Trabajo Fin de Máster

Desarrollo e implantación de un sistema de ML enteramente basado en tecnologías SQL de la nube

Alba Camino Enríquez

Máster en Técnicas Estadísticas

Curso 2023-2024

Propuesta de Trabajo Fin de Máster

Título en galego: Desenvolvemento e implantación dun sistema de ML enteiramente baseado en tecnoloxías SQL da nube

Título en español: Desarrollo e implantación de un sistema de ML enteramente basado en tecnologías SQL de la nube

English title: Development and Implementation of an Entirely SQL-Based Cloud ML System

Modalidad: Modalidad B

Autora: Alba Camino Enríquez, Universidad de Santiago de Compostela

Directora: Rosa María Crujeiras Casais, Universidad de Santiago de Compostela.

Tutor: Andres Padrones Garcia, SDG Group.

Breve resumen del trabajo:

Cada vez es más habitual que servicios de data warehousing en la nube incorporen capacidades de ML con el objetivo de permitir la creación y/o explotación de modelos algorítmicos mediante interfaces de fácil uso y conocidas por el analista de datos —i.e., SQL—, al mismo tiempo que se logre realizar todo el proceso allí donde estén los datos y, gracias a ello, conseguir grandes capacidades de procesamiento dada la alta escalabilidad de estas arquitecturas. En base a eso, este TFM propone la evaluación de todos estos aspectos en lo relativo a BigQuery ML, el componente con capacidades de ciencia de datos que Google Cloud ha habilitado en su producto de data warehousing en la nube. Así pues, el foco de este trabajo estará en dar cobertura al ciclo de vida completo de la ciencia de datos (análisis exploratorio, feature engineering, experimentación algorítmica, evaluación, fine tuning, operativización, explotación y monitorización) usando como componente central BigQuery ML, y enriqueciendo todo ello con un enfoque MLOps para garantizar la ágil iteración sobre la solución. Este trabajo se realizará sobre un caso de uso vehicular, con el fin último de demostrar la viabilidad de este tipo de enfoques a la hora de aportar valor a negocio desde una perspectiva de analítica avanzada en un corto tiempo, y utilizando tecnologías que posiblemente ya estén aplicando las compañías en relación a su estrategia de datos global. Beneficio esperado: Establecer el grado de madurez de las tecnologías de data warehousing en la nube con respecto a sus capacidades de ML, validando la viabilidad de su aplicación para el desarrollo e implantación end-to-end de soluciones de ciencia de datos.

Doña Rosa María Crujeiras Casais, Catedrática del área de Estadística y Investigación Operativa de la Universidad de Santiago de Compostela, don Andres Padrones Garcia, Head Data Scientist de SDG Group, informan que el Trabajo Fin de Máster titulado

Desarrollo e implantación de un sistema de ML enteramente basado en tecnologías SQL de la nube

fue realizado bajo su dirección por doña Alba Camino Enríquez para el Máster en Técnicas Estadísticas. Estimando que el trabajo está terminado, dan su conformidad para su presentación y defensa ante un tribunal.

En Santiago de Compostela, a 3 de Junio de 2024.

La directora:
Doña Rosa María Crujeiras Casais

El tutor:
Don Andres Padrones Garcia

La autora:
Doña Alba Camino Enríquez

Declaración responsable. Para dar cumplimiento a la Ley 3/2022, de 24 de febrero, de convivencia universitaria, referente al plagio en el Trabajo Fin de Máster (Artículo 11, Disposición 2978 del BOE núm. 48 de 2022), **el/la autor/a declara** que el Trabajo Fin de Máster presentado es un documento original en el que se han tenido en cuenta las siguientes consideraciones relativas al uso de material de apoyo desarrollado por otros/as autores/as:

- Todas las fuentes usadas para la elaboración de este trabajo han sido citadas convenientemente (libros, artículos, apuntes de profesorado, páginas web, programas,...)
- Cualquier contenido copiado o traducido textualmente se ha puesto entre comillas, citando su procedencia.
- Se ha hecho constar explícitamente cuando un capítulo, sección, demostración,... sea una adaptación casi literal de alguna fuente existente.

Y, acepta que, si se demostrara lo contrario, se le apliquen las medidas disciplinarias que correspondan.

Índice general

Resumen

Resumen en español

Este proyecto ilustra la implementación de capacidades de aprendizaje automático, especialmente a través de BigQuery ML, con el propósito de aprovechar la accesibilidad de interfaces SQL para la creación de diversos modelos en BigQuery. Los modelos evaluados se explican tanto de forma teórica como práctica, se comparan con sus equivalentes en Python y abarcan desde la regresión lineal múltiple o la regresión logística hasta métodos combinados, técnicas de *forecasting*, sistemas de recomendación y *clustering*. La comparativa entre BigQuery ML y Python se lleva a cabo mediante el desarrollo de los respectivos modelos en ambos entornos, utilizando conjuntos de datos específicos para evaluar su desempeño y capacidades. El objetivo principal de este estudio es resaltar y comparar las funcionalidades de ambas plataformas, con el objetivo de ofrecer una guía que facilite una elección más adecuada en base a las necesidades específicas de cada proyectos de ciencia de datos.

English abstract

This project illustrates the implementation of machine learning capabilities, especially through BigQuery ML, with the purpose of leveraging the accessibility of SQL interfaces for creating several models in BigQuery. The evaluated models are explained both theoretically and practically, compared to their Python equivalents, and range from multiple linear regression or logistic regression to ensemble methods, forecasting techniques, recommendation systems, and clustering. The comparison between BigQuery ML and Python is conducted by developing the respective models in both environments, using specific datasets to assess their performance and capabilities. The main objective of this study is to highlight and compare the skills of both platforms, aiming to provide a guide that facilitates a more suitable choice based on the specific needs of each data science project.

Capítulo 1

Introducción

1.1. Motivación y objetivos

En el actual ámbito empresarial, la convergencia entre el almacenamiento de datos en la nube y las capacidades de aprendizaje automático ha impulsado una revolución en la forma en que las empresas aprovechan sus recursos analíticos. En el marco de las prácticas llevadas a cabo en la empresa SDG Group, se exploran las sinergias entre el almacenamiento de datos en la nube y el potencial del aprendizaje automático para ofrecer soluciones más ágiles y efectivas. En este contexto, se aborda la evaluación exhaustiva de BigQuery ML, el componente de ciencia de datos habilitado por Google Cloud en su plataforma de almacenamiento en la nube.

Este trabajo se enfoca en analizar, desde una perspectiva técnica, el ciclo completo de la ciencia de datos utilizando BigQuery ML como eje central. En particular, este ciclo pasa por diversas fases, comezando por una compresión empresarial en la cuál se determinan los objetivos y se evalúa la situación. Seguidamente, se recopilan y exploran los datos disponibles, para su posterior preparación. Una vez tenemos los datos limpios y seleccionados se puede pasar al modelado. Se debe seleccionar una técnica de modelado, construir el modelo y evaluarlo. Además de evaluar dicho modelo se deberán evaluar los resultados obtenidos y finalmente llevar a cabo una implementación y mantenimiento del plan. Desde el análisis exploratorio hasta la monitorización, se busca profundizar en el potencial de Bigquery, en comparación con las correspondientes versiones en Python.

Pero, ¿qué es exactamente BigQuery y qué lo hace tan relevante en este contexto? Como decíamos unas líneas más arriba, BigQuery es un *data warehouse* de Google Cloud de bajo coste, multinube, escalable y totalmente administrado. Integra herramientas de análisis capaces de procesar petabytes de datos en segundos, todo ello de una forma rentable y adaptable a las necesidades de cada organización. Su facilidad de uso, su capacidad para manejar grandes cantidades de datos y su integración con herramientas de *Machine Learning* e IA hacen de BigQuery una herramienta muy poderosa. Los datos provenientes de distintas fuentes se almacenan en tablas que pueden organizarse por líneas de negocio y proyectos. Bigquery, integra varias herramientas en una sola, entre las cuáles destacamos explotar adecuadamente grandes volúmenes de datos o construir modelos predictivos. A continuación, destacaremos las principales funcionalidades de Bigquery y sus ventajas frente a otros *softwares*.

En definitiva, nuestro objetivo principal será ilustrar las funcionalidades de Bigquery destacando su simplicidad y agilidad en comparación a otros *software* como es el caso de Python.

1.2. Introducción a Bigquery

BigQuery representa un pilar fundamental para procesar enormes volúmenes de información y emplear herramientas avanzadas de análisis y aprendizaje automático. En esta sección, exploraremos aquellas funcionalidades que hacen de BigQuery una potencial herramienta para el análisis de datos.

1.2.1. Reducción del plazo de lanzamiento

Bigquery ML es el servicio para la creación de modelos de *machine learning* integrado en Bigquery. Este, acelera el proceso de innovación y construcción de modelos, ya que quita la necesidad de exportar datos desde el almacén de datos. Esto no solo acelera la capacidad de BigQuery para procesar rápidamente enormes volúmenes de datos y implementar soluciones analíticas, sino que también desempeña un papel fundamental en la agilización del prototipado de modelos. Al aprovechar la sintaxis de BigQuery, la creación de modelos se convierte en un proceso rápido y directo. Como muestra la Figura 1.1, basta con especificar los datos de entrada mediante una consulta de SQL y algunos parámetros básicos para el tipo de modelo deseado dentro de las opciones. Esto automatiza múltiples pasos que,

```
{CREATE MODEL | CREATE MODEL IF NOT EXISTS | CREATE OR REPLACE MODEL} model_name
OPTIONS(model_option_list)
AS query_statement
```

Figura 1.1: Sintaxis de Bigquery para la creación de un modelo.

de otra manera, requerirían tiempo y conocimientos especializados. Por ejemplo, al tratar con datos faltantes, el proceso se vuelve más eficiente ya que la plataforma puede manejarlos automáticamente, evitando la necesidad de desarrollar estrategias específicas para manipularlos. Esto simplifica el flujo de trabajo y permite a los usuarios enfocarse más en la construcción y experimentación de modelos en lugar de dedicar tiempo a la limpieza y preprocesado de los datos.

Además, la división de datos para entrenamiento, validación y test es una parte crucial en la creación de modelos de aprendizaje automático. BigQuery ML automatiza este proceso, permitiendo segmentar los datos de forma sencilla y rápida. En concreto, no sólo ofrece una división automática basada en el tamaño total de la tabla sino que también brinda opciones más personalizadas, permitiendo la segmentación a través de una columna específica. Además, la plataforma facilita divisiones aleatorias, útiles cuando no hay un criterio definido o se busca evitar sesgos, otorgando mayor flexibilidad en la ejecución de esta división a las necesidades de los usuarios. En el caso de la división aleatoria, evitar sesgos, sugiere que al utilizar este enfoque, se busca prevenir cualquier inclinación injusta o desigual en la asignación de datos a diferentes conjuntos. Cuando se realizan divisiones aleatorias, cada observación o instancia tiene la misma probabilidad de ser asignada a un conjunto específico, lo que contribuye a la imparcialidad en la selección de datos de entrenamiento y prueba. Este enfoque puede ser especialmente útil cuando no hay un criterio definido para la división o cuando se quiere minimizar la posibilidad de introducir sesgos subjetivos en la selección de datos.

Otro punto importante es la obtención automática de métricas de evaluación asociadas a cada modelo. BigQuery ML proporciona estas métricas de manera integrada, lo que facilita la evaluación y validación de múltiples modelos sin requerir un esfuerzo adicional. Esta capacidad permite probar diferentes enfoques o algoritmos de manera rápida y precisa, identificando el modelo más adecuado para un conjunto de datos específico. Este proceso de validación eficiente permite a los equipos tomar decisiones informadas sobre qué modelos implementar sin perder tiempo en pruebas extensas y manuales.

En resumen, la capacidad de la plataforma para gestionar automáticamente aspectos complejos como datos faltantes o la división de conjuntos de datos para entrenamiento, validación y pruebas simplifica el flujo de trabajo y permite a los usuarios enfocarse en la construcción y experimentación de modelos. Esto junto a la capacidad para proporcionar las métricas asociadas a cada modelo agiliza la evaluación y validación de múltiples enfoques, permitiendo a los equipos comparar múltiples opciones para así tomar decisiones sobre qué modelos implementar, sin perder tiempo en pruebas extensas y manuales.

En conjunto, estas características consolidan a BigQuery ML como una herramienta poderosa y versátil que acelera la velocidad y desarrollo de modelos de aprendizaje automático, permitiendo a las empresas optimizar su tiempo y mejorar su productividad.

1.2.2. Habilitación de capacidades de aprendizaje automática

El aprendizaje automático es la ciencia que se centra en el desarrollo de algoritmos y modelos que permiten a las computadoras aprender patrones y tomar decisiones sin ser programadas explícitamente. En lugar de seguir instrucciones específicas, los sistemas de aprendizaje automático utilizan datos para mejorar su rendimiento en una tarea específica a medida que adquieren experiencia, (Norman, 2019). Desarrollar proyectos exitosos de aprendizaje automático requiere una variedad de conocimientos y, por lo tanto, es un trabajo extremadamente colaborativo. Un equipo de ML normalmente consta de analistas de datos, ingenieros e ingenieras de datos, científicos e científicas de datos, científicos e científicas aplicados, ingenieros e ingenieras de aprendizaje automático y desarrolladores. Analizaremos cada uno de estos roles en detalle (Karani, 2022).

- Ingenieros/as de datos. Son responsables de construir y mantener la infraestructura que permite recopilar, transformar, procesar y almacenar datos de manera eficiente y segura. Trabajan con herramientas como Spark o Hadoop para manejar grandes volúmenes de datos y crean almacenes de datos utilizando plataformas en la nube. Su enfoque principal está en asegurar la integridad y disponibilidad de los datos, así como en gestionar procesos ETL (Extract Transform Load), lo que esencialmente significa tomar datos de una fuente, procesarlos y almacenarlos en almacenes de datos. De modo que manejan tareas computacionales intensivas.

- Analistas de datos. Estos profesionales son expertos en extraer información valiosa de los datos de los usuarios. Colaboran estrechamente con equipos comerciales para comprender las necesidades y comportamientos de los clientes. Utilizan herramientas como SQL, Excel, Tableau y Power BI para explorar y analizar datos. Sus habilidades principales incluyen el análisis estadístico y la inferencia para interpretar y presentar datos de manera comprensible y útil.

- Científicos/as de datos. Estos profesionales tienen habilidades en análisis estadístico avanzado y aprendizaje automático. Son capaces de procesar, analizar y interpretar datos complejos para extraer información valiosa y generar modelos predictivos. Utilizan herramientas y lenguajes como Python, R, SQL y técnicas de aprendizaje automático para entender los patrones en los datos y realizar predicciones significativas. Dentro de este marco, englobaríamos a los perfiles del Máster en Técnicas Estadísticas cursado.

- Científicos/as investigadores. Su enfoque se centra en la investigación y el desarrollo de nuevos algoritmos y técnicas de mejora. Realizan investigaciones profundas, publican artículos académicos y resuelven desafíos científicos complejos para mejorar la eficiencia y la precisión de los modelos existentes o para crear nuevos enfoques innovadores.

- Ingenieros/as de aprendizaje automático. Estos, se centran en la implementación práctica de modelos de aprendizaje automático. Trabajan en la infraestructura necesaria para entrenar, actualizar y desplegar modelos. Colaboran estrechamente con científicos de datos para implementar modelos desarrollados por ellos. A menudo, su trabajo abarca múltiples escenarios de implementación, como implementaciones en la nube, locales o perimetrales.

- Desarrolladores. Su rol implica el diseño y desarrollo de interfaces y aplicaciones que permiten a los usuarios interactuar con los modelos de aprendizaje automático. A menudo diseñan las API y dan formato a la predicción del modelo para que sea fácil de usar.

Como hemos visto, cada uno de estos roles aportan conocimientos bastante específicos al proceso de creación de un modelo de *machine learning*. Por lo que en esta sección, nos centraremos fundamentalmente en analizar que beneficios aporta Bigquery a un no *data scientist*.

En el contexto de BigQuery y su impacto en roles no especializados en ciencia de datos, hay varias automatizaciones que facilitan la vida de profesionales como ingenieros de datos o desarrolladores, los cuáles no tienen un conocimiento tan profundo en ciencia de datos.

Por ejemplo, un ingeniero de datos puede enfrentarse a desafíos técnicos al trabajar con variables categóricas, donde la transformación de estas variables en formatos utilizables para modelos de aprendizaje automático requiere cierto conocimiento específico. En concreto, en Python esto puede requerir el conocimiento específico de métodos como *One-Hot Encoding* o *Label Encoding* para convertir datos categóricos en formatos numéricos adecuados. Por otro lado, BigQuery ML ofrece funciones integradas que pueden manejar automáticamente estas transformaciones, convirtiendo variables categóricas en representaciones numéricas sin que el usuario necesite entender los detalles técnicos del proceso.

Otro caso común es la gestión de datos faltantes, como indicamos en el apartado anterior. Para un profesional que no está especializado en ciencia de datos, lidiar con valores nulos o datos faltantes puede ser un desafío. En Python, el manejo de valores faltantes implica la comprensión y aplicación de técnicas como la imputación en media o mediana, eliminación de registros o métodos más avanzados como el uso de modelos para predecir valores faltantes. En cambio, BigQuery ML automatiza el manejo de datos faltantes, permitiendo que estos sean tratados sin necesidad de una intervención manual extensa. Por ejemplo, al utilizar ciertas funciones de BigQuery ML para entrenar modelos, la plataforma puede manejar internamente los datos faltantes sin requerir que el usuario realice ningun tipo de procedimiento para imputarlos.

Además, de nuevo podemos destacar las funcionalidades de BigQuery ML para la división automática de datos en conjuntos de entrenamiento, validación y prueba. Esta característica es beneficiosa para profesionales que no tienen experiencia en la selección y partición de datos para modelos de aprendizaje automático. Estas funciones automatizadas permiten que estos profesionales obtengan resultados de modelos sin la necesidad de comprender los detalles técnicos de cómo se realiza la división de datos o cómo se deben tratar. En Python, esta partición de datos generalmente se realiza utilizando librerías como scikit-learn, lo que requiere la comprensión de funciones específicas y cómo se deben aplicar correctamente para obtener conjuntos de datos equilibrados y representativos.

En general, trabajar con Python para tareas de ciencia de datos y aprendizaje automático requiere un conocimiento más profundo de la programación, la manipulación de datos y el uso de librerías específicas. Esto puede ser una barrera para profesionales que no tienen un enfoque técnico o que están más especializados en otros campos. En contraposición, BigQuery ML simplifica y automatiza procesos complejos de ciencia de datos, permitiendo a profesionales no especializados en esta área utilizar herramientas de aprendizaje automático de manera efectiva sin necesidad de comprender los aspectos técnicos más profundos, lo que les permite aprovechar al máximo las capacidades de ML para sus proyectos específicos.

1.2.3. Simplificación de infraestructuras

Debido a su potente infraestructura, Bigquery se utiliza en múltiples ámbitos con el objetivo de mejorar y agilizar los proyectos. En este contexto, puede resultar interesante introducir terminología como IaaS, PaaS, SaaS.

Cuando nos referimos a IaaS, estamos hablando de infraestructuras como servicio. Es decir, las empresas contratan la infraestructura de hardware a un tercero a cambio de una cuota, esto permite elegir la capacidad del procesador, la memoria a utilizar o el espacio de almacenamiento. Por otro lado, la plataforma como servicio o PaaS es un conjunto de servicios basados en la nube que permite a los desarrolladores y usuarios empresariales crear aplicaciones ya que aporta herramientas de diseño y flujo de trabajo y API completas. Por último, si nos enfocamos, en el modelo SaaS, este aloja software y datos en servidores externos, permitiendo a empleados acceder a las aplicaciones desde cualquier lugar sin instalaciones locales. La empresa se concentra en usar los programas sin preocuparse por recursos como hardware o mantenimiento, ya que son proporcionados y gestionados por el proveedor.

Estos modelos, ofrecen distintos niveles de control y responsabilidad al usuario. IaaS otorga acceso a la infraestructura y su configuración, PaaS permite acceso al software de la plataforma, mientras que SaaS no brinda acceso a la configuración del software. Además, el nivel de seguridad varía, en IaaS, el usuario debe mantener y actualizar las aplicaciones, mientras que en SaaS y PaaS, el proveedor se encarga de la seguridad. De este modo, dependiendo de sus necesidades las empresas o los profesionales,

pueden emplear un modelo IaaS, PaaS o SaaS. Permitiendo así enfocarse en procesos clave al delegar la gestión de recursos al proveedor, lo que reduce costos de infraestructura, sistemas operativos y software.

En particular, BigQuery se clasificaría principalmente como un servicio de PaaS (Plataforma como Servicio). Puesto que ofrece una plataforma integral para el análisis de datos en la nube, brindando a los usuarios las herramientas y la infraestructura necesaria para realizar consultas y análisis complejos sin preocuparse por la configuración o el mantenimiento de la infraestructura subyacente. De modo que los usuarios pueden acceder a él para realizar análisis sin la necesidad de gestionar la infraestructura subyacente, centrando sus esfuerzos en el análisis de datos y no en la administración de la infraestructura.

Comparativamente, Python requiere el uso de herramientas como Docker para encapsular aplicaciones y configuraciones, lo que puede implicar costos adicionales de aprovisionamiento y configuración de recursos específicos. Además, Python necesita servicios adicionales, como almacenamiento de modelos o sistemas de orquestación, mientras que BigQuery, en contraste, simplifica su infraestructura al no requerir estas herramientas o costos asociados.

1.2.4. Interoperabilidad de modelos

La interoperabilidad de modelos se basa en la premisa fundamental de acercar el modelo a los datos, ya sea con el objetivo de reducir costos o de aprovechar la capacidad física disponible de manera eficiente. De modo que buscar incrementar la capacidad de integrar y trabajar con diversos modelos y herramientas es fundamental para fomentar la colaboración entre equipos en el ámbito del Machine Learning. Esto facilita la implementación de soluciones analíticas más completas y eficientes, puesto que permite la transferencia fluida de modelos entre diferentes entornos y plataformas. Dos estándares destacados en este campo son PMML (*Predictive Model Markup Language*) y ONNX (*Open Neural Network Exchange*). Ambos se centran en lograr la interoperabilidad entre modelos, aunque difieren en su enfoque y alcance. PMML se basa en XML y es más antiguo en comparación con ONNX. Está diseñado para representar modelos de Machine Learning de una manera más genérica y puede cubrir una variedad de algoritmos y técnicas de ML. Además, tiene una amplia compatibilidad con diversas herramientas y entornos de Machine Learning. Por otra parte, ONNX está altamente especializado en modelos de redes neuronales profundas y tiene una mayor integración con marcos de Deep Learning específicos, lo que facilita la transferencia de modelos entre ellos. En particular, Python, tiene una gran flexibilidad para trabajar con ambos formatos. Podemos exportar modelos desde bibliotecas de Python, como scikit-learn, a PMML y utilizarlos en BigQuery. Si bien ONNX se enfoca más en redes neuronales, bibliotecas como TensorFlow y PyTorch ofrecen herramientas para exportar modelos a ONNX, los cuales podrían ser utilizados en entornos compatibles con este estándar. Sin embargo, BigQuery, sólo puede trabajar con modelos exportados en formatos compatibles como PMML, es decir, directamente no ofrece soporte nativo para ONNX.

1.2.5. Modelos disponibles

BigQuery ML ofrece una selección específica pero potente de modelos integrados y externos para abordar una amplia gama de tareas de aprendizaje automático y análisis de datos, como se puede ver en la Figura 1.2. Dentro del conjunto de modelos disponibles, diferenciamos entre los modelos entrenados de forma interna y los entrenados de forma externa. Modelos como la regresión lineal, la regresión logística, el agrupamiento en clústeres de k-medias, la factorización de matrices, el análisis de componentes principales (PCA) o las series temporales se entrenan de forma interna. Mientras que, los árboles con *boosting* o los bosques aleatorios son modelos externos a BigQuery ML y están entrenados en Vertex AI. Si bien la oferta de modelos no es tan extensa como la de modelos clásicos, esta limitación se convierte en una ventaja al simplificar significativamente el proceso de elección.

Por ejemplo, si consideramos un problema de *forecasting*, en Python existen librerías que ponen a nuestra disposición una amplia gama de modelos diferentes, desde ARMA y ARIMA hasta SARI-

MAX, AR y MA. Sin embargo, BigQuery ML ofrece una única opción específica para esta tarea. Esta singularidad no limita las capacidades de predicción, sino que simplifica el proceso. La estrategia de

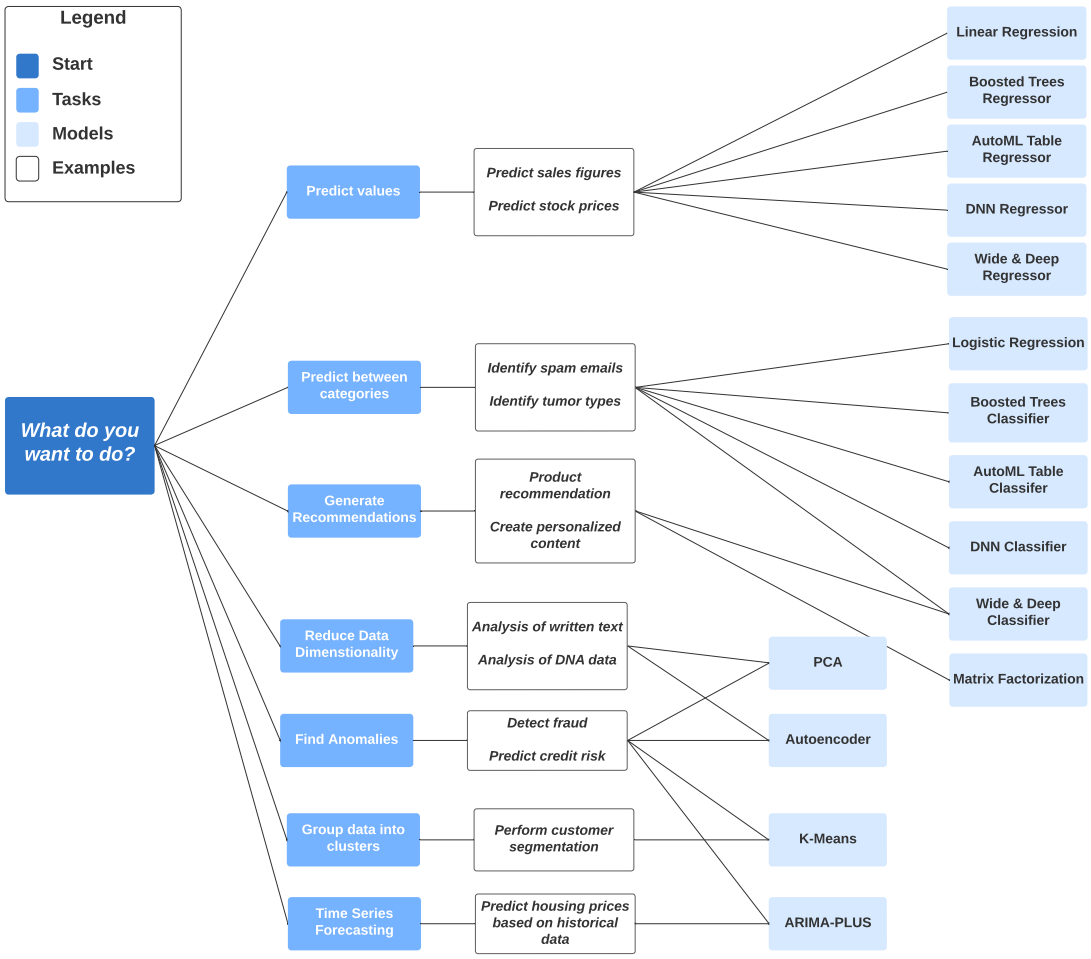


Figura 1.2: Guía de selección de modelos (Google Cloud, 2024).

BigQuery ML de reducir la complejidad de selección de modelos puede ser una ventaja para aquellos que buscan una solución rápida y eficiente para sus necesidades de aprendizaje automático y no tienen muchos conocimientos específicos en el área del problema. En capítulos posteriores, analizaremos con mayor profundidad la metodología asociada a cada uno de estos modelos y introduciremos ejemplos ilustrativos tanto en Python como en Bigquery ML.

Capítulo 2

Metodología

Este capítulo, se presenta como un puente entre la teoría y la práctica, sirviendo de cimiento para la posterior implemetación de los modelos. En ella, se establece el marco teórico necesario para fundamentar la aplicación práctica de los modelos, proporcionando unas bases que facilitarán su comprensión y análisis en capítulos posteriores.

En nuestro recorrido por las técnicas analíticas, nos sumergiremos en los diversos métodos indicados en la Figura 1.2. Comenzaremos explorando la regresión, donde trabajaremos con el modelo de regresión lineal (Sección 2.1), boosted trees y random forest (Sección 2.3) para entender y predecir relaciones entre variables. En el ámbito de la clasificación, nos adentraremos en el modelo de regresión logística (Sección 2.2), así como en boosted trees y random forest, que nos permitirán asignar observaciones a categorías específicas de manera eficiente. En el terreno del forecasting (Sección 2.4), nos enfocaremos en el modelo ARIMA para desentrañar patrones temporales y realizar predicciones precisas. Para la recomendación (Sección 2.5), emplearemos la factorización de matrices, una técnica que revela patrones y preferencias ocultas en grandes conjuntos de datos. Por último, clustering (Sección 2.6) nos llevará a explorar k-medias, una técnica para agrupar datos similares.

Cada técnica se presenta como una herramienta específica, abordando distintos problemas. A lo largo de esta exploración, buscaremos en primer lugar comprender estas metodologías para poder aplicarlas de manera práctica en capítulos posteriores en el análisis de datos.

2.1. Regresión lineal múltiple

La regresión lineal múltiple permite generar un modelo lineal en el que el valor de la variable dependiente o respuesta se determina a partir de un conjunto de variables independientes llamadas predictores. Es una extensión de la regresión lineal simple, por lo que es fundamental comprender esta última. Los modelos de regresión múltiple pueden emplearse para predecir el valor de la variable dependiente o para evaluar la influencia que tienen los predictores sobre ella, (Faraway, 2015).

2.1.1. Introducción

El modelo de regresión lineal múltiple es un caso particular del modelo lineal general dado por la expresión:

$$Y = X\beta + \epsilon, \tag{2.1}$$

donde, Y es el vector de respuestas, la matriz X es la matriz de diseño que contiene las observaciones de las variables explicativas,, β es el vector de parámetros a estimar y ϵ es el vector que contiene los errores.

Para el caso específico de la regresión lineal múltiple, la matriz X, tiene una estructura particular. Si tenemos, las variables explicativas enumeradas de la 1 a la $q-1$ y n observaciones, la matriz de

diseño X se define como:

$$X = \begin{bmatrix} 1 & x_{11} & x_{12} & \dots & x_{1,q-1} \\ 1 & x_{21} & x_{22} & \dots & x_{2,q-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n1} & x_{n2} & \dots & x_{n,q-1} \end{bmatrix}$$

En esta matriz, la primera columna es un vector de unos, que se utiliza para representar el término constante o intercepto en el modelo. Las columnas restantes representan los valores de las $q-1$ variables explicativas para cada una de las n observaciones.

Por lo tanto, el modelo de regresión lineal múltiple se puede expresar como:

$$Y = \beta_0 + \beta_1 X_1 + \cdots + \beta_{q-1} X_{q-1} + \epsilon, \tag{2.2}$$

siendo como se ha indicado, Y, la variable respuesta y $X_1, X_2, \cdots, X_{q-1}$ las variables explicativas, $\beta_0, \beta_1, \cdots \beta_{q-1}$ los q parámetros que acompañan a las variables y ϵ el error.

Para plantear este modelo, al igual que el modelo de regresión lineal simple, se debe asumir una cierta linealidad y que la media del error sea nula. Sin embargo, para hacer inferencia sobre el modelo, será necesario que el error ϵ verifique una serie de hipótesis de homocedasticidad, normalidad e independencia. Dichas hipótesis se pueden expresar como sigue:

$$\epsilon_1, \cdots, \epsilon_n \in N(0, \sigma^2) \text{ y son independientes,}$$

donde la varianza del error σ^2, es un parámetro a estimar.

2.1.2. Hipótesis del modelo

Como hemos indicado, al igual que en el caso de la regresión lineal simple, debemos considerar una serie de hipótesis básicas sobre el modelo.

- Linealidad: es la hipótesis principal de este modelo. Sugiere que la relación entre las variables es un hiperplano, proporcionando un marco lineal en el cual interpretamos los efectos de las variables independientes en la variable dependiente. En el caso del modelo lineal general, donde podemos tener tanto variables continuas como categóricas, la relación también debe ser lineal, pero será en general una colección de hiperplanos, paralelos o no en función de si hay o no interacciones.

- Homocedasticidad. La varianza del error es la misma para cualquier valor de la variable explicativa:
$$Var(\epsilon | X = x) = \sigma^2 I_n \text{ para todo } x.$$

- Normalidad. El error tiene distribución normal:
$$\epsilon \in N_n(0, \sigma^2 I_n),$$
siendo σ^2, la matriz de covarianzas y I_n la matriz identidad de dimensión n.

- Independencia: Dado que estamos considerando una distribución normal multivariante para los errores, la estructura diagonal de la matriz de covarianzas indica que la covarianza entre los errores de diferentes observaciones (ϵ_i y ϵ_j para $i \neq j$) es cero. Esto implica independencia entre los errores, ya que la covarianza cero entre dos variables indica que no hay relación lineal entre ellas.

En definitiva, la hipótesis de linealidad en el modelo de regresión múltiple sugiere que la relación entre las variables es un hiperplano, proporcionando un marco lineal en el cual interpretamos los efectos de las variables independientes en la variable dependiente. Por otro lado, las hipótesis de homocedasticidad y normalidad constituyen simplificaciones muy útiles para poder llevar a cabo las tareas de inferencia. Finalmente, la suposición de independencia de los errores también es conveniente para poder desarrollar inferencia, pero además es razonable suponerla cierta, por ejemplo, en los casos en que la muestra está constituida por experimentos sobre individuos diferentes.

2.1.3. Estimación de los coeficientes: mínimos cuadrados.

Partiendo del planteamiento que hemos hecho del modelo de regresión lineal (2.1), debemos estimar el vector de parámetros β y la varianza del error, σ^2.

Comenzaremos con la estimación de β mediante el método de mínimos cuadrados. De forma que, escogeremos como estimador aquel $\hat{\beta}$ donde se alcance:

$$\min_{\beta} \sum_{i=1}^{n} (Y_i - x_i\beta)^2,$$

donde x_i representa la i-ésima fila de la matriz de diseño X. Análogamente, en notación matricial, el problema de minimización se puede expresar como:

$$\min_{\beta} (Y - X\beta)'(Y - X\beta).$$

En este punto, derivando $(Y - X\beta)'(Y - X\beta)$, respecto de β, e igualando a cero, se obtienen lo que se conocen como ecuaciones normales de regresión:

$$X'X\beta = X'Y.$$

Resolviéndolas, se tiene que el estimador de β por mínimos cuadrado es:

$$\hat{\beta} = (X'X)^{-1}X'Y. \tag{2.3}$$

Para definir correctamente este estimador es necesario que exista la matriz inversa de $X'X$. Una vez obtenida esta estimación, se puede plantear el ajuste del modelo y obtener las predicciones como sigue:

$$\hat{Y} = X\hat{\beta}. \tag{2.4}$$

Es relevante señalar que para la construcción de estos estimadores, solo necesitamos que se verifique la hipótesis de linealidad. Sin embargo, es esencial reconocer que, para realizar inferencias robustas y válidas sobre los parámetros del modelo, se requiere que se cumplan las hipótesis adicionales de normalidad, homocedasticidad e independencia de los errores. Es importante subrayar que, aunque estas hipótesis son ideales y facilitan la interpretación, en la práctica se puede adaptar el enfoque incluso cuando estas condiciones no se cumplen de manera estricta.

2.1.4. Estimación de la varianza

A partir de las expresiones, (2.1) y (2.4), podemos definir los residuos del modelo como la diferencia entre las observaciones y las predicciones, es decir:

$$\hat{\epsilon}_i = Y_i - \hat{Y}_i = Y_i - x_i\hat{\beta},\ i \in 1, \cdots, n.$$

Utilizaremos los residuos definidos, para obtener una estimación de la varianza del error, $\hat{\sigma}^2$:

$$\hat{\sigma}^2 = \frac{1}{n-q}\sum_{i=1}^{n} \hat{\epsilon}_i{}^2 = \frac{1}{n-q}\sum_{i=1}^{n} (Y_i - x_i\hat{\beta})^2 = \frac{\text{RSS}}{n-q},$$

donde q es el rango de la matriz $X'X$ y RSS denota la suma residual de cuadrados.

2.2. Regresión Logística

En esta sección, nos adentraremos en un contexto donde la variable respuesta ya no es de naturaleza continua, como en el caso de la regresión lineal, sino que toma valores categóricos. Por ende, el modelo lineal utilizado previamente resulta limitado para explicar respuestas discretas. En este escenario, surge la necesidad de explorar alternativas que nos permitan modelar eficazmente la relación entre nuestras variables explicativas y la respuesta categórica.

Una herramienta valiosa en este contexto es el modelo de regresión logística, parte de la familia de modelos lineales generalizados (GLM), (Faraway, 2016). Antes de profundizar en la regresión logística, puede resultar interesante introducir brevemente los GLM, que proporcionan un marco teórico unificado para abordar una amplia gama de modelos de regresión. Los modelos lineales generalizados se caracterizan por tener una función de enlace, una distribución de probabilidad y una función de varianza que los define, todas ellas determinadas por la distribución que sigue la variable respuesta.

El modelo de regresión logística se destaca dentro de los GLM, siendo el más sencillo o específico en términos de su función de enlace y distribución de probabilidad. A diferencia del modelo lineal, la variable respuesta en la regresión logística no sigue una distribución normal, sino una distribución de Bernoulli. Esta particularidad es esencial para tratar respuestas binarias o categóricas con dos niveles.

