MACHINE LEARNING

LAB - WORKING ENVIRONMENTS FOR MACHINE LEARNING

Danilo Croce – croce@info.uniroma2.it

Machine Learning
October 2025

GOALS

- Introduce tools and environments for running Python code.
- Show local and cloud execution options.
- Learn to manage Python environments and notebooks.

PYTHON AS THE COURSE LANGUAGE

- The course will primarily use **Python**, the most popular language for Data Science and Machine Learning.
- Python provides an extensive **ecosystem of libraries and frameworks** that make experimentation and prototyping fast and accessible.
- Core tools for data handling and analysis:
  - 🧮 **NumPy** – numerical computing and linear algebra
  - 📊 **Pandas** – data manipulation and tabular analysis
  - 📈 **Matplotlib** – data visualization and plotting
- Machine Learning and Deep Learning frameworks:
  - ⚙️ **scikit-learn** – classical ML algorithms and model evaluation
  - 🔥 **PyTorch**, **TensorFlow**, **Keras** – neural networks and GPU acceleration
- Advantages of using Python:
  - 🟢 **Open-source** and widely supported by both academia and industry
  - 💬 Huge community, tutorials, and pre-trained models available
  - 🧩 **Seamless integration** with notebooks (Jupyter, Colab) and visualization tools

WHERE TO RUN YOUR CODE

- Two main options:
  - **Local execution** — on your own computer
  - **Cloud execution** — on external systems (e.g., *Google Colab*, *Kaggle Notebooks*)
- Pros and cons:
  - *Local:* full control, persistent environment, better performance
  - *Cloud:* no installation needed, but sessions are temporary and limited

LOCAL SETUP WITH CONDA

- **Conda** is an *environment and package manager*
- It allows you to:
  - Create isolated environments with specific dependencies
  - Avoid version conflicts between projects
  - Reproduce setups easily on different machines

- Basic commands:

```
conda create -n ml-lab python=3.11
conda activate ml-lab
conda install numpy pandas scikit-learn
```

WHAT IS AN ENVIRONMENT?

- An **environment** is an isolated workspace containing its own Python interpreter and libraries.
  - It helps prevent **dependency conflicts** between different projects.
  - Think of it as a "box" that holds:
    - A specific **Python version**
    - A **controlled set of packages** and dependencies
- You can export it using an environment.yml file so others can share or reproduce it:
  - `conda env export > environment.yml`
- **Learn more (official docs):**
  - Managing environments in Conda — user guide:

  https://docs.conda.io/docs/user-guide/tasks/manage-environments.html

ANACONDA DISTRIBUTION

- **Anaconda** is a Python distribution that includes Conda and many preinstalled tools
- It also provides:
  - **Anaconda Navigator** — a graphical interface to manage environments
  - **Jupyter Notebook/Lab** preinstalled
- Great for beginners: everything ready to use

- For advanced users, **Miniconda** or **Mambaforge** are lighter alternatives

Official links:
  - Anaconda distribution → https://www.anaconda.com/products/distribution
  - Installing Miniconda → https://docs.anaconda.com/miniconda/

ANACONDA VS MINICONDA (AND MAMBAFORGE)

- **Conda** → the core *package and environment manager* used by both.
- **Anaconda**
  - Full Python **distribution** for data science and ML
  - Includes: Conda, Python, Anaconda Navigator, Jupyter, and many preinstalled packages (NumPy, Pandas, scikit-learn…)
  - ✅ *Best for beginners*: ready-to-use setup, no configuration needed
  - ⚠️ Large install (~3–4 GB), slower updates
- **Miniconda**
  - Minimal installer with **only Conda + Python**
  - You install packages as needed (e.g., conda install numpy pandas)
  - ✅ *Best for intermediate users*: lightweight, clean, flexible
  - ⚠️ Manual package setup required
- **Mambaforge**
  - Community alternative with **mamba** (faster Conda) and *conda-forge* channel by default
  - ✅ *Best for advanced users*: fast, reproducible, open-source friendly

CONDA AND THE SANDBOX CONCEPT

- **Problem in ML projects:** *dependency hell* (conflicts between library versions across different projects).
- **Conda solution:**
  - Each **environment** is a sandbox (separate folder) with its own Python + packages.
  - No interference between projects: one can use TensorFlow 2.10, another PyTorch 2.4.
- You must **activate the environment** to work inside it:

  ```
  conda activate myenv
  ```
- Empty at the start → you install only what you need.

INSTALLING PACKAGES

CONDA VS PIP

- Once the environment is created, you **fill it with packages**:
  - `conda install numpy pandas scikit-learn`
  - or `pip install package-name` if not available on conda.
- **Differences:**
  - **pip** → standard Python package manager, installs from **PyPI**.
  - **conda** → package + environment manager, can install Python libs **and non-Python dependencies** (C/Fortran libs, CUDA…).
- Rule of thumb:
  - Use **conda** for core packages (ML, scientific stack).
  - Use **pip** only if the package isn't available in conda.

REPRODUCIBILITY
SHARING ENVIRONMENTS

- Good practice: always save the list of packages with their versions.
- Options:
  - **Conda**:
    - `conda env export > environment.yml`
  - **pip**:
    - `pip freeze > requirements.txt`
- Why important?
  - Another person can rebuild your environment **exactly**:
    - `conda env create -f environment.yml`
    - `pip install -r requirements.txt`
- A well-documented ML project includes these files to ensure **reproducibility** and avoid “it works on my machine” issues.

