# Vertex AI - Course Guide

**AutoML**

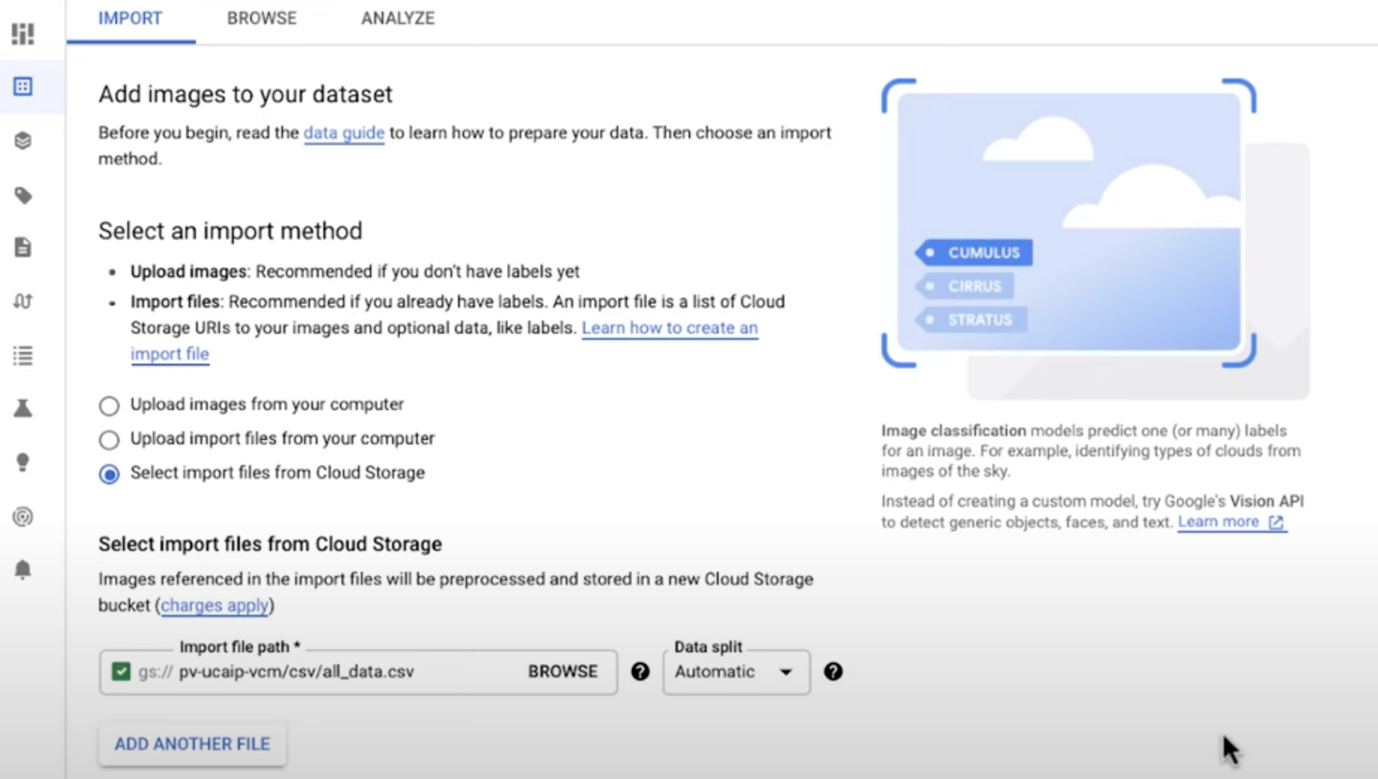
Cuando utilizar AutoML

* Tenemos un problema que involucra una de estas opciones.
  + Imagenes
    - Clasificacion (A nivel de imagen)
    - Detección de objectos (A nivel de sección)
    - Segmentation (A nivel de pixel)
  + Texto
    - Clasificación de texto
    - Extracción de entidades
    - Analisis de sentimiento
  + Datos tabulares (CSV)
    - Regresión
    - Clasificacion
    - Predicción (Forecasting)
  + Video
    - Clasificación
    - Reconocimiento de acciones
    - Seguimiento de objetos (Object tracking)
* El problema es básico.
* El equipo es pequeño o no tiene experiencia.
* Se búsca un prototico o prueba de concepto rápida.
* El modelo no necesitaria ser mejorado

