Machine Learning Project, 2025/2026

Margarida Silveira, Catarina Barata, Alexandre Bernardino

September 13, 2025

1 Introduction

The project is split into two parts: regression and classification. The first part uses synthetic data and comprises one code deliverable for evaluation. The second part comprises two problems with code deliverables for evaluation. At the end of the project the students must deliver a final written report for evaluation, describing the work done and the results obtained in both parts.

The programming language used in the project is Python because it has powerful libraries for building Machine Learning applications, and it is a widespread language in the industry.

The Machine Learning project should be done in groups of two. Each group should work alone. Code and report will be checked for plagiarism, and detected cases will be penalized according to the university rules.

2 Student Evaluation

Student evaluation will take into account:

- code deliverables - statistical performance of the algorithms developed by the group, evaluated on independent data (test) sets - 25% of the lab grade;
- final written report (maximum length of 10 pages, font size 12 pt) describing the methodologies adopted by the group, including figures and statistical evaluation, and interaction with the teacher during the laboratory sessions - this is an individual assessment of each group member, thus both students must actively solve the project and be acquainted with the work; - 75% of the lab grade

The code deliverables consist in the code used in training the models, and code for testing the model in independent test data. Students do not have access to the test data, so each group must submit test code in a special format according to strict rules specified in the next section. The test code submitted by the student will be run by the teaching team in a benchmark framework to compare the prediction with the ground truth, and the results (a leaderboard with the scores achieved by each group) will be published on the Fenix web page. **If the teaching team is not able to run the test code delivered in the benchmarking framework, the deliverable will not be ranked**. A dummy benchmark code example will be provided for students verifying the compatibility of their test code with the bechmarking framework. More details are given in the next section.

The deliverables are due on:

- September 27: code for the Regression Problem
- October 11: code for the first part of the Classification Problem
- October 18: code for the second part of the Classification Problem
- October 24: final report

Attendance at the laboratory sessions is mandatory, and submissions from groups who fail to do so will not be evaluated.

3 Dataset and Project Submissions

The training data for each problem will be available through the course webpage on Fenix, in `numpy` (.npy) or pickle (.pkl) formats. For each problem, the students will have access to a training set with the feature vectors `X_train` and corresponding target outputs `Y_train`.

The test data is not available to the students. It will be used by the teaching team to evaluate the performance of the models submitted by the groups.

There are no restrictions on the number of machine learning models that students can research and try, but **the minimum is two models to be discussed in the report**. However, in each of the project questions, they must **pick only one model** to submit in the test code.

Project submissions should be made through Fenix, in the appropriate section. For each question, the students must submit a zip file containing the python code for model training (free format) and the python code and model description file for testing the model. **The test code follows a strict format specified in the following. Failure to follow the specified format will invalidate the deliverable**.

3.1 Format of the Test Code

The test code must include a file with the name `mymodel.py` (mandatory name) containing a function named `predict` (strict name) that accepts as parameter the test features and returns the test predictions. Test data has the same format of the training data with the exception of size, that is defined in the specific description of the project parts, in the following sections.

The `predict` function should not include training code. To transfer the model learned in the training stage to the `predict` function, students are advised to use the libraries `pickle` or `joblib`. These libraries have functions to save and load to file the parameters and other information about the models to be run in the `predict` function. So, all `pickle` or `joblib` files required to run the `predict` function must be included in the deliverable.

To verify the compatibility of the delivered test code and the benchmarking framework used to score the models, a dummy scoring script will be provided for each part of the project. Below is the code of the dummy scoring script for the first problem:

Listing 1: The code for validating the format of the delivered test code in the first problem. File `validate_test_code.py`

```
import numpy as np
from mymodel import predict
#Generate dummy random data X of size (300,6) and y of size (300,)
X_test = np.random.rand(300, 6)
y_test = np.random.rand(300)
# Make the predictions
y_pred = predict(X_test)
#validate the size of y_pred
if y_pred.shape != y_test.shape:
    raise ValueError(f"Shape␣mismatch:␣{y_pred.shape}␣vs␣␣{y_test.shape}")
print("Prediction␣format␣is␣valid.")
```

In summary, the delivered zip file must contain:

- the Python code for all the experiments.
- a special Python file called `mymodel.py` that defines a function called `predict`.
- the model specification files (e.g. `pickle` or `joblib` files) required by the function `predict`
- the function `predict` must run without errors from the provided script `validate_test_code.py`

The teaching team will assess the performance of the provided test code in the test set using appropriate statistical metrics. The scores achieved by each group will be made available on a leaderboard on the course website.

