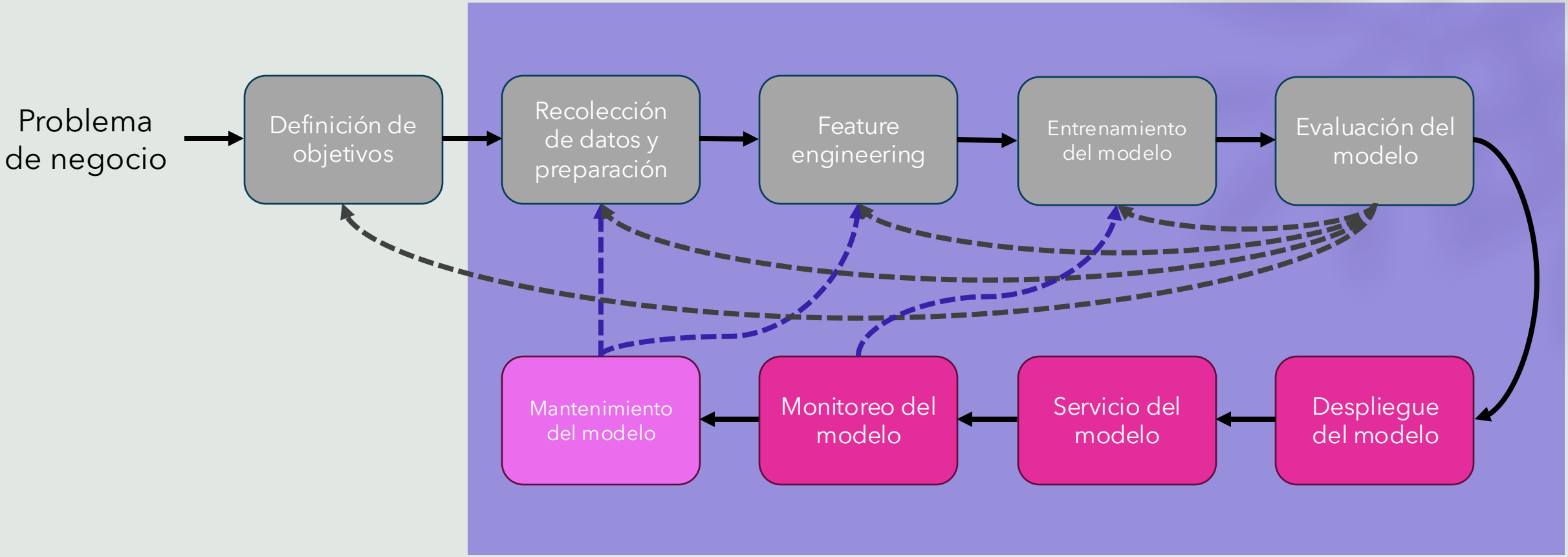
Desarrollo de modelos

Operaciones de Aprendizaje Automático I - CEIA - FIUBA

Dr. Ing. Facundo Adrián Lucianna

Repaso de la clase anterior

Ciclo de vida de un proyecto de Aprendizaje Automático



Ciclo de vida de un proyecto de Aprendizaje Automático

Data Engineer

El Data Engineer es responsable de la **preparación y limpieza de los datos**, la creación de **pipelines de datos** y la integración de diferentes fuentes de datos.
Su trabajo también incluye la selección de las herramientas y tecnologías adecuadas para la gestión de datos y la implementación de soluciones de **almacenamiento y procesamiento de datos escalables**.

Data Scientist

El Data Scientist se encarga de **definir y crear modelos de machine learning** que permitan hacer predicciones a partir de los datos. Su trabajo implica seleccionar los algoritmos adecuados, entrenar los modelos y optimizar su rendimiento.
Los Data Scientists también pueden participar en la identificación de variables relevantes y en la **exploración de los datos para encontrar patrones y tendencias.**

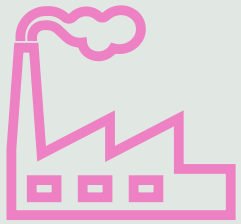
Data Analyst

El Data Analyst trabaja con datos para descubrir patrones y tendencias que puedan ser útiles para la **toma de decisiones empresariales**. Su trabajo implica realizar análisis estadísticos y visualizaciones de datos para **entenderlos mejor** y hacer recomendaciones sobre cómo pueden utilizarse **para mejorar el negocio.**

Machine Learning Engineer

El Machine Learning Engineer es responsable de llevar los modelos de machine learning a **producción y asegurarse de que estén funcionando correctamente.**
Su trabajo implica seleccionar la infraestructura adecuada para el despliegue de los modelos, integrar los modelos con otras aplicaciones y sistemas, y **supervisar el rendimiento de los modelos.**

Consideraciones para aplicaciones en industria

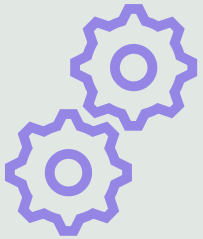


Producción

Para que el modelo pueda entregar valor al negocio debe estar productivo.

Usabilidad

Un modelo con 70% de exactitud en producción produce mucho más valor que uno con 100% de exactitud que no se puede usar.



Dependencia

Los modelos en producción requieren mantenimiento para prevenir el *desvío* en los datos o en el target.

Escalabilidad

El proceso debe ser implementado para que otras personas del equipo lo entiendan, debe ser transparente y replicable.

Pipelines/flujos de trabajo reproducibles dentro de ML

¿Qué es un pipeline?

Un pipeline de datos es una construcción lógica que representa un proceso dividido en fases.

Los pipelines de datos se caracterizan por definir el conjunto de pasos o fases y las tecnologías involucradas en un proceso de movimiento o procesamiento de datos.

Esto nos permite encapsular el código, hacerlo más legible, más ordenado, estandarizar y automatizar los procesos, entre otros beneficios.

Los pipelines de Machine Learning permiten a los equipos de datos iterar rápidamente sobre diferentes modelos y ajustes y mejorar continuamente el rendimiento del modelo.

Los pipelines están conformados por **componentes** y por **artefactos** (artifacts).

Machine Learning Operations (MLOps)

MLOps, o **Machine Learning Operations**, es un término que se refiere a las prácticas y herramientas utilizadas para gestionar y desplegar modelos de aprendizaje automático a gran escala en producción de manera efectiva y eficiente.

MLOps es una **disciplina emergente** que se enfoca en la gestión de los modelos de machine learning en producción y busca establecer procesos y herramientas para garantizar que los modelos de machine learning sean precisos, escalables y adaptables a diferentes situaciones.

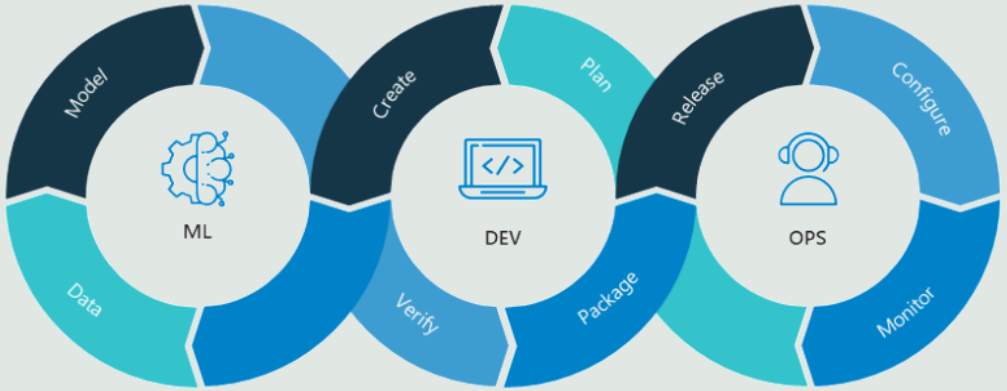


Imagen obtenida de Neal Analytics

Niveles de MLOps

Los 3 niveles de MLOps

Frecuentemente dentro de la industria se pueden encontrar diferenciados tres niveles de MLOps. Estos niveles se diferencian en cuanto a la cantidad de herramientas/prácticas de MLOps que incluyen dentro de su funcionamiento.