## **Datasets**

En esta sección es donde gestionaremos el proceso de convertir los datos que tenemos en un dataset valido para su uso en Vertex AI. Este proceso puede ser realizado desde la CLI o desde la interfaz, a continuación, se muestran unas capturas de los pasos a realizar en la interfaz.

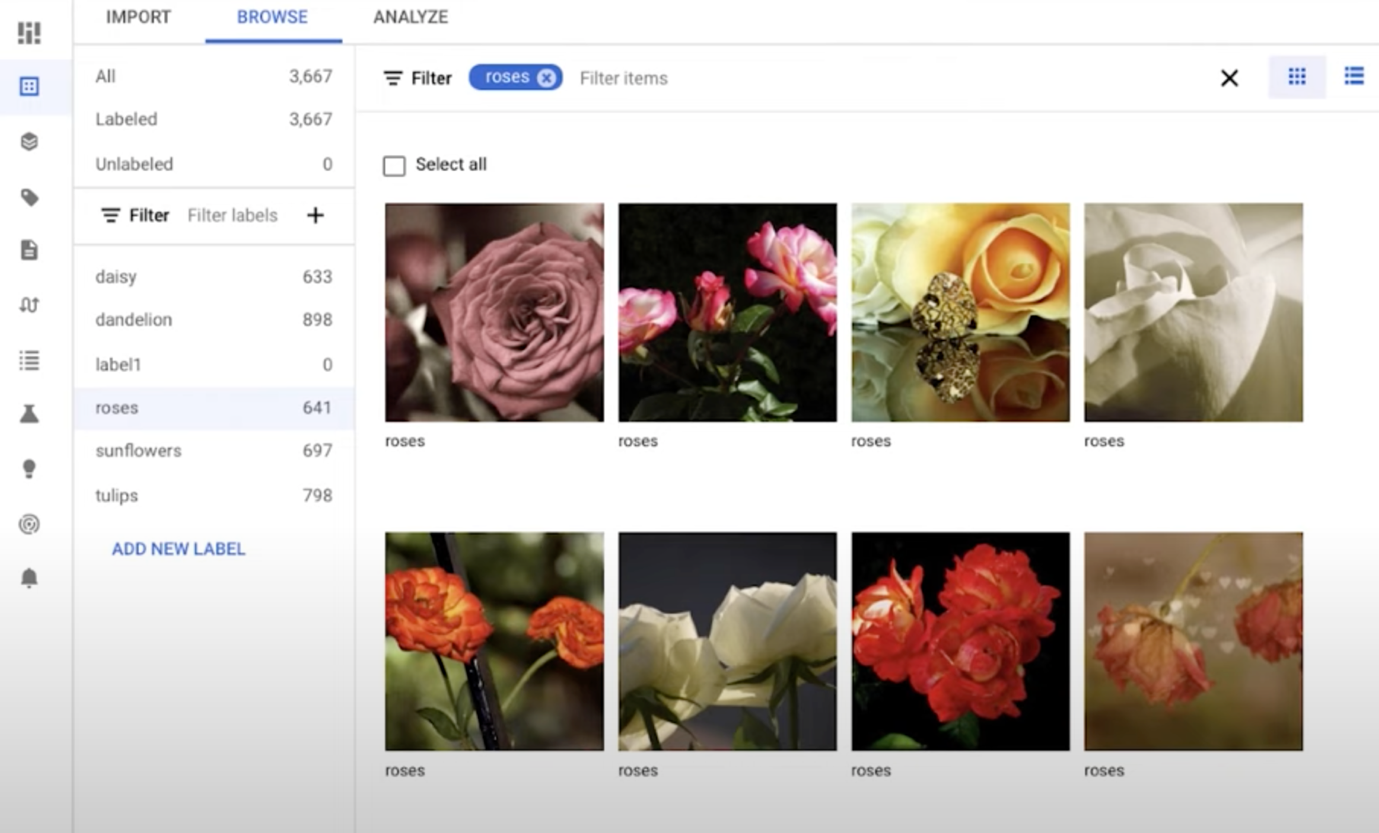
La sección de datasets tiene el siguiente aspecto.



