Machine Learning Project

Gloria Segurini, Andrea Carnevale, Federico Canepuzzi
Data Science and Business Informatics g.segurini@studenti.unipi.it
Data Science and Business Informatics a.carnevale2@studenti.unipi.it
Data Science and Business Informatics f.canepuzzi@studenti.unipi.it
ML course (654AA), Academic Year: 2021/2022
Date: 22/05/2022
Type of project: **B**

Abstract

The aim of this report is to present the developement of several models and the evaluation of the relative performances. Once we tuned the single models, we combined them through two distinct ensemble models so as to achieve the best generalization capability.

1 Introduction

For this project it is required to solve a regression task on the CUP dataset. It is focused on trying out the different models, selecting the best hyperparameters configuration and compare their performances. We used: KNN, SVM, LBE, Random Forest and Neural Network. These models were first used to perform a classification task on the well-known MONK dataset as a benchmark. Finally, we implemented from scratch a Stacking and a Voting ensemble by combining the tuned estimators above.

2 Method

2.1 Code

The notebooks were written in Python language with the Numpy and Pandas libraries to manage the data structures, Scikit-learn and Keras libraries to develop the models. The code was run both on Google Colab and Kaggle remote machines in order to perform many gridsearch simultaneously and exploit a more powerful computational capability.

For every model we always follow the same schema: first we choose the best hyper-parameters through a gridsearch and then we do the retraining on the entire training set to build the final model.

The library of Keras is specific for NN models and it doesn't support an own gridsearch; it is possibile to use a wrapper from Scikit-learn but we decided to implement from scratch a gridsearch in order to avoid the switch between Keras and Sklearn, have more flexibility and try to speed up the grids.

Moreover, since Scikit-learn library doesn't allow to manage a multi-labeled output for the ensemble models, we implemented them from scratch. For the Voting model we simply

computed the mean of the prediction of the base estimators, while for the stacking model we collected the base estimators' predictions in two dataframes (one for each output label) and on top of them we trained two Ridge regressors. The performances are evaluated putting together the outputs of the Ridges (for further reference see paper [1]).

2.2 Preprocessing procedure

On the MONK dataset we used the *getdummies* function which changes the number of features from 6 to 17 all in binary format, while for the Cup dataset it wasn't necessary because we don't have categorical attributes. Moreover, since the mean and standard deviation of the patterns on the CUP are similar, we didn't apply any normalization.

2.3 Validation schema

On the MONK, the test set is already separated from the training set, while on the CUP we performed an holdout splitting (90% TR - 10% TS). In order to select the best hyperparameters, we applied a Cross Validation to every model with 5 folds, except for the Neural Network where we only used 4 folds, so as to save computational time. Since on the MONK we performed a classification task, we also applied a stratification procedure of the folds. In NN and in SVM models for the Cup, we tested a great number of parameters through a coarse to fine gridsearch approach: this method let us save a great quantity of time.

For the Voting ensemble the predictions from the base estimators are done with a 4-folds-cross-validated approach so that we can evaluate the MEE between the average base estimators' predictions and the target values. For the Stacking we followed the same cross-validated approach for the prediction from the base models in order to have two new training datasets; then we fit the data on the two ridge regression models (using a gridsearch with 5 fold cross-validation for the choice of the hyper-parameters) and calculate the MEE.

3 Experiments

3.1 Monk Results

The MONK results are shown in table 1.

For all the three tasks, in NN model, we used the Relu activation function for the hidden layer and Sigmoid activation function for the output. The loss was computed using Mean Square Error (MSE) on a full batch size over 400 epochs. We can notice from the results of the MONK 3 task the importance of the regularization in order to avoid overfitting. Indeed, by using L2 regularization we obtain a worse performance on TR but a better one on TS. The learning curves of the NN models are in figure 1, 2, 3, 4, 5, 6, 7 and 8.

