Universidad ORT Uruguay

Facultad de Ingeniería

Investigación de mercado y aplicación práctica de MLOps: machine learning operations

Entregado como requisito para la obtención del título de Master en Big Data

Diego Rivero - 249082

Tutora: Mikaela Pisani

2022

Declaración de autoría

Yo, Diego Rivero declaro que el trabajo que se presenta en esta obra es de mi propia mano. Puedo asegurar que:

- La obra fue producida en su totalidad mientras realizaba el Proyecto Final del Master en Big Data;
- Cuando he consultado el trabajo publicado por otros, lo he atribuido
con claridad;
- Cuando he citado obras de otros, he indicado las fuentes. Con excepción de estas citas, la obra es enteramente mía;
- En la obra, he acusado recibo de las ayudas recibidas;
- Cuando la obra se basa en trabajo realizado conjuntamente con otros, he explicado claramente que fue contribuido por otros, y que fue contribuido por mi;
- Ninguna parte de este trabajo ha sido publicada previamente a su entrega, excepto donde se han realizado las aclaraciones correspondientes.

20-12-2022

Abstract

La metodología *MLOps* ha cobrado gran importancia dentro del área de ciencia de datos los últimos años, motivada por la necesidad desde la industria de capitalizar la inversión en modelos de aprendizaje automático. Para ello es necesario integrar los modelos a los sistemas productivos, de forma mantenible y escalable, para que así puedan generar un impacto económico real.

La evolución se ha visto impulsada por la creación de muchas herramientas orientadas a no solo desarrollo de modelos, sino también a su integración y despliegue continuo. En este trabajo se hizo un análisis de las herramientas líderes del mercado, para finalmente elegir VertexAI de Google y realizar un proyecto utilizando la herramienta.

El caso de negocio consistió en generar un *pipeline* que automatice la totalidad del ciclo de vida de un modelo de aprendizaje automático. Desde la ingesta de los datos hasta el despliegue del modelo. El problema es de regresión y consiste en predecir el precio de un auto o camioneta, basado en las publicaciones activas de Mercadolibre, utilizando un modelo supervisado.

Desde Vertex AI se accede a los datos a través de la API pública de Mercadolibre, dichos datos se limpian y transforman en *features,* se entrena un modelo utilizando el componente predefinido de *AutoML* de VertexAI y finalmente se despliega el modelo si el mismo supera los umbrales mínimos en sus *KPI*.

Como resultado se obtuvo un buen modelo que se puede mantener en el tiempo con poca intervención de un experto. Además se hizo un uso exhaustivo de la herramienta VertexAI, líder en el mercado para *MLOps* en la nube.

Como conclusiones se validó la ideología *data-centric AI* para lograr el desarrollo de un buen modelo. Se validó la herramienta VertexAI para lograr un proyecto de *MLOps* y se alcanzó el objetivo de implementar un proyecto de aprendizaje automático *end to end*.

Por último se plantean dos opciones para extraer valor real del modelo, construyendo herramientas de software que utilizarían al modelo como centro de sus funcionalidades.

Palabras clave

Google Cloud; Aprendizaje Automático; GCP ; Ciencia de datos; AutoML; MLOps; Regresión; Supervisado; Vertex; Pipeline; Nube; Kubeflow;

Índice general

Glosario

- **Ciencia de datos:** campo de la matemática aplicada y la estadística que provee información útil basada en el procesamiento de grandes cantidades de datos.
- **Analítica de datos:** disciplina que analiza los datos pasados para sacar conclusiones sobre esa información.
- **Inteligencia artificial (IA):** rama de la ciencia que permite que las máquinas piensen, aprendan y encuentren soluciones tal como lo hacen los humanos
- **Aprendizaje automático:** Subárea de la inteligencia artificial que consiste en desarrollar algoritmos con la capacidad de aprender y mejorar a través de experiencias, sin necesidad de programarlas explícitamente.
- **Procesamiento de lenguaje natural:** es una forma de IA que da a las máquinas la capacidad no solo de leer, sino de entender e interpretar el lenguaje humano
- **Visión de computador:** es una forma de IA que da a las máquinas la capacidad de procesar y entender los diferentes componentes de una imagen o conjunto de imágenes.
- **AutoML:** se le llama AutoML a las diversas herramientas que automatizan la calibración de modelos de aprendizaje automático.
- **Inteligencia artificial responsable**: Es la práctica de diseñar, desarrollar e implementar IA con la intención de impactar de manera justa a los clientes y la sociedad, reduciendo lo mayor posible los sesgos en los datos.
- **dataframe:** es un conjunto de datos. En ciencia de datos también se refiere a un tipo de objeto de la librería pandas de python y de el lenguaje R.
- **features:** son las variables de entrada que se utilizan para entrenar el modelo.
- **benchmarking:** consiste en comparar distintas herramientas / soluciones. En IA se suele utilizar el término para referirse a un modelo sencillo que debe ser superado como mínimo requisito de un proyecto.
- **varianza**: es una medida de la dispersión de los datos. A mayor varianza, mayor dispersión.
- **sesgo**: es la diferencia que se da entre la teoría y la realidad.
- **SDK**: Un kit de desarrollo de software (SDK) es un conjunto de herramientas proporcionado usualmente por el fabricante de una plataforma de hardware, un sistema operativo o un lenguaje de programación.
- **Kubernetes**: es una plataforma de código abierto para automatizar la implementación, el escalado y la administración de aplicaciones en contenedores.
- **Kubeflow**: Es una plataforma de código abierto nativa de Kubernetes para desarrollar, organizar, implementar y ejecutar cargas de trabajo de aprendizaje automático escalables y portátiles.

- **input:** son los parámetros de entrada otorgados a una función, un componente, un modelo, entre otros.
- **output:** son los elementos de salida, que surgen como resultado de procesar los input en la función, componente o modelo.
- **bucket:** es un contenedor para almacenar datos en la nube.

1 Introducción

1.1 Caso de estudio

El proyecto consiste en desarrollar un modelo de regresión para predecir el precio de un automóvil en base a ciertas características del mismo.

La predicción está basada en las publicaciones activas de vehículos de MercadoLibre. Por lo tanto, se espera que dicha predicción responda bien a principios importantes de la economía, como son la variación de precios de un bien acorde a la economía local, y a la oferta y demanda del mismo.

El modelo será desarrollado utilizando la metodología *MLOps*, haciendo énfasis en su mantenibilidad y escalabilidad.

Para ello, se realiza un análisis de mercado de las plataformas más importantes de *MLOps* y posteriormente se elige una herramienta basándose en dicho análisis.

La primera etapa del desarrollo del modelo será en carácter de investigación, buscando la combinación de variables que genere mejores resultados, así como la mejor combinación de parámetros y/o hiper-parámetros del modelo.

Una vez que la investigación del modelo y sus variables esté culminada, se procede con la construcción de la arquitectura del modelo en producción.
La misma no va a incluir elementos de investigación, y va a estar enfocada en desplegar el modelo de la forma más eficiente.

Finalmente, se programan pruebas automatizadas buscando asegurar que el modelo esté performando con el mínimo aceptable antes de ser puesto en producción, y que la arquitectura está funcionando correctamente.

1.2 Motivaciones personales

La motivación que llevó a elegir esta temática es complementar los amplios conocimientos de desarrollo de modelos adquiridos en la maestría, con conocimientos de despliegue de modelos.

Si bien en la industria los científicos de datos se suelen dedicar a desarrollar los modelos y los ingenieros de aprendizaje automático a su despliegue, es muy buena práctica que ambos tengan conocimiento sobre el rol del otro. De esa forma el equipo multidisciplinario puede trabajar de forma mucho más coordinada y lograr mejores resultados.

Se eligió la metodología *MLOps* por su gran crecimiento en la industria y el desafío técnico que conlleva llevarla a la práctica.

Teniendo un título de grado de ciencias económicas, este desafío técnico puede ser la elección más desafiante para mi desarrollo personal, ya que es lo más alejado a mis bases académicas previas a la realización de la maestría. Por ello es una gran oportunidad de crecimiento personal y de consolidación del conocimiento adquirido sobre grandes datos y aprendizaje automático.

1.3 Objetivos

Los objetivos establecidos para este proyecto son los siguientes:

1. Crear una solución de aprendizaje automático de punta a punta: desde el planteamiento del problema desde la perspectiva de negocio hasta el modelo desplegado en producción.
2. Diseñar la solución en base a la metodología de *MLOps*, de forma que el modelo se mantenga actualizado en el tiempo con la menor intervención humana posible.
3. Lograr un modelo aceptable en cuanto a sus métricas de predicción
4. Realizar un análisis de las herramientas disponibles en la industria para llevar a cabo el proyecto
5. Desarrollar conocimiento en la herramienta escogida

1.4 Necesidades desde la industria

Una encuesta realizada en 2018 por Databricks [1], generaba la siguiente conclusión:

“Si bien la IA es la tendencia más candente en tecnología, el ritmo de adopción e implementación real es una historia diferente. En particular, solo 1 de cada 3 proyectos de IA tienen éxito . Esto significa que la gran mayoría de los proyectos no se completan a tiempo, superan los presupuestos previstos y no producen los resultados previstos.”

Además, en 2019, Venturebeat informó en el artículo [2] que solo el 13 % de los modelos de aprendizaje automático llegan a producción.

Por otro lado, en una encuesta en KDNuggets en 2022 [3], el 79% de los votantes afirmaron que entre el 0% y 40% de los modelos desarrollados por ellos llegaron a producción.

Si bien los números varían, todos los artículos tienen el consenso de que hay una gran brecha entre desarrollar un modelo y realmente llegar a desplegarlo.

2 Marco Teórico

2.1 Ciencia de datos, analítica de datos, inteligencia artificial y aprendizaje automático.

La ciencia de datos es un campo de la matemática aplicada y la estadística que provee información útil basada en el procesamiento de grandes cantidades de datos.

[4] Mientras que la inteligencia de negocio se caracteriza por explicar qué pasó, la ciencia de datos suele responder a preguntas cómo “por qué pasó eso”, “cómo accionamos sobre ello” y “qué va a pasar a continuación”.

[4] La inteligencia artificial es la simulación de la inteligencia humana en máquinas. La IA permite que las máquinas piensen, aprendan y resuelvan problemas de forma similar a cómo lo haría un cerebro humano.

[5] El aprendizaje automático es una rama de la inteligencia artificial que utiliza algoritmos para extraer datos y luego predecir tendencias futuras. El software está programado con modelos que realizan análisis estadísticos para comprender patrones en los datos.

Los modelos de aprendizaje automático son sistemas complejos, ya que tienen todas las complejidades de cualquier software, sumado a las complejidades de datos que cambian todo el tiempo. Para lograr desplegar un modelo, se requiere de un equipo multidisciplinario que debe realizar varios pasos desde el entendimiento del problema de negocio hasta el despliegue y monitoreo del modelo en producción.

2.2 Ciclo de vida de un modelo de aprendizaje automático

Existen varios pasos en el ciclo de vida de un modelo de aprendizaje automático, comenzando con un problema de negocio y terminando en un modelo desplegado en producción.

Si bien los procesos pueden variar según el tipo de problema a resolver, la gran mayoría de los desarrollos de aprendizaje automático incluyen los siguientes pasos:

1. Definición del problema de negocio.
2. Obtención y análisis de los datos.
3. Preparación y transformación de los datos.

4. Entrenamiento y tuneo del modelo y sus hiperparámetros.
5. Evaluación y despliegue del modelo.

Figura 2.1: Flujo de aprendizaje automático

A su vez dichos pasos se subdividen en varios otros. A continuación se incluirá una explicación detallada de cada uno de los mismos [6]

2.2.1 Definición del problema de negocio

Figura 2.2: Flujo de aprendizaje automático - Definición del problema de negocio

2.2.1.1 Contexto de negocio y definición del problema

El primer paso consiste en comprender el contexto en el negocio. Eso implica hacer las preguntas correctas, como por ejemplo: qué precisa ser resuelto, con qué nivel de precisión, con qué urgencia y cuáles son los recursos disponibles.

Una vez que se hicieron las preguntas necesarias y se obtuvo un consenso del equipo sobre la respuesta de las mismas, se puede avanzar en la definición formal del problema a resolver, así como las expectativas y desafíos del mismo.

2.2.1.2 Traducir el problema a un problema de aprendizaje automático

El paso siguiente es entender si es un problema que se podría resolver con un modelo de clasificación o regresión, y el nivel de complejidad del mismo.

También es importante entender la posible ganancia económica de la implementación, para en base a eso definir si puede ameritar la construcción de un modelo de aprendizaje profundo, que suele ser costoso, o es mejor ir por una

solución más sencilla como puede ser una regresión lineal si algunos puntos más de precisión no implican una ganancia mucho mayor en términos económicos.

2.2.1.3 Planificación y definición de KPI

Planificación de un cronograma, definición de los recursos computacionales y humanos disponibles, y también definiciones de los indicadores clave (KPI) que determinarán el éxito o fracaso del proyecto.

2.2.2 Obtención y análisis de los datos

Figura 2.3: Flujo de aprendizaje automático - Obtención y análisis de los datos

2.2.2.1 Recopilación y comprensión de datos

El siguiente paso es el referido a la obtención de los datos.

Es fundamental comprender cómo se va a consumir la data que será el input para los modelos y gestionar los permisos para acceder a la data.

Al momento de consumir los datos hay varios aspectos que pueden variar, uno de ellos es la fuente desde donde se consumen. Puede estar en una *API*, en un *data warehouse,* en un *data lake,* en csv, entre otros formatos.

Además otro aspecto a tomar en cuenta es la estructura de los datos: puede ser data tabular, o datos no estructurados como texto o imágenes.

Otro punto importante es entender si el SLA (service level agreement) de la actualización de la data es suficiente para lo que precisa el modelo. Por ejemplo, podemos precisar un modelo en casi tiempo real, pero tener la data actualizada a día vencido. En ese caso el primer paso es aumentar la frecuencia de actualización de la data, o directamente consumirla desde otra fuente.

2.2.2.2 Unificación de los datos

En caso de que haya varias fuentes diferentes de datos, se deberá acceder a cada una de ellas y unificar los datos en un mismo dataframe (siempre que sean datos estructurados).

En el caso de imágenes, también pueden recopilarse de diversas fuentes y ser guardadas en una misma carpeta por ejemplo.

2.2.3 Preparación y transformación de los datos

Figura 2.4: Flujo de aprendizaje automático - Definición del problema de negocio

La transformación de los datos suele ser una de las etapas más desafiantes y donde el científico de datos pasa más tiempo para construir su modelo.

Como la transformación depende mucho de los datos de entrada, es una de las tareas que menos se ha podido automatizar con el avance de la tecnología, y muchas veces es donde la persona experta puede agregar mayor valor.

2.2.3.1 Limpieza de datos

La limpieza de datos incluye algunos pasos estandarizados que suelen aplicar a la gran mayoría de datasets, y algunos más personalizados dependiendo de la data que esté utilizando. Algunos de los pasos más comunes son:

2.2.3.1.1 Datos duplicados

Puede suceder que los datos contengan valores duplicados, es común que este factor se deba a errores en el data entry o al combinar diferentes datasets.

En el caso más sencillo, si el array es totalmente igual, se procede a eliminar los duplicados.
También existen casos más complejos, por ejemplo puede suceder que un mismo id de un cliente tenga más de una fila asociada ya que tiene dos domicilios. En dichos casos se debe definir un criterio de negocio y utilizarlo no solo en ese caso particular, sino en todos los datos de dicha columna que presenten el mismo inconveniente.

2.2.3.1.2 Validación de datos

La validación de datos consiste en verificar la coherencia de los mismos. Por ejemplo, en las predicciones de precio de un ítem no debería haber precios negativos.

2.2.3.1.3 Valores nulos

El análisis de nulos se suele hacer columna a columna, si una columna tiene un gran porcentaje de nulos, se suele eliminar.

Es importante evaluar si los nulos son efectivamente datos faltantes, o si son en sí mismo un dato. Por ejemplo, si en una columna booleana el dato es *True* o *NULL*, posiblemente el *NULL* sea equivalente a *False*, y habría que validarlo con la fuente de los datos.

En caso de que haya pocos nulos, se pueden evaluar diferentes técnicas de imputación, como puede ser imputar la media o la mediana según la distribución de los datos.

2.2.3.1.4 Unificar nomenclaturas

Al unir datos desde diferentes fuentes, hay que verificar que tengan la misma nomenclatura. Por ejemplo, una marca podría llamarse de una forma en un sistema y de otra en otro, ya que podría tener previo a la marca el nombre del distribuidor concatenado por ejemplo.

2.2.3.2 Análisis exploratorio de los datos (EDA)

El análisis exploratorio se utiliza para conocer la distribución de los datos.

No existe una forma única de realizar el análisis pero se suele comenzar por análisis univariado, es decir analizar de a una variable.
Para ello se suelen graficar histogramas y diagramas de caja.

Es común que una variable tenga siempre el mismo valor, en cuyo caso se procede a eliminarla, ya que no aporta valor para las predicciones del modelo.

También es común realizar análisis multi-variado. Es decir, cómo se relacionan unas variables con otras. En esta etapa se suelen utilizar gráficos de dispersión y matrices de correlación.

Además se puede observar los datos de forma agrupada en tablas.

En todo el proceso se debe ir tomando nota de los patrones descubiertos para avanzar a accionar en la selección de features.

2.2.3.3 Ingeniería y selección de features

Esta etapa consiste en seleccionar las features que ayudarán al modelo a realizar la mejor predicción. Para ello son fundamentales los insights obtenidos en el EDA.

Además, suele ser una etapa de mucha iteración, ya que los modelos lineales así como los de árboles, otorgan la posibilidad de entender cómo influyeron las *features* en sus resultados (feature importance). Es una buena práctica utilizar esta información para volver unos pasos atrás e iterar sobre las *features* a utilizar.

A medida que se automatiza la creación de modelos, con librerías como *sklearn* o *AutoML* que cobran cada vez más importancia en el mercado, pasa a ser la ingeniería de atributos un paso fundamental para obtener buenas predicciones. Esto es parte del nuevo paradigma del aprendizaje automático que solía ser centrado en los modelos y actualmente está siendo centrado en los datos.

2.2.3.4 Preparación de datos para modelar

2.2.3.4.1 Creación de variables booleanas

La mayoría de los modelos no pueden procesar texto, por lo que para las variables categóricas que tengan diferentes strings como posibles valores, se utiliza la técnica de one hot encoding.

La misma consiste en generar una variable para cada uno de los posibles valores y poner un dato de 1 o *True*, si esa fila tenía ese valor, y 0 ó *False* en caso contrario.

2.2.3.4.2 Oversampling y undersampling

Existen técnicas como *oversampling* y *undersampling* que son utilizadas cuando el conjunto de datos está desbalanceado, es decir cuando no es pareja la cantidad de datos que hay para cada categoría de predicción.

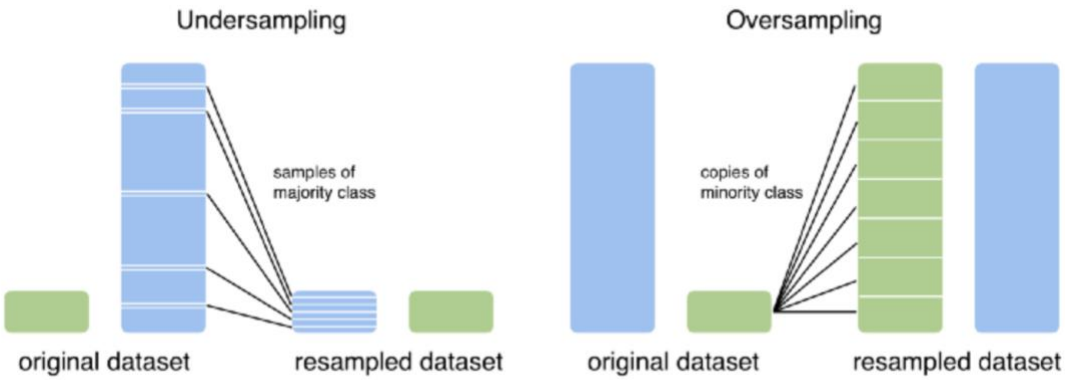


Figura 2.5: Técnicas de balanceo. Fuente: [7]

2.2.3.4.3 Dividir datos en train y test

Los datos son divididos en 2 conjuntos: entrenamiento y testeo, para que el modelo sea entrenado con los datos de entrenamiento y el conjunto de testeo sea nuevo para el mismo. Se suele utilizar un 20% o 30% de los datos para testeo.

2.2.4 Entrenamiento y calibración del modelo

Figura 2.6: Flujo de aprendizaje automático - Entrenamiento y tuneo del modelo

2.2.4.1 Construcción del modelo

Una vez que tenemos prontos los datos, se puede comenzar la construcción del modelo.
En general los modelos con mayores capacidades de predicción son los menos explicativos, por ello según el problema de negocio a resolver se elegirá un modelo más preciso pero menos explicativo o viceversa.

Los modelos lineales (regresiones) son los más explicativos y suelen ser menos precisos que los demás, aunque pueden ser una buena forma de generar un mínimo al cual los demás modelos deban superar.

Sin embargo, en muchos casos de negocio pueden tener una precisión aceptable si el *feature engineering* fue realizado adecuadamente, y en la realidad tienen una alta adopción por las compañías.

En la misma línea, es buena práctica generar un modelo con *AutoML*, casi sin costo de desarrollo, para utilizar como *benchmarking*, es decir como unos kpi mínimos a superar.

Los modelos de árboles (*Random Forest, xgboost, catboost, lightboost*, entre otros) suelen tener mayor precisión, manteniendo un alto nivel de explicabilidad y sin generar una complejidad computacional muy grande, por lo que también son muy utilizados.

Por otro lado, los modelos de redes neuronales son los que pueden generar una mayor precisión en la teoría, especialmente cuando se cuenta con muchos datos. Además pueden manejar mejor el ruido en los datos, por lo que si bien el *feature engineering* sigue siendo muy importante, puede ser corregido por el mismo modelo al darle diferentes pesos a cada variable.

Pueden ser complejos computacionalmente de llevar a producción, especialmente en tiempo real, y no tienen una gran explicabilidad, lo que disminuye su uso en ciertos casos de negocio donde se precisa ejecutar con una arquitectura sencilla o con una gran explicabilidad, como por ejemplo en casos socioeconómicos como la solicitud de un crédito, donde es importante explicar las razones en caso de que no se otorgue.

