure Functions<sup>5</sup>.

## MLflow

MLflow (<https://mlflow.org>) es una plataforma de código abierto que permite administrar el ciclo de vida completo de ML. Aunque sea una herramienta *open-source*, viene integrada en el área de trabajo de Azure Databricks, debido a que los creadores de MLflow y Azure Databricks son los mismos. Gracias a esta integración, es posible tanto en local como en el *cloud* llevar a cabo la administración de experimentos y ejecuciones y la captura de revisiones de *notebooks* [20], lo que permitirá hacer una comparación de dicha herramienta en ambos entornos, debido a que el funcionamiento final es el mismo pero la manera de ejecutar y poner el servicio de MLflow varía de un entorno a otro.

En MLflow, los experimentos son la unidad principal de organización; donde todas las ejecuciones de MLflow pertenecen a un experimento. Cada experimento permite visualizar, buscar y comparar ejecuciones, así como descargar *artifacts* o metadatos de ejecución para su análisis en otras herramientas (en el capítulo 3 veremos más en detalle su funcionamiento). A continuación, comprendamos las cuatro funciones principales que engloba MLflow [17, 22] (figura 2.9):

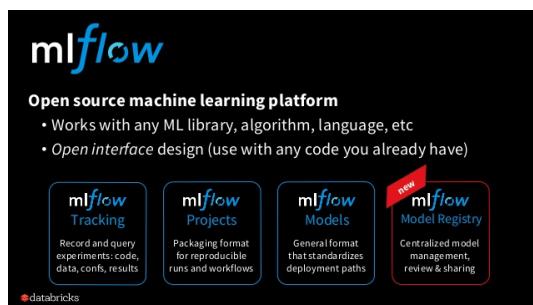


Figura 2.9: Elementos de MLflow [14]

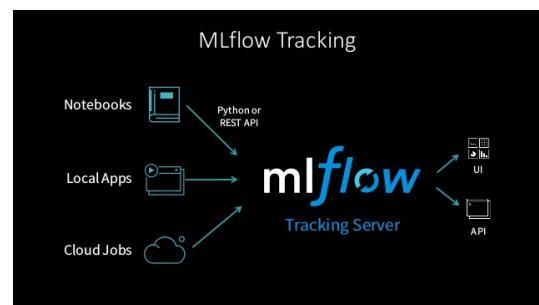


Figura 2.10: MLflow Tracking [14]

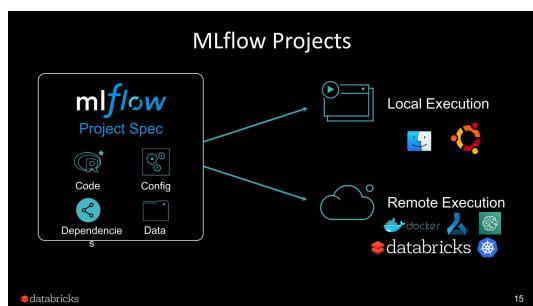


Figura 2.11: MLflow Projects [14]

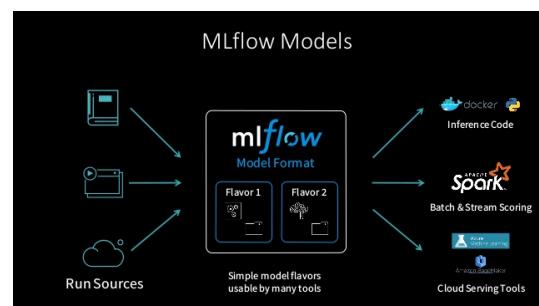


Figura 2.12: MLflow Models [14]

<sup>5</sup>Azure Functions permite ejecutar pequeños fragmentos de código (denominados "funciones") sin preocuparse por la infraestructura de la aplicación. Con Azure Functions, la infraestructura de la nube proporciona todos los servidores actualizados que necesita para mantener la aplicación en ejecución a gran escala. En Visual Studio Code, haciendo uso de la librería `scikit-learn`, es posible empaquetar el modelo y obtener un modelo en `.pkl` que puede ser desplegado luego en Azure Machine Learning.

1. **MLflow Tracking - Seguimiento:** este componente permite realizar un seguimiento de los experimentos para registrar y comparar parámetros y resultados (figura 2.10).
2. **MLflow Projects - Proyectos:** este componente permite usar el código ML de forma reutilizable y reproducible para compartir con otros científicos de datos o para transferir a producción (figura 2.11). Se puede trabajar con entornos exportables en conda, Docker o en nuestro propio entorno local.
3. **MLflow Models - Modelos:** este componente permite administrar e implementar modelos desde una gran variedad de bibliotecas de ML para distintas plataformas, es decir, permite empaquetar modelos de forma estandarizada, lo que facilita su posterior puesta en producción (figura 2.12). Con esta herramienta, es posible beneficiarse de diversas formas de empaquetar modelos para que puedan ser interpretados por los diferentes *frameworks* de producción. Además, admite subir los modelos a Azure ML o servirlos mediante una API REST.
4. **MLflow Model Registry - Registro de Modelos:** este componente ha sido el último en integrarse y se basa en centralizar en un almacén los modelos para administrar las transiciones entre las etapas del ciclo de vida completo de los modelos, desde el almacenamiento provisional a la producción, con funcionalidades que permitan crear versiones y anotaciones.

En tal caso, en esta fase se despliega o implementa el modelo. Por ejemplo, en el *cloud* se hace uso de Azure Container Instances (o Azure Kubernetes Service) que ofrecen una manera simple de ejecutar un contenedor en Azure sin la necesidad de aprovisionar máquinas virtuales o adoptar un servicio de nivel superior<sup>6</sup>, o mediante el uso de herramientas en local, como veremos en el capítulo siguiente.

#### 2.4.3. Fase 3: Automatizar flujos de trabajo, supervisar y administrar

En esta fase se automatiza el ciclo de vida del modelo completo ya que los modelos no son activos estáticos, por lo que se entrenan constantemente con nuevos datos y se mejoran con el tiempo [1]. Gracias al enfoque de MLOps, se proporciona un centro unificado para rastrear el origen y el rendimiento a medida que se desarrollan nuevos modelos y se implementan en producción (figura 2.13). En otras palabras, mediante el uso de contenedores, como Azure Container Instances (o Azure Kubernetes Service) nuestro enfoque cambia a conectar el *pipeline* operacionalizado (veremos como funciona en MLflow) con una robusta herramienta de *model serving* para administrar y (re)implementar los modelos a medida que maduran.

Por otro lado, las herramientas y técnicas utilizadas para monitorear el software tradicional no son adecuadas para el aprendizaje automático. MLOps proporciona sistemas de monitoreo especialmente diseñados, en donde las métricas específicas del modelo deben rastrearse para medir y comparar el rendimiento con otros modelos implementados [1], parte de esta funcionalidad viene integrada en MLflow. Es importante, mantener la “salud” de los modelos después de su despliegue.

<sup>6</sup>Para servir modelos en el *cloud*, se suele aprovechar Azure Container Instances, mientras que para las cargas de trabajo de producción se usaría Azure Kubernetes Service por sus capacidades de escalamiento robustas.

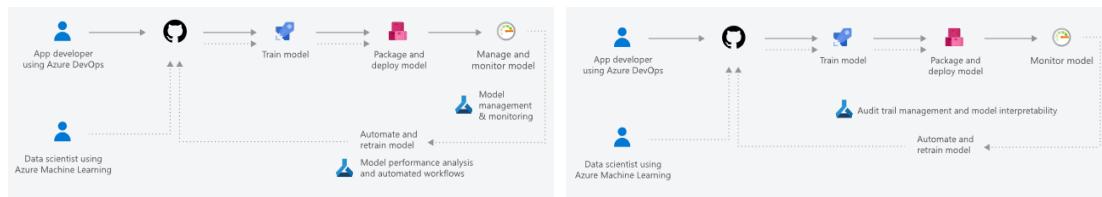


Figura 2.13: Fase 3 de MLOps [15]

Figura 2.14: Fase 4 de MLOps [15]

#### 2.4.4. Fase 4: Aplicar gobernanza y control

Finalmente, se monitoriza el modelo (figura 2.14), es decir, se capturan los datos necesarios para establecer un registro de auditoría completo del ciclo de vida de ML, que incluye quién publica los modelos, el motivo por el que se realizan los cambios y el momento en que los modelos se implementan o usan en producción [15]. Bajo el modelo de gobierno, los datos, los modelos y otros recursos deben controlarse estrictamente para evitar cambios no deseados. Gracias a MLOps se proporciona control de acceso centralizado, trazabilidad y registros de auditoría para minimizar el riesgo y garantizar el cumplimiento normativo [1].

### 2.5. Conclusiones

Este capítulo ha introducido el enfoque de MLOps, el cual se retroalimenta de las fases que constituyen el proceso tradicional de ML, como el análisis descriptivo o pre-procesamiento de los datos, modelado y evaluación. No obstante, para llevar a cabo un MLOps de éxito es necesario incorporar las mejoras prácticas llevadas a cabo por ingenieros de *software* en DevOps, con el fin de poner en producción el despliegue de dicho modelo. Esto implica que profesionales de muy diversa índole colaboren para poner una aplicación de IA en producción. MLOps surge de esta unión, como un conjunto de prácticas que proporcionan determinismo, escalabilidad, agilidad y gobernanza en el desarrollo del modelo y la implementación del *pipeline*. Como hemos visto, MLOps engloba 4 grandes fases, entre las que destaca la obtención del modelo entrenado, el empaquetado e implementación del modelo haciendo uso de MLflow para su posterior consumo y la monitorización del mismo. De manera que en el siguiente capítulo, veremos un caso de uso para desarrollar el enfoque de MLOps, tanto en local como en el *cloud*, haciendo uso de las distintas herramientas de los entornos que nos permiten la industrialización de los modelos de ML/IA.



# Capítulo 3

## Implementación de MLOps

**RESUMEN:** Este capítulo abarca el desarrollo del trabajo, es decir, la creación de la aplicación usando el enfoque de MLOps mediante la administración, implementación y supervisión de los modelos a partir de los datos extraídos de Kaggle (*Microsoft Malware Prediction*).

### 3.1. Introducción

Como hemos comentado, uno de nuestros objetivos es llevar a cabo la **implementación de MLOps**. De esta manera, se van a proponer alternativas de arquitectura (herramientas y configuración del sistema) para efectuar el enfoque actual de MLOps. En las figuras 3.8 y 3.14 se puede apreciar el esquema de arquitectura que se va a implantar, tanto en local como en la nube. Para poner en funcionamiento este proceso, es necesario partir de un problema. En este caso, se va a detectar si un sistema Windows se ha infectado de *malware* en función de las diferentes propiedades de dicho dispositivo<sup>1</sup>. Se contará con la presencia de unos datos tratados en su mayoría con Python, con el fin de obtener información útil que permita realizar buenas predicciones con los modelos de ML/IA seleccionados. Una vez que estemos satisfechos con el modelo seleccionado (entrenado y evaluado)<sup>2</sup>, deberemos hacer uso de las herramientas y arquitecturas que permiten la puesta en marcha del modelo en producción, creando una API REST del modelo al que realizar consultas o peticiones. Por esto, se creará una aplicación web que se conecte tanto a la API (local) como a Azure ML, y mediante un formulario permita predecir si el dispositivo Windows se infectará o no de alguna familia de *malware*. Partiendo de que la idea de MLOps reside en juntar ML y DevOps, nos centraremos inicialmente en el proceso tradicional de *Data Mining* que vimos en el capítulo 2, donde posteriormente lo uniremos a la idea de DevOps comentada.

**Nota:** se irá haciendo mención al repositorio de Github del proyecto ([enlace](#)).

### 3.2. Ciclo de vida del proyecto en la parte de ML

Como estudiamos en el capítulo 2, se va hacer uso de la **metodología CRISP-DM** para realizar la parte correspondiente al ciclo de vida de Machine Learning en

---

<sup>1</sup>Es posible extrapolar el uso o enfoque de MLOps con otro conjunto de datos.

<sup>2</sup>En general se da por terminado el ciclo de *Data Mining* cuando el modelo es entrenado y evaluado.

MLOps. Sin embargo, en este proyecto se le va a dar más importancia al modelo final, poniéndolo en producción y realizando una aplicación que permita realizar consultas al mismo. A continuación, veremos las acciones realizadas para cada una de las fases.

### 3.2.1. Fase I: *Business Understanding*

El proyecto se centra en resolver un problema particular. El dataset [Microsoft Malware Prediction](#), proporcionado por Kaggle, contiene datos de telemetría de propiedades e infecciones de máquinas con sistema Windows; dicha información nace de la combinación de informes y amenazas de la solución de protección de Microsoft, Windows Defender. El objetivo de este dataset es **predecir la probabilidad de que una máquina Windows se infecte por varias familias de malware en función de sus propiedades**<sup>3</sup>. Cada fila de este conjunto de datos corresponde a una máquina, identificada de forma exclusiva por la variable `MachineIdentifier`. Para saber si se detectó *malware* en la máquina, se hace uso de la variable binaria `HasDetections`, donde 0 (falso) indica que no se infectó y 1 (verdadero) que sí se infectó. Por lo que usando la información y las etiquetas del conjunto de aprendizaje, se debe predecir el valor de `HasDetections` en una máquina nueva. Es importante destacar que este conjunto de datos no es representativo de las máquinas de los clientes de Microsoft; ya que está muestreada para incluir una proporción mucho mayor de máquinas *malware* [11].

