Universitat
de les Illes Balears

Machine Learning

Lesson 6: Productization

Machine Learning in the context of Artificial Intelligence

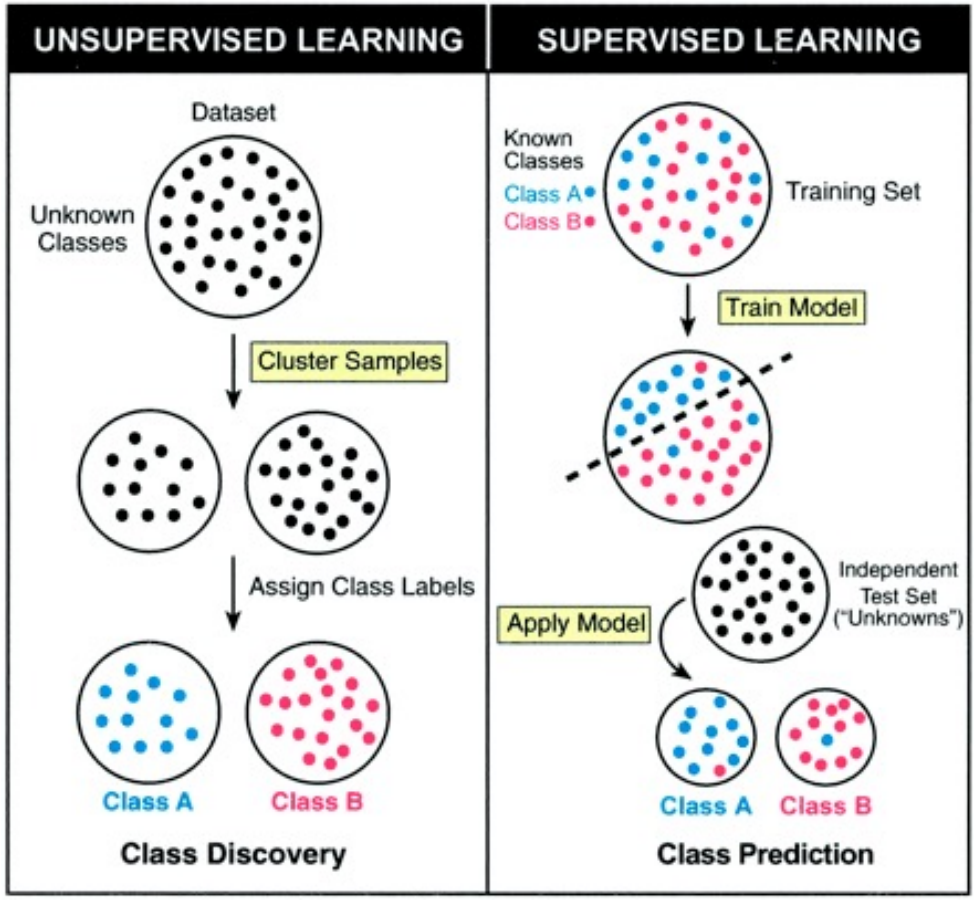
Learning main types

Supervised

Data is provided (through a "master") as pairs of cases-labels, indicating whether or not they belong to the type of association to be learned (classification, regression). Practical case: prediction

Unsupervised

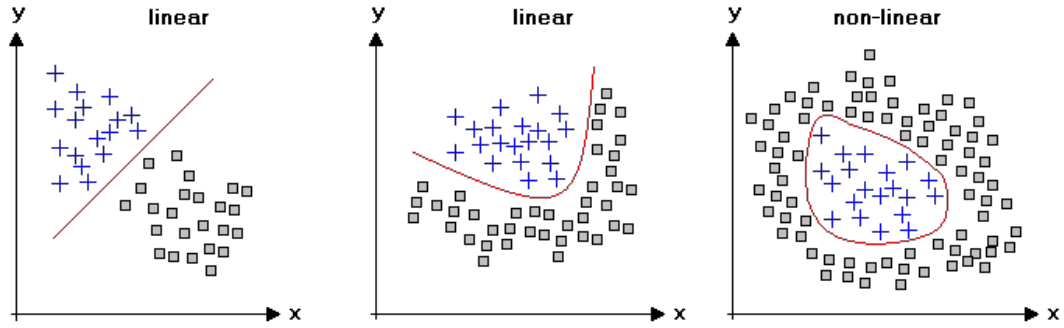
The algorithm must use other funds to obtain a "feedback" that tells him whether he does it well or not, since he does not have a "master" (clustering). Practical case: segmentation



Task types

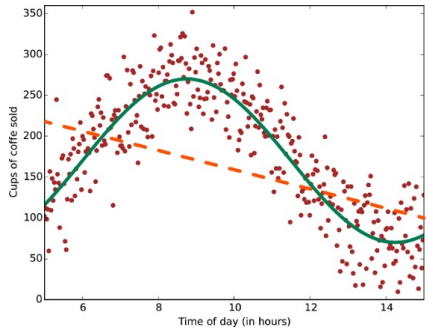
Classification

Classification algorithms are used when the desired output is a discrete label.



