Type B Project Report

Academic year 2019/2020

Motoc Stefan Daniel
stemotoc@gmail.com

Tenucci Gabriele
gabriele.tenucci@gmail.com

Machine Learning

30/01/2020

Abstract

In this report we train different models: Multi-Layer Perceptron (MLP), Random Forest (RF) and Support Vector Machine (SVM) on the given dataset. We've analysed performance and accuracy after choosing the best parameters for each model implementation, by using grid searches and k-fold cross validation. According to our results, the best model is the one obtained with Extremely Randomized Trees, which is therefore assessed on the test set and used as final model for the ML Cup.

1 Introduction

This report is divided in two parts. In the first one we solve a classification problem, the MONK dataset, using a different MLP model for each of the three datasets. After preprocessing the dataset with 1-of-k encoding for better precision, we have performed a grid search with k-fold cross validation and reported the performances of the best models for each task.

In the second part we solve a regression problem using different models. After splitting the dataset into Training and Test, for each model we have performed a grid search with k-fold cross validation. Our aim is to explore how these different models behave when applied on the same problem.

2 Method

We've developed our code in Python and tested it in a Jupyter notebook running both locally and on Google Colab. The MLP network was developed in Keras, using Tensorflow 1.14, and we have used ScikitLearn[2] for Random Forest, Extremely Randomized Trees (ETR)[3] and SVM (for both Radial Basis Function and Polynomial kernels).

First of all we have divided the dataset in two separate sections (Training and Test set). The test set is only used in the final testing phase, and it consists of 20% of the original dataset, taken randomly from the whole set.
We have tried using different Scalers on the dataset, but we have found no differences in the model performances, so we have decided not to use them to save computation time.

For every problem (including MONK) we have implemented a grid search to find the best hyperparameters, and in order to perform the model selection, we have used k-fold cross validation. In RF, ETR and SVM we have used 10-fold for both the grid search and the final training, while

for MLP we have used 3-fold on the grid phase, mainly because of how slow the training on this model is compared to the others, and 10-fold for the training of the final model.

For the MLP models we have tried dropout and weight decay, and compared the performances with and without them. We have also used minibatch, with batch size = 64 as a trade-off between timing constrains and model performances. We have decided not to implement any early stopping technique, working directly on the learning curves obtained in the training phase.

We've used Stochastic Gradient Descent (SGD) with Mean Squared Error (MSE) as a loss function and Mean Euclidean Error (MEE) as a metric for the model selection and the final test.

$$MSE = \frac{1}{N}\sum_{i=1}^{n}\Big(o_i - t_i\Big)^2 \qquad MEE = \frac{1}{N}\sum_{i=1}^{n}\Big\|o_i - t_i\Big\|_2$$

As activation functions for MLP, we have used Rectified Linear Unit (ReLU) for the intermediate layers, and Linear for the output layer.

For SVM we have decided to use two different kernels: RBF and Polynomial.

3 Experiments

3.1 MONK

For every MONK problem we've used a MLP net consisting of 17 input nodes (due to the 1-of-k representation), 7 nodes in the hidden layer, and 1 output node. The best hyperparameters have been found using a grid search on the learning rate (η), momentum (γ), weight decay (λ) and the number of nodes in the hidden layer (which turned out to be 7 for every MONK).
According to experimental results, we've also decided not to use Nesterov's momentum, and to set a batch size of length 8 for every problem, as our tests showed no significant performance decay compared to not using any batch.

We've used k-fold cross validation with a randomized validation split of 10%. Due to the low amount of samples, the obtained accuracy curves (see Fig.1, 2, 3) look very segmentated. Also, while MONK 2 reaches convergence early (less than 200 epochs), MONK 1 and 3 converge to the optimal values much later with our models.
The best hyperparameters found are shown in Table 1, along with each model's accuracy on the test sets.

Task	γ	η	λ	**MSE (TR/ TS)**	**Accuracy** (TR / TS)
MONK 1	0.9	0.5	1e-7	0.0001 / 0.0021	100% / 100%
MONK 2	0.9	0.035	1e-6	0.0001 / 0.0004	100% / 100%
MONK 3	0.7	0.0075	1e-6	0.0483 / 0.0525	93% / 97%

Table 1: Best models found for MONK.

