Sistemas Cognitivos Artificiales

Roberto Casado Vara

Tema 10:Ecosistemas en la nube y puesta en producción de sistemas de IA

UNIVERSIDAD INTERNACIONAL DE LA RIOJA

La Universidad en Internet

Ecosistemas en la nube y puesta en producción de sistemas de IA

- Servidores de modelos de inteligencia artificial
- Ecosistemas en la nube
- Aspectos prácticos de la puesta en producción de sistemas de machine learning

Puesta en producción de sistemas de IA

- En este tema hablaremos sobre varias aspectos de la puesta en producción de un sistema de inteligencia artificial.
- La puesta en producción se refiere a utilizar un modelo entrenado en un entorno real, ya sea con usuarios finales o con nuevos datos a evaluar.

Tema 10.1: Servidores de modelos de inteligencia artificial

Inference time

- Hasta ahora nos hemos centrado en el entrenamiento de redes neuronales. Sin embargo, otro aspecto fundamental es utilizar los modelos entrenados para **inferencia** y extraer valor a partir de ellos.

- El proceso de inferencia suele llamarse también *serving time* (vs *training time*), *test time* o *prediction time*.

- Utilizar una red neuronal para inferencia es un proceso menos costoso que el entrenamiento, ya que sólo tenemos que hacer el *forward pass*.

- Sin embargo, esto no significa que utilizar un modelo entrenado no tenga sus complejidades y desafíos. Un modelo podría ser utilizado para un gran número de usuarios o realizar cientos o miles de predicciones por segundo.
    - Por ejemplo, un sistema de recomendaciones en un portal de música o vídeo.

- Vamos a ver una serie de características que nos gustaría tener en un sistema que sirva modelos de machine learning.

Servidores para IA: **concurrencia**

- Una característica que nos gustaría que cumpliera un servidor de modelos es la de **concurrencia**: la capacidad de atender múltiples *requests* (llamadas a nuestro modelo) a la vez.

- Esto es de gran importancia para sistemas que atienden a usuarios finales. Si nuestro modelo devuelve recomendaciones a los usuarios y varios usuarios se conectan a la vez, un servidor no concurrente haría esperar al último usuario demasiado tiempo.

- La concurrencia se consigue habitualmente mediante el uso de **varios procesos o hilos** que están a la espera de recibir y tratar *requests* según van llegando. Si llega una *request* y un proceso está ocupado, otro proceso libre se encarga de la nueva solicitud, de modo que esta no tiene que esperar a que la anterior termine.

Servidores para IA: *batching* de requests

- Sabemos que durante el entrenamiento hacemos *batches* de training data para aproximar el gradiente real al ejecutar SGD.
- De hecho, hacer *batches* de elementos también ayuda a acelerar el entrenamiento de redes neuronales: normalmente ejecutamos el *forward pass* y el *backward pass* de una *mini-batch* como un producto de matrices y, como sabemos, podemos **optimizar las operaciones con matrices** con GPUs (y, de hecho, también con CPUs).
- Si tenemos muchas *requests* por segundo, podemos aplicar también *batching* al usar nuestro modelo entrenado. La idea es ir guardando varias requests según llegan y esperar hasta tener una batch para obtener la salida del modelo. Esto nos da resultados de una manera más eficiente.
- Por supuesto, esto no es útil si recibimos pocas *requests* por segundo, ya que haríamos esperar innecesariamente a los usuarios.

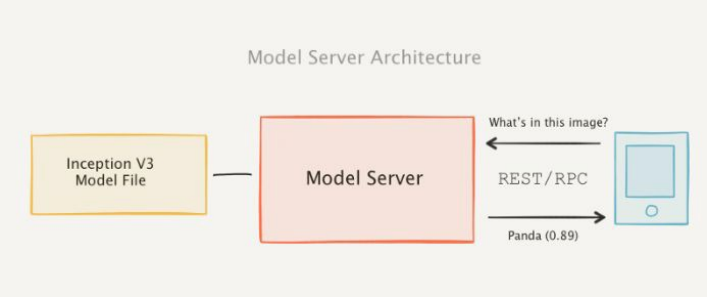
Servidores para IA: **versionado de modelos**

- Otra característica importante de un servidor de modelos es la posibilidad de versionar los modelos disponibles (asignarles un número de versión).
- De este modo, podemos tener una lista de versiones de los modelos que hemos entrenado en el tiempo, e incluso volver atrás a un modelo anterior si los resultados del último modelo no son los esperados.

