A comparison among regression models

Authors: (Dawit Mezemir Anelay / Marco Petix)
Master degree in Computer Science (AI Curriculum)
Mail: (d.anelay1@studenti.unipi.it / m.petix@studenti.unipi.it)
ML course (654AA), Academic Year: 2020/21
Date: *25/01/2021*
Type of project: **B**
Github repo: (https://github.com/dawitanelay/ML-Project-20)

ABSTRACT

Our goal was to build and compare different models from different software tools. We compare a Neural Network from Keras[1] and both a Support Vector Machine and a K-nn regressors from the Scikit-learn[2] framework. We perform model selection and validate via a combination of screening phases and cross-validation, also testing on an internal test set extracted via hold-out.

1. INTRODUCTION

Our main aim is to study the influence of the various hyper-parameters upon the implementation of a neural network model. This is achieved through intensive use of screening phases which also provide the added benefit of alleviating the computational burden of the many grid-searches.

The result of said effort, in the form of a neural network implementing a mini-batch Stochastic Gradient Descent, is then put to the test through a comparison with two main other well known regression models. The former, a Regressor-chain SVR, presents a Radial Basis Function kernel (rbf) with a degree of polynomial kernel function and regularization parameters while the latter, a K-Neighbors Regressor, weights the influence of each neighbor with respect to their euclidean distance from the record fed to it.

Assuming the potential benefits of applying feature suppression on the training dataset, in terms of both improved processing time and accuracy, a performance comparison was implemented on the neural network models trained on the original dataset or one with no highly correlated characteristics. The object of this record corresponds to the model trained on the complete dataset mentioned so far but, in the appendix, there is information on the comparison between the two and on the surprisingly worse performance of the second.

This appendix also includes information regarding the performance on the dataset of other machine learning regression models initially considered for the aforementioned comparison, such as Extra Tree and Decision Trees Regressors.

2. METHOD

The neural network is implemented through the Keras framework. The Pandas[3] library is used to manage the import / output of data while the Numpy[4] library handles numerical manipulations and, finally, the Matplotlib[5] and Seaborn[6] libraries provide the means for the visualization graphs and learning curves. In addition to the implementation of the various regression models other than the

aforementioned neural network, the Sklearn library is used for all its numerous methods dedicated to model validation and selection.

The Mean Euclidean Error, implemented by ourselves, is used as a metric of evaluation of the model's performances on both the validation and internal test set. The neural network presents two dense hidden layers both made up by 17 units and activated, respectively, by the sigmoid and tanh function. The dense output layer counts two units, obviously activated by the linear function, referring to the two target variables involved in the regression task.

The Stochastic Gradient Descent is implemented as the learning algorithm due to the familiarity with it developed by the authors during the course. The choice of implementing a mini-batch version of the same is dictated by a comparison with the alternatives during the validation process. The initialization of the weights is taken from a truncated normal distribution with zero mean and standard deviation deriving from the fan-in of the units.
Finally, the regularization of the weights is imposed by introducing a lambda decay coefficient. The stop condition corresponds to reaching the hundredth epoch.

The model selection process is firstly implemented through a considerable use of screening phases dedicated to each of the hyperparameters. These tend to guide the definition of the ranges of values then explored through the application of the grid-search.

The training set is initially split into developer sets and internal test sets with a ratio of 75% and 25%. The screening phases see the use of one third of the remaining training set as a validation set, thus obtaining a division of 50% TR, 25% VL and 25% TS. The subsequent grid-search instead bases its validation on a k-fold validation process centered on 3 folds.

Excluding the tests related to the application of the feature elimination process, which resulted in the comparison between the models cited in the introduction and described in the appendix, the development of the neural network model did not include any particular preprocessing steps.

Both the Regressor Chain, from the family of the Support Vector Regressors, and the K-Neighbors Regressor, are implemented through the Sklearn library. These models were implemented together with other well-known regression models from the same library. After the definition of a grid-search for each of these and the comparison of the best model defined for each of these types of estimators, the aforementioned Support Vector and the K-Neighbors Regressors were chosen as having the best performances.

3. EXPERIMENTS

3.1 MONK'S RESULTS

Task	#Units, eta, lambda, ..	MSE (TR/TS)	Accuracy (TR/TS) (%)
MONK 1	5 units, eta = 0.45, momentum= 0.1	0.0002/0.0006	100%/100%
MONK 2	2 layers of 3 and 4 units, eta = 0.1, momentum = 0.1	0.0003/0.0005	100%/100%
MONK3	2 layers of 2 units each, eta = 0.6, momentum = 0.3	0.0367/0.0043	96.1%/100%
MONK3 (reg.)	2 layers of 3 units each, eta = 0.4, momentum = 0.1, ???	0.0766/ 0.0631	94.13%/97.22%

Table 3.1. Average prediction results obtained for the MONK's tasks.