2.2.1. Introducción

En particular, tendremos una variable respuesta binaria, cuya distribución será Bernoulli y su media una probabilidad de éxito. Esto hace que un modelo lineal resulte inadecuado ya que se quebrantan casi todas sus suposiciones básicas.

En concreto, si nos centramos en la linealidad. En este caso, la media de la respuesta no puede expresarse de forma lineal, ya que debe estar en el intervalo $[0, 1]$, es decir, es una probabilidad de éxito p_i. Por otro lado, en el modelo de regresión lineal se suponía que la varianza de las observaciones era constante. Sin embargo, en el caso de la regresión logística, la varianza depende de la probabilidad de éxito, que a su vez era la media condicionada. Como la media condicionada depende de la variable explicativa, también lo hace la varianza. En consecuencia, un modelo de regresión con respuesta binaria es esencialmente heterocedástico. Además, la distribución de probabilidad de la variable respuesta no es normal, sino que es Bernoulli. Por último, la hipótesis de independencia entre los distintos individuos de la muestra debe cumplirse, igual que en el caso del modelo lineal. En cualquier caso, el quebrantamiento de las suposiciones de linealidad, homocedasticidad y normalidad, justifica plenamente la necesidad de considerar otro tipo de modelo.

De esta forma, la regresión logística es el resultado de combinar un modelo de regresión lineal de la forma:

$$\eta = \beta_0 + \beta_1 X_1 + ... + \beta_q X_q, \tag{2.5}$$

con una función enlace g como:

$$\eta_i = g(p_i).$$

Puesto que la relación lineal η_i=p_i no es viable ya que, como indicamos, necesitaremos que $0 \leq p_i \leq 1$ y $\eta_i \in \mathbb{R}$ usaremos una función enlace g. Será necesario que g sea monótona y verifique $0 \leq g^{-1}(\eta) \leq 1$ para cualquier valor η. Aunque podemos pensar en distintas funciones g que cumplan estas propiedades, la elección más popular es la función logística o función *logit* dada por la siguiente expresión:

$$\eta = g(p) = \ln\left(\frac{p}{1-p}\right),$$

o equivalentemente

$$p = \frac{\exp^{\eta}}{1+\exp^{\eta}}. \tag{2.6}$$

La combinación de esta función logit junto con un predictor lineal es lo que recibe el nombre de Regresión Logística.

Recapitulando, recordemos que p en las expresións anteriores hace referencia a la probabilidad de éxito frente a $1-p$ que es la de fracaso. La función logística consiste en aplicar un logaritmo al cociente de ambas probabilidades. El cociente de estas probabilidades indicadas es lo que se conoce como odds:

$$\text{odds} = \frac{p}{1-p},$$

o de forma equivalente:

$$p = \frac{\text{odds}}{1+\text{odds}}.$$

Estas no son más que la probabilidad de éxito entre la de fracaso. Una importante ventaja matemática asociada a ellas, es que pueden tomar cualquier valor real positivo, mientras que la probabilidad de éxito sólo podía tomar valores en el intervalo [0,1]. Dado que obviamente al ser un cociente de cantidades positivas las odds van a ser positivas, podemos aplicarle un logaritmo y transformalas en una cantidad real.

Utilizando el concepto de odds y combinándolo con (2.10), el modelo de regresión logística es:

$$\eta = \ln \text{odds} = \ln\left(\frac{p}{1-p}\right) = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + ... + \beta_q X_q.$$

De aquí podemos interpretar β_1 como sigue: un aumento unitario en X_1 con $X_2, \ldots, X_q$ fijas aumenta los $ln(odds)$ de éxito de β_1 o equivalentemente aumenta las probabilidades de éxito de un factor $\exp^{\beta_1}$. Por lo que en cierto modo la interpretación de los coeficientes exponeciales puede ser más práctica.

2.2.2. Estimación de los coeficientes: método de máxima verosimilitud

A continuación queremos estudiar la estimación de los parámetros de nuestro modelo de regresión logística, para ello, usaremos el método de máxima verosimilitud, (Faraway, 2016).

En particular, para nuestro modelo logístico la variable respuesta sigue una distribución de Bernoulli con parámetro p_i, y su función masa de probabilidad aplicada a una observación y_i, donde $i \in \{1, \ldots, n\}$, es :

$$P(Y = y_i; p_i) = p_i^{y_i} \left(1 - p_i\right)^{1-y_i},$$

y su función de verosimilitud tiene la siguiente expresión:

$$\alpha(\boldsymbol{y}, \boldsymbol{p}) = \prod_{i=1}^{n} p_i^{y_i} (1-p_i)^{1-y_i} = \prod_{i=1}^{n} \left(\frac{p_i}{1-p_i}\right)^{y_i} (1-p_i),$$

donde $\mathbf{y}$ es el vector $\mathbf{y} = (y_1, \ldots, y_n)$ y análogamente $\mathbf{p} = (p_1, \ldots, p_n)$. Aplicando las funcines exponencial y logarítmo y recurriendo a las propiedades de estas, llegamos a:

$$\alpha(\boldsymbol{y}, \boldsymbol{p}) = \left\{\prod_{i=1}^{n} (1-p_i)\right\} \left\{\prod_{i=1}^{n} \exp\left[\ln\left(\frac{p_i}{1-p_i}\right)^{y_i}\right]\right\},$$

Aplicando $\ln\left(\frac{p_i}{1-p_i}\right)^{y_i} = y_i \ln\left(\frac{p_i}{1-p_i}\right) = y_i \eta_i$ e $p_i = \frac{\exp(\eta_i)}{1+\exp(\eta_i)}$ y a continuación remplazando $\eta_i = \ln\left(\frac{p_i}{1-p_i}\right)$ y $1 - p_i = \frac{1}{1+\exp(\eta_i)}$ obtenemos la siguiente expresión simplificada:

$$\alpha(\boldsymbol{\eta}) = \left\{\prod_{i=1}^{n} \frac{1}{1+\exp(\eta_i)}\right\} \exp\left[\sum_{i=1}^{n} y_i \eta_i\right]. \tag{2.7}$$

Aplicando el logaritmo a esta última expresión (2.11) y la propiedad del logaritmo de un cociente se tiene:

$$l(\boldsymbol{\eta}) = \sum_{i=1}^{n} y_i \eta_i - \ln\left(1 + \exp(\eta_i)\right). \tag{2.8}$$

A partir de (2.8) y considerando que $\eta_i = \beta_0 + \beta_1 x_{i1} + \ldots + \beta_q x_{iq}$ o análogamente usando $\eta_i = x_i\beta$, siendo x_i la fila correspondiente de la matriz de diseño X, para la cuál se debe destacar que la primera componente es un uno para poder introducir el intercepto. Tenemos así:

$$l(\beta) = \sum_{i=1}^{n} y_i x_i \beta - \ln(1 + \exp(x_i\beta)). \tag{2.9}$$

Planteamos a continuación las ecuaciones de la verosimilitud, las cuáles se corresponden con la derivada de la log-verosimilitud con respecto a las diferentes componentes del vector de parámetros. De modo que, podremos obtener el parámetro candidato a ser máximo, y por tanto el estimador de β

$$\frac{\partial l(\beta)}{\partial \beta} = \sum_{i=1}^{n} \left[y_i x_i - \frac{\exp(x_i\beta)}{1 + \exp(x_i\beta)} x_i \right] = 0. \tag{2.10}$$

Finalmente para obtener el estimador buscado, debemos resolver las ecuaciónes (2.10) y así obtener una expresión para β. En el caso del modelo lineal, obteníamos las estimaciones de los parámetros a partir del método de mínimos cuadrados, con el cuál llegabamos a un sistema de n ecuaciones y n incógnitas. Sin embargo, en este caso la ausencia de solución explícita, hace que tengamos que recurrir a métodos iterativos.

Para resolver numéricamente las ecuaciones (2.10) es necesario acudir a la matriz hessiana, dada por:

$$\frac{\partial^2 l(\beta)}{\partial \beta \partial \beta'} = -\sum_{i=1}^{n} x_i \left(\frac{\exp(x_i\beta) x_i}{(1 + \exp(x_i\beta))^2} \right). \tag{2.11}$$

A partir de esta expresión (2.11) y aplicando $\eta_i = x_i\beta$, $p_i = \frac{\exp \eta_i}{1+\exp \eta_i}$ e $1 - p_i = \frac{1}{1+\exp \eta_i}$ podemos reescribir la expresión para la matriz hessiana como:

$$\frac{\partial^2 l(\boldsymbol{\beta})}{\partial \boldsymbol{\beta} \partial \boldsymbol{\beta'}} = -\sum_{i=1}^{n} [x_i' x_i p_i (1 - p_i)]. \tag{2.12}$$

En este punto, para resolver las ecuaciones planteadas puede ser de gran utilidad escribir matricialmente la expresión para a matriz hessiana, considerando:

$$X = \begin{bmatrix} 1 & x_{11} & \ldots & x_{1,q} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n1} & \ldots & x_{n,q} \end{bmatrix}, \tag{2.13}$$

donde (2.13) se corresponde con la matriz que contiene los valores de las variables explicativas. Denotando por V La matriz diagonal, con valores $p_i(1 - p_i)$, se tiene:

$$\mathrm{V} = \begin{pmatrix} p_1(1-p_1) & 0 & \ldots & 0 \\ 0 & \ddots & & \vdots \\ \vdots & & \ddots & 0 \\ 0 & \ldots & 0 & p_n(1-p_n) \end{pmatrix},$$

con lo que finalmente la matriz hessiana se puede escribir como sigue:

$$H = \frac{\partial^2 l(\beta)}{\partial \beta \partial \beta'} = -\sum_{i=1}^{n} [x_i' x_i p_i (1 - p_i)] = -X'VX. \tag{2.14}$$

Para resolver las ecuaciones de verosimilitud planteadas, existen diversos métodos iterativos. Uno de los más conocidos es el método de Newton-Raphson. Este método parte de un iterante inicial, $\boldsymbol{\beta^0}$, y obtiene sucesivamente el valor del siguiente parámetro mediante esta expresión (Hastie, Tibshirani, y Friedman, 2009):

$$\boldsymbol{\beta^{k+1}} = \boldsymbol{\beta^k} - \left(\frac{\partial^2 l(\boldsymbol{\beta})}{\partial \boldsymbol{\beta} \partial \boldsymbol{\beta'}} \Big| \boldsymbol{\beta} = \boldsymbol{\beta^k} \right)^{-1} \frac{\partial l(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}} \Big| \boldsymbol{\beta} = \boldsymbol{\beta^k}, \tag{2.15}$$

siendo k el número de iteraciones. Dada la expresión general para el método de Newton-Raphson simplemente queda sustituir en (2.15) para el caso particular de regresión logística. Para ello, debemos recordar la expresión matricial para a matriz hessiana descrita en (2.14) y las relacións dadas en (2.9) y en (2.10) así como la expresión de la matriz de variables explicativas (2.13). Así, podremos expresar las ecuaciones de verosimilitud dadas en (2.10) con notación matricial:

$$\frac{\partial l(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}} = X'(\mathbf{y} - \mathbf{p}) = 0, \tag{2.16}$$

Utilizando estas expresiones, junto co la formulación general del método (2.15) y sustituyendo por las expresiones relativas a la matriz hessiana y a las ecuaciones de verosimilitud ambas en forma matricial, obtendremos:

$$\boldsymbol{\beta^{k+1}} = \boldsymbol{\beta^k} + (X'V^kX)^{-1}X'(\mathbf{y} - \mathbf{p^k}), \tag{2.17}$$

donde V^k es la matriz diagonal V con valores $p_i(1-p_i)$ y $\mathbf{p^k}$ es el vector $\mathbf{p}$ cambiando $\boldsymbol{\beta}$ por el valor de $\boldsymbol{\beta}$ en la iteración anterior, es decir, por $\boldsymbol{\beta^k}$.

A partir de esto, podemos iniciar el proceso iterativo. En primer lugar, debemos partir de un iterante inicial $\boldsymbol{\beta^0}$ y establecer un umbral ϵ . Este valor ϵ nos permitirá saber si nuestro algoritmo converge o no. Principalmente el proceso iterativo a seguir consistirá en el cálculo de $\mathbf{p}$ usando $\mathbf{p} = \frac{\exp(X\boldsymbol{\beta^k})}{1+\exp(X\boldsymbol{\beta^k})}$ y en el cálculo de V cuya diagonal viene dada por $V_{ii} = p_i(1-p_i)$. Estos términos calculados se irán introduciendo progresivamente en la expresión expresión (2.17) para cada iteración. El proceso se repite hasta que $\|\boldsymbol{\beta^k} - \boldsymbol{\beta^{k+1}}\| < \epsilon$, cundo esto pase, si es que sucede, usaremos el valor de β^k obtenido, como estimador del vector de parámetros $\boldsymbol{\beta}$. Si esto último, no ocurre después de una cantidad determinada de iteraciones, podemos decir que no hay convergencia.

2.3. Métodos combinados (*ensemble learning*)

En la década de 1990 empiezan a utilizarse los métodos *ensemble* (métodos combinados), esto es, métodos predictivos que se basan en combinar las predicciones de diversos modelos (Breiman, 1996). Dos de estos métodos son el boosting y el bagging. En el método de boosting, los modelos posteriores otorgan una importancia adicional a los puntos que han sido incorrectamente predichos por los modelos previos, generando así un consenso ponderado para la predicción final. Por otro lado, en el bagging, cada modelo se construye sin depender de los modelos anteriores, utilizando muestras bootstrap del conjunto de datos. La predicción final se determina mediante un voto mayoritario simple entre los diferentes modelos generados. A mayores, Breiman introduce los bosques aleatorios (*random forest*), (Breiman, 2001) como una extensión del método bagging, añadiendo un nivel extra de aleatoriedad. A diferencia de los árboles de decisión estándar, donde cada nodo se divide utilizando la mejor división entre todas las variables, en los bosques aleatorios, cada nodo se divide utilizando la mejor entre un conjunto aleatorio de predictores. Esta estrategia ha demostrado ser altamente efectiva en comparación con otros clasificadores como las máquinas de soporte vectorial o las redes neuronales. Además, en comparación, los bosques aleatorios son más robustos frente al sobreajuste y fáciles de usar, puesto que tan solo presentan dos parámetros para ajustar: el número de variables en el subconjunto aleatorio en cada nodo y el número de árboles en el bosque (Liaw y Wiener, 2002).

2.3.1. Boosting

El boosting es una metodología general de aprendizaje lento en la que se combinan múltiples modelos obtenidos mediante un método con poca capacidad predictiva para mejorar la precisión de un predictor. Inicialmente fue concebido para problemas de clasificación y tardó varios años en extenderse a problemas de regresión.

La idea del boosting surge de la mano de Valiant (1984) y Kearns y Valiant (1994) (Fernández-Casal, Costa Bouzas, y Oviedo de la Fuente, 2021), quienes se plantearon la pregunta de si un algoritmo de aprendizaje débil, con un rendimiento ligeramente mejor que el azar, puede ser mejorado para convertirse en un algoritmo de aprendizaje fuerte y preciso (Schapire, 1999). Esto resultó ser una díficil tarea que hasta que en 1996, Freund y Schapire presentan el algoritmo AdaBoost, (Fernández-Casal y cols., 2021). De forma esquemática, podremos describir este algoritmo como sigue, (Schapire, 2002): En definitiva, el algoritmo AdaBoost inicia asignando un peso inicial a cada ejemplo de entrenamiento y luego se elige el número de iteraciones, determinando cuántos clasificadores débiles se entrenarán y combinarán. En cada iteración, se construye un clasificador débil, h_t, y se le asigna un peso α_t basado en su tasa de error en la clasificación. Esto permite ajustar la contribución de cada clasificador a la combinación final. Luego, se actualizan los pesos de los ejemplos de entrenamiento para dar más importancia a los ejemplos mal clasificados en la iteración anterior, permitiendo que los clasificadores siguientes se centren más en corregir esos errores. Finalmente, una vez completadas todas las iteraciones, se combina la salida de todos los clasificadores débiles ponderados para obtener un clasificador final. Este clasificador final utiliza una suma ponderada de las predicciones de los clasificadores débiles, utilizando la función de signo para decidir la etiqueta de clase final. Este enfoque iterativo y adaptativo del algoritmo AdaBoost permite la construcción de un clasificador robusto y preciso a partir de clasificadores débiles individuales.

Por otro lado, otro avance significativo en este campo fue la introducción del método conocido como *gradient boosting machine* por Friedman en el año 2001, (Breiman, 2001). Este método, que se encuentra dentro de la categoría de métodos iterativos, marcó un hito importante en el aprendizaje automático. Una de las características destacadas de este enfoque es su capacidad para abordar una amplia gama de problemas, incluyendo tanto la clasificación como la regresión, lo que amplió significativamente su aplicabilidad y utilidad en diversos contextos de análisis de datos. A continuación exponemos una sencillo esquema de este algoritmo, (Friedman, 2001) :

Algoritmo 1 Algoritmo *Gradient Boosting Machine*

Entrada:

- Un conjunto de entrenamiento: $\{(x_i, y_i)\}_{i=1}^n$,
- Una función de pérdida diferenciable: $L(y, \hat{y})$ (por ejemplo RSS),
- Un algoritmo de aprendizaje base,
- Un cierto número de iteraciones M.

Inicialización:

Inicializamos el modelo con un valor constante: $F_0 = \arg\min_\gamma \sum_{i=1}^n L(y_i, \gamma)$.

Para cada iteración $m = 1, \cdots, M$

- Calculamos los residuos o gradientes de la función de pérdida: $\tilde{y}_{im} = -\frac{\partial L(y_i, F_{m-1}(x_i))}{\partial F_{m-1}(x_i)}$,

para $i = 1, \cdots, n$.

- Introducimos un parámetro de aprendizaje $h_m(x)$ al conjunto de entrenamiento $\{(x, \tilde{y}_{im})\}_{i=1}^n$.
- Actualizamos el modelo como: $F_m(x) = F_{m-1}(x) + h_m(x)$.

Salida:

Obtenemos la salida final: $F_M(x)$.

Resumiendo lo anterior, a partir de un conjunto de entrenamiento $\{(x_i, y_i)\}_{i=1}^n$, una función de pérdida diferenciable (como la suma de los errores cuadráticos, RSS), un algoritmo de aprendizaje base y un número determinado de iteraciones M, se inicia el modelo con un valor constante F_0 que minimiza la función de pérdida. Para cada iteración, se calculan los residuos y se introduce un parámetro de

aprendizaje $h_m(x)$ al conjunto de entrenamiento. Tal parámetro es un factor que controla la magnitud de las actualizaciones en los pesos del modelo durante el entrenamiento, determinando así la velocidad y estabilidad con la que el modelo converge hacia una solución óptima. Luego, se actualiza el modelo $F_m(x)$ agregando el parámetro de aprendizaje. Finalmente, se obtiene la salida final $F_M(x)$. Este procedimiento iterativo busca mejorar continuamente el modelo, ajustándolo a los residuos y obteniendo así una salida final que optimiza la función de pérdida en el conjunto de entrenamiento.

En conclusión, el boosting ha demostrado ser una metodología poderosa en el campo del aprendizaje automático, permitiendo mejorar la precisión de los predictores combinando múltiples modelos débiles de manera inteligente. Desde sus primeras conceptualizaciones hasta los desarrollos más recientes como el algoritmo AdaBoost y el método de gradient boosting machine, este enfoque ha sido fundamental en la creación de clasificadores robustos y precisos, ampliando su aplicabilidad a una variedad de problemas en clasificación y regresión. Con su capacidad para adaptarse y mejorar iterativamente, el boosting continúa siendo un área de investigación activa y prometedora en la búsqueda de mejores técnicas de aprendizaje automático.

2.3.2. Bagging

Como indicamos, durante la década de 1990, surgen los métodos de ensamblado, los cuales se basan en combinar las predicciones de múltiples modelos predictivos. Uno de los primeros métodos *ensemble* utilizados fue el bagging (*bootstrap aggregation*), propuesto por Breiman en 1996 (Breiman, 1996). Este método, diseñado para reducir la varianza, combina el bootstrap con un modelo de regresión o clasificación. Su idea principal consiste en si se tienen múltiples conjuntos de entrenamiento, cada uno puede usarse para entrenar un modelo individual para hacer predicciones. Luego, al promediar estas predicciones, se simplifica la solución y se reduce significativamente la varianza. Sin embargo, en la práctica, a menudo solo se dispone de un único conjunto de entrenamiento. Aquí es donde el bootstrap cobra importancia, ya que, si bien es una técnica útil para estimar varianzas, en este contexto, se emplea para mitigar el sesgo y reducir la varianza. Se generan cientos o miles de muestras bootstrap a partir del conjunto de entrenamiento original, y cada una de estas muestras se utiliza como un nuevo conjunto de entrenamiento (*bootstrapped training data set*).

El desarrollo de este algoritmo se basa en tres pasos principales que enumeramos a continuación (Breiman, 1996):

1. Bootstrap: El bagging utiliza el bootstrap para crear muestras diversas del conjunto de datos de entrenamiento. Este método de remuestreo genera subconjuntos del conjunto de datos seleccionando puntos de datos al azar y con reemplazamiento. Esto significa que cada vez que se elige un punto de datos del conjunto de entrenamiento, puede seleccionarse más de una vez, lo que resulta en la repetición de ciertos valores o instancias en una muestra.

2. Entrenamiento paralelo: Una vez que se han generado las muestras de bootstrap, estas se utilizan para entrenar modelos de forma independiente y en paralelo entre sí. Cada muestra bootstrap se utiliza como un nuevo conjunto de entrenamiento para entrenar un modelo débil o básico, como por ejemplo un árbol de decisión.

3. Agregación: Finalmente, en función del tipo de tarea (regresión o clasificación), se promedian o se toma la mayoría de las predicciones de los modelos individuales para calcular una estimación más precisa. En el caso de la regresión, se promedian todas las salidas predichas por los modelos, lo que se conoce como votación suave. Para problemas de clasificación, se acepta la clase con la mayoría de votos, lo que se conoce como votación en firme o votación por mayoría. Este proceso de agregación ayuda a reducir la varianza y mejorar la precisión del modelo final.

La explicación anterior proporciona una descripción general del algoritmo. Ahora, para abordar el algoritmo con mayor rigor, podemos describirlo en detalle utilizando un enfoque más técnico, (Breiman, 1996):

Algoritmo 2 Algoritmo de Bagging para Regresión con votación suave

Entrada:

• Conjunto de entrenamiento: $\{(x_i, y_i)\}_{i=1}^{n}$

• Función de pérdida diferenciable: $L(y, \hat{y})$ (por ejemplo, RSS)

• Algoritmo de aprendizaje base

• Número de iteraciones: M

Inicialización:

Inicializamos el modelo con un valor constante: $F_0(x) = \arg\min_{\gamma} \sum_{i=1}^{n} L(y_i, \gamma)$.

Para cada iteración $m = 1, \cdots, M$

• Generamos una muestra bootstrap D_b del conjunto de entrenamiento

• Entrenamos un modelo débil f_b utilizando la muestra bootstrap D_b

• Actualizamos el modelo: $F_m(x) = F_{m-1}(x) + f_b(x)$

• b es el índice de la muestra bootstrap

Salida:

Obtenemos la salida final: $F_M(x)$.

Cabe destacar, que el uso de una muestra bootstrap dota al método bagging de una ventaja adicional al permitir estimar el error de predicción directamente, sin necesidad de utilizar la muestra test o métodos como la validación cruzada. La cantidad de muestras bootstrap tomadas, es una decisión bastante importante. Para ello se realiza una aproximación Monte Carlo y se estudia la convergencia del error OOB (*out-of-bag*). Este error surge cuando se utiliza bootstrap para crear múltiples conjuntos de entrenamiento, algunos datos pueden aparecer repetidos en varias muestras de entrenamiento, mientras que otros pueden no aparecer en ninguna muestra. Los datos que no están presentes en una muestra bootstrap particular se denominan datos *out-of-bag* (fuera de la bolsa). El error *out-of-bag* se calcula utilizando estos datos *out-of-bag*. Para cada observación en el conjunto de datos original, se calcula la predicción utilizando solo los modelos que no utilizaron esa observación en su conjunto de entrenamiento (es decir, los modelos en los que esa observación es *out-of-bag*). Luego, se calcula el error de predicción promediando estos errores individuales para todas las observaciones. Este error promedio se denomina error *out-of-bag*. Es decir, para cada observación en el conjunto de datos de entrenamiento original, se determina en cuántos modelos (árboles, en el caso de Bagging con árboles de decisión) esa observación no fue incluida en la muestra bootstrap correspondiente. Luego, se utiliza la predicción de estos modelos OOB para calcular el error de predicción para cada observación. De forma que el error OOB proporciona una estimación del rendimiento del modelo en datos no vistos, lo que lo hace útil para evaluar la capacidad de generalización del modelo sin necesidad de un conjunto de datos de prueba separado. De esta forma, si hay convergencia con unos pocos cientos de modelos, no se espera una mejora significativa al aumentar el número de estos. Además, aumentar demasiado el número de modelos no aumenta el riesgo de sobreajuste, pero sí aumenta los costos computacionales. Una ventaja adicional de Bagging es que, aunque mejora la precisión de las predicciones, se pierde interpretabilidad.

En resumen este método realiza remuestreo repetido del conjunto de datos de entrenamiento y entrena un modelo con cada conjunto de datos remuestreado. Con ello, se obtienen las predicciones finales promediando las predicciones de todos los modelos (o mediante la decisión mayoritaria en el caso de la clasificación). Finalmente, la precisión de estas predicciones se puede estimar utilizando el error OOB (*out-of-bag*).

2.3.3. Random forests

Los árboles de decisión son una herramienta simple y fácilmente interpretable en el campo del aprendizaje automático. Se desarrollaron en los años 70 como una alternativa a los métodos estadísticos tradicionales, (Fernández-Casal y cols., 2021). La idea detrás de un árbol de decisión es dividir el espacio predictor en regiones simples representadas por un árbol binario. Cada nodo del árbol representa una pregunta sobre una característica del conjunto de datos, y las ramas del árbol conducen a nodos hijos con respuestas binarias a esas preguntas. Estos nodos terminales, o hojas, se utilizan

para realizar predicciones. Los árboles de decisión son válidos tanto para problemas de regresión como de clasificación, y pueden manejar tanto variables numéricas como categóricas. Sin embargo, la simplicidad del modelo puede limitar su capacidad para describir relaciones complejas entre las variables predictoras y la variable objetivo. Los bosques aleatorios (*random forest*) son una variante del bagging para trabajar con dichos árboles de decisión. Surgen de la necesidad de añadir aleatoriedad al proceso de construcción de los árboles, con el objetivo de evitar la correlación entre ellos. Tras varios intentos de introducir esta aleatoriedad Breiman (Breiman, 2001), propuso un algoritmo llamado bosques aleatorios. Este algoritmo, tanto para el caso de regresión como de clasificación se puede explicar como sigue, (Liaw y Wiener, 2002):

Algoritmo 3 Algoritmo Random Forest

Entrada:
- Un conjunto de entrenamiento: $\{(x_i, y_i)\}_{i=1}^n$,
- Tamaño de la muestra: n
- Número de características a considerar en cada división: m,
- Tipo de problema: clasificación o regresión.

Inicialización:
Inicializamos el modelo con un valor constante: $F_0(x) = \arg\min_\gamma \sum_{i=1}^n L(y_i, \gamma)$.

Para cada árbol $b = 1, \cdots, n$
- Seleccionar n muestras bootstrap del conjunto de datos original.
- Para cada una de las muestras bootstrap:
- Seleccionar aleatoriamente m características de las p disponibles.
- Construir un árbol de clasificación o regresión sin podar usando las características seleccionadas.

Salida:
Bosque de árboles de decisión $\{T_b\}_{b=1}^n$.

En definitiva, el proceso anterior se basa en lo que sigue:

1. Seleccionar n muestras bootstrap del conjunto de datos original.
2. Para cada una de las muestras bootstrap, construir un árbol de clasificación o regresión sin podar, teniendo en cuenta la siguiente modificación: en cada nodo, en lugar de elegir la mejor división entre todos los predictores, se seleccionan de forma aleatoria m de los q predictores y se elige la mejor división entre esas variables.
3. Predecir nuevos datos agregando las predicciones de los n árboles es decir, votos mayoritarios para clasificación, promedio para regresión.

Tras aplicar el algoritmo, se podrá medir la importancia de las variables mediante una estimación de la tasa de error, basada en los datos de entrenamiento. El proceso a seguir consiste en que en cada iteración de bootstrap, se predicen los datos que no están en la muestra bootstrap utilizando el árbol construido con la muestra bootstrap. Posteriormente, se agregan estas predicciones o predicciones OOB para calcular la tasa de error denominada estimación OOB de la tasa de error.

Volviendo al algoritmo, el hiperparámetro a seleccionar es m, este se podrá seleccionar mediante diversas técnicas habituales como por ejemplo validación cruzada. Sin embargo, como puntos de partida razonables se suelen considerar $m = \sqrt{q}$ para problemas de clasificación y $m = q/3$ para problemas de regresión, (Fernández-Casal y cols., 2021). Por otro lado, también se puede tratar como un hiperparámetro el número de árboles a construir, aunque esto se suele tratar más bien como un problema de convergencia (Fernández-Casal y cols., 2021).

En resumen, los bosques aleatorios son un método computacionalmente más eficientes que bagging puesto que, aunque requieren más árboles, la construcción de cada árbol es mucho más rápida al evaluarse sólo unos pocos predictores en cada corte. Además, los bosques aleatorios no solo ofrecen un rendimiento computacional superior al bagging, sino que también destacan por su versatilidad y

robustez en la práctica del aprendizaje automático. Su capacidad para manejar grandes conjuntos de datos, así como su eficacia en la reducción del sobreajuste, los convierte en una herramienta muy relevante para la modelización predictiva. Además, cabe destacar su flexibilidad para adaptarse a una amplia variedad de problemas tanto de clasificación como de regresión, su facilidad de uso y su rendimiento confiable en una amplia variedad de aplicaciones del mundo real. Lo que los hace una opción muy sólida en muchos casos.

2.4. Forecasting

El forecasting es una actividad fundamental que nos permite calcular o predecir eventos o condiciones futuras mediante el análisis de datos relevantes de manera racional (Makridakis, 1996). Existen dos enfoques principales para realizar forecasting, cualitativo y cuantitativo. Los métodos cualitativos son más intuitivos y se utilizan en casos en los que no hay datos disponibles, o si los datos disponibles no son relevantes para los pronósticos. Mientras que en los métodos cuantitativos, se dispone de información numérica sobre el pasado y se utilizan modelos matemáticos o estadísticos para generar predicciones a partir de dichos datos pasados. Los métodos cuantitativos tienen la ventaja de estar respaldados por teoría matemática y estadística (Hyndman y Athanasopoulos, 2018).

Dentro de los métodos cuantitativos, uno de los más ampliamente utilizados es el modelo ARIMA (*Autoregressive Integrated Moving Average*). Este modelo es especialmente útil para el análisis de series temporales, que son conjuntos de observaciones registradas en intervalos de tiempo predefinidos, como horas, días, meses o años. El modelo ARIMA, junto con otras técnicas, forma parte de un conjunto diverso de métodos disponibles para el forecasting.

En definitiva, a lo largo de esta sección, exploraremos en detalle los métodos y técnicas utilizadas en el forecasting, centrándonos especialmente en el modelo ARIMA y su aplicación práctica en el análisis de series temporales. [1]

2.4.1. Conceptos de Forecasting

En esta sección, exploraremos los conceptos básicos del forecasting, centrándonos en la comprensión de las series temporales y los elementos fundamentales que influyen en ellas. Una serie de tiempo es una colección de observaciones de una variable, X, tomadas secuencialmente a lo largo del tiempo (Cryer y Chan, 2008). Es decir, tendremos una colección de variables aleatorias, $\{X_t\}_t$, con media mu_t y varianza σ_t^2. Al observar la representación gráfica secuencial de una serie de tiempo, podemos identificar características clave, como la posible presencia de tendencia, estacionalidad y heterocedasticidad. En particular, la tendencia se relaciona con un cierto comportamiento a largo plazo de la serie, la estacionalidad con un comportamiento periódico o con un patrón repetitivo y la presencia de heterocedasticidad hace referencia a que la variabilidad de la serie no es constante (Cryer y Chan, 2008).

Visto esto, antes de presentar los modelos ARIMA, primero debemos discutir el concepto de estacionariedad y la técnica de diferenciación de series temporales. Una serie temporal estacionaria es aquella cuyas propiedades no dependen del tiempo en el que se observa la serie, es decir es aquella $\{X_t\}_t$ que verifica (Cryer y Chan, 2008):

$$\begin{aligned} \mu_t &= \mu \ \forall t. \\ \sigma_t^2 &= \sigma^2 \ \forall t. \\ \gamma(t, t+k) &= \gamma_k, \ \forall t, k. \end{aligned}$$

[1] Utilizaremos notación muy similar a los apuntes de la asignatura Series de tiempo impartida en el Departamento de Matemáticas, Universidad da Coruña, Máster Universitario en Técnicas Estadísticas, Curso 2022-23 (Aneiros Pérez, 2022).

Es decir, será aquél proceso con media y varianzas constantes a lo largo del tiempo. Por lo tanto, las series de tiempo con tendencias o con estacionalidad no son estacionarias (Hyndman y Athanasopoulos, 2018). En general, una serie temporal estacionaria no tendrá patrones predecibles a largo plazo y los gráficos de tiempo asociados mostrarán que la serie es aproximadamente horizontal. De este modo, existen transformaciones como aplicar logaritmos que nos pueden ayudar a estabilizar la varianza de una serie de tiempo. Por otro lado, la diferenciación puede ayudar a estabilizar la media eliminando cambios en el nivel de una serie temporal y, por lo tanto, eliminando o reduciendo la tendencia y la estacionalidad.

En cuanto a la diferenciación, podemos distinguir entre diferenciación regular y estacional. La diferenciación regular implica calcular las diferencias entre observaciones consecutivas de una serie temporal. Es decir, debemos trabajar con los siguientes datos (Svetunkov, 2023):

$$\Delta x_t = x_t - x_{t-1} = (1 - B)x_t,$$

donde $(1 - B)$ es un operador de diferenciación tal que $Bx_t = x_{t-1}$. Por ejemplo, al calcular las diferencias entre los valores diarios de una serie de tiempo, estamos realizando una diferenciación regular para eliminar la tendencia y la estacionalidad. Esta técnica ayuda a estabilizar la media de la serie temporal al eliminar los cambios en el nivel de la misma.

Por otro lado, la diferenciación estacional implica tomar la diferencia entre una observación y la observación anterior de la misma temporada. Es decir, tendremos (Hyndman y Athanasopoulos, 2018):

$$\Delta x_t = x_t - x_{t-s}.$$

Por ejemplo, al calcular las diferencias entre los valores de una serie temporal en el mismo mes de diferentes años, estamos realizando una diferenciación estacional para eliminar la estacionalidad. Esta técnica es útil para eliminar los efectos de la estacionalidad y hacer que la serie temporal sea más estacionaria. Estas técnicas de diferenciación son fundamentales para el análisis de series temporales y para la aplicación de los modelos conocidos como modelos de Box-Jenkins que introduciremos en las siguientes secciones.

Además de comprender los conceptos de estacionariedad y las técnicas de diferenciación, es fundamental explorar las características de los procesos estocásticos subyacentes en el análisis de series temporales. Dos aspectos clave en este sentido son la causalidad y la invertibilidad de dichos procesos.

Un proceso causal, también conocido como MA(∞), es aquel que puede representarse como una combinación lineal de los términos de error pasados, con un número finito de coeficientes no nulos. Es decir, aquél que admite la siguiente representación (Peña, 2005):

$$X_t = c + \psi_0 a_t + \psi_1 a_{t-1} + \psi_2 a_{t-2} + \dots \quad \text{con} \sum_{i=0}^{\infty} |\psi_i| < \infty.$$

Por otro lado, un proceso invertible, o AR(∞), se caracteriza por una representación que incluye términos autorregresivos pasados, nuevamente con un número finito de coeficientes no nulos. Es decir, áquel que admite una expresión:

$$X_t = c + a_t + \pi_1 X_{t-1} + \pi_2 X_{t-2} + \dots \quad \text{con} \sum_{i=1}^{\infty} |\pi_i| < \infty.$$

Estas propiedades, junto con la comprensión de la estacionariedad y las técnicas de diferenciación, forman la base para el análisis y modelado avanzado de series temporales. Ahora exploraremos más a fondo los modelos disponibles, así cómo sus características y cómo estas influyen en la predicción y el pronóstico de datos temporales.

2.4.2. Modelos de Box-Jenkins

Para abordar los modelos de Box-Jenkins, comenzaremos estableciendo una notación general (Aneiros Pérez, 2022) que mantendremos a lo largo de estas secciones:

- $x_1, x_2, \ldots, x_T$: serie de tiempo observada.
- $\{X_t\}_t$: proceso generador de la serie de tiempo.
- $\{a_t\}_t$: proceso de ruido blanco (innovaciones). El ruido blanco es un proceso estacionario en el que:
  - $\mu = 0$.
  - $\sigma^2 = \sigma_a^2$.
  - $\gamma_k = \begin{cases} \sigma_a^2, & \text{si } k = 0 \\ 0, & \text{si } k \neq 0 \end{cases}$.

Además, asumiremos que, (Aneiros Pérez, 2022):

$$a_t \text{ es independiente de } X_{t-1}, X_{t-2}, \ldots$$

Esta suposición implica que el conocimiento de los valores pasados de la serie de tiempo no proporciona información sobre el ruido blanco en el instante actual a_t. Establecer esta independencia es crucial para el desarrollo y la interpretación de los Modelos de Box-Jenkins. Ahora, dada esta notación y suposición previa, podemos analizar en profundidad los conceptos y aplicaciones de los Modelos de Box-Jenkins en el análisis y pronóstico de series temporales.