INTRODUCTION TO JUPYTER NOTEBOOKS

- Jupyter Notebooks allow writing and executing Python code interactively
- Combine:
  - Code
  - Text (Markdown)
  - Figures and tables
- Excellent for analysis, experimentation, and teaching
- Basic shortcuts:
  - Shift + Enter → run a cell
  - "Kernel → Restart & Run All" → execute everything from scratch

GOOGLE COLAB (ONLINE ALTERNATIVE)

- **Google Colab**: a free, cloud-based notebook environment

- Provides:
  - CPU, GPU, or TPU runtime
  - Integration with Google Drive
  - No local setup required
- Perfect if your laptop cannot handle heavy computation

- 📘 **Introductory guide (Politecnico di Torino):**
  - https://dbdmg.polito.it/dbdmg_web/wp-content/uploads/2024/04/Colab_intro.pdf

USING NOTEBOOKS LOCALLY AND IN COLAB

- You can run notebooks in **two main ways**:
  - Online with **Google Colab** (no setup required)
  - Locally in **Anaconda/JupyterLab** (persistent, customizable)
- Download options from Colab or Jupyter:
  - `.ipynb` → open again in Jupyter/Colab
  - `.py` → plain Python script for any environment
- Good practice: keep both the notebook and environment file (environment.yml / requirements.txt) in your project.

INSTRUCTOR'S RECOMMENDATIONS

- Prefer **local setup with Conda/Anaconda** whenever possible
- Use **Colab** for quick tests or when you lack GPU resources
- Always keep:
  - Your notebooks organized and backed up
  - An exported environment file (environment.yml)
- Avoid installing packages globally on your system Python
- Recommended Python version: **3.10+**

TRAINING VS TEST
WHY DO WE SPLIT THE DATA?

- We start with a **labeled dataset**
- Goal: train a model that can **generalize** to unseen data
- **Training set** → the model *learns* from this data
- **Test set** → kept aside, simulates future data, used only once for **final evaluation**

⚠️ Warning: models may achieve **very high accuracy on training data** but perform poorly on test data → **overfitting**

AVOIDING "PEEKING" AT THE TEST SET

- Training a model often requires tuning **hyperparameters**
  - Example: parameter C in an SVM, number of layers in a neural network

❌ Wrong approach: optimizing hyperparameters by checking test accuracy (*cheating!*)

✅ Best practice: split the training data further:
  - **Train set** → fit the model
  - **Development (Validation) set** → tune hyperparameters
  - **Test set** → untouched, only for final unbiased evaluation

- **Takeaway:**
  - Never use the test set until the very end
  - Use train/dev/test for a fair and reproducible workflow

SIMPLE BUT IMPORTANT NOTES ON EVALUATION

- ⚠ **Do not change the test set** when comparing results
  - Different test sets may be easier or harder → results not comparable
- A possible approach:
  - Use multiple random splits of the test set
  - Compute **mean and variance** of performance across runs
- But in general use a well-assessed benchmark!!!

IS 80% ACCURACY GOOD?

- Always compare with **bounds**:
  - **Lower bound (baseline):**
    - *Weak baseline:* predict the most frequent class
    - *Stronger baseline:* use a simple model (e.g. SVM on basic features)
  - **Upper bound:**
    - Human performance on the same task
    - Best results reported in the **literature on the same dataset**
- **Why it matters:**
  - An accuracy value (e.g. *80%*) is meaningless alone
  - **Context comes from comparison with baselines and references**
- **Research practice:** In a good conference, a paper **without baselines is not accepted**

LABORATORY 1.1

FIRST STEPS WITH DATA

- This first exercise is mainly about:
  - Getting familiar with the **working environment** (Colab or Conda + Jupyter)
  - Loading a simple dataset (**Iris**)
  - Trying a first data manipulation technique: **shuffle & train/test split**
- 👉 Goal: **Practice running Python code in notebooks and understand the idea of holding out data for testing.**

- **LINK:**

https://colab.research.google.com/drive/1s2feJiR902pIrOEhJrUdedN7eQOC1aqA?usp=sharing

LABORATORY 1.2

TRAINING A NEURAL NETWORK

- In this lab, we move from **data preparation** to **model training**
- We will use:
  - **MNIST dataset** — handwritten digit recognition
  - **PyTorch** — one of the most popular deep learning frameworks
  - **MLP (Multi-Layer Perceptron)** — a simple feed-forward neural network
- **Learning objectives**:
  - Understand the pipeline:
    - Dataset → Model → Training Loop → Evaluation
  - Get hands-on with **PyTorch syntax**
  - Compare performance between **training and test sets**
- 👉 The purpose is **not** to reach state-of-the-art accuracy,
  - but to **understand the key steps** in training a neural network.
- **LINK:**

https://colab.research.google.com/drive/1kwUyurdARvIP28CWbvkM-TV-kuFfW-MP?usp=sharing

WHAT YOU NEED TO DO AFTER LAB 1

- Use **Colab Notebook of Lab 1.1**:
  - Open it in **Google Colab**
  - Download it as:
    - **Jupyter Notebook (.ipynb)**
    - **Python script (.py)**
  - Run it also on your **local machine** with Anaconda/Jupyter and using Python on the command line
- The **Colab Notebook of Lab 1.2** is **illustrative only**:
  - It shows the objectives of the course
  - 👉 For now, you don't need to go deeper into it