Servidores de modelos

- En los últimos años, han surgido una serie de **servidores de modelos** de machine learning que facilitan la puesta en producción de modelos entrenados con frameworks como TensorFlow, Caffe, etc.

- La idea es que el desarrollador aporta un modelo salvado en disco (por ejemplo, un fichero con un modelo salvado desde TensorFlow) y el servidor se encarga de cargar el modelo y responder a las requests de manera escalable y eficiente.



- Ejemplos:
  - TensorFlow Serving
  - Clipper
  - DeepDetect

Fuente de la imagen: https://medium.com/@vikati/the-rise-of-the-model-servers-9395522b6c58

Tema 10.2:Ecosistemas en la nube

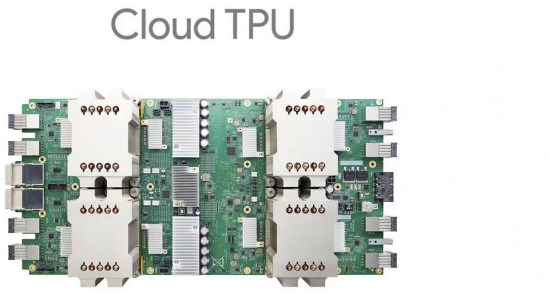
Ecosistemas en la nube

- Mayores proveedores de computación en la nube: Amazon Web Services (AWS), Google Cloud Platform (GCP) y Microsoft Azure.
- Plataformas de servicios en la nube que permiten el desarrollo de soluciones informáticas. Ofrecen capacidad de cómputo (servidores), bases de datos, almacenamiento masivo de datos, etc.
- Permiten construir sistemas informáticos sin tener que preocuparnos por la instalación y mantenimiento de una costosa infraestructura. Se paga bajo demanda por los servidores y máquinas que utilicemos.
- Permite escalar a un mayor número de usuarios fácilmente. Basta con pagar por más recursos.

Ecosistemas en la nube

- Los ecosistemas en la nube ofrecen también recursos para IA. Por ejemplo, es posible contratar máquinas específicas con GPU y el software necesario (*frameworks* estilo TensorFlow, Keras, cuDNN, etc.) para entrenar redes neuronales de manera efectiva.

- Algunos proveedores ofrecen también hardware específico para el entrenamiento de modelos de deep learning, como las TPUs (*tensor processing units*) de Google.



Ecosistemas en la nube

- Igualmente, los ecosistemas en la nube proporcionan APIs para tareas generales y comunes de IA:

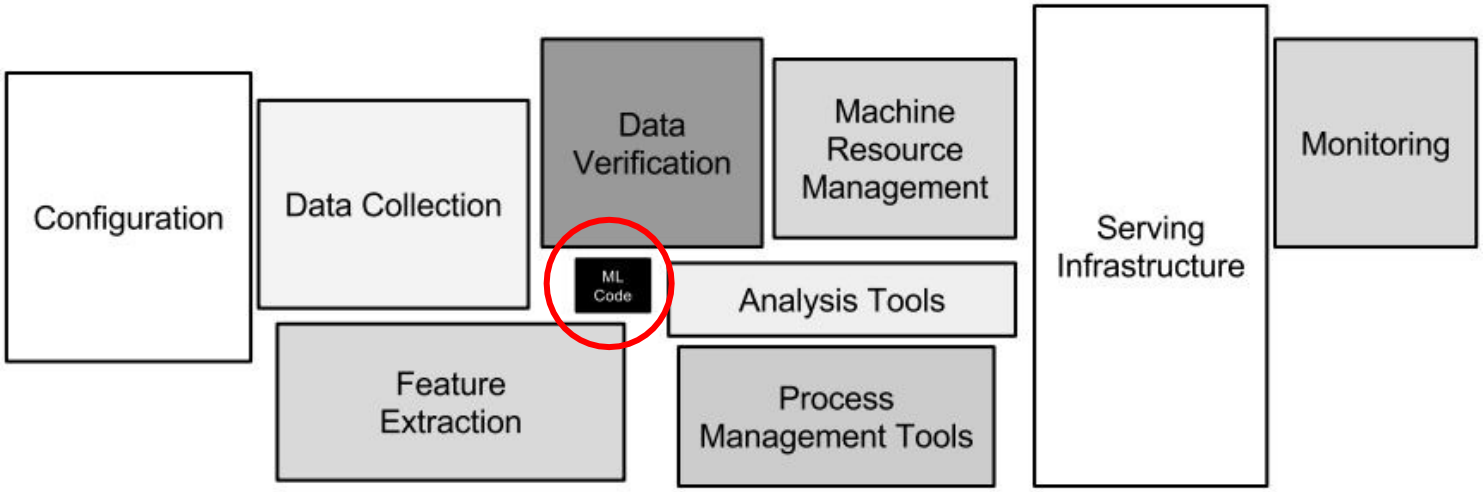
  - Procesamiento del lenguaje natural: análisis de sentimiento, *entity extraction*, *POS tagging…*
  - *Speech-to-Text* y *Text-to-Speech*
  - Traducción automática
  - Visión: clasificación de imágenes, detección de objetos, extracción de texto en imágenes…
  - Análisis de vídeo: extracción de contenidos en vídeos y etiquetado.
  - Chatbots: desarrollo de asistentes conversacionales.

- Entrenar modelos que resuelvan estas tareas generales de manera adecuada es complejo y requiere de grandes inversiones en obtener los datasets necesarios, por lo que en ocasiones merece la pena recurrir a estas soluciones *off-the-shelf*.

Tema 10.3: Aspectos prácticos de la puesta en producción de sistemas de *machine learning*

