**Machine Learning Project Report**

*By Dennis Bettels*

[*dbettels1@gmail.com*](mailto:dbettels1@gmail.com)

ML course (654AA), Academic Year:2019/20

Date:

Type of project: B

Abstract

An extensive analysis of various ML tools such as keras and scikit-learn applied to a variety of different models such as Neural Networks, K-Nearest Neighbour and Support Vector Models, along with a fine tuning of hyperparameters using Gridsearch to best fit the models to a dataset describing the Adult Dataset[[1]](#footnote-1) predicting whether an individual earns over 50K a year based on 14 attributes. Additionally, a classification is run on Monk’s Problems Dataset to ascertain the quality of the models, and a regression is run on the official ML-CUP dataset for the competition using all models, selecting the best one.

Introduction

The purpose of this assignment was to evaluate a set of popular tools which are used to make Neural networks, and compare their efficacy in terms of precision, cross-validation score and loss. The tools which were examined for this research were Scikit-learn, Keras and Tensorflow. To evaluate these tools, we used them to create and manipulate a range of Neural Networks: Multi-Layered Perceptrons, Support Vector Models and K-Nearest Neighbour. These models were trained on 2 datasets: one a binary classification predicting whether the income of an individual exceeds 50K based on a variety of factors and another on MONKS’s problem dataset. The latter was done with the intention of testing how solid our machine learning algorithms were, as the Monk’s dataset is quite useful to determine classification networks quality and tends to show high improvements when calibrated properly.

In order to complete the assignment, some assumptions had to be made. For the MLP, we had to keep only one hidden layer to avoid adding complexity to the model.

Additionally, the Support Vector Regressor was too slow with the ‘poly’ and ‘linear’ kernel to include those two in a Gridsearch. A basic implementation of randomized search was done towards the end to address this issue.

Method

The methodology used for the project was as follows: the MLP was made on keras which runs on top of Tenserflow as well as Scikit-learn, and the KNN and SVM (as well as their regression versions (KNR) and SVR)were made with the respective Scikit-learn packages. To import and carry out the pre-processing of the data both numpy and pandas were used across all models and tools. In order to the plot the graphs, matplotlib was used.

While all the models were created separately, the evaluation methods were similar. Once implemented, the hyperparameters were fine-tuned using gridsearch to return the most optimal results.

For each model, the hyperparameter which were tuned were as follows:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| SVM | Kernel  ('rbf','sigmoid', ‘poly’,’linear’)  Specifies the type of kernel to be used with the algorithm  Note:poly and linear not used for regressor | Regularization parameter C | Gamma coefficient  Λ  Kernel coefficient for rbf, poly and sigmoid kernel types | Decision Function Shape  OVO, OVR  One vs rest (ovr) decision function or one v one. Ignored in binary classification | Shrinking Heuristic  True,False  Whether to use the shrinking heuristic | Tol  Tolerance for stopping criterion | Coef0  Independent term in kernel function. Only used in poly and sigmoid |
| KNN | n-neighbours  number of neighbours to be used | Weight Function  ('uniform', 'distance')  Weight function. Either uniform weights will be used or weight points by inverse of their distance | Algorithm  ('ball\_tree', 'kd\_tree', 'brute')  Type of algorithm to compute nearest neighbours | Leaf size  Leaf size passed to ball tree and kd\_tree | Power Parameter  P | Metric  (minkowski, Euclidean, manhattan, chebyshev)  Distance metric to calculate the tree |  |
| MLP | Alpha  [0.025,0.05,0.1,0.2,0.4,0.7]  To remove | Hidden Layer Size  [8],[14],[16],[24],[28] | Maximum iteration  [64, 96, 112, 128, 160, 256] | Destination function  [‘relu’,  ’tanh’,  ’sigmoid’] | Momentum  [0.1,0.5,0.7,0.8,0.9] | Nesterov’s momentum  [True,False] | Learning rate initiation  [0.001, 0.01, 0.05, 0.1, 0.15,0.2 |
|  | Dropout  [0,0.2,0.5] | Decay  [0,0.01,  0.0001,  0.000005] |  |  |  |  |  |

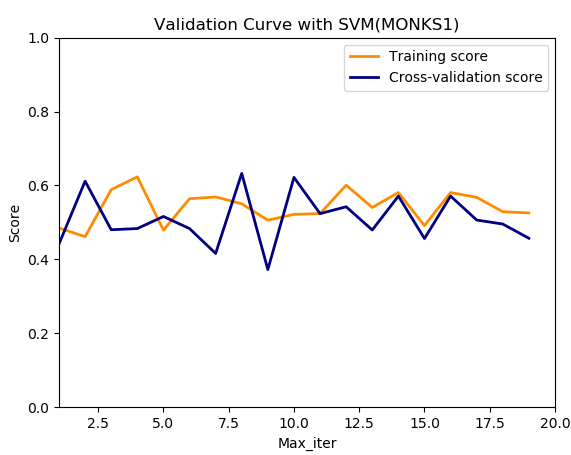
The primary evaluation method which was used was the final accuracy of the model and the mean squared error. This was done with both the Adult dataset and MONK dataset.