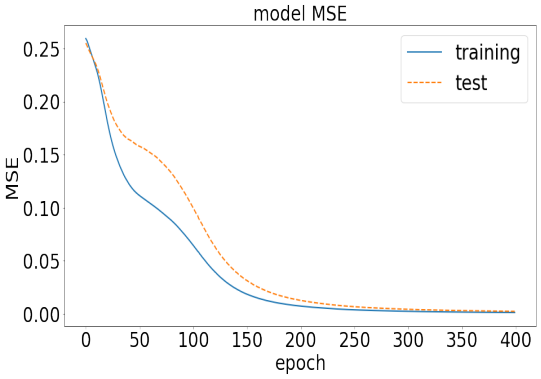

Figure 1: MONK 1 MSE

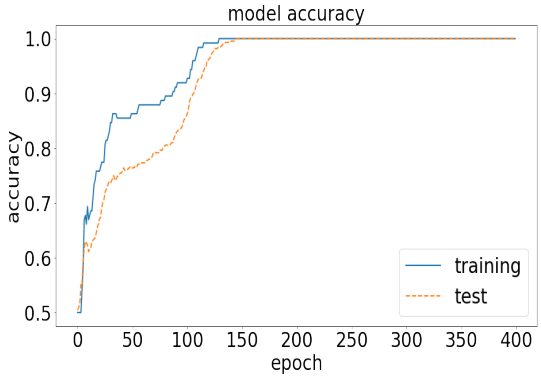

Figure 2: MONK 1 accuracy

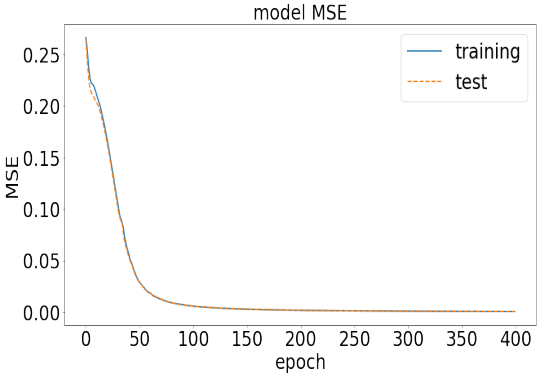

Figure 3: MONK 2 MSE

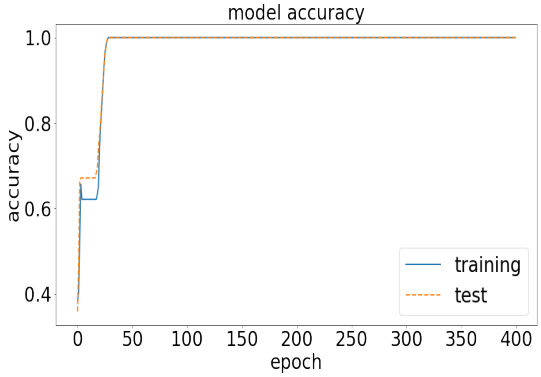

Figure 4: MONK 2 accuracy

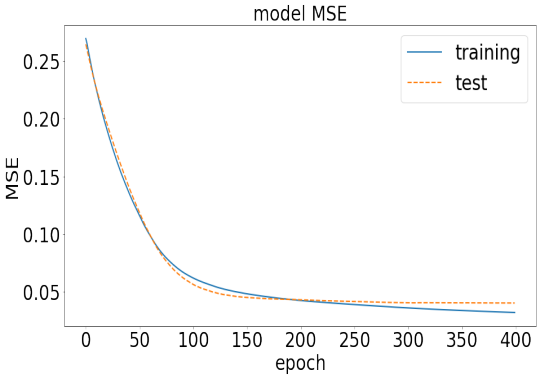

Figure 5: MONK 3 MSE

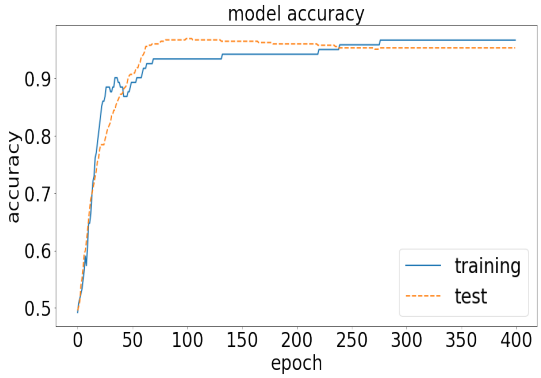

Figure 6: MONK 3 accuracy

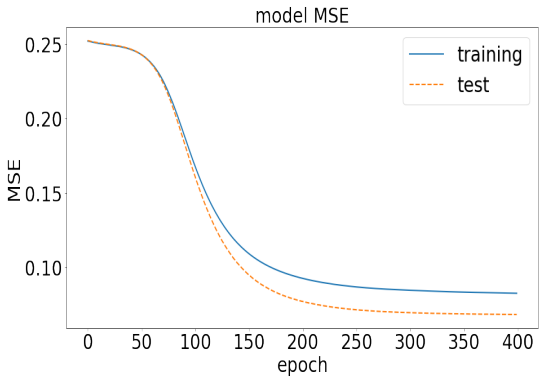

Figure 7: MONK 3 + reg. MSE

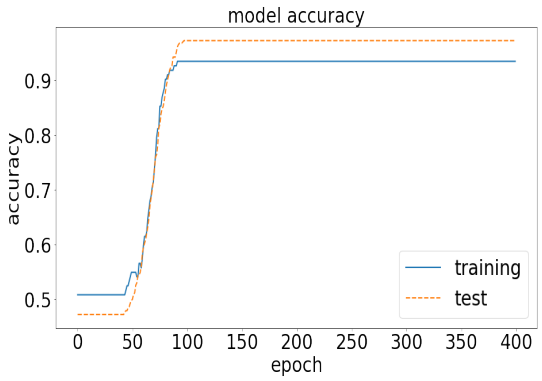

Figure 8: MONK 3 + reg. accuracy

Task	**Hyper-parameters**					**MSE**		**Acc**	
NN	*weight*	*archit*	*eta*	*alpha*	*lambda*	*TR*	*TS*	*TR*	*TS*
MONK1	norm (0.4)	(17,4,1)	0.5	0.7	0	0.0014	0.0024	100%	100%
MONK2	norm (0.2)	(17,4,1)	0.7	0.7	0	6.9e-4	6.8e-4	100%	100%
MONK3	norm (0.4)	(17,5,1)	0.3	0.3	0	0.03	0.04	97%	95%
MONK3+reg	uni [-0.3,0.3]	(17,3,1)	0.3	0.3	0.01	0.08	0.06	93%	97%
KNN	*n_neighbors*	*weights*	*p*					*TR*	*TS*
MONK1	8	uniform	2					80%	74%
MONK2	23	uniform	2					67%	66%
MONK3	69	distance	1					100%	95%
LBE	*solver*	*alpha*	*degree*	*interact*				*TR*	*TS*
MONK1	saga	1	2	False				100%	98%
MONK2	saga	100	2	False				69%	67%
MONK3	saga	100	4	True				96%	96%
SVM	*kernel*	*degree*	*C*					*TR*	*TS*
MONK1	poly	2	10					100%	100%
MONK2	poly	2	100					100%	100%
MONK3	linear	-	0.1					93%	97%
FOREST	*max_depth*	*criterion*	*min_split*	*min_leaf*	*max_feat*			*TR*	*TS*
MONK1	5	gini	15	5	8			100%	100%
MONK2	15	gini	35	35	17			62%	67%
MONK3	25	gini	15	5	17			93%	97%

Table 1: Prediction results obtained for the MONK tasks

3.2 Cup Results

We first divided the dataset into TR(90%) and TS(10%) with the Hold-out method. Afterwards, we executed a gridsearch with Cross-Validation for every model, select the best hyperparameters on which we retrain the model on the full TR. For the model selection phase, we used a Cross-Validation with 5 Folds, except for the NN model, where we used 4 in order to speed up the process. The choice of the best hyperparameters was made on the basis of the means of the performances on the VL. After the retraining with the best hyperparameters we executed the model assessment on the test set. The performances were evaluated both on the VL and on the TS by the Mean Euclidean Error (MEE).

