ML 2023 Project

Nunzio Canduci (n.canduci@studenti.unipi.it),
Paolo Junior Mollica (p.mollica@studenti.unipi.it),
Andrea Marino (a.marino47@studenti.unipi.it).

ShrimPropagation team

Master's degree in Computer Science (Artificial Intelligence curriculum),
Master's degree in Data Science and Business Informatics

Date: 01/02/2023

Type of project: **B**

Objectives

Our aim for this project was to test various models, built with the help of some libraries (see bibliography), on the Monk and ML23 CUP tasks.

In particular, we were interested in seeing how various combinations of both standard and non-standard techniques would perform in the tasks at hand (especially in the ML23 CUP).

Ultimately, we aimed at achieving the best possible result for the blind competition, while also learning more about the impact that our choices have on such results. Therefore, an exploration of the performance of various models and hyperparameters' configurations was performed.

We implemented three classes of models: Support Vector Machine, Neural Network and Random Forest.

Code structure, models overview

Basic structure of the code:

- GitHub repo in which we stored the datasets, images, notebooks and saved models.
- A notebook for each of the three classes of models, for each of the tasks.
- In each notebook: import and process the data, create the model (often by calling a function provided by the library), set up the grid search and training process, select best hyperparameters and plot the learning curve, assess the error, save the model.
- Extra notebooks for data exploration and models comparison.
- Extra notebooks for utils (NeuralNetwork classes, save/load models and plots…).

We used the **Random Forest** and **SVM** from Scikit-Learn, and implemented a multilayer feedforward **Neural Network** using Keras. The `NeuralNetwork` and `MonkNeuralNetwork` classes in our code are designed to be seamless integrated with Scikit-Learn extending the native classes of the library and implementing 3 main methods (`fit`, `predict`, `score`).

For the Neural Networks, we have 2 models, given by the choice of the strategy for the exploration of the hyperparameters space (full grid search/bayesian optimizer). So we compared 4 models in total.

For all these models we tried out many configurations of hyperparameters, as detailed in the next sections.

Models details & contributions – 1

MONK tasks models: Scikit-Learn's `SVC`, Scikit-Learn's `RandomForestClassifier`, a custom class implementing a `Sequential` Neural Network with Keras, named `MonkNeuralNetwork`.

All of the models used for the ML23 CUP task are wrapped in a `Pipeline` with a `RobustScaler`.

- `RobustScaler` is chosen for its robustness to outliers, through the following formula: $(X - median(X)) \,/\, IQR$, where IQR is the interquartile range (75th - 25th percentile).
- A `Pipeline` groups together many steps of a ML process into a single estimator, allowing the scaler to be applied for each subset of data in the cross-validation process.

Structure of the Neural Networks:

- `Sequential` model from Keras.
- `Dense` (hidden) layers are interleaved by `Dropout` layers. The dropout rate (for input and hidden layer) is a hyperparameter selected by hyperparameters tuning.
- As a further form of regularization, the weights of the `Dense` layers are constrained through Keras' `MaxNorm()`.

Models details & contributions – 2

Weight initialization for the ML23 CUP Neural Networks: `HeNormal()` from `keras.initializers` [2][7]. This strategy is tailored for layers that use the `ReLU` activation function, like the ones in our neural network.

Stop conditions for the ML23 CUP Neural Networks: we decided to keep the number of epoch fixed during training in the hyperparameters tuning process, but incrementing the epochs and adding the early stopping callback in the retraining of the best model. Precisely, early stopping has been implemented by using `EarlyStopping()` from `keras.callbacks`.

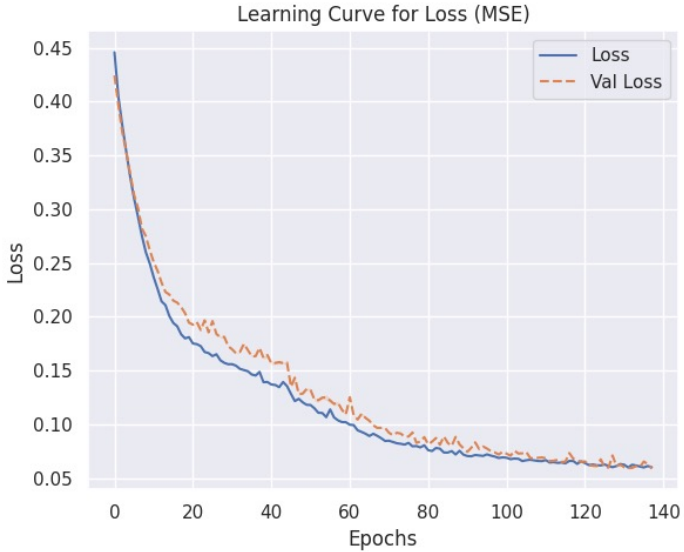
Preliminary trials and constraints in terms of time and computational resources leds us to make choices in model's features. In particular, for ML23 CUP Neural Networks:

- N° layers: we opted for a 2 layers architecture to make the grid search feasible, while for the bayesian optimizer the number of layers is selected in a range from 2 to 5.
- Activation function: we only used ReLU as is mostly used in state-of-the-art NNs.
- Learning algorithm: we choose the SGD optimization, but varying the batch size.

MONK 1 Results

Table 1: Neural Network grid search hyperparameters and results

Task	*Architecture / learning alg / act. f / η / α / λ / epochs / batch_size / patience*	MSE (TR/VL)	Accuracy (TR/VL)	Accuracy (TS)
Monk 1	*(8,8) / SGD / ReLU / 0.1 / 0.6 / 0.01 / 200 / 8 / 6*	*0.060 / 0.059*	*100% / 100%*	*100%*



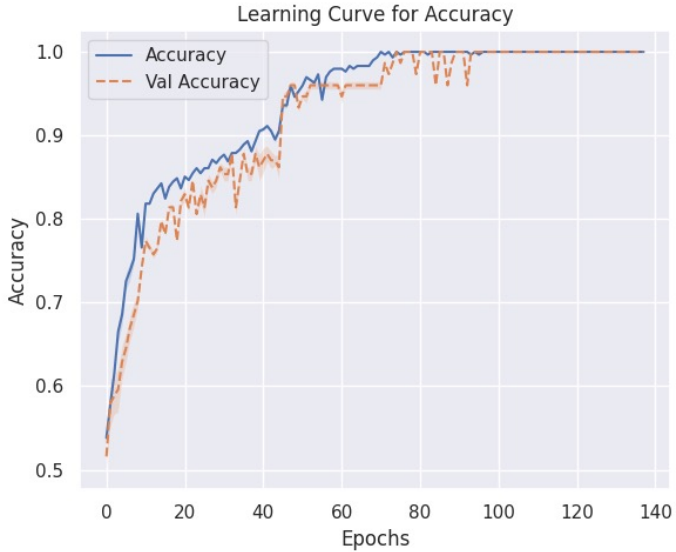
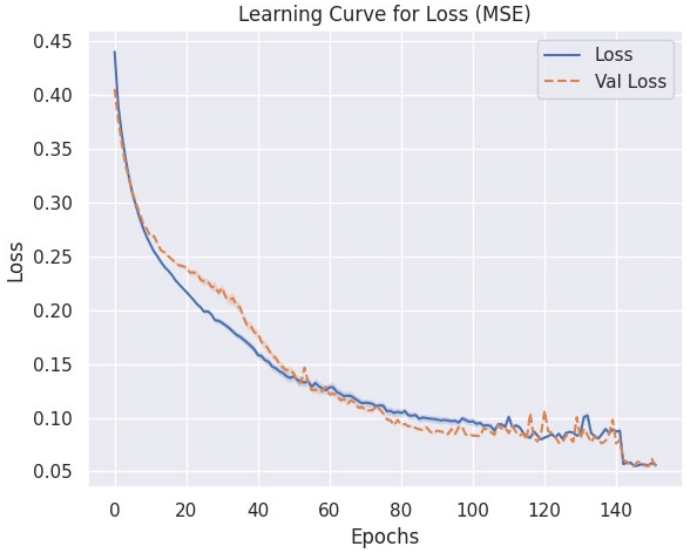