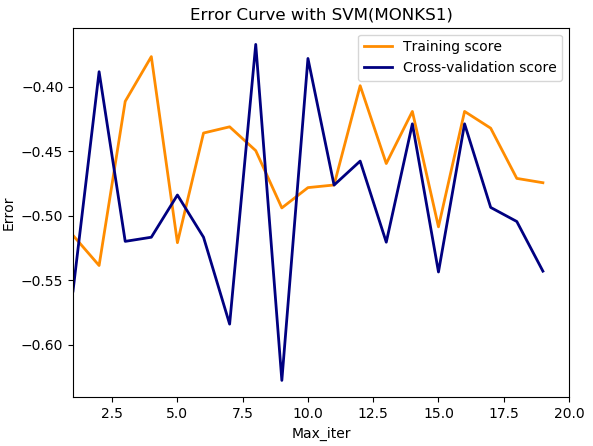
The gridsearch was performed by running every available value where fixed values were required. For numeric values, orders of magnitude were used at first and gradually centred on the optimal values over multiple gridsearches. Occasionally, for the sake of simplicity, dead-weight parameters (such as coef0 for rbf) were used, which increased computational time. However, this had no adverse effect on the final results. Various graphs were plotted in **Section 3** covering the effect that hyperparameters had on the final accuracy score of the model.

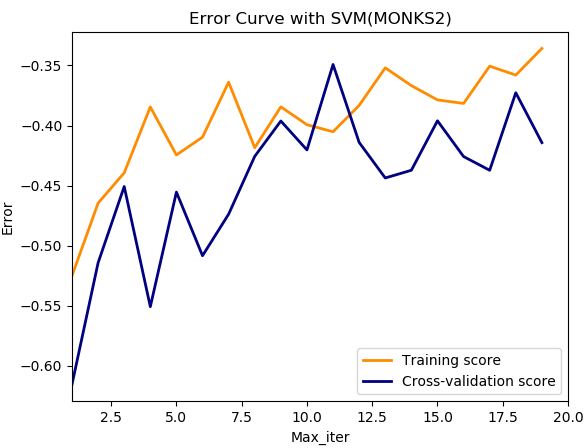
The gridsearch for the multilayer perceptron was performed in Scikit-learn. To test the best chosen hyperparameters, the classifier of Scikit-learn was used (MLPClassifier). The model was then translated into a sequential Keras model and we applied an ulterior hyperparameter fine-tuning not present on the Scikit-learn classifier, which was the dropout and decay. Both of these values were set to 0.0. Finally, to plot the results for the MLP the learning curve was utilised in Scikit-Learn.

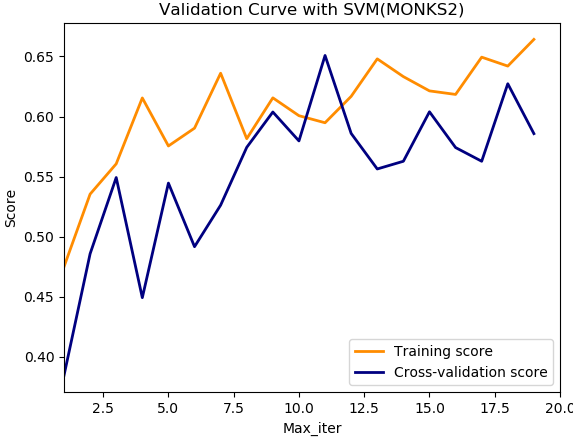
**3. Experiments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Task** | **Hyperparameters** | **MSE** | **Accuracy(TR/TS)** |
| Monk 1(MLP) | activation = 'tanh'  learn\_rate = 0.2  neurons = 8  dropout = 0.0  momentum = 0.7  nesterov = True  epochs = 1024  batch\_size = 64  alpha= 0.001 | TR: 0.04251  TS: 0.05389 | TR: 100%  TS: 98.8% |
| Monk 1 (SVM) | {'C': 810, 'coef0': 0, 'decision\_function\_shape': 'ovo', 'gamma': 0.005, 'kernel': 'rbf', 'shrinking': True, 'tol': 1e-06} | TR: 0.0  TS: 0.07889 | TR: 100%  TS: 92.11% |
| Monk 1 (KNN) | {'algorithm': 'brute', 'leaf\_size': 7, 'metric': 'chebyshev', 'n\_neighbors': 28, 'p': 1, 'weights': 'distance'} | TR: 0.0  TS: 0.19954 | TR:100%  TS:80.05% |
| Monk 2(MLP) | activation = 'tanh'  learn\_rate = 0.2  neurons = 8  dropout = 0.0  momentum = 0.7  nesterov = True  epochs = 1024  batch\_size = 256  alpha= 0.001 | TR: 0.18411  TS: 0.20616 | TR: 76.9%  TS:71.3% |
| Monk 2 (SVM) | {'C': 1000, 'coef0': 0, 'decision\_function\_shape': 'ovo', 'gamma': 'auto', 'kernel': 'rbf', 'shrinking': True, 'tol': 0.0001} | TR:0.0  TS: 0.17865 | TR:100%  TS:82.13% |
| Monk 2 (KNN) | {'algorithm': 'brute', 'leaf\_size': 30, 'metric': ' mahalanobis ', 'n\_neighbors': 1, 'p': 1, 'weights': 'uniform'} | TR: 0.0  TS: 0.17865 | TR:100%  TS:82.13% |
| Monk 3(MLP) | activation = 'tanh'  learn\_rate = 0.2  neurons = 13  dropout = 0.0  momentum = 0.7  nesterov = True  epochs = 1024  batch\_size = 128  alpha= 0.001 | TR: 0.06792  TS: 0.08541 | TR:95%  TS:93% |
| Monk 3 (SVM) | {'C': 10, 'coef0': 1, 'decision\_function\_shape': 'ovo', 'gamma': 'auto', 'kernel': 'poly', 'shrinking': True, 'tol': 0.0001,’class\_weight’:’balanced’} | TR: 0.00826  TS: 0.05568 | TR:99.17%  TS:94.43% |
| Monk 3 (KNN) | {'algorithm': 'auto', 'leaf\_size': 60, 'metric': 'manhattan', 'n\_neighbors': 30, 'p': 2, 'weights': 'distance'} | TR: 0.0  TS: 0.07657 | TR:100%  TS:92.34% |
| Adult (SVM) | {'C': 10, 'coef0': 0, 'decision\_function\_shape': 'ovo', 'gamma': 1e-05, 'kernel': 'rbf', 'shrinking': True, 'tol': 0.0001} | TR: 0.23127  TS: 0.17532 | TR: 78%  TS:72% |
| Adult (KNN) | {'algorithm': 'auto', 'leaf\_size': 1, 'metric': 'manhattan', 'n\_neighbors': 18, 'p': 1, 'weights': 'uniform'} | TR: 0.23836  TS: 0.22511 | TR: 76%  TS: 77% |
| Adult (MLP) | activation = 'tanh'  learn\_rate = 0.15  neurons = 16  dropout = 0.0  momentum = 0.7  nesterov = True  epochs = 512  batch\_size = 307  alpha = 0.001 | TR: 0.15554  TS: 0.16274 | TR:78%  TS:76% |