Regression

Approximate a continuous function (predicting outputs that are continuous)



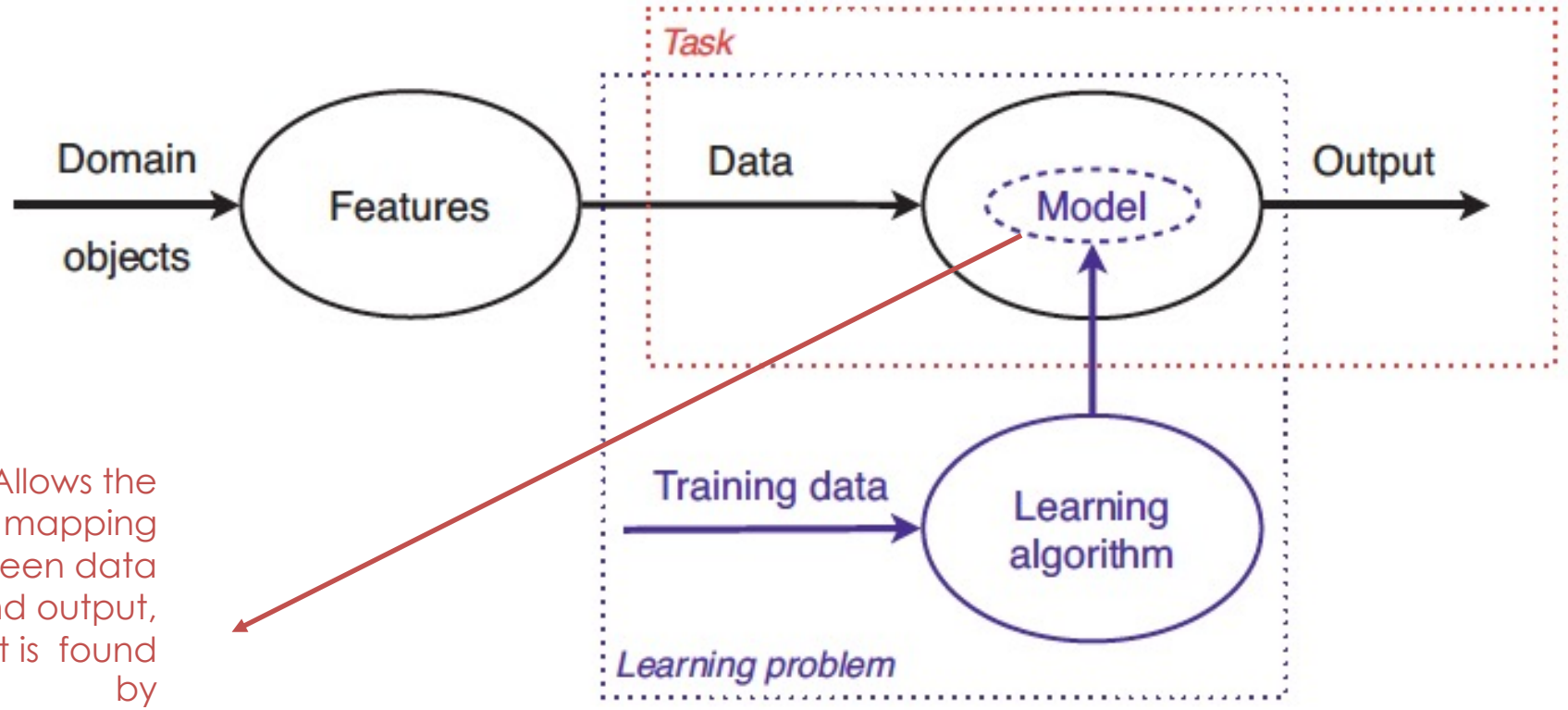
Useful to remember…

Most of the value will come from supervised models !