- Nivel 0
- Nivel 1
- Nivel 2

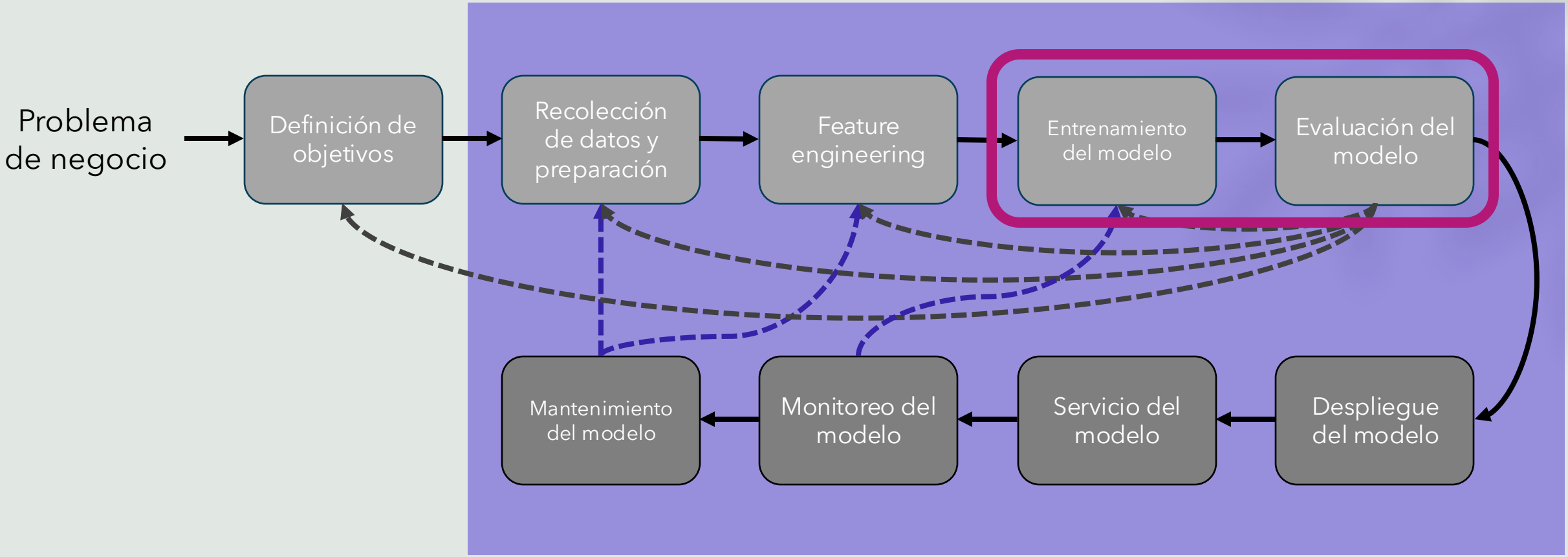
Buenas prácticas de programación

Para comenzar a trabajar pensando en un modelo de aprendizaje automático que será productivo, y visto por otras personas, el código debe cumplir con ciertos estándares de buenas prácticas de programación.

Cuando nuestro código va a ser potencialmente usado en producción, debe cumplir con ser **legible**, **simple** y **conciso**.

Desarrollo de modelos

Desarrollo de modelos en producción



Desarrollo de modelos en producción

En AMq1 vimos diferentes modelos, y estudiamos el proceso para entrenarlo y evaluarlo. Ahora volveremos un poco sobre esto, pero ahora estamos pensando que este modelo no solo terminará en un notebook, sino que llegará a producción.

El desarrollo de modelos es un proceso iterativo.

Cada iteración, uno debe comparar el rendimiento con iteraciones anteriores.

Seleccionar el tipo de modelo

Seleccionar el tipo de modelo

Si uno tuviese tiempo infinito y recursos infinitos, el camino más racional es evaluar todas las posibles soluciones y ver cuál es la mejor al problema que queremos resolver. Pero no tenemos el tiempo y recursos infinitos, sino que tenemos un presupuesto y un tiempo para desarrollar. Por lo que hay que ser estratégico.

Un punto importante es enfocarse en modelos apropiados para el problema, y para eso usamos lo que vimos en AMq1.

Otro punto importante, en general dejado de lado, es **la implementación del modelo**, el cual no solo debemos mirar la métrica de entrenamiento, sino cuanta data, procesamiento y tiempo llevaría entrenarlo. También cuál sería su latencia de inferencia e interoperabilidad.

Seleccionar el tipo de modelo

Ocho consejos a la hora de seleccionar un modelo:

- **Evitar la trampa del estado del arte**: En general se asume que estos modelos van a ser la mejor solución para el problema, dado que todos están usando ese modelo. No solo eso, a nivel de negocio, "*vende*" más usar un modelo moderno.

Que un modelo sea el que tenga un rendimiento mejor que otros en un dataset estático, no significa que el modelo será lo suficientemente rápido o lo suficientemente barato.

Inclusive tampoco significará que el modelo será mejor que otros modelos con tus datos.

Seleccionar el tipo de modelo

Ocho consejos a la hora de seleccionar un modelo:

- **Comienza con el modelo más simple**: Dentro del Zen de Python, encontramos:

Simple es mejor que complejo

Y este principio es aplicable al desarrollo de modelos. Simplicidad sirve de tres propósitos:

1. **Modelos sencillos son más fáciles de desplegar en producción**. Desplegarlo temprano permite validar un pipeline de predicción es consistente con tus datos de entrenamiento.
2. Comenzando algo más sencillo y agregando complejidad paso a paso lo hace **más fácil de entender** y de depurar.
3. Los modelos más simples sirven como **baseline**.

Seleccionar el tipo de modelo

Ocho consejos a la hora de seleccionar un modelo:

- **Explicabilidad**: ¿Las predicciones del modelo requieren explicación para una audiencia no técnica? Los modelos de aprendizaje automático más precisos son los llamados **cajas negras**. Cometen muy pocos errores de predicción, pero puede ser difícil de entender, y aún más difícil de explicar, por qué un modelo hizo una predicción específica. Ejemplos de tales modelos son las redes neuronales profundas y los modelos de conjuntos.

  Por el contrario, los algoritmos de aprendizaje de árboles de decisión, kNN y regresión lineal no siempre son los más precisos. Sin embargo, sus predicciones son fáciles de interpretar por parte de un usuario no experto.

Seleccionar el tipo de modelo

Ocho consejos a la hora de seleccionar un modelo:

Velocidades de entrenamiento y predicción: ¿Cuánto tiempo se le permite usar a un algoritmo de aprendizaje para construir un modelo y con qué frecuencia será necesario volver a entrenar el modelo con datos actualizados? Si el entrenamiento dura dos días y necesita volver a entrenar su modelo cada 4 horas, el modelo nunca estará actualizado. Las bibliotecas especializadas contienen implementaciones muy eficientes de algunos algoritmos. Es posible que prefiera investigar en línea para encontrar dichas bibliotecas.

Para produccion se suele usar bibliotecas especializadas que tienen modelos optimizados para entrenar.

¿Qué tan rápido debe ser el modelo al generar predicciones? ¿Se utilizará el modelo en un entorno de producción donde se requiere un rendimiento muy alto? Modelos como SVM y modelos de regresión lineal y logística, y redes neuronales no muy profundas, son extremadamente rápidos en el momento de la predicción. Otros, como kNN, ensambles y redes neuronales profundas o recurrentes, son más lentos.

Si estamos en un contexto de prediccion online, es importante analizar modelos que sean rapidos.

Seleccionar el tipo de modelo

Ocho consejos a la hora de seleccionar un modelo:

Evita sesgos humanos en la selección: Todos tenemos preferencias y gustos sobre ciertos tipos de modelos.

Si un tipo de modelo nos gusta más, vamos a destinar más tiempo jugando con sus hiperparámetros que otros. Los que nos puede dar la errónea idea de que un modelo es mejor que otro.

Cuando se compara dos arquitecturas diferentes, es importante compararlo bajo configuraciones similares. Si se ejecutan 100 experimentos en una, no sería justo si se corren un par en otra arquitectura.

Dado que el rendimiento del modelo depende fuertemente del contexto, es prácticamente **imposible poder asegurar que una arquitectura es mejor que otra**.

Seleccionar el tipo de modelo

Ocho consejos a la hora de seleccionar un modelo:

→ **Evalúa rendimiento de hoy versus rendimiento posterior**: El mejor modelo hoy no necesariamente lo será en dos meses.

Por ejemplo, un modelo de árbol puede al inicio rendir mejor, pero más adelante con mucha más data acumulada, una red neuronal lo supere.

Una forma de estimar que tanto puede cambiar el rendimiento en el futuro con más datos es usan **curvas de aprendizaje**.

