ORIGINAL ARTICLE
Journal Section

FYS-STK Project 2 on Machine Learning Fall 2019

Alexander D. Aliferis | Marius Stette Jessen

Department of Physics, University of Oslo, Norway

[1]Department of Physics, University of Oslo, Norway

Correspondence
Email: alexadal@fys.uio.no

This report considers a study of machine learning methods conducted on a data set of credit card clients in Taiwan over several months in 2005. Algorithms are deployed in an attempt of creating models to predict if a client will default their payment in June 2005. The methods used are logistic regression and artificial neural networks. The performance of the different methods is compared and analysed. The python code supporting the analysis in this report is uploaded to the GitHub repository.

KEYWORDS
Logistic Regression, Artificial Neural Network, Machine Learning

1 | INTRODUCTION

Consumer borrowing is rapidly increasing worldwide, substantially driven by lenders providing easily applicable credit card loans to individuals. These high interest loans are great sources of income to consumer banks and they are broadly discussed as the interest rates seem to be ever increasing in contrast to the latest development of interbank offered rates e.g LIBOR[5]. However, as with all high cash-yield loans, there is a risk of counterpart default, especially with accelerating rates. That is, the consumer is not able to cover their outstanding credit or minimum down payment for 6 successive months, and consequently defaults the loan. To avoid this, banks and other loan issuing companies pursue models and algorithms for prediction of default, given customer information. The larger banks have already solid databases of customer conduct, and with proper usage of Machine Learning (ML) algorithms, risk can be minimized without too wide exclusion of potential loan-takers, maximizing profit. The question then arises on which ML model to choose, giving the most accurate predictions based on historical credit card data with target values of default or no-default.

This study, similarly to Yeh and Lien[8], utilize credit card data from Taiwan (2005) in different ML algorithms and compare their results. However, given the conclusive comparison of Artificial Neural Networks (ANNs) and Logistic regression (LR) in Yeh and Lien[8], this study focus on the elaboration of these two exact models. Simply put, the

purpose of this paper is to give a more general analysis of how the different models are optimized in terms of prediction accuracy and other scoring assessments by tuning model parameters. For the case of ANNs, the study also cover the topic of regression with targets outside the binary range, fitting the Franke-function.

2 | THEORY AND METHODOLOGY

The general theory in this section is obtained from the course lecture notes Hjorth-Jensen [4] and the textbook Hastie et al. [3]. For a more comprehensive description on the theory behind ANNs, reading Nielsens book on Neural Networks is strongly advised.

2.1 | Logistic Regression (log-reg)

In logistic regression, the aim is to obtain the conditional probability of achieving a binary outcome $p(Y = 1 \mid X = x)$ or $p(Y = 0 \mid X = x)$. The mathematical expressions for these probabilities are given by the following exponential properties

$$p(y_i = 1 | x_i, \hat{\beta}) = \frac{\exp(\beta_0 + \beta_1 x_i)}{1 + \exp(\beta_0 + \beta_1 x_i)},$$
$$p(y_i = 0 | x_i, \hat{\beta}) = 1 - p(y_i = 1 | x_i, \hat{\beta}),$$

For easier explanation, $p(Y = 1 \mid X = x)$ is simply stated as $p(X)$. To estimate the unknown βs or "weights" in the expressions above, maximum likelihood is used

$$\ell(\beta_0, \beta_1) = \prod_{i, y_i = 1} p(x_i) \prod_{i, y_i = 0} (1 - p(x_i)) \tag{1}$$

The likelihood, is the probability of the observed zeros and ones in the data, $\mathcal{D} = \{(y_i, x_i)\}$. Hence, one is to choose the "weights" that maximize $\ell(\beta_0, \beta_1)$. To do this, it is common to restate equation 1 as the negative log-likelihood

$$C(\hat{\beta}) = -\sum_{i=1}^{n} \left(y_i(\beta_0 + \beta_1 x_i) - \log(1 + \exp(\beta_0 + \beta_1 x_i)) \right). \tag{2}$$

which then serves as a cost function to minimize. As the expression in equation 2, or more precisely the cross-entropy cost function, is convex, any local minima serves as a global minima. This is ever important as analytically optimizing the cost function wrt. to its weights is impossible as seen below

$$\frac{\partial C(\hat{\beta})}{\partial \beta_0} = -\sum_{i=1}^{n}\left(y_i - \frac{\exp(\beta_0+\beta_1 x_i)}{1+\exp(\beta_0+\beta_1 x_i)}\right)$$

and

$$\frac{\partial C(\hat{\beta})}{\partial \beta_1} = -\sum_{i=1}^{n}\left(y_i x_i - x_i\frac{\exp(\beta_0+\beta_1 x_i)}{1+\exp(\beta_0+\beta_1 x_i)}\right).$$

or in a more compact matrix form

$$\frac{\partial C(\hat{\beta})}{\partial \hat{\beta}} = -\hat{X}^T(\hat{y}-\hat{p}) = \nabla_\beta C(\beta),$$

where $\hat{X}$ have the dimensions, n×m, respectively for number of data and number of features.

Consequently, to yield results for the credit card data, numerical optimizers such as gradient descent (GD), Newton Raphson (NR) and Stochastic Gradient descent (SGD) are put to play. These method all share the same representation of computing weights that optimize the cost function in the form of

$$\beta_{new} = \beta_{old} - \eta\nabla_\beta C(\beta_k), k = 0, 1, \cdots \tag{3}$$

Where the learning rate, η, is replaced by $\left(\frac{\partial^2 C(\hat{\beta})}{\partial\hat{\beta}\partial\hat{\beta}^T}\right)^{-1}$ in NR, and the gradient $\nabla_\beta C(\beta_k)$ is replaced by a subset-gradient $\sum_i^n \nabla_\beta c_i(\boldsymbol{x_i}, \beta)$ in SGD. Usually, SGD is preferred over GD as randomness is introduced by only taking the gradient of a subset of the data, so-called minibatches. The data-points in the batches are taken arbitrarily by shuffling, and an iteration over a new set of batches, epoch, is repeated several times. This method is then capable of escaping shallow local minima while updating the weights of the model faster as fewer amounts of data are used in the computation.

2.2 | Artificial Neural Networks (ANNs)

The logic behind Artificial Neural Networks, is not too different to that of Log-reg. The model consist of a number of neurons or nodes in different layers, that "communicate" in mathematical form. Between each neuron in neighboring layers, there is a connection represented by a weight variable, and before information is propagated forwards, the accumulated signal at a node has to go through an activation threshold, $f(z)$, to yield an output. This activation is often the same as the binary conditional probability in the previous section

$$p(y_i = 1|x_i, \hat{\beta}) = \frac{\exp(\beta_0+\beta_1 x_i)}{1+\exp(\beta_0+\beta_1 x_i)},$$

also known as the sigmoid function, $\sigma(\beta_i, x_i)$. As follows, log-reg is actually a subset of ANN classifiers, with one

hidden layer of a single neuron. Hidden layers are all layers that are neither an input or output.