El problema que vamos a tratar entra dentro de la categoría de aprendizaje supervisado, donde cargamos los datos etiquetados al modelo para que “aprenda” a asignar la etiqueta de salida adecuada a un nuevo valor, puesto que conocemos los datos de entrada y los datos de la salida esperados [2]. No obstante, disponemos de dos datasets, uno etiquetado (conjunto de aprendizaje) al cual se le realizará una partición para entrenamiento y validación; y otro dataset no etiquetado (conjunto de prueba), que servirá para “probar” el comportamiento del modelo en el *submission* de Kaggle<sup>4</sup>.

Dado que nuestro proyecto tiene un enfoque empresarial y, en menor medida, académico, la selección de unos datos de gran dimensionalidad, permiten que podamos manejar y poner en funcionamiento el término Big Data en un caso real. También en esta fase, nos tenemos que plantear qué herramientas vamos a usar. En este caso, se ha optado por hacer las fases de ML en local o en el *cloud* con *notebooks* de Python (en el apéndice B se puede encontrar más información acerca de las herramientas usadas).

### 3.2.2. Fase II: *Data Understanding*

En esta fase se ha realizado un análisis descriptivo de los datos con el fin de conocer y entender el conjunto de datos y descubrir qué información interesante puede ser de utilidad para los modelos y sus predicciones. El análisis descriptivo se ha realizado en el *notebook* `1-MicrosoftMalwarePrediction-CargarVisualizar.ipynb`, disponible [aquí](#). A continuación, se mencionan algunas de las anotaciones iniciales de los datos (para saber más acerca del análisis descriptivo de los datos ir al apéndice C):

- El conjunto de datos etiquetado está balanceado (figura C.1).

---

<sup>3</sup>Para poder realizar la descarga de los datos en Kaggle es necesario aceptar las reglas.

<sup>4</sup>La realización del *submission* de Kaggle permite comprobar que las predicciones hechas a los datos no etiquetados son correctos o no, ya que Kaggle tiene la solución para dichas predicciones ([enlace](#)).

- La dimensionalidad de los datos de aprendizaje (etiquetados) es de (8921483, 83) y la de los datos de prueba (no etiquetados) de (7853253, 82).
- Al realizar la carga de los datos de aprendizaje, aparece un *warning* que sugiere que es necesario cambiar el tipo de dato a las variables con el fin de reducir su tamaño (espacio) en memoria.
- Hay muchas variables con pocos *missings* (menos de un 10 %) y pocas variables con más del 90 % de observaciones vacías (figura C.2).
- Alrededor de 30 variables tienen más del 80 % de las observaciones en la primera categoría o etiqueta (asimetría).

Por último, se realiza un análisis exhaustivo para cada variable. Dicho estudio se puede encontrar [aquí](#) y se basa en sacar el mayor conocimiento de los datos disponibles.

### 3.2.3. Fase III: *Data Preparation*

Después de comprender tanto el problema que queremos abordar y los datos que disponemos, es necesario realizar distintas operaciones y/o transformaciones a los datos “brutos” con el fin de construir el conjunto final de datos (los datos que se utilizarán en el modelado). Este tratamiento o preprocesamiento se ha realizado en el *notebook* 2-MicrosoftMalwarePrediction-Preprocesamiento.ipynb, disponible [aquí](#). De manera resumida, se han tratado y limpiado los datos haciendo uso de las siguientes técnicas (para saber más acerca del preprocesamiento de los datos ir al apéndice D):

- Eliminar variables que no aportan información.
- Estudiar reducción dimensionalidad mediante el uso del PCA.
- Tratar los *missings*, tanto por variables como por filas.
- Estudiar y tratar los *outliers*.
- Pasar a minúsculas los valores de las variables categóricas.
- Realizar transformaciones en ciertas variables, como transformar *missings* a ‘unknown’ en variables categóricas, fusionar la etiqueta ‘unspecified’ con ‘unknown’ o fusionar la etiqueta ‘portable’ con ‘notebook’ en *Census\_ChassisTypeName*.
- Convertir las variables categóricas a *Label Encoding*.
- Estudiar y tratar la correlación.

### 3.2.4. Fase IV: *Modeling*

Una vez que disponemos de los datos limpios, la siguiente etapa consiste en seleccionar y aplicar las técnicas de modelado. Gracias a los conocimientos adquiridos durante la realización del Máster, me he centrado en hacer uso de tres algoritmos que a priori sé que tienen un buen comportamiento [28]:

1. **Regresión Logística**
2. **Random Forest**
3. **Gradient Boosting**

## Regresión Logística

La técnica estadística de **Regresión Logística** permite describir dependencias significativas entre las variables incluidas en el dataset para investigar y modelizar la relación entre las variables (pincha [aquí](#) para saber más sobre el algoritmo de Regresión Logística). En base al conjunto de muestras de entrenamiento, el algoritmo es capaz de calcular una probabilidad de clasificación con respecto a las posibles categorías de la variable *target HasDetections*. Si dicha probabilidad sobrepasa el valor 0.5, se considera que la muestra pertenece a la clase 1 y, si su probabilidad es menor que 0.5, entonces se considera que se encuentra en la clase 0. Una vez predichas las etiquetas, y a partir de las probabilidades, se calcularán una serie de métricas (como veremos en la fase de evaluación) [28]. Para aplicar este modelo en Python nos ayudaremos de la librería `sklearn` y concretamente del módulo `linear_model` [26].

```

1 # Configuracion del algoritmo de Regresion Logistica
2 LogisticRegression(C=1.0, class_weight=None, max_iter=100,
3                     penalty='l2', tol=0.0001)

```

- `C`: aplica regularización con el objetivo de reducir el *overfitting*, cuanto más pequeño es el valor, mayor es la regularización.
- `class_weight`: puede ser *balanced/None* dependiendo de si se quiere balancear el conjunto de datos o no (usar cuando las clases son desbalanceadas).
- `penalty='l2'`: indica la regularización, en este caso L2 hace referencia a la regresión *Ridge*, que estima  $\beta$  pequeños y sirve para controlar el sobreajuste.
- `tol`: es un parámetro que se encuentra dentro de los *early stopping*. Sirve para que el algoritmo pare de iterar una vez alcanzado el criterio de tolerancia fijado.

## Random Forest

El método de **Random Forest** es una modificación del método *Bagging*<sup>5</sup> que utiliza una serie de árboles de decisión con el fin de mejorar la tasa de clasificación, es decir, combina la idea básica de dicho método de sortear muestras de entrenamiento con la idea de hacer una selección aleatoria de variables (pincha [aquí](#) para saber más sobre el algoritmo de Random Forest). En general, lo que busca esta técnica es descorrelacionar los diversos árboles que son generados por las diferentes muestras de los datos de entrenamiento, y luego simplemente reducir la varianza en los árboles promediándolos [28]. Como desventaja, se debe considerar que no suelen ser buenos clasificadores cuando existe un desequilibrio de datos y que a veces, pueden producir sobreaprendizaje. Para aplicar este algoritmo nos ayudaremos de la misma librería que antes pero esta vez haciendo uso del módulo `RandomForestClassifier` [27].

```

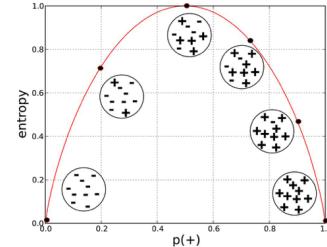
1 # Configuracion del algoritmo Random Forest
2 RandomForestClassifier(criterion='entropy', max_depth=12, n_jobs=-1,
3                         oob_score=True, n_estimators=100, max_features="auto",
4                         min_samples_leaf=50)

```

---

<sup>5</sup>El *Bagging* es una técnica que consiste en crear diferentes modelos usando muestras aleatorias con reemplazo y luego combinar o ensamblar los resultados.

- `criterion='entropy'`: se utiliza para construir los nodos, en este caso se hace uso de la **entropía**, medida que mide la calidad del nodo en base a la homogeneidad de la muestra. Si la muestra es completamente homogénea la entropía será 0 y si la muestra está igualmente dividida (50 % - 50 %) la entropía será máxima, es decir, 1. El objetivo de un nodo es clasificar lo mejor posible, esto es, que haga una distinción entre clases buscando la menor entropía posible y con la mayor ganancia de información.
- `max_depth=12`: máxima profundidad del árbol, mayores valores de profundidad suelen conducir a problemas de sobreajuste.
- `n_jobs=-1`: hace referencia al número de trabajos a ejecutar en paralelo, en este caso -1 significa que se usan todos los procesadores de la máquina.
- `oob_score=True`: indica que se va a utilizar la técnica de *Bagging* para la construcción del “bosque”.
- `n_estimators=100`: se usan 100 árboles para mejorar la precisión del modelo.
- `max_features='auto'`: número de parámetros utilizados para buscar el mejor corte (`auto` es igual a  $\sqrt{n\_features}$ ). Además, la elección de este parámetro ayuda a controlar el sobreajuste.
- `min_samples_leaf=50`: número mínimo de observaciones por hoja.



**Figura 3.1: Entropía**

## Gradient Boosting

Otra de las técnicas de ensamblaje a tener en cuenta es el **Gradient Boosting** (pincha [aquí](#) para saber más sobre el algoritmo de Gradient Boosting). Esta técnica de aprendizaje automático conduce a modelos de predicción que combinan conjuntos de modelos más débiles, normalmente árboles de decisión. En definitiva, se basan en ir ajustando modelos sucesivamente de manera que cada uno de ellos dé mayor peso a las observaciones peor clasificadas por los modelos previos [28]. Los árboles de decisión con Gradient Boosting tienen un buen comportamiento, no obstante, su principal inconveniente es que requieren un ajuste cuidadoso de los parámetros y mucho tiempo de entrenamiento. Esta técnica internamente representa todos los problemas como un caso de modelado predictivo de regresión que sólo toma valores numéricos como entrada, ya que el hecho de trabajar sólo con datos numéricos es lo que hace que esta librería sea tan eficiente (por eso el hecho transformar los datos usando la técnica de *Label Encoding*). Para aplicar este algoritmo nos ayudaremos de misma librería que antes pero esta vez mediante el módulo `GradientBoostingClassifier` [25].

```

1 # Configuración del algoritmo Gradient Boosting
2 GradientBoostingClassifier(n_estimators=50, learning_rate=0.75,
3                             max_depth=4, validation_fraction=0.2, random_state=9)

```

- **n\_estimators**: es el número de etapas *Boosting* a realizar. Al ser un algoritmo bastante robusto, un valor grande generalmente resulta en un mejor rendimiento.
- **learning\_rate=0.75**: es la tasa de aprendizaje, la cual incide en la reducción de contribución de cada árbol.
- **max\_depth=4**: igual que antes, indica la profundidad máxima de los estimadores. La profundidad máxima limita el número de nodos en el árbol. Este parámetro se debe ajustar adecuadamente para obtener el mejor rendimiento.
- **validation\_fraction**: este parámetro dice la proporción de datos de entrenamiento que se reservan como conjunto validación, comprendido entre 0 y 1.
- **random\_state**: determina la semilla a usar con el fin de reproducir el modelo.