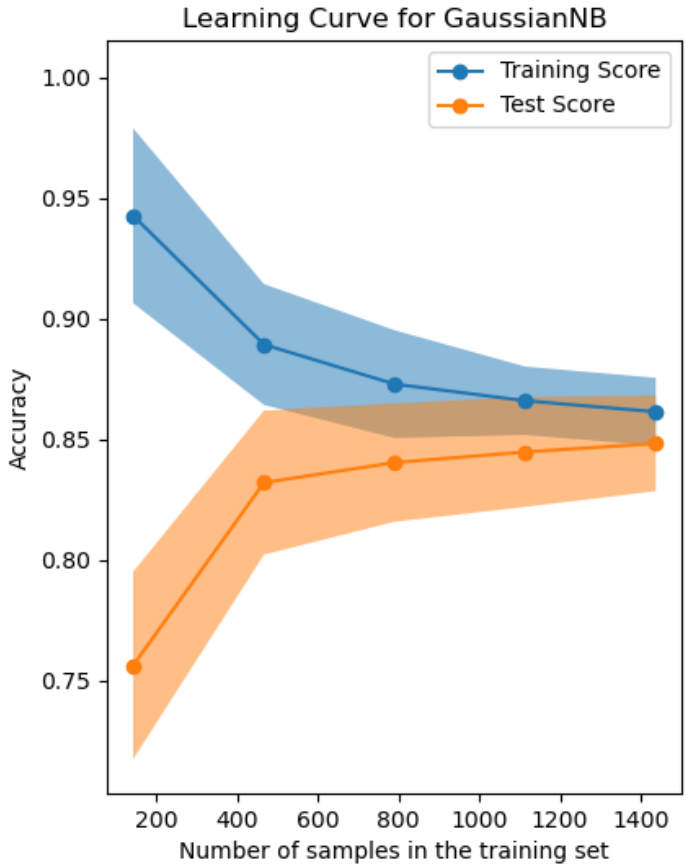
Seleccionar el tipo de modelo

Ocho consejos

Evalúa rendimi no necesariamente

Por ejemplo, un on mucha más data acumu

Una forma de e s datos es usan **curvas de**



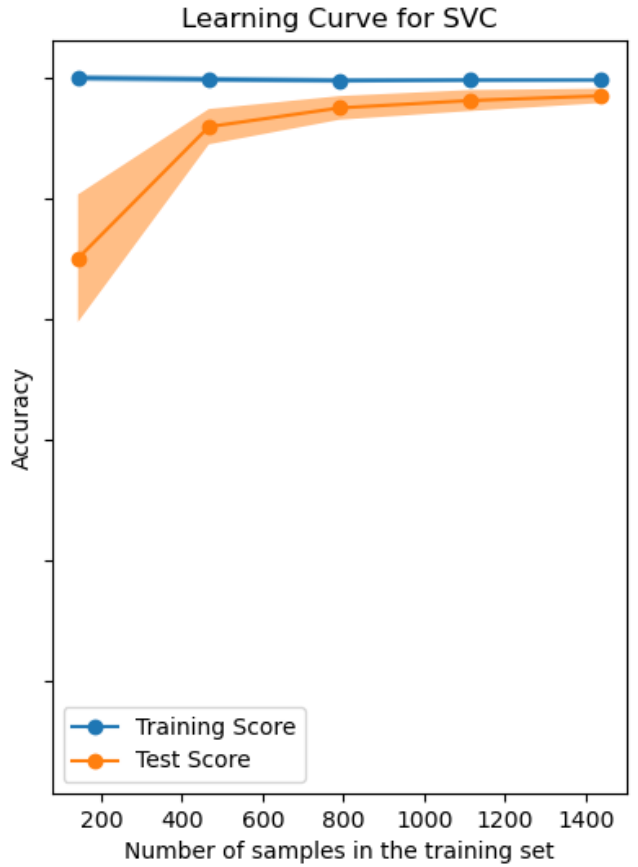


Imagen obtenida de Scikit-Learn

Seleccionar el tipo de modelo

Ocho consejos a la hora de seleccionar un modelo:

Evalúa rendimiento de hoy versus rendimiento posterior: El mejor modelo hoy no necesariamente lo será en dos meses.

Por ejemplo, un modelo de árbol puede al inicio rendir mejor, pero más adelante con mucha más data acumulada, una red neuronal lo supere.

Una forma de estimar que tanto puede cambiar el rendimiento en el futuro con más datos es usan **curvas de aprendizaje**.

Cuando evalúas modelos, es importante tener en cuenta su potencial de mejoras en el futuro y que tan fácil es lograr estas mejoras.

Seleccionar el tipo de modelo

Ocho consejos a la hora de seleccionar un modelo:

→ **Evalúa trade-offs**: Hay muchas consideraciones que se deben tomas a la hora de elegir un modelo.

- Una clásica es trade-off de falsos positivos y falsos negativos que vimos en AMq1.
- Requerimientos de cómputos versus exactitud. Un modelo más complejo puede llevar a una alta exactitud, pero requiere una maquina más poderosa para lograr inferencias con buena latencia.
- Rendimiento versus interpretabilidad. Un modelo más complejo puede tener mejores resultados, pero sus resultados son menos interpretables.

Seleccionar el tipo de modelo

Ocho consejos a la hora de seleccionar un modelo:

→ **Entiende las suposiciones del modelo**: Veamos algunas suposiciones,

- Suposiciones de predicción. Cada modelo apunta a predecir una salida Y de una entrada X, hace la suposición de que Y está basado en X.
- IID: En general, lo modelos asumen que los datos de entrenamientos son obtenidos independiente de una misma distribución. Para series de tiempo esto no se cumple, ya que es importante el orden en que se procesan los datos
- Suavidad: Cada método supervisado asume que existe un set de funciones que puede transformar la entrada en salida, de tal forma que entradas similares nos da salidas similares.
- Tratabilidad: Todo modelo generativo que una entrada X y una representación latente Z (por ejemplo, componentes principales) hace la suposición de que el cálculo de la probabilidad es tratable.

Seleccionar el tipo de modelo

Ocho consejos a la hora de seleccionar un modelo:

Entiende las suposiciones del modelo: Veamos algunas suposiciones,

- Bordes. Un clasificador lineal asume que la frontera de decisión es lineal. Ejemplo, SVM
- Independencia condicional: Un clasificador Bayes ingenuo asume que los atributos son independientes entre ellos para una clase dada.
- Distribución normal: Muchos métodos estadísticos asumen que los datos están distribuidos de forma normal.

Las 4 fases del desarrollo de modelos

Las 4 fases del desarrollo de modelos

La estrategia que uno debe llevar a la hora de adoptar ML para un problema específico dependerá de en qué fase nos encontremos.

- **Fase 1: Antes de Machine Learning**

Si es la primera vez que se quiere intentar realizar un tipo de predicción, comienza con una solución que no involucra un modelo. Lo primero puede ser una simple heurística.

Por ejemplo, el feed de noticias de Facebook, cuando comenzó en 2006, los post se mostraban de forma cronológica. Recien en 2011, Facebook comenzó a mostrar actualizaciones en base a intereses.

Las 4 fases del desarrollo de modelos

La estrategia que u specífico dependerá de en

- **Fase 1: Antes d**

Si es la primera ve enza con una solución que no in stica.

Por ejemplo, el fee ost se mostraban de forma cronológ aciones en base a intereses.

Las 4 fases del desarrollo de modelos

La estrategia que uno debe llevar a la hora de adoptar ML para un problema específico dependerá de en qué fase nos encontremos.

- **Fase 2: Modelo de Machine Learning más sencillo posible**

Para el primer modelo, empieza con el más básico posible que le brinde visibilidad de su funcionamiento para validar la utilidad del enfoque del problema y los datos. Casi cualquier modelo de AMq1.

También son más fáciles de implementar y desplegar, lo que permite crear rápidamente un marco desde la ingeniería de datos hasta el desarrollo.

Las 4 fases del desarrollo de modelos

La estrategia que uno debe llevar a la hora de adoptar ML para un problema específico dependerá de en qué fase nos encontremos.

- **Fase 3: Optimizar el modelo sencillo**

Una vez que está productivo el framework del modelo, nos podemos enfocar en optimizar el modelo simple modificando la función objetivo, búsqueda de hiper-parámetros, feature engineering, más datos o ensambles.

Las 4 fases del desarrollo de modelos

La estrategia que uno debe llevar a la hora de adoptar ML para un problema específico dependerá de en qué fase nos encontremos.

- **Fase 4: Modelos complejos**

Una vez que se considera que se llegó al límite de los modelos más simples, y el uso de caso lo demanda, se puede experimentar con modelos más complejos.

Además, va a ser importante experimentar que tan rápido decae un modelo en producción (es decir, cada cuanto hay que re-entrenarlo) para que se construya la infraestructura de entrenamiento.

Depurando modelos

Depurando modelos

Un gran problema de los modelos es que cuando fallan, **fallan silenciosamente**. Por lo que la tediosa tarea de depurar, en Aprendizaje Automático es mucho más tedioso.

- Un problema típico es: el código compila, la función de pérdida disminuye como debería, se llaman las funciones correctas, las predicciones se hacen… pero las predicciones están **mal**. Lo peor es que los usuarios no notan error y utilizan las predicciones como si la aplicación estuviera funcionando como debería.
- Una vez que se encuentra el error, es sumamente lento validar si el error se corrigió.
- Recordemos que el trabajo de implementar un modelo es un trabajo inter-disciplinario (datos, etiquetas, features, algoritmo de ML, infraestructura, etc.). Estos componentes pueden ser de equipos diferentes. Una falla puede ser de alguno de estos componentes o de la interacción entre ellos.