Moreover, for a given layer, the input can be represented as z_i^l (as opposed to x_i for the input layer), and the output as $y_i = a_i^l = f(z_i^l)$. z_i^l is again a function of weights, w, and biases, b

$$z_i^l = \sum_j w_{ij}^l a_j^{l-1} + b_i^l, \tag{4}$$

where the superscript $l-1$ indicate outputs of the prior layer. Consequently, the cost function in the final layer, that is to be minimized, is

$$C(\hat{W}) = -\sum_{i=1}^{n} \left(t_i \log a_i^L + (1 - t_i) \log (1 - a_i^L) \right) \tag{5}$$

where the targets are defined as t_i. To minimize the cost function, back-propagation with SGD is used. Back-propagation is, simply put, utilization of the chain rule wrt. to all the activations, weights and biases from the final to the first layer. A short, but more detailed explanation follow the steps below, values are presented by matrix-notation. Please note that $\boldsymbol{w}^T$ has the shape $(n_{prev.layer}, n_{next.layer})$ and $\circ$ denotes as the Hadamard product.

1. Do a feed-forward. That is, send the input signal throughout the network and evaluate the cost.
2. Divide the computation of derivatives into two pieces, one for the final outer layer and the other for the inner hidden layers
   - Outer Layer
     - calculate $\delta_{out} = \frac{\partial C}{\partial a_{out}} \circ \frac{\partial a_{out}}{\partial z_{out}} = \boldsymbol{a_{out}} - \boldsymbol{y}$ (for the cross-entropy cost)
     - $\frac{\partial z_{out}}{\partial w_{out}} = \boldsymbol{a_{out-1}}$
     - $\frac{\partial C}{\partial w_{out}} = \delta out^T \circ \frac{\partial z_{out}}{\partial w_{out}}$ and $\frac{\partial C}{\partial b_{out}} = \sum_{n_{inputs}} \delta_{out}$
   - Hidden layers
     - For layer h, ranging from the second last to the first layer, calculate
     - $\delta_{h+1} = \frac{\partial C}{\partial a_{h+1}} \circ \frac{\partial a_{h+1}}{\partial z_{h+1}}$, i.e delta from the previous layer in the iteration, that goes backwards from the outer layer
     - $\frac{\partial C}{\partial a_h} = \frac{\partial C}{\partial a_{h+1}} \circ \frac{\partial a_{h+1}}{\partial z_{h+1}} \circ \frac{\partial z_{h+1}}{\partial a_h} = \delta_{h+1} \circ \boldsymbol{w_{h+1}}$
     - $\frac{\partial z_h}{\partial a_h} = f_h'(z_h)$
     - $\delta_h = \frac{\partial C}{\partial a_h} \circ \frac{\partial z_h}{\partial a_h} = \delta_{h+1} \circ \boldsymbol{w_{h+1}} \cdot f_h'(z_h)$
     - $\frac{\partial z_h}{\partial w_h} = \boldsymbol{a_{h-1}}$
     - To finally give, $\frac{\partial C}{\partial w_h} = \delta_h^T \circ \frac{\partial z_h}{\partial w_h}$ and $\frac{\partial C}{\partial b_h} = \sum_{n_{inputs}} \delta_h$

As for log-reg, with a "SGD-solver" for every batch in its corresponding epoch, the unique biases and weights are computed as in equation 8

2.2.1 | Other cost and activation functions

In order to utilize the Neural Network onto other problems such as linear regression under the noisy Franke-Function (used by the authors in a previous study), the cost function has to be replaced by

$$MSE = C(\hat{W}) = \frac{1}{2N}\sum_{i}^{N}(y_i - t_i)^2 \tag{6}$$

Also, to obtain answers outside of the binary range, the activation in the final layer have to be substituted by either ReLU:

$$f(x) = max(0, x)$$
$$f'(x) = 0 \text{ or}$$
$$f'(x) = 1$$

or

Linear:

$$f(x) = x$$
$$f'(x) = 1$$

For any input passed to the function, x.

2.3 | Regularization

Regularization is implemented in order to constrain the size of the weights so that they do not grow out control, and hence prevent over-fitting. This is done by the following additions to the cost functions and to each gradient corresponding to its weight in weight computation

$$Cost_{regularized} = Cost_{original} + \frac{1}{2}\lambda \cdot \sum_{ij} {w_{ij}}^2 \tag{7}$$

$$\nabla_{w_i} C(w)_{regularized} = \nabla_{w_i} C(w)_{original} + \lambda \cdot w_i \tag{8}$$

The addition in 7, goes under the category of L2-norm regularization. Note that the regularization only yields on the weights and not the biases.

2.4 | Model-Evaluation Properties

To evaluate how the models perform in sense of prediction, the following metrics are studied

2.4.1 | Accuracy

The accuracy is measured as the number of correct classifications from the output of a model. The formula is given by equation 9

$$Accuracy = \frac{\sum_{i=1}^{n}(t_i = y_i)}{n} \tag{9}$$

where I is the indicator function that give 1 if $t_i = y_i$ and 0 otherwise. All of the models are constructed in such way that the output is categorized to 0 or 1 for probabilities below and above 0.5, respectively.

2.4.2 | R2-score

As for the previous study done by the authors, R2 score is given by

$$R2 = 1 - \frac{\sum_{i}^{N}(z_i - \tilde{z}_i)}{\sum_{i}^{N}(z_i - \bar{z}_i)} \tag{10}$$

R2 measures how well the variance in the data are explained by the (linear) regression line. It has the most optimal score of 1.

2.4.3 | Cumulative Gains plot

Cumulative gains is a visual representation of model performance that sort outputs wrt. to highest probabilities along the x-axis of the plot. Hence moving along the x-axis, one select the X best deciles, and along the y-axis values give the percentage of the target class members that the model returns up until decile X[6]. Along with this curve, the plot also visualizes a perfect curve that represent a theoretical best model performance and a baseline representing random guessing.

3 | CREDIT CARD DATA

As mentioned, the study is conducted on a data set for credit card clients in Taiwan over several months in 2005. All 24 of the attributes in the data-set are described in Table 1, and the goal of the analysis is to predict whether or not the client defaults the payment in June 2005. With 24 attributes or features, the data-set thus contain 24 columns, in addition to a final column containing target values of "default payment next month". In its raw form, the data consists of 30000 samples.

