POWER UP

Piyush Tada, p.tada@studenti.unipi.it
Sharath Chandra Yanamandra, s.yanamandra@studenti.unipi.it
Meghna Singh, m.singh4@studenti.unipi.it

ML course (654AA), A.Y. 2021-22

1st January 2023

Project Type B

Abstract

The preliminary goal of our project is to compare the different machine learning models. To achieve this we need to have extensive access to the chosen models and find out the best among them. Here in our project we chose: Support vector machine regressor, K-Nearest neighbour regressor and Multi layer Perceptron on the given dataset. Later we validate them using 5 fold cross validation and compare on the same dataset. According to our results the best model is Neural Network.

1 Introduction

We begin with a sufficient theoretical underpinning of the machine learning algorithms from the course as well as prior knowledge of various library implementation. The project's initial goal was to be able to utilize and evaluate the three unique learning models. From the Machine learning course we came across many efficient learning algorithms, in which we were much interested in a few mentioned below and wanted to know how the algorithm works for this scenario. We choose Support vector machines regressor for a very specific reason: because of their entirely different method to learning hidden functions due to the fact that we don't have to learn the weights of the model directly as with the artificial neural network, which also allowed us to construct a model in less time. Secondly, we chose K-Nearest neighbour because of its uniqueness in lazy learning approach and behaves well for mixture of data from various distributions. Since SVM is an eager learning and KNN is lazy learning, we were excited to know the best model for this scenario. And finally, we needed an Artificial neural network in action as Multi layered perceptron would be the best choice and taught well in the course and we were interested to discover how practical this model was.

1.1 Pre-sets for the code:

We used anaconda Jupyter notebooks for model implementation. We used python as a primary programming language. We began by using the standard machine learning process to construct the models:

- Pandas package to import the dataset.
- Our dataset was divided into a training set and a test set, with the training set containing 70% of the data and the test set 30%. The dataset was divided using the SKlearn package.
- We utilized the Matplotlib library to illustrate the progression of the project's learning curves.

- Using the SKlearn tools, we conducted a grid search to optimize the parameters. Despite offering less flexibility in terms of the results, we liked this tool since it contains a function that allows you to parallelize the search.

- Our model was validated using k-fold cross validation. We selected K=5 so that we would have 80% of the data as a training set and 20% of the data as a validation set. We determine the mean score across all k-folds as we collect the results of all k folds.

2 Method:

We developed the code and tested on Jupyter notebooks and python scripts running them both locally and on Google Colab. Firstly, Our Neural Network is developed in Keras, TensorFlow as backend. The Neural network is having two hidden layers and sigmoid as an activation function. We used gradient descent for training and stochastic gradient descent batch form for manage training instances which helps in getting delta.

Initially we divided the dataset into training and test sections, the test is used only in final testing phase, as it is randomly selected from original dataset.

In order to enable the model to produce several outputs, we used the Multioutput Regressor in conjunction with the SKlearn package for the SVR. We employ regularization parameter C for this model, which we obtained by grid search. And for KNR we have the k value used for model complexity.

For MONK 1,2, and 3 we have implemented grid search to find the best hyperparameter for model selection. Followed by k fold cross validation. Each of the parameters are explained in the experiment section below.

We have used stochastic gradient descent (SGD) with Mean square error (MSE) the loss function and Mean Euclidean error (MEE) for model selection and for final test.
And finally for activation function, we used ReLU and did a few experiments on other activation functions.

