ML Project Report

Giuliano Galloppi - 646443 - g.galloppi@studenti.unipi.it - MSc in Big Data Technologies

Niko Paterniti Barbino - 638257 - n.paternitibarbino@studenti.unipi.it - MSc in Artificial Intelligence

Ilaria Salogni - 640991 - i.salogni@studenti.unipi.it - MSc in Humanistic Computer Science

ML Course (654AA) 9 CFU - Academic Year: 2022/2023

Date: 23/01/2023

Type of project: **B**

Abstract

In this project we have chosen to evaluate the functionality and performance of various models, the picked one was: Neural Networks, SVM, KNN and Random Forest. In order to make an exhaustive comparison between them we used as validation technique the K-Fold Cross Validation, also using a Grid Search approach. Random Forest model, after the validation of all the models it became the best performing.

1 Introduction

The aim of our course project was to explore the different models and to evaluate their approach towards a classification and a regression task. All the models were tested for both the tasks, and for each task a different dataset was given. Mainly two libraries were used, PyTorch[2] and ScikitLearn[1]. The models were Neural Networks, SVM, KNN and Random Forest. Several strategies for the hyperparameter tuning phase were tested out, as well as the combination of the more significant parameters for each model through cross-validation. The performance of the models was compared using the 'Mean Euclidean Error' on the validation set as a metric. Finally, the best performing model was tested against new data for the CUP competion 2022.

2 Method

The first thing we did was to divide the ML cup set into a design set and a test set. We decided to use 80% of the dataset for the model selection and the remaining 20% for the internal model assessment.
For the monk there were no hold-out performed on the training set, and the search has been carried on keeping the three monk datasets separated, getting three respective scoring. The same train-test split has been used for all models to prevent any bias due to different sample selections. The given datasets did not contain any missing or NaN values, so there was no data cleaning phase other from checking for those.

2.1 NN-Neural Networks

For the implementation of the NN we used Pytorch and scikit-lern for validation and pre-processing. For this model we decided to use Adam as optimizer since it is generally well performing even without properly tuning the hyper-parameters. So, after having tuned the learning rate n, keeping 1=0.9 2=0.09, we noticeably reduced the number of MLP to train which is essential since in general MLP is much slower to train if compared with other models. The training has been performed using minibatches of size 32 and LeakyRelu as activation function. We applied Early Stopping for regularization, meaning that the training stops if it fails to converge after having analyzed 500 epochs or if an improvement has not been found in the last 20 epochs. The number of units in the hidden layers has been kept constant so each architercture is defined as "LXN"

- L: Number of hidden layers
- N: Number of units per hidden layers

2.2 RF-Random Forest

RF is an ensemble method that works by putting together more decision trees at training time. If we have a classification task, the output class is the one selected by most trees. For regression tasks, the output is the mean prediction of the individual trees. Also for RFs, we used the scikitlearn library and we tuned the hyperparameters performing grid searches. With RANDOM FOREST the number of trees in the forest represents an ulterior hyperparameter. Also in this case we used the mean Euclidean distance as a metric and we performed the GridSearchCV class to perform grid search and cross-validation

2.3 K-Neares Neighbors

The model was fully implemented using Scikit Learn library. We've used K-Fold Cross Validation technique, following GridSearch approach to look for the best combination of parameters, evaluating: values of n-neighbors, two different scalers, weights, metric and p. We used a custom scoring function, based on the MEE, to evaluate the performance of the pipeline on the training data.

Although hyperparameter tuning is a basic part of the design of all the machine learning models, it is crucial for KNN, where there is no actual "learning" carried on, but something more similar to a majority voting.

The numbers of neighbors had been experimented first by trying values in large ranges, like the multiples of five from 0 to 50, and then squashing the range to explore the values in the proximity of the more promising ones. In example, if in a first session the best value was chosen between 1, 5, 10, 15, 20 and so on up to 50, in a latter moment, when the 20 was picked as the best value by the GridSearch, we repeated the search with a range of values that lay around 20 (so 18, 19 ... 22, 23 an so on). The best combination (best params attribute of the GridSearchCV) is then used for a model that is fitted using the training data.

2.4 Support Vector Machine (SVM)

The intuition behind the SVM training was to use a 'MultiOutputRegressor' that has a dedicated strategy in which fit one regressor per output target. Since we had to perform a 'regression task' we have used the SVR model, it and the MultiOutputRegressor are provided by scikit-learn [1] library. The common hyperparameters that were tuned in the model selection process are 'C' the regularization coefficient and the 'epsilon'- tube. Instead for each different kernel:

- 'rbf' : gamma
- 'poly': coefficient, degree
- 'sigmoid': gamma, coefficient

We used the `GridSearchCV` taking as estimator/model the `MultiOutputRegressor` performing the grid search and the cross-validation for each model, obtain respectively the best estimator and scores as results.

