Analisys and model comparison among several ML appraches

Authors:*Batignani Alessandro, Di Sante Matteo, Pecorella Francesco.*
Master Degree: Physics; curriculum: Complex Systems.
a.batignani@studenti.unipi.it, m.disante1@studenti.unipi.it, f.pecorella3@studenti.unipi.it.
ML course (654AA), Academic Year: 2022/2023
Date: 23/01/2023
Type of project: **B**

Abstract

The models implemented are KNN, MLP, SVM and Linear. For model selection we used a grid search approach, exploiting a KFold-CV. For model assessment we adopted a Hold-Out procedure. Comparing the results obtained in such a way, we found that the best performing model is KNN.

1 Introduction

The main focus of the project is to explore different ML paradigms, analyzing their general behavior and comparing that with what we have studied along the course. Having done so, it was possible to find the model that performed best over the regression task. The candidate models for this exploration are the following: linear model, KNN, Feedforward MLP, SVM.

2 Method

- The project was developed in Python3 and the library used for the implementation is Scikit-Learn. The supporting libraries used are Numpy, Pandas, Matplotlib and Seaborn.
- For linear model was used SGD training algorithm, with regularization techniques (Lasso and Ridge). For Feedforward MLP model were used: SGD training algorithm; an architecture up to 3 layers and 100 units; mini-batch with size 100; tanh, logistic activation function; random weights initialization (standard Sklearn approach); regularization L^2 (the only possible in Sklearn). The stopping criteria of MLP and linear models are treated in their respective sections. The core of a SVM is a quadratic programming problem (QP), separating support vectors from the rest of the training data.
- The preprocessing procedure is a standardization of the data (mean=0, $\sigma = 1$).
- For model assessment we adopted an hold-out procedure: 30% test + 70% design set. This procedure has been applied for each model. In order to evaluate the performances, the Mean Absolute Error (MAE - eq.1) is used when the model has only one variable as output, while the Mean Euclidean Error (MEE - eq.1) is used both for models with two output variables and for evaluating the performance of two one-variable models combined.

- For model selection we adopted slightly different strategies for each model, but all of them followed a common approach based on a 5-Fold-CV grid search.
- In the preliminary trials, for each model, we explored a set of hyperparameters values around the default ones suggested by the library.

$$MAE = \frac{1}{N}\sum_{p=1}^{N} \|y_p - f(\vec{x}_p)\|_1 , MEE = \frac{1}{N}\sum_{p=1}^{N} \left\|\vec{y}_p - \vec{f}(\vec{x}_p)\right\|_2 \quad (1)$$

3 Experiments

3.1 Monk Results

For SVM and KNN we decided to perform an encoded (ENC) and a not encoded (NENC) preprocessing procedure for the data. For MLP have been used an on-line SGD (batch_size=1) and the minimized *Loss* is the standard one that Sklearn uses for binary classification tasks, the Average Cross-Entropy:

$$Loss = -\frac{1}{n}\left(\sum_{i=0}^{n} y_i \log(f(\vec{x})) + (1-y_i)\log(1-f(\vec{x}))\right) + \frac{\alpha}{2n}\|W\|_2^2$$

Task	**Hyperparameters**	**MSE (TR/VL/TS)**	**Accuracy (TR/VL/TS)**
MONK1	4, invscaling 0.2, 0.1, tanh, 0	0/0/0.02	1/1/0.98
MONK2	3, invscaling 0.1, 0.1, tanh, 0	0/0/0.02	1/1/0.98
MONK3	3, invscaling 0.3, 0.01, tanh, 0	0.05/0.12/0.05	0.95/0.88/0.95
MONK3+reg	3, constant, 0.01, tanh, 0.1	0.06/0.08/0.03	0.94/0.92/0.97

Table 1: NN model. In the hyperparameters section, in order, are reported: number of units in the hidden layer, type of learning rate, power of the denominator of the learning rate in the case of invscaling, learning rate initialization value, activation function and regularization parameter.

Task	**Hyperparameters**	**Accuracy TR/VL/TS (± std)**
MONK1	NENC, 8, distance, distance_nominal	1.0(0.0), 0.82(0.07), 0.85
MONK2	NENC, 4, distance, distance_ordinal	1.0(0.0), 0.71(0.07), 0.84
MONK3	ENC, 42, distance, cityblock	1.0(0.0), 0.91(0.05), 0.95

Table 2: KNN model. In the hyperparameters section, in order, are reported: encoding (ENC) or not encoding (NENC) of data, best n_neighbors, weights and metric. In case of NENC we have two customized metrics: nominal and ordinal.