Figure 1:
Average MSE and accuracy plots for the model in Table 1, MONK 1 task, using a cross-validation with 5 folds

MONK 2 Results

Table 2: Neural Network grid search hyperparameters and results

Task	***Architecture / learning alg / act. f / η / α / λ / epochs / batch_size / patience***	**MSE (TR/VL)**	**Accuracy (TR/VL)**	**Accuracy (TS)**
Monk 2	*(8,8) / SGD / ReLU / 0.1 / 0.6 / 0.01 / 200 / 8 / 6*	*0.056 / 0.054*	*100% / 100%*	*100%*



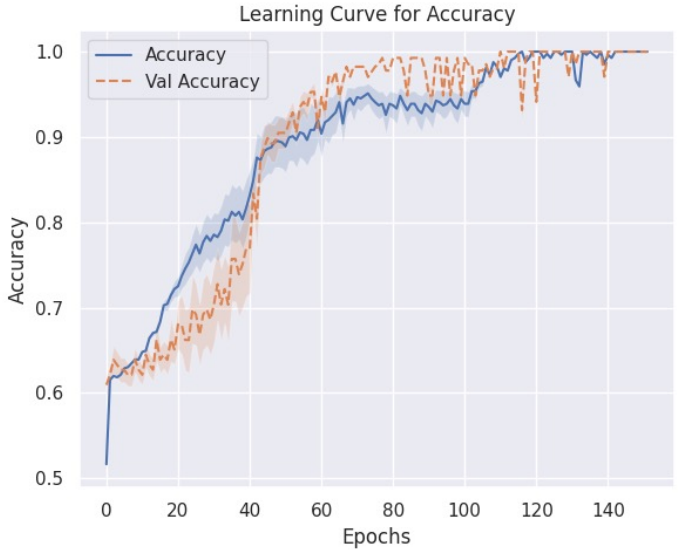
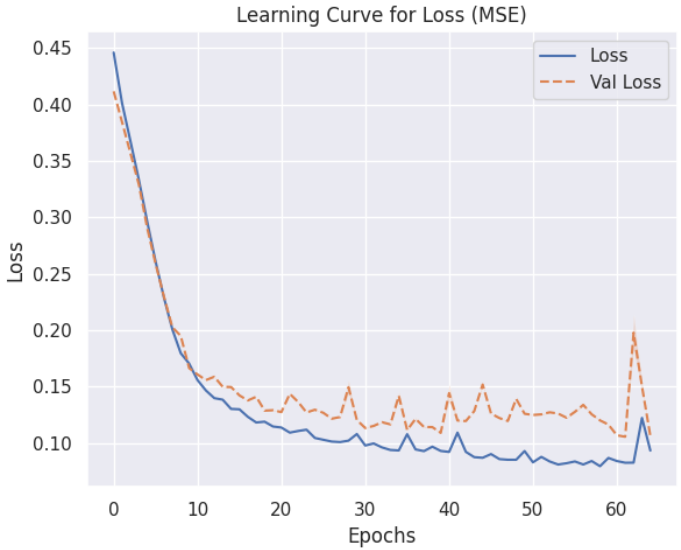


Figure 2:
Average MSE and accuracy plots for the model in Table 2, MONK 2 task, using a cross-validation with 5 folds

MONK 3 Results

Table 3: Neural Network grid search hyperparameters and results

Task	*Architecture / learning alg / act. f / η / α / λ / epochs / batch_size / patience*	MSE (DS)	Accuracy (TR/VL)	Accuracy (TS)
Monk 3	*(8,8) / SGD / ReLU / 0.1 / 0.6 / 0.01 / 200 / 8 / 6*	*0.093 / 0.106*	*93,32% / 91,83%*	*97,22%*



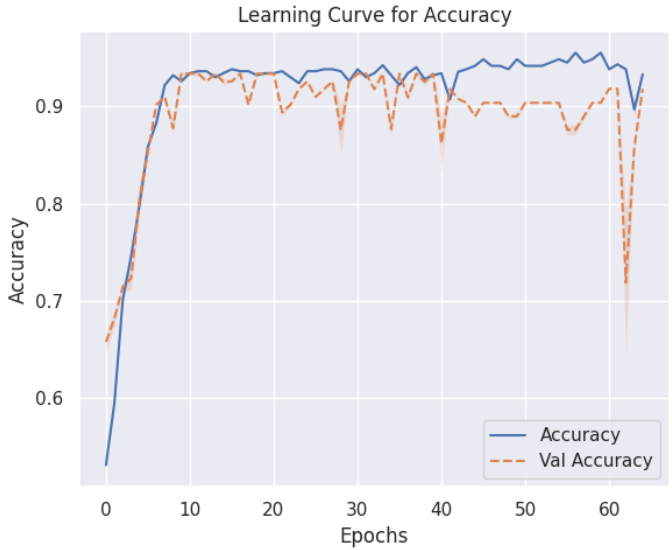


Figure 3:
Average MSE and accuracy plots for the model in Table 3, MONK 3 task, using a cross-validation with 5 folds

CUP Dataset exploration

The test set and the training set show similar features distribution, as shown in Figure 4(b).
Furthermore, there aren't cluster of test data points that are separated from the training ones, as shown in Figure 4(a). For these reasons, the test set seems to be well represented by the training set.

Figure 4(a) shows the effect of applying the UMAP dimensionality reduction technique to the ML23 training and blind test sets. We choose this because it aims to preserves data neighborhood.

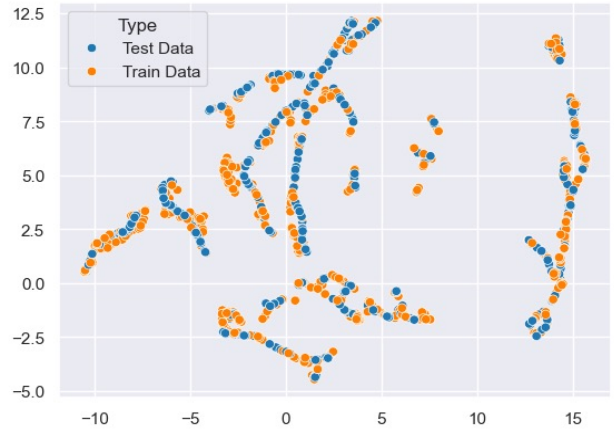


Figure 4: **(a)** UMAP embedding for the whole dataset, **(b)** Features distribution for training and test data

Hardware resources & Computing times

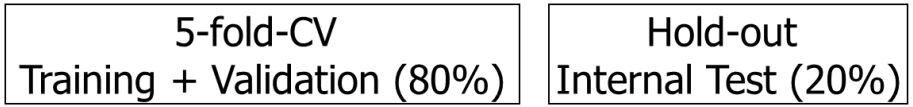
All the models were fitted using Google Colaboratory using the standard CPU, which has 8 threads (physical and logical)

Table 4: Hardware resources and computing times for the compared models

Model	HW resources	Grid search time	(Re)training time
SVR	Google Colaboratory's CPU	2h40m *(with an average of 3.99s/fit and a total of 2400 fits)*	12m10s
RF	Google Colaboratory's CPU	*2h26m (with an average of 0.60s/fit and a total of 1800 fits)*	13.6s
NN, Grid Search	Google Colaboratory's CPU	*8h06m (with an average of 5.10s/fit and a total of 5760 fits)*	14s *(with an average of 8,6epoch/s)*
NN, Optuna	Google Colaboratory's CPU	4h40m (for 400 trials)	9s *(with an average of 8,6epoch/s)*

CUP Validation schema

We held out 20% of the data (randomly sampled by fixing the seed for reproducibility) to use it as an internal test set. Since the seed is fixed, all of the models use the same internal test set.
On the remaining part of the data the model is trained and validated following a 5-fold Cross Validation schema.