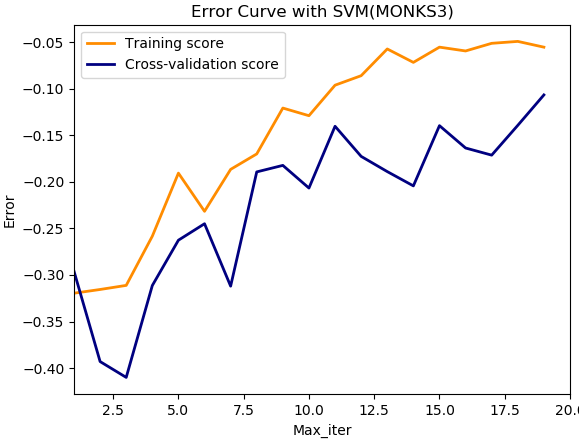
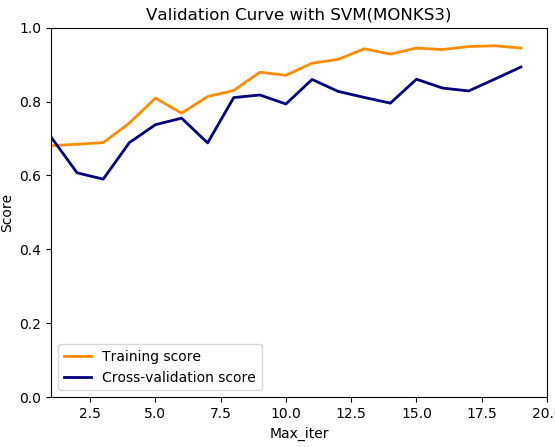
**Figure 1.** Plot of accuracy and MSE of the Monk’s 1 on the SVM



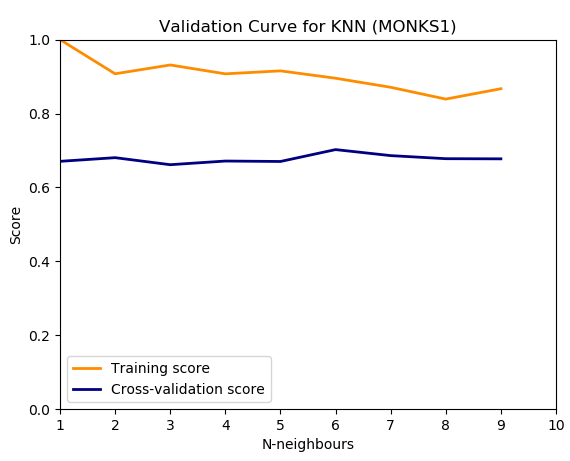
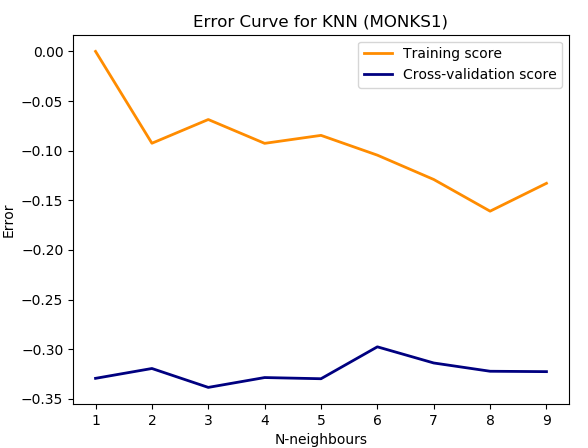
**Figure 2.**  Plot of accuracy and MSE of the Monk’s 2 on the SVM



