An extensive experimental setup of ML models for multivariable regression

Veronica Pistolesi, Francesca Poli

Computer Science Master Degree, AI curriculum.

v.pistolesi6@studenti.unipi.it, f.poli12@studenti.unipi.it

ML course 654AA, Academic Year: 2022/2023

Date: 01/06/2023

Type of project: **B**

Abstract

This project tests the effectiveness of multiple Machine Learning models and libraries on a regression task, namely: Decision Tree, Extra Tree, K-Nearest Neighbors, Support Vector Machine, Random Forest, Bagging and MLP. At the end, we implemented an ensemble model with the top five performing models.

1 Introduction

Our project's aim is to compare the results of different Machine Learning models and libraries applied to a regression task on the ML-CUP22 dataset. Our experiments cover the following models: Decision Tree, Extra Tree, K-Nearest Neighbors, Support Vector Machine, Random Forest, Bagging and MLP. The goal of this comparison is to identify the top models for this task and to try to confirm the expected results given the qualities of each model. Furthermore, an ensemble model consisting of the top five models was selected to be presented to the CUP competition. Some of the previous models were also implemented to carry out a classification task with MONK dataset [3].

2 Method

All the experiments were executed on a laptop with an `Intel Core i7-1255U` processor, belonging to the `x64` architecture, by using `Visual Studio Code`. The chosen models for the regression task are: *DecisionTreeRegressor* and *ExtraTreeRegressor* for the class **TREE**; *SupportVectorRegressor* for the class **SVM**; *KNeighborsRegressor* for the class **NEIGHBORS**; *RandomForestRegressor*, and *BaggingRegressor* for the class **ENSEMBLE**; three *MLP* models for the class **NN**. The classes were implemented by using `scikit-learn` [5] library but for the neural networks `SciKeras` [2] and `Skorch` [6] were also used. At the beginning, we extracted a development set (70% of the data) and an internal hold-out test set (30% of the data) from the ML-CUP22 dataset and the input features were normalized with the standard scaler transformation. Then, each model's hyperparameters were subjected to a grid search, with 5-fold cross-validation being used for each configuration of hyperparameters with `scikit-learn`'s `GridSearchCV`. Models' results were ranked and analyzed according to the Mean Euclidean Error (MEE) score on the validation set (20% of the data), before proceeding with a final model assessment phase. The parameter configurations tested for each model and more details on the implementation choices are presented in Section 3.2.

3 Experiments

3.1 Monk Results

The following steps summarize the experimental setting we adopted for this task:

1. One hot encoding to all the input features
2. Choice of four model classes for comparison (TREE, SVM, NEIGHBORS, and NN)
3. Grid searches on each model's hyperparameters, with stratified 5-fold cross-validation applied to each configuration.

3.1.1 Tree: DecisionTreeClassifier (`scikit-learn`)

M	ccp alpha	class weight	criterion	max depth	max leaf nodes	min samples leaf	min samples split	min weight fraction leaf	ACC (TR/TS)
1	0	None	Gini	10	30	1	2	0.01	$96,77\%$ / $89,81\%$
2	0.01	None	Entropy	100	100	1	2	0	$99,41\%$ / $86,34\%$
3	0	Balanced	Entropy	10	10	1	2	0.001	$95,90\%$ / $95,37\%$

Table 1: Best parameters and prediction results with Decision Tree algorithm

3.1.2 SVM: SupportVectorClassifier (`scikit-learn`)

M	C	coef0	gamma	kernel	shrinking	ACC (TR/TS)
1	1	0.5	Scale	Poly	True	100% / 100%
2	1.8	7	Scale	Poly	True	100% / $99,54\%$
3	0.02	1.5	Scale	Poly	True	$93,44\%$ / $97,22\%$

Table 2: Best parameters and prediction results with SVM algorithm

3.1.3 Neighbors: KNeighborsClassifier (`scikit-learn`)

M	algorithm	leaf size	metric	#neighbors	p	weights	ACC (TR/TS)
1	BallTree	25	Minkowski	10	5	distance	100% / 84, 49%
2	auto	10	Minkowski	21	1	uniform	66, 86% / 66, 20%
3	BallTree	2	Minkowski	18	1	distance	100% / 92, 82%

Table 3: Best parameters and prediction results with K-Neighbors algorithm

3.1.4 Neural Networks: MLP (`scikit-learn`, `SciKeras`, `PyTorch Lightning`)

M	topology	act	η	α	momentum	MSE (TR/TS)	ACC (TR/TS)
1	$(4, 4, 4,)$	TanH	0.5	0	0.7	$0, 0$ / 0.009	100% / 99, 07%
2	$(4, 4, 4,)$	TanH	0.9	0	0.7	$0, 006$ / $0, 037$	99, 41% / 96, 30%
3	$(2,)$	ReLU	0.7	0.1	0.9	$0, 057$ / $0, 06$	94, 26% / 93, 98%

Table 4: Best parameters and prediction results with scikit-learn's MLP

M	topology	act	η	λ	momentum	MSE (TR/TS)	ACC (TR/TS)
1	$(8,)$	ReLU	0.9	0	0.7	$0, 003$ / $0, 015$	100% / 97, 92%
2	$(8,)$	ReLU	0.2	0	0.9	$0, 0002$ / $0, 0003$	100% / 97, 92%
3	$(12,)$	ReLU	0.01	0.001	0.7	$0, 056$ / $0, 039$	100% / 97, 92%

Table 5: Best parameters and prediction results with MLP in Keras

M	topology	act	η	λ	momentum	MSE (TR/TS)	ACC (TR/TS)
1	$(8,)$	ReLU	0.1	0	0.1	$0, 0005$ / $0, 0027$	100% / 100%
2	$(8,)$	ReLU	0.1	0	0.1	$0, 0002$ / $0, 0003$	100% / 100%
3	$(12,)$	ReLU	0.1	0.001	0.3	$0, 019$ / $0, 028$	98.97% / 98, 15%

Table 6: Best parameters and prediction results with MLP in Pytorch

3.2 Cup Results

For this part of the experiments we proceeded as previously stated in Section 2. Details about each model can be found in the related Subsection.