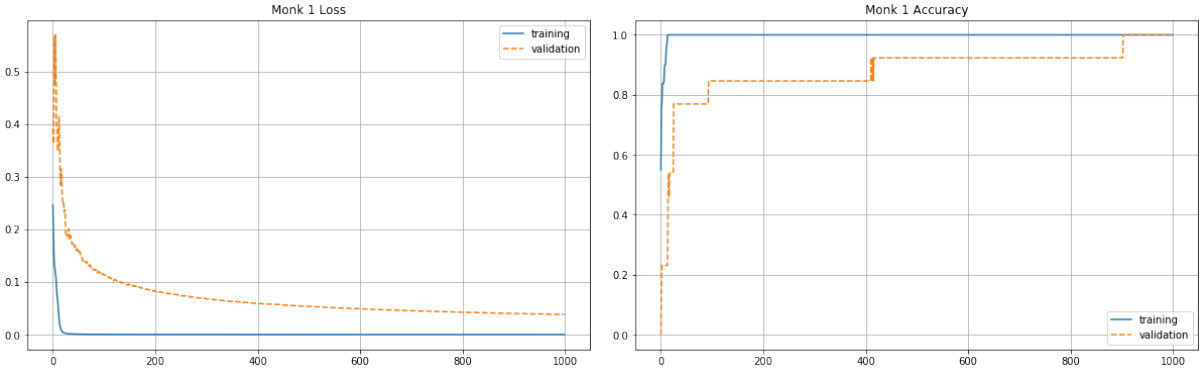


Figure 1: MONK 1 Loss and Accuracy curves.

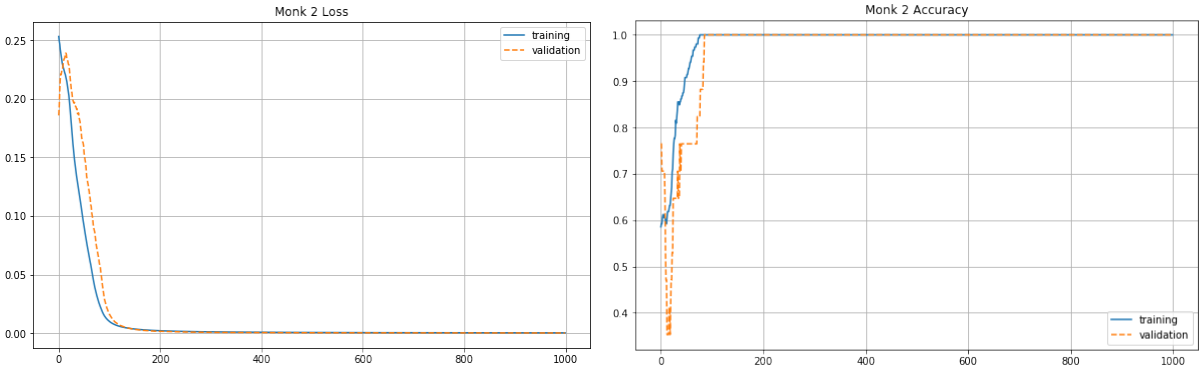


Figure 2: MONK 2 Loss and Accuracy curves.

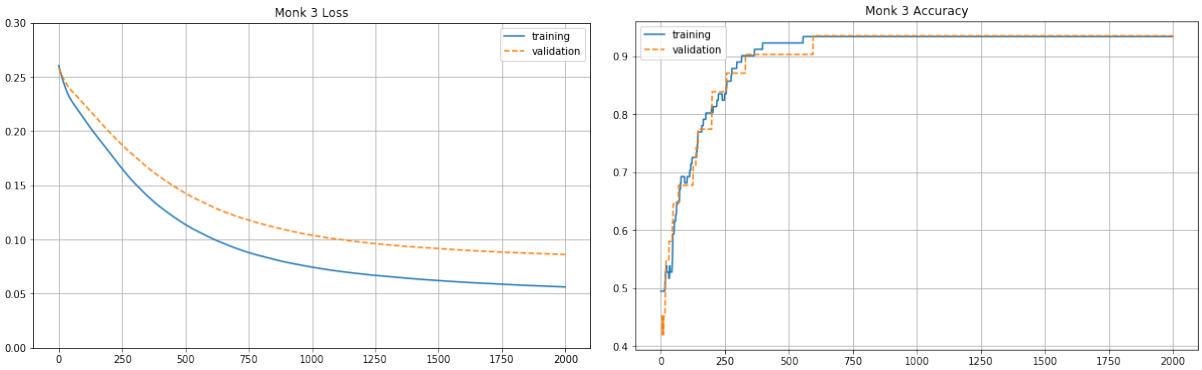


Figure 3: MONK 3 Loss and Accuracy curves.

3.2 Multi-Layer Perceptron

For our MLP models we have used the Keras implementation of fully connected networks, choosing SGD as optimizer with MSE as loss function. In the first phase of the search, we have decided not to use early stopping, and to look at the learning curves first.
We've implemented a grid search on the number of nodes and their configurations, the learning rate (η), the momentum (γ), and the weight decay (λ). For timing issues, the grid search was executed with 3-fold cross validation, while the batch size was set to 64 and the dropout value was fixed at 0.2. For the same same reason, we have performed the search on 200 epochs, under the (weak) assumption that the best model found for those epochs will also be among the best models after 1000 or more epochs.

The values on which we have performed the grid search are the following:

- $Configuration$ = [[100, 200, 300, 200, 100], [100, 200, 100], [200, 200]]
- η = [0.025, 0.05, 0.1]
- γ = [0.7, 0.9]
- λ = [0, 1e-3]

The best hyperparameters we have found are shown in Table 2.