- Do I need ML ?
  - Do not underestimate the power of simple solutions
    - Filter and aggregate are also your tools
  - Always compare your ML model against a baseline:
    - Mean / Median imputation
    - Linear Regression

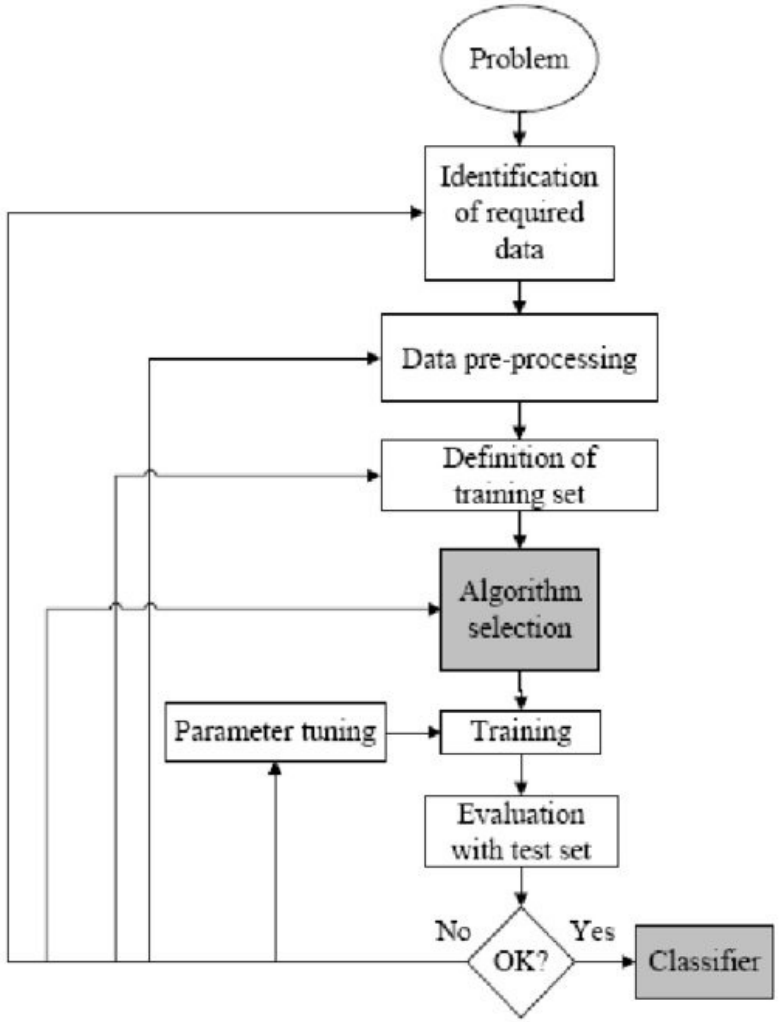
- How can I solve a given business problem using a classification or a regression
  - Try to the possible extent to work under a "supervised" framework

Task: machine learning model



Allows the mapping between data and output, and it is found by computational learning algorithms

How to develop a ML solution



Framing the problem:

From Real world problem to ML problem

Getting the data

Do I have data to solve this problem?

Explore the data

Is the data representative?

Train and select the model

Model selection and feature engineering

Deploy

How it will be consumed?

ML in production

A model which does not work on production is worth nothing

Always take into account…

- Don't use data that wouldn't be available in production when prototyping
  - you won't be able to make predictions

- After iterative prototyping stage → productization
  - Notebooks: good for prototyping, bad for productization
  - Notebook productization: **papermill**

- Productization needs:
  - Real time response ? (web app)
  - Offline predictions --> volumetry ? Time to get predictions ? Python ?