Depurando modelos

Veamos algunas causas típicas de fallas:

- **Restricciones teóricas**: Cada modelo viene con sus propias suposiciones y los datos no cumplieron estas suposiciones. Es importante poner alarmas en las metricas de entrenamiento.
- **Mala implementación del modelo**: El modelo podría ajustarse bien a los datos, pero los errores están en la implementación del modelo. Ejemplo, en Pytorch no poner el modo correcto en entrenamiento o evaluacion.
- **Elección pobre de hiperparámetros**: El modelo se adapta perfectamente a los datos y la implementación es correcta, pero un conjunto deficiente de hiperparámetros puede hacer que el modelo sea inútil.
- **Problemas de data**: Hay muchas cosas que podrían salir mal en la recopilación y el preprocesamiento de datos y que podrían causar que el modelo tenga un rendimiento deficiente.
- **Mala elección de features**: Hay muchísimas opciones de features, muchas features pueden ocasionar overfitting o causar **data leakage**.

Depurando modelos

No hay métodos precisos para depurar, pero podemos ver algunas estrategias para evitar bugs:

- Comienza simple y agrega gradualmente más componentes.
- Una vez que tienes una implementación simple, intenta sobre-entrenarlo con unos pocos datos, y observa si la metrica es casi perfecta. Si el modelo no puede overfittear significará que hay algo raro con la implementación.
- Usa una semilla para la aleatoriedad. Esto permite consistencia entre ejecuciones y facilita la reproducción de errores.

Entrenamiento distribuido

Entrenamiento distribuido

A pesar de que se insiste en el uso de modelos sencillos, hay una realidad que ni podemos escapar. Cada vez hay más datos, y cada vez se necesitan modelos más grandes, lo que nos llevan a casos más intensos de recursos.

El primer problema que vemos en estos casos es que los datos no entran en memoria. Casi todos los problemas de aprendizaje profundo o de visión por computadora entran en esa categoría.

Cuando los datos no entran en memoria, o no contamos con el procesamiento suficiente, debemos pasar a técnicas de paralelismo.

Entrenamiento distribuido

Paralelismo de datos

La forma más común de paralelismo es la de datos. Se reparte los datos en muchas maquinas, se entrena el modelo en cada maquina y se acumulan los gradientes.

El desafío es cómo acumular gradientes de diferentes máquinas de manera precisa y efectiva.

Hay dos formas de realizar la acumulación de gradientes:

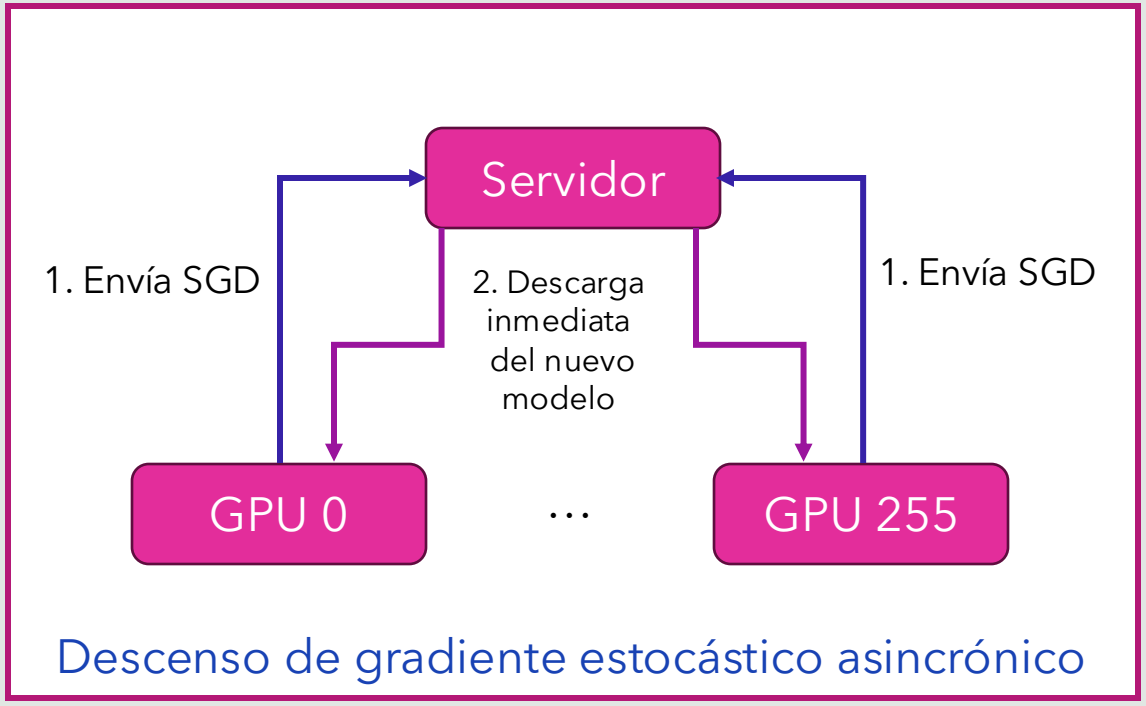
- Descenso de gradiente estocástico sincrónico
- Descenso de gradiente estocástico asincrónico

Ejemplos: Dataset y Dataloads de Pytorch, Polars

Entrenamiento distribuido

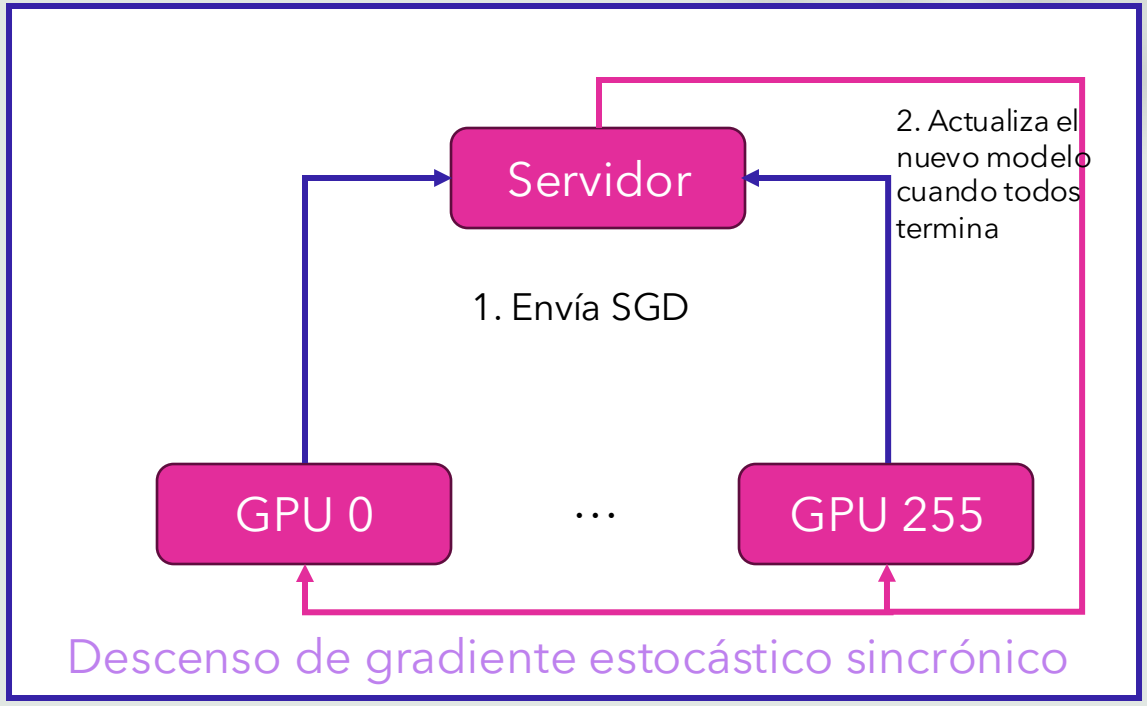
Paralelismo de datos

En este caso, el servidor no espera que terminen todos, actualiza los pesos cada vez que recibe los gradientes de algun GPU



El problema aca es que una GPU puede estar atrasado por lo cual el servidor va a actualizar los pesos con gradientes viejos

Cada GPU envia el gradiente y cuando llegan todos, el servidor actualiza los pesos usando la suma de los gradientes. Luego, el servidor envia la informacion a cada GPU



