FYS-STK3155 - Project 2 Autumn 2019[5]

Johannes A. Barstad, Olve Heitmann

November 10, 2019

Abstract

In this paper we discuss and implement logistic regression (LR), as well as artifiscial neural networks (ANNs) for use in regression as well as classification problems. The LR and ANN models are trained using batch gradient descent, combined with backpropagation for the ANNs. To test our models we use simulated Franke Function data for regression, and Credit Card data from I-Cheng, Yeh. and Che-hui, Lien's article The comparisons of data mining techniques for the predictive accuracy of probability of default of credit card clients[7] *from 2008. For the simulated Franke Function data our results neural network are able to reproduce results similar to those obtained by OLS and manual feature engineering (polynomials and interaction terms)[1] with a simple two-dimensional input. For the credit card data our results are similar for both LR and ANNs to those obtained by I-Cheng and Che-hui.*

1 Introduction

The main aim of this project was to study classification and regressions problems, including implementing Logistic Regression (LR) as well as implementing and training a Multi Layer Perceptron (MLP) Neural Network using gradient descent and backpropagation. We then apply our models for i) predicting outputs from simulated data using the Franke Function, as well as ii) Predicting credit card defaults.
Dataset i) is our main test subject for the regression task. Our goal here was to study whether or not our MLP were able to reproduce and/ or improve upon the results we obtained in project 1 through manual feature engineering (i.e. the creation of polynoms and interaction terms) simply by taking in a two-dimensional input.

Our study on credit card defaults builds on the work of I-Cheng Yeh and Che-hui Lien[7], which outlines a comparison of various data mining techniques for predicting credit card defaults in the Taiwanese credit card market. I-Cheng and Che-hui's paper focus on a consumer payment dataset from October, 2005, retrieved from an "important bank in Taiwan"[7], as well as reviewing other relevant literature on credit card customers. Our focus here was to test whether or not we could obtain similar results for our implemented LR and MLP models in a classification task.

To justify our various choices made during the implementation we start of by discussing classification, LR and ANNs in a general theory section. For a general discussion on regression techniques, see our first project[1].

2 Theory

2.1 On classification problems

Classification problems, as opposed to regression, are concerned with taking on the form of discrete variables (i.e. categories). The perhaps most common, and definitely simplest classification problem, is the *binary classification problem*, where we have two possible outcomes. It is here common to encode the two possible outcomes as

$$y = \begin{bmatrix} 0 & \text{outcome1} \\ 1 & \text{outcome2} \end{bmatrix}$$

In the practical application part of this project we will study the binary case, namely credit card client defaults (two possible outcomes, the client ended up defaulting or the client did not end up defaulting) with a dataset from UCI's Machine Learning Repository[12].

In general however, the problem can be described as follows: We aim to study the case where the dependent variables (i.e. the responses or outcomes) are discrete and only can take on value from $k = 0, \ldots, K-1$ — i.e. K classes. We aim to predict the output classes based on *the design matrix* $\hat{\mathbf{X}} \in \mathbb{R}^{n\times p}$. The design matrix consists of n-samples, where each contain data on p features (predictors).

On the use of regressions techniques in classification
One way to employ regressions techniques in classification problems is to have a sign function for mapping the output of the regressor to the discrete values in our list of possible outcomes (e.g. for the binary classification problem to $\{0, 1\}$). An example of such a sign function could be $f(s_i) = sign(s_i) = 1$ if $s_i \geq 0$ and 0 otherwise. This approach is what historically has been called *the perceptron model* in machine learning literature.

