Project 2: Classification and Regression: from linear and logistic regression to neural networks

A. Mattiroli[1], A. Pera[1], and J. Štorek[2]

[1]University of Milano Bicocca
[2]Czech Technical University in Prague

November 2019

Abstract

This report aims to provide insights into the performance of different methods on both categorical and linear data. Two datasets have been studied regarding classification for categorical data and the Franke function's data has been used for linear regression. Algorithms such as gradient descent – both simple and stochastic – and neural networks have been applied and compared to scikit-learn results. The obtained conclusions agree with the ones from scikit-learn, whose algorithms generally perform better. Out of all studied methods, neural network emerges to be the most adequate predictive technique for both classification and regression problems.

1 Introduction

The main purpose of the project is to explore classification, as prediction of qualitative responses through logistic regression, and to re-evaluate linear regression problems, already addressed in project 1, by introducing multi-layer neural networks.

In particular, in the first part of the analysis, logistic regression is implemented by using a gradient descent algorithm on two different data sets: a classification case, in which the default payment is predicted, and the so-called Wisconsin Cancer data, a scikit-learn's dataset consisting of a binary classification problem of benign or malignant tumors.

A neural network algorithm has been applied on the same two datasets in order to find the optimal weights and biases and to analyze which combination of regularization parameters and learning rates leads to the best performance measure.

In the second part of the study, the neural network algorithm is adapted to a linear regression problem. For this part, Franke function from project 1 [2] has been used and results have been compared with the ones obtained previously.

A background theory of the used techniques can be found in the methods section. A description of the implemented code for the study is contained in the implementation section. The analysis of results can be read in the results and discussion section.

2 Methods

2.1 Logistic regression

Logistic regression is a widely-used classifier which allows to predict qualitative responses. In particular, logistic regression models the probability of Y to fall into a particular category rather than modelling directly the response itself.

The probability that a data point belongs to a category is given by the Sigmoid or logit function

$$p(t) = \frac{1}{1+\exp(-t)} = \frac{\exp t}{1+\exp t} \quad , \tag{1}$$

which represents the likelihood of an event.

Logistic regression models are fit by applying the Maximum Likelihood Estimation (MLE) criterion. In a binary case with independent data points, the likelihood expression is

$$P(D|\vec{\beta}) = \prod_{i=1}^{n} [p(y_i = 1|x_i, \vec{\beta})]^{y_i} [1 - p(y_i = 1|x_i, \vec{\beta})]^{1-y_i} \tag{2}$$

from which it is possible to obtain the negative log-likelihood and the cost function, the so-called Cross Entropy, as

$$C(\vec{\beta}) = -\sum_{i=1}^{n} (y_i(\beta_0 + \beta_1 x_i) - \log(1 + \exp(\beta_0 + \beta_1 x_i)) \tag{3}$$

which is easier to minimize in order to find the parameter estimates.

The binary logistic regression has multiple-class and multinomial extensions in which the gradient descent method is typically used in order to find the optimal parameters' values.

2.2 Softmax function

The Sotftmax is a function that takes as input a vector of the results of K linear functions and returns a probability distribution.

$$\sigma(z)_i = \frac{\exp z_i}{\sum_{j=1}^{K} \exp z_j} \tag{4}$$

for $i = 1, ..., K$ and $z = (z_1, ..., z_K)$.

The softmax function is used in multiclass classification methods such as multinomial logistic regression, discriminant analysis and neural networks in order to compute the probability of a unit to be in a certain class.

2.3 Simple gradient descent

Gradient descent is an iterative optimization algorithm for finding the minimum of a function and estimating the optimal values of specific parameters.

When fitting a model to a data set, the fit is actually performed by finding the values of β that minimize the cost function. Ideally, they should be determined analytically but when that is not possible the gradient descent can be used.

The basic idea of the algorithm is that function $F(x)$ decreases faster in the direction of the negative gradient $-\nabla F(x)$ and that the old value of x_k is replaced by the value x_{k+1} shifted closer to the minimum: $x_{k+1} = x_k - \gamma_k \nabla F(x)$, where the parameter $\gamma_k > 0$ is called learning rate. The number of iterations can be then limited by reaching a particular absolute value of the gradient $||\nabla F(x)||$ which corresponds to the distance from the minimum of $F(x)$. The first x_0 can be chosen randomly. For γ_k small, $F(x_{k+1}) \leq F(x_k)$. In this way, starting from an initial random guess x_0, for a sufficiently small γ_k the movement is always towards smaller values.

2.4 Stochastic gradient descent

Stochastic gradient descent is a stochastic approximation of the simple gradient descent method, in the sense that in the process it replaces the gradient calculated on the entire data with an estimate of it calculated on a subset of the data itself.

The main idea behind the method is that the gradient can be computed as a sum of gradients, obtained considering the cost function as a sum over data points

$$C(\beta) = \sum_{i=1}^{n} c_i(\vec{x}_i, \beta) \tag{5}$$

from which

$$\nabla_\beta C(\beta) = \sum_{i}^{n} \nabla_\beta c_i(\vec{x}_i, \beta) \tag{6}$$

In order to approximate the gradient, the sum over all data points is replaced by a sum over the data points in one of the subsets, called minibatches, randomly picked at each gradient descent step.

Typically, a dataset is divided into n/M minibatches (where n is the total number of data points and M is size of minibatches) and the process iterates over all of them for a chosen number of epochs.

2.5 Neural Networks

Neural networks are computational systems that consist of layers of connected units sharing mathematical functions. Neural networks can learn to perform tasks by considering examples, generally without being programmed with any task-specific rule.

They can be classified in feed-forward neural networks, convolutional and recurrent ones. Only the first type, in which the information moves forward through the layers, is used for this project.

Each neural network is made of an input layer, one or more hidden layers (single or multi-layer perceptron) and an output layer. Each layer can contain a different number of neurons which need to be activated by a so-called activation function.

The algorithm is based on the output of the activation function

$$y = f\Big(\sum_{i=1}^{n} w_i x_i + b_i\Big) = f(z) \tag{7}$$

in which the inputs x_i are the outputs of previous layers, b_i is the bias and w_i are weights. In this way, each layer receives a weighted sum of the outputs of all units (or neurons) in the previous layer.

From a general point of view, all the layers could have different activation functions and their own number of units. The complete functional form for the output of the l^{th} layer is

$$y_i^l = f^l\Big[\sum_{j=1}^{N_{l-1}} w_j^l f^{l-1}\Big(\sum_{k=1}^{N_{l-2}} w_k^{l-1} f^{l-2}\Big(...f^1\Big(\sum_{n=1}^{N_0} w_n^1 x_n + b_n^1\Big)...\Big) + b_k^{l-1}\Big) + b_j^l\Big] \tag{8}$$

where the only independent variables are the input x_n.