Configuration	η	γ	λ	**VL MSE**	**VL MEE**
100x200x100	0.1	0.9	0.001	1.0182	1.0802
100x200x300x200x100	0.05	0.9	0.001	1.0138	1.0806
200x200	0.1	0.9	0.001	1.0366	1.0885
100x200x100	0.05	0.9	0.001	1.0175	1.1151
100x200x300x200x100	0.025	0.9	0	1.0780	1.1405

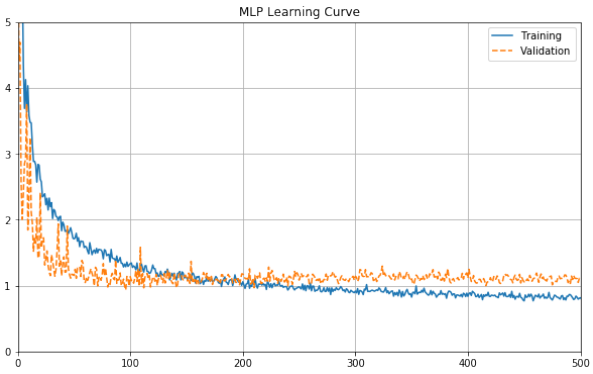
Table 2: Grid search results for MLP, ordered by MSE value on Validation.

Since the model with 3 layers and 400 nodes seems to be the best, we assume 3 layers are enough to obtain a satisfying model.

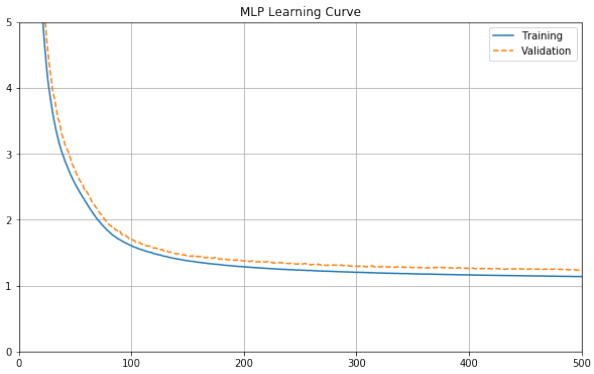
In Fig.4a we can see that the learning curve of the best model is discontinuous, so we have worked on the hyperparameters to obtain a smooth learning curve. Considering we only used 200 epochs in the grid search, we have come to the conclusion that the best grid search results shown in Table 2 had to have a higher than normal learning rate, as it is needed to reach convergence in little time. Because of that, we have decided to lower the learning rate (from 0.1 to 0.001), and to remove dropout, as our experimental results showed a smoother convergence when no dropout was used.

The resulting learning curve is smoother and, although the final accuracy seems to be slightly lower than the optimal model from the grid search, we have considered this model as our best MLP model, as it seems to be the most reliable one.

In Fig.4b we can see the learning curve of our chosen model.



(a) Noisy learning curve obtained by using the result of the grid search, with η=0.1 and dropout=0.2.



(b) Smooth learning curve obtained by using the same model with η=0.001 and dropout=0.

Figure 4: MLP Loss curve.

3.3 Random Forest

We've used an implementation of Random Forest regressor with SKlearn, by choosing MSE as loss function and k = 10 for the k-fold cross validation.
The number of iterations have been set to 100 for the grid search, which has been performed using the following values:

- $Number\ of\ estimators$ = [5, 10, 50, 100, 200, 250, 500, 1000]
- $Max\ depth$ = [2, 4, ..., 100]
- $Max\ features$ = ['auto', 'sqrt', 'log2']
- $Min\ samples\ split$ = [2, 3, 4, ..., 10]
- $Min\ samples\ leaf$ = [2, 3, 4, ..., 10]
- $Bootstrap$ = [True, False]

The results of the grid search are shown in Table 3, from which we can see that the best model achieves a low MSE value on validation even with a lower amount of estimators, which indicates a small forest.
There seems to be no pattern among the hyperparameters in the most performing models, so we have decided not to fine-tune ulteriorly the found models.

Estimators	**Min split**	**Min leaves**	**Max features**	**Max depth**	**Bootstrap**	**MSE**	**MEE**
200	4	2	log2	90	False	0.6288	0.7797
1000	6	2	log2	58	False	0.6318	0.7907
1000	5	3	sqrt	28	False	0.6529	0.8124
200	8	3	log2	68	False	0.6654	0.8242
250	10	2	log2	56	False	0.6622	0.8250

Table 3: Best models found by the grid search for Random Forest.

3.3.1 Extremely Randomized Trees

Given the good results of the RF models, we have used a different kind of implementation of Randomized Trees (Extra Trees Regressor from SciKitLearn), where the splits are chosen randomly for each feature, instead of choosing the best split as it normally happens in a Random Forest model. This allows the learning phase to take even less computational resources.

