Machine Learning Project Report

Andrea Castagna, Computer Science - SW, a.castagna5@studenti.unipi.it
Martina Leocata, Informatica Umanistica, m.leocata1@studenti.unipi.it
Irene Mondella, Informatica Umanistica, i.mondella@studenti.unipi.it

ML course (654AA), Academic Year: 2022/2023
Date: 22/01/2023
Type of project: **B**

Abstract

In this report, we trained many Machine Learning models from different classes: MLP (Multi-Layer Perceptron), DT (Decision Tree), RF (Random Forest), KNN (K-Nearest Neighbors), and SVR (Support Vector Regressor), whose best models have been chosen through an exhaustive search and a k-fold cross-validation. The best model happened to be the RF, which therefore has been selected to predict the values of the blind test set.

1 Introduction

Our work focused on developing many different Machine Learning models, searching for the best combinations of hyperparameters, and training and evaluating them on different datasets, aiming to select the best performing one and to predict the target variables of the ML Cup's blind test set.

The classes of models we considered are Multi-Layered Perceptron, K-Nearest Neighbors, Decision Tree, Random Forest, and Support Vector Machine. All of them have been used both for classification tasks (on the MONK datasets) and for a regression task (on the ML-CUP dataset).

2 Method

In order to develop our models we used Python's libraries *Scikit-learn* (for partitioning datasets, developing Support Vector Machine, K-Nearest Neighbors, Decision Tree and Random Forest models, and evaluating performances) and *Keras* (for Multi-Layer Perceptron models), while for data manipulation we used *Pandas* and *Numpy* and for plotting results *Matplotlib* and *Seaborn*.

Before implementing our models, we conducted some data exploration, in order to better understand the data and to perform, whenever necessary, some preprocessing, e.g. *one-hot encoding* on categorical features or scaling on continuous ones. As for the MONK datasets, we implemented the *one-hot encoding* of all the features, while for the ML Cup dataset, after trying different possible scaling strategies, we found no significant changes in the performance of the models, therefore we decided not to scale the values and to keep the original distributions.

As for the *splitting* phase, we implemented different validation schemas depending on the dataset. Since the MONK dataset was already split into train and test set, we didn't perform a further Hold-out in order to split the data in train and validation set but a 5-fold stratified cross-validation. Instead, since for the ML Cup dataset we only had a train and a blind test set, we first partitioned our training set through Hold-out technique in order to obtain an internal test set, equal to 20% of the data, and, finally, we performed a 5-fold cross-validation.

Before searching for the best hyperparameters for each model, we performed some preliminary trials, to narrow our hyperparameter space and save up some computational time. For instance, while developing our MLP model we tried out different possible model architectures, such as different hidden layers or different units per layer; for the Random Forest Regressor we tested different depths and number of estimators; for the K-Nearest Neighbors we tested different possible values of k; for SVM we tested different C and γ parameters.

For the ML-CUP, finally, for each class of models, we implemented a Grid Search on the obtained hyperparameter space, in order to find the optimal configurations. After obtaining the best models, we tested them on the internal test set using the MSE, MAE, MEE, and r^2 as performance measures.