### 3.2.5. Fase V: *Evaluation*

Tras haber comentado y comprendido los tres algoritmos a usar, es hora de evaluar los modelos antes de la puesta en producción. Entonces, cuando ya se dispone de un modelo (o modelos) con una buena calidad desde la perspectiva del análisis de datos, es importante evaluar el modelo a fondo para revisar que se han logrado adecuadamente los objetivos del problema planteado. Para evaluar, se hace uso del conjunto de datos de validación obtenido previamente. Además, se contrastará su comportamiento mediante los *submissions* de Kaggle (ver [código](#)). Por otro lado, en nuestro proyecto no nos vamos a quedar con un algoritmo por encima del otro, sino quearemos uso de los mejores parámetros de cada modelo para poner en producción los tres y hacer uso indistintamente de la información que aporta cada uno de ellos en nuestra aplicación final. Por tanto, después de haber ejecutado los *notebooks* que podemos encontrar [aquí](#), pasamos a evaluar mediante diversas métricas cada modelo (figuras 3.2, 3.3 y 3.4)

	<b>Log loss</b>	<b>Accuracy</b>	<b>F1 Score</b>	<b>Precision</b>	<b>Recall</b>	<b>AUC</b>	<b>Tiempo de entrenamiento</b>
<b>Random Forest</b>	12.598056	0.635253	0.644785	0.628454	0.661989	0.635249	2331.510788

Figura 3.2: Métricas del algoritmo Random Forest

	<b>Log loss</b>	<b>Accuracy</b>	<b>F1 Score</b>	<b>Precision</b>	<b>Recall</b>	<b>AUC</b>	<b>Tiempo de entrenamiento</b>
<b>Regresión Logística</b>	13.628411	0.605421	0.598193	0.609451	0.587343	0.605424	8016.766741

Figura 3.3: Métricas del algoritmo Regresión Logística

	<b>Log loss</b>	<b>Accuracy</b>	<b>F1 Score</b>	<b>Precision</b>	<b>Recall</b>	<b>AUC</b>	<b>Tiempo de entrenamiento</b>
<b>Gradient Boosting</b>	12.172721	0.647567	0.640917	0.653346	0.628952	0.64757	3280.820529

Figura 3.4: Métricas del algoritmo Gradient Boosting

Una vez estudiados todos los algoritmos que hemos tratado, es interesante hacer un análisis para seleccionar el mejor predictor. A continuación, veremos brevemente qué modelo es más preciso a la hora de realizar la clasificación de infección de *malware*. A simple vista, se observa que de los tres modelos el “mejor” es Gradient Boosting (figura 3.4). Para determinar esto se ha revisado que tanto la *precision*<sup>6</sup> (0.653), el área bajo la curva ROC (0.647) como el *accuracy*<sup>7</sup> (0.647) tienen los mejores valores. Por contraposición, si hacemos uso del *submission* de Kaggle, éste nos determina que el mejor modelado ha sido con Random Forest, puesto que es el que obtiene mayor *accuracy* (ver [aquí](#)), siendo aquel que ha obtenido el menor tiempo de entrenamiento en local. Por contraposición, la Regresión Logística queda lejos de los resultados esperados. Adicionalmente, como comparativa se sacan las variables más importantes de cada algoritmo. Se observa que dichas variables son similares en Random Forest (figura 3.5) y Gradient Boosting (figura 3.6), puesto que tienen conclusiones parecidas (métricas).

Valor	Variable	Valor	Variable
57 0.401149	SmartScreen	57 0.401244	SmartScreen
56 0.168516	AVProductsInstalled	56 0.169904	AVProductsInstalled
55 0.083330	EngineVersion	55 0.091885	EngineVersion
54 0.034807	AppVersion	54 0.054166	Census_TotalPhysicalRAMGB
53 0.029097	Census_TotalPhysicalRAMGB	53 0.037232	AppVersion
52 0.021547	Census_InternalPrimaryDiagonalDisplaySizeInInches	52 0.030377	Wdft_IsGamer
51 0.019983	Census_PrimaryDiskTotalCapacityGB	51 0.026119	Census_InternalPrimaryDiagonalDisplaySizeInInches
50 0.018912	Wdft_IsGamer	50 0.016714	Wdft_RegionIdentifier
49 0.013920	Census_OSInstallTypeName	49 0.014922	Census_OSBUILDRevision
48 0.012147	IsProtected	48 0.013348	Census_OSInstallTypeName

Figura 3.5: Variables más importantes para Random Forest

Figura 3.6: Variables más importantes para Gradient Boosting

Valor	Variable
57 0.086621	OsSuite
56 0.085686	IsProtected
55 0.069224	Census_ProcessorCoreCount
54 0.056616	ProductName
53 0.042874	Census_IsSecureBootEnabled
52 0.036846	Census_GenuineStateName
51 0.035804	Census_OSEdition
50 0.031611	EngineVersion
49 0.029470	Census_HasOpticalDiskDrive
48 0.022990	Census_ProcessorManufacturerIdentifier

Figura 3.7: Variables más importantes para Regresión Logística

<sup>6</sup>Proporción de identificaciones positivas correctas.

<sup>7</sup>Relación entre el número de predicciones correctas y el número total de muestras de entrada.

### 3.2.6. Fase VI: Deployment

Si hacemos mención al estado del arte, la literatura en algunos casos ha concluido un proyecto de *Data Mining* después de la creación y evaluación del modelo. No obstante, como se ha visto en la metodología CRISP-DM existe un último proceso conocido como *deployment* (despliegue) o puesta en producción del modelo final, donde el propósito es aumentar el conocimiento de los datos al disponer del modelo en producción. Esta es la fase que permite unir, en su mayoría, ML con DevOps, obteniendo así lo que hasta ahora hemos visto como MLOps. Esta fase la veremos en el siguiente apartado.

## 3.3. Puesta en producción del modelo

A lo largo del proyecto se ha hecho mención a la puesta en producción tanto en local como en el *cloud* con Microsoft Azure. A continuación, veremos ese proceso en ambos entornos, en donde para ambas formas de trabajar se construirán servicios (modelos) a consumir mediante una aplicación web. Adicionalmente, si hacemos mención a las fases que componen MLOps (como vimos en el capítulo 2), la primera fase donde se entrena el modelo se obtiene de lo visto anteriormente en la fase de evaluación. Entonces, después de haber seleccionado el modelo (en este caso usaremos el óptimo de cada modelo), se empaqueterá el mismo en una instancia para llevar a cabo su implementación.

### 3.3.1. Caso de uso en local

El primer caso de uso que analizaremos será la **puesta en producción en local**. En la figura 3.8 se ve la arquitectura a implementar. A través de los *notebooks* de Jupyter se crea la instancia del modelo en MLflow. A su vez, MLflow contiene los modelos a desplegar, los cuales una vez desplegados se consumen haciendo uso de la aplicación web. En otras palabras podemos decir que tenemos una arquitectura de microservicios, en donde para conectarse a cada microservicio en local se usa el puerto correspondiente.

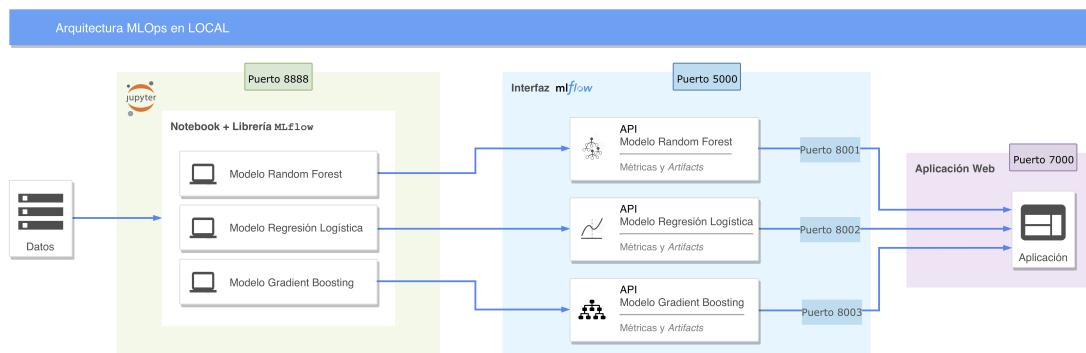


Figura 3.8: Esquema de la arquitectura de MLOps en local

Siguiendo las fases de MLOps, lo primero que hay que hacer es **crear la instancia del modelo con MLflow** en los *notebooks*. Para ello, es necesario seguir una estructura de código como la siguiente (pincha [aquí](#) para saber más):

```
1 with mlflow.start_run(): # 0. Iniciamos 'mlflow'
```

```

2 # 1. Creamos (o cargamos) el modelo en cuestión (RL, RF, GB)
3
4 # 2. Obtenemos las predicciones para el modelo
5 predict = model.predict(X_val)
6
7 # 3. Calculamos las medidas (metricas) que queramos
8 (rmse, mae, logloss, accuracy, F1, precision, recall, auc) = metricas(y_val, predict)
9
10 # 4. Cargamos los parametros de las metricas en MLflow
11 mlflow.log_metric("accuracy", accuracy)
12 mlflow.log_metric("precision", precision)
13 mlflow.log_metric("auc", auc)
14
15 # 5. Cargamos el modelo generado
16 mlflow.sklearn.log_model(model, "Model")
17

```

Automáticamente, después de implementar y ejecutar el anterior código (ver en este [notebook](#)), se crea una instancia del modelo en MLflow (figura 3.9). El anterior fragmento de código crea tantas instancias como ejecuciones se hayan realizado, tanto si terminan con éxito como si ha ocurrido un error o excepción durante la ejecución. Básicamente, hasta este punto sólo hemos “subido” los modelos a un lugar donde poder compararlos y registrarlos. Además, para acceder a la interfaz de MLflow (figura 3.9), debemos disponer de la librería instalada (dependencias) y ejecutar en un terminal la sentencia `mlflow ui` a la altura de donde se ha creado la instancia del modelo o donde se encuentra el *notebook* que la ha creado. Con esto podremos acceder a dicho servicio en el puerto 5000 (<http://127.0.0.1:5000>).

The screenshot shows the MLflow UI homepage. At the top, there's a dark header with the MLflow logo, GitHub, and Docs links. Below the header, the main interface has a sidebar on the left with tabs for Experiments, TFM, and Notes. The TFM tab is selected, showing Experiment ID: 0 and Artifact Location: file:///Users/gema/Desktop/TFM/codigo/mlruns/0. Under the TFM tab, there are three sections: Modelos Random Forest, Modelos Regresión Logística, and Modelos Gradient Boosting, each listing specific runs with their creation dates. Below these sections is a search bar for runs, filters for State (Active), and buttons for Search and Clear. At the bottom, there's a table titled 'Showing 11 matching runs' with columns for Start Time, User, Source, Metrics (F1, accuracy, auc), and a checkbox column. The table lists 11 runs from May 22, 2020, with various performance metrics and source details.

	Start Time	User	Source	F1	accuracy	auc
<input type="checkbox"/>	2020-05-22 20:27:44	gema	ipykernel_launcher.py	-	-	-
<input type="checkbox"/>	2020-05-22 19:34:41	gema	ipykernel_launcher.py	0.598	0.605	0.605
<input type="checkbox"/>	2020-05-22 19:33:49	gema	ipykernel_launcher.py	0.573	0.601	0.601
<input type="checkbox"/>	2020-05-22 19:30:49	gema	ipykernel_launcher.py	0.641	0.648	0.648
<input type="checkbox"/>	2020-05-22 19:28:23	gema	ipykernel_launcher.py	0.598	0.605	0.605
<input type="checkbox"/>	2020-05-22 19:27:57	gema	ipykernel_launcher.py	0.642	0.624	0.624

Figura 3.9: Página de inicio de MLflow

A continuación, vamos a ver como funciona MLflow. Podemos ver cada una de las ejecuciones dentro de la interfaz. En las pestañas *Runs* se tienen los detalles sobre cada una de las ejecuciones realizadas. En la figura 3.9 las instancias de los modelos que se han creado correctamente están con un *tick* verde y aquellas que durante su creación han tenido problemas están en rojo. Al hacer *click* en cada entrada, se devuelven los detalles de dicha ejecución, los archivos que definen el modelo y mucho más (figura 3.12). También es posible comparar varios modelos (figuras 3.10 y 3.11).

Search Runs: metrics.rmse < 1 and params.model = "tree" and tags.mlflow.source.type = "LOCAL"

Showing 11 matching runs

	Start Time	User	Source	F1	accuracy	auc
<input type="checkbox"/>	2020-05-22 20:27:44	gema	ipykernel_launcher.py	-	-	-
<input type="checkbox"/>	2020-05-22 19:34:41	gema	ipykernel_launcher.py	0.598	0.605	0.605
<input checked="" type="checkbox"/>	2020-05-22 19:33:49	gema	ipykernel_launcher.py	0.573	0.601	0.601
<input checked="" type="checkbox"/>	2020-05-22 19:30:49	gema	ipykernel_launcher.py	0.641	0.648	0.648

Figura 3.10: Página seleccionando las ejecuciones para hacer la comparación

Run ID	Run Name	Start Time	End Time	Metrics
aac82aa84f6d4b82adaf19fc2975050	326ab82080934288afe93177737d3844	2020-05-22 19:33:49	2020-05-22 19:30:49	72fd515589624214b2c41bcfa44aef17
				Metrics
				>
				F1
				accuracy
				auc
				logloss
				mae
				precision
				recall
				rmse

Figura 3.11: Página con la comparativa de las tres ejecuciones seleccionadas

Run 72fd515589624214b2c41bcfa44aef17

Date: 2020-05-21 18:26:09 Duration: 45.0s

Source: ipykernel\_launcher.py User: gema Status: FINISHED

Notes:

```
Modelo 01 de Random Forest
RandomForestClassifier(criterion = 'entropy', max_depth = 12, n_estimators = 100, max_features = "auto", min_samples_leaf = 50)
```

Parameters:

Name	Value
F1	0.645
accuracy	0.635
auc	0.635
logloss	12.6
mae	0.365
precision	0.628
recall	0.662
rmse	0.604

Metrics:

Name	Value
F1	0.645
accuracy	0.635
auc	0.635
logloss	12.6
mae	0.365
precision	0.628
recall	0.662
rmse	0.604

Figura 3.12: Página con información detallada de la ejecución de un modelo

A pesar de toda la información que se puede encontrar en cada experimento, lo más importante o necesario para desplegar la instancia del modelo y crear así el contenedor<sup>8</sup> es hacer uso del fichero YAML<sup>9</sup>, el cual se crea a la par que se construye la instancia del modelo (al ejecutar el fragmento de código que hemos visto). Además, en el experimento podemos ver su contenido (figura 3.13). Básicamente, este fichero nos implementa el contenedor que contiene nuestro modelo con las dependencias necesarias. El hecho de haber usado la librería `scikit-learn` para los modelos, se debe a que MLflow integra por defecto esta biblioteca, con lo que no tenemos que cambiar ni añadir ninguna dependencia. Sin embargo, si hubiéramos hecho uso del algoritmo XGBoost (ver [aquí](#)), el cual está en la librería `xgboost`, hubiera sido necesario añadir la dependencia de esa librería para incluirla en el fichero YAML y tener en cuenta que la función de ese modelo está ubicada en otra biblioteca.