The most typical functions used as activation functions are the logistic Sigmoid, the hyperbolic tangent function and the softmax function, which is normally used in the last layer for classification problems.

Neural Networks also allow to use the final output in order to modify and adjust the weights, so that the errors are as small as possible, through a back propagation algorithm implemented by using gradient descent updates. The gradient which starts the algorithm depends on the derivatives of the cost function at the output layer

$$\frac{\partial C(\hat{W}^L)}{\partial w_{jk}^L} = \delta_j^L a_k^{L-1} \tag{9}$$

where a_k^{L-1} is the output of the activation function of the last hidden layer and

$$\delta_j^L = \frac{\partial C}{\partial b_j^L} \tag{10}$$

is the error of the output layer itself, equal to the rate of change of the cost function C as a function of the bias b.

In order to propagate backwards, the error in the last layer is used to compute the error in the previous layer and so on as

$$\delta_j^{l-1} = \sum_k \delta_k^l w_{kj}^l f'(z_j^{l-1}) \tag{11}$$

The weights and biases are updated through a gradient descent procedure as

$$w_{jk}^l = w_{jk}^l - \eta \delta_j^l a_k^{l-1} \tag{12}$$

$$b_j^l = b_j^l - \eta \delta_j^l \tag{13}$$

where η is the learning rate.

2.6 Performance measurements

In the machine learning context, checking the performance of a classification model is an important task. In order to evaluate and compare the performance of different methods, several metrics can be used.

Firstly, considering the number of correctly guessed targets over the total amount of targets, the accuracy score metric is defined as

$$A(t_i, y_i) = \frac{\sum_{i=1}^{n} I(t_i = y_i)}{n} \tag{14}$$

where n is the total number of targets and I is indicator function which returns 1 for $t_i = y_i$ and 0 otherwise . Due to their definition, accuracy score values lay in the interval [0,1], where 1 represents the accuracy score of the best possible classification method.

Another method which can be applied is called Area Under Curve (AUC), which computes the area under a Receiving Operating Characteristics (ROC) curve. ROC is a probability curve used to understand the extent to which the model is capable to distinguish between different classes. An example can be seen in Figure 7 in the appendix. This curve plots two parameters: the true positive rate (TPR) and the false positive rate (FPR). The first one is the ratio of true positive to all positives, and the second is the ratio of false positives to all negatives (binary classification). AUC ranges in value from 0 to 1. A model whose predictions are 100% wrong has an AUC of 0.0; one whose predictions are 100% correct has an AUC of 1.0, while an AUC of 0.5 reflects random predictions.

Finally, confusion matrices can be produced in order to study the amount of correctly guessed targets after the use of a specific predictive algorithm. A confusion matrix is a table which allows to show the way the classification method is confused when making predictions and it is strictly connected to the concept of accuracy score. In particular, the the diagonals of the table contain the number of correctly predicted targets in all the classes studied and the number of wrong guesses.

For linear regression, instead, in order to quantify the capacity of the methods on fitting the data, two indices are used: the Mean Squared Error (MSE) and the R^2 (variance score) statistics.

The MSE calculates the mean value of the squared difference between data $\vec{y}$ and prediction $\vec{\tilde{y}}$:

$$MSE(\vec{y}, \vec{\tilde{y}}) = \frac{1}{n} \sum_{i=0}^{n-1} (y_i - \tilde{y}_i)^2. \tag{15}$$

R^2 statistic provides an alternative method to express the quality of a fit by evaluating fraction of residual (RSS) and total sum of squares (TSS). It is calculated as

$$R^2(\vec{y}, \vec{\tilde{y}}) = 1 - \frac{RSS}{TSS} = 1 - \frac{\sum_{i=0}^{n-1} (y_i - \tilde{y}_i)^2}{\sum_{i=0}^{n-1} (y_i - \bar{y})^2}, \tag{16}$$

where $\bar{y} = \frac{1}{n} \sum_{i=0}^{n-1} y_i$.

3 Implementation

3.1 Taiwan credit card dataset

The Taiwan credit card data dataset describes customers' default payments in Taiwan and contains a response binary variable (default payment: Yes=1, No=0) and 23 explanatory ones, among which 9 are categorical variables – sex, education, marriage, repayment status and default payment. Table 7 in the appendix summarizes all the recorded variables with their possible values.

3.2 Winsconsin breast cancer diagnostic dataset

The dataset, stored in the sklearn's repository, contains 569 istances and 30 numeric explanatory variables which describe different diagnostic features of benign and malignant tumors.

3.3 Reading and pre-processing the categorical data - credit card case

Firstly, the data has been read from the excel source file [3] and the rows containing values which have been found to be outside of the possible choices for each categorical predictor have been deleted using `pandas.DataFrame.isin` functionality. This was not the case for repayment status because of a vast fraction of badly labeled data.

Hence, a closer look at repayment status data has been taken, noticing the presence of the value "-2" which does not match the right set of recorded values for those categorical variables. It has been decided to set all values "-2" equal to "0" in order to consider them as irrelevant during the analysis.

For repayment status, zero value is also considered as badly labeled and after hotencoding is transformed into zero. This is done manually by storing positions of hotencoded zeros in list_bad array. Values at these positions in design matrix are then set to zeros, i.e. repayment status for zero values is encoded only into zero vector which implies that it does not involve fitting procedure.

Secondly, the function `delete_zeros` (see GitHub link [1]) is applied in order to discard all the rows which contain more than 10 zero values in the columns regarding the amount of bill statement and previous payments because they do not contain any useful information. In particular, for each row the number of zeros contained in the previously stated columns is counted and a list is created aiming to track the lines which are to discard. Finally, the list of rows to be deleted is removed from the dataframe, with a loss of more than 1000 lines.

Thirdly, data from numerical predictors has been standardized using the `standardize` function (see GitHub link), which subtracts the mean from the value and divides it by the standard deviation for each column.

Features and targets have been split in order to be able to apply the one-hot-encoding procedure to each categorical variable by scikit-learn's `OneHotEncoder` and `ColumnTransformer` functionalities, obtaining a feature matrix which contains, for each one-hot-encoded variable, a number of columns corresponding to the number of possible categories, which consists of only ones and zeros.

The design matrix is then built from the feature matrix. In order to split the data between the set used for training (train set) and the one applied in the evaluation process (test set), scikit-learn's `train_test_split` function has been applied by setting 0.8 as training share.