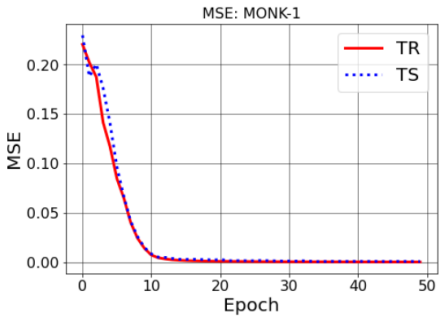
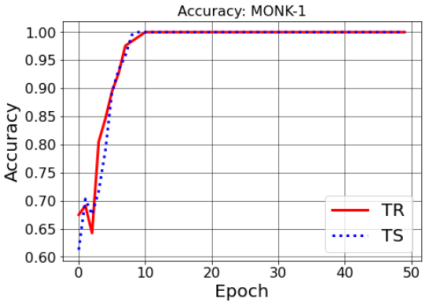
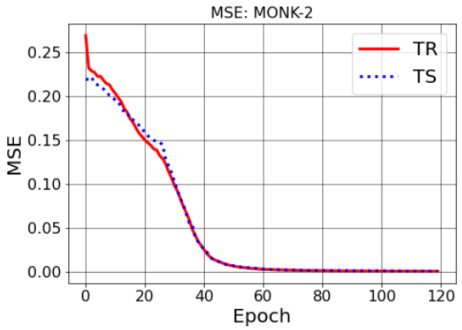
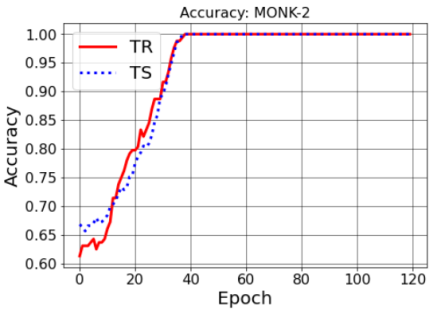
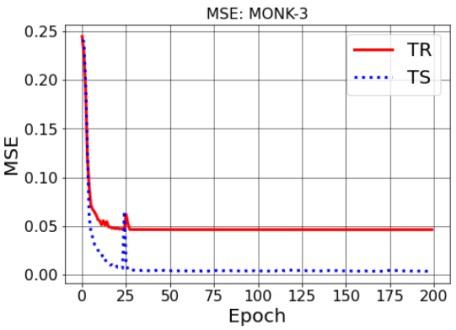
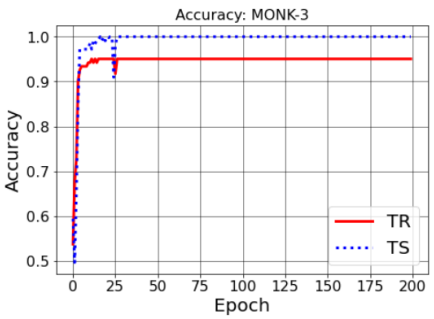
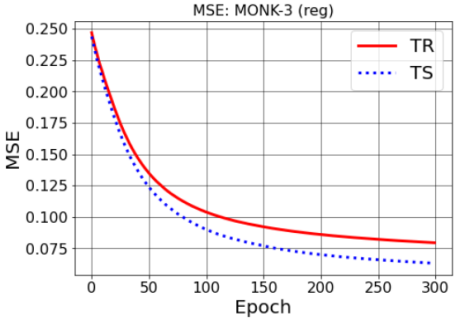
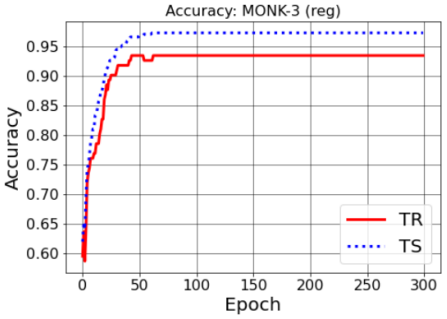
MSE	Accuracy
MONK 1	
	
MONK 2	
	
MONK 3	
	
MONK 3 (with regularization)	
	

Figure 3.1. Plot of the MSE and accuracy for the 3 MONK's benchmarks.

3.2 CUP RESULTS

3.2.1 Neural Network model from Keras

Validation: Splitting the training-set

The original Training set is initially divided, as usual in hold-out, into the Developer set and (Internal) Test set with a ratio of 75% and 25%.

