

A Comprehensive Comparative Analysis of Machine Learning Models

ML course (654AA), A.Y. 2020-21

Abstract

Our main aim of this project was to comprehensively access three different learning models in order to comparatively analyze their significance. The three models we opted for are: Multi layer Perceptron using Keras [1] framework, K-Nearest Neighbors Regressor [2] and Support Vector Regressor [3] using SKlearn framework. Finally, we validate them using a 5-fold cross-validation and test all of them on the same dataset to compare their performance.

1 Introduction

The first objective of the project was to being able to use and assess the three distinct learning models. We start with enough theoretical background of learning algorithm of the models as well as the prior knowledge of different library usage. The reasons behind choosing these models are many for instance the rationale to chose Multi-layer Perceptron was that it has been one of the major argument of the Machine Learning course and we were intrigued to see practicality of this model, then we picked up Support Vector Regressor due to their completely disparate approach to the learning of the hidden functions owing to the fact that we don't have to learn the weights of the model directly as for the artificial neural network, this also lead us to the achieve to develop a model in less time. Eventually, we selected K-Nearest Neighbors Regressor because it is a totally different paradigm of the machine learning since it has a lazy learning algorithm, unlike the previous two models that have eager learning algorithm and they construct directly a model, but

also due to the fact that it works very well on data coming from mixture of different distributions, and since we are unaware of the origin of the data, using this algorithm we cover more hypothesis.

2 Method

All the models have been developed by implementing Python programming in the Jupyter notebook. We started developing the models by following the common procedure of Machine Learning:

1. *Importing Data*: We imported the dataset using Pandas library [4].
2. *Splitting Data*: We split our dataset into a development set and a test set, The development set consists on 85% of data while test set consists on 15% of data. The split of the dataset is performed by using SKlearn library [5] which allows the shuffle of the data automatically. The preprocesssing procedure was trivial in our case so we decided not to implement it.
3. *Visualizing Data*: To visualize the results we achieved and the learning curves throughout the development of the project we used Matplotlib library [6].
4. *Hyperparameter Tuning*: For the optimization of the parameters we ran a grid search using the tools provided by SKlearn [5]. We preferred this tool as it has a feature with the possibility to parellelize the search, even if it provides less flexibility on the results.
5. *Validation method*: To validate our model we used k-folds cross validation. We chose $K=5$ in such a way that we are left with 80% data as a training set and 20% data as a validation set. As we get the scores of all the k-folds, we calculate the mean of the score over all the k-folds. We report the variance in the graph.
6. *Model Assessment*: The assessment of the model was carried out using hold-out method. Thus, the assessing of the selected model has been done on the overall test set.

We used Keras [1] in order to implement the neural network. To observe the behavior of neural network we designed our architecture which depicts a pyramid structure with 2 hidden layers, each of them with a Sigmoid activation function. For training algorithm we used standard gradient descent as it was the most emphasized algorithm discussed during the course and we wanted to explore its practicality of it. We used SGD in its batch form in order to control training instances, so we processed all the training instances before calculating the delta. In an effort to regularize our neural network we used the weight decay technique and to optimize the learning algorithm we used momentum.

For the SVR we used the SKlearn [5] library combined with the MultiOutput Regressor [7] in such a way to enable the model to attain multi output. For this model we use the regularization parameter C , obtaining it through grid search. Finally, we used KNR [2] which includes only one parameter, $K neighbor$, which acts as the controller of model complexity.

3 Experiments

3.1 MONK Result

Before to start with the comparison of the model on the main dataset, we tried to solve a benchmark classification problem called Monk dataset, composed by three different datasets. For solving it we have implemented a small neural network for each of the three datasets using Keras framework. All the parameters of the networks have been tuned using a 3-fold grid search cross-validation with a fixed number of units, 4, SGD with mini-batch equal to 25, and ReLu as activation function. The results for each part are showed in the Table 1, while the learning curve of each monk are presented in the Table 2.

