Classification and Regression, from linear and logistic regression to neural networks

Johan Nereng

Department of Physics, University of Oslo, Norway

Nov 13, 2019

Abstract

This project's main focus is comparing and evaluating logistic regression and neural networks in qualitative prediction, where the project found that the two methods predicted with similar quality (area ratio scores ~ 0.44) on the Taiwan credit card data, using one hidden layer in the neural network. Cursory examination of the connection between the number of epochs, batch size, learning rate (η), and regularization (λ) used in the stochastic gradient descent (SGD) in the back-propagation of the neural network, suggests that all four parameters are linked and must be found through trial and error for each unique data set - but that λ and the number epochs appeared to be the strongest connected parameters. In addition, this project examines the effects of data pre-processing on prediction accuracy and gains, concluding that accuracy is unsuited as a measurement of prediction quality in categorical modeling for skewed data sets (accuracy 0.78 on the same predictions that gave an area ratio score= 0.00), and that scaling had the largest effect on raising prediction quality of all the pre-processing methods examined in this project. Lastly, this project compares OLS and neural networks in quantitative regression on data generated using Franke's function, where the project found that the OLS (MSE $\sim 10^{-30}$) outperformed the single layer neural network (MSE $\sim 10^{-5}$), and as such appear to be better suited. The project was unable to ascertain a more firm conclusion due to faulty implementation of the neural network program intended to handle additional hidden layers.

Authors' comment: *Writing this project has taught me a lot about neural networks, and classification in general. Having spent the most amount of time I have ever done on any project, I've reached the conclusion that I cannot correctly code and understand all the methods in a vacuum. As such, I've decided that I need to spend more time at CCSE, and less at my home office, so that I may discuss theory, code, and results with co-students. This was especially true in the case of neural networks, were there are fewer "right" answers, and more trial and error, such as the trick dividing the weights gradient by batch size, or choosing sensible values for the parameters. I especially struggled to make the the neural network with multiple layers work, even with the extended deadline, as I was already somewhat behind when it came to finishing the report. In the end I had to give up on using the more complex program I had written, and rely on the object oriented program from the lecture notes. I wish I had the time to implement PCA, and work at the multi-layered NN until I had matched the results from the scientific paper. For future project I would like to limit the length of my reports, both for my own sake and for the sake of the reader.*

1 Introduction

Classifiers, such as logistic regression, models the probability that a data entry belongs in a certain category. Due to this, the regression methods used for numerical output values are ill suited. If one insist on using for example least squares in categorical prediction, one runs into the problem of ordering of responses and category gaps (explained eloquently and in detail in [3][p.130]). Neural networks are flexible in this regard. The basic structure of the network remains the same for any type of regression - the only changes required are the activation functions which determine if a neuron activates, and how the prediction quality is measured (the cost function). When training neural network, logistic classifiers, and a variety of other regressors, adjusting the weights or regression coefficients require using minimization techniques. Where analytic expressions are not viable or available, gradient descent (GD) methods are often applied. One such method, the stochastic gradient descent (SGD) is much used for neural networks, and the parameters which decide the trajectory of the cost function are numerous and influence each other. Because of this, successful prediction using neural networks require practical experience in data analysis, as well as insights into the problem at hand.

Credit card data: The data set which this project mainly uses is the Taiwan credit card data [10], which has 23 predictors and a binary response - default payment on credit card or not default payment. Of the 30000 data entries, only 22% of them are default payment, making the data set skewed. This skewness has implications on training the classifiers, as they easily tend to favor the most numerous category. Pre-processing of the data, such as up-sampling of the default category in training data, may therefor increase prediction quality. The different predictors in the data set also have a large range of values, where some are $\sim 10^0$, others are $\sim 10^6$, which further complicates modeling. Because of this, scaling of the design matrix vastly improves the resulting predictions for both neural networks and logistic regression.

Project structure: This project starts by giving a brief explanation of logistic regression, and how it may be used to predict the probability of a response falling within a category. In order to fit a logistic regression model, and later also a neural network, the project moves on to the gradient descent method for minimizing cost functions. Then, in order to build neural networks for both qualitative and quantitative response variables, the project looks into neural networks in general, connecting the training of the network to gradient descent, and explaining in brief how the network may be trained using feed forward and back-propagation, followed by a brief description of a selection of data pre-processing. After verifying the integrity of the logistic regression implementation and neural network implementation with one hidden layer, the project applies the implementations (logreg, NN) on the Taiwan credit card data, followed by application of neural networks on quantitative responses. The next section

covers the project's results, followed by a discussion of the results. Based on the discussion, the project's **main findings**, which are summarized in the conclusion of the project, are presented.

Main findings: The project finds, after comparing gains charts, confusion matrices, area ratio scores, and accuracy, that accuracy is unsuited as a measurement of prediction quality in categorical modeling for skewed data sets. The project also found scaling of predictor to greatly increase prediction quality, which in the end gave the neural network an area ratio score (0.43) (using $\eta = 0.1$,$\lambda = 1.0$, with 10000 epochs (SGD) and $batchsize = 1000$) comparable with the logistic regression (0.46) classifier. Based on this, the project deemed it likely that a single hidden layer is insufficiently complex to achieve optimal prediction. The examination of the four parameter learning rate η, regularization λ, epochs (SGD) and batch size, revealed close linkage, suggested that practice, trial and error, and knowledge of the data set is crucial when using neural networks. The strongest connection appeared to be between regularization and the number epochs. Lastly, the project found that for predicting quantitative targets, neural networks (MSE $\sim 10^{-5}$)of one layer appear to under-perform significantly when compared with ordinary least squares (MSE $\sim 10^{-30}$). The faulty implementation of the more complex neural network algorithm for multiple hidden layers unfortunately meant that no general conclusions could be reached with regards to neural networks of more than one hidden layer. The project does however suggest, pending further examination, that neural networks are best suited for categorization, and perhaps in some cases, too complex for easy implementation in numerical regression.