Un problema es que actualizar un paso puede demorar mucho al ser todo sincronico

Entrenamiento distribuido

Paralelismo de datos

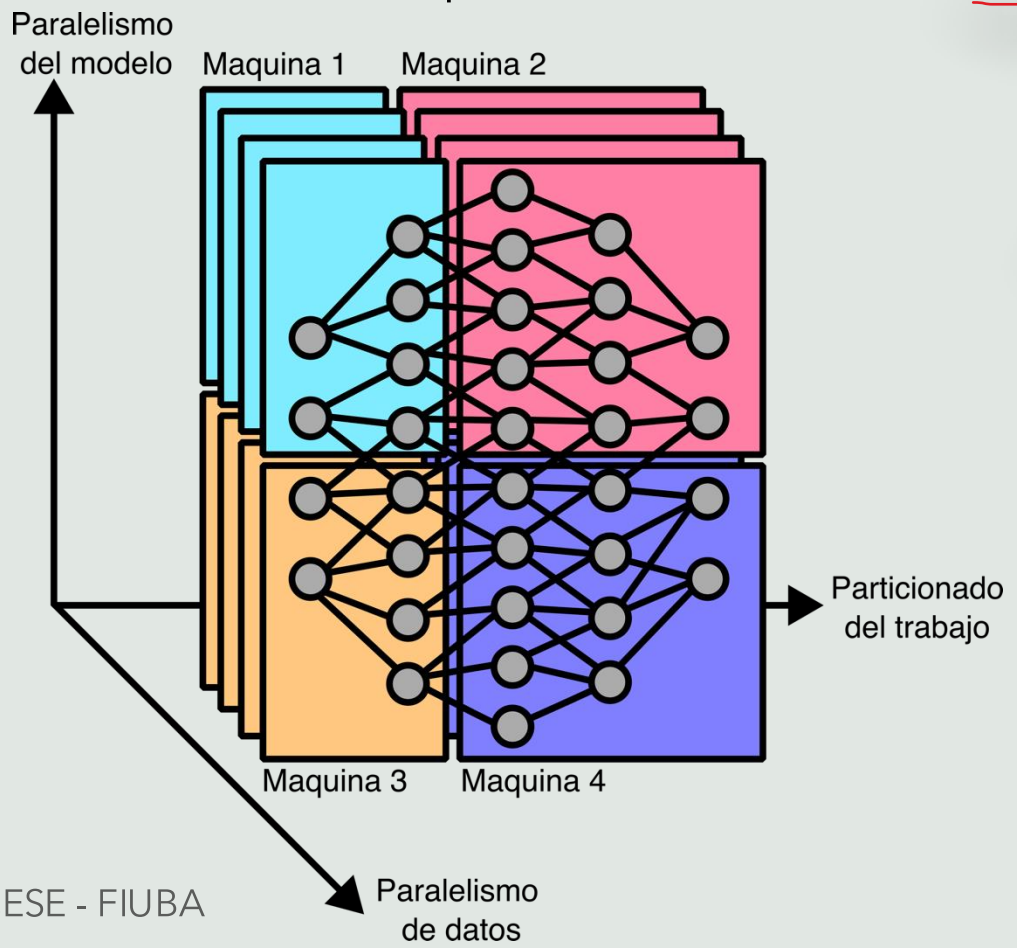
El descenso de gradiente estocástico asincrónico converge, pero requiere muchos más pasos que el descenso de gradiente estocástico sincrónico. Sin embargo, en la práctica cuando el número de parámetros es grande, las actualizaciones de gradiente solo modifican pequeñas fracciones de los parámetros, y es poco probable que dos actualizaciones de diferentes maquinas modifiquen el mismo parámetro.

En esos casos es preferible usar asincrónico, ya que se tiene el beneficio de la velocidad y no se pierde nada.

Entrenamiento distribuido

Paralelismo del modelo

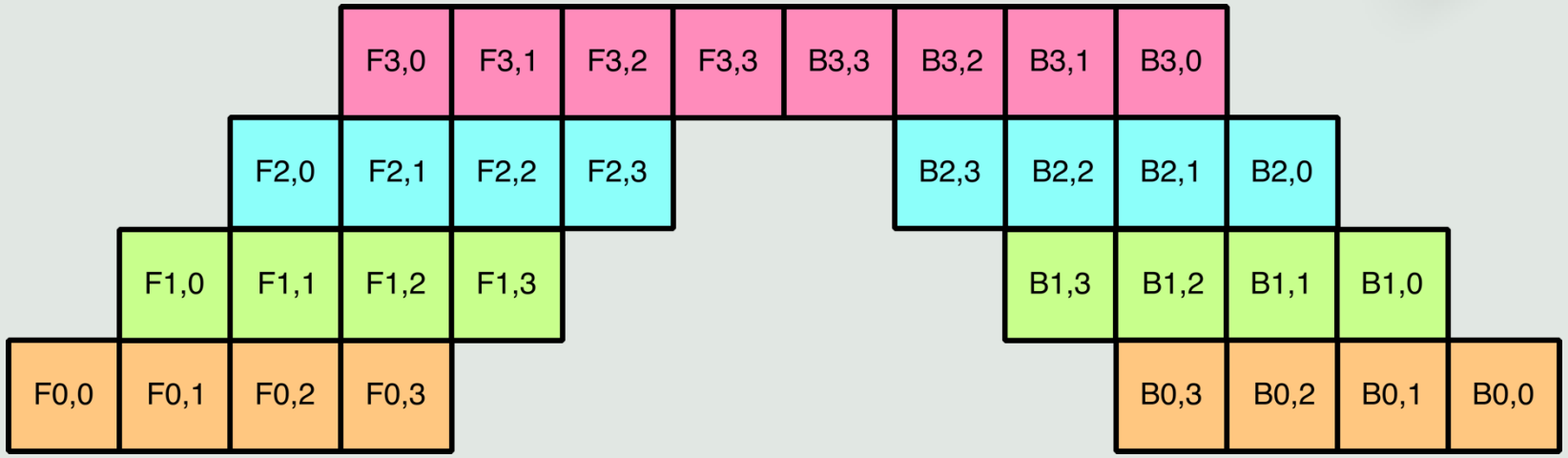
Paralelismo del modelo es cuando diferentes componentes del modelo se encuentran en diferentes maquinas.



Entrenamiento distribuido

Paralelismo del pipeline

La idea clave en este paralelismo es dividir el cálculo en varias partes. Cuando la maquina 1 termina, para el resultado a la maquina 2, y continua con la segunda parte del cálculo y así sucesivamente. La máquina 2 puede ejecutar la primera parte, mientras que la maquina 1 ejecuta su segunda parte del cálculo.



Métodos de evaluación

Métodos de evaluación

En AMq1 vimos como evaluar modelos usando sus métricas de rendimiento. Pero en producción tenemos más evaluaciones que nos importan, ya que nuestros modelos deben ser robustos, justos, calibrados y en general tener sentido.

Veamos algunos métodos de evaluación que nos permitan caracterizar un modelo.

Métodos de evaluación

Test de perturbación

Idealmente, las observaciones utilizadas para desarrollar el modelo deberían ser lo más similar a las que se encontrará cuando esté funcionando, pero hay casos que no es posible.

Para tener una idea de cómo será el rendimiento del modelo con dato ruidosos, se puede realizar pequeños cambios en el set de testing para ver como estas perturbaciones afectan el modelo.

Cuando más sensible a perturbaciones es el modelo, más difícil será de mantener, ya que, si el comportamiento de los usuarios varía un poco, el rendimiento puede cambiar significativamente.

Métodos de evaluación

Test de invarianza

Ciertos cambios en las entradas del modelo no deberían generar cambios en la salida del modelo. Principalmente en modelos con datos demográficos. Si esto ocurre, es que hay sesgos en el modelo, el cual puede volver inutilizable al mismo, sin importar que tan bueno sea.

Para evitar estos sesgos, una solución es mantener las entradas iguales, pero cambiar la información sensible para ver si las salidas cambian.

Sensibles (raza, religion, etc)

Mejor aún, en primer lugar, se debería excluir la información confidencial de las funciones utilizadas para entrenar el modelo.