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

3 Experiments

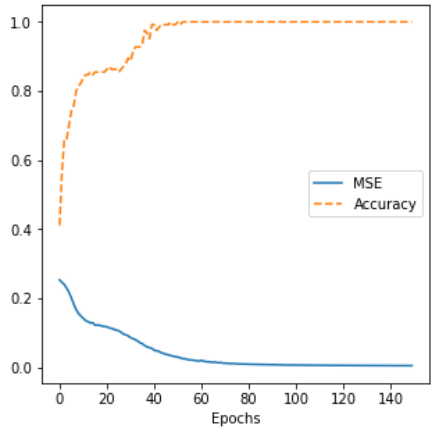
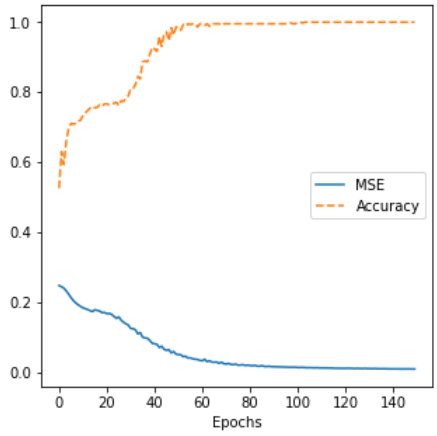
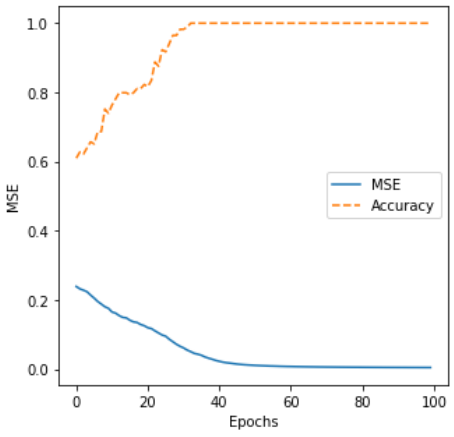
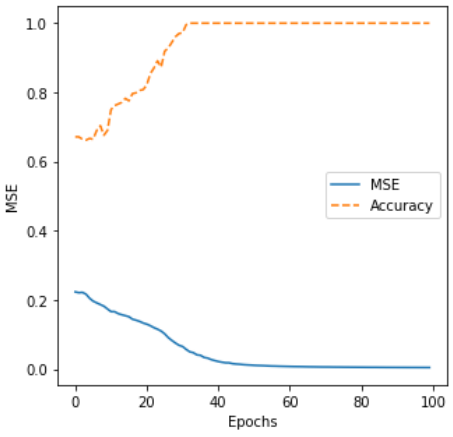
3.1. Results for MONK

Monk dataset is composed of 3 sub datasets and we started by solving the benchmark classification. To solve this we implemented a neural network using keras framework, our neural network consist of 17 input dimensions and 1 output node. To find the best hyperparameter we used grid search on the learning rate (n), momentum (y) and wight decay (lambda). We ran the monk all dataset for 150 epochs and the results are shown in the table 2.

The parameters are tuned using 3 fold cross validation with 4 number of units. We also used stochastic gradient descent on mini batch = 25 with ReLU as an activation function. The results of MONK dataset along with the accuracy and MSE along with the learning curve are mentioned in the table 1 and 2.

Task	Learning Rate	Momentum	Lambda	MSE (TR/TS)	Accuracy (TR/TS)
Monk 1	0.1	0.9	0.0001	TR: 0.00514 TS: 0.00865	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: best models found for MONK

	Training Loss curves	Test Accuracy Curves
Monk 1		
Monk 2		

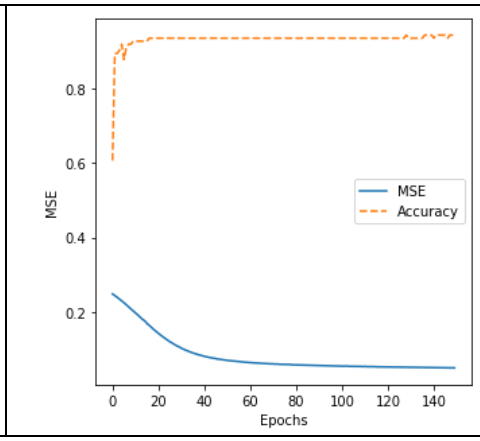
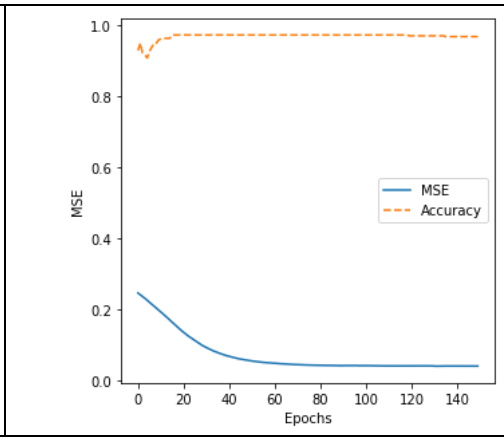
Monk 3		

Table 2: MONK 1,2 and 3 loss and accuracy curves

3.2. ML Cup results

For the model implementation first we split the dataset into 70% and 30% ratios. Then we implemented different validated models with k fold cross validation, in which we trained and validated the model.