Tools used in the project include: Python 3.7.5, NumPy [8], scikit-learn [9] and imbalanced-learn [4], as well as a number of books, web-pages and articles - of which most are listed under references. All the code required to reproduce the results may be found in the author's github page .

2 Methods

2.1 Logistic Regression for two classes

Logistic regression models the probability that a qualitative response falls within a category. In the case of two mutually exclusive categories or classes, such as "True" ($Y = 1$) and "False" ($Y = 0$), this translates to modeling the relationship between the probability of $Y = 1$ given X as $P(Y = 1|X; \beta) = p(X; \beta)$, using the *logistic function* (1) [3][p.132], which due to mutual exclusivity, models $Y = 0$ as a function of X, or $P(Y = 0|X) = p(X'; \beta)$, as $1 - p(X; \beta)$.

$$p(Y = 1|X, \boldsymbol{\beta}) = p(\boldsymbol{X}; \beta) = \frac{e^{\boldsymbol{\beta X}}}{1 + e^{\boldsymbol{\beta X}}} \tag{1}$$

where $\boldsymbol{\beta} = [\beta_0, \beta_1, ...\beta_p]$, $\boldsymbol{X} = [1, X_1, X_2, ...X_p]$, and p is the number of predictors that the the binary response Y is modeled upon. When fitting the model (estimating the β coefficients) to a data set $\mathcal{D} = \{y_i, \boldsymbol{x}_i\}_{i=1}^n$, of n observations, the desired fit is the one where the predicted probabilities of $p(\boldsymbol{x}_i)$ most precisely corresponds to y_i. In other words, one seeks the model which has the highest probability of predicting the data set, which is achieved by utilizing the *maximum likelihood* method (2)[3][p.133].

$$P(\boldsymbol{\beta}) = \prod_{i:y_i=1} p(\boldsymbol{x}_i; \boldsymbol{\beta}) \prod_{i':y_{i'}=0} (1 - p(\boldsymbol{x}_{i'}; \boldsymbol{\beta})) \tag{2}$$

Taking the log of the maximum likelihood, one obtains the log-likelihood. The (reordered) negative log-likelihood constitutes the cost function, $\mathcal{C}(\hat{\beta})$ (3) [1][p.120] (in Hastie et al, the log-likelihood is maximized) of the logistic regression, or the *cross entropy*. When minimized, this convex function yields the desired β estimates.

$$\mathcal{C}(\boldsymbol{\beta}) = -\sum_{i=1}^{n} \left(y_i(\boldsymbol{\beta}\boldsymbol{x}_i) - log(1 + e^{\boldsymbol{\beta}\boldsymbol{x}_i})\right) \tag{3}$$

2.2 Gradient Decent

The minimization of a cost function may in some cases be carried out analytically, for example when using least squares error. For some cost function however, the analytic expression may be inconvenient to use. Factors such as the size of a matrix which requires inverting may be to computationally intensive, and in yet other cases, close form solutions are entirely unavailable. For problems such as these, numerical methods for obtaining the desired model fit is instead applied. One such method is the gradient descent (GD) (4) [5][p.247], also known as steepest descent. When utilizing GD, one makes an initial guess for $\boldsymbol{\beta}$, evaluates the gradient, $\boldsymbol{g}$ of the cost function in order to obtain an estimate closer to the desired minimizing value. This process is then iteratively repeated till the sufficient convergence of the coefficients is reached. In order to not overshoot the $k+1$'th estimate, a *learning rate* of η is applied to the gradient - which may also serve to lengthen the step in an iteration. The value of the learning rate can either be fixed, usually by trial and error, such that it works well for the problem at hand, or it may be adapted to each step.

$$\boldsymbol{\beta}_{k+1} = \boldsymbol{\beta}_k - \eta_k \boldsymbol{g}(\boldsymbol{\beta}_k) \tag{4}$$

A regularization parameter, λ, is often added to the gradient in an extra term, effectively modifying the gradient descent (and SGD), which effectively reduces the growth of the weights, thus reducing overfitting to the training data.

$$\boldsymbol{\beta}_{k+1} = \boldsymbol{\beta}_k - \eta_k \boldsymbol{g}(\boldsymbol{\beta}_k)(1 + \lambda) \tag{5}$$

For logistic regression, $\boldsymbol{g}$, may be expressed as: [5][p.247]:

$$\boldsymbol{g}(\boldsymbol{\beta}) = \frac{\partial \mathcal{C}(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}} = -\hat{X}^T(\boldsymbol{y} - \boldsymbol{p}) \tag{6}$$

Where $\hat{X}$, is the design matrix, and $\boldsymbol{p}$ a vector with elements $p(y_i|x_i;\boldsymbol{\beta})$.

A common improvement on GD is the stochastic gradient descent (SGD), which involves splitting the data into min batches, then randomly selecting one batch at a time from which to compute the gradient. More detailed information is available in literature such as Kevin P. Murphy's book on machine learning [5][p.262].