The aforementioned Developer set is then:

- divided, as in hold out again, into the Training set and the Validation set with a ratio of 66% and 33% (50% and 25% of the original Training set) during the Screening Phase.
- involved in a K-fold Validation with K = 3 during the Grid-Search.

Navigating the hyper-parameters space: Screening Phase and Grid-Search

The Screening Phase is a tool extensively used during the first steps of the development of the model in question in order to analyze the behavior of the individual hyper-parameters and reduce the ranges of values to be analyzed through Grid-Search.
The aforementioned analysis was carried out by printing and comparing the learning curves of models characterized by the modification of a particular parameter or a pair of them. These analyzes by pairs of parameters have proved useful in analyzing the interaction between related parameters such as the learning rate and the momentum.

The preliminary trials pursued during this step refers to various comparisons among model differing by:

- the weights initialization mode
- the number of units and activation function of the first layer
- the learning rate and momentum
- the number of units and the activation function of the second layer
- the learning rate and decay
- the epochs and batch size

The set of values associated with each of the aforementioned parameters during this phase is reported within the second column of **Table 3.2**.

The second step of the Model Selection and Assessment process saw the implementation of a Grid-Search upon intervals of values “inspired” by the results obtained during the previous step. These values are displayed within the third column of **Table 3.2**.

Parameters	Screening phase	Grid parameters
Eta	0.01, 0.1, 0.3, 0.5, 0.7	0.4, 0.5, 0.6
Num. of units of the first hidden layer	5, 10, 15, 20	13, 15, 17
Activation function of the units in the first layer	tanh, sigmoid, softmax, relu	sigmoid
Num. of units of the second hidden layer	5, 10, 15, 20	13, 15, 17

Activation function of the units in the second layer	tanh, sigmoid, softmax, relu	tanh
Activation function of the units in the output layer	---	linear
Momentum	0.0, 0.01, 0.1, 0.3, 0.5, 0.7	0.3, 0.4, 0.5
Lambda	0.0, 0.01, 0.1, 0.3	0.01, 0.1
Num. of epochs	50, 100, 200, 300	50, 100, 200
Size of the batch	None, 10, 50, 100	10, 15, 20

Table 3.2. Values of the parameters explored via the screening phase and grid search of the

Results from the Grid-Search

As mentioned within the previous subsection, the Grid-Search was included a K-fold validation with k = 3 for the evaluation of the 1458 potential models.

Said candidates for the optimal set of parameters were then sorted with respect to their score as Mean Euclidean Error on the Cross-Validation. The MME scores for the 5 most promising candidates with respect to the training, validation and (internal) test set are displayed in **Table 3.3**.

Rank	TR: MEE	VL: MEE	TS: MEE
5	2.492712	3.003236	2.999846
4	2.388998	3.003206	2.896262
3	2.320047	2.995402	3.076121
2	2.383637	2.994617	2.928943
1	2.387664	2.973705	2.847089

Table 3.3. MEE on the TR, VL and TS of the most promising results from the Grid-search of the NN

3.2.2 Support Vector Regressor from Sklearn

After the comparison among the models from Sklearn mentioned above the Chain Regressor was among the two with the best performances.

Validation and Grid-search

The training dataset was initially split into developer sets and internal test sets with a ratio of 75% and 25%. In order to obtain the optimal parameters, the model was involved in an extensive grid search. This included a k-fold validation process centered on 3 folds. The intervals of values explored by the grid-search are displayed in **Table 3.4**.

Hyper Parameters	Range of values for the Grid-Search
kernel	rbf, poly, sigmoid, precomputed

epsilon	0.0001, 0.001, 0.01, 0.1
Gamma	numpy.logspace(-3, 2, 6)
C	numpy.logspace(-3, 2, 6)

Table 3.4. Range of hyper-parameters explored via grid-search for the Chain Regressor

Results from the Grid-Search

As for the model previously analysed, the candidates for the optimal set of parameters resulting from the grid-search on the SVR were then sorted with respect to their score as Mean Euclidean Error on the Cross-Validation. The MME scores for the 4 most promising candidates with respect to the training, validation and (internal) test set are displayed in **Table 3.5** together with the set of parameters which they're associated with.

Rank	TR: MEE	VL: MEE	TS: MEE	Kernel	Gamma	Epsilon	C
4	1.637049	3.403059	3.206134	rbf	0.1	0.001	100
3	1.641912	3.401634	3.201590	rbf	0.1	0.01	100
2	2.520116	3.397486	3.065184	rbf	0.1	0.1	10
1	1.696402	3.391483	3.165636	rbf	0.1	0.1	100

Table 3.5. Best four hyperparameter combination obtained for the Chain Regressor

3.2.3 K-Neighbors Regressor from Sklearn

As for the Chain Regressor, after the comparison among the models from Sklearn, the K-Neighbors Regressor turned out as one of the two with the best performances.