Since the approaches are very similar, we have decided to run the grid search for the ETR on the same hyperparameters used for Random Forest.

Estimators	Min split	Min leaves	Max features	Max depth	Bootstrap	MSE	MEE
250	4	3	auto	52	False	0.6004	0.7697
100	10	2	auto	50	False	0.6155	0.7915
200	2	2	log2	26	False	0.6378	0.8137
250	3	2	sqrt	34	False	0.6362	0.8139
200	7	3	auto	30	True	0.6772	0.8684

Table 4: Best models found for by the grid search on Extremely Randomized Trees.

As we can see from Table 4, the best values for the hyperparameters found with ETR are on average lower than RF, while keeping comparable (or even better) MSE values.
Fig.5 shows the learning curves of the best models we have found for Random Forest and Extremely Randomized Trees. As we can see from the curves, our ETR model reaches convergence more smoothly than the RF one, so we have considered this one as the best Randomized Trees approach.

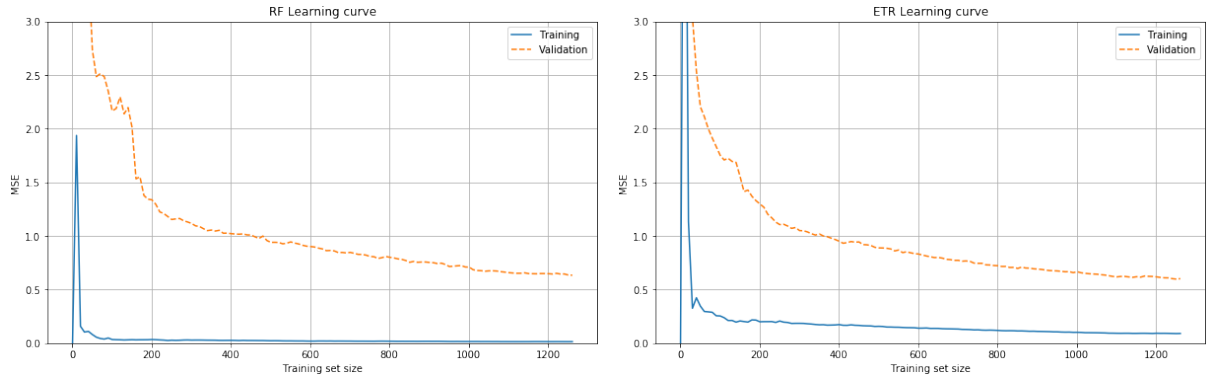


Figure 5: RF & ETR training curves.

3.4 SVM

As we assumed the hypothesis space to be non-trivial, we have chosen to use Radial Basis Function (RBF) as a complex kernel, and Polynomial as a simpler one (which still can represent a wide hypothesis space), skipping the linear one as it seemed to be too simple for our problem.
Initially, we have performed a grid search for each kernel to find the order of magnitude of the hyperparameters and reduce their range. Then we've performed another grid search to find slightly better values, and after that we've refitted a final model with the best hyperparameters found, using k-fold cross validation with 10 splits on the training set. During the grid search, we have also used the Shrinking method[1] to reduce the complexity of the model in the training phase.

3.4.1 RBF Kernel

For the RBF kernel we've performed a 10-fold grid search over the regularization parameter (C), the toleration margin (ϵ), and the kernel coefficient (γ) using the following values:

- C = [0.001, 0.01, 0.1, 1, 10, 100, 1000]
- ϵ = [0.001, 0.01, 0.1, 1, 10, 100, 1000]
- γ = ['scale', 'auto']
- *Shrinking* = [True, False]

Table 5 shows around which orders of magnitude we should look for the best hyperparameters.

C	ϵ	γ	**Shrinking**	**VL MSE**	**VL MEE**
100	0.1	scale	True	0.9467	1.0056
100	0.1	scale	False	0.9467	1.0056
100	0.1	auto	True	0.9443	1.0076
100	0.1	auto	False	0.9443	1.0076
100	0.01	auto	True	0.9632	1.0107

Table 5: Best orders of magnitude for SVM + RBF.

Then, we've performed another grid search in order to fine-tune the hyperparameters, starting from the results of the previous search. As we've seen in Table 5, the shrinking algorithm boosts speed, but doesn't change the accuracy of the model. Therefore, we have decided to set it as True from now on. The second search, whose results are shown in Table 6, was therefore performed with these ranges:

- C = [50, 75, 100, 125, 175, 250]
- ϵ = [0.01, 0.025, 0.5, 0.75, 0.1]
- γ = ['scale', 'auto']

C	ϵ	γ	**VL MSE**	**VL MEE**
125	0.1	auto	0.9460	1.0056
100	0.1	auto	0.9443	1.0076
125	0.1	scale	0.9471	1.0077
100	0.1	scale	0.9467	1.0107
75	0.1	auto	0.9474	1.0128

Table 6: Best hyperparameters for SVM + RBF.

