Análisis de Datos

Clase 7. Ingeniería de variables. Caso de estudio end-to-end.

Plan

- Ingeniería de variables.
- Caso de estudio. Clasificación de señales 1D.
  - Features en dominio de tiempo y frecuencia.
  - Repaso de Transformada de Fourier, DFFT. Espectrograma.
- Conceptos de ingeniería de SW para definición de cadenas de procesamiento complejas.
  - Programación orientada a objetos.
  - Algunos patrones de diseño.
- Organización y despliegue de un proyecto de ML (ejemplo)
  - Preparación de datos, ingeniería de variables, entrenamiento y exportación del modelo.

1. Introducción a la ingeniería de variables

1. Introducción a la ingeniería de variables

- En general lo que tenemos es la variable objetivo y los datos crudos (raw data), producto de trabajo de recolección de datos. Debemos extraer features de los datos crudos (*featurization*). Esta extracción de features va a depender del dominio en el cual estemos trabajando. Algunos ejemplos:
  - BMI: Tengo el peso(w) en kg y la altura (h) en cm de una persona y obtengo $BMI = w/h^2$
  - Fechas: días de la semana, mes, minutos desde un cierto momento
  - Texto: Bag of words (+stemming), Bigrams (miro de a pares), embedding (word2vec), TF-IDF(Term frequency-Inverse Document Frequency, peso los features)
  - Imágenes: Transformaciones sobre los pixels (blur, blanqueado), Features descriptivo (histograma), Detectores de features locales (detección de bordes)
  - Audio: transformaciones en el espacio del tiempo y la frecuencia.

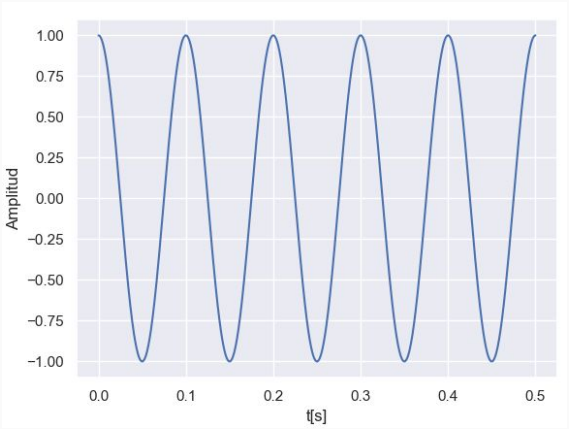
2. Caso de estudio. Señales 1D.

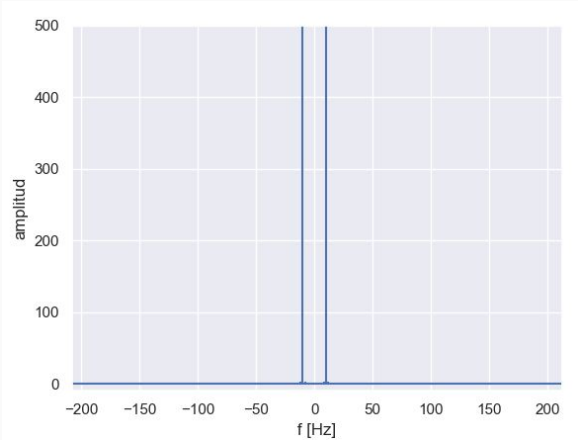
2. Caso de estudio. Señales 1D.

- El objetivo de este caso de estudio es clasificar entre distintas familias de aves utilizado principalmente grabaciones de audios de los cantos de dichos pájaros.
- Usar como entrada la señal de audio cruda resulta muy costoso, por lo cual se busca extraer características a partir de dichas señales.
    - Features basados en las señales temporales
    - Features basados en el espectro de la señal
    - Features basados en transformaciones tiempo-frecuencia.

Repasemos algunas nociones del espectro de una señal

- El espectro de una señal nos habla de su contenido de frecuencia.

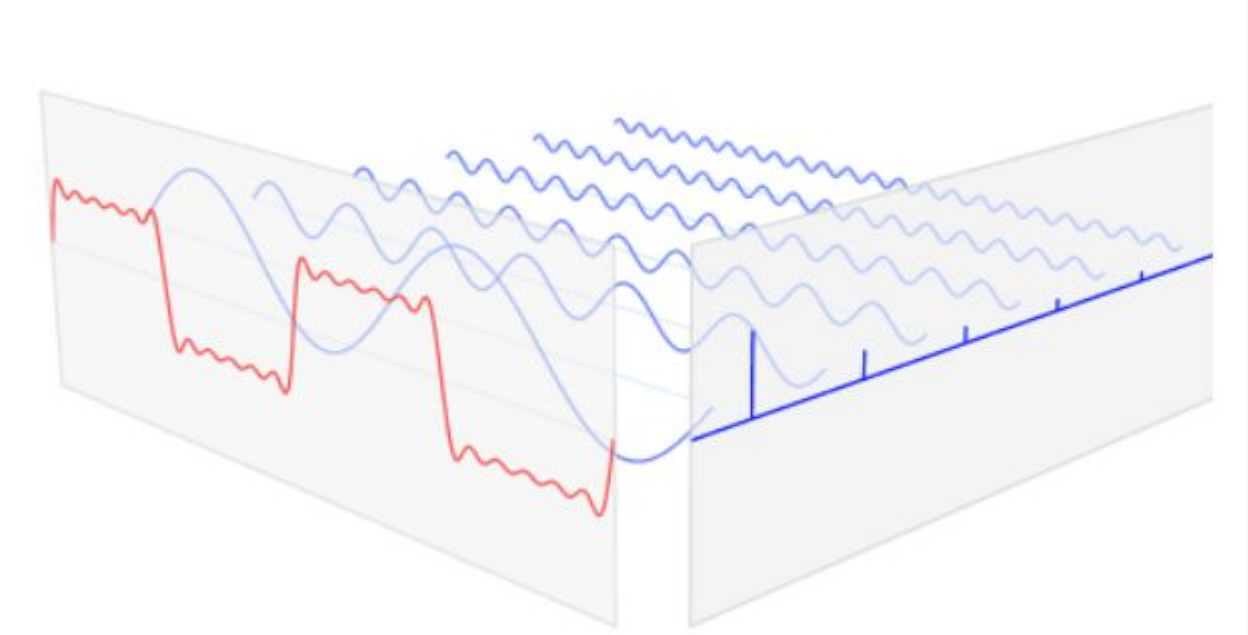




$$x(t)$$

$$X(\omega) = \int_{-\infty}^{\infty} x(t)\, e^{-j\,2\pi\,t\,\omega}\, dt$$

Repasemos algunas nociones del espectro de una señal



DTFT

Si la señal que tenemos es discreta

$$x[n] \overset{FFT}{\Rightarrow} X(\Omega) = \sum_{n=-\infty}^{\infty} x[n]\, e^{-j\Omega n}$$

$$X(\Omega) \overset{IFFT}{\Rightarrow} x[n] = \frac{1}{2\pi} \int_{2\pi} X(\Omega) \cdot e^{i\Omega n} d\Omega$$

DFT

En la computadora no podemos trabajar con señales continuas ➡ muestreo en tiempo y en frecuencia

$$X[k] = \sum_{n=0}^{N-1} x[n] e^{j\,2\pi\,k\,n/N} = \sum_{n=0}^{N-1} x[n] W_N^{kn}$$

$$x[n] = \frac{1}{N} \sum_{k=0}^{N-1} X[k] e^{j\,2\pi\,k\,n/N}$$

Para computar la DFT, se usa la FFT (Fast Fourier Transform), que es una canasta de algoritmos para computar la DFT de forma eficiente.