Task	Hyperparameters	Accuracy TR/VL/TS(± std)
MONK1	ENC, poly, 0.001, -10, 2, 100	1.0(0.0), 1.0(0.0), 1.0
MONK2	ENC, poly, 0.001, -0.1, 2, 100	1.0(0.0), 0.93(0.08), 1.0
MONK3	ENC, sigmoid, 1000, 0.01, 0.01	0.93(0.01), 0.93(0.05), 0.97

Table 3: SVM model. For Monk 1 and 2 hyperparameters reported are: encoding (ENC) or not encoding (NENC) of data, type of kernel function, C, r, degree and gamma. For Monk 3 hyperparameters reported are: encoding (ENC) or not encoding (NENC) of data, type of kernel function, C, r and gamma. In particular the winning kernels are given by the formulas: $k_{poly}(x, x') = (\gamma < x, x' > +r)^d$, $k_{sigmoid}(x, x') = tanh(\gamma < x, x' > +r)$.

3.2 Cup Results

3.2.1 Linear Models

For linear models we decided to use the Scikit-Learn library implementation SGDRegressor(), using stochastic gradient descent (SGD) as learning algorithm. First we implemented a linear model without any kind of regularization. After that we decided to use Lasso and Ridge regularization in order to check if there were any improvement. This model has also been implemented with a polynomial basis expansion (LBE). In particular, the LBE was implemented by a polynomial expansion of degree 2 that increased the number of features from 9 to 55.
For the model assessment we have used the Hold-Out procedure described before. Regarding model selection, first has been done a 5-Fold-CV with the aim of find the best hyperparameters. This operation has been repeated 10 times (RepeatedKFold()). After doing that, we split the design set into two parts: 80% training + 20% validation. Fixed the set of the best hyperparameters obtained through the grid search, we trained the model for a number of epochs such that it reached the plateau on the MSE validation learning curve. This procedure assures us to avoid an overtrained model.
Initially, grid-search were performed based on the range around the default values. After some trial runs to find the best ranges over which to operate the grid-search, it was decided to adopt a max_iter value of 3000 or 4000 in order to achieve training convergence. The tolerance ("tol") varies within the range [10^{-4}, 10^{-1}] as larger values led to irregular loss curves while smaller values would require a time consuming number of iterations. "eta_0" was chosen in the range [10^{-4}, 10^{-2}] and as "learning_rate" the library gave us the following choices: "invscaling," "adaptive" and "constant." The parameter "power_t" was taken to be 1/4 or 1/5. Finally "n_iter_no_change" was set to 50 or 100 depending on regularization (50 for unregularized linear and 100 for ridge).
α has been chosen from a vector of 100 components equally log-space in the range $[10^{-3}, 1]$. The results obtained for the best models have been reported in table (6) for each target variable. The results obtained by performing an LBE are significantly better than those obtained without. This is probably due to the fact that the original dataset did not have the structure of a linearly separable problem: LBE increased the degree of linear separability of the dataset, resulting in better performance of the linear model.