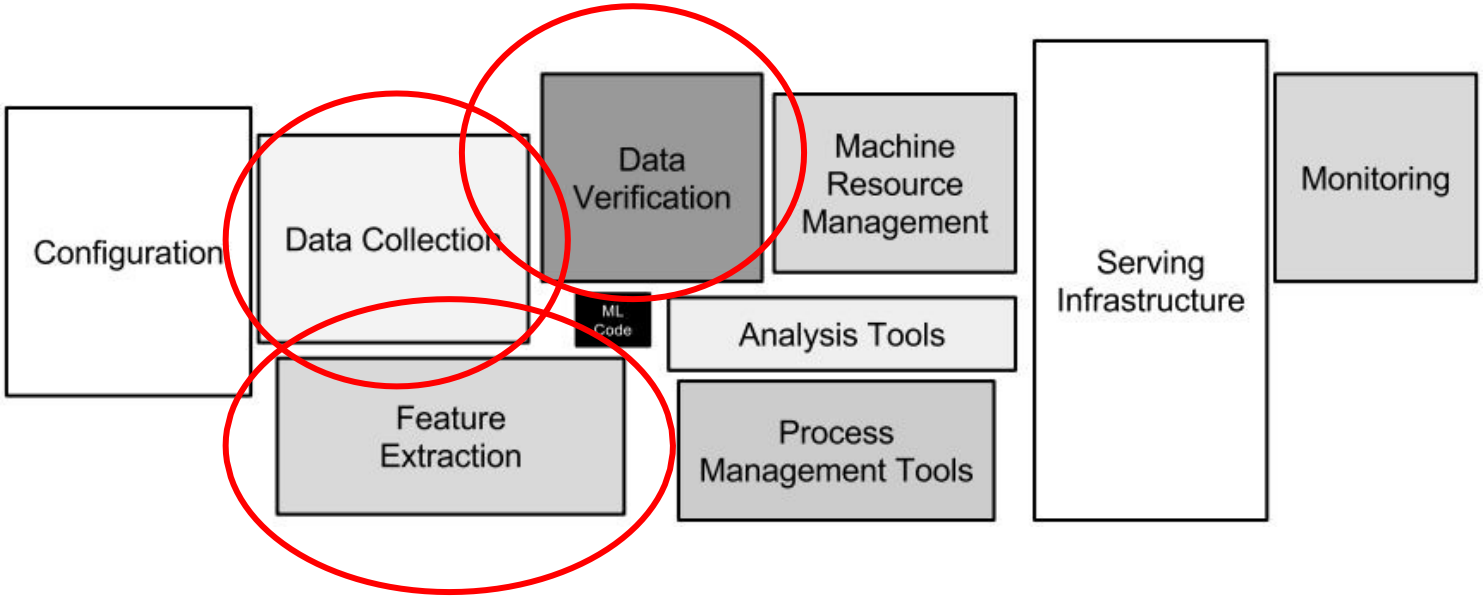
Desarrollo de un sistema de ML

- Código de ML…¡sólo una pequeña parte!
  - Código TF, Keras, etc.



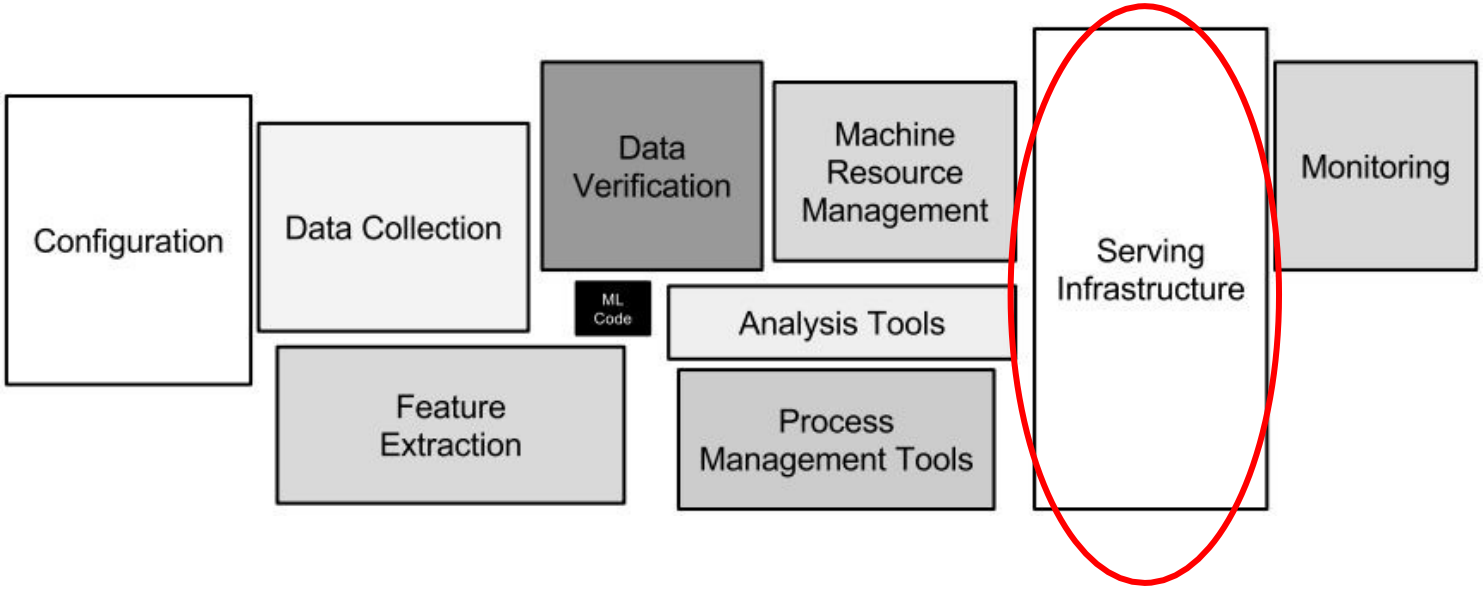
Desarrollo de un sistema de ML

- Gran parte del trabajo: **colecta y tratamiento de datos**
  - Elegir qué datos vamos a utilizar en nuestros modelos.
  - Recopilar los datos de las diversas fuentes (logs, bases de datos, etc).
  - Verificar su consistencia y corrección.