3.1 | Pre-processing

Before assessing the models and their performance, the quality of the data-set needs investigation. By inspection, there are some entries where either all past bill statements (BILL_AMTX) or all pay amounts (PAY _AMTX) are zero. These entries make no sense and are therefore removed from the dataset. Further, by inspecting Figure 2, it is obvious that the data in the PAY_X columns contain some undefined categories, namely -2 and 0. The entries containing -2

TABLE 1 Results Terrain

Column	Description	Possible Values
LIMIT_BAL	Amount of given credit in NT Dollars	Numerical
SEX	Gender of client	Categorical: 1 = male, 2 = female
EDUCATION	Client's level of education	Categorical: 1 = graduate school, 2 = university, 3 = high school, 4 = others
MARRIAGE	Maritual status of client	Categorical: 1 = married; 2 = single; 3 = others
AGE	Client's age	Numerical
PAY_0 .. PAY_6	Payment status from April(PAY_6) to September(PAY_0) 2005.	Categorical: -1 = Duly pay, 1 = 1 month delay,..., 9 = payment delayed for 9 months and above.
BILL_AMT1 .. BILL_AMT6	Amount on bill statements per month	Numerical
PAY_AMT1 .. PAY_AMT6	Amount paid per month	Numerical
default payment next month	Payment default June 2005	Categorical: 1 = yes, 0 = no

will be removed, while the number of entries containing 0 is significantly large so these will be kept in the dataset as an NA-class.

Figures 3 and 4 show the data for BILL_AMTX and PAY_AMTX respectively. The black lines in the plots show the outlier limit based on the well-known "rule of thumb"[2] for what is considered as extreme values in data sets. These limits are defined as everything, 1.5 Interquartile ranges below the first quartile and above the third quartile. By observing the Figures, it is somewhat obvious that these limits would exclude too much data. Thus, a constants of 225000 and 200000 are added to the upper-limits of the bill amounts and pay amounts respectively. The new limit for outlier exclusion is illustrated by the blue line in the Figures. After excluding the data as described above, the dataset is reduced to 22911 samples. For all the analysis, the numerical features are scaled using numpy's StandardScaler[7]. Prior to initiation of the models used, data is split into training and test sets. The training set make up 80% of the data and is used to train the models. The remaining 20% of the data is used to validate and evaluate the models.
Figure 5 depicts the target values in the dataset. By observing the Figure, it becomes clear that the data set is biased towards non-default (0). To account for this, the exercises will be conducted on a dataset where the training data is re-sampled so that 50% of the data is predicting default, for comparison.