3.4 Simple Gradient Descent

The gradient descent procedure is developed by the function `our_gradient_descent_beta` which requires the design matrix, the target vector, the learning rate and a small number epsilon as an input and returns vector beta as an output. In particular, the function firstly sets a random vector of length equal to the number of parameters to estimate as

```
betas=np.ones(X_1.shape[1])*0.001
```

then a variable `gradient_norm`, set equal to a small number ($2 * \epsilon$ specifically) and a null vector of probabilities with length equal to the number of observed cases are created. Afterwards a loop, which stops whenever the norm of the gradient is less or equal to ϵ, is settled. Inside the loop, another loop over the different data lines is performed, computing the probability associated with each case according to the logistic regression probability formula

```
p[i] = (np.exp((X_1[i,:]).dot(betas)))/(1+np.exp((X_1[i,:]).dot(betas)))
```

for the i-th subject of the dataset. After completing the inner loop, it is possible to compute the gradient, update the parameters and check the gradient norm. The number of total iterations is also stored, in order to be able to refer to the number of `while` loop iterations. The function returns the parameters which are then optimized through the gradient descent method itself and can be used in order to compute predicted values for the targets.

3.5 Stochastic Gradient Descent

The stochastic gradient descent is applied through the function `stochastic_GD` (see GitHub link), which requires the design matrix, the targets vector, the learning rate, the number of inputs (in terms of rows of the design matrix), the mini-batch size and the number of epochs as arguments. Firstly, the function sets a random vector of parameters which will be updated afterwards and computes the number of mini-batches by dividing the total number of inputs by the batch size.

Secondly, a first loop is set in order to go over all the epochs and another one is created to go over all the mini-batches. In the latter, a number `k` is randomly chosen among all possible mini-batches and both the design matrix and the target vector are sliced according to `k`, obtaining a subset for the two of them. A null vector of probabilities is then set and filled with the actual probabilities computed for each element of the subset of the design matrix according to the logistic regression formula for probabilities. After that, the gradient is calculated and used to update the parameters betas, which are returned as the result of the function after the loop over the epochs is concluded.

3.6 Performance measurements

In the classification cases, accuracy score, AUC and confusion matrices are used to evaluate the performance of different methods. The `accuracyscore` function is implemented in order to compute the accuracy of predictions made by using the parameter estimates defined by the gradient descent procedures. In particular, a loop over all values in the probability vector is set aiming to count the number of predictions which correspond exactly to the target values. Finally, the accuracy is computed as the number of right predictions over the total amount of them according to equation (14). The accuracy is then also computed using scikit-learn's functionality `accuracy_score` and similar results are obtained.

The area under curve function is computed using sklearn's functionality `roc_auc_score`, while also storing results for the ROC curve through `roc_curve` functionality.

A confusion matrix is also produced for each of the methods studied, by applying the sklearn's `confusion_matrix`.

In terms of linear regression analysis, mean square error and R^2 metrics are computed by using sklearn's functions `mean_squared_error` and `r2_score`.

3.7 Neural Network with feed forward and back propagation algorithms for classification

The neural network has been implemented using both back propagation and feed forward algorithms. Firstly, it has been decided to use three hidden layers, with 50, 40 and 30 neurons each respectively. The number of categories is set equal to two, therefore there is the need to one-hot encode the response vector. This is processed using the `to_categorical_numpy` function (see GitHub link), which allows to one-hot-encode both the train and test y subsets.

Secondly, weights and biases for the three hidden layers and the output layer are initialized as random vectors.

Combinations of different learning rates (eta) and the regularization parameters (lmbd), where values are set as

```
eta_vals = [0.0000001,0.000001,0.00001,0.0001,0.001]
lmbd_vals = [0.0001,0.001,0.01,0.1,1,10]
```

for the credit card data and as

```
eta_vals = [1e-5,1e-4,1e-3,1e-2,1e-1]
lmbd_vals = [1e-5,1e-4,1e-3,1e-2,1e-1,1]
```

for the breast cancer data, have been studied.

By running a loop over different iteration (with maximum number set to 100) it is possible to update weights and biases through the neural network procedure itself and to determine which of the combination gives the best results in terms of accuracy score and AUC values.

The back propagation algorithm is firstly applied using the `backpropagation` function, which requires the training subsets of both the design matrix and the one-hot-encoded response vector. The function immediately recalls the `feed_forward_train` process, which computes weighted sums of inputs for each layer and respective activations using the sigmoid activation function

```
def sigmoid(x):
    return 1/(1 + np.exp(-x)).
```

Weighted sums and activations are passed to next layer until the output layer is reached. The three activations and the vector of probabilities (output) are then returned as an output of `feed_forward_train` and used in the back propagation process.

In particular, the back propagation algorithm computes the error of each layer and the respective gradient for both weights and biases, starting from the output layer and going back to the first hidden one, considering the Cross Entropy as the cost function to be minimized. The back propagation function then returns all different computed gradients.

Next step is then to regularize all gradients through the parameter lambda and to update weights and biases using the learning rate and the computed gradients, applying a gradient descent process.

After a set number of iterations of the loop, accuracy score can be computed and predictions for the response vector obtained by applying the function `predict`, which recalls a feed forward process in order to get a matrix of probabilities and then evaluates the maximum value within each column, returning its index as a result. The validity of the algorithm itself is checked by using sklearn's `MLPClassifier` and computing both accuracy score and AUC on the test data.

3.8 Neural Network with feed forward and back propagation algorithms for linear regression

The neural network algorithm, applying both back propagation and feed forward, has been used to study a linear regression case from the previous project, in particular regarding the analysis of the Franke function's data.

After setting the design matrix, all the available data is split in test and training subsets, in order to perform more accurate analysis.

The applied neural network algorithm is the same as the one used for the classification case, except of the cost function which is no longer the Cross Entropy but becomes the Mean Square Error (MSE), which leads to a different definition of the error for the output layer. Another difference consists of the output produced in the feed forward algorithm: the softmax function is no longer needed in order to get probabilities in the output layer since the output is now given by the sigmoid function applied to the weighted sum of inputs to the output layer and depends on the output weights, output bias and activation in the previous hidden layer. As a result of this slightly different approach, predictions now depend only on the output produced by the feed forward algorithm.

Results are studied for different combinations of learning rates and regularization parameters as done in the classification case.

4 Results and discussion

Breast cancer data, credit card data and Franke function's data have been studied by different methods such as simple and stochastic gradient descent and neural networks. The validity of all the results has been evaluated by accuracy score, Area Under Curve and confusion matrices in the classification case, by R^2 and MSE statistics in the regression case.

A first insight into the classification problem datasets is given by the heatmaps in Figure 1 and Figure 2, representing the correlations among features of both the credit card data and the breast cancer one.

In the two cases, some variables seem to be highly correlated and even though a dimensionality reduction approach could have been taken, it has been preferred to consider all the variables during the analysis in order to maintain the original problem frame.