**Figura 3.13:** Página de información con el fichero YAML del modelo

En tal caso, se despliega el modelo creando una instancia para cada modelo con Docker<sup>10</sup> donde podremos acceder a la API de dicho modelo. Para acceder a la API del modelo y crear el servicio web, debemos lanzar en un terminal la siguiente sentencia, en donde el `host` hace referencia al servidor donde nos vamos a conectar, en este caso usaremos `0.0.0.0` para decirle que use cualquier dirección y el `<puerto>` será `8001` por defecto (cada modelo tendrá un puerto distinto `800X`). Por ejemplo, si `<host>` es `0.0.0.0` y el `<puerto>` es `8001`, se consume del modelo a partir de la API `http://0.0.0.0:8001/invocations`. En la siguiente sección hay un ejemplo de ello mediante la API generada en el *cloud* (Azure ML). No obstante, veremos más adelante como se consume en la aplicación web creada.

```

1 # Poner en producción un modelo (http://0.0.0.0:8001/invocations)
2 mlflow models serve -m <ruta-del-modelo-en-local> -h <host> -p <puerto>

```

En el siguiente [vídeo](#) hay una demo con lo que se acaba de comentar.

<sup>8</sup>Un contenedor es como una máquina virtual más ligera. La idea de crear el contenedor es la misma a si tuviéramos el modelo desplegado en una máquina virtual que está siempre iniciada y nosotros podemos acceder a ella y realizar todas las predicciones al modelo que queramos.

<sup>9</sup>YAML es el lenguaje en el que se definen los *deployments* y demás estructuras.

<sup>10</sup>La idea detrás de Docker es crear contenedores ligeros y portables para las aplicaciones *software* que puedan ejecutarse en cualquier máquina con Docker instalado, independientemente del sistema operativo que la máquina tenga por debajo, facilitando así también los despliegues.

### 3.3.2. Caso de uso en el cloud

El segundo caso de uso a analizar será la **puesta en producción en el *cloud*** en **Microsoft Azure**, en concreto, mediante los *notebooks* de Azure Databricks pero la puesta en producción de los modelos será de igual manera que antes (pincha [aquí](#) para saber más). En este caso, se seguirá la estructura de la figura 3.14. La idea es la misma que antes, solo que ahora MLflow viene integrado en Databricks (puesto que son de los mismos creadores), lo que facilita crear el servicio, pero el funcionamiento sigue siendo el mismo, lo único que ahora viene todo integrado en el ecosistema de Azure.

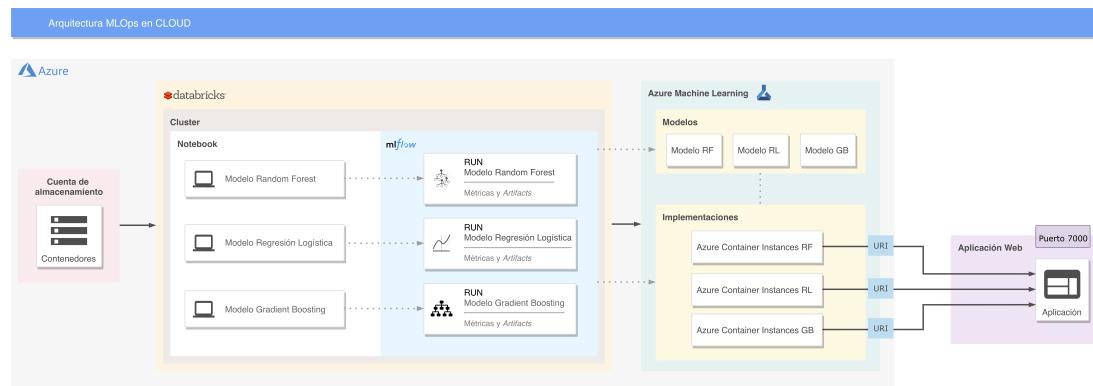


Figura 3.14: Esquema de la arquitectura de MLOps en el cloud

Por consiguiente, para crear los modelos en los *notebooks* de Databricks se accede a los datos “limpios” depositados en un Azure Storage (figura 3.15), para posteriormente crear la instancia del modelo en cuestión (tal y como vimos antes). En el *cloud* podemos ver cada una de las ejecuciones dentro de la GUI de Databricks gracias a la integración de MLflow (figura 3.16). Los experimentos se encuentran a la derecha del panel en la interfaz de Databricks y al hacer *click* en cada entrada se devuelven los detalles de la ejecución. Además, al *clickar* en la pestaña *Runs*, en la esquina superior derecha, observamos que podemos ver detalles sobre cada una de las ejecuciones realizadas con MLflow (figura 3.16). En términos generales, tenemos la misma herramienta aunque el acceso a ella es distinto pero más fácil e intuitivo.

Nombre	Última modificación	Nivel de acceso público	Estado de concesión
datos	21/5/2020 20:23:32	Contenedor	Disponible
modelos	21/5/2020 20:59:07	Contenedor	Disponible

Figura 3.15: Crear la instancia del modelo en MLflow

Al ejecutar el código que aparece en la figura 3.16, que es el mismo que vimos en local, se crea la instancia del modelo en cuestión, pero en este caso se deben de cargar los modelos, es decir, ubicar que dichos modelos se guarden en Azure Machine Learning Workspace. Básicamente, nosotros en local teníamos todo en nuestro ordenador pero en Azure el lugar donde se gestionan los modelos es en Azure Machine Learning. No

obstante, antes de que los modelos se puedan implementar en Azure ML, se debe crear un Azure ML Workspace. La función `azureml.core.Workspace.create()` se encarga de ello y carga un espacio de trabajo con un nombre especificado o crea uno si aún no existe. En nuestro caso, haremos uso de un recurso ya creado, y buscaremos los valores de la configuración necesarios (nombre del *workspace*, localización del *workspace*, nombre del *resource group* y el ID de la suscripción).

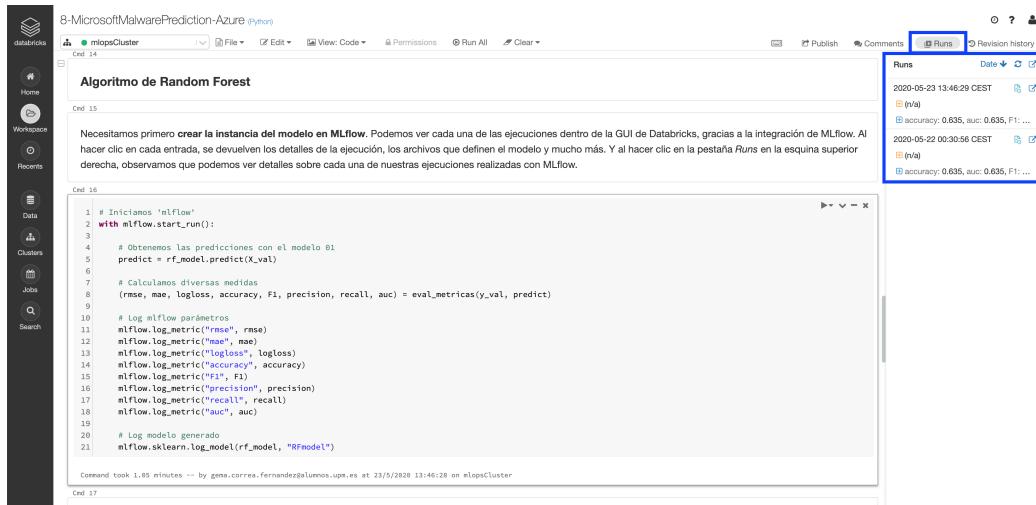


Figura 3.16: Integración de MLflow en Azure Databricks

```

1 # Definir los valores de la configuracion
2 workspace_name = "ML-TFM"
3 workspace_location = "southcentralus"
4 resource_group = "TFM_GroupResource"
5 subscription_id = "XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX"
6
7 # Crear un Azure Machine Learning Workspace
8 workspace = Workspace.create(name=workspace_name,
9                               subscription_id=subscription_id,
10                             resource_group=resource_group,
11                             location=workspace_location,
12                             exist_ok=True)

```

Cuando se ejecuta la función que acabamos de ver, es necesario autenticarse a través de una URL, y meter el código de autenticación que nos devuelve. Una vez que el modelo ya está almacenado, pasamos a desplegar la imagen del contenedor Docker del mismo. Para crear el contenedor, se vuelve hacer uso del fichero YAML y con la siguiente ejecución registramos el modelo en Azure ML Workspace, necesario para posteriormente desplegarlo. En la figura 3.17 se observa como en la parte derecha hay una pestaña que se llama “modelos”, la cual contiene el nuestro.

```

1 # Registrar el modelo en Azure ML Workspace
2 model_image, azure_model = mlflow.azureml.build_image(
3     model_uri=model_path,
4     workspace=workspace,
5     model_name="model",
6     image_name="model-container-image",
7     description="mlflow model machine malware infect",
8     synchronous=False)
9 model_image.wait_for_creation(show_output=True)

```

The screenshot shows the Microsoft Azure portal interface for the ML-TFM workspace. The left sidebar has 'Modelos' selected under 'Activos'. The main content area displays the details of a registered model named 'model-rf-malware'. The 'Detalles' tab is active, showing attributes like Version (2), Id. (model-rf-malware:2), Date registered (23/05/2020 11:56:41 UTC), Location (aml://asset/6d7d652495b44566841fc7b28a26194c), Description (mlflow rf model for scoring machine malware infection status), and Tags (model\_uri : dbfs:/databricks/mlflow/4431065003331548/b654e005f60fd4040bbd62a0e9fc0e50f/artifacts/RFmodel, python\_version : 3.7.3). A message at the top indicates that the page will move to a new immersive experience for managing the end-to-end machine learning lifecycle.

Figura 3.17: Modelo registrado en Azure ML Workspace

Una vez tenemos nuestro modelo, pasamos a la implementación del mismo. Para ello, optamos por crear un ACI (Azure Container Instances) y realizar el despliegue del servicio web usando la imagen del contenedor del modelo. En este caso, se genera en la pestaña “implementaciones” una URL con la cual consumir el modelo. En local se consumía el modelo a partir de la API <http://0.0.0.0:8001/invocations>, en el *cloud* haremos uso de <http://44c1017e-78f3-4de2-825e-5baec39e2e7c.southcentralus.azurecontainer.io/score>.

```

1 # Crear el servicio web del modelo
2 aci_webservice_name = "model-rf-malware-aci"
3 aci_webservice_deployment_config = AciWebservice.deploy_configuration()
4 aci_webservice = Webservice.deploy_from_image(workspace=workspace,
5                                              name=aci_webservice_name, image=model_image,
6                                              deployment_config=aci_webservice_deployment_config)
7 aci_webservice.wait_for_deployment(show_output=True)

```

The screenshot shows the Microsoft Azure portal interface for the ML-TFM workspace. The left sidebar has 'Implementaciones' selected under 'Activos'. The main content area displays the implementation details for the 'model-rf-malware' model. The 'Detalles' tab is active, showing attributes like Condition (Healthy), Type of process (ACI), Identifier of the service (model-rf-malware-aci), and Tags (model-rf-malware-container-image:2). The 'URI de puntuación' field contains the URL <http://44c1017e-78f3-4de2-825e-5baec39e2e7c.southcentralus.azurecontainer.io/score>. A message at the top indicates that compute targets will be manageable from both locations.

Figura 3.18: Implementación del modelo

Por último, consultaremos el *endpoint* del servicio web ACI enviando una solicitud HTTP POST que contenga el vector de entrada en formato JSON y devuelva las predicciones a partir de la URI (funciona de igual manera para la API creada en local). Para comprobar que funciona se ha realizado un ejemplo con Postman<sup>11</sup>, pero indistintamente podría ser usando la aplicación web creada (como veremos a continuación).