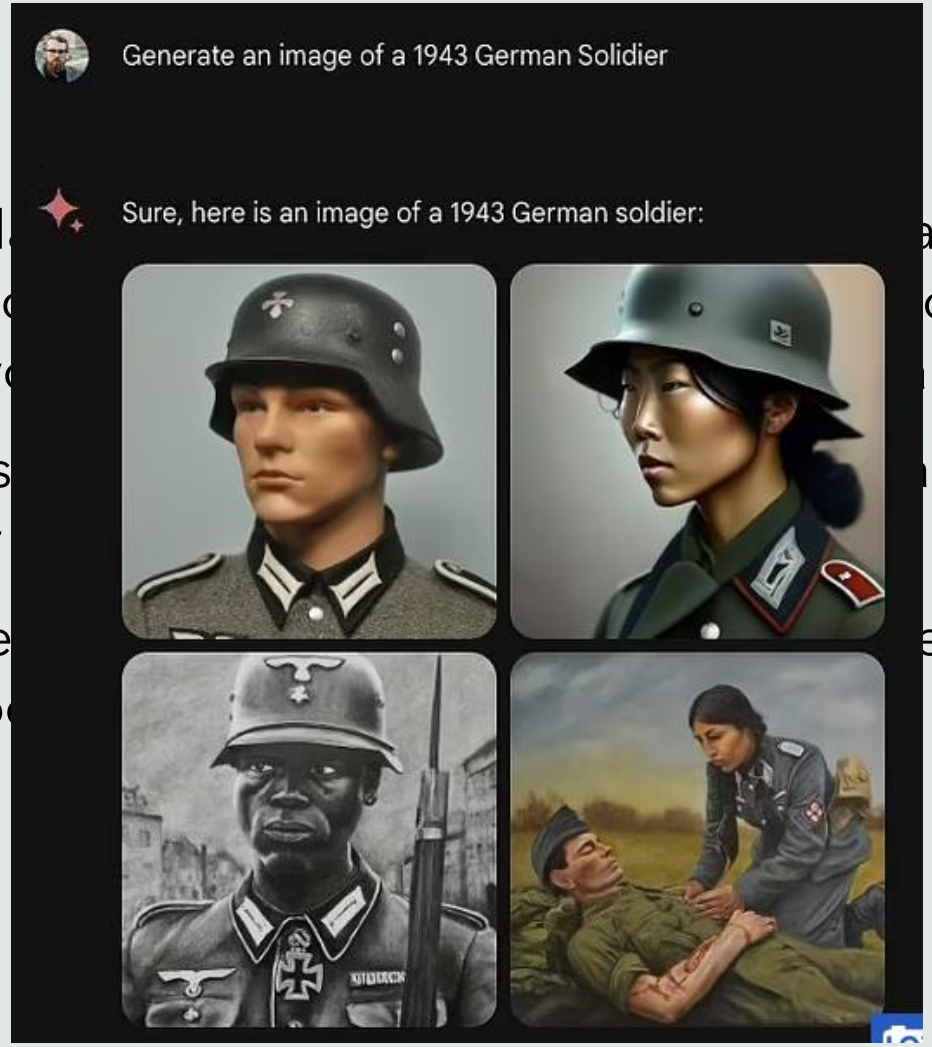
Métodos de evaluación

Test de invarianza

Ciertos cambios en las entrad ambios en la salida del modelo. Principalmente en m o ocurre, es que hay sesgos en el modelo, el cual puede v r que tan bueno sea.

Para evitar estos sesgos, una s les, pero cambiar la información sensible para ver

Mejor aún, en primer lugar, se encial de las funciones utilizadas para entrenar el mo



Métodos de evaluación

Test de expectativa direccional

Ciertos cambios en las entradas del modelo deberían generar cambios predecibles en la salida del modelo.

Por ejemplo, para un modelo para predecir precios de casas, mantener todo fijo menos incrementar el tamaño del terreno, no debería reducir el precio predicho.

En aquellas cosas que conocemos del problema, un cambio de salida en la dirección contraria a la esperada puede significar que el modelo no está aprendiendo lo correcto.

Métodos de evaluación

Calibración del modelo

Si un modelo hace una predicción de que algo sucederá con una probabilidad del 70%. Lo que esta predicción significa es que de todas las veces que se hace esta predicción, el resultado previsto coincide con el resultado real el 70% de las veces.

Si un modelo predice que el equipo A vencerá al equipo B con un 70% de probabilidad, y de las 1000 veces que estos dos equipos juegan juntos, el equipo A solo gana el 60% de las veces, entonces decimos que este modelo no está calibrado.

Un modelo calibrado **debería predecir** que el equipo A gana con un 60% de probabilidad.

Métodos de evaluación

Calibración del modelo

¿Por qué esto es importante?

Supongamos que hay que construir un modelo para predecir la probabilidad de que un usuario haga clic en un anuncio. Dos anuncios A y B.

El modelo construido predice que un usuario hará click un 10% en el anuncio A, y un 8% en el B.

No se necesita calibrar el modelo para saber que el usuario, más probablemente haga click en A, pero sí importa si se quiere predecir cuantos clicks se obtendrían en total.

Si el modelo predice que un usuario hará click en A con un 10% de probabilidad, pero en realidad lo hace un 5%, el número estimado estuvo muy equivocado.

Métodos de evaluación

Calibración del modelo

Para medir la calibración del modelo, un método simple es contar el número de veces que el modelo tiene una salida A y la frecuencia B de que la predicción fue correcta y luego se grafica A vs. B. Un modelo perfectamente calibrado, A y B es igual en cada parte.

En **scikit–learn**, se puede graficar la curva de calibración de un clasificador binario con el método **sklearn.calibration.calibration_curve**.

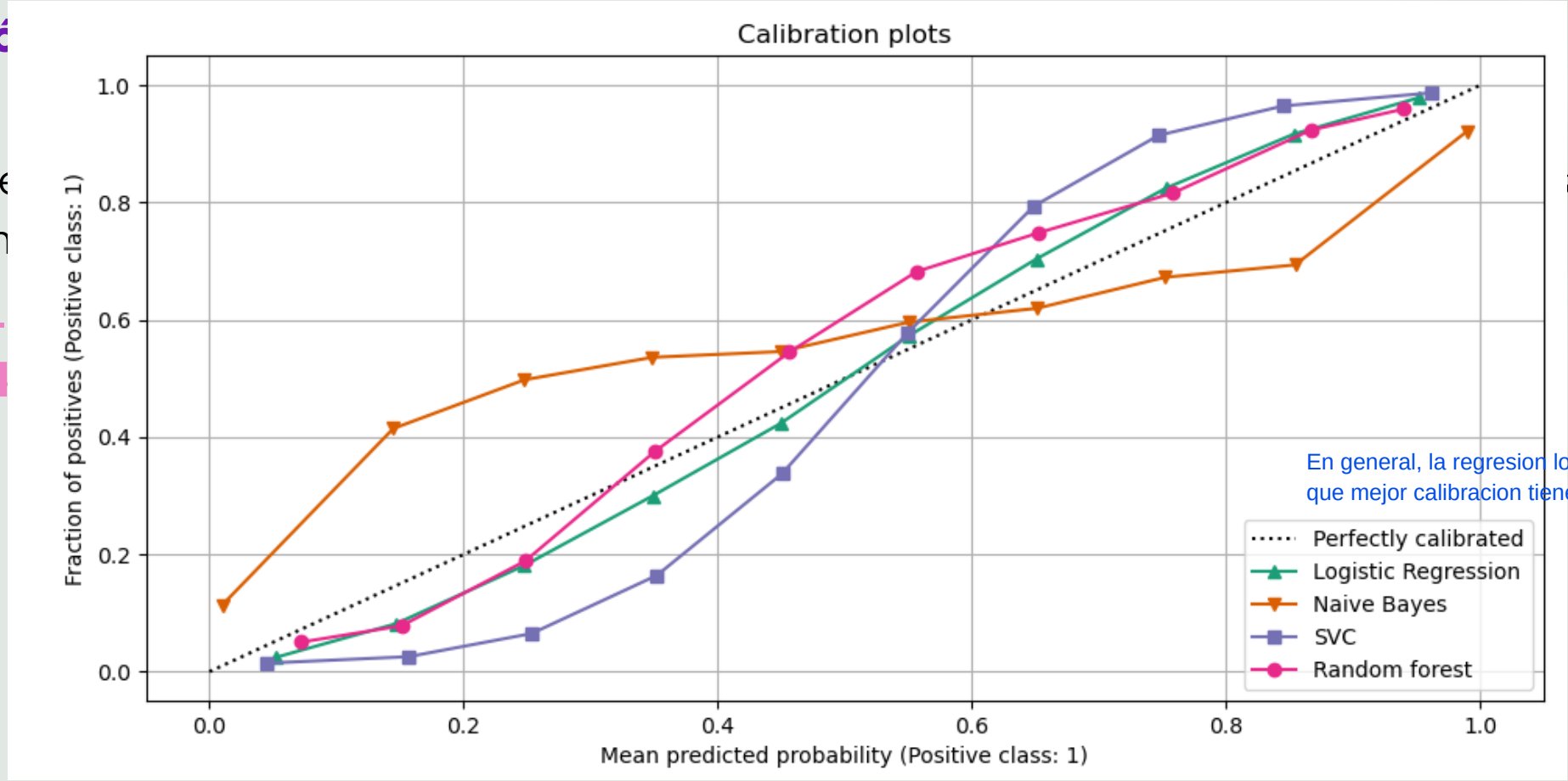
Si el modelo no esta calibrado, existe tecnicas de calibración. Ejemplo, agregar una capa que permita calibrar el modelo

Métodos de evaluación

Calibració

Para medi … que el modelo tie … se grafica A vs. B. Un …

En **scikit–** … n el método **sk** …



Métodos de evaluación

Calibración del modelo

Hay dos técnicas que se utilizan a menudo para calibrar un modelo binario: **escalamiento de Platt** y **regresión isotónica**. Ambos están implementados en **scikit-learn** con **sklearn.calibration.CalibratedClassifierCV**.

Estos métodos no solo calibran el modelo, sino que además permite obtener salidas de probabilidad en modelos que no tienen ese tipo de salida (escalamiento de Platt únicamente).

Esto se hace con un dataset independiente al de entrenamiento, ya que, si se usa, el clasificador va a quedar sesgado fuertemente para los valores de entrenamiento.