2.2.5 Evaluación y despliegue del modelo

Figura 2.7: Flujo de aprendizaje automático - Evaluación y despliegue del modelo

2.2.5.1 Validación y evaluación del modelo

2.2.5.1.1 Testear el modelo

El modelo fue entrenado con los datos de entrenamiento, por lo que en esta etapa se evaluará con los datos de testeo que el modelo aún no ha visto. Es decir que se va a testear su capacidad de generalizar sobre datos que aún no ha visto.

Según el caso de negocio se pueden utilizar diferentes métodos de evaluación. Para los problemas de clasificación uno de los más usados es la matriz de confusión y para casos de regresión algunos de los métodos más utilizados son *RMSE*, *MAE* Y *R2.*

2.2.5.1.2 Tuneo de parámetros o hiperparámetros

Los parámetros son variables de entrada al modelo, mientras que los hiper parámetros son variables que aprende el modelo por sí mismo.

Existen muchas librerías que se encargan de automatizar esta parte de la construcción del modelo, como puede ser *AutoML*, aunque muchas veces no logran superar a un modelo calibrado por un experto.

En general se genera una primera predicción de forma automática y se utiliza como *benchmark* para superar con conocimiento experto.

El calibrado de parámetros es de lo más importante para construir un buen modelo, y suele requerir muchas iteraciones.

Existen técnicas para utilizar un algoritmo preexistente, así como también existen formas de acelerar el proceso de entrenamiento como pueden ser *gridsearch* y *randomsearch*.

2.2.5.1.3 Cross validation

Cross validation es una técnica de evaluación que consiste en variar los conjuntos de datos que van a ser utilizados para entrenar y para testear.

Es particularmente útil cuando hay pocos datos para entrenar, ya que permite entrenar con todos los datos originales.

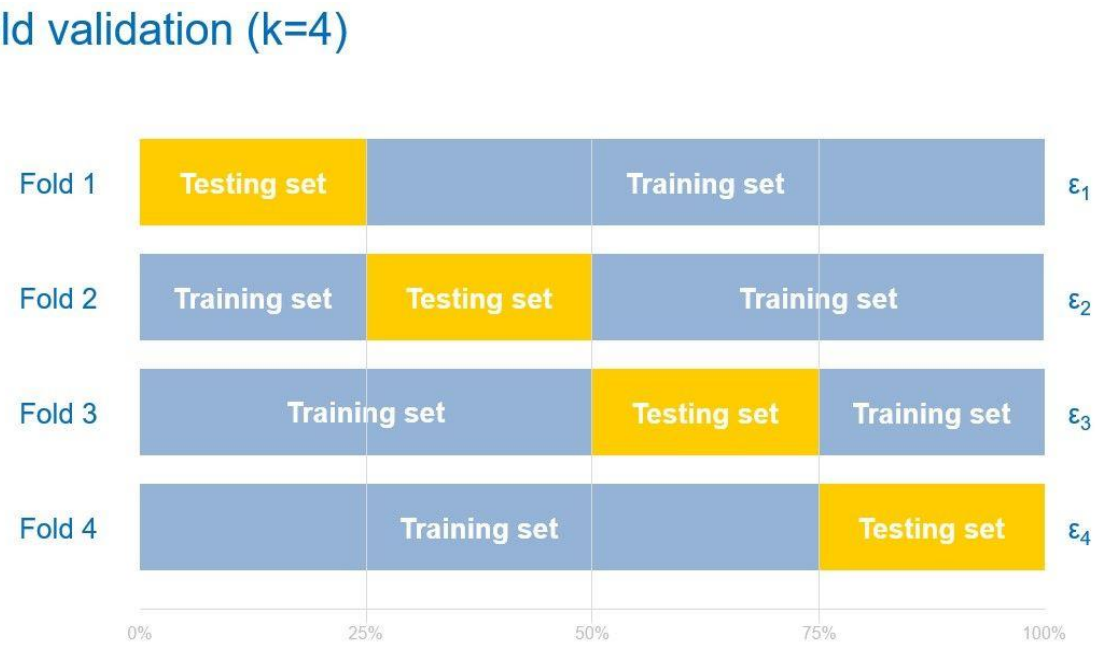


Figura 2.8: Validación cruzada. Fuente [8]

2.2.5.1.4 Tradeoff en métricas de evaluación

Al elegir el mejor modelo es importante encontrar el punto de equilibrio entre sesgo y varianza. Un modelo muy ajustado a los datos de entrenamiento (poco error), no suele generalizar muy bien cuando ve datos nuevos ya que no aprendió patrones, sino que memorizó.

2.2.5.1.5 Overfitting / Underfitting

Overfitting es el caso en que el modelo memoriza los datos de entrenamiento, y por lo tanto no generaliza bien.

Underfitting es cuando el modelo no aprende de forma suficiente de los datos de entrenamiento, en cuyo caso tampoco generaliza bien pero por razones opuestas a overfitting.

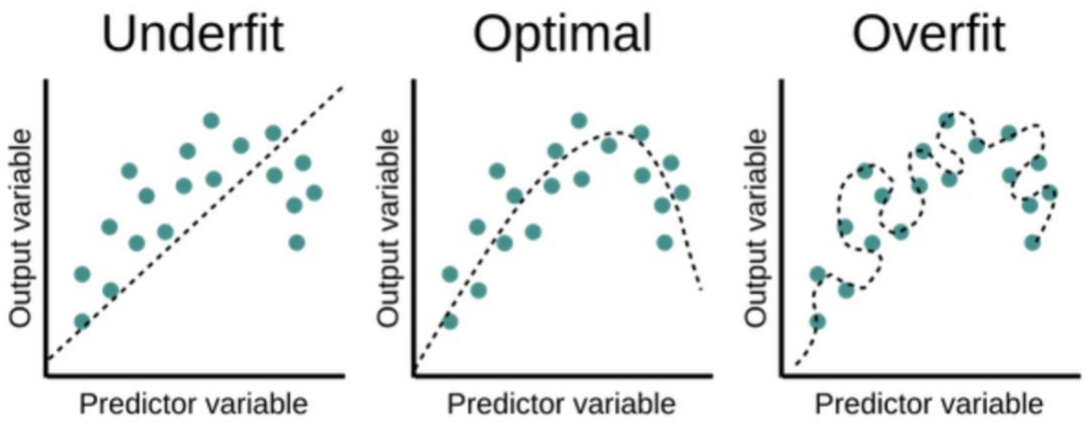


Figura 2.9: Overfitting. Fuente: [9]

2.2.5.2 Puesta en producción del modelo

Muchos de los proyectos de aprendizaje automático no logran pasar de la fase de experimentación a producción.

Pueden existir múltiples desafíos como que los datos de experimentación no estén disponibles en producción, o integrar el modelo a los demás sistemas de la compañía.

Si bien el campo está en constante desarrollo, hoy en día existen múltiples herramientas que se utilizan para poner modelos en producción, algunas de las cuales logran también englobar todo el ciclo de vida de un modelo.

En este documento vamos a analizar a los jugadores más grandes del mercado que pueden ofrecer el servicio de soportar el ciclo de vida completo de un modelo, y en base a sus variables elegir uno para profundizar.

2.2.5.3 Seguimiento del modelo

Una buena práctica es crear tableros de métricas sobre el modelo una vez en producción, para monitorear sus KPI y medirlo de la forma más objetiva posible, así como tomar acción si el modelo disminuye su rendimiento.

En ciertas industrias que evolucionan constantemente, como la de prevención de fraude, es común que un modelo pierda performance con el correr del tiempo, ya que el fraude va evolucionando.

Además las empresas pueden cambiar sus procesos, lo que puede cambiar la distribución de los datos y generar una caída en la performance del modelo si el mismo no es mantenido adecuadamente.

2.3 MLOps

En el capítulo anterior se mencionó la necesidad de mantener los modelos para asegurar su performance, pero hacerlo manualmente puede tener un costo muy alto, especialmente si se toma en cuenta el alto costo de tener un equipo de expertos en ciencia de datos, así como la gran cantidad de modelos que puede tener una compañía en producción.

Por ello muchas empresas están apostando a MLOps, es decir utilizar una metodología similar a DevOps pero orientada a aprendizaje automático, basada en el mantenimiento automático de los modelos.

Según [10] "*DevOps* se basa en herramientas, automatización y flujos de trabajo para abstraer la complejidad accidental y permitir a los desarrolladores centrarse en los problemas reales que deben resolverse. *DevOps* ayuda a romper la barrera entre desarrollo (Cambios constantes) y operaciones (la necesidad de mantener estable el sistema)".

No podemos simplemente utilizar *DevOps* para los modelos de *machine learning (ML)* ya que hay una diferencia fundamental entre *ML* y el software tradicional: *ML* no es solo código, es código más datos.

[10] "*MLOps* es un conjunto de prácticas que combinan *Machine Learning*, *DevOps* e Ingeniería de datos, cuyo objetivo es implementar y mantener sistemas *ML* en producción de manera confiable y eficiente."

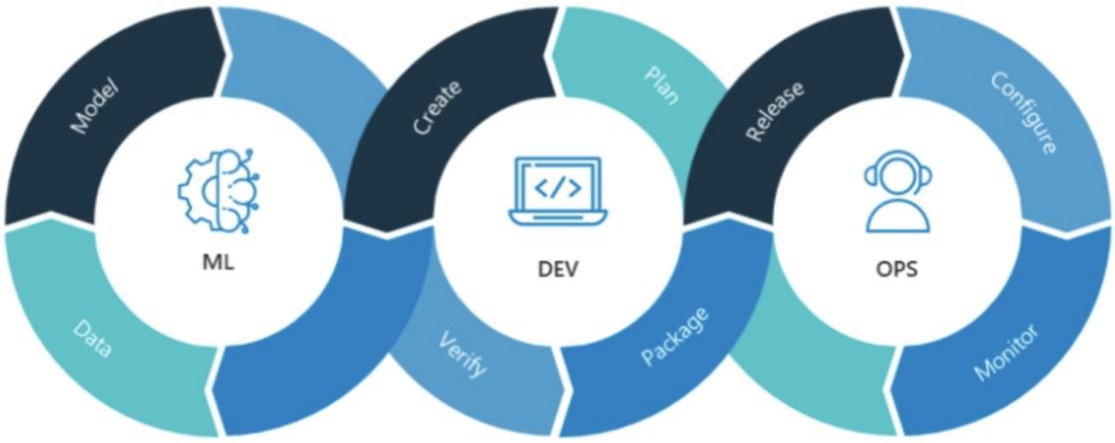


Figura 2.10: MLOps. Fuente: [9]

2.4 Buenas prácticas de MLOps

De acuerdo al artículo [10] MLOps es la combinación de las siguientes buenas prácticas:

2.4.1 Equipos híbridos

Si bien puede suceder, no es común que un científico de datos tenga los conocimientos de ingeniería para realizar un modelo de *MLOps*, por lo que en general debería estar acompañado de un ingeniero de datos y de un ingeniero de *DevOps*.

Para poder trabajar en conjunto, si bien cada rol tiene sus responsabilidades definidas, todos deberían tener una idea básica del rol de sus compañeros. Es decir, si bien el científico de datos no necesariamente tiene que saber hacer el trabajo ingenieril por sí solo, sí debería manejar ciertos conceptos como modularización, reutilización, prueba y control de versiones de código para poder trabajar en equipo de forma eficiente.

2.4.2 Flujos de ML

Los flujos de datos, también llamados *pipelines* de datos o procesos *ETL* (extraer, transformar y cargar), son procesos en que la data es obtenida desde una fuente, para ser transformada y luego cargada en la misma fuente o en otro sistema.

Las transformaciones de los datos se refiere a todos los cambios necesarios sobre los mismos, para que quede con el formato que es requerido en el destino final (el modelo de *ML*).

Por ejemplo, si la extracción es desde diversas fuentes, la transformación consistiría en combinar dicha data y pasar todo a la misma nomenclatura. Un ejemplo donde esta transformación sería necesaria podría ser un caso en el que un sistema tiene en su campo de marcas del producto un prefijo con el nombre del proveedor, y en el otro sistema no tiene ese prefijo. Se le debería agregar el prefijo al último sistema o quitárselo al primero, para poder agrupar los datos por marcas de forma correcta.

El entrenamiento de un modelo de aprendizaje automático también puede considerarse como una transformación de los datos, de ahí el nombre de flujos de *machine learning (ML)*.

La mayoría de los modelos precisarán dos flujos de *ML*, el primero sería un flujo construido para el entrenamiento del modelo, y el segundo uno para el modelo que realiza predicciones en producción. Esto se debe a que la forma de acceder a los datos para entrenar un modelo pueden ser muy diferentes a la forma de acceder a los mismos datos en producción, especialmente si el modelo realiza predicciones en tiempo real.

Por ejemplo, podría suceder que el modelo sea entrenado con datos tabulares en una base de datos, pero que en producción deba invocar directamente a las APIs que dan origen a los datos en dicha base.

2.4.3 Versionado de modelos y de datos

En el desarrollo de software tradicional, es fundamental el versionado del código para poder volver a versiones anteriores en caso de que los últimos despliegues no funcionaran como se esperaba.

En aprendizaje automático es igual de importante mantener registro de las diferentes versiones del código, pero se le suma también la necesidad de registrar las versiones del modelo, así como de los datos que fueron utilizados para entrenarlo, y de otros datos como hiper-parámetros del modelo.

2.4.4 Validación de modelos

Una parte fundamental de *DevOps* es la automatización de testeos, generalmente pruebas unitarias y pruebas de integración. De pasarse estos testeos automáticos, el *pipeline* avanza con la implementación, y en caso contrario la implementación es detenida. Estas pruebas automáticas son uno de los pasos más importantes de la metodología, ya que tienen un gran impacto en acelerar las implementaciones en producción, manteniendo la confianza en el sistema.

Al testear procesos de software, es claro entender si un proceso funciona como debe o si falla. Sin embargo, al testear un modelo puede ser más complejo definir si es lo suficientemente bueno.

Para lograr esta definición es importante definir *KPI* previo a la medición de la performance, y setear umbrales mínimos aceptables para cada uno.

Para medir un modelo de clasificación es importante tener en cuenta:

- Precisión del modelo: de los casos que el modelo clasificó como TP, cuántos lo fueron lo realmente

- Recall del modelo: la precisión abierta por categorías. Un modelo puede ser muy bueno para detectar ciertos valores pero pésimo en otros. Según el problema de negocio, puede haber algunas categorías que importen más que otras, o puede ser necesario tener un mínimo aceptable en todas. Es importante conocer el problema para tomar definiciones, por ejemplo puede entenderse como menos costoso que un modelo de prevención de fraude notifique un caso como fraude que realmente no lo sea, que no notificar y sufrir la pérdida económica.
- Medición de sesgo en los datos: es importante realizar mediciones que ayuden a entender si el modelo está sesgado, especialmente si se trata de datos socioeconómicos. Por ejemplo, un modelo de selección de personal podría predecir que la gente de orígenes locales son mejores candidatos que los extranjeros, simplemente porque históricamente se ha contratado gente local, es decir que tiene un sesgo en los datos. Una forma de medir este sesgo es analizar los *KPIs* por diferentes cortes, para entender si hay algún subsegmento que esté dando valores muy diferentes y si existe una explicación razonable para ello.
- Casos críticos: un modelo puede tener buena precisión, recall y no estar sesgado, pero no agarrar los casos más importantes en la historia reciente. Por ejemplo los casos de mayor pérdida económica en un modelo de predicción de fraude, o los talentos con mejor performance en un modelo de selección de personal. Es importante evaluar los casos críticos ya que puede ser muy costoso para una compañía si el modelo no es bueno identificándolos.

Por otro lado, para medir un modelo de regresión los siguientes puntos son fundamentales:

- Métricas globales del modelo como RMSE, MAE Y R2.
- Las mismas métricas pero aperturadas por diferentes cortes, para entender si hay algún segmento en el cual el modelo tenga oportunidades de mejora.
- Medición del sesgo en los datos, igual que en los problemas de clasificación.
- Casos críticos, igual que en los problemas de clasificación.

2.4.5 Validación de datos

Lo más importante en un modelo son los datos que recibe. Si los mismos no son correctos, el modelo no podrá realizar buenas predicciones. Hay un dicho muy popular que dice que si entra basura al modelo, saldrá basura del mismo.

La validación de datos consiste de dos niveles.

El más básico es la calidad de los mismos, lo cual incluye verificar cantidad de nulos, el *SLA (service level agreement)* de actualización de los datos, que todos los datos sean del tipo esperado, entre otros.

El nivel más complejo consiste en monitorear la distribución de los datos. Un cambio en la distribución puede ser originada por una falla en alguno de los pipelines o por cambios en los datos .

La imagen siguiente es un ejemplo de métricas clave para poder monitorear la distribución de los datos, proporcionadas por *Tensorflow*

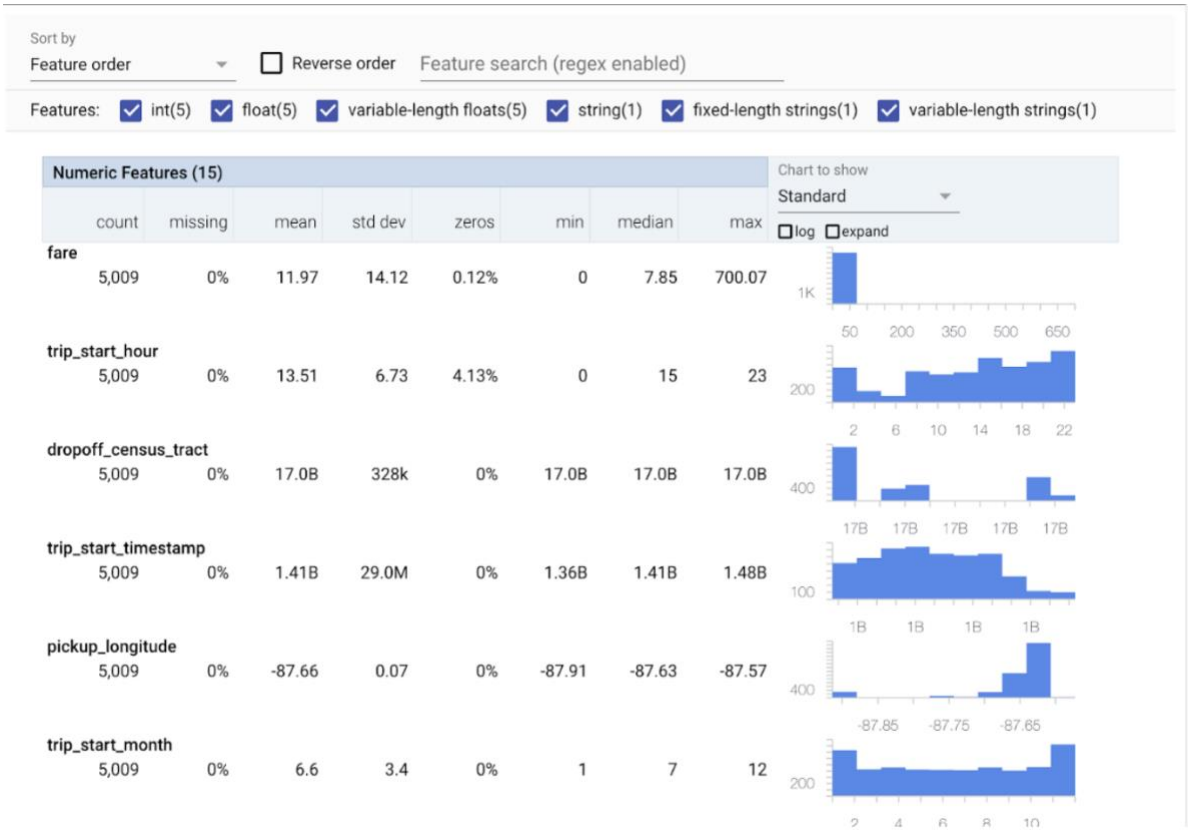


Figura 2.11: Distribución de datos en Tensorflow. Fuente: [11]

2.4.5 Resumen

MLOps es un conjunto de prácticas que tiene muchas similitudes con *DevOps*, con la diferencia principal de que *MLOps* está centrada en datos además de código.

Otra disciplina que tiene a los datos en su eje es la ingeniería de datos, de la cual también *MLOps* toma algunos procesos como inspiración.

De la combinación de datos con el agregado de algunas prácticas particulares de machine learning, se creó el concepto de *MLOps*.

La siguiente imagen muestra un comparativo entre las tres disciplinas:

Práctica	DevOps	Ingeniería de Dados	ML Ops
Control de versiones	Versionamiento de código	Versionamiento de código Linaje de datos	**Versionamiento de código + datos + modelos (conectados)**
Pipeline	n/a	Flujo de datos/ETL	**Flujo de ML de entrenamiento, Flujo de ML para predicción**
Validación de comportamento	Pruebas unitarias	Pruebas unitarias	**Validación de Modelo**
CI/CD	Despliegue de código en producción	Despliega código en flujo de datos	**Despliega código a producción+flujo de datos de ML**
Validación de datos	n/a	Validación de negocio y formato	**Validación estatística**
Monitoreo	Basado en SLOs	Basado en SLOs	**SLOs + monitoreo diferencial y estatístico**

Figura 2.12: DevOps / Inigeniería de datos / MLOps. Fuente: [12]

2.5 La evolución de MLOps

MLOps está en pleno auge y constante evolución, y una forma de medirlo es la gran cantidad de nuevas herramientas que han surgido y siguen surgiendo en el mercado orientadas a ello.

El artículo [13] explica la evolución de las herramientas.

“A principios de la década de 2000, cuando las empresas necesitaban implementar soluciones de aprendizaje automático, usaban el software con licencia de los proveedores, como *SAS, SPSS y FICO*. Con el auge del software de código abierto y la disponibilidad de datos, más profesionales de software comenzaron a usar bibliotecas *Python o R* para entrenar modelos *ML*. Sin embargo, el uso de los modelos en producción seguía siendo un problema. A medida que iba surgiendo la tecnología de contenerización, se resolvió el despliegue del modelo de forma escalable mediante el uso de contenedores *Docker* y *Kubernetes*. Recientemente, vemos la evolución de esas soluciones hacia plataformas de implementación de *ML* que cubren toda la iteración de experimentación, capacitación, implementación y monitoreo de modelos. La siguiente Figura visualiza la evolución de los *MLOps*.”