DFT

Observación: Si *x[k]* se obtuvo al muestrear una señal en tiempo continuo *x(t)*, lo que querríamos aproximar es la TF de *x(t)*. La DFT puede pensarse como una sucesión de muestras equiespaciadas de $X(\omega)$.

Llamando f_s a la frecuencia de muestreo, podemos hacer un mapeo entre las frecuencias 1,..., N con el intervalo de frecuencias $-f_s/2$ y $f_s/2$.

Señales que varían en el tiempo
Transformaciones espacio-tiempo

¿Qué pasa si la señal que estoy analizando cambia su contenido espectral a lo largo del tiempo?

Espectrograma: se basa en calcular la DFT de la señal de a tramos (ventaneo).

El largo y tipo de ventana elegidos

juegan un rol muy importante en la

calidad del espectrograma obtenido

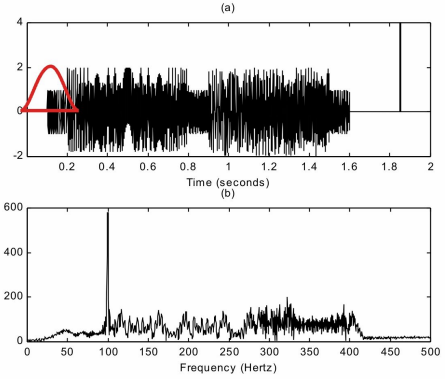
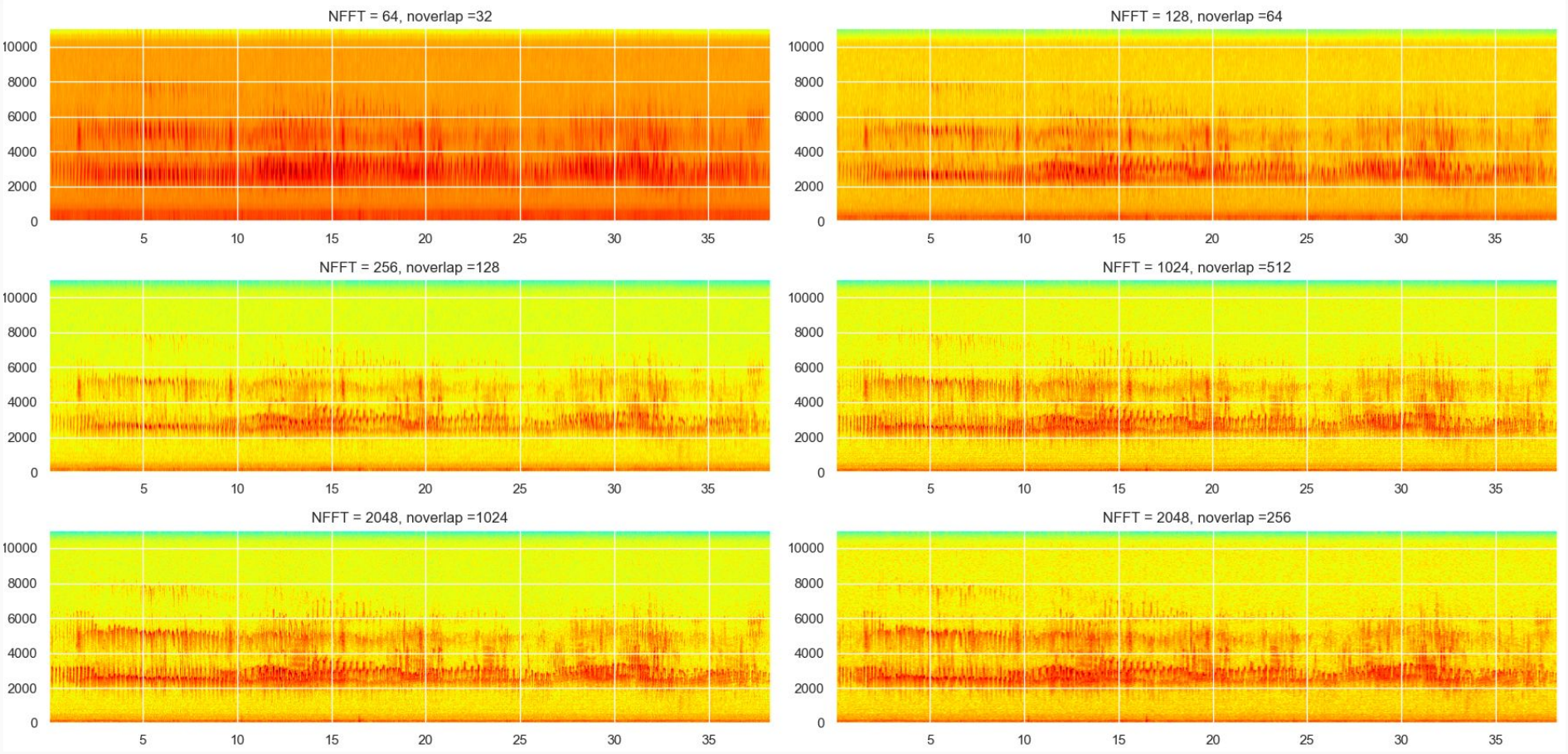


Fig. 9.2 (a) Real part of a multicomponent signal and (b) its Fourier transform

Fig. 9.3 Absolute value of the STFT for the signal depicted in Figure 9.2

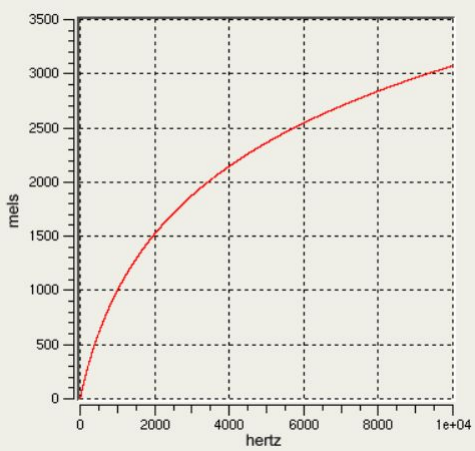
http://home.cc.umanitoba.ca/~thomasg/MP/Research/Time-Frequency/STFT.htm

Espectrograma



Escala MEL

En audio se suele trabajar con lo que se llama escala MEL, que se basa en cómo funciona el oído humano



$$mel = 1127.01048 \ln(f/700 + 1), \quad f[Hz]$$

Asociados a esta escala se encuentran los Mel Frequency Cepstral Coefficients (Coeficientes Cepstrales en las Frecuencias de Mel) o MFCCs, que son coeficientes para la representación del habla basados en la percepción auditiva humana.

MFCCs

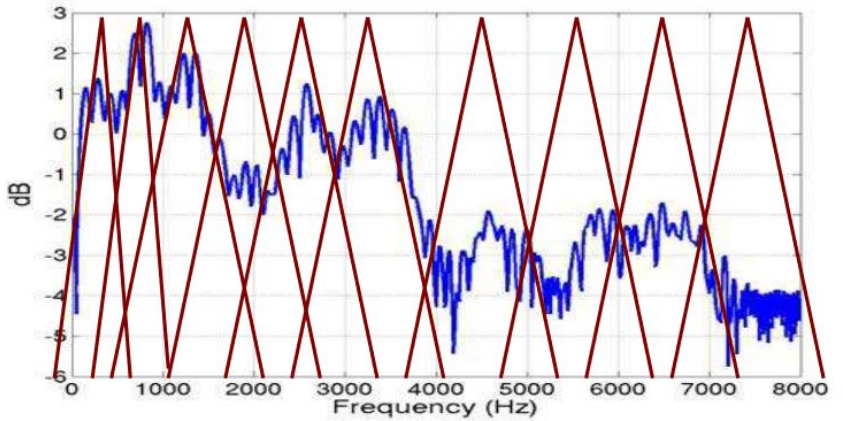
MFCCs se calculan comúnmente de la siguiente forma:

Espectro -> Filtros Mel -> Espectro-Mel

1. Separar la señal en pequeños tramos (ventaneo) y a cada tramo aplicarle la DFT
2. Transformo cada DFT con los filtros MEL, para obtener el espectro MEL

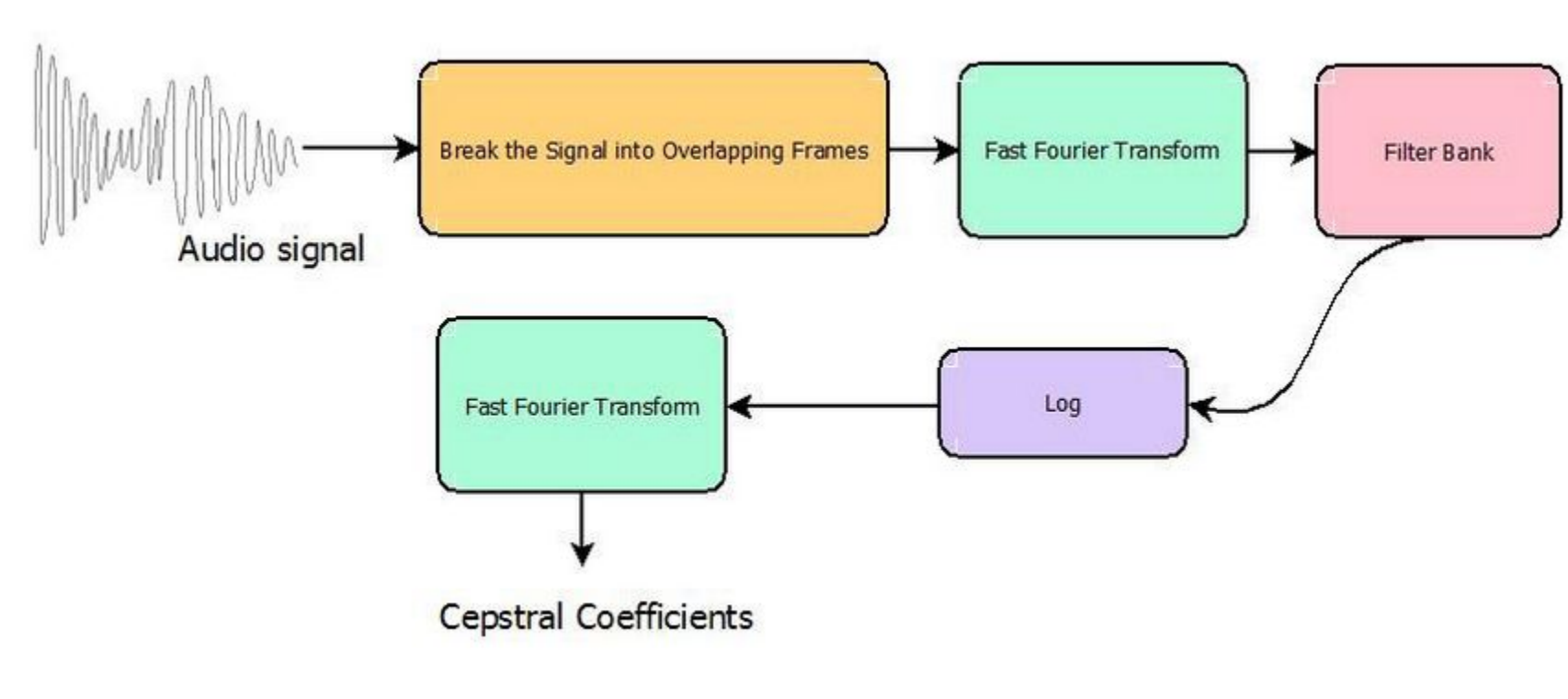
$$\ln(X[k]) = \ln(\text{espectro Mel}) = \ln(H[k]) + \log(E[k]) \overset{IFFT}{\Rightarrow} x[k] = h[k] + e[k]$$

3. Tomo el logaritmo
4. Los h[k] son los coeficientes Mel



http://www.speech.cs.cmu.edu/15-492/slides/03_mfcc.pdf

MFCCs



https://medium.com/prathena/the-dummys-guide-to-mfcc-aceab2450fd

3. Implementación de sistemas de ML

Implementación de sistemas de ML

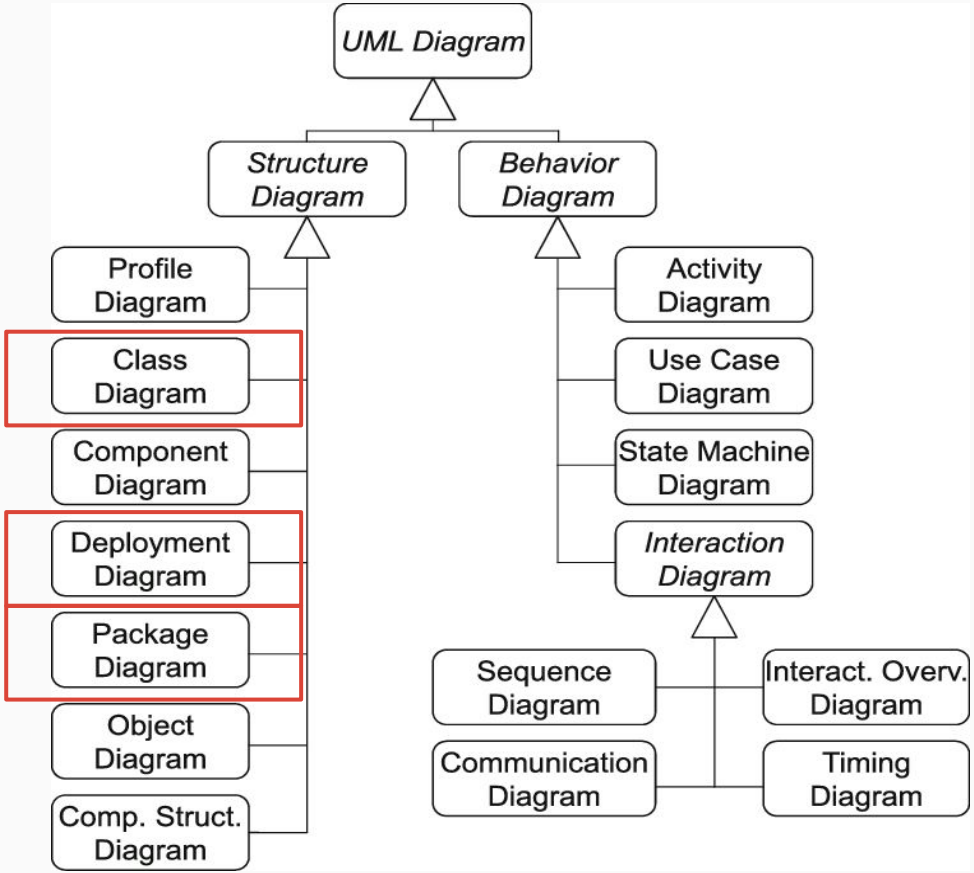
- Un sistema de ML en producción requiere múltiples componentes:
  - Infraestructura.
  - Aplicaciones.
  - Datos
  - Documentación
  - Configuración.
  - Etc.
- La modularización en todos los niveles contribuye a la organización y mantenimiento de sistemas complejos
  - A nivel de componentes de SW
    - Programación orientado a objetos
      - Herencia y polimorfismo
      - UML.
    - Caso de estudio: Scikit learn
  - A nivel de despliegue:
    - Arquitecturas de microservicios
- Ejemplo completo con microservicios (docker).