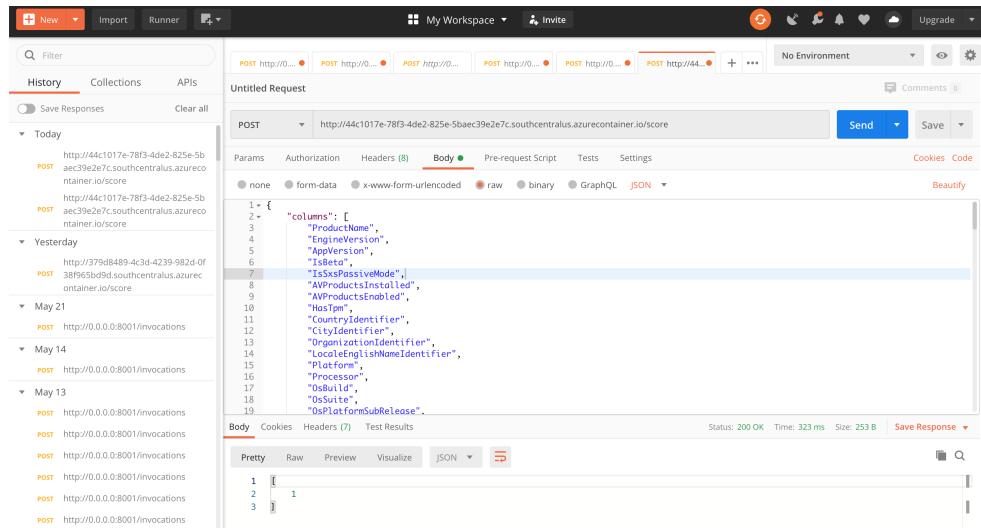


Figura 3.19: Consulta al modelo a través de Postman

En el siguiente [vídeo](#) se muestra del funcionamiento de MLflow en Azure y funcionamiento final de MLOps con la herramienta Postman.

Como punto final, estaría interesante estudiar este mismo procedimiento que hemos visto tanto de manera local como en el *cloud*, haciendo uso de la herramienta AutoML de Azure, quién nos realizaría todo el preprocesamiento y modelado, reduciendo de manera significativa el desarrollo del proyecto. No obstante, dado la dimensionalidad de los datos, AutoML no ha permitido trabajar con ella (ver apéndice E).

### 3.4. Funcionamiento de la aplicación web

El **funcionamiento de la aplicación web es el mismo independiente de que la puesta en producción sea en local o en el *cloud***, simplemente se necesita acceso a la API o URI del modelo para poder realizar las diversas peticiones o consultas. El código de la aplicación se encuentra [aquí](#) y ha sido realizado mediante el lenguaje de programación Python y el *framework* Flask, para las páginas web se ha usado HTML con Bootstrap (ir al apéndice B para saber más acerca de las herramientas usadas).

Entonces, hemos visto que ya disponemos de los modelos desplegados tanto en local como en el *cloud*. En local accederemos a la API mediante la URL `http://0.0.0.0:8001/invocations` y en el *cloud* accederemos al modelo subido en Azure ML mediante la URI `http://44c1017e-78f3-4de2-825e-5baec39e2e7c.southcentralus`.

<sup>11</sup>Postman es un entorno de desarrollo de APIs que nos permite diseñar, probar y monitorizar servicios REST.

[azurecontainer.io/score](http://azurecontainer.io/score). Seguidamente, el siguiente paso a realizar es establecer la conexión con el modelo desplegado. Para ello se ha creado una función que a partir de la información recibida en un formulario, crea una consulta (petición POST) a dicho modelo (es la misma idea que hemos visto en la figura 3.19).

```

1 # codigo app.py
2
3 @app.route('/probabilidad_random_forest', methods=['POST'])
4 def probabilidad_random_forest():
5
6     """
7         Funcion que nos redirige a visualizar la probabilidad del modelo
8     """
9
10
11    # Definimos la API del modelo
12    url = 'http://localhost:8001/invocations'
13
14    # Dado que los datos van en JSON es necesario definir la cabecera
15    headers = {"content-type": "application/json; format=pandas-split"}
16
17    # Realizamos la peticion POST
18    r = requests.post(url, data=data, headers=headers)
19
20    # Una vez hecha la peticion, devolvemos la probabilidad en una nueva pagina
21    return render_template("probabilidad_random_forest.html", malware = r.text)

```

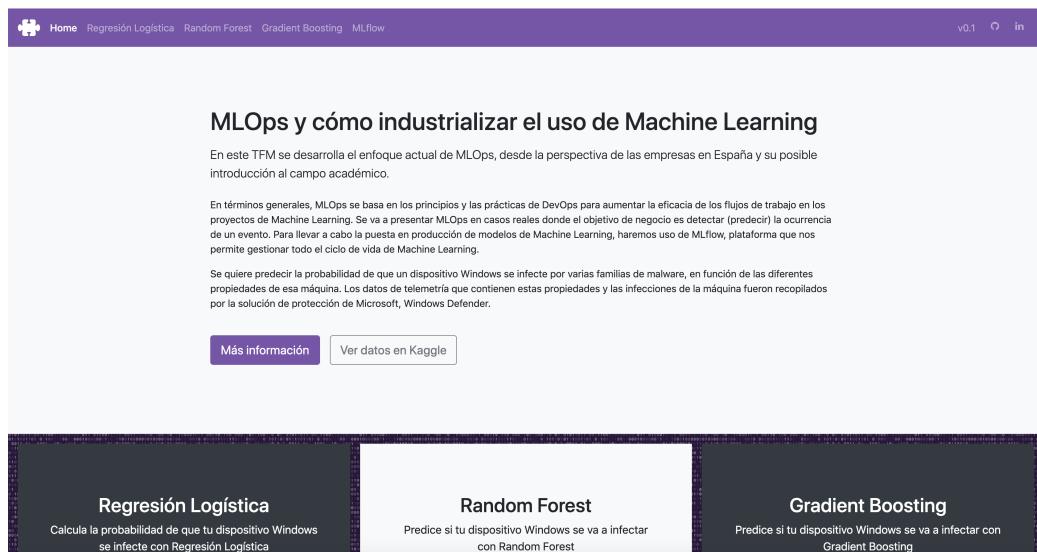


Figura 3.20: Página de inicio de la aplicación

A continuación, veamos de manera gráfica como funciona la app. Para ello, se ha creado una página de inicio en donde podemos encontrar toda la información relacionada con el proyecto (figura 3.20). En la página inicial de la aplicación nos podemos mover por los distintos modelos o algoritmos. Por defecto, cada modelo presenta un formulario<sup>12</sup> a llenar como el de la figura 3.22. Adicionalmente, en la barra de menú

<sup>12</sup>En los iconos de información de cada campo del formulario se puede ver una pequeña descripción de qué significa la variable y sus posibles valores, puesto que estamos haciendo uso de *Label Encoding*.

podemos pasar por cada unos de los modelos que tenemos y así predecir cuál es la predicción de que un sistema Windows se infecte de *malware*. Por ejemplo, en las figuras 3.22, 3.23 y 3.24 se observa como para unos mismos datos la probabilidad de infección es distinta. No obstante, esa distinción permite ser contrastada con la la Regresión Logística, ya que predice que existe un 0.49 de probabilidad de infectarse la máquina, lo que amplía el entendimiento de la información extraída de los otros dos modelos.

**Figura 3.21:** Página para realizar la petición POST a la API del modelo de Regresión Logística (rellenar formulario)

**Figura 3.22:** Página que devuelve la probabilidad de que un sistema Windows se infecte haciendo uso del modelo de Regresión Logística

**Figura 3.23:** Página que devuelve la probabilidad de que un sistema Windows se infecte haciendo uso del modelo de Random Forest (para ello se ha completado el formulario que realiza la petición POST a la API del modelo)



## Capítulo 4

# Conclusiones y Trabajo Futuro

**RESUMEN:** Este capítulo recoge los principales aspectos y contribuciones obtenidos para el ejemplo práctico realizado con el enfoque de MLOps. De igual manera, comprobaremos el cumplimiento de los objetivos que introducimos inicialmente en el capítulo 1 y veremos una comparación entre la implementación de MLOps en un entorno local contra la del *cloud*. Asimismo se comentarán posibles propuestas de mejora de cara a sucesivas iteraciones del proyecto en el futuro.

### 4.1. Conclusiones

En este TFM se ha profundizado en la puesta en producción de varios modelos en diversos entornos, tanto en local como en Azure, llevados a cabo mediante el uso de *notebooks*, el lenguaje Python y la herramienta MLflow, esencial para desplegar el modelo y unir las disciplinas que conforman el enfoque de MLOps. Para ello, se han introducido los conceptos básicos necesarios sobre Machine Learning y DevOps, así como las prácticas que permiten integrar ambas tecnologías.

Dentro de las posibilidades de análisis que ofrece MLOps, se ha prestado especial atención a la detección (predicción) de si un sistema Windows se ha infectado de *malware* a partir de la información recopilada por la solución de protección de Microsoft, Windows Defender. Este proceso de análisis, preprocesamiento, modelado y evaluación se ha llevado a cabo mediante *notebooks* de Python y diversos *frameworks* que nos han facilitado el trabajo, como podemos encontrar en el apéndice B. Gracias a este conjunto de datos se ha podido implementar y explicar el enfoque actual de MLOps.

Se han usado distintos *frameworks* de Python existentes para los dos entornos. Esto ha supuesto un trabajo de evaluación y selección de herramientas para determinar las que más se adaptaban a nuestro proyecto. De igual manera, se ha llevado a cabo un análisis sobre la elección de usar *notebooks* de Jupyter y no los de Google Colab, por ejemplo, puesto que no sólo es posible implementar el enfoque de MLOps en *notebooks* de Jupyter. Dado que el rendimiento de la carga de datos, considerados de gran dimensionalidad, disminuía al usar el entorno de Google. Básicamente, esta ha sido una de las principales diferencias que han hecho decantarse por *notebooks* en Jupyter.

A la vista de los resultados obtenidos, considero que desplegar un modelo en un entorno *cloud* es mejor opción si se requiere de una puesta de producción rápida, robusta y con escalabilidad, puesto que la infraestructura donde se está implementando es ajena a nuestro entorno local. No obstante, desplegar en la nube conlleva un coste económico, aunque se dispongan de créditos gratuitos<sup>1</sup>. Por lo que si se tienen recursos ilimitados no supone ningún problema hacer uso de ello, pero si existe un límite dificulta realizar tantas pruebas como se hayan podido realizar en un entorno local. Por otro lado, desplegar un modelo en local es más pedagógico, puesto que se comprende mejor lo que hay por debajo al no disponer de las herramientas integradas, cosa que pasa en el ecosistema Azure, ya que en local se debe controlar más dicho entorno y lanzar de forma manual los servicios que componen la arquitectura. Adicionalmente, Python no sólo permite implementar el proceso de *Data Mining*, sino que al ser un lenguaje multiparadigma permite crear aplicaciones web. En nuestro caso, se ha creado una aplicación web como un recurso extra para poder consumir y controlar la puesta en producción de los modelos. Además, dado su enfoque empresarial, consumir datos con una interfaz enriquece en contraposición a usar otra aplicación como podría ser Postman. Por añadidura, Github ha sido el centro de control del código para ambos entornos, siendo muy fácil de usar independientemente de donde trabajemos.

Por tanto, mediante el desarrollo de este TFM, considero que se han alcanzado los objetivos señalados en el capítulo 1, en concreto:

1. Se ha presentado una propuesta de industrialización de modelos de Machine Learning conocida como MLOps, la cual ha permitido hacer uso de la información proporcionada por los modelos desplegados para extraer conocimiento.
2. Se ha entendido cómo se utiliza la ciencia de datos en un entorno empresarial y académico, en donde en ambos está presentes el uso de *notebooks*. Además, una de las principales diferencias es que hasta ahora en el ámbito académico no solíamos pasar de la evaluación del modelo, mientras que en el empresarial dicho modelo sí pasa a ser desplegado o puesto en producción.
3. Se ha mostrado como se puede adoptar la ciencia de datos con tecnologías *cloud* mediante el uso de Microsoft Azure.
4. Se han desarrollado modelos predictivos de Machine Learning para predecir si un sistema Windows se ha infectado de *malware*. En concreto, se ha hecho uso de los modelos de Regresión Logística, Gradient Boosting y Random Forest, puesto que se conocía a priori su buen comportamiento.
5. Se han conocido las capacidades que ofrecen diversas herramientas para la implementación de modelos ML/IA, entre las que destaca **scikit-learn**.
6. Se han mostrado y comparado las dos arquitecturas implementadas para el despliegue del servicio web usando la imagen del contenedor del modelo. En donde, dependiendo de los requisitos y las necesidades es interesante hacer uso del entorno local o del entorno *cloud*.

---

<sup>1</sup>Microsoft Azure tiene una suscripción gratuita de 100\$ para los estudiantes.

7. Se ha prestado especial atención en desarrollar un trabajo autocontenido, tratando de que sea accesible a cualquier persona con un mínimo de conocimientos en esta área, y documentando un repositorio con todo el código disponible.
8. En cuanto a lo personal, he adquirido experiencia en el trabajo sobre MLOps, en concreto sobre qué une Machine Learning con DevOps, puesto que conocía ambas áreas debido a mi formación y gracias al trabajo he podido ver que ambas se necesitan para mejorar proyectos de MLOps. Además, he ganado soltura en diversas herramientas, al igual que he trabajado con datos de gran dimensionalidad, lo que ha permitido ver como se llevan a cabo los análisis en Big Data.