In the NN models we used the Early stopping technique with 'patience' equal to 50 epochs, in order to avoid overfitting (together with L2 regularization) and speed up the grids. During the retraining phase of these models, we chose the stopping epoch based on the average MEE of the training set obtained on the folds of the cross-validation. In this way it was possible to use the entire TR dataset for the retraining. The training error values at which the models have to stop are shown in table 2.

Moreover, in NN models we used the Hyperbolic Tangent (Tanh) activation function for the hidden layers and linear activation function for the two output units. The loss was computed using the mean euclidean error (MEE) together with L2 regularization. The learning curves of the NN models are reported in figures 9, 10, 11, 12, 13, 14, 15 and 16 (APPENDIX A).

The hyper-parameters evaluated and then chosen through the gridsearch for the various models, are shown in table 3.

Model	**TRAINING MEE** *TR mean (±std)*
NN SGD batch	0.834 (±0.072)
NN SGD mb (100)	1.007 (±0.020)
NN SGD online	0.966 (±0.027)
NN ADAM batch	0.876 (±0.042)

Table 2: Average MEE on TR from CV - CUP task

Grid-search Hyper-parameters						
NN SGD batch						
Param	*weight*	*layer*	*unit*	*eta*	*alpha*	*lambda*
Range	normal [0.2...0.4]	[2,3]	[10...50]	[0.0005...0.07]	[0.8...0.975]	[0.0005...0.01]
Best	normal (0.4)	3	40	0.03	0.95	0.001
NN SGD mb 100						
Param	*weight*	*layer*	*unit*	*eta*	*alpha*	*lambda*
Range	normal [0.2...0.4]	[2,3]	[10...50]	[0.0001...0.05]	-	[0.0001...0.05]
Best	normal (0.2)	3	50	0.025	-	0.0005
NN SGD online						
Param	*weight*	*layer*	*unit*	*eta*	*alpha*	*lambda*
Range	normal [0.2...0.4]	[2,3]	[10...50]	[0.000001...0.0005]	-	[0.0005...0.05]
Best	normal (0.2)	2	50	0.0005	-	0.001
NN ADAM batch						
Param	*weight*	*layer*	*unit*	*eta*	*beta1/beta2*	*lambda*
Range	normal [0.2...0.4]	[2,3]	[10...50]	[0.0001...0.025]	[0.6...0.99]	[0.0005...0.01]
Best	normal (0.4)	3	40	0.025	0.9 / 0.7	0.001
SVM						
Param	*kernel*	*degree*	*C*	*epsilon*	*gamma*	
Range	[linear, poly, rbf]	[1...5]	[0.1...100]	[0.01...10]	[scale, auto]	
Best	rbf	-	10	0.5	auto	
RANDOM FOREST						
Param	*n_estimator*	*max_depth*	*min_split*	*min_leaf*	*max_feat*	*bootstrap*
Range	[100]	[5...60]	[2...20]	[2...20]	[2...10]	[True,False]
Best	100	20	2	2	3	False
LBE						
Param	*solver*	*degree*	*alpha*	*interact*		
Range	[saga]	[2...6]	[0.001...10]	[True,False]		
Best	saga	4	10	True		
KNN						
Param	*n_neighbors*	*weights*	*p*			
Range	[2...200]	[unif, dist]	[1,2]			
Best	9	dist	1			

Table 3: Tested and best parameters - CUP task

3.2.1 Models comparison

In table 4 are shown the performances of the models with the best parameters. The 'VL mean (±std)' value is the result of the cross-validation: such values are the base for the model selection phase. The 'TR' and 'TS' values are computed after the retraining. The 'TS' values are the ones used for the model assessment of the single models.