Herencia y polimorfismo

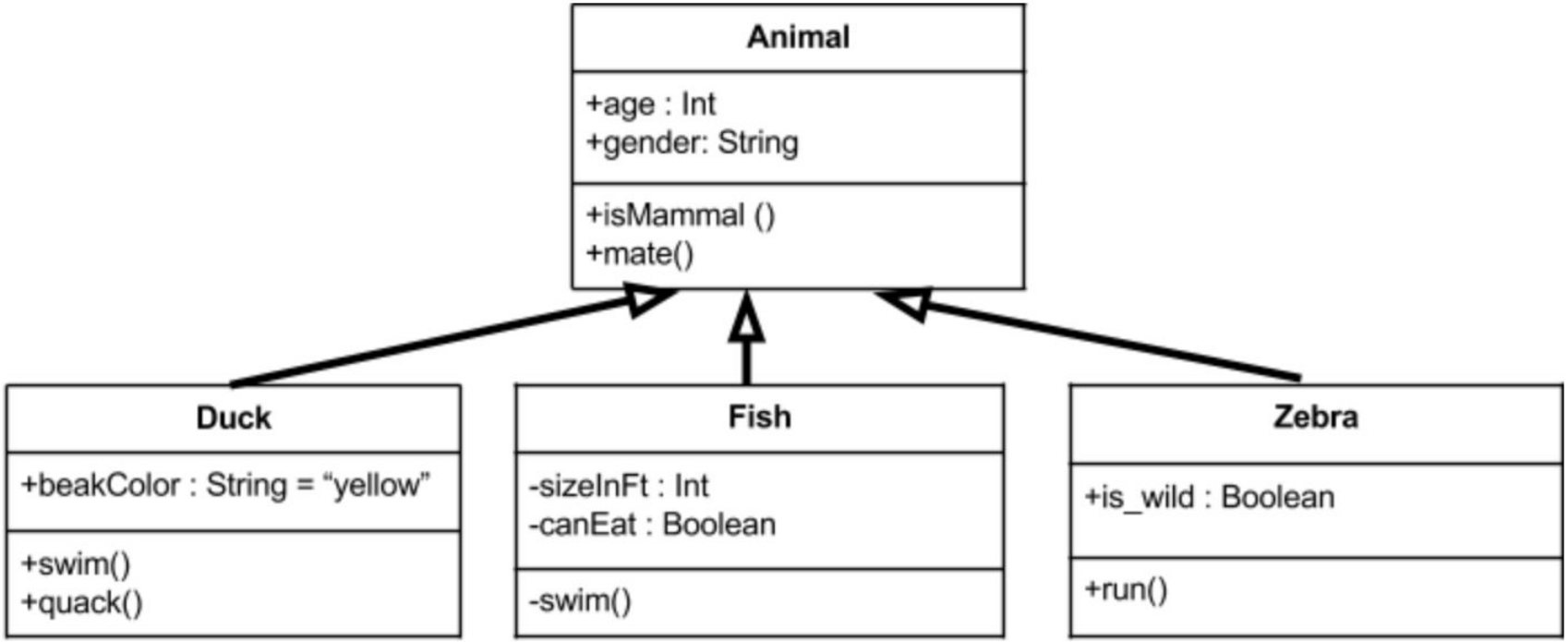
- Hemos introducido algunos conceptos de POO, herencia y polimorfismo en la clase 1.
  - **Herencia**: mecanismo por el cual una clase se deriva de otra para extender su funcionalidad. La clase de la que se hereda se suele denominar clase base/padre/superclase.
  - **Polimorfismo**: propiedad por la que es posible enviar mensajes sintácticamente iguales a objetos de tipos distintos. El único requisito que deben cumplir los objetos que se utilizan de manera polimórfica es saber responder al mensaje que se les envía.
- Estos conceptos serán de interés para la composición de cadenas de procesamiento.

Unified Modelling Language (UML)

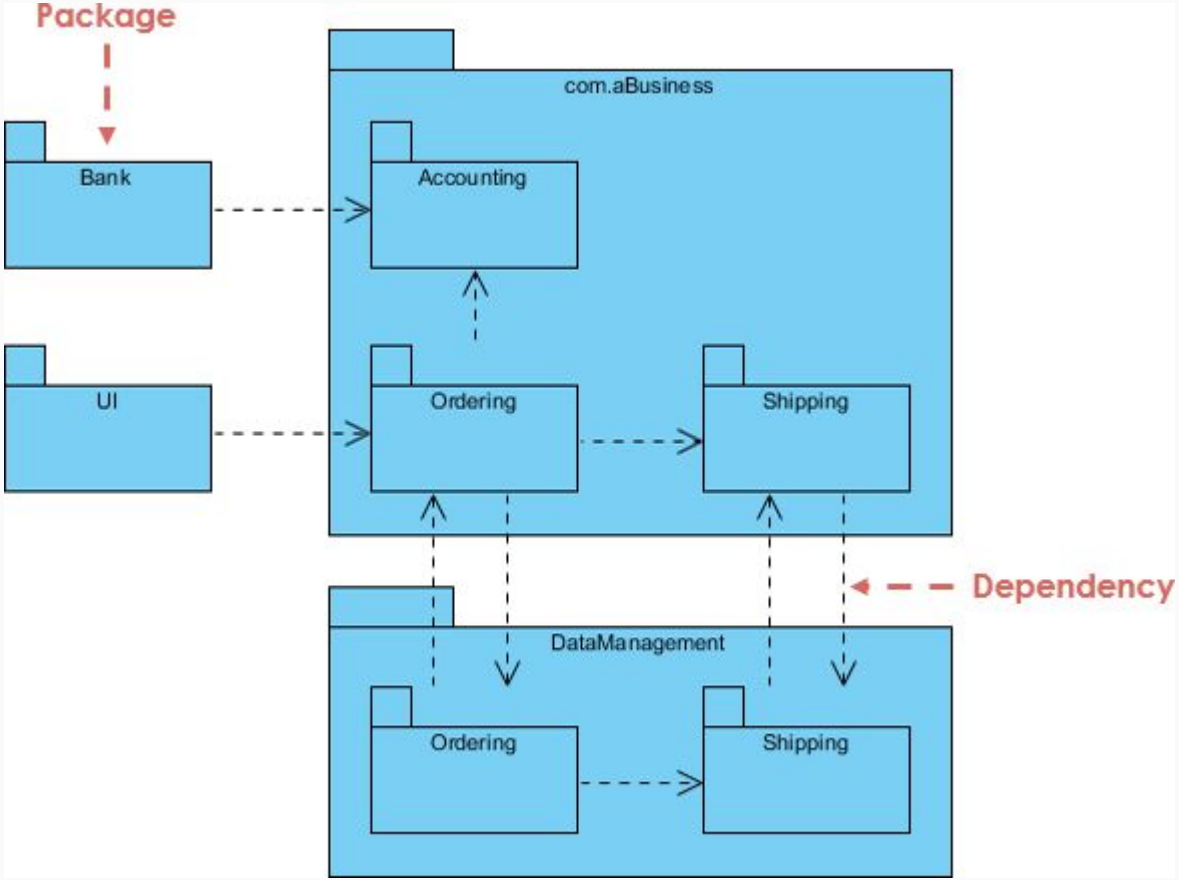
- Estándar para crear esquemas, diagramas y documentación relativa a desarrollos de SW, particularmente orientado a POO.
- Proyecta las diferentes **vistas** de un sistema.
- Organiza los diagramas en dos grandes grupos:
  - **Estructurales**
  - **De comportamiento**