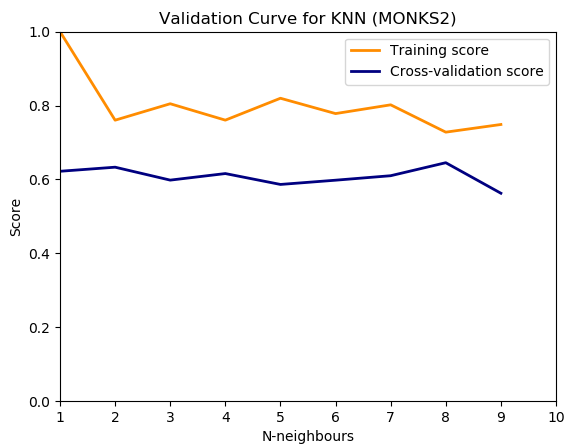
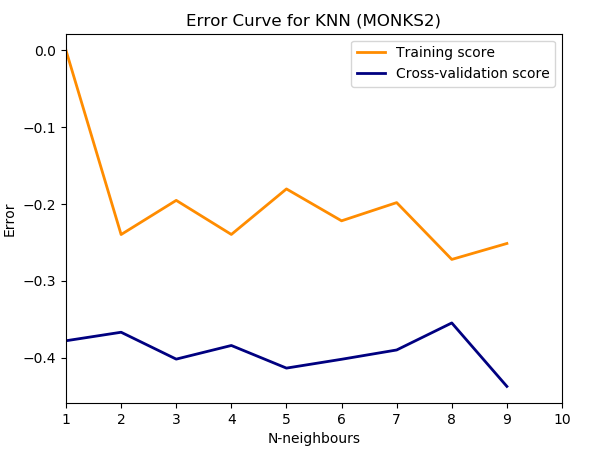
**Figure 3.** Plot of accuracy and MSE of the Monk’s 3 on the SVM

****

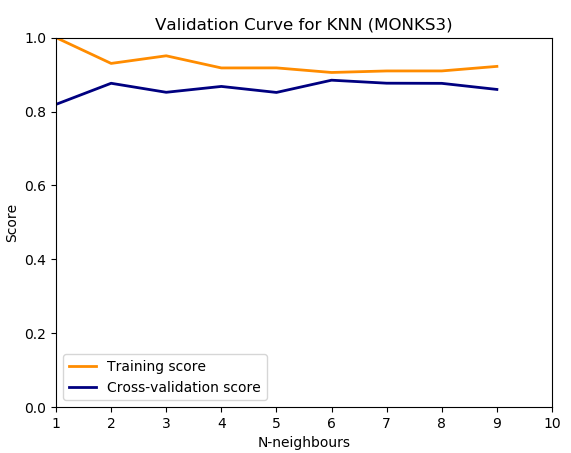
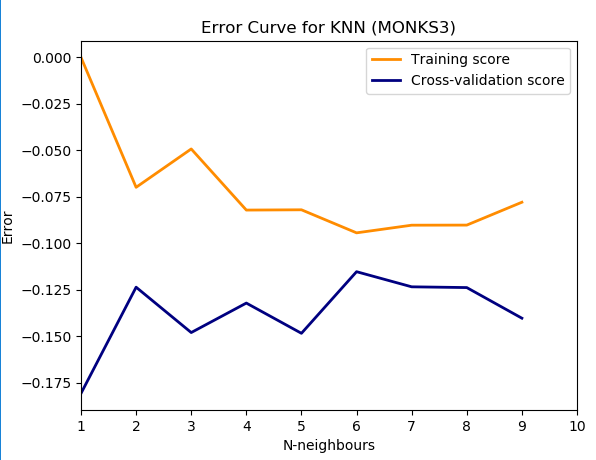
**Figure 4 .** Plot of accuracy and MSE of the Monk’s 1 on the KNN