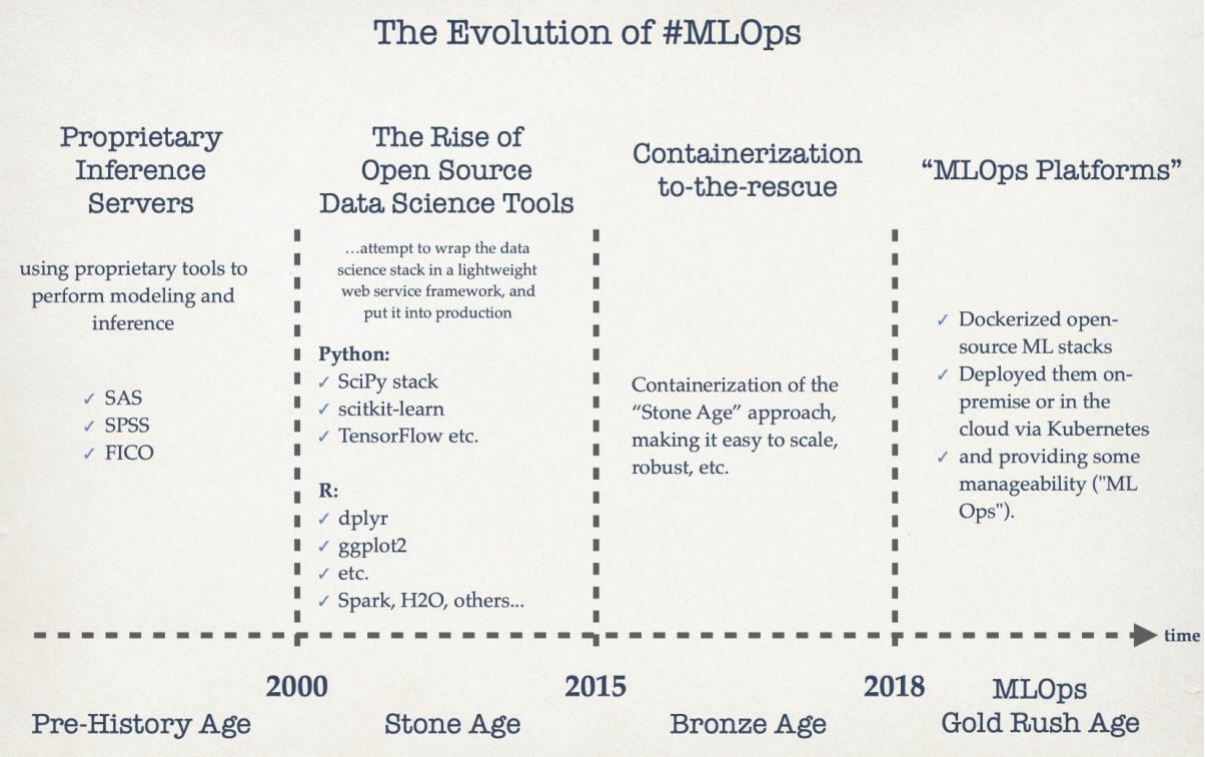


Figura 2.13: Evolución de MLOps. Fuente: [14]

3 Investigación sobre herramientas de MLOps

3.1 Benchmarking

Gartner es una compañía dedicada a la consultoría e investigación de las tecnologías de la información.

La misma tiene un prestigio muy importante a nivel mundial, y sus análisis son recibidos con mucha confianza por la comunidad tecnológica.

Cada año Gartner publica el Cuadrante Mágico de Gartner, que incluye a las empresas de tecnología más importantes a nivel mundial.

Existe una gran cantidad de cuadrantes, cada uno orientado a una temática diferente.
Para la temática de *MLOps*, corresponde el cuadrante para servicios en la nube de inteligencia artificial.

La descripción de Gartner de los servicios en la nube de inteligencia artificial es la siguiente:

“Los servicios para desarrolladores de IA en la nube (*CAIDS*) son servicios alojados en la nube o en contenedores que permiten a los equipos de desarrollo y a los usuarios comerciales que no son expertos en ciencia de datos usar modelos de inteligencia artificial (IA) a través de API, kits de desarrollo de software (*SDK*) o aplicaciones . Ayudan a estos usuarios a brindar servicios con capacidades en las áreas de aprendizaje automático automático (*AutoML*), lenguaje y visión , por ejemplo, comprensión del lenguaje natural (*NLP*), análisis de sentimientos, reconocimiento de imágenes y servicios de canalización de modelos de aprendizaje automático (*ML*).

Nuestra visión del mercado de CAIDS se centra en la capacidad de cada proveedor para satisfacer las necesidades futuras de los usuarios finales. No nos enfocamos en el mercado tal como es hoy. Los proveedores que ofrecen sólo servicios de idiomas o servicios de visión, así como los proveedores que se especializan solo en casos de uso específicos, fueron excluidos de este Cuadrante Mágico.”

En febrero de 2022 el cuadrante de servicios en la nube de IA se compuso de la siguiente manera:

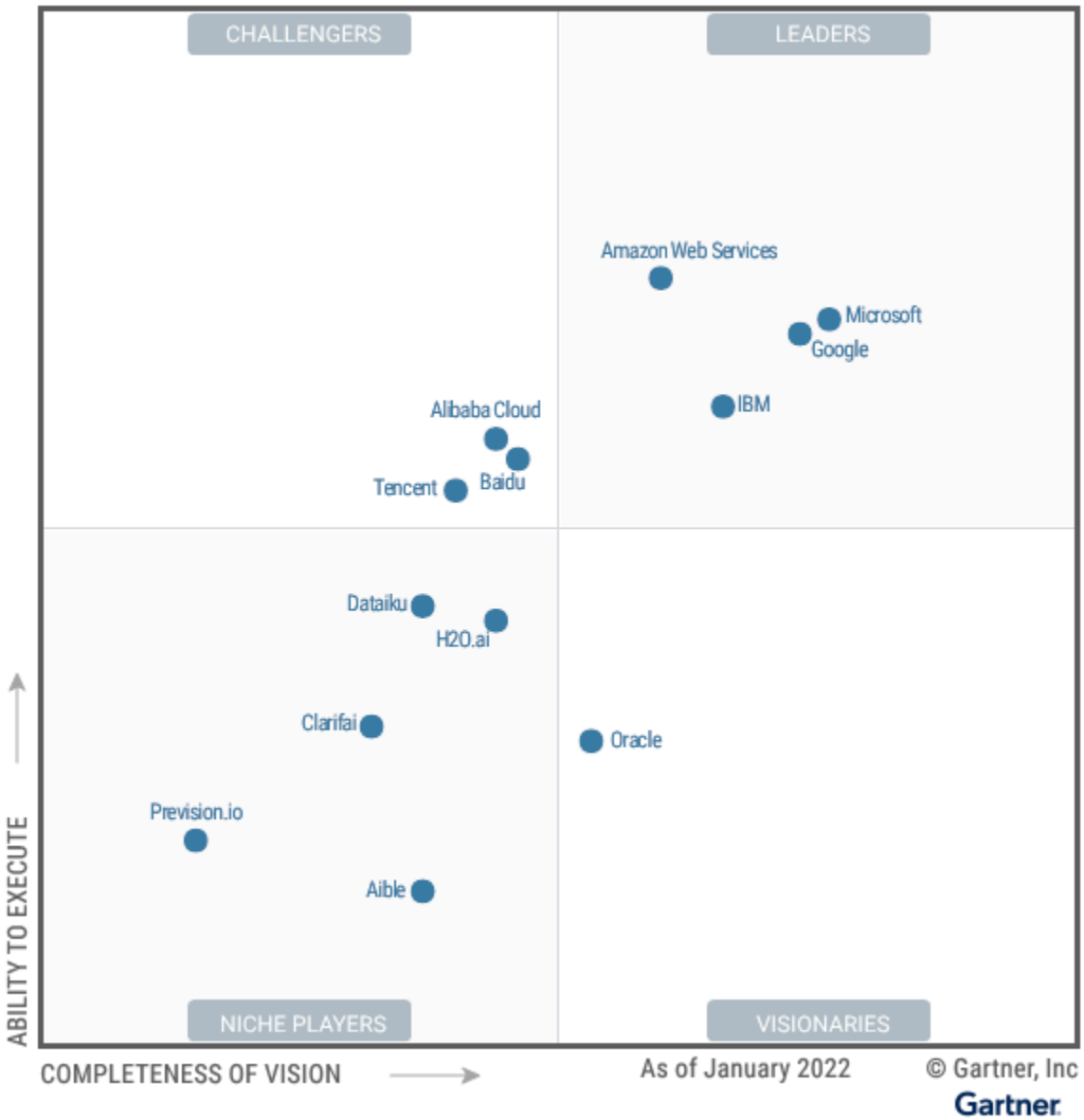


Figura 3.1: Cuadrante de Vertex de *CAIDS*. Fuente: [15]

En dicho cuadrante, Gartner clasifica como líderes a:

- Amazon Web Services: SageMaker
- Microsoft: Azure
- IBM: Watson
- Google: Vertex AI

Basándose en dicho análisis, en este proyecto se van a analizar estos cuatro servicios para la selección de la herramienta a utilizar.

3.2 Fortalezas y debilidades de cada herramienta

Gartner además de generar su cuadrante mágico, proporciona un análisis exhaustivo de cada herramienta. Del mismo se pueden obtener los principales puntos a favor y en contra de cada una de las herramientas.

3.2.1 Amazon Web Services

Los servicios de IA de Amazon Web Services (AWS), incluyendo Amazon Sagemaker y otros servicios de procesamiento de lenguaje natural y visión de computador, fueron diseñados para automatizar el ciclo de vida completo de desarrollo y despliegue de inteligencia artificial.

Figura 3.2: AWS y algunas de sus herramientas

AWS tiene una fuerte presencia global en el mercado, con clientes de todas las industrias. AWS permite a los clientes personalizar sus soluciones, ya sea con asistencia de su personal o con la ayuda de socios consultores.

Es una opción interesante para grandes cargas de trabajo en producción, ya que tiene bajos costos de operación y una amplia gama de opciones de infraestructura.

Una de sus fortalezas es la ejecución, donde tiene el mayor puntaje de todas las compañías del cuadrante. Esto es debido a su fuerte estrategia orientada a IA, su comprensión del mercado, sus estrategias de marketing y ventas y la ejecución del modelo de negocio. Como resultado de todos estos factores, AWS está expandiéndose rápidamente en el mercado de servicios en la nube de IA.

Además posee una amplia disponibilidad en diferentes zonas del mundo, convirtiéndose en un verdadero servicio global y en el mejor en este aspecto entre sus competidores.
En cuanto al producto, es excelente en la operativa y escalabilidad de inteligencia artificial. También permite su uso sin necesidad de conocimiento experto en el área.

Una de sus oportunidades de mejora, es que AWS es muy bueno en términos generales, pero le falta adaptabilidad a ciertas industrias que precisan herramientas más personalizadas.

En cuanto a desarrollo de IA responsable, está por detrás de sus competidores, aunque Amazon SageMaker Clarify, que es una herramienta de Amazon para detección de sesgos en modelos de ML y comprensión de las predicciones de los modelos, es un paso en la dirección correcta.

Por otra parte, en lo que respecta a su estrategia de producto, es el único líder sin una solución nube-híbrida o multi nube.

3.2.2 Microsoft

Microsoft Azure tiene una oferta integral para los tres casos de uso: procesamiento de lenguaje natural, visión por computador y AutoML.

Sus operaciones son globales y tiene clientes en todas las industrias. Sus servicios pueden ser consumidos por expertos vía *APIs* o *SDKs*, y por desarrolladores menos experimentados a través de Microsoft Power Platform.

Microsoft lidera la industria en términos de propiedades de calidad empresarial , como integración, escalabilidad, rendimiento, seguridad, privacidad, transparencia, explicabilidad y uso responsable de la IA.

Figura 3.3: Logo Microsoft Azure

Una de sus mayores fortalezas es que los servicios están utilizados para ser utilizados tanto por expertos como por no expertos, lo que amplía significativamente su potencial de clientes.

Otro punto a favor es su estrategia de ventas, la cual se adapta muy bien a las diferentes realidades de sus clientes. Por un lado ofrece sus características de *AutoML* de forma gratuita, mientras cobra por computación y almacenamiento. Para

atender a los clientes con un alto volumen de cargas de trabajo de IA, Microsoft presenta una opción que ofrecerá un precio por unidad más bajo pero requiere un compromiso de uso constante.

La empresa se basa y contribuye a muchos proyectos en la comunidad de código abierto. Mejoró continuamente sus servicios en 2021, con más de 30 lanzamientos, incluidos lanzamientos de funciones importantes. Además, Microsoft alienta a los desarrolladores a comprometerse con su mercado al reducir las comisiones al 3 % (por debajo del promedio de la industria del 20 %).

En cuanto a sus oportunidades de mejora, Microsoft ofrece su cartera completa de servicios en las Américas, pero no ofrece todos estos servicios en otros lugares. Además, su servicio de post venta no es tan bueno como el de sus competidores, generando una peor experiencia de cliente.

3.2.3 IBM

Las operaciones de IBM son globales, con clientes en todas las industrias.
IBM fortaleció la integración entre su área de investigación, líder en la industria, y sus áreas de producto, logrando que sus innovaciones sean desplegadas y lleguen al mercado.
Esto ha logrado fortalecer su marca Watson, bajo la cual consolida su oferta de servicios de IA.

IBM tiene una fuerte presencia en los mercados verticales y es líder en IA responsable y confiable. Lidera con una estrategia de nube híbrida que apela a la mayoría de los clientes.

Figura 3.3: Logo IBM Watson

Una de las fortalezas de IBM es su producto, tanto sus servicios de procesamiento de lenguaje natural, de visión de computador y de *AutoML* son sólidos.

La empresa se ha centrado en ofrecer soluciones sólidas para los desarrolladores. Sus herramientas son fáciles de usar para los desarrolladores y proporcionan activos de desarrollo codificados y de bajo código que les permiten crear soluciones rápidamente.
IBM es fuerte en su estrategia geográfica y en su enfoque hacia diferentes verticales del mercado.

Dentro de sus desventajas, se lista que su reputación en servicios en la nube de IA se ven influenciados por las soluciones que IBM Consulting crea e implementa, las cuales han tenido resultados mixtos.

Además, los precios de IBM son considerados excesivos por algunos de sus clientes.

3.2.4 Google

Google centra su oferta de servicios en la nube de IA en Vertex AI, dentro de Google Cloud Platform (GCP). A través de Vertex AI ofrece servicios de lenguaje, visión y *AutoML*.

Los servicios de Google se centran en modelos de redes neuronales profundas, junto con modelos *ML* pre entrenados que los desarrolladores pueden personalizar.

Es líder en investigación de IA e IA responsable, con más de 3500 investigadores que han producido más de 6000 artículos de investigación.

Figura 3.4: Logo Google Vertex.AI

Una de sus fortalezas es la cercanía con su comunidad de desarrolladores. El lanzamiento de Vertex AI ha integrado sus herramientas para construir, implementar y administrar modelos de IA y ML. Esto ha sido muy bien recibido por su comunidad ya que la integración ayuda a reducir el tiempo de despliegue de proyectos de IA.

En lo que respecta a IA responsable, Google tiene procesos de ética bien definidos y puede abordar rápidamente las inquietudes sobre la IA caso por caso.

La arquitectura de Google es clara y comprensible para los desarrolladores.

Una de sus oportunidades de mejora es que los esfuerzos de investigación y desarrollo de productos de Google se centran casi exclusivamente en las redes neuronales, y se presta poca atención a la IA simbólica.

Google no admite completamente la implementación de servicios de IA en nubes privadas o en las instalaciones. Sin embargo, los clientes pueden crear soluciones alternativas, utilizando *Kubeflow* por ejemplo.

Aunque Google tiene fuertes ofertas de mercado vertical, la cantidad de industrias a las que se dirigen es menor que la de algunos de sus competidores.

3.3 Comparación de herramientas

3.3.1 Requisitos mínimos

Para calificar para la inclusión en el cuadrante mágico, cada proveedor tenía que cumplir una serie de requisitos muy desafiantes.

Cada proveedor tenía que haber generado al menos 20 millones de USD en ingresos como resultado de sus servicios de inteligencia artificial en la nube en 2021; o tener al menos 75 clientes empresariales que paguen por sus servicios de inteligencia artificial en la nube.

Gartner define a los líderes de servicios de inteligencia artificial en la nube como:

“Los líderes tienen ofertas sólidas en las tres áreas de servicio clave: *AutoML*, lenguaje y visión. Se puede acceder a sus ofertas de IA como un servicio a través de API y no requieren que los desarrolladores tengan experiencia en ciencia de datos. Los líderes también brindan capacidades de soporte para mejorar sus servicios principales, incluida la detección y mitigación automatizadas de sesgos, ingeniería de características, NLP, etiquetado de imágenes, *MLOps* e IA explicable

e interpretable. Los líderes prestan servicios en varias regiones y admiten varios idiomas.”

Por lo tanto, se entiende que cualquiera de las herramientas líderes son suficientes para implementar un proyecto de inteligencia artificial con la metodología *MLOps*, siempre y cuando el servicio sea prestado en la región.

3.3.2 Criterios de selección

Como se estableció en el subcapítulo anterior, cualquier de las cuatro herramientas líderes sería más que suficiente para realizar este proyecto.

Por ello los criterios de selección serán los siguientes:

- **Zonas de disponibilidad:** una mayor cantidad de zonas de disponibilidad, cerca de donde los clientes van a utilizar los servicios en la nube, implica una mejora en la latencia del servicio.
- **Costos:** si todas las herramientas son lo suficientemente buenas para realizar el trabajo, el costo de las mismas se vuelve una de las variables más importantes. Para así realizar el proyecto lo más eficiente posible, gastando la menor cantidad de recursos. De esa forma podremos incrementar los márgenes de ganancia.
- **Usabilidad:** cuanto más sencillo sea realizar la arquitectura necesaria para que el modelo corra en producción, más tiempo se le podrá dedicar al *fine-tuning* del modelo. Por ello, que la plataforma sea fácil de utilizar es otro punto importante.

3.4 Zonas de disponibilidad

En los servicios en la nube, una zona de disponibilidad es el centro de datos de un proveedor de nube pública que contiene su propia energía y conectividad de red. Cada región es un área geográfica separada y, por lo general, cada región tiene varias ubicaciones aisladas conocidas como zonas de disponibilidad.

Si bien una única zona de disponibilidad puede abarcar varios centros de datos, dos zonas no pueden compartir un centro de datos.

Tener los servidores más cerca de los usuarios finales de los clientes disminuye la latencia. Además los clientes pueden implementar sus aplicaciones en más de una zona de disponibilidad, de forma que si falla una zona de disponibilidad, el servicio continúe en otra zona.

Por ello, cuanto más zonas de disponibilidad tenga cada proveedor en América del Sur, mejor será para la latencia del proyecto en cuestión.

Las zonas disponibles de Vertex AI en las Américas es la siguiente:

América Europa Asia-Pacífico

- Oregón (us-west1)
- Los Ángeles (us-west2)
- Las Vegas (us-west4)
- Iowa (us-central1)
- Carolina del Sur (us-east1)
- Virginia del Norte (us-east4)
- Dallas (us-south1)
- Montreal (northamerica-northeast1)
- Toronto (northamerica-northeast2)
- São Paulo (southamerica-east1)

Figura 3.5: Zonas de disponibilidad Google Vertex.AI. Fuente: [16]

Es decir que tiene disponible a San Pablo en América del Sur.

La disponibilidad en América del Sur de AWS es:

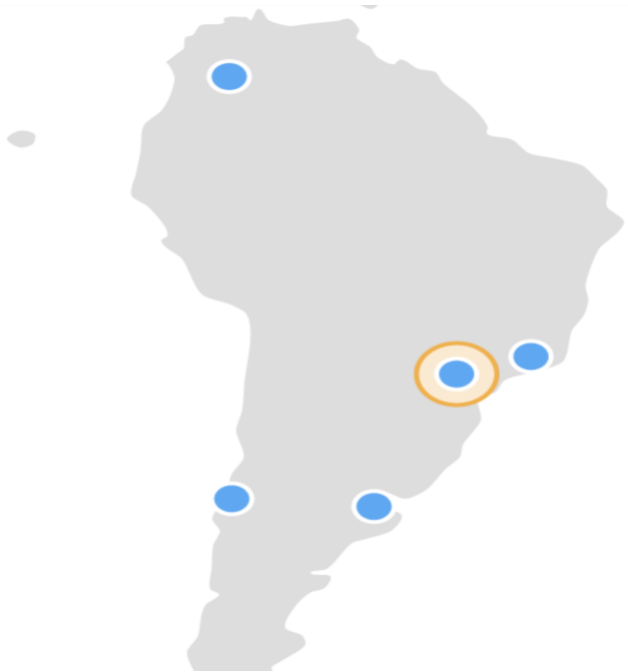


Figura 3.6: Zonas de disponibilidad de AWS. Fuente: [17]

Es decir:

- Río de Janeiro, Brasil
- São Paulo, Brasil
- Bogotá, Colombia
- Buenos Aires, Argentina
- Santiago, Chile

Por otro lado, IBM también tiene zonas de disponibilidad en San Pablo, dentro de Sudamérica.

Americas | Europe | Asia Pacific

Location	Region	Zone	Data center
Dallas	us-south	us-south-1 us-south-2 us-south-3	DAL10 DAL12 DAL13
Sao Paulo	br-sao	br-sao-1 br-sao-2 br-sao-3	SAO01 SAO04 SAO05
Toronto	ca-tor	ca-tor-1 ca-tor-2 ca-tor-3	TOR01 TOR04 TOR05
Washington DC	us-east	us-east-1 us-east-2 us-east-3	WDC04 WDC06 WDC07

Figura 3.7: Zonas de disponibilidad de IBM. Fuente: [18]

En cuanto a Microsoft, también tiene zonas de disponibilidad en el Sur de Brasil.

Americas	Europe	Middle East	Africa	Asia Pacific
Brazil South	France Central	Qatar Central*	South Africa North	Australia East
Canada Central	Germany West Central	UAE North*		Central India
Central US	North Europe			Japan East
East US	Norway East			Korea Central
East US 2	UK South			Southeast Asia
South Central US	West Europe			East Asia
US Gov Virginia	Sweden Central			China North 3
West US 2	Switzerland North			
West US 3				

Figura 3.8: Zonas de disponibilidad de Microsoft. Fuente: [19]

En conclusión, Amazon tiene una mayor oferta de zonas de disponibilidad en América del Sur, teniendo cinco a su cargo. Los demás competidores tienen una única zona ubicada en Brasil.