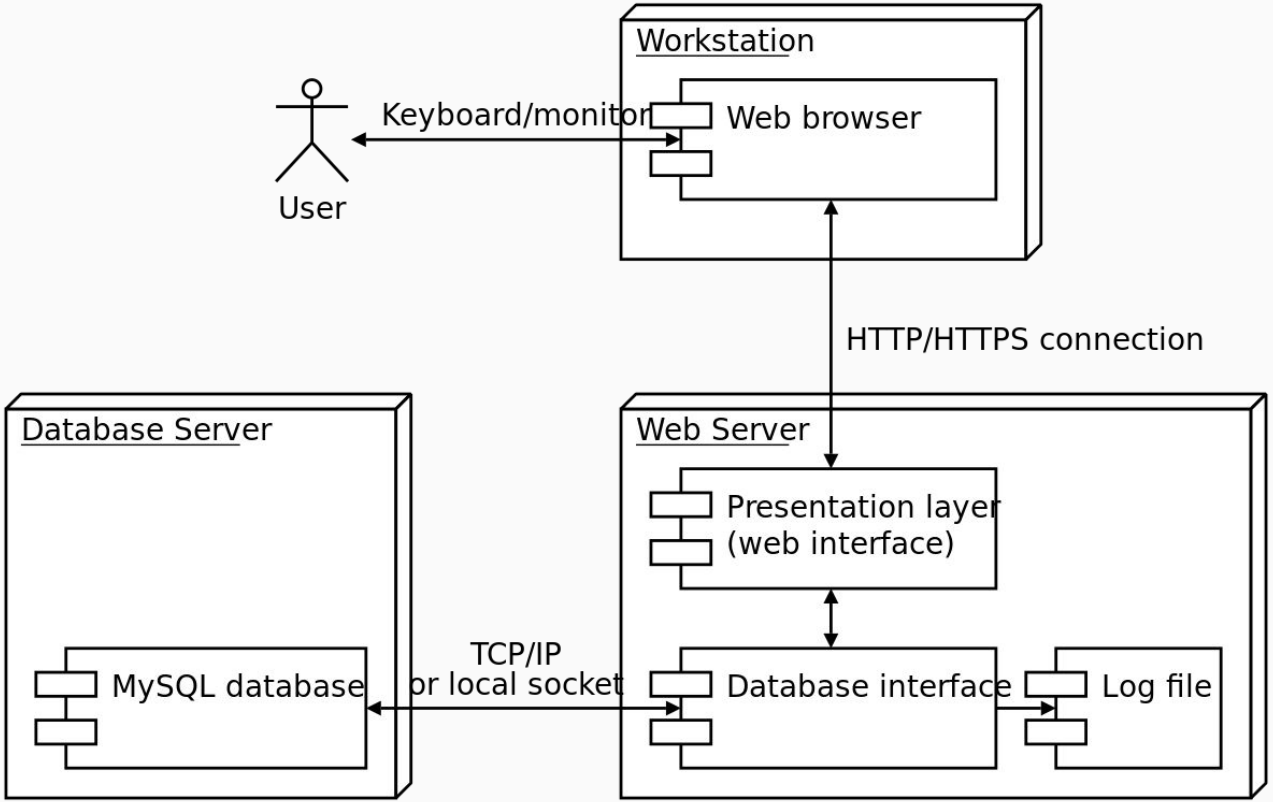
UML. Diagramas de clase



UML. Diagramas de paquetes



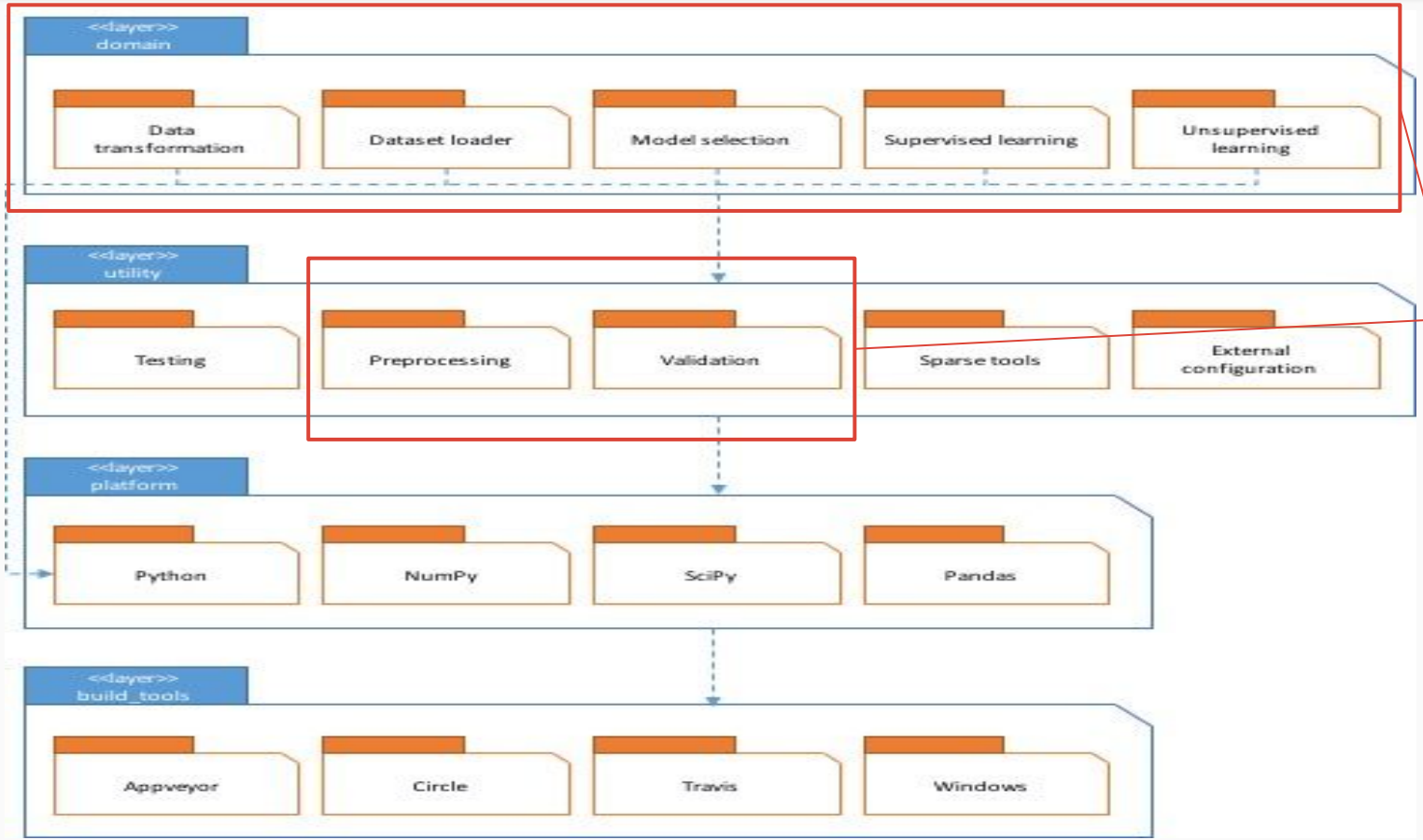
UML. Diagramas de despliegue



Caso de estudio: Scikit Learn

- Inicialmente desarrollada por David Cournapeau como proyecto de Google Summer of code en 2007.
- ¿Qué problema intenta resolver y cómo presenta una interfaz a los usuarios?
  - *“Simple and efficient tools for predictive data analysis · Accessible to everybody, and reusable in various contexts”.*
  - Es un buen ejemplo de cómo diseñar una librería de SW que exponga una interfaz sencilla, y al mismo tiempo sea mantenible, escalable, y flexible.

