MLiCB Assignment 1

Nikos Charisis 7115152400008

This is the report of the 1st assignment in MLiCB

I. General overview

The purpose of this assignment is multi-fold. First, it intends to familiarize the student with some basic software design principles and tools and to introduce version control tool 'git'. Secondly, it aims to provide some hands on experience on tasks such as data cleaning, plotting and visualization. Thirdly, it serves as an excellent way to delve into some of the most famous ML algorithms for regression, such as Bayesian Ridge, Elastic Net and SVR, all of them being implemented by the sci-kit python library. It is worth mentioning that the main task of this assignment is BMI prediction using regression models and ONLY metagenomic data, which by itself is a quite interesting and challenging task as it is well known that BMI is mostly linked to one's height and weight, parameters that are completely absent in our datasets. Nevertheless, we approach this task with scientific curiosity and enthusiasm, hoping for the best.

II. Data exploration

A. First peak

By peaking just a tad in our sets, we notice that both have 140 columns and therefore 140 potential features. The development set has 489 rows (entries) and the validation set 211 rows (entries). So, whilst not tiny, our dataset is not very large.

B. Closer Look

Moving on and looking closer to our data, we notice that out of the 140 columns 134 refer to actual bacteria, with the rest carrying information about the age and the sex of the host, their BMI, the type of the experiment and its respective IDs. When the dataset is also loaded using pandas an extra 'Unnamed: 0' column is added, but since it is not part of the original dataset, we will not refer to it any further even though we will consider it when the time for data cleaning comes.

C. Deeper Look

By performing some actual data exploration, we notice that none of the bacterial, age or sex columns is normally distributed, with outliers being present in a great deal of them. This is something that we need to keep in mind when performing the cleaning and scaling of our data by using scalers that are robust in such cases. The non-normality of our data can be noticed by observing at the respective qq-plot of the feature. In **Figure1**, we can observe such a plot for BMI. As a final step of this investigation, we calculate the Pearson correlation between BMI and all of the other significant columns (that being 'Host Age', 'Sex' and all the bacterial ones). We observe that none of them are strongly correlated, so chances are that most play a part of their own in BMI prediction, making feature selection a tad risky and tricky. In **Figure2**, we can observe the heatmap of the most correlated bacterium with BMI

D. Data Cleaning

When it comes to data cleaning, we check our datasets for NA and duplicates values. We fill the possible NA entries with '0.0' and remove the duplicate values. It is worth mentioning here that our dataset contains none of the aforementioned, an indication of the meticulous work required to produce it. Finally, as the new clean datasets we save the 'BMI', 'Age' and 'Sex' along with all of the bacterial columns. The idea behind saving 'Age' and 'Sex', even though our predictions are solely based on the micro biome metrics, has its foundation outside of the scope of this exercise, where we know that both the age and the sex of the host play a part in their BMI interpretation. So, maybe in a future project this type of information can be used to our advantage. Still, for this purpose of this assignment only the cleaned 'BMI' and bacterial columns will be used.

E. Data Scaling

Even though it does not occur in this section of the assignment, a couple of words about our approach towards data scaling is in order. Since we have observed that our data are not normally distributed and have outliers, our approach towards scaling is to use a pipeline comprising of scikit's RobustScaler and PowerTransformer. RobustScaler implements the scaling part of the pipeline while PowerTransformer, contrary to our expectation of turning into a muscle-car and fighting alien invaders, makes the data more Gaussian, a quite helpful property in the grand scheme of things, as a lot of models work tremendously well with the assumption of the normality. As a final, we will point out that scaling DOES NOT occur before the saving of our new cleaned data but AFTER. This is in order to minimize the risk of data leakage.

III. PREDICTION WITH REGRESSION MODELS

Before we begin our discussion about the results of the regression methods, we should highlight once again the need for a tiny bit of data preprocessing, as pandas tends to add a 'Unnamed: 0' column every time we load a dataset. It poses no significant threat to our work as we just need to drop this column before we scale our data.

We begin our work by scaling both our bacterial and our BMI data. For that intend and purpose we use the respective function that can be found in the functions.py file. After that, we move on to the comparison of the different methods and approaches:

A. Baseline models

For the intends and purposes of this task, we use the IO and DefaultPredictor classes, whose implementation can be found in the functions.py file. The IO class handles the loading and saving of the models, while the DefaultPredictor class implements the actual training of all three models by using the default values for the hyper-parameters, provided by sci-kit. The class Evaluator is used to evaluate each method and return some keys metrics such as RMSE,MAE and R2, which is the third evaluation metric we opted to use along side with RMSE and MAE. The reason for that selection lies in R2's purpose, which is to provide a measure of how well observed outcome are replicated by the model, based on the proportion of the total variation of outcomes explained by the model. With that in mind, for each model we get the following results:

a) Bayesian Ridge

	mean	std	median	CI Low	CI High
RMSE	0.9373	0.0516	0.9372	0.8368	1.0375
MAE	0.6905	0.0413	0.6918	0.6103	0.7690
R2	0.1112	0.0490	0.1122	0.0094	0.2027

b) Elastic Net

	mean	std	median	CI Low	CI High
RMSE	0.9974	0.0507	0.9989	0.8906	1.1033
MAE	0.8000	0.0391	0.8025	0.7209	0.8800
R2	-0.0054	0.0070	-0.0023	-0.0260	-0.0000

c) SVR

	mean	std	median	CI Low	CI High
RMSE	0.8810	0.0502	0.8809	0.7833	0.9808
MAE	0.6189	0.0403	0.6177	0.5391	0.6993
R2	0.2145	0.0492	0.2182	0.1113	0.3052

Judging from the above tables, we notice a superiority of SVR compared with the other two methods. Its RMSE and MAE values are significantly lower, while its R2 value is significantly higher, something that can be comprehended easily by looking at **Figure3**. For now at least, it seems that SVR is a prime candidate for our top performer. Let's see what happens when we perform feature selection

B. Feature selection with no tuning

In order to perform feature selection, we use sci-kit's RFECV with DecisionTreeRegressor as the estimator. The reason behind this combination is multi-fold. First and foremost RFECV is a feature selection method that utilizes CrossValidation, with the default number of folds being 5. This trait of RFECV ensures the high performance of the method, combined with the fact that our datasets are not considerably large. Secondly, the selection of the DecisionTreeRegressor is somewhat of compromise between two dilemmas. Those are whether we should use a different feature set for each model and whether, should we use only one feature set for all models, we should use one of the evaluated models as the estimator ins RFECV. The compromise is that we do generate only a single feature set, common to all models, but with the aid of a 'third party'. The goal is for our feature set to be the right amount of generalized and specified. We gather that the usage of the DecisionTreeRegressor achieves that. Having said that, let's take a look at the results

a) Bayesian Ridge

	mean	std	median	CI Low	CI High
RMSE	0.9334	0.0514	0.9338	0.8371	1.0311
MAE	0.6884	0.0414	0.6883	0.6093	0.7681
R2	0.1112	0.0490	0.1122	0.0080	0.2226

b) Elastic Net

	mean	std	median	CI Low	CI High
RMSE	0.9974	0.0507	0.9989	0.8906	1.1033
MAE	0.8000	0.0391	0.8025	0.7209	0.8800
R2	-0.0054	0.0070	-0.0023	-0.0260	-0.0000

c) SVR

	mean	std	median	CI Low	CI High
RMSE	0.8764	0.0500	0.8759	0.7776	0.9750
MAE	0.6161	0.0404	0.6144	0.5358	0.6984
R2	0.2227	0.0510	0.2272	0.1201	0.3201

Taking a look into the above tables, we notice an improvement in terms of SVR and Bayesian Ridge. Elastic Net has no improvement whatsoever, which is a tad troublesome, but we still have the tuning part of the exercise left. Regarding the comparison between the methods, it is still abundantly clear that, even in the 106 feature set that we used, SVR is the superior method. A visual representation of that can be found in **Figure4**. Finally, let's move on to the hyper-parameter tuning section of this assignment

C. *Hyper-parameter tuning on selected features*

Having selected our optimal feature set in the step before, we move on to tuning the hyper-parameters of the models. For that purpose, we are being aided by sci-kit's GridSearchCv, which, among other methods, uses 5 fold cross validation in order to determine the best value for the given hyper-parameters. It must be pointed here, that for some methods we opted to not search for some specific hyper-parameters (e.g, alpha_init in BayesianRidge) as we lack the required knowledge for a decent starting point. Also, instead of using the DefaultPredictor class, in this task we make use of the TunedPredictor class, which is a subclass of the DefaultPredictor and has the capabilities to tune the models at its initialization. With that in mind, let's take a look at the results:

a) Bayesian Ridge

	mean	std	median	CI Low	CI High
RMSE	0.9334	0.0514	0.9338	0.8371	1.0311
MAE	0.6884	0.0414	0.6883	0.6093	0.7681
R2	0.1112	0.0490	0.1122	0.0080	0.2226

b) Elastic Net

	mean	std	median	CI Low	CI High
RMSE	0.9255	0.0520	0.9234	0.8296	1.0309
MAE	0.6789	0.0410	0.6797	0.5974	0.7583
R2	0.1336	0.0461	0.1341	0.0380	0.2267

c) SVR

	mean	std	median	CI Low	CI High
RMSE	0.8812	0.0504	0.8811	0.7842	0.9824
MAE	0.6280	0.0403	0.6266	0.5479	0.7098
R2	0.2144	0.0460	0.2173	0.1214	0.3017