4.1 Logistic regression for breast cancer data set

Method	Accuracy score
Simple gradient descent	0.9300699300699301
Stochastic gradient descent	0.9370629370629371
Sklearn accuracy score	0.958041958041958
Neural network: $\lambda = 0.01, \eta = 0.01$	0.9790209790209791
Neural network sklearn: $\lambda = 0.01, \eta = 0.01$	0.972027972027972

Table 1: Summary of accuracy score for different methods for breast cancer data.

Method	AUC
Simple gradient descent	0.9289308176100629
Stochastic gradient descent	0.9422431865828093
Neural network: $\lambda = 0.01, \eta = 0.01$	0.979454926624738
Neural network sklearn: $\lambda = 0.01, \eta = 0.01$	0.970020964360587

Table 2: Summary of Area Under Curve (AUC) of different used method for breast cancer data.

Results of all used methods for the breast cancer data, in terms of accuracy and AUC, are summarized in Tables 1 and 2.

It is possible to state that neural network is the algorithm which performs in the most satisfying way on the data analyzed. In particular the best accuracy, according to scikit-learn's functionality MPLClassifier, is reached by using $\lambda = 0.01$ and $\eta = 0.01$ as parameters. The own neural network code returns 0.979 as the highest accuracy computed, using the same parameters' combination as scikit-learn's.

Choosing the learning rate η is one of the most crucial parts of building a neural network, since a very small parameter may result in a long training process with long convergence time, whereas a very large value could lead to the learning of a sub-optimal set of weights in an extremely fast way or to an unstable training process. Another important decision to make is regarding the choice of the regularization parameter λ, which allows to put a constraint on the size of the weights implied, aiming to contain their growth and to avoid overfitting. In particular, larger weight values are more penalized if λ is large. A deeper analysis of how differences among several combinations of learning rates and regularization parameters affect the accuracy score values can be found in Figures 3 and 4 in the appendix. Interestingly, it has been noticed that having the learning rate fixed, changes in the regularization parameter result in very small differences in terms of the performance metrics studied, underlining the key role that the learning rate plays in determining the best method to be used.

Accuracy is a very intuitive method used to evaluate the performance of a model, however it is reliable only when the number of false positive and false negative values in a dataset is almost the same. Having to deal with biased data, it seems reasonable to compute an alternative statistic which aims to evaluate the performance of the algorithms implemented. In particular, the AUC (Area Under Curve) metric is chosen to determine which method performs the best. In the case of breast cancer data, AUC agrees with accuracy score suggesting the neural network algorithm as the best for the data analyzed in terms of predictions given.

Confusion matrices showed in Figures 5 and 6 in the appendix are displayed in order to give an insight of how well the different methods applied perform on the classification of the data itself. The dataset used is quite small, however it is possible to see that the neural network algorithm is the one giving the highest rate of correctly guessed targets, by looking at the percentages in the anti-diagonal elements.

Lastly, a visualisation of AUC has been made by Receiver Operating Characteristics (ROC) curves in Figure 7 in the appendix.

4.2 Logistic regression on credit card data

Method	Accuracy score
Simple gradient descent	0.8238606369875066
Stochastic gradient descent	0.8215731127925392
Sklearn accuracy score	0.8272039415801513
Neural network: $\lambda = 10, \eta = 0.00001$	0.7858525426711244
Neural network sklearn: $\lambda = 0.0001, \eta = 0.0001$	0.8272039415801513

Table 3: Summary of accuracy score of different methods for credit card data.

Method	AUC
Simple gradient descent	0.6600631977964128
Stochastic gradient descent	0.6524705934208744
Neural network: $\lambda = 10, \eta = 0.0001$	0.6591697411268396
Neural network sklearn: $\lambda = 0.0001, \eta = 0.0001$	0.683187409779005

Table 4: Summary of Area Under Curve (AUC) for different used method for credit card data.

Results of all used methods in terms of accuracy and AUC for the credit card data are summarized in Tables 3 and 4.

In the credit card data case, the neural network algorithm does not perform as well as it does for the breast cancer problem. Particularly looking at the results obtained by using scikit-learn, it seems that the algorithm itself performs quite well, although the outcome is not totally reliable due to convergence issues which are encountered during the iterations. The best conclusions, according to the accuracy score, are obtained by setting λ=10 and η=0.00001 with the own algorithm and λ=0.0001 and η=0.0001 with scikit-learn's (further studies of parameters' combinations can be seen in Figure 8 in the appendix).

In general, all the values obtained for accuracy scores with different methods are similar. It can be interesting to study the performance of these approaches using some other kind of statistic in order to see whether results would differ more or not. Aiming to achieve a conclusion which is reliable in the case of biased data, AUC index is computed. Using this alternative criterion, it is possible to state that neural network is the method which performs the best on the data in terms of its classification, in particular with the combination of λ=0.0001 and η=0.0001 for scikit-learn and λ=10 and η=0.0001 with the own code (see Figure 9). As noticed for the breast cancer data case, by looking at accuracy score and AUC values over different parameters' combinations, it is clear that the learning rate has an important role in determining the best method. In fact, given a specific value for that parameter, changes of the regularization term λ do not lead to significant changes in the performance measurement values.

Confusion matrices (Tables 10 and 11 in the appendix) show that all algorithms tend to perform in a similar way, even though the neural network seems to be the one which can classify the targets y=1 better.

4.3 Regression for Franke's function

	OLS	Ridge	Lasso
MSE	0.0020510889	0.0020673088	0.0087683865
R^2	0.9737030822	0.9734951268	0.8937880494

Table 5: Statistics obtained with scikit-learn functions from project 1.

	MSE	R^2
NN own code: $\lambda = 0.00001, \eta = 1$	0.017093883312645722	0.9014514754185878
NN sklearn: $\lambda = 0.0001, \eta = 0.1$	0.00013406661880399936	0.9992270880035212

Table 6: Statistics obtained with the neural network algorithm

In Table 5 results from previous studies of Franke function with Ordinary Least Squares, Ridge and Lasso regression methods are summarized (see Project 1 [2]).

Aiming to check whether there exist more effective methods for prediction in a linear regression context, the analysis of the data produced by the so-called Franke function has been extended to a more complex method, which is neural network.

Particularly, the neural network algorithm implemented to perform a prediction analysis on the regression problem considered has been evaluated in terms of reliability by studying the convergence of MSE to 0 for the training data over

100 iterations, simultaneously checking the progressive increase of the R^2 metric towards 1. The described behaviours can be observed in Figure 12.

Different combinations of learning rate and regularization parameters have been studied (Figure 13 in the appendix) in order to evaluate performance metrics over different models, resulting in λ=0.00001 and η=1 in the case of the own neural network algorithm and in λ=0.0001 and η=0.1 for scikit-learn as the best combinations of parameters to use.