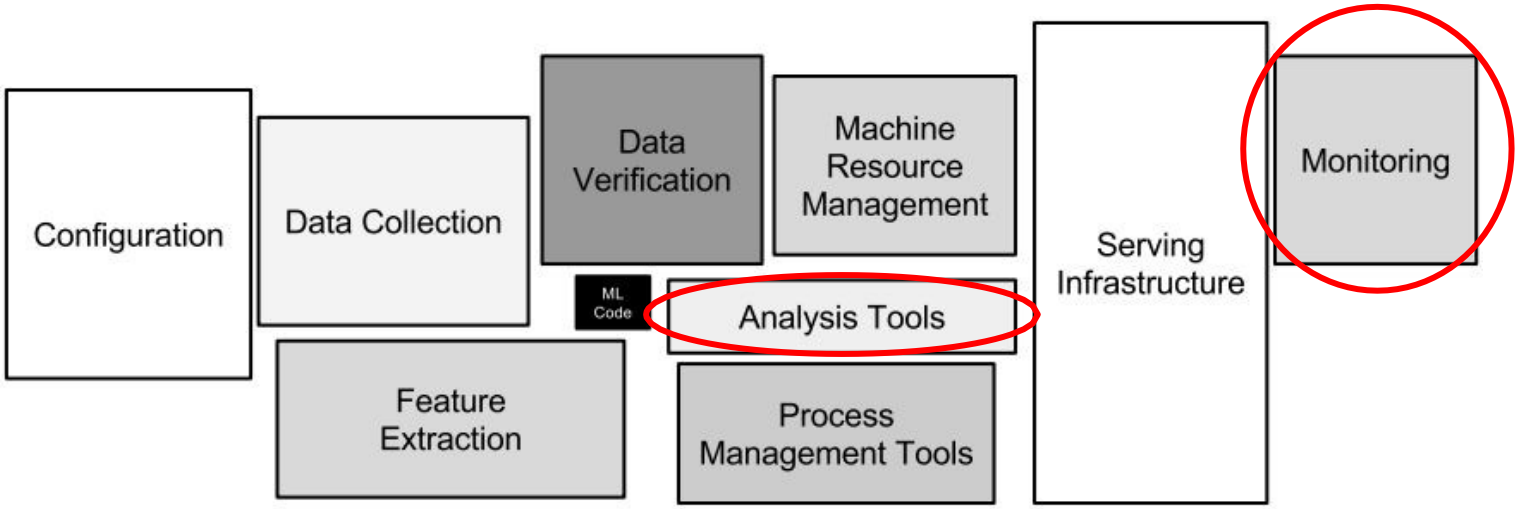
Desarrollo de un sistema de ML

- Servidores de modelos / infraestructura de producción



Desarrollo de un sistema de ML

- Monitoring, análisis.



Training-serving skew

- Problema muy común en sistemas de ML en producción. Consiste en la degradación del rendimiento entre el entrenamiento de un modelo y su uso para inferencia.

- El modelo tiene muy buenas métricas de entrenamiento (*training*), pero funciona muy mal una vez que se utiliza con datos reales en un entorno de producción (*serving*).

- Posibles motivos:
  a. Los datos utilizados durante el entrenamiento son tratados de una forma distinta a cómo se usan para *serving*.
     - ¿Se normalizan las features durante *serving*?
     - ¿Están disponibles todas las features durante *serving*?
     - El código que trata las features debería ser el mismo durante *training* y *serving.*
  b. La distribución de los datos de entrenamiento es distinta que la que el modelo ve durante *serving*.
     - Sistema entrenado con datos de 2010-2015 es usado en 2018.
     - Sistema entrenado con datos de usuarios de China es usado con usuarios de México.

- Los modelos no son mágicos: **funcionan bien sólo con el tipo de datos que han visto durante el entrenamiento.**

Comportamiento del modelo en subpoblaciones

- Un fenómeno que puede dar lugar a problemas en nuestro modelo, sobre todo si tratamos con usuarios.

- Las métricas son buenas al entrenar y en producción, pero el comportamiento del modelo es inestable y funciona mal para ciertas subpoblaciones (compensado en las métricas globales por otros usuarios para los que funciona bien).

- Ejemplos de subpoblaciones:
  a. Usuarios de cierta edad.
  b. Usuarios de cierta zona geográfica.
  c. Usuarios de pago (graves consecuencias para el negocio).