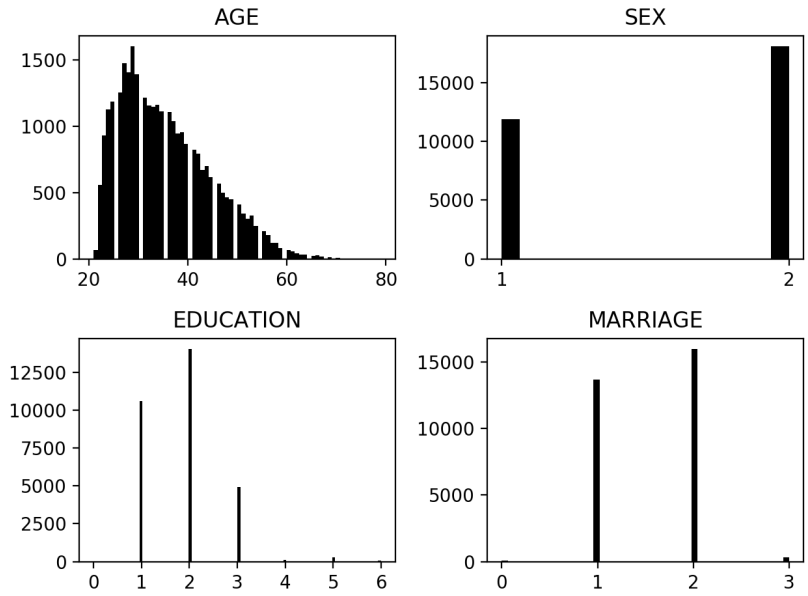

FIGURE 1 Demographical features

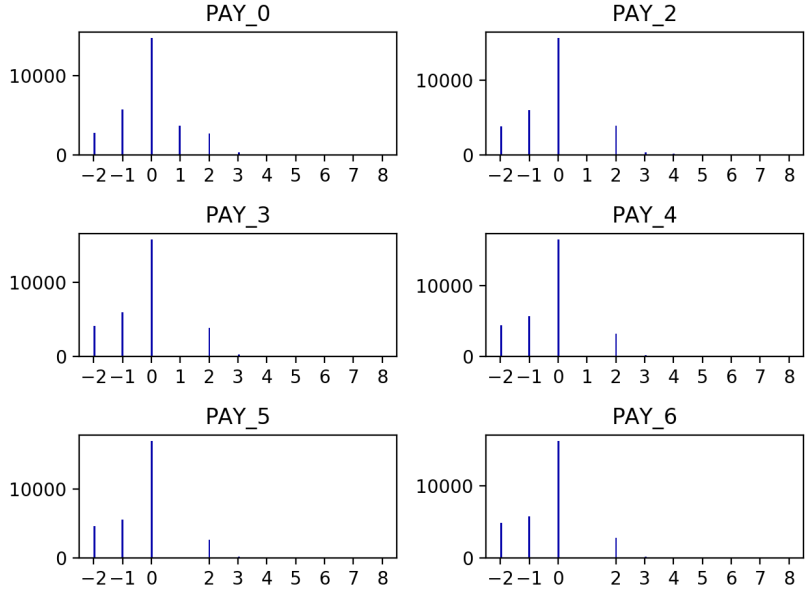

FIGURE 2 Pay status data

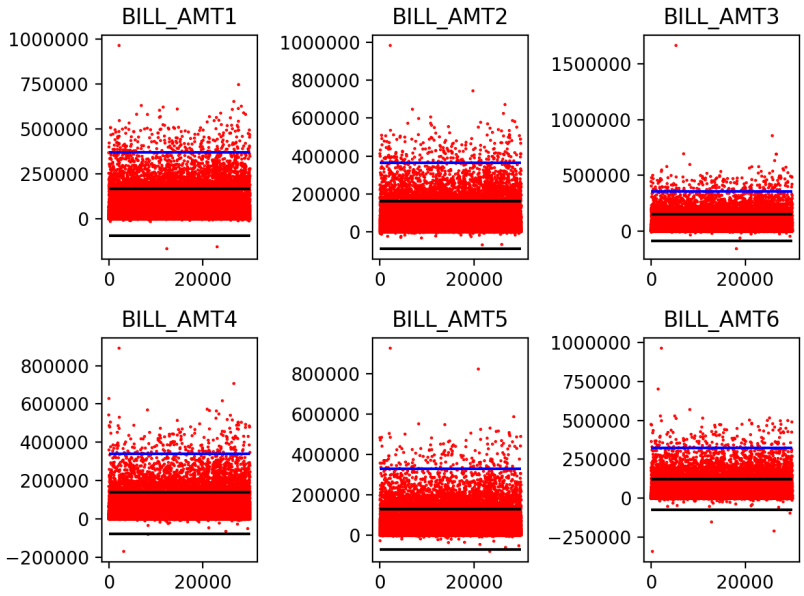

FIGURE 3 Scatter plot of bill amounts

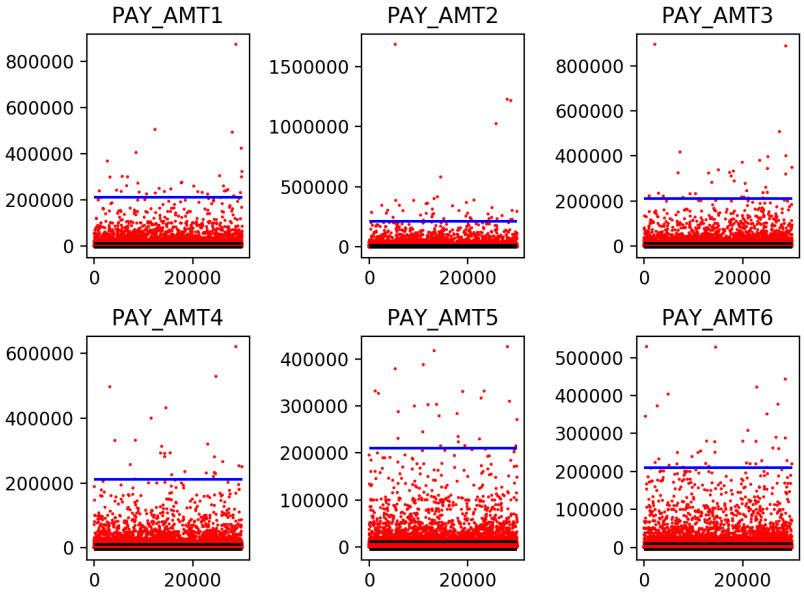

FIGURE 4 Scatter plot of pay amounts

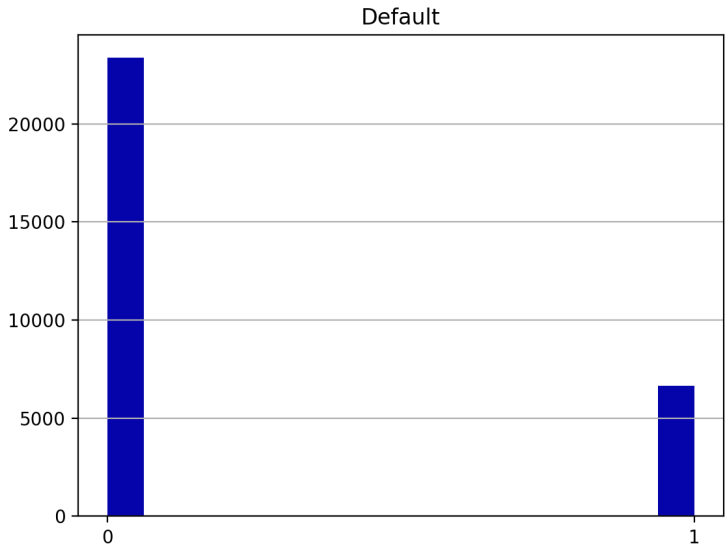

FIGURE 5 Target Values

4 | RESULTS

4.1 | Grid Search

A grid search is a method to refine the model by performing multiple runs over different regularization constants, λ, and learning rates, η. The routine hence measures the accuracy of predictions created from logistic regression and neural networks as showcased in Figures 6 and 12. Both Figures are created by varying λ and η in log-spaces ranging from 10^{-5} to 100 and from 10^{-5} to 100, respectively.

4.2 | Logistic regression

Figures 6 and 7 show results from grid-searching the logistic regression model. By inspection of the latter Figure, optimal learning rate η ranges from 0.001 to 0.01. In this region, the accuracy is also seemingly less dependent on λ. The maximum accuracy for SGD with 50 epochs and batch-size of 100, is 0.828 for η = 0.1 and λ = 0.001. Correspondingly for original gradient descent with 1000 iterations, result are not that different giving the same max accuracy for η = 0.01 and $\lambda = 10^{-5} - 0.1$.