Task	Learnin g Rate	Momentum	Lambda	MSE (TR/TS)	Accuracy (TR/TS)
Monk 1	0.4	0.5	0.0001	TR: 0.0043 TS: 0.0061	TR: 100% TS: 100%
Monk 2	0.5	0.2	0.0001	TR: 0.0042 TS: 0.0043	TR: 100% TS: 100%
Monk 3	0.1	0.1	1e-5	TR: 0.0503 TS: 0.0405	TR: 94.26% TS: 96.75%

Table 1: Monk results

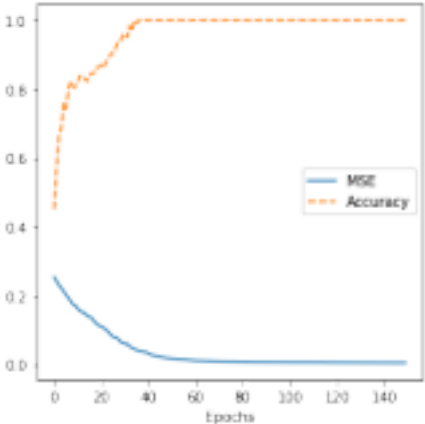
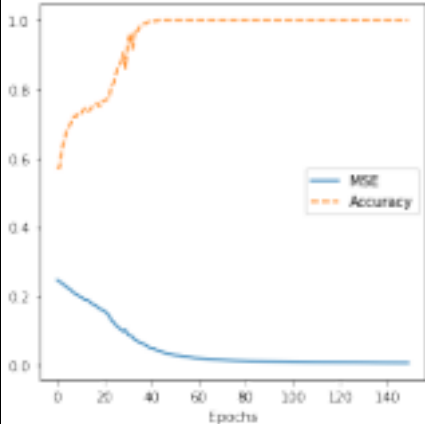
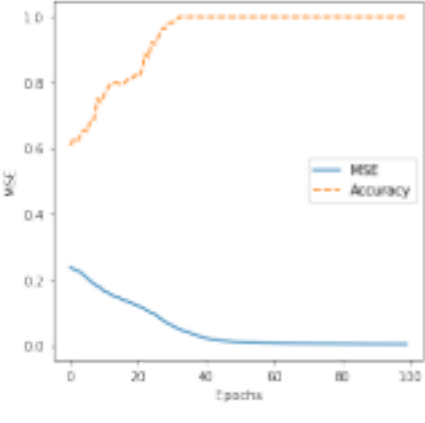
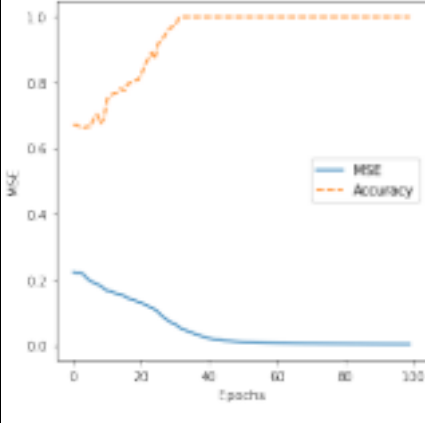
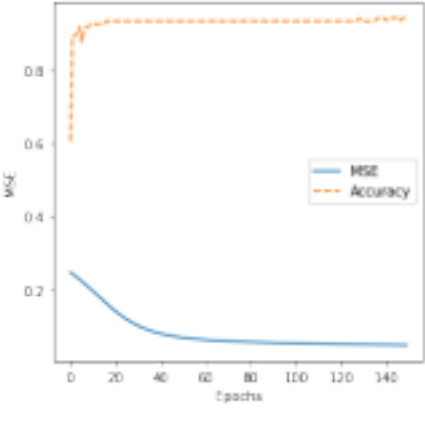
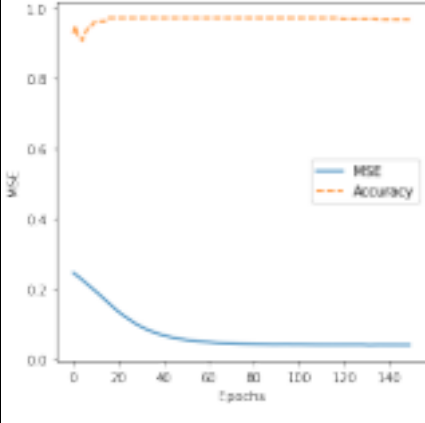
	Training curves	Test curves
Monk 1		
Monk 2		
Monk 3		

Table 2: Monk learning curves

3.2 CUP Result

After defining the methods of the overall process, we started the implementation of the different models. Each model has been validated, as already mentioned, with a

5-fold cross-validation on the development set previously selected with the 85% of the overall dataset. So, for each of the 5 folds, we trained the model on the 68% data (1036 samples) of the initial dataset and we validate it on the remaining 17% data. Then, we selected the best model, for each learning algorithm, with respect to the finest value returned by the mean over the 5 folds of their best configuration.

