Machine Learning

Del ~~Notebook~~ Lab a Produccion

Quien Soy

German Bourdin - Technical Leader @ ~~Machinalis~~ MercadoLibre

@g2k88 (http://twitter.com/g2k88)

gbourdin (https://github.com/gbourdin)

german.bourdin@gmail.com (mailto:german.bourdin@gmail.com)

De donde vengo

Machinalis

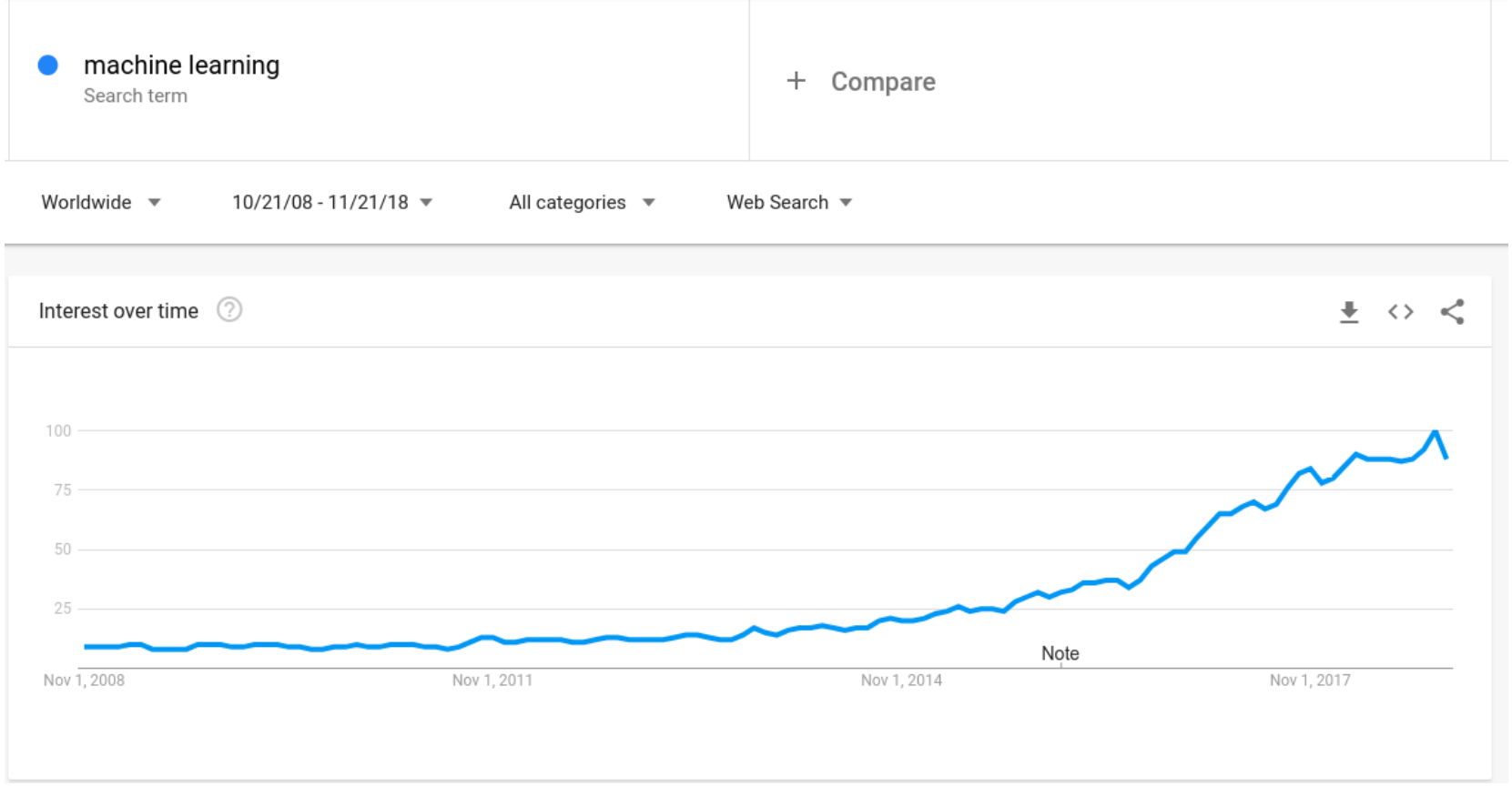
- Complex Web
- Data Science
- Machine Learning
- Machine Learning para Fintech e E-Commerce