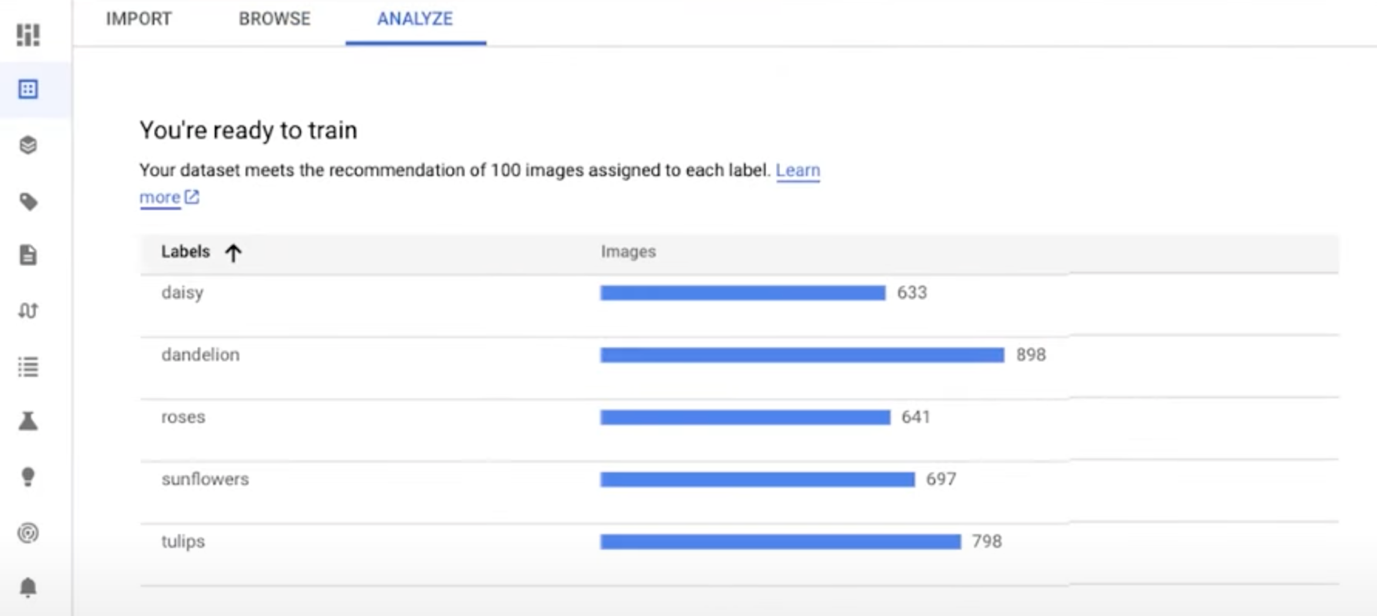
Podemos observar que tenemos diferentes opciones para importar datos. En este caso se trata de un dataset de imágenes.

* Podemos subir las imágenes directamente desde nuestro ordenador.
* Subir el *import file* que más adelante explicaremos.
* Seleccionar los datos desde en el Cloud Storage (Esto implica que previamente han sido subidos)

Una vez realizamos la importación tendremos una sección de Browse/Exploración donde podemos ver las clases, cuantas imágenes por clase e imágenes de la clase que tenemos seleccionada.