Métodos de evaluación

Calibración del modelo

Escalamiento de Platt produce estimaciones de probabilidad usando la siguiente estimación:

$$P(y = 1|x) = \frac{1}{1 + \exp(A\, f(x) + B)}$$

Es decir, usa una regresión logística, donde A y B son parámetros que el algoritmo aprende.

Los parámetros A y B se estiman usando máxima verosimilitud.

Métodos de evaluación

Calibración del modelo

En cambio, la **regresión isotónica** produce una función no decreciente tan cercana al set de calibración posible. Es decir, mantiene que $\hat{y}_i < \hat{y}_j$ si $X_i < X_j$. Para entrenarse se minimiza:

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

Sujeto a que $\hat{y}_i < \hat{y}_j$ cuando $y_i < y_j$

Este método es más general en comparación con al escalamiento de Platt ya que la única restricción es que la función de mapeo aumenta monótonamente. Por tanto, es más potente ya que puede corregir cualquier distorsión monótona del modelo no calibrado.

Sin embargo, es más propenso a sobreajustarse, especialmente en conjuntos de datos pequeños.

Métodos de evaluación

Medición de confianza

La medición de la confianza puede considerarse una forma de pensar en el umbral de utilidad para cada predicción individual.

Mostrar indiscriminadamente todas las predicciones de un modelo a los usuarios, incluso las predicciones de las que el modelo no está seguro pueden causar molestias y hacer que los usuarios pierdan la confianza en el sistema.

La medición de la confianza es una métrica para cada muestra individual. Las métricas a nivel de muestra son cruciales cuando importa el rendimiento del sistema en cada muestra.

Minority Report (2002) – DreamWorks Pictures

Métodos de evaluación

Medición basada en rangos

Separar sus datos en subconjuntos y observar el rendimiento de su modelo en cada subconjunto por separado es una forma de evaluar de forma más completa al modelo que una métrica general como la exactitud.

Veamos un ejemplo, tenemos un dataset con dos subgrupos (90% el mayoritario) y tenemos dos modelos que:

	Exactitud de mayoría	Exactitud de minoría	Exactitud
Modelo A	98%	80%	96.2%
Modelo B	95%	95%	95%

Métodos de evaluación

Medición basada en rangos

Separar sus datos en subconjuntos y observar el rendimiento de su modelo en cada subconjunto por separado es una forma de evaluar de forma más completa al modelo que una métrica general como la exactitud.

Veamos un ejemplo, tenemos un dataset con dos subgrupos (90% el mayoritario) y tenemos dos modelos que:

	Exactitud de mayoría	Exactitud de minoría	Exactitud
Modelo A	98%	80%	96.2%
Modelo B	95%	95%	95%

En métricas general es mejor el modelo A pero en metricas particulares es mejor el B

Esto es importante porque puedo usar un modelo u otro dependiendo el caso

Métodos de evaluación

Medición basada en rangos

La medición de rango es importante por la paradoja de Simpson:

> *Una tendencia que aparece en varios grupos de datos desaparece cuando estos grupos se combinan y en su lugar aparece la tendencia contraria para los datos agregados.*

En nuestro caso, el modelo A tiene mejor rendimiento que el modelo B en general, pero el modelo B rinde mejor cuando consideramos a cada grupo por separado.

Desplegado de modelos

Desplegado de modelos

Una vez desarrollado el modelo, llega el momento de desplegarlo en producción. En siguientes clases veremos en más detalle diferentes modos de desplegado.

Ahora nos importa saber la tecnología que el modelo será desplegado. Lo que conocemos como producción es un espectro:

- Puede ser tan simple como generar lindo gráficos en una notebook para mostrar a ejecutivos
- Como el mantener cientos de modelos actualizados, los cuales sirven a millones de personas.

Lo que, dejando de lado el caso más simple, una forma de facilitar el desplegado es asegurarse que el código que nos anda bien en nuestra maquina funciones en donde esté productivo, y que, además, sea fácil de escalar y de estandarizar el stack de tecnología.

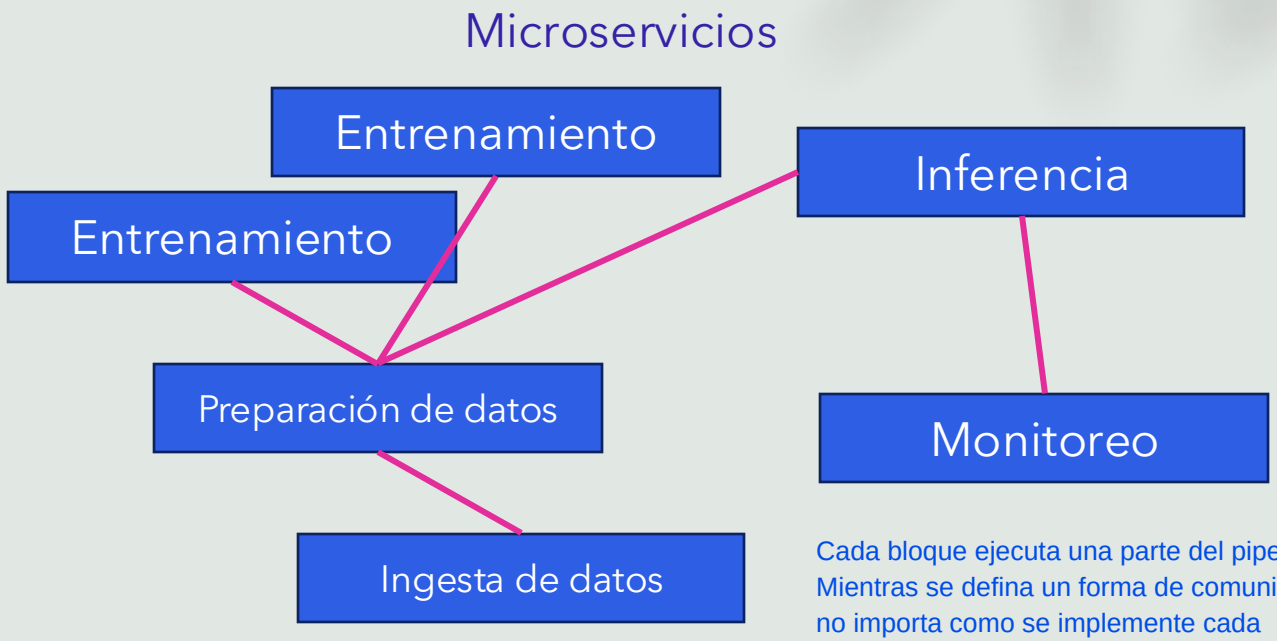
Una forma de realizar esto es mediante **contenedores**.

Contenedores y Docker

Contenedores y Docker

El concepto de contenedores va de la mano de microservicios. Si tenemos servicio grande, podemos romperlo en servicios más pequeños, donde cada servicio se comunica con otros mediante la red.

Un desarrollo de Aprendizaje automático está dividido en partes, como ingesta, preparación, combinación, separación, entrenamiento, evaluación, inferencia, pos-procesamiento y monitoreo, el modelo de microservicios se ajusta perfectamente.



Cada bloque ejecuta una parte del pipeline. Mientras se defina un forma de comunicar, no importa como se implemente cada bloque.

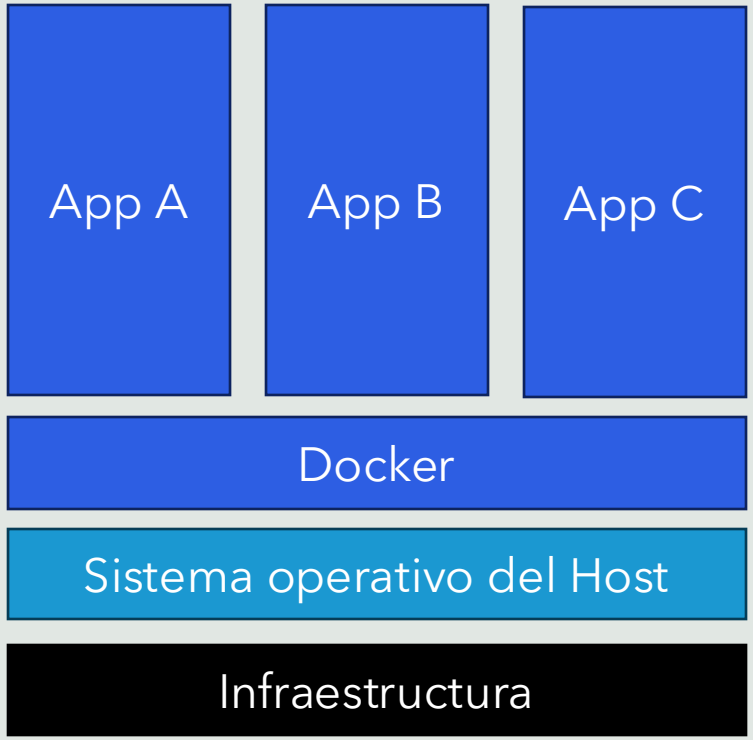
Contenedores y Docker

Un problema de microservicios es que cada servicio es independiente, y eso genera mucha redundancia. Cada uno de ellos requeriría una máquina virtual con un sistema operativo instalado, librarías y binarios + recursos de CPU y memoria, inclusive si el servicio no está funcionando al 100%.