Fig.8a shows an approximated learning curve for the best SVM model with RBF kernel, obtained by performing an incremental fitting with the same parameters. Fig.6 shows the heatmaps of the first and the second searches.

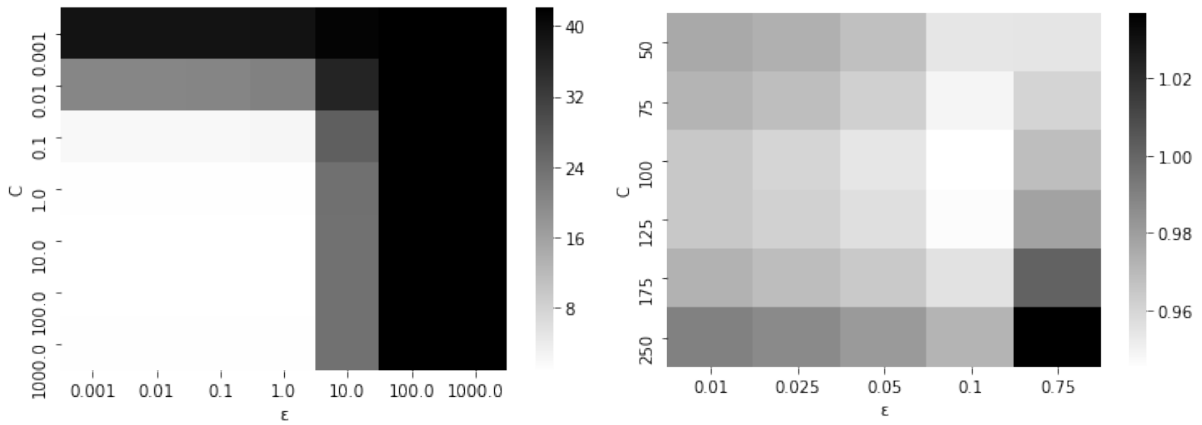


Figure 6: Heatmaps of the two grid searches. On the left the one that allowed us to find the order of magnitude of the parameters and on the right a more refined version.

3.4.2 Polynomial Kernel

For the Polynomical kernel the initial grid search was performed over parameters C and ϵ, again using k-fold cross validation, to find their order of magnitude. The results are the same as the ones in Fig.6. We've therefore fixed $\epsilon = 0.1$ and then we performed another grid search, until we got to narrower parameter ranges, shown below:

- C = [0.1, 0.5, 0.75, 1, 1.25]
- $Degree$ = [4, 5, 6]
- r $(coef0)$ = [0.6, 0.7, 0.8, 0.9]

Fig.7 shows the heatmaps of the parameters of the final grid search, that allowed us to obtain the values in Table 7. As we can see in Fig.8, the Polynomial kernel model achieves results that are similar to those of the model based on RBF kernel, while converging slightly slower but with a smoother curve.

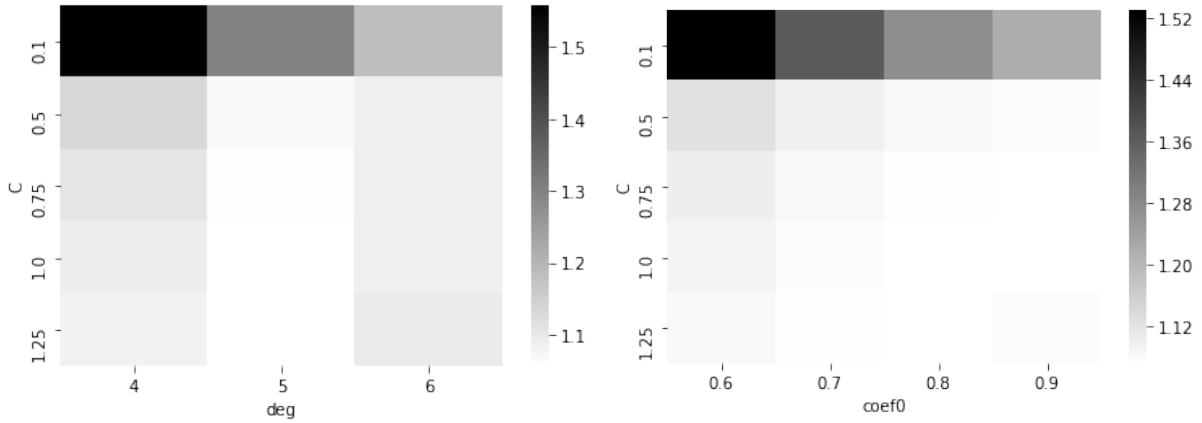
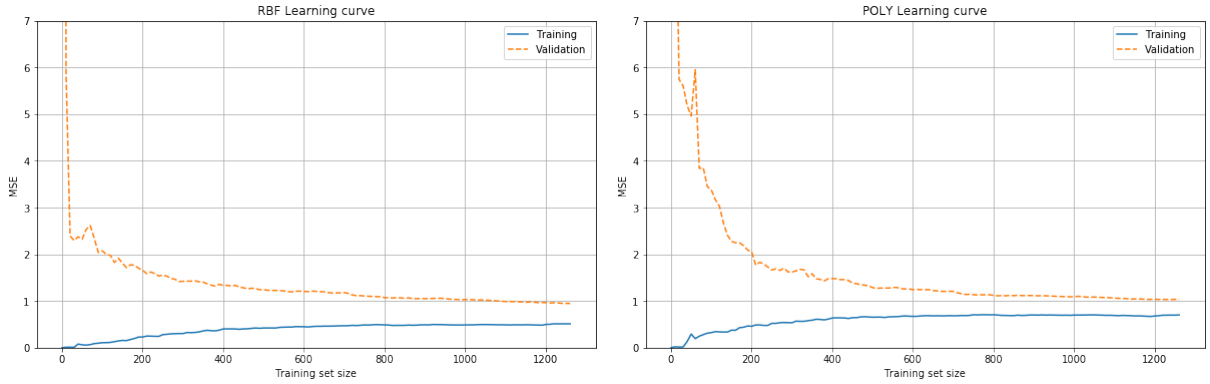