Proceso autorregresivo de orden p

El proceso autorregresivo de orden p (AR(p)) es un concepto fundamental en el análisis de series temporales. Este modelo describe cómo el valor actual de una serie temporal depende linealmente de sus p valores anteriores, más el término a_t o ruido blanco. Formalmente, este proceso estacionario $\{X_t\}_t$ se representa como (Hyndman y Athanasopoulos, 2018):

$$X_t = c + \phi_1 X_{t-1} + \phi_2 X_{t-2} + \cdots + \phi_p X_{t-p} + a_t,$$

donde c es una constante, $\phi_1, \ldots, \phi_p$ son los coeficientes autorregresivos($\phi_p \neq 0$) y a_t es ruido blanco.

La representación de un proceso AR(p) tiene algunas propiedades importantes que cabe destacar (Peña, 2005). En primer lugar, es estacionario si y solo si el polinomio $1 - \phi_1 z - \phi_2 z^2 - \cdots - \phi_p z^p$ no tiene raíces en el círculo unitario $|z| = 1$. En cuanto a la causalidad, el proceso es causal únicamente si las raíces de dicho polinomio se encuentran fuera del círculo unitario $|z| \leq 1$. Además, este tipo de proceso es siempre invertible, lo que implica que puede ser descrito como un proceso de media móvil (MA). Finalmente, la función de autocorrelación parcial del proceso AR(p) se anula para todos los retardos mayores que p, pero no para el retardo p. En definitiva, el proceso autorregresivo de orden p es una herramienta poderosa para modelar y predecir series temporales, especialmente cuando se cumplen las condiciones de estacionariedad y causalidad.

Proceso de medias móviles de orden q

El proceso de medias móviles de orden q (MA(q)) es otro componente esencial en el análisis de series temporales. Este modelo, en lugar de utilizar valores pasados de la variable de pronóstico, utiliza errores de pronóstico pasados en un modelo similar a una regresión. Formalmente, dicho proceso estacionario $\{X_t\}_t$ se representa como (Hyndman y Athanasopoulos, 2018):

$$X_t = c + a_t + \theta_1 a_{t-1} + \theta_2 a_{t-2} + \cdots + \theta_q a_{t-q},$$

donde c es una constante, $\theta_1, \ldots, \theta_q$ son los coeficientes de las medias móviles($\theta_q \neq 0$) y a_t es ruido blanco.

En relación a sus propiedades podemos destacar (Peña, 2005). En primer lugar, el proceso es siempre estacionario y causal. Además, es invertible si y solo si el polinomio $1 + \theta_1 z + \theta_2 z^2 + \cdots + \theta_q z^q$ no presenta raíces fuera del círculo unitario $|z| \leq 1$. Por último, la función de autocorrelación simple del proceso se anula para todos los retardos mayores que q, pero no para el retardo q.

El proceso de medias móviles de orden q proporciona otra forma efectiva de modelar y predecir series temporales, especialmente en conjunción con otros modelos como los procesos autorregresivos.

Procesos ARMA(p,q)

La combinación en un mismo proceso de una estructura autorregresiva (AR) y de medias móviles (MA) da lugar a los procesos conocidos como ARMA. Un ARMA, es un proceso estacionario $\{X_t\}_t$ que admite la siguiente representación (Peña, 2005):

$$X_t = c + \phi_1 X_{t-1} + \phi_2 X_{t-2} + \cdots + \phi_p X_{t-p} + a_t + \theta_1 a_{t-1} + \theta_2 a_{t-2} + \cdots + \theta_q a_{t-q}, \tag{2.18}$$

donde c,$\phi_1, \ldots, \phi_p$, $\theta_1, \ldots, \theta_q$ ($\phi_p, \theta_q \neq 0$) son constantes. Dicha ecuación (2.18) también se puede escribir de forma compacta como sigue:

$$\phi(B) X_t = c + \theta(B) a_t,$$

donde

$$\phi(B) = (1 - \phi_1 - \phi_2 B^2 - \cdots - \phi_p B^p),$$
$$\theta(B) = (1 + \theta_1 B + \theta_2 B^2 + \cdots + \theta_q B^q)$$

y B denota al operador retardo, definido por $BX_t = X_{t-1}$.

En cuánto a las características asociadas a este proceso debemos mencionar:

- Para que la representación anterior dé lugar a un proceso estacionario (equivalentemente, a un ARMA(p, q)) es necesario y suficiente que $1 - \phi_1 z - \phi_2 z^2 - \cdots - \phi_p z^p \neq 0$ para todo z con $|z| = 1$.
- Es causal si y solo si $1 - \phi_1 z - \phi_2 z^2 - \cdots - \phi_p z^p \neq 0$ para todo z con $|z| \leq 1$.
- Es invertible si y solo si $1 + \theta_1 z + \theta_2 z^2 + \cdots + \theta_q z^q \neq 0$ para todo z con $|z| \leq 1$.

Además, la clase de procesos ARMA(p,q) es una familia de procesos estacionarios muy flexible y puede modelizar una gran variedad de series generadas por procesos estacionarios.

Estimación de un proceso ARMA(p,q)

Una etapa relevante en el análisis de series temporales es la estimación de los parámetros en los modelos ARMA(p,q), donde P y q representan los órdenes del componente autorregresivo y de media móvil, respectivamente. Por tanto, dada una serie de tiempo $x_1, \ldots, x_T$ generada por un proceso ARMA(p,q) causal, invertible y gaussiano, con parámetros $c, \phi_1, \ldots, \phi_p, \theta_1, \ldots, \theta_q$ y varianza del término de error σ_a^2. El objetivo es estimar estos parámetros utilizando los datos disponibles de la serie de tiempo. Para ello, existen diferentes métodos de estmación de parámetros, como la estimación por mínimos cuadradps, mínimos cuadrados condicionados o máxima verosimilitud (Peña, 2005).

- Estimación por mínimos cuadrados: Este método, busca minimizar la suma de los cuadrados de los residuos entre los valores observados y los valores ajustados por el modelo ARMA. Es decir, La estimación de los parámetros $(c, \phi_1, \ldots, \phi_p, \theta_1, \ldots, \theta_q)$ por medio del método de mínimos

cuadrados se obtiene a través de los valores $(\hat{c}, \hat{\phi}_1, \ldots, \hat{\phi}_p, \hat{\theta}_1, \ldots, \hat{\theta}_q)$ que minimizan a la función dada como sigue:

$$\sum_{t=1}^{T} \hat{a}_t^2,$$

siendo $\hat{a}_t = x_t - (c + \widetilde{\phi}_1 x_{t-1} + \ldots + \widetilde{\phi}_p x_{t-p} + \widetilde{\theta}_1 a_{t-1} + \ldots + \widetilde{\theta}_q a_{t-q})$.

- Estimación por mínimos cuadrados condicionados: Cuando el orden p del componente autorregresivo es mayor que cero, surge una dificultad en la estimación de los primeros residuos $\hat{a}_1, \hat{a}_2, \ldots, \hat{a}_p$ debido a la dependencia de los valores no observados. El método de mínimos cuadrados condicionados aborda esta limitación minimizando la función de pérdida condicionada a partir del momento en que los primeros p residuos son conocidos. Es decir, minimizando esta función:

$$\sum_{t=p+1}^{T} \hat{a}_t^2,$$

en lugar de la función dada para el caso anterior.

- Estimación de máxima verosimilitud: Dicho método, busca encontrar los valores de los parámetros que maximizan la probabilidad de observar los datos en la serie temporal. Por tanto, en base a este enfoque, la estimación de los parámetros se obtiene a partir de aquellos valores $(\hat{c}, \hat{\phi}_1, \ldots, \hat{\phi}_p, \hat{\theta}_1, \ldots, \hat{\theta}_q)$, $\hat{\sigma^2}_a$ que máximizan la función de verosimilitud. Es decir:

$$(\hat{c}, \hat{\phi}_1, \ldots, \hat{\phi}_p, \hat{\theta}_1, \ldots, \hat{\theta}_q, \hat{\sigma^2}_a) = \arg \max_{\widetilde{c}, \widetilde{\phi}_1, \ldots, \widetilde{\phi}_p, \widetilde{\theta}_1, \ldots, \widetilde{\theta}_q, \widetilde{\sigma^2}_a} L(\widetilde{c}, \widetilde{\phi}_1, \ldots, \widetilde{\phi}_p, \widetilde{\theta}_1, \ldots, \widetilde{\theta}_q, \widetilde{\sigma^2}_a).$$

La elección del método de estimación depende de las características específicas de los datos y de las suposiciones sobre el proceso subyacente. Cada enfoque tiene sus ventajas y limitaciones, y es importante considerar cuidadosamente el contexto y los objetivos del análisis al seleccionar el método más apropiado.

Procesos ARIMA(p,d,q)

En el apartado anterior, hemos explorado los modelos ARMA, una familia flexible de procesos estacionarios que pueden capturar una amplia gama de comportamientos en series temporales. Sin embargo, en la realidad, nos encontramos con que muchas series temporales exhiben tendencias o patrones repetitivos que no son inherentes a los procesos estacionarios. Debido a esto, necesitaremos una alternativa para abordar estas situaciones, y es aquí donde entra en juego el concepto de los procesos ARIMA (p, d, q). La idea detrás de los procesos ARIMA radica en que mediante la aplicación de diferenciación regular a una serie temporal, podemos transformarla en un proceso estacionario, sobre el cual podemos aplicar los modelos ARMA.

En esencia, un proceso ARIMA(p, d, q) es aquel que, después de aplicarle d diferencias regulares, se convierte en un proceso ARMA(p, q). Es decir (Hyndman y Athanasopoulos, 2018):

$$\{X_t\}_t \text{ es ARIMA(p,d,q)} \Leftrightarrow (1-B)^d X_t \text{ es ARMA(p,q)}.$$

Equivalentemente $\{X_t\}_t$ es un proceso ARIMA(p,d,q) si admite una representación del tipo (Peña, 2005):

$$\phi(B)(1-B)^d X_t = c + \theta(B) a_t,$$

donde el polinomio $\phi(z)$ no tiene raíces de módulo 1. Esto refleja que un proceso ARIMA permite modelar series temporales que exhiben tendencia mediante la combinación de componentes autorregresivos y de medias móviles.

A continuación, nos centraremos en los detalles de cómo identificar y modelar adecuadamente estos procesos ARIMA. Exploraremos los métodos para determinar los parámetros p, d y q óptimos para un modelo ARIMA específico. Para identificar dichos parámetros óptimos, es común utilizar métodos como la función de autocorrelación (ACF) y la función de autocorrelación parcial (PACF) para determinar el orden de autorregresión p y el orden de medias móviles q, respectivamente. Además, la diferenciación de la serie temporal d puede ser ajustada iterativamente para lograr estacionariedad.

Una vez identificado el modelo ARIMA adecuado, es crucial verificar su validez y evaluar su capacidad predictiva utilizando técnicas como la validación cruzada y la comparación de errores de pronóstico o métricas disponibles. Además, la interpretación de los resultados del modelo puede proporcionar información valiosa sobre las relaciones dinámicas en los datos y guiar la toma de decisiones en función de las predicciones realizadas.

Estimación de un modelo ARIMA(p,d,q)

La relación estrecha entre los modelos ARIMA(p,d,q) y los modelos ARMA(p,q) simplifica significativamente el proceso de estimación de un modelo ARIMA. Para dicho proceso, es crucial identificar el orden de diferenciación regular d. Este paso implica identificar cuántas veces se debe diferenciar la serie temporal para convertirla en estacionaria. Una vez que la serie temporal ha sido diferenciada d veces y se ha vuelto estacionaria, se procede a estimar un modelo ARMA(p,q).

Estos dos pasos en conjunto, ofrecen una metodología efectiva para la estimación de modelos ARIMA(p,d,q), lo que permite un análisis detallado de las series temporales y la generación de pronósticos precisos.

2.5. Sistemas de recomendación basados en Factorización de Matrices

En la actualidad, las consumidoras y consumidores se enfrentan a una abrumadora cantidad de opciones, ya que los proveedores de contenido ofrecen una amplia selección de productos. La clave para mejorar la satisfacción del usuario radica en la capacidad de emparejar a los consumidores con los productos más adecuados. Por esta razón, cobran especial relevancia los sistemas de recomendación, los cuales analizan patrones de interés de los usuarios para ofrecer recomendaciones personalizadas que se ajusten a sus gustos. Estos sistemas de recomendación son especialmente útiles para productos de entretenimiento, como películas, música y programas de televisión. Los usuarios suelen estar dispuestos a indicar su nivel de satisfacción con productos específicos, generando un gran volumen de datos sobre las preferencias de los clientes. Las empresas pueden analizar estos datos para recomendar productos a clientes específicos, mejorando así su experiencia de usuario.

En particular, existen dos estrategias principales en los sistemas de recomendación, el filtrado basado en contenido y el filtrado colaborativo. El primero crea perfiles para cada usuario o producto, caracterizando su naturaleza mediante atributos específicos. El segundo se basa únicamente en el comportamiento pasado del usuario, sin necesidad de crear perfiles explícitos. Nosotros nos centraremos dentro del filtrado colaborativo, donde los modelos de factorización de matrices han demostrado ser efectivos (Koren, Bell, y Volinsky, 2009).

Los métodos de factorización de matrices son reconocidos por su capacidad para gestionar grandes volúmenes de datos, su popularidad se debe a su flexibilidad para modelar una variedad de situaciones reales y su eficacia en generar recomendaciones precisas y personalizadas. En las siguientes secciones, exploraremos en detalle cómo los métodos de factorización de matrices han evolucionado para mejorar la calidad de las recomendaciones, revisando diversas técnicas y su aplicabilidad en diferentes contextos.

2.5.1. Técnicas de factorización matricial

Los sistemas de recomendación basados en filtrado colaborativo generalmente se componen de un conjunto de usuarios, un conjunto de ítems y las preferencias de los usuarios por varios ítems, las cuales suelen representarse en forma de tuplas [Usuario, Elemento, Calificación]. Al agrupar estas tuplas, se obtiene una matriz de calificaciones R de tamaño $m \times n$, donde m es el número de usuarios y n el número de ítems. En esta matriz, cada entrada r_{ij} representa la calificación que el usuario i le ha dado al ítem j. Dado que la matriz de calificaciones suele ser extremadamente dispersa, con muchos valores faltantes, uno de los principales desafíos es predecir estas calificaciones ausentes para poder recomendar ítems no evaluados por los usuarios. Aquí es donde entran las técnicas de factorización matricial, como la Descomposición en Valores Singulares (SVD, por sus siglas en inglés) (Guan, Li, y Guan, 2017).

Descomposición en Valores Singulares (SVD)

La Descomposición en Valores Singulares (SVD) es una técnica de factorización matricial que descompone la matriz original R en dos matrices de rango inferior U y V^T. En términos matemáticos, se tiene que la SVD de una matriz R de tamaño $m \times n$ se expresa como (Guan y cols., 2017):

$$R = U^T V \tag{2.19}$$

donde U^T es la transpuesta de una matriz ortogonal de tamaño $m \times k$, donde k es el número de factores latentes. Por otro lado, V es una matriz ortogonal de tamaño $k \times n$. De este modo, la valoración r_{ij} con prediccción $\tilde{r}_{ij}$, que el i-ésimo usuario da a la j-ésima película puede representarse como:

$$\tilde{r}_{ij} = {U_i}^T V_j \tag{2.20}$$

donde U_i, V_j son los vectores asociados al i-ésimo usuario y la j-ésima película, respectivamente. El mayor reto consiste en calcular la correspondencia de cada artículo y usuario con los vectores indicados. Una vez el sistema de recomendación ha completado este mapeo, puede estimar fácilmente la valoración que un usuario dará a cualquier artículo utilizando la ecuación (2.20) (Koren y cols., 2009).

La aplicación de la Descomposición en Valores Singulares (SVD) en el ámbito del filtrado colaborativo implica la factorización de la matriz de valoración usuario-artículo. Este proceso suele enfrentar dificultades debido a la alta proporción de valores faltantes. La SVD convencional puede resultar problemática cuando el conocimiento sobre la matriz es incompleto, lo que puede llevar a un sobreajuste si solo se consideran las entradas conocidas. De este modo, para optimizar las aproximaciones de U y V, el sistema busca minimizar la discrepancia entre las puntuaciones existentes y los valores predichos, lo cual se logra mediante la minimización de la suma de errores al cuadrado (Guan y cols., 2017):

$$E = \frac{1}{2} \sum_{i,j \in \kappa} (r_{ij} - \tilde{r}_{ij})^2 + \frac{k_u}{2} \sum_{i=1}^{m} {U_i}^2 + \frac{k_v}{2} \sum_{j=1}^{n} {V_j}^2, \tag{2.21}$$

donde κ es un conjunto de pares (u, i) a los que se han asignado valores originalmente en la matriz de calificación R (también conocida como conjunto de entrenamiento) y los parámetros de regularización k_u y k_v se utilizan para aliviar el sobreajuste mencionado y pueden calcularse por ejemplo por validación cruzada (Koren y cols., 2009).

Con el objetivo de resolver el problema de optimización que minimice la ecuación (2.21), podemos recurrir al descenso de gradiente estocástico (stochastic gradient descent, SGD). Este algoritmo itera sobre las valoraciones de entrenamiento (κ) y modifica U y V en la dirección opuesta al gradiente. A continuación, el Algoritmo 4 nos indica el procedimiento a seguir (Gemulla, Nijkamp, Haas, y Sismanis, 2011).

Algoritmo 4 SGD para la Factorización Matricial

Entrada:

• Un conjunto de entrenamiento: Z.

• Valores iniciales de: U_0 y V_0.

Mientras no haya convergencia:

Seleccionamos un punto de entrenamiento $(i, j) \in Z$ uniformemente al azar. Se reitera de modo:

$U_i \leftarrow U_i - \alpha \frac{\partial E_{ij}}{\partial U_i}$

$V_j \leftarrow V_j - \alpha \frac{\partial E_{ij}}{\partial V_j}$

donde α es la tasa de aprendizaje.

Finalmente se actualizan los valores de U_i y V_j hasta lograr convergencia.

Dicha técnica es una herramienta poderosa y eficiente para la factorización matricial. Al iterar sobre las valoraciones de entrenamiento y actualizar los factores latentes en la dirección opuesta al gradiente del error, SGD puede manejar de manera efectiva la dispersión de datos y la presencia de valores faltantes en la matriz de calificaciones. Esto permite generar recomendaciones precisas y personalizadas para los usuarios. El uso de SGD en combinación con técnicas de regularización, como se muestra en el algoritmo anterior, ayuda a mitigar el riesgo de sobreajuste, mejorando así la generalización del modelo a nuevos datos.

En resumen, la integración de SGD en la factorización matricial no solo optimiza la precisión de las predicciones de calificaciones, sino que también permite la escalabilidad necesaria para aplicaciones del mundo real, haciendo de esta técnica una elección valiosa para el desarrollo de sistemas de recomendación robustos y eficientes.

2.6. Clustering con k-means

El clustering es una técnica fundamental en el análisis de datos que permite clasificar y buscar patrones ocultos en los conjuntos de datos. Este proceso implica agrupar objetos de datos en clusters disjuntos de forma que los datos dentro del mismo cluster sean similares, mientras que los datos pertenecientes a diferentes clusters difieran entre sí. La creciente demanda de organizar el aumento exponencial de datos y extraer información valiosa de ellos ha hecho que las técnicas de clustering se apliquen ampliamente en diversas áreas. Entre los algoritmos de clustering más utilizados en la actualidad se encuentra k-means, un método numérico, no supervisado, no determinístico y iterativo. Su simplicidad y velocidad lo han convertido en una herramienta eficaz para obtener buenos resultados de clustering en muchas aplicaciones prácticas (Bock, 2007).

Las técnicas de clustering de datos son métodos de análisis descriptivo que se pueden aplicar a conjuntos de datos multivariados para descubrir la estructura presente en ellos. Son particularmente útiles cuando las estadísticas clásicas de segundo orden (la media muestral y la covarianza) no se pueden utilizar. En el análisis exploratorio de datos, se asume que no se tiene conocimiento previo sobre el conjunto de datos ni sobre su distribución. En tal situación, el clustering de datos puede ser una herramienta valiosa. El clustering de datos es una forma de clasificación no supervisada, ya que los clusters se forman evaluando las similitudes y disimilitudes de características intrínsecas entre diferentes casos, y la agrupación se basa en estas similitudes emergentes y no en un criterio externo. Además, estas técnicas pueden ser útiles para conjuntos de datos con una dimensionalidad superior a tres, ya que es muy difícil para los humanos comparar elementos de tal complejidad de manera confiable sin un soporte que ayude en la comparación.

El k-means pertenece a las técnicas de agrupación basadas en particionamiento, las cuales se basan en la reubicación iterativa de puntos de datos entre clusters. En particular, se utiliza para agrupar individuos. Si el algoritmo se aplica a los casos o a las variables del conjunto de datos depende de qué dimensiones de este conjunto de datos se desean reducir. El objetivo es producir grupos de casos/variables con un alto grado de similitud dentro de cada grupo y un bajo grado de similitud entre

grupos (Hastie y cols., 2009). La técnica de clustering k-means también se describe como un modelo de centroides, ya que se utiliza un vector que representa la media para describir cada cluster. MacQueen (1967), creador de uno de los algoritmos k-means, consideró que el principal uso del clustering k-means es proporcionar a los investigadores una comprensión cualitativa y cuantitativa de grandes conjuntos de datos multivariados, más que encontrar una agrupación única y definitiva para los datos (Morissette y Chartier, 2013).

Este capítulo presenta una revisión teórica del clustering con especial énfasis en el algoritmo k-means. En las siguientes secciones, se explicará en detalle el funcionamiento del algoritmo y su aplicabilidad

2.6.1. k-means

El algoritmo k-means es una técnica de *clustering* ampliamente utilizada debido a su simplicidad y eficiencia. Se trata de un método iterativo, no supervisado, diseñado para particionar un conjunto de datos en k *clusters*, donde k es un número predefinido de *clusters*. El objetivo principal de k-means es minimizar la variabilidad dentro de cada *cluster* y maximizar las diferencias entre estos (Na, Yong, y Xumin, 2010). Para ello, este método agrupa objetos de datos en clusters basándose en la minimización de la suma del error cuadrático entre los objetos y los centros de sus respectivos clusters. Los objetos se representan como vectores reales de n dimensiones, $x = (x_1, x_2, \ldots, x_n)$, y el algoritmo k-means construye k grupos $S = \{S_1, S_2, \ldots, S_k\}$, minimizando la suma de distancias de los objetos dentro de cada grupo a su centroide. En el contexto del algoritmo k-means, el centroide de un cluster es el punto de referencia calculado como la media aritmética de las coordenadas de todos los puntos en ese cluster. Supongamos que el objeto a clasificar es x. El algoritmo k-means tiene como objetivo elegir centroides que minimicen la siguiente expresión (Na y cols., 2010):

$$E = \sum_{i=1}^{k} \sum_{x \in C_i} \|x - \mu_i\|^2,$$

donde E es la suma del error cuadrático de todos los objetos en la base de datos, C_i es el conjunto de objetos en el cluster i, y μ_i es el centroide del cluster i. En particular, el algoritmo k-means se basa en la distancia euclidiana para determinar la cercanía entre cada objeto de datos y el centroide del cluster. La distancia euclidiana entre un vector $x = (x_1, x_2, \ldots, x_n)$ y otro vector $y = (y_1, y_2, \ldots, y_n)$ se define como (Na y cols., 2010):

$$d(x, y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}.$$

Dicho algoritmo consta de tres pasos principales inicialización, asignación y actualización. En la inicialización, se eligen k centroides, típicamente de forma aleatoria. Luego, cada objeto de los datos se asigna al centroide más cercano en el paso de asignación. En el paso de actualización, se recalculan las posiciones de los centroides como el promedio de los objetos asignados a cada grupo. Estos pasos de asignación y actualización se repiten iterativamente hasta que los centroides no se muevan significativamente, o se muevan por debajo de una distancia umbral. Como ya indicamos, el principal objetivo del algoritmo k-means es resolver el problema de optimización minimizando la suma de las distancias cuadráticas de cada objeto al centroide de su cluster (Pedregosa y cols., 2011), es decir minimizar E. El procedimiento a seguir se muestra en detalle en el desarrollo del Algoritmo 5 (Na y cols., 2010).

Algoritmo 5 Algoritmo k-means

Entrada:

• Número de clusters deseados: k,

• Una base de datos $D = \{d_1, d_2, ...d_n\}$ que contiene n objetos de datos

Inicialización:

• Seleccionar aleatoriamente k objetos de datos del conjunto de datos D como centros de cluster iniciales.

• Calcular la distancia entre cada objeto de datos d_i (donde $(1 \leq i \leq n)$ y todos los centros de clusters c_j (donde $1 \leq j \leq k$)

• Asignar el objeto de datos d_i al cluster más cercano.

• Para cada cluster j $(1 \leq j \leq k)$, recalcular el centro del cluster.

• Repetir el proceso hasta que no haya cambios.

Salida:

Un conjunto de k clusters.

El algoritmo k-means, a pesar de su popularidad y eficiencia, presenta ciertas limitaciones que pueden afectar su rendimiento en la práctica. Una de las principales deficiencias es la necesidad de calcular la distancia de cada objeto de datos a cada centro de cluster en cada iteración. Esto ocasiona un elevado costo computacional, especialmente cuando se trabaja con grandes conjuntos de datos. Sin embargo, existen posibles alternativas, pues se ha observado que no siempre es necesario recalcular estas distancias en cada iteración. Por ejemplo, una vez que un objeto de datos se asigna a un cluster y permanece en él durante varias iteraciones, la distancia a otros centros de clusters rara vez cambia significativamente. Sin embargo, el algoritmo k-means sigue realizando estos cálculos redundantes, lo que incrementa el tiempo de ejecución y afecta la eficiencia del algoritmo. Estas limitaciones han llevado al desarrollo de variantes y mejoras del algoritmo k-means, entre las que destaca el algoritmo k-means++. En la siguiente sección, se presentará el algoritmo k-means++, que aborda algunas de estas deficiencias introduciendo un método más eficiente para la inicialización de los centros de clusters, mejorando así tanto la velocidad como la precisión del proceso de clustering (Na y cols., 2010).

2.6.2. k-means++

Como indicamos anteriormente, la implementación del algoritmo k-means presenta importantes limitaciones, especialmente cuando trabajamos con grandes conjuntos de datos. Además de la necesidad de recalcular las distancias en cada iteración, otra limitación significativa del algoritmo k-means es su sensibilidad a la elección inicial de los centros. Este problema puede conducir a agrupaciones erróneas, ya que la selección inicial de centros aleatorios puede resultar en configuraciones erróneas que afectan negativamente la calidad de los clusters formados.

Para abordar tal deficiencia, se desarrolló una variante conocida como k-means++, el cuál se describe en detalle en Algoritmo 6 (Na y cols., 2010). Esta técnica mejora la selección inicial de los centros al elegirlos de forma ponderada, basándose en la distancia al cuadrado de cada punto al centro más cercano ya seleccionado. Este método no solo es rápido y sencillo, sino que también proporciona mejores garantías de rendimiento que el k-means tradicional (Arthur y Vassilvitskii, 2007). Al mejorar la dispersión inicial de los centros, k-means++ reduce significativamente las probabilidades de converger a mínimos locales pobres, mejorando la eficacia y la eficiencia del clustering resultante.

Algoritmo 6 Algoritmo k-means++

Entrada:
Conjunto de datos $D = \{d_1, d_2, \ldots, d_n\}$
Salida:
Un conjunto de k centros iniciales
Procedimiento:

1. Seleccionar un punto d_1 de D al azar como el primer centro.
2. Para cada punto $x \in D$, calcular $D(x)$, que es la distancia entre x y el centro más cercano que ya ha sido seleccionado.
3. Seleccionar un nuevo punto d_i al azar de D como un nuevo centro, utilizando una distribución de probabilidad ponderada donde un punto x es seleccionado con una probabilidad proporcional a $D(x)^2$.
4. Repetir los pasos 2 y 3 hasta que se hayan seleccionado k centros.
5. Utilizar los k centros como inicialización para el algoritmo k-means estándar.

En definitiva, el algoritmo k-means++ tiene como objetivo mejorar la precisión del k-means inicializando los centros de los clusters de manera más efectiva. Mientras que k-means selecciona los centros iniciales de forma aleatoria, k-means++ elige los centros de manera que se maximicen las distancias entre ellos, minimizando la probabilidad de obtener malos resultados. El procedimiento de selección de centros en k-means++ se realiza a través de una distribución de probabilidad que da mayor peso a los puntos que están más alejados de los centros ya seleccionados. Esto asegura una mejor dispersión inicial, lo que generalmente conduce a una mejor convergencia del algoritmo y a una mayor calidad de los clusters obtenidos.

2.7. Métricas de evaluación

En la evaluación de modelos de aprendizaje automático, es fundamental comprender y utilizar diversas métricas para medir su rendimiento y generalización. En esta sección, exploraremos de manera teórica las principales métricas de evaluación tanto para modelos de regresión como de clasificación, así como aquellas asociadas a modelos más especializados, como la factorización de matrices.

2.7.1. Métricas de evaluación para regresión

En esta sección, exploraremos las métricas de evaluación esenciales para analizar el desempeño de modelos de regresión. En particular, nos centraremos en el modelo de regresión lineal múltiple aunque estas métricas se pueden extrapolar a los modelos de regresión restantes. Dichas métricas proporcionan una medida cuantitativa de la precisión de las predicciones en comparación con los valores reales. Con ellas, podremos evaluar de manera efectiva la eficacia de nuestro modelo. Denotaremos por Y_i a la observación y por $\hat{Y}_i$ a la predicción de la respuesta.

Error absoluto medio

El error absoluto medio (*Mean Absolute Error*, MAE) se calcula como la suma de los errores absolutos dividida por el tamaño de la muestra:

$$\text{MAE} = \frac{\sum_{i=1}^{n} |\hat{\epsilon}_i|}{n} = \frac{\sum_{i=1}^{n} |Y_i - \hat{Y}_i|}{n}. \tag{2.22}$$

Este valor, proporciona una medida promedio de cuánto se desvían las predicciones del modelo de los valores reales. Un MAE más bajo indica que el modelo tiene una mejor capacidad para prever las respuestas reales.

Error cuadrático medio

El error cuadrático medio (*Mean Squared Error*, MSE) es la suma de los cuadrados de las diferencias entre valores observados y valores predichos:

$$\text{MSE} = \frac{\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2}{n}. \tag{2.23}$$

Es obvio, que este valor siempre será positivo. En particular, mide lo cerca que está la línea de mejor ajuste al conjunto de puntos. De modo que, cuanto más se acerque a 0, más precisa es la predicción.

Error logarítmico cuadrático medio

Esta métrica, adopta un enfoque similar al MSE. Sin embargo, utiliza un logaritmo en su expresión lo que implica que Y tiene que ser una variable con soporte positivo y las predicciones también deben tomar valores positivos (Pedregosa y cols., 2011):

$$\text{MSLE} = \frac{\sum_{i=1}^{n}[\log(1 + Y_i) - \log(1 + \hat{Y}_i)]^2}{n}. \tag{2.24}$$

Este valor, mide la media de los cuadrados de las diferencias logarítmicas entre las predicciones y los valores reales. Aplicar logaritmo ayuda a compensar valores atípicos elevados en el conjunto de datos puesto que los trata como si estuvieran en la misma escala. De modo que, puede ser especialmente adecuado para problemas donde las variables tienen escalas muy diferentes. Al igual que el MSE, se busca minimizar el MSLE, es decir, obtener un modelo que reduzca en general el tamaño de los errores logarítmicos.

Error Absoluto Mediano

El error absoluto mediano, (*Median Absolute Error*, MedAE) es una métrica de evaluación para modelos de regresión que se basa en la mediana de los valores absolutos de las diferencias entre las predicciones y los valores reales. Aquí tenemos su expresión asociada:

$$\text{MedAE} = \text{Mediana}(|Y_i - \hat{Y}_i|),\ i \in \{1, \cdots, n\}. \tag{2.25}$$

En específico, la mediana representa el punto que separa el conjunto de datos en dos partes iguales, por lo tanto, el MedAE indica que al menos el 50 % de los errores son iguales o menores a este valor. Este valor, al igual que el anterior es más robusto ante valores atípicos, ya que no se ven afectados de manera significativa por errores extremadamente grandes.

R-cuadrado o coeficiente de determinación

Este índice, determina la proporción de varianza de la variable dependiente que puede explicarse por las variables independientes. Su expresión es la siguiente:

$$\text{R}^2 = 1 - \frac{\text{RSS}}{\text{TSS}} = 1 - \frac{\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2}{\sum_{i=1}^{n}(Y_i - \bar{Y})^2},$$

siendo RSS la suma residual de cuadrados y TSS la suma total de cuadrados. Este valor proporciona una medida de la bondad de ajuste del modelo. Cuanto más alto sea R^2, mejor se ajusta el modelo a los datos. Un valor cercano a 1 indica que el modelo explica una gran parte de la variabilidad, mientras que

un valor cercano a 0 indica que el modelo no explica casi ninguna variabilidad. Sin embargo, es crucial comprender que un R^2 elevado no garantiza automáticamente la validez del modelo, ya que no aborda por completo la veracidad de las hipótesis subyacentes. Por ejemplo, si la relación real entre las variables no es lineal, un alto R^2 podría resultar engañoso, ya que el modelo puede no estar capturando de manera adecuada la verdadera estructura. Además, un valor elevado de dicho coeficiente, no garantiza que se cumplan las hipótesis de homocedasticidad, independencia de los errores o normalidad. Podrían darse patrones de heterocedasticidad, correlación entre los residuos o desviaciones de la normalidad, que afectarían a la validez de las inferencias realizadas a partir del modelo. Adicionalmente, un R^2 alto puede ser sensible a la presencia de valores atípicos, ya que un solo dato influyente puede afectar significativamente la magnitud de este valor.

En definitiva, será fundamental complementar la evaluación del R-cuadrado, con un análisis más detallado de las hipótesis del modelo y considerar otras métricas de evaluación para obtener una imagen completa de la validez del modelo de regresión.

2.7.2. Métricas de evaluación para modelos de clasificación

Las métricas de clasificación son herramientas valiosas para evaluar la eficacia de los modelos, proporcionando así, información detallada sobre su rendimiento. En esta sección, introduciremos una exploración teórica, con la que desentrañaremos estas métricas para ofrecer una comprensión más profunda de la capacidad predictiva y la robustez de los modelos de clasificación.

Matriz de confusión

La matriz de confusión es una herramienta fundamental en la evaluación de modelos de clasificación. Su propósito es mostrar la relación entre las predicciones de un modelo y las clases reales de los datos. Se organiza en una matriz cuadrada $N \times N$, donde N es el número de clases y se diferencian diferentes elementos. En particular, tendremos los Verdaderos Positivos (*True Positives*, TP) que representan la cantidad de instancias positivas que el modelo ha predicho correctamente. Por otro lado, los Falsos Positivos (*False Positives*, FP) indican la cantidad de instancias negativas que el modelo ha predicho erróneamente como positivas. En contraste, los Falsos Negativos (*False Negatives*, FN) revelan la cantidad de instancias positivas que el modelo ha predicho incorrectamente como negativas. Finalmente, los Verdaderos Negativos (*True Negatives*, TN) reflejan la cantidad de instancias negativas que el modelo ha predicho correctamente. De esta forma, se tiene que los elementos diagonales representan el número de clasificaciones correctas para cada clase, mientras que los elementos fuera de la diagonal representan clasificaciones incorrectas. En la Figura 2.1, vemos la estructura de la matriz de confusión asociada a una clasificación binaria.

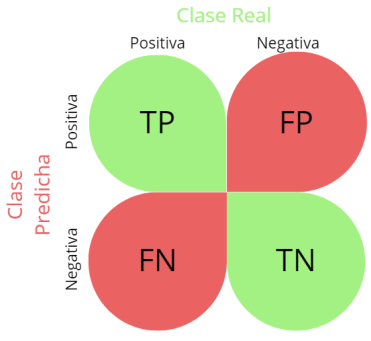


Figura 2.1: Matriz de confusión asociada a una clasificación binaria.

En esta matriz, los elementos diagonales representan el número de clasificaciones correctas para cada clase, mientras que los elementos fuera de la diagonal representan clasificaciones incorrectas. A partir de la información dada por la matriz de confusión se pueden calcular diferentes métricas asociadas como se verá a continuación.

Especificidad

La especificidad (en inglés *Precision*) mide la exactitud de un resultado positivo predicho. Se calcula como la proporción de verdaderos positivos respecto a la suma de verdaderos positivos y falsos positivos:

$$Precision = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{TN}}. \tag{2.26}$$

En definitiva, este valor indica qué tan confiables son las predicciones positivas del modelo. Por lo que será deseable un valor lo más próximo a uno posible.

Exhaustividad o Sensibilidad (*Recall*)

La sensibilidad nos indica cuántos de los casos positivos reales hemos podido predecir correctamente con nuestro modelo.

$$Recall = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}}. \tag{2.27}$$

Esta es una métrica muy útil en los casos en los que los falsos negativos superan a los falsos positivos, (Karimi, 2021). Un valor alto de sensibilidad indica que el modelo tiene una capacidad sólida para identificar correctamente las instancias positivas, lo que sugiere que el modelo es efectivo para detectar la presencia de la clase de interés. Por otro lado, un valor bajo de sensibilidad indica que el modelo está perdiendo muchos casos positivos, lo que puede ser problemático en situaciones donde es crucial detectar todos los casos positivos.

Precisión global (*Accuracy*)

La precisión global (*accuracy*) es una métrica fundamental en la evaluación de modelos de clasificación. Representa la proporción de predicciones correctas realizadas por el modelo sobre el total de predicciones realizadas en el conjunto de datos. En otras palabras, la precisión global indica qué tan exacto es el modelo en general. Se calcula dividiendo el número de predicciones correctas (verdaderos positivos más verdaderos negativos) entre el número total de predicciones realizadas, (Karimi, 2021). Matemáticamente, se expresa como:

$$Accuracy = \frac{\mathrm{TP} + \mathrm{TN}}{\mathrm{TP} + \mathrm{TN} + \mathrm{FP} + \mathrm{FN}}. \tag{2.28}$$

Un valor alto de precisión indica que el modelo está haciendo predicciones correctas en la mayoría de los casos, lo que sugiere que es confiable y preciso en su capacidad de clasificación. Por otro lado, una precisión baja indica que el modelo está cometiendo un número significativo de errores en sus predicciones, lo que puede ser problemático en aplicaciones donde la precisión es crucial.