3.2.1 Keras Neural Network

We start the configuration of the Keras Neural Network by running a grid search over a set of different parameters. The parameters and their range being tried are illustrated in the Table 3, all the combinations have been run for 600 epochs, with a full batch SGD and with a weight initialization generated by a uniform distribution ranging between -0.7 and 0.7.

No. of units of first hidden layer	[25, 30, 40, 50, 80]
No. of units second hidden layer	[5, 15, 25, 35]
Learning Rate	[0.001, 0.01, 0.1, 0.2, 0.3, 0.5]
Momentum	[0.01, 0.1, 0.5, 0.7, 0.9]
Alpha	[0.1, 1e-5, 1e-8, 1e-10, 1e-20]
Activation Function	[sigmoid, softmax]

Table 3: Grid search parameters range

The Grid search took around 15 hours to complete its execution and returned us 6000 combination of parameters over the 5 folds, some of them are showed in Table 4.

Parameters	MEE on train set	MEE on valid set	Rank
alpha: 1e-10, lr: 0.3, mom: 0.5, unit1: 30, unit2:	2.4611(\pm 0.0469)	2.8131(\pm 0.1404)	1

15			
alpha: 1e-8, lr: 0.2, mom: 0.5, unit1: 80, unit2: 35	2.5123(± 0.0309)	2.8188(± 0.0635)	3
alpha: 1e-10, lr: 0.3, mom: 0.5, unit1: 50, unit2: 25	2.4689(± 0.0277)	2.8488(± 0.1988)	168
alpha: 1e-20, lr: 0.2, mom: 0.7, unit1: 25, unit2: 5	2.4689(± 0.1572)	2.6587(± 0.1661)	517

Table 4: Some Grid Search result

The parameters we received from Grid Search were not preeminent since the learning curves were unstable and we did not obtain the smoothness which we were trying to achieve. Thus, we modified the number of the units in both the hidden layers by

increasing them to 50 and 25 respectively, and in this way we got the best parameters with a smooth learning curve, illustrated in Figure 1.

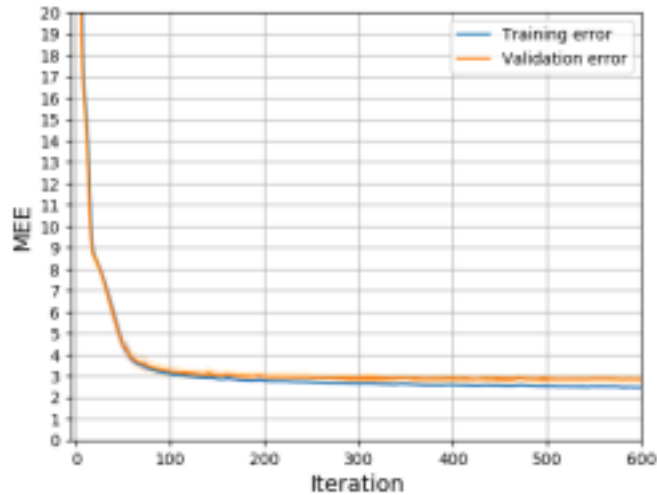


Figure 1: Learning curve with the best parameters

The final best parameters of the model are shown in the Table 5, and in the Table 6 we can see their Mean Euclidean Error on the training set and on the validation set.

No.Unit 1° Layer	No.Unit 2° Layer	Lear. Rate	Momentum	Alpha
50	25	0.3	0.5	1e-10

Table 5: Parameters of the best model

MEE on training set	MEE on validation set
2.4689(± 0.0277)	2.8488(± 0.1988)

Table 6: Result of the best model

We also tried different configuration of the parameters, to observe how they change our model and the learning process. We tried to change the batch size of the SGD, using first 100 and then 1, we tried to decrease the Learning Rate and the Momentum and we also tried to run the training without any weight initialization, some of the learning curves of these experiments are showed in the Appendix A. After reaching to this point we was interested if this type of result can be achieved using a less complex model. In order to do that we recreate the model, using the best parameters illustrated in the Table 5, but, this time, implementing an early stopping technique monitoring the MEE on the validation. This new model achieves better results, both on mean and standard deviation, with less training cycles (around 400), that means a best model for the results as well as for the complexity. Thus, we choose

this final model as the best model for the neural network, the result achieved can be seen in Table 7 while the learning curve is illustrated in the Figure 2.