Figure 7: Heatmaps obtained by the final grid search over the parameters C, $Degree$ and $coef0$.

C	Degree	coef0	VL MSE	VL MEE
1.25	5	0.9	1.0925	1.0676
1.25	5	0.8	1.0959	1.0680
1	5	0.9	1.0348	1.0711
1	5	0.8	1.0386	1.0713
0.75	5	0.9	1.0420	1.0755

Table 7: Best models obtained for SVM + POLY after the final grid search.



(a) Learning curve of SVM with RBF kernel. (b) Learning curve of SVM with Polynomial kernel.

Figure 8: Learning curves of SVM with different kernels.

4 Model selection and final results

In order to perform the model selection and choose the final model for the ML CUP, each of the best models found by the grid searches has been retrained with 10-fold cross validation, and had its average MEE performances on validation recorded. The hardware configuration that we've used for training consisted of an i5 5200U 2.2GHz with 12GB RAM and a Nvidia GeForce 920M.
The MLP model with the smooth curve has been retrained over 2000 epochs, while the other MLP with the noisier learning curve has only been retrained over 200 epochs to avoid overfitting (Fig. 4).

According to Table 8, the best model for this dataset is Extreme Randomized Trees, which provided the best performances in terms of MEE, while also having the lowest training time. Therefore it's the model that we have chosen for the CUP blind test set and its parameters are shown in Table 9.

Model	MEE	Training Time
ERT	0.7697	1.1583s
RF	0.7797	1.3793s
SVM (RBF)	1.0056	2.6938s
SVM (Poly)	1.0676	1.3805s
MLP	1.0802	17.2249s
MLP (smooth)	1.2316	178.6565s

Table 8: Best performing models on validation.

Estimators	Min split	Min leaves	Max feat.	Max depth	Bootstrap	TR/VL/TS (MEE)
250	4	3	auto	52	False	0.2836 / 0.7697 / 0.7543

Table 9: ETR, the model selected for the CUP and its performances on Training, Validation and Test. The results on the blind Test are in file MoTeN_ML-CUP19-TS.csv

Since the target values of the regression represent (x,y) coordinates, we've decided to plot the results to get a better insight into the performances that each model offers. All the plots are shown in Appendix A, from which we can see that models with a lower MEE value are able to better predict a curve that is very similar to the real one, plotted in Fig.9.

The performance of our models on the test set are comparable with the ones they had on validation, as seen in Table 10, but it is however worth noticing that RF achieves a better MEE value on the Test Set.

Finally, it was interesting to observe that even simple and randomized approaches like RF and ETR can easily outperform complex MLP models, which are hard to work with due to the amount of time that they require to train and therefore to perform a grid search, which in our (small) case took more than 6 hours to complete. In order to decrease training time, we've analysed the dataset and we found out that there was a high correlation between couple of columns (our input features), which were actually copies of each other, as shown in Fig.13. For this reason we've tried to drop the duplicate columns so that we could obtain a smaller dataset on which we could perform the training, but there were no relevant improvements over the results in terms of neither training time nor MEE.

Model	MEE
RF	0.7378
ERT	0.7543
MLP	0.9726
SVM (RBF)	1.0444
SVM (Poly)	1.0866
MLP (smooth)	1.2449

Table 10: Best performing models on test set.

References

[1] Chih-Jen Lin Chih-Chung Chang. *LIBSVM: A Library for Support Vector Machines*. URL: `https://www.csie.ntu.edu.tw/~cjlin/papers/libsvm.pdf`.