3.5 Comparación de costos

3.5.1 Microsoft Azure

Microsoft contiene una calculadora de costos, en la cual podemos observar claramente que el costo es más caro en la modalidad de pagar al ir usando, comparando con si reservamos cierta cantidad y capacidad de cómputo en forma anticipada.

La calculadora es muy clara y fácil de configurar, ya que solo se debe determinar el servicio a contratar y la máquina a elegir. Cada máquina tiene el detalle de sus características.

Para los siguientes parámetros:

- Región Brasil
- Pagar al usar
- 300 horas
- 4 CPU
- 16 GB RAM
- 150 GB almacenamiento temporal

El precio es de USD 0.360 por hora. Es decir un total de USD 108 por 300 horas.

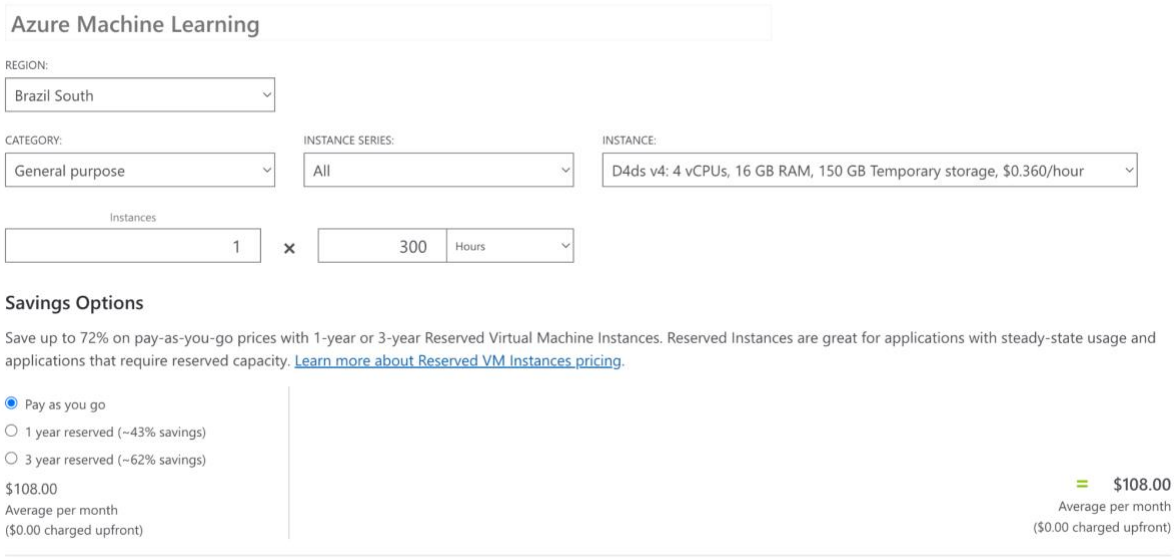


Figura 3.9: Calculadora de precios de Microsoft Azure. Fuente: [20]

3.5.2 Amazon SageMaker

Con la calculadora de AWS, configurando los siguientes parámetros:

- Región Brasil
- 1 notebook
- 300 horas
- 4 CPU
- 16 GB RAM

Da como resultado USD 0.323 por hora, es decir USD 96.9 por 300 horas.

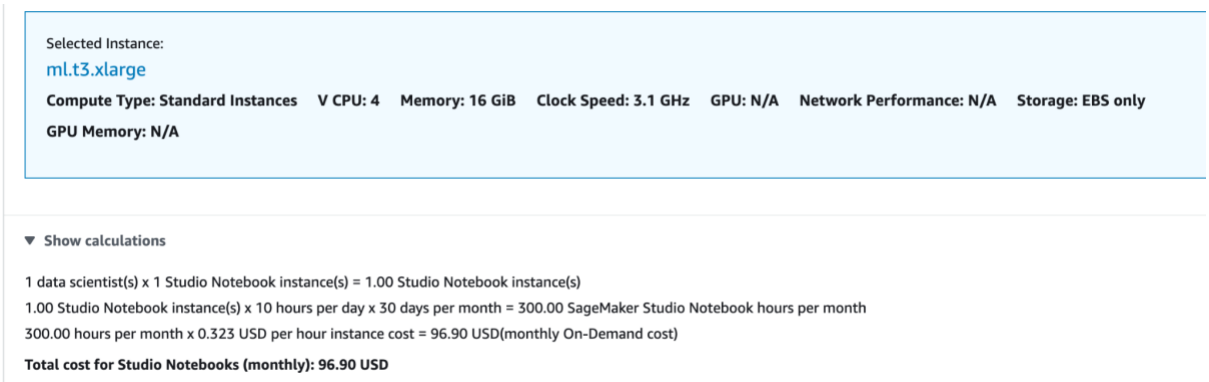
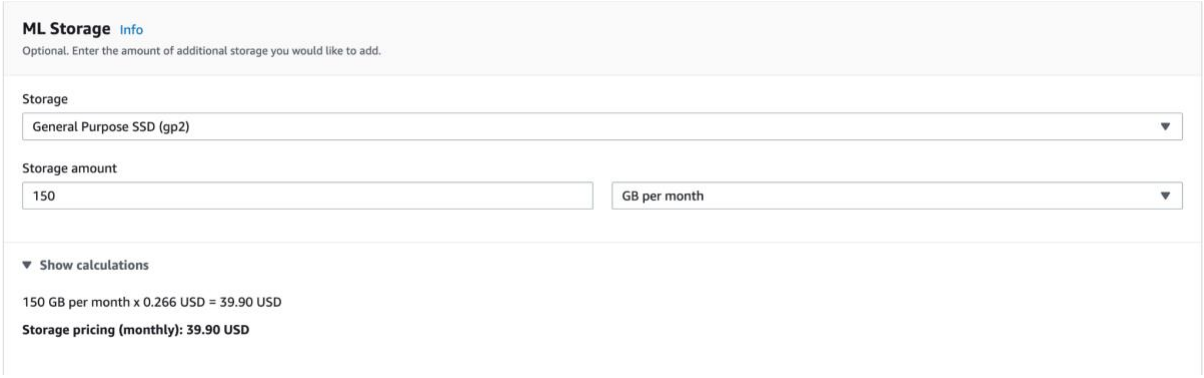


Figura 3.10: Calculadora de precios de AWS. Fuente: [21]

Además, para agregar 150 GB de almacenamiento mensual, los costos son los siguientes:

0.266 USD por GB, para un total de USD 39.90



Por lo tanto el costo total en AWS sería de 136.8 USD

3.5.3 IBM Watson

La calculadora de costos de IBM es menos clara que las de AWS y Microsoft. Es menos configurable.

Según el cuadro, una unidad de capacidad tiene un costo de USD 0.50 por hora. Y una arquitectura de 4 CPU y 16 GB de ram serían dos unidades de capacidad.

Por ello el costo por hora sería de 0.50 * 2 = USD 1 y el costo de 300 horas de USD 300

Es el precio más caro de los software analizados, lo cual condice con las desventajas de IBM mencionadas en el artículo de Gartner.

v2 Standard	**Instancia de servicio** Horas de unidades de capacidad (CUH) de pago por uso Tipo de capacidad: • 1 vCPU y 4 GB RAM = 0,5 unidades de capacidad necesarias por hora • 2 vCPU y 8 GB RAM = 1 unidad de capacidad necesaria por hora • 4 vCPU y 16 GB RAM = 2 unidades de capacidad necesarias por hora • 8 vCPU y 32 GB RAM = 4 unidades de capacidad necesarias por hora • 16 vCPU y 64 GB RAM = 8 unidades de capacidad necesarias por hora IA automática • 8 vCPU y 32 GB RAM = 20 unidades de capacidad necesarias por hora	0,50 $ USD/Capacity Unit-Hour

Figura 3.11: Calculadora de precios de IBM. Fuente: [22]

3.5.4 Vertex AI

La estimación de costos de Google es más difícil ya que se debe elegir la máquina virtual en base a su nombre, en lugar de seleccionar las características deseadas de la misma.

La máquina e2-standard-4 tiene 4 CPU y 16 GB de memoria

E2 estándar | E2 con alta capacidad d... | CPU elevada E2 | Núcleo compartido E2

Los tipos de máquinas estándar E2 tienen 4 GB de memoria de sistema por CPU virtual.

Tipos de máquina	CPU virtuales*	Memoria (GB)	Cantidad máxima de discos persistentes (PD)†	Tamaño total máximo de PD (TB)	SSD local	Ancho de banda de salida máximo (Gbps)‡
e2-standard-2	2	8	128	257	No	4
e2-standard-4	4	16	128	257	No	8
e2-standard-8	8	32	128	257	No	16
e2-standard-16	16	64	128	257	No	16
e2-standard-32	32	128	128	257	No	16

Figura 3.12: Máquinas virtuales de Google.

La máquina e2-standard-4 tiene un costo por hora de entrenamiento de USD 0.19. Es decir de USD 57 por 300 horas.

En cuanto al almacenamiento offline, tiene un costo de 0.023 USD por GB al mes. Calculando 150 GB, serían USD 3.45

Vertex AI Feature Store

Los precios de Vertex AI Feature Store se basan en la cantidad de datos que ofrecen las funciones de almacenamiento online y offline, así como en la disponibilidad de servicios online. Una *hora de nodo* representa el tiempo que emplea una máquina virtual para servir datos de funciones o para mantenerse lista a la espera de gestionar solicitudes de datos de funciones.

Nota: Los precios son los mismos en todas las regiones donde se admite Vertex AI Feature Store.

Operación	Precio
Almacenamiento online	0,25 USD por GB al mes
Almacenamiento offline	0,023 USD por GB al mes
Servicio online	0,94 USD por nodo y hora
Exportación por lotes	0,005 USD por GB

Figura 3.13: calculadora de costos de Google.

Por lo tanto el costo total en Google sería de USD 60.45

3.5.5 Comparativo

En conclusión, contratar 4 CPU y 16 GB de RAM por 300 horas y 150 GB de almacenamiento, tendrían los siguientes costos en USD:

Google	60.5
Microsoft	108
Amazon	136.8
IBM	300

Tabla 3.1: Comparativo de costos

4 Selección de la herramienta

4.1 Comparaciones y selección

En el capítulo anterior se listaron las fortalezas y debilidades de cada herramienta, así como sus comparativos en zonas de disponibilidad y costos de sus servicios.

En cuanto a fortalezas y debilidades, las cuatro herramientas se muestran muy sólidas y en constante desarrollo, por lo que la selección no tiene una única respuesta correcta, y dependerá del caso de uso a desarrollar y del contexto del problema.

Amazon tiene mayores zonas de disponibilidad en América del Sur, lo que puede implicar una menor latencia. Sin embargo, en este problema en particular, las cantidades de datos manejadas no son muy grandes, por lo que no es un punto prioritario a tener en cuenta.

Por otro lado, en la comparativa de costos Google resultó ser el servicio más económico. Este factor sí es importante en un proyecto como este, ya que el precio del producto puede determinar si es viable y rentable.

Además, tengo experiencia trabajando con *Big Query* en *Google Cloud Platform (GCP)*, lo que podría disminuir la curva de aprendizaje de la herramienta *Vertex AI*, ya que como ambas pertenecen a *GCP*, seguramente tengan una interfaz de usuario similar.

Por estos puntos la herramienta seleccionada para desplegar el modelo del proyecto con metodología *MLOps* es *Vertex AI* de Google.

4.2 Costos de la herramienta elegida

La herramienta Google Cloud Platform otorga un crédito de usd 300 a sus usuarios nuevos, con un límite de 90 días

Además se deben cumplir los siguientes requisitos:

- Nunca haber sido un cliente de pago de Google Cloud, Google Maps Platform o Firebase.
- Que el cliente no se ha registrado anteriormente para la prueba gratuita.

.

El período de prueba gratuito de $ 300 de 90 días comienza automáticamente cuando completa su registro.

Según la página oficial de Google Cloud Platform “Para completar su registro de prueba gratuita, debe proporcionar una tarjeta de crédito u otro método de pago para configurar una cuenta de facturación en la nube y verificar su identidad. No se preocupe, configurar una cuenta de facturación en la nube no nos permite cobrarle. No se le cobra a menos que habilite explícitamente la facturación al actualizar su cuenta de Facturación de Cloud a una cuenta paga. Puede actualizar a una cuenta paga en cualquier momento durante la prueba. Después de actualizar, aún puede usar los créditos restantes (dentro del período de 90 días).”

En este extracto hay dos puntos muy importantes. El primero es que se debe otorgar una tarjeta de crédito para la creación de la cuenta. La misma debe ser validada por Google.
Por ello es no es fácil que el usuario cree varias cuentas para abusar de los créditos gratis.

El segundo punto es que no se le cobra al usuario hasta que él mismo elija la opción paga. Esto es una enorme ventaja por sobre sus competidores. Por ejemplo, Amazon no da aviso de que se terminó la versión de prueba, por lo que los usuarios suelen incurrir en altos costos inesperados.

Para este proyecto se comenzó con la versión gratuita, pero luego se actualizó la versión paga para utilizar GPU en el entrenamiento, ya que los mismos están disponibles solo en la versión paga. Igualmente, como dice el extracto de texto, te

permiten seguir utilizando los usd 300 aunque cambies a pago, aunque ya queda en responsabilidad del usuario si se sobrepasa y tiene cargos en su tarjeta.

5 Desarrollo

Ya teniendo definidas las diferentes etapas de un proceso de punta a punta de aprendizaje automático, así como de la metodología *MLOps*, se procede a poner en práctica los conocimientos teóricos con un caso de estudio.

Anteriormente definimos las siguientes etapas en el desarrollo de un modelo de aprendizaje automático de punta a punta:

Figura 5.1: Flujo de aprendizaje automático

En este capítulo profundizaremos en cada etapa, aplicada al caso de estudio de predicción del precio de un auto o camioneta basado en la oferta de productos de MercadoLibre.

5.1 Desarrollo del pipeline

La documentación de Google [23] define a los pipelines de la siguiente manera: “Vertex AI Pipelines te permite organizar los flujos de trabajo de aprendizaje automático (AA) sin servidores. Antes de que *Vertex AI Pipelines* pueda organizar el flujo de trabajo de AA, debes describirlo como una canalización. Las canalizaciones de AA son flujos de trabajo de AA portátiles y escalables que se basan en contenedores y servicios de Google Cloud.”

Además comenta que “Vertex AI Pipelines puede ejecutar canalizaciones compiladas con la versión 1.8.9 o superior del SDK de Kubeflow Pipelines, o una versión 0.30.0 o superior de TensorFlow Extended.”

En este caso fue construido con la *SDK* de *Kubeflow*.

VertexAI se creó en mayo del 2021, por lo que es una herramienta muy nueva y no tiene una gran cantidad de casos de uso que se puedan encontrar en internet para apoyarse al momento de desarrollar. Por ello fueron fundamentales dos factores:

1. La extensa documentación oficial proporcionada por Google, así como los cursos que ofrecen dentro de la misma plataforma
2. La documentación oficial de Kubeflow, así como los casos de uso de *Kubeflow* que los usuarios han subido a internet.

Si bien VertexAI simplifica el no tener que administrar un cluster de *Kubernetes*, su curva de aprendizaje para desarrollar un pipeline es alta y se ve incrementada por la escasa comunidad online.
Se puede suponer que dicha falta se expresa por el costo de la herramienta, lo que no la hace accesible para todo el mundo, así como de su poca antigüedad en el mercado.

Los *pipelines* están basados en un sistema de contenedores.

Los componentes de canalización de *Kubeflow* son funciones de fábrica que crean pasos de canalización. Cada componente describe las entradas, salidas y su implementación.

Los componentes se usan para crear los pasos de la canalización. Cuando se ejecuta una canalización, los pasos se ejecutan a medida que los datos de los que dependen están disponibles. Por ejemplo, un componente de entrenamiento puede tomar un archivo CSV como entrada y usarlo para entrenar un modelo.

En este capítulo se describirá en detalle la completitud del *pipeline*, el cual podemos observar en la imagen siguiente:

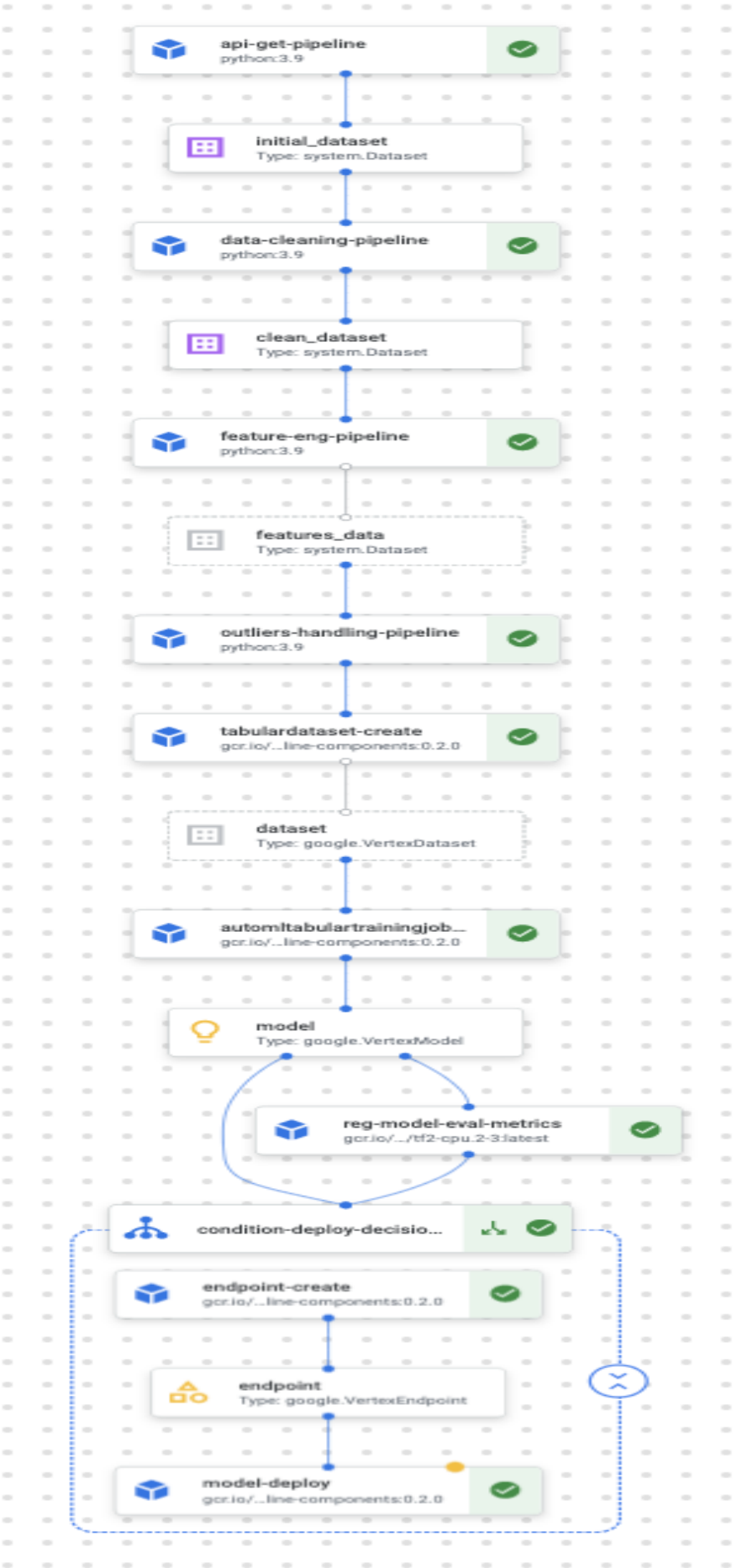

api-get-pipeline
python:3.9
initial_dataset
Type: system.Dataset
data-cleaning-pipeline
python:3.9
clean_dataset
Type: system.Dataset
feature-eng-pipeline
python:3.9
features_data
Type: system.Dataset
outliers-handling-pipeline
python:3.9
tabulardataset-create
gcr.io/...line-components:0.2.0
dataset
Type: google.VertexDataset
automltabulartrainingjob...
gcr.io/...line-components:0.2.0
model
Type: google.VertexModel
reg-model-eval-metrics
gcr.io/.../tf2-cpu.2-3:latest
condition-deploy-decisio...
endpoint-create
gcr.io/...line-components:0.2.0
endpoint
Type: google.VertexEndpoint
model-deploy
gcr.io/...line-components:0.2.0

Figura 5.2: *Pipeline* completo

5.2 Definición del problema de negocio

Figura 5.3: Flujo de aprendizaje automático - Definición del problema de negocio

El problema consiste en realizar la predicción del precio de un auto o camioneta, tanto nuevo como usado, en Uruguay. Para dicha predicción se utilizarán variables sencillas de conseguir, como el modelo del auto, el kilometraje y el año de compra; datos que se entiende que cualquier dueño de un vehículo puede obtener fácilmente.

Se eligió este problema porque el vehículo de un uruguayo suele ser una parte importante de su patrimonio, y por ello al momento de cambiarlo se requiere mucho análisis para definir el precio del mismo.

Si bien el precio de los vehículos nuevos no varía tanto, se puede encontrar diferencias según la zona geográfica o la oferta y demanda. Por otro lado, calcular el precio de un auto usado sí es una tarea difícil y es donde el modelo puede generar mayor valor.

Una vez desarrollado el modelo, se entiende que puede ser utilizado principalmente de dos maneras:

1. Siendo la columna vertebral de un producto orientado a los datos, que consista en una web o una app que realiza la predicción del precio del vehículo del usuario, basándose en los datos que el usuario ingresa como input de su vehículo.
   Con la misma información, la aplicación podría también realizar recomendaciones de vehículos a los usuarios, basándose en su presupuesto y de las prestaciones que el usuario busca en su vehículo.
   Este producto estaría orientado a clientes personas físicas, y sería un producto de consumo masivo con el objetivo de democratizar el rápido acceso a información de vehículos, basado en el presupuesto del usuario y sus preferencias.

2. La segunda aplicación está orientada a personas jurídicas y consiste en un buscador de oportunidades en la compra de vehículos usados.
   Una herramienta que puede realizar la predicción del precio de un vehículo, puede también identificar vehículos que estén significativamente por debajo de su precio de mercado. Ese dato puede ahorrar muchísimas horas de exploración e investigación de las ofertas de vehículos en búsqueda de una oportunidad financiera. Esta aplicación no sería democratizable ya que es información valiosa si no la tiene todo el mundo. Por lo tanto está orientada hacia el mundo empresarial.