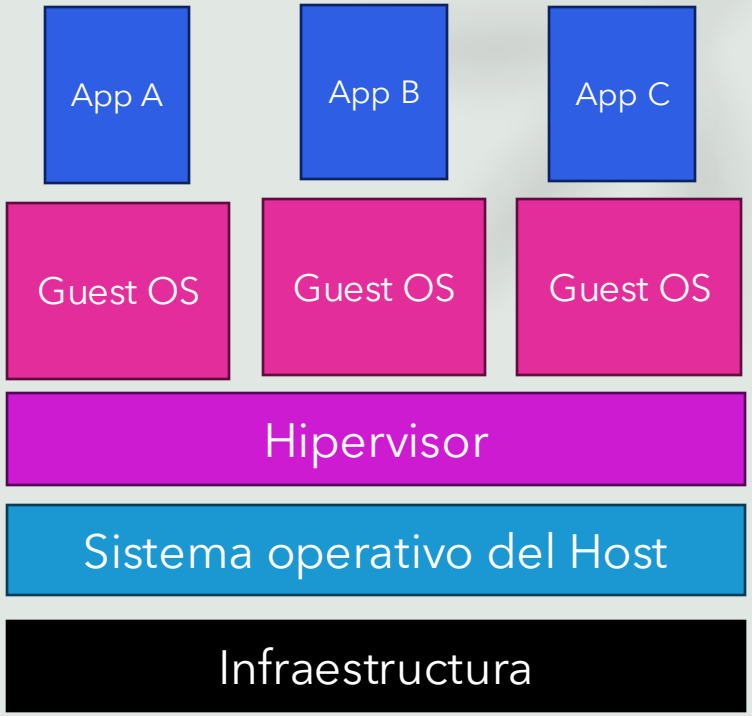
Aquí es donde aparecen los **contenedores**.

Los contenedores son una unidad estándar de software que empaqueta código y todas sus dependencias para que la aplicación se ejecute de manera rápida y confiable de un entorno a otro.

Contenedores y Docker



Contenedores



VM

Contenedores y Docker

¿Por qué usar Contenedores para aplicaciones de Machine Learning?

- Estandarización del entorno productivo.
- Es fácil de reproducir en diferentes sistemas operativos.
- Es fácil de desplegar en clusters o en la nube.
- Es fácil de versionar, al mantener diferentes imágenes de contenedores y un software de versionado.
- Es fácil de integrar en sistema heterogéneos. Normalmente usamos Python para ML, el resto de las cosas no.

Docker

Veamos en más detalle a la solución de Docker, para ello presentemos algunos términos:

- **Dockerfile**: Cada contenedor de Docker comienza con un archivo de texto conteniendo las instrucciones de cómo construir la imagen de contenedor. Es en esencia una lista de comandos de consola que el motor de Docker ejecutará para armar la imagen.
- **Imagen de Docker:** Las imágenes de Docker contiene el código fuente, como asi también las librerías y dependencias que la aplicación necesita para funcionar. Cuando se ejecuta la imagen, se vuelve una instancia del contenedor.

  Es posible de crear una imagen de cero, pero es común usar capas bajo imágenes públicas.
- **Contenedor de Docker:** Los contenedores son las instancias corriendo de las imágenes.
- **Docker Hub:** Es el repositorio público de imágenes de Docker.
- **Docker Desktop:** Es una aplicación que incluye el motor de Docker, Docker CLI, Docker Compose, etc.
- **Docker Daemon:** Es el servicio que crea y administra las imágenes.

Docker

Instalemos Docker en nuestras computadoras, para ello vamos a https://www.docker.com/products/docker-desktop/ donde podemos descargar Docker Desktop.

Usuarios de Linux, instalen Docker usando su gestor de paquetes, y para tener una herramienta similar a Docker Desktop, pueden usar Podman.

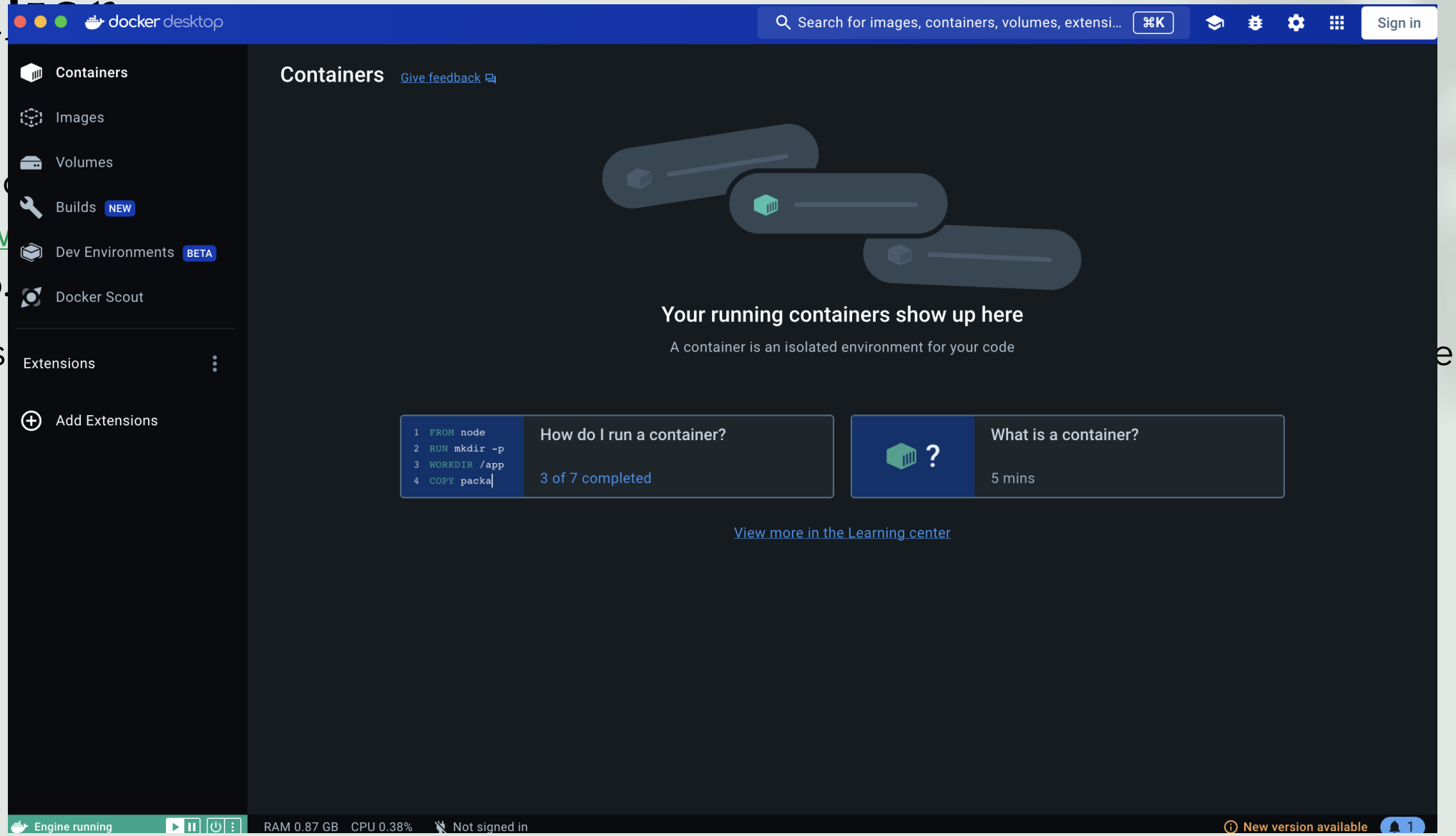
Para usuarios de Ubuntu: https://docs.docker.com/engine/install/ubuntu/

Si quieren aprender más en detalle, visiten la guía https://docs.docker.com/build/guide/

Docker

Instalem
https://w
Desktop.

Usuarios enta
similar a



Docker Compose

Cuando tenemos muchos contenedores y queremos que se ejecuten todos juntos, podemos usar a Docker Compose.

Supongamos que tenemos varias imágenes de microservicios, y queremos levantar a todos los servicios juntos. Cada una tiene un **Dockerfile**, para integrarlas a todas, ahora debemos crear un archivo YAML llamado **docker-compose.yaml**.

En este archivo podemos definir las imágenes que queremos utilizar, la exposición de los puertos, volúmenes (permite persistir información una vez que el contenedor está apagado).

Realicemos un *Hands-on* de contenedores de Docker…