2.3 Neural Networks

For simplicity's sake, matrices are denoted without bold font or hats from here on forwards. The feed forward neural network (FFNN), or multi-layer perceptron model (MLP), is an artificial neuron network consisting of an ordered series of regression models with connected inputs and outputs [5][p.563]. The network consists of L layers, with a number of neurons (regression models), or nodes, in each layer. The neurons are not necessarily of the same type, as one can for example use logistic regression models for all nodes but the ones in L, which could be linear regression models if the problem is a regression problem. The outputs, a^1, of the M_1 nodes in layer $l = 1$ (one), the input layer, are fed to the nodes in layer $l = 2$, and so on, until the outputs, a^{L-1}, from layer $l = L - 1$ is fed to layer $l = L$, the output layer. Outputs from layer L, a^L are the network's outputs, and predict the target(s) of the model. The layers in-between layer 1 and layer L, are deterministic functions of the input, and are called hidden layers. Each node in each layer uses weights for each input, similar to the coefficients corresponding to the predictors described in the logistic regression subsection. In addition, each node has it's own *bias*, b, which regulates the output tendency of neuron. The sum of the weighted inputs and the bias, z, is usually called the activation of the neuron, not to be confused with it's output. The output is however a function of the activation, such that $a^l = f^l(z^l)$, where $f(z)$ is called an activation function.

The remaining subsections on neural networks are based on the informative and well written book by Michael Nielsen book [7] (using a slightly different matrix indexation than what this project does), and lectures notes in FYS-STK4155 [2].

2.3.1 Feed forward

The propagation of neuron outputs in the network, resulting in prediction, is called feed forward. The network used in this project is set up such that it takes n_{inputs} observations, of $n_{predictors}$ predictors, thus working through the entirety of the data in one go. As with other regression methods, the design matrix X ($n_{inputs} \times n_{predictors}$) is the input, which in the end outputs a matrix

y ($n_{inputs} \times n_{targets}$) of predicted values for the $n_{targets}$ targets. By arranging the weights of layer l, w^l ($M_{l-1} \times M_l$), the activations of layer l is calculated by $z_l = Xw^l$ ($n_{inputs} \times M_l$). Taking $a^l = f^l(z^l)$ ($n_{inputs} \times M_l$), means that a^l holds the output for every neuron in l.

2.3.2 Cost function and backpropagation

Once a^L is obtained through the feed forward process, the network has made a prediction on the target(s), y, for each of the n_{inputs} observations. However, unless one has divine insights, initiating the weights and biases such that $a^L \approx y$ with sufficient accuracy is unlikely. The quality of these predictions is quantified by the cost function $C(w, b)$, which as indicated is a function of the weights and biases of network. The effect on prediction quality when adjusting w and b is thus measurable by the cost function, which, as in the case of the logistic regression, is to be minimized. In the case of FFNNs, this is done through backpropagation, which is the process of calculating the error of each layer, (7) and (8), and the gradient of the cost, ∇C, function w.r.t the biases (9) and weights (10).

$$\delta_L = \nabla_a C \odot f'(z^L) \tag{7}$$

$$\delta_l = \delta^{l+1}(w^{l+1})^T \odot f'(z^l) \tag{8}$$

$$\frac{\partial C}{\partial b^l} = \delta^l \tag{9}$$

$$\frac{\partial C}{\partial w^l} = (a^{l-1})^T \delta^l \tag{10}$$

Having obtained the gradient ∇C, gradient decent is applied in order to calculate new weights and biases. Once the weights and biases are updated, a new feed forward process is initiated, followed by backpropagation and corresponding adjustments of parameters until the cost function is sufficiently minimized or the number of iterations (feed forward + backpropagation) is reached.

2.3.3 Neural network for classification

In the logistic regression case, this project did not use one hot encoding - using cross entropy ensures equal importance of both target options when training. For the NN implementation however, one-hot encoding is used, along with cross entropy (3) as the cost function. As the one-hot encoding means that the network is expected to output two values, representing the likelihood of each of the two categories for each data point, the activation function of the output layer is the softmax function (11) (output of neuron $j = 1, 2$ in L), which normalizes the sum of the outputs to one.

$$a_j^L = \frac{e^{z_j^L}}{\sum e^{z^L}} \tag{11}$$

Using the cross entropy as the cost function, sigmoid for hidden layer(s), and softmax as the activation for layer L, means that the backpropagation equations (7)-(10) for the two last layers of network are:

$$\delta_L = a_L - y$$
$$\delta_{L-1} = \delta_L (w^L)^T \odot a_{L-1} \odot (1 - a_{L-1})$$
$$\frac{\partial C}{\partial b^L} = \delta^L$$
$$\frac{\partial C}{\partial w^L} = (a^{L-1})^T \delta^L$$
$$\frac{\partial C}{\partial b^{L-1}} = \delta^{L-1}$$
$$\frac{\partial C}{\partial w^{L-1}} = (a^{L-2})^T \delta^{L-1}$$

It's worth noting that in the output error, when using softmax with cross-entropy, the $f'(z)$ term cancels with parts of the derivative of C w.r.t a.

2.3.4 Neural network for regression

In this project, the neural network for regression is set up using the quadratic cost, with $tanh(z)$ as activation function in hidden layer(s), and with $f(z^L) = z^L$, the identity activation, as activation for the output layer. Due to the change in activation functions, the backpropagation changes slightly, however, as the cost function is also changed, the output error remains the same.

$$\delta_{L-1} = \delta_L (w^L)^T \odot (1 - tanh(z_{l-1}^2))$$

2.4 Implementation, pre-processing data, testing, and evaluation

Logistic regression for classification is implemented in it's own program using python 3.7.5. The program also uses functionalities from scikit-learn [9] for splitting data into test and training sets, metrics such as producing a confusion matrix, and scaling functions. A neural network for classification is also implemented, building on the template provided in the lecture slides on neural networks from the course FYS-STK4155 [2], using the activation functions and cost function as described in the subsection on neural networks for classification.