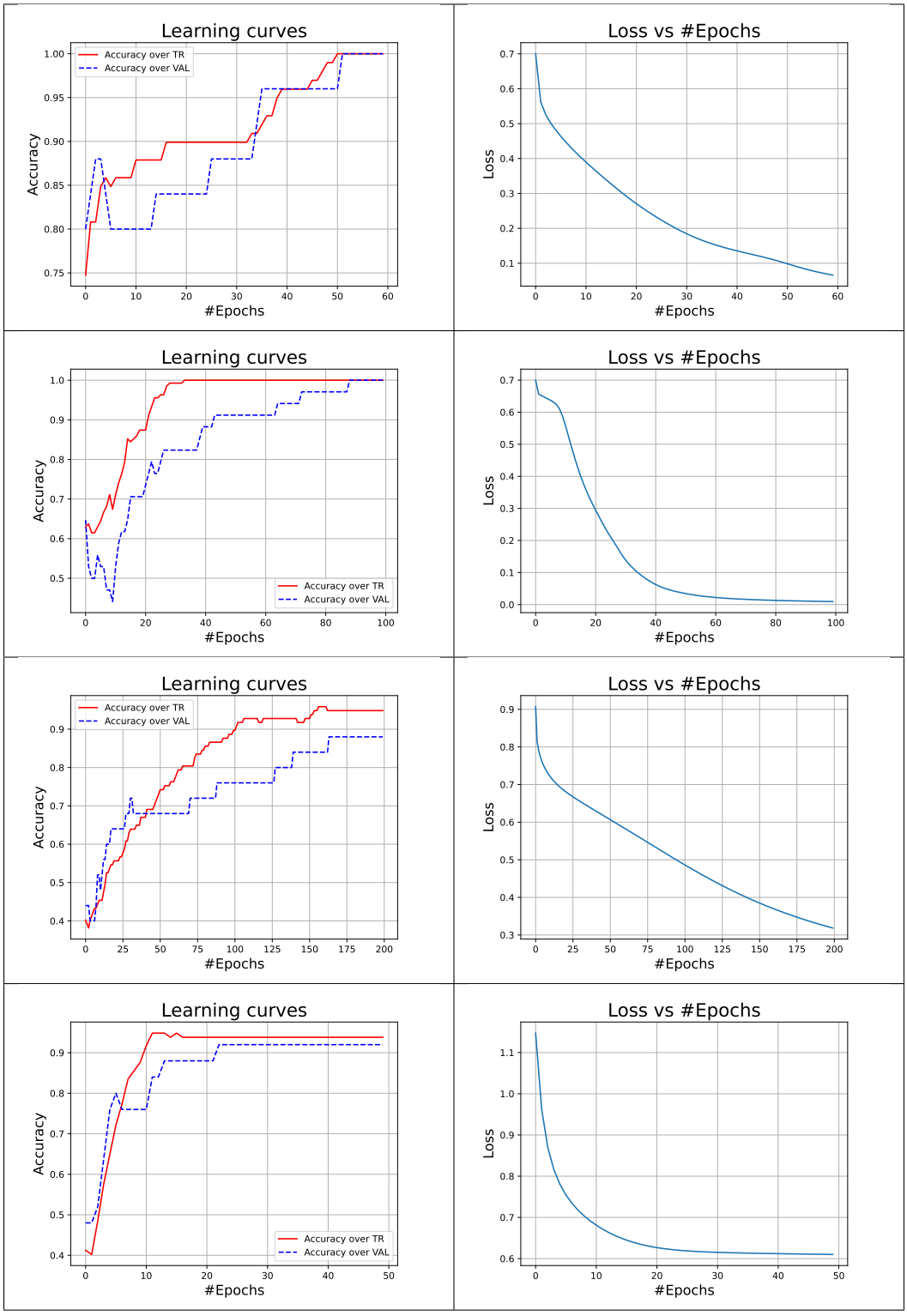

Table 4: From above: MONKS1, MONKS2, MONKS3, MONKS3+reg solved using MLPClassifier().