The scoring method used with all models was the *mean ecludian error*:

$$E_{MEE} = \frac{1}{l} \sum_{p=1}^{l} ||o_p - t_p||$$

where *l=number of data, p=pattern, o=output, t=target.*

3 Experiments

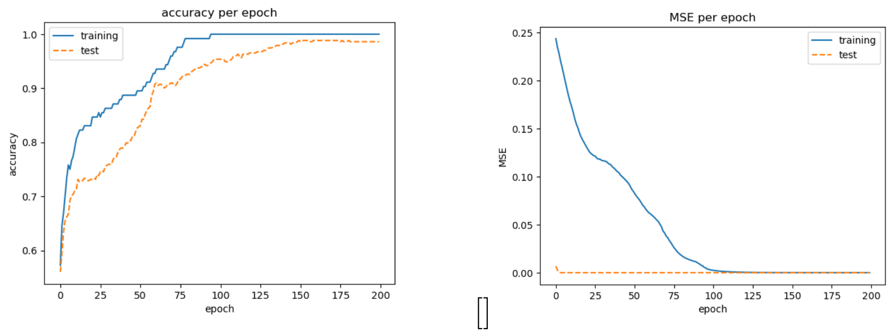
"The best values of the hyperparameters were obtained for each model through a model selection phase, using the lowest Mean Euclidean Error (MEE) validation score as the criteria. The model with the best validation results was chosen and then evaluated on an internal test set in the model assessment phase. This section presents the results of the experiments conducted with various models."

3.1 Monk Results

In this section we report the summarized table and plots of the neural network model trained using PyTorch library to classify the three monk datasets.

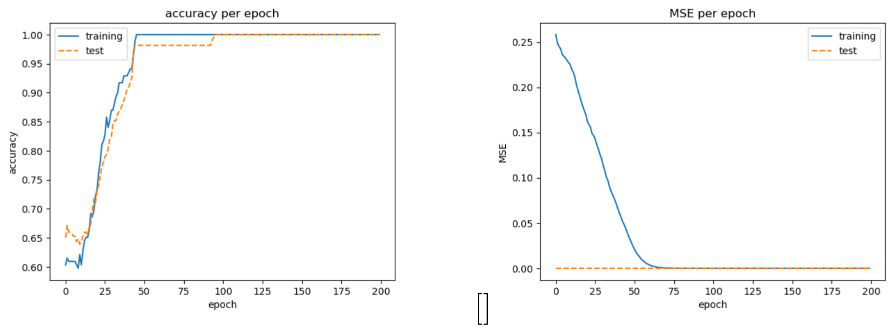
The hyperparameters were found performing a grid search; in Monk1 and Monk2 the hyperparameters taken into account were: the number of units of the hidden layer, the momentum coefficient and the learning rate, while in Monk3 we added also L2 regularization to the loss, therefore we used the grid search also to search the best coefficient for L2 regularization. To train the models we used again the mean squared error as loss. Table 1 reports relevant features regarding the final model built for each monk dataset using PyTorch, figure 1 shows the learning curves for each final model created for the monk datasets using PyTorch.

Task	**Model**	**MSE (TR/TS)**	**Accuracy (TR/TS)(%)**
MONK1	NN	0.959 / 0.916	100% / 99%
MONK2	NN	0.242/0.006	100% / 100%
MONK3	NN	0.409 / 0.033	94% / 97%
MONK3+reg.	NN	-	-



[] []

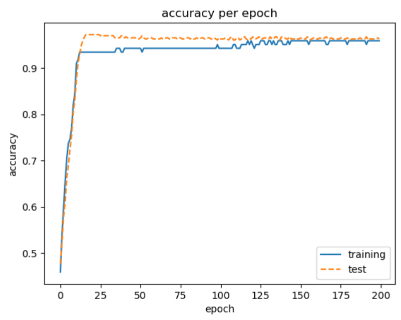
Figure 1: monk1 metrics



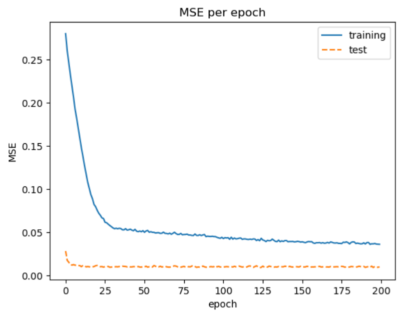
[] []

Figure 2: monk2 metrics

In the project source also the other model were used to classify the monk datasets.