3.2.1. Neural Network.

On a set of parameters, we used grid search to discover the optimum hyperparameters. In order to do this, we used stochastic gradient descent across more than 500 epochs and a weight initialization by uniform distribution with a range between -0.7 and 0.7.

First hidden layer units	[25, 30, 40, 50, 80]
Second hidden layer units	[5, 15, 25, 35]
Alpha	[0.1, 1e-5, 1e-8, 1e-10, 1e-20]
Momentum	[0.01, 0.1, 0.5, 0.7, 0.9]
Learning Rate	[0.001, 0.01, 0.1, 0.2, 0.3, 0.5]
Activation Functions used	[sigmoid, softmax]

Table 3: Grid search combinations for Neural network.

After performing the grid search on the following combinations we are able to find a few best combinations. The top are mentioned in the below table.

Alpha	Learning rate	Momentum	Unit 1	Unit 2	MEE on train	MEE on test
1e-10	0.3	0.5	30	15	2.3110	2.6715
1e-8	0.2	0.5	80	35	2.4123	2.6818
1e-10	0.3	0.5	50	25	2.4698	2.6588

Table 4: Grid search results

From the results of the grid search we came to a conclusion of choosing best parameters and to obtain a smooth learning curve between MEE and Iterations. The graph is given below in figure 1.1.

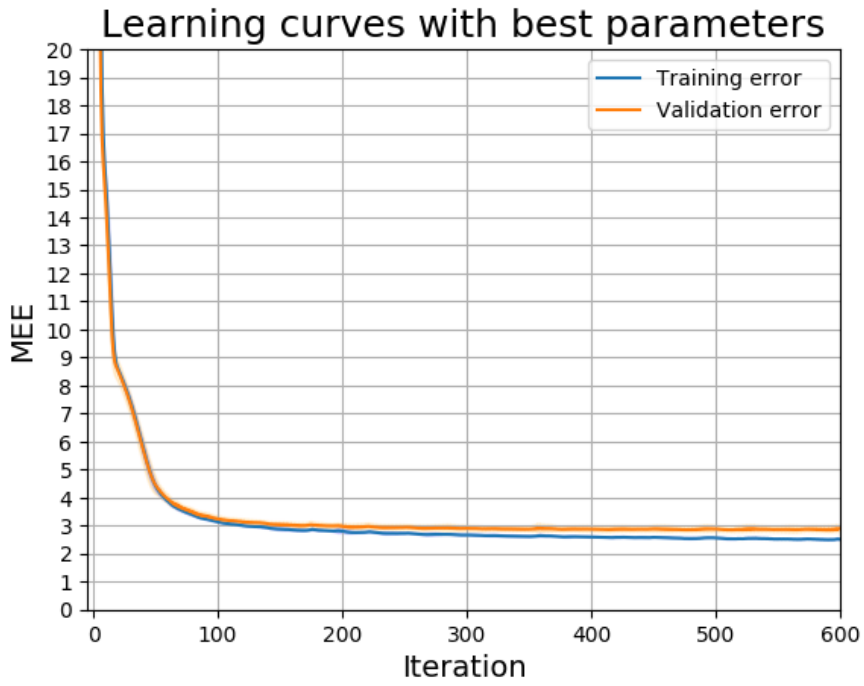


Figure 1.1. Learning curve from the best parameters chosen

To obtain a smooth learning curve we choose the below hyperparameters listed in the table 5 followed by Mean Euclidean error on validation and training sets.