Validation and Grid-search

The process of model selection for the model in question was implemented through a grid-search whose parameter space is displayed in **Table 3.6.** Each potential candidate as optimal set of values was evaluated, as for the other models, via K-fold Validation with k = 3.

Hyper Parameters	Range of values for the Grid-Search
algorithm	auto, ball_tree, kd_tree, brute
leaf_size	from 1 to 30
n_neighbors	from 1 to 50
weights	uniform,distance

Table 3.6. Range of hyper-parameters explored via grid-search for the K-Neighbors Regressor

Results from the Grid-Search

Once again we provide a list of the most promising candidates for the optimal set of parameters resulting from the grid-search on the model. They are, as usual, sorted with respect to their score as Mean Euclidean Error on the Cross-Validation. As for the SVR, the MME scores for the 4 most promising candidates with respect to the training, validation and (internal) test set are displayed in **Table 3.7** together with the set of parameters which they're associated with.

Rank	TR: MEE	VL: MEE	TS: MEE	Neighbors	Weights	Algorithm	p	Leaf size
1	---	2.968047	2.664215	11	distance	ball_tree	2	10
3	---	2.964121	2.649795	12	distance	ball_tree	2	10
2	---	2.996065	2.649795	15	distance	auto	2	10
1	---	2.727770	2.66509	10	distance	auto	2	1

Table 3.7. Best four hyperparameter combination obtained for the K-Neighbors Regressor

3.2.4 Comparing the Neural Network, Chain Regressor and K-Neighbors Regressor

A comparison with respect to the computing training time

Table 3.8 provides the time spent in training the various models with respect to different hardwares. Obviously, being K-Neighbors Regressor an instance-based estimator, the time associated with its training results null.

Model	Training time	Hardware
Neural Network	11.73 s	Dell XPS15, Intel® Core™ i7 – 8750H, NVIDIA® GeForce® GTX 1050 Ti, 16 GB RAM , Windows 10
K-Neighbors Regressor	---	---
SVR: Chain Regressor	0.122 s	Accer swift 3 Amd - Ryzen 5, GB RAM,linux mint 20

Table 3.8. The training time of each model and the hardware that performed the computation

A comparison with respect to the Mean Euclidean Error (MEE)

Table 3.9 provides a comparison between the MEE values for the three models with respect to their interaction with the training, validation and (internal) test sets.

The Neural Network model, while still obtaining relatively good results with both training and test set, reveals a state of overfitting that would require a more careful employment of regularization.

Given the figurative place on the regression performance podium occupied in this case by the K-Neighbors Regressor, the model in question is selected as the final model. Therefore, the predictions returned by this towards the two targets of the CUP test set are used by the group in order to participate in the competition.

Model	TR: MEE	VL: MEE	TS: MEE
Neural Network	2.387664	2.973705	2.847089
K-Neighbors Regressor	---	2.727770	2.66509
SVR: Chain Regressor	1.696402	3.391483	3.165636

Table 3.9. MEE on the TR, VL and (internal) TS of the three models

3.2.5 The final model: K-Neighbors Regressor

As mentioned within the previous subsection, the instance-based K-Neighbors is selected as the final model for the participation at the CUP competition. This is due to its surprising ability to beat the performances of both other models.

The ease of implementation and optimization, represented by the absence of training and relatively short calculation times of the grid-search, especially when compared with the multiple hours of calculation for models such as the neural network, also acts as an added value.

In the following we present the parameters used within the implementation of the model and a summary of its performance in the form of **Table 3.10**.

Parameters of the models:

- **Num. of neighbors**: 10
- **Algorithm**: auto
- **Weights**: distance (Influence on the regression based upon the distance from the input)
- **p** : 2 (**Euclidean Distance**)
- **Leaf size**: 1

Model	TR: MEE	VL: MEE	TS: MEE
K-Neighbors Regressor	---	2.727770	2.66509

Table 3.10. Summary on the performance of the final model on the VL and TS sets

Learning curve of the final model

Figure 3.2 shows the learning curve of the model in question. As mentioned above, this particular instance-based estimator class stores the elements of the training set. The curve associated with the performance on the training set, now representing a full accuracy score, is therefore to be considered both expected and not very useful for the purpose of examining the actual performance of the model. These however, are still largely represented by the curve associated with the results of the interaction with the internal test set.