[2] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python." In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[3] Louis Wehenke Pierre Geurts Damien Ernst. *Extremely Randomized Trees*. 2006. URL: `https://orbi.uliege.be/bitstream/2268/9357/1/geurts-mlj-advance.pdf`.

Appendices

A Output Values Plots

Figure 9: Test set real output shape.

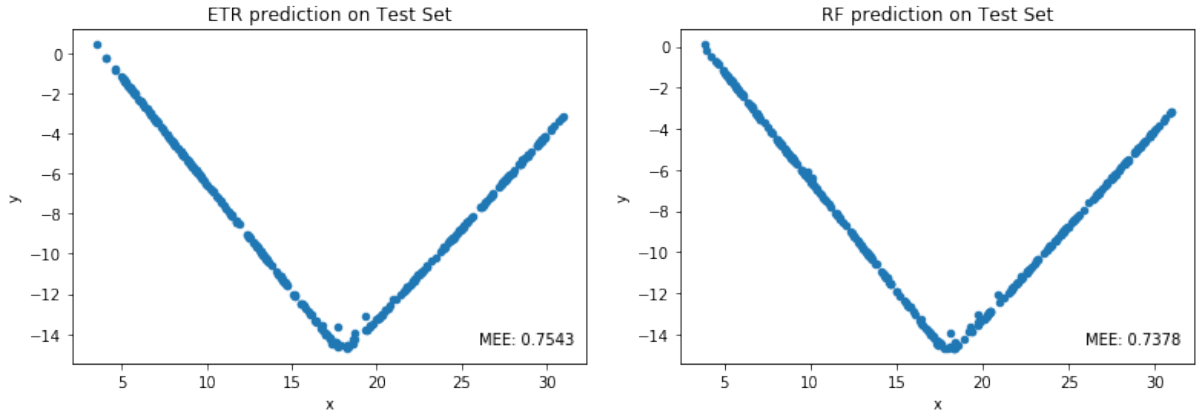


Figure 10: Random Forest and Extra Tree Regressor output shapes.

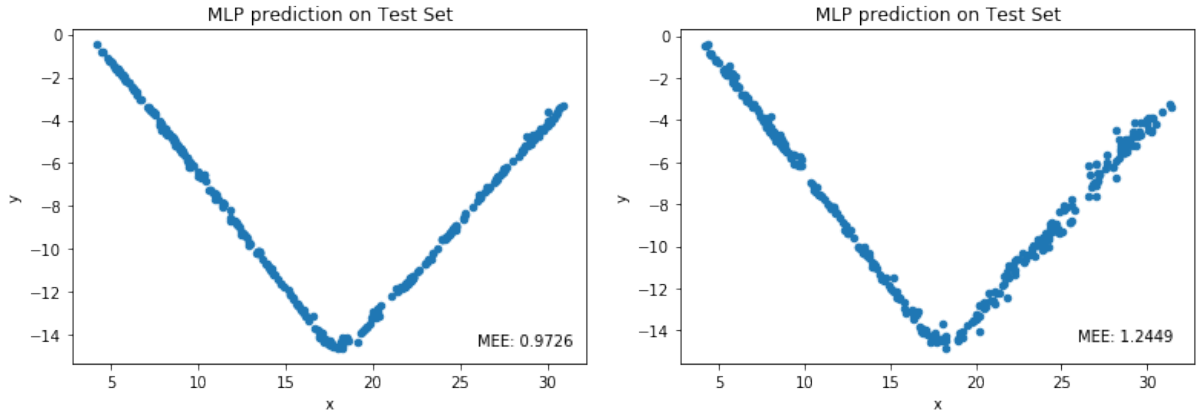


Figure 11: MLP output shape for the best model (left) and the one with the smooth curve (right).

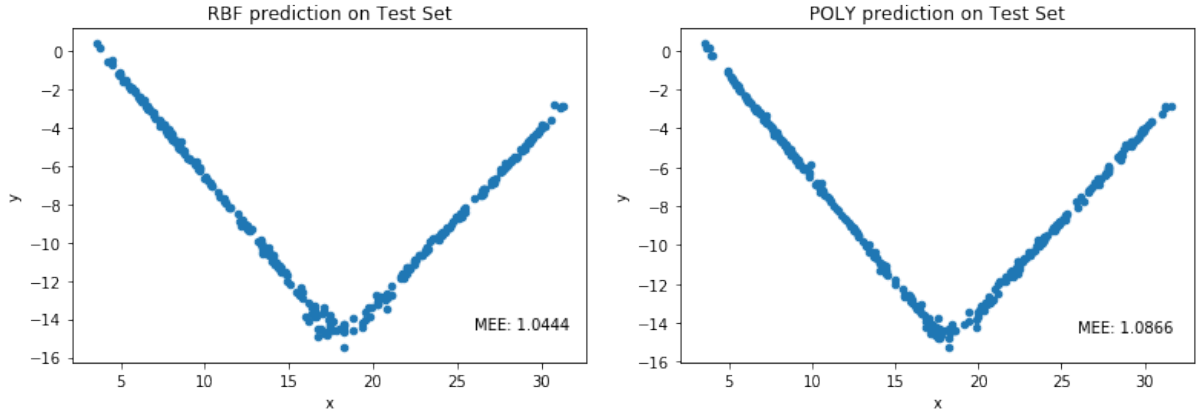


Figure 12: SVM output shape for the best models with different kernels.