2.4.1 Pre-processing of data

As the credit card data (and to some extent the cancer data) is skewed (22% default), to some extent faulty, and in general not processed for regression, pre-processing of the data is required in order to make reasonable model predictions.

First, invalid entries in the categorical predictors are removed from the data all together - such as entries with a value corresponding to another marital status than married (1), single (2) or other (3)]. Then, the categorical predictors; gender, education, and marital status are one-hot encoded, as it is undesirable to train the model on the assumption that the options for each category have some "distance" between them, which is equal for say married (1) and single (2), and single(2) and other (3). As it is desirable to limit the number of predictors, the remaining categorical predictors are assumed to not suffer in the same extent from the equal-distance problem and are not one-hot encoded. Having removed some entries and reshaped the design matrix, it is then standardized (using scikit-learn's standard scaler) by substracting the mean of each predictor and scaling to unit variance. The data is then split into testing and training data, using a stratified split (scikit-learn), ensuring integrity of target category ratio in training and test sets. Lastly, the training data is up-sampled using Smote from imbalanced-learn [4] in order prevent the model from over-training on the over represented non-default case.

2.4.2 Model evaluation and quality measurements

In order to ascertain the quality of the classifiers in this project, accuracy score (12), area (ratio) score (13), and (intermittently), confusion matrixes are used.

$$\text{Accuracy} = \frac{\sum_{i=1}^{i=n} I(t_i = y_i)}{n} \tag{12}$$

Where I is the indicator function.

$$\text{Area ratio score} = \frac{\text{Area between model curve and baseline curve}}{\text{Area between optimal model curve and baseline curve}} \tag{13}$$

Where the curves in question are the curves obtained in a gains analysis.

2.4.3 Initial testing of the logistic regression implementation

Initial testing of the logistic regression implementation (using both GD and SGD) is carried out on data from the breast cancer set available through scikit-learn, which contains 569 data points with 30 predictors and targets (212 WDBC-Malignant, 357 WDBC-Benign). The program features an option to plot (set plot=True in the function call for GD/SGD) the cost function vs. number of iterations/epochs - which shows that the cost function decreases with iterations/epochs. Before training, the data set is scaled using scikit-learn's standard scaler (without any additional pre-processing), and split into test (1/3) and training (2/3) sets. In addition, scikit-learn's logistic regression model is fitted and used to confirm the integrity of the algorithm developed for this project. Based on comparison of area and accuracy scores as well as confusion matrices (see appendix 1), the implementation is deemed successful.

2.4.4 Initial testing of neural network implementation

A neural network with one hidden layer, with a number of nodes equal to the mean between input and output, is now tested on the same data set as when testing the logistic regression implementation, using both smote and scaling. The SGD in the neural network is set to use 1500 epochs with batch size 20, and $\eta = 0.1$, and $\lambda = 0.0$. This produced an area ratio score of 0.87 (where as the logistic regression program scored 0.97). The plot of the cost function (see appendix 1), shows that the network minimizes the cost function. Based on these indications, the implementation is deemed successful.

2.4.5 Logistic regression on credit card data

Having confirmed the integrity of the implementation, the project now moves on to a brief evaluation of the importance of pre-processing, while simultaneously establishing their connection to evaluation measurements (accuracy and area score) in skewed data sets when using logistic regression. In order to do so, the credit card data is loaded with outlier-removal and one-hot encoding, and fitted with the logistic regression program (using GD with iterations=5000, $\eta = 0.1$) without application of stratified split, smote and scaling. Similar fits, with scaling, with smote and scaling, and with smote, scaling and stratified split, is then carried out.

2.4.6 Neural network on credit card data

The neural network implementation is now used to predict on the credit card data. The conclusions reached under discussion of results on pre-processing from the logistic regression case it is assumed to be applicable for the neural network - although this has not been tested. The credit card data is loaded using outlier removal, one-hot encoding (also on the target in this case), scaling, and smote. As was the case under the initial test of the neural network, the number of nodes in the hidden layer corresponds to the average between the number of nodes in the input layer and the number nodes in the output layer - 16 (the credit card data, after one-hot encoding, has 29 predictors and 2 targets). First, a gridsearch over batch size and number of epochs for the SGD is carried out, using $\eta = 0.1$ and $\lambda = 0.1$, followed by an additional gridsearch using $\eta = 0.1$ and $\lambda = 1.0$. Then, in order to visualize the relationship between the four paramters (epochs, batch size, η, λ), an additional gridsearch over values for the regularization parameter λ and learning η is made, using 1000 epochs, with batch size 1000 . All gridsearches use area ratio score as their primary evaluation measurement, as the logistic regression section showed how misleading accuracy can be in the case of skewed data.

2.4.7 Neural network on Franke's Frunction

The neural network is modified in accordance with the back-propagation alterations described in the subsection on neural networks for regression, such that

the network (with one hidden layer) may be used for regression when the target has a numerical value. In order to evaluate it's performance, a 30×30 xy-grid for $x, y \in [0, 1]$ using Franke's function [6] to obtain targets, is generated and used to set up a design matrix corresponding to $p = 5$ predictors on the form $[x, y, xy, x^2, y^2, xy, ...]$. The predictors are then scaled to a range of $0, 1$, with no further pre-processing as most data pre-processing methods for classification do not apply. The network is first trained and tested on the same 30×30 grids, before being trained and tested on a split data set - it's MSE compared with that of the OLS model from scikit-learn. The regression is done using batch size 50 with 1000 epochs, and a grid search for η and λ, recording the lowest MSE.