ML in production

A model which does not work on production is worth nothing

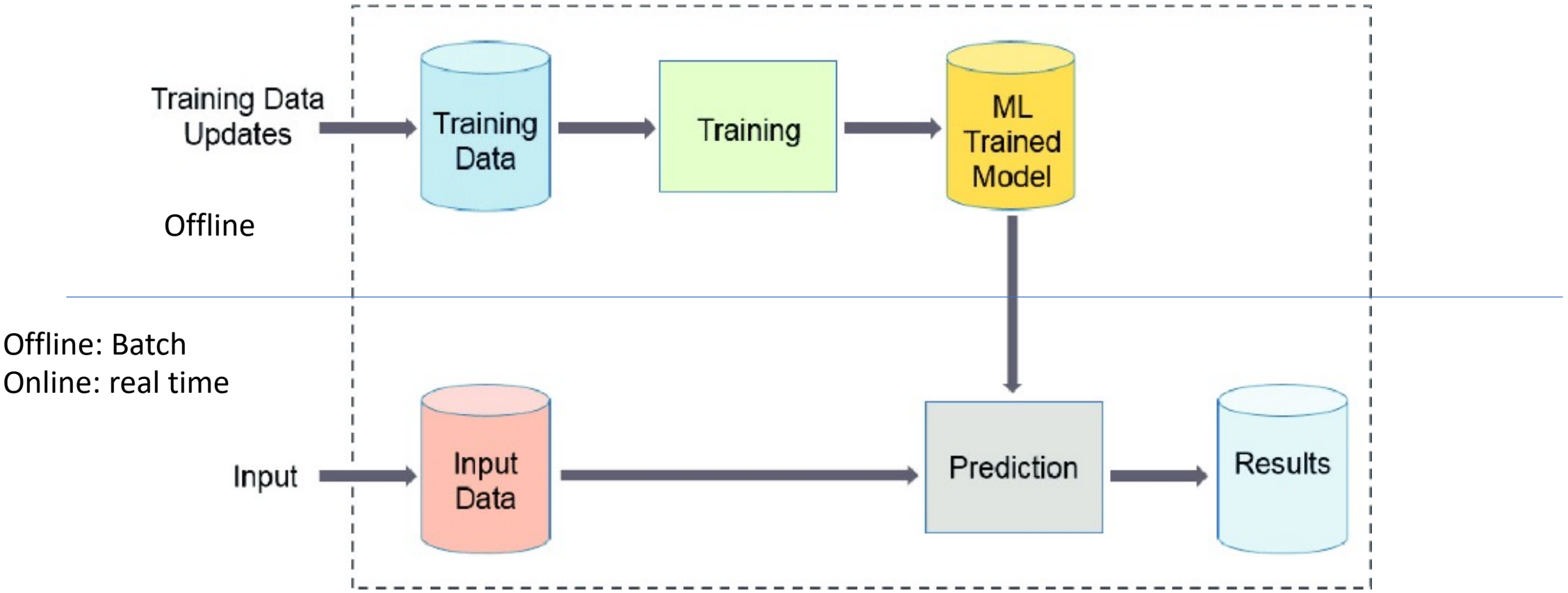
- Enterprise maturity ?
  - Do I have a ML platform ?
  - Do I need to build it from scratch ?
  - Can I leverage managed solutions ?
    - sagemaker, google cloud, azure, …

- Model Monitoring ?
  - Is my model behaving correctly ?
  - Do I need to retrain it?
    - When, how often…how ?
    - Can it be automated ?
    - Can I do A/B testing in my current framework ?
  - How do I monitor it ?

ML in production

- We as ML engineers / Data Scientist will normally seat in the middle of a cross-disciplinary team in order to get things done !
- Understand the data
- Understand the modelling process
- **Understand from minute 0 how the model is going to be used and plan ahead for it**