We ran a full grid search for all the three classes of considered models.
In addition to that, we used Optuna (an hyperparameter optimization framework. See [6] and Appendix) for the exploration of the hyperparameters' space. The range of hyperparameters explored by Optuna is in the following slides.

Range of hyperparameters in the grid search: for most of the hyperparameters, we followed the advice in [8] and considered an uniform sampling in the log-domain. That is, the ratio between two consecutive values to explore is constant.

CUP Validation schema

The best model returned by the hyperparameters tuning process is then retrained using a 5-fold cross-validation, obtaining a model for each fold and thus an ensemble. Using this approach, we were able to use all the training data, given the considerations made earlier.

Then, we measure the test error (of the ensemble) on the hold-out TS. After this model assessment phase, the ensemble is re-built through a cross-validation on the whole dataset.

Random Forest

Table 5: Range of explored hyperparameters in the grid search, Random Forest

Hyperparameter	Values	Hyperparameter	Values
N° of estimators (aka trees)	100, 150, 200, **300**	*Maximum depth of the tree*	**None**, 4, 6, 8, 10
Minimum n° of samples to split an internal node	**2**, 8, 10	*Minimum n° of samples required to be a leaf node*	**1**, 3, 4

Hyperparameter	Values
N° of features to consider when looking for the best split	**`sqrt()`**, `log_2()`

The best hyperparameters found in the grid search are in bold. Table 6 shows the error of this model.

Table 6: MEE of the best Random Forest using grid search on the datasets

Dataset	TR	VL	TS
MEE	1.033	2.734	2.489
stdev	0.005	0.094	n.d.

Support Vector Regressors

Table 7: Range of explored hyperparameters in the grid search, SVM

Hyperparameter	Values	Hyperparameter	Values
Kernel	RBF, **poly.**, sigmoid (aka two-layer perceptron), linear	β_1 *coefficient* (for two-layer perceptron kernel)	0, 1, 2
C regularization hyperparameter	0.1, 1, 10, **100**	*Translation term k0* (for polynomial kernel: $(\gamma \cdot \langle x, x_i \rangle + k_0)^p$)	0, 1, **2**
Epsilon-tube hyperparameter ε	0.01, **0.1**, 1	*Degree p* (for polynomial kernel)	2, 3, **4**

Hyperparameter	Values
Kernel's parameter γ ($1/(2\sigma^2)$ for RBF, $\gamma \cdot \langle x, x \rangle$ for poly kernel, β_0 for sigmoid)	**0.1**, 0.01, 0.001

The best hyperparameters found in the grid search are in bold. Table 8 shows the error of this model.

Table 8: MEE of the best SVM

Dataset	TR	VL	TS
MEE	0.472	0.804	0.813
stdev	0.006	0.031	n.d.

Neural Network (Grid Search)

Table 9: Range of explored hyperparameters in the grid search, Neural Network

Hyperparameter	Values	Hyperparameter	Values
Learning rate η	10^{-3}, **10^{-2}**	*L2 regularization hyperparameter λ*	10^{-3}, **10^{-2}**
Nesterov's momentum	True, **False**	*Activation function* (*hidden layers*)	**ReLU**
Momentum hyperp. α	**0.5**, 1	*Dropout hyperp.* (*input layer*)	**0**, 0.1
Learning algorithm	**SGD**	*epochs*	50, 100, **150**
Batch size	**16**, 32		

Hyperparameter	Values
Architecture ((x_1,…,x_n): *n hidden layers, layer i has x_i units*)	(128,64), (128,128), **(256,128)**
Dropout hyperparameter (*hidden layers,* (x_1,…,x_n): *layer i has dropout hyperparameter i*)	**(0.1,0.1)**, (0.2,0.2)

Best hyperparameters are in bold.

Neural Network (Bayesian Optimizer)

Table 10: Range of hyperparameters explored by Optuna

Hyperparameter	Range	Hyperparameter	Range
Learning rate η	[10^{-4}, 10^{-2}] **0.002**	*L2 regularization hyperp. λ*	[10^{-5}, 10^{-2}] **0.001**
Nesterov's momentum	[**True**, False]	*Activation function* (*hidden layers*)	ReLU
Momentum hyperp. α	[0, 1] **0.095**	*Dropout hyperp.* (*input*)	[0, 0.5] **0.022**
N° (hidden) layers	[1,5] **2**	*Dropout hyperp. (hidden)*	[0, 0.5] **(0.091, 0.267)**
Optimization algorithm	SGD	*epochs*	[5, 400]
Batch size	[16, 128] **40**	*Patience hyperparameter*	[5, 10] **5**
Hyperparameter		**Range**	
N° units per layer		[32, `max_units`] with a step of 32 **(480,480)**	

Using `max_units=512.` Moreover, Optuna's objective function is designed to ensure a funnel structure to the network (number of units in a layer less than or equal to those in the previous layer).

Neural Networks learning curves, Bayesian optimizer

Figure 5: Learning curve (MSE, left) and MEE (right) for the NN ensemble with hyperparameters selected using Optuna framework

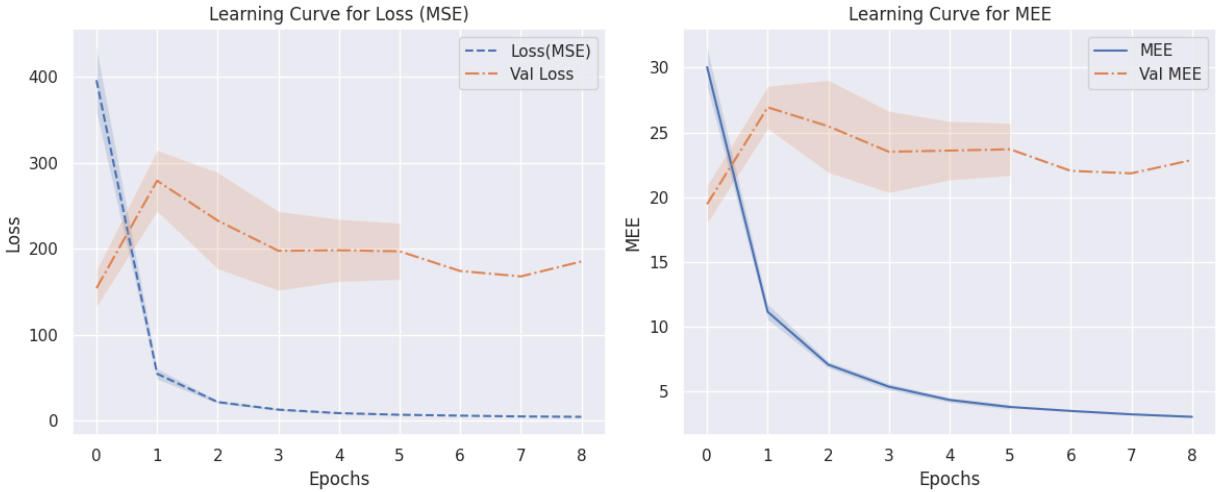


Table 11: Error of best NN using Bayesian Optimizations

Dataset	MSE	MEE
TR	4.369	3.037
VL	185.2	22.87
TS	2.200	2.162

The observed variance in the graph resulting from using five different splits for TR and VL reinforces our decision to employ an ensemble approach.