No: units 1st layer	No: units 2nd layer	Learning rate	Momentum	Alpha
50	25	0.3	0.5	1e-05

As we have many different parameters and wanted to perform a few experiments by considering the parameters achieved. For instance changing the learning rate to observe if there is any changes. Switching up and down the values of the momentum and batch size of SGD and sometimes training without weight initialization and hoping to get a better results. Some were pretty good and some were not satisfactory. To achieve this we need to re train the model with the best parameters we found by

grid search and we did it. We had a question of why not using an early stopping technique because the model is performing best for the last few cycles. And hence its true,, we implemented early stopping on MEE on validation and it achieved best results on mean and standard deviation. With this we came to a conclusion by considering the best hyperparameters we are able to get the best model for neural network.

The results of the models are illustrated below in the table and the graphs for our experiment purposes are listed after the bibliography.

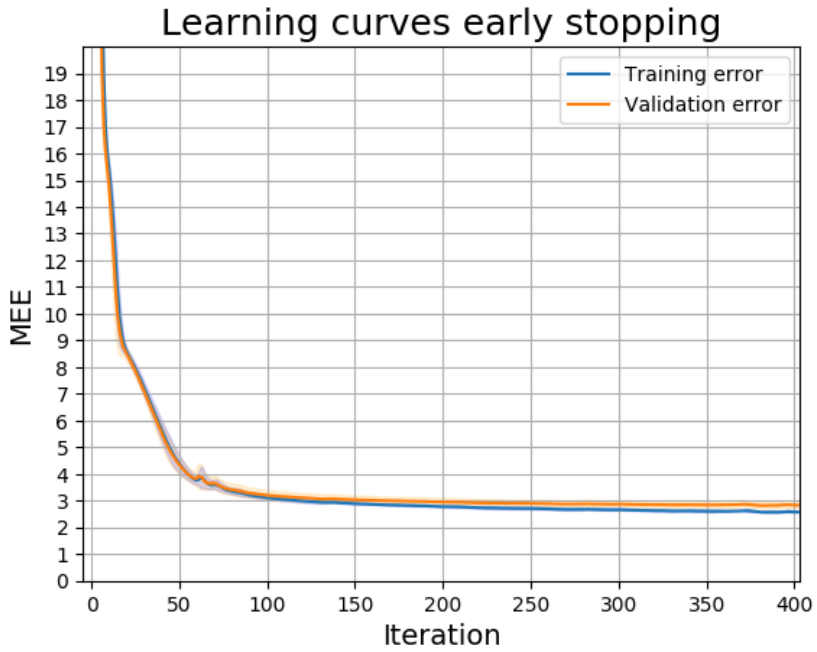


Figure 1.2. learning curve with early stopping

MEE on training set	MEE on validation set
2.5731(±0.047)	2.8313(±0.1322)

Table 5. results with early stopping

3.2.2. SVM regressor

Support vector machines have a unique quality of giving double outputs, hence we implemented using Multioutput regressor from SKlearn by which we can manage and differentiate the double output problem. We considered a list of parameters for grid search to find the best ones among the following. The gird search parameters are listed in the table 6. In SVM we have many kernels in which we choose

RBF for execution. After running the grid search we are able to consider the values of the c, Gamma and the Epsilon from the table 7 which also is having the MEE on train and Validation sets.

C	[5,8,10,12,15]
Gamma	[0.01, 0.1, 0.5]
Epsilon	[0.001 , 0.1, 0.2, 0.5, 0.09, 1]

Table 6. grid search parameters for SVM

C	Gamma	Epsilon	MEE on train set	MEE on val set
8	0.1	0.2	1.2138 (0.0106)	1.4465 (0.0353)
15	0.1	0.9	2.4332(0.0298)	2.946 (0.0845)