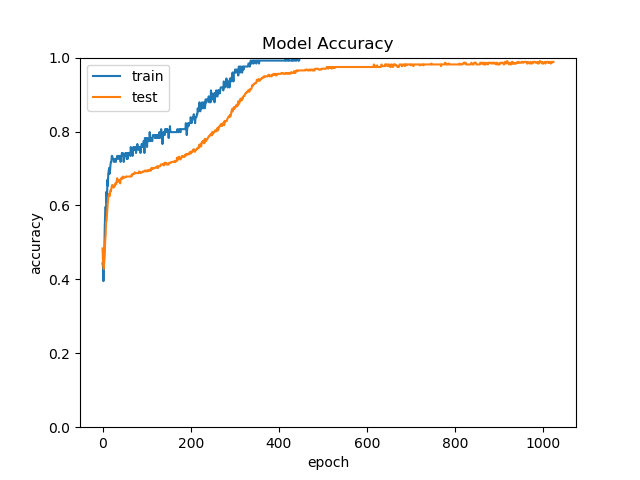
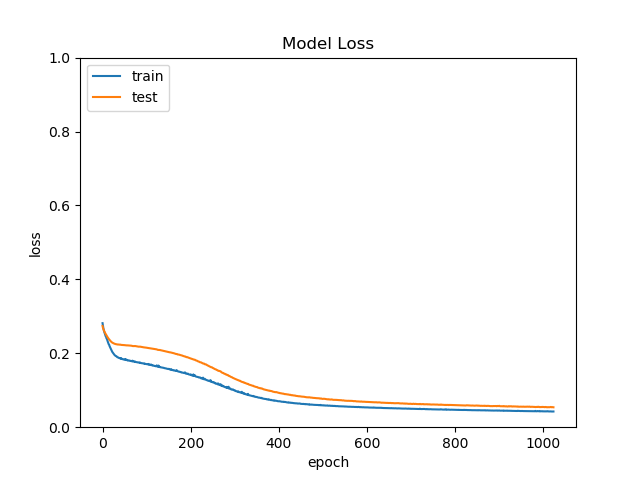
****

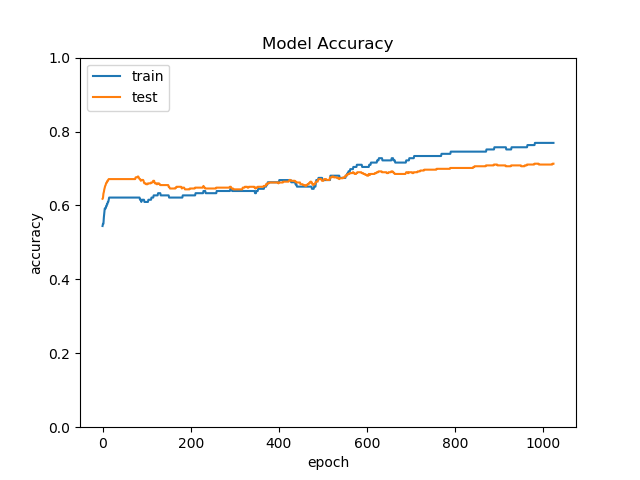
**Figure 5.** Plot of accuracy and MSE of the Monk’s 2 on the KNN

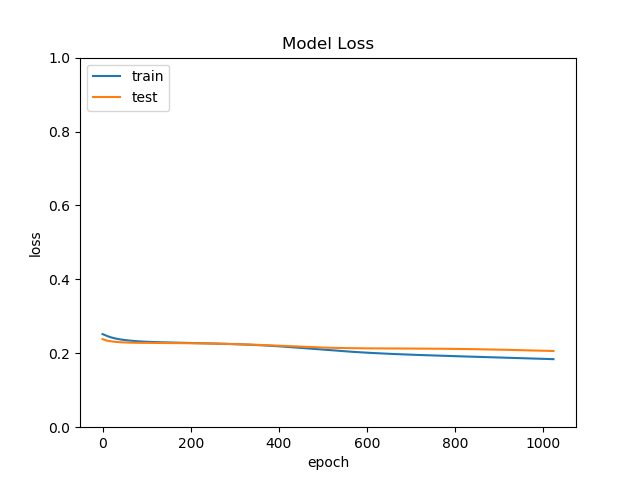


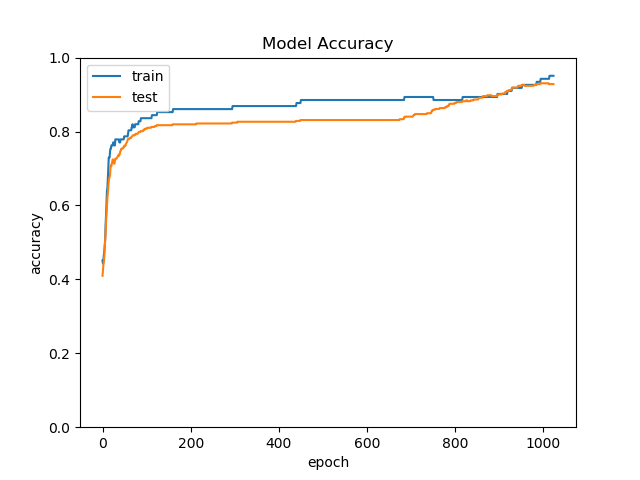
**Figure 6.** Plot of accuracy and MSE of the Monk’s 3 on the KNN