Los datos son obtenidos desde la API pública de MercadoLibre. Esto se debe a que es donde hay mayor cantidad de información sobre precios de vehículos, tanto nuevos como usados, así como a la posibilidad de obtener dicha información sin un costo asociado.

La ventaja de obtener los datos desde dicha fuente, es como mencionamos anteriormente, que los mismos se mantengan actualizados acorde a la oferta del momento. Por ello es clave incorporar *MLOps* en el proyecto, para así mantener el modelo actualizado sin un gran costo en horas de desarrollo.

Traduciendo el problema a términos de aprendizaje automático, el mismo es un problema de regresión, es decir que su resultado no será binario ni multiclase, sino que su output será una variable numérica continua.

En términos de las métricas de evaluación, es importante obtener la menor diferencia posible en términos económicos entre la predicción y el precio real.

Dicha diferencia es importante tanto en términos absolutos como porcentuales. Es decir, predecir que un auto de precio 13.000 usd, vale 10.000 usd, es peor que predecir que un auto de precio 70.000 usd, vale 67.000 usd.

Si bien la diferencia absoluta es la misma (3.000 usd), la diferencia porcentual respecto de su precio es muy diferente.

Se retomará este punto más adelante, al momento de elegir las métricas que determinen el éxito o fracaso del modelo. Lo que se puede adelantar es que se va a buscar una manera que tome en cuenta tanto la diferencia absoluta como la porcentual.

5.3 Obtención y análisis de los datos

Figura 5.4: Flujo de aprendizaje automático - Obtención y análisis de los datos

El acceso a los datos es a través de la API pública de Mercadolibre.
La documentación de dicha API está en la siguiente url:
https://developers.mercadolibre.com.uy/es_ar/api-docs-es

Para poder conectarse a la API es necesario crear una aplicación desde la web de desarrolladores de Mercadolibre.

5.3.1 Access secret y tokens

En el anexo 11.1 se encuentra el detalle de los desafíos de esta sección del pipeline y cómo se resolvieron.

Fue particularmente difícil ya que MercadoLibre aplica muchas trabas para que los desarrolladores no extraigan muchos datos desde su API y para que tampoco lo hagan de manera programática, sin la interacción de un humano.

La forma de esquivar dichas trabas es hacer un primer pedido con un humano, ya que es necesaria una validación de segundo factor. En dicho pedido nos devuelven un código que puede ser utilizado una única vez para consultar la API.

Utilizando dicho código, podemos obtener un token y un refresh token.

El token es válido por una sola ejecución, pero el refresh token no.

Además el refresh token habilita otro tipo de consulta, que nos devuelve un nuevo refresh token, es decir que lo podemos renovar en cada consulta.

Por ello es viable lograr consultar los datos de forma programática, una vez que pudimos acceder al primer refresh token.

Por otro lado, se utilizó el Secret Manager de Google Cloud Platform, para asegurar la seguridad de las claves.

De acuerdo al detalle en el anexo, se pudo acceder a los datos de MercadoLibre sin la necesidad de la interacción de un humano y asegurando las claves de acceso desde la misma herramienta de Google.

5.3.2 Primer acceso a los datos

En el Anexo 11.2 se detallan los procesos a nivel de código de esta parte del *pipeline.*

La cantidad de publicaciones de autos y camionetas ha oscilado entre 17.000 y 19.000 registros, por lo que se buscó la forma de obtener todos esos registros esquivando el límite de 4.000 impuesto por la compañía . Para así lograr construir el mejor modelo posible y no dejar afuera información valiosa.

La obtención de los datos se realizó en dos pasos, acorde a las limitaciones de la API:

- Obtención del listado de ítems (un ítem es un artículo que se vende en una publicación, el mismo modelo de auto vendido por dos vendedores diferentes son dos ítems diferentes).
- A partir del listado de ítems, la obtención de los datos de cada publicación

El límite de 4.000 registros afecta al primer punto, ya que el punto 2 se ejecuta sin *offset* ni *limit*, sino item por item.

La API permite consultar por categoría, y esa es la primera consulta a realizar y la más sencilla. Con ella se obtienen 4.000 publicaciones sin repetir.

Otra opción de la API es consultar una búsqueda específica (como si escribiera en el buscador de MercadoLibre). Se validó que uno de los campos devueltos es la categoría, por lo que luego de utilizar esta función se filtra la categoría para limpiar resultados indeseados.

Luego de diferentes intentos, se determinó que la combinación de auto + tipo de cuerpo de auto y camioneta + tipo de cuerpo de camioneta; así como de la marca de los vehículos, eran buenas maneras de obtener nuevos registros relacionados de forma escalable y que pueda perdurar en el tiempo. Ya que no es común que salgan nuevos tipos de cuerpos de vehículos (sedán, hatchback, entre otras), ni que surjan nuevas marcas de vehículos constantemente.

De esta manera, se obtuvieron más del 90% de las publicaciones de autos y camionetas.

Luego se transformaron los datos para pasar de formato json a *pandas dataframe.*

El código de esta sección está contenido en el primer componente del pipeline.

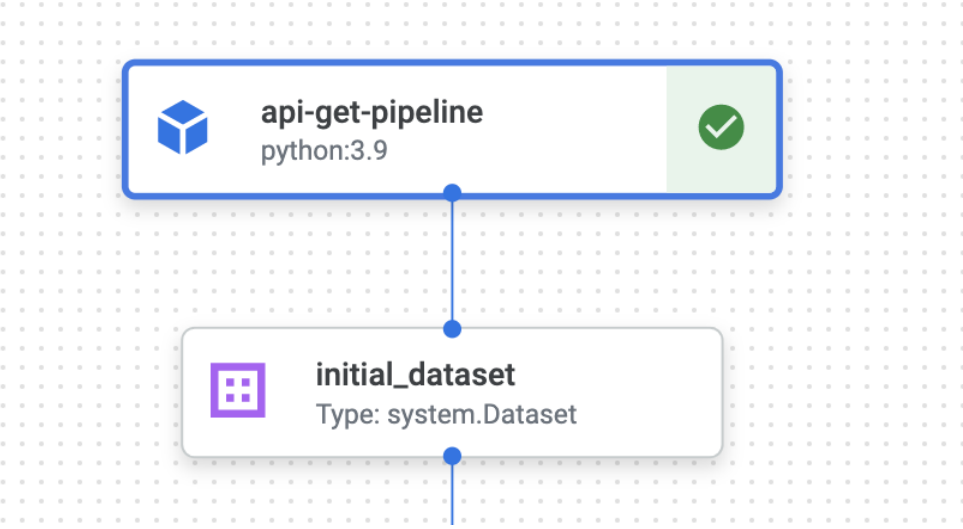


Figura 5.6: *Pipeline - API Get*

5.4 Preparación y transformación de los datos

Figura 5.7: Flujo de aprendizaje automático - Preparación y transformación de los datos.

5.4.1 Análisis exploratorio y Limpieza de datos

El análisis exploratorio es realizado en su mayoría con la librería *pandas de Python*.

El primer paso consiste en observar los datos a través de impresiones (print) del dataframe.
Podemos observar que el dataframe tiene 93 columnas, de las cuales 14 son columnas “array”, es decir columnas que tienen dentro un json, con muchos más campos que la columna original.

Las columnas *array* son muy flexibles al momento de almacenar datos, ya que pueden variar en la cantidad de campos clave valor que contienen, es decir variar el número de columnas (hablando en términos de dataframe). Por la misma razón, es un formato incómodo al momento de consultar los datos.

Los datos están muy sucios ya que el formato es el mismo para todas las publicaciones de MercadoLibre, sin importar su categoría. Esto es posible gracias a la utilización de las columnas *array* mencionadas. Por ello se hizo un fuerte foco en la limpieza de datos para obtener un buen modelo. Además hubo varias columnas que no requirieron mucho análisis para ser eliminadas, ya que no estaban pensadas para publicaciones de vehículos o directamente no traían datos.

Dentro de cada columna, se analiza si la misma está vacía o si tiene un único valor repetido en todas sus filas. Si este es el caso se procede a eliminar dicha columna, ya que no aporta valor al momento de realizar una predicción.

Se descartan dichas columnas y quedan como resultado 43 columnas.

El paso siguiente es analizar el contenido de cada columna y entender si aporta valor aplicando conocimiento de negocio. Por ejemplo los id de las publicaciones no aportan información de las mismas y confundirían al modelo. Por lo tanto, no es necesario realizar un análisis estadístico para remover dicha variable.

También se analiza si existe alguna variable que tenga absolutamente todos sus valores diferentes, ya que esa variable también sería descartada.

Por otro lado, puede haber dos o más variables que tengan los mismos valores, en cuyo caso se mantiene solo una de las variables, ya que tendrían una correlacionalidad del 100% y no agregaría valor. Por ejemplo las variables “ITEM_CONDITION” y “condition” tienen exactamente los mismos valores.

Luego de realizar estas limpiezas, quedan 9 columnas, de las cuales dos son en formato array.

El siguiente paso podría catalogarse como del capítulo 6.3: Data cleaning y feature engineering. Consiste en cambiar el formato de las columnas array, abriendo cada variable dentro de ellas en una nueva columna. La limpieza de datos y el análisis exploratorio se superponen en ciertos casos, ya que para explorar la data a veces es necesario limpiarla primero, pero para limpiarla hay que comprenderla y por lo tanto explorarla. Más que un proceso en serie son dos procesos que conviven en el tiempo.

El proceso de apertura de las columnas *array* está detallado en el anexo 11.3

Luego de la apertura de las columnas arrays, el *dataframe* queda con 29 columnas.

Conectando este paso con el anterior, este es el pipeline al momento:

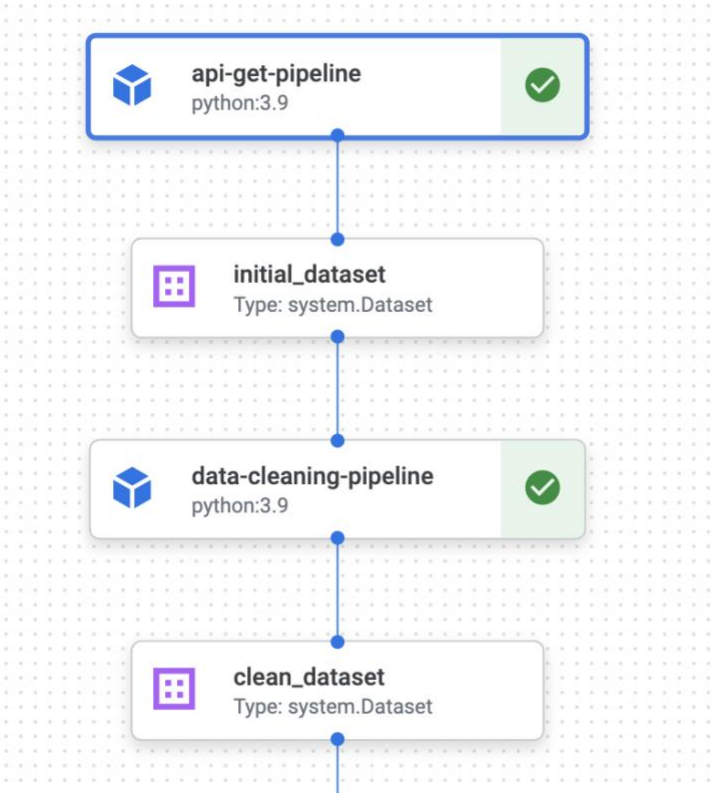


Figura 5.8: *Pipeline* - API Get + limpieza de datos

5.4.2 Ingeniería de atributos

El primer paso de la limpieza de los datos es identificar valores nulos en las columnas, ya que los mismos dan error al entrenar en la mayoría de los modelos.

Para entender las limpiezas necesarias, se realizan exploraciones sobre los datos agrupando la data en tablas, como en el siguiente ejemplo:

```
1 df.groupby(df["vehicle_body_type"].tolist(),as_index=False).size()
```

	index	size
0		1224
1	Cabriolet	27
2	Coupé	134
3	Crossover	33
4	Furgón	469
5	Hatchback	2390
6	Light Truck	20
7	Minivan	69
8	Monovolumen	23
9	Off-Road	27
10	Pick-Up	1514
11	Rural	235
12	SUV	1936
13	Sedán	1565
14	Van	20

De esa forma se puede observar la distribución de los datos, más allá de ver solo una muestra de los mismos como se venía haciendo hasta el momento.

Luego de analizar la distribución de los datos, así como la cantidad de nulos de los mismos, se descartan algunas variables.

Por ejemplo, al analizar las publicaciones en pesos uruguayos, se observa que la mayoría son errores del usuario al momento de publicar, pero no todas.

Como es muy difícil distinguir programáticamente cuáles están bien y cuáles no como para hacer la conversión de las erróneas, se procede a filtrar sólamente las publicaciones en USD. Luego de ello el dato de moneda (que hasta el momento tenía mucho valor) deja de tener valor agregado y se elimina.

Luego se analiza el tipo de dato de cada columna. Los tipos aceptados para entrenar son numéricos y booleanos. Por lo tanto se hacen las transformaciones correspondientes, como transformar “si” en 1 y “no” en 0.

También se realizan transformaciones en los datos, como pasar a mayúsculas y quitar los tildes a las variables de texto.

Agregando este paso, la foto del pipeline es la siguiente:

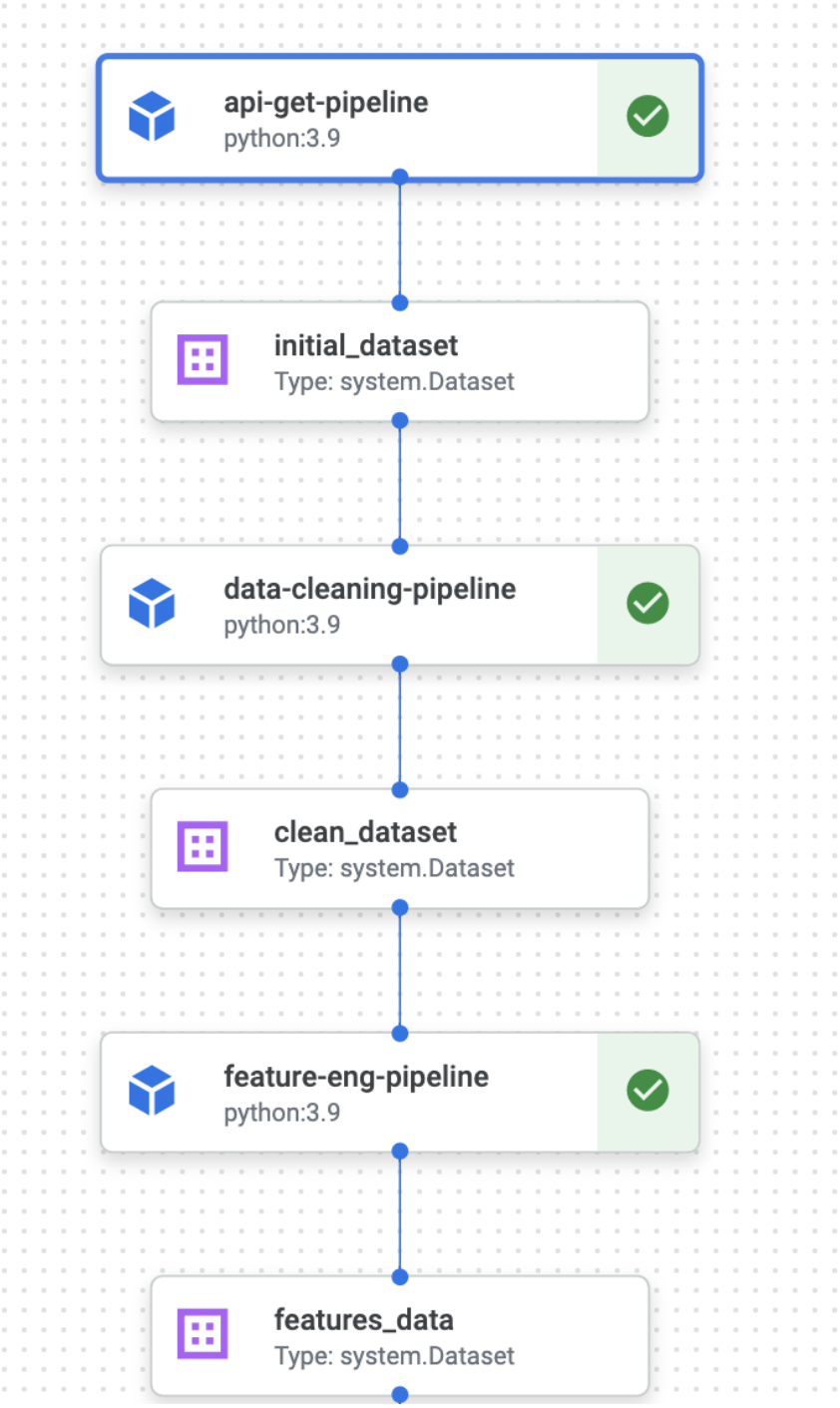


Figura 5.9: *Pipeline* - API Get + limpieza de datos + ingeniería de atributos

5.4.3 Tratamiento de outliers

Los outliers son los datos que se definen como excepciones. Muchas veces dichos datos son la consecuencia de un error del usuario al momento de ingresarlos, otras son excepciones que pueden aportar valor o no según el caso de uso, e incluso pueden ser muy dañinos para el modelo.

En este caso al tratarse de publicaciones de los usuarios, el usuario tiene la libertad de publicar su auto muy lejos del precio del mercado sin restricciones. Por lo que se entiende que los outliers no son de valor para el modelo por el origen de los datos. Además, como ya se observó en el caso de la divisa, los usuarios incurren en errores al momento de ingresar los datos de la publicación.

Diferente sería si los datos provinieran de ventas y no de publicaciones, ya que podría haber una explicación lógica detrás de esa excepción, y es más difícil que se efectúe la venta si hay errores importantes en la publicación.

Se analizan todas las publicaciones de precio mayor a 1 millón de USD, y se concluye que la mayoría son errores en el ingreso de los datos. Por ello se procede a eliminar dichos registros. Esta exclusión tuvo un impacto muy positivo en el modelo.

Previo a la exclusión de las publicaciones en pesos y de las publicaciones mayores a un millón de usd, el modelo daba resultados extremadamente malos, llegando a predecir valores negativos como precios de autos. Luego de analizar en detalle la situación se llegó a la conclusión de que los datos estaban muy sucios, lo que no permitía generalizar bien al modelo.

Dos de las variables más representativas del precio de un vehículo son su marca y su modelo. Dichas variables tienen una alta cardinalidad.

En esos casos se suelen agrupar los valores menos repetidos como la categoría “otros”. Luego de analizar los datos se entiende que dicha práctica puede ser muy peligrosa en este caso ya que hay marcas como Ferrari que tienen valores muy diferentes que las demás y no deberían agruparse por tener pocos registros.

Se hizo el experimento de probar el modelo [27] *catboost* de árboles, ya que tiene la particularidad de aceptar variables categóricas como *input*. Los resultados del mismo no fueron satisfactorios, y luego de investigar el modelo en detalle, se descubre que si bien soporta variables categóricas, establece un máximo a la cantidad de categorías por atributo. Por ello no era un buen modelo para este caso de uso.

Luego de probar otras alternativas se buscó reemplazar la variable categórica, por el valor promedio de la marca en las publicaciones activas, así como el modelo por el

valor promedio del modelo. De esa forma ambas variables se convirtieron en numéricas y su alta cardinalidad dejó de ser un problema.

Estos cambios generaron una gran mejora en la performance del modelo.

También se utilizaron estas nuevas variables para excluir outliers generados por errores del usuario al ingresar los datos. Si bien se habían excluido los autos de precio mayor a 1MM usd, aún podían quedar errores de menor magnitud pero que igual podían tener un alto impacto en el modelo.

Se procedió a establecer un umbral conformado por el precio promedio modelo del auto + la desviación estándar * 3. Si un ítem del mismo modelo supera ese umbral, es descartado del dataframe.

Una vez que se terminaron de limpiar los datos, se realiza el *one hot encoding*. Es decir la transformación de todas las variables categóricas en variables booleanas (1 ó 0).

Por definición esta transformación aumenta la cantidad de columnas, por lo que el dataset final con el que se va a entrenar el modelo quedó definido con 54 columnas. Este es el último componente creado por el usuario en lo referido a la obtención de los datos. El pipeline al momento es el siguiente:

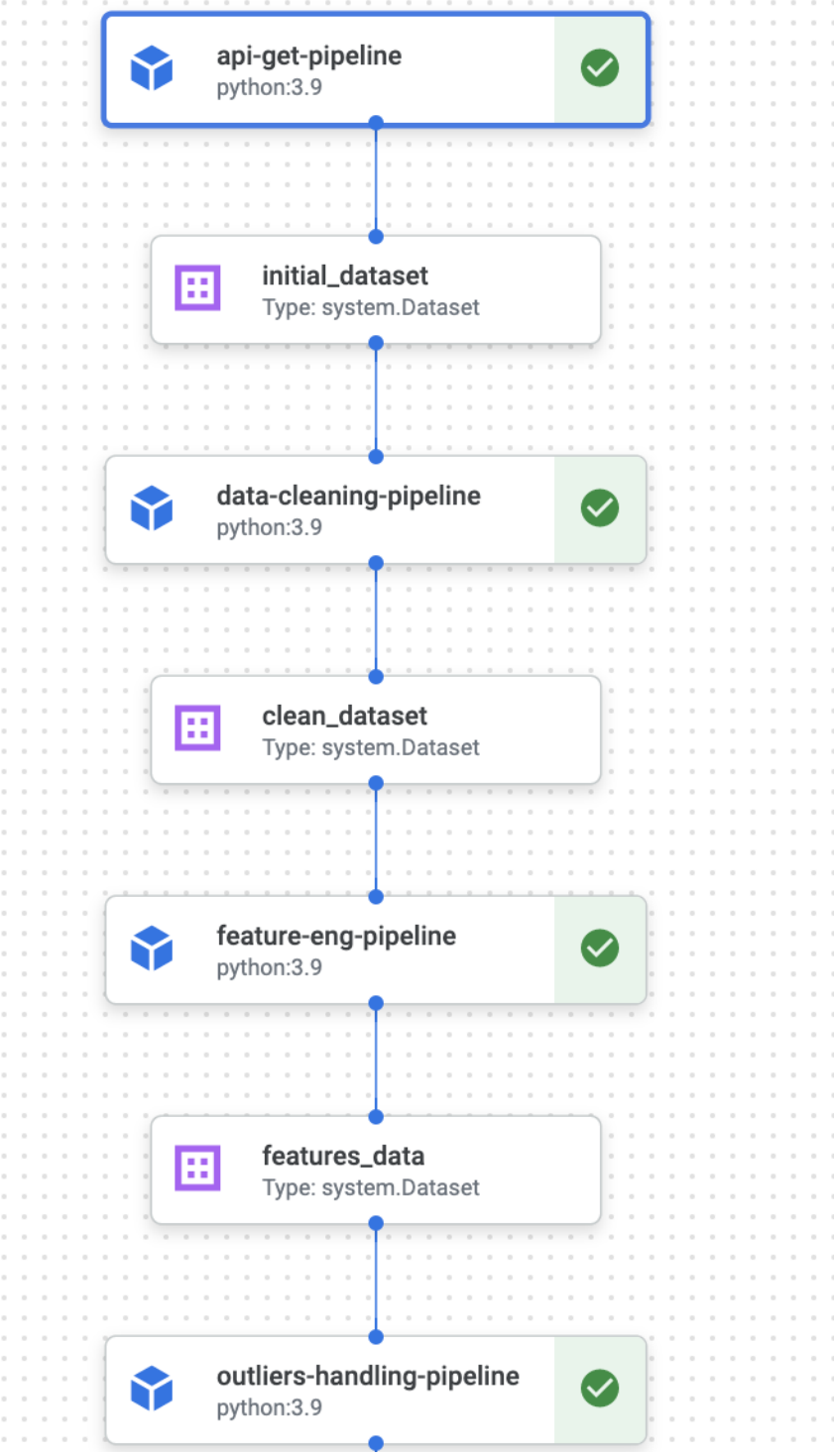


Figura 5.10: *Pipeline* - API Get + limpieza de datos + ingeniería de atributos + outliers

5.4.4 Listado de parámetros

Los parámetros con las que se entrena el modelo son las siguientes:

- **brand_mean (int)**: el promedio de precio de la marca del vehículo.