MercadoLibre

- Platarforma de E-Commerce mas grande de America Latina
- Mucho de lo mismo que veniamos haciendo pero varias veces mas grande

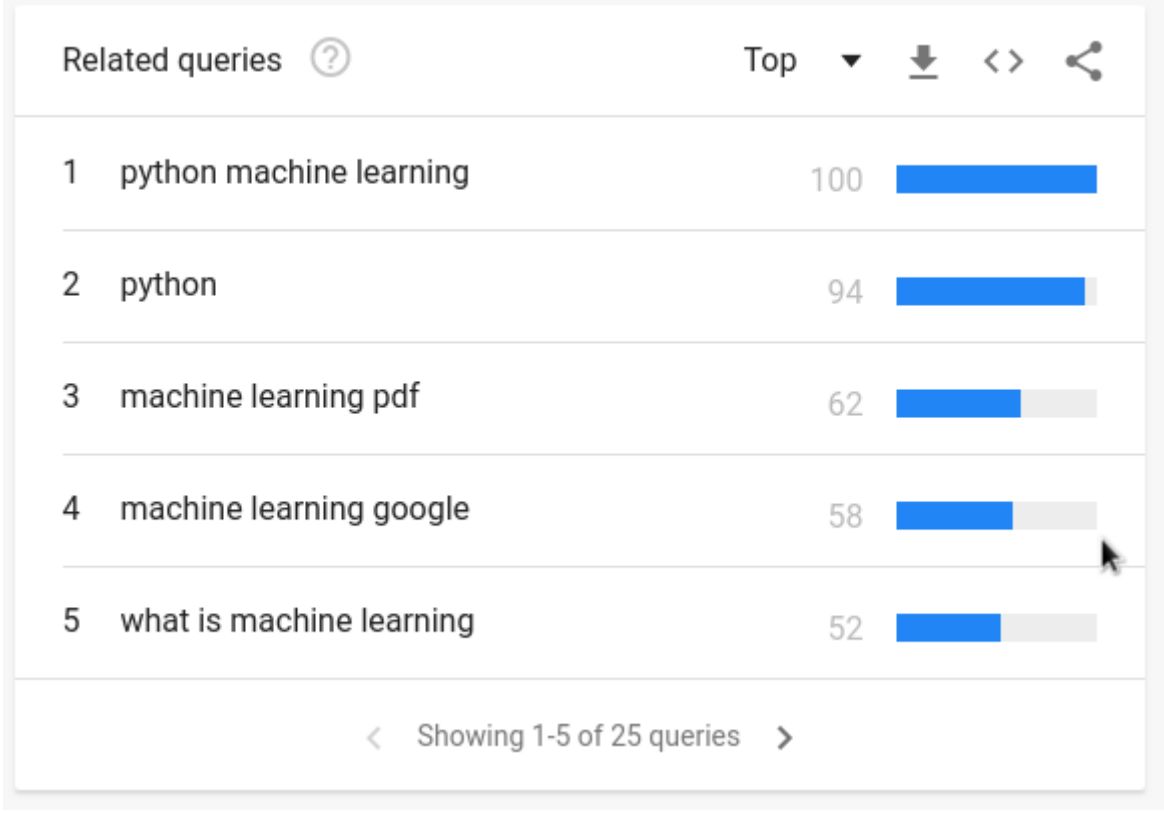
Intro

Machine Learning, es uno de los temas mas populares de los ultimos 10 años y sigue en aumento