4 Part 1 - Regression with Non-linear Models

The first problem illustrates the application of linear regression techniques to a non-linear problem. We will use non-linear transformations of the input features, such as polynomial functions, radial basis functions, or other, to emulate a non-linear input-output relationship.

4.1 Motivation

In many industrial or scientific applications, a certain target measurement (e.g., stress concentration, radiation level, fluid density) can only be obtained via a very expensive, high-precision sensor. To reduce cost and improve scalability, we would like to replace this sensor with a software model that estimates its output from a set of 6 cheap sensors which measurements are related non-linearly with the target measurement.

4.2 Objectives

The main objective of this exercise is to illustrate the use of linear regression techniques with non-linear transformations to predict the values on a non-linear input-output mapping.

A dataset of 1000 samples was acquired, containing the values of the cheap sensors in the data matrix X, of size $(1000, 6)$ and the corresponding values of the expensive sensor in the data vector Y of size $(1000,)$. For training, a set of 700 samples is randomly selected from the original dataset, to create the data matrix X_{train}, and the output vector Y_{train} provided to the students. The remaining 300 samples will be used by the teaching team as test data, to score the models trained by the students. The evaluation metric will be the **Coefficient of Determination (R^2)**.

4.3 Challenges

There are several known facts that challenge regression methods, in particular linear regression, like feature redundancy, input scale imbalance, high-signal-to-noise ratios, irrelevant variables (low correlation with output), and unreliable sensors (with large fractions of outliers). You should explore techniques to diagnose and mitigate such cases, and the possibility of removing problematic features or points from the model.

Additionally, non-linear models can easily overfit to input data and perform badly in the test set, so you should use methods to prevent overfitting.

4.4 Provided files

The following files will be provided:

- `X_train.npy` – input features for training in `numpy` format with shape $(700, 6)$
- `Y_train.npy` – target values for training in `numpy` format with shape $(700,)$
- `validate_test_code.py` – a script file that students can use to confirm that the submitted test code will run without errors in the teachers evaluation.

4.4.1 Loading Data

You can use this code to load the training data:

```
import numpy as np
X_train = np.load("X_train.npy")
Y_train = np.load("Y_train.npy")
```

4.4.2 Saving and Loading Models

When a model is trained, its parameters can be saved into a file. Later, the model can be loaded from that file and used for making predictions. Two main Python libraries are used with `scikit-learn` to save and load machine learning models to file: `pickle` and `joblib`.

Below we show some examples using `pickle` to load and save linear regression models:

Listing 2: Example: Dumping a model in `pickle`

```
import joblib
import numpy as np
from sklearn.linear_model import LinearRegression
# Example data: y = 1 + 2x
X = np.array([[i] for i in range(-5, 6)])   # Inputs: -5 ... 5
y = 1 + 2*X[:, 0]                           # Outputs: 1 + 2x
# Train the model
model = LinearRegression()
model.fit(X, y)
# Save model to pickle file
joblib.dump(model, "linreg.pkl")
```

Listing 3: Example: Loading a model in `pickle`

```
import joblib
import numpy as np
# Load the saved model
loaded_model = joblib.load("linreg.pkl")
# New input data
X_new = np.array([[7], [8], [9]])
# Make predictions
y_pred = loaded_model.predict(X_new)
```

Note that if you do some data pre-processing (e.g. normalization, feature extraction), these processes must be done both on training and testing. Pre-processing step can also be saved and loaded in `pickle` files, using the context of pipelines.

