Machine Learning in Computational Biology

Spring 2025

10/3/2025

Assignment #1

Assignment goals

The first assignment for this semester is organized into three tasks. The first is related to setting up a development environment and following proper software design principles, using Python files to build up your codebase and notebooks for the presentation or operations you will perform, such as plotting, data exploration, and cleaning. It also serves as an introduction to git, a powerful version control software that helps the developers track the codebase changes, collaborate efficiently, and manage different versions of their projects.

The second task aims to familiarize you with some basic data engineering actions. Data in the real world is rarely in the proper form for machine learning (ML). Additionally, you will learn to implement one of the most popular ML techniques, cross-validation, in the context of regression.

You will apply the above skills in the third task to develop and evaluate regression models with or without feature selection. The ML pipeline you will develop in task 3 aims to develop effective regression models to predict a person's body mass index (BMI) using as input the composition of probiotic bacteria in their gut!

Task 1: Set up your development environment (Non-graded)

1. Select and download your integrated development environment (IDE); one of the best free options is Visual Studio Code.
2. Establishing a Linux environment is essential for this assignment. Linux users can skip this step. However, Windows users should install Linux alongside Windows without the need for a dual-boot by utilizing the Windows Subsystem for Linux (WSL). You can also connect the VS code to your WSL environment [tutorial].
3. Install Python in your Linux WSL environment via the terminal with Anaconda/Miniconda package manager [overview, tutorial].
4. Create a GitHub account (if you haven't done it already) and install git via the Linux terminal [tutorial].

Task 2: Download and explore the dataset

You will be working with metagenomic data from gut microbiome analysis to predict the body mass index (BMI) of study subjects using regression models and feature selection. To get started, run "gh repo fork https://github.com/MLCB2025Class/Assignment-1" in your terminal to fork the

project directory from GitHub to your computer. This will create a copy of the repository containing the project data under your GitHub account and download it to your computer. **You should work in your forked repository rather than the original one to track your changes and submit your work.** If you are new to GitHub, you can follow this guide on how to fork a repository (you do not have to sync your fork with the upstream repository - more info on git).

1. First, create the sub-directories to structure your project. These should be "src" for the source code, "notebooks" for the Jupyter notebook files (.ipynb), and "models" to store your trained model instances. The dataset provided should exist in the ./data directory. Create a notebook "data_exploration.ipynb".
2. In the "data_exploration.ipynb", load the given datasets and process them until they are in a format you deem ready for the machine learning tasks at hand. Once you do that, you will need to export the cleaned datasets in the "./data" directory by the names "development_final_data.csv" and "evaluation_final_data.csv". (*Note: The datasets are not scaled. We recommend to ensure proper scaling that avoids data leakage, especially during cross-validation.)*
3. In this task, you are free to choose how to visualize the data to best describe the datasets. You can explore visualization techniques provided by the package of your choice (matplotlib, plotly, seaborn).
4. Provide in your report (see below) a justification of any cleaning operations you deemed necessary to perform. Are all the columns necessary for the data analysis?

Task 3: BMI prediction using regression models and metagenomic data

In the ./src directory, create a "functions.py" file. There, you will write your codebase to identify the optimal model instance of three regression algorithms: Elastic Net, SupportVectorRegression, and BayesianRidge. The main subtasks of the ML workflow that you are asked to implement are summarized below:

- **Establish a baseline** model to set minimum performance benchmarks and a reference point for more complex models. For this baseline, you should use all features (bacteria species) and default hyperparameters for all regression algorithms (no feature selection, no model tuning).
- **Feature selection**: Using your preferred feature selection (FS) method, identify the smallest feature set that can solve the regression problem effectively and certainly better than the baseline without hyperparameter tuning.
- **Model Tuning:** Having selected the best feature set, use cross-validation to fine-tune the hyperparameters of your FS models. It should:
    - Take as input, at least, the selected features to be used for training.
    - Perform cross-validation with a number of folds (**K**) of your choice.

- Identify the best hyperparameter combination based on the minimization of Root Mean Squared Error (RMSE) across the validation splits.
- Output the final trained model with the optimal hyperparameters.

All the above three subtasks are performed using the development dataset.

- **Evaluation:** The goal of this subtask is to assess the generalization performance of a trained and fine-tuned, if requested, model instance using a separate evaluation set not utilized for any task related to the model's development. It should return statistics such as the mean, median, and confidence intervals for key metrics like RMSE (Root Mean Squared Error), the Mean Absolute Error (MAE), and at least one additional metric of your choice (*hint*: To obtain meaningful statistics, the evaluation process must be repeated multiple times.). Consider different methods to achieve this:
    - How can you generate multiple evaluations from a single dataset?
    - What strategies could help ensure that your performance estimates are stable and generalizable?
    - How would you quantify uncertainty in your metric estimates?

The final model instance should be stored in the ./models directory, where they will be loaded for **inference** on **unseen data** we have excluded from the provided datasets.