Puntuación F1 (*F1 score*)

Cuando intentamos aumentar la precisión del modelo, la sensibilidad disminuye y viceversa. La puntuación F1 es una media armónica de la precisión y la sensibilidad, por lo que da una idea combinada de estas dos métricas, (Karimi, 2021). Es particularmente útil en problemas de clasificación donde hay un desequilibrio entre las clases, es decir, cuando hay un número significativo de falsos positivos y falsos negativos. Se calcula como la media armónica de precisión y sensibilidad, y se expresa matemáticamente como, (Vujović, 2021):

$$F1\ Score = \frac{2 \cdot Precision \cdot Recall}{Precision \cdot Recall} \tag{2.29}$$

Una puntuación F1 alta indica que el modelo tiene un equilibrio entre precisión y sensibilidad, lo que significa que está haciendo un buen trabajo tanto en la identificación de verdaderos positivos como en la minimización de falsos positivos y falsos negativos. Por otro lado, una puntuación F1 baja sugiere que el modelo puede estar sesgado hacia la precisión o la sensibilidad, lo que significa que puede estar sacrificando una de las métricas a expensas de la otra. Esto puede ocurrir en situaciones donde hay un desequilibrio significativo entre las clases y el modelo está optimizado para una métrica a costa de la otra.

En resumen, una puntuación F1 alta indica un buen equilibrio entre precisión y sensibilidad, mientras que una puntuación F1 baja sugiere que el modelo puede necesitar ajustes para mejorar su rendimiento en términos de clasificación correcta y minimización de errores.

Log Loss

La pérdida logarítmica, también conocida como *log loss*, es una medida utilizada para evaluar el rendimiento de un modelo de clasificación. Se expresa como una función de la discrepancia entre las probabilidades predichas por el modelo y las etiquetas reales de los datos. La fórmula de la pérdida logarítmica (Log Loss) se expresa como (Pedregosa y cols., 2011):

$$\text{Log Loss} = -\frac{1}{N}\sum_{i=1}^{N}\left(Y_i \log(p_i) + (1 - Y_i)\log(1 - p_i)\right), \tag{2.30}$$

donde N es el número total de muestras, Y_i es la etiqueta real (1 si el evento ocurrió, 0 si no) y p_i es la probabilidad predicha por el modelo de que el evento ocurra para la muestra i.

En esencia, la pérdida logarítmica cuantifica qué tan bien el modelo hace sus predicciones. Cuanto más cerca estén las probabilidades predichas de las etiquetas reales, menor será la pérdida logarítmica. Por otro lado, cuanto mayor sea la discrepancia entre las probabilidades predichas y las etiquetas reales, mayor será la pérdida logarítmica.

En resumen, la pérdida logarítmica proporciona una medida de la calidad de las probabilidades predichas por un modelo de clasificación. Cuanto menor sea la pérdida logarítmica, mejor será el rendimiento del modelo en términos de precisión y confianza en sus predicciones.

Área bajo la curva ROC (ROC AUC)

La curva ROC (*Receiver Operating Characteristic*) es una herramienta gráfica utilizada para evaluar y visualizar el rendimiento de un modelo de clasificación. Representa la tasa de verdaderos positivos (sensibilidad) en el eje Y y la tasa de falsos positivos (1 - especificidad) en el eje X, (Vujović, 2021). Cada punto en la curva ROC representa un umbral de decisión diferente, y trazar la curva ROC permite evaluar cómo cambia el rendimiento del modelo a medida que se ajusta el umbral de clasificación.

La curva ROC es útil porque proporciona una representación completa del rendimiento del modelo en todas las posibles configuraciones de sensibilidad y especificidad. Cuanto más alejada esté la curva ROC del punto de referencia diagonal (que indica un rendimiento aleatorio), mejor será el rendimiento del modelo.

El área bajo la curva ROC (ROC AUC, del inglés *'Area Under the Curve'*) es una medida numérica de la capacidad discriminatoria del modelo. Es el área bajo la curva ROC y proporciona una única medida del rendimiento global del modelo en lugar de evaluar su rendimiento en diferentes umbrales de decisión. El valor de ROC AUC varía entre 0 y 1, donde un valor de 1 indica un modelo perfecto que puede distinguir perfectamente entre las clases positiva y negativa, mientras que un valor de 0.5 indica un rendimiento aleatorio similar al lanzamiento de una moneda.

2.7.3. Métricas de Evaluación para Clustering

En esta sección teórica, exploraremos dos métricas fundamentales para evaluar la calidad de los resultados obtenidos mediante algoritmos de clustering. Estas métricas proporcionan una medida ob-

jetiva de la cohesión y separación de los clusters generados por un algoritmo de agrupamiento. A través de cada métrica, analizaremos cómo pueden ayudarnos a comprender y comparar el rendimiento de diferentes enfoques de clustering.

Índice de Davies-Bouldin

En el proceso de aprendizaje automático, evaluar modelos es esencial para garantizar su efectividad. Cuando se trata de evaluar la agrupación de datos en clústeres, es crucial determinar cuántos clústeres son óptimos. El Índice de Davies-Bouldin nos ofrece una forma de evaluar la calidad de la agrupación al comparar la similitud promedio entre los clústeres más similares entre sí. La expresión matemática que define este índice y proporciona una medida cuantitativa de la separación entre los clústeres es la siguiente:

$$\bar{R} = \frac{1}{k} \sum_{i=1}^{k} \max_{i \neq j} R_{ij}, \tag{2.31}$$

donde k es el número de clústeres y R_{ij} indica la relación entre las similitudes medias de cada conglomerado con su conglomerado más similar y la distancia entre los centroides de los conglomerados. Es decir (Vergani y Binaghi, 2018):

$$R_{i,j} = \frac{S_i + S_j}{M_{i,j}},$$

siendo:

$$S_i = \left\{ \frac{1}{T_i} \sum_{j=1}^{T_i} \|X_j - A_i\|^q \right\}^{\frac{1}{q}}$$

$$M_{i,j} = \left\{ \sum_{k=1}^{N} \|a_{ki} - a_{kj}\|^p \right\}^{\frac{1}{p}},$$

en el que T_i representa el número de vectores en el clúster i, X_j son los vectores en i, y A_i es el centroide del clúster i. Cuando $q = 1$, S_i es la distancia euclidiana promedio de los vectores a los centroides de i, mientras que si $q = 2$, S_i es la desviación estándar de la métrica sobre las muestras en el clúster en relación con su centroide. S_j está definido de manera equivalente. Por otro lado, a_{ki} es el k-ésimo componente del vector n-dimensional a_i, que es el centroide del clúster i. $M_{i,j}$ es la métrica de Minkowski de los centroides i y j; luego, si $p = 1$, es la distancia de bloque de ciudad/Manhattan, mientras que si $p = 2$, es la distancia euclidiana entre los centroides y si $p = q = 2$, $R_{i,j}$ es la medida de similitud de Fisher calculada entre el clúster i y el clúster j.

De esta forma, un valor del índice de Davies-Bouldin(DB) más bajo indica una mejor calidad de la solución de agrupamiento. Por otro lado, valores más altos de dicho índice DB están asociados con soluciones de agrupamiento menos óptimas. Esto se debe a que un valor elevado sugiere que los grupos no están bien diferenciados entre sí o que estos no son compactos (Wijaya y cols., 2021).

Distancia media cuadrada

Esta métrica representa la distancia cuadrática media del modelo de clustering. Se calcula como la media de las distancias entre los puntos de datos de entrenamiento y su centroide más cercano (Google Cloud, 2024):

$$\text{Distancia media cuadrada} = \frac{1}{n} \sum_{i=1}^{n} \|x_i - \mu_{c_i}\|^2,$$

donde n es el número total de puntos de datos de entrenamiento, x_i es el i-ésimo punto de datos y μ_{c_i} es el centroide del cluster al que se asigna el punto de datos x_i. En otras palabras, para cada punto de

datos, se calcula la distancia al centroide de su cluster asignado, y luego se toma el promedio de estas distancias cuadráticas para obtener la distancia media cuadrada del modelo. Esta métrica proporciona una medida de la dispersión de los datos con respecto a sus centroides en el espacio de características. Cuanto mayor sea el valor de esta métrica, mayor será la dispersión de los datos, lo que puede indicar una menor cohesión de los clusters. Por otro lado, un valor más bajo de esta métrica sugiere una mayor cohesión de los clusters, lo que generalmente se considera deseable en el análisis de clustering.

Capítulo 3

Análisis de datos

[1]

Este capítulo nos permitirá ilustrar la teoría expuesta anteriormente, mediante ejemplos concretos. En particular, enfocándonos específicamente en la comparación entre la ejecución de modelos en BigQuery y Python. El contraste con Python no sólo nos permitirá comparar ambas opciones, sino que también destacará la sencillez que caracteriza a BigQuery ML. En el texto de esta parte de la memoria empleamos el lenguaje habitual del *machine learning*.

3.1. Modelo de regresión lineal

En primer lugar, en base a los conceptos teóricos explicados en el Capítulo 2, aplicaremos el modelo de regresión lineal a un conjunto de datos predeterminado. Es decir, analizaremos un conjunto de datos sobre la edad del abulón, obtenidos del repositorio de aprendizaje automático UCI que en inglés se conoce como *UCI Machine Learning Repository* (Nash, Sellers, Talbot, Cawthorn, y Ford, 1995).

3.1.1. Descripción del conjunto de datos

El conjunto de datos *abalone*, nos permitirá predecir la edad del abulón a partir de medidas físicas. La edad del abulón se determina cortando la concha a través del cono, teñiéndola y contando el número de anillos a través de un microscopio. Sin embargo, se pueden utilizar otras medidas más simples de obtener. En particular, nuestras variables explicativas serán, *Sex*, hace referencia al sexo del abulón, diferenciando entre macho (M), hembra (F) y infante (I), *Length*, medida más larga de la concha en milímetros, *Diameter*, medida perpendicular a la longitud, altura *Height*, peso entero en gramos ,*Whole_ weight*, peso sin cáscara, *Shucked_ weight*, peso intestinal, *Viscera_ weight* y peso después de secarse, *Shell_ weight*. Por otro lado, nuestra variable respuesta en el modelo es *Rings*, que hace referencia al número de anillos, a los cuáles si les sumamos 1,5 nos dará la edad en años.

De los datos originales se eliminaron ejemplos con valores faltantes (a la mayoría les faltaba el valor predicho) y los rangos de los valores continuos se escalaron para su uso con una ANN (dividiendo por 200).

3.1.2. Modelo de regresión lineal en Python

El código de Python comienza importando las bibliotecas necesarias: `pandas` para manipulación de datos, `google.cloud` para el uso de BigQuery, y módulos específicos de `scikit-learn` para la regresión lineal y métricas de evaluación. Luego, se inicializa el cliente de BigQuery y se especifica la información del proyecto, conjunto de datos y tabla que se utilizarán en la consulta. La consulta

[1] Utilizaremos lenguaje de machine learning a lo largo de está sección.

se realiza para obtener todos los datos de la tabla especificada en BigQuery y se almacenarán en un *DataFrame* de `pandas`. Este proceso es únicamente necesario porque los conjuntos de datos se importan directamente de Bigquery, esto se hace para que el origen de los datos sea el mismo y así poder tener una comparación más justa.

Posteriormente, los datos se dividen en características (X) y la variable objetivo (y). Cabe, destacar que en Python es necesario realizar una codificación *one-hot* para la columna 'sex' para convertir variables categóricas en variables *dummy*. Esto se lleva a cabo con la función `get_dummies` de `pandas` , como muestra la Figura 3.1, y crea nuevas columnas ('sex_M', 'sex_F', 'sex_I') con valores binarios para cada categoría. Este enfoque asegura que el modelo de regresión lineal y otros algoritmos puedan

```
# Dividir los datos en características (X) y la variable objetivo (y)
#Codificar las variables categóricas
df = pd.get_dummies(df, columns=["sex"], prefix=["sex"])
X = df.drop(["rings", "split", "is_test"], axis=1)
y = df["rings"]
```

Figura 3.1: División de los datos y codificación *one-hot* para la columna 'sex'.

interpretar correctamente la información de la columna sexo durante el análisis y entrenamiento del modelo.

Posteriormente, se obtienen conjuntos de entrenamiento (X_train, y_train) y conjuntos de prueba (X_test, y_test) filtrando los datos según la columna 'split'. Luego, se crea un modelo de regresión lineal utilizando `LinearRegression` de `scikit-learn` y se entrena el modelo con el conjunto de entrenamiento. Se realizan predicciones en el conjunto de prueba y se calculan métricas de evaluación del modelo para cotejar el rendimiento del modelo. Finalmente, se imprime la información del modelo, incluyendo los coeficientes para cada característica, el intercepto y las métricas calculadas.

Como resultado del modelo de regresión lineal entrenado utilizando los datos de abulones, con características como longitud, diámetro, altura, peso y la categoría de género obtenemos los coeficientes indicados en el Cuadro 3.1.

Estos coeficientes proporcionan información sobre la contribución relativa de cada característica en la predicción del número de anillos de los abulones. En particular, el intercepto, que representa el valor estimado de la variable de respuesta (número de anillos) cuando todas las características tienen un valor de cero, es de 3.6670. Se asume que las variables predictoras no toman valores negativos, lo que implica que esta interpretación se limita al rango de valores observados en los datos utilizados para el modelo.

Por otro lado, tendremos coeficientes positivos como la longitud, diámetro o altura entre otros y coeficientes negativos como el peso de la carne, el peso de las vísceras o el género indeterminado. En el contexto del modelo de regresión lineal, los coeficientes positivos indican una relación proporcional directa, lo que significa que a medida que el valor de una variable aumenta, el número de anillos de los abulones también tiende a aumentar. Por ejemplo, en el caso de la variable longitud, un aumento de una unidad de esta, está asociado con un incremento de aproximadamente 0.1478 unidades en el número de anillos de los abulones, siempre y cuando las demás variables se mantengan constantes.

Por el contrario, los coeficientes negativos indican una relación proporcional inversa, lo que implica que a medida que el valor de una variable aumenta, el número de anillos tiende a disminuir. Por ejemplo, coeficientes negativos para las variables 'shucked_weight' y 'viscera_weight' indican que un mayor peso de la carne y las vísceras se asocia con un menor número de anillos. Por ejemplo, un aumento de una unidad en el peso de la carne, suponiendo las variables restantes constantes, está relacionado con una disminución de aproximadamente 19.4627 unidades en el número de anillos.

Esta interpretación proporciona un entendimiento claro de cómo cada variable contribuye a la

Coeficiente	**Valor**
Longitud (length)	0.1478
Diámetro (diameter)	10.6433
Altura (height)	9.3834
Peso total (whole_weight)	8.5211
Peso de la carne (shucked_weight)	-19.4627
Peso de las vísceras (viscera_weight)	-9.4110
Peso de la concha (shell_weight)	8.9982
Género Femenino (sex_F)	0.2530
Género Indeterminado (sex_I)	-0.5903
Género Masculino (sex_M)	0.3373
Intercepto	3.6670

Cuadro 3.1: Estimadores de los parámetros del modelo de regresión lineal.

Métrica	**Valor**
Error Cuadrático Medio (MSE)	4.8132
R-cuadrado (R^2)	0.5520
Error Absoluto Medio (MAE)	1.5533
Error Logarítmico Cuadrado Medio (MSLE)	0.0316
Error Absoluto Mediano (MedAE)	1.1189

Cuadro 3.2: Métricas de evaluación del modelo de regresión lineal.

variable respuesta de nuestro modelo.

Por ejemplo, un MSE de 4.8132 indica la magnitud promedio del error cuadrático entre las predicciones y los valores reales.

Como indicamos en la sección de métricas de evaluación del modelo (2.3.1), estas métricas proporcionan una evaluación cuantitativa del rendimiento del modelo en el conjunto de validación. En particular, el error absoluto medio es 1.5533, representa que las predicciones se desvían en 1.5533 unidades de los valores reales. Por otro lado, el error cuadrático medio, que mide la discrepancia entre los valores predichos por el modelo y los valores reales, como se mencionó anteriormente, es de 4.8132. Es importante destacar que el error cuadrático medio no tiene un rango de valores específico que lo haga mejor o peor, ya que su magnitud depende del contexto del problema y de la escala de las variables

involucradas. Sin embargo, en términos generales, un error cuadrático medio más bajo indicaría una mayor precisión en las predicciones del modelo. Por lo tanto, aunque 4.8132 es el valor actual, se espera que este número sea lo más pequeño posible para mejorar la precisión del modelo y su capacidad para predecir con mayor exactitud los valores reales. El error logarítmico cuadrático Medio es 0.0316 y el error absoluto mediano es 1.1189. Ambas medidas son menos sensibles a la presencia de valores atípicos en comparación con el MAE, y de nuevo la idea es obtener los valores más pequeños posibles. Por último, el valor del coeficiente de determinación o R-cuadrado (R^2) es 0.5520, lo que indica que aproximadamente el 55,20 % de la variabilidad en la variable de respuesta se explica por el modelo. En resumen, nuestro modelo tiene cierta capacidad predictiva, pero existen áreas de mejora, especialmente en términos de reducción de errores y explicación de la variabilidad observada. Considerar ajustes o explorar modelos más complejos podría ser útil para mejorar la precisión y la capacidad explicativa del modelo.

3.1.3. Modelo de regresión lineal en Bigquery

En el contexto de BigQuery, la creación de un modelo de regresión lineal sigue un proceso claro y estructurado. Se inicia con la cláusula CREATE OR REPLACE MODEL, donde se especifica el proyecto, conjunto de datos y nombre del modelo, indicando la creación de uno nuevo o la actualización de uno existente. Luego, la sección OPTIONS se utiliza para establecer configuraciones específicas, es relevante señalar que las principales opciones disponibles se detallan en el Cuadro 3.3. Entre estas opciones, la estrategia para la tasa de aprendizaje (LEARN_RATE_STRATEGY) destaca como un factor crucial en BigQuery ML. Esta estrategia controla la magnitud de los ajustes en los coeficientes del modelo, y LINE_SEARCH, el valor predeterminado, dinámicamente ajusta la tasa de aprendizaje según la reducción de la función de pérdida.

Adicionalmente, es importante resaltar que BigQuery realiza la codificación de variables no numéricas de forma predeterminada mediante ONE_HOT_ENCODING. Esta configuración crea columnas binarias para cada categoría, siguiendo una lógica similar a la implementación en Python. Por último, en cuanto a la gestión de los datos para el entrenamiento, BigQuery simplifica notablemente el proceso. AUTO_SPLIT, la opción predeterminada, ajusta automáticamente la división según el tamaño del conjunto de datos. RANDOM introduce aleatoriedad, CUSTOM permite la división mediante una columna específica, SEQ realiza divisiones secuenciales y NO_SPLIT utiliza todos los datos para el entrenamiento. Este enfoque en la sintaxis de BigQuery proporciona un marco sencillo para el desarrollo de modelos de regresión lineal.

En nuestro caso particular, para la creación del modelo para predecir el número de anillos de los abulones, podemos aplicar el código indicado en la Figura 3.2. Seleccionamos el tipo de modelo como 'linear_reg', y optamos por una división personalizada de los datos, indicando la columna 'is_test' como el criterio de división. Además, especificamos que las etiquetas de entrada estarán en la columna 'Rings'. Esta configuración refleja el enfoque en la predicción de la variable ´Rings' basándonos en las características proporcionadas en el conjunto de datos de abalones. En cuánto a los opciones por defecto, ONE_HOT_ENCODING, creará columnas binarias para cada categoría de la variable sexo, siguiendo una lógica similar a la implementación en Python. Por otro lado, en el caso de Bigquery no es necesario especificar la división de los datos como se hacía en Python, si no que de manera implícita al usar CUSTOM se lleva a cabo la división en base a una columna específica, en nuestro caso la columna 'is_test'. Esta columna debe ser de tipo booleano.

En relación a las limitaciones de BigQuery para el modelo de regresión lineal, es importante tener en cuenta que la columna de etiquetas debe contener valores reales y no puede tener valores infinito o NaN (*not a number*). La estrategia de optimización NORMAL_EQUATION, que calcula directamente la solución de mínimos cuadrados del problema de regresión lineal con la fórmula analítica, no admite una cardinalidad total de las funciones superior a 10.000. En otras palabras, cuando se utiliza esta estrategia de optimización, el modelo de regresión lineal no puede procesar conjuntos de datos que tengan más de 10.000 características distintas o variables independientes que se utilicen para predecir

```
# Define la consulta SQL para crear un modelo de regresión lineal con división personalizada de datos
query = f"""
CREATE OR REPLACE MODEL `{project_id}.{dataset_id}.{model_name}`
OPTIONS(
  model_type='linear_reg',
  data_split_method='CUSTOM',
  data_split_col='is_test',  # Nombre de la columna de división de datos
  input_label_cols=['Rings']
) AS
SELECT
  length,
  diameter,
  height,
  whole_weight,
  shucked_weight,
  viscera_weight,
  shell_weight,
  rings,
  is_test
FROM
  `{project_id}.{dataset_id}.abalone`
"""
```

Figura 3.2: Sintaxis de Bigquery para la creación del modelo de regresión lineal múltiple.

la variable dependiente. Asimismo, la búsqueda de líneas, un método que ajusta dinámicamente la tasa de aprendizaje en función de cómo se está reduciendo la función de pérdida o error del modelo, puede ralentizar el entrenamiento y aumentar la cantidad de bytes procesados, aunque generalmente converge incluso con una tasa de aprendizaje inicial mayor. Además, la utilización del algoritmo de búsqueda 'GRID_SEARCH', búsqueda en cuadrícula, solo es posible cuando el espacio de búsqueda de cada hiperparámetro es discreto. Estas consideraciones son esenciales para garantizar la efectividad y eficiencia del modelo en el entorno de BigQuery.

Mediante la sintaxis proporcionada en BigQuery, se generan las estadísticas de entrenamiento para evaluar el rendimiento del modelo y las métricas de evaluación asociadas al mismo. Para acceder a las estadísticas de entrenamiento, Figura 3.3, debemos abrir el panel de información del modelo, seleccionar la pestaña Entrenamiento y, a continuación, hacer clic en Tabla. Un algoritmo de aprendizaje automático examina muchos ejemplos y trata de encontrar un modelo que minimice la pérdida. La pérdida (*loss*) es una sanción por una predicción incorrecta, un número que indica qué tan mala fue la predicción del modelo en un solo ejemplo. Si la predicción del modelo es perfecta, la pérdida es cero; de lo contrario, es mayor. El objetivo de entrenar a un modelo es encontrar un conjunto de pesos y ordenadas al origen que tengan, en promedio, una pérdida baja en todos los ejemplos (Google Cloud, 2024).

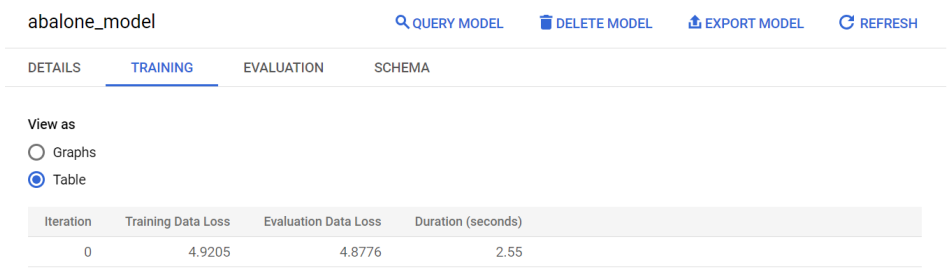


Figura 3.3: Estadísticas de entrenamiento del modelo de regresión lineal múltiple en Bigquery.

En particular, la columna *Training Data Loss* (Pérdida de datos de entrenamiento) representa la métrica de pérdida calculada después de que se entrena el modelo en el conjunto de datos de entrena-

miento. Como se realizó una regresión lineal, esta columna muestra el valor del error cuadrático medio. Se utiliza automáticamente una estrategia de optimización 'normal_equation' para este entrenamiento, lo que significa que no son necesarias iteraciones para converger en el modelo final. Esto se debe a que las ecuaciones normales de la regresión proporcionan una solución explícita.

Por otro lado, si en el panel de información del modelo en Google Cloud vamos al apartado de evaluación, obtendremos las métricas asociadas al modelo. Estas métricas varían en función del modelo especificado. En el caso de la regresión lineal, obtenemos las métricas indicadas en la Figura 3.4.

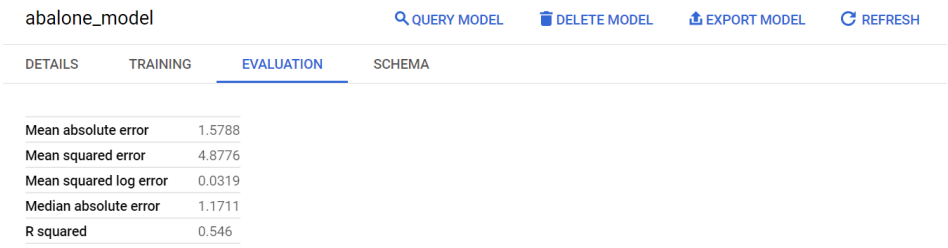


Figura 3.4: Métricas del modelo de regresión lineal múltiple en Bigquery.

La interpretación de dichas métricas es análoga a lo indicado en el apartado anterior. El error absoluto medio (MAE), 1.5788, representa que las predicciones se desvían en 1.5788 unidades de los valores reales. Por otro lado, el error cuadrático medio (MSE) es 4.8776, este valor debería aproximarse más a cero. El error logarítmico cuadrático medio (MSLE) es 0.0319 y el error absoluto mediano (MedAE) es 1.1711. Por último, el valor del coeficiente de determinación o R-cuadrado (R^2) es 0.546, lo que indica que aproximadamente se explica el 54.6 % de la variabilidad en la variable de respuesta.

Después de haber creado el modelo de regresión lineal mediante consultas SQL en BigQuery, se abren diversas posibilidades para explorar y comprender mejor el rendimiento del modelo. Una de estas opciones es utilizar la función ML.PREDICT para realizar predicciones basadas en el modelo entrenado. Luego, para profundizar en la interpretación de los resultados de predicción, la función ML.EXPLAIN_PREDICT proporciona información detallada sobre la contribución de cada atributo a las predicciones, brindando una visión más completa del proceso. Adicionalmente, para entender qué características son generalmente más influyentes en la determinación del número de anillos de un abulón, se puede explorar la función ML.GLOBAL_EXPLAIN. Aunque en este caso no nos enfocamos en estas opciones específicas, es importante mencionar que ML.GLOBAL_EXPLAIN ofrece explicaciones globales sobre las características más relevantes del modelo. Cabe destacar que para aprovechar esta opción, se requeriría volver a entrenar el modelo con ENABLE_GLOBAL_EXPLAIN configurado como TRUE. Aunque no nos adentramos en estas acciones específicas, son herramientas valiosas para aquellos que deseen explorar más allá de la creación inicial del modelo de regresión lineal.

Opción	**Descripción**
MODEL_TYPE	Tipo de modelo.
OPTIMIZE_STRATEGY	Estrategia de optimización.
LEARN_RATE_STRATEGY	Estrategia de tasa de aprendizaje.
LEARN_RATE	Valor de tasa de aprendizaje.
LS_INIT_LEARN_RATE	Tasa de aprendizaje inicial para la búsqueda de línea cuando se está utilizando la estrategia de tasa de aprendizaje LEARN_RATE_STRATEGY.
CALCULATE_P_VALUES	Calcula o no los p-valores.
FIT_INTERCEPT	Indica si se debe ajustar o no el intercepto del modelo.
CATEGORY_ENCODING_METHOD	Método de codificación para variables no numéricas.
CLASS_WEIGHTS	Especifica los pesos de clase.
ENABLE_GLOBAL_EXPLAIN	Habilita o deshabilita la explicación global.
INPUT_LABEL_COLS	Lista de columnas de etiquetas de entrada.
L1_REG	Regularización L1.
L2_REG	Regularización L2.
MAX_ITERATIONS	Número máximo de iteraciones durante el entrenamiento.
WARM_START	Inicia el entrenamiento desde un modelo previamente ajustado.
EARLY_STOP	Habilita o deshabilita la detención temprana durante el entrenamiento.
MIN_REL_PROGRESS	Progreso mínimo relativo para la detención temprana.
DATA_SPLIT_METHOD	Método de división de datos.
DATA_SPLIT_EVAL_FRACTION	Fracción de datos utilizada como conjunto de evaluación.
DATA_SPLIT_TEST_FRACTION	Fracción de datos utilizada como conjunto de prueba.
DATA_SPLIT_COL	Columna utilizada para la división de datos en 'CUSTOM'.
NUM_TRIALS	Número de intentos para la búsqueda de hiperparámetros.
MAX_PARALLEL_TRIALS	Número máximo de intentos paralelos.
HPARAM_TUNING_ALGORITHM	Algoritmo de ajuste de hiperparámetros.
HPARAM_TUNING_OBJECTIVES	Objetivos para la búsqueda de hiperparámetros.

Cuadro 3.3: Configuraciones específicas para el modelo en Bigquery.

3.1.4. Comparación del modelo de regresión lineal en Python y Bigquery

En la comparación entre el modelo de regresión lineal implementado en Python y en BigQuery, se pueden destacar algunas diferencias significativas en términos de simplicidad y facilidad de uso.

En primer lugar, la gestión de datos en BigQuery simplifica notablemente el proceso. Mientras que en Python es necesario realizar manualmente la división de datos en conjuntos de entrenamiento y prueba, en BigQuery esta tarea se puede simplificar notablemente con la opción AUTO_SPLIT, que ajusta automáticamente la división según el tamaño del conjunto de datos. En nuestro caso particular, dicha división se hace mediante la opción CUSTOM, para la cuál simplemente indicamos la columna booleana para la división de los datos. Dicha opción garantiza que las divisiones de datos utilizadas sean exactamente las mismas en Python y en Bigquery ML. Además, su automatización en Bigquery ML elimina la necesidad de escribir código adicional para dividir los datos de manera manual.

Además, en BigQuery ML la codificación de variables no numéricas se realiza de forma predeterminada mediante ONE_HOT_ENCODING. En Python, la codificación one-hot para la columna sexo requiere el uso de la función `get_dummies` de pandas, lo que implica escribir código adicional para crear nuevas columnas binarias. En BigQuery, esta tarea se realiza de manera implícita, simplificando el proceso de preparación de datos. Esto hace que el tiempo de implementación y ejecución de Bigquery sea inferior al de Python. En nuestro ejemplo particular, el Cuadro 3.4 muestra una comparativa entre los tiempos de cálculo de ambos *software*. La tabla revela que BigQuery muestra un tiempo total

Plataforma	**Tiempo de ejecución (segundos)**
Python	3.097
BigQuery	2.550

Cuadro 3.4: Comparación del tiempo total de ejecución entre Python y BigQuery.

de ejecución de 2.55 segundos, mientras que Python tiene un tiempo total de ejecución ligeramente mayor, alcanzando los 3.097 segundos. Esto sugiere que, en este escenario particular, BigQuery puede proporcionar una ejecución más eficiente en términos de tiempo en comparación con Python para el procesamiento de los datos y la ejecución de las operaciones correspondientes.

En cuanto a la obtención de métricas de evaluación del modelo, BigQuery proporciona automáticamente estadísticas de entrenamiento y métricas asociadas al modelo. Por otro lado, en Python, es necesario realizar predicciones en el conjunto de prueba, calcular manualmente métricas como el Error Cuadrático Medio (MSE), R-cuadrado (R^2), Error Absoluto Medio (MAE), entre otras, y luego imprimir la información del modelo. Este proceso puede resultar más laborioso en comparación con la obtención automática de métricas en BigQuery. A continuación, en el Cuadro 3.5, mostramos una comparativa de las métricas obtenidas en ambos casos. Dicha comparativa muestra que las métricas de Python y BigQuery son generalmente similares, con ligeras variaciones en el rendimiento del modelo, como el Error Absoluto Medio (MAE) siendo ligeramente más alto en BigQuery.

En resumen, la integración de BigQuery para la creación y evaluación de modelos de regresión lineal proporciona una sintaxis más sencilla y automatizada en comparación con la implementación en Python. La capacidad de BigQuery para manejar tareas como la división de datos, codificación one-hot y generación automática de métricas simplifica el flujo de trabajo y reduce la cantidad de código necesario para lograr los mismos resultados. Sin embargo, es importante tener en cuenta las limitaciones y consideraciones específicas de cada plataforma al elegir entre Python y BigQuery para un proyecto en particular.

Métrica	Python	BigQuery
Error Cuadrático Medio (MSE)	4.8132	4.8776
R-cuadrado (R^2)	0.5520	0.546
Error Absoluto Medio (MAE)	1.5533	1.5788
Error Logarítmico Cuadrado Medio (MSLE)	0.0316	0.0319
Error Absoluto Mediano (MedAE)	1.1189	1.1711

Cuadro 3.5: Comparación de métricas de evaluación de modelos.

3.2. Modelo de regresión logística

La regresión logística es una herramienta esencial para abordar problemas de clasificación binaria. En contraste con la regresión lineal, la regresión logística se adapta a situaciones en las que la variable dependiente es categórica y presenta dos posibles resultados. Como indicamos en el Capítulo 2, este enfoque utiliza una función específica para transformar una combinación lineal de variables predictoras, generando así probabilidades que se interpretan como la certeza de pertenencia a una clase. En la implementación de la regresión logística en un conjunto de datos concreto, se siguen pasos que abarcan desde la exploración inicial de los datos hasta la evaluación del modelo, ofreciendo una herramienta valiosa para la clasificación.

3.2.1. Descripción del conjunto de datos

Este conjunto de datos proporciona información simulada sobre la calidad del agua en un entorno urbano y está disponible en Kaggle (Kaggle, 2019). En concreto, los datos contienen información detallada sobre diversos atributos del agua, todos expresados como variables numéricas. Cada atributo corresponde a un componente específico, como aluminio, amoníaco, arsénico, bario, cadmio, cloramina, cromo, cobre, fluoruro, bacterias, virus, plomo, nitratos, nitritos, mercurio, perclorato, radio, selenio, plata, y uranio. Cada uno de estos atributos tiene un umbral asociado que indica si su concentración es considerada peligrosa. La columna 'is_safe' es el atributo de clase y toma dos valores: 0 para indicar que el agua no es segura y 1 para indicar que es segura. Dicho conjunto de datos es especialmente apropiado para tareas de clasificación binaria, como la que nos ocupa.

3.2.2. Modelo de regresión logística en Python

El código refleja la implementación de un modelo de regresión logística utilizando Python, específicamente las bibliotecas `pandas`, `scikit-learn` y `google.cloud`.

En primer lugar, el código importa las bibliotecas necesarias, como `pandas` para manipulación de datos, la API de BigQuery para acceder a datos almacenados en BigQuery, y varias funciones relacionadas con el aprendizaje automático de scikit-learn, que nos permitirán tanto crear el modelo como calcular las métricas de evaluación del mismo.

Luego, se inicializa un cliente de BigQuery utilizando las credenciales de autenticación del usuario. Se especifican detalles como el proyecto, conjunto de datos y nombre de la tabla para la consulta de datos. A continuación, se ejecuta una consulta SQL en BigQuery para obtener todos los datos de la tabla especificada, y se carga en un DataFrame de pandas para su posterior manipulación.

Después de cargar los datos, es necesario realizar una limpieza o preprocesado de los datos. En concreto, se reemplazan los valores nulos específicos en la columna 'is_safe' y se convierte esta columna

en categórica, empleando el código indicado en la Figura 3.5. De modo que se añade una categoría

```
# Reemplazar valores #NUM! por NaN en la columna 'is_safe'
df["is_safe"].replace({"#NUM!": None}, inplace=True)

# Convertir la columna 'is_safe' a tipo categórico y agregar una categoría para los valores faltantes
df["is_safe"] = df["is_safe"].astype("category")
df["is_safe"] = df["is_safe"].cat.add_categories(["Missing"])
df["is_safe"].fillna("Missing", inplace=True)
```

Figura 3.5: Código de Python para convertir la columna 'is_safe' a tipo categórico y agregar una categoría para los valores faltantes.

adicional llamada *'Missing'* para los valores faltantes. Además es necesario codificar dichos valores categóricos para que puedan ser utilizados en un modelo de regresión logística. Para ello se convierte la columna categórica a tipo númerico asignando a cada categoría un código numérico único, como muestra la Figura 3.6. De esta forma, ya se puede utilizar dicha columna como variable respuesta de nuestro modelo de regresión logística.

```
# Convertir la columna 'is_safe' a tipo numérico para entrenar el modelo
df["is_safe"] = df["is_safe"].cat.codes
```

Figura 3.6: Código de Python para convertir la columna 'is_safe' a tipo numérico.

A continuación, siguiendo el mismo procedimiento utilizado en la regresión lineal, se procede a la división de los datos en conjuntos de entrenamiento y prueba basándose en la columna 'split'. También será necesario abordar la presencia de valores nulos en el conjunto de variables explicativas, para lo cuál emplearemos un imputador, como se observa en la Figura 3.7. En este caso, se crea una instancia

```
# Crear un imputador que reemplace NaN con la media
imputer = SimpleImputer(strategy="mean")

# Entrenar el imputador con los datos de entrenamiento y transformar los datos
X_train_imputed = imputer.fit_transform(X_train)
X_test_imputed = imputer.transform(X_test)
```

Figura 3.7: Código de Python para imputar los valores faltantes.

de SimpleImputer con la estrategia de imputación establecida como 'mean' (media). Este imputador se utiliza para reemplazar los valores nulos en los conjuntos de entrenamiento y prueba, sustituyéndolos por la media correspondiente de cada columna. Esta estrategia de imputación contribuye a mantener la integridad de los datos y prepara el conjunto para el entrenamiento del modelo de regresión logística, asegurando así un manejo adecuado de los valores faltantes en el proceso de aprendizaje automático. Sin embargo, es importante destacar que la imputación por media es solo una de las estrategias disponibles, se tienen otras opciones que incluyen la eliminación de filas o columnas, la imputación por un valor constante, la mediana, la moda o incluso métodos más avanzados como la imputación basada en modelos predictivos, (Miguel, 2021).