By comparing previous results with the ones displayed in Table 6, it can be noticed that scikit-learn's neural network performs better than the other regression methods studied. This is probably due to the usage of more efficient methods for weight optimization and prediction, such as so-called Adam gradient descent optimizer. On the contrary, results obtained by the own neural network are found to be even less precise than the ones obtained by OLS or Ridge method. Especially in case of MSE, results obtained by the own neural network differ by one order of magnitude from OLS, Ridge and Lasso methods and by two order of magnitude from scikit-learn's neural network. Difference for R^2 is less but still around 10%. Hence, scikit-learn's outcomes are considered to be more reliable and they can be used to point neural network as the algorithm which works the best on the given linear regression problem.

5 Conclusions

Different methods, such as simple gradient descent, stochastic gradient descent and neural networks, have been applied in order to evaluate the predictive performance on three different datasets – two classification and one linear regression problems. All the considered algorithms behave similarly over all kinds of analyzed cases, recognizing the neural network as the best method in terms of performance compared to the others.

In the linear regression case, neural networks bring better results than the regression methods studied in the previous project, such as OLS, Ridge and Lasso. In both the classification problems, neural networks overcome gradient descent techniques in terms of prediction accuracy, even though the results they give for credit card data are less impressive than the ones obtained for the breast cancer data case.

In comparison to the own algorithms, scikit-learn's functionalities performed always better except for neural network's breast cancer data case. However, with the exclusion of Franke function's results, outcomes obtained with the own algorithms agree with the scikit-learn's ones within 7% error, in most cases having this agreement to be even better (error within 4%). Considering Franke function's data, the own algorithm did not perform comparably well.

To summarize the approaches examined, gradient descent has been found to be a simple method, sensitive to the choice of learning rate γ. However, obtaining a precise result is quite time demanding. This can be improved by using stochastic gradient descent, which utilizes the idea of batches and avoids falling into local minimum. Neural networks, anyway, provide better results than gradient descent in a variety of problems. One disadvantage of this algorithm, though, is the great consumption of computational memory and the inability for humans to understand the actual meaning of values found the back propagation algorithm during the learning process.

An interesting comparison could be performed by studying the behaviour of other techniques, such as decision trees and convolutional neural networks, in the given context.

References

[1] Project repository https://github.com/storejar/Data_analysis_and_machine_learning/tree/master/Project2_wd

[2] Project1 repository https://github.com/storejar/Data_analysis_and_machine_learning/tree/master/Project1

[3] Excel source file "default of credit card clients.xls" https://github.com/storejar/Data_analysis_and_machine_learning/blob/master/Project2_wd/default%20of%20credit%20card%20clients.xls

[4] James, G., Witten, D., Hastie, T., Tibshirani, R. (2013). An Introduction to Statistical Learning. Springer Texts in Statistics. Springer New York. https://doi.org/10.1007/978-1-4614-7138-7

[5] Hastie, Trevor, et al. The Elements of Statistical Learning: Data Mining, Inference, and Prediction. Springer, 2017.

[6] Morten Hjorth-Jensen, (2019), Lectures Notes in FYS-STK4155. Data Analysis and Machine Learning: Optimization and Gradient Methods, https://compphysics.github.io/MachineLearning/doc/pub/Splines/html/Splines.html

[7] Morten Hjorth-Jensen, (2019), Lectures Notes in FYS-STK4155. Data Analysis and Machine Learning: Logistic Regression, https://compphysics.github.io/MachineLearning/doc/pub/LogReg/html/LogReg.html

[8] Morten Hjorth-Jensen, (2019), Lectures Notes in FYS-STK4155. Data Analysis and Machine Learning: Neural networks, from the simple perceptron to deep learning, https://compphysics.github.io/MachineLearning/doc/pub/NeuralNet/html/NeuralNet.html

A Appendix

Label	Description
LIMIT_BAL	Amount of given credit in New Taiwan (NT) dollars
SEX	Gender (1=male, 2=female)
EDUCATION	(1=graduate school, 2=university, 3=high school, 4=others)
MARRIAGE	Marital status (1=married, 2=single, 3=others)
AGE	Age in years
PAY_0	Repayment status in September, 2005 (-1=pay duly, 1=payment delay for one month, 2=payment delay for two months, . . . 8=payment delay for eight months, 9=payment delay for nine months and above)
PAY_2	Repayment status in August, 2005 (scale same as above)
PAY_3	Repayment status in July, 2005 (scale same as above)
PAY_4	Repayment status in June, 2005 (scale same as above)
PAY_5	Repayment status in May, 2005 (scale same as above)
PAY_6	Repayment status in April, 2005 (scale same as above)
BILL_AMT1	Amount of bill statement in September, 2005 (NT dollar)
BILL_AMT2	Amount of bill statement in August, 2005 (NT dollar)
BILL_AMT3	Amount of bill statement in July, 2005 (NT dollar)
BILL_AMT4	Amount of bill statement in June, 2005 (NT dollar)
BILL_AMT5	Amount of bill statement in May, 2005 (NT dollar)
BILL_AMT6	Amount of bill statement in April, 2005 (NT dollar)
PAY_AMT1	Amount of previous payment in September, 2005 (NT dollar)
PAY_AMT2	Amount of previous payment in August, 2005 (NT dollar)
PAY_AMT3	Amount of previous payment in July, 2005 (NT dollar)
PAY_AMT4	Amount of previous payment in June, 2005 (NT dollar)
PAY_AMT5	Amount of previous payment in May, 2005 (NT dollar)
PAY_AMT6	Amount of previous payment in April, 2005 (NT dollar)
default.payment.next.month	Default payment in June, 2005 (1=yes, 0=no)

Table 7: Summary of the all data variables.

