TÉCNICO LISBOA

2025-2026

Data Science

Prediction Server

Deadline – January 9th 2025 @ (23:59)

I. Deployment

Here are the guidelines for deploying the best classification models for the **second dataset** used in the classification part of the project.

A. Project Goal

The goal is to design, implement, and evaluate a web-based classification system that exposes a set of existing machine learning models through a browser interface.

The models and shared preparation pipeline are those identified as the best during **the classification part of the project**, while addressing the second classification task.

To do this, students must:

- <u>Design a client–server architecture</u> that allows a <u>web client</u> to use <u>multiple pre-trained models</u> via a clean API, <u>without any model retraining or updating</u>.
- <u>Implement a backend</u> that loads the stored preparation pipeline and models and exposes HTTP endpoints for prediction and evaluation.
- <u>Implement a frontend</u> web interface that enables non-expert users to
    - submit a single instance for prediction and
    - upload a validation dataset to compare model performances.

Suggested technologies (not mandatory):

- Python, scikit-learn and joblib for models and preparation pipeline.
- FastAPI or Flask for the backend.
- HTML/CSS/JavaScript (optionally a framework like React or Vue) for the front-end.

B. System architecture

Students must adopt a strict **client–server architecture**. But before doing so, students must save the preparation pipeline and the best models trained for each classification technique to be used by the system as described below in section Prediction Objects.

- Server (backend):
  - Runs as a web service (e.g., FastAPI/Flask app).
  - Loads models and the preparation pipeline specified in the input file, at startup.
  - Provides endpoints such as:
    - `GET /api/models` – list available models and metadata.
    - `POST /api/predict-single` – single-instance prediction.
    - `POST /api/evaluate-models` – performance estimation on uploaded validation data.
  - Handles all validation, preparation, prediction, and evaluation.
- Client (frontend):
  - Runs in the browser and never accesses model files directly.
  - Communicates with the server only via HTTP requests to the API.
  - Implements the interface described below.
  - Is responsible for user interaction and visualizing responses returned by the server.

Prediction Objects

Prediction objects include the best models identified for each classification technique required and the preparation pipeline selected to handle the dataset.

These objects must be made available to the system to be developed through several different files:

a) A text file, in any format (either semi-structured, like JSON, or not) listing the files that contain the implementation of the prediction objects, and that will be used as the main **input to the system server**.

b) One file per classification technique, containing the best model trained (for example, in a JOBLIB format)

c) One single file containing the preparation pipeline (for example, in a JOBLIB format).
d) One or more files containing the hyper-parametrizations for each one of the models made available, in a manner that the training of updated models would be possible.
e) One file containing the preparation pipeline description, specifying the preparation tasks applied and their parametrizations.

The descriptions available in d) and e) can be included in the file described in a) if students prefer.

Backend functionality

The server must implement:

- Startup loading:
  - Load the preparation pipeline and each one of the available models
  - If anything critical fails (missing file, invalid file, incompatible model), the server should clearly report the error.
- Prediction logic (`/api/predict-single)`:
  - Construct a feature vector compatible with the training data used (`instance`).
  - Apply the shared pipeline to transform the instance.
  - Use one of the models to classify the instance and compute its prediction.
  - Return a structured response with the predicted class.
- Evaluation logic (`/api/evaluate-models)`:
  - Accept an uploaded data file compatible with the training data used (`validation dataset`), including the target variable.
  - Load the validation data into a suitable structure. Split the target variable to compare it against the predictions made by each model.
  - Transform the data with the preparation pipeline.
  - Generate predictions for the validation data using any available models. Compute an evaluation of the predictions by comparing them with the target values received. Use at least two standard classification metrics.
  - Return results per model in a machine-readable format (e.g., JSON list of models with their metrics), which the client will then display.

- Error handling and robustness:
    - Validate inputs and provide meaningful error messages (e.g., missing fields, unknown model, malformed file).

Web interface requirements

The frontend must include a main page with a form offering two clearly visible modes:

- Predict a single instance
- Estimate model performance

The interface must include:

- A mode selector (e.g., radio buttons, tabs, or a dropdown) that switches between "Single prediction" and "Performance estimation".
- For single-instance prediction:
    - One input field per feature describing the dataset (text, numeric, dropdown, date picker, etc., as appropriate).
    - An input field to choose which model to use; if omitted, the server should apply the default model. Students must choose the default model.
    - A "Predict" button that sends the form data to the server and displays the predicted class (and optionally any other metadata) in a readable way (text, colored label, small chart, etc.).
- For performance estimation:
    - A file upload field to select a validation dataset (e.g., CSV).
    - An input field to choose which model to use; if none is chosen, the default model will be used. Optionally, more than one model can be chosen.
    - An "Evaluate" button that sends the file to the server and then displays the performance results for the model(s) (e.g., accuracy, precision, recall, F1-score).

Graphical visualizations (e.g., charts) are totally optional.

II. Evaluation

A. Development timeline and checkpoints

Students must structure their work around three evaluation checkpoints:

- **First intermediate evaluation – Architecture and prediction objects**
  - *Deliverables*:
    - Architecture diagram showing the server and its main modules, client, data folders, model artifacts, and data flows.
    - File formats for describing the prediction objects.
  - *Focus*: clarity of design, separation of concerns, and feasibility.

The architecture must be documented with at least one diagram (e.g., UML component diagram or high-level architecture diagram) showing the separation of concerns and main data flows (user → client → server → models → client → user).

- **Second intermediate evaluation – Backend prototype and API contract**
  - *Deliverables*:
    - Demonstration of a running server prototype **in the lab**, with basic tests that call the endpoints with small example data to verify that model loading and simple predictions work.
  - *Focus*: progress and correctness of model loading and prediction.
- **Final evaluation – Full system**
  - Deliverables:
    - Link to access a client of the fully implemented system.
    - Demo – small video (2 minutes or less) demonstrating the system working **uninterruptedly** in both modes (single-instance prediction and performance estimation using any compatible file).
    - Zip file with all required files to install a new copy of the system (server, client, models, preparation pipeline, and text files with available prediction objects). Data files can’t be included.
  - Focus:
    - **Correctness**: models and pipeline load correctly; predictions and metrics run without errors for valid inputs.
    - **Robustness:** the system gracefully handles invalid inputs.
    - **Quality of architecture**: clear separation between server and client; clean, modular structure; adequate choice for models and pipeline storage.
    - **Quality of demo**: coverage of most relevant functionalities, including both modes, data input, results presentation and error handling.

B. Evaluation criteria

Below are the evaluation criteria broken down by topic. All elements in the final delivery are mandatory.

DEPLOYMENT	**20%**
Preparation pipeline	
Description	2%
Running pipeline over dataset	1%
Running pipeline over instance	1%
Models (values per technique)	
Description of models' hyperparameters	0.25%
Each running model	0.75%
System	
Architecture (modularity, adaptability, ---)	2%
Server functionalities	3%
Client functionalities (input, presentation, data load)	1%
Error handling	1%
Demo	2%

Good Work!