Organización de Scikit-Learn



Más utilizados en este curso

Interfaces

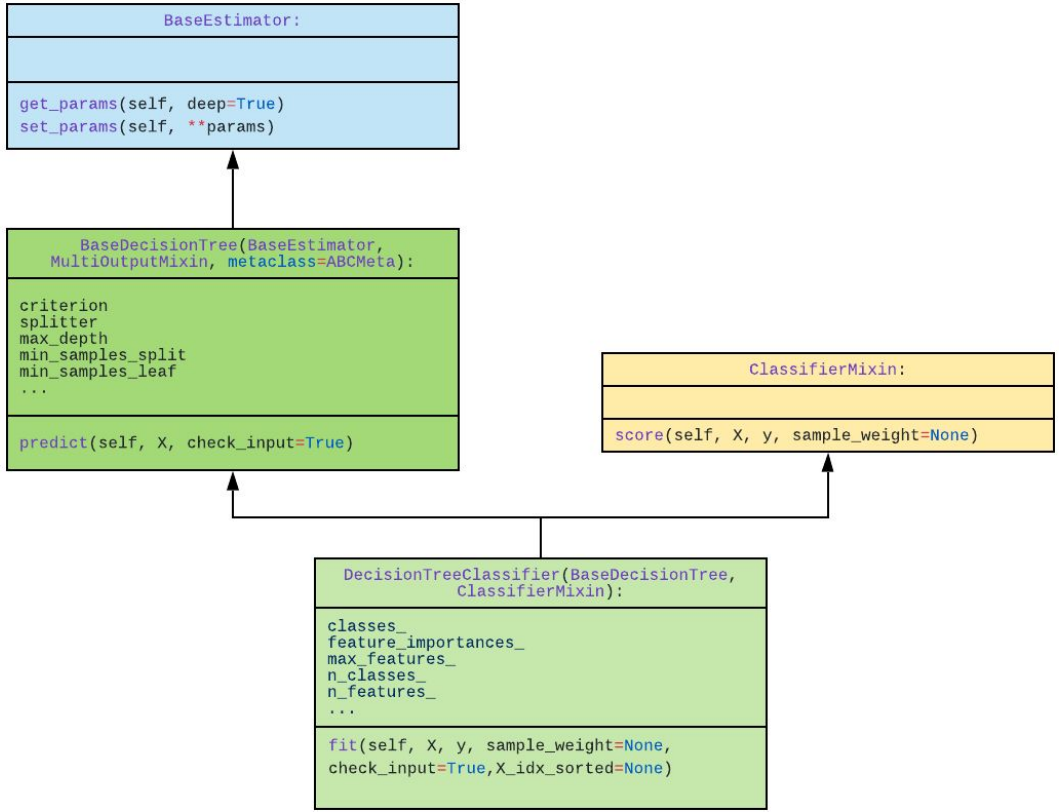
- La librería se organiza en torno a tres interfaces (APIs) fundamentales:
  - **Estimators**
  - **Predictors**
  - **Transformers**
- Estas interfaces son complementarias, y por lo tanto pueden superponerse. Un algoritmo clasificador, como por ejemplo *DecisionTreeClassifier*, puede ser simultáneamente un **estimador** y un **predictor**.

Estimators

- Exponen el método *fit()* para ajustar parámetros a un set de datos.
- La definición de parámetros (HPs) está separada de su ajuste de parámetros. Es un ejemplo de aplicación de funciones parciales, que en este caso permite mantener la misma firma para el método fit:
  - model = MyModel(param1,param2,...)
  - fitted_model = model.fit(X_train,y_train).

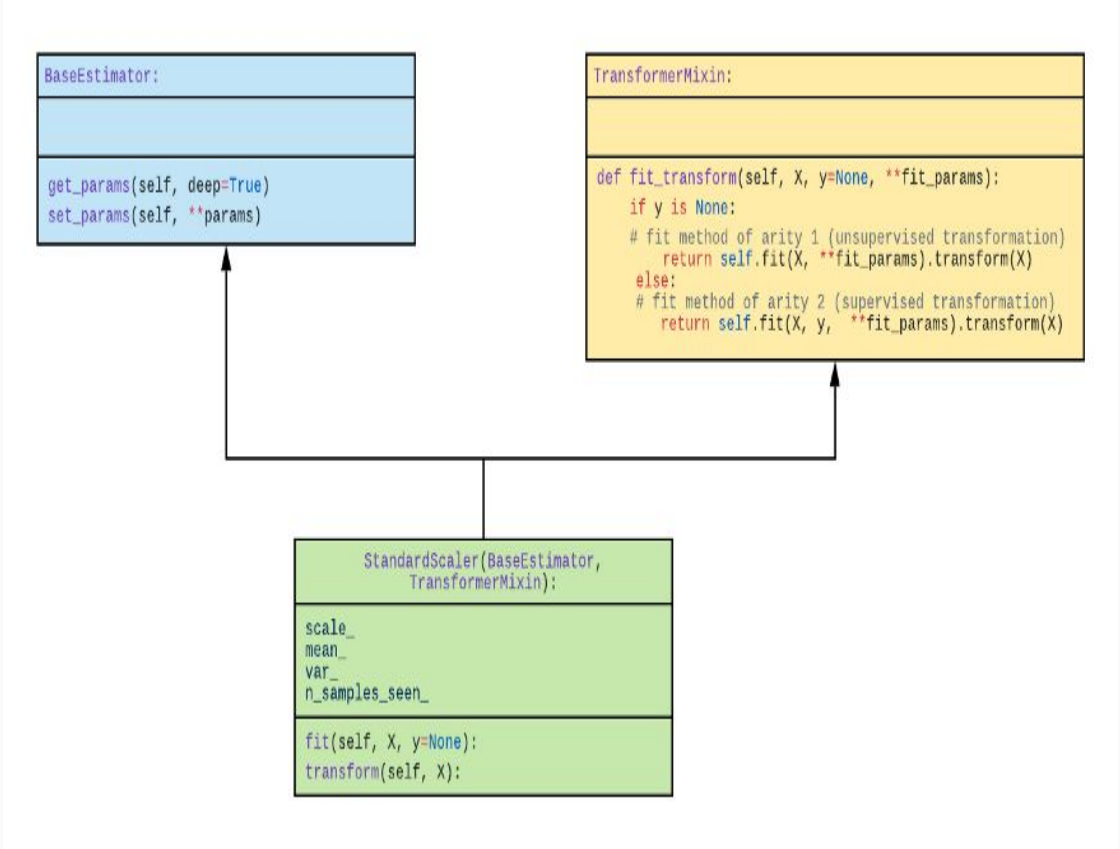
Predictors

- Extienden la funcionalidad de un Estimador agregando la capacidad de predecir, es decir, agregan:
  - predict()/fit_predict()



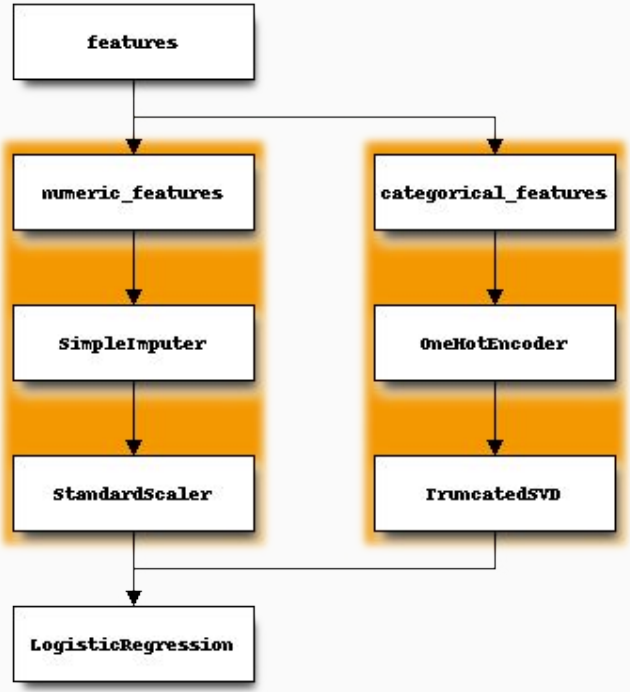
Transformers

- Cumplen la función de aplicar una transformación a los datos de entrada.
- Observar que en algunos casos, pueden también ser un estimador (por ejemplo StandardScaler(), o las transformaciones BoxCox).