A continuación, se crea un modelo de regresión logística y se entrena con los datos de entrenamiento imputados. Se imprimen el intercepto y los coeficientes del modelo para su análisis, Cuadro 3.4. Estos

Coeficiente	**Valor**
Aluminium	-0.3393
Ammonia	0.0535
Arsenic	2.1644
Barium	0.0271
Cadmium	0.4958
Chloramine	-0.0454
Chromium	-0.7653
Copper	0.2617
Flouride	0.0265
Bacteria	-0.5206
Viruses	0.7799
Lead	0.1849
Nitrates	0.1408
Nitrites	0.2211
Mercury	0.0098
Perchlorate	0.1109
Radium	0.0392
Selenium	0.1018
Silver	0.1297
Uranium	0.2092
Intercepto	**0.7354, -0.5530, -0.1824**

Cuadro 3.6: Coeficientes del modelo de regresión logística.

coeficientes, proporcionan información valiosa sobre la relación entre las características de entrada y la probabilidad de que una observación pertenezca a una clase específica. El intercepto, representado por un vector con tres valores en este caso (0.7354, -0.5530, y -0.1824), establece una probabilidad base para cada clase independientemente de las características observadas. Por otro lado, los coeficientes

asociados a cada característica indican cuánto cambia la probabilidad logarítmica de pertenecer a una clase en respuesta a un cambio unitario en esa característica, manteniendo constantes todas las demás. Un coeficiente positivo sugiere que un aumento en el valor de la característica está asociado con un aumento en la probabilidad de pertenecer a la clase, mientras que un coeficiente negativo indica lo contrario. Por ejemplo, en el caso del aluminio con un coeficiente de -0.3393, un aumento unitario en la concentración de aluminio en el agua implica una disminución en la probabilidad logarítmica de que el agua sea clasificada como segura, siempre que se mantengan constantes todas las demás características. Por otro lado, en el caso del arsénico cuyo coeficiente asociado es 2.1644, a medida que aumenta la concentración de arsénico, es más probable que el agua sea clasificada como segura, siempre y cuando las demás variables permanezcan constantes. Interpretar estos coeficientes permite identificar qué características son más influyentes en la clasificación de las observaciones en las diferentes clases.

Posteriormente, se evalúa el modelo utilizando los datos de prueba imputados y se imprimen diversas métricas de evaluación, Cuadro 3.5, como la precisión, la sensibilidad, la exactitud y el puntaje F1. Además, se calcula la pérdida logarítmica y el área bajo la curva ROC (ROC AUC) utilizando las probabilidades predichas por el modelo y el enfoque utilizado en Bigquery ML. En particular, cuando nos enfrentamos a un problema de clasificación con más de dos clases, el enfoque indicado es *One-vs-Rest* (OvR). Este, trata de convertir el problema inicial en múltiples problemas de clasificación binaria (uno contra el resto). Para cada clase, se entrena un clasificador binario que distingue entre esa clase y todas las demás clases. En resumen, se construye un clasificador por cada clase, y luego se elige la clase con la mayor puntuación o probabilidad entre todos los clasificadores. Lo que implica calcular una curva ROC por cada clase, comparando esa clase con todas las demás agrupadas, (Chmielnicki y Stąpor, 2016). Con lo que finalmente se obtiene un valor de ROC AUC que refleja la capacidad general del modelo para discriminar entre las clases en el problema multiclase. En Python, basta acudir a la función `roc_auc_score` de scikit-learn y añadir un parámetro especificando el método de cálculo del ROC AUC, `multi_class=‘‘ovr’’`, como muestra la Figura 3.8.

```
# Calcula el ROC AUC con método 'ovr' (One-vs-Rest)
roc_auc = roc_auc_score(y_test, predicciones1, multi_class="ovr")
print("ROC AUC:", roc_auc)
```

Figura 3.8: Calculo del ROC AUC con el enfoque *One-vs-Rest* (OvR).

Métrica	**Valor**
Precision	0.5025
Recall	0.4107
Accuracy	0.8949
F1 score	0.4331
Log loss	0.2751
ROC AUC	0.7730

Cuadro 3.7: Métricas de evaluación del modelo de clasificación.

También podemos observar el resultado obtenido para la matriz de confusión, la cuál proporciona

una visión detallada de cómo el modelo ha clasificado las instancias de prueba en comparación con sus clases reales. En esta matriz, las filas representan las clases reales, mientras que las columnas muestran las predicciones del modelo. Cada celda de la matriz indica el número de instancias clasificadas correctamente o incorrectamente para cada clase. Veáse ahora la matriz de confusión correspondiente al modelo evaluado en el Cuadro 3.6. En concreto, dicha matriz muestra que el modelo ha clasificado correctamente la mayoría de las instancias de las clases 0 y 1, como se refleja en los valores altos en la diagonal principal (1376 para la clase 0 y 46 para la clase 1), clasificando bastante peor esta última. Sin embargo, parece haber dificultades importantes en clasificar la clase "Missing", es decir los valores faltantes, ya que todas las instancias de esta clase han sido mal clasificadas como clase 0.

	Predicción Missing	**Predicción 0**	**Predicción 1**
Clase real Missing	0	1	0
Clase real 0	0	1376	31
Clase real 1	0	135	46

Cuadro 3.8: Matriz de confusión del modelo de clasificación.

Volviendo a las métricas obtenidas en el Cuadro 3.5, estas nos ofrecen una visión completa del rendimiento del modelo de clasificación. Como indicamos en el capítulo anterior, la precisión, mide la proporción de predicciones correctas entre las clasificadas como positivas, en este caso es aproximadamente el 50.25 %, lo que sugiere una capacidad razonable del modelo para identificar correctamente las instancias positivas. Por otro lado, la exhaustividad, o *recall*, muestra que el modelo captura alrededor del 41.07 % de todas las instancias positivas, indicando una capacidad moderada para predecir correctamente las clases positivas. La precisión global, que representa la proporción total de predicciones correctas realizadas por el modelo, es del 89.49 %, lo que sugiere un rendimiento generalmente alto en la clasificación de las instancias. La puntuación F1, que como indicamos, combina precisión y sensibilidad en una sola métrica, muestra un equilibrio moderado entre ambas medidas, con un valor de aproximadamente 0.4331. La pérdida logarítmica, que cuantifica qué tan bien el modelo hace sus predicciones, es de aproximadamente 0.2751, indicando un nivel aceptable de confianza en las predicciones del modelo. Finalmente, el área bajo la curva ROC (ROC AUC), que evalúa la capacidad discriminativa del modelo, alcanza aproximadamente 0.7730, lo que sugiere un rendimiento moderado en la capacidad del modelo para distinguir entre las clases positiva y negativa.

En resumen, el modelo de regresión logística muestra una capacidad aceptable para detectar instancias positivas y ofrece un rendimiento generalmente sólido en la clasificación. No obstante, también evidencia necesidades de mejora, especialmente en cuanto a su sensibilidad y capacidad para discriminar entre clases.

3.2.3. Modelo de regresión logística en Bigquery

Al igual, que en el caso de la regresión lineal, un modelo de regresión logística se inicializa con la cláusula CREATE OR REPLACE MODEL, donde se especifica el proyecto, conjunto de datos y nombre del modelo. Luego, en la sección OPTIONS se establecenn las configuraciones específicas, estas coinciden con las indicadas en el Cuadro 3.3.

De nuevo BigQuery realizará la codificación de variables no numéricas de forma predeterminada mediante ONE_HOT_ENCODING. Esta configuración crea columnas binarias para cada categoría, siguiendo una lógica similar a la implementación en Python. En cuanto a la gestión de los datos para el entrenamiento, seguiremos un proceso totalmente análogo al de la Sección 3.1.3.

En definitiva para la creación de un modelo de regresión logítica en Bigquery que nos permita clasificar la calidad del agua aplicamos el código indicado en la Figura 3.9. Con dicho código selec-

```
query = f"""
CREATE OR REPLACE MODEL `{project_id}.{dataset_id}.{model_name}`
OPTIONS(
  model_type='logistic_reg',
  data_split_method='CUSTOM',
  data_split_col='is_test',  # Nombre de la columna de división de datos
  input_label_cols=['is_safe']
) AS
SELECT
  aluminium,
  ammonia,
  arsenic,
  barium,
  cadmium,
  chloramine,
  chromium,
  copper,
  flouride,
  bacteria,
  viruses,
  lead,
  nitrates,
  nitrites,
  mercury,
  perchlorate,
  radium,
  selenium,
  silver,
  uranium,
  is_safe,
  is_test
FROM
  `{project_id}.{dataset_id}.waterquality`
```

Figura 3.9: Sintaxis de Bigquery para la creación del modelo de regresión logística.

cionamos el tipo de modelo indicando 'logistic_reg', obtamos de nuevo por una división de los datos en base a la columna 'is_test' y especificamos la columna asociada variable respuesta, 'is_safe'. En cuánto a las opciones específicas que se podrían indicar, se tiene que estas coinciden con las dadas para el modelo de regresión lineal múltiple, indicadas en el Cuadro 3.3. Podemos destacar, la opción `auto_class_weights`, que de forma predeterminada es FALSE, pero cuando se establece en TRUE, equilibra automáticamente las etiquetas de clase en los datos de entrenamiento. Esto ayuda a contrarrestar el sesgo que puede surgir cuando las etiquetas están desequilibradas. Por ejemplo, si la mayoría de los datos pertenecen a una clase específica, el modelo podría aprender a predecir esa clase con mayor frecuencia. Los pesos de clase equilibran esto asignando pesos más altos a las clases menos frecuentes y pesos más bajos a las clases más comunes, de manera que el modelo no favorezca una clase sobre otra durante el entrenamiento. Además, es relevante mencionar que no es necesario especificar un modelo de regresión logística binaria frente a un modelo de regresión logística de varias clases, puesto que BigQuery puede determinar cuál entrenar según la cantidad de valores únicos en la columna de la etiqueta. Como limitación para este modelo, tendremos que las columnas de etiquetas pueden contener hasta 50 valores únicos; es decir, el número de clases debe ser menor o igual a 50.

Después de crear el modelo, podemos evaluar su rendimiento simplemente observando el panel de información del modelo en la consola de Google Cloud para ver las métricas de evaluación calculadas

durante el entrenamiento, Figura 3.10. Además, en el mismo apartado de evaluación del modelo también

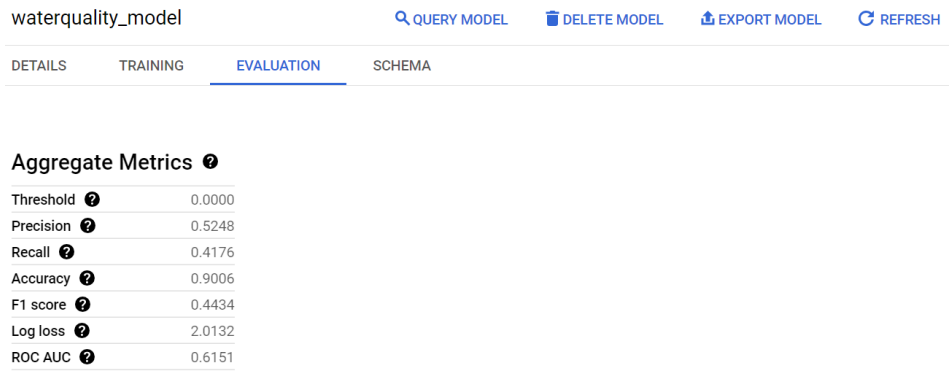


Figura 3.10: Métricas del modelo de regresión logística en Bigquery.

se proporciona la matriz de confusión asociada al modelo, como muestra la Figura 3.11.

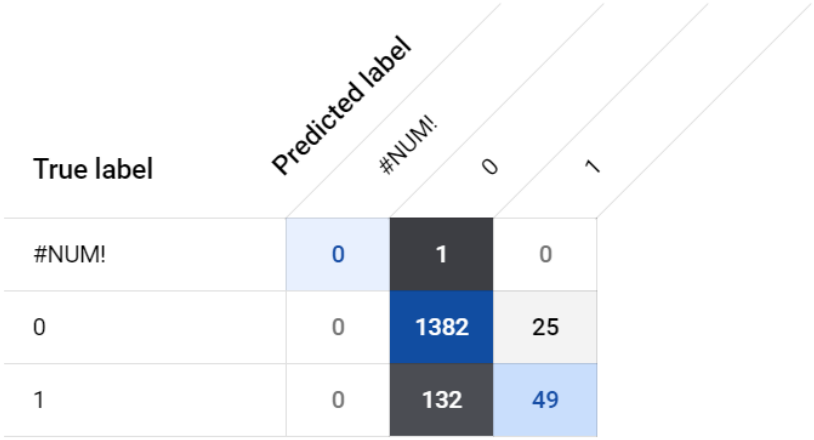


Figura 3.11: Matriz de confusión del modelo de regresión logística en Bigquery.

Estas métricas indican que el modelo de regresión logística tiene una precisión aceptable del 52.48 %, lo que significa que más de la mitad de las predicciones positivas son correctas, mientras que la sensibilidad es del 41.76 %, lo que sugiere que el modelo podría estar perdiendo algunos casos positivos. La exactitud general es alta, alcanzando el 90.06 %. El *F1 Score*, que combina precisión y *recall*, está en 0.4434, señalando un equilibrio moderado entre ambas métricas. Sin embargo, la pérdida logarítmica es de 2.0132, indicando que el modelo podría mejorar en la precisión de sus predicciones. Además, el área bajo la curva ROC es de 0.6151, lo que sugiere un rendimiento moderado en la capacidad de distinguir entre las clases positiva y negativa. Por otro lado, la matriz de confusión dada en la Figura 3.11, proporciona una visión detallada del rendimiento del modelo al clasificar las instancias en sus respectivas clases. En este caso, el modelo predijo correctamente 1382 instancias como agua no potable (clase 0) y 49 instancias como potable (clase 1). Sin embargo, también clasificó incorrectamente 25

instancias no potables como potables y 132 instancias potables como no potables. Esta información muestra que el modelo tiene un sesgo hacia la clasificación de instancias como no potables y podría ser necesaria una mejora para la correcta identificación de los casos potables. En resumen, el modelo muestra una buena precisión general pero podría mejorar en la identificación de casos de agua potable y en la distinción entre clases.

En relación a las limitaciones del uso de Bigquery para este tipo de modelos se tiene que las columnas de etiquetas pueden contener hasta 50 valores únicos; es decir, el número de clases debe ser menor o igual a 50.

3.2.4. Comparación del modelo de regresión logística en Python y Bigquery

En Python, la implementación del modelo de regresión logística implica una serie de pasos adicionales en comparación con BigQuery. Primero, se debe manejar la limpieza de datos, incluyendo el reemplazo de los valores #NUM! por NaN y la conversión de las variables categóricas a numéricas teniendo en cuenta una categoría a mayores para los respectivos valores faltantes. Además, se debe dividir manualmente el conjunto de datos en conjuntos de entrenamiento y prueba. Luego, se realiza el preprocesamiento de los datos, que incluye la imputación de valores faltantes utilizando la media. Después de entrenar el modelo, se calculan una a una las métricas de evaluación indicadas, utilizando funciones específicas de `scikit-learn` para cada una de ellas. Si comparamos las métricas obtenidas en ambos casos, como muestra el Cuadro 3.9, similitudes en el rendimiento general, con algunas diferencias notables. BigQuery muestra una mayor precisión, recall y exactitud, indicando una mejor capacidad para identificar y clasificar correctamente las instancias. Sin embargo, Python exhibe un log loss considerablemente menor y un área bajo la curva ROC más alto, sugiriendo una mejor calibración de las probabilidades de predicción y un rendimiento superior en la clasificación binaria. Estas diferencias resaltan la importancia de considerar las fortalezas y limitaciones de cada plataforma al elegir la adecuada para implementar y evaluar modelos de aprendizaje automático.

Métrica	**Python**	**BigQuery**
Precision	0.5025	0.5248
Recall	0.4107	0.4176
Accuracy	0.8949	0.9006
F1 score	0.4331	0.4434
Log loss	0.2751	2.0132
ROC AUC	0.7730	0.6151

Cuadro 3.9: Comparación de métricas de evaluación del modelo de regresión logística.

Por otro lado, en BigQuery, muchas de las tareas mencionadas anteriormente se simplifican significativamente. BigQuery se encarga automáticamente de manejar los valores faltantes y la codificación de variables categóricas mediante el uso de ONE-HOT-ENCODING. Además, la división de los datos en conjuntos de entrenamiento y prueba se puede realizar de manera mucho más eficiente sin más que filtrar los datos según una columna específica, en lugar de hacerlo manualmente como en Python. Además, BigQuery proporciona las métricas de evaluación del modelo de manera automática después de entrenar el modelo, lo que elimina la necesidad de calcular estas métricas manualmente. En cuánto al

tiempo de ejecución, en este caso como se puede observar en el Cuadro 3.10 , Python es notablemente más rápido que Bigquery, el cuál lleva a cabo 9 iteraciones para obtener el modelo final.

Plataforma	**Tiempo de ejecución (segundos)**
Python	4.546
BigQuery	40.100

Cuadro 3.10: Comparación del tiempo total de ejecución entre Python y BigQuery.

En resumen, aunque ambos enfoques son efectivos para entrenar y evaluar modelos de regresión logística, BigQuery ofrece una ventaja significativa en términos de facilidad de uso y eficiencia al simplificar muchas de las tareas de preprocesamiento de datos y cálculo de métricas de evaluación.

3.3. Regresión con métodos combinados(*ensemble learning*)

En esta sección, nos adentraremos en la aplicación práctica de los métodos combinados de aprendizaje, específicamente, boosting y random forest. Nuestro objetivo es aplicar estas técnicas tanto en BigQuery como en Python utilizando un conjunto de datos previamente preparado. Este conjunto de datos, llamado *abalone* y descripto en detalle en la Sección 3.1.1, nos proporciona la oportunidad de predecir la edad del abulón a partir de diversas medidas físicas.

Después de haber explorado la regresión lineal anteriormente, ahora nos adentraremos en modelos más avanzados. Como sabemos por la Sección 2.3, estos modelos, aprovechan la información de múltiples estimadores y han demostrado su eficacia en una gran variedad de problemas de predicción y clasificación. En particular, compararemos la implementación de boosting y random forest tanto en BigQuery como en Python. No solo nos enfocaremos en la precisión de las predicciones, sino también en la facilidad de uso y las ventajas particulares que ofrece cada plataforma. A través de este análisis comparativo, esperamos proporcionar una visión más completa de las capacidades de estos métodos combinados de aprendizaje y su aplicabilidad en problemas de regresión como el que abordamos con el conjunto de datos *abalone*.

3.3.1. Aplicación del método boosting para regresión en Python

Al igual que ejemplos anteriores, comenzaremos importando las bibliotecas necesarias. Se utiliza `matplotlib.pyplot` para la visualización de datos, `pandas` para manipulación de datos, `google.cloud` para interactuar con BigQuery, `GradientBoostingRegressor` de `scikit-learn` para implementar el modelo de regresión con Boosting, y varias métricas de evaluación de modelos de regresión de `scikit-learn.metrics` (Pedregosa y cols., 2011).

Luego, se inicializa el cliente de BigQuery y se especifica la información del proyecto, conjunto de datos y tabla que se utilizarán en la consulta. Se ejecuta una consulta en BigQuery para recuperar todos los datos de la tabla especificada, y se almacenan en un DataFrame de `pandas`. Posteriormente, se realizan ciertas manipulaciones de los datos. Es importante destacar que al igual que en el caso de la regresión lineal, se dividen los datos en conjuntos de entrenamiento (TRAIN) y prueba (TEST) filtrando los datos según la columna split. Se separan las características (X_train, X_test) y la variable objetivo (y_train, y_test). Se lleva a cabo una codificación one-hot para la columna sex utilizando la función `get_dummies` de `pandas`. Esto crea nuevas columnas (sex_M, sex_F, sex_I) con valores binarios para cada categoría, asegurando que el modelo pueda interpretar correctamente la información de la columna sexo durante el análisis y entrenamiento. Esto resulta totalmente análogo a lo visto en la sección 3.1.2.

A continuación, se inicializa el modelo de regresión con Boosting (GradientBoostingRegressor) y se entrena el modelo con los datos de entrenamiento. Después de entrenar el modelo, se realizan predicciones en el conjunto de prueba y se calculan varias métricas de rendimiento del modelo, como el Error Cuadrático Medio (MSE), el Coeficiente de Determinación (R^2), el Error Absoluto Medio (MAE), el Error Logarítmico Cuadrático Medio (MSLE) y el Error Absoluto Mediano (MedAE). Finalmente, se imprimen las métricas de rendimiento calculadas como muestra el Cuadro 3.7 y en la Figura 3.12 se visualizan los valores originales y predichos mediante un gráfico de dispersión utilizando `matplotlib`,.

Métrica	**Valor**
Error Cuadrático Medio (MSE)	4.6977
Coeficiente de determinación (R^2)	0.5628
Error Absoluto Medio (MAE)	1.4960
Error Logarítmico Cuadrado Medio (MSLE)	0.0293
Error Absoluto Mediano (MedAE)	1.0694

Cuadro 3.11: Métricas de evaluación del modelo de regresión con Boosting.

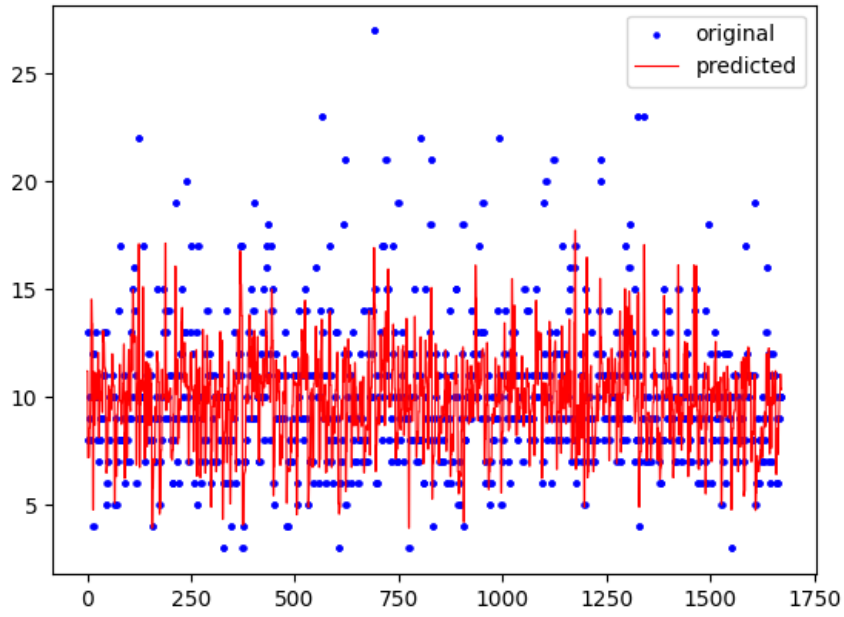


Figura 3.12: Gráfico de dispersión de valores predichos y originales utilizando `matplotlib`.

Si observamos las métricas de evaluación del modelo de regresión con Boosting, Cuadro 3.7, mues-

tran un Error Cuadrático Medio (MSE) de aproximadamente 4.6977, lo que indica la magnitud promedio de los errores al cuadrado. El coeficiente de determinación (R^2) es de alrededor de 0.5628, lo que significa que aproximadamente el 56.28 % de la variabilidad en la variable de respuesta es explicada por el modelo. El Error Absoluto Medio (MAE) es de aproximadamente 1.4960, proporcionando una medida de la precisión promedio del modelo en términos de unidades de la variable de respuesta. El Error Logarítmico Cuadrado Medio (MSLE) es de alrededor de 0.0293, penalizando los errores más grandes de manera más significativa que los errores más pequeños. Finalmente, el Error Absoluto Mediano (MedAE) es de aproximadamente 1.0694, ofreciendo una medida de la precisión del modelo menos sensible a los valores atípicos en los datos. Estas métricas proporcionan una evaluación cuantitativa del rendimiento del modelo en la predicción de la edad de los abulones, abordando diferentes aspectos de la calidad del ajuste y la capacidad predictiva del modelo. Como se puede observar estos valores son bastante similares a los obtenidos para la regresión lineal en el cuadro 3.2, siendo en este caso el coeficiente de determinación ligeramente mayor.

En resumen, con nuestro script de Python reflejamos cómo implementar y evaluar un modelo de regresión con Boosting en utilizando datos almacenados en BigQuery, y destacamos la importancia de dividir los datos manualmente y transformar las variables categóricas a tipo dummies para el correcto funcionamiento del modelo.

3.3.2. Aplicación del método boosting para regresión en Bigquery

En esta sección, se presenta la aplicación del método Boosting para regresión. En Bigquery, un árbol con boosting se utiliza para crear modelos de clasificación o regresión basados en XGBoost, estos modelos son externos a BigQuery ML y están entrenados en Vertex AI, una plataforma de aprendizaje automático que permite entrenar y implementar modelos . La plataforma permite a los desarrolladores y científicos de datos centrarse más en sus proyectos y menos en la infraestructura, pues ayuda a reducir el tiempo de entrenamiento y a implementar modelos en producción con facilidad mediante los frameworks de código abierto que elijas y la infraestructura de IA optimizada. En concreto, nuestros modelos de árbol impulsados se entrenan utilizando la biblioteca XGBoost, que permite el ajuste de hiperparámetros para mejorar el rendimiento del modelo. XGBoost es una biblioteca optimizada de aumento de gradiente distribuido diseñada para ser altamente eficiente, flexible y portátil. Implementa algoritmos de aprendizaje automático bajo el marco de Gradient Boosting. XGBoost proporciona un impulso de árbol paralelo (también conocido como GBDT, GBM) que resuelve muchos problemas de ciencia de datos de forma rápida y precisa, (*XGBoost Documentation xgboost 2.1.0-dev documentation*, s.f.). Este método de regresión basado en árboles impulsados ofrece flexibilidad y rendimiento, permitiendo el ajuste de hiperparámetros para optimizar la precisión del modelo.

Para implementar el método boosting, se presentan unas tablas resumen, detallando los posibles parámetros que ofrece Bigquery, Cuadro 3.12 y Cuadro 3.13. En particular, para la creación del modelo que nos ocupa, ver Figura, se utiliza el método asociado a la regresión con boosting. Además, como en modelos anteriores, personalizamos la división de datos utilizando el método CUSTOM, donde la columna is_test fue designada para esta función y establecemos la variable objetivo como rings. Los demás parámetros no mencionados en esta consulta se consideran por defecto según las configuraciones predeterminadas de BigQuery ML. Por ejemplo, para el parámetro ENABLE_GLOBAL_EXPLAIN, determina si se deben calcular explicaciones globales utilizando inteligencia artificial explicativa para evaluar la importancia de las características globales para el modelo, el valor por defecto es FALSE, lo que significa que no se calculan explicaciones globales automáticamente a menos que se especifique lo contrario. Otro ejemplo de parámetro es AUTO_CLASS_WEIGHTS, que determina si se deben equilibrar las etiquetas de clase utilizando pesos para cada clase en proporción inversa a la frecuencia de esa clase. El valor predeterminado es FALSE, lo que significa que el conjunto de datos de entrenamiento no está ponderado automáticamente en función de la frecuencia de las clases. También cabe destacar, que al igual que en los ejemplos anteriores Bigquery lleva a cabo la codificación automática de las variables no numéricas mediante el ONE_HOT_ENCODING mencionado en secciones previas.

```
query = f"""
CREATE OR REPLACE MODEL `{project_id}.{dataset_id}.{model_name}`
OPTIONS(
  model_type='BOOSTED_TREE_REGRESSOR',
  data_split_method='CUSTOM',
  data_split_col='is_test',  # Nombre de la columna de división de datos
  input_label_cols=['rings']
) AS
SELECT
  sex,
  length,
  diameter,
  height,
  whole_weight,
  shucked_weight,
  viscera_weight,
  shell_weight,
  rings,
  is_test
FROM
  `{project_id}.{dataset_id}.abalone`
"""
```

Figura 3.13: Sintaxis de Bigquery para la creación del modelo de regresión con boosting.

Opción	**Descripción**
MODEL_TYPE	Tipo de modelo.
CATEGORY_ENCODING_METHOD	Método de codificación para características no numéricas.
BOOSTER_TYPE	Tipo de impulsor a usar.
DART_NORMALIZE_TYPE	Tipo de algoritmo de normalización si se está utilizando el impulsor DART.
NUM_PARALLEL_TREE	Número de árboles paralelos construidos durante cada iteración.
MAX_TREE_DEPTH	Profundidad máxima de un árbol.
DROPOUT	Tasa de abandono, fracción de árboles anteriores a dejar durante el abandono.
L1_REG	Cantidad de regularización L1 aplicada.
L2_REG	Cantidad de regularización L2 aplicada.

Cuadro 3.12: Resumen de opciones del modelo XGBoost en BigQuery ML (Parte 1)

Opción	**Descripción**
LEARN_RATE	Tamaño del paso de reducción.
TREE_METHOD	Tipo de algoritmo de construcción de árboles.
MIN_TREE_CHILD_WEIGHT	Suma mínima de pesos para particionar.
COLSAMPLE_BYTREE	Proporción de submuestra al construir cada árbol.
COLSAMPLE_BYLEVEL	Proporción de submuestra para cada nivel.
COLSAMPLE_BYNODE	Proporción de submuestra para cada nodo.
MIN_SPLIT_LOSS	Pérdida mínima para hacer una partición adicional en un nodo hoja.
SUBSAMPLE	Proporción de submuestra de las instancias de entrenamiento.
INSTANCE_WEIGHT_COL	Especificar los pesos.
XGBOOST_VERSION	Versión de XGBoost para el entrenamiento.
AUTO_CLASS_WEIGHTS	Determina si equilibrar o no las etiquetas de clase.
CLASS_WEIGHTS	Pesos a usar para cada etiqueta de clase.
ENABLE_GLOBAL_EXPLAIN	Determina si calcular o no explicaciones globales.
EARLY_STOP	Decide si detener el entrenamiento basándose en la mejora de la pérdida relativa.
MIN_REL_PROGRESS	Mejora mínima relativa de la pérdida necesaria para continuar cuando EARLY_STOP está configurado en TRUE.
INPUT_LABEL_COLS	El nombre de la columna de etiqueta en los datos de entrenamiento.
MAX_ITERATIONS	Número máximo de rondas para el aumento.
DATA_SPLIT_METHOD	Método utilizado para dividir los datos de entrada.
DATA_SPLIT_EVAL_FRACTION	Fracción de los datos a utilizar como evaluación.
DATA_SPLIT_TEST_FRACTION	Fracción de los datos a utilizar como prueba.
DATA_SPLIT_COL	Nombre de la columna a utilizar para ordenar los datos de entrada.
NUM_TRIALS	Número máximo de submodelos a entrenar.
MAX_PARALLEL_TRIALS	Número máximo de ensayos que se ejecutarán al mismo tiempo.
HPARAM_TUNING_ALGORITHM	El algoritmo utilizado para ajustar los hiperparámetros.
HPARAM_TUNING_OBJECTIVES	El objetivo de ajuste de hiperparámetros para el modelo.

Cuadro 3.13: Resumen de opciones del modelo XGBoost en BigQuery ML (Parte 2)

Una vez implementado el modelo, podemos observar el panel de evaluación del mismo y analizar las métricas obtenidas, Figura 3.14.

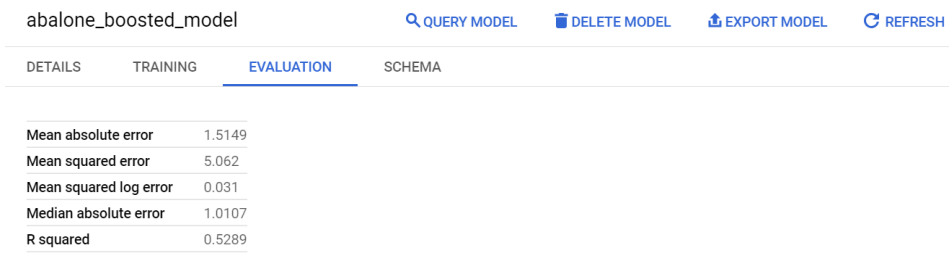


Figura 3.14: Métricas del modelo de regresión con boosting en Bigquery.

En cuánto a los valores obtenidos, el Mean Absolute Error (MAE) es de 1.5149 lo que indica que, en promedio, las predicciones difieren de los valores reales en aproximadamente 1.5149 unidades, mientras que el Mean Squared Error (MSE) de 5.062 muestra una dispersión de errores más amplia. El Mean Squared Log Error (MSLE) es de 0.031 y el Median Absolute Error de 1.0107 proporciona una medida adicional de dispersión de errores, mostrando que el 50 % de los errores están por encima de este valor y el 50 % por debajo. Finalmente, el R-squared (R^2) de 0.5289 indica que aproximadamente el 52.89 % de la variabilidad en los datos puede ser explicada por el modelo, lo que sugiere un ajuste moderado pero significativo del modelo a los datos.

En relación a las limitaciones del uso de Bigquery para este tipo de modelos, cabe destacar que al ser externo a BigQuery ML y estar entrenado en Vertex AI no se puede realizar una prueba de validación en las declaraciones CREATE MODEL para obtener una estimación de la cantidad de datos que procesarán si los ejecutas. Esto implica que no se puede obtener una estimación precisa de la cantidad de datos que procesará el modelo si se ejecuta en BigQuery. Esta restricción puede afectar a la capacidad de los usuarios para evaluar adecuadamente el rendimiento del modelo en términos de eficiencia y recursos necesarios para su implementación. Por lo tanto, se deberá tener en cuenta esta limitación al trabajar con modelos externos en BigQuery y considerar alternativas para evaluar y ajustar el rendimiento del modelo de manera efectiva.

3.3.3. Comparación del método boosting para regresión en Python y Bigquery

En el caso de Python, se emplea la biblioteca scikit-learn para implementar el modelo GradientBoostingRegressor, lo que ofrece flexibilidad y control sobre el proceso de entrenamiento y evaluación del modelo. Se deben realizan manipulaciones de datos, como la división en conjuntos de entrenamiento y prueba, y la codificación one-hot para las variables categóricas. Por otro lado, en el caso de BigQuery, se utiliza un modelo externo entrenado en Vertex AI, lo que ofrece facilidades para el entrenamiento y la implementación de modelos en producción, pero limita la capacidad de realizar pruebas de validación durante la creación del modelo en BigQuery ML. Esto significa que no se puede obtener una estimación precisa de la cantidad de datos que procesará el modelo al ejecutarse en BigQuery, lo que puede afectar la capacidad de los usuarios para evaluar el rendimiento del modelo en términos de eficiencia y recursos necesarios.

Si observamos las métricas obtenidas en ambos casos, el Cuadro 3.14, muestra una comparativa de las métricas de evaluación del modelo de regresión con Boosting en Python y en BigQuery. Se observa que las métricas son bastante similares en ambas plataformas, lo que sugiere un rendimiento comparable del modelo en términos de precisión y capacidad predictiva. Sin embargo, existen pequeñas

Métrica	Python	BigQuery
Mean Absolute Error (MAE)	1.4960	1.5149
Mean Squared Error (MSE)	4.6977	5.0620
Mean Squared Log Error (MSLE)	0.0293	0.0310
Median Absolute Error (MedAE)	1.0694	1.0107
R-squared (R^2)	0.5628	0.5289

Cuadro 3.14: Comparación de métricas de evaluación del modelo de regresión con Boosting en Python y BigQuery.

diferencias en algunas métricas, como el MAE y el MSE, que podrían atribuirse a variaciones en la implementación o en la configuración de los modelos en cada plataforma.

Al igual, que para los modelos anteriores podemos comparar los tiempos de ejecución de ambas plataformas. El Cuadro 3.15 muestra una comparación de los tiempos de ejecución entre Python y BigQuery para el mismo proceso de modelado y evaluación. Se observa que BigQuery logra una ejecución más rápida con un tiempo total de 111.02 segundos, mientras que Python requiere significativamente más tiempo con 363.23 segundos. Esto resalta la eficiencia de BigQuery para procesar y analizar grandes conjuntos de datos en comparación con la implementación local en Python.

Plataforma	Tiempo de ejecución (segundos)
Python	363.23
BigQuery	111.02

Cuadro 3.15: Comparación del tiempo total de ejecución entre Python y BigQuery.

En conclusión, la comparación entre la implementación del método de Boosting para regresión en Python y en BigQuery revela un rendimiento similar en términos de métricas de evaluación. Sin embargo, es importante tener en cuenta las diferencias en el proceso de desarrollo y ejecución de los modelos. Mientras que Python brinda mayor flexibilidad y control sobre el proceso de modelado y evaluación, requiere más esfuerzo en términos de manipulación de datos y configuración del entorno. Por otro lado, BigQuery ofrece una solución integrada y escalable para el procesamiento de grandes conjuntos de datos, lo que resulta en tiempos de ejecución más rápidos y una mayor facilidad de implementación en entornos de producción.

3.3.4. Aplicación del método *Random Forest* para regresión en Python

Al igual que en ejemplos anteriores, comenzaremos importando las bibliotecas necesarias. Se utilizará `matplotlib.pyplot` para la visualización de datos, `pandas` para la manipulación de datos, `google.cloud` para interactuar con BigQuery, `RandomForestRegressor` de `scikit-learn` para implementar el modelo de regresión con Random Forest, y varias métricas de evaluación de modelos de regresión de `scikit-learn.metrics` (Pedregosa y cols., 2011).

Luego, se inicializa el cliente de BigQuery y se especifica la información del proyecto, conjunto de

datos y tabla que se utilizarán en la consulta. Se ejecuta una consulta en BigQuery para recuperar todos los datos de la tabla especificada, y se almacenan en un DataFrame de `pandas`. Posteriormente, se realizan ciertas manipulaciones de los datos. Es importante destacar que al igual que en el caso de la regresión lineal, se dividen los datos en conjuntos de entrenamiento (TRAIN) y prueba (TEST) filtrando los datos según la columna split. Se separan las características (`X_train`, `X_test`) y la variable objetivo (`y_train`, `y_test`). Se lleva a cabo una codificación one-hot para la columna sex utilizando la función `get_dummies` de `pandas`. Esto crea nuevas columnas (`sex_M`, `sex_F`, `sex_I`) con valores binarios para cada categoría, asegurando que el modelo pueda interpretar correctamente la información de la columna sexo durante el análisis y entrenamiento. Esto resulta totalmente análogo a lo visto en la sección 3.1.2.