More specifically, for the tasks at hand:

1. After setting up your codebase, create a notebook named "model_analysis.ipynb". Then, import your codebase from functions.py and load the clean datasets you created before.
2. Generate (for every regression algorithm) a baseline model with default hyperparameters and no feature selection to populate ./models using the entire development set for training. Then, evaluate the trained baseline model using the evaluation dataset. Read the evaluation paragraph above carefully to learn what a solid evaluation process entails.
3. Use the development set for feature selection in order to reduce the dimensionality of the original dataset. You can employ a method of your choice for this task. The features you select should be "stable" i.e., do not change too much if the training set is perturbed. Then, train a model on the selected features using the development set and test it thoroughly using the evaluation set. Compare this model to your baseline for the same algorithm. Describe and justify your feature selection approach and briefly explain the observed results in your report.
4. Using the development set with selected features, perform hyperparameter tuning with cross-validation to populate the ./models directory with the optimal model instance (best hyperparameter combination based on the minimization of Root Mean Squared Error

(RMSE) across the validation splits) for each regressor. Thoroughly evaluate the tuned models on the evaluation set.

5. For every model you evaluate at any stage, create boxplots of the used metrics so that you can compare models rigorously as you move from the baseline (no FS, no tuning) to the FS model (no tuning) and, finally, to the FS+tuning final model for every algorithm.
6. Establish a directory named "final_models" to house only three best model instances (1 per regressor), trained on the selected features and tuned. Add the best model instances per regressor into "final models". Compare the results and declare the **best overall model/regressor**. Explain your reasoning and consider model complexity vs performance vs interpretability tradeoffs in your decision.
7. Train what you consider your best overall model instance on the whole dataset (development+evaluation set). Save the best overall model in the "final_models" by the name of "winner". Your best work will be evaluated by the instructors using an unknown, to you, hold out set based on your winner model instance (*Hint*: Keep in mind that this best model will be used on new **unseen data**. This means that it should somehow include the transformations you applied to the data **before** using them as input to train the model. Take a good look at sci-kit learn Pipeline.)

In any case, to guarantee the **reproducibility** of your results, you should use **seed=42**.

For extra bonus points (50% extra points for each part)

The following bonus parts are **independent** of each other. You can work on as many as you want, **and they are all totally optional**. How bonus points contribute towards your final class grade will be explained in class. If you choose to complete bonus tasks, do so in a notebook named "Bonus_{x}.ipynb" in the ./notebooks directory, where x is the bonus task number.

1. Add an alternative option for hyperparameter tuning in your "functions.py" codebase via an input parameter using the Optuna package with trial pruning and Bayesian optimization. Compare and discuss the performance of the resulting models against the optimal models from subtask 5 above.
2. Convert the regression problem into a binary classification problem (establish a BMI threshold, hint)! Your models should be classifiers, not regressors, for this problem. Minimally modify your pipeline now, using Logistic Regression and Gaussian Naïve Bayes as simple classifiers. Comment on your results.

Deliverables

- Your main deliverable will be a concise, **maximum 10-page report (typed)** in which you should summarize your methods and results (using evaluation boxplots) in a format used in scientific papers. Use a concise yet precise methods-results-discussion exposition format for every subtask.

- Your report **must** include a link to your **GitHub repository**, which, as previously discussed, should contain all the required sub-directories. This will serve as an introduction to **version control systems**, an essential skill in computational work. Make sure your repositories are public.
- For a perfect report, you should also incorporate **relevant commentary (max. 2 pages) based on the literature** to explain and support your results. Can you find pertinent literature that supports the results of your feature selection?

Hints

- When creating functions for your codebase, it is a good practice to follow the "Single responsibility principle" (SRP), which states that each function should have a single specific purpose.
- We encourage you to use **object-oriented Python programming** with reusable classes. If you have not done this before, it is time to step up and explore it! It will be required for your next assignment anyway! Also, the code you write now will follow you in the next assignment (and for much longer, hopefully).
- Your **repository and codebase** should be structured to facilitate easy **reproducibility and extension** of your pipeline. This will be beneficial for future assignments. Repeated code in your codebase indicates the need for **modularization** through functions. Structuring your code into reusable modules improves readability and simplifies the process of **adding new features**. A well-organized codebase also makes it easier to integrate bonus tasks seamlessly into your pipeline!
- Maintaining a **well-structured notebook** with **clear markdown documentation** is crucial for both yourself and the reviewer, ensuring readability and organization**. It is your job to make the evaluation of your work easy for the reviewer.** Put yourself in the position of someone who is submitting a paper for publication, and the quality, readability, and ease of code evaluation are main elements of the review process..

An important note on LLM usage for coding

- Learning programming/Data Science isn't about mastering a single thing. It is mostly about gathering bits and pieces of knowledge and fitting them together like a puzzle. Once you have built "muscle memory" from practicing with assignments like this one, you can easily use your knowledge to tackle similar problems in the future. However, if you rely on LLMs without a 100% understanding of the code they provide, you enter an infinite cycle where you will keep returning to it for the same tasks over and over again, and each time it is varied, often error-prone solutions will only lead to further confusion. It is much better to use them as a teaching assistant if needed, although the SciKit Learn package provides great documentation with in-depth explanations for all of the tools it provides,

and it is a great idea to familiarize yourself with its use as early as possible. After all, assignment solutions that utilize copy/pasted LLM code are most often distinguishable, and we have the experience to spot them! **In any case, if you use an LLM, you should disclose which one and how you used it in your report.**

Deadline: You should submit your work (max 10-page report in PDF, including a link to your GitHub repository) via **e-class** by **Monday, April 7, 2025**. If you work on bonus parts, you should submit a separate report for each one of them, so please include all your reports in a zip file with your name.

Start working on this assignment early and well before the deadline. Do not forget that "The devil is in the details."