3 Experiments

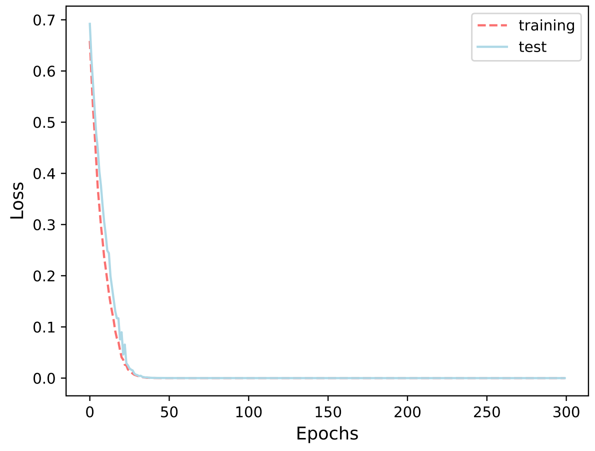
3.1 Monk Results

For each MONK dataset, we developed a MLP model with 17 input units, due to the *one-hot encoding* transformation of the features, 1 hidden layer made up of 6 units, and 1 output node, using 'ReLu' as hidden activation function, 'sigmoid' as output function, and RMSprop as optimizer. As for the other hyperparameters, such as *learning rate* (η), *momentum* (γ), and *weight decay* (λ), we performed a grid search. The best model was chosen according to the 5-fold cross-validation results (table 1).

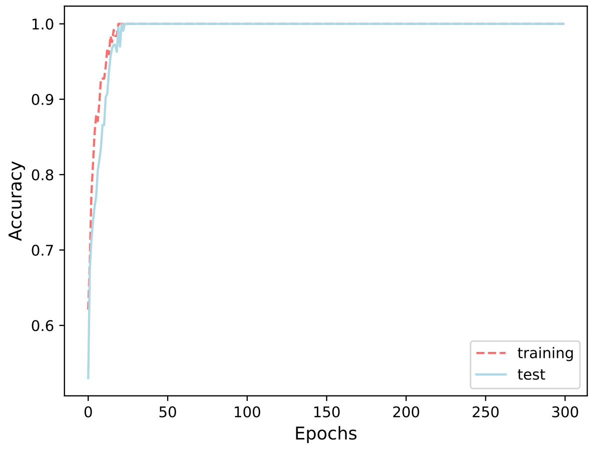
Task	$\boldsymbol{\eta}$	$\boldsymbol{\lambda}$	$\boldsymbol{\gamma}$	**MSE (TR/TS)**	**Accuracy (TR/TS)(%)**
MONK1	0.01	0.001	0.2	4.5e-11/5.5e-10	100% /100%
MONK2	0.01	0.001	0.0	2.7e-11/4.1e-11	100%/100%
MONK3	0.001	0	0.1	0.0089/0.027	99%/97%

Table 1: Performance of each best model for MONK's tasks with a MLP

As we can see from the plots (figures 1, 2 and 3), the best performing models are the ones implemented for MONK 1 and MONK 2, obtaining the maximum accuracy and very low MSE, and reaching convergence around the 25th epoch. Instead, for MONK 3 we obtained a 97% accuracy reached slightly before the 50th epoch.