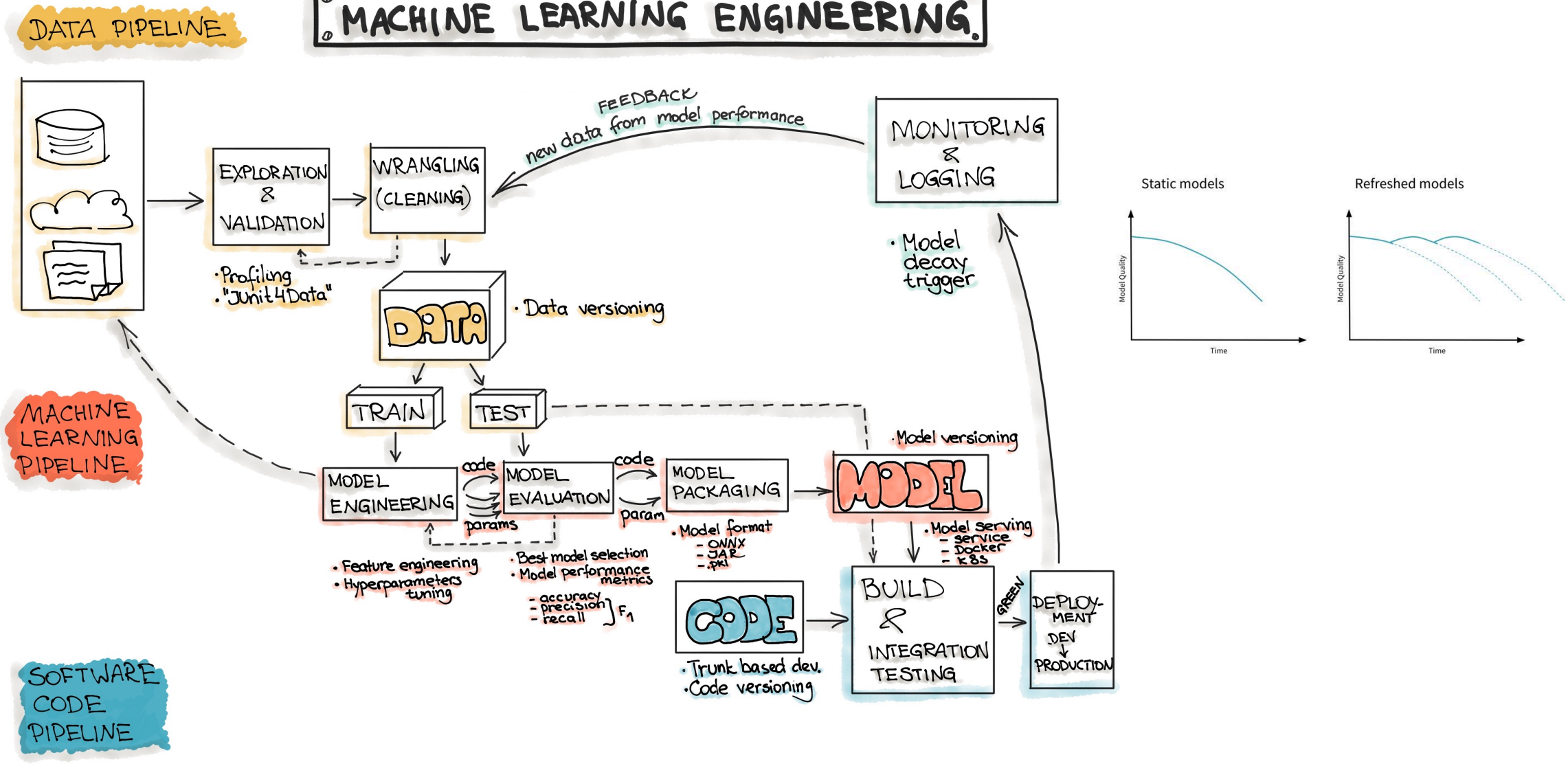
Basic ML pipeline



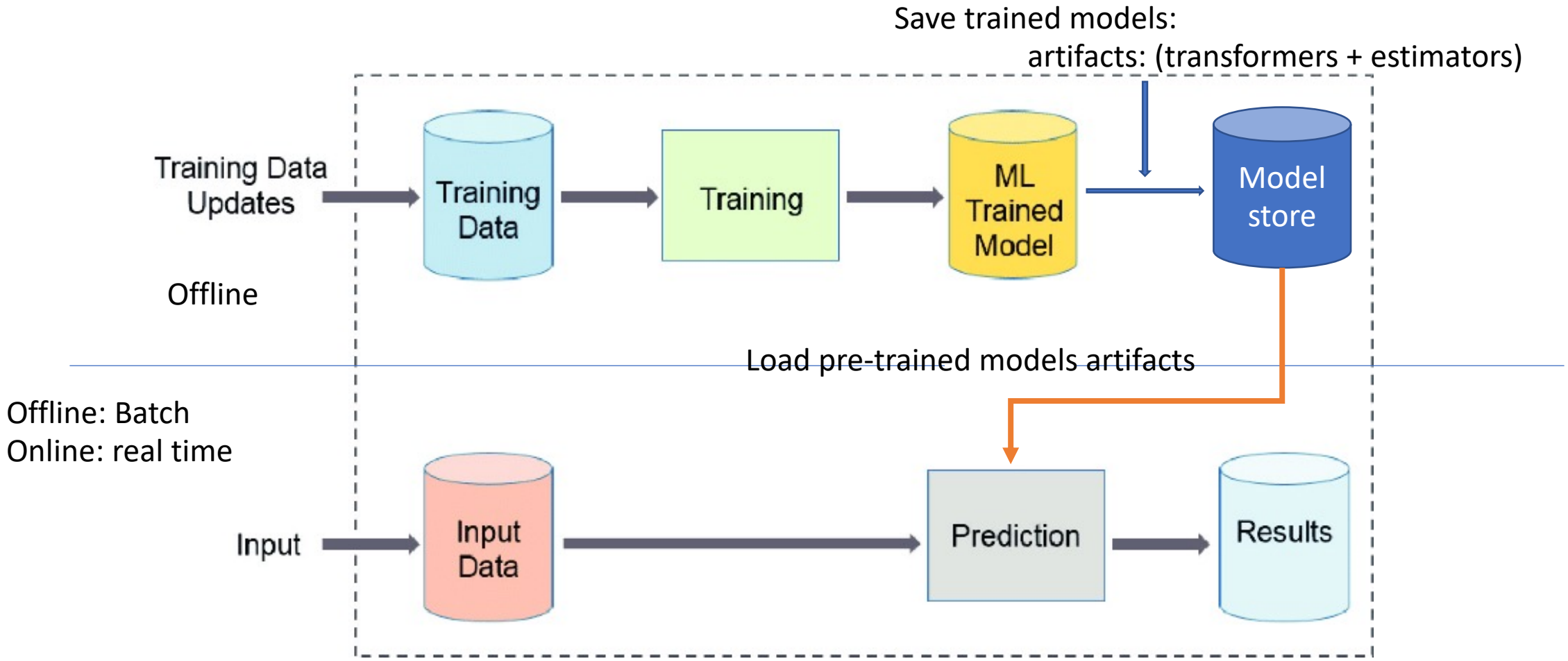
Two basic productization ways

- Batch prediction
  - Inference on large datasets
  - Can be “easily” scaled
  - More time allocated for predictions:
    - If I need fresh predictions every morning, can set up a night batch reprocess the data
- Real time prediction
  - Inference on small datasets via http
  - Need an endpoint to send requests
  - API-REST
    - Load ?