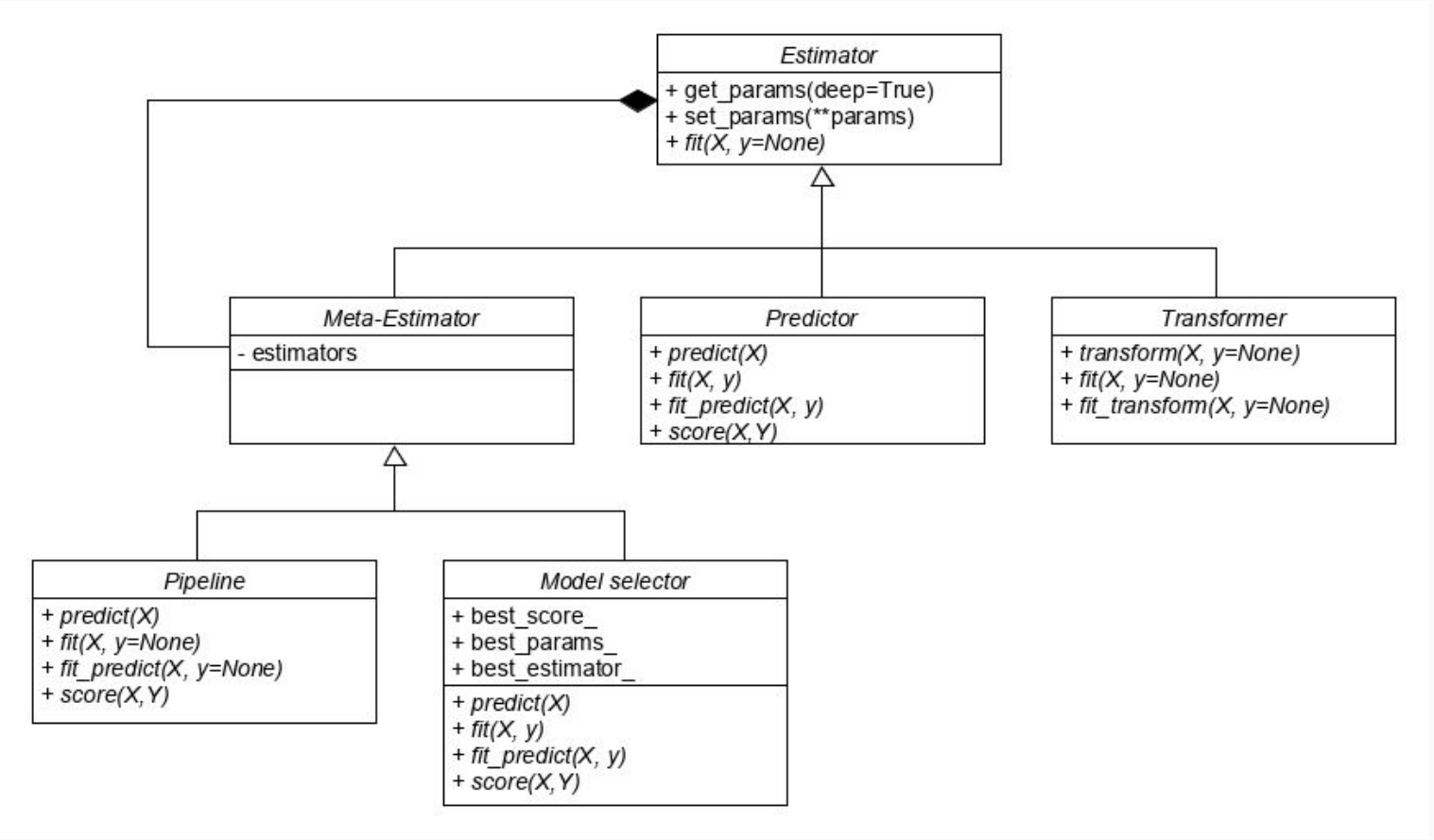


Composición de estimadores: Pipelines

- Organizar la API a partir de ***Estimadores***, ***Transformadores*** y ***Predictores*** permite encadenar estas operaciones para definir cadenas de procesamiento complejas.

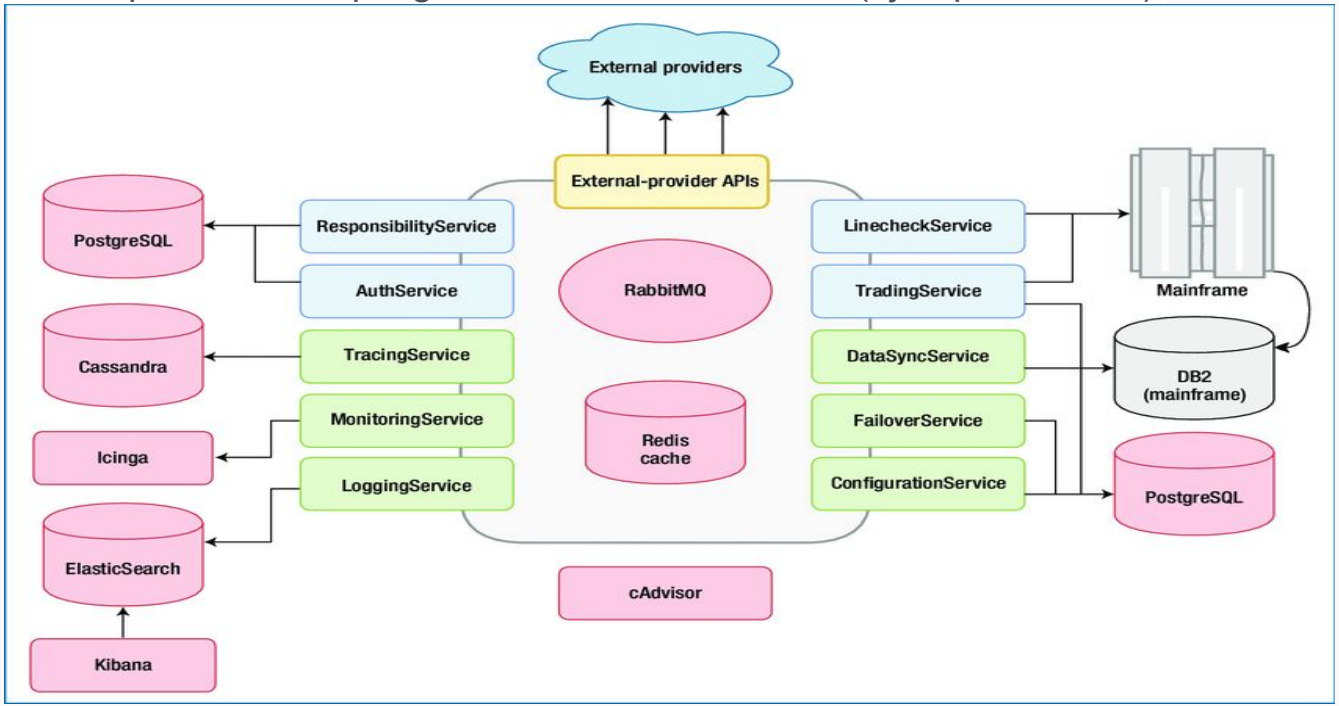


Jerarquía de clases: Estimator, Predictor, Transformer y cadenas.