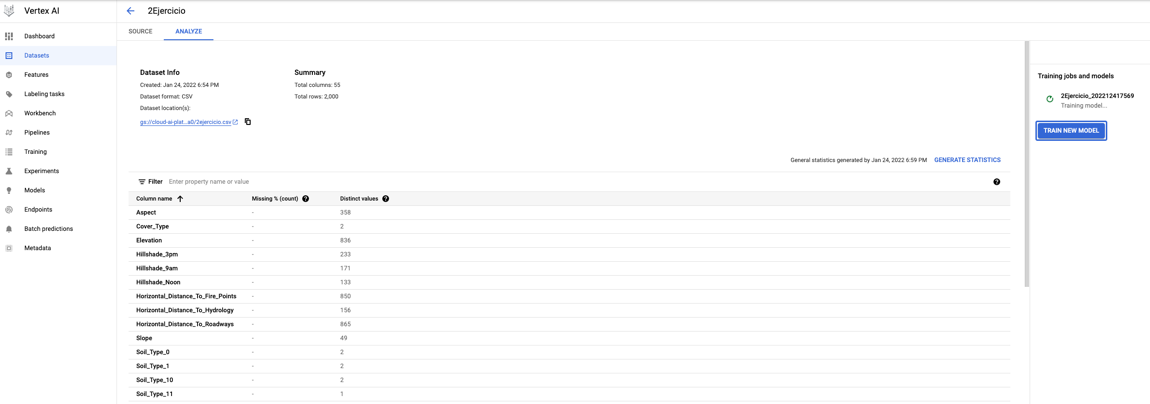
Por último, tenemos una sección de Analyze/Analisis donde podemos ver un breve resumen del dataset. Esta sección es particularmente interesante si previamente no hemos realizado un análisis por nuestra cuenta.



Aquí mostramos otro ejemplo del aspecto que tendría un dataset de tipo texto. Vemos que tenemos distintas clases, y una selección de ejemplos de dichas clases, muy similar al caso de imágenes.

## 

Una vez tenemos un dataset configurado en Verte AI, a la izquierda de la pantalla tendremos un botón que nos permitirá comenzar un nuevo entrenamiento.



# AutoML

## Ejemplo con datos tabulares

* La primera linea del CSV ha de ser la cabecera, estos son los nombres de las columnas
* Las columnas pueden contener caracteres alfanúmericos y la barra baja "\_" (no puede comenzar por)
* Cada CSV no puede pasar de 10GB, si pesa mas de 10GB lo puedes repartir en varios CSV hasta un máximo de 100GB
* El delimitador ha de ser la coma ",".
* Al menos 1000 filas para Tabular Data, 100 imagenes por clase para Vision AI
* El CSV se sube con la variable ha predecir incluida

No hace falta delimitar el schema del CSV (si las columnas son enteros, flotantes, strings..., etc), Vertex AI lo hace por ti. Se puede repartir los datos entre entrenamiento, validación y test de forma automática o manual