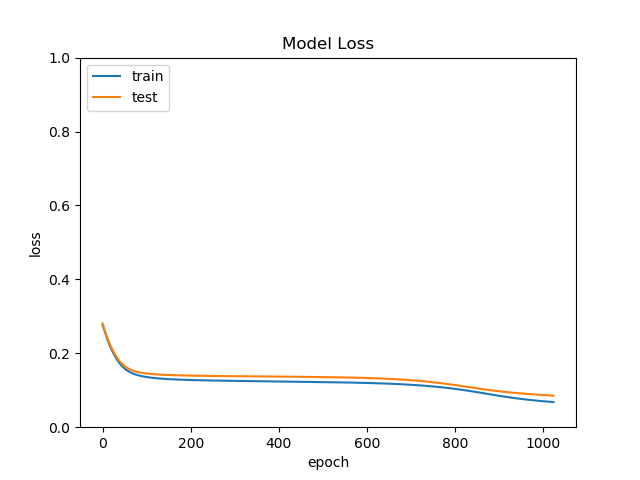


**Figure 7** Plot of accuracy and Loss of the Monk’s 1 on the MLP

**Figure 8** Plot of accuracy and Loss of the Monk’s 2 on the MLP

****

**Figure 9** Plot of accuracy and Loss of the Monk’s 3 on the MLP



**CUP Results**

***MLP Regression***

Before, GridSearch (kfold=5) to choose the model:

First Run:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Hidden Layer Size | Alpha | Batch Size | Learning rate init | momentum |
| [8],[16],[24],[32] | [0.0, 0.00001, 0.001] | [32,128,512] | [0.01,0.1] | [0.7,0.9] |

The best hyperparameters that were found for this were alpha = 0.001, batch size = 32, hidden layer sizes = 512, learning rate init = 0.01 and momentum = 0.7

Second Run:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Hidden Layer Size | Alpha | Batch Size | Learning rate init | momentum |
| [14],[16],[18],[20] | [0.0005, 0.001, 0.0015] | [256,512,809] | [0.001,0.01] | [0.7] |

The best hyperparameters that were found for this were alpha = 0.0005, batch size = 809, hidden layer sizes = 14, learning rate init = 0.01 and momentum = 0.7

Third Run:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Hidden Layer Size | Alpha | Batch Size | Learning rate init | momentum |
| [14],[15],[16] | [0.0, 0.0001, 0.0005] | [809] | [0.01] | [0.7] |

The best hyperparameters that were found for this were alpha = 0.0001, batch size = 809, hidden layer sizes = 14, learning rate init = 0.01 and momentum = 0.7

The final value was calculated using the third and final model and resulted in a training loss of 2.9683 and a testing loss of 3.5121.

**Support Vector Regression**

The SVR was run similarly to the method used in the previous section for the binary classification. A simple gridsearch was run using the same parameters and methodology, with the difference that SVR does not support multiply outputs, and therefore the hyperparameters were tuned differently for the two columns (referred to as column x and column y). The testing for this model was obtained by using 20% of the training dataset as a test set, so it’s possible the final results (with the entire dataset)are even better. The measures used were sklearn’s explained\_variance\_score (E.V.S) function and mean\_squared\_error (M.S.E) function (E.V.S is the desired metric for this section, M.S.E is mainly here for comparison with the previous section).

Column X- shrinking=True, kernel='rbf',C=15,epsilon=0.1,gamma=0.098,tol=0.07)

Training E.V.S - 0.9915876813764759

Testing E.V.S - 0.9884323625581125

Training M.S.E - 0.5475270726888952

Testing M.S.E - 0.7478276153325918

Column Y - (coef0=0,kernel='rbf',shrinking=True,C=8,tol=0.003,epsilon=0.29,gamma=0.23)

Training E.V.S - 0.9817062130384528

Testing E.V.S - 0.9530609681504781

Training M.S.E - 0.32628611908072835

Testing M.S.E - 0.8065032501288617

**K-Nearest Neighbours Regression**

Methodology, hyperparameter tuning and testing methodology were identical to the ones laid out in the previous section. The following results were obtained:

Column X - (algorithm='auto',metric='manhattan',weights='distance',n\_neighbors=15,p=2)

Training E.V.S - 1.0

Testing E.V.S - 0.9916695548331144

Training M.S.E - 0.0

Testing M.S.E - 0.5422168843319535

Column Y - (algorithm='auto', metric='manhattan', weights='distance',p=2,n\_neighbors=7)

Training E.V.S - 1.0

Testing E.V.S - 0.9649924689685742

Training M.S.E - 0.0

Testing M.S.E - 0.6051727370272808

These obtained results indicate that KNR is our best model to perform the regression, and was therefore used for the final prediction of the ML-CUP dataset.

**Conclusion**

Over the course of our research, we attempted to use various models and tools to fit to the Adult dataset. This was a binary classification problem(<50k and >50k were set to 0 and 1 respectively) and our models were trained to correctly predict whether, based on certain criteria, a person would earn more or less than 50k annually using data it had not seen before in a test set. We found out that the most efficient model we had at our disposal was the blablabla. To test the general efficacy of our models, we also tried a series of tests on the MONK’s dataset, and concluded with testing the K-Nearest Neighbours Regression on the ML Cup dataset, resulting in a final Explained Variance Score of 0.99 and 0.96 for columns x and y.