MACHINE LEARNING ENGINEERING



Basic ML pipeline



Artifacts

- **Models take into account:**
  - Transformers → OHE, PCA, …
  - Estimators → Random Forest, linear regressions…

- **How to persist the "trained" state of a model**
  - **Python**
    - **Pickle:**
      - Serializing and de-serializing Python object structures → process of converting an object in memory to a byte stream can then be retrieved and de-serialized back to a Python program
      - OJO!
        - Only works with Python
        - It is environment specific → Pickle de-serialization will need the same environment from which was serialized
    - What if not python…

Artifacts

- **What if not python…**
  - Pre processing stages may not be easily shared and will need to be re-written in chosen language
  - Cross-language standards to store models
    - If it is a linear/logistic regression → just need to save the weights in any form and load them into any language (easy cheese)
    - ONNX the Open Neural Network Exchange format, is an open format that supports the storing and porting of predictive model across libraries and languages. Most **deep learning librarie**s support it and sklearn also has a library extension to convert their model to ONNX's format.
    - PMML or Predictive model markup language, is another interchange format for predictive models. Like for ONNX sklearn also has another library extension for converting the models to PMML format. It has the drawback however of only supporting certain type of prediction models.
- **It is worth to leverage other options such as:**
  - Using pub/sub messaging systems
  - Containerize the whole application and publish it as a microservice

Project considerations

- Knowledge data discovery and data modelling is by definition an iterative process.
- Notebooks are for data discovery, prototyping and communication
- Productization of Machine Learning models are development projects
- Data science code quality is about correctness and reproducibility

use [cookiecutter](#) and adapt it to your needs!

Cookiecutter Data Science

A logical, reasonably standardized, but flexible project structure for doing and sharing data science work.

```
├── LICENSE
├── Makefile           <- Makefile with commands like `make data` or `make train`
├── README.md          <- The top-level README for developers using this project.
├── data
│   ├── external       <- Data from third party sources.
│   ├── interim        <- Intermediate data that has been transformed.
│   ├── processed      <- The final, canonical data sets for modeling.
│   └── raw            <- The original, immutable data dump.
│
├── docs               <- A default Sphinx project; see sphinx-doc.org for details
│
├── models             <- Trained and serialized models, model predictions, or model summaries
│
├── notebooks          <- Jupyter notebooks. Naming convention is a number (for ordering),
│                         the creator's initials, and a short `-` delimited description, e.g.
│                         `1.0-jqp-initial-data-exploration`.
│
├── references         <- Data dictionaries, manuals, and all other explanatory materials.
│
├── reports            <- Generated analysis as HTML, PDF, LaTeX, etc.
│   └── figures        <- Generated graphics and figures to be used in reporting
│
├── requirements.txt   <- The requirements file for reproducing the analysis environment, e.g.
│                         generated with `pip freeze > requirements.txt`
│
├── src                <- Source code for use in this project.
│   ├── __init__.py    <- Makes src a Python module
│   │
│   ├── data           <- Scripts to download or generate data
│   │   └── make_dataset.py
│   │
│   ├── features       <- Scripts to turn raw data into features for modeling
│   │   └── build_features.py
│   │
│   ├── models         <- Scripts to train models and then use trained models to make
│   │   │                 predictions
│   │   ├── predict_model.py
│   │   └── train_model.py
│   │
│   └── visualization  <- Scripts to create exploratory and results oriented visualizations
│       └── visualize.py
│
└── tox.ini            <- tox file with settings for running tox; see tox.testrun.org
```

Final considerations

Data Transformations are considered as a DAG

Test set is kept away from modelling

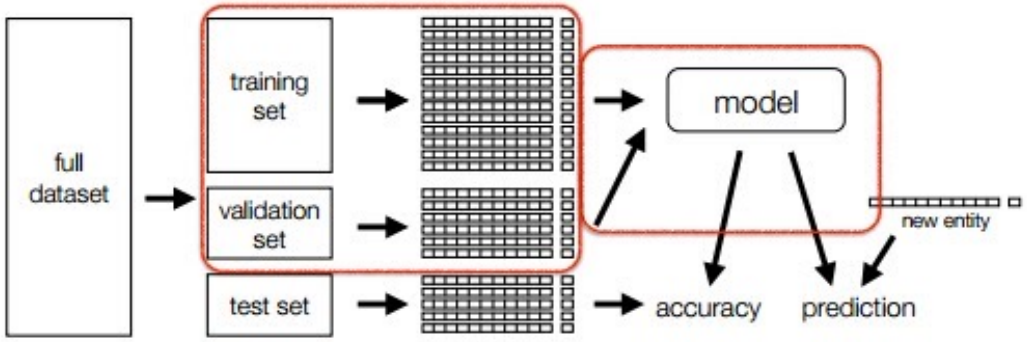
Train set is subdivided in train & validation