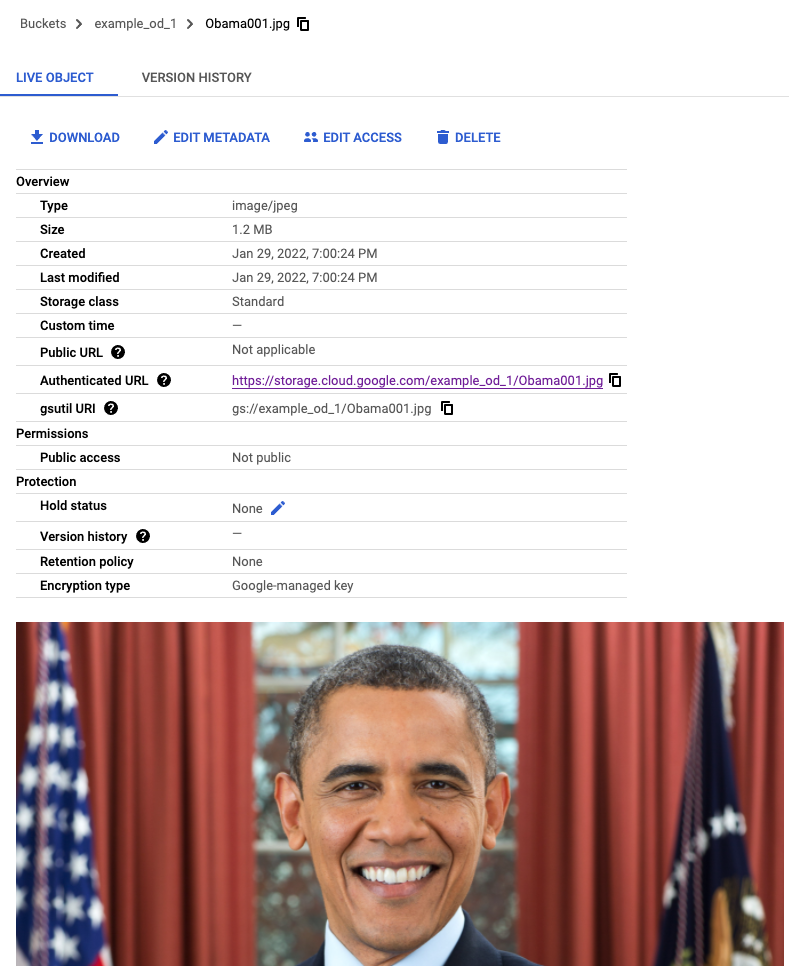
Vamos a entrenar un modelo con datos tabulados. Tenemos un ejemplo en el notebook llamado 2-AutoML que esta entrenando, usando la famosa librería Scikit-Learn.

El ejercicio consiste en preparar los datos que usamos en este mismo ejercicio, crear un dataset en Vertex AI y entrenar un modelo.

## Ejemplo con imágenes

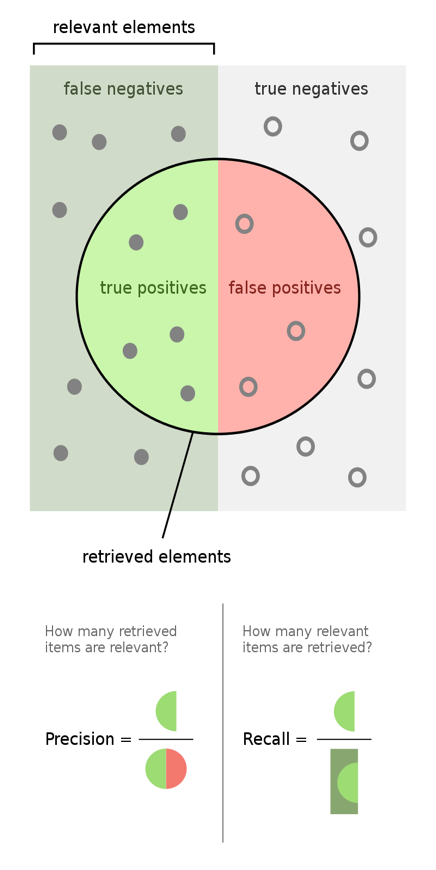
Para este ejercicio, vamos directamente a la documentación para aprender por nuestra cuenta como realizar este caso. La documentación la podemos encontrar en el [LINK](https://cloud.google.com/vertex-ai/docs/datasets/prepare-image#csv_2) los pasos son los siguiente

* Creamos un dataset subiendo las imágenes
* Etiquetamos las imágenes.



# Métricas

Como interpretar los resultados, dependiendo del dataset obtendremos unos tipos de resultados u otros. En el caso de datos tabulados por ejemplo tendremos las siguientes métricas.