Model	**MEE**		
	VL mean (±std)	*TR*	*TS*
NN SGD batch	1.050 (±0.031)	0.832	1.047
NN SGD mb (100)	1.090 (±0.022)	1.027	1.088
NN SGD online	1.089 (±0.019)	0.992	0.991
NN ADAM batch	1.041 (±0.017)	0.896	1.014
SVM	1.130 (±0.024)	0.968	1.081
RANDOM FOREST	1.080 (±0.040)	0.191	1.103
LBE	1.234 (±0.044)	1.073	1.232
KNN	1.068 (±0.022)	-	1.057

Table 4: Performances - CUP task

The performances of the various models were compared on the VL. The model with the highest MEE is the LBE. In our opinion, this could be caused by the phi function: we used a polynomial transformation which could not be the best one for this kind of task. The best performances were achieved by the NN with ADAM and batch mode. Similar results are obtained by NN with SGD and batch mode and KNN, whose grid_search take much less time with respect to NN. While SVM can not reach the same generalization capability. We also tried NN in minibatch (with batch size = 100) and online mode: in both cases the computational time for the grid increases a lot (in particular with the online) and the performances are worse with respect to NN with SGD and batch mode. Finally, although Random Forest is an ensemble, it cannot reach the three best models' performances and it is quite computationally slow.

3.2.2 Ensemble models

In order to improve the generalization capability of the single models, we also execute 2 ensembles: a Voting regressor and a Stacking model. We observed that the best result on the cross-validation was reached with only 4 estimators: NN with SGD and batch mode, KNN, NN with ADAM and batch mode and Random Forest. For the Stacking we did two gridsearch to select the best 'alpha' value for the 2 ridge regression model (one for every output). The alpha's values selected are 75 (model of the first output) and 250 (model of the second output).

In table 5 we show the performance on validation evaluated through the cross-validation, while TR and TS MEE are reached by retraining the ensembles on the full TR set.

Comparing the value on the VL with the ones of the others simple estimators, we can notice the improvement of the performance in both the ensembles. In particular the Voting regressor

has the best MEE on VL.

Ensemble Model	**MEE**		
	VL mean (±std)	*TR*	*TS*
Voting regressor	1.024 (±0.008)	0.461	1.006
Stacking	1.025 (±0.013)	0.383	1.011

Table 5: Ensemble performances - CUP task

3.2.3 Final Model For Blind Test

After trying 8 simple estimators and two ensemble, we chose to select the Voting regressor ensemble like the final model to predict the data on the Blind Test. This our choice is based on the value of the MEE on VL. This ensemble takes the mean of NN with SGD and batch mode, KNN, NN with ADAM and batch mode and Random Forest, each of them build with the best parameters reported in table 3 and the characteristics see above. Thanks to the value on 'TS' we predict a generalization capability of this ensemble of 1.006 on the MEE.

3.2.4 Computational time

All the gridsearch were executed on Kaggle platform: in this way we obtained high performance computing regardless of our hardwares. The only inconvenient was notebook execution limit of 12 hours: this problem was easily overcome by the possibility of launching multiple notebooks at the same time per user.

4 Conclusion

Thanks to this work we can say that the best models to predict the Cup dataset is the NN and in particular the one with Adam optimizer. But the use of one ensemble composed by different estimators can improve further the performance and can lead to reach very high generalization capability. In particular, the use of NN with SGD and batch mode, KNN, NN with ADAM and batch mode and Random Forest in the Voting regressor ensemble give the best prediction capabilities. According to our estimate, this ensemble will have a MEE on Blind Test around 1.006.