Intro

Python es el termino mas buscado junto a machine learning en los ultimos años

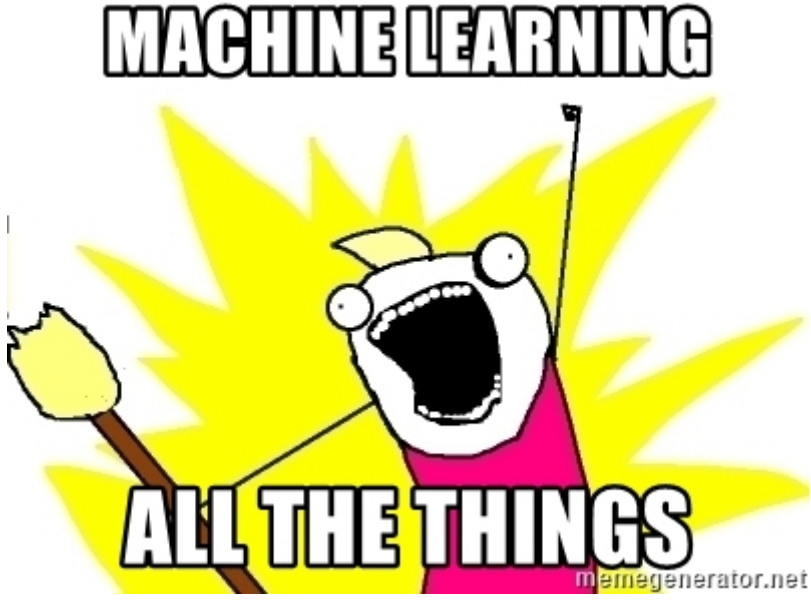


Intro

Basicamente

Intro

Basicamente



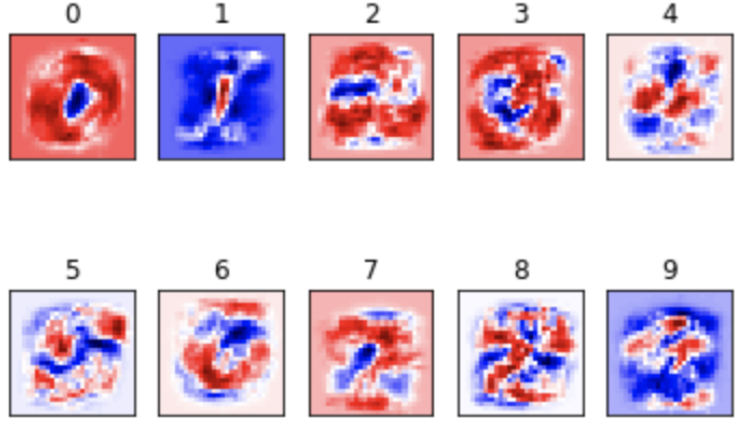
Intro

Intro

Como ir de esto:

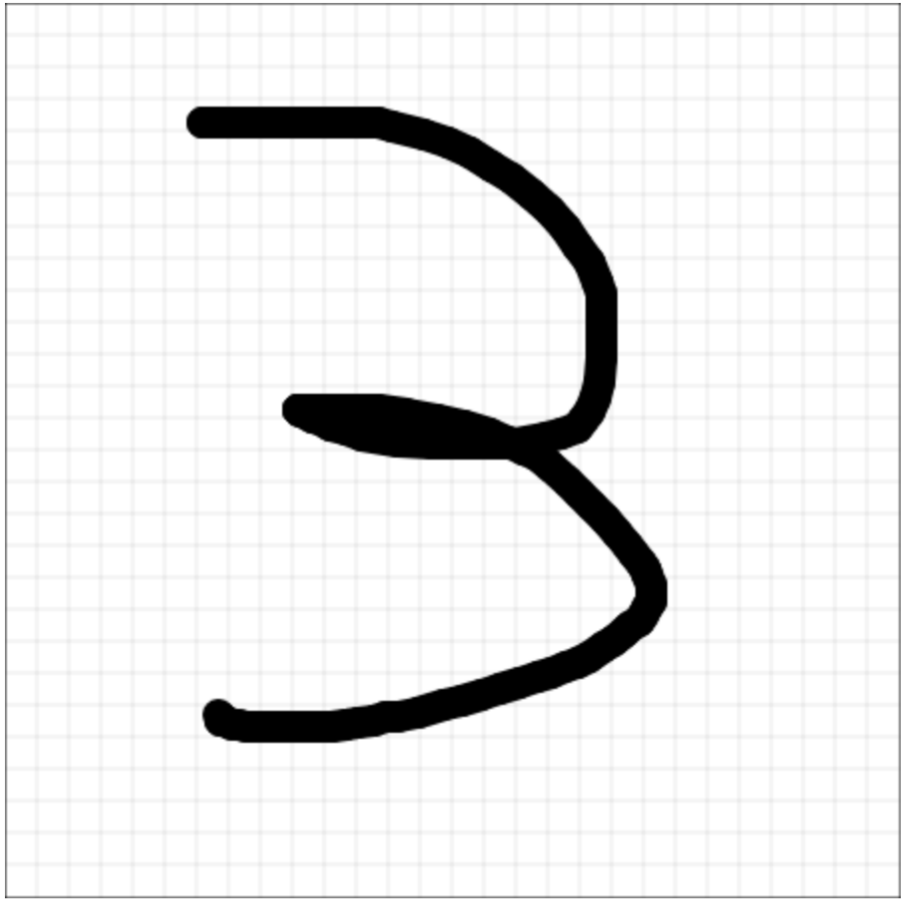
```
Training Step:3600  Accuracy =  0.9061  Loss = 0.2604808
Training Step:3700  Accuracy =  0.906  Loss = 0.25868016
Training Step:3800  Accuracy =  0.9063  Loss = 0.25693423
Training Step:3900  Accuracy =  0.9065  Loss = 0.2552398
Training Step:4000  Accuracy =  0.9066  Loss = 0.25359392
```

In [15]:
```
for i in range(10):
    plt.subplot(2, 5, i+1)
    weight = sess.run(W)[:,i]
    plt.title(i)
    plt.imshow(weight.reshape([28,28]), cmap=plt.get_cmap('seismic'))
    frame1 = plt.gca()
    frame1.axes.get_xaxis().set_visible(False)
    frame1.axes.get_yaxis().set_visible(False)
plt.show()
```

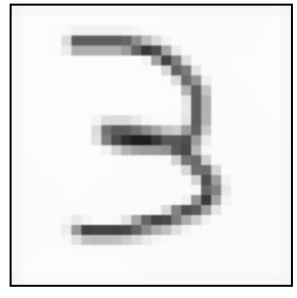


Intro

A esto:



input:



output:

Prediction!

0 0.000
1 0.000
2 0.001
3 0.997
4 0.000
5 0.001
6 0.000
7 0.000
8 0.000
9 0.000

Overview

Disclaimers

Disclaimers

- Esta no es una charla de machine learning

Disclaimers

- Esta no es una charla de machine learning

Disclaimers

- Esta no es una charla de machine learning
- No soy data scientist
- Hay mucho código de soporte de la charla, la idea no era construir los mejores predictores si no mostrar como llevar eso a producción, asi que no esperen que sean precisos
- La mayoria de lo que diga, es una serie de recomendaciones, no hablo en absolutos

El notebook

- Es una herramienta de experimentación y exploración
- Es documentación
- A veces deja registro de varios experimentos y cosas que en el camino no andaron
- En el mejor de los casos, hace una sola cosa

El Notebook

En general, si estamos experimentando vamos a tener mas o menos las siguientes etapas en nuestro notebook:

- Análisis y limpieza de datos
- Definicion/configuracion del modelo
- Training
- Predicción
- Evaluación

Probablemente con varias iteraciones de los últimos pasos hasta que llegamos a un modelo que nos convence

Demo Time!