**Precision =** TP / (FN + FP)

**Recall** = TP / (FN + TP)

**F1 Score** =

## Precision Recall Curve

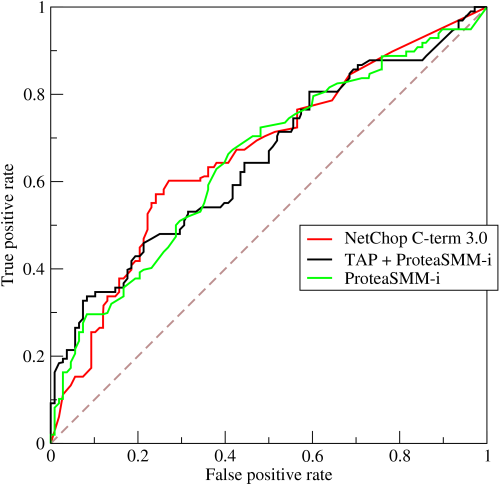
Para los problemas binarias de clasificación con clases desbalanceadas esta curva es muy útil para rápidamente de un vistazo observar si las dos clases son correctamente clasificadas o de lo contrario hay una tendencia a clasificar la clase mayoritaria como comúnmente ocurre en estas situaciones.

### ROC AUC

Para problemas de clasificación binarios tenemos una gran métrica entre TPR (True Positive Rate) y FPR (False Positive Rate)

**TPR =** TP / (TP + FN)

**FPR** = FP / (FP + TB)



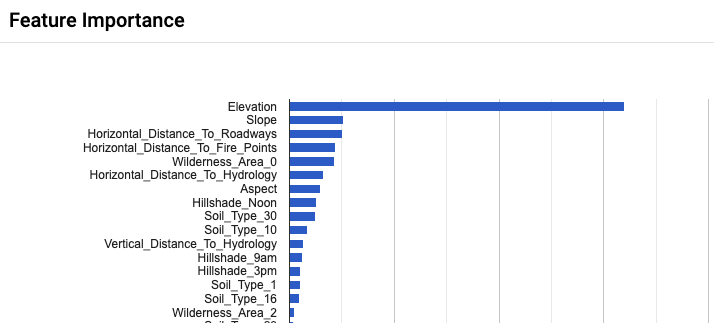
## Matriz de confusion



Tenemos los porcentajes de TP FP TN FN respectivamente en una matriz. Este caso es binario, para casos de más clases tendremos una matriz de NxN donde N es el número de clases.

## Feature Importance

Ranking de importancia de las variables para lograr la clasificación del modelo.