Table 7. Gird search results for SVM

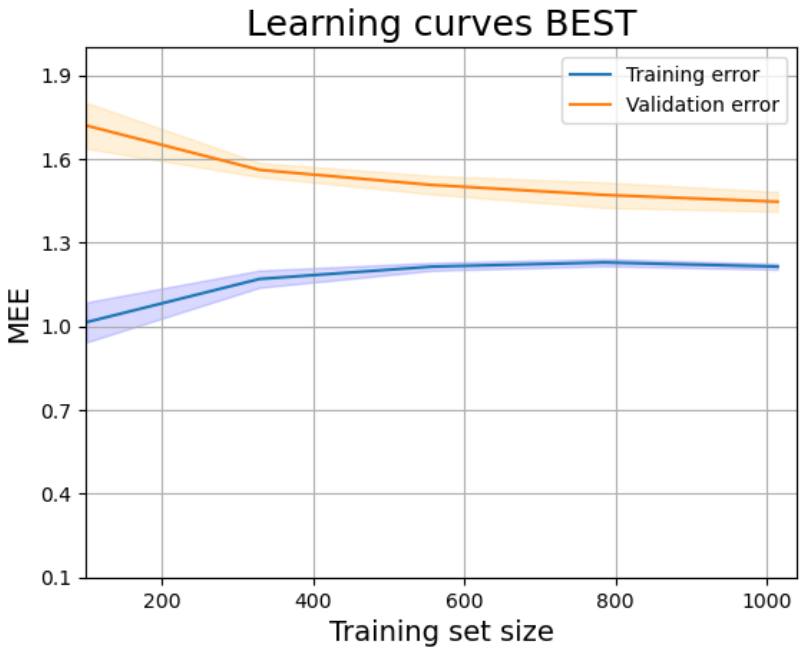


Figure 1.3. Learning curve with best parameters of SVM

Later after this implementation we wanted to do a few experiments on altering the C and Epsilon values. So we had a learning curves with high and low C, high and low epsilon. But the results were not what we were expecting so we need to stick with the results from grid search. The graphs for these are appended after the conclusion section.

3.2.3. K Nearest Neighbor regressor

In K Nearest Neighbor algorithm initially our task is to find the value of K, later we need to find the MEE for validation and training sets. To determine the K, we implemented in SKlearn with the values range between 1 and 51. We considered Minkowski as our metric and are able to achieve K = 44. The MEE on training and validation sets are listed in the table 8.

K value	MEE on training set	MEE on val set
44	1.4228 (0.0126)	1.4517 (0.0604)

Table 8. results of K nearest neighbor

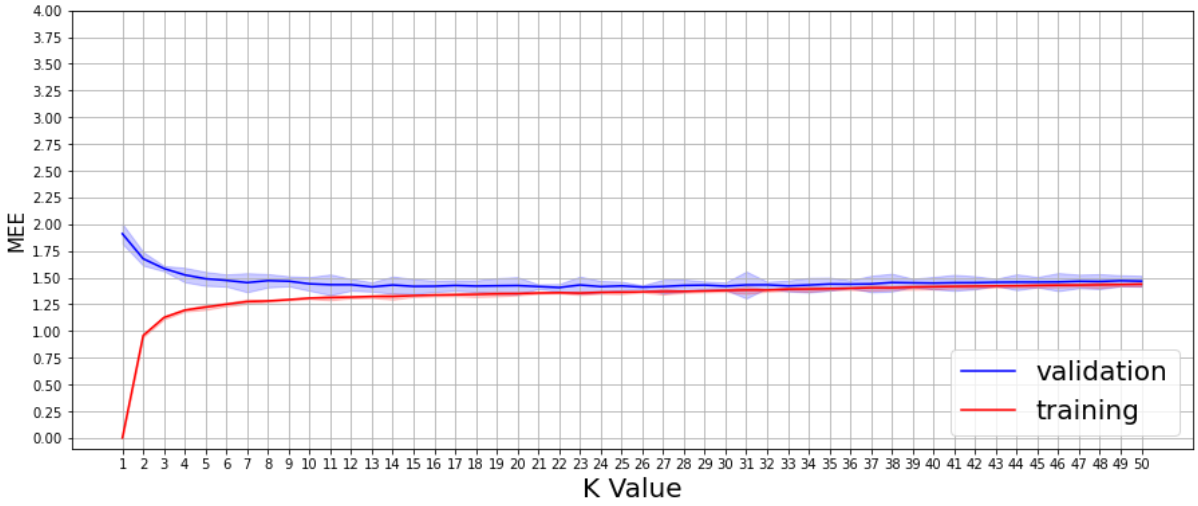


Figure 1.4. MEE on training and validation set by K value

4 Conclusion

	Training score	Validation score	Test score
SVM regressor	1.2138 (0.0106)	1.4465 (0.0353)	1.5049
KNN regressor	1.4228 (0.0126)	1.4517 (0.0604)	1.5368
Neural network	2.5731(0.0470)	2.8313(0.1322)	2.9293