TODO: New graphs, Student dataset, quick randomized search for KNR. Say you found hyperparams with aid of randomized gridsearch. Finish Student Dataset with other models

**ACKNOWLEDGEMENTS**

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

**References**

[1]: A. Micheli: Lecture slides for Machine Learning 2018/2019

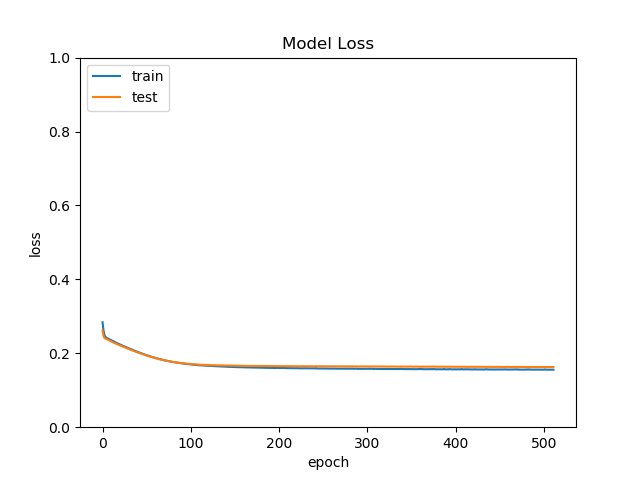
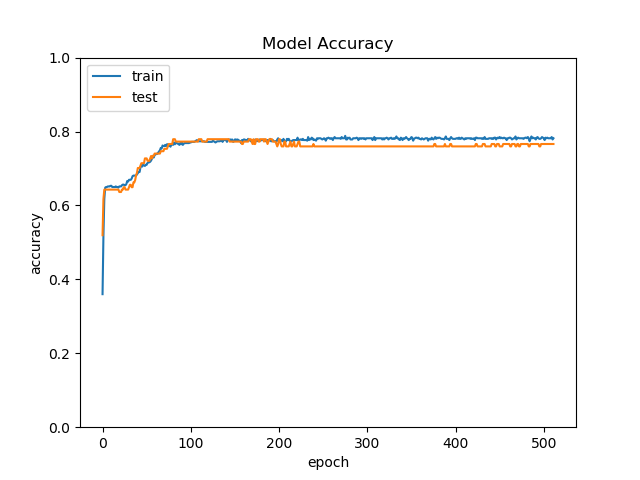
[2]: <https://www.andreagrandi.it/2018/04/14/machine-learning-pima-indians-diabetes/>

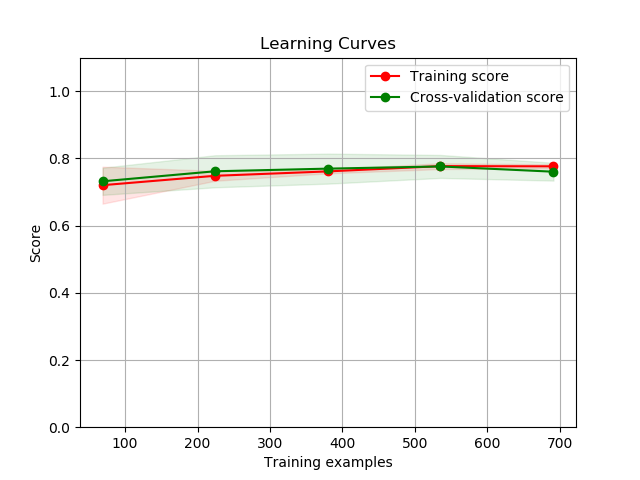
[3]: Keras: https://keras.io/

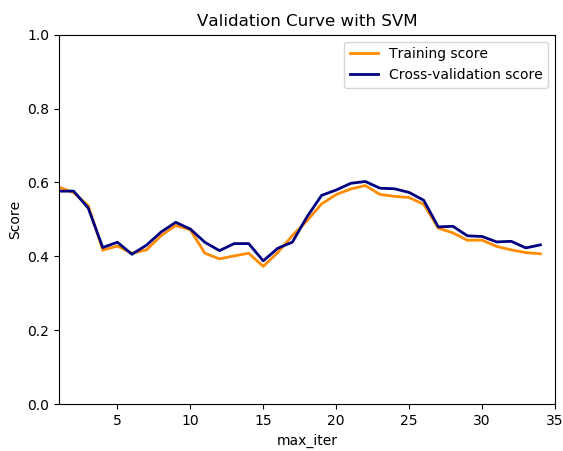
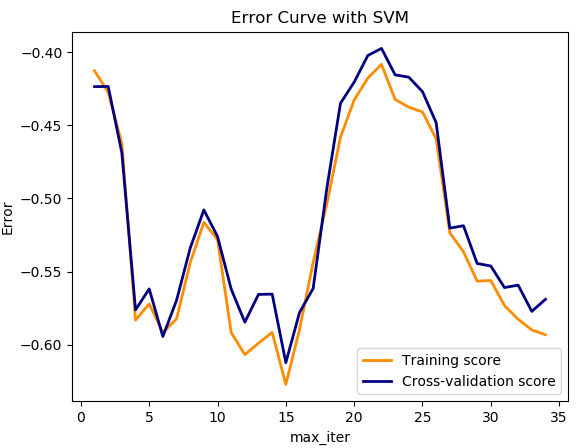
[4]: Scikit-Learn: https://scikit-learn.org/stable/user\_guide.html