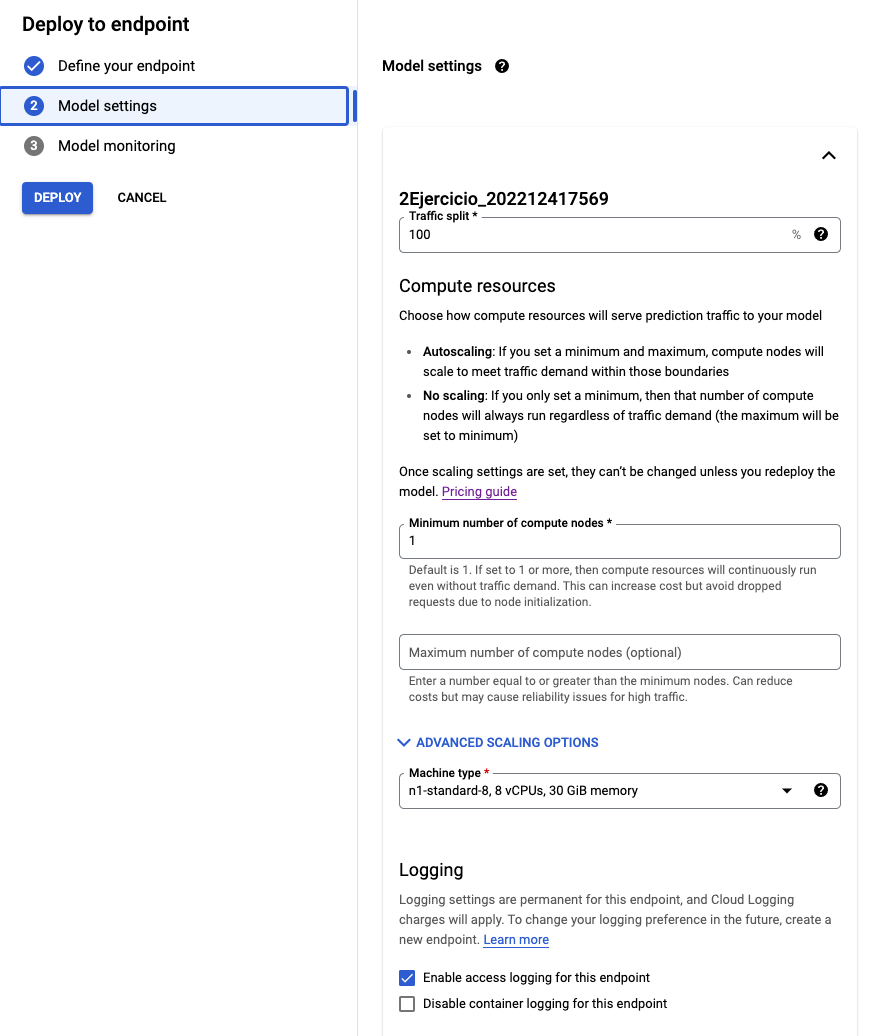
## IoU Intersection Over Union

Esta es una métrica exclusivamente de computer visión. Se trata de una métrica de similitud, por lo tanto, cuanto mayor es la cifra mayor similitud. Su valor esta acotado entre 0 y 1.

|  |  |
| --- | --- |
|  |  |

# Deploy

Una vez entrenado un modelo, en la sección de models seleccionamos el modelo el cual deseamos crear un endpoint.



**Traffic Split**, esta variable puede ser entendida en el siguiente escenario.

Cuando tenemos un modelo en un endpoint y queremos realizar una actualización de dicho modelo, sí cambiamos el modelo en el endpoint lógicamente va a devolver resultados que pueden variar con el modelo anterior. Para evitar cambios abruptos en los resultados podemos tener dos modelos en un mismo endpoint. Traffic Split nos permite desviar un porcentaje de las peticiones a un modelo e ir incrementado con el paso del tiempo, permitiendo una transición entre modelos mucho más suave, esto puede ser de especial importancia cuando es un cliente el que usa el endpoint.

**Minimum de compute nodes** indica como mínimo cuantos modos vamos a tener en funcionamiento en todo momento, con el consecuente gasto económico.

**Maximum compute nodes** indica cuantos nodos como máximo podemos tener, esto es especialmente útil para situaciones donde queremos autoscaling.

Cuando tenemos el endpoint creado, si pulsamos en el veremos la información básica que lo define, si nos fijamos en la URL tendremos un número, como en la siguiente foto.

# 

Una vez desplegado el endpoint, podemos realizar queries desde la interfaz o desde código Python.

Si necesitamos procesar datos en forma de batch tenemos la opción Batch Processing desde la interfaz.