3 Results

3.0.1 Logistic regression on credit card data

Figure 1 shows the gains chart produced by the logistic regression program on the credit card data with no other pre-processing of the data than one-hot encoding and removal of outliers (accuracy= 0.78, area ratio= 0.00 - row one table 1). As the plot suggests, and the area ratio score shows, the model performs similar to the random classifier.

Figure 2 shows the gains chart associated with the highest area ratio (0.46) from table 1. The figure indicates that the model significantly outperforms the random classifier, while not matching optimal model performance.

Table 1 shows the accuracy and area ratio score for logistic regression with GD (5000 iterations and $\eta = 0.1$) for different levels of pre-processing - showing that 0.44 area ratio outlier removal, one-hot, scaling, and smote produce the highest area ratio score of 0.46. In addition, the table suggests that scaling may be the most important pre-processing in the case of the credit card data.

Pre-processing	Accuracy score	Area ratio score
Outlier removal, one-hot	0.78	0.00
Outlier removal, one-hot, scaling	0.60	0.44
Outlier removal, one-hot, scaling, smote	0.58	0.46
Outlier removal, one-hot, scaling, smote, stratified split	0.57	0.44
Outlier removal, one-hot, smote, stratified split	0.78	-0.02

Table 1: Logistic regression on credit card data; accuracy and area ratio score as functions of different levels of pre-processing (GD with 5000 iterations and $\eta = 0.1$)

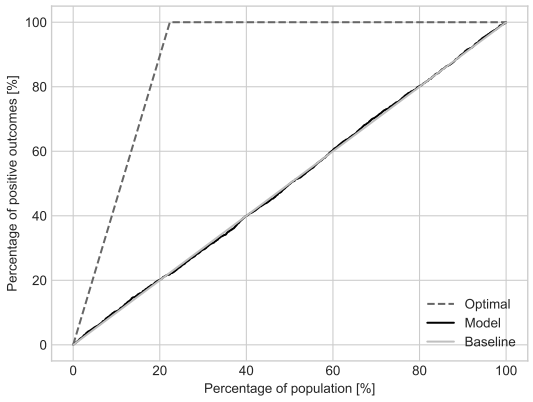

Figure 1: Gains plot for logistic regression using GD with 5000 iterations and $\eta = 0.1$ on credit card data, no pre-processing of data other than one-hot encoding.

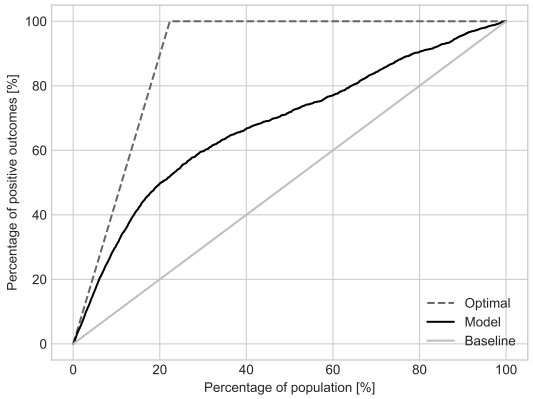

Figure 2: Gains plot for logistic regression using GD with 5000 iterations and $\eta = 0.1$ on credit card data, using outlier removal, one-hot, scaling and smote.

3.0.2 Neural network on credit card data

Figure 3 shows a gridsearch over epochs and batch sizes, for $\eta = 0.1$ and $\lambda = 0.1$, using outlier removal, one-hot encoding, scaling and smote. The figure indicates that 500 epochs and batch sizes of 500, and 1000 epochs and batch sizes of 1000, yield the best area ratio score (0.42).

Figure 4 shows a gridsearch over epochs and batch sizes, for $\eta = 0.1$ and $\lambda = 0.1$, using outlier removal, one-hot encoding, scaling and smote. The figure indicates that 500 epochs and batch sizes of 1500, and 1000 epochs and batch sizes of 2000, yield the best area ratio score (0.43).

Figure 5 shows a gridsearch over η and λ, using 1000 epochs over a batch size of 1000 using outlier removal, one-hot encoding, scaling and smote. The figure indicates that $\eta = 0.1$ and $\lambda = 0.001$, and $\eta = 0.1$ and $\lambda = 1.0$, yield the best area ratio score (0.43)e.