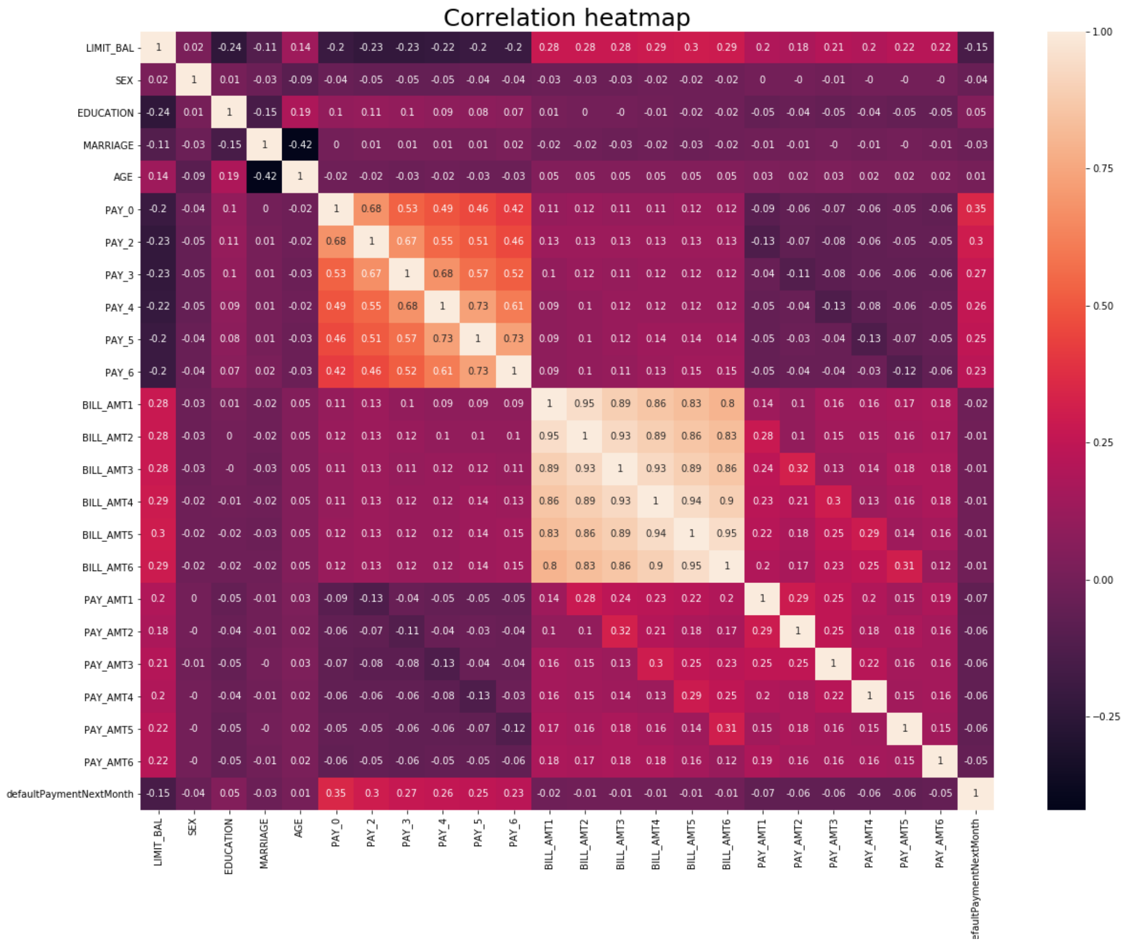

Figure 1: Correlation matrix heatmap for credit card data.

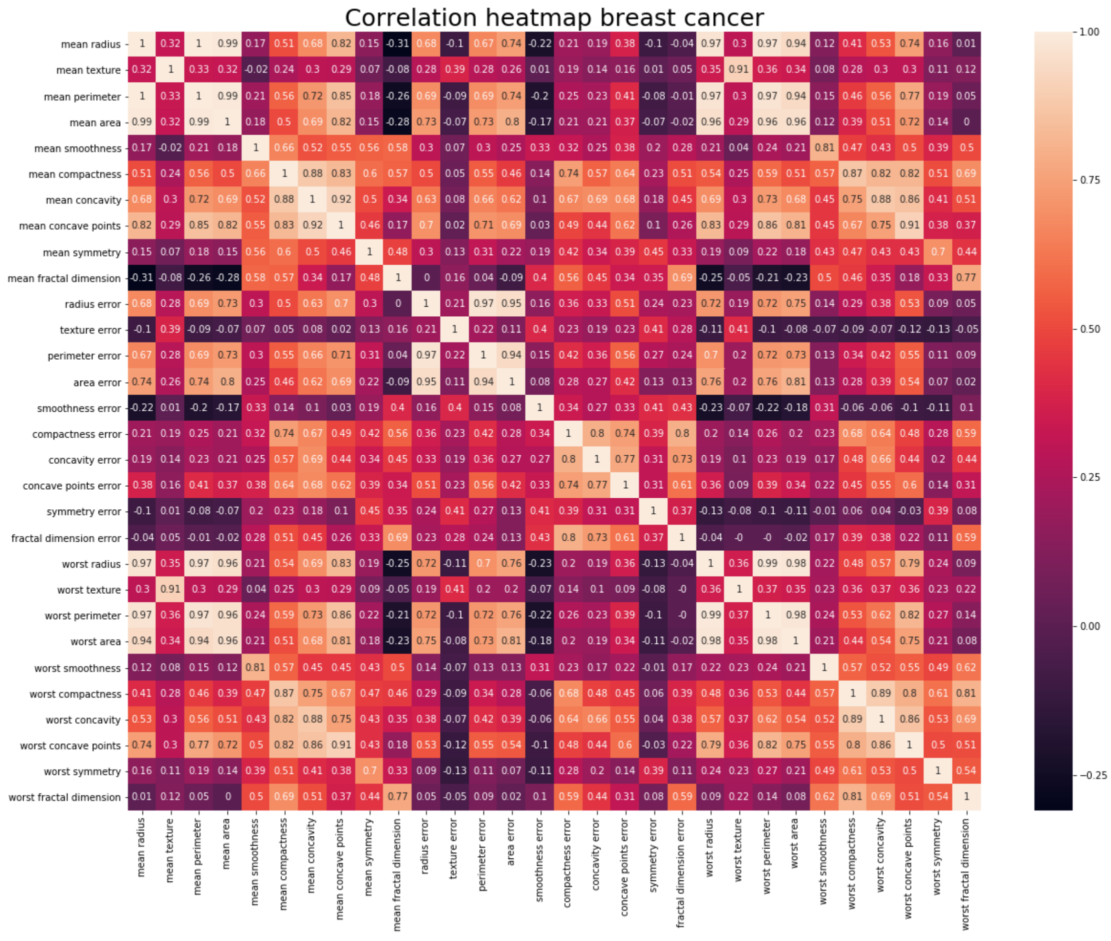

Figure 2: Correlation matrix Breast Cancer data set

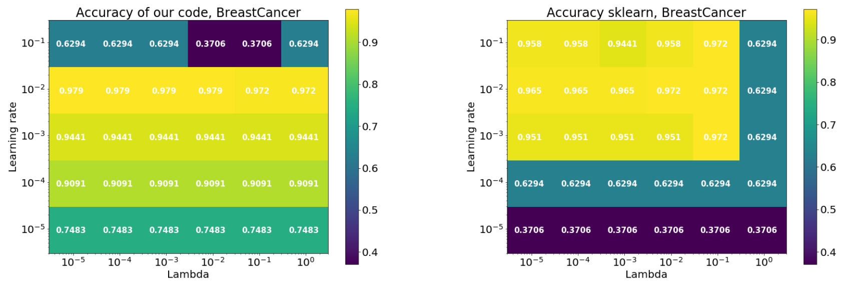

Figure 3: Accuracy dependence on learning rate η and lambda λ for breast cancer data left for the own algorithm and right for scikit-learn.