Below are examples of saving and loading a linear regression pipeline with data normalization and degree-2 polynomial features:

Listing 4: Example: Dumping a pipeline in `pickle`

```
import joblib
import numpy as np
from sklearn.preprocessing import PolynomialFeatures, StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.pipeline import Pipeline
# Example data: y = 1 + 2x + 3x^2
X = np.array([[i] for i in range(-5, 6)])
y = 1 + 2*X[:, 0] + 3*X[:, 0]**2
# Build pipeline: Polynomial -> StandardScaler -> LinearRegression
model = Pipeline([
    ("poly", PolynomialFeatures(degree=2)),
    ("scaler", StandardScaler()),
    ("linreg", LinearRegression())
])
model.fit(X, y)
# Save pipeline to pickle
joblib.dump(model, "poly_linreg.pkl")
```

Listing 5: Example: Loading a pipeline in `pickle`

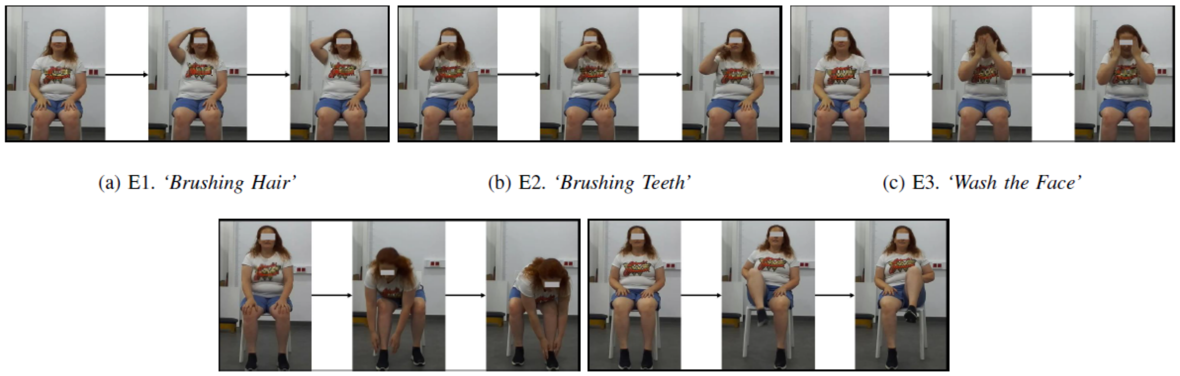
```python
import joblib
import numpy as np
# Load pipeline
loaded_model = joblib.load("poly_linreg.pkl")
# New input data
X_new = np.array([[7], [8], [9]])
# Make predictions
y_pred = loaded_model.predict(X_new)
```

4.4.3 Important!

Please make sure you comply with the naming of the test script, `mymodel.py` and test function `predict`; otherwise, the instructors will not be able to assess your score. Make sure that the provided script `validate_test_code.py` runs successfully in the folder to be submitted as a zip file.

5 Part 2 - Classification of Physical Therapy Sessions

The second part of the project is devoted to analyzing video data from physical therapy sessions of 18 stroke survivors. The patients were recorded while performing different functional exercises that simulate Activities of Daily Living, as shown in Figure 1. The repetitive execution of such exercises contributes to the improvement of motor function. The videos were recorded using a ZED stereo camera from StereoLabs. The data provided consists of 2D skeletons extracted with MediaPipe[1] from the RGB frames. There is a total of 33 keypoints distributed across the entire body, as show in Figure 2.



(a) E1. *'Brushing Hair'* (b) E2. *'Brushing Teeth'* (c) E3. *'Wash the Face'*

(d) E4. *'Put on Socks'* (e) E5. *'Hip Flexion'*

Figure 1: Example of functional exercises performed by the patients.

Our first goal is to distinguish between three types of functional exercises. The second goal is to predict which side (left or right) was left impaired by the stroke.

5.1 First Problem - Exercise Recognition

The first classification task is a multiclass one, where we want to create a model that distinguishes between exercises *E1* (brushing hair), *E2* (brushing teeth), and *E5* (hip flexion). For this task the labels are 0 (*E1*), 1 (*E2*), and 2 (*E5*).