Bah, les muestro un par de notebooks asi nomás

Modelado de modelos ?

Saliendo del notebook, hay tareas que nos van a seguir interesando hacer con nuestro modelo y tareas que ya no tanto.

- Analisis de datos -> Y no, la verdad que no
- Limpieza de datos -> A menos que vaya a entrenar de nuevo, no
- Definicion/Configuracion del modelo -> Si
- Training -> Si
- Predicciones -> SI!
- Evaluacion -> Y, no, la verdad que no (ya lo hicimos en otro lado, no voy a estar re-evaluando a cada rato)

Modelado de modelos

A las cosas que ya veniamos haciendo, le agregaria la posibilidad de cargar y restaurar entrenamientos anteriores

- dump
- load

Modelado de modelos

Proponemos una interfaz standard para estas cosas. Basada en la interfaz de scikit-learn + dump y load. Provee justo lo básico que esperamos de un "coso que predice", es suficientemente facil de extender y la mayoria de los que hacen DS/ML estan familiarizados con esa interfaz.

Interfaz Propuesta

In [1]:
```
from abc import ABCMeta, abstractmethod
class BaseModel(metaclass=ABCMeta):
    @abstractmethod
    def predict(self, X):
        passs

    @abstractmethod
    def fit(self, X, y):
        pass

    def dump(self, path):
        """
        Dumps the current trained model to path
        """
        pass

    @classmethod
    def load(cls, path):
        """
        Load saved model from path.
        """
        pass
```

Packages - Libraries

- No necesitas ser Jose Tensorflow para desarrollar un paquete en python
- No es necesario que lo publiques en pypi para instalarlo
- Pero dependiendo del tamaño de tu infraestructura, tal vez podes tener un pypi privado y publicar ahi!
- Los modelos de machine learning pueden pensarse como cosas reusables, no necesariamente va a servir solo para este proyecto, no necesariamente va a solamente servirse desde la app X
- Hacer una libreria pip instalable en lugar de tener un archivo que pones en el path de tu proyecto cuando lo necesitas lleva, nada de esfuerzo extra y tiene muchos beneficios

Packages - Libraries

Hiciste tu predictor pip instalable, de paso te aseguraste:

- De listar todas las dependencias
- Que este documentado claro como se instala en otros entornos
- Si tenias command line scripts, los podes agregar al path y eso queda lindo!
- Ahhh y si querés pegar la vuelta, ahora podes importar desde un notebook tu modelo entero configurado y hacer metricas sobre eso limitandote solo a hacer análisis

Packages - Libraries

Y ya que estamos, pensemos en este punto en hacer tests!

Demo Time!

Armando una API Rest para servir modelos

Armando una API Rest para servir modelos

- REST es mas o menos standard para comunicar micro-servicios
- Nos permite tener una capa agnostica del lenguaje con nuestro cosumidor (no necesariamente estamos proveyendo servicios a una web, no necesariamente el consumidor esté escrito tambien en python, etc)
- Tenemos herramientas para armar y servir agregando una capa muy finita arriba de nuestro modelo

Flask + FlaskRESTPlus

- Descripcion de flask (microframework y eso)
- Descripcion de flaskRestPlus

Demo de Código!

Armando una API Rest para servir modelos

Por que separar la api del predictor?

- Modularidad, no es poco normal que cambiemos el modelo, la api en general va a cambiar poco
- Distintos concerns, en general en la API vamos a enfocarnos mas en temas de como se serializan los datos, de performance, de seguridad, etc
- Podemos decidir que una API Rest no es lo que necesitamos y solamente tirar esa parte del proyecto
- Podemos decidir usar otra tecnologia para servir y no afectar al modelo con el que estamos trabajando

Armando una API Rest para servir modelos

Por que no Django + DRF?