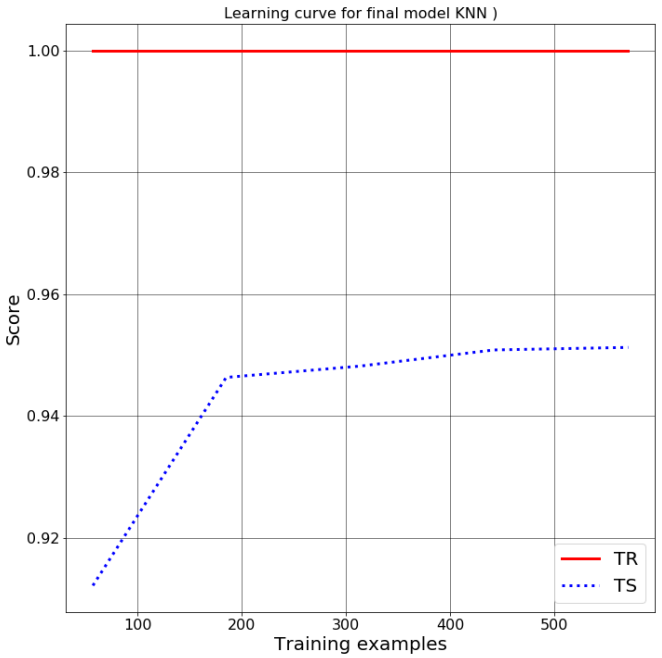


Figure 3.2. Learning Curve of the final model: K-Neighbors Regressor

4. CONCLUSIONS

During the development of the project we became more competent in the practice of model selection and assessment. The analysis of the learning curves of models such as the neural network, as happened during the implementation of the various screening phases, has allowed us to acquire greater familiarity, stemming from a practical finding, of the influence that the individual parameters such as decay and momentum can have on a model's behavior.

An additional lesson that we can say to have learned from the experience with the project is the sometimes underestimated value of simple but nevertheless effective models like that of the K-Neighbors.

Nickname: smile
Blind Test Results: smile_ML-CUP20-TS.csv

We agree to the disclosure and publication of my name, and of the results with preliminary and final ranking .

REFERENCES

[1]: https://keras.io/

[2]: https://scikit-learn.org/stable/

[3]: https://pandas.pydata.org/

[4]: https://numpy.org/

[5]: https://matplotlib.org/

[6]: https://seaborn.pydata.org/

APPENDIX.

Model	Parameters of the best model	TR:MEE	VL:MEE	TS: MEE
KNeighborsRegressor	algorithm='auto', leaf_size=1, metric='minkowski', n_neighbors=7, p=2, weights='distance'	---	2.727770	2.665090
ExtraTreeRegressor	criterion='mse', max_depth=6, max_features='auto', max_leaf_nodes=20, min_samples_leaf=20, min_samples_split=10, splitter='random'	5.346413	5.737935	5.218577
DecisionTreeRegressor	criterion='mse', max_depth=8, max_leaf_nodes=20, min_samples_leaf=20, min_samples_split=10, presort='deprecated', random_state=0, splitter='best'	3.490605	4.719817	4.331061
Regression Chain	C=10.0, cache_size=200, coef0=0.0, degree=3, epsilon=1.0, gamma=0.1, kernel='rbf', max_iter=-1, shrinking=True, tol=0.001,	1.696402	3.125407	3.165635
MultiOutputRegressor (Direct)	C=10.0, cache_size=200, coef0=0.0, degree=3, epsilon=1.0, gamma=0.1, kernel='rbf', max_iter=-1, shrinking=True, tol=0.001,	2.64765	2.94582	2.95417