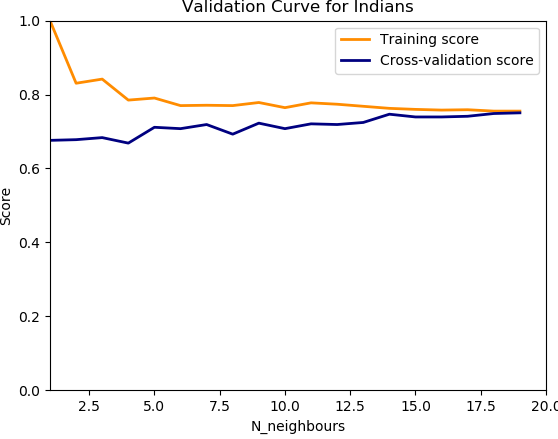
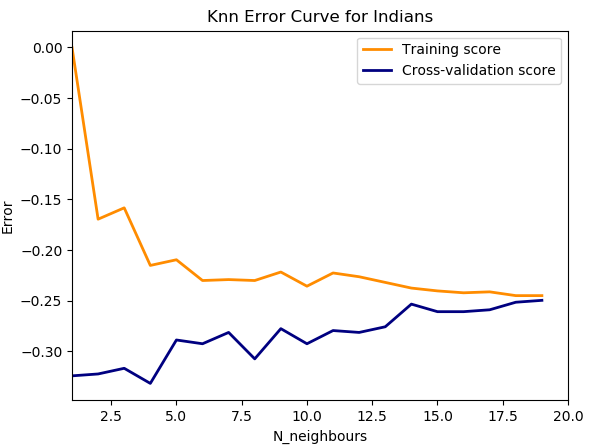
**Appendix**

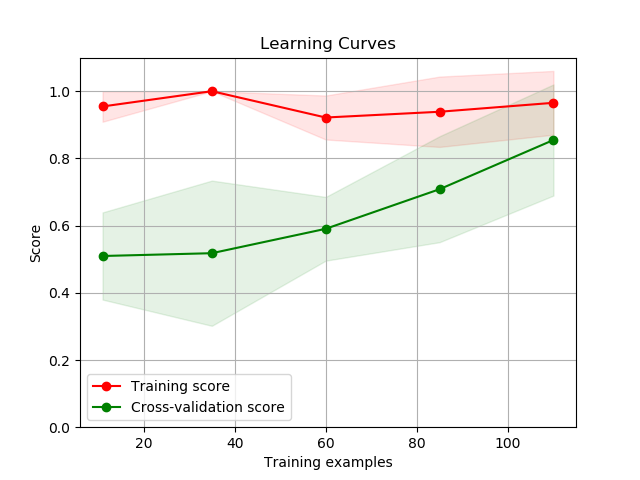
Figure 10. Accuracy and Error of Indians dataset on MLP with Keras

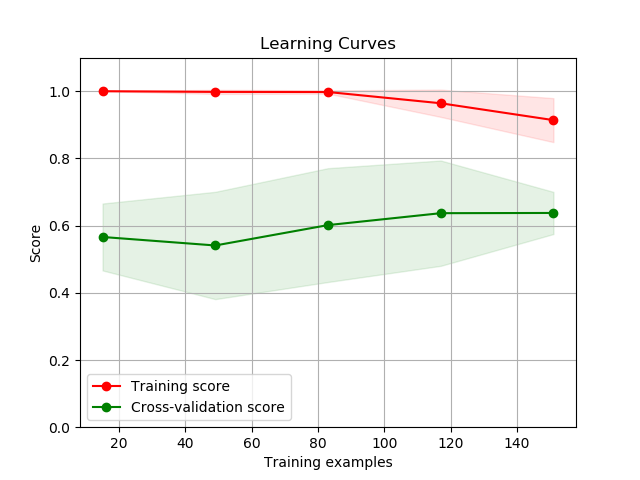
****

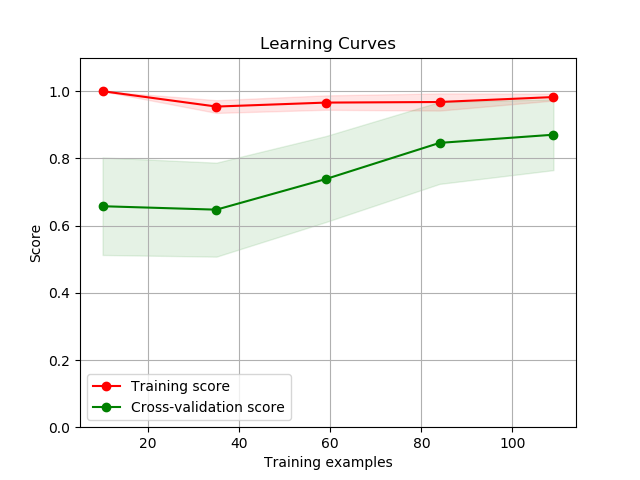
Figure 11 Accuracy and Error of Indians dataset on MLP with Scikit-learn

Figure 12. SVM on the Indians dataset with accuracy and Error(Scikit)

Figure 13. KNN on the Indians dataset with accuracy error (Scikit)

Figure 14 and 15. Learning Curve for Monks1 and 2 MLP(Scikit)



Figure 16. Learning Curve for Monks3 MLP(Scikit)

1. https://archive.ics.uci.edu/ml/datasets/Adult [↑](#footnote-ref-1)