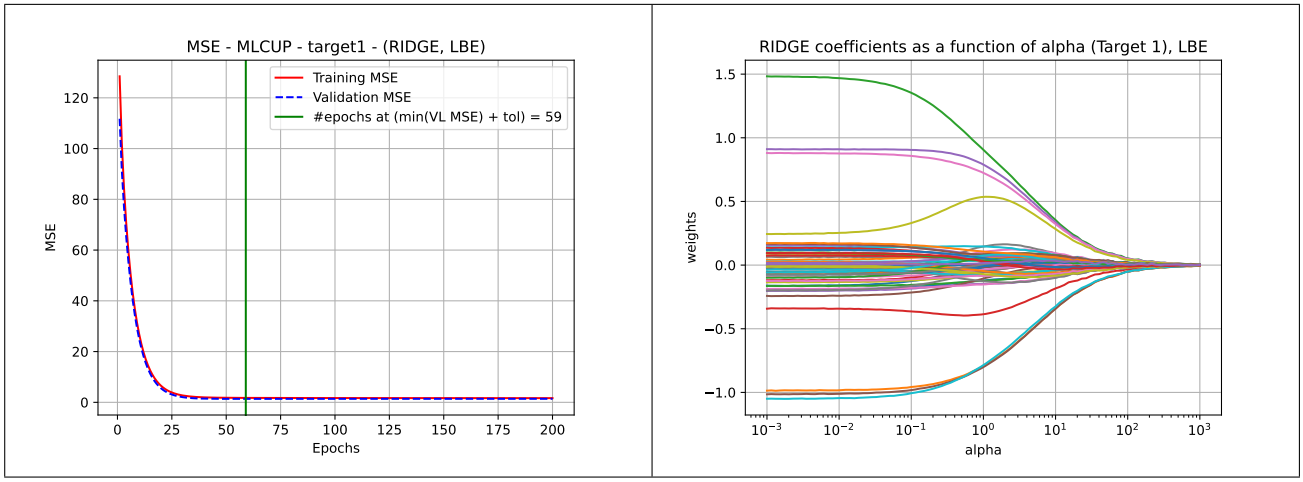

Table 5: *On the left:* MSE curves for Ridge regularization (LBE, target2) evaluated over TR and VL as the number of epochs changes. *On the right:* the linear coefficients of the best Ridge regularization (LBE, target2) in which the regularization parameter α was allowed to vary; it is observed that as α increases, the weights tend to 0.

Target	**Model**	**Hyperparameters**	**MAE TR/VL/TS**	**MEE TR/VL/TS**
1	Linear	Lasso, $\alpha = 0.016$	1.23, 1.09, 1.2	2.18, 2.01, 2.12
2	Linear	Ridge, $\alpha = 0.081$	1.52, 1.47, 1.50	
1	Linear(LBE)	Ridge, $\alpha = 0.043$	0.99, 0.95, 1.04	1.70, 1.76, **1.75**
2	Linear(LBE)	Lasso, $\alpha = 0.017$	1.14, 1.23, 1.15	

Table 6: Best linear models for target 1 and 2, including LBE

3.2.2 NN - Feedforward MLP

For the Feedforward NN we decided to use the implementation of the Scikit-Learn library and the standard SGD solver method. The procedure for selecting the best model is approximately the same as the one described in the Linear Models section. The main difference from what we have done before is the absence of the 10 repetitions of the 5-Fold-CV, infeasible because of being too time consuming for a NN grid search.
We choose to use just a single NN for solving both tasks and to explore no further than 3 layers. This choices have been taken to prevent time wasting grid searches (2 NN, one for each output variable, take more time to be explored; more than 3 layers can be too time consuming to train, due to the increasing number of weights to optimize). After some previous runs, we also decided to set momentum=0 to make more stable (smooth) loss curves. We also avoided early stopping, since we decided to visualize the curve of the best estimator found. In addition, we fixed the $batch\_size = 100$, after having seen that was more stable than the on-line method and faster in convergence than the full-batch. The tolerance parameter (tol) and $n\_iter\_no\_change$ have been fixed, to prevent the reaching of the maximum number of iteration for the SGD algorithm without a convergence. Table 7 shows the grid that allowed us to find the best model. For this grid we choose the invscaling learning rate, after having seen that was the best one for obtaining smooth results for the loss curves. In this way we limited our exploration only to the exponents $powet\_t$. Said that, we set the $learning\_rate\_init$ to values ≥ 0.001. The first three best models obtained with this grid differ only by the regularization factor (alpha). The mean learning curve and the loss for the best model are shown in table 9, while the results are given in table 8.

Hyperparameters	Set chosen
Activation	'tanh', 'logistic'
regularization	0, 0.01, 0.1
hidden_layer_size	(10), (100), (10,10), (10,10,10)
learning_rate_init	0.1, 0.01, 0.001
power_t	0.3, 0.4, 0.5, 0.6

Table 7: The grid search done for the MLPRegressor, with an *invscaling* learning rate type. NB: for hidden_layer_size the i-th element of the vector represents the number of neurons in the i-th hidden layer, e.g. (10,10) means 10 units in the first layer and 10 units in the second.