## 4.2. Trabajo futuro

Durante la realización de este TFM se han encontrado temas, ejemplos o aplicaciones bastante relacionadas pero que no se han estudiado en el mismo. Por este motivo, comentaremos algunos aspectos a profundizar para desarrollar el futuro:

1. Realizar la puesta en producción en la nube haciendo uso de MLflow y la plataforma Google Cloud, para realizar una comparativa entre plataformas del *cloud*.
2. Añadir más modelos de ML para tener más información en las predicciones.
3. Hacer uso de modelos procedentes de las herramientas conocidas como AutoML, con el objetivo de comparar cuan de buenas son frente al análisis llevado a cabo por un *Data Scientist* o *Data Engineer*.
4. Dado el enfoque académico, la aplicación web ha sido desarrollada con el *framework* Flask, debido a su facilidad en el aprendizaje. No obstante, se quiere implementar dicha aplicación haciendo uso de Django, *framework* más conocido y usado en el escenario DevOps.
5. Ampliar la aplicación web creada, para que mande alertas cuando el modelo se quede obsoleto o dotarla de más funcionalidad.

Adicionalmente, considero que este tema y el enfoque que le he dado puede tener interés para el público en general por lo que valoro la posibilidad de darle más difusión al trabajo, tanto en el ámbito académico como profesional.



## Apéndice A

# Planificación del TFM

TAREAS	TIEMPO					
	Diciembre	Enero	Febrero	Marzo-Abril	Mayo	Junio
Elección del tema	Reuniones con la empresa					
Búsqueda del dataset	Obtención de los datos					
Documentar		Lectura de <i>papers</i> y documentación	Redacción de los capítulos			
Modelos		Repaso de conceptos				
MLOps		Formación y estudio de los conceptos de MLOps				
MLflow			Formación y estudio de los conceptos de MLflow			
Ejemplo práctico			Creación de la aplicación y casos de usos			
Github				Documentación de la aplicación y TFM en Github		
Anexos					Agregación de documentación a temas poco detallados	
Conclusiones						Realización de las conclusiones
Repaso y entrega						Repaso para su entrega
Reuniones	Durante todo el proceso he estado en contacto con la tutora del TFM y la empresa					



## Apéndice B

# Herramientas

Veamos de manera resumida las herramientas que se han usado:

- Se ha hecho uso de la versión 3.7 de [Python](#).
- Para tratar los datos se han usado [Pandas](#) y [Spark](#), principalmente. Y para la representación gráfica la biblioteca [Plotly](#).
- Para la creación de los modelos de Machine Learning se ha usado la biblioteca de aprendizaje automático [scikit-learn](#).
- En el entorno local se han usado los *notebooks* de [Jupyter](#). Se planteó usar [Google Colab](#), no obstante, se comprobó que el rendimiento con los 9 de millones de datos era peor por lo que se descartó esta alternativa.
- En el entorno *cloud* se han usado los *notebooks* de [Azure Databricks](#) en Azure.
- Para el despliegue de los modelos y el control del ciclo de vida de los modelos ML se ha usado [MLflow](#).
- Respecto el desarrollo de la aplicación web, se ha hecho uso del *framework* [Flask](#) junto con Python, ya que Flask es un *framework* escrito en Python que permite crear aplicaciones web rápidamente y con un mínimo número de líneas de código. Y para la creación de las páginas web se ha usado [HTML](#) y [Bootstrap](#).

Como sabemos, las prestaciones de la infraestrura en la nube son las proporcionadas por Microsoft Azure de acuerdo a lo que paguemos. Pero en local las prestaciones del ordenador donde se han realizado las implementaciones son (figura B.1):



Figura B.1: Especificaciones de MacOS



## Apéndice C

# Análisis descriptivo de los datos

### Lectura y carga de los datos

Al cargar los datos nos aparece el *warning* “*Columns (28) have mixed types. Specify dtype option on import or set low\_memory=False*” (como podemos ver [aquí](#)). Esto es debido a que los datos de aprendizaje en memoria ocupan más de 5.5GB, siendo los tipos de datos: `float64`, `int64` y `object`. Si tenemos en cuenta algunas referencias como [10], es necesario modificar dichos tipos con el fin de reducir el uso en memoria:

- Cambiar `object` por `category` para ahorrar memoria sin modificar el formato del dato [21] y, en general, bajar la codificación de 64 bits a 32 ó 16.
- Cambiar los valores binarios a `int8` (p. ej. la variable `IsBeta` toma valores 0 ó 1, pero al estar en `int64` se está reservando más tamaño del necesario). Los valores binarios con *missings* se modifican a `float16`, debido a que `int` no entiende *nan*.

Se realizan las transformaciones tanto en el conjunto de aprendizaje como en el de prueba; y al efectuar estos cambios, se reduce de 5.5GB a 1.9GB el conjunto de aprendizaje y de 4.8GB a 1.7GB el de prueba (pincha [aquí](#) para ver las modificaciones).

### Resumen de los datos

Para saber más sobre los datos `train` y `test`, pincha [aquí](#). De primeras, vemos como el conjunto etiquetado está balanceado (figura C.1).

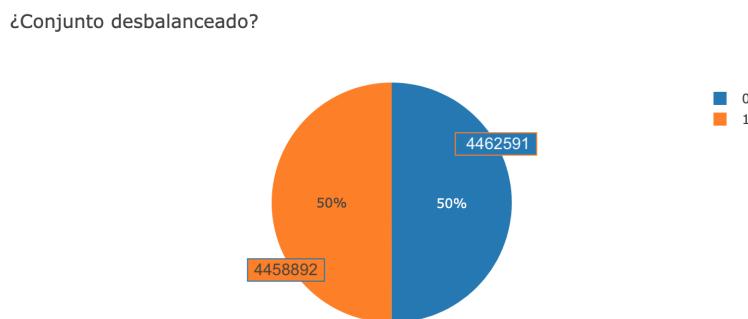


Figura C.1: Gráfico con la distribución de la variable `target` `HasDetections` para el conjunto de datos de aprendizaje

La dimensión del conjunto de datos de aprendizaje es de (8921483, 83), donde hay 33 variables numéricas, 21 binarias y 29 categóricas. Y la del conjunto de prueba de (7853253, 82), una variable menos dado que no se dispone del conjunto etiquetado.

### Missing

En el conjunto de aprendizaje se dispone de 60686127 celdas con observaciones vacías (ver [aquí](#)), es decir, hay 8.2 % de observaciones vacías en total, en donde no existe una distribución equitativa en cuanto al reparto de *missings*, puesto que hay variables que no tienen ninguno y otras que tienen más del 90 % (figura C.2). Entonces, para el tratamiento de los *missings* se requiere un estudio minucioso, en donde existen varios escenarios a aplicar [13]:

1. **Dejar los datos faltantes.** Esta opción se contempla, pero debe ser analizada.
2. **Verificar si es posible encontrar el valor real faltante.** Esta posibilidad no puede ser contrastada, puesto que los datos han sido obtenidos de Kaggle.
3. **Eliminar los datos donde se encuentra el valor perdido, eliminando la variable faltante o la fila completa donde se encuentre.** Esta alternativa debe ser analizada, ya que si se opta por eliminar filas con datos faltantes, se debería escoger aquellas variables con pocas observaciones vacías, para hacer el menor impacto posible (eliminar ruido sin perder información).
4. **Reemplazar los datos faltantes, mediante la imputación.** Al aplicar esta opción se reemplazan los datos faltantes por una conjeta que puede ser tanto cierta como falsa. En este caso, dejar el valor faltante beneficiaría, debido a que se dispone tanto de variables categóricas como numéricas, algunas con muchos valores, por lo que tendríamos un bajo porcentaje para aceptar dicho valor.

	TotalNA	PorcentajeNA
PuaMode	8919174	99.97
Census_ProcessorClass	8884852	99.59
DefaultBrowsersIdentifier	8488045	95.14
Census_IsFlightingInternal	7408759	83.04
Census_InternalBatteryType	6338429	71.05
Census_ThresholdOptIn	5667325	63.52
Census_IsWIMBootEnabled	5659703	63.44

Figura C.2: Tabla con las variables que tienen más de 60 % de *missings*

### Información sobre variables demasiado sesgadas (asimetría)

Otra manera para reducir la dimensionalidad de las variables y eliminar ruido en el dataset, es estudiar aquellos campos donde más del 80 % de sus valores forman parte de la primera categoría. En la figura C.3, se puede contemplar cómo hay variables con más de un 99 % de sus valores en una sola categoría (pincha [aquí](#) para saber más).

variable	asimetria	valores_unicos
Census_IsWIMBootEnabled	99.999969	2
IsBeta	99.999249	2

Figura C.3: Tabla con variables más asimétricas

## Apéndice D

# Preprocesamiento de los datos

Después del análisis descriptivo de los datos, y de la extracción de información sobre los mismos, veamos qué técnicas aplicar en el preprocesamiento (se modificarán tanto los datos *train* como *test*). Además, es importante destacar que este preprocesamiento se ha ido revisando a la par que se entrenaban los modelos, con el fin de obtener los mejores datos para el modelado (para encontrar más información ir a [aquí](#)).

### Eliminar variables que no aportan información

A partir de lo comentado en el [análisis exhaustivo para cada variable](#), se van a eliminar 20 variables. Entonces, con dichas modificaciones, se pasa de tener un conjunto de datos de 83 variables a 63 para *train*, y de 82 a 62 para *test*. Veamos la justificación:

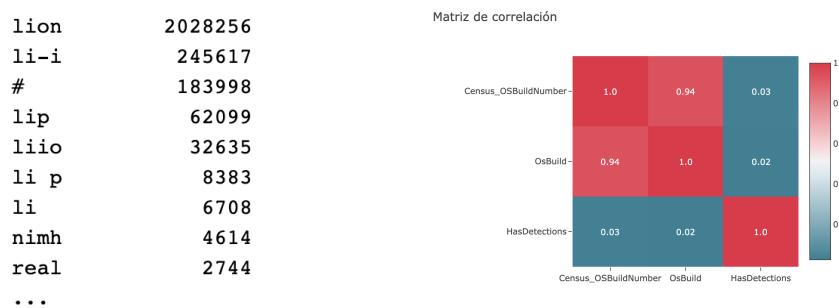
1. `MachineIdentifier`: identificador único que no aporta información.
2. `AvSigVersion`: variable redundante y correlacionada con `EngineVersion`.
3. `DefaultBrowsersIdentifier`: identificador con 8488045 *missings* (95.14 %).
4. `AVProductStatesIdentifier`: identificador con 28970 valores únicos que tiene 36221 *missings* (0.41 %) y correlación negativa con `AVProductsInstalled`.
5. `GeoNameIdentifier`: altamente relacionada con `CountryIdentifier` y otras.
6. `OsBuildLab`: aporta información redundante y hace uso de información segmentada en `OsBuild`, `OsPlatformSubRelease` y `Processor`, entre otras (figura D.1).

<code>OsBuildLab[0]</code>	17134.1.amd64fre.rs4_release.180410-1804
<code>OsPlatformSubRelease[0]</code>	rs4
<code>OsBuild[0]</code>	17134
<code>Processor[0]</code>	x64

**Figura D.1: Información de la variable OsBuildLab**

7. `AutoSampleOptIn`: no hay información de esta variable binaria pero tiene 258 valores para el 1 y 8921225 para la categoría 0 (99.99 %).
8. `Census_ProcessorModelIdentifier`: no hay información de esta variable pero presenta una alta correlación con `Census_ProcessorManufacturerIdentifier`.

9. **Census\_ProcessorClass**: a partir de **SkuEdition** y **Processor** se puede deducir la clase a la que pertenece el procesador y tiene 8884852 *missings* (99.59 %).
10. **Census\_InternalPrimaryDisplayResolutionVertical**: presenta correlación muy alta con **Census\_InternalPrimaryDisplayResolutionHorizontal**.
11. **Census\_OSVersions**: indica la versión del SO (10.0.17134.228), esta información se encuentra segmentada en variables como **OsVer** y **Census\_OSBUILDRevision**.
12. **Census\_InternalBatteryType**: tiene 6338429 *missings* (71.05 %) y dispone de etiquetas escritas de forma inconsistente (figura D.2).
13. **Census\_OSBUILDNumber**: alta correlación con la variable **OsBuild** (figura D.3).



**Figura D.2: Información de la variable *Census\_InternalBatteryType***

**Figura D.3: Información de la variable *Census\_OSBUILDNumber***

14. **Census\_OSSkuName**: variable sesgada y alta correlación con **Census\_OSEdition**.
15. **Census\_OSBBranch**: amplía la información de la variable **OsPlatformSubRelease**, teniendo una alta correlación con ella.
16. **Census\_IsFlightingInternal**: variable binaria que tiene 21 observaciones para el valor 1 y 7408759 *missings* (83.04 %).
17. **Census\_IsWIMBootEnabled**: no hay información de esta variable binaria pero tiene un único valor para el 1 y 5659703 *missings* (63.44 %).

