Data Science and Machine Learning Lab
Lab 05 - Regression Exam Simulation

Politecnico di Torino

1 Overview

In this laboratory, you will learn how to prepare and submit your exam solution within the CrownLab environment. You will explore each step of the submission workflow, from developing and packaging your code to understanding how the evaluator verifies, reproduces, and scores your results.

1.1 Platform and Exam Environment

All submissions will be carried out through the Moodle exercise page, which provides access to the CrownLab environment where you will develop and execute your code. The environment is entirely offline. All the main libraries used throughout the course, such as `numpy`, `pandas`, `matplotlib`, `scikit-learn`, `pytorch`, together with other commonly used ecosystem packages, will already be installed and ready to use. During the exam, you will be given two datasets: a **labeled development set** for training and validation, and an **unlabeled test set** on which you will generate predictions for your final submission.

1.2 Tasks

Given the development dataset (with labels), you must:

1. **Analyze** the data and design appropriate **transformations/preprocessing**.
2. **Train** a model (training *must* happen on-the-fly when `main.py` is executed; it is possible, anyway, to run a grid search for parameter tuning and then use in the `main.py` file the desired configuration).
3. **Predict** labels for the provided test set.
4. **Save** predictions to `submission.csv` (the file to be graded).

1.3 What You Must Submit

- A single executable file: `main.py` containing all your code that makes the **preprocessing** on the development dataset, **trains** the model, **predicts** the labels for the test set, and **saves** the predictions to `submission.csv`.
- A CSV file: `submission.csv` containing predictions for *all* test rows.

Important: `main.py` will be re-run by the evaluator. The predictions it produces will be compared with your uploaded `submission.csv` and considered in the evaluation.

1.4 How the Evaluator Works

The evaluator automatically:

1. Re-runs your `main.py` in a clean environment with the provided `dev.csv` and `test.csv`.
2. Checks that your script **trains on the fly** and generates a valid `submission.csv`.

3. Verifies the **schema** and **cardinality** of `submission.csv` (header names, row count, ID coverage).
4. Compares the re-generated `submission.csv` to your uploaded one to validate **consistency**.
5. It evaluates the result obtained with the actual (hidden) ground truth.

1.5 Validation Rules (Hard Requirements)

A submission *will not be evaluated or will be penalized* if any of the following is true:

- `main.py` **does not run** to completion under the provided environment. A **wall-clock time limit** will also be enforced to discourage excessively slow or inefficient solutions. If your program exceeds this limit, the submission will be deemed invalid.
- `submission.csv` is **missing**, **wrong** (e.g., wrong headers), or has a **row count different from the test set**.
- The **file naming is different** (must be exactly `main.py` and `submission.csv`).
- The evaluator cannot reproduce your `submission.csv` from `main.py` (**significant mismatch**).

1.6 End of the Exam

At the end of the exam, all your code (including `main.py` and `submission.csv`) will be automatically archived and uploaded to the **Elaborati** section. You will be able to review the delivered files there. After the exam window, you may refine your solution and submit a new attempt. Part of the evaluation will consider the **amount of code changed** between the first and second submissions.

1.7 Checklist Before You Submit

1. Run `python main.py` in a clean workspace and confirm it finishes without errors.
2. Open `submission.csv` and check: correct headers, correct number of rows, no missing values.
3. Re-run `python main.py` twice and verify the `submission.csv` is identical (or within stated tolerance).
4. Ensure you did not use Internet access or external files beyond the provided datasets.

1.8 FAQ

- **Can I use additional packages?** Only those that are pre-installed on CrownLab.
- **Can I save intermediate artifacts?** Yes, but they must be created by `main.py` at runtime. You cannot rely on pre-trained weights.
- **How are labels formatted?** Follow the exam statement (binary, multi-class, or regression). Match the required dtype and naming exactly. A submission example will always be already available on the Crownlab page.
- **What if my model is stochastic?** You should control randomness to make results reproducible, since this is also one part of the evaluation.

2 Exercises

2.1 Regression Exam Simulation

In this exercise, you will work on a regression problem where the goal is to predict a continuous target variable based on a set of heterogeneous features. The dataset includes:

- **Numerical features**, such as continuous measurements or ratios;
- **Categorical features**, representing discrete values without intrinsic order (e.g., color, brand, type);
- **Ordinal features**, representing ordered categories (e.g., "low", "medium", "high").

During the lab, focus on the preprocessing and modeling steps required to build a functional regression pipeline. In particular, you should:

1. Load the dataset and inspect its structure (number of records, columns, and feature types), and perform the right preprocessing.
2. Split the dataset into training and validation sets if you want to test locally your solution, or if you want to perform hyperparameter tuning, since the test set will not have the actual labels.
3. Train a regression model of your choice and evaluate its performance.
4. Once you have selected your best model, the whole pipeline with the correct hyperparameters should be run in the `main.py`, which should contain the dataset preprocessing, the training of the best model, and saving your predictions into `sample_submission.csv`, following the provided template.

2.2 Seeding for reproducibility

When running experiments that involve random operations (such as data shuffling, weight initialization, or sampling), results may vary slightly from one execution to another. To ensure that results are reproducible (i.e., that the same code always produces the same output), we set a *random seed*.

A random seed initializes the pseudo-random number generator used by libraries like `numpy`, `random`, and `torch`. Setting it to a fixed integer value guarantees deterministic behavior, which is essential for debugging, comparing models, and verifying experimental results.

Below is an example of how to fix the random seed across different libraries:

```
import os, random, numpy as np, torch

SEED = 42

# Python built-in random
random.seed(SEED)

# NumPy random
np.random.seed(SEED)

# Torch random (CPU and GPU)
torch.manual_seed(SEED)
torch.cuda.manual_seed_all(SEED)

# Ensure deterministic behavior in CUDA operations
torch.backends.cudnn.deterministic = True
torch.backends.cudnn.benchmark = False
```

Once the seed is set, every operation involving randomness will yield consistent results across multiple runs of the same code.