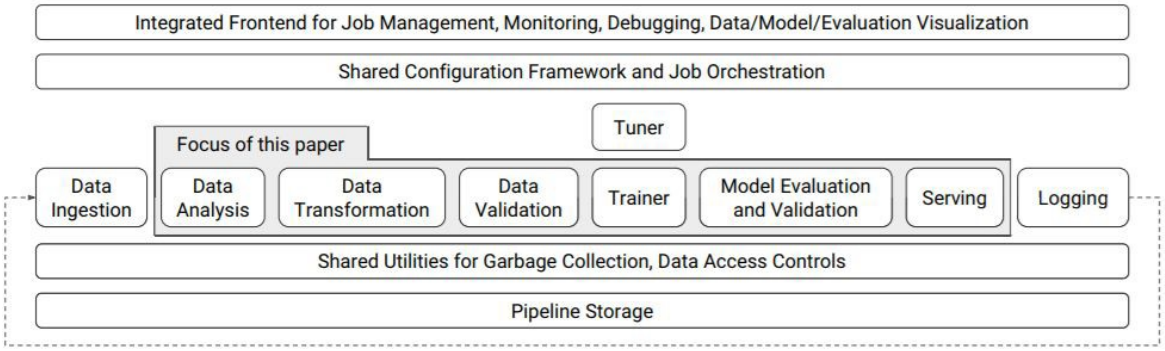
- Difícil de detectar y de solucionar.

Continuous training pipelines

- En ocasiones, necesitamos actualizar nuestro modelo periódicamente. Por ejemplo, un sistema de *spam* necesita ser re-entrenado cada pocos días para ser capaz de detectar nuevas formas de spam.

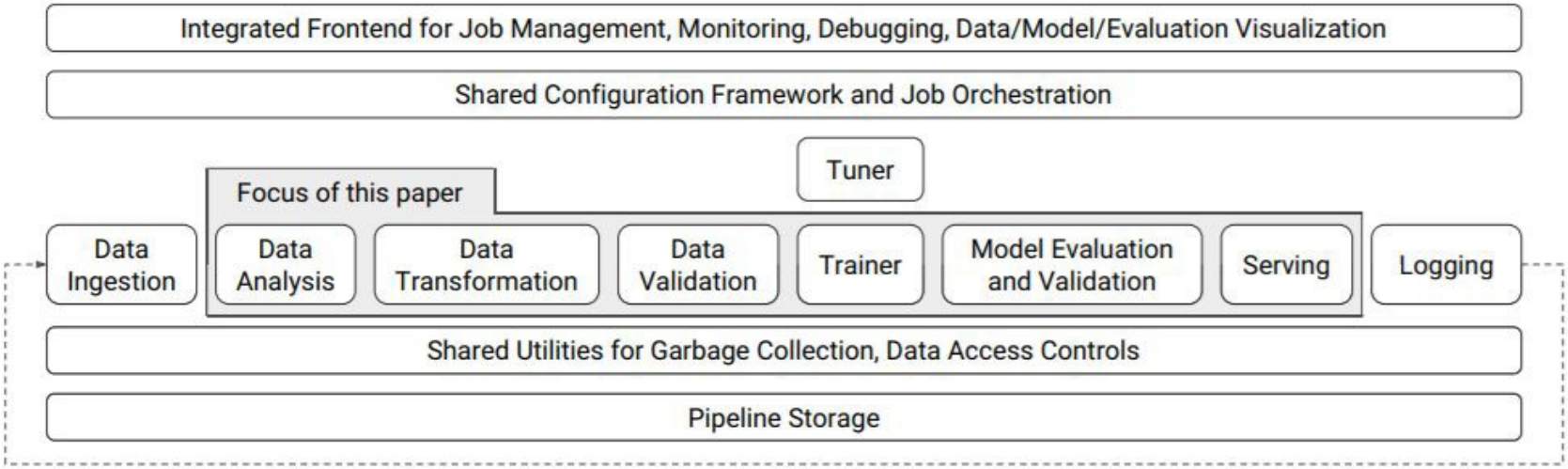
- Cuando esto tiene que pasar cada poco tiempo, se hace necesario automatizar la pipeline de entrenamiento y puesta en producción, añadiendo una mayor complejidad al sistema.