### Estudiar el uso del PCA (Análisis de Componentes Principales)

El método de componentes principales transforma un conjunto de variables (originales) en un nuevo conjunto de variables denominadas componentes principales, las

cuales se caracterizan por estar incorreladas entre sí [4]. Al realizar este análisis perdemos la interpretabilidad de los datos. En consecuencia, se descarta aplicar un PCA, puesto que el objetivo final del proyecto reside en tener el significado de cada variable.

### ***Missing***

Después de haber reducido la dimensionalidad del dataset, al haber eliminado 20 variables, se dispone de una variable con más del 60 % de observaciones vacías, `PuaMode`. Dicha variable tiene dos categorías: `on` con 2307 valores y `audit` con 2. También posee 99.97 % observaciones faltantes (8919174), por lo que se decide prescindir de ella. No obstante, no solamente es importante eliminar columnas (variables), sino que es interesante ver qué filas (observaciones) podemos eliminar. Como vimos [aquí](#), una buena idea para reducir la dimensionalidad del dataset es eliminar aquellas observaciones que pertenecen a variables con bajo número de *missings*. Por tanto, se van a extraer y eliminar las observaciones de las variables que tienen menos de un 0.5 % de *missings*<sup>1</sup>.

En las dos situaciones que acabamos de comentar, no se han imputado valores perdidos, debido a que en muchos casos disponer del valor faltante beneficia en lugar de su imputación. A causa de que se dispone de variables con muchos valores, por lo que se tendría un bajo porcentaje para aceptar el dato de la celda correspondiente. Además, si tenemos dos máquinas con las mismas propiedades a excepción de una propiedad, es posible que dicha característica sea determinante para decidir si un sistema Windows se va infectar o no de *malware*.

### ***Outliers***

Para detectar los *outliers*, nos centramos en aquellas variables que son numéricas. Como sabemos, un valor atípico es algo diferente de la multitud [29]. En este caso, después de estudiar los posibles *outliers*, se llega a la conclusión que no son “*outliers*”, por lo que no se elimina nada en este apartado.

### **Pasar a minúsculas todas las variables categóricas**

Otra manera de limpiar el dataset, es aplicando un tratamiento a aquellas variables categóricas que posean etiquetas tanto en mayúsculas como en minúsculas. Por ejemplo, `SmartScreen` tiene `Off` y `off`, etiquetas que significan lo mismo. Por esa razón, se analizan las variables para pasar todos sus valores a minúsculas (figuras D.4 y D.5).

### **Realizar transformaciones en ciertas variables**

De forma más detallada, se realizan algunas transformaciones específicas. Por ejemplo, los valores *missings* en las variables categóricas se convierten en la etiqueta `unknown`, para así disponer de una categoría que contabilice lo “desconocido”. Además, algunas de esas variables tienen la etiqueta `unspecified` definida, por lo que se debe unificar con `unknown`. En la figura D.7, podemos ver un ejemplo con el campo `Census_PrimaryDiskTypeName`.

<sup>1</sup>El hecho de haber cogido las variables con menos de un 0.5 % de *missings*, se ha debido a que la variable `IsProtected`, la cual se considera muy importante para discriminar si se ha infectado o no un dispositivo de malware, tiene 0.5 % de observaciones vacías, por lo que al eliminar dichas filas estaríamos eliminando información que se considera importante.

```
RequireAdmin 4316183
ExistsNotSet 1046183
Off 186553
Warn 135483
Prompt 34533
Block 22533
off 1350
On 731
&#x02;
416
&#x01;
335
on 147
requireadmin 10
OFF 4
0 3
Promt 2
```

**Figura D.4:** Variable SmartScreen sin tratar mayúsculas

```
requireadmin 4316194
existsnotset 1046183
off 187907
warn 135484
prompt 34534
block 22533
on 878
&#x02;
416
&#x01;
335
0 3
promt 2
&#x03;
1
enabled 1
00000000 1
```

**Figura D.5:** Variable SmartScreen tratando mayúsculas

```
requireadmin 4266939
unknown 3108503
existsnotset 1032045
off 176181
warn 133970
prompt 32572 *
block 22237
on 869
&#x02;
415
&#x01;
321
0 3
promt 2 *
enabled 1
&#x03;
1
00000000 1
```

**Figura D.6:** Errores ortográficos en la variable SmartScreen

Otras modificaciones importantes para aplicar en las variables categóricas, consiste en arreglar los errores ortográficos que contienen. Por ejemplo, la variable `SmartScreen` tiene algunas inconsistencias como `prompt` y `promt` (figura D.6). Por otro lado, otra transformación interesante es la creación de nuevas variables. Por ejemplo, la variable `Census_TotalPhysicalRAM` da información acerca de la RAM física en MB, pero la mayoría de las RAM se suelen dar en GB. Si dejamos la variable en MB se tiene 3290 valores únicos, pero si creamos una nueva variable para contabilizar la RAM en GB y redondeamos hacia arriba, pasamos de 3290 valores únicos a 91.

```
hdd      5744261
ssd      2446206
unknown  309926 *
unspecified 273667 *

hdd      5744261
ssd      2446206
unknown  583593 *
```

**Figura D.7:** Variable `Census_PrimaryDiskTypeName` sin y con la unificación de `unspecified` con `unknown`

### Convertir las variables categóricas a *Label Encoding*

En el [análisis descriptivo de los datos](#) se realizaron algunas comparaciones para saber la correlación entre variables con el mismo tipo. Sin embargo, es posible que haya variables con tipos diferentes y qué mantengan una relación muy alta entre ellas, es decir, comparar variables numéricas con categóricas. Por eso, se estudia pasar las variables categóricas a *Label Encoding* (figura D.8) o a *One Hot Encoding* (figura D.9).

Se descarta hacer uso de *One Hot Encoding*, dado que disponemos de 14 variables categóricas, cada una con muchas etiquetas, por lo que realizar *One Hot Encoding* aumentaría considerablemente el tiempo de computación a la par que la dimensionalidad del dataset [9]. Es por eso que se ha considerado aplicar *Label Encoding* a las columnas categóricas (figura D.8). Además, algunos algoritmos como Random Forest necesitan de esta transformación para las variables categóricas no binarias, puesto que en el caso de las variables categóricas binarias la división del nodo se produciría en 0.5 y las categorías quedarían separadas.

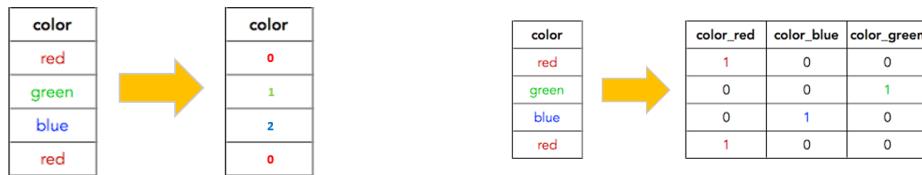


Figura D.8: Ejemplo gráfico con la idea de Label Encoding [7]

Figura D.9: Ejemplo gráfico con la idea de One Hot Encoding [8]

### Estudiar y tratar la correlación

Una forma de medir la relación existente entre las variables en nuestro dataset, es calcular la correlación entre todas las variables y contra la variable *target*. La correlación indica la fuerza y la dirección de una relación lineal y proporcionalidad entre dos variables estadísticas. Dado que disponemos del dataset transformado a *Label Encoding* podemos aplicar la correlación para todas las variables, puesto que después de realizar las anteriores modificaciones es posible que haya variables que se correlacionen:

- **IsSxsPassiveMode** y **RtpStateBitfield**: ambas variables están relacionadas negativamente. En ambos casos la primera categoría abarca más del 96 % de los datos. **IsSxsPassiveMode** tiene 2 categorías y una de ellas contiene el 98.27 % del total; para **RtpStateBitfield** se tienen 7 categorías donde la primera contiene un 97.32 % del total. Por tanto, se elimina **RtpStateBitfield**, ya que tiene más categorías y aporta casi lo mismo que **IsSxsPassiveMode**.
- **OsVer** y **Platform**: **OsVer** indica la versión del SO y **Platform** indica el nombre de la plataforma (propiedades relacionadas con el SO y el procesador). Ambas tienen una alta correlación (figura D.10), pero se acaba eliminando **OsVer**.

Matriz de correlación

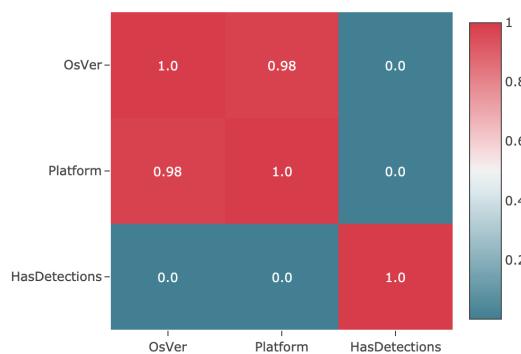


Figura D.10: Correlación entre OsVer y Platform

### Partición de los datos

Por último, guardamos el nuevo dataset con todas las transformaciones realizadas. Además, se lleva a cabo una partición del conjunto de datos de aprendizaje, donde un 75 % va para entrenamiento (6580545, 59) y un 25 % va para validación (2193515, 59).



## Apéndice E

# AutoML

Por último, es interesante estudiar el funcionamiento y la aplicación que supone usar herramientas de AutoML sobre este proyecto. Como queremos predecir con la mayor precisión posible, debemos saber que existen otros métodos a aplicar para obtener predicciones, en concreto, hablamos del **aprendizaje automático automatizado** o AutoML, el cual combina los principales algoritmos de Machine Learning y aprendizaje estadístico con el Big Data. Dentro de las posibles herramientas que nos podemos encontrar tanto privadas como de código abierto, se ha considerado probar la opción en la nube de **Microsoft Automated Machine Learning** y la alternativa de código abierto **H2O AutoML**. No obstante, dado que no se obtienen resultados sustancialmente mejores con H2O en comparación con nuestros modelos, se prueba AutoML que se encuentra dentro del recurso de Azure Machine Learning.

Configuración y vista previa

Esta configuración se ha detectado automáticamente. Compruebe que se hayan seleccionado las opciones correctas o modifíquelas.

Formato de archivo

Delimitado

Delimitador Ejemplo  
Comma Campo1,Campo2,Campo3

Codificación UTF-8

Encabezados de columna Sin encabezados

Omisión de filas Ninguno

Column1	Column2	Column3	Column4	Column5	Column6	Column7	Column8	Column9	Column10	Column11	Column12	Column13	Column14	Column15	Column16
Machine...	Product...	EngineL...	AppVersi...	AvgEngVer...	isBeta	RunState...	isExpPas...	DefaultB...	AIProduc...	AIProduc...	HasTpm	CountryId...	CityIdentif...	Organiza...	
0000028...	win8def...	1.1.1510...	4.18.180...	1.273.17...	0	7	0	53447	1	1	1	29	128035	18	
0000075...	win8def...	1.1.1460...	4.13.171...	1.263.48.0	0	7	0	53447	1	1	1	93	1462	18	
0000078...	win8def...	1.1.1510...	4.18.180...	1.273.13...	0	7	0	53447	1	1	1	86	153579	18	
0000091...	win8def...	1.1.1510...	4.18.180...	1.273.15...	0	7	0	53447	1	1	1	88	20710		
0000149...	win8def...	1.1.1510...	4.18.180...	1.273.13...	0	7	0	53447	1	1	1	18	37376		
0000161...	win8def...	1.1.1510...	4.18.180...	1.273.10...	0	7	0	53447	1	1	1	97	13598	27	
0000161...	win8def...	1.1.1510...	4.18.180...	1.273.84...	0	7	0	43927	2	1	1	76	81215		
0000195...	win8def...	1.1.1510...	4.18.180...	1.273.13...	0	7	0	53447	1	1	1	97	150323	27	
0000196...	win8def...	1.1.1520...	4.18.180...	1.275.98...	0	7	0	53447	1	1	1	164	155006	27	

Atrás Siguiente Cancelar

Figura E.1: Paso 1 en AutoML - Cargar los datos

Para iniciar un proceso de AutoML, dentro de las posibles configuraciones que conlleva hacer uso de una herramienta de este tipo, simplemente debemos subir nuestros datos de aprendizaje (figuras E.1, y E.2), indicar cuál es la variable objetivo a predecir

(figura E.3), decir que queremos usar validación cruzada y que queremos obtener el mejor modelo de acuerdo al mejor valor obtenido para el área bajo la curva. Una de las ventajas de hacer uso de esta herramientas, es que aunque los datos estén limpios, él mismo los preprocesa y te da una explicación de lo que ha realizado exactamente.