3.2.1 TREE

For both the models *DecisionTreeRegressor* (DTR) and *ExtraTreeRegressor* (ETR) we performed a grid search as previously stated in Section 2. For these models we chose to test the same set of hyperparameter values (see Table 7) while keeping in mind their main difference, namely the splitter criterion which is *best* for DTR and *random* for ETR. The best two models selected by the search are a DTR with *ccp_alpha* = 0.0, *max_depth* = 7, *min_samples_leaf* = 6, *min_samples_split* = 2 and *min_weight_fraction_leaf* = 0.0 and an ETR with *ccp_alpha* = 0.0, *max_depth* = 15, *min_samples_leaf* = 1, *min_samples_split* = 2 and *min_weight_fraction_leaf* = 0.005. The grid searches lasted for about 3 and 1 minutes, respectively.

Hyperparameters	**Values**
max_depth	5, 7, 10, 15, 20, 25, 30, 50
min_samples_split	2, 3, 4, 5, 6, 8, 10
min_samples_leaf	1, 2, 3, 4, 5, 6, 8, 10
min_weight_fraction_leaf	0.0, 0.1, 0.5, 0.01, 0.05, 0.001, 0.005, 0.0001, 0.0005
ccp_alpha	0.0, 0.1, 0.5, 0.01, 0.05, 0.001, 0.005, 0.0001, 0.0005

Table 7: Grid search parameters for both DecisionTreeRegressor and ExtraTreeRegressor.

3.2.2 SVM

SupportVectorRegressor (SVR) is not specifically designed for multi-target regression but we were able to use it for this task thanks to `MultiOutputRegressor` wrapper class available in `scikit-learn` which extended it to handle multiple target variables. The range of tested hyperparameter values by grid search is shown in Table 8. The best model has $C = 3$, *coef0* = 0.0, *gamma* = *auto*, *kernel* = rbf and *shrinking* = $False$. The training and validation MEE were respectively **1.2855** and **1.4690**. The time required for the grid search was around 30 seconds.

Hyperparameters	**Values**
kernel	linear, poly, rbf, sigmoid
C	0.1, 0.5, 1, 2, 3
coef0	0.0, 0.01, 0.1, 0.5
gamma	auto, scale
shrinking	True, False

Table 8: Grid search parameters for SupportVectorRegressor.

3.2.3 NEIGHBORS

For the *KNeighborsRegressor* (KNR), a preliminary search for the ideal number of neighbors was performed by keeping the other parameters with their default values. Figure 1 illustrates how such parameter below 30 results in a lower MEE score. The range of tested hyperparameter values by grid search is shown in Table 9. The best model has *algorithm* = *auto* and *n_neighbors* = 16 and the training and validation MEE were **1.3488** and **1.4555**, respectively. The grid search required around 2 minutes.

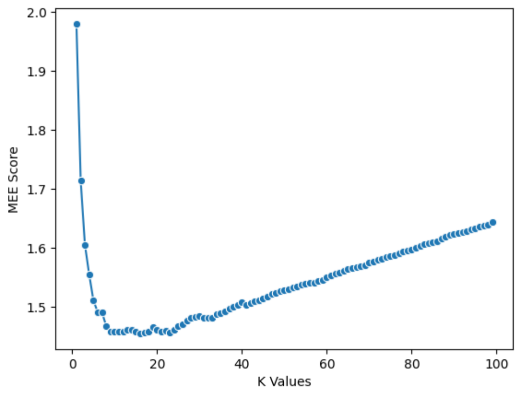

Figure 1: Preliminary search of the optimal number of neighbors for KNeighborsRegressor.

Hyperparameters	Values
n_neighbors	np.arange(2,30)
algorithm	auto,ball_tree, kd_tree, brute

Table 9: Grid search parameters for KNeighborsRegressor.

3.2.4 ENSEMBLE

Since we had more training samples available for the ML-CUP22 dataset than the MONK dataset (1044 as opposed to 124, 169 and 122), we decided to include tests with this class of models in this part of the experiments. The number of estimators is the ensemble models' most significant parameter. For this reason, an initial investigation of that number was made, leaving the other parameters unchanged, before proceeding with the actual grid search. The parameter's upper bound was set to 100 due to the still limited number of training samples available for this task. All the hyperparameters tested by grid search are shown in Table 10 and Table 11, for *RandomForestRegressor* (RFR) and *BaggingRegressor* (BR) resepctively. The best RFR has ccp_alpha = 0.0, max_depth = 10, $min\_samples\_leaf$ = 1, min_sample_split = 5, $min\_weight\_fraction\_leaf$ = 0.0 and $n\_estimators$ = 70 with a training MEE equals to **0.7417** and a validation one equals to **1.4792**. The time required for its grid search was around 5 minutes. The best BR has $bootstrap$ = $False$, $bootstrap\_features$ = $False$, $max\_samples$ = 0.3 and $n\_estimators$ = 70 with **1.0305** and **1.4753** as training and a validation MEE, respectively. The time required for this grid search was less than 1 minute.