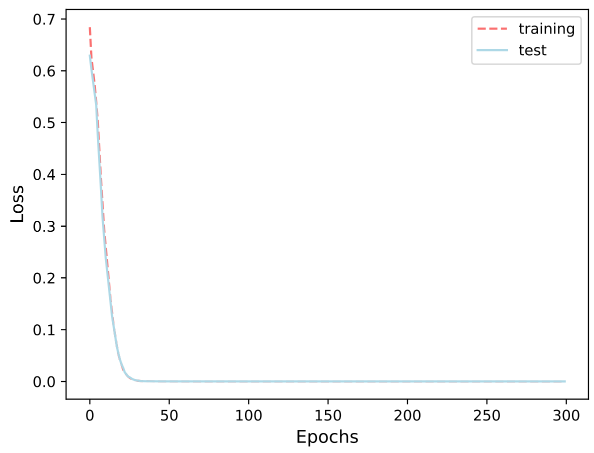
(a) MSE for MONK 1



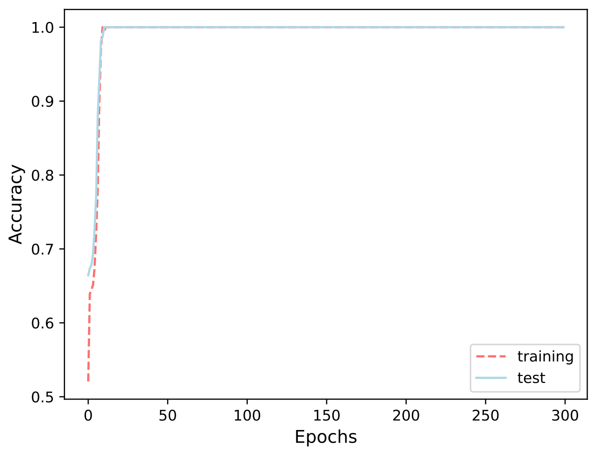
(b) Accuracy for MONK 1

Figure 1: Plots of the MSE and accuracy of the MLP for MONK 1

We also tested our other ML models: Decision Tree, Random Forest, Support Vector Classifier and K-Nearest Neighbors. The best models obtained through grid search, and their performance (MSE and Accuracy), are reported in tables 2, 3, 4 and 5.

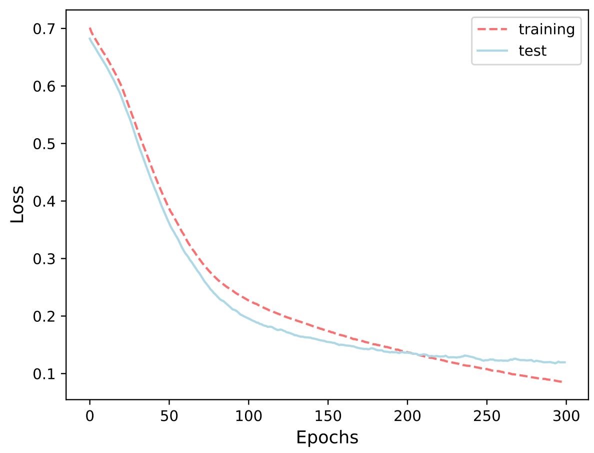


(a) MSE for MONK 2

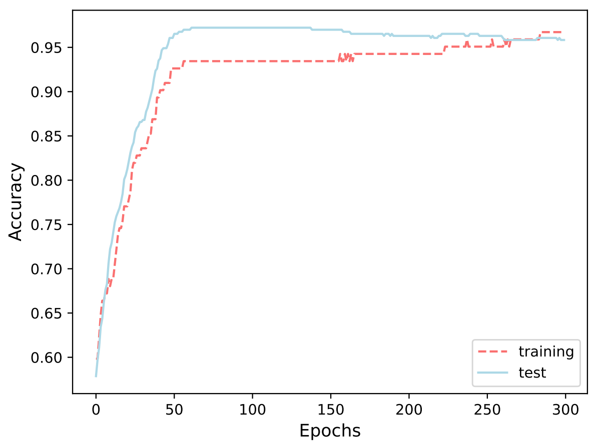


(b) Accuracy for MONK 2

Figure 2: Plots of the MSE and accuracy of the MLP for MONK 2



(a) MSE for MONK 3



(b) Accuracy for MONK 3

Figure 3: Plots of the MSE and accuracy of the MLP for MONK 3

Task	**ccp_alpha**	**criterion**	**max depth**	**Accuracy (TS)**
MONK1	0.003	gini	9	96%
MONK2	0.010	entropy	11	86%
MONK3	0.019	gini	4	97%

Table 2: Results obtained for the MONK's tasks with DTR

3.2 Cup Results

Before implementing our models, we performed some data exploration and found out that all the independent features have a normal or bimodal distribution, with a mean value around 0

Task	estimators	Max depth	Max features	Min sample leaf	Min sample split	Bootstrap	Accuracy (TS)
MONK1	15	8	log2	2	7	False	94%
MONK2	15	6	sqrt	2	7	False	76%
MONK3	15	6	sqrt	2	7	True	97%

Table 3: Results obtained for the MONK's tasks with RFR

Task	kernel	C	γ	Accuracy (TS)
MONK1	polynomial (degree 2)	0.125	2	100%
MONK2	polynomial (degree 2)	0.125	4	100%
MONK3	linear	0.125	0.01	97%