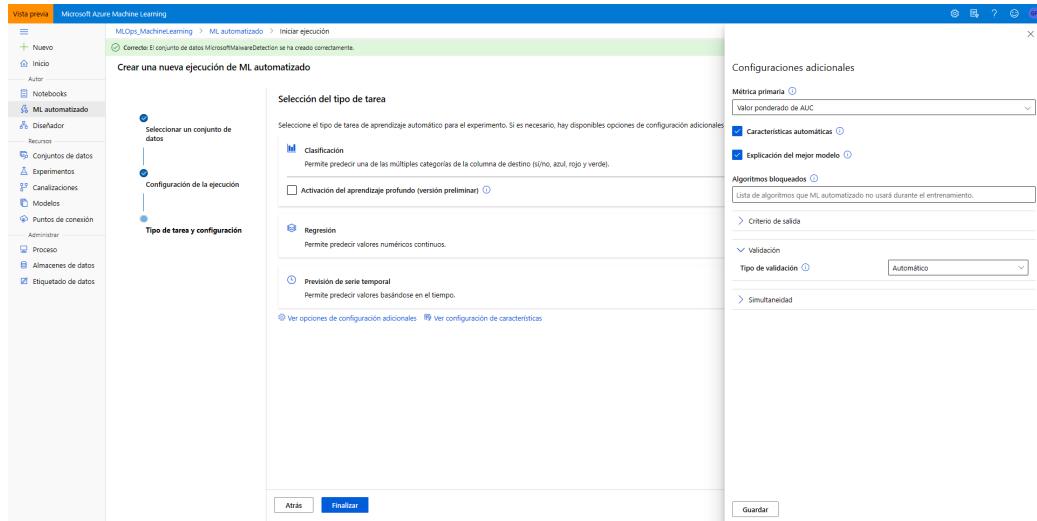
Incluir	Nombre de columna	Propiedades	Tipo	Ejemplo y configuración de formato
<input checked="" type="checkbox"/>	Path	No aplicable al tipo seleccionado	Cadena	UI/02-28-2020_091959_UTC/train_malware.csv, UI/02-28-2020_091959_UTC/train_mal...
<input checked="" type="checkbox"/>	MachineIdentifier	No aplicable al tipo seleccionado	Cadena	0000028988387b115f6931a3b04f09, 000007535c3f730ef9ea0b7ef1bd645, 0000079...
<input checked="" type="checkbox"/>	ProductName	No aplicable al tipo seleccionado	Cadena	win8defender, win8defender, win8defender
<input checked="" type="checkbox"/>	EngineVersion	No aplicable al tipo seleccionado	Cadena	1.1.15100.1, 1.1.14600.4, 1.1.15100.1
<input checked="" type="checkbox"/>	AppVersion	No aplicable al tipo seleccionado	Cadena	4.18.1807.18075, 4.13.17134.1, 4.18.1807.18075
<input checked="" type="checkbox"/>	AvSigVersion	No aplicable al tipo seleccionado	Cadena	1.273.1735.0, 1.263.48.0, 1.273.1341.0
<input checked="" type="checkbox"/>	IsBeta	No aplicable al tipo seleccionado	Entero	0, 0, 0
<input checked="" type="checkbox"/>	RtpStateBitField	No aplicable al tipo seleccionado	Entero	7, 7, 7
<input checked="" type="checkbox"/>	IsSxsPassiveMode	No aplicable al tipo seleccionado	Entero	0, 0, 0
<input checked="" type="checkbox"/>	DefaultBrowsersIdentifier	No aplicable al tipo seleccionado	Entero	null, null, null
<input checked="" type="checkbox"/>	AVProductStatesIdentifier	No aplicable al tipo seleccionado	Entero	53447, 53447, 53447
<input checked="" type="checkbox"/>	AVProductsInstalled	No aplicable al tipo seleccionado	Entero	1, 1, 1
<input checked="" type="checkbox"/>	AVProductsEnabled	No aplicable al tipo seleccionado	Entero	1, 1, 1
<input checked="" type="checkbox"/>	HasTpm	No aplicable al tipo seleccionado	Entero	1, 1, 1

Figura E.2: Paso 2 en AutoML - Definir tipo y datos a usar

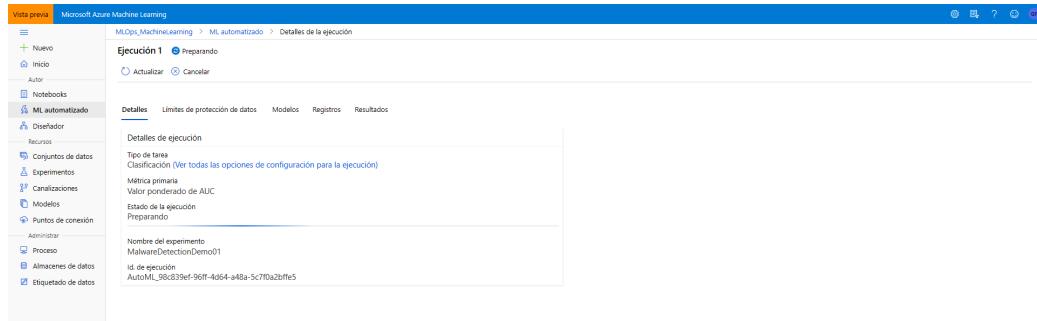
Figura E.3: Paso 3 en AutoML - Indicar la variable objetivo a predecir

A continuación, una vez que disponemos de los datos, hay que crear una ejecución para el ML automatizado (figura E.4), donde se le indique qué problema es (clasificación, regresión), el tipo de validación a usar y la métrica de acuerdo con la que obtener el mejor modelo (área bajo la curva o AUC). Después de añadir esas configuraciones

ya podemos ejecutar nuestro ML automatizado (figura E.5), y él solo se encarga de realizar todo lo necesario.



**Figura E.4: Paso 4 en en AutoML - Crear ejecución con ML automatizado**



**Figura E.5: Paso 5 en en AutoML - Ejecutar ejecución del ML automatizado**

La idea de usar AutoML surge de proporcionarle el conjunto de datos sin limpiar, para que él haga todo. Básicamente en este caso de uso, todo el proceso sería llevado a cabo por la máquina. No obstante, estas herramientas tienen limitaciones, y una de ellas es que no es capaz de preprocesar tal cantidad de datos, por lo que actualmente esta herramienta no tiene sentido usarla en problemas grandes de Big Data.



# Bibliografía

- [1] AI Wiki. *Machine Learning Operations (MLOps)*. Acceso en: 2 de Junio 2020. Disponible en: <https://docs.paperspace.com/machine-learning/wiki/machine-learning-operations-mlops>.
- [2] Y. E. Arias. *Machine Learning: Conceptos básicos*. Acceso en: 19 de Mayo 2020. Disponible en: <https://www.linkedin.com/feed/update/urn:li:activity:6663148358632296449/>.
- [3] Y. N. Bellini Saibene, M. Volpacchio, S. Banchero, and R. Mezher. Desarrollo y uso de herramientas libres para la explotación de datos de los radares meteorológicos del inta. 09 2014.
- [4] S. de la Fuente Fernández. *Componentes Principales*. Acceso en: 20 de Mayo 2020. Disponible en: [http://www.estadistica.net/Master-Econometria/Componentes\\_Principales.pdf](http://www.estadistica.net/Master-Econometria/Componentes_Principales.pdf).
- [5] A. Desarda. *MLOps: The Upcoming Shining Star. The right path to building a full-stack machine learning system. MLOps is the new emerging practise to streamline managing the ML lifecycle*. Acceso en: 26 de Mayo 2020. Disponible en: <https://towardsdatascience.com/mlops-the-upcoming-shining-star-dcf9444c493>.
- [6] Gartner Research. *Accelerate Your Machine Learning and Artificial Intelligence Journey Using These DevOps Best Practices*. Acceso en: 17 de Mayo 2020. Disponible en: <https://www.gartner.com/en/documents/3975098/accelerate-your-machine-learning-and-artificial-intellig>.
- [7] GeeksforGeeks: A computer science portal for geeks. *ML / Label Encoding of datasets in Python*. Acceso en: 20 de Mayo 2020. Disponible en: <https://www.geeksforgeeks.org/ml-label-encoding-of-datasets-in-python/>.
- [8] GeeksforGeeks: A computer science portal for geeks. *ML / One Hot Encoding of datasets in Python*. Acceso en: 20 de Mayo 2020. Disponible en: <https://www.geeksforgeeks.org/ml-one-hot-encoding-of-datasets-in-python/>.
- [9] Interactive Chaos. *Label Encoding and One Hot Encoding*. Acceso en: 20 de Mayo 2020. Disponible en: <https://www.interactivechaos.com/manual/tutorial-de-machine-learning/label-encoding>.
- [10] Kaggle. *Load the Totality of the Data*. Acceso en: 20 de Mayo 2020. Disponible en: <https://www.kaggle.com/theoviel/load-the-totality-of-the-data>.
- [11] Kaggle. *Microsoft Malware Prediction*. Acceso en: 18 de Mayo 2020. Disponible en: <https://www.kaggle.com/c/microsoft-malware-prediction>.

- [12] M. A. Kate Baroni, Software Architect. *Getting AI/ML and DevOps working better together*. Acceso en: 17 de Mayo 2020. Disponible en: <https://azure.microsoft.com/es-es/blog/getting-ai-ml-and-devops-working-better-together/>.
- [13] Ligdi González: Aprende todo sobre Inteligencia Artificial. *Manipulando datos perdidos en Python*. Acceso en: 19 de Mayo 2020. Disponible en: <https://ligdgonzalez.com/manipulando-datos-perdidos-en-python/>.
- [14] F. J. P. López and K. Albes. *MLOps: Software best practices for building Machine Learning solutions*. Marzo 2020. Virtual Coffee, Plain Concepts.
- [15] L. Matsumota. *Machine and Operations Learning (MLOps): Learn more about machine and operations learning (MLOps) and see benefits and the different phases*. Acceso en: 25 de Mayo 2020. Disponible en: <https://dzone.com/articles/machine-and-operations-learning-mlops>.
- [16] G. Mendels and N. Laskaris. *Why software engineering processes and tools don't work for ML*. Acceso en: 26 de Mayo 2020. Disponible en: <https://medium.com/comet-ml>.
- [17] Microsoft. *MLflow*. Acceso en: 26 de Mayo 2020. Disponible en: <https://docs.microsoft.com/es-es/azure/databricks/applications/mlflow/>.
- [18] Microsoft. *¿Qué es el aprendizaje automático automatizado (AutoML)?* Acceso en: 17 de Mayo 2020. Disponible en: <https://docs.microsoft.com/es-es/azure/machine-learning/concept-automated-ml>.
- [19] Microsoft Azure. *Operaciones de Machine Learning (MLOps)*. Acceso en: 17 de Mayo 2020. Disponible en: <https://azure.microsoft.com/es-es/services/machine-learning/mlops/>.
- [20] MLflow. *MLflow Documentation*. Acceso en: 26 de Mayo 2020. Disponible en: <https://www.mlflow.org/docs/latest/index.html>.
- [21] Pandas. *Categorical data*. Acceso en: 19 de Mayo 2020. Disponible en: [https://pandas.pydata.org/pandas-docs/stable/user\\_guide/categorical.html](https://pandas.pydata.org/pandas-docs/stable/user_guide/categorical.html).
- [22] Paradigma Digital. *MLOps: Machine learning en producción con Google y MLflow*. Acceso en: 26 de Mayo 2020. Disponible en: <https://www.paradigmadigital.com/dev/mlops-produccion-google-mlflow/>.
- [23] Red Hat. *¿Qué significa DevOps?* Acceso en: 25 de Mayo 2020. Disponible en: <https://www.redhat.com/es/topics/devops>.
- [24] J. V. Román. *CRISP-DM: La metodología para poner orden en los proyectos*. Acceso en: 25 de Mayo 2020. Disponible en: <https://www.sngular.com/es/data-science-crisp-dm-metodologia/>.
- [25] Scikit Learn. *Documentation Gradient Boosting classifier*.
- [26] Scikit Learn. *Documentation Logistic Regression classifier*. Acceso en: 4 de Junio 2020. Disponible en: [https://scikit-learn.org/stable/modules/generated/sklearn.linear\\_model.LogisticRegression.html](https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html).

- [27] Scikit Learn. *Documentation Random Forest classifier.* Acceso en: 4 de Junio 2020. Disponible en: <https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html>.
- [28] D. V. Serrano. *Minería de Datos.* Febrero 2020. Apuntes de la asignatura de Minería de Datos en el Máster TECI, Curso 2019/2020.
- [29] N. Sharma. *Ways to Detect and Remove the Outliers.* Acceso en: 20 de Mayo 2020. Disponible en: <https://towardsdatascience.com/ways-to-detect-and-remove-the-outliers-404d16608dba>.
- [30] N. Talagala. *Why MLOps (and not just ML) Is Your Business' New Competitive Frontier.* Acceso en: 26 de Mayo 2020. Disponible en: <https://aibusiness.com/mlops-parallelml-competitive-edge/>.
- [31] N. Thacker. *MLflow and Azure ML. The Power Couple for ML Lifecycle Management.* Acceso en: 18 de Mayo 2020. Disponible en: <https://databricks.com/sparkaisummit/europe/schedule>.