CUP Results: final model

We chose the ensemble of SVRs as our final model, because it is the one that achieves the best result on the internal test set by far. The chosen hyperparameters were shown in the previous slides.

Figure 6: Learning curve for the SVR model (MEE)

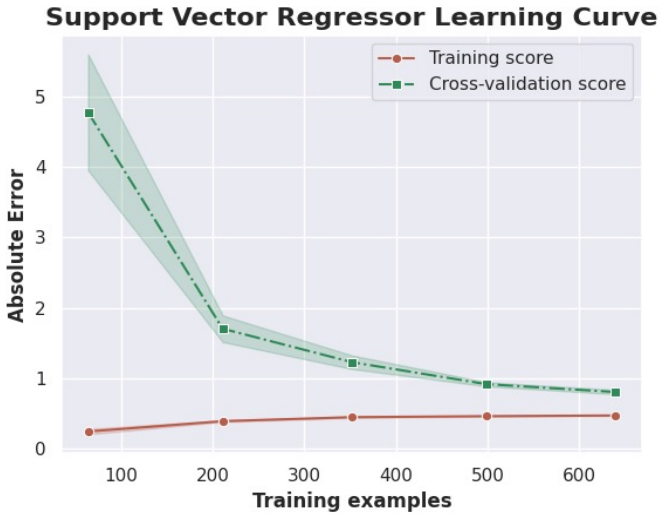
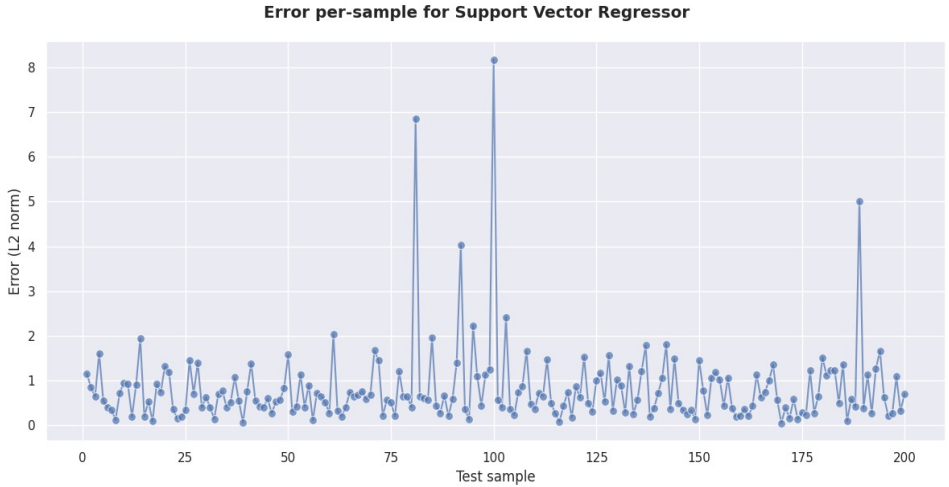


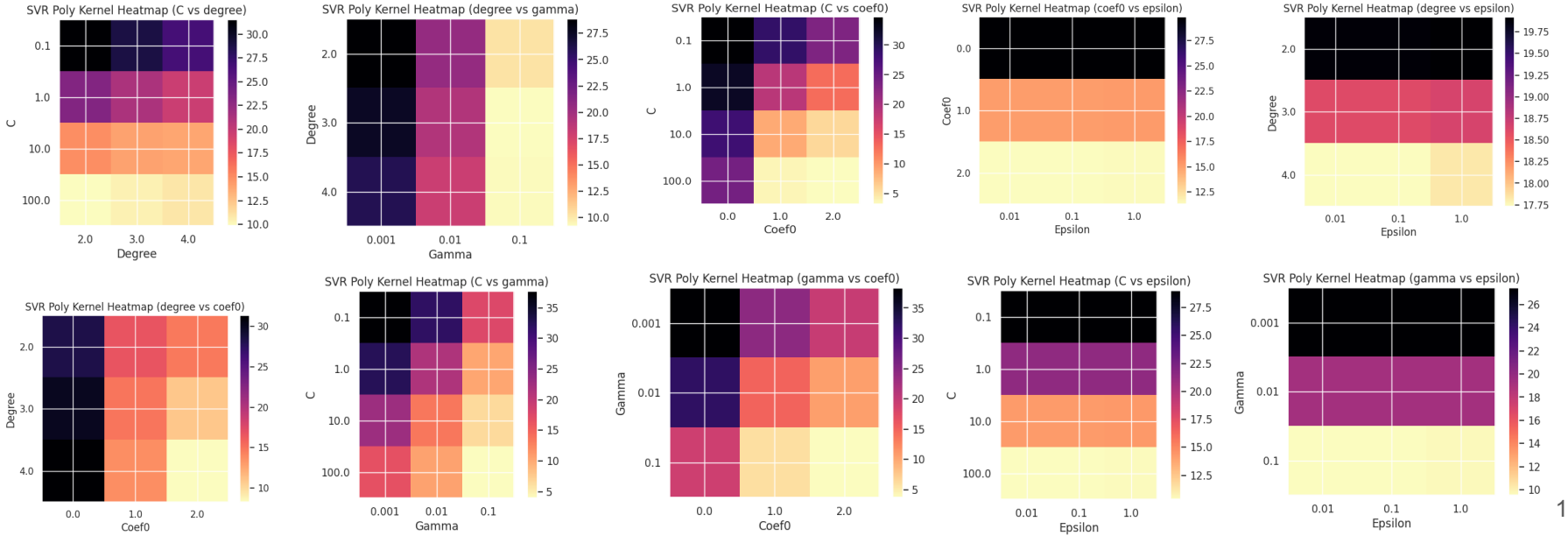
Figure 7: Euclidean distance between the true and predicted value, for each point in the internal TS



CUP Results: hyperparameters' impact on the error – 1

The following images present the mean (validation) MEE achieved across all combinations of hyperparameters' values, fixed the polynomial kernel. Specifically, for each pair of hyperparameters, the average MEE is calculated over the 5 folds of the cross-validation. It can be noted, for example, that a high value of the C parameter implies better results.

Figure 8: Heatmaps showing the impact on the MEE of various combinations of hyperparameters, SVR



CUP Results: hyperparameters' impact on the error – 2

From the analysis of the previous images (and the underlying numbers) the following conclusion can be drawn:

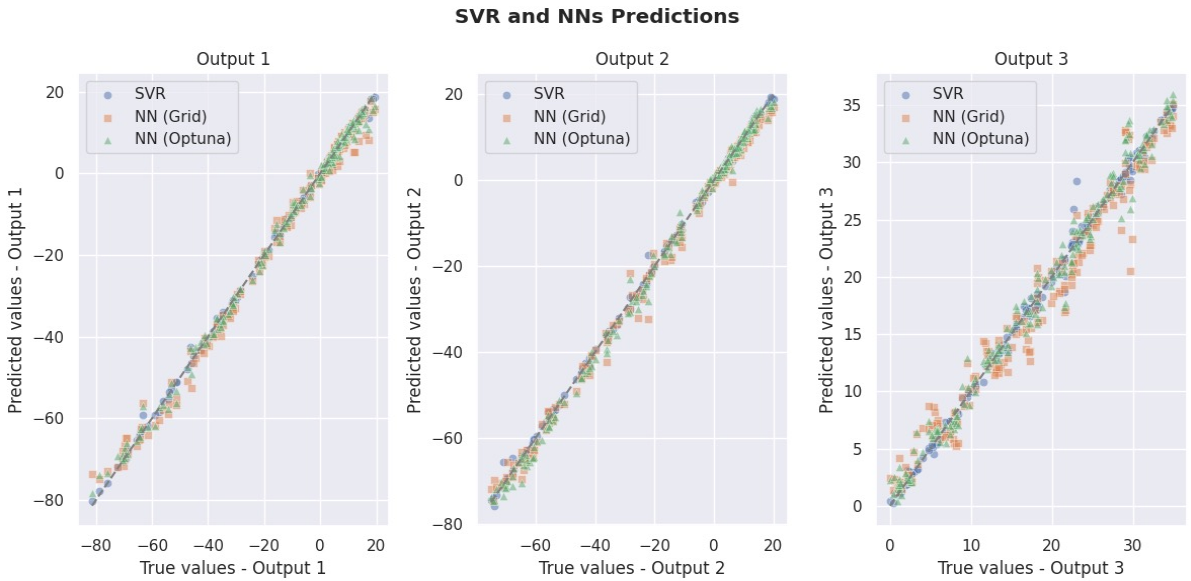
- The value of ε is nearly irrelevant, as in all heatmaps in which ε appears the color is constant along the axis corresponding to the hyperparameter.
- C=100 consistently yields the lowest error. This is not very surprising, as high C corresponds to less regularization and to a tighter margin (i.e., less tolerance on error).
- The translation term k_0 consistently yields the best results at value 2.
- When the degree is 4, by varying k_0 the error changes tremendously.
- The C vs γ heatmap suggests a linear dependency, i.e. that the average MEE decrease when C and γ increase.
- γ has a big impact on the error, at a glance the heatmaps that involve it are the most colourful. Furthermore, high γ corresponds to better values.

Discussion: comparisons between models – 1

Comparison between the predictions of the three best models on the hold-out test set. The comparisons are done component-wise, each shown in a different subplot.

The comparison shows how the SVR's predictions are consistently aligned to the diagonal (perfect prediction), with a lower dispersion than the other two.

Figure 9: True and predicted values of the three best models



On the third coordinates the NNs models struggle more, and a few points show substantial discrepancies between predicted and true values. Noticeably, the third coordinate is the one in which the range of values is smaller.
More such comparisons can be found in our `exploration` notebook.

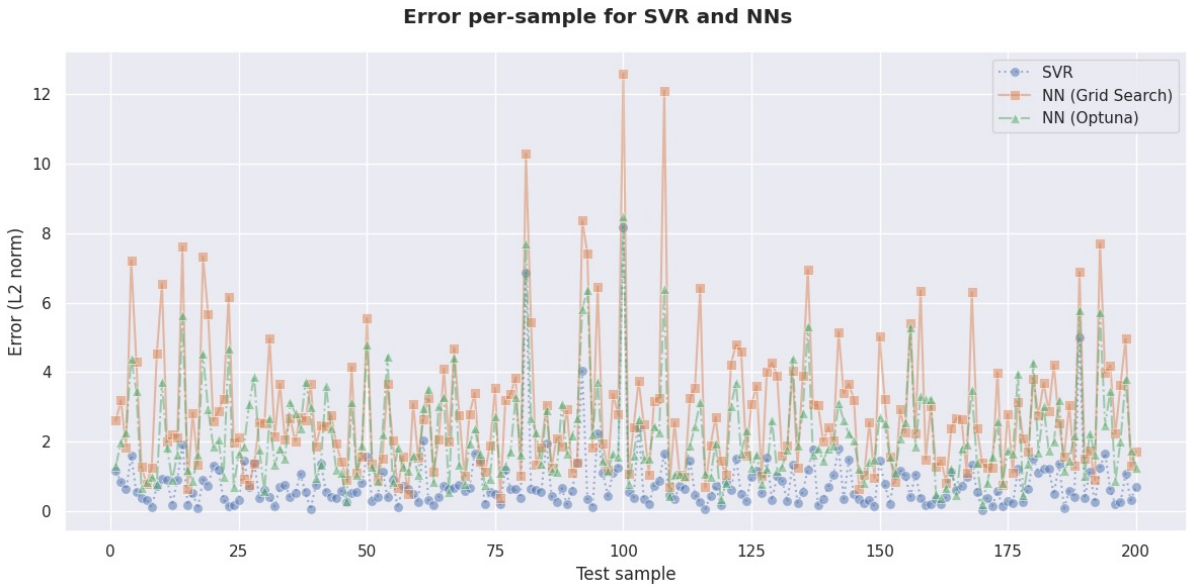
Discussion: comparisons between models – 2

Error of the three best model's predictions on each sample of the internal TS. The error is measured in the L_2 norm:

$$\|y_i - \hat{y}_i\|_2, \; i = 1, \ldots, |TS|.$$

The SVR consistently makes less mistakes, and the neural network constructed with Optuna yields lower prediction errors than those made by the one built with grid search, on all samples.

Figure 10: Error per-sample for the the three best models



The three models struggle to predict the same samples: the curves show similar peaks, but with different values. The SVR and Optuna NN show narrower error spreads than the grid search NN.

Discussion: comparisons between models – 3

Figure 11: Different views of the SVR predictions and true outputs in $\mathbb{R}^3$

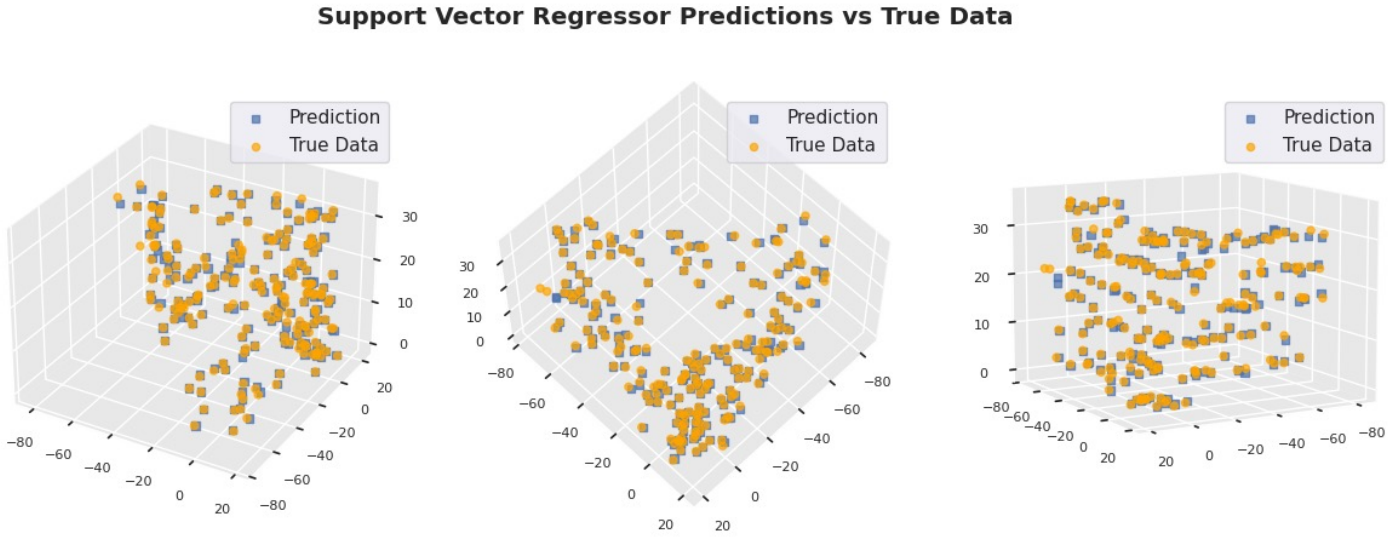


Figure 11 shows the spatial collocations of true outputs and SVR's predictions, for each sample in the hold-out test set. This figure shows how the data density is not homogeneous in the output space.