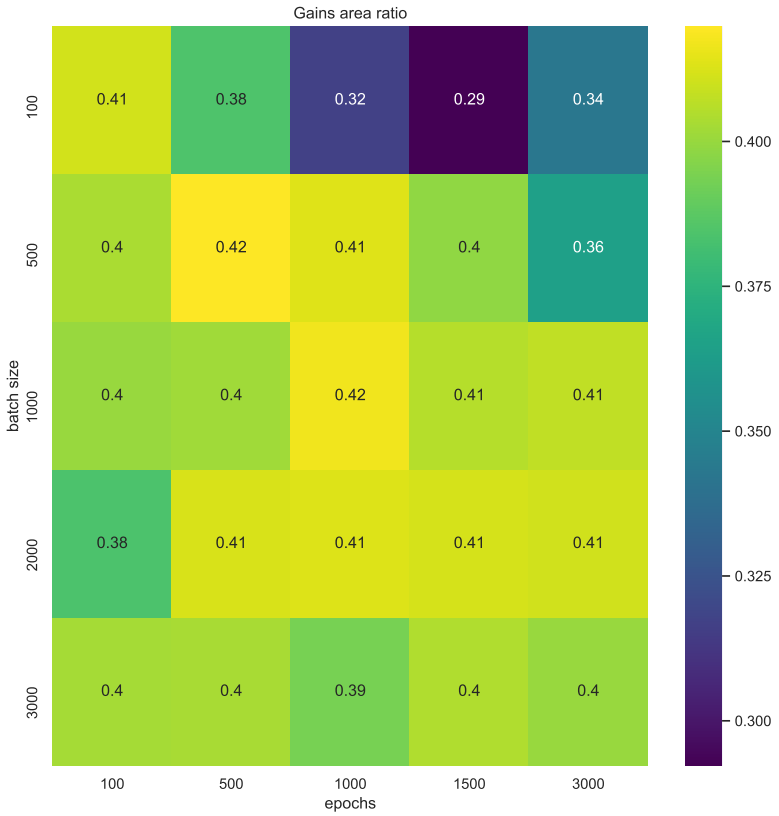

Figure 3: Gridsearch showing area ratio score for a number of epochs and batch sizes using values of $\eta = 0.1$ and $\lambda = 0.1$.

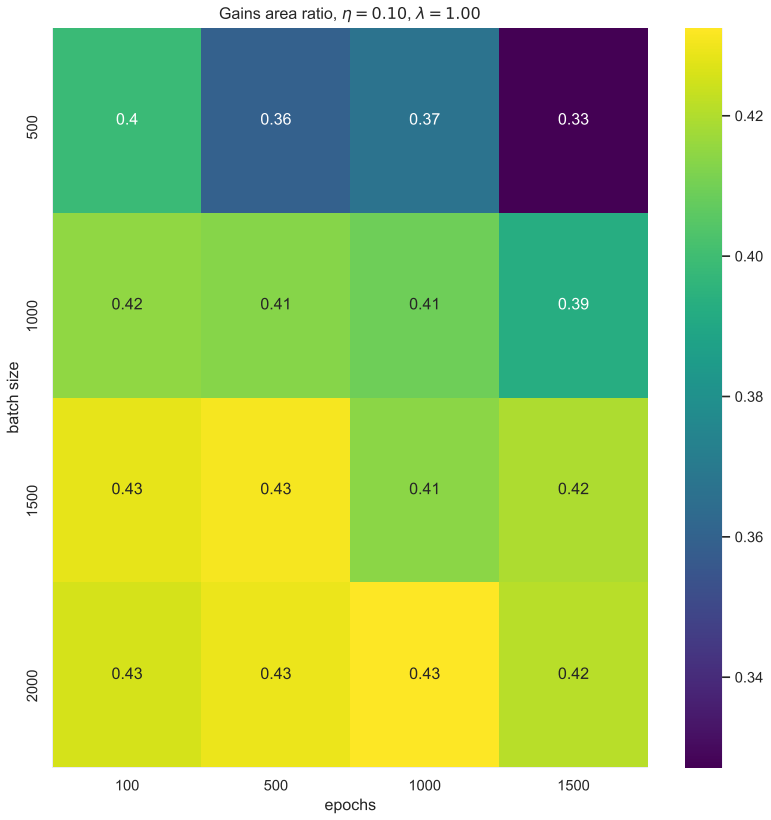

Figure 4: Gridsearch showing area ratio score for a number of epochs and batch sizes using values of $\eta = 0.1$ and $\lambda = 1.0$.

3.0.3 Neural network on Franke's Frunction

Table 2 shows a comparison of averaged MSE for the neural network and OLS on (30×30) Franke's function generated data sets with noise $\sim N(0, 0.3)$. The table indicates that the neural network produces an MSE that is up to 26 orders of magnitude larger than what the OLS from scikit-learn produces.

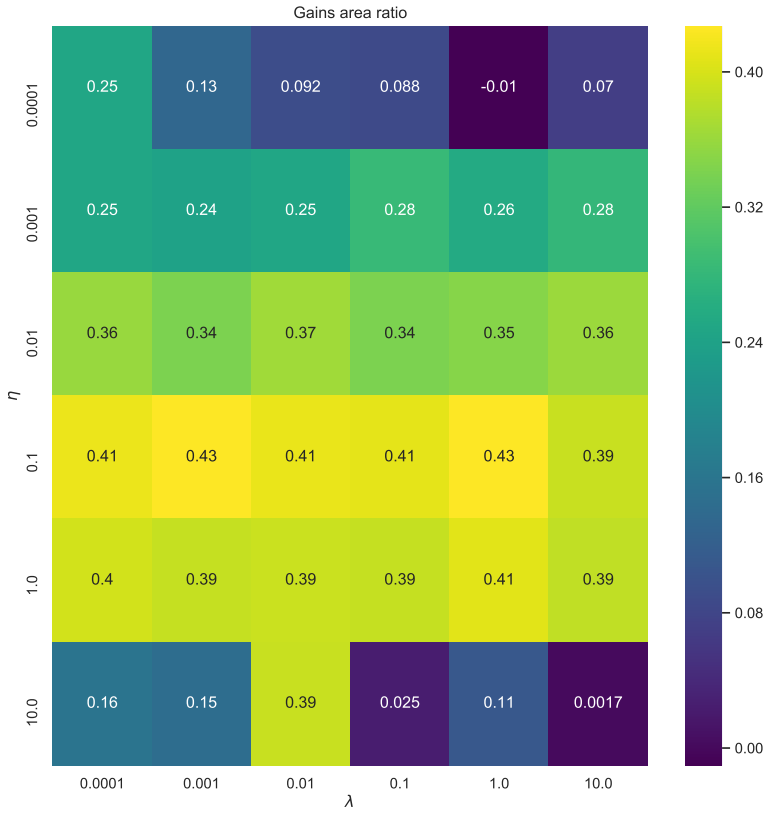

Figure 5: Gridsearch showing area ratio score for a number of values of η and λ, using 1000 epochs over a batch size of 1000.