Table 4: Performance of each best model for MONK's tasks with a SVC

Task	K	algorithm	metric	weights	Accuracy (TS)
MONK1	8	ball tree	Manhattan	distance	87%
MONK2	21	bruteforce	Manhattan	uniform	65%
MONK3	36	ball tree	Manhattan	distance	94%

Table 5: Results obtained for the MONK's tasks with a KNN

and a standard deviation value around 1. Also, through the Scikit-learn function Select K-Best and the R-regressor method, we found correlation between every feature and each target variable, with different degree depending on which one.

Our strategy for building models was slightly different depending on the class of models: while in the case of MLP we developed a single model with two output units, one for each target variable, in the case of DT, RFR, KNN, and SVR we first trained a model for each target, in some cases using different param grids (e.g. in the case of DTR), and then we obtained a single ensemble model, in order to better evaluate them.

3.2.1 MLP

For MLP, we tested different network architectures: after some preliminary trials, we tried to implement 3 hidden layers with different possible number of units per layer, ranging from 20 to 200. We later implemented a grid search with 5-fold cross-validation. In this way, we searched for the best architectures and hyperparameters, deciding also to fix some of them, such as batch size, set to 10, loss function, set to MSE, and weight initialization, whose Keras' default value is *glorot uniform*. According to the optimizer, we decided to perform a grid search on slightly different parameters.

1. Optimizer: SGD

   Network architecture = {[20, 50, 50], [20, 50, 100], [50, 50, 100], [50, 100, 200], [100, 200, 200]}

   η = {0.001, 0.0001, 0.0001}

   λ = {0, 0.0001}

   γ = {0.5, 0.7 , 0.9}

Nesterov = {True, False}

2. Optimizer = RMSprop

   Network architecture = {[20, 50, 50], [20, 50, 100], [50, 50, 100]}

   η = {0.001, 0.0001, 0.0001}

   λ = {0, 0.0001}

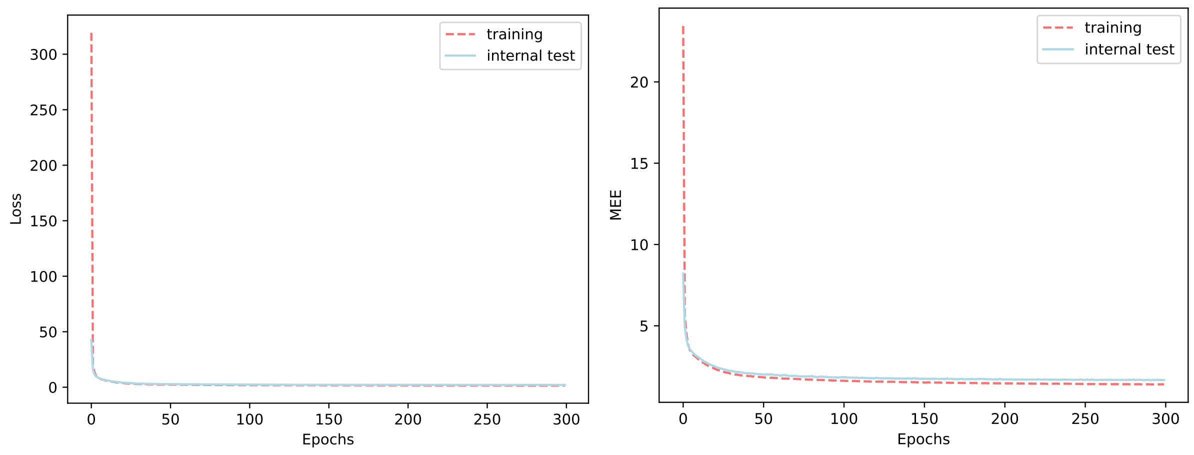
   γ = {0.0, 0.1, 0.2}

The best models found have the hyperparameters reported in table 6. As we can see, in almost all cases the best value for λ is 0 in both optimizers, suggesting that the weight initialization is already optimal or that the conformity of the dataset doesn't require regularization; as for the η, we noticed that the lowest value set in our grid was the best according to the results, suggesting that the best way to reach the global minimum is through a slow, smooth convergence instead of a fast one.