BLIND TEST RESULTS: Looney_Tuned_ML-CUP21-TS.csv

Acknowledgments

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

References

[1] David H. Wolpert. Stacked generalization. *Neural Networks*, 5(2):241–259, 1992.

Appendix A

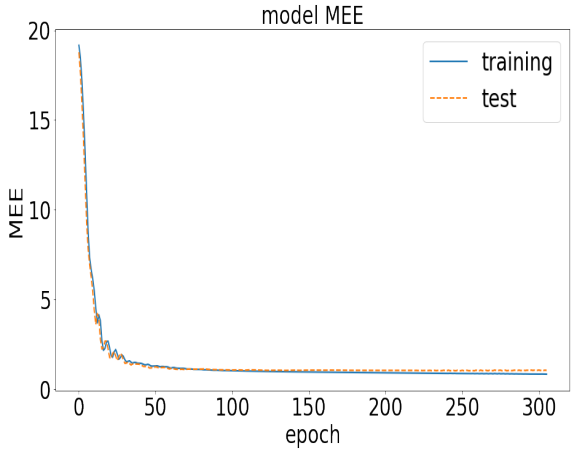

Figure 9: NN with SGD and batch mode

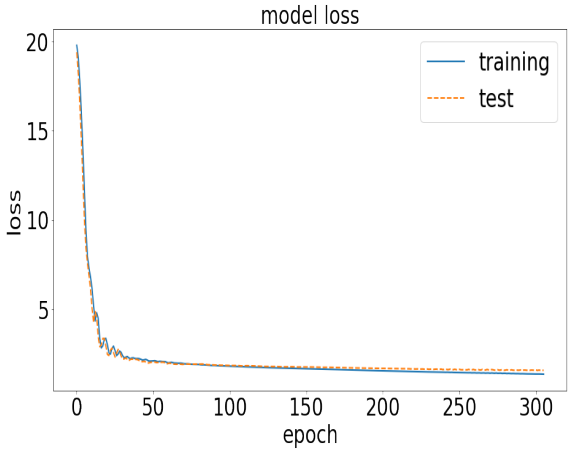

Figure 10: NN with SGD and batch mode

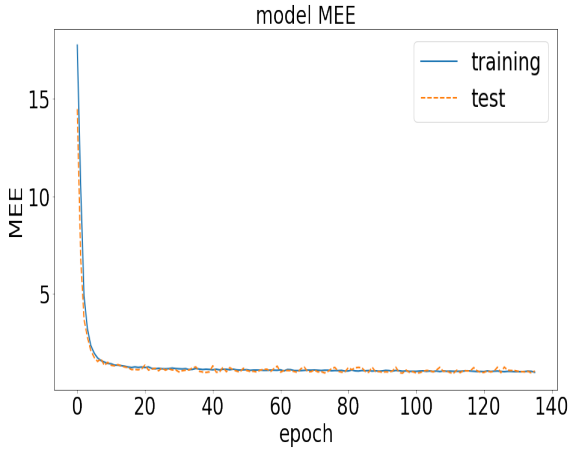