From the tests we performed the standard SVM regressor showed the best results among the KNN and Neural Network. As SVM and KNN are very similar in terms of results. The neural network could be more efficient if fine tuned and trained for much longer time. But, as a standard benchmark algorithm like SVM is able to achieve best results in very short amount of time. We considered the parameters of each algorithm from table 1,4,7 for SVM, KNN and NN. From the above results the model we chose for blind test is support vector machine and the results are given : PowerUp_ML-CUP22-TS.csv

Bibliography

[1] F. Pedregosa et al. “Scikit-learn: Machine Learning in Python.” In: Journal of Machine Learning Research

[2] https://keras.io/api/

[3] https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVR.html

[4] https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsRegressor.html

Appendices

Additional Learning curves of Neural Network:

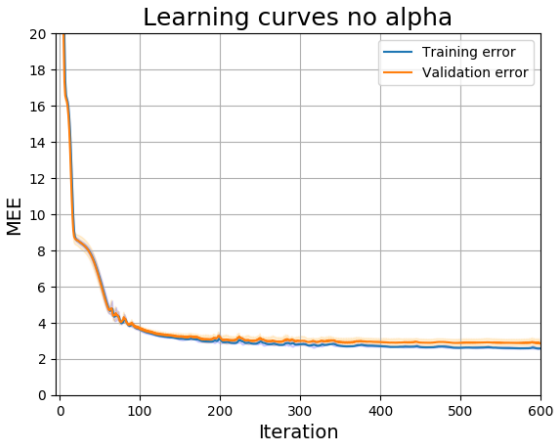


Figure 2.1. no alpha learning curve

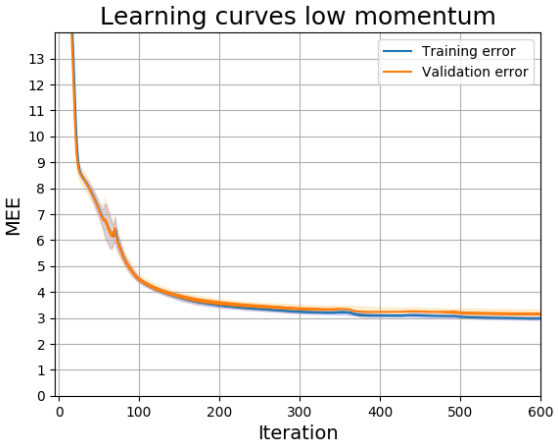


Figure 2.2. Learning curve with low momentum

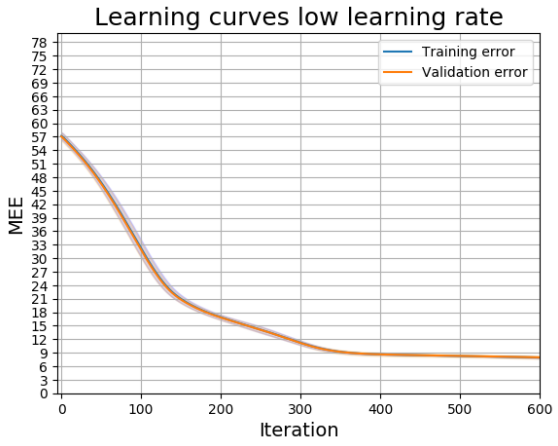


Figure 2.3. Learning curve when low learning rate

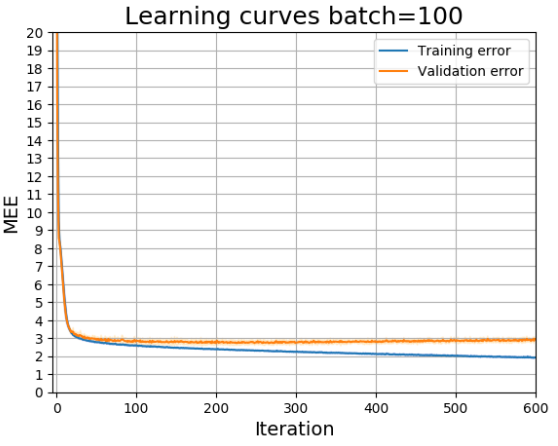


Figure 2.4. Learning curve with batch 100

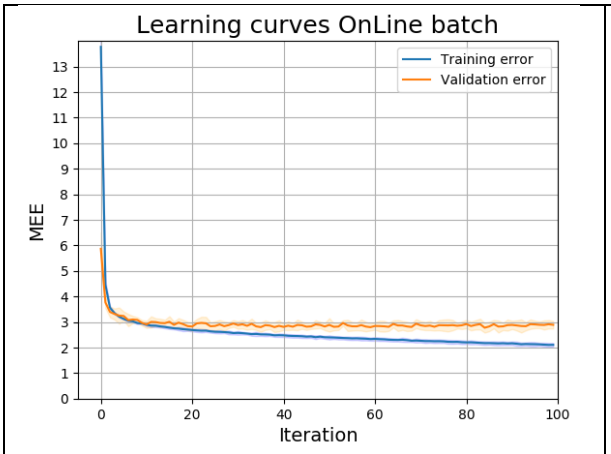


Figure 2.5. Learning curve online batch

Additional learning curve charts for Support vector machine regressor:

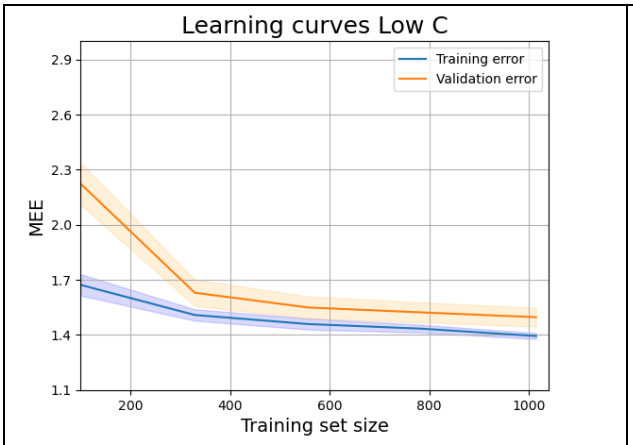


Figure 2.6. Low c learning curve

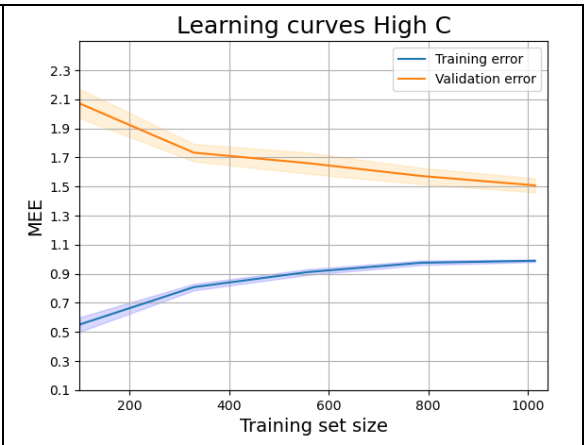


Figure 2.7. high c learning curve

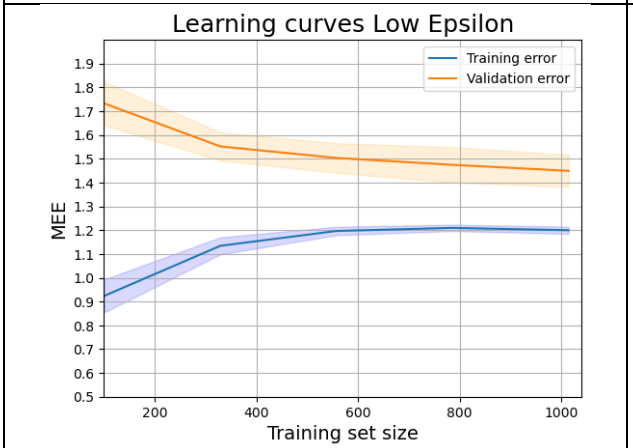


Figure 2.8. low epsilon learning curve