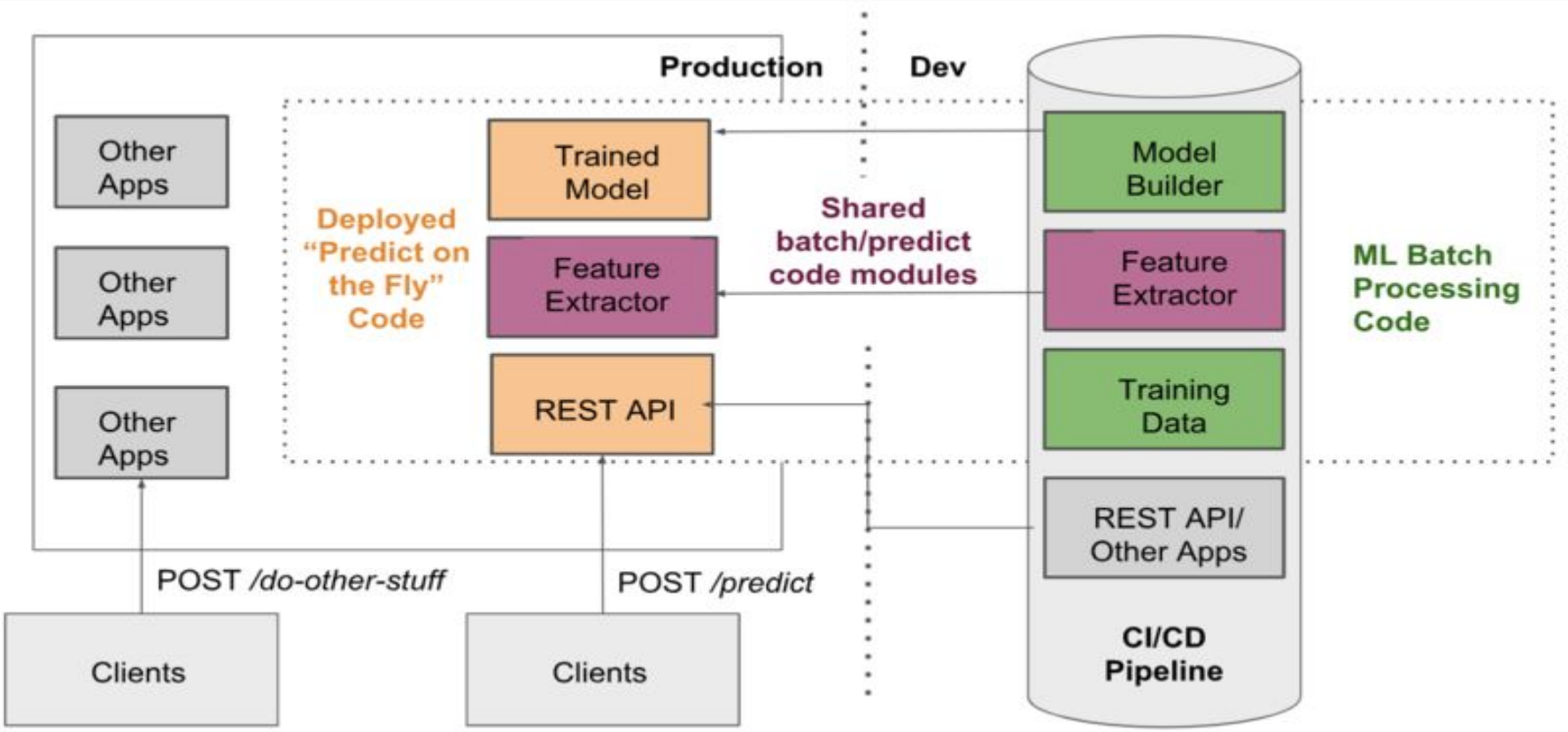


Despliegue con microservicios

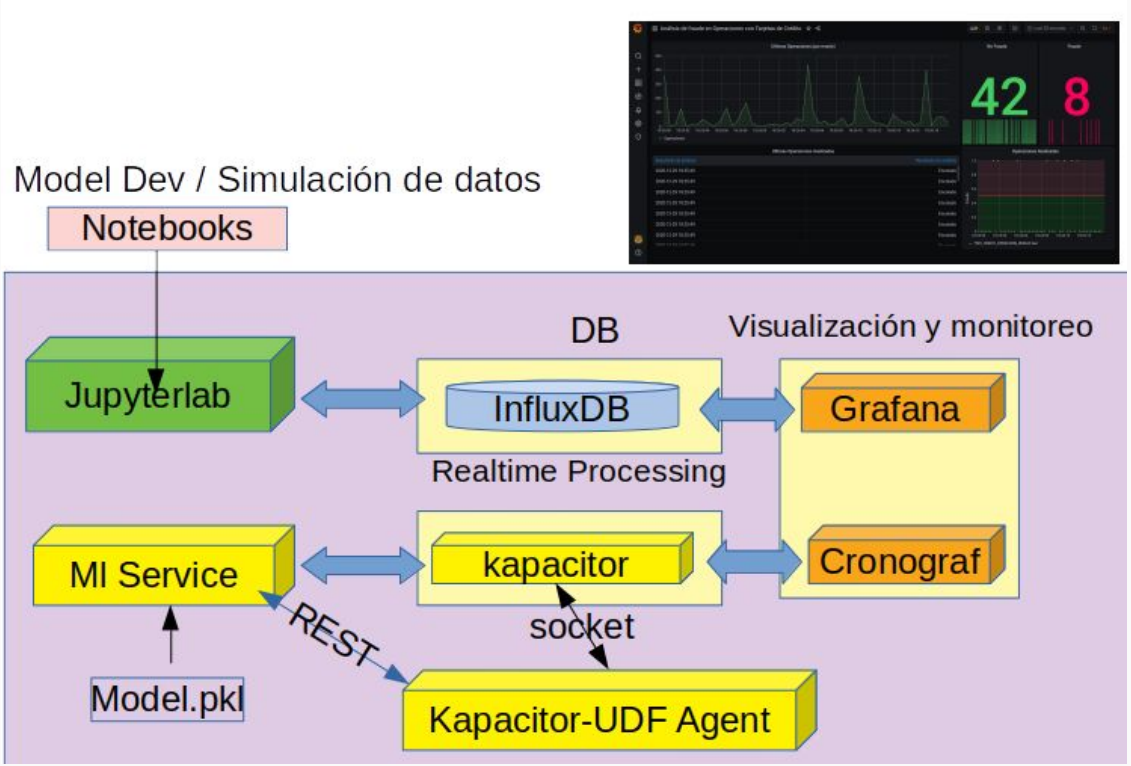
- Es una aproximación para el desarrollo de software que consiste en construir una aplicación como un conjunto de pequeños servicios, los cuales se ejecutan en su propio proceso y se comunican con mecanismos ligeros (por ejemplo REST, protobuf, Kafka RabbitMQ, etc.)
- Una forma adoptada de despliegue es con contenedores (ejemplo: docker).



Despliegue


Production
Dev
Other Apps
Other Apps
Other Apps
Trained Model
Feature Extractor
REST API
Deployed "Predict on the Fly" Code
Shared batch/predict code modules
Model Builder
Feature Extractor
Training Data
REST API/ Other Apps
ML Batch Processing Code
CI/CD Pipeline
POST /do-other-stuff
POST /predict
Clients
Clients

Ejemplo



Ver proyecto completo en repositorio: código (notebooks) + despliegue con docker

Referencias

- Diseño de Scikit-Learn
  - https://github.com/gopala-kr/10-weeks/blob/master/Projects-Blogs/06-ml-dl-frameworks/scikit-learn-architecture.md
  - https://towardsdatascience.com/scikit-learn-design-principles-d1371958059b