Model	Validation	η	λ	MSE
Neural Network	Training (no split)	0.01	0.17	2×10^{-5}
Neural Network	Test (0.25 of the data)	0.31	0.01	3×10^{-5}
OLS	Training (no split)	-	-	6×10^{-30}
OLS	Test (0.25 of the data)	-	-	3×10^{-31}

Table 2: MSE after gridsearch for η and λ on (30×30) Franke's function generated data sets with noise $\sim N(0, 0.3)$ for the neural network (1000 epochs, batch size 50), compared with MSE for OLS using scikit-learn

4 Discussion of results

4.1 Implementation, testing, and evaluation

The accuracy score of the logistic regression model on the raw credit card data (with one-hot encoding and outlier removal) is 0.78, a deceivingly high score, considering that the area ratio score is 0.00. This indicates that that the model is classifying all the test data as 0 (1 being defaulting on payment) - which the confusion matrix (see appendix 1) confirms. The discrepancy in evaluation scores, and the gains chart (1), shows that accuracy is not necessarily a meaningful measurement on model performance. Additionally, having performed no better than a random classifier suggest that additional pre-processing of data is necessary, which is not surprising considering that only 22% of the entries after outlier removal have $y = 1$, ie. the data set is considerably skewed. As indicated in table 1, scaling is by far the most important data pre-processing in the case of credit card data - which may be due to the wide range of predictor values - the highest un-scaled value is 1684258, while the lowest are 0. The best performance evaluated by area ratio (0.46) is obtained with outlier removal, one-hot, scaling, smote. Area ratio score of 0.46 slightly outperforms similar analysis from [10] (0.44), which may be due to factors such as split ratio. Corresponding performance when using stochastic gradient decent and sci-kit were 0.44 and 0.33, respectively.

4.2 Neural network on credit card data

The first grid search (figure 3) points to 500 epochs over batches of size 500, or 1000 by 1000 for $\eta = 0.1$ and $\lambda = 0.1$ (area ratio score = 0.42), and shows a trend for intermediate batch sizes and epochs for (relatively) low values of the regularization (λ). The trend visible in figure 4, would suggest that lower number of epochs and larger batch sizes (again, relatively speaking) produce the best results (area ratio score = 0.43). This could indicate that higher values of λ allow for a larger number of epochs, as the model does not over-fit as easily with severe regularization penalty, thus does not display a drop in area ratio score on test data (bias variance-tradeoff). The two best epoch and batch size combinations in for low λ would also suggest this. Figure 5 exemplifies however that the choice of λ is not necessarily as directly linked to the number of epochs and batch size as the above discussion would suggest. The two parameter pairs of $\eta = 0.1$ and $\lambda = 0.001$, and $\eta = 0.1$ and $\lambda = 1.0$ with the highest area ratio score (0.43) for 1000 epochs and batch size 1000, may suggest that some values of epochs and batch size have several optimal combinations for learning rate and regularization - suggesting further that choosing suitable combinations of the four parameters is challenging at best, and may require more permutations than what this project shows. Additionally, comparison of the figures shows a discrepancy for $\eta = 0.1, \lambda = 1.0, epochs = 1000, batch = 1000$; figure 4 yields a score of 0.41, while figure 5 yields 0.43 - which may indicate some error in implementation, or alternatively display the stochastic properties of SGD and/or

the splitting and smote. The neural network appears to predict on par with logistic regression, albeit this may not be the case if more layers and implemented and/or better parameter adjustment is carried out. The paper by I-Cheng Yeh and Che-Hui Lien [10] reports area ratio scores of 0.54 for neural networks on the credit card data, which very strongly suggests that further improvement on the current implementation is achievable.

4.3 Neural network on Franke's Frunction

As table 2 clearly indicates; the neural network using one hidden layer does not perform nearly as well as OLS from scikit-learn, producing a MSE score to 26 orders of magnitude larger than what the OLS does on a (30×30) Franke's function generated data set with noise $\sim N(0, 0.3)$, both using validation sets and not. This may either indicate an ineptitude for neural networks in general, although the (relatively) high MSE is assumed to not be representative of neural networks as a whole. A more likely option is either failure to find suitable parameters (epochs, batch size, η and λ), one hidden layer being inadequate, or implementation error that was not triggered by initial testing, or a combination of these. As such, further testing is required in order to establish neural network aptitude for quantitative prediction. This project is unable to make any certain conclusions on the matter other than that the current implementation with one layer appears to favor OLS for predicting terrain.