Hyperparameters	Values
n_estimators	50, 60, 70
max_depth	None, 5, 10, 15
min_samples_split	2, 3, 5
min_samples_leaf	1, 5, 10
min_weight_fraction_leaf	0.0, 0.01, 0.05, 0.1
ccp_alpha	0.0, 0.01, 0.05, 0.1

Table 10: Grid search parameters for RandomForestRegressor.

Hyperparameters	Values
n_estimators	30, 40, 50, 60, 70
max_samples	1.0, 0.1, 0.3, 0.5, 0.7, 0.8, 0.9
bootstrap	False, True
bootstrap_features	False, True

Table 11: Grid search parameters for BaggingRegressor.

3.2.5 NN

For this class of models we decided to implement the same model (MLP) by using different libraries for its implementation. In particular, we used *MLPRegressor* from `scikit-learn`, *KerasRegressor* class from `SciKeras`, which integrates `Keras` functionality within `scikit-learn`, and *NeuralNetRegressor* class from `Skorch`, which instead integrates `PyTorch` functionality within `scikit-learn`. This allowed us to use in all three cases the `GridSearchCV` model selection method of `scikit-learn`, which is common to all CUP exepriments. The three models share the following implementation choices: the mini-batch stochastic method, also known as SGD; Mean Squared Error (MSE) as the loss function; batch size set at 16; maximum number of epochs set at 100; early stopping as the stop condition based on the validation loss with a tolerance of 20 epochs, possible through a 20% validation fraction; and the use of L2 regularization. The default minimum change in the monitored quantity to qualify as an improvement in the early stopping condition is zero but it has been changed into 0.01 in the non `scikit-learn` MLPs. Even with this kind of stopping condition, a previous testing activity demonstrated how more than 4 layers and 50 units can easily result in overfitting. Because of this, our final grid search does not test a larger number of them. For what concern the activation functions, ReLu and TanH proved to be the most effective while the Linear one was used for the 2-units output layer. For the weights and biases initialization we decided to take advantage of their default values: `scikit-learn` and `SciKeras` MLPs uses the Glorot uniform initializer [4], also known as the Xavier uniform initializer, which selects samples from a uniform distribution within the range $[-limit, limit]$ where $limit = \sqrt{\frac{6}{fan\_in + fan\_out}}$ as the kernel initializer and zero for the bias initialization, while `Skorch` MLP initializes both weights and biases from a uniform distribution within the range $[-limit, limit]$ where $limit = \sqrt{\frac{1}{fan\_in}}$ [1]. In this context, fan_out is the number of output units, whereas fan_in is the number of input units in the weight tensor. The hyperparameter values tested by grid search for the `scikit-learn` library are shown in Table 12 while for `SciKeras` and `Skorch` in Table 13. The best *MLPRegressor* has $hidden\_layer\_sizes = (50, 50, 50)$, $activation = tanh$, $learning\_rate\_init = 0.01$, $momentum = 0.0$ and $alpha = 0$ and the training and validation MEE are **1.4187** and **1.4628**, respectively. The grid search required around 12 minutes. The best *KerasRegressor* has $num\_hidden\_layers = 1$ (4 layers in total), $h\_units = 40$, $activation = tanh$, $learning\_rate = 0.01$, $momentum = 0.5$ and $weight\_decay = 0.0001$ and the training and validation MEE are **1.3976** and **1.4151**, respectively. The grid search required around 3 hours. The best *NeuralNetRegressor* has $num\_hidden\_layers = 1$ (4 layers in total), $h\_units = 40$, $activation = tanh$, $learning\_rate = 0.01$, $momentum = 0.0$ and $weight\_decay = 0.001$ and the training and validation MEE are **1.3627** and **1.4564**, respectively. The grid search required around 45 minutes. The MEEs of the last two models correspond to those of the models with the median validation MSE score among five retrainings with relative best parameters and different initializations.

Hyperparameters	**Values**
hidden_layer_sizes	(50,), (50, 50,), (50, 50, 50,), (40,), (40, 40,), (40, 40, 40,), (30,), (30, 30,), (30, 30, 30,)
activation	tanh, relu
learning_rate_init	0.001, 0.01, 0.1
momentum	0.0, 0.01, 0.02, 0.03
alpha	0, 0.0001, 0.001, 0.01, 0.1

Table 12: Grid search parameters for scikit-learn's MLP.

Hyperparameters	**Values**
num_hidden_layers	$0, 1$
h_units	$50, 40, 30$
lr	$0.008, 0.01, 0.02$
momentum	$0.0, 0.01, 0.03, 0.1, 0.5$
weight_decay	$0.0001, 0.001, 0.01, 0.1$
activation	ReLU, Tanh

Table 13: Grid search parameters for Skorch and SciKeras's MLP.