- **condition_new (bool)**: si el vehículo es nuevo (1) o no (0).
- **doors (int)**: cantidad de puertas del vehículo, de 2 a 5.
- **FUEL_TYPE_Diesel (bool)**: si el vehículo utiliza combustible diesel
- **FUEL_TYPE_Electrico (bool):** si el vehículo utiliza energía eléctrica como combustible
- **FUEL_TYPE_Hibrido (bool):** si el vehículo es híbrido (utiliza combustible y energía eléctrica)
- **FUEL_TYPE_Hibrido_Diesel (bool)**: si el vehículo es diesel e híbrido
- **FUEL_TYPE_Hibrido_Nafta (bool):** si el vehículo es híbrido y nafta
- **FUEL_TYPE_Nafta (bool):** si el vehículo utiliza como combustible nafta
- **FUEL_TYPE_Nafta_GNC (bool):** si el vehículo utiliza como combustible nafta o gnc
- **has_abs_brakes (bool)**: si tiene frenos abs.
- **has_alarm (bool)**: si tiene alarma.
- **has_air_conditioning (bool)**: si el vehículo tiene aire acondicionado o no.
- **has_driver_airbag (bool)**: si el conductor tiene airbag.
- **has_gps (bool)**: si el vehículo tiene gps o no.
- **has_passenger_airbag (bool)**: si el acompañante tiene airbag.
- **IS_FINANCEABLE (bool)**: si el vehículo es financiable (se puede vender en cuotas).
- **kilometers (int)**: cantidad de kilómetros recorridos.
- **listing_type_id_gold_premium (bool)**: si la publicación es premium_gold, es decir el tipo de publicación más caro. Una publicación más cara implica una mayor exposición en la plataforma.
- **official_store_id (bool)**: valor booleano sobre si la publicación fue hecha por una tienda oficial de Mercadolibre o no.
- **std_id_mean (int)**: el promedio de los precios del modelo
- **STEERING_Asistida (bool)**: si la dirección del volante es asistida
- **STEERING_Electrica (bool)**: si la dirección del volante es eléctrica
- **STEERING_Hidraulica (bool)**: si la dirección del volante es hidráulica
- **TRANSMISSION_Automatica (bool)**: si la caja de cambios es automática.
- **TRANSMISSION_Manual (bool):** si la caja de cambios es manual.
- **VEHICLE_BODY_TYPE_Hatchback (bool)**: si el cuerpo del vehículo es hatchback
- **VEHICLE_BODY_TYPE_Pick_Up (bool):** si el cuerpo del vehículo es pick up
- **VEHICLE_BODY_TYPE_Sedan (bool):** si el cuerpo del vehículo es sedán
- **VEHICLE_BODY_TYPE_SUV (bool):** si el cuerpo del vehículo es suv
- **VEHICLE_BODY_TYPE_Van (bool):** si el cuerpo del vehículo es van
- **vehicle_year (int)**: el año en que el vehículo fue construido.

Inicialmente se buscó entrenar con variables categóricas con el modelo *catboost*, pero como los resultados no fueron prometedores se procedió a transformar todas las variables a *integer* y *bool*.

5.3.5 Creación de Vertex Dataset

IBM define a los componentes de la siguiente manera [31]:
"Los componentes de software proporcionan las funciones en sus patrones de sistemas virtuales. Puede configurar los componentes y definir las interacciones entre los componentes para crear un entorno totalmente operativo cuando despliega su patrón en la nube.
En un patrón desplegado, los componentes que contiene se convierten en máquinas virtuales de la instancia del sistema virtual"

Vertex AI permite crear tus propios componentes a través de componentes de *Kubeflow* o utilizar componentes predefinidos por Google. Luego dichos componentes son conectados entre sí a través de un *pipeline.*

El listado de componentes que proporciona Google es muy extensivo y se puede consultar en la documentación oficial: https://cloud.google.com/vertex-ai/docs/pipelines/gcpc-list.

Una desventaja que se pudo observar de estos componentes es que no están hechos para interactuar directamente con los componentes creados por el usuario, ya que muchas veces los input que soportan están pensados para ser los output de otros componentes de Google.

Para la ingestión y limpieza de datos se utilizaron componentes creados por el usuario, ya que la limpieza requerida para llegar a el dataframe final fue muy compleja y específica de este caso de estudio y este dataset.
Sin embargo, las demás partes del proceso son más estándar y por lo tanto automatizables. Por ello se entendió que era viable utilizar los componentes predefinidos.

Además, utilizar *AutoML* aumentaría la autonomía del proceso, ya que no requiere de un humano para parametrizar el modelo. Por eso se decidió utilizar el componente predefinido *AutoMLTabularTrainingJobRunOp.*

Al estudiar el componente se observa que requiere como uno de sus argumentos un google.VertexDataset, mientras que los dataset generados hasta el momento eran system.Dataset.

Se llegó a la conclusión de que para poder utilizar el componente era necesario generar una capa intermedia que haga esa conversión del dataset.

Para ello fue utilizado el componente predefinido *TabularDatasetCreateOp*

El componente toma como *input* un *path* a un *bucket* o una tabla de BigQuery. Por ello el componente anterior (outliers-handling-pipeline) retornaba además de un system.Dataset, la escritura de un csv en el *bucket* del proyecto.

El path de dicho csv se incluye como input de *TabularDatasetCreateOp,* lo que retorna un archivo google.VertexDataset. Podemos acceder al mismo desde la sección Datasets de VertesAI.

En la imagen podemos observar que la consola nos indica el bucket donde está guardado el csv. Además nos devuelve el dato de cantidad de columnas (54) y filas (17.790).

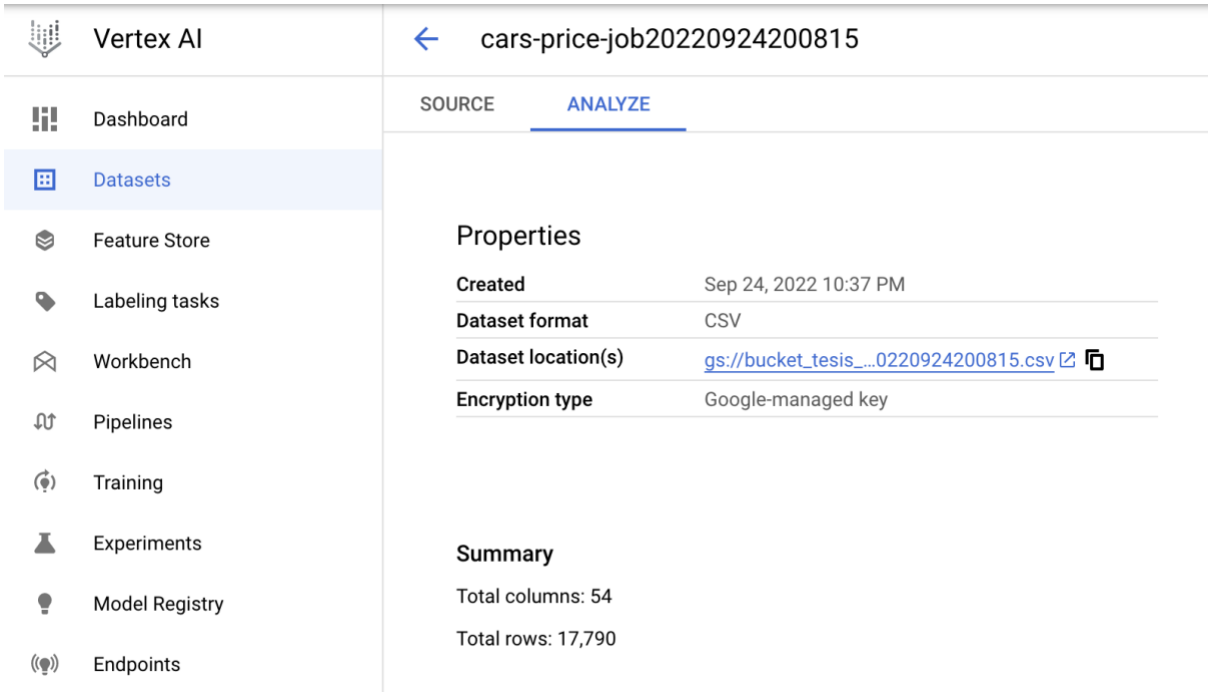


Figura 5.11: *Vertex Dataset*

Además muestra algunas estadísticas de los datos como la cantidad de nulos y la cantidad de valores distintos.

General statistics generated by Sep 25, 2022 9:22 PM GENERATE STATISTICS

Filter Enter property name or value

Column name	Missing % (count)	Distinct values
brand_mean	-	112
condition_new	-	2
DOORS	-	4
FINANCEABLE_BY_0	-	2
FINANCEABLE_BY_Creditel	-	2
FINANCEABLE_BY_Santander	-	2
FINANCEABLE_BY_Scotiabank	-	2
FUEL_TYPE_Diesel	-	2
FUEL_TYPE_Electrico	-	2

Figura 5.12: Fragmento de estadísticas de *features* de VertexAI

Con este paso se finaliza la parte de ingestión y limpieza de datos. El pipeline al momento es el siguiente:

Figura 5.13: *Pipeline* - obtención y análisis de datos + prep. y transformación de datos

5.5 Entrenamiento y parametrización del modelo

Figura 5.14: Flujo de aprendizaje automático - Entrenamiento y tuneo del modelo

5.5.1 Catboost

El modelo fue elegido por su capacidad de entrenar utilizando variables categóricas.

Según sus creadores [28], catboost se define como:

“CatBoost es un algoritmo para descensos de gradientes en árboles de decisión . Está desarrollado por investigadores e ingenieros de Yandex, y se utiliza para búsqueda, sistemas de recomendación, asistente personal, automóviles autónomos, predicción del tiempo y muchas otras tareas en Yandex y en otras empresas, incluidas *CERN, Cloudflare, Careem taxi.* Es de código abierto y puede ser utilizado por cualquier persona.”

La metodología utilizada fue *grid search*:

```
[ ]   1 grid = {'iterations': [500],
      2         'learning_rate': [0.06, 0.05, 0.04],
      3         'depth': [6, 8, 10, 14],
      4         'l2_leaf_reg': [2]}
```

Las primeras predicciones dieron resultados irracionales, llegando a predecir precios negativos o precios millonarios para vehículos comunes.

Luego de un análisis exhaustivo, se llegó a la conclusión de que esto se debía a la calidad de los datos, más específicamente de la divisa.

Cuando se analizaron los datos, no existía ningún valor en UYU. Sin embargo, en el momento del entrenamiento (que se ejecutó nuevamente el API GET para tener los últimos datos), estos sí tenían algunos valores en UYU. Se procedió a eliminar

dichas publicaciones del dataset ya que la mayoría eran errores del usuario al ingresar los datos.

Por otro lado, había publicaciones catalogadas en USD, pero al observar detenidamente el precio era incoherente, y seguramente el usuario de la plataforma eligió moneda USD pero publicó el monto como si fuera UYU. Por ello se procedió a eliminar los vehículos listados en más que 1MM USD, ya que la mayoría de ellos eran vehículos publicados en pesos, erróneamente catalogados como en USD.

Luego de esos cambios el modelo comenzó a predecir valores razonables, aunque aún no conformaban sus resultados.

Los parámetros que mejor performaron fueron:

```
1 print(model.get_params())
```

```
{'loss_function': 'RMSE', 'depth': 10, 'l2_leaf_reg': 2, 'iterations': 500, 'learning_rate': 0.06}
```

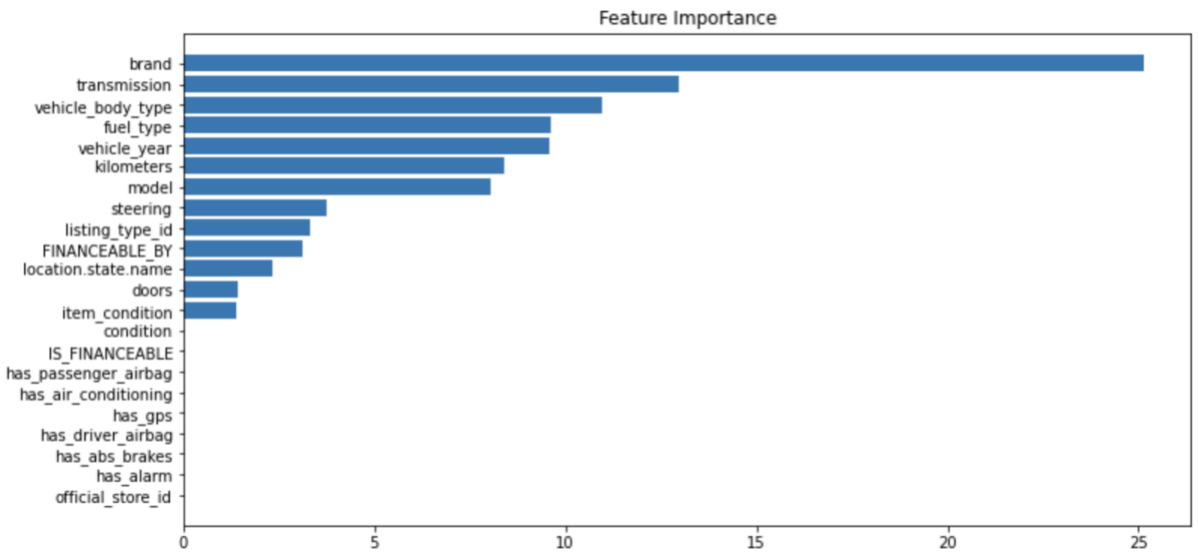
Sus métricas fueron un RMSE de 10.840 y un R2 de 0.68

```
1 pred = model.predict(x_test)
2 rmse = (np.sqrt(mean_squared_error(y_test, pred)))
3 r2 = r2_score(y_test, pred)
4 print("Testing performance")
5 print('RMSE: {:.2f}'.format(rmse))
6 print('R2: {:.2f}'.format(r2))
```

```
Testing performance
RMSE: 10840.13
R2: 0.68
```

En un principio en el *feature importance* se observan oportunidades de mejora, las variables booleanas no estaban siendo tenidas en cuenta por el modelo y algunas importantes como el kilometraje no están en las primeras posiciones.

Algunas variables tienen muy alta cardinalidad, lo cual puede estar jugando en contra al momento de predecir.



Más adelante se corrigió lo de las variables booleanas y las comenzó a tomar en cuenta el modelo, pero los resultados no mejoraron significativamente y se decidió avanzar con otros modelos.

5.5.2 Random Forest

Luego de los resultados insatisfactorios del modelo *catboost* se cambió la estrategia sobre las variables categóricas y fueron convertidas a variables booleanas.

Además se resolvió la cardinalidad de los datos de marca y modelo, tomando el promedio del precio de la marca y modelo. Reemplazando el texto por números y por lo dejando dejando de tener un problema con su cardinalidad.

Ambos cambios fueron muy satisfactorios, llevando a mejores resultados con un modelo poco iterado de Random Forest.

```
[ ]  1 predictions = rf.predict(test_features)
     2
     3 errors = abs(predictions - test_labels)
     4 mae = round(np.mean(errors), 2)
     5
     6 print('Mean Absolute Error:', mae, 'USD.')

     Mean Absolute Error: 3117.84 USD.
```

```
[ ]  1 mape = 100 * (errors / test_labels)
     2
     3 accuracy = round(100 - np.mean(mape),2)
     4 print('Accuracy:', round(accuracy, 2), '%.')

     Accuracy: 87.26 %.
```

Mejoraron tanto las métricas de *Mean Absolute Error (MAE)* como la precisión significativamente.

5.5.3 Vertex *AutoML*

Luego de los buenos resultados de *Random Forest,* se destaca que se llegó a un modelo satisfactorio con muy poca iteración en los hiper-parámetros. Lo que lleva a la hipótesis de que tanta inversión en el *dataframe* dió sus frutos, y no tanto el modelo en sí mismo.

Si esto fuese cierto, permitiría generar un modelo de *AutoML* con buenos resultados. Un modelo de *AutoML* sería lo ideal para *MLOps* ya que permitiría una ejecución del pipeline completo sin intervención humana, escalable en el tiempo.

Si bien hay un consenso en el campo de la inteligencia artificial de las definiciones de *AutoML*, hay varias formas diferentes de llevarlo a cabo.

Google define a *AutoML* de forma muy sencilla como: [30] “AutoML permite a los desarrolladores con experiencia limitada en aprendizaje automático entrenar modelos de alta calidad específicos para sus necesidades comerciales. Así pueden crear su propio modelo de aprendizaje automático personalizado en minutos.’

De forma de tener todo en un mismo ecosistema, y de continuar evaluando la herramienta de Google, se tomó la decisión de utilizar *AutoML* de *VertexAI*.

VertexAI provee de un componente para ejecutar *AutoML* en datos tabulares. El nombre del mismo es *AutoMLTabularTrainingJobRunOp.*

El mismo toma como input un google.VertexDataset como se mencionó anteriormente en el documento. Además es necesario aclarar si el modelo es de clasificación o regresión, cuál es la variable a predecir y la métrica a optimizar. En este caso fue elegido el error cuadrático medio (RMSE) ya que dicha métrica tiene buena respuesta ante los errores de alto monto.
Además hay que especificar el tiempo máximo de entrenamiento, en este caso se estableció 1000 milli horas, es decir una hora.

Por último se detallan las variables a tomar en cuenta así como su tipo de dato. Esto es positivo para el *pipeline* ya que si eventualmente MercadoLibre decide agregar variables a su API, el modelo no las tomaría en cuenta por defecto.

El código del componente de entrenamiento en el *pipeline* se encuentra en el Anexo

Las métricas del modelo son las siguientes:

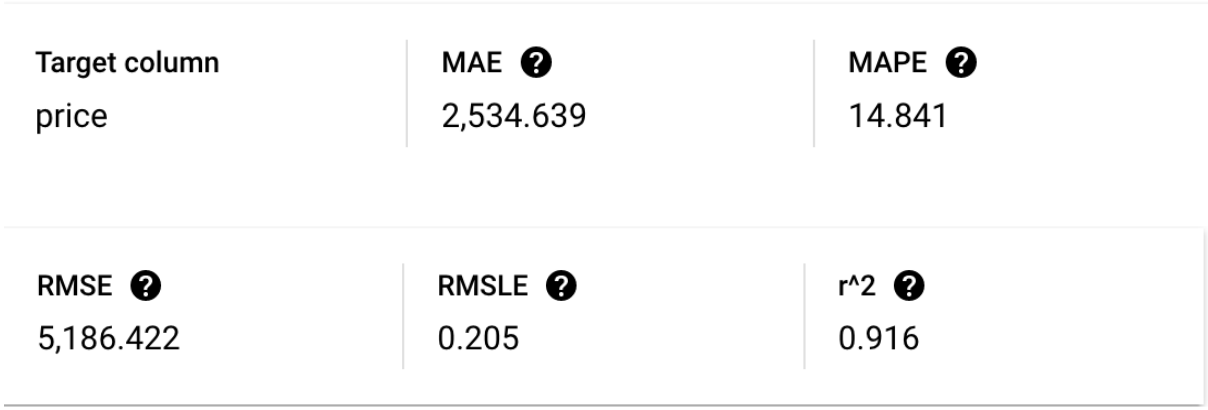


Figura 5.15: Métricas de VertexAI del modelo de regresión

Las métricas que proporciona la consola son las clásicas de un modelo de regresión, y no permiten seleccionar otras con el componente prehecho.

Google también proporciona una definición de cada métrica, las cuales son detalladas en el anexo 11.5

Los resultados del modelo son excelentes, llegando a un r^2 de 0,91 y el menor MAE hasta el momento con un valor de USD 2.535.

Por lo tanto, efectivamente se puede utilizar *AutoML* para este caso de estudio.

El Output de esta etapa del proceso es el modelo ya construido y optimizado.