Por que no tensorflow serving?

Por que no Falcon?

Por que no el microframework de la semana?

Armando una API Rest para servir modelos

Q: Los microservicios son una buena idea para conseguirte dolores de cabeza gratis, por qué no lo integrás todo con tu sistema existente?

A: Hablemos mas de esto despues, pero en general, los sistemas que sirven modelos de machine learning suelen necesitar prestar atencion a cosas muy distintas a por ejemplo un sitio de clasificados tradicional, donde el tiempo que vas a pasar esperando, en general va a ser de latencia de red o de disco y podes hacer cosas con el CPU mientras esperas.

Demo Time!

Implementación de las APIs de los ejemplos!

El Frontend

Siguen las demos!

El frontend

Descubrimos que mi predictor de precios de casas, está roto y tengo hasta despues del partido para arreglarlo antes de que mi prima me mate.

El frontend

Descubrimos que mi predictor de precios de casas, está roto y tengo hasta despues del partido para arreglarlo antes de que mi prima me mate.

Veamos como reemplazar el modelo en 5 minutos (por uno que ande menos mal)

El frontend

Descubrimos que mi predictor de precios de casas, está roto y tengo hasta despues del partido para arreglarlo antes de que mi prima me mate.

Veamos como reemplazar el modelo en 5 minutos (por uno que ande menos mal)

Live Coding! Nada puede salir mal!

Extras

Algunas cosas para tener cuidado

Extras

Algunas cosas para tener cuidado

ThreadSafety

Hay muchas herramientas para construir modelos de Machine Learning, que no son threadsafe, xgboost por ejemplo aclara que el predict de los modelos que construye tiene este problema.

Esto es, no puede ejecutarse en muchos threads al mismo tiempo, y si eso ocurre, no garantizan que funcione o que funcione bien. A la hora de configurar servidores para modelos de ese tipo, hay que tenerlo en cuenta, es una buena idea aclararlo en nuestros READMEs!

Extras

Algunas cosas para tener cuidado

Lazy Loading

Otra cosa que suele pasar con algunas de las herramientas mas comunes, es que generan cosas en memoria que no son serializables o que no pueden compartirse entre distintos procesos. Es el caso de TensorFlow entre otros, donde las sesiones no se pueden compartir y algunas cosas de los modelos no son serializables.

Para evitar problemas con esto, conviene habilitar el "lazy load" en la config de uwsgi. Por defecto uwsgi antes de forkear los workers carga la app en memoria y hace algo de inicializacion, eso ayuda a que el footprint en memoria sea mas chico. Para cosas como tensorflow, queremos que pase exactamente lo opuesto.

Extras

Algunas cosas para tener cuidado

Cuidado con el GIL

Usar threads para aumentar el throughput the una aplicacion web suele ser tentador, tengan en cuenta sin embargo, a la hora de configurar un servidor para modelos de Machine Learning que lo mas probable es que su cuello de botella sea en CPU: ie, sus procesos no pierden tiempo buscando cosas del disco o esperando a la red, estan usando el CPU para hacer calculos. Por como funciona el GIL, agregarle mas threads a esto, lo unico que va a conseguir es aumentar la latencia de todos los threads que estan compartiendo el interprete!

Usen procesos en su configuracion, apunten a no mas de uno por CPU y tuneen desde ahi

Extras

Métricas, que cosas mirar

Más allá de los clásicos precission, recall, etc, que cosas queremos ver de los modelos que estamos sirviendo

Métricas, que cosas mirar

Latencia y tiempos de respuesta

No suele ser un tema menor cuanto tiempo tarda en responder un modelo, antes de poner algo en produccion, si van a tener requerimientos sobre esto, hacer benchmarks y load tests del modelo solo y del modelo montado sobre el servidor para al menos entender como se comporta. Tener una idea de con que volumen de requests vamos a empezar a encolar pedidos en lugar de responder enseguida y cuanto vamos a tener que escalar para satisfacer los requerimientos.