Table A.1. Comparison among several regression models from Sklearn

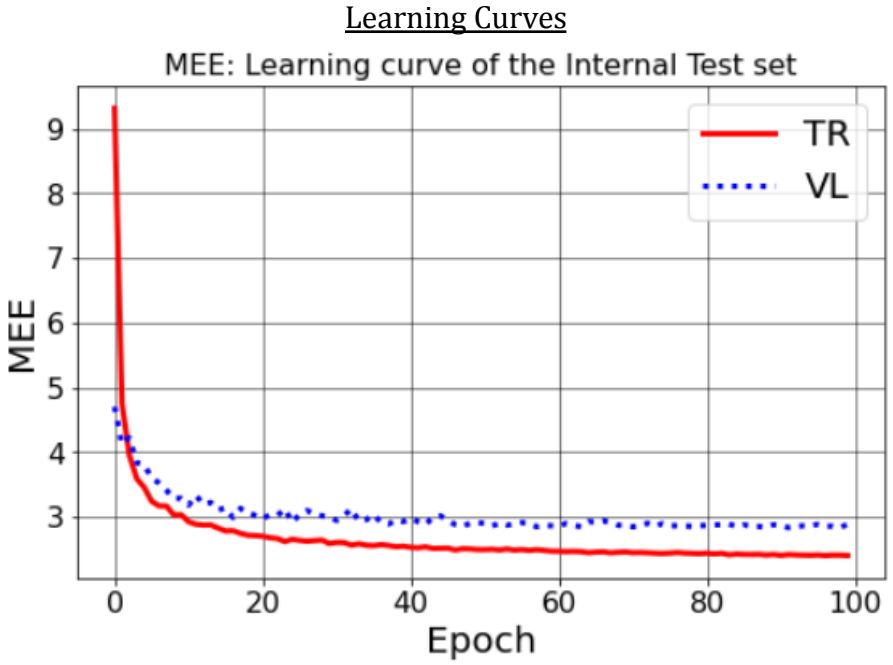


Figure A.1. Learning curve of the Neural Network from Keras

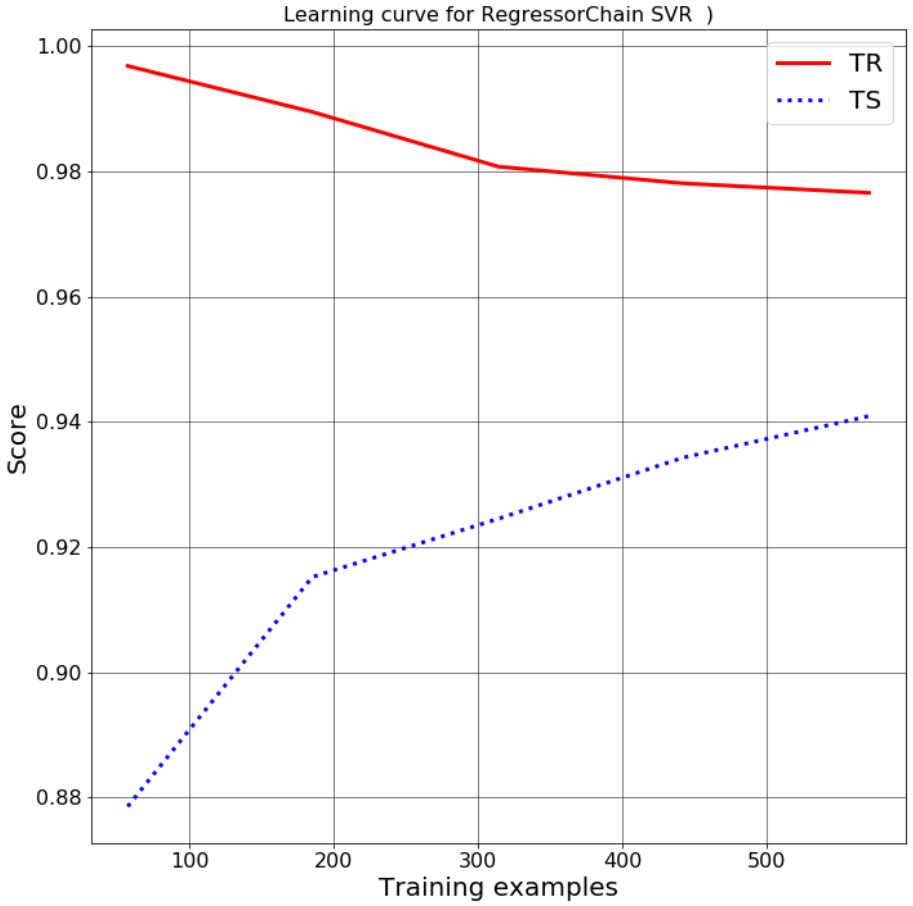


Figure A.4. Learning curve of the Support Vector Regression Chain from Sklearn

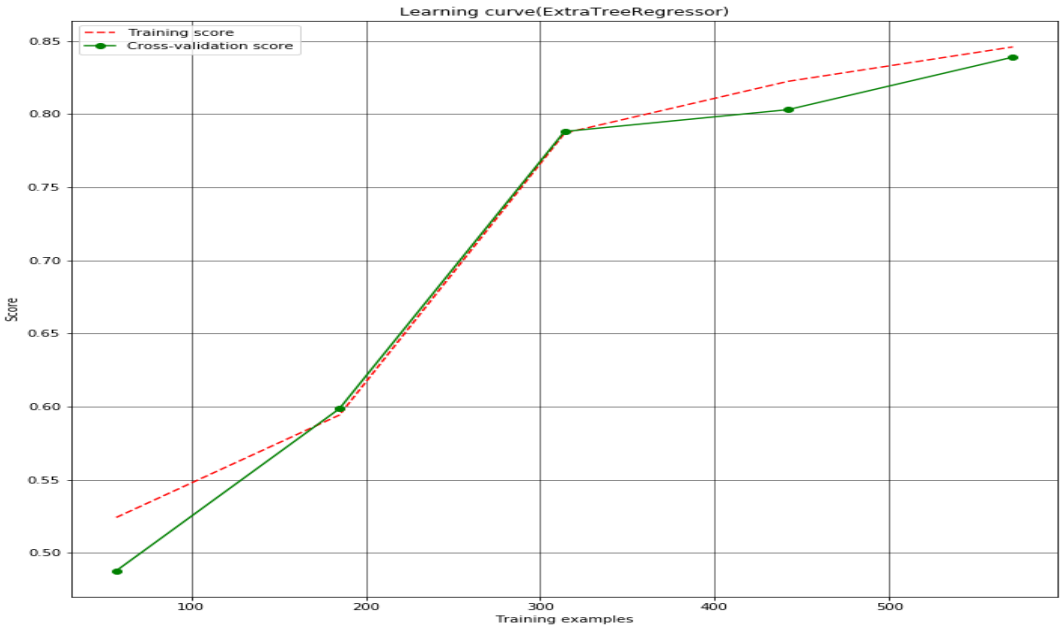