3.2.3 SVM

For the SVM we used SVR(), implemented by the Scikit-Learn library. This tool allows to predict one task at the time, so we performed two separate validation phase, both following the same reasonings. The approach used for model assessment is already discussed in the section Method. For model selection we adopted for each cell of the Grid Search a 5-fold CV repeated 10 times. The winning model that comes out from model selection was retrained over the entire design set. Our real goal is not to choose brutally the model with best validation score(i.e. MEE). We want to take into account the efficiency of the model as well (w.r.t. amount of data

Model	Hyperparameters	MEE TR/VL/TS
MLP NN	activation function: tanh, reg: 0.1, hidden_layer_sizes: 10 (1 layer), learning_rate: 'invscaling', LR_init: 0.1, power for the invscaling: 0.3	1.56, 1.61, 1.70

Table 8: Best set of h.params. found from the MLP grid search proposed in table 7.

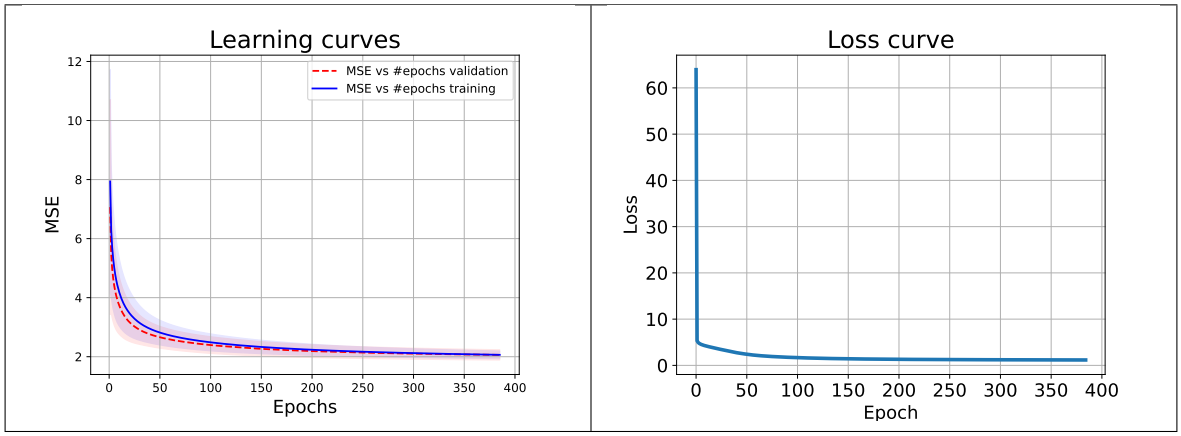

Table 9: On the left: the mean learning curves obtained with 100 iterations of the validation procedure for the best MLP model chosen (Table 8). The halo is the std dev associated to each point. It is not a noisy mean curve, nor very widespread. This shows that the training and validation procedures are stable. On the right: the *Loss* curve; notice the peak of the first value, due to the computation of the *Loss* for the randomly initialized weights.

Kernel	Target	C	ϵ	γ	fsv	MAE TR/VL/TS	MEE TS
RBF	1	11.9	1.0	0.07	0.3	0.74(0.01), 0.81(0.04), 0.80	1.51
RBF	2	3.8	1.0	0.14	0.35	0.98(0.02), 1.08(0.07), 1.09	