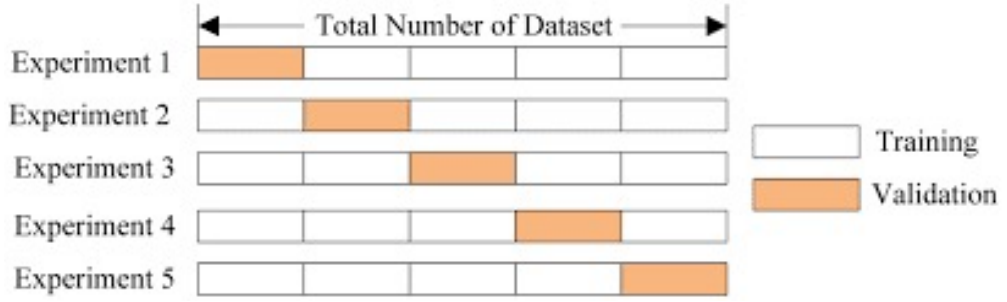
5-fold cross validation has been used for model hyperparameter tuning

A unique metric should be used to compare model performance:

Class: AUC-ROC, neg_log_loss

Regression: MSE, R2

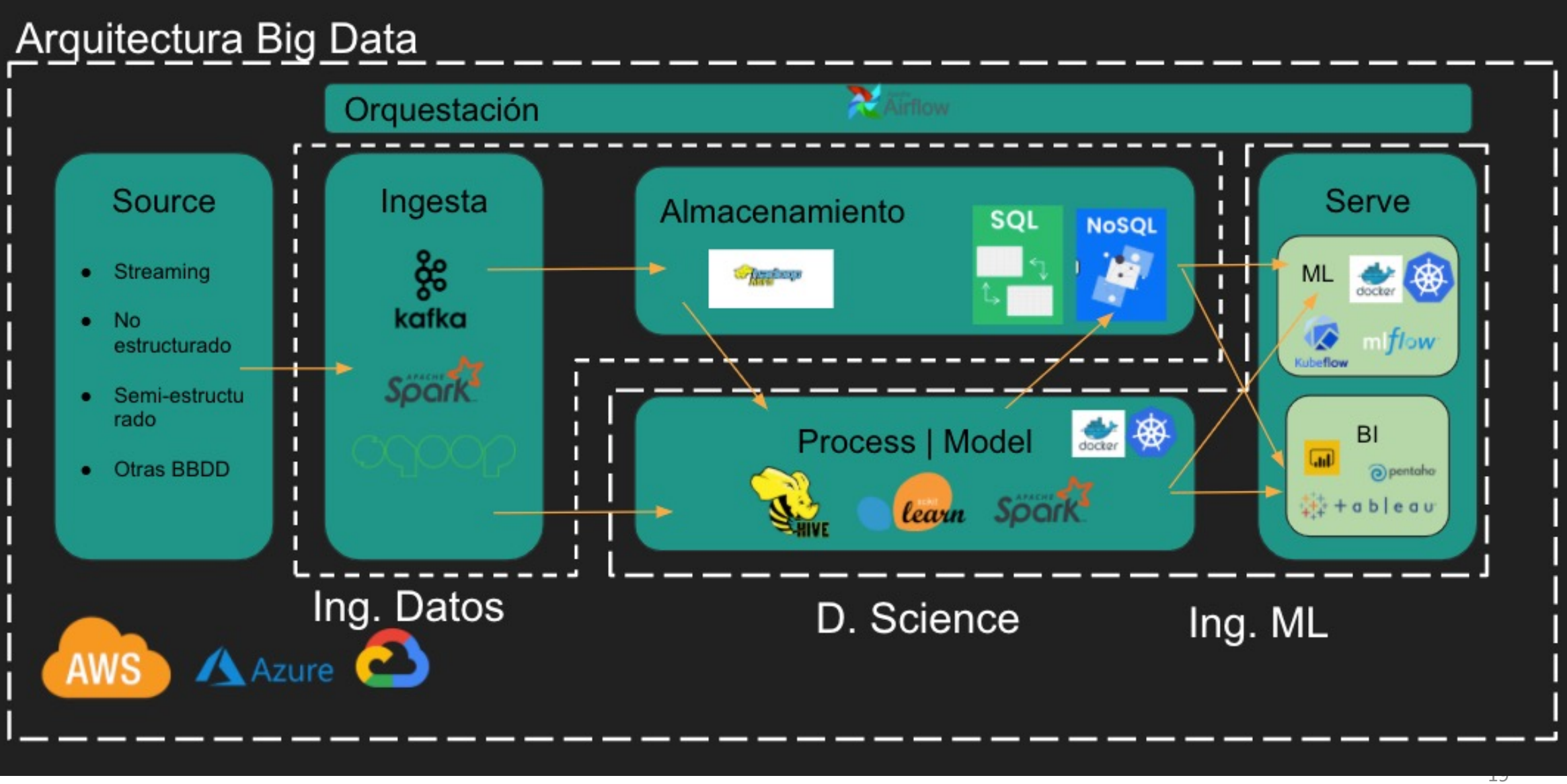




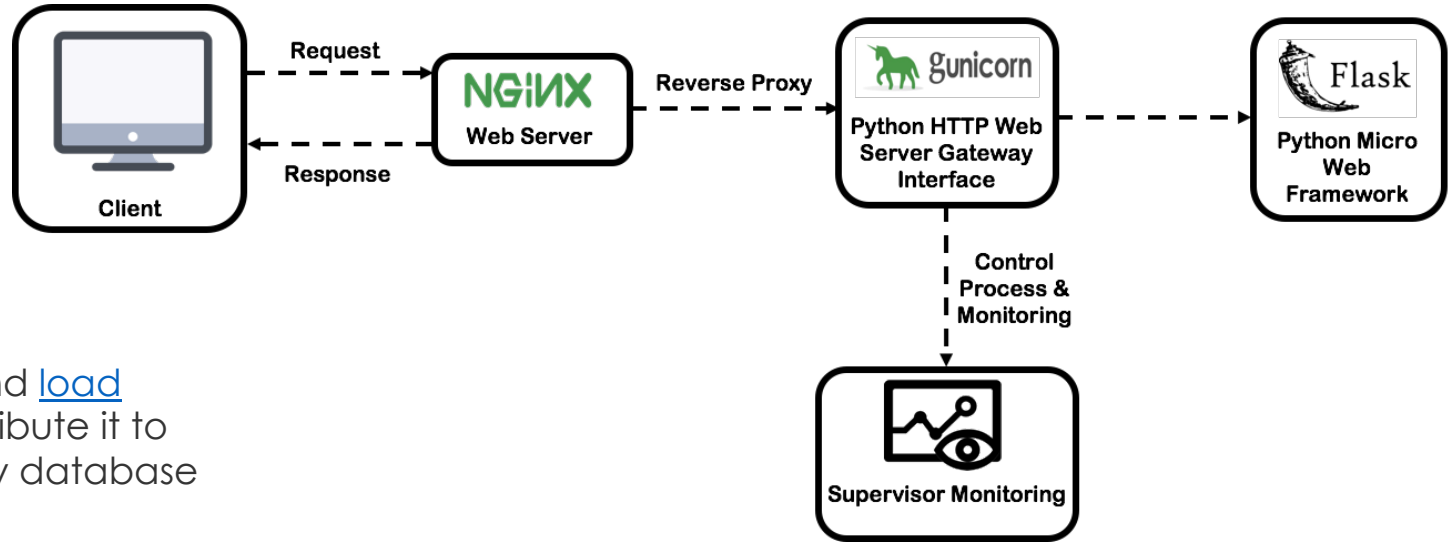
What should I know / learn…

- SQL
- Start thinking in OOP and SOLID
- Git: version control for code
- DVC: version control for data
- Docker:
  - Docker is a tool which helps to create, deploy, and run applications by using containers in a simpler way.
  - The container helps to **package up** an **application with all of the parts it needs**, such as libraries and other dependencies, **and deploy it as one package**.
- Model database:
  - Modeldb
  - mlflow
- Model deployment:
  - kubeflow
  - Mlflow
  - Airflow

Modern Data Application: de la fuente al consumo



A basic api-rest



NGINX is commonly used as a reverse proxy and load balancer to manage incoming traffic and distribute it to slower upstream servers – anything from legacy database servers to microservices.

WSGI (Web Server Gateway Interface)server. This way, you deployment will be more stable, be able to handle more requests at once and be fast about it. The application code does not care about anything except being able to process single requests.

Flask is a web micro-framework in python. It provides you with tools, libraries and technologies that allow you to build a web application.

source

Hands – on!

- https://github.com/juanhuguetgarcia/ml_project_template