4.4 Comparison of regression methods

Judging solely on the results produced in this project and project 1 [6], neural networks would appear perform on par with logistic regression for classification (area score ~ 0.44). However, the paper by I-Cheng Yeh and Che-Hui Lien [10] reports an area ratio score of 0.55 for neural networks, indicating that the results in this project are not indicative of the aptness of the method in classification, and that neural networks are instead the better suited option for classification. Neural networks (MSE $\sim 10^{-5}$) would also appear to be inferior for numerical prediction when compared with OLS (MSE $\sim 10^{-30}$). As the multiple hidden layer code did not pass initial implementation tests, the project is unable to make any conclusion on whether neural networks in general are less suited for regression, and concludes only that neural networks with one hidden layer is less suited. However, producing such a large difference in MSE for only 5 predictors on 900 data points, suggest that it is a lot easier to achieve low MSE with OLS than with neural networks. Neural networks appear to require care and practice when choosing parameters, such that obtaining accurate results when modeling a numerical function may be easier with methods such as OLS.

5 Conclusions

This project concludes that accuracy is unsuited as a measurement of prediction quality in categorical modeling for skewed data sets, having calculated an accuracy score of 0.78 for a model performing equivalent to a random classifier (area ratio score= 0.00)on the credit card data. This project goes on to concluding that scaling of predictors may be highly effective in increasing prediction quality, it's isolated application having produced the highest area ratio score (0.46 using logistic regression with gradient decent) on the credit card data of the pre-processing methods applied in this project. In the case of neural networks in classification, this project deems it likely that a single hidden layer is insufficiently complex to achieve optimal prediction on data with complexity and skewness (target ratio of 0.22) akin to that of the credit card data, having produced an area ratio score of 0.43 using $\eta = 0.1$,$\lambda = 1.0$, with 10000 epochs (SGD) and $batchsize = 1000$ on the credit card data. Cursory examination on the connection between the number of epochs, batch size, learning rate (η), and regularization (λ), suggests that all four parameters are closely linked and must be found through trial and error for each unique data set - but that regularization and the number epochs seem to have the strongest connection. Lastly, for predicting quantitative targets, neural networks (MSE $\sim 10^{-5}$)of one layer appear to underperform significantly when compared with ordinary least squares (MSE $\sim 10^{-30}$) on both training and validation data. The project is not able to reach any conclusion on whether additional layers would increase the predictive qualities of the network due to faulty implementation. Further studies of neural networks with multiple hidden layers are required to properly establish which areas neural networks are best suited for - the project does however suggest that it may not be in numerical regression problems.

References

[1] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer Series in Statistics. Springer New York, New York, NY, second edition edition, 2009.

[2] Morten Hjorth-Jensen. Lectures notes in fys-stk4155. data analysis and machine learning: Neural networks, from the simple perceptron to deep learning, Oct 4 2019.

[3] Gareth James. *An Introduction to statistical learning : with applications in R*. Springer texts in statistics. Springer, New York, corrected at 8th printing 2017. edition, 2017.

[4] Guillaume Lemaître, Fernando Nogueira, and Christos K. Aridas. Imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning. *Journal of Machine Learning Research*, 18(17):1–5, 2017.

[5] Kevin P. Murphy. *Machine learning : a probabilistic perspective*. Adaptive computation and machine learning. MIT Press, Cambridge, 2012.

[6] Johan Nereng. Project 1 in fys-stk4155: Regression analysis and re-sampling methods, Oct 8 2019.

[7] Michael A. Nielsen. *Neural Networks and Deep Learning*. Determination Press, 2015.

[8] Travis Oliphant. *Guide to NumPy*. 01 2006.

[9] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[10] I-Cheng Yeh and Che-Hui Lien. The comparisons of data mining techniques for the predictive accuracy of probability of default of credit card clients. *Expert Systems With Applications*, 36(2):2473–2480, 2009.

Appendices

Appendix 1.

Initial test on cancer data

GD - Own model: Accuracy: 0.97 Area score: 0.99

$$confusion = \begin{bmatrix} tn = 63 & fp = 2 \\ fn = 3 & tp = 120 \end{bmatrix} \tag{14}$$

SGD - Own model: Accuracy: 0.96 Area score: 0.99

$$confusion = \begin{bmatrix} tn = 61 & fp = 4 \\ fn = 3 & tp = 120 \end{bmatrix} \tag{15}$$

Sklearn model: Accuracy: 0.97 Area score: 0.90

$$confusion = \begin{bmatrix} tn = 61 & fp = 4 \\ fn = 2 & tp = 121 \end{bmatrix} \tag{16}$$

Credit card data, no pre-processing

$$confusion = \begin{bmatrix} tn = 7591 & fp = 0 \\ fn = 2178 & tp = 0 \end{bmatrix} \tag{17}$$

Implementation test of NN

See figre 6

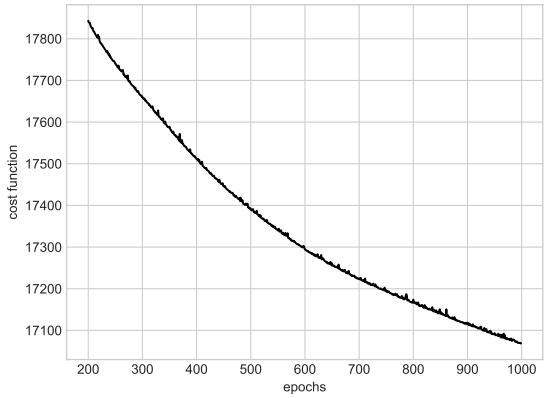

Figure 6: Plot of cost function for NN for initial implementation test

Iterations and batch size NN on credit card data

Epochs	Batch size	Area ratio score
2000	1000	0.39
1500	2000	0.407
1500	500	0.408
1000	1000	0.41
600	200	0.4

Table 3: Neural network on credit card data; area ratio score as functions of epochs and batch size using $\eta = 0.1$ and $\lambda = 0.0$