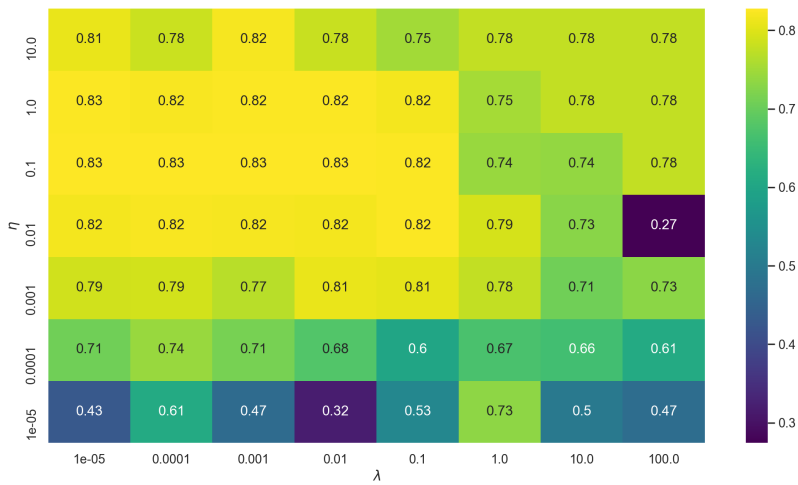

FIGURE 6 Grid search using Stochastic gradient descent

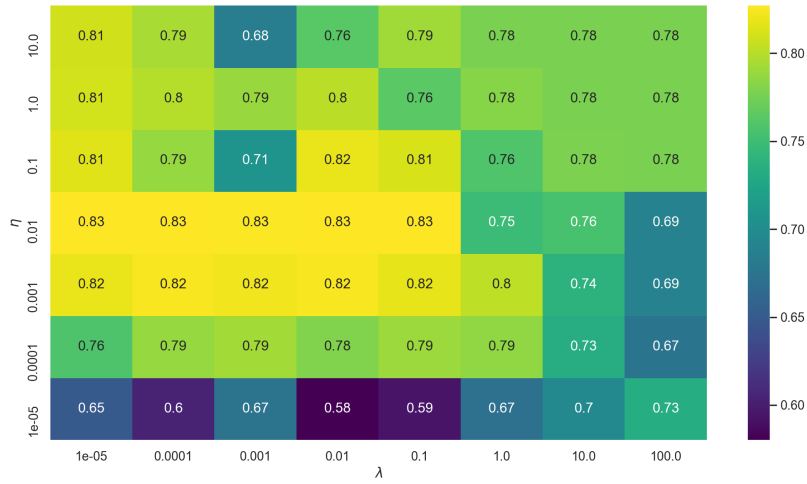

FIGURE 7 Grid search using gradient descent

Results for NR were not obtained, as collinearity in the inputs made it impossible to compute $\left(\frac{\partial^2 C(\hat{\beta})}{\partial\hat{\beta}\partial\hat{\beta}^T}\right)^{-1}$. A Pearson correlation matrix of $\left(\frac{\partial^2 C(\hat{\beta})}{\partial\hat{\beta}\partial\hat{\beta}^T}\right)$ is depicted in Figure 8. Note that there are many features that have correlation coefficients above 0.7 which is considered as significant.

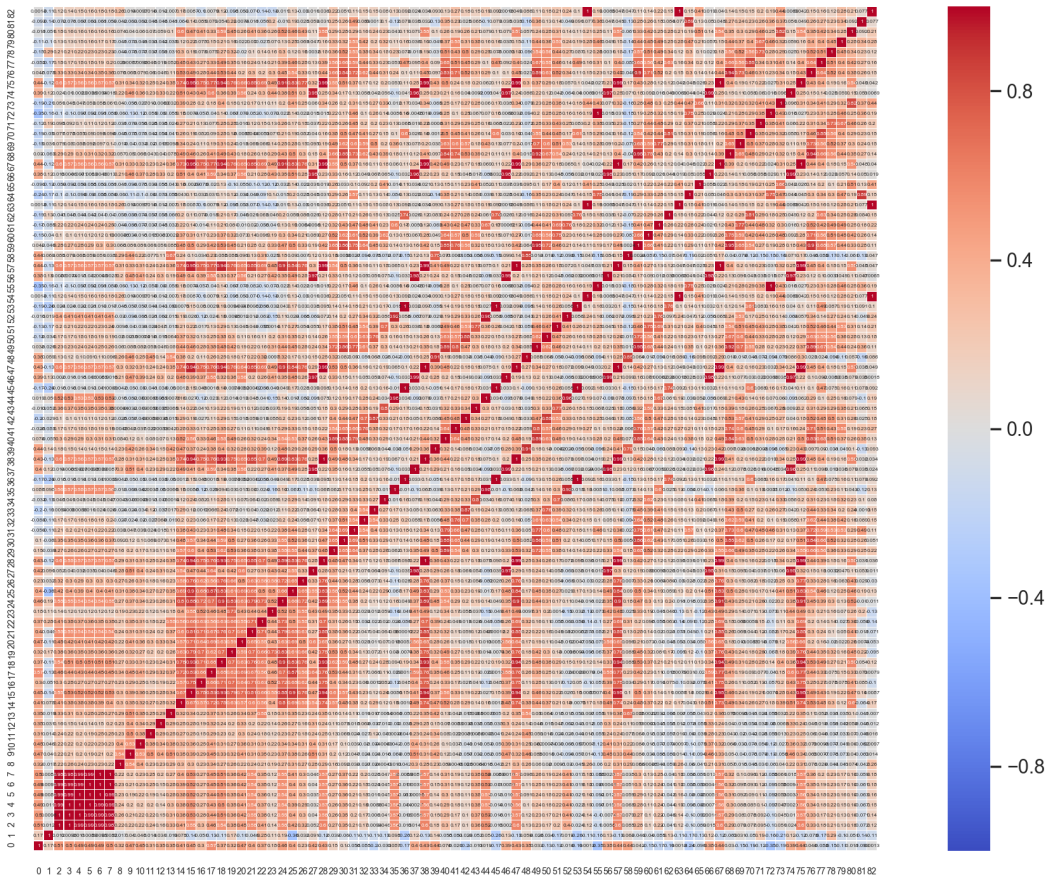

FIGURE 8 Pearson Correlation Matrix for $\left(\frac{\partial^2 C(\hat{\beta})}{\partial \hat{\beta} \partial \hat{\beta}^T}\right)$

Re-sampling the data reduced the accuracies slightly as shown in Figures 9 and 10, giving best result of 0.797 and 0.813, respectively. The tuning-parameters yielding maximum accuracy are also somewhat offset.

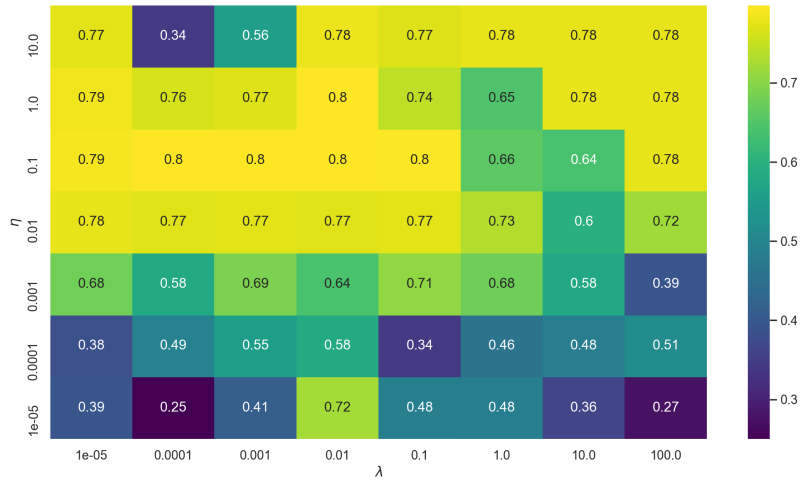

FIGURE 9 Grid search using stochastic gradient descent on re-sampled training data

FIGURE 10 Grid search using gradient descent on re-sampled training data

4.3 | Neural Network

After some test and trial, the ANN was initiated with a structure of 82 inputs, 2 hidden layers of 70 and 50 neurons, in addition to an output of 1 neuron. For regression analysis, the default sigmoid activation function in the output layer was alternated to either ReLU or linear. It should. Outputs of the "home-made" ANN are compared to outputs of a Neural Network built with Tensorflow/Keras. Epochs and batchsizes are the same as for SGD in log-reg.