5.1.1 Provided Data

To tackle the first classification challenge, you will receive a training dataset `Xtrain1.pkl` containing features from 700 exercise recordings and the patient IDs. We also provide the corresponding target

[1] https://github.com/google-ai-edge/mediapipe

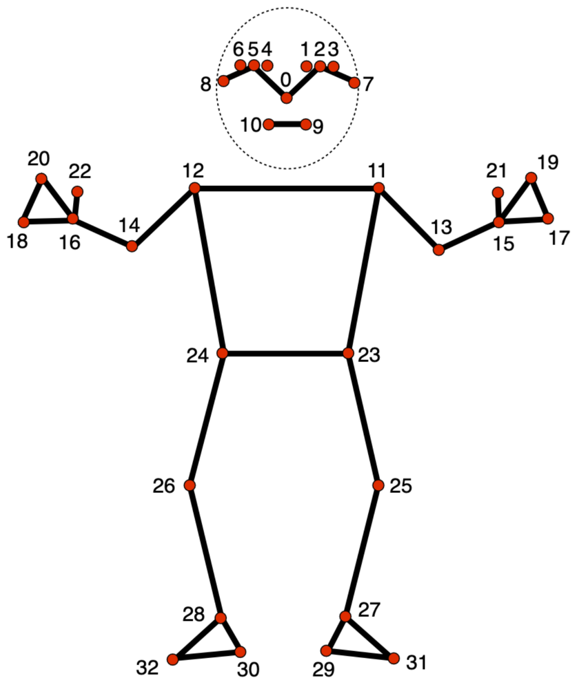


Figure 2: MediaPipe Skeleton Keypoints. Each number corresponds to a keypoint - this is the ordering followed in the project.

(ground truth) values in `Ytrain1.npy`. The exercise samples were collected from 14 out of the 18 patients, while the exercises from the remaining 4 patients were kept as the test set.

The teaching team computed the average and standard deviations of the positions x and y of each keypoint from the skeleton. These values were combined to obtain the features of the data, meaning that the training dataset must have a shape (700,132). The test set will have a shape (200,132). Each feature vector contains the average values for each keypoint (x and y), followed by the corresponding standard deviations. Thus, the first 66 positions of the feature vector correspond to the average of the positions for each keypoint and the last 66 to the standar deviations.

5.1.2 Challenges

Each of the patients from the dataset has a different degree of impairment. Moreover, they will have different heights, body poses, and overall anatomical structure. Finally, each patient can be affected on the left or right side of the body. This means that there can be a significant variability across the skeletons of the various patients, while doing exactly the same exercise. Thus, you must explore mechanisms to cope with this variability and ensure that your model will work well on a new set of patients that it has not seen before.

5.1.3 Deliverable and Evaluation

Similarly to the regression problem, students must submit a `mymodel.py` (mandatory name) containing a function named `predict`. This function must receive a test matrix `Xtest` of shape (200,132) and return a vector of predictions of shape (200,). The metric the teaching team uses to evaluate the submissions for this task is the F_1 **score**.

5.2 Second Problem - Identification of the impaired side

The second task is a binary one, consisting of predicting whether a patient is affected on the left (label 0) or right (label 1) side.

For this task, you will receive a training dataset `Xtrain2.pkl` containing a data frame with the following information: patient ID, type of exercise, and a sequence of skeletons. This means that each row of the data frame will be associated with one patient and a specific exercise, containing the evolution of the skeleton keypoints over time. Overall, you will have access to 446 exercise sequences, divided by the 14 training patients. The `Ytrain2.npy` will have a shape (14,), comprising patient-level labels w.r.t the affected side.

The test data will follow the same structure as the training one, containing 126 exercise sequences for the 4 remaining patients. The goal will be to predict the affected side of each of these patients.

5.2.1 Challenges

Several challenges are associated with this task. Besides the differences across patients, already mentioned for the first classification task, students will need to seek for strategies to handle the following challenges:

1. How to get a patient-level label from multiple observations over time? You will need to define a strategy to aggregate the information of the multiple exercise sequences to make a final prediction for the patient.
2. How to work with skeleton sequences that have different lengths/durations? There is no guarantee that different patients take the same time performing an exercise. In fact, this can even change for the same patient across different trials. You must identify a suitable approach to convert the skeleton sequences into features that are standardized across trials and patients.
3. Each patient performed a different combination of exercises and you must account for this while developing strategies to extract the features.

Finally, note that there is a high imbalance between left and right-sided affected patients. Thus, you will need to come up with an approach to mitigate this.

5.2.2 Deliverable and Evaluation

Students must submit a `mymodel.py` containing a function named `predict` (mandatory names). This function must receive a data frame `Xtest` comprising 115 rows and return a vector of predictions of shape (4,).

The metric the teaching team uses to evaluate the submissions for this task is the **Balanced Accuracy**.