3.2.6 Model Assessment

Table 14 summarizes the scores for each of the models. For this part of the experiments we chose to take into account a group of the top models ranked by considering their validation MEE, in particular the first five. Therefore, the resulting ensemble consists of the three *MLP*s implementations, *KNeighborsRegressor* and *SupportVectorRegressor*, the ones highlighted in gray in Table 14. Due to the other models' validation MEE score and the perceived quite high disparity between training and validation MEE scores, which may indicate overfitting of the model, they have not been used for the final ensemble. In order to assess the MEE on the development set and the internal test set, extracted at the beginning of the task (see Section 2), we used the computed average of the predictions of the five selected models. The final results are **1.3113** and **1.4349**, respectively.

Model	**Validation MEE**	**Training MEE**	**GAP**
MLP (`SciKeras`)	1.4151	1.3976	0.02
KNR	1.4555	1.3488	0.11
MLP (`Skorch`)	1.4564	1.3627	0.09
MLP (`scikit-learn`)	1.4628	1.4187	0.04
SVR	1.4690	1.2855	0.18
BR	1.4753	1.0305	0.44
RFR	1.4792	1.7417	0.74
DTR	1.7264	1.1848	0.54
ETR	1.8040	1.5208	0.28

Table 14: Models' rank based on their validation MEE score.

4 Conclusion

Our research on this multi-target regression task reveals that Neural Networks (NN), K-Nearest Neighbors (KNN) and Support Vector Machines (SVM) are the most effective classes of models, while Ensemble and Trees tend to perform worse according to the MEE score. Even if a model performance could depend on many factors, this kind of behaviour confirms what we expected from a theoretical point of view due to the adaptability, capacity for generalization, and suitability for various data structures of NN, SVM and KNN. Ensemble techniques and Decision Trees, on the other hand, could have a propensity for overfitting and restrictions in modeling complex relationships.
For a complete plot perspective, please refer to Appendix A.
Our implementation code is fully replicable and available on `GitHub` at `https://github.com/VeronicaPistolesi/MachineLearning`. Our team's name is Ladydebugs and the final output for the CUP competition is in `Ladydebugs_ML-CUP22-TS.csv` file.

Acknowledgments

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

References

[1] Linear layer initialization in pytorch. `https://pytorch.org/docs/stable/generated/torch.nn.Linear.html#torch.nn.Linear`. Accessed: April, 2023.

[2] Adrian Garcia Badaracco. Scikit-learn wrapper for keras. `https://github.com/adriangb/scikeras/`, 2022. Accessed: April, 2023.

[3] Dheeru Dua and Casey Graff. UCI machine learning repository, 2017.

[4] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In Yee Whye Teh and Mike Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR.

[5] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[6] Marian Tietz, Thomas J. Fan, Daniel Nouri, Benjamin Bossan, and skorch Developers. *skorch: A scikit-learn compatible neural network library that wraps PyTorch*, July 2017.

Appendix A

Please note that with *learning curves* we refer to plots with MEE scores on the y-axis and **training set sizes** on the x-axis, whereas with *training curves* we refer to plots with MEE scores on the y-axis and **epochs** on the x-axis.

Monk plots

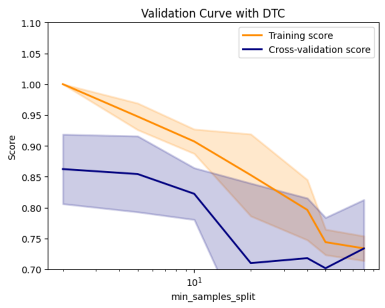

Figure 2: Learning curve of Decision Tree on MONK 1.

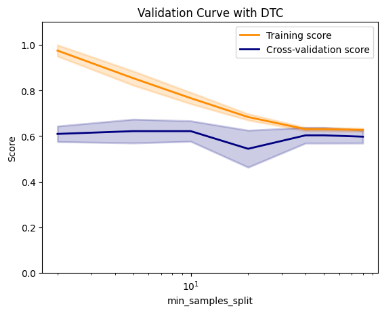

Figure 3: Learning curve of Decision Tree on MONK 2.

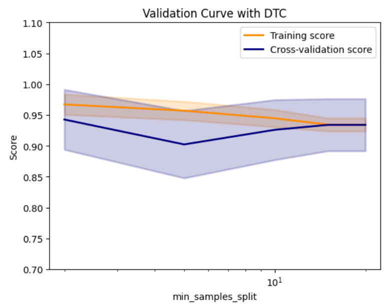

Figure 4: Learning curve of Decision Tree on MONK 3.

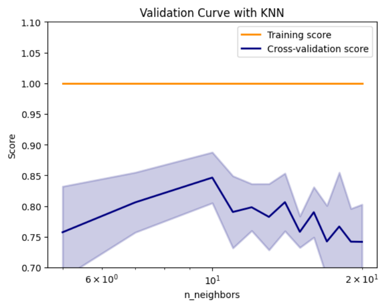

Figure 5: Learning curve of KNN on MONK 1.

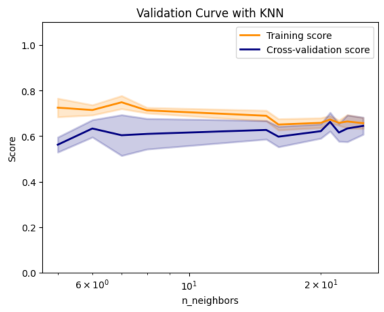

Figure 6: Learning curve of KNN on MONK 2.