Final discussion

- Support Vector Machine (SVM), Neural Network (NN) and Random Forest (RF) models were examined.
- The SVM showed excellent accuracy, with the best model using a polynomial kernel, and the best trade-off between convergence time and results, partly due to the reduced hyperparameter space compared to the Neural Networks.
- The Neural Networks exhibited good performance, especially using Bayesian optimization, which allowed a more efficient hyperparameter configuration than grid search. RF provided satisfactory results, although not to the level of SVM or NN. In general, the careful choice of hyperparameters had a significant impact on model performance.
- Early Stopping proved to be an excellent choice in both theoretical and practical terms, as it allowed us to considerably reduce the training time in refitting, as well as apply good regularization to the model.
- The use of ensemble proved to be an effective choice, both experimentally and theoretically, to reduce variance. This approach helped to decrease the error in the various models and allowed full utilization of the dataset, given its small size as well.
- Finally, the adoption of a `RobustScaler` allowed us to benefit in scale-sensitive algorithms, such as Neural Networks, by obtaining zero mean and unit variance and thus allowing faster convergence, and in distance-based algorithms, such as SVM.

Acknowledgments

Team name: **shrimPropagation**.

The predictions on the blind test set made by the SVR ensemble (i.e., our best model) are stored in the file `shrimPropagation_ML-CUP23-TS.csv`.

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

Bibliography – 1

Main libraries and software tools used for this project, with documentation:

1. F. Chollet, *Keras*, GitHub, GitHub Repository, (2015) version 3.0.2
2. Keras 3 documentation
3. M. Abadi, A. Agarwal et al., *TensorFlow: Large-scale machine learning on heterogeneous systems* (2015). www.tensorflow.org
4. F. Pedergosa, G. Varoquaux et al, *Scikit-learn: Machine Learning in Python*, JMLR, 12(Oct), (2011), pp.2825-2830
5. Scikit-learn API reference
6. T. Akiba, S. Sano, T. Yanase, T. Ohta, M. Koyama, *Optuna: A Next-generation Hyperparameter Optimization Framework*, KDD, (2019), pp.2623-2631

Bibliographical sources:

7. K. He, X. Zhang, S. Ren, J. Sun, *Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification*, *2015 IEEE International Conference on Computer Vision (ICCV)*, Santiago, Chile, 2015, pp. 1026-1034,
8. Bengio, Y. *Practical Recommendations for Gradient-Based Training of Deep Architectures*. In: Montavon, G., Orr, G.B., Müller, KR. (eds) Neural Networks: Tricks of the Trade. Lecture Notes in Computer Science, vol 7700 (2012). Springer, Berlin, Heidelberg.
9. S. Thrun. J. Bala, E. Bloedorn, I. Bratko et al. (1992). *The MONK's Problems: A Performance Comparison of Different Learning Algorithms*.

Appendices

Appendix – 1

Full grid search for the MONK tasks: SVM

Table 12: Hyperparameter's values for the SVMs used for the three MONK tasks

Hyperparameter	Values	Hyperparameter	Values
Kernel	RBF, polynomial, sigmoid (aka two-layer perceptron)	β_1 *coefficient* (for two-layer perceptron kernel)	0, 1, 2
C regularization hyperparameter	0.1, 1, 10, 100	*Translation term k0* (for poly. kernel: $(\gamma \cdot \langle x, x \rangle + k_0)^p$)	0, 1, 2
ε (regression only)	0.01, 0.1, 1	*Degree p* (for polynomial kernel)	2, 3, 4

Hyperparameter	Values
Kernel's parameter γ ($1/(2\sigma^2)$ for RBF, $\gamma \cdot \langle x, x \rangle$ for poly kernel, β_0 for sigmoid)	0.1, 0.01, 0.001

Appendix – 2

Full grid search for the MONK tasks: Random Forests

Table 13: Hyperparameter's values for the Random Forests used for the three MONK tasks

Hyperparameter	Values	Hyperparameter	Values
N° of estimators (*aka trees*)	20, 50, 100, 200	*Maximum depth of the tree*	None, 10, 20 (*not for MONK 3*), 30
Minimum n° of samples to split an internal node	2, 4, 6	*Minimum n° of samples required to be a leaf node*	1, 2, 4
Bootstrap	True, False	*Criterion to evaluate split quality*	`gini`, `entropy`

Hyperparameter	Values
Number of features to consider when looking for the best split	`Sqrt()`, `log_2()`, None

Appendix – 3

Full grid search for the MONK tasks: Neural Networks

Table 14: Hyperparameter's values for the Neural Networks used for the three MONK tasks

Hyperparameter	**Values**
Architecture ((x_1,…,x_n): *n hidden layers, layer i has* x_i *units*)	(8), (8,8), (8,8,8), (8,8,8,8)

Hyperparameter	**Values**	**Hyperparameter**	**Values**
Learning rate η	0.1, 0.3, 0.5	*Learning algorithm*	SGD
L2 regularization hyperparameter λ	0.01	*Activation function* (*hidden layers*)	ReLU
Momentum hyperparameter α	0.6, 0.8, 0.9	*epochs*	200
Batch size	8		

Appendix – 4

Further comparisons on MONK tasks: grid search results for RF and SVM

Table 15: Prediction results obtained for the MONK's task, with a SVM

Task	***Kernel* / *C* / *γ* / *p* / *k0***	**Accuracy (TR/VL)**	**Accuracy (TS)**
MONK 1	`Poly` / *1* / *1* / *2* / *0.5*	*100%±0% / 80,43%±17,95%*	*100%*
MONK 2	`Poly` / *10* / *1* / *2* / *0.0*	*100%±0% / 71,71%±10,20%*	*100%*
MONK 3	`RBF` / *1* / *0.1* / *n.d.* / *n.d.*	*93,40%±1,05% / 93,40%±4,23%*	*97,22%*

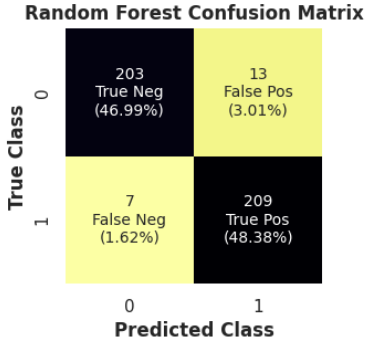
Table 16: Average prediction results obtained for the MONK's task, with a Random Forest

Task	***# trees* / *maxDepth* / *min_samples_split* / *min_samples_leaf* / *maxFeatures* / *bootstrap* / *criterion***	**Accuracy (TR/VL)**	**Accuracy (TS)**
MONK 1	*20* / *None* / *4* / *1* / *None* / *True* / `gini`	*100%±0% / 78,03%±15,44%*	*95,37%*
MONK 2	*100* / *None* / *2* / *1* / *None* / *False* / `gini`	*100%±0% / 65,17%±9,91%*	*81,94%*
MONK 3	*50* / *None* / *6* / *1* / `sqrt()` /*True* / `gini`	*95,88%±1,13% / 92,57%±4,07%*	*96,99%*