MEE on training set	MEE on validation set
2.5731(± 0.0408)	2.8313(± 0.1322)

Table 7: Result of the best model with early stopping

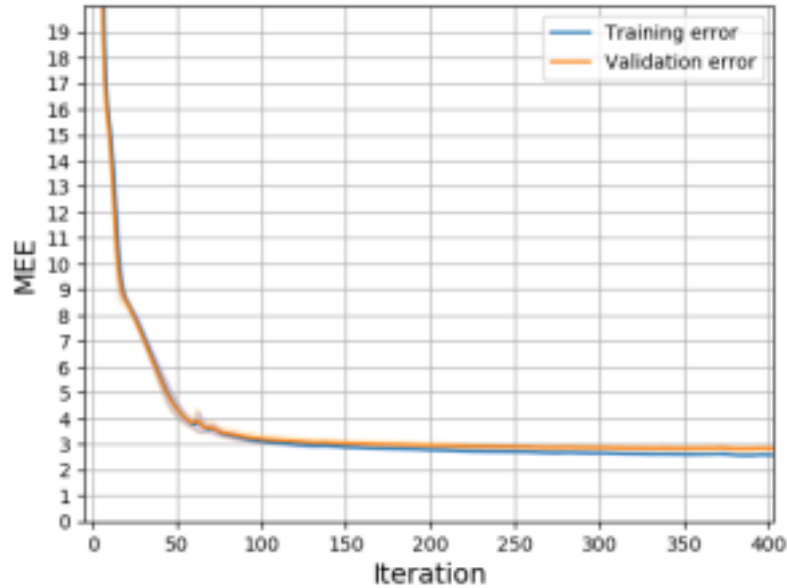


Figure 2: Learning curve with the best parameters and early stopping

3.2.2 SKLearn Support Vector Regressor

For implementing the Support Vector Regressor learning model we used the library MultiOutputRegressor [7] of SKLearn in such a way that double output of the problem can be managed. Then, we ran a grid search, using pipeline library [8], in order to find the best parameters for our data. The range of parameters, we used for running the grid search, are illustrated in the Table 8 and all the combinations have been executed with a RBF kernel. This grid search was quite quick, it took around 15 minutes and returned us around 100 combination of parameters, from which some of them, including the best results, are illustrated in Table 9.

We also tried multiple configuration of the parameters, to observe how they would affect our model and the learning process. We ran the model again using low value of C and high value of C , observing the change of the learning curves, we also execute the learning algorithm with a very low epsilon. In this last case we noted that the low

C	[5, 8, 10, 12, 15]
Gamma	[0.01, 0.1, 0.5]
Epsilon	[0.01, 0.1, 0.2, 0.5, 0.09,

	1]
--	----

Table 8: Grid search parameters range for SVR

Parameters	MEE on train set	MEE on valid set	Rank
C: 15, Epsilon: 0.9, Gamma: 0.1	2.4338(± 0.0379)	2.9465(± 0.1412)	1
C: 15, Epsilon: 1, Gamma: 0.1	2.4497(± 0.0107)	3.0043(± 0.0853)	2
C: 5, Epsilon: 0.5, Gamma: 0.1	2.6205(± 0.0348)	3.0172(± 0.1291)	25

Table 9: Some Grid Search result for SVR

epsilon has a trivial effect on the results and they are quite similar with the model with the best parameters returned from the grid search. The learning curves of these models can be seen in Appendix B.

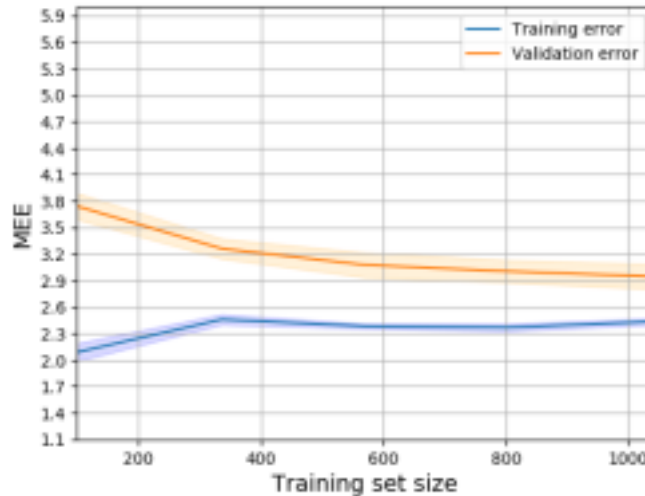


Figure 3: Learning curve with the best parameters of SVR

Finally, to select the best model for the support vector regressor, we opted for the model trained with the best parameters returned from the grid search. The

parameters and the achieved results from then on the training and validation set can be seen on the first row of the Table 9, while, the learning curves of the model are shown in the Figure 3.