A continuación, se inicializa el modelo de regresión con Random Forest (`RandomForestRegressor`) y se entrena el modelo con los datos de entrenamiento. Después de entrenar el modelo, se realizan predicciones en el conjunto de prueba y se calculan varias métricas de rendimiento del modelo, como el Error Cuadrático Medio (MSE), el Coeficiente de Determinación (R^2), el Error Absoluto Medio (MAE), el Error Logarítmico Cuadrático Medio (MSLE) y el Error Absoluto Mediano (MedAE). Finalmente, se imprimen las métricas de rendimiento calculadas como muestra el Cuadro 3.16 y en la Figura 3.15 se visualizan los valores originales y predichos mediante un gráfico de dispersión utilizando `matplotlib`.

Métrica	**Valor**
Error Cuadrático Medio (MSE)	5.3206
Coeficiente de determinación (R^2)	0.5048
Error Absoluto Medio (MAE)	1.6132
Error Logarítmico Cuadrado Medio (MSLE)	0.0333
Error Absoluto Mediano (MedAE)	1.1200

Cuadro 3.16: Métricas de evaluación del modelo de regresión con Random Forest.

Si observamos las métricas de evaluación del modelo de regresión con Random Forest, Cuadro 3.16, muestran un Error Cuadrático Medio (MSE) de aproximadamente 5.3206, lo que indica la magnitud promedio de los errores al cuadrado. El coeficiente de determinación (R^2) es de alrededor de 0.5048, lo que significa que aproximadamente el 50.48 % de la variabilidad en la variable de respuesta es explicada por el modelo. El Error Absoluto Medio (MAE) es de aproximadamente 1.6132, proporcionando una medida de la precisión promedio del modelo en términos de unidades de la variable de respuesta. El Error Logarítmico Cuadrado Medio (MSLE) es de alrededor de 0.0333, penalizando los errores más grandes de manera más significativa que los errores más pequeños. Finalmente, el Error Absoluto Mediano (MedAE) es de aproximadamente 1.12, ofreciendo una medida de la precisión del modelo menos sensible a los valores atípicos en los datos. Estas métricas proporcionan una evaluación cuantitativa del rendimiento del modelo en la predicción de la edad de los abulones, abordando diferentes aspectos de la calidad del ajuste y la capacidad predictiva del modelo. Como se puede observar, estos valores son comparables a los obtenidos para la regresión lineal, siendo en este caso el coeficiente de determinación ligeramente menor.

En resumen, con nuestro script de Python reflejamos cómo implementar y evaluar un modelo de regresión con Random Forest en Python utilizando datos almacenados en BigQuery, y destacamos la importancia de dividir los datos manualmente y transformar las variables categóricas a tipo dummies para el correcto funcionamiento del modelo.

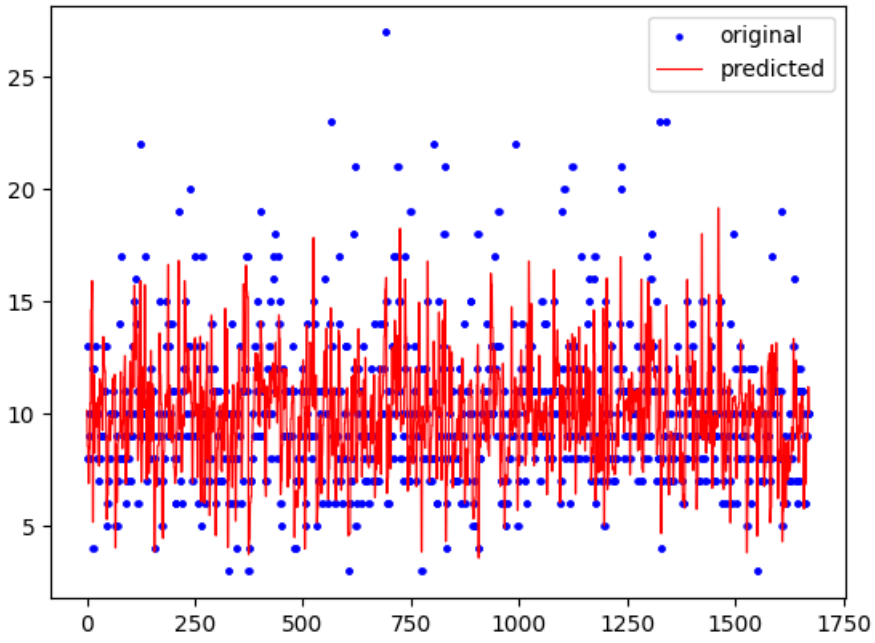


Figura 3.15: Gráfico de dispersión de valores predichos y originales utilizando `matplotlib`.

3.3.5. Aplicación del método *Random Forest* para regresión en Bigquery

En esta sección, exploraremos la aplicación del método de bosque aleatorio para regresión utilizando BigQuery. Similar al método de boosting, para construir modelos de bosque aleatorio en BigQuery se requiere el uso de la biblioteca XGBoost, como se menciona en (*XGBoost Documentation xgboost 2.1.0-dev documentation*, s.f.). Estos modelos se encuentran fuera del alcance de BigQuery ML y están entrenados en Vertex AI.

Al igual que en secciones anteriores, presentaremos una tabla que resume las configuraciones específicas para el modelo en BigQuery. Estas configuraciones abarcan una variedad de opciones que afectan el comportamiento y el rendimiento del modelo de bosque aleatorio. A continuación, se muestra la tabla dividida en dos partes, detallando cada opción y su descripción correspondiente.

Opción	**Descripción**
MODEL_TYPE	Tipo de modelo.
APPROX_GLOBAL_FEATURE	Aproximación para los cálculos de contribución de características.
NUM_PARALLEL_TREE	Número de árboles paralelos construidos durante cada iteración.
MAX_TREE_DEPTH	Profundidad máxima.
L1_REG	Regularización L1.
L2_REG	Regularización L2.

Cuadro 3.17: Configuraciones específicas para el modelo en Bigquery (Parte 1).

Opción	**Descripción**
TREE_METHOD	Tipo de algoritmo de construcción de árbol.
MIN_TREE_CHILD_WEIGHT	Suma mínima de pesos para particionar.
COLSAMPLE_BYTREE	Proporción de submuestra por árbol.
COLSAMPLE_BYLEVEL	Proporción de submuestra por nivel.
COLSAMPLE_BYNODE	Proporción de submuestra por nodo.
MIN_SPLIT_LOSS	Pérdida mínima para hacer una partición adicional en un nodo hoja.
SUBSAMPLE	Proporción de submuestra de entrenamiento.
INSTANCE_WEIGHT_COL	Pesos en el conjunto de datos de entrenamiento..
XGBOOST_VERSION	La versión XGBoost para entrenamiento.
AUTO_CLASS_WEIGHTS	Determina si equilibrar o no las etiquetas de clase.
CLASS_WEIGHTS	Pesos a usar para cada etiqueta de clase.
ENABLE_GLOBAL_EXPLAIN	Determina si calcular o no explicaciones globales.
EARLY_STOP	Decide si detener el entrenamiento basándose en la mejora de la pérdida relativa.
MIN_REL_PROGRESS	Mejora mínima relativa de la pérdida necesaria para continuar cuando EARLY_STOP está configurado en TRUE.
INPUT_LABEL_COLS	El nombre de la columna de etiqueta en los datos de entrenamiento.
DATA_SPLIT_METHOD	Método utilizado para dividir los datos de entrada.
DATA_SPLIT_EVAL_FRACTION	Fracción de los datos a utilizar como evaluación.
DATA_SPLIT_TEST_FRACTION	Fracción de los datos a utilizar como prueba.
DATA_SPLIT_COL	Nombre de la columna a utilizar para ordenar los datos de entrada.
NUM_TRIALS	Número máximo de submodelos a entrenar.
MAX_PARALLEL_TRIALS	Número máximo de ensayos que se ejecutarán al mismo tiempo.
HPARAM_TUNING_ALGORITHM	El algoritmo utilizado para ajustar los hiperparámetros.
HPARAM_TUNING_OBJECTIVES	El objetivo de ajuste de hiperparámetros para el modelo.

Cuadro 3.18: Configuraciones específicas para el modelo en Bigquery (Parte 2).

Una vez revisadas las configuraciones específicas del modelo, procedemos a implementar el modelo de bosque aleatorio en BigQuery. A continuación, en la Figura 3.16, se presenta la sintaxis de la consulta de BigQuery para la creación del modelo de regresión con bosque aleatorio. La consulta anterior

```
query = f"""
CREATE OR REPLACE MODEL `{project_id}.{dataset_id}.{model_name}`
OPTIONS(
  model_type='RANDOM_FOREST_REGRESSOR',  -- Seleccionamos BOOSTED_TREE_REGRESSOR para simular un Random Forest
  data_split_method='CUSTOM',
  data_split_col='is_test',  -- Nombre de la columna de división de datos
  input_label_cols=['rings']
) AS
SELECT
  sex,
  length,
  diameter,
  height,
  whole_weight,
  shucked_weight,
  viscera_weight,
  shell_weight,
  rings,
  is_test
FROM
  `{project_id}.{dataset_id}.abalone`
"""
```

Figura 3.16: Sintaxis de Bigquery para la creación del modelo de regresión con random forest.

define las características del modelo y especifica los parámetros necesarios para su entrenamiento. Una vez ejecutada la consulta y entrenado el modelo, procedemos a evaluar su desempeño mediante diversas métricas de evaluación, Figura 3.17. En cuanto a los valores obtenidos, el Mean Absolute Error

Mean absolute error	1.5363
Mean squared error	5.0514
Mean squared log error	0.0311
Median absolute error	1.06
R squared	0.5299

Figura 3.17: Métricas del modelo de regresión con random forest en Bigquery.

(MAE) es de 1.5363, lo que indica que, en promedio, las predicciones difieren de los valores reales en aproximadamente 1.5363 unidades. El Mean Squared Error (MSE) de 5.0514 muestra una dispersión de errores más amplia. El Mean Squared Log Error (MSLE) es de 0.0311 y el Median Absolute Error de 1.06 proporciona una medida adicional de dispersión de errores, mostrando que el 50 % de los errores están por encima de este valor y el 50 % por debajo. Finalmente, el R-squared (R^2) de 0.5299 indica que aproximadamente el 52.99 % de la variabilidad en los datos puede ser explicada por el modelo, lo que sugiere un ajuste moderado pero significativo del modelo a los datos.

Estas métricas nos brindan una comprensión detallada del rendimiento del modelo de regresión con bosque aleatorio en BigQuery, permitiéndonos evaluar su precisión y capacidad para generalizar a partir de los datos de entrenamiento.

3.3.6. Comparación del método Random Forest para regresión en Python y BigQuery

En el caso de Python, se emplea la biblioteca scikit-learn para implementar el modelo RandomForestRegressor, lo que ofrece flexibilidad y control sobre el proceso de entrenamiento y evaluación del modelo. Se deben realizar manipulaciones de datos, como la división en conjuntos de entrenamiento y prueba, y la codificación one-hot para las variables categóricas.

Por otro lado, en el caso de BigQuery, se utiliza un modelo externo entrenado en Vertex AI, lo que ofrece facilidades para el entrenamiento y la implementación de modelos en producción, pero limita la capacidad de realizar pruebas de validación durante la creación del modelo en BigQuery ML. Esto significa que no se puede obtener una estimación precisa de la cantidad de datos que procesará el modelo al ejecutarse en BigQuery, lo que puede afectar la capacidad de los usuarios para evaluar el rendimiento del modelo en términos de eficiencia y recursos necesarios.

Métrica	**Python**	**BigQuery**
Mean Absolute Error (MAE)	1.6132	1.5363
Mean Squared Error (MSE)	5.3206	5.0514
Mean Squared Log Error (MSLE)	0.0333	0.0311
Median Absolute Error (MedAE)	1.1200	1.0600
R-squared (R^2)	0.5048	0.5299

Cuadro 3.19: Comparación de métricas de evaluación del modelo de regresión con Random Forest en Python y BigQuery.

Si observamos las métricas obtenidas en ambos casos, el Cuadro 3.19 muestra una comparativa de las métricas de evaluación del modelo de regresión con Random Forest en Python y en BigQuery. Se observa que las métricas son bastante similares en ambas plataformas, lo que sugiere un rendimiento comparable del modelo en términos de precisión y capacidad predictiva. Sin embargo, existen pequeñas diferencias en algunas métricas, como el MAE y el MSE, que podrían atribuirse a variaciones en la implementación o en la configuración de los modelos en cada plataforma.

Al igual que para los modelos anteriores, podemos comparar los tiempos de ejecución de ambas plataformas. La tabla muestra una comparación de los tiempos de ejecución entre Python y BigQuery para el mismo proceso de modelado y evaluación. Los tiempos de ejecución obtenidos revelan una diferencia significativa entre la implementación de Random Forest en Python y en BigQuery. Mientras que Python muestra un rendimiento notablemente más rápido, con un tiempo total de ejecución de aproximadamente 5.08 segundos, la ejecución en BigQuery requiere considerablemente más tiempo, con alrededor de 92.73 segundos. Esta disparidad puede influir en la elección de la plataforma dependiendo de los requisitos de velocidad de procesamiento y la escalabilidad del proyecto. Por ejemplo, para conjuntos de datos pequeños o análisis rápidos, Python puede ser preferible debido a su eficiencia en términos de tiempo. Sin embargo, para conjuntos de datos masivos y necesidades de procesamiento a gran escala, la infraestructura escalable y distribuida de BigQuery ofrece una solución adecuada, a pesar de los tiempos de ejecución más largos. En conclusión, la comparación entre la implementación del método de Random Forest para regresión en Python y en BigQuery revela un rendimiento similar en términos de métricas de evaluación. Sin embargo, en términos de tiempos de ejecución, se observa que la implementación en Python es significativamente más rápida. Esta diferencia puede ser importante

Plataforma	Tiempo de ejecución (segundos)
Python	5.0804
BigQuery	92.73

Cuadro 3.20: Comparación del tiempo total de ejecución entre Python y BigQuery para Random Forest.

dependiendo de las necesidades de velocidad de procesamiento y el tamaño del conjunto de datos a analizar.

3.4. Clasificación con métodos combinados(*ensemble learning*)

En esta sección, nos adentraremos en el análisis de clasificación utilizando un conjunto de datos sobre la calidad del agua en entornos urbanos (Kaggle, 2019). Este conjunto de datos contiene una variedad de características ambientales y químicas que nos permiten explorar diferentes técnicas de clasificación para categorizar la calidad del agua. Después de haber cubierto los fundamentos de la regresión logística, ahora nos expandiremos hacia métodos más avanzados de clasificación, como boosting y random forest. A lo largo de esta sección, exploraremos cómo estos algoritmos pueden aplicarse para comprender y predecir la calidad del agua, evaluando su precisión y capacidad para generalizar en diferentes escenarios.

3.4.1. Aplicación del método boosting para clasificación en Python

Comenzamos importando las bibliotecas necesarias, que incluyen `matplotlib.pyplot` para visualización de datos, `pandas` para manipulación de datos, `google.cloud` para interactuar con BigQuery y `GradientBoostingClassifier` de `scikit-learn` para implementar el modelo de clasificación con boosting. Además, importamos varias métricas de evaluación de modelos de clasificación de `scikit-learn.metrics`, (Pedregosa y cols., 2011).

Al igual que en ocasiones anteriores debemos realizar ciertas manipulaciones de los datos, como la limpieza de valores nulos y la codificación de variables categóricas. Después dicha preparación de datos, dividimos el conjunto en características (X) y la variable objetivo (y), y filtramos los datos en conjuntos de entrenamiento (TRAIN) y prueba (TEST) según la columna de división (*split*).

A continuación, creamos un modelo de clasificación con boosting (`GradientBoostingClassifier`) y lo entrenamos con los datos de entrenamiento. Una vez entrenado el modelo, realizamos predicciones en el conjunto de prueba y calculamos varias métricas de rendimiento del modelo junto con la matriz de confusión, como la precisión, la sensibilidad y la pérdida logarítmica, tal y como se muestra en e los Cuadros 3.21 y 3.22.

En general, el modelo de clasificación con boosting ha demostrado un buen rendimiento en la tarea de clasificación. Con una precisión de 0.9583 y una sensibilidad promedio de 0.5798, el modelo logra clasificar correctamente la mayoría de las muestras. Además, el alto valor de Accuracy (0.9604) indica que el modelo es capaz de predecir con precisión las etiquetas de clase. Sin embargo, es importante tener en cuenta que algunas clases pueden estar desbalanceadas, lo que puede afectar la precisión promedio y la sensibilidad promedio del modelo. La matriz de confusión, Cuadro 3.22, revela cómo el modelo de clasificación con boosting ha realizado las predicciones en relación con las clases reales. Destaca que el modelo logró identificar correctamente la gran mayoría de las muestras de las clases 0 y 1, con 1390 y 136 predicciones correctas respectivamente. Sin embargo, muestra cierta tendencia a

Métrica	Valor
Precisión	0.9583
Precisión promedio ('macro')	0.6190
Sensibilidad promedio ('macro')	0.5798
Accuracy	0.9604
Puntaje F1	0.5974
Pérdida logarítmica	0.1169
ROC AUC	0.7880

Cuadro 3.21: Métricas de evaluación del modelo de clasificación.

	Predicción Missing	Predicción 0	Predicción 1
Clase real Missing	0	1	0
Clase real 0	0	1390	17
Clase real 1	0	45	136

Cuadro 3.22: Matriz de confusión del modelo de clasificación.

clasificar erróneamente algunas muestras de la clase 0 como clase 1 (17 casos) y algunas de la clase 1 como clase 0 (45 casos).

3.4.2. Aplicación del método boosting para clasificación en Bigquery

Al igual que en el caso del método boosting para regresión, Bigquery entrena este tipo de modelos se entrenan utilizando la biblioteca XGBoost. Es decir, son modelos externos a Bigquery ML y están entrenados en Vertex AI.

Al implementar el modelo de clasificación con boosting, se hace uso de los Cuadros 3.12 y 3.13, que detallan los parámetros relevantes para la configuración del método en BigQuery, similar a lo realizado para la regresión. En nuestro caso, definimos la consulta SQL para la creación del modelo de Boosted Trees con clasificación, utilizando una división personalizada de los datos. Los parámetros clave incluyen la especificación del tipo de modelo, el método de división de datos, la columna de división de datos y las columnas de etiquetas de clasificación, como muestra la Figura 3.18.

Después de implementar el modelo en BigQuery, con los parámetros detallados en la Figura 3.18 y entrenar el modelo, se evaluó su rendimiento utilizando métricas estándar de clasificación. Los resultados obtenidos se pueden observar en la Figura 3.19. Además, en el mismo apartado de evaluación del modelo también se proporciona la matriz de confusión asociada al modelo, como muestra la Figura .

Las métricas obtenidas por el modelo en BigQuery muestran un rendimiento generalmente positivo en la clasificación. La precisión, que indica la proporción de predicciones positivas correctas entre

```
query = f"""
CREATE OR REPLACE MODEL `{project_id}.{dataset_id}.{model_name}`
OPTIONS(
  model_type='BOOSTED_TREE_CLASSIFIER',
  data_split_method='CUSTOM',
  data_split_col='is_test',  # Nombre de la columna de división de datos
  input_label_cols=['is_safe']  # Columna de etiqueta de clasificación
) AS
SELECT
  aluminium,
  ammonia,
  arsenic,
  barium,
  cadmium,
  chloramine,
  chromium,
  copper,
  flouride,
  bacteria,
  viruses,
  lead,
  nitrates,
  nitrites,
  mercury,
  perchlorate,
  radium,
  selenium,
  silver,
  uranium,
  is_safe,
  is_test
FROM
  `{project_id}.{dataset_id}.waterquality`
"""
```

Figura 3.18: Sintaxis de Bigquery para la creación del modelo de clasificación con boosting.

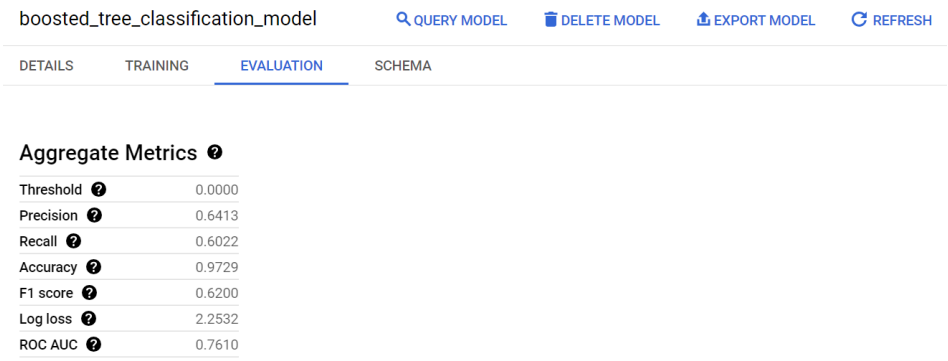


Figura 3.19: Métricas del modelo de clasificación con boosting en Bigquery.

todas las predicciones positivas, es del 64.13 %, lo que sugiere que el modelo tiende a ser preciso en sus clasificaciones positivas. La sensibilidad, o recall, es del 60.22 %, lo que significa que el modelo captura

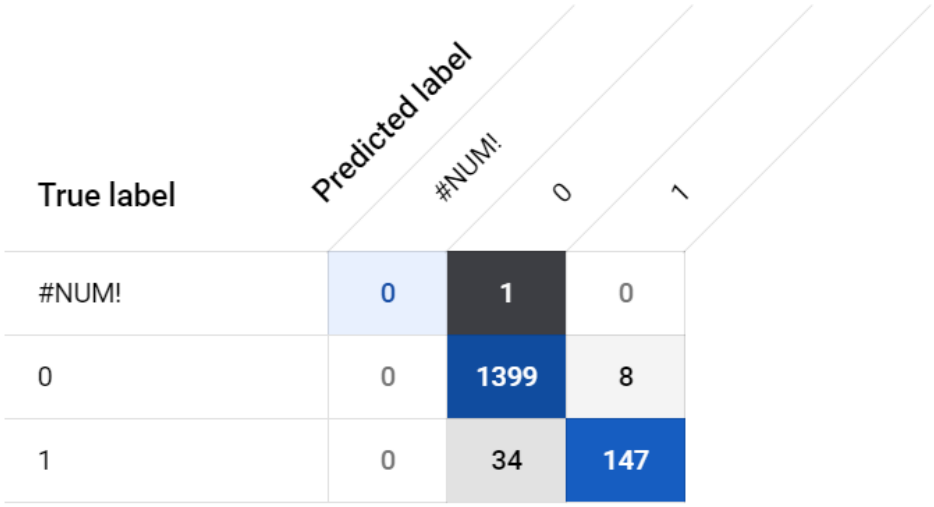


Figura 3.20: Matriz de confusión del modelo de clasificación con boosting en Bigquery.

aproximadamente el 60.22 % de todos los casos positivos. La precisión global del modelo, medida por la exactitud, es bastante alta, alcanzando el 97.29 %, lo que indica que la mayoría de las predicciones del modelo son correctas. El puntaje F1, que combina precisión y sensibilidad en una sola métrica, es del 62.00 %. Sin embargo, el valor de pérdida logarítmica es relativamente alto, con un valor de 2.2532, lo que indica que el modelo podría beneficiarse de una mayor calibración de sus predicciones. Por último, el área bajo la curva ROC (ROC AUC) es del 76.10 %, lo que sugiere que el modelo es capaz de distinguir entre las clases con una tasa razonable de verdaderos positivos y falsos positivos. La matriz de confusión dada en la Figura 3.20, muestra cómo el modelo clasificó cada etiqueta en el conjunto de evaluación. En particular, el modelo predijo correctamente 1399 instancias como agua no potable (clase 0) y 147 instancias como potable (clase 1). Sin embargo, también clasificó incorrectamente 8 instancias no potables como potables y 34 instancias potables como no potables. En definitiva, se destaca la precisión en la clasificación de la clase no potable (0) y potable (1), con un alto número de predicciones correctas. Sin embargo, se observa una cantidad significativa de falsos positivos y falsos negativos, lo que indica áreas de mejora para una clasificación más precisa.

3.4.3. Comparación del método boosting para clasificación en Python y Bigquery

Al implementar el modelo de clasificación con boosting tanto en Python como en BigQuery, se observan diferencias significativas en los enfoques y el rendimiento. En Python, se utiliza la biblioteca scikit-learn para implementar el modelo GradientBoostingClassifier, lo que proporciona flexibilidad y control sobre el proceso de entrenamiento y evaluación. Por otro lado, en BigQuery, se emplea un modelo externo entrenado en Vertex AI, lo que facilita el entrenamiento y la implementación de modelos en producción, pero puede limitar la capacidad de realizar pruebas de validación durante la creación del modelo en BigQuery ML.

La comparación de métricas de evaluación, Cuadro 3.23, revela que, aunque ambos enfoques logran un rendimiento razonable, existen diferencias en la precisión, la sensibilidad y la pérdida logarítmica entre ambos. Python muestra una mayor precisión y sensibilidad y una pérdida logarítmica más baja. Sin embargo, BigQuery muestra una precisión ligeramente mayor y una accuracy más alta que Python.

Métrica	Python	BigQuery
Precisión	0.9583	0.6413
Sensibilidad	0.5798	0.6022
Accuracy	0.9604	0.9729
Puntaje F1	0.5974	0.6200
Pérdida logarítmica	0.1169	2.2532
ROC AUC	0.7880	0.7610

Cuadro 3.23: Comparación de métricas de evaluación del modelo de clasificación con Boosting en Python y BigQuery.

En cuanto a la matriz de confusión, ambos enfoques muestran una tendencia similar en la clasificación correcta de las muestras de las clases 0 y 1, pero BigQuery tiende a clasificar menos muestras erróneamente en comparación con Python.

Plataforma	Tiempo de ejecución (segundos)
Python	8.19
BigQuery	115.91

Cuadro 3.24: Comparación del tiempo total de ejecución entre Python y BigQuery para el método boosting.

En términos de tiempo de ejecución, Cuadro 3.24, BigQuery requiere considerablemente más tiempo que Python para entrenar y evaluar el modelo, con 115.91 segundos frente a 8.19 segundos. Esta diferencia en el tiempo de ejecución puede influir en la elección de la plataforma.

En resumen, aunque ambos enfoques ofrecen buenos resultados, la elección entre Python y BigQuery para la implementación del modelo de clasificación con boosting dependerá de varios factores, como la precisión deseada, la velocidad de procesamiento, la infraestructura disponible o las necesidades específicas del proyecto.

3.4.4. Aplicación del método *Random Forest* para clasificación en Python

Siguiendo el esquema de la sección anterior, utilizaremos en este caso, el método *Random Forest* para clasificar el agua en potable o no potable. Para clasificación en Python, utilizamos la biblioteca scikit-learn, que proporciona una implementación eficiente y fácil de usar del algoritmo RandomForestClassifier, (Pedregosa y cols., 2011). Este proceso implica varias etapas, desde la preparación de los datos hasta la evaluación del modelo. Entre estas, cabe destacar que se realizan varias manipulaciones de datos, como la imputación de valores faltantes y la conversión de variables categóricas a numéricas, preparando así los datos para el entrenamiento del modelo. Posteriormente, es necesario dividir el conjunto de datos en conjuntos de entrenamiento y prueba para evaluar el rendimiento del modelo.

En cuánto a la evaluación del modelo, se calculan las métricas indicadas en el Cuadro , utilizando scikit-learn.

Métrica	**Valor**
Precisión	0.9545
Precisión promedio ('macro')	0.6226
Sensibilidad promedio ('macro')	0.5623
Accuracy	0.9566
Puntaje F1	0.5878
Pérdida logarítmica	0.1272
ROC AUC	0.9831

Cuadro 3.25: Métricas de evaluación del modelo de clasificación con Random Forest en Python.

En general, el modelo muestra un rendimiento positivo en la clasificación de las muestras. La alta precisión (95.45 %) indica que el modelo tiene una alta proporción de predicciones correctas. Además, la precisión promedio ('macro') y la sensibilidad promedio ('macro') sugieren que el modelo tiene una precisión aceptable y puede capturar correctamente las muestras positivas de todas las clases. La exactitud (95.66 %) indica que la mayoría de las predicciones del modelo son correctas. Sin embargo, el puntaje F1 y la pérdida logarítmica sugieren que el modelo podría beneficiarse de una mayor calibración de sus predicciones. Por último, el alto valor de ROC AUC (98.31 %) indica que el modelo es capaz de distinguir entre clases con una tasa razonable de verdaderos positivos y falsos positivos. Además, la matriz de confusión obtenida proporciona información detallada sobre cómo el modelo clasificó cada etiqueta en el conjunto de evaluación. En este caso, el modelo logró predecir correctamente la gran mayoría de las muestras de las clases 0 y 1, con 1394 y 126 predicciones correctas respectivamente. Sin embargo, también clasificó erróneamente algunas muestras de la clase 0 como clase 1 (13 casos) y algunas de la clase 1 como clase 0 (55 casos). A pesar de estas clasificaciones incorrectas, el modelo sigue mostrando un rendimiento positivo en la clasificación, como se refleja en las métricas de evaluación.

	Predicción Missing	**Predicción 0**	**Predicción 1**
Clase real Missing	0	1	0
Clase real 0	0	1394	13
Clase real 1	0	55	126

Cuadro 3.26: Matriz de confusión del modelo de clasificación.

3.4.5. Aplicación del método *Random Forest* para clasificación en Bigquery

En esta sección, exploraremos la aplicación del método de bosque aleatorio para regresión utilizando BigQuery. Similar al método de boosting, para construir modelos de bosque aleatorio en BigQuery se requiere el uso de la biblioteca XGBoost, como se menciona en (*XGBoost Documentation xgboost 2.1.0-dev documentation*, s.f.). Estos modelos se encuentran fuera del alcance de BigQuery ML y están entrenados en Vertex AI.

Al igual que en secciones anteriores, presentaremos una tabla que resume las configuraciones específicas para el modelo en BigQuery. Estas configuraciones abarcan una variedad de opciones que afectan el comportamiento y el rendimiento del modelo de bosque aleatorio. Para más detalles, se pueden consultar los Cuadros 3.17 y 3.18.

Una vez revisadas las configuraciones específicas del modelo, procedemos a implementar el modelo de bosque aleatorio en BigQuery. La consulta Figra 3.23, define las características del modelo y especifica los parámetros necesarios para su entrenamiento. Una vez ejecutada la consulta y entre-

```
query = f"""
CREATE OR REPLACE MODEL `{project_id}.{dataset_id}.{model_name}`
OPTIONS(
  model_type='RANDOM_FOREST_CLASSIFIER',  -- Seleccionamos RANDOM_FOREST_CLASSIFIER para clasificación
  data_split_method='CUSTOM',
  data_split_col='is_test',  -- Nombre de la columna de división de datos
  input_label_cols=['is_safe']  -- Columna de la variable objetivo
) AS
SELECT
  aluminium,
  ammonia,
  arsenic,
  barium,
  cadmium,
  chloramine,
  chromium,
  copper,
  flouride,
  bacteria,
  viruses,
  lead,
  nitrates,
  nitrites,
  mercury,
  perchlorate,
  radium,
  selenium,
  silver,
  uranium,
  is_safe,
  is_test
FROM
  `{project_id}.{dataset_id}.waterquality`
"""
```

Figura 3.21: Sintaxis de Bigquery para la creación del modelo de clasificación con *Random forest*.

nado el modelo, procedemos a evaluar su desempeño mediante diversas métricas de evaluación, que se muestran en la Figura 3.22. Observando dichas métricas, se tiene con una precisión del 62.55 % y una *Recall* del 59.13 %, el modelo demuestra una capacidad razonable para predecir correctamente las instancias positivas y capturar la mayoría de las instancias positivas reales. Además, la alta exactitud del 96.54 % indica un buen rendimiento general en la clasificación de las instancias. La puntuación F1, que equilibra precisión y recuperación, es del 60.70 %, lo que sugiere un buen equilibrio entre ambas métricas. Sin embargo, la pérdida logarítmica de 1.0685 revela cierta incertidumbre en las predicciones del modelo. Por otro lado, el área bajo la curva ROC (ROC AUC) muestra un rendimiento moderado del modelo en la capacidad de discriminación entre clases, con un valor del 74.22 %.

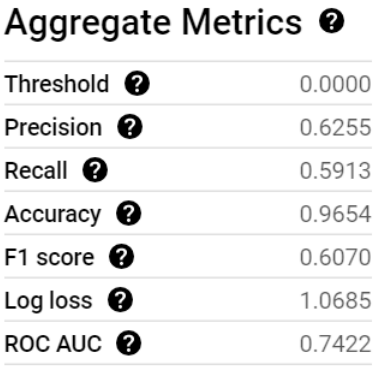


Figura 3.22: Métricas del modelo de clasificación con *Random forest* en Bigquery.

Además, igual que en casos anteriores, en el mismo apartado de evaluación del modelo también se proporciona la matriz de confusión asociada al modelo, como muestra la Figura 3.23. La matriz de confusión proporciona un detalle adicional sobre cómo el modelo clasifica correctamente e incorrectamente las instancias, con 1392 verdaderos positivos, 15 falsos positivos, 39 falsos negativos y 142 verdaderos negativos. En conjunto, estas métricas y la matriz de confusión ofrecen una evaluación completa del rendimiento del modelo de clasificación con Random Forest en BigQuery.

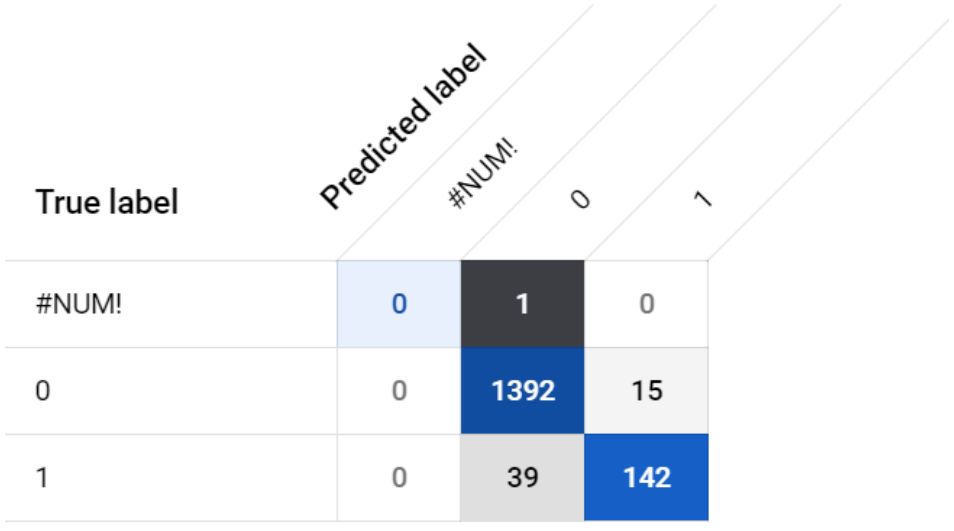


Figura 3.23: Matriz de confusión del modelo de clasificación con *Random forest* en Bigquery.

3.4.6. Comparación del método *Random Forest* para clasificación en Python y Bigquery

Igual que en modelos anteriores, el uso de Bigquery facilita la implementación, dado que no es necesario un preprocesado o división de los datos como en el caso de Python. Por otro lado, en Python,

se utiliza la biblioteca scikit-learn para implementar el modelo lo que nos permite tener un control sobre el proceso de entrenamiento y evaluación. Sien embargo, en BigQuery, se emplea un modelo externo entrenado en Vertex AI, lo que facilita el entrenamiento y la implementación de modelos, pero puede limitar la capacidad de realizar pruebas de validación.

Métrica	**Python**	**BigQuery**
Precisión	0.9545	0.6255
Sensibilidad	0.5623	0.5913
Accuracy	0.9566	0.9654
Puntaje F1	0.5878	0.6070
Pérdida logarítmica	0.1272	1.0685
ROC AUC	0.9831	0.7422

Cuadro 3.27: Comparación de métricas de evaluación del modelo de clasificación con *Random forest* en Python y BigQuery.

Si comparamos las métricas obtenidas, Cuadro 3.27, observamos algunas diferencias significativas en el rendimiento de los modelos en ambas plataformas. En Python, el modelo muestra resultados superiores en términos de precisión, sensibilidad y área bajo la curva ROC (ROC AUC), con valores de 95.45 %, 56.23 % y 98.31 % respectivamente. Estas métricas sugieren que el modelo en Python tiene una capacidad sólida para predecir correctamente las clases y discriminar entre ellas. Por otro lado, el modelo implementado en BigQuery exhibe una precisión ligeramente inferior del 62.55 %, pero con una sensibilidad ligeramente mayor del 59.13 %. Sin embargo, la pérdida logarítmica y el área bajo la curva ROC muestran un rendimiento más bajo en comparación con Python, con valores de 1.0685 y 74.22 % respectivamente. Estos resultados pueden atribuirse a las diferencias en la implementación y el entorno de ejecución entre Python y BigQuery.

En cuánto a las matrices de confusión obtenidas en ambos casos, en Python muestra que la mayoría de las predicciones son correctas, con 1394 verdaderos positivos para la clase 0 y 126 verdaderos positivos para la clase 1. Sin embargo, se observa un número relativamente alto de falsos positivos para la clase 1 (13 casos) y falsos negativos para la clase 1 (55 casos), lo que indica que el modelo puede tener dificultades para distinguir correctamente entre las clases en ciertos casos. Por otro lado, la matriz de confusión del modelo de clasificación en BigQuery muestra resultados similares en términos de verdaderos positivos y verdaderos negativos, con 1392 y 142 casos respectivamente. Sin embargo, se observa una diferencia en los falsos positivos y falsos negativos, con 15 y 39 casos respectivamente. Esto sugiere que el modelo en BigQuery puede tener una tendencia ligeramente diferente en la clasificación incorrecta de las instancias en comparación con el modelo en Python.