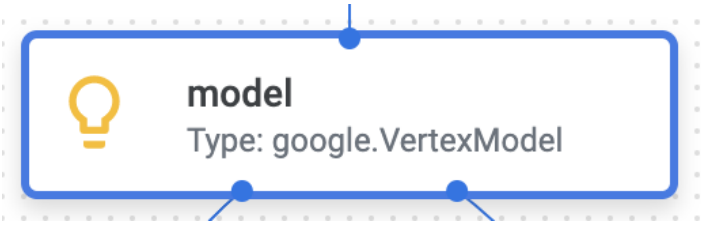


Figura 5.16: Modelo de VertexAI

Los modelos creados se pueden administrar en la sección “Model Registry” de VertexAI

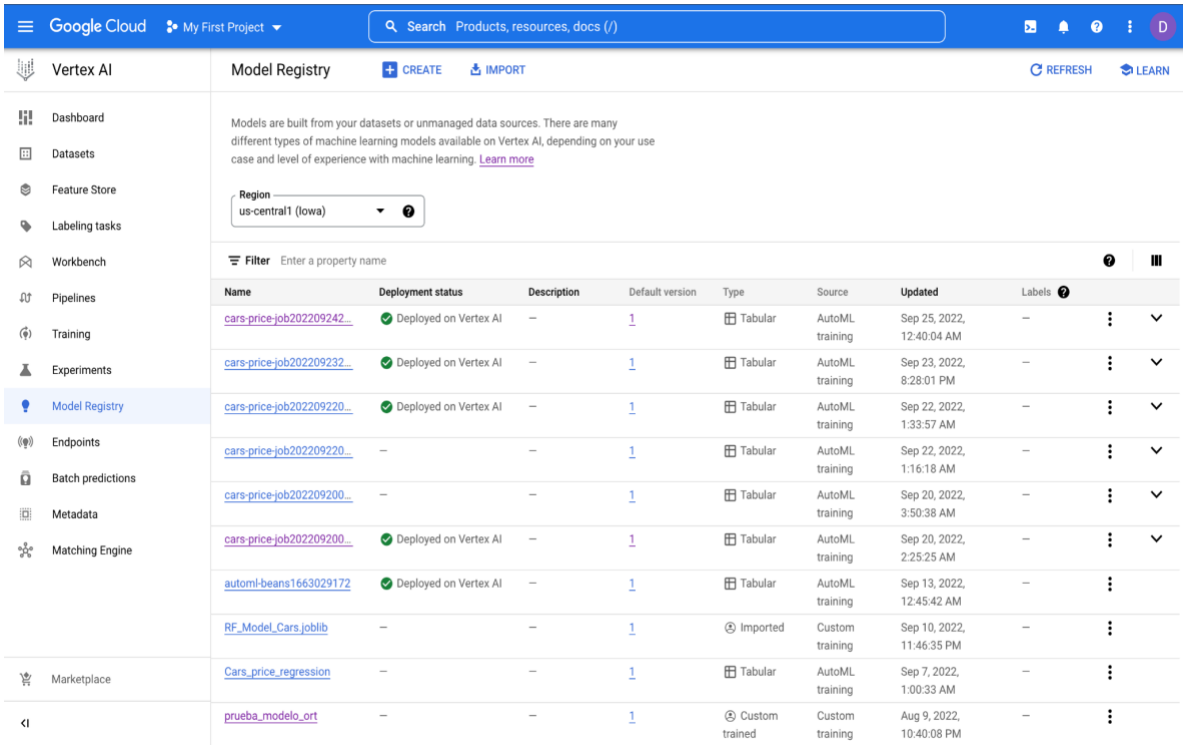


Figura 5.17: Registro de modelos de VertexAI

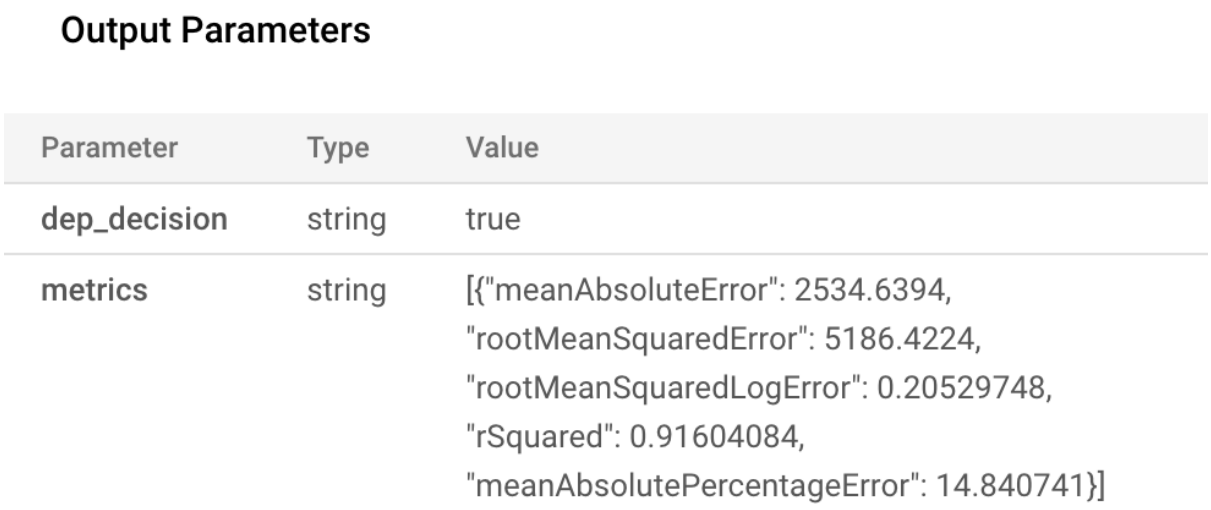
5.6 Evaluación y despliegue del modelo

Figura 5.18: Flujo de aprendizaje automático - Evaluación y despliegue del modelo

Un paso fundamental de *MLOps* son los tests automatizados, para asegurar la calidad de lo que se está desplegando en producción.

En los componentes creados por Google no se encontró uno de evaluación, por lo que creamos nuestro propio componente para esta etapa.

Como las métricas que retornó el modelo son suficientes para evaluar el éxito o fracaso de este caso de estudio, el componente consistió en acceder a las mismas programáticamente.



Output Parameters

Parameter	Type	Value
dep_decision	string	true
metrics	string	[{"meanAbsoluteError": 2534.6394, "rootMeanSquaredError": 5186.4224, "rootMeanSquaredLogError": 0.20529748, "rSquared": 0.91604084, "meanAbsolutePercentageError": 14.840741}]

Figura 5.19: output del componente de evaluación: KPI del modelo y decisión de desplegar

En el *pipeline* se establece que los pasos siguientes se ejecutarán solamente si la evaluación del modelo retorna que el mismo es lo suficientemente bueno.

En el Anexo 11.5 se puede observar a nivel de código la condición del *pipeline* que según las métricas del modelo, decide desplegar el modelo o no.

Para la evaluación se determinó a la métrica MAPE como la definitiva para tomar la decisión de desplegar o no.

Dicha decisión se basó en que al ser una métrica porcentual, tiene mayores probabilidades de mantenerse vigente ante cambios en la distribución de los datos.

Una vez que la evaluación resulta positiva, se procede al despliegue del modelo

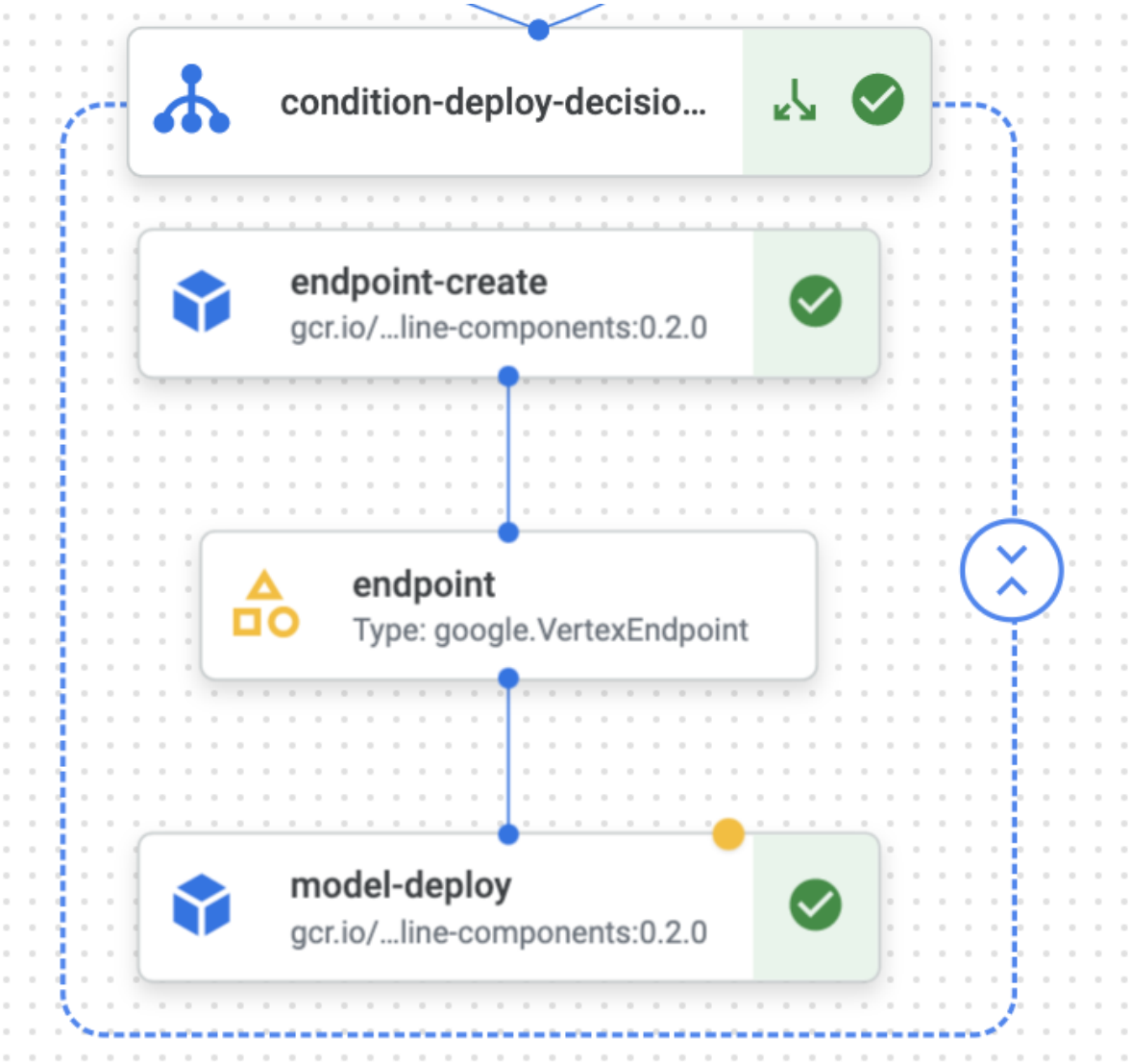


Figura 5.20: *Pipeline* - despliegue

Tanto la creación del *endpoint* como el despliegue del mismo se hicieron utilizando componentes pre-hechos por Google.

Para la creación del endpoint se utilizó el componente *EndpointCreateOp*

Los endpoint creados se pueden administrar en la sección “Endpoints” de VertexAI

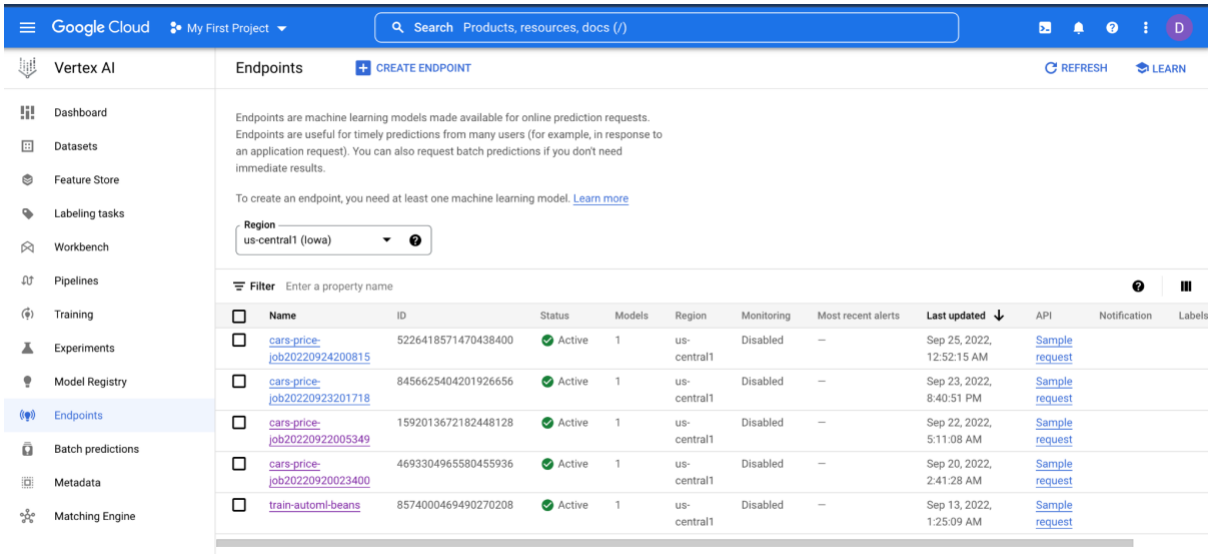
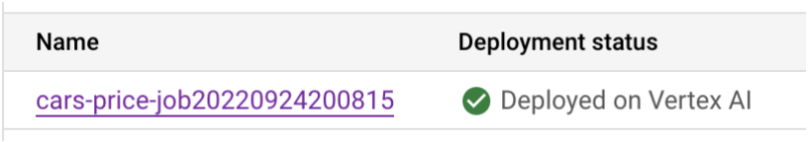


Figura 5.21: Registro de *endpoints* en VertexAI

Una vez que el endpoint es creado, el siguiente paso y el final es el despliegue del modelo sobre el mismo. Para ello se utilizó el componente de Google *ModelDeployOp*

En la sección Model Registry de Vertex se puede observar el estatus de desplegado en el modelo



6 Resultados

A continuación se presentan los resultados haciendo foco en los objetivos planteados al inicio del proyecto.

6.1 Aprendizaje automático de punta a punta

Este objetivo consiste en crear una solución de aprendizaje automático de punta a punta: desde el planteamiento del problema desde la perspectiva de negocio hasta el modelo desplegado en producción.

El objetivo se cumplió en su totalidad, ya que se generó un *pipeline* que inicia obteniendo datos de una API pública y termina con la creación de un endpoint y despliegue de un modelo a través del mismo.

6.2 Utilizar la metodología MLOps

El objetivo es diseñar la solución en base a la metodología de *MLOps*, de forma que el modelo se mantenga actualizado en el tiempo con la menor intervención humana posible.

El objetivo se cumplió en su totalidad ya que el pipeline no precisa de intervención humana en ninguna parte de su proceso. Además está construido en su totalidad sobre *VertexAI*, lo que facilita su mantenimiento.

6.3 KPI

Este objetivo consiste en lograr un modelo aceptable en cuanto a sus métricas de predicción.

Este objetivo se cumplió con creces, ya que se pudo obtener un r^2 de 0,91 sobre un máximo de 1. Así como buenas métricas de MAE, MAPE y RMSE.

Que esto se haya logrado utilizando *AutoML*, significa que se pudo lograr este objetivo priorizando también la escalabilidad y mantenimiento.

6.4 Benchmarking

El siguiente objetivo se trata de realizar un análisis de las herramientas disponibles en la industria para llevar a cabo el proyecto

El objetivo se cumplió en su totalidad, haciendo un análisis exhaustivo de la industria y de sus cuatro mayores jugadores, basándose en información objetiva de la consultora Gartner.

6.5 Conocimiento de la herramienta

Por último, otro objetivo es desarrollar conocimiento en la herramienta escogida.

Este objetivo se cumplió en su totalidad, desarrollando conocimiento sobre variados aspectos de Google Cloud Platform. Destacándose el aprendizaje de Vertex AI y de *Kubeflow* para hacer un mejor uso de los *pipelines* de Vertex.

7 Discusión

En la presente sección se describen los aspectos que tienen oportunidades de mejora en el proyecto así como en lo personal.

A nivel del proyecto el modelo resultó satisfactorio. Sin embargo se podría profundizar el trabajo haciendo un análisis de las métricas del modelo por diferentes particiones, para así entender si hay alguna rama en particular donde performa peor de lo esperado.

Las variables de entrada del modelo son sencillas de ingresar por el usuario a excepción del precio promedio de la marca y del modelo. Esos datos son obtenibles ya que el mismo pipeline los obtiene, pero tendría que agregarse una capa intermedia que transforme el modelo y marca ingresados por el usuario por sus precios promedio, de forma de que se mantenga fácil de utilizar el modelo.

Si bien el modelo se logró desplegar, sería útil conectar el endpoint con un frontend, para facilitar el uso de cualquier usuario.

A nivel personal desarrollé mucho conocimiento sobre *Kubeflow y VertexAI*. Pero entiendo que hay una oportunidad en entender más a fondo cómo trabaja *AutoML* de Vertex, así como de explorar *Explainable AI* que por su nombre parece una feature muy interesante de Google.

8 Conclusiones

Caso de negocio

- Se verifica que los datos públicos de MercadoLibre son suficientes para realizar buenas predicciones sobre precios de vehículos nuevos y usados en el Uruguay. Si bien para lograrlo se tiene que hacer una limpieza exhaustiva de los datos y pensar fuera de la caja para obtener la totalidad de los mismos.
- En este caso de estudio se confirma que es posible obtener un buen modelo haciendo foco en los datos y no en los algoritmos de modelado. Este principio se conoce como “data-centric AI” y tiene a Andrew Ng, CEO de Google como uno de sus mayores promotores. En el artículo [29] del MIT, se habla de como Andrew Ng y otros referentes de la industria comparten esta visión.

Vertex AI y Kubeflow pipelines

- Se implementó exitosamente un pipeline de Vertex AI que cumple los requisitos de la metodología *MLOps*.
- Se observa que la curva de aprendizaje para utilizar la herramienta es alta, así como los costos de la misma. Por ambos factores puede no valer la pena para proyectos que por su poca complejidad no ameriten un sistema con tanto poder de escalabilidad.
- La documentación oficial es buena pero no hay una gran comunidad de desarrolladores utilizando la herramienta y escribiendo artículos de la misma, lo que dificulta el desarrollo. Se entiende que esto se debe a que la herramienta es muy nueva así como a los elevados costos de utilizarla.

Conocimiento personal

- Personalmente cumplí el objetivo de implementar un proyecto de aprendizaje automático de punta a punta. Si bien en la industria las diferentes etapas del proceso pueden ser realizadas por diferentes perfiles (el científico de datos limpia los datos y entrena el modelo y el ingeniero en aprendizaje automático lo despliega), considero que es importante que los profesionales de datos tengan un conocimiento mínimo sobre todo el proceso para poder tener una mejor comunicación con sus colegas y aumentar la probabilidad de éxito de los proyectos.
- Profundizar en la metodología *MLOps* fue muy desafiante y enriquecedor, especialmente utilizando las últimas tecnologías y aprendiendo sobre nuevas herramientas en el camino.

9 Referencias bibliográficas

[1] Databricks, “Encuesta de CIO: Los 3 principales desafíos para adoptar la IA y cómo superarlos”. [Online]. Available: https://databricks.com/blog/2018/12/06/cio-survey-top-3-challenges-adopting-ai-and-how-to-overcome-them.html

[2] VentureBeat, “Why do 87% of data science projects never make it into production?”.[Online]. Available: https://venturebeat.com/2019/07/19/why-do-87-of-data-science-projects-never-make-it-into-production/

[3] KDNuggets, “Models Are Rarely Deployed: An Industry-wide Failure in Machine Learning Leadership”. [Online]. Available: https://www.kdnuggets.com/2022/01/models-rarely-deployed-industrywide-failure-machine-learning-leadership.html

[4] Sakshi Gupta, “Data Science vs Data Analytics vs. Machine learning vs. Artificial Intelligence. [Online]. Available: https://www.springboard.com/blog/data-science/data-science-vs-data-analytics-vs-machine-learning-vs-artificial-intelligence/

[5] Coursera, “Data Science vs. Machine Learning: What’s the Difference?”. [Online]. Available: https://www.coursera.org/articles/data-science-vs-machine-learning

[6] Rakesh Kumar Maddipati, “Machine Learning Life-cycle Explained!”. [Online]. Available: https://www.analyticsvidhya.com/blog/2021/05/machine-learning-life-cycle-explained/

[7] Analytics Vidhya, “Machine Learning Life-cycle Explained!“. [Online]. Available: https://www.analyticsvidhya.com/blog/2021/05/machine-learning-life-cycle-explained/

[8] MathWorks, “Cross-Validation”. [Online]. Available: https://ch.mathworks.com/discovery/cross-validation.html

[9] Cambridge Spark, “The year of MLOps”. [Online]. Available: https://www.cambridgespark.com/info/the-year-of-mlops

[10] Cristiano Breuel, “ML Ops: Machine Learning como disciplina de ingeniería“. [Online]. Available: https://medium.com/@cbreuel/ml-ops-machine-learning-como-disciplina-de-ingenier%C3%ADa-7e0c49f0a6f6

[11] Tensorflow, “Comience con la validación de datos de Tensorflow” [Online]. Available: https://www-tensorflow-org.translate.goog/tfx/data_validation/get_started?_x_tr_sl=en&_x_tr_tl=es&_x_tr_hl=en

[12] Cristiano Breuel, “ML Ops: Machine Learning como disciplina de ingeniería”.[Online]. Available: https://medium.com/@cbreuel/ml-ops-machine-learning-como-disciplina-de-ingenier%C3%ADa-7e0c49f0a6f6

[13] Dra. Larysa Visengeriyeva, Anja Kammer, Isabel Bär, Alexander Kniesz y Michael Plöd “Por qué es posible que desee utilizar el aprendizaje automático”. [Online]. Available: https://ml-ops.org/content/motivation

[14] MLOpsOrg, “Why you Might Want to use Machine Learning” [Online]. Available: https://ml-ops.org/content/motivation

[15] Gartner, “Magic Quadrant for Cloud AI Developer Services” [Online]. Available: https://www.gartner.com/doc/reprints?id=1-29DOQV50&ct=220311&st=sb

[16] Google, “Ubicaciones”, [Online]. Available: https://cloud.google.com/vertex-ai/docs/general/locations?hl=es-419#americas

[17] Amazon, “Zonas de disponibilidad y regiones” [Online]. Available: https://aws.amazon.com/es/about-aws/global-infrastructure/regions_az/

[18] IBM, “Region and data center locations for resource deployment” [Online]. Available: https://cloud.ibm.com/docs/overview?topic=overview-locations

[19] Microsoft, “Regions and availability zones”. [Online]. Available: https://docs.microsoft.com/en-us/azure/availability-zones/az-overview

[20] Microsoft, “Price calculator”. [Online]. Available: https://azure.microsoft.com/en-us/pricing/calculator/

[21] Amazon, “Calculator”. [Online]. Available: https://calculator.aws/#/addService/SageMaker

[22] IBM, “Unidades de capacidad”. [Online]. Available: https://www.ibm.com/docs/es/cdfsp/7.6.1.x?topic=overview-capacity-units

[23] Google, “Compila una canalización” . [Online]. Available: https://cloud.google.com/vertex-ai/docs/pipelines/build-pipeline?hl=es-419

[24] MercadoLibre, ”Autenticación y autorización” [Online]. Available:https://developers.mercadolibre.com.uy/es_ar/autenticacion-y-autorizacion

[25] Google, “Secret manager”. [Online]. Available: https://cloud.google.com/secret-manager?hl=es

[26] Google, “Access secret version”. [Online]. Available: https://cloud.google.com/secret-manager/docs/samples/secretmanager-access-secret-version

[27] Catboost, “Categorical features” [Online]. Available: https://catboost.ai/en/docs/features/categorical-features

[28] Catboost, “Catboost“. [Online]. Available: https://catboost.ai/en/docs/

[29] MIT, “Why it’s time for 'data-centric artificial intelligence' “. [Online]. Available: https://mitsloan.mit.edu/ideas-made-to-matter/why-its-time-data-centric-artificial-intelligence#:~:text=Ng%20advocates%20for%20%E2%80%9Cdata%2Dcentric,experts%2C%20delivered%20every%20Tuesday%20morning.