Figure 11: NN with SGD and batch 100

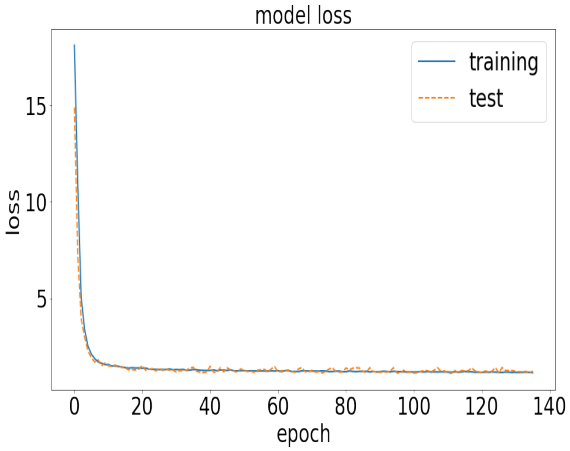

Figure 12: NN with SGD and batch 100

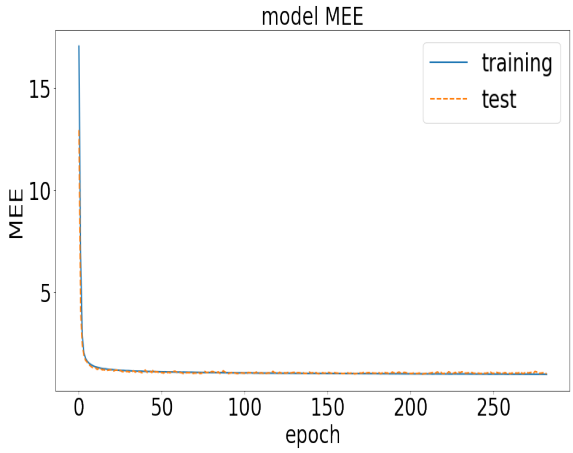

Figure 13: NN with SGD and ONLINE

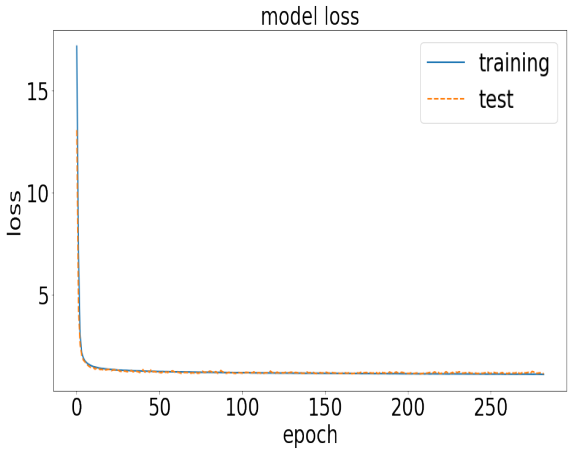

Figure 14: NN with SGD and ONLINE

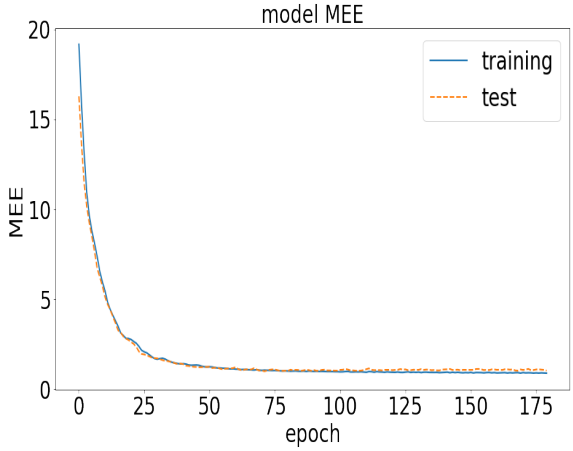

Figure 15: NN with ADAM and batch mode

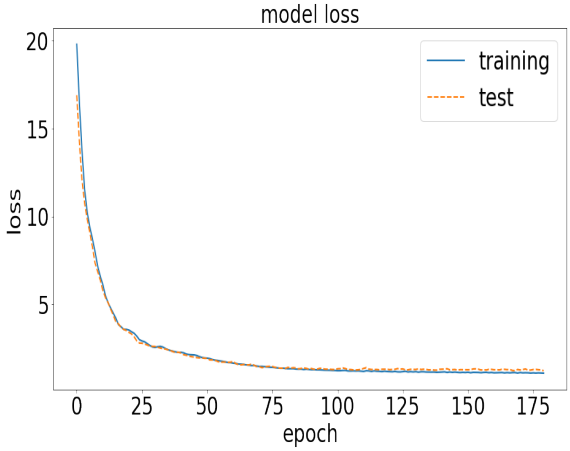

Figure 16: NN with ADAM and batch mode