Appendix – 5

In-depth analysis MONK task

Figure 12: Confusion matrix and ROC curve for the Random Forest, MONK 1 task



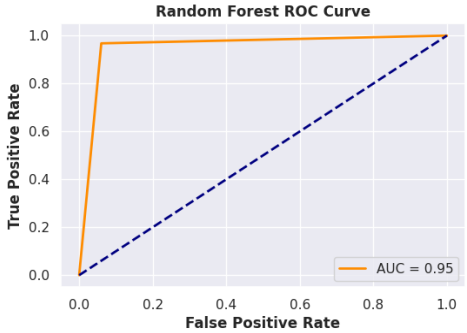
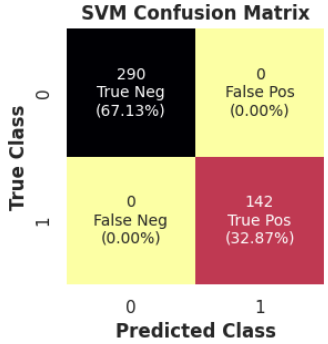


Figure 13: Confusion matrix and ROC curve for the SVM, MONK 2 task



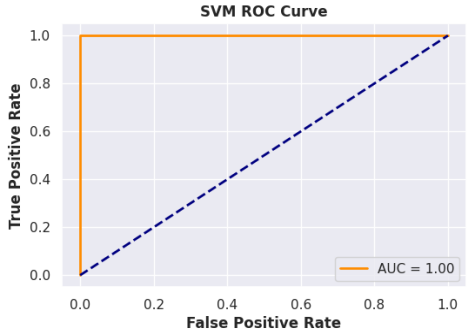
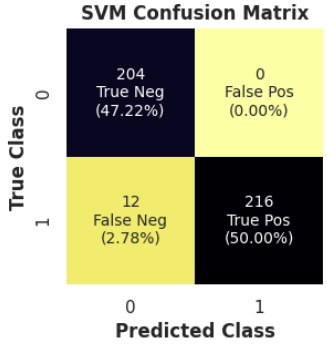
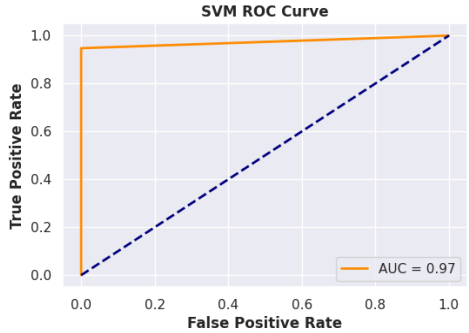


Figure 14: Confusion matrix and ROC curve for the SVM, MONK 3 task





Appendix – 6

Further specifications on model details & contributions

Weight initialization:
In the weight initialization performed by `HeNormal()` starting weights' values are sampled from a $N(0, \sigma^2)$ distribution, with $\sigma = \sqrt{2/\text{fan\_in}}$, where fan_in is the number of incoming connections for each weight.
The main idea is to preserve the variance of the response (aka the output) of the layers, and this leads to the weights having the afromentioned law.
This initialization has been proven superior to `GlorotNormal()` and `GlorotUniform()` (aka Xavier) for deep networks with ReLU activation function. [7]

Weight constraints:
Keras' `MaxNorm(n)` constraints the norm of the weights in the following way: for each hidden unit, the vector of the weights that are incident in that unit are constrained to have maximum norm n (i.e. $\left\|(w_{i,j})_j\right\| \leq n \; \forall i$).
We chose the value n=3 because it yielded better result in explorative trials. Due to time constraints, we opted not to do a grid search on this value.

Early stopping:
When doing model selection, in both the full grid search and Optuna, the number of epochs is an hyperparameter. We opted to not do early stopping at this stage, following the advice in [8]: early stopping at this stage implies that the hyperparameters' values are tested for different number of epochs. As a consequence, the effect that these values have is hidden. This makes the grid search less reliable and hampers the analysis of the effect of individual hyperparameters.

Appendix – 7

Further specifications on model detils & contributions

Optuna:

Optuna is a hyperparameter optimization software using a define-by-run approach, allowing dynamic construction of the search space. It optimizes an objective function, defined by input hyperparameters, which is built gradually during runtime. Trials, representing objective function evaluations, generate hyperparameters dynamically using suggest methods with specified ranges.

These hyperparameters are statistically sampled based on past trials' history. It also incorporates pruning mechanisms to terminate low-promise trials and employs efficient strategies for searching and performance estimation. Additionally, it can identify informative objective function evaluations regarding hyperparameters' concurrence relations.

The choice was driven in particular because optuna adopts the Tree-structured Parzen Estimator (TPE) algorithm for its Bayesian optimisation method.

TPE is generally more computationally efficient than methods based on Gaussian processes, especially when dealing with large hyper-parameter spaces.

Furthermore, the adoption of TPE, like other Bayesian processes, allows us to improve the speed of the search, at the same time achieving an excellent trade-off between exploration and exploitation in the hyperparameter space.

Appendix – 8

Further details on data exploration

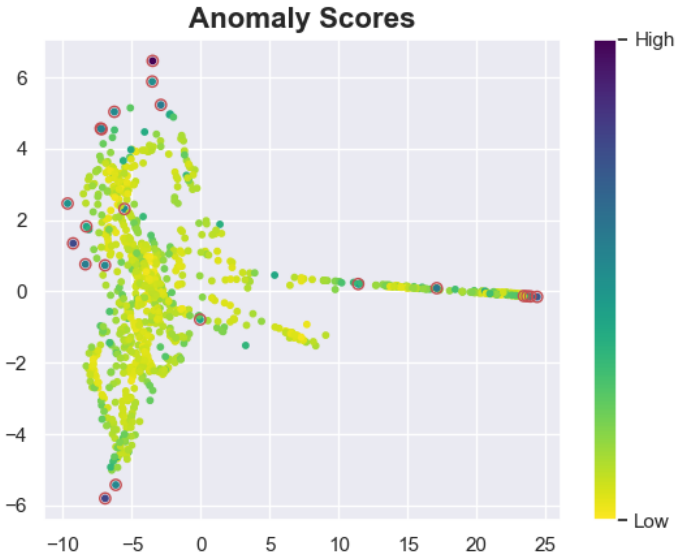
For the outlier detection we applied the LOF algorithm to the original 10-dimensional training data. This algorithm assigns a score to each datapoint, based on the ratio between its local density and the local density of its neighbours.

Figure 15 shows the effect of the IsoMap embedding of the 10-dimensional training data in the plane. Points are colored based on their anomaly score.

The algorithm detected 21 outliers, based on a threshold of 1.3. We concluded that there aren't many outliers in the dataset.

The presence of outliers (although very few), motivated us in using the `RobustScaler`.