- Una **continuous training pipeline** automatiza el proceso de recolecta de datos, entrenamiento del modelo y deployment del modelo a servidores.



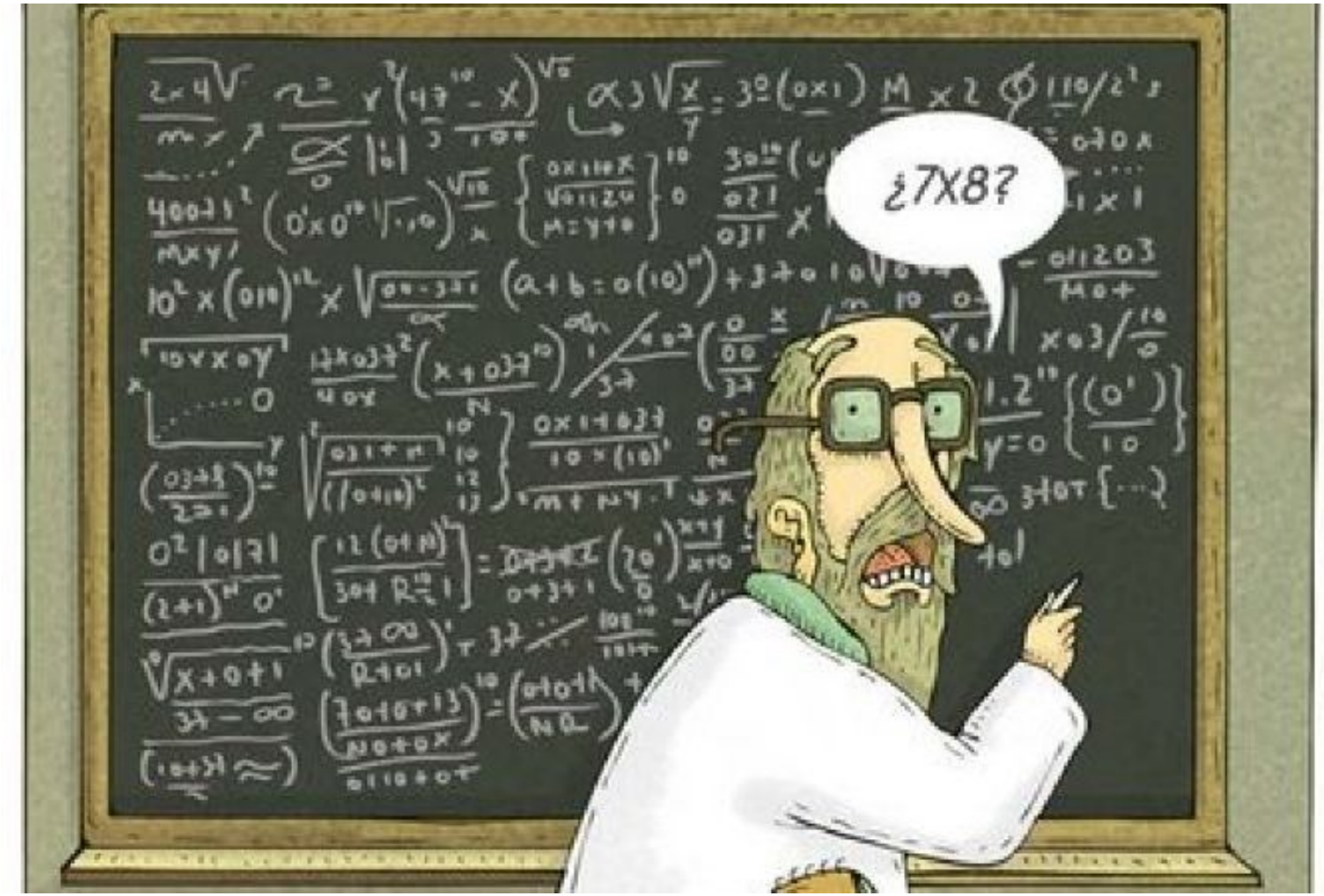
Fuente de la imagen: https://dl.acm.org/citation.cfm?id=3098021

Continuous training pipelines



Fuente de la imagen: https://dl.acm.org/citation.cfm?id=3098021

¿Dudas?



www.unir.net