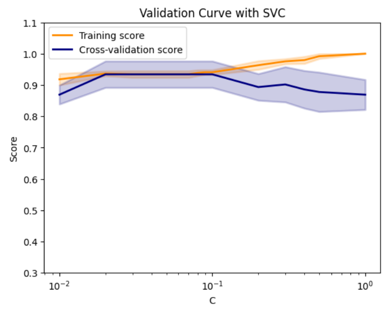

Figure 7: Learning curve of KNN on MONK 3.

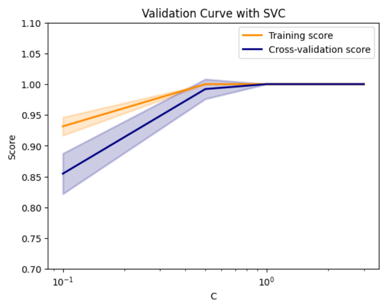

Figure 8: Learning curve of SVM on MONK 1.

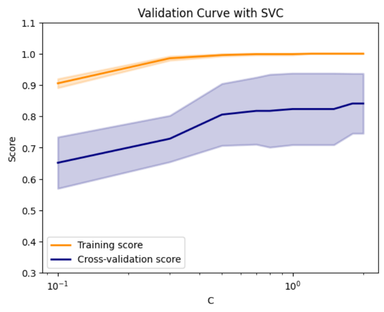

Figure 9: Learning curve of SVM on MONK 2.

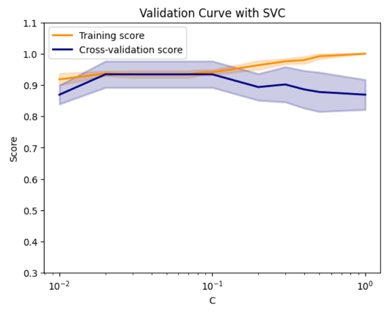

Figure 10: Learning curve of SVM on MONK 3.

MLP (scikit-learn)

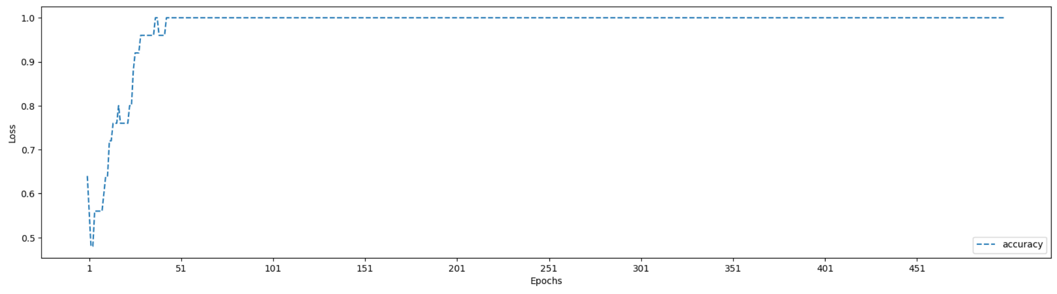

Figure 11: Accuracy in training curve of scikit-learn's MLP on MONK 1.

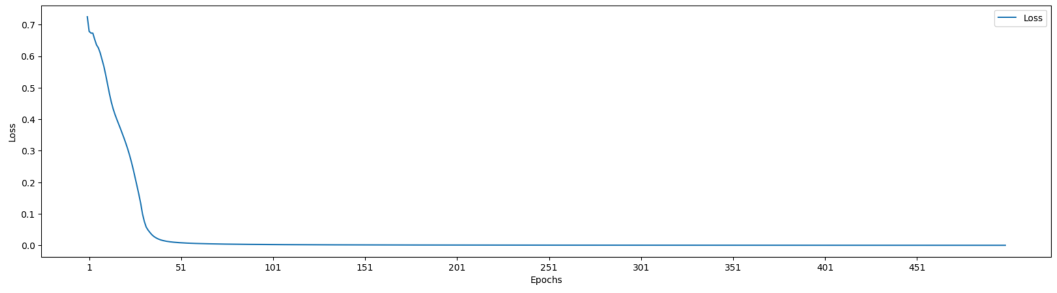

Figure 12: Loss in training curve of scikit-learn's MLP on MONK 1.

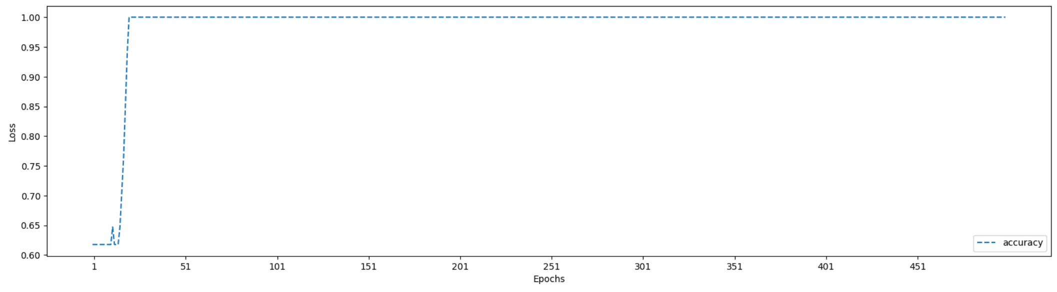

Figure 13: Accuracy in training curve of scikit-learn's MLP on MONK 2.