A careful look at the above results reveals a really interesting peculiarity. That is the simultaneous improvement of Elastic Net, the lack of any change in Bayesian Ridge and the ever so slight worst performance of SVR. A possible explanation for that phenomenon could be the fact that sci-kit's default values for the hyper-parameters are somewhat carefully selected, making the out-of-the-box models pretty resilient and versatile. That would explain the lack of any change in the Bayesian Ridge model. Regarding the worsening of the SVR model, this behavior hints towards some sort of over-fitting during the training, as the values of the hyper-parameters are indeed different from the default ones (check model_analysis.ipynb for more). What has remained unchanged however, is the dominance of SVR compared to all the other methods, something that can be observed visually in **Figure5**.

IV. Selecting the best models

Taking all of the aforementioned into consideration it is to conclude that the overall best method, at least by our evaluation standards, is baseline SVR with feature selection. The default hyper-parameter values of sci-kit proved just too good and impossible to beat, which is to be expected considering the phenomenal work of everyone involved with that library. Regarding Bayesian Ridge, we elect to save our hyper-tuned model on the feature set, just to boost our ego a little bit. It goes without mentioning, that the best model for Elastic Net is the one that was tuned and trained on the selected feature set. As a side note here, we would like to point out that this model highlights the value of fine-tuning as it transformed what seemed to be a terribly underperforming model, to a strong candidate for our best solution.

All in all, the final pipeline for the winner model includes the SVR model trained on the selected feature set along with all the required tools need to transform the unseen data into the proper form for predictions

V. code REMARKS

Regarding the implementation of the assignment, it should be mentioned that the heavy lifting is being done by the classes and functions in functions.py. These are the ones that train, predict, evaluate, save, load and plot. Also, a considerably part of the data cleaning and scaling process is done by the respective functions. Something worth pointing out is that you may need to install certain libraries first for qqplotting and the heatmaps. Besides, that the trained models can be found in their respective directory per the instructions

LLM Usage

The LLM used in the assignment was ChatGPT's free model by openAI (if I recall correctly that's their 4.0 model with limited quiries). Its role has been mostly advisory, by suggesting alternatives to method's, trying to give an explanation to weird phenomena (like the worsened performance of the fine-tuned SVR model) and clarifying proper syntax and usage for various sci-kit’s methods and classes.

FIGURES

(For better versions of these images please refer to the respective notebooks)

Figure1 (qq-plot for BMI).

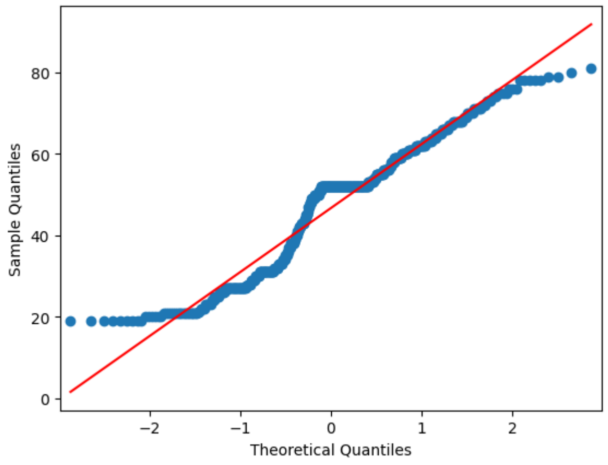


Figure2 (heatmap of the most correlated bacterium)

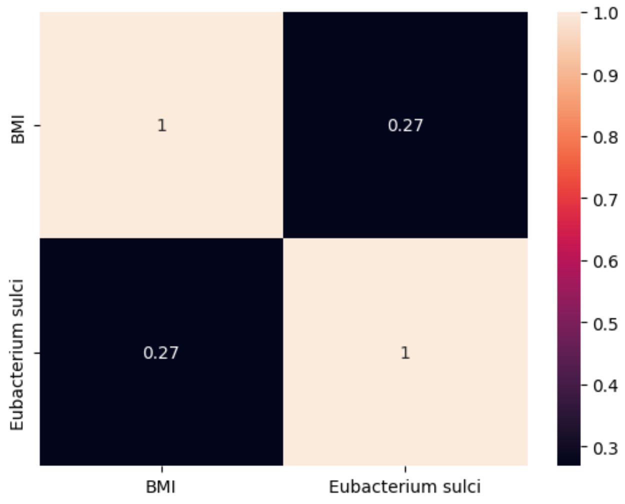


Figure3 (boxplot of baseline models perfromance)