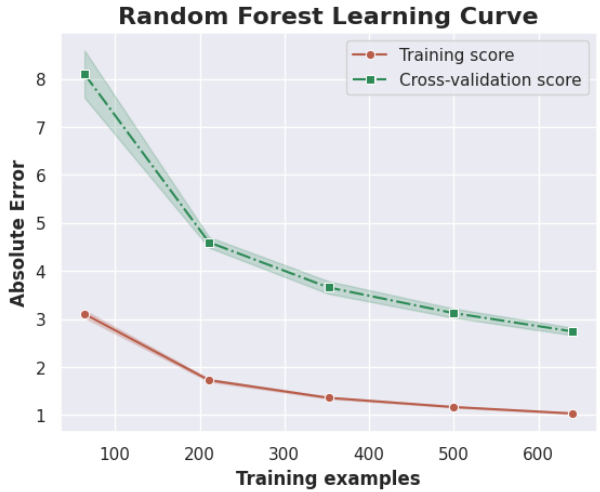
Figure 15: IsoMap embedding and LOF score of the training data

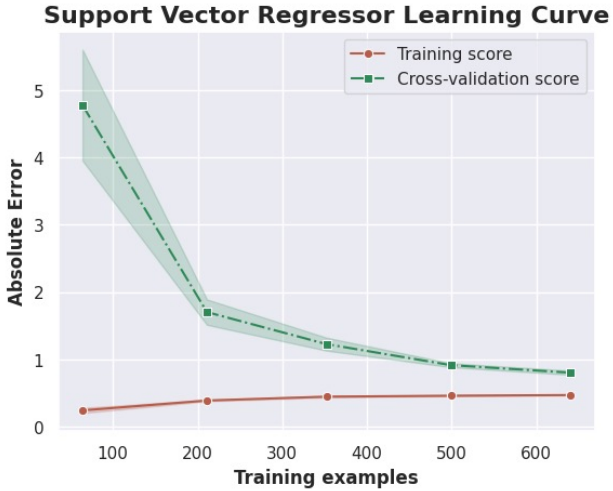


Appendix – 9

Learning curves for SVR and Random Forest

Figure 16: Learning curves displaying the MEE for the SVM (right) and RF (left) with the best hyperparameters found in their respective grid searches.





The learning curves are obtained through Scikit-Learn's `learning_curve` function. The function performs 5-fold CV on increasingly portions of the design set, to determine cross-validated training and validation scores. The curve tells how well the model scales as the amount of data increases. With this approach, we have the mean and the standard deviation over the folds, for each point (corresponding to the loss of the model on amount of data on which it is trained).

Appendix – 10

Heatmap for the Neural Network (bayesian optimization)

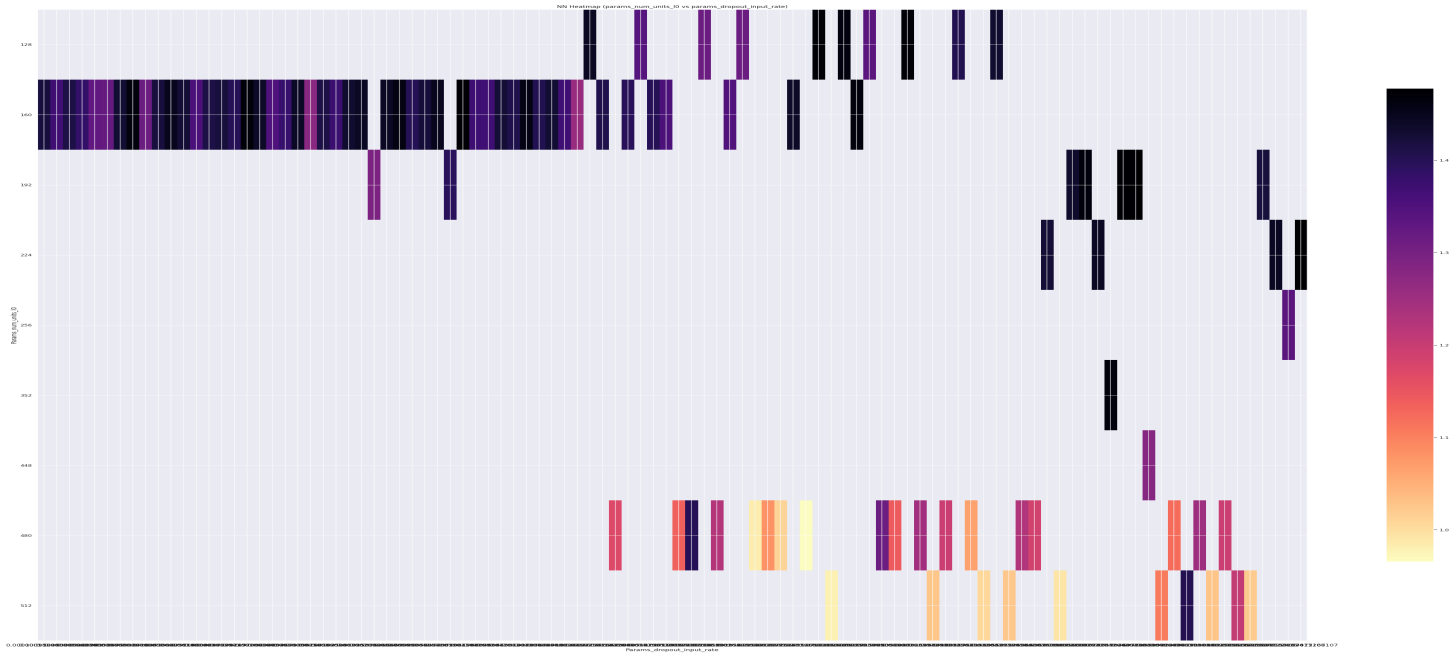


Figure 17: Heatmap showing the impact in terms of (VL) MEE of the Optuna choosen values for number of units and dropout rate for the first hidden layer

Appendix – 11

Further comparisons among models: spatial collocation of predictions & true values

Figure 18(a): True and predicted values on the internal test set, Optuna Neural Network

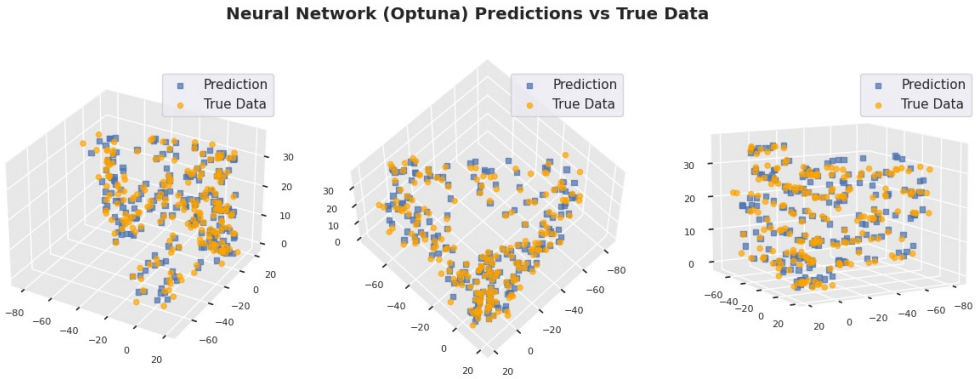


Figure 18(b): Error per test sample, SVR and Random Forest

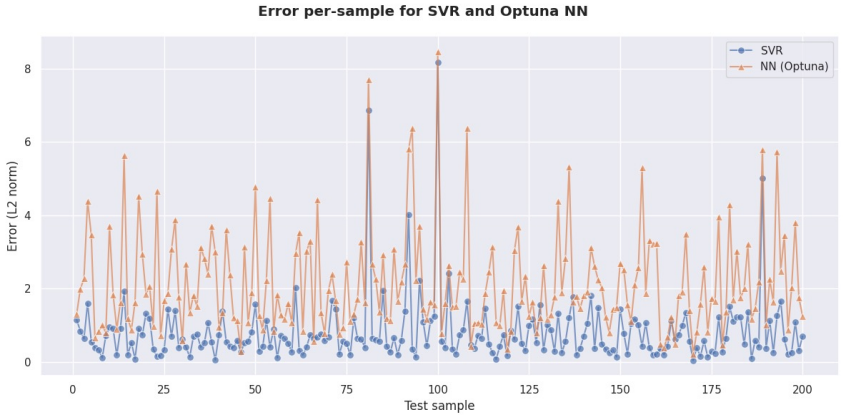


Figure 18 **(a)** illustrates that the Neural Network predictions are less aligned to true values than those of the Support Vector Regression (SVR), **(b)** Reveals superior performance of the Support Vector Machine (SVM) over the NN in the entire test set.