Figure A.3. Learning curve of the Extra Tree Regressor from Sklearn

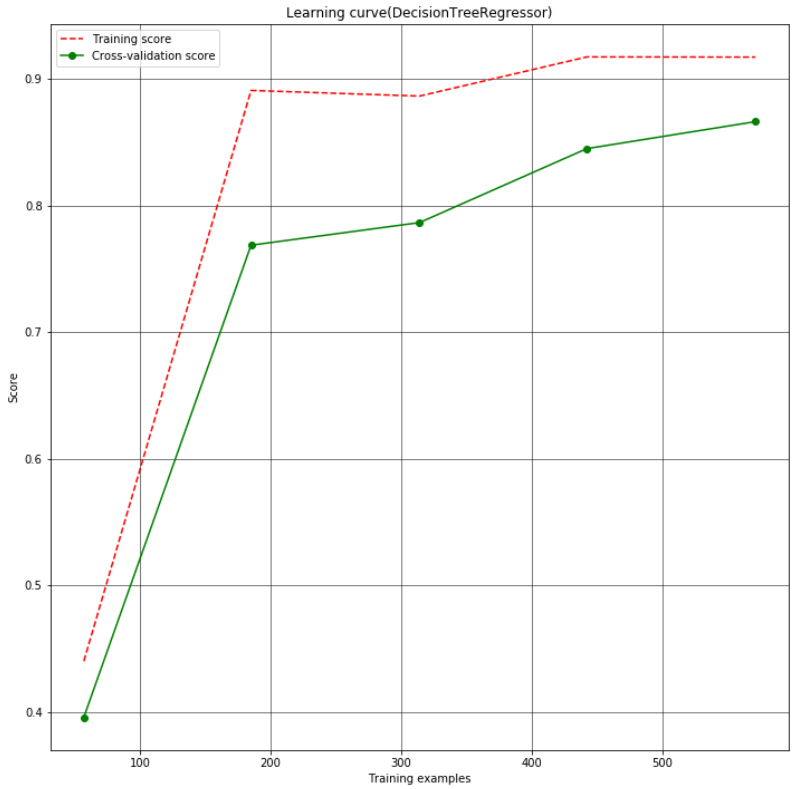


Figure A.4. Learning curve of the Decision Tree from Sklearn

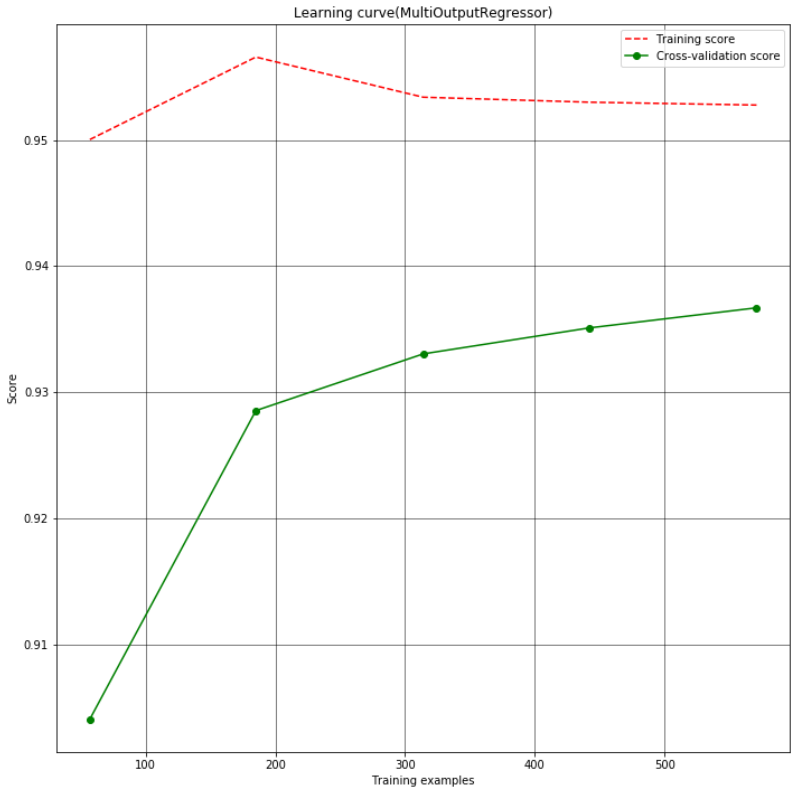


Figure A.5. Learning curve of the MultiOutputRegressor from Sklearn

Miscellanea: samples from the project

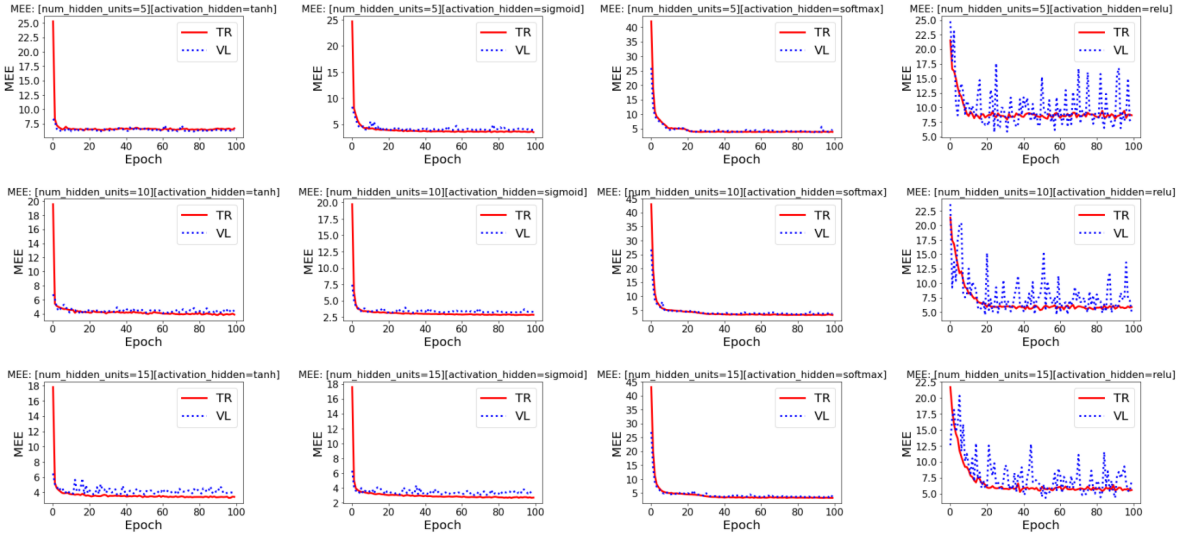