Por último, en términos de tiempo de ejecución, Cuadro 3.2, BigQuery requiere considerablemente más tiempo que Python para entrenar y evaluar el modelo, con 91.18 segundos frente a 4.16 segundos. Esta diferencia en el tiempo de ejecución puede deberse a las diferencias entre ambos e influir en la elección de la plataforma.

Plataforma	Tiempo de ejecución (segundos)
Python	4.16
BigQuery	91.18

Cuadro 3.28: Comparación del tiempo total de ejecución entre Python y BigQuery para Random Forest.

3.5. *Forecasting*

En esta sección, abordaremos el proceso de forecasting utilizando los datos proporcionados por la NOAA (Administración Nacional Oceánica y Atmosférica) a través de su Programa Global de Monitoreo de Gases de Efecto Invernadero (GML). Estos datos se ofrecen libremente al público y a la comunidad científica con el objetivo de fomentar una comprensión más profunda y nuevas perspectivas en el ámbito científico.

Nuestro objetivo principal es ilustrar las técnicas de *forecasting* detalladas en la sección 2.4. Para ello, nuestro conjunto de datos específico comprende las mediciones diarias de concentraciones de CO2, recopiladas desde 1974 hasta el 21 de noviembre de 2023. Es importante tener en cuenta que los datos más recientes pueden estar sujetos a procedimientos de control de calidad y son considerados preliminares. Además, es relevante destacar que debido a la erupción del volcán Mauna Loa, las mediciones del Observatorio de Mauna Loa se interrumpieron a partir del 29 de noviembre de 2022 y se reanudaron en julio de 2023. Durante este período, las observaciones se llevaron a cabo desde los Observatorios de Maunakea, aproximadamente a 21 millas al norte del Observatorio de Mauna Loa. Para obtener más detalles sobre los datos y su uso, se puede acceder a la página web oficial de NOAA (*USE OF NOAA GML DATA*, s.f.).

En base a dichos datos, exploraremos técnicas de *forecasting* para comprender mejor las tendencias y variaciones en los gases de efecto invernadero, lo que nos permitirá realizar pronósticos precisos. En particular, como en ejemplos anteriores, llevaremos a cabo este análisis desde dos vertientes, Python y BigQuery.Para facilitar el análisis de estos datos en el entorno de BigQuery, primero realizaremos algunas transformaciones preliminares. En particular, destacamos la creación de una nueva columna de fecha combinando las columnas '*year*', '*month*' y '*day*'.

Luego, emplearemos la librería de BigQuery de Google Cloud Platform para cargar los datos transformados en una tabla específica de un proyecto y conjunto de datos definidos, como muestra la Figura 3.24. Cabe destacar, que este proceso es estándar para cualquier carga de datos en BigQuery. Además, ordenaremos la tabla resultante por la columna de fecha, asegurando así que los datos estén organizados cronológicamente en la nueva tabla creada.

En definitiva, en base a estos datos y con un enfoque combinado de análisis en Python y BigQuery, exploraremos el proceso de *forecasting*, centrándonos específicamente en la implementación de un modelo ARIMA. Este modelo nos permitirá realizar pronósticos sobre las concentraciones de CO2 y comprender mejor las tendencias a lo largo del tiempo.

3.5.1. *Forecasting* en Python

En esta sección, nos centramos en el proceso de *forecasting* utilizando Python. Para ello, en concreto, implementaremos diferentes técnicas y evaluaremos la idoneidad de un modelo ARIMA para pronosticar las concentraciones de CO2 en la estación Mauna Loa.

En primer lugar, igual que en ejemplos anteriores, se inicia importando las bibliotecas necesarias, tales como `matplotlib` para visualización, `pandas` para manipulación de datos, y herramientas específicas para análisis de series temporales como `pmdarima` y `statsmodels`. Además, se establecen las

```python
# Nombre de la tabla que se creará en BigQuery
table_name = "C02_MaunaLoa"

# Inicializa el cliente de BigQuery
client = bigquery.Client(project=project_id)

# Carga los datos en BigQuery con el esquema detectado automáticamente
job_config = bigquery.LoadJobConfig(
    autodetect=True,
)
job = client.load_table_from_dataframe(
    data, f"{dataset_id}.{table_name}", job_config=job_config
)
job.result()

print(
    f"El archivo CSV se ha cargado en la tabla {dataset_id}.{table_name} en BigQuery."
)
print(data.dtypes)
```

Figura 3.24: Carga de datos en Bigquery.

conexiones con BigQuery para extraer los datos de CO2 de la estación Mauna Loa. Estos datos se cargan en un DataFrame de pandas, asegurando que la columna de fecha sea del tipo adecuado para su posterior manipulación.

Una vez cargados los datos, se procede a realizar una exploración inicial dividiendo los datos en conjuntos de entrenamiento y prueba. Con esta división, se crea un gráfico para visualizar la distribución de los datos a lo largo del tiempo, lo que proporciona una primera impresión de la serie temporal de CO2, como muestra la Figura 3.25. En dicha representación, se aprecia claramente la presencia

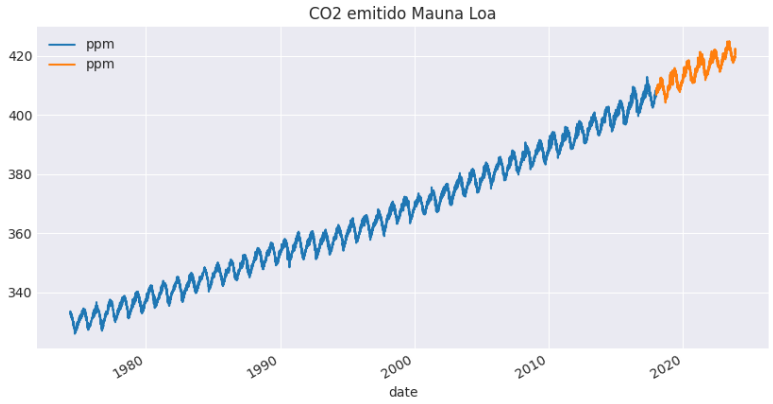


Figura 3.25: Serie temporal del CO2 emitido por el Mauna Loa.

de tendencia. Además, debemos evaluar su estacionariedad, para ello recurriremos a los test ADF (Augmented Dickey-Fuller) y KPSS (Kwiatkowski-Phillips-Schmidt-Shin). Para el primero, la hipótesis nula implica la no estacionariedad de la serie y se obtiene un p-valor asociado al contraste de 0,88, por lo que claramente no estamos en condiciones de rechazar la hipótesis nula. De forma análoga, para el test de KPSS se tiene que la hipótesis nula afirma la estacionariedad de la serie y el p-valor obtenido para este test es 0,01, por lo tanto con un nivel de significación del 5 % podremos rechazar H_0.

Debido a la presencia de tendencia, será necesario diferenciar la serie para eliminar dicha característica, como muestra la Figura 3.26. Una vez diferenciada podemos volver a evaluar su estacionariedad de forma análoga a la serie original. En este caso el p-valor asociado al test ADF es muy pequeño por lo tanto podremos rechazar la hipótesis nula, por otro lado el p-valor asociado al test de KPSS es 0.1, por lo tanto con un nivel de significación del 5 % no podemos rechazar H_0. En definitiva, ambos test nos permiten afirmar que la serie diferenciada es estacionaria.

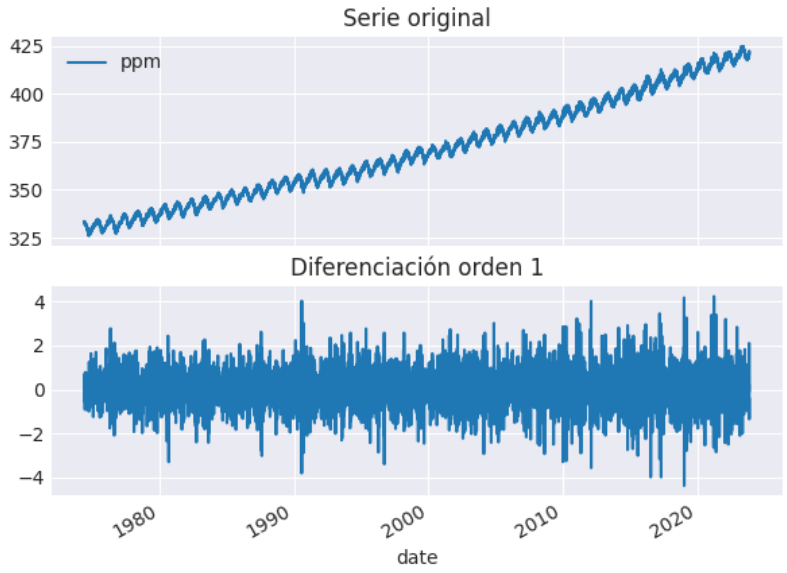


Figura 3.26: Serie temporal y serie temporal diferenciada del CO2 emitido por el Mauna Loa.

Además de evaluar la estacionariedad de la serie diferenciada, es fundamental comprender la estructura de autocorrelación en los datos. Para ello, la autocorrelación y la autocorrelación parcial son medidas cruciales de asociación entre los valores de una serie temporal, tanto actuales como pasados. Estas medidas revelan qué valores de la serie son más relevantes para predecir los valores futuros. Esencialmente, nos permiten entender cómo los datos están correlacionados a lo largo del tiempo y cuánto influyen las observaciones anteriores en las futuras. En particular, la Función de Autocorrelación (FAS) en el retardo k representa la autocorrelación entre los valores de la serie que están separados por k intervalos de tiempo. En otras palabras, nos muestra cómo los valores de la serie están relacionados a lo largo de períodos específicos de tiempo. Por otro lado, la Función de Autocorrelación Parcial (FAP) en el retardo k es la autocorrelación entre los valores de la serie que están separados por k intervalos de tiempo, pero teniendo en cuenta los valores de los intervalos intermedios. Esta medida nos proporciona información sobre la correlación directa entre observaciones individuales, eliminando la influencia de los intervalos intermedios (Cryer y Chan, 2008).

En nuestro caso de estudio, tendremos las funciones de autocorrelación y autocorrelación parcial que muestran las Figuras 3.27 y 3.28. Estos gráficos son útiles para identificar el orden de los términos autorregresivos (AR) y de medias móviles (MA) en modelos ARIMA. En el contexto de nuestra serie temporal de CO2, estos gráficos nos ayudarán a determinar los valores de p y q para plantear nuestro modelo ARIMA. Observando las funciones relativas a la serie diferenciada, se puede ver la presencia de correlaciones significativas en el primer retardo de ambas funciones lo que indica una dependencia

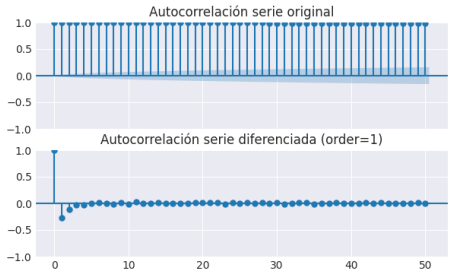


Figura 3.27: Funciones de autocorrelacion de la serie original y diferenciada.

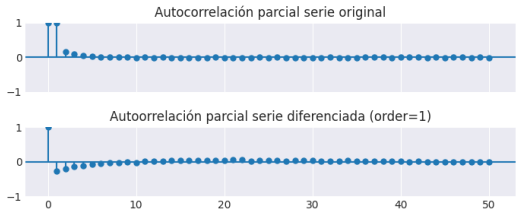


Figura 3.28: Funciones de autocorrelacion parcial de la serie original y diferenciada.

lineal entre los valores actuales y los valores inmediatamente anteriores de la serie temporal, así como una correlación residual después de eliminar las contribuciones de los retardos intermedios. Por lo tanto, optaríamos por un componente autorregresivo de orden 1(p=1) y un término de promedio móvil de orden 1 (q=1) y tendríamos que analizar la posible estacionalidad de la serie.

Sin embargo, Python facilita todo este proceso. Ya que ofrece herramientas como `auto_arima` de `pmdarima`, que automatiza la selección del modelo ARIMA, teniendo en cuenta diferentes combinaciones de parámetros. En particular, `auto_arima` funciona realizando pruebas de diferenciación (utilizando tests como Kwiatkowski–Phillips–Schmidt–Shin o Dickey-Fuller) para determinar el orden de diferenciación apropiado. Luego, si la opción seasonal está habilitada, busca identificar los hiperparámetros óptimos para determinar el orden de diferenciación estacional. Por último, para encontrar el mejor modelo posible, optimiza para un determinado criterio de información, ya sea el criterio de información de Akaike(AIC), el criterio de información bayesiano(BIC) o algún otro y finalmente devuelve el ARIMA que minimiza dicho valor. Es importante tener en cuenta que debido a problemas de estacionariedad, es posible que dicha función no encuentre un modelo adecuado. En este caso, se lanzará un mensaje sugiriendo que se tomen medidas para inducir la estacionariedad antes de ajustar el modelo, o que se seleccione un nuevo rango de valores para los parámetros del modelo (Smith y cols., 2017). En resumen, esta función es especialmente útil para identificar el modelo ARIMA más adecuado facilitando así el proceso de *forecasting*.

En nuestro caso en concreto, ajustaremos el modelo ARIMA utilizando `auto_arima` en base al criterio AIC, como muestra la Figura 3.29. En base a ello, identificamos que el modelo ARIMA(0,1,3)(0,0,0)[0] con intercepto presenta el menor AIC, siendo este de 24750.984. Esto implica que dicho modelo logra un buen equilibrio entre el ajuste a los datos y la complejidad del modelo, lo que lo convierte en una elección sólida para el pronóstico de las concentraciones de CO2 en la estación Mauna Loa.

```
# Ajustar el modelo ARIMA utilizando auto_arima y el criterio AIC
modelo_arima = auto_arima(
    datos["ppm"],
    seasonal=False,
    trace=True,
    error_action="ignore",
    suppress_warnings=True,
    stepwise=True,
)
print(modelo_arima.summary())
```

Figura 3.29: Sintaxis para el ajuste del modelo ARIMA.

En base a los resultados obtenidos, ajustaremos el modelo a los datos de entrenamiento y reali-

zamos predicciones aplicando la función `predict` sobre los datos de prueba. Es importante destacar, la necesidad de preparar las fechas del conjunto de prueba para que coincidan con las fechas de las predicciones y asignar estas fechas a las predicciones generadas. Finalmente, visualizamos los datos de entrenamiento, los datos de prueba y las predicciones del modelo en la Figura 2.30 con la finalidad de evaluar la precisión del modelo en la predicción de las concentraciones de CO2. En base a la Figura

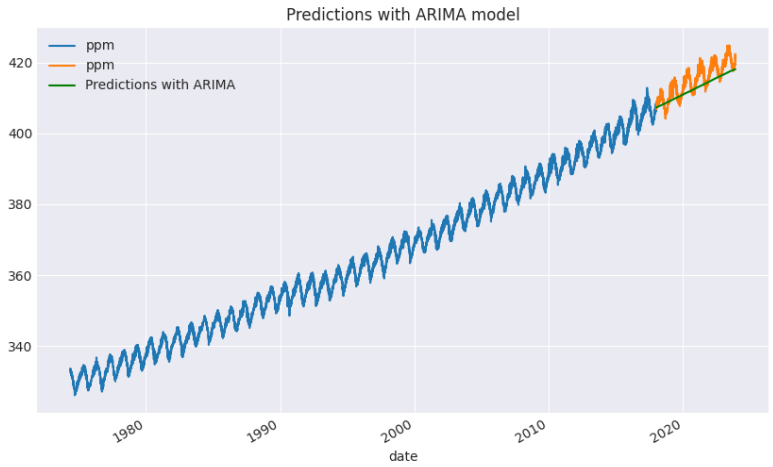


Figura 3.30: Predicciones del modelo ARIMA.

3.30, podremos afirmar que las predicciones obtenidas por nuestro modelo son razonablemente buenas, puesto que se asemejan bastante a los datos de prueba reservados.

3.5.2. *Forecasting* en Bigquery

En esta sección, exploraremos la potente capacidad de BigQuery para realizar forecasting en series temporales utilizando el modelo ARIMA_PLUS. Este modelo ofrece una herramienta robusta y eficiente para el análisis y la predicción en conjuntos de datos de series temporales a gran escala.

En particular, el modelo ARIMA_PLUS en BigQuery ML automatiza los desafíos más comunes en el análisis de series temporales, desde la detección y ajuste de valores atípicos hasta la identificación de patrones estacionales y de tendencia. El proceso de forecasting se puede realizar de manera integral utilizando funciones como `ML.FORECAST` y `ML.EXPLAIN_FORECAST`, las cuáles permiten obtener y evaluar las predicciones. Una de las características más destacadas de este modelado es su capacidad para manejar series temporales a gran escala, con la posibilidad de pronosticar hasta 100000000 series temporales simultáneamente con una sola consulta.

Para comprender mejor cómo se configura este modelo en BigQuery, es importante revisar los parámetros específicos que se pueden ajustar. Estos parámetros, detallados en el Cuadro 3.29, permiten personalizar el proceso de modelado según las necesidades y características de los datos de entrada.

Cuadro 3.29: Parámetros del modelo ARIMA_PLUS en BigQuery

Parámetro	**Descripción**
MODEL_TYPE	Especifica el tipo de modelo.
TIME_SERIES_TIMESTAMP_COL	Columna de puntos temporales para el entrenamiento.
TIME_SERIES_DATA_COL	Columna de datos a pronosticar.
TIME_SERIES_ID_COL	Columnas de ID.
HORIZON	Número de puntos de tiempo a pronosticar.
AUTO_ARIMA	Uso o no de auto.ARIMA .
AUTO_ARIMA_MAX_ORDER	Valor máximo para la suma de p y q no estacionales.
AUTO_ARIMA_MIN_ORDER	Valor mínimo para la suma de p y q no estacionales.
NON_SEASONAL_ORDER	Tupla de p, d, q no estacional para el modelo ARIMA_PLUS.
DATA_FREQUENCY	Frecuencia de datos.
INCLUDE_DRIFT	Incluir un término de deriva lineal o no.
HOLIDAY_REGION	La región geográfica para el efecto de vacaciones.
CLEAN_SPIKES_AND_DIPS	Detección y limpieza automática de picos y caídas.
ADJUST_STEP_CHANGES	Detección y el ajuste automático de cambios de paso.
TIME_SERIES_LENGTH_FRACTION	Fracción de la longitud de la serie para modelar la tendencia.
MIN_TIME_SERIES_LENGTH	Número mínimo de puntos temporales para modelar la tendencia.
MAX_TIME_SERIES_LENGTH	Número máximo de puntos temporales para modelar la tendencia.
TREND_SMOOTHING_WINDOW_SIZE	Tamaño de la ventana de suavizado para la tendencia.
FORECAST_LIMIT_LOWER_BOUND	Límite inferior de los valores de pronóstico.
FORECAST_LIMIT_UPPER_BOUND	Límite superior de los valores de pronóstico.
SEASONALITIES	La estacionalidad de los datos.
HIERARCHICAL_TIME_SERIES_COLS	Columnas utilizadas para generar pronósticos de series temporales jerárquicas.

Una vez comprendidos estos parámetros, podemos proceder a aplicar el modelo ARIMA_PLUS a nuestro propio conjunto de datos, como refleja la Figura 3.31. Los resultados obtenidos de BigQuery

```
# Define la consulta SQL para crear un modelo ARIMA+
query = f"""
CREATE OR REPLACE MODEL `{project_id}.{dataset_id}.{model_name}`
OPTIONS(
  model_type='ARIMA_PLUS',
  time_series_timestamp_col='date',
  time_series_data_col='ppm',
  auto_arima=TRUE,
  data_frequency='AUTO_FREQUENCY',
  decompose_time_series=TRUE
) AS
SELECT
  date,
  ppm
FROM
  `{project_id}.{dataset_id}.C02_MaunaLoa_ordered`
"""
```

Figura 3.31: Sintaxis de Bigquery para la creación de un modelo ARIMA_PLUS.

proporcionan una visión detallada de las configuraciones y métricas asociadas con el modelo ARIMA_PLUS utilizado para el análisis de series temporales. Estos resultados revelan los parámetros clave del modelo, incluidos los términos autorregresivos, de diferenciación y de media móvil (p, d y q respectivamente), que son fundamentales para modelar la autocorrelación y la estacionalidad. Además de los parámetros básicos, Bigquery especifica la presencia o ausencia de características como la deriva, picos y caídas, efectos de días festivos, cambios bruscos y la periodicidad de los datos. Estas características adicionales proporcionan información sobre aspectos específicos del modelo que pueden influir en su rendimiento. También muestra métricas de evaluación que ofrecen una medida objetiva de la calidad del ajuste del modelo a los datos observados, como el logaritmo de la verosimilitud, el criterio de información de Akaike (AIC) y la varianza. De esta forma, un mayor logaritmo de verosimilitud y varianza, junto con un menor valor de AIC, indican un mejor ajuste del modelo a los datos de series temporales.

En base al criterio de información de Akaike, Bigquery seleccionaría el modelo con p= 2, d= 1, q= 2. Dicho modelo no presenta deriva, pero sí tiene picos y caídas, no considera efecto de días festivos, pero sí tiene en cuenta cambios bruscos y la serie temporal presenta una estacionalidad tanto semanal como anual. En relación a las métricas asociadas, se tiene que el logaritmo de la verosimilitud es -3815.616, el AIC es 7643.231 y la varianza de 0.089. Como indicamos, para seleccionar el mejor modelo, podríamos considerar alternativas, como maximizar el logaritmo de verosimilitud o también podríamos evaluar la estacionalidad de los datos y seleccionar el modelo que mejor se ajuste a esa periodicidad. De modo que, el modelo seleccionado dependerá de los objetivos específicos del análisis y de cómo se prioricen las métricas de evaluación.

En resumen, el modelo ARIMA_PLUS representa una buena herramienta para el análisis de datos a gran escala. Este modelo automatiza procesos clave en el análisis de series temporales, desde la detección de patrones hasta la generación de predicciones precisas, lo que facilita significativamente el trabajo con conjuntos de datos complejos y extensos. Por otro lado, la amplia gama de parámetros permite adaptar el modelo a las particularidades de cada conjunto de datos, lo que maximiza su

capacidad predictiva y su capacidad para capturar tendencias y estacionalidades. Además, igual que en ejemplos anteriores, cabe destacar la capacidad de BigQuery para manejar grandes volúmenes de datos y pronosticar múltiples series temporales simultáneamente mediante una sola consulta.

3.5.3. Comparación de *forecasting* en Python y Bigquery

Al comparar el proceso de forecasting en Python y BigQuery, podemos identificar algunas diferencias significativas en cuanto a la implementación y las capacidades de cada plataforma.

En Python, el proceso de forecasting implica una serie de pasos manuales que incluyen la importación de bibliotecas específicas, la preparación de datos, la exploración y visualización de la serie temporal, la identificación de la estacionariedad, la selección del modelo adecuado (como ARIMA) y la evaluación de su desempeño. Aunque Python ofrece una amplia gama de herramientas y bibliotecas para análisis de series temporales, como `pandas`, `statsmodels` y `pmdarima`, la implementación de estos procesos puede requerir un conocimiento técnico sólido y un tiempo considerable.

Por otro lado, BigQuery simplifica significativamente el proceso de forecasting al automatizar muchas de las tareas clave, como la detección de valores atípicos, la identificación de patrones estacionales y de tendencia, o la selección del modelo adecuado (como ARIMA_PLUS). Además, la capacidad de BigQuery para manejar grandes volúmenes de datos y pronosticar múltiples series temporales simultáneamente con una sola consulta ofrece una ventaja significativa en términos de escalabilidad y eficiencia.

Si bien Python ofrece flexibilidad y control total sobre el proceso de forecasting, BigQuery destaca por su facilidad de uso, escalabilidad y capacidad para manejar grandes conjuntos de datos. La elección entre Python y BigQuery dependerá de las necesidades específicas del proyecto, así como del nivel de experiencia y recursos disponibles. En resumen, Python es ideal para proyectos que requieren un enfoque personalizado y detallado, mientras que BigQuery es más adecuado para proyectos que involucran grandes volúmenes de datos y requieren una solución rápida y escalable.

3.6. Sistemas de Recomendación Basados en Factorización de Matrices

En los capítulos anteriores, hemos discutido la teoría detrás de las técnicas de factorización de matrices y su aplicabilidad en sistemas de recomendación. Ahora, nos centraremos en un análisis práctico de estos modelos, implementándolos con herramientas modernas como Python y BigQuery.

3.6.1. Descripción del conjunto de datos

Para la implementación práctica de los sistemas de recomendación basados en factorización de matrices, utilizaremos el conjunto de datos MovieLens (Harper y Konstan, 2015). Este conjunto de datos fue recopilado por el Grupo de Investigación GroupLens en la Universidad de Minnesota. El conjunto de datos MovieLens consta de 100,000 calificaciones (de 1 a 5) proporcionadas por 943 usuarios sobre 1682 películas. Cada usuario ha calificado al menos 20 películas. Los datos fueron recopilados a través del sitio web MovieLens (movielens.umn.edu) durante un periodo de siete meses, desde el 19 de septiembre de 1997 hasta el 22 de abril de 1998. Este conjunto de datos ha sido limpiado, eliminando usuarios que tenían menos de 20 calificaciones. El propósito de esta limpieza fue garantizar la calidad y la integridad de los datos para su uso en investigaciones.

Para llevar a cabo la implementación, utilizaremos Python y BigQuery. Comenzaremos cargando los datos a partir de los archivos CSV que contienen los datos de películas y calificaciones. Para ello, se inicializa un cliente de BigQuery y se leen los archivos CSV en DataFrames de pandas, se definen los nombres de las tablas que se crearán en BigQuery y, finalmente, se configuran y ejecutan los trabajos de carga de datos para subir los DataFrames de pandas a las tablas de BigQuery especificadas. A

dichos datos, se le añade una columna de división de los datos, que nos permitirá efectuar la división en conjunto de entrenamiento y test.

3.6.2. Factorización de Matrices en Python

Para implementar la factorización de matrices en Python, utilizamos una combinación de bibliotecas de análisis de datos y de aprendizaje automático, como `pandas`, `numpy`, `matplotlib`, `seaborn`, y `surprise`, junto con BigQuery para la gestión y consulta de los datos indicados. Como en ejemplos anteriores, comenzamos importando las bibliotecas necesarias y configurando el cliente de BigQuery para conectar con el proyecto y el conjunto de datos específicos. Luego, se consultan las tablas de BigQuery, que contienen las calificaciones de las películas y los detalles de las mismas y se cargan en un dataFrame de pandas.

Antes de plantear el modelo, podemos visualizar la distribución de las calificaciones para entender mejor los datos. Esta visualización nos permite identificar patrones y características importantes en las calificaciones de los usuarios. Al visualizar el gráfico resultante, podremos hacer varias observaciones importantes. Si la mayoría de las calificaciones se agrupan en torno a valores altos (por ejemplo, 4 o 5), esto indica un sesgo hacia calificaciones positivas entre los usuarios. En cambio, una cantidad significativa de calificaciones bajas sugiere una mayor variabilidad en las opiniones. Entender la proporción de calificaciones en cada rango puede ayudar a ajustar los algoritmos de recomendación para manejar mejor las distribuciones sesgadas. La visualización también puede revelar la presencia de datos atípicos y mostrar si la distribución es unimodal o multimodal, lo cual afecta las estrategias de modelado y evaluación. Además, conocer la distribución de las calificaciones es útil para evaluar el desempeño del modelo, ya que un modelo que predice consistentemente calificaciones altas puede parecer preciso pero en realidad no capturar la variabilidad real en los datos. En nuestro caso en particular, Figura 3.32, se tiene que la mayoría de las calificaciones se corresponden con valores altos. Esto podría tenerse en cuenta para el modelado.

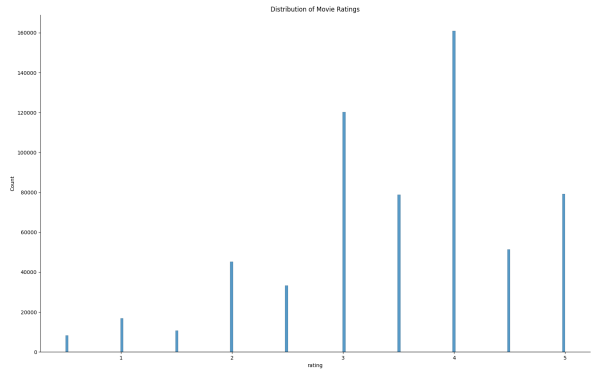


Figura 3.32: Distribución de las calificaciones.

Ya visualizados los datos, se procede a prepararlos para el modelo de factorización de matrices, como indica la Figura 3.33. Para ello, definimos un objeto para especificar el rango de calificaciones y cargamos los datos en un formato compatible con `surprise`. Cabe destacar, la necesidad de dividir los datos en conjuntos de entrenamiento o prueba y la necesidad de utilizar un formato concreto compatible con `surprise`.

Finalmente, se entrena un modelo de factorización de matrices usando la técnica SVD (*Singular Value Decomposition*) y se evalúa su desempeño en el conjunto de prueba. Inicialmente, se utilizan valores predeterminados para los hiperparámetros del modelo. Los resultados de las métricas de eva-

```python
# Define un objeto Reader para especificar el rango de ratings
reader = Reader(rating_scale=(0.5, 5))

# Crea un dataset Surprise con las columnas requeridas
data = Dataset.load_from_df(movie_ratings_data[["userId", "movieId", "rating"]], reader)

# Filtrar el DataFrame para obtener trainset y testset
datatrainset = movie_ratings_data[movie_ratings_data['split'] == 'TRAIN']
datatestset = movie_ratings_data[movie_ratings_data['split'] == 'TEST']

# Crea un dataset Surprise con las columnas requeridas
trainset = Dataset.load_from_df(datatrainset[["userId", "movieId", "rating"]], reader)
testset = Dataset.load_from_df(datatestset[["userId", "movieId", "rating"]], reader)

# Convertir el conjunto de entrenamiento y test a un formato compatible con Surprise
trainset = trainset.build_full_trainset()
testset = testset.build_full_trainset().build_testset()
```

Figura 3.33: Procesamiento de los datos.

luación para este primer modelo se presentan en el Cuadro 3.30. Estas métricas de evaluación ofrecen

Métrica	**Valor**
Error Cuadrático Medio (MSE)	0.7867
R-cuadrado R^2	0.2738
Error Absoluto Medio (MAE)	0.6822
Error Logarítmico Cuadrado Medio (MSLE)	0.0561
Error Absoluto Mediano (MedAE)	0.5412

Cuadro 3.30: Métricas de evaluación del modelo de factorización de matrices con valores predeterminados para los hiperparámetros en Python.

una evaluación cuantitativa del rendimiento del modelo de factorización de matrices utilizando la técnica SVD. En el modelo inicial, con valores predeterminados para los hiperparámetros, observamos un MSE de 0.7867, que representa la magnitud promedio del error cuadrático entre las predicciones del modelo y los valores reales; un MSE más bajo indica una mayor precisión del modelo. El R^2 de 0.2738 muestra que el modelo explica aproximadamente el 27.38 % de la variabilidad en las calificaciones de los usuarios, lo que sugiere un rendimiento relativamente bajo en la explicación de la variabilidad. El MAE de 0.6822 representa la magnitud promedio del error absoluto entre las predicciones y los valores reales, lo que indica una discrepancia significativa entre las predicciones y los datos reales. El MSLE de 0.0561 mide el error cuadrático medio de los logaritmos de las predicciones, mientras que el MedAE de 0.5412 indica la mediana de los errores absolutos entre las predicciones y los valores reales. Estos resultados revelan la necesidad de mejorar el modelo inicial para lograr una mayor precisión y capacidad explicativa.

En base al objetivo de mejorar el modelo, se pueden ajustar sus hiperparámetros utilizando GridSearchCV, como indica la Figura 3.34. Posteriormente, entrenamos el modelo final los mejores paráme-

```python
# Ajuste del modelo
# utilizamos algoritmo GridSearchCv
param_grid = {"n_epochs": [5, 10], "lr_all": [0.002, 0.005]}

gs = GridSearchCV(
    SVD, param_grid, measures=["rmse", "mae"], cv=3, n_jobs=-1, joblib_verbose=True
)

gs.fit(data)
gs.best_score["rmse"]
gs.best_params["rmse"]
```

Figura 3.34: Ajuste de hiperparámetros utilizando GridSearchCV.

tros encontrados y se valida nuevamente en el conjunto de prueba. Las métricas calculadas se muestran en el Cuadro 3.31.

Métrica	**Valor**
Error Cuadrático Medio (MSE)	0.1495
R-cuadrado R^2	0.8619
Error Absoluto Medio (MAE)	0.2858
Error Logarítmico Cuadrado Medio (MSLE)	0.0113
Error Absoluto Mediano (MedAE)	0.2182

Cuadro 3.31: Métricas de evaluación del modelo de factorización de matrices con los mejores hiperparámetros encontrados en Python.

Los resultados muestran una mejora significativa en las métricas de evaluación después del ajuste del modelo utilizando GridSearchCV. En particular, el RMSE y el MAE se redujeron considerablemente, lo que indica que el modelo ajustado es más preciso en sus predicciones. El $\mathbf{R^2}$ del modelo final es significativamente más alto (0.8619) comparado con el modelo inicial (0.2738), lo que sugiere que el modelo final explica mucho mejor la variabilidad en las calificaciones de los usuarios. Esto destaca la importancia de ajustar los hiperparámetros para mejorar el desempeño de los modelos de recomendación basados en factorización de matrices.

3.6.3. Factorización de Matrices en Bigquery

La factorización de matrices es una técnica ampliamente utilizada en el campo del aprendizaje automático y la minería de datos, especialmente en sistemas de recomendación. En BigQuery ML, esta técnica permite descomponer una matriz grande y dispersa en el producto de dos matrices más pequeñas y densas, facilitando así la identificación de patrones latentes y la reducción de dimensionalidad.

BigQuery ML admite la factorización de matrices a través de modelos como la Descomposición en Valores Singulares (SVD) y la Factorización de Matrices No Negativas (NMF), los cuales son particularmente efectivos para nuestras necesidades actuales. En términos de modelado, BigQuery proporciona una variedad de parámetros configurables para optimizar el rendimiento del modelo, veáse Cuadro 3.32.

Parámetro	**Descripción**
MODEL_TYPE	Especifica el tipo de modelo.
FEEDBACK_TYPE	Determina el tipo de feedback utilizado durante el entrenamiento.
NUM_FACTORS	Especifica el número de factores latentes a usar.
USER_COL	Nombre de la columna de usuario.
ITEM_COL	Nombre de la columna de ítem.
RATING_COL	Nombre de la columna de calificación.
WALS_ALPHA	Ajusta el impacto del feedback del usuario en la confianza de la recomendación.
L2_REG	Cantidad de regularización L2 aplicada.
MAX_ITERATIONS	Número máximo de iteraciones de entrenamiento.
EARLY_STOP	Determina si el entrenamiento debe detenerse tras una mejora mínima relativa.
MIN_REL_PROGRESS	Mejora mínima relativa del error necesaria para continuar el entrenamiento con EARLY_STOP.
DATA_SPLIT_METHOD	Método utilizado para dividir los datos de entrada.
DATA_SPLIT_EVAL_FRACTION	Fracción de los datos utilizados como datos de evaluación.
DATA_SPLIT_TEST_FRACTION	Fracción de los datos utilizados como datos de prueba (solo con ajuste de hiperparámetros).
DATA_SPLIT_COL	Nombre de la columna utilizada para dividir los datos de entrada.
NUM_TRIALS	Número máximo de submodelos a entrenar (solo con ajuste de hiperparámetros).
MAX_PARALLEL_TRIALS	Número máximo de pruebas a ejecutar al mismo tiempo.
HPARAM_TUNING_ALGORITHM	Algoritmo utilizado para ajustar los hiperparámetros.
HPARAM_TUNING_OBJECTIVES	Objetivo del ajuste de hiperparámetros.

Cuadro 3.32: Parámetros del modelo de factorización de matrices en BigQuery ML

En nuestra configuración específica, los hiperparámetros del modelo se definen directamente en la consulta SQL utilizada para crear el modelo en BigQuery como muestra la Figura .

Estos parámetros son esenciales para adaptar el rendimiento del modelo a nuestras necesidades.

```
query = f"""
CREATE OR REPLACE MODEL `{project_id}.{dataset_id}.{model_name2}`
    OPTIONS(
     MODEL_TYPE='MATRIX_FACTORIZATION',
     FEEDBACK_TYPE='EXPLICIT',
     NUM_FACTORS=3,
     L2_REG=500.0,
     USER_COL='userId',
     ITEM_COL='movieId',
     RATING_COL='rating',
     data_split_method='CUSTOM',
    data_split_col='is_test') AS
SELECT
  userId,
  movieId,
  rating,
  is_test
FROM
  `{project_id}.{dataset_id}.factormatrix_istest`
"""
```

Figura 3.35: Sintaxis de Bigquery para crear el modelo de factorización de matrices.

Por ejemplo, determinamos el número de factores latentes con el hiperparámetro NUM_FACTORS y controlamos la regularización con L2_REG. Además, especificamos los nombres de las columnas que contienen información sobre usuarios, elementos y calificaciones con USER_COL, ITEM_COL y RATING_COL, respectivamente. También establecemos el método y la columna utilizados para dividir los datos de entrada en conjuntos de entrenamiento y evaluación al igual que en ejemplos anteriores. Además, es importante tener en cuenta que algunos hiperparámetros, como HPARAM_TUNING_ALGORITHM y HPARAM_TUNING_OBJECTIVES, tienen valores predeterminados que se aplican si no se especifican explícitamente en la consulta. Por ejemplo, el valor predeterminado para HPARAM_TUNING_ALGORITHM es VIZIER_DEFAULT, que utiliza el algoritmo por defecto de Vertex AI Vizier para ajustar los hiperparámetros. Este algoritmo realiza una mezcla de algoritmos de búsqueda avanzada, incluida la optimización bayesiana con procesos gaussianos y utiliza el aprendizaje por transferencia para aprovechar modelos previamente sintonizados.