Network Architecture	**Optimizer**	$\boldsymbol{\eta}$	$\boldsymbol{\lambda}$	$\boldsymbol{\gamma}$	**Nesterov**	**TS MSE**	**TS MEE**
[50, 50, 100]	SGD	0.0001	0	0.5	True	1.521	1.389
[20, 50, 50]	RMSprop	0.0001	0	0.2	False	1.614	1.408

Table 6: Best MLP parameters and performance on internal TS

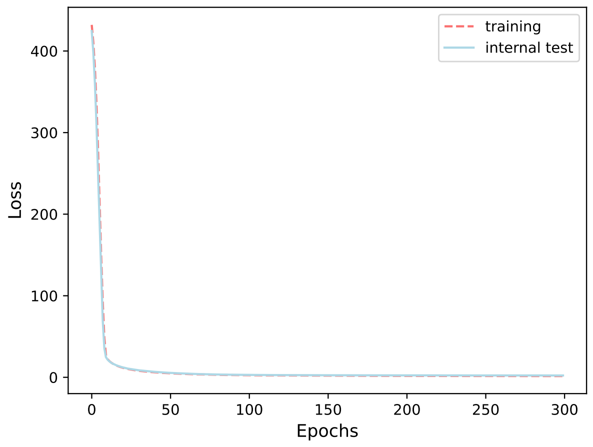
In figures 4 and 5 we can see the loss curves and the MEE curves of the best model with SGD and the best one with RMSprop.



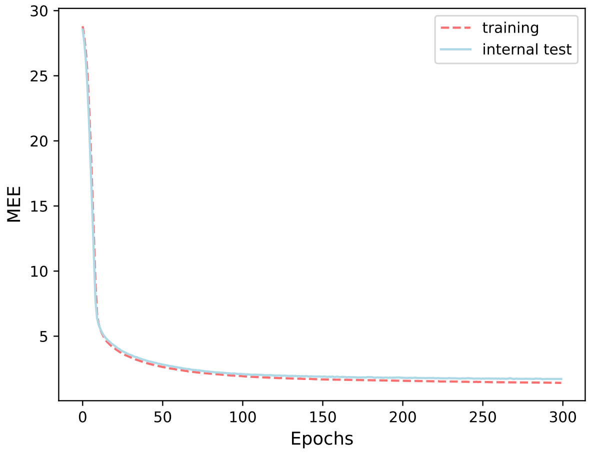
(a) Loss

(b) Mean Euclidean Error

Figure 4: Plots of the MSE and MEE for the best network with SGD



(a) Loss



(b) Mean Euclidean Error

Figure 5: Plots of the MSE and MEE for the best network with RMSprop

3.2.2 Decision Tree

The next model we implemented was the *Decision Tree Regressor*, whose best hyperparameters had been searched through a 5-fold cross-validation and MEE as loss function. Specifically, we searched for the best between the following values:

- *Criterion*: {squared error, Friedman mse, absolute error}
- *Max depth*: {4, 5, 6, 7, 8}
- *CCP_alpha*: $\left\{\frac{M}{50}i : i \in \mathbb{N}, 0 \leq i \leq 50\right\}$, $M \approx 3$ for the first target variable, $M \approx 1$ for the second target variable.

The best model found has the hyperparameters shown in table 7: as we can see, both the models built to predict the values for the first target variable and the second one have a best *ccp_alpha* value of 0.0, meaning that no regularization is needed for the model in order to avoid overfitting. Therefore, the data may either be not too complex or the dataset be small.