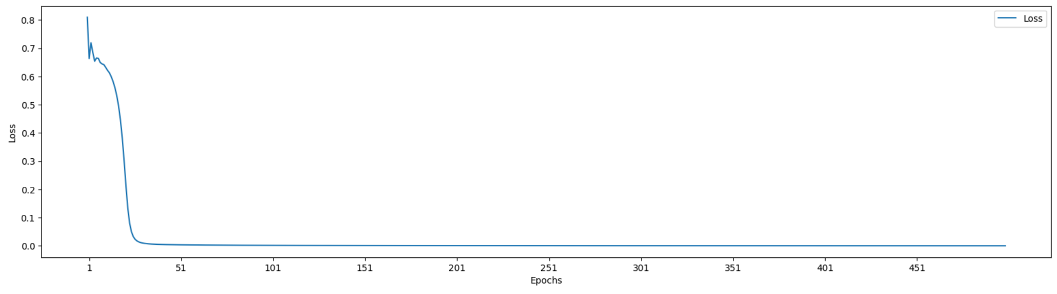

Figure 14: Loss in training curve of scikit-learn's MLP on MONK 2.

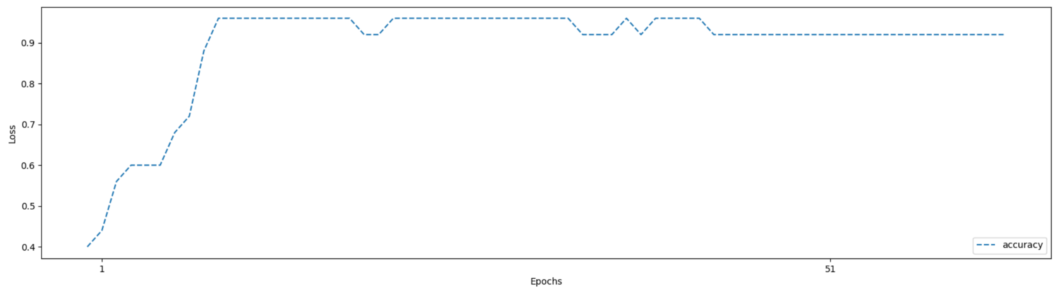

Figure 15: Accuracy in training curve of scikit-learn's MLP on MONK 3.

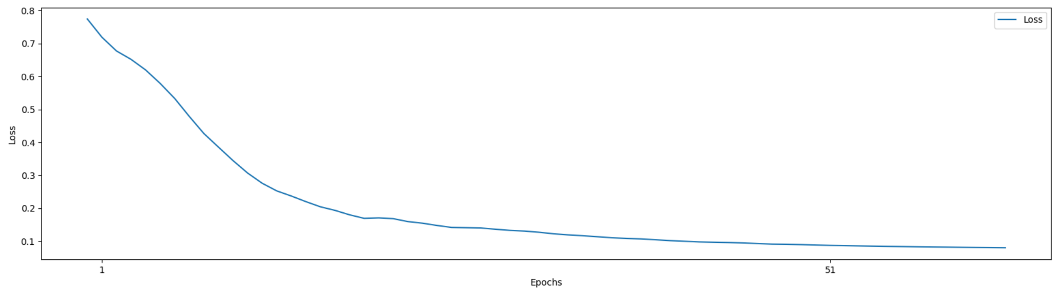

Figure 16: Loss in training curve of scikit-learn's MLP on MONK 3.

MLP (SciKeras)

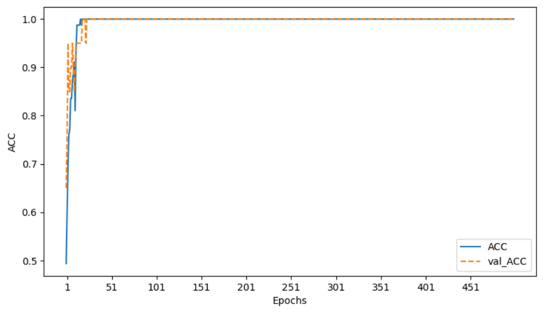

Figure 17: Accuracy in training curve of SciKeras' MLP on MONK 1.

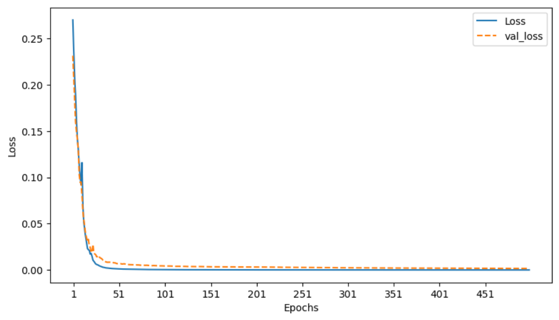

Figure 18: Loss in training curve of SciKeras' MLP on MONK 1.

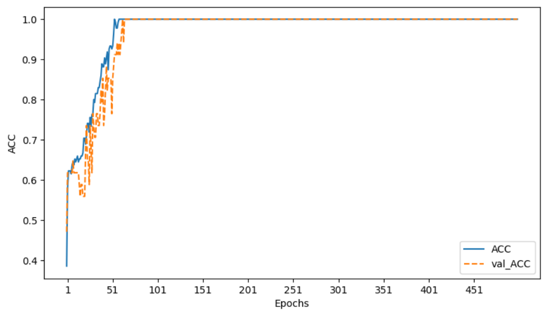

Figure 19: Accuracy in training curve of SciKeras' MLP on MONK 2.