B Correlations

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	1	-0.86	0.23	-0.82	-0.75	0.85	0.87	-0.82	1	-0.75	0.85	0.54	0.54	-0.82	0.89	-0.82	0.87	0.23	-0.86	0.89
2	-0.86	1	-0.13	0.86	0.79	-0.78	-0.9	0.89	-0.86	0.79	-0.78	-0.39	-0.39	0.86	-0.89	0.89	-0.9	-0.13	1	-0.89
3	0.23	-0.13	1	0.016	0.14	0.39	0.19	-0.17	0.23	0.14	0.39	0.58	0.58	0.016	0.19	-0.17	0.19	1	-0.13	0.19
4	-0.82	0.86	0.016	1	0.88	-0.7	-0.84	0.76	-0.82	0.88	-0.7	-0.28	-0.28	1	-0.84	0.76	-0.84	0.016	0.86	-0.84
5	-0.75	0.79	0.14	0.88	1	-0.62	-0.78	0.71	-0.75	1	-0.62	-0.17	-0.17	0.88	-0.79	0.71	-0.78	0.14	0.79	-0.79
6	0.85	-0.78	0.39	-0.7	-0.62	1	0.81	-0.77	0.85	-0.62	1	0.67	0.67	-0.7	0.83	-0.77	0.81	0.39	-0.78	0.83
7	0.87	-0.9	0.19	-0.84	-0.78	0.81	1	-0.87	0.87	-0.78	0.81	0.45	0.45	-0.84	0.91	-0.87	1	0.19	-0.9	0.91
8	-0.82	0.89	-0.17	0.76	0.71	-0.77	-0.87	1	-0.82	0.71	-0.77	-0.4	-0.4	0.76	-0.88	1	-0.87	-0.17	0.89	-0.88
9	1	-0.86	0.23	-0.82	-0.75	0.85	0.87	-0.82	1	-0.75	0.85	0.54	0.54	-0.82	0.89	-0.82	0.87	0.23	-0.86	0.89
10	-0.75	0.79	0.14	0.88	1	-0.62	-0.78	0.71	-0.75	1	-0.62	-0.17	-0.17	0.88	-0.79	0.71	-0.78	0.14	0.79	-0.79
11	0.85	-0.78	0.39	-0.7	-0.62	1	0.81	-0.77	0.85	-0.62	1	0.67	0.67	-0.7	0.83	-0.77	0.81	0.39	-0.78	0.83
12	0.54	-0.39	0.58	-0.28	-0.17	0.67	0.45	-0.4	0.54	-0.17	0.67	1	1	-0.28	0.48	-0.4	0.45	0.58	-0.39	0.48
13	0.54	-0.39	0.58	-0.28	-0.17	0.67	0.45	-0.4	0.54	-0.17	0.67	1	1	-0.28	0.48	-0.4	0.45	0.58	-0.39	0.48
14	-0.82	0.86	0.016	1	0.88	-0.7	-0.84	0.76	-0.82	0.88	-0.7	-0.28	-0.28	1	-0.84	0.76	-0.84	0.016	0.86	-0.84
15	0.89	-0.89	0.19	-0.84	-0.79	0.83	0.91	-0.88	0.89	-0.79	0.83	0.48	0.48	-0.84	1	-0.88	0.91	0.19	-0.89	1
16	-0.82	0.89	-0.17	0.76	0.71	-0.77	-0.87	1	-0.82	0.71	-0.77	-0.4	-0.4	0.76	-0.88	1	-0.87	-0.17	0.89	-0.88
17	0.87	-0.9	0.19	-0.84	-0.78	0.81	1	-0.87	0.87	-0.78	0.81	0.45	0.45	-0.84	0.91	-0.87	1	0.19	-0.9	0.91
18	0.23	-0.13	1	0.016	0.14	0.39	0.19	-0.17	0.23	0.14	0.39	0.58	0.58	0.016	0.19	-0.17	0.19	1	-0.13	0.19
19	-0.86	1	-0.13	0.86	0.79	-0.78	-0.9	0.89	-0.86	0.79	-0.78	-0.39	-0.39	0.86	-0.89	0.89	-0.9	-0.13	1	-0.89
20	0.89	-0.89	0.19	-0.84	-0.79	0.83	0.91	-0.88	0.89	-0.79	0.83	0.48	0.48	-0.84	1	-0.88	0.91	0.19	-0.89	1

Figure 13: Correlations between couples of input variables.