3.2.3 SKLearn K-Nearest Neighbor Regressor

For the K-Nearest Neighbor Regressor we used the SKLearn library [2] to implement the overall algorithm. This learning algorithm is model-less and has only single parameters that allow us to control everything. So, with the aim to tune the learning process, we ran a grid search using our own script, setting the algorithm with the value of the parameters 'k_neighbor' in the range between 1 and 51. For each iteration we get the mean value of the MEE for validation and training set and their standard deviation. The results of this script have been plotted and can be observed in the Figure 4.

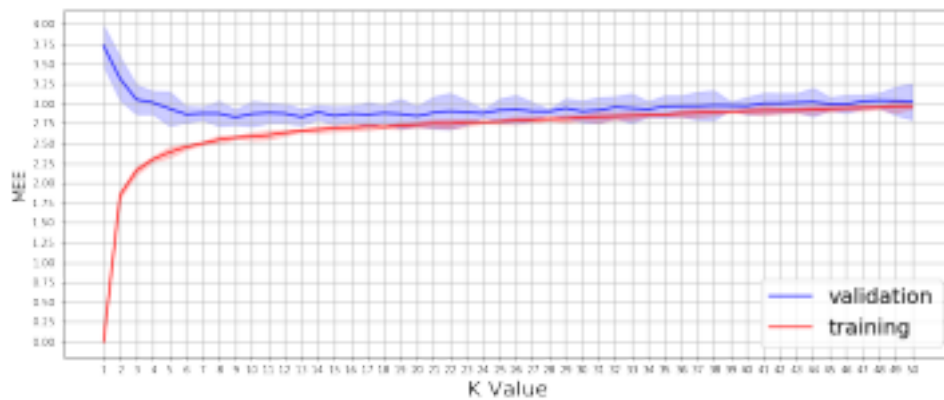


Figure 4: MEE on validation and training set by K_neighbor value

The best model we achieved has the value of K-neighbour equal to 32, which allows us to get the result shown in Table 10.

MEE on training set	MEE on validation set
2.8395(± 0.0310)	2.9477(± 0.1498)

Table 10: Result of the best KNR

4 Conclusion

In conclusion of our work, we went to assess the models on the internal test set created at the start of the project. The models assessed are the best ones founded for each learning algorithms. Thus, the parameters of these models are showed in the Table 5 trained with early stopping for neural network and in the Table 9 for the

support vector regressor, while concerning the k-nearest neighbor regressor we used $k=32$.

	Training score	Validation score	Test score
Neural Network	2.5731(± 0.0408)	2.8313(± 0.1322)	2.9293
Support Vector Regressor	2.4338(± 0.0379)	2.9465(± 0.1412)	3.1954
K-Nearest Neighbor Regressor	2.8395(± 0.0310)	2.9477(± 0.1498)	3.1417

Table 11: Assessment of the models

Thus, according to the Table 11, the best model chosen for the blind test is the neural network.

Nickname: Airric

BLIND TEST RESULTS: Airric ML-CUP20TS.csv

References

- [1] <https://keras.io/api/>
- [2] <https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsRegressor.html>
- [3] <https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVR.html>
- [4] <https://pandas.pydata.org>
- [5] <https://scikit-learn.org/stable/>
- [6] <https://matplotlib.org>
- [7] <https://scikit-learn.org/stable/modules/generated/>

`sklearn.multioutput.MultiOutputRegressor.html`

[8] <https://scikit-learn.org/stable/modules/generated/sklearn.pipeline.Pipeline.html>