	Criterion	**Max depth**	**CCP_alpha**
First target	Friedman mse	6	0.0
Second target	squared error	5	0.0

Table 7: Best hyperparameters of the DT models

After finding the best hyperparameters, we tested the models on our internal test set, obtaining the results shown in table 8.

3.2.3 Random Forest

Since the results of the DTR seemed to be encouraging, especially for the first target variable, we decided to further investigate using a *Random Forest Regressor*, which combines several

	R^2	MSE	MAE	MEE
First target	0.924	3.300	1.064	1.064
Second target	0.763	4.059	1.338	1.338
Both targets	0.829	3.676	1.198	1.982

Table 8: Performance of the best DT models

decision trees in order to predict a target variable, meaning that it is a slower yet more reliable model. As for the DTR, we implemented a grid search in order to find the best values of the following hyperparameters:

- *Estimators*: {10, 50, 100, 250}
- *Max features*: {auto, sqrt, log2}
- *Max depth*: {4, 6, 8, 10}
- *Min sample split*: {6, 8, 10, 12, 14}
- *Bootstrap*: {True, False}

In table 9 we can see the hyperparameters of the best RTR models: both the models built for each target variable adopt the bootstrap method, meaning that some data points may be selected more than one time in order to create subsamples of data, each of them later used to train a specific tree of the forest. This result seems reasonable, given the small size of the data and the risk of overfitting because of the large number of estimators.

	Estimators	**Max features**	**Max depth**	**Min samples split**	**Bootstrap**
First target	100	log2	10	6	True
Second target	100	log2	10	6	True

Table 9: Best hyperparameters of the RFR models

After finding the best hyperparameters, we tested the models on our internal test set, obtaining the following results (table 10).

	R^2	MSE	MAE	MEE
First target	0.977	0.992	0.744	0.744
Second target	0.85	2.568	1.054	1.054
Both targets	0.892	1.795	0.899	1.456

Table 10: Performance of the best RFR models

3.2.4 KNN

The next model we implemented was *K-Nearest Neighbours Regressor* and searched for best hyperparameters in the following spaces, according to the MEE evaluated through a 5-fold cross-validation:

- K : $\{k \in \mathbb{N} : 0 < k \leq 60\}$,
- *Algorithm* : {bruteforce, ball tree, kd tree},
- *Metric* : {Manhattan, Euclidean, Chebyshev, Minkowski},
- *Weights*: {distance, uniform}

Selecting a relatively low number of neighbors ($\approx 2\%$ of training examples), computing Euclidean distance without any type of clustering and weighting them accordingly to their distance seems the best strategy to predict both target variables as shown by the following table (11).

	K	**algorithm**	**metric**	**weights**
First target	23	bruteforce	Euclidean	distance
Second target	24	bruteforce	Euclidean	distance

Table 11: Best hyperparameters of the KNN models

We then assessed the performance of the best models on our internal test set shown in the following table (12).

	R^2	**MSE**	**MAE**	**MEE**
First target	0.979	0.932	0.729	0.729
Second target	0.869	2.253	0.961	0.961
Both targets	0.924	1.592	0.845	1.360

Table 12: Performance of the best KNN models

3.2.5 SVR

The last model we implemented was the Support Vector Machine for regression. As hyperparameters, we considered:

- *Kernel function*: {linear, sigmoid, Radial Basis Function, polynomial}, considering polynomials of degree 0, 1, 2 and 3
- *Regularization term* C: {0.125, 0.25, 0.5, 1, 2}
- γ: {0.125, 0.25, 0.5, 1, 2}
- ϵ*-tube*: {0.0001, 0.001, 0.01, 0.1, 1}

The search for the best values of the hyperparameters was performed through a 5-fold cross-validation, comparing the Mean Euclidean Errors of the models. The two models we selected, one for each target variable, are described in table 13.