Métricas, que cosas mirar

Uso del CPU

De nuevo, hagan experimentos con carga similar a la que esperan de produccion, observen, aunque sea de la forma más rustica la carga y el nivel de uso del CPU. Algunas librerias llaman a rutinas en C que se escapan de las limitaciones de python y usan threads o forkean para usar 2, 4, 8 o TODOS los cpus disponibles. Si esto les pasa, en seguida van a notar que aunque tengan 1 servidor cada 2 CPUs, si hay requests contestandose en paralelo, la performance de todas va a estar degradada. Enterensé lo antes posible de esto, tomen medidas para evitarlo.

Memoria

Si van a estar cargando modelos grandes en memoria, tengan en cuenta cuanta hay disponible, no quieren irse a swap (o que los mate el OOM)

Métricas, que cosas mirar

Resultados de las predicciones y su distribución

Esto no se cae tan de maduro, pero, antes de poner algo en produccion, solemos tener idea de como performa contra datos productivos que teniamos ejecutado. Con los datos que estemos prediciendo una vez que vayamos a produccion, a menos que tengamos un loop donde los datos lo mismo se revisen y etiqueten a mano, solamente tenemos la estadística de como suelen estar compuestos.

Si por ejemplo, hicieron un detector de panchos, y en las muestras que tomaron de produccion, el 20% de las imagenes tenian panchos, monitoreen sus resultados, si en testing tenian buenas metricas pero en produccion de repente pasan a detectar que el 80% de las imagenes estan teniendo panchos, o estamos sufriendo una invasion de vendedores de panchos o estamos por algun motivo devolviendo falsos positivos.

Si pueden, monitoreen de forma automatica, si no, al menos colecten los logs y parseen de vez en cuando!

Extras

Performance Tweaks

Algunas cosas que se pueden hacer si estamos teniendo tiempos de respuesta poco aceptables o se nos encolan muchos requests, etc. Es una lista incompleta y son recursos para considerar cuando ya otras cosas fallaron.

> *Premature optimization is the root of all evil!*

Performance Tweaks

Buffering

Algo que suele pasar por la naturaleza de los algotimos que hay debajo (aunque no siempre es el caso, midanlo si estan en esta situación) es que hacer una prediccion para 1 dato o para 50 lleva el mismo o casi el mismo tiempo.

Ejemplo: Supongamos que obtener el precio sugerido de una casa me lleva 100ms, y el de 10 casas juntas me lleva 150ms, si predijera en serie 10 pedidos que entran juntos, el primero va a tardar 100ms, el segundo 200, el ultimo 1000ms.

Un posible tweak que se puede meter acá es una ventana que junte requests y cada X ms mande a ejecutar, es complejo de implementar pero ayuda a reducir la latencia media en estos casos. No quiero dar ejemplos puntuales porque es complicado y hay que tunearlo mucho.

Performance Tweaks

Caching

El universo de cosas que tenes que predecir es finito y acotado?

Respondes muchas veces la misma pregunta?

Tiene sentido tener pre-calculados los resultados para estos casos?

Preguntas?

Preguntas?

Agradecimientos

- A pyar y a la organizacion
- A los sponsors
- @elnassto por el frontend para el predictor de propiedades

Recursos

- Slides
- Ejemplos
- Cookiecutters

Todo disponible en https://github.com/gbourdin/charlas/ (https://github.com/gbourdin/charlas/)


THANK YOU FOR YOUR TIME
quickmeme.com

Cierre

During a conversation I had with Peter Norvig, Norvig quoted a friend of him who said:

> *"Machine Learning development is like the raisins in a raisin bread: 1. You need the bread first 2. It's just a few tiny raisins but without it you would just have plain bread."*

Marcos Sponton - Don't Buy Machine Learning (2016)