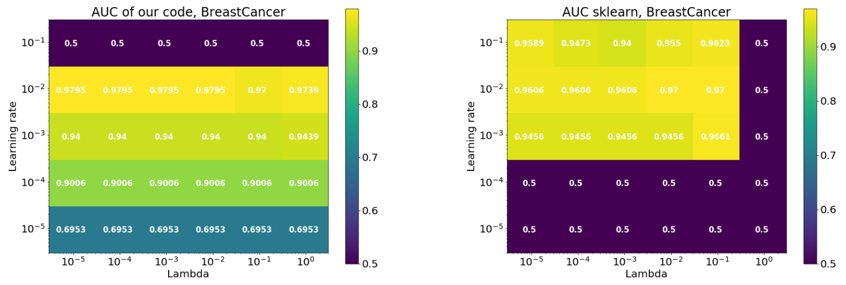

Figure 4: Area Under Curve dependence on learning rate η and lambda λ for breast cancer data left for the own algorithm and right for scikit-learn.

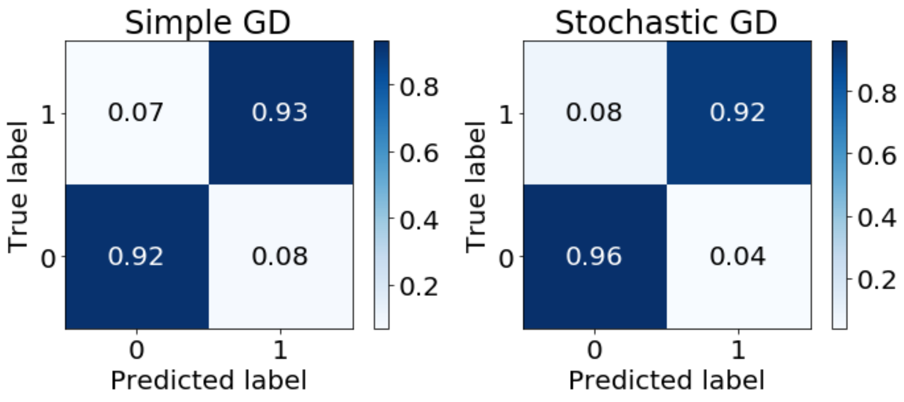

Figure 5: Confusion matrices for breast cancer data for the own simple gradient descent (left) and stochastic gradient descent (right).

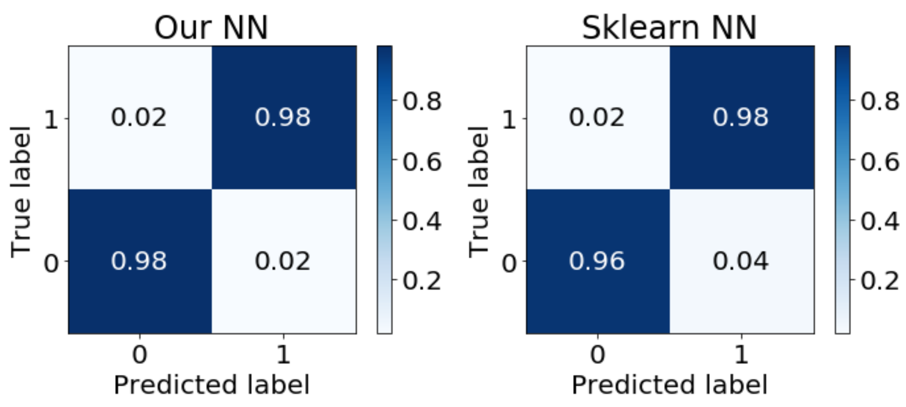

Figure 6: Confusion matrices for breast cancer data for Neural Networks with the own code on the left and from scikit-learn on the right.

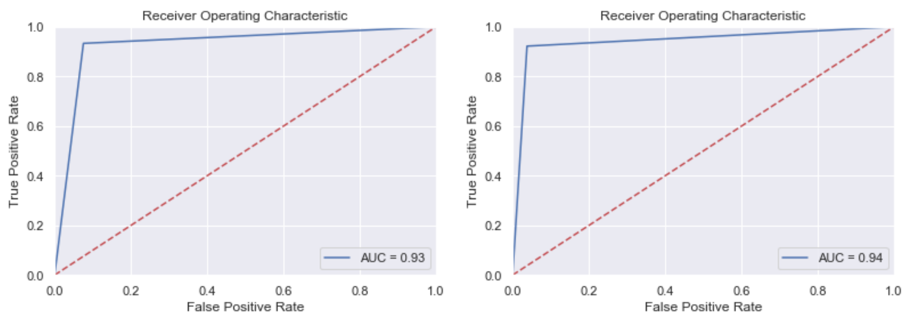

Figure 7: Receiver Operating Characteristic for simple gradient descent on the left and for stochastic gradient descent on the right on breast cancer data

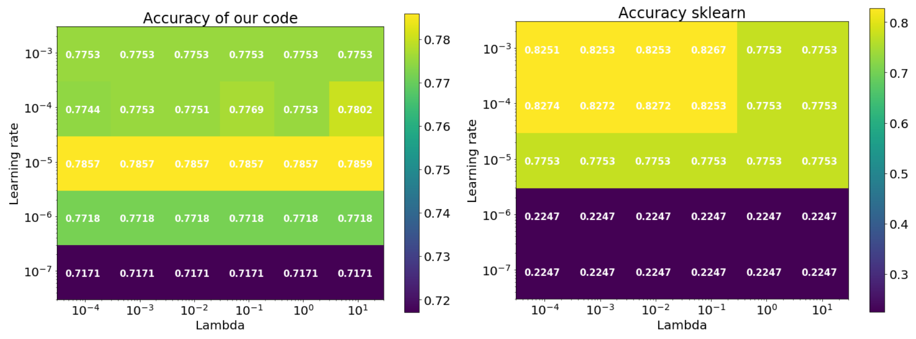

Figure 8: Accuracy dependence on learning rate η and lambda λ for credit card data left for the own algorithm and right for scikit-learn.