Table 10: Best SVM models for target 1 and 2

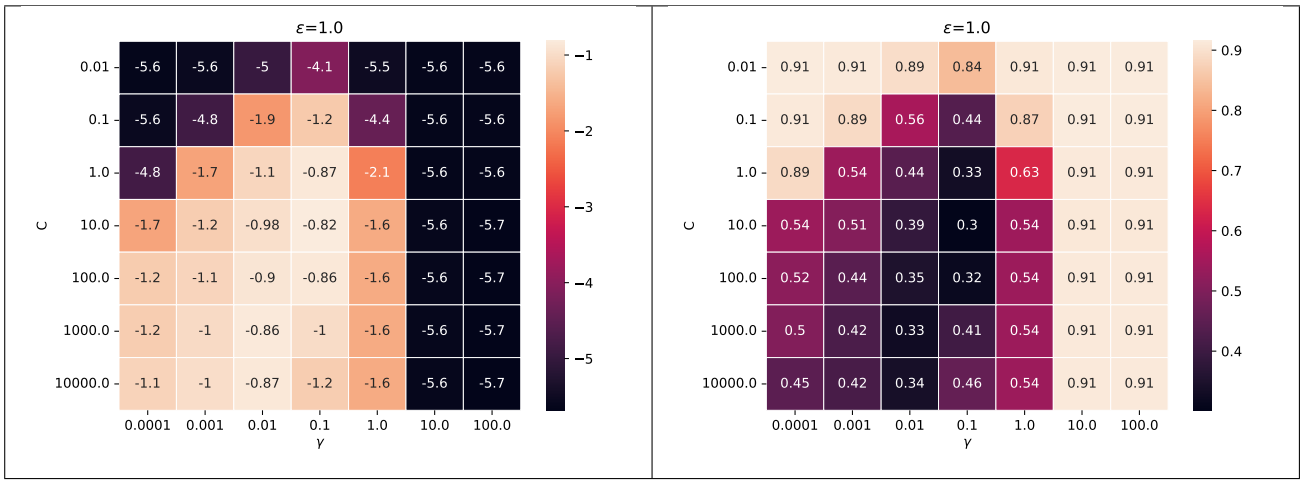

Table 11: Heatmap representing grid search projected over different values of ϵ

stored to make a prediction). For this reason we adopted a strategy that involved reasonings about the number of support vectors. A coarse grid search was performed for both target outputs using RBF kernel $k_{RBF}(x, x') = exp(-\gamma \|x - x'\|^2)$ and Sigmoidal kernel (see Table 3 for the formula), taking into account equally log-spaced values of C in the range $[10^{-2}, 10^4]$, γ in the range $[10^{-4}, 10^2]$, ϵ in the range $[10^{-2}, 1]$ and finally r in the range $[10^{-3}, 10^{-1}]$. To choose these values, we were inspired by the observed behaviors in the Monk and the well-known relationships between γ and C. Varying C and γ, and fixed the others hyperparameters, we observe that areas of grid search with higher validation score have lower fraction of support vectors (fsv) (look at the Table 11). In the previous context fsv provides an estimation of generalization capabilities, but varying the others hyperparameters it's not still true that lower fsv leads to higher validation score. The ϵ has a great influence on fsv, but it doesn't provide relevant differences in validation score. Given the same patterns accuracy for different fixed ϵ, we explored the one characterized by the lowest fsv, in order to provide a model with higher efficiency. For both kernel ϵ=1 leads to lower fsv. After further finer grid search around the model with best MEE, we obtained the final results. It comes out that RBF kernel is better in terms of validation MEE. The evaluation over TR, VL and inner-TS are reported in Table 10.

3.2.4 KNN

To implement regression via KNN, the KNeighborsRegressor() module of the Scikit-Learn library was used. This module allows prediction of both one target at a time and two together. Both the model with two output variables and two models with a single output were implemented, and the performance of the two approaches was found to be comparable. However,

Model	n_neighbors	Metric	Weights	MEE TR/VL/TS (± std)
KNN	15	Euclidean (2-norm)	distance	0.0 (0.0), 1.45 (0.07), 1.46

Table 12: Hypeparameters and evaluation scores for the best KNN found. The MEE equal to zero on the TR is probably due to the weights given to the points