[30] Google, “AutoML”. [Online]. Available: https://cloud.google.com/AutoML

[31] IBM, “Componentes de software”. [Online]. Available: https://www.ibm.com/docs/es/wsr-and-r/8.5.6?topic=artifacts-software-components

10 Anexos

10.1 Access Secret y API Get

Para obtener datos desde la API es necesario ingresar las siguientes credenciales:

```
credentials = {
        "grant_type":"authorization_code",
        "client_id":client_id,
        "client_secret":client_secret,
        "code":"TG-63274b136110a000011b93cf-184062779",
        "redirect_uri":"https://prestabot-ui.herokuapp.com/"
}
```

- grant_type: es el tipo de acceso, y no varía.
- client_id: es el identificador de mi app creada en la web de desarrolladores de MercadoLibre, está protegido por el secret manager.
- client_secret: es la clave de mi app creada en la web de desarrolladores de MercadoLibre, también está protegido por el secret manager.
- code: es un código que puede utilizarse una única vez y al cual no se puede acceder programáticamente por definiciones de negocio de MercadoLibre.
- redirect_url: es la dirección web que registré en mi app, a la cual me redirigen y desde la cual obtengo el “code” en la url de redirección.

El “code” se obtiene clickeando el siguiente link: https://auth.mercadolibre.com.uy/authorization?response_type=code&client_id={}&redirect_uri=https://prestabot-ui.herokuapp.com/

Dicho link contiene mi client_id así como mi redirect_url.

Una vez ingresadas las credenciales, se procede a realizar la solicitud del token y el refresh token.

```
url = 'https://api.mercadolibre.com/oauth/token'
token_response = requests.post(url, data = credentials)
```

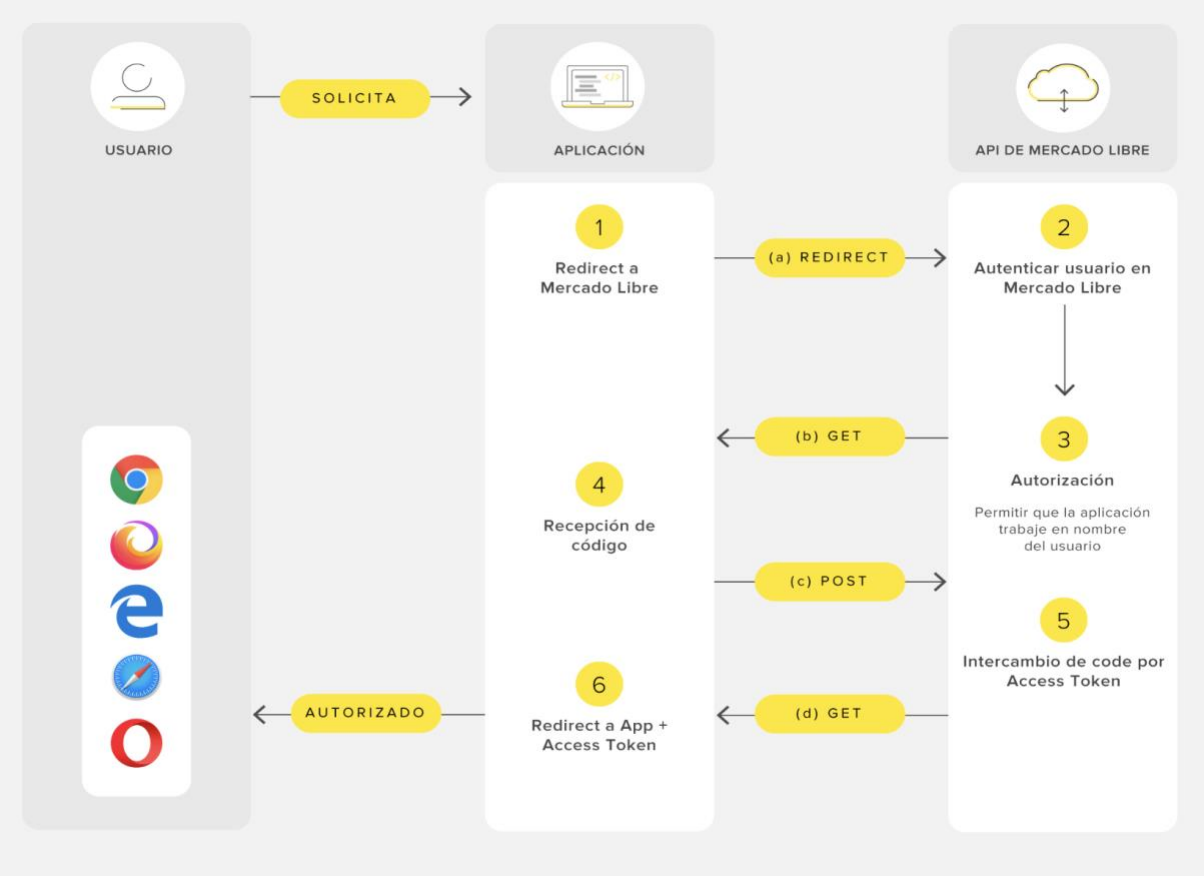


Figura 10.1: Flujo de la API de MercadoLibre. Fuente: [24]

Si intento obtener el *"code"* programáticamente, MercadoLibre lo devuelve hasheado. Además si intento obtenerlo mediante un bot, la empresa pide autorización de acceso mediante second factor.

Por ello, la primera vez que se ejecuta el código, se debe hacer este paso manualmente. Sin embargo, una vez hecha esa primera ejecución, se puede acceder a la API mediante un *refresh token* que devuelve esa primera ejecución.


```
{"access_token":"APP_USR-3053736294689162-091812-f83e97bfd97900a59ad63de7b5b1d593-184062779","token_type":"bea
rer","expires_in":21600,"scope":"offline_access read","user_id":184062779,"refresh_token":"TG-63274b1e02139000
01c0a03d-184062779"}
```


Las interacciones con la API mediante el *refresh_token* son de la siguiente manera:

```
myobj = {
        "grant_type":"refresh_token",
        "client_id":client_id,
        "client_secret":client_secret,
        "refresh_token": "{}".format(refresh_token),
}

url = 'https://api.mercadolibre.com/oauth/token'
token_response = requests.post(url, data = myobj)
```

Ello nos devuelve otro *refresh_token*, con el cual podemos hacer la siguiente consulta. Generando así un ciclo que ya no requiere la interacción de un humano.

A diferencia del *access_token*, el *refresh_token* no expira. Por ello podemos armar un proceso de *MLOps* desde el mismo, siempre y cuando la máquina virtual se mantenga activa para no perder el dato del último *refresh_token*; o si el dato es guardado de manera segura.

Anteriormente se mencionó que había datos protegidos por el *secret manager. Secret Manager* es una funcionalidad de *Google Cloud Platform* que permite almacenar de forma segura información sensible. La definición de Google es: [25] “Secret Manager es un sistema práctico y seguro para almacenar claves de API, contraseñas, certificados y otros datos sensibles. Además, ofrece una única fuente de información centralizada para acceder a los secretos, gestionarlos y auditarlos en todos los componentes de Google Cloud” .

La función a utilizar se obtuvo de la documentación oficial [26] y es la siguiente:

```python
def access_secret_version(project_id, secret_id, version_id):
    """
    Accede a un dato protegido por el secret manager de gcp
    """

    # Librerias.
    import google_crc32c
    from google.cloud import secretmanager

    # Crear el cliente de secret manager.
    client = secretmanager.SecretManagerServiceClient()

    # Nomenclatura
    name = f"projects/{project_id}/secrets/{secret_id}/versions/{version_id}"

    # Acceso a la version
    response = client.access_secret_version(request={"name": name})

    # Verificacion datos
    crc32c = google_crc32c.Checksum()
    crc32c.update(response.payload.data)
    if response.payload.data_crc32c != int(crc32c.hexdigest(), 16):
        print("Data corruption detected.")
        return response

    # Retorno

    payload = response.payload.data.decode("UTF-8")
    return payload
```

Con dicha fórmula el acceso se vuelve muy sencillo

```python
client_secret = access_secret_version(project_id="15207959361", secret_id="client_secret", version_id="latest")
```

En conclusión de este capítulo, se pudo acceder a los datos de MercadoLibre sin la necesidad de la interacción de un humano y asegurando las claves de acceso desde la misma herramienta de Google.

10.2 Primer acceso a los datos

Por medio de la API, se llega a la conclusión de que la categoría a consultar es la de id MLU1734 llamada 'Autos, Motos y Otros'. Dentro de esa categoría, se consulta la subcategoría de id MLU1744 llamada 'Autos y Camionetas'.

El paginado permite consultar de a 50 registros, por lo que se combinan los parámetros *offset* con *limit* para extender la capacidad de consulta. Una dificultad es que por definición de negocio Mercadolibre topea las consultas en 4.000 registros.

La cantidad de publicaciones de autos y camionetas ha oscilado entre 17.000 y 19.000 registros, por lo que se buscó la forma de obtener todos esos registros esquivando el límite de 4.000 impuesto por la compañía . Para así lograr construir el mejor modelo posible y no dejar afuera información valiosa.

La obtención de los datos se realizó en dos pasos, acorde a las limitaciones de la API:

- Obtención del listado de ítems (un ítem es un artículo que se vende en una publicación, el mismo modelo de auto vendido por dos vendedores diferentes son dos ítems diferentes).
- A partir del listado de ítems, la obtención de los datos de cada publicación

El límite de 4.000 registros afecta al primer punto, ya que el punto 2 se ejecuta sin *offset* ni *limit*, sino item por item.

La fórmula a utilizar para la interacción con la API fue la siguiente:

```
def api_get_operation(url, max_offset, item_list):

    offset = 0
    r = get_response(url, offset)
    maximum = get_max_paging(url, offset)

    while r.status_code == 200 and offset < maximum and offset < max_offset:
        r = get_response(url, offset)
        data = r.json()
        length = len(data['results'])
        for i in range(length):
            item_id = data['results'][i]['id']
            item_list.append(item_id)
        offset += 50
```

Es decir, ejecuta el loop siempre y cuando:

- la API responda correctamente
- el parámetro offset no sobrepase la cantidad de publicaciones
- el parámetro offset no sobrepase el límite máximo (4.000)

El resultado de dicha fórmula es una lista (item_list) con cada uno de los ítems obtenidos.

La API permite consultar por categoría, y esa es la primera consulta a realizar y la más sencilla. Con ella se obtienen 4.000 publicaciones sin repetir.

Otra opción de la API es consultar una búsqueda específica (como si escribiera en el buscador de MercadoLibre). Se validó que uno de los campos devueltos es la categoría, por lo que luego de utilizar esta función se filtra la categoría para limpiar resultados indeseados.

Luego de diferentes intentos, se determinó que la combinación de auto + tipo de cuerpo de auto y camioneta + tipo de cuerpo de camioneta; así como de la marca de los vehículos, eran buenas maneras de obtener nuevos registros relacionados de forma escalable y que pueda perdurar en el tiempo. Ya que no es común que salgan nuevos tipos de cuerpos de vehículos (sedán, hatchback, entre otras), ni que surjan nuevas marcas de vehículos constantemente.

Por lo tanto, a los 4.000 registros obtenidos de categoría se suman tres consultas.

1. Auto + cuerpo de auto ['Cabriolet','Coupé','Hatchback','Monovolumen','Sedán']
2. Camioneta + cuerpo de camioneta ['Crossover','Furgón','Light Truck','Minivan','Pick-Up','Rural','SUV','Van']
3. Marca ['Chevrolet','Volkswagen','Renault','Fiat','Peugeot','Nissan','Suzuki','Ford','Toyota','Hyundai', 'Citroën','Mercedes-Benz','BMW','Honda','Chery','Mitsubishi','Great Wall','Kia','Geely', 'DFSK','BYD','Jeep','Audi','Faw','Porsche','RAM','Land Rover','Maxus','Jaguar','Ferrari']

Luego se quitan duplicados, y como resultado se obtiene más del 90% de las publicaciones de autos y camionetas.

Una vez obtenida la lista de items, se procede a utilizarla para consultar los atributos de los mismos de la siguiente manera:

```
for i in range(len(item_list)):
    item="https://api.mercadolibre.com/items/{}".format(item_list[i])
    item_add = requests.get(item)
    item_add = item_add.json()
    final_list.append(item_add)
```

Es decir, extrae los datos de cada artículo con una interacción diferente con la API. Dichos datos son devueltos en formato json.

Con la librería pandas de manipulación de datos, dichos los datos son convertidos desde json a *dataframe* de la siguiente forma:

```
for row in range(1, list_len):
    new_row = final_list[row]
    new_row = json_normalize(new_row)
    df = pd.concat([df, new_row], ignore_index = True)
```

Ya en formato dataframe, se procede a filtrar las filas que son de la categoría ‘MLU1744’ (Autos y camionetas).

10.3 Apertura de columnas array

El siguiente es un ejemplo de una fracción de la columna *array* atributos. La misma columna contiene una gran cantidad de diccionarios con la misma clave en todos y diferentes valores según la publicación.


```
{
   "id":"ITEM_CONDITION",
   "name":"Condición del ítem",
   "value_id":"2230581",
   "value_name":"Usado",
   "value_struct":"NULL",
   "values":[
      {
         "id":"2230581",
         "name":"Usado",
         "struct":"NULL"
      }
   ],
   "attribute_group_id":"",
   "attribute_group_name":""
},
{
   "id":"COLOR",
   "name":"Color",
   "value_id":"52055",
   "value_name":"Blanco",
   "value_struct":"NULL",
   "values":[
      {
         "id":"52055",
         "name":"Blanco",
         "struct":"NULL"
      }
   ],
   "attribute_group_id":"ADICIONALES",
   "attribute_group_name":"Adicionales"
}
```


Existen 168 diccionarios por cada fila de dicha columna. Cada diccionario tiene una clave llamada “id” cuyo valor corresponde a la variable que incluye el diccionario.

También tienen una clave llamada “value_name” cuyo valor se corresponde con el valor que toma el id en esa publicación. Es decir que de cada diccionario precisamos extraer los id que pasarían a ser el nombre de la nueva columna, y value_name que sería el valor a insertar en ella.
Esta transformación de datos es un proceso muy costoso, por lo que luego de mucho análisis exploratorio se procedió a extraer solamente los id que se consideraba que tienen potencial de agregar valor al modelo, según conocimiento de negocio. El listado de dichos id se almacenó en la lista “attribute_features”.

La fórmula para aperturar las columnas array es la siguiente:

```
attribute_features = ['ITEM_CONDITION','STEERING','HAS_AIR_CONDITIONING','HAS_GPS', 'GEAR_NUMBER',
                      'CITY_AVERAGE_CONSUMPTION','BRAND','DOORS','ENGINE','FUEL_TYPE','KILOMETERS',
                      'MODEL','TRANSMISSION','TRIM','VEHICLE_BODY_TYPE','VEHICLE_YEAR','HAS_ABS_BRAKES',
                      'HAS_ALARM','HAS_DRIVER_AIRBAG','HAS_PASSENGER_AIRBAG']
```

```
for fila in df_atr: # para cada fila
    for diccio in range(len(ast.literal_eval(df_atr[fila]))): # para cada uno de los diccionarios
        if ast.literal_eval(df_atr[fila])[diccio]["id"] not in attribute_features:
            pass
        else:
            variable_name = ast.literal_eval(df_atr[fila])[diccio]["id"]
            dict_atributos[variable_name]=ast.literal_eval(df_atr[fila])[diccio]["value_name"]
    df2= pd.DataFrame(dict_atributos, index=[fila])
    data_atributos = pd.concat([pd.DataFrame(data_atributos), df2], ignore_index=True)
```

Es decir:

1. Para cada fila, y para cada diccionario en cada fila
2. Observa si el id (nombre columna) está en el listado de features predefinido. Si no lo está, pasa al siguiente diccionario.
3. Si lo está, guarda el id (nombre columna) y el valor (valor fila) en un diccionario
4. Transforma el diccionario en *dataframe*

10.4 Componente de entrenamiento de modelo con AutoML en VertexAI

El siguiente es como se definió a nivel de código el componente de entrenamiento de AutoML en VertexAI

```
    training_task = gcc_aip.AutoMLTabularTrainingJobRunOp(
        project=project,
        display_name=display_name,
        optimization_prediction_type="regression",
        budget_milli_node_hours=1000,
        dataset=dataset_create_task.outputs["dataset"],
        target_column="price",
        optimization_objective="minimize-rmse",
        column_transformations=[
            {"numeric": {"column_name": "brand_mean"}},
            {"categorical": {"column_name": "condition_new"}},
            {"numeric": {"column_name": "DOORS"}},
            {"categorical": {"column_name": "FUEL_TYPE_Diesel"}},
            {"categorical": {"column_name": "FUEL_TYPE_Hibrido"}},
            {"categorical": {"column_name": "FUEL_TYPE_Nafta"}},
            {"categorical": {"column_name": "HAS_ABS_BRAKES"}},
            {"categorical": {"column_name": "HAS_AIR_CONDITIONING"}},
            {"categorical": {"column_name": "HAS_ALARM"}},
            {"categorical": {"column_name": "HAS_DRIVER_AIRBAG"}},
            {"categorical": {"column_name": "HAS_GPS"}},
            {"categorical": {"column_name": "HAS_PASSENGER_AIRBAG"}},
            {"categorical": {"column_name": "IS_FINANCEABLE"}},
            {"numeric": {"column_name": "KILOMETERS"}},
            {"categorical": {"column_name": "listing_type_id_gold_premium"}},
            {"categorical": {"column_name": "official_store_id"}},
            {"numeric": {"column_name": "std_id_mean"}},
            {"categorical": {"column_name": "STEERING_Asistida"}},
            {"categorical": {"column_name": "STEERING_Electrica"}},
            {"categorical": {"column_name": "STEERING_Hidraulica"}},
            {"categorical": {"column_name": "TRANSMISSION_Automatica"}},
            {"categorical": {"column_name": "TRANSMISSION_Manual"}},
            {"categorical": {"column_name": "VEHICLE_BODY_TYPE_Hatchback"}},
            {"categorical": {"column_name": "VEHICLE_BODY_TYPE_Pick_Up"}},
            {"categorical": {"column_name": "VEHICLE_BODY_TYPE_Sedan"}},
            {"categorical": {"column_name": "VEHICLE_BODY_TYPE_SUV"}},
            {"categorical": {"column_name": "VEHICLE_BODY_TYPE_Van"}},
            {"numeric": {"column_name": "VEHICLE_YEAR"}},
        ],
```

Imagen: código de entrenamiento del modelo con AutoML de VertexAI

10.5 Definición de métricas para modelos de regresión

Google nos presenta las siguientes definiciones sobre métricas de modelos de regresión:

MAE: El error absoluto medio (MAE) es el promedio de las diferencias absolutas entre los valores observados y predichos. Un valor bajo indica un modelo de mayor calidad, donde 0 significa que el modelo no cometió errores. La interpretación de MAE depende del rango de valores de la serie. MAE tiene la misma unidad que la columna de destino.

MAPE: El error porcentual absoluto medio (MAPE) es el promedio de los errores porcentuales absolutos. MAPE varía de 0% a 100%, donde un valor más bajo indica un modelo de mayor calidad. MAPE se convierte en infinito si los valores 0 están presentes en los datos de verdad del terreno.

RMSE: El error cuadrático medio (RMSE) es la raíz de las diferencias al cuadrado entre los valores observados y predichos. Un valor más bajo indica un modelo de mayor calidad, donde 0 significa que el modelo no cometió errores. La interpretación de RMSE depende del rango de valores de la serie. RMSE responde mejor a grandes errores que MAE.

RMSLE: La raíz del error logarítmico cuadrático medio (RMSLE) es la raíz de los promedios cuadrados de las diferencias logarítmicas entre los valores observados y predichos. La interpretación de RMSLE depende del rango de valores de la serie. RMSLE responde menos a los valores atípicos que RMSE, y tiende a penalizar las subestimaciones un poco más que las sobreestimaciones.

r^2: r al cuadrado es el cuadrado del coeficiente de correlación de *Pearson* entre los valores observados y predichos. Esto varía de 0 a 1, donde un valor más alto indica un modelo de mayor calidad.

10.5 Condiciones en el pipeline y despliegue

En el siguiente fragmento de código se encuentra el componente de decisión, y si el mismo retorna “true” se prosigue con la creación del endpoint y el despliegue del modelo.

```
    model_eval_task = reg_model_eval_metrics(
    project = project,
    location = gcp_region,  # "us-central1",
    api_endpoint = api_endpoint,  # "us-central1-aiplatform.googleapis.com",
    thresholds_dict_str = '{"meanAbsolutePercentageError": 20}',
    model = training_task.outputs["model"]
)

    with dsl.Condition(
        model_eval_task.outputs["dep_decision"] == "true",
        name="deploy_decision",
    ):

        endpoint_task = gcc_aip.EndpointCreateOp(
            project=project,
            location=gcp_region,
            display_name=display_name,
        )

        gcc_aip.ModelDeployOp(
            model=training_task.outputs["model"],
            endpoint=endpoint_task.outputs["endpoint"],
            dedicated_resources_min_replica_count=1,
            dedicated_resources_max_replica_count=1,
            dedicated_resources_machine_type="n1-standard-4",
        )
```