4.3.1 | Binary Classification - Credit Card Data

Results on credit card data in its original form are displayed in Figure 12. For the self produced ANN, max accuracy 0.825 for λ = 0.01 and η = 0.01. This optimal result is similar to the one from Keras, although the model seems to be more sensitive to regularization and learning rate. This is probably due to different initiation of the weights and biases i.e by use of Xavier initialization [1].

FIGURE 11 Grid search using Neural Network

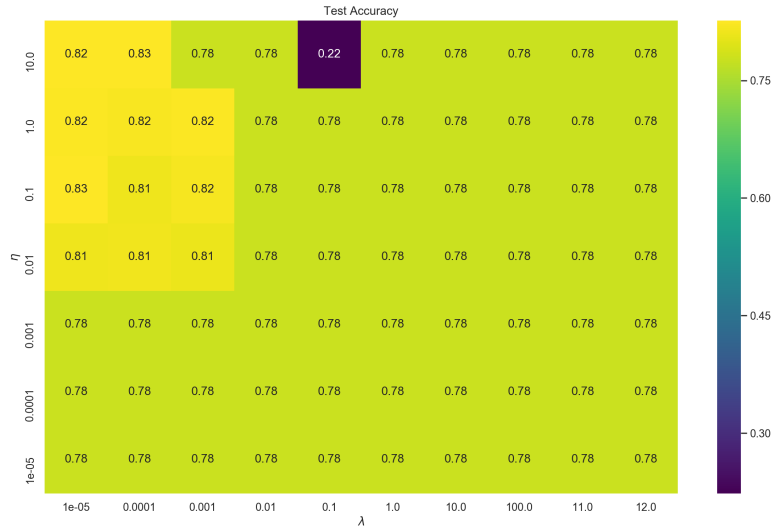

FIGURE 12 Grid search using Keras Neural Network

Figure 14 show the results from Neural Network on re-sampled data. The maximum accuracy in this case is again slightly lower, at 0.82. Compared to Keras, model results are satisfactory.

FIGURE 13 Grid search using Neural Network on re-sampled training data

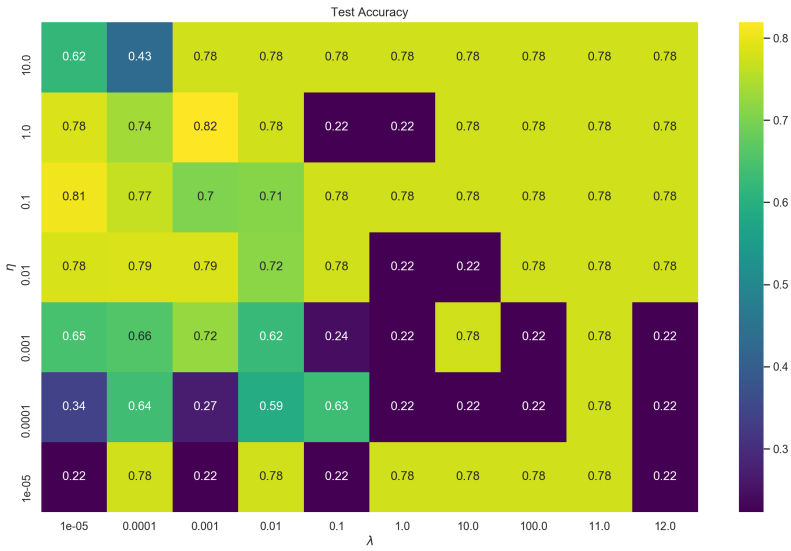

FIGURE 14 Grid search using Keras Neural Network on re-sampled training data

4.4 | Comparison

All methods gave comparable results in terms of accuracy, however, the time the algorithms used to process the data was very different. Figure 15 shows the accuracy obtained by using each method in addition to the elapsed time of the code. It is clear that for this case, the SGD method was by far the quickest.

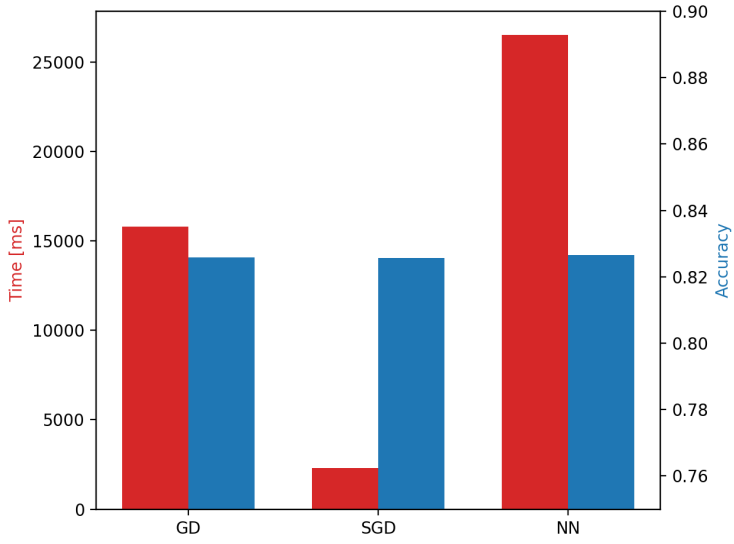

FIGURE 15 Time and accuracy of the methods

Further evaluation of cumulative gains plots and confusion matrices in Figure 17 to 27, give a better perspective of how good the models are to predict the different classes. For the cumulative gains charts, particularly in the re-sampled case, there is an evidently poorer performance of gradient descent as the curve does not bend that much towards the green best model curve. Distinguishing between ANN and SGD by these plots on the other hand is not easy, so quantifying performance with confusion matrices is a better way. As the prediction of defaulters is the center of attention, (1,1) and the corresponding number yields best performance of ANN. Also notably is the gain of this value for all the models when the training data is re-sampled to ensure that 50% of the target values are default.