Appendices

A - Neural Network additional learning curves

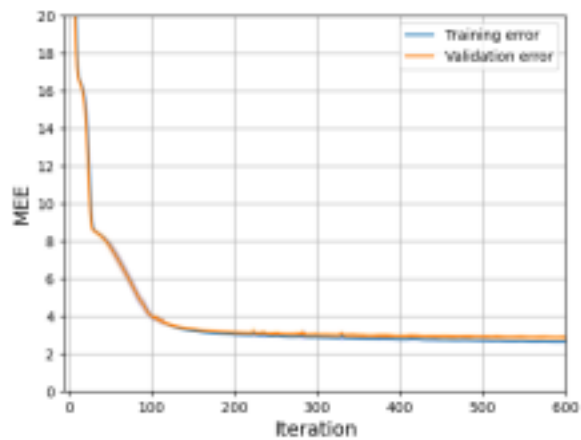


Figure 5: NN Learning curves with no weight initialization

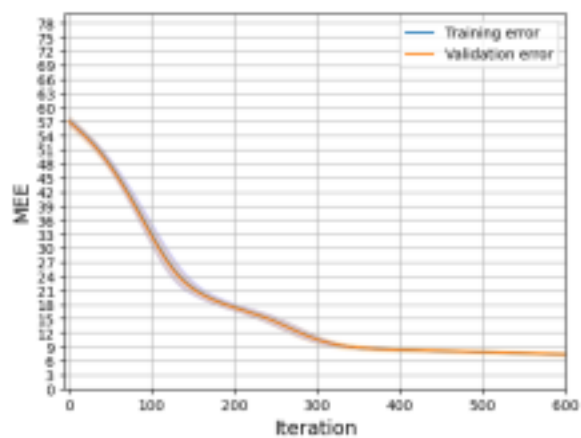


Figure 6: NN Learning curves with learning rate=0.01

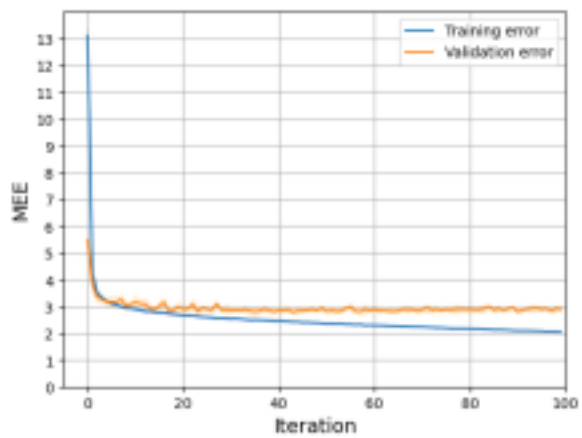


Figure 7: NN Learning curves with On-Line SGD and learning rate=0.01

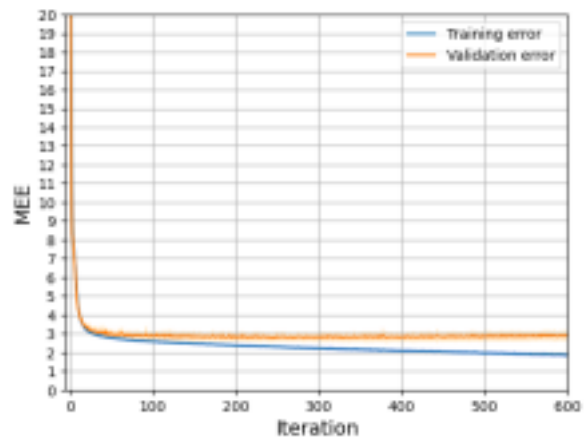


Figure 8: NN Learning curves with Mini Batch (100) SGD and learning_rate=0.1

B- Support Vector Regressor additional learning curves

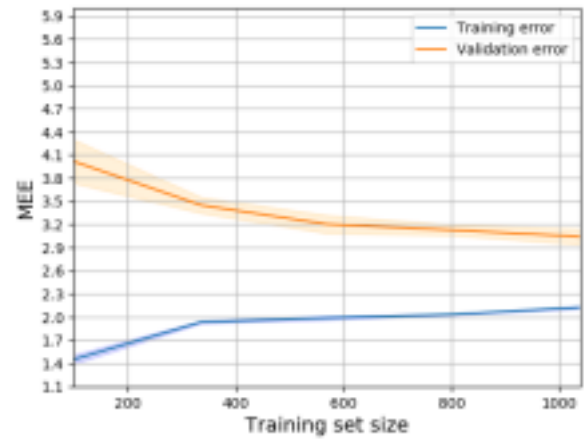


Figure 9: SVR Learning curves with C=70