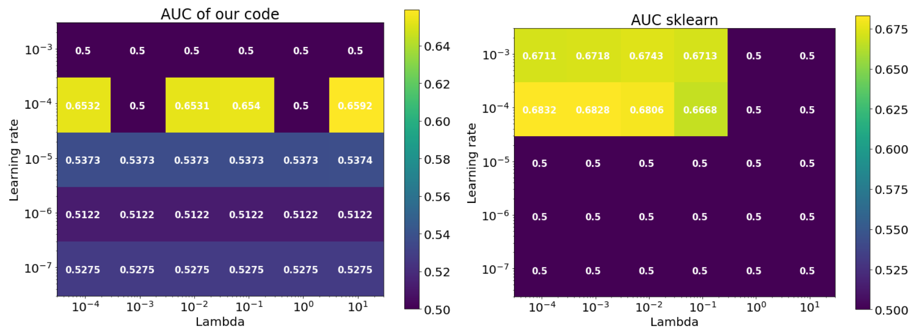

Figure 9: Area Under Curve dependence on learning rate η and lambda λ for credit card data left for the own algorithm and right for scikit-learn.

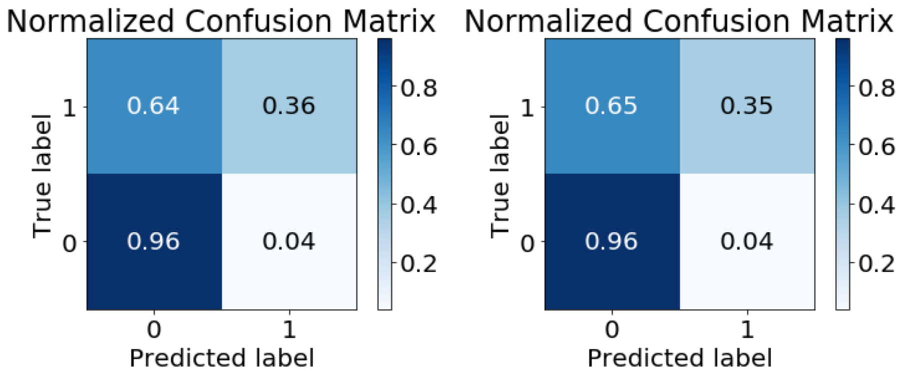

Figure 10: Confusion matrices on credit card data respectively for simple gradient descent on the left and stochastic gradient descent on the right

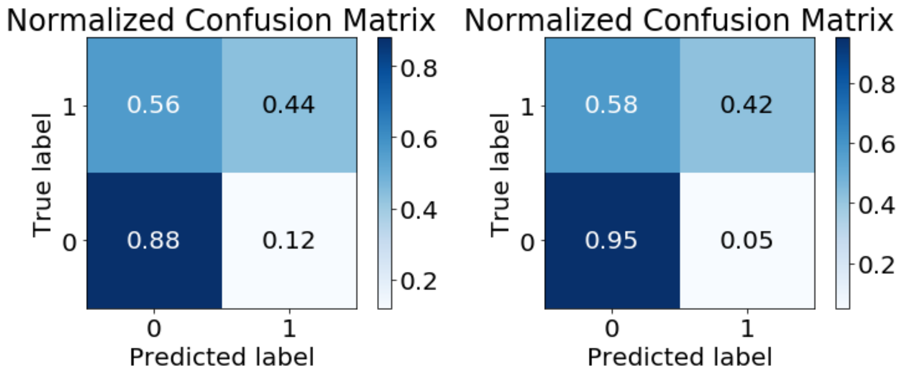

Figure 11: Confusion matrices on credit card data respectively for Neural Networks with the own code on the left and from scikit-learn on the right

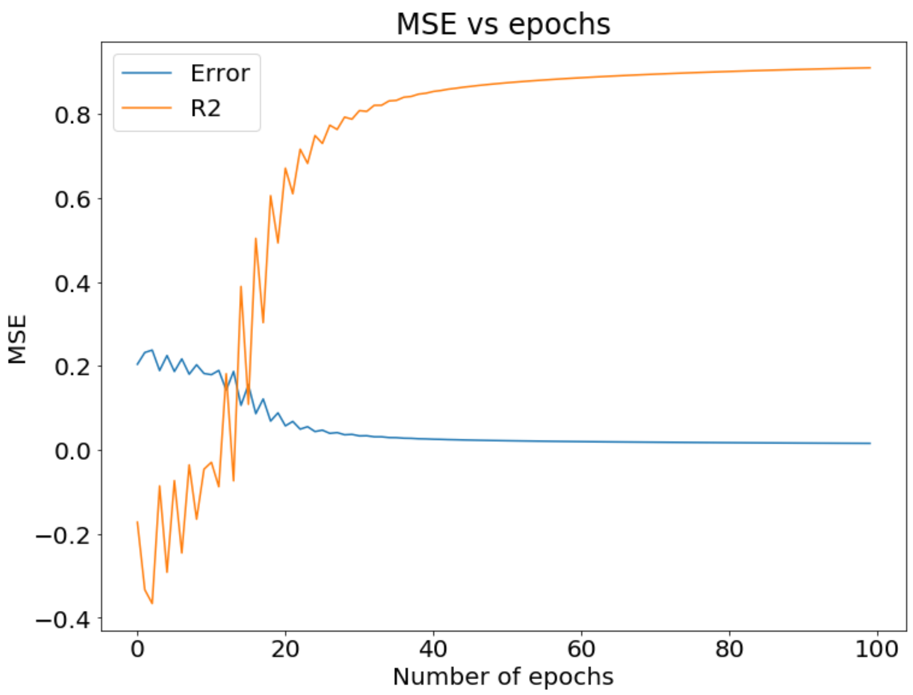

Figure 12: MSE and R2 computed with the own Neural network code and plotted against the number of epochs for Franke function.

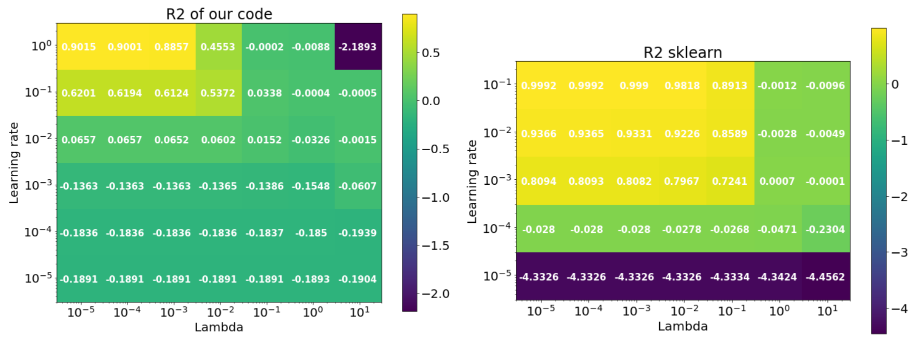

Figure 13: R^2 dependence on learning rate η and lambda λ for Franke function, left for the own algorithm and right for scikit-learn.