	Kernel	**C**	ϵ	γ
First target	RBF	1	0.1	0.125
Second target	RBF	1	0.1	0.25

Table 13: Best hyperparameters of the SVR models

These two models were tested on our internal test set, obtaining the results in table 14.

	R^2	**MSE**	**MAE**	**MEE**
First target	0.975	1.046	0.804	0.804
Second target	0.834	2.234	0.949	0.949
Both targets	0.905	1.640	0.877	1.411

Table 14: Performance of the best SVR models

3.2.6 Model selection for the ML Cup

In order to select the final model to predict the values of the blind test set for the ML Cup, we performed a k-fold cross-validation, setting k to 10 and testing all the models obtained for each grid search, using the MEE as a scoring function. In this phase, since we noticed little to no difference between the hyperparameters of the models obtained from grid search for each target variable for the DT, RF, KNN, and SVR classes, we decided not to use two but only one model to predict the target variables. In particular, we decided to pick the one developed for the second target variable, since there is a lower correlation value between it and the features and as backed up by empirical evidence, as we can see from the performance measures of the previous phase. In table 15 we report the best models' performances, MEE scores (mean and standard), and training times, according to our computation on Google Colab: as we can see, the best model we obtained for this dataset is Random Forest, which scored the lowest MEE and also seems the most stable one and the second best as training time, especially compared to the MLP models, which seem the most expensive and the most dependent on random initialization ones. The parameters of our chosen model are shown in table 16, and its learning curve in figure 6.

Model	**MEE** (mean)	**MEE** (std)	**Training time**
Random Forest	1.432	0.074	8.34 μs
MLP (RMSprop)	1.462	0.074	174 s
KNN	1.511	0.078	9.3 μs
MLP (SGD)	1.543	0.109	218 s
Decision Tree	1.823	0.077	10.6 ms
SVR	1.830	0.091	8.11 μs

Table 15: Performance of the best models after a 10-fold cross-validation

Estimators	**Max Features**	**Max Depth**	**Min Sample Split**	**Bootstrap**	**TR/VL/TS (MEE)**
100	log2	10	6	True	0.975/ 1.477/ 1.445

Table 16: Hyperparameters and performance of the final model

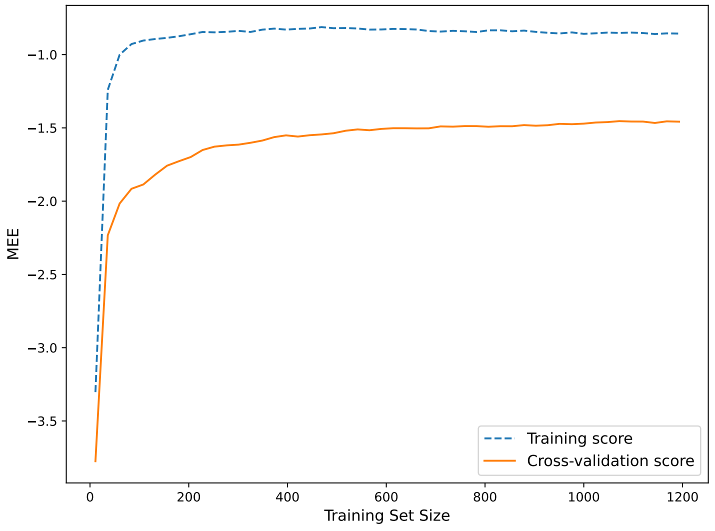


Figure 6: Learning curve of the Random Forest

4 Conclusion

The results of our experiments bring us to the conclusion that the best-performing model for this dataset is Random Forest, a simple yet effective approach, suggesting that for simpler tasks a more complex model may be not the optimal choice. Although the most powerful model is undoubtedly the MLP, it is also the one with the largest number of hyperparameters to fine-tune, for this reason it requires a very computationally expensive and time-consuming grid search. This implies two possible yet incomplete approaches: one consisting of a randomized search over a lot of hyperparameters or, like in our case, one based on a small grid search defined after several preliminary trials in order to restrict ”manually” the hyperparameter space. For simpler models, on the other hand, it is possible to perform an exhaustive search in a reasonable amount of time and through any ordinary hardware.