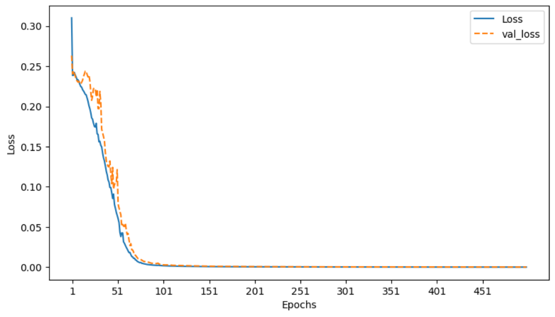

Figure 20: Loss in training curve of SciKeras' MLP on MONK 2.

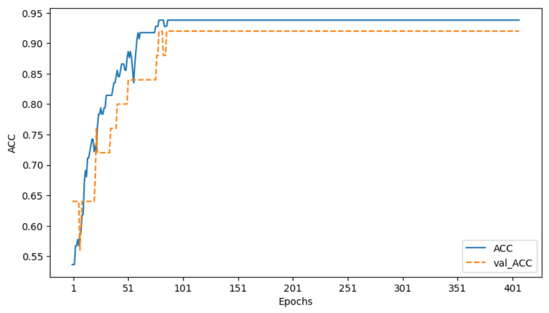

Figure 21: Accuracy in training curve of SciKeras' MLP on MONK 3.

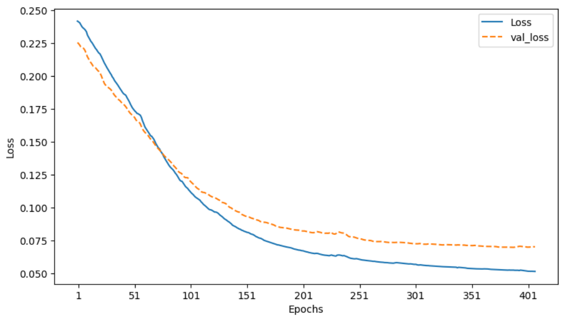

Figure 22: Loss in training curve of SciKeras' MLP on MONK 3.

MLP (Skorch)

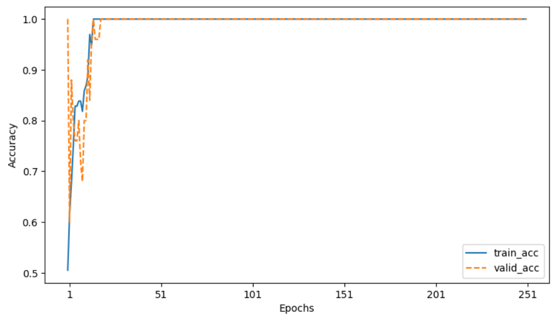

Figure 23: Accuracy in training curve of Skorch's MLP on MONK 1.

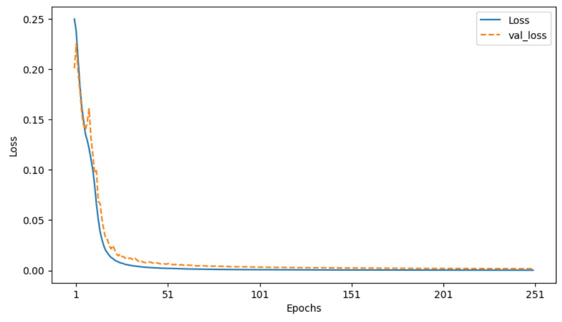

Figure 24: Loss in training curve of Skorch's MLP on MONK 1.

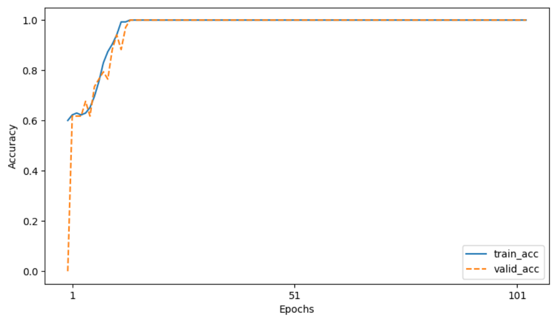

Figure 25: Accuracy in training curve of Skorch's MLP on MONK 2.

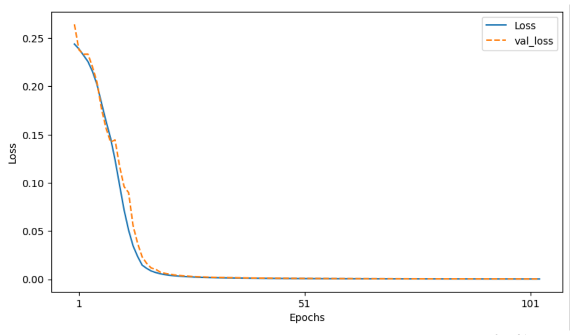

Figure 26: Loss in training curve of Skorch's MLP on MONK 2.

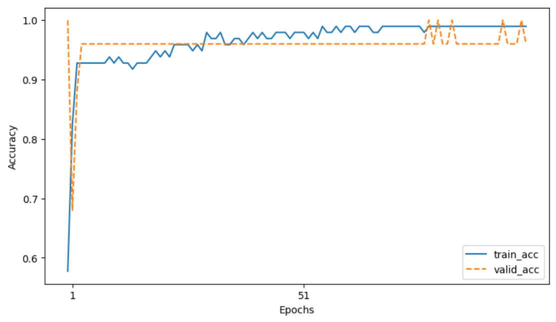

Figure 27: Accuracy in training curve of Skorch's MLP on MONK 3.