Después de configurar y entrenar el modelo de factorización de matrices en BigQuery ML, podemos a evaluar su rendimiento utilizando diversas métricas de evaluación dadas en la Figura 3.36.

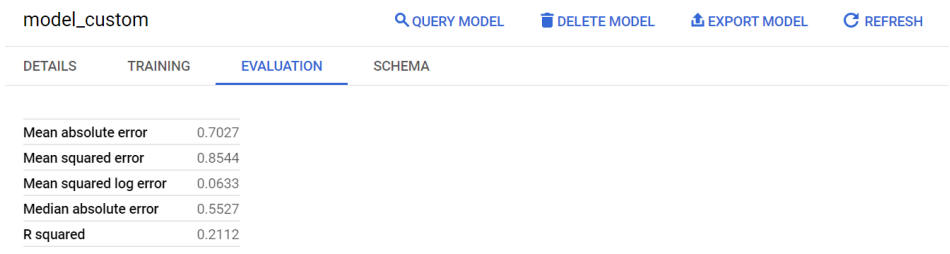


Figura 3.36: Métricas del modelo de factorización de matrices en Bigquery.

En cuánto a los valores obtenidos, se puede destacar un coeficiente de determinación, R^2 de 0.2112,

lo que indica un mal ajuste del modelo a los datos observados. Por otro lado, las métricas de error absoluto medio, error cuadrático medio y error absoluto mediano presentan valores elevados, mientras que lo ideal sería obtener valores más bajos que indiquen un mejor ajuste del modelo a los datos de entrada. Si bien estos resultados proporcionan una base para evaluar el rendimiento del modelo, es importante tener en cuenta que existen posibilidades de mejora mediante la modificación de hiperparámetros. Sin embargo, es crucial señalar que BigQuery ML presenta una limitación importante para este tipo de modelos, ya que los modelos de factorización de matrices solo están disponibles para clientes con reservas. Entrenar modelos de factorización de matrices generalmente requiere recursos computacionales e infraestructura que podrían no estar disponibles fácilmente para su uso y requieren un coste a mayores. Por lo tanto, para proceder con la mejora del modelo, se necesitaría reconfigurar una reserva o utilizar una infraestructura específica según las instrucciones detalladas en la documentación pública de BigQuery.

3.6.4. Comparación de factorización de matrices en Python y Bigquery

A lo largo de esta sección, hemos explorado la aplicación de la factorización de matrices tanto en entornos Python como en BigQuery ML, con el objetivo de desarrollar un sistema de recomendación efectivo para el conjunto de datos de películas. Ambos enfoques ofrecen sus propias ventajas y desafíos, los cuales analizaremos a continuación.

La implementación en Python se basa en una combinación de bibliotecas de análisis de datos y aprendizaje automático. Este enfoque proporciona una gran flexibilidad en términos de procesamiento de datos y ajuste de modelos. Se puede destacar, la facilidad de Pyhon para realizar un análisis exploratorio de los datos y así, comprender la distribución de las calificaciones de los usuarios y identificar posibles patrones. También se destaca la necesidad de un preprocesado de los datos, en particular, la división o el ajuste de estos para que sean un dataframe de `surprise`. Por otro lado, la implementación en BigQuery ML aprovecha las capacidades de esta plataforma para procesar y modelar grandes conjuntos de datos de manera eficiente. Mediante consultas SQL, se define y configura el modelo de factorización de matrices directamente. Si bien este enfoque simplifica el proceso de modelado y evaluación, presenta limitaciones significativas en términos de disponibilidad y acceso. Además, la capacidad de ajustar los hiperparámetros puede estar restringida por la necesidad de reservas y la limitación de recursos computacionales. En cuánto a los resultados obtenidos, no son del todo comparables a nivel métricas, puesto que no se están utilizando exactamente los mismos parámetros pero en términos generales, el modelo de Python muestra un mejor comportamiento.

De este modo, la elección entre Python y BigQuery ML para la implementación de la factorización de matrices depende de los requisitos específicos del proyecto y las limitaciones de recursos. Si se busca flexibilidad y control total sobre el proceso de modelado, Python puede ser la mejor opción. Sin embargo, si se prioriza la escalabilidad para el procesamiento de grandes volúmenes de datos, BigQuery ML puede ser una buena alternativa. Es importante tener en cuenta las limitaciones de acceso y ajuste de modelos en BigQuery ML, especialmente en entornos donde la disponibilidad de recursos es un factor crítico.

3.7. Clustering con k-means

En esta sección, abordaremos la implementación práctica del algoritmo k-means, utilizando tanto Python como BigQuery. Mientras que en el Capítulo 2 se presentó una revisión metodológica detallada del algoritmo, aquí nos centraremos en su aplicación práctica. Veremos cómo aplicar este algoritmo para realizar clustering en conjuntos de datos reales, destacando las ventajas y desafíos asociados con cada enfoque de implementación.

En particular, trabajaremos con el mismo conjunto de datos que se utilizó para la implementación de la regresión logística. Este conjunto de datos proporciona información simulada sobre la calidad del agua en un entorno urbano y está disponible en Kaggle (Kaggle, 2019).

3.7.1. Clustering con k-means en Python

En la implementación de k-means en Python, igual que en ejemplos anteriores, se comienza por la preparación de los datos, los cuales se obtienen de un conjunto de datos que proporciona información simulada sobre la calidad del agua en un entorno urbano. Este conjunto de datos, el mismo utilizado para la implementación de la regresión logística y se extrae de BigQuery mediante consultas SQL. Una vez obtenidos, se realiza un procesado inicial, como la sustitución de valores nulos por NaN en la columna ïs_safe", que indica si el agua es segura o no. Además, igual que en el caso de la regresión logística debemos convertir esta columna a tipo categórico para facilitar su manejo durante el entrenamiento del modelo.

Posteriormente, se lleva a cabo una visualización de los datos mediante histogramas para comprender mejor la distribución de las características como se puede ver en la Figura 3.37. Luego, se realiza

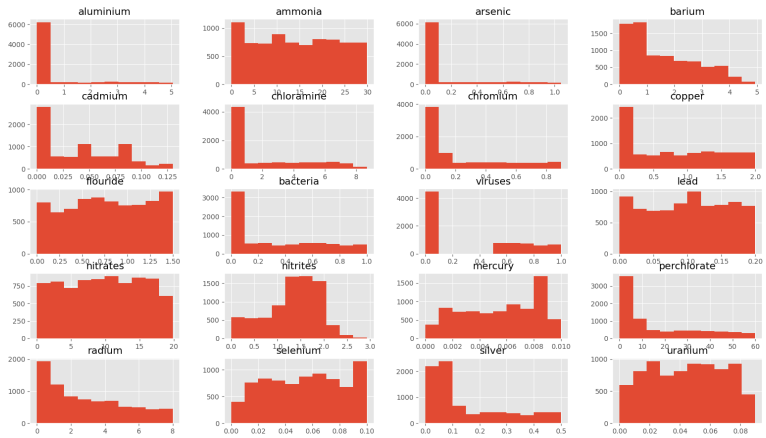


Figura 3.37: Distribución de las variables explicativas.

una división de los datos en características (X) y la variable objetivo (y), donde al igual que en el caso de la regresión logística, X contiene todas las características del agua y y indica su seguridad. El siguiente paso es abordar el problema de los datos faltantes. Una opción es eliminar los registros con valores nulos, lo cual se realiza creando un nuevo conjunto de datos sin valores nulos. A continuación, se determina el número óptimo de clústeres utilizando el método del codo o método Elbow. La idea detrás de este método es bastante sencilla, se trata de identificar el número de clústeres para el que se observa un cambio significativo en la tasa de disminución de la varianza intra-cluster (también conocido como suma total de las distancias al cuadrado). Para ello se debe ejecutar el algoritmo de k-means con diferentes números de clústeres (k) y calcular la suma de las distancias al cuadrado de cada punto respecto a su centroide. Usando los resultados para crear una gráfica con los valores de k en el eje x y la suma de las distancias al cuadrado en el eje y. En esta gráfica se busca el punto donde se produce un cambio brusco en la disminución de la suma de las distancias al cuadrado. Punto en el que la curva muestra la forma de un codo. El número óptimo de clústeres se corresponde con este punto que se conoce como el "codo" (Umargono, Suseno, y Gunawan, 2020). Ante esto, en base a la gráfica que muestra la Figura 3.38, podremos deducir que el número de clústeres de nuestro conjunto de datos es $k = 2$. Dicho esto, estamos en condiciones de ajustar el modelo k-means a los datos utilizando la función `Kmeans` de `scikit-learn` (Pedregosa y cols., 2011).

A continuación, se evalúa su desempeño utilizando métricas como el índice Davies-Bouldin y la

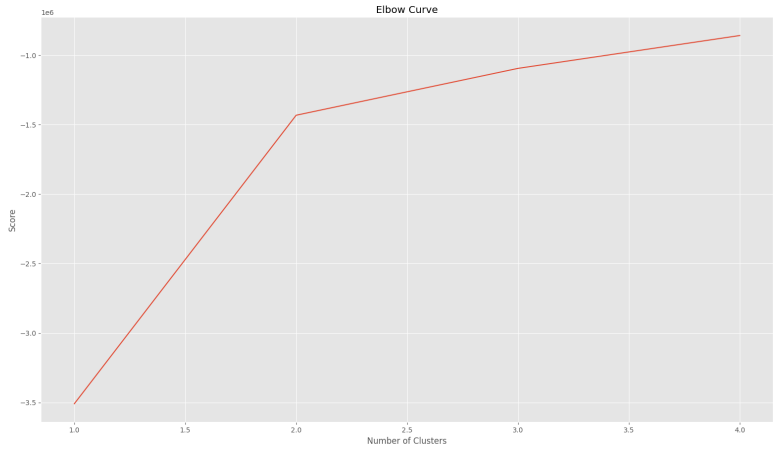


Figura 3.38: Curva de Elbow (método del codo).

distancia media cuadrada. En primer lugar, se calculó el índice Davies-Bouldin, el cual arrojó un valor de 0.7617, indicando una calidad de agrupación aceptable. Además, se calculó la distancia media cuadrada, la cual se estimó en 780.695. Este valor anómalo puede deberse a que los datos no están normalizados.

Por lo que posteriormente, se exploraron otras variantes del algoritmo k-means, incluyendo una versión con los datos normalizados y una métrica similar al coseno y otra con los datos estandarizados. El índice Davies-Bouldin para el modelo con métrica similar al coseno fue de 0.9661, indicando una calidad ligeramente inferior en comparación con el modelo original. Además, la distancia cuadrada media para esta variante fue de 0.4578. Por otro lado, al estandarizar los datos, se observó un aumento significativo en el índice Davies-Bouldin, con un valor de 2.1263. Esto sugiere una calidad de agrupación inferior en comparación con los otros enfoques. La distancia cuadrada media para el modelo con datos estandarizados fue de 23.5589, indicando una mayor dispersión de los datos alrededor de los centroides.

Estos resultados muestran la importancia de explorar diferentes enfoques y técnicas de preprocesamiento de datos al aplicar el algoritmo k-means, ya que pueden afectar significativamente la calidad de la agrupación obtenida.

3.7.2. Clustering con k-means en Bigquery

La implementación del algoritmo k-means en BigQuery permite manejar grandes volúmenes de datos utilizando la infraestructura escalable de Google Cloud. Esto facilita la creación y entrenamiento de modelos de clustering directamente en BigQuery a través de consultas SQL, aprovechando la potencia y eficiencia de esta plataforma.

Para crear un modelo k-means en BigQuery, es esencial entender los diversos parámetros de configuración que se pueden especificar. Estos parámetros permiten ajustar el comportamiento del modelo y optimizar el proceso de clustering según las necesidades específicas del análisis de datos. A continuación, se presenta una tabla, Cuadro 3.33, con los parámetros admitidos para la creación de un modelo k-means en BigQuery.

Parámetro	**Descripción**
MODEL_TYPE	Especifica el tipo de modelo. En este caso, debe ser 'KMEANS'.
NUM_CLUSTERS	Define el número de clusters a identificar. Puede ser un valor entero o especificar un rango o candidatos para la sintonización de hiperparámetros.
KMEANS_INIT_METHOD	Método de inicialización de los clusters. Puede ser 'RANDOM', 'KMEANS++' o 'CUSTOM'.
KMEANS_INIT_COL	Nombre de la columna utilizada para inicializar los centroides, debe ser de tipo BOOL.
DISTANCE_TYPE	Tipo de métrica utilizada para calcular la distancia entre dos puntos. Puede ser 'EUCLIDEAN' o 'COSINE'.
STANDARDIZE_FEATURES	Determina si se deben estandarizar las características numéricas. Valores posibles: TRUE o FALSE.
MAX_ITERATIONS	Número máximo de iteraciones de entrenamiento. Valor predeterminado: 20.
EARLY_STOP	Determina si el entrenamiento debe detenerse tras la primera iteración en la que la mejora relativa de la pérdida sea menor que MIN_REL_PROGRESS.
MIN_REL_PROGRESS	Mínima mejora relativa de la pérdida necesaria para continuar el entrenamiento cuando EARLY_STOP está en TRUE.
WARM_START	Determina si se debe entrenar un modelo con nuevos datos de entrenamiento, nuevas opciones de modelo, o ambos.
NUM_TRIALS	Número máximo de submodelos a entrenar para la sintonización de hiperparámetros.
MAX_PARALLEL_TRIALS	Número máximo de pruebas a ejecutar al mismo tiempo.
HPARAM_TUNING_ALGORITHM	Algoritmo utilizado para sintonizar los hiperparámetros: 'VIZIER_DEFAULT', 'RANDOM_SEARCH' o 'GRID_SEARCH'.
HPARAM_TUNING_OBJECTIVES	Objetivo de sintonización de hiperparámetros. El valor predeterminado es 'DAVIES_BOULDIN_INDEX'.

Cuadro 3.33: Parámetros para la creación de un modelo k-means en BigQuery

En nuestra configuración específica, los hiperparámetros del modelo se definen directamente en la consulta SQL utilizada para crear el modelo en BigQuery, como se muestra a continuación en la Figura 3.39. Entre las opciones del modelo, podemos destacar que se establece un rango de hiperparámetros para el número de clusters utilizando num_clusters=HPARAM_RANGE(2,5), lo que significa que el número de clusters se seleccionará automáticamente dentro del rango especificado durante la sinto-

nización de hiperparámetros. Por otro lado, la distancia utilizada para calcular la similitud entre los puntos (entre x e y) se establece en 'COSINE' mediante distance_type='COSINE', la cuál se obtiene como sigue (Google Cloud, 2024):

$$\sqrt{1 - \frac{x \cdot y}{\|x\|\|y\|}},$$

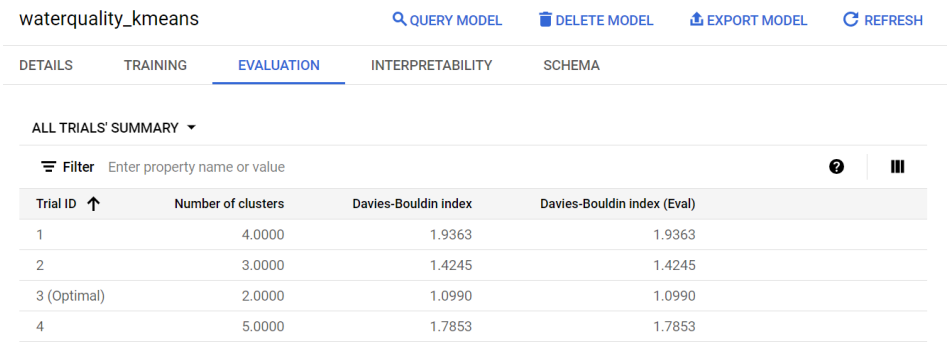
donde $\|x\|$ representa la norma L2 de x.

```
# Define la consulta SQL para crear un modelo K-means
query = f"""
CREATE OR REPLACE MODEL `{project_id}.{dataset_id}.{model_name}`
OPTIONS(
  model_type='kmeans',
  num_clusters=HPARAM_RANGE(2,5),  # Define el número de clusters según tu requerimiento
  distance_type='COSINE',  # Opcional: tipo de distancia a utilizar
  num_trials=10,  # Número de pruebas de hiperparámetros
  max_parallel_trials=2,  # Número máximo de pruebas en paralelo
  hparam_tuning_algorithm='VIZIER_DEFAULT'  # Algoritmo de sintonización de hiperparámetros
) AS
SELECT
  aluminium,
  ammonia,
  arsenic,
  barium,
  cadmium,
  chromium,
  copper,
  flouride,
  bacteria,
  viruses,
  lead,
  nitrates,
  nitrites,
  mercury,
  perchlorate,
  radium,
  selenium,
  silver,
  uranium
FROM
  `{project_id}.{dataset_id}.waterquality`
"""
```

Figura 3.39: Sintaxis de Bigquery para crear el modelo de clustering con k-means.

Además, se especifica que se realizarán 10 pruebas de hiperparámetros (num_trials=10) y que como máximo se ejecutarán 2 pruebas en paralelo (max_parallel_trials=2). El algoritmo utilizado para la sintonización de hiperparámetros es 'VIZIER_DEFAULT', por ser el más potente entre las opciones disponibles. Finalmente, la consulta selecciona las columnas relevantes del conjunto de datos *waterquality* y entrena el modelo basado en estos datos.

Los resultados de la implementación del modelo k-means en BigQuery, véase Figura 3.40, muestran que se realizaron cuatro pruebas, variando el número de clusters entre 2 y 5. El rendimiento del modelo se evaluó utilizando el índice Davies-Bouldin, que como vimos, es una medida de la compacidad y separación de los clusters, donde valores más bajos indican una mejor calidad de clustering. La prueba óptima es aquella con el ID número 3, que utiliza 2 clusters y obtiene el índice Davies-Bouldin más bajo (1.0990). Esto sugiere que dividir los datos en dos clusters proporciona la mejor combinación de compacidad y separación de los clusters. Las pruebas con más clusters (3, 4 y 5) obtienen índices Davies-Bouldin más altos, lo que indica una menor calidad del clustering comparado con la prueba óptima. Estos resultados demuestran la eficacia del modelo k-means en identificar la estructura óptima del conjunto de datos de calidad del agua cuando se ajustan adecuadamente los hiperparámetros.



Trial ID	Number of clusters	Davies-Bouldin index	Davies-Bouldin index (Eval)
1	4.0000	1.9363	1.9363
2	3.0000	1.4245	1.4245
3 (Optimal)	2.0000	1.0990	1.0990
4	5.0000	1.7853	1.7853

Figura 3.40: Resultados del de clustering con k-means en Bigquery.

La implementación del algoritmo k-means en BigQuery no solo es poderosa, sino también accesible y sencilla de usar. La infraestructura de Google Cloud, combinada con la capacidad de crear y entrenar modelos de clustering directamente a través de consultas SQL, hace que este proceso sea eficiente y fácil de manejar incluso para aquellos con conocimientos técnicos limitados. En particular, nuestra configuración del modelo k-means demostró que se pueden definir y ajustar diversos parámetros de configuración con gran facilidad. La sintonización de hiperparámetros, realizada automáticamente por BigQuery utilizando el algoritmo 'VIZIER_DEFAULT', permite determinar el número óptimo de clusters sin necesidad de una intervención manual complicada. Además, los resultados de las pruebas muestran que con solo unos pocos pasos y se indica como óptimo el cluster que proporcionó el mejor resultado, validado por el índice Davies-Bouldin más bajo. Este enfoque resalta la sencillez y la facilidad de uso de BigQuery para tareas de machine learning, permitiendo a los usuarios llevar a cabo análisis avanzados de grandes conjuntos datos.

3.7.3. Comparación de clustering con k-means en Python y Bigquery

Al comparar la implementación del algoritmo K-means en Python y BigQuery, se destacan varias diferencias clave que resaltan la simplicidad y accesibilidad de BigQuery para usuarios con menos conocimientos técnicos.

En primer lugar, la creación del modelo k-means en BigQuery es notablemente sencilla. Con una única consulta SQL, es posible definir el modelo, ajustar los parámetros y entrenar el modelo. BigQuery maneja automáticamente el preprocesamiento de datos, así como el tratamiento de valores nulos, lo cual simplifica significativamente el proceso. Además, la elección automática del número óptimo de clusters mediante la sintonización de hiperparámetros en BigQuery elimina la necesidad de una intervención manual compleja.

Por otro lado, la implementación en Python requiere varios pasos adicionales. Se debe realizar un preprocesado manual de los datos, incluyendo la gestión de valores faltantes, la normalización o estandarización de las características, y la división del conjunto de datos en conjuntos de entrenamiento y prueba. Además, se debe aplicar el método del codo para determinar el número óptimo de clusters, lo que implica una mayor complejidad y más tiempo de configuración.

En cuanto a la evaluación de las métricas de rendimiento, ambas implementaciones utilizan el índice Davies-Bouldin para medir la calidad del clustering. En Python, el modelo inicial arrojó un índice de 0.7617, indicando una calidad aceptable. Sin embargo, al experimentar con la normalización y estandarización de los datos, los índices variaron, mostrando la sensibilidad del modelo a las técnicas de preprocesamiento. En BigQuery, el índice Davies-Bouldin más bajo fue 1.0990 con 2 clusters, sugiriendo que este modelo proporcionaba una combinación óptima de compacidad y separación de clusters. La facilidad de ajuste de los hiperparámetros en BigQuery permite obtener resultados comparables sin necesidad de ajustes manuales extensivos. En términos de tiempos de ejecución, la implementación

en BigQuery toma 128.60 segundos, mientras que la de Python es significativamente más rápida, con 15.47 segundos. Sin embargo, es importante considerar que BigQuery maneja automáticamente grandes volúmenes de datos y realiza múltiples pruebas en paralelo, lo que puede justificar el tiempo adicional.

En resumen, aunque la implementación en Python permite un mayor control y personalización, BigQuery ofrece una solución más accesible y menos técnica para la creación y ajuste de modelos k-means. La automatización del preprocesado y la sintonización de hiperparámetros en BigQuery facilitan el proceso de clustering, haciéndolo ideal para usuarios con menos experiencia técnica y que buscan una manera eficiente de manejar grandes conjuntos de datos.

Capítulo 4

Conclusiones

A lo largo de este trabajo, hemos explorado en profundidad el uso de BigQuery ML como herramienta para la creación de modelos, comparándola con Python, una de las opciones más comunes y flexibles en el campo del aprendizaje automático. En particular, analizamos la facilidad de uso, la integración de diversas funciones y la eficacia de BigQuery ML para gestionar tareas complejas de un forma sencilla. A continuación, a lo largo de este capítulo, presentaremos una recopilación de los aspectos clave.

4.1. Ventajas de BigQuery ML

En esta sección nos centraremos en una visión bastante general de las ventajas y limitaciones derivadas del uso de Bigquery (Google Cloud, 2024).

En primer lugar, de igual modo que a lo largo de este proyecto, debemos destacar su simplicidad y accesibilidad. Una de las principales ventajas de BigQuery ML es su sencillez. Al estar integrado en BigQuery, los usuarios pueden crear, entrenar y evaluar modelos de aprendizaje automático utilizando consultas SQL. Esto elimina la necesidad de trasladar datos entre diferentes sistemas, reduciendo significativamente el tiempo y la complejidad del proceso. Además, BigQuery ML automatiza muchos de los pasos necesarios en la preparación de datos y en la evaluación de modelos, lo que resulta especialmente útil para profesionales que no tienen un conocimiento profundo de ciencia de datos. En relación a los tiempos de desarrollo, BigQuery ML acelera la implementación de modelos al permitir que las transformaciones de datos y la creación de estos se realicen en el mismo entorno. Además cabe destacar otros aspectos, la división automática de datos, el manejo de datos faltantes y la generación automática de métricas de evaluación que agilizan el flujo de trabajo y permiten experimentar con los modelos sin preocuparse por detalles técnicos.

Por otro lado, como se menciona al inicio, Bigquery proporciona interoperabilidad de modelos y simplificación de infraestructura. En particular, sabemos que se clasifica como una plataforma PaaS, lo cuál permite a las empresas y profesionales enfocarse en el análisis de datos sin tener que gestionar la infraestructura subyacente. Esto es una fuerte ventaja en contraste con Python, que a menudo requiere la configuración y mantenimiento de entornos de desarrollo, bibliotecas adicionales y sistemas de orquestación como Docker.

Por último, se debe mecionar que BigQuery ML está diseñado para ser accesible incluso para aquellos que no son expertos en ciencia de datos. La automatización de tareas complejas y la integración con herramientas de Google Cloud hacen que esta plataforma sea ideal para aquellos perfiles menos técnicos.

4.2. Limitaciones de Bigquery ML

A pesar de que esta plataforma ofrece grandes ventajas también presenta ciertas limitaciones que se deben tener en cuenta (Google Cloud, 2024).

Una de estas limitaciones radica en su menor flexibilidad y control en comparación con otras herramientas. Aunque simplifica muchos aspectos del desarrollo de modelos al permitir la creación de modelos a través de consultas SQL, esta simplicidad conlleva una reducción en la capacidad de los usuarios para ajustar los detalles específicos del preprocesamiento de datos y la configuración de los modelos. Para proyectos que requieren una personalización más avanzada, esta falta de flexibilidad puede ser una limitación. Otra limitación notable es la variedad limitada de modelos ofrecidos por BigQuery ML. Si bien proporciona una selección de modelos predefinidos que son adecuados para muchas tareas comunes de aprendizaje automático, esta oferta puede resultar insuficiente para aquellos que necesitan modelos específicos o algoritmos más avanzados. En comparación con Python y sus bibliotecas asociadas, las opciones disponibles en BigQuery ML pueden ser más limitadas, lo que podría restringir las posibilidades de desarrollo en ciertos proyectos. Además, la dependencia de la infraestructura de Google Cloud es otra consideración importante. Si bien la integración con Google Cloud ofrece beneficios significativos en términos de facilidad de uso y escalabilidad, también significa que los usuarios están vinculados al servicio de Google. Esto puede ocasionar costes adicionales, como por ejemplo en el caso de las reservas necesarias para llevar a cabo la factorización de matrices.

En resumen, si bien BigQuery ML ofrece simplificación y eficiencia en muchos aspectos del desarrollo de modelos de aprendizaje automático, estas limitaciones deben ser consideradas cuidadosamente al evaluar su idoneidad para un proyecto específico.

4.3. Ventajas de Python

Python, como lenguaje de programación en el ámbito de la ciencia de datos, presenta una serie de ventajas significativas que lo convierten en una herramienta fundamental para el desarrollo de modelos de aprendizaje automático y análisis de datos (Cutting y Stephen, 2021).

En primer lugar, el control que ofrece Python es un aspecto a destacar. Este lenguaje proporciona a los usuarios un nivel de control detallado sobre cada aspecto del proceso de ciencia de datos. Desde la manipulación de datos hasta la personalización de modelos y la implementación de algoritmos avanzados, Python permite una gran libertad. Los científicos de datos pueden utilizar una amplia colección de bibliotecas y *frameworks*, como `scikit-learn`, `TensorFlow` y `PyTorch`, para crear soluciones que se adapten específicamente a las necesidades de su proyecto. Además, la amplitud de herramientas y modelos disponibles en Python es otro punto a su favor. Este lenguaje cuenta con una alta variedad de bibliotecas que soportan una amplia gama de algoritmos y modelos de aprendizaje automático. Desde técnicas clásicas como regresión lineal hasta enfoques más avanzados como redes neuronales convolucionales. Esta diversidad de herramientas y modelos permite a los científicos de datos seleccionar la herramienta o enfoque más adecuado para su proyecto específico, lo que contribuye a la versatilidad y eficacia de Python en el campo de la ciencia de datos. Por último, la fuerte comunidad y los recursos educativos disponibles para Python son un factor importante a tener en cuenta. La comunidad de Python es extensa y activa, proporcionando una gran cantidad de recursos educativos o foros de discusión. Esto facilita el aprendizaje y la resolución de problemas técnicos, además de promover el intercambio de conocimientos y buenas prácticas dentro de la comunidad. La abundancia de recursos educativos disponibles hace que Python sea accesible para aquellos que deseen aprender y mejorar sus habilidades en ciencia de datos, lo que contribuye a su popularidad y adopción generalizada en la industria.

4.4. Limitaciones de Python

Aunque Python es una herramienta poderosa y versátil en el campo de la ciencia de datos y el aprendizaje automático, también presenta ciertas limitaciones que deben ser consideradas (Cutting y Stephen, 2021).

Una de las principales desventajas de Python es la necesidad constante de mantenimiento y actualización de bibliotecas. Las bibliotecas de Python citadas anteriormente, como `scikit-learn`, `TensorFlow` y `PyTorch`, están en continuo desarrollo, lo que significa que los usuarios deben mantenerse al día con las nuevas versiones y cambios en las APIs. Esto puede ser un desafío, especialmente en proyectos de larga duración, donde las actualizaciones pueden introducir cambios incompatibles que requieren ajustes en el código existente. Además, la gestión de dependencias puede volverse compleja, ya que diferentes proyectos pueden necesitar versiones específicas de las bibliotecas, lo que puede resultar en conflictos y dificultades de integración. Otra limitación significativa de Python es su rendimiento. Python es un lenguaje interpretado, lo que puede hacerlo más lento en comparación con otros lenguajes compilados como C++ o Java, especialmente en tareas que requieren un procesamiento intensivo. Además, en Python resulta complejo manejar grandes volúmenes de datos, lo que requiere conocimientos técnicos avanzados. Esto puede ser una barrera para equipos con menos experiencia en la gestión de infraestructuras de gran escala.

En resumen, aunque Python ofrece un control detallado y una gran flexibilidad en el desarrollo de modelos de aprendizaje automático, su uso implica la necesidad de un mantenimiento continuo de las bibliotecas, posibles problemas de rendimiento y desafíos en la escalabilidad. Estas limitaciones deben tenerse en cuenta al elegir Python como herramienta para proyectos específicos.

4.5. Conclusiones finales

En conclusión, la elección entre BigQuery ML y Python para proyectos de aprendizaje automático depende en gran medida del contexto y de las necesidades específicas del usuario. BigQuery ML sobresale por su simplicidad, accesibilidad y capacidad para acelerar el desarrollo de modelos en un entorno integrado, siendo ideal para profesionales con menos experiencia en ciencia de datos. Por otro lado, Python ofrece una flexibilidad y control mayores, junto con una amplia gama de herramientas y modelos, lo que lo convierte en la opción más común en la industria.

Sin embargo, ambas herramientas tienen su lugar en el mundo del análisis de datos. BigQuery ML es especialmente valioso para organizaciones que buscan implementar rápidamente soluciones de aprendizaje automático sin una gran inversión en recursos técnicos, mientras que Python sigue siendo la opción más robusta para desarrollos complejos que demandan una mayor personalización y control detallado. La clave está en evaluar las necesidades específicas del proyecto y el nivel del equipo para elegir la herramienta más adecuada.

Referencias

Aneiros Pérez, G. (2022). *Series de tiempo.* (Apuntes de la asignatura impartida en el Departamento de Matemáticas, Universidad da Coruña, Máster Universitario en Técnicas Estadísticas, Curso 2022-23)

Arthur, D., y Vassilvitskii, S. (2007, 01). K-means++: The advantages of careful seeding. En (Vol. 8, p. 1027-1035). doi: 10.1145/1283383.1283494

Bock, H.-H. (2007). Clustering Methods: A History of K-means Algorithms. *Selected Contributions in Data Analysis and Classification*, 161–172.

Breiman, L. (1996). Bagging predictors. *Machine Learning*, *24*(2), 123-140.

Breiman, L. (2001). Statistical modeling: The two cultures (with comments and a rejoinder by the author). *Statistical Science*, *16*(3), 199-231.

Chmielnicki, W., y Stąpor, K. (2016). Using the one–versus–rest strategy with samples balancing to improve pairwise coupling classification. *International Journal of Applied Mathematics and Computer Science*, *26*(1), 191-201.

Cowpertwait, P. S. P., y Metcalfe, A. V. (2009). *Introductory Time Series with R* (First ed.). New York: Springer.

Cryer, J., y Chan, K. (2008). *Time Series Analysis: With Applications in R.* Springer New York.

Cutting, V., y Stephen, N. (2021, 08). A Review on using Python as a Preferred Programming Language for Beginners. , *8*, 4258-4263.

Faraway, J. J. (2015). *Linear Models with R (2nd ed.).* Boca Raton, FL: Chapman & Hall/CRC.

Faraway, J. J. (2016). *Extending the Linear Model with R: Generalized Linear, Mixed Effects and Nonparametric Regression Models.* Boca Raton: CRC Press.

Fernández-Casal, R., Costa Bouzas, A., y Oviedo de la Fuente, M. (2021). *Aprendizaje estadístico.* (Edición: Septiembre de 2021)

Fox, J. (2008). *Applied Regression Analysis and Generalized Linear Models (2nd ed.).* Los Angeles: Sage.

Friedman, J. H. (2001). Greedy Function Approximation: A Gradient Boosting Machine. *The Annals of Statistics*, *29*(5), 1189-1232.

Gemulla, R., Nijkamp, E., Haas, P. J., y Sismanis, Y. (2011). Large-scale Matrix Factorization with Distributed Stochastic Gradient Descent. En *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (p. 69–77). New York, NY, USA: Association for Computing Machinery.

Google Cloud. (2024). *Introducción a BigQuery ML.* Descargado de `https://cloud.google.com/bigquery/docs/bqml-introduction?hl=es-419`

Guan, X., Li, C.-T., y Guan, Y. (2017). Matrix Factorization With Rating Completion: An Enhanced SVD Model for Collaborative Filtering Recommender Systems. *IEEE Access*, *5*, 27668-27678.

Harper, F. M., y Konstan, J. A. (2015, December). The movielens datasets: History and context. *ACM Transactions on Interactive Intelligent Systems (TiiS)*, *5*(4), 19.

Hastie, T. J., Tibshirani, R. J., y Friedman, J. (2009). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction* (2nd ed.). New York: Springer.

Hyndman, R., y Athanasopoulos, G. (2018). *Forecasting: Principles and Practice* (2nd ed.). Australia: OTexts.

Kaggle. (2019). *Water quality.* Descargado de `https://www.kaggle.com/datasets/mssmartypants/water-quality`

Karani, D. (2022). *Roles in ML team and how they collaborate with each other.* Descargado de `https://neptune.ai/blog/roles-in-ml-team-and-how-they-collaborate`

Karimi, Z. (2021). *Confusion Matrix.*

Koren, Y., Bell, R., y Volinsky, C. (2009). Matrix Factorization Techniques for Recommender Systems. *Computer*, *42*(8), 30-37.

Liaw, A., y Wiener, M. (2002). Classification and Regression by RandomForest. *R News*, *2/3*, 18-22.

Makridakis, S. (1996, December). Forecasting: its role and value for planning and strategy. *International Journal of Forecasting*, *12*(4), 513-537.

Miguel, S. (2021). *Guía completa para el manejo de datos faltantes.* Descargado de `/blog/manejo-datos-faltantes/`

Morissette, L., y Chartier, S. (2013, 02). The K-means Clustering Technique: General Considerations and Implementation in Mathematica. *Tutorials in Quantitative Methods for Psychology*, *9*, 15-24. doi: 10.20982/tqmp.09.1.p015

Na, S., Yong, G., y Xumin, L. (2010, apr).
En *Intelligent Information Technology and Security Informatics, International Symposium on research on K-means Clustering Algorithm: An Improved K-means Clustering Algorithm* (p. 63-67). Los Alamitos, CA, USA: IEEE Computer Society.

Nash, W., Sellers, T., Talbot, S., Cawthorn, A., y Ford, W. (1995). *Abalone.* Descargado de `https://doi.org/10.24432/C55C7W`

Norman, A. T. (2019). *Aprendizaje Automático En Acción: Un Libro Para El Lego, Guía Paso A Paso Para Los Novatos.* Italy: Tektime.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, *12*, 2825–2830.

Peña, D. (2005). *Análisis de Series Temporales.* Alianza.

Schapire, R. E. (1999). A Brief Introduction to Boosting. En *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence.*

Schapire, R. E. (2002). The Boosting Approach to Machine Learning: An Overview. En *MSRI Workshop on Nonlinear Estimation and Classification.*

Smith, T. G., y cols. (2017). *pmdarima: Arima estimators for Python.* Descargado de `http://www.alkaline-ml.com/pmdarima`

Svetunkov, I. (2023). *Forecasting and Analytics with the Augmented Dynamic Adaptive Model (ADAM).* Chapman and Hall/CRC. doi: 10.1201/9781003452652

Tatachar, A. V. (2021). Comparative Assessment of Regression Models Based On Model Evaluation Metrics. *International Research Journal of Engineering and Technology (IRJET)*, *8*(9), 853-860.

Umargono, E., Suseno, J., y Gunawan, S. (2020, 01). K-means Clustering Optimization Using the Elbow Method and Early Centroid Determination Based on Mean and Median Formula.. doi: 10.2991/assehr.k.201010.019

USE OF NOAA GML DATA. (s.f.). Available online: gml.noaa.gov/ccgg/trends/. Descargado de `gml.noaa.gov/ccgg/trends/`

Vergani, A. A., y Binaghi, E. (2018). A Soft Davies-Bouldin Separation Measure. En (p. 1-8).

Vujović, (2021). Classification Model Evaluation Metrics. *International Journal of Advanced Computer Science and Applications*, *12*(6), 1. doi: 10.14569/IJACSA.2021.0120670

Wijaya, Y. A., Kurniady, D. A., Setyanto, E., Tarihoran, W. S., Rusmana, D., y Rahim, R. (2021). Davies Bouldin Index Algorithm for Optimizing Clustering Case Studies Mapping School Facilities. *TEM Journal*, *10*(3), 1099–1103. doi: 10.18421/TEM103-13

Xgboost documentation xgboost 2.1.0-dev documentation (Vol. 2024) (n.º Mar 28,). (s.f.). Descargado de `https://xgboost.readthedocs.io/en/latest/#`