The final predictions of our selected model may be found in the file Hoddmìmir_ML-CUP22_TS.csv

Acknowledgments

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

Appendix A

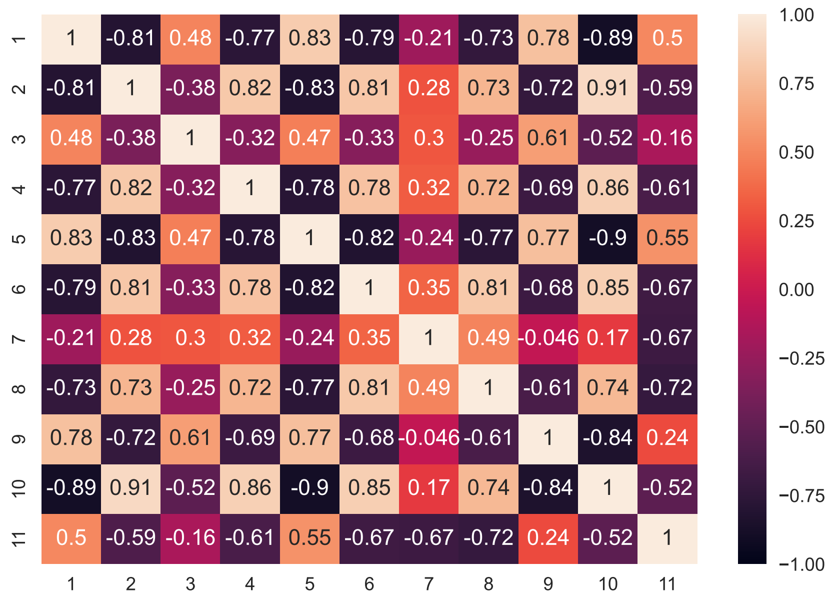
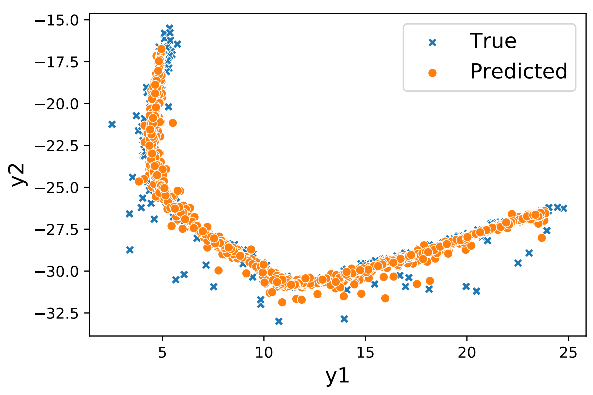
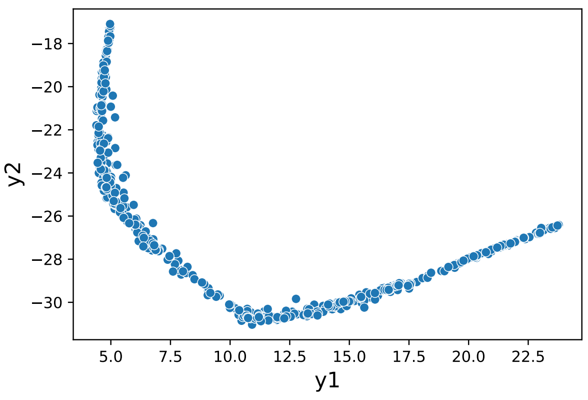


Figure A1: Correlation plot



(a) Y values



(b) Predicted y values of the blind test set

Figure A2: Scatter plots of the true and predicted target variables values