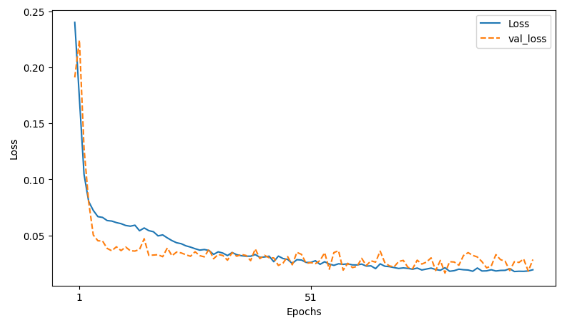

Figure 28: Loss in training curve of Skorch's MLP on MONK 3.

CUP plots

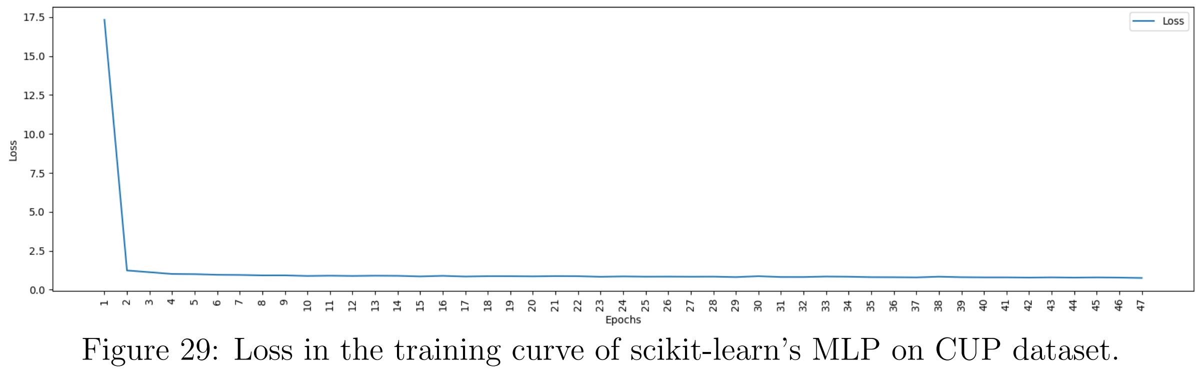

Figure 29: Loss in the training curve of scikit-learn's MLP on CUP dataset.

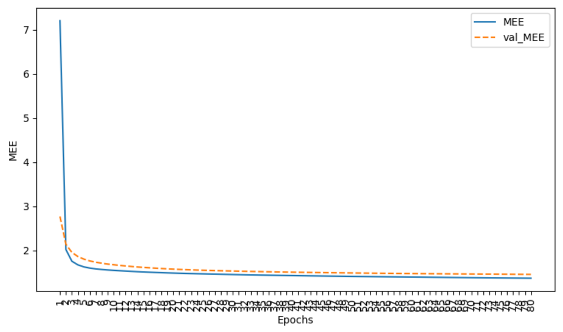

Figure 30: MEE in training curve of Skorch's MLP on CUP dataset.

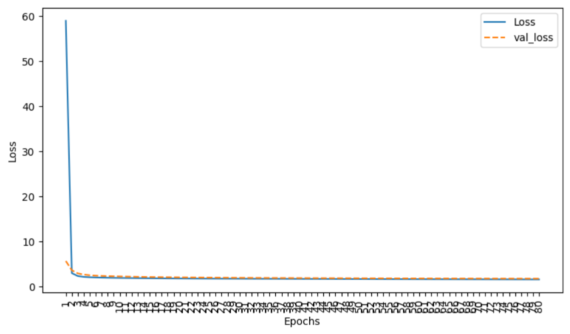

Figure 31: Loss in training curve of Skorch's MLP on CUP dataset.

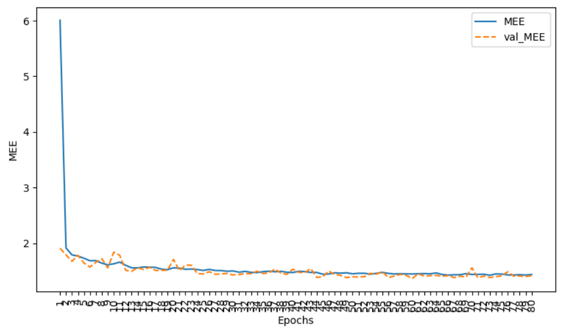

Figure 32: MEE in the training curve of SciKeras' MLP on CUP dataset.

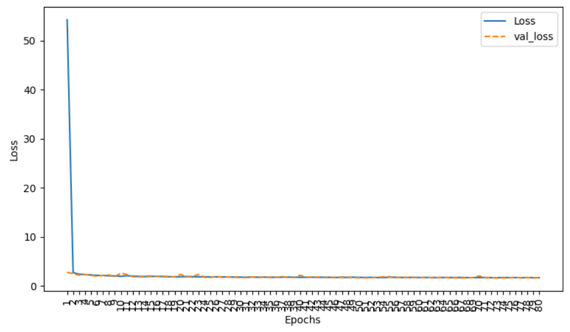

Figure 33: Loss in the training curve of SciKeras' MLP on CUP dataset.