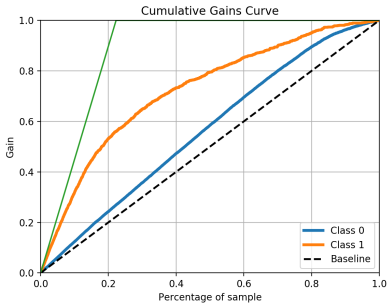

FIGURE 16 Stochastic Gradient Descent

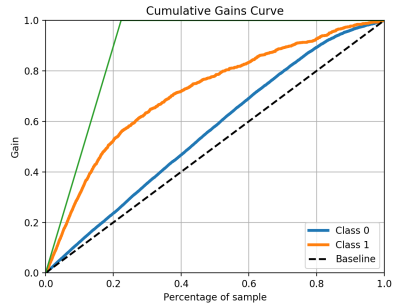

FIGURE 17 Stochastic Gradient Descent re-sampled

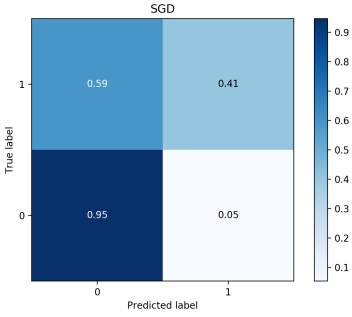

FIGURE 18 Confusion Matrix

FIGURE 19 Confusion Matrix

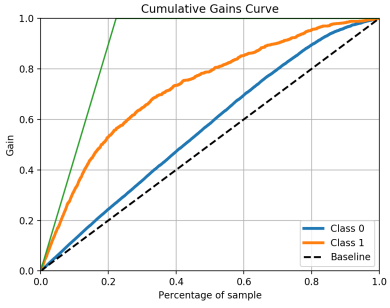

FIGURE 20 Gradient Descent

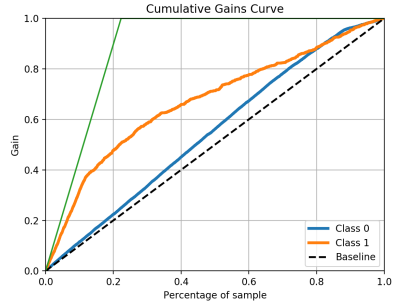

FIGURE 21 Gradient Descent re-sampled

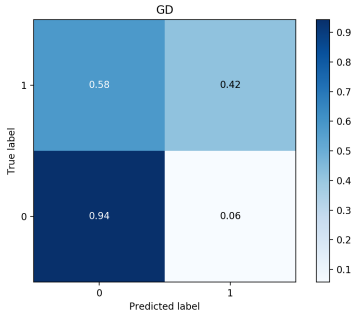

FIGURE 22 Confusion Matrix

FIGURE 23 Confusion Matrix

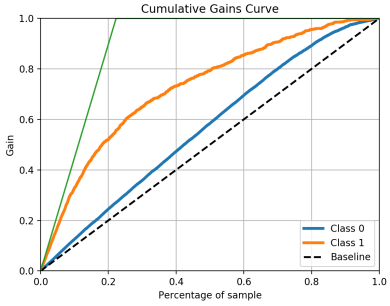

FIGURE 24 Neural Network

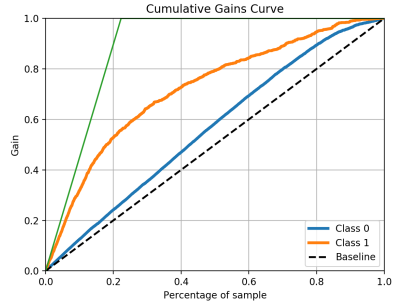

FIGURE 25 Neural Network re-sampled

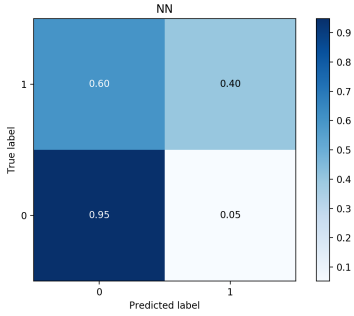

FIGURE 26 Confusion Matrix

FIGURE 27 Confusion Matrix

4.4.1 | Regression on Noisy Franke Function

Below are results from applying the ANN (2 hidden layers with 50 neurons), to the Franke Function with added noise in the form of MSE and R2-score. Carefully note that the color-range for MSE is reversed. For a combination of sigmoid activation functions in the hidden layers and linear in the output, the highest scores are $MSE = 0.013$ and $R2 = 0.83$. A combination of sigmoid and ReLU yields the same results.

FIGURE 28 ANN MSE Sigmoid-Linear, 2 hidden layers of 50 neurons

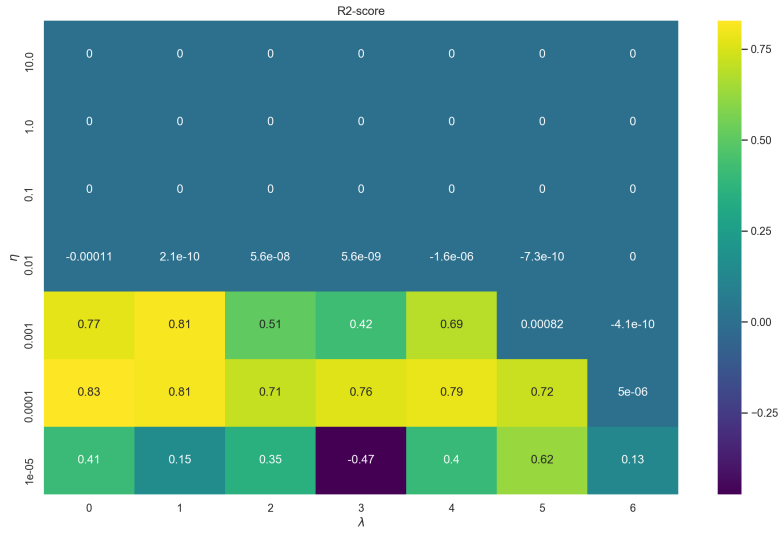

FIGURE 29 ANN R2-score Sigmoid-Linear, 2 hidden layers of 50 neurons

FIGURE 30 ANN MSE Sigmoid-ReLU, 2 hidden layers of 50 neurons

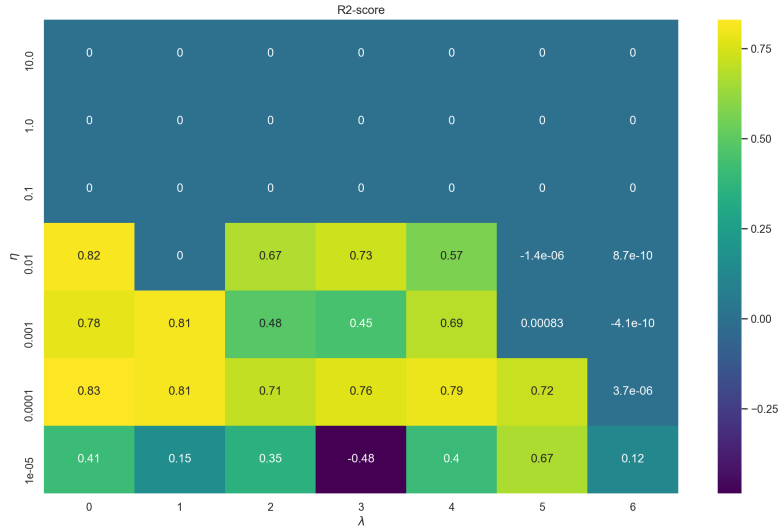

FIGURE 31 ANN R2-score Sigmoid-ReLU, 2 hidden layers of 50 neurons

Increasing the amount of neurons in the hidden layers to 500 and 100, improves the model performance. This is evident by Figures 32 to 34, showing most optimal results of $MSE = 0.013$, $R2 = 0.87$ and $MSE = 0.0096$, $R2 = 0.87$, respectively for final activation function of linear and ReLU.