# Custom model (Avanzado)

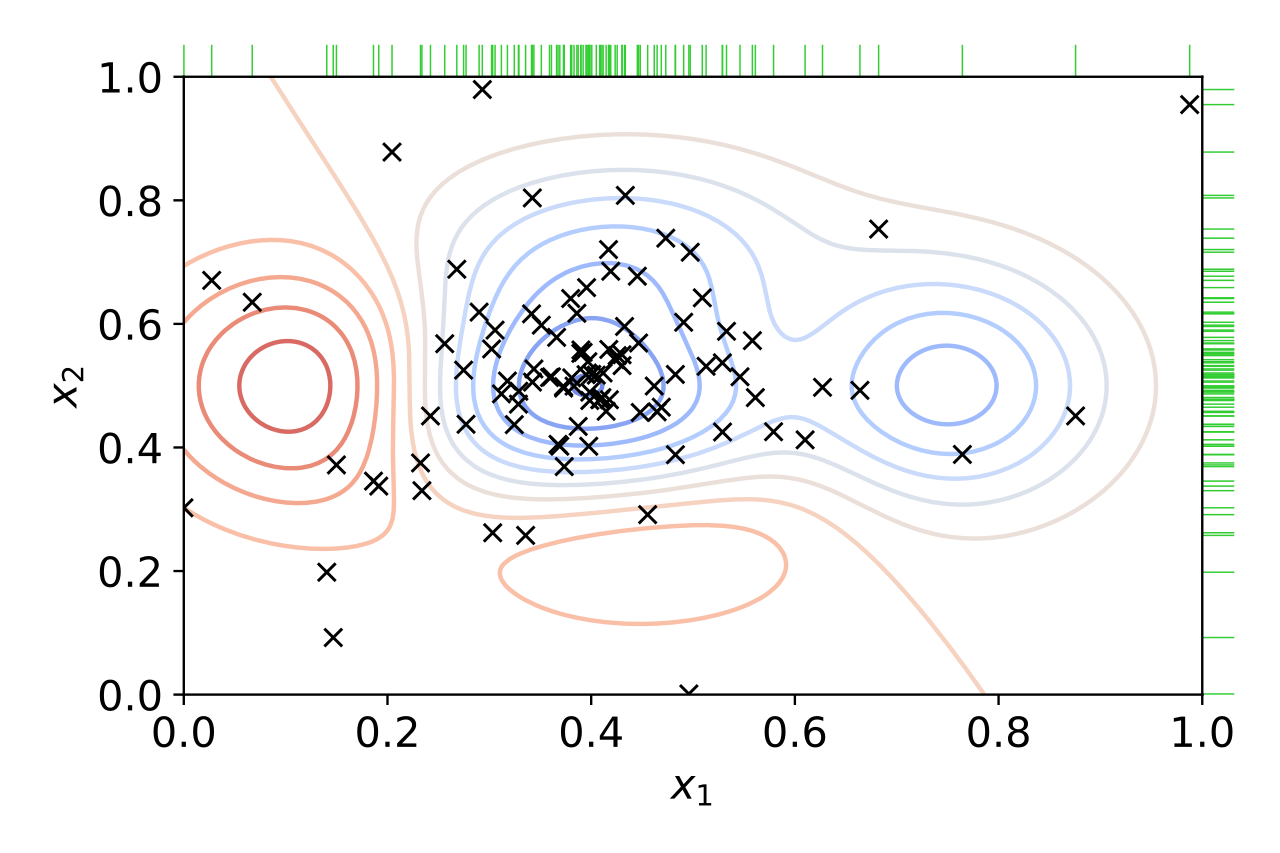
Cuando utilizar emplear un custom model

* Tenemos un problema que involucra más de un problema elemental (ie. Imágenes con datos tabulados)
* El problema no es básico.
* Necesitamos un control particular sobre el modelo, como el framework (TensorFlow, PyTorch, Scikit-Learn)
* El equipo que se encarga del modelo no es pequeño.
* El equipo que se encarga del modelo tiene conocimientos.
* Hay tiempo para realizar ajustes y mantenimientos.
* El modelo ha de poder ser mejorado.

## ¿Qué necesitamos?

* Training Package
  + Pre-built (Tensorflow, Scikit-learn, PyTorch)
  + Container o Artifact Registry (Custom Container)
* Model Artifacts
  + Se deben de guardar en un bucket del Storage
* Hyperparameter Tuning
  + Cambiar parametros de entrenamiento, el sistema se encarga de búscar los mejores parametros que den el mejor resultado
* Compute resources
  + Para entrenar el modelo debemos asignar un máquina.
* Endpoint
  + Deploy pre-built (Tensorflow, Scikit-Learn, PyTorch)
  + Deploy Custom

**HP Tunning (Avanzado)**

****

[**GCP HP Tunning**](https://cloud.google.com/ai-platform/training/docs/hyperparameter-tuning-overview)

[**Third Party HP Tunning**](https://optuna.org/)

# Edge Computing

# [Coral.ai](https://coral.ai/)

# [Jetson Nano](https://developer.nvidia.com/embedded/jetson-nano-developer-kit)