Figure A.6. Example of Screening Phase performed while developing the Neural Network model

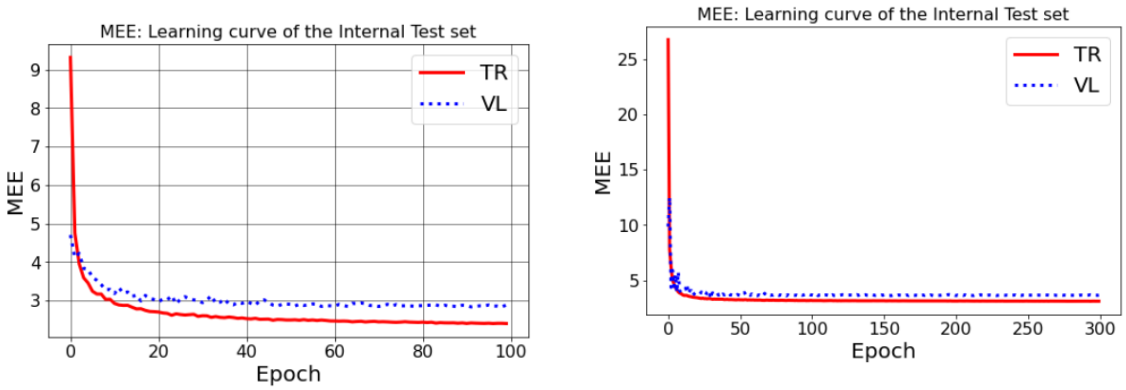


Figure A.7. Comparison of the performance from the neural network models developed and trained with the complete dataset (on the left) and the dataset target of feature selection (on the right)