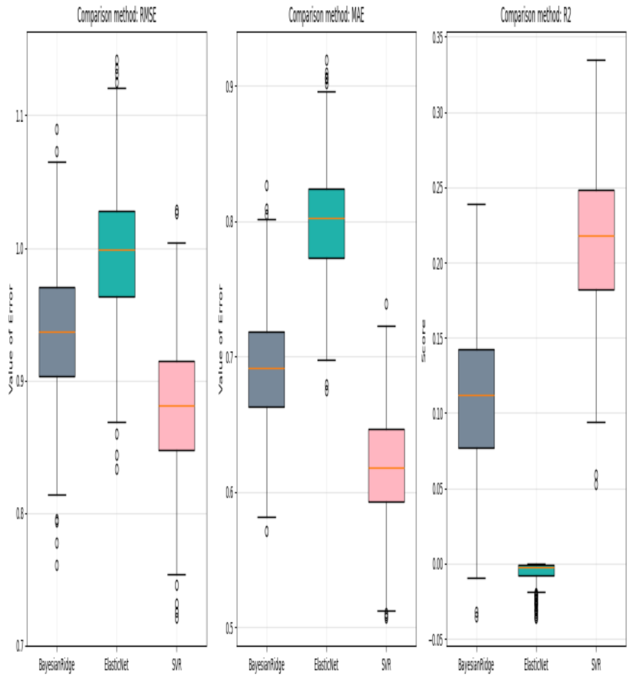


Figure5 (boxplot of tuning on feature selection)

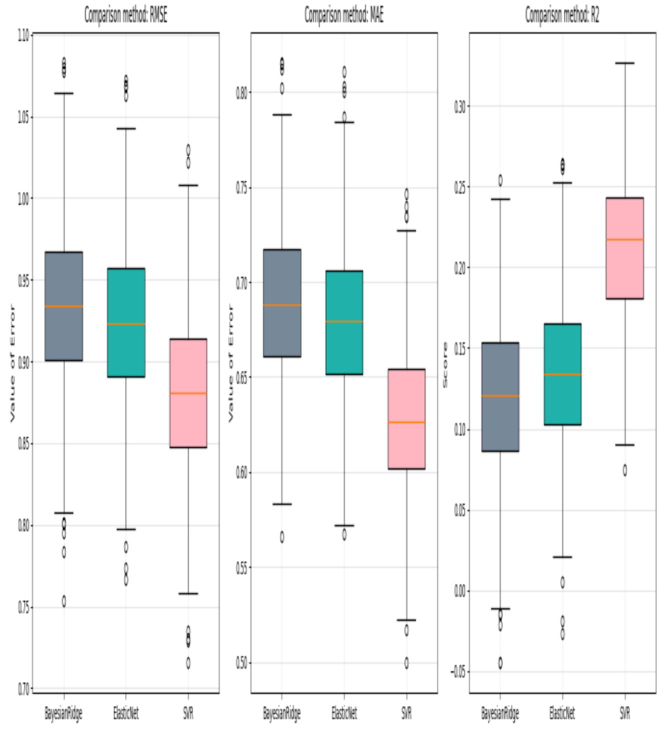
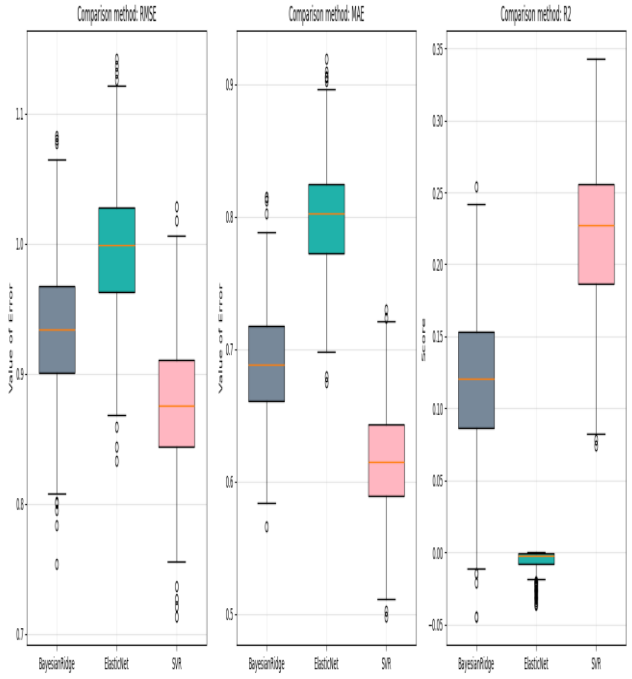


Figure4 (boxplot of feature selection and baseline models)



REFERENCES/ LINKS USED

- https://chatgpt.com/
- https://github.com/vex-me-not/Assignment-1
- https://scikit-learn.org/stable/index.html
- https://pandas.pydata.org/
- https://ojs-services.com/journals-publication-indexing/single-and-2-column-academic-scientific-word-templates/