FIGURE 32 ANN MSE Sigmoid-Linear, 2 hidden layers of 500 and 100 neurons respectively

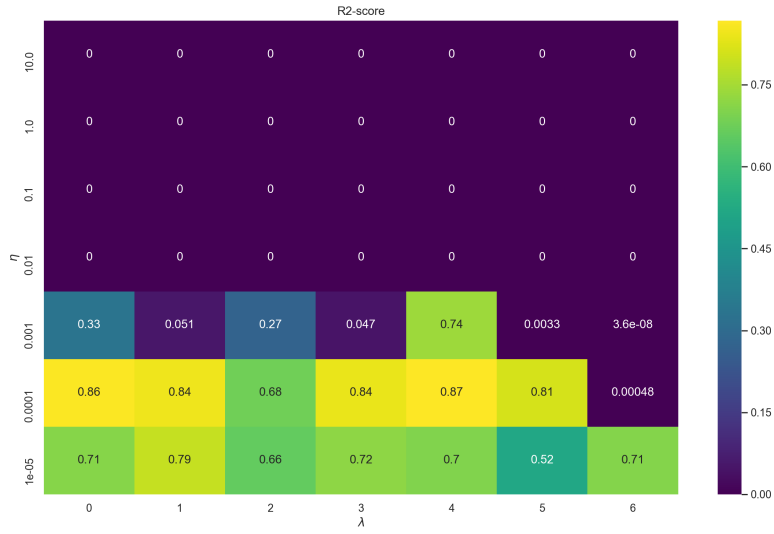

FIGURE 33 ANN R2-score Sigmoid-Linear, 2 hidden layers of 500 and 100 neurons respectively

FIGURE 34 ANN MSE Sigmoid-ReLU,2 hidden layers of 500 and 100 neurons respectively

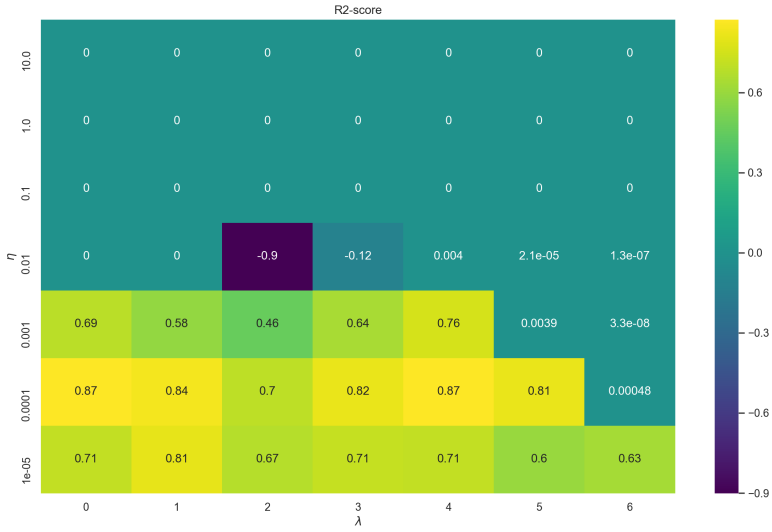

FIGURE 35 ANN R2-score Sigmoid-ReLU,2 hidden layers of 500 and 100 neurons respectively

The overall performance of the ANN is very close to linear regression models as evident by table 2, taken from a previous study by the authors. Compared to the best linear regression model, namely Ridge, results are almost identical with only a slight reduction in R2-score.

TABLE 2 Results Franke Function

Method	MSE	R2
OLS(Deg = 4)	0.0107	0.8698
Ridge(Deg = 5, $\lambda = 10^{-3}$)	0.0096	0.8836
Lasso (Deg = 3, $\lambda = 10^{-4}$)	0.0103	0.8739

5 | CONCLUSION

This study maps model performance of logistic regression and artificial neural networks on credit card data from Taiwan obtained in 2005. The outcome of the study is in accordance with Yeh and Lien [8] that ANN give the most accurate predictions and targets defaulted customers best. Results of the self-built model are also adequate compared to those of the Keras model.

In terms of logistic regression with a NR-solver for cross-entropy cost optimization, results were not obtained as the data showed too much colinearity. Generally as the data is biased, re-sampling gave reduced accuracies, while increasing each model's ability to predict default.

Finally, ANN also show good performance on regression fitting with results similar to those of linear-regression. Changing the activation function slightly increased the scores as the model became less exposed to noise.

It should be noted that it is difficult to surely determine the optimal number of layers and hidden neurons, as optimization can become very time-consuming. Thus in a more detailed study with an increased time-frame, one could asses a sensitivity analysis on layering with cross-validation to pursue even more rigorous model outputs. Another point of interest could also be to evaluate other activation functions and implementation of different weight initialization techniques such as the Xavier method.

references

1. Glorot, X. and Bengio, Y. (2010) Understanding the difficulty of training deep feedforward neural networks. In *In Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS'10). Society for Artificial Intelligence and Statistics*.

2. Han, J. and Kamber, M. (2006) *Data Mining: Concepts and Techniques*. 500 Sansome Street, Suite 400, San Francisco, CA 94111: Morgan Kaufmann Publishers, second edn.

3. Hastie, T., Tibshirani, R. and Friedman, J. H. (2009) *The Elements of Statistical Learning: Data Mining, Inference, and Prediction, 2nd Edition*. Springer Series in Statistics. Springer. URL: https://doi.org/10.1007/978-0-387-84858-7.

4. Hjorth-Jensen, M. (2019) Lecture notes fys-stk4155.

5. LAKE, R. (2019) Why is credit card delinquency rising as interest rates fall? https://www.investopedia.com/credit-card-delinquency-rises-even-as-interest-rates-fall-4764123. Accessed: 2019-10-28.

6. Marcus, P. and Nagelkerke, J. (2018) Plot information (modelplotpy). https://modelplot.github.io/intro_modelplotpy.html. Accessed: 2019-11-09.

7. Scikit (Year Not specified) Scikit standardscaler. https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html. Accessed: 2019-11-09.

8. Yeh, I. and Lien, C.-H. (2009) The comparisons of data mining techniques for the predictive accuracy of probability of default of credit card clients. *Expert Systems with Applications*, **36**, 2473–2480.