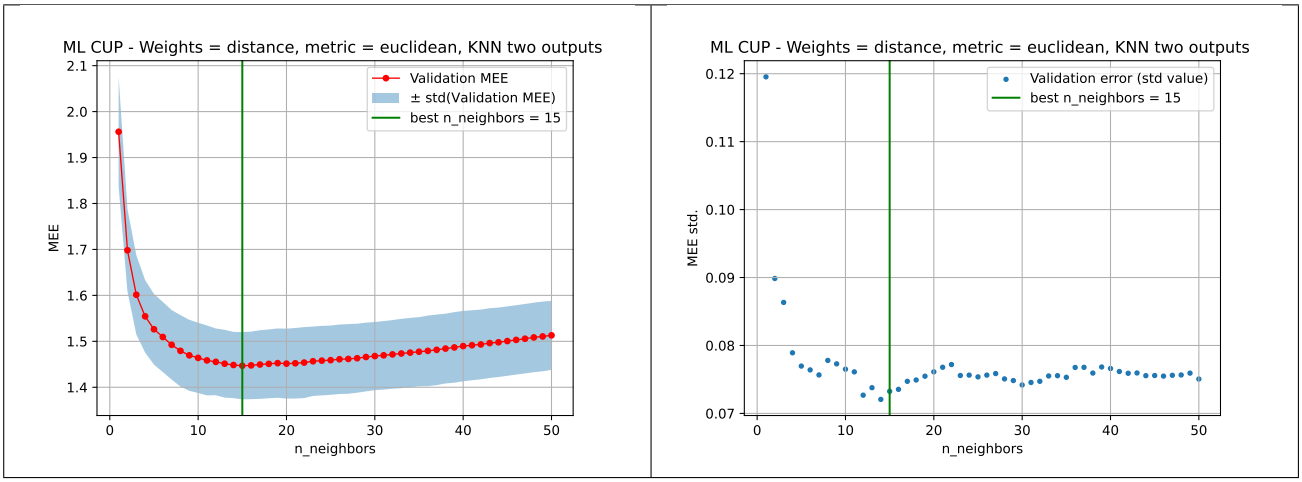

Table 13: Mean and std value of MEE as the number of neighbors varies, for a KNN with weighted points and euclidean metric

since the final results of the model with two outputs is slightly better it was decided to use this as the model for KNN. Regarding the procedure used for model assessment, the one already discussed in the Methods section was used. Regarding model selection a grid-search was used as an approach to find the best hyperparameters. A 5-folds CV repeated 10 times was implemented for each grid-search cell. The KNeighborsRegressor() module admits as hyperparameters: the number of neighbors, the metric used for calculating distances, and whether or not points have to be weighted by the inverse of their distance from a new point whose output is to be predicted. The hyperparameter "metric" admits as options: norm1, norm2 and cosine similarity. Looking at the first ten models of the grid search by order of MEE on the validation set, it is observed that these share Euclidean metric and all of them weights the points by the inverse of their distance ("weights" = distance). So the hyperparameter that most affects the MEE on the validation set is the number of neighbors. It is also clear from the graphs shown in the table (13) that the number of n_neighbors associated with the best model is also the one for which the minimum value of std(MEE) is obtained. This indicates that the result found in addition to being the best in terms of MEE score is also the one that returns the closest and most stable results to the mean value. The evaluation over TR, VL and inner-TS are reported in the table (12).

3.2.5 Computing times

To compare the computing times for each model we repeated a fit and a prediction on the same dataset for 10 times. From the vector obtained in such a way, we computed a mean and a

std dev. All of this procedure was performed over the same device: i7-1065G7 1.30GHz CPU with 16Gb of RAM. The results are shown in table 14. Some results are directly interpretable: all the SGD-based models take some time to fit data (the worst one is obviously the MLP), while the memory-based models (KNN, SVM) have the highest timing for the predictions. In principle we expect a compatibility between the timing of the KNN and of the SVM: in the library is written that both go like $O[DN^2]$ (D number of features, N number of samples).

MODEL	FIT TIME [s] ($\pm\sigma$)	PREDICTION TIME [s] ($\pm\sigma$)
MLP NN	0.72(0.05)	0.0002(0.0004)
SVM	0.04(0.01), 0.05(0.01)	0.04(0.02), 0.05(0.01)
LINEAR (no LBE)	0.03(0.01), 0.009(0.005)	0.0004(0.0005), 0.0001(0.0003)
LINEAR (LBE)	0.093(0.006), 0.06(0.01)	0, 0
KNN	0.0018(0.0004)	0.019(0.003)

Table 14: For every model: the mean fit time and the mean prediction time. If there are two models, two times are reported, one for each output.

4 Conclusion

The implementation of each model and its estimation of future performance by MEE on the inner-test set (TS) inferred that KNN is the best one. For this reason, it was decided to make the predictions on the blind test using the best KNN found. Regarding the linear models, it became clear that the LBE leads to better results. Another peculiarity that emerged in the study of the behavior of linear models is that the predictions on target 1 are better than those on target 2. The grid search for the NN has shown interesting results for the number of layers: just 10 units seem to be enough to solve the given task (based on the MEE over TS), regardless of the regularization factor. Eventually, RBF kernel represents a powerful tool for prediction and provides a more intuitive way to manage the complexity of SVM.
The file name containing the blind test predictions is "team_bruschetta_ML-CUP22-TS.csv". The nickname of the group is "team_bruschetta".

Acknowledgments

We agree to the disclosure and publication of our names, and of the results with preliminary and final ranking.