[]



[]

Figure 3: monk3 metrics

3.2 CUP Results

3.2.1 K-Nearest Neighbors

As we described early, also for K-Nearest Regression (KNR) we performed a grid-search to explore the best parameters. Where the parameters seemed more promising, we expanded the research (for example in the environs of the k of neighbors that seemed to be more promising). For the regression task, the target values were stored in a two-column matrix, and we decided not to split them. As a result, in the scatterplot we arranged the first column values on the x-axis, and the second column on the y-axis. We did this both from our real values, from the original cup dataset (in blue), and for our predicted (red) values

3.2.2 SVM

For the SVM regression we have used three kernels: The RBF kernel is defined as $k(x_i, x_j) = e^{\gamma|x_i - x_j|^2}$, Polynomial as $(a \times b + r)^d$ and Sigmoid as $k(x, y) = \tanh(\gamma \cdot x^T y + r)$.

For our regression task that involves two output target, as said before, we have used the class `MultioutputRegressor` that bring as parameter the `SVR` model adapting it for the multioutput regression. We have setted the `GridSerachCV` class with these estimators Differentiating executions for each kernel, which accepts different parameters based on this. We have investigate how each combination of these parameters give us the best hyperparameters evaluating their performance with the MEE score. In the following table are reported the metrics with the best tuned parameters achieved:

kernel	**C**	**EPS**	γ	**COE**	**DEG**	**MEE_Val**	**MEE_Tr**	**MSE_Tr**
rbf	10	0.001	auto	-	-	1.465	1.167	1.162
poly	0.1	0.01	-	1.5	5	1.507	1.246	1.212
sig	0.1	0.001	auto	2.0	-	4.428	4.514	3.563

3.3 NN-Neural Network

Each network architecture is tested with three values of $2.00E-02, 1.00E-02, 1.00E-03$, with 1.00E-02 proving to be optimal in almost all cases. For bigger values of L, we have chosen to explore smaller values of N. In fact, if we kept N fixed as we increase L, we would have reached a number of hidden units disproportionate to the number of training samples models, which makes the model prone to overfitting (aside from taking unfeasible long times to train). This is supported by the results we have obtained, as the optimal number for N becomes smaller as L increases.

Table 1: Mean MEE Val. Error for different MLP Architectures and learning rates

LR	**2x25**	**2x50**	**2x100**	**3x15**	**3x25**	**3x50**	**3x100**
2,00E-2	1.02718	0.92256	0.97069	0.75256	0.87856	0.99453	1.03842
1,00E-2	0.89463	0.88884	1.03394	0.89143	0.87737	0.93827	0.88460
1,00E-3	1.03642	1.05011	0.97772	0.94495	1.04055	0.85946	0.83542

LR	**4x10**	**4x15**	**4x25**	**4x50**	**5x10**	**5x15**
2,00E-2	0.95294	0.92968	0.86932	0.92866	0.88129	0.96781
1,00E-2	0.95388	0.94786	0.92623	0.90285	0.88157	0.91467
1,00E-3	0.96863	0.85917	0.93753	0.93977	0.97671	0.93988

LR	**5x25**	**5x50**	**6x10**	**6x15**	**6x25**	**6x50**
2,00E-2	0.91141	0.90605	1.13389	1.02312	1.08312	0.98643
1,00E-2	0.90085	0.87043	0.98635	0.95813	0.93584	0.91485
1,00E-3	0.89477	0.85061	1.06600	0.95045	0.83970	0.91879

3.4 RF-Random Forest

Random forests or random decision forests is an ensemble learning method for classification or regression that acts by constructing a multitude of decision trees at training time. In the model selection phase it was passed to a `GridSearchCV` an object `RandomForestRegressor` (also provided by scikit-learn) that follows the same procedure such the other models. In RF there we haven't normalized the data because with it we don't needed. For what it concerns the number of estimators, we wanted to use a number in the range $[1000, 5000]$ but due to high computational time we decided to test the Random Forest using a number in the range $[10, 50]$. After a tuning of the parameters, we reach a best score of MEE on the validation set of $0,55$ almost, proclaiming this model the chosen one for the model selection phase wrt the others.

4 Conclusion

Random Forest model performed the best on the validation set, according to the MEE metric and was therefore selected for further training and evaluation on the test set. However, it is important to note that MEE may not be the only metric to consider when evaluating the performance of a model: using other metrics could also lead to a different assessment. Overall working in this project has been very useful and stimulating for all of us since we have been forced to put the concepts studied during the course putting them into practice with the models described above by deepening.

References

[1] Lars Buitinck, Gilles Louppe, Mathieu Blondel, Fabian Pedregosa, Andreas Mueller, Olivier Grisel, Vlad Niculae, Peter Prettenhofer, Alexandre Gramfort, Jaques Grobler, Robert Layton, Jake VanderPlas, Arnaud Joly, Brian Holt, and Gaël Varoquaux. API design for machine learning software: experiences from the scikit-learn project. pages 108–122, 2013.

[2] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Z. Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. *CoRR*, abs/1912.01703, 2019.