Another way to use regression techniques to solve and describe classification problems is to interpret the output of the regression model as a *probability* of a given category. This can be extremely useful in real world applications where we are interested in the expected value of some variable that *depends* on the outcome of the problem we are trying to solve. One example of this could be a banks expected loss on credit debts over a given period, which would be given by *probability of loss*× *loss* (assuming "binary loss" — i.e. you either default all the money you owe, or nothing at all).
This approach to using regression techniques in classification problems are sometimes referred to as a "soft" classifier — we will study one example of this type of classifier, the *logistic regression.*

2.2 The Sigmoid function

The Sigmoid (logit) function is defined as:

$$f(z) = \frac{1}{1 + \exp{-z}}$$

with the first order derivative given by:

$$f'(z) = f(z)(1 - f(z))$$

Moving on we will refer to the sigmoid function as $\sigma(z)$. Note also that $1 - \sigma(z) = \sigma(-z)$, and that the function will output a number $(0, 1)$. This is an important attribute making it possible to interpret the output of a sigmoid function as the probability of a binary event.

2.3 Logistic Regression

For the binary case, logistic regression let us model the probability that a sample is of type "outcome2", as opposed to "outcome1". The binary case model has the form:

$$\log \frac{p(\hat{\beta}\hat{x})}{1 - p(\hat{\beta}\hat{x})} = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots \beta_p x_p$$

Logistic regression is widely used in biostatistical applications where binary problems occur frequently (patients survive or not, have heart disease or not, etc.)[6]

2.3.1 Fitting Logistic Regression Models

Logistic Regression models are typically fit by *maximum likelihood*, using the conditional likelihood of G given X, where G can be interpreted as the class of the observed sample, and X the observed explanatory factors.
Maximimum likelihood in general is not discussed in detail here, but can be thought of as methods were we aim at maximizing the probability of seeing the observed data — for a more mathematical rigorous explanation, see e.g. Hastie et. al chapter 8.[6].
As $Pr(G|X)$ completely specifies the conditional distribution, the *multinomial* distribution is appropriate (Hastie. et al page 120[6]). The log-likelihood is then given by

$$l(\theta) = \sum_{i=1}^{N} \log p_{g_i}(x_i; \theta)$$

where $p_{g_i}(x_i; \theta) = Pr(G = k|X = x_i; \theta)$. Coding $y_i = 1$ when $g_i = 1$, and $y_i = 0$ when $g_i = 2$, and denoting $p_1(x; \theta) = p(x; \theta)$ and $p_2(x; \theta) = 1-p(x; \theta)$ we can write

$$\begin{aligned} l(\beta) &= \sum_{i=1}^{N} \{y_i \log p(x_i; \beta) +)(1 - y_i) \log (1 - p(x_i; \beta))\} \\ &= \sum_{i=1}^{N} \{y_i \beta^T x_i - \log (1 + \exp^{\beta^T x_i})\} \end{aligned}$$

assuming that the vector of inputs x_i includes the constant term 1 to accommodate the intercept. To maximize the log-likelihood, we set the derivatives to 0. The score equations are then given by

$$\frac{\partial l(\beta)}{\partial \beta} = \sum_{i=1}^{N} x_i (y_i - p(x_i; \beta)) = 0$$

which are $p + 1$ equations *nonlinear in* β

2.4 Neural Networks

Neural networks (NNs) are a class of learning methods developed originally developed separately in two different fields — statistics and artificial intelligence. The two fields however ended up creating what is essentially identical models[6].

NNs work by extracting linear combinations of the inputs as a new set of *derived features*, and then modeling the target as a nonlinear function of the *derived features* (In simpler statistical models the construction of these features is typically more manual, and is often referred to as *feature engineering*). They are constructed as a *two-stage* regression or classification model, with the possible of multiple quantitative final outputs. For K-class classification we typically want K outputs, with the kth output modeling the probability of the sample being in class k. For regression problems 1 output suffice. The two stages can be interpreted as passing the inputs into the hidden layers, and extracting the output layer from the last hidden layer. For training our neural networks we will apply *gradient descent-* and *backpropagation-algorithms* to minimize our chosen cost functions.

2.4.1 Choosing cost functions

For the regression part we apply *quadratic loss*;

$$C = \frac{1}{2}\sum_{i}^{n}(\hat{y}_i - y_i)^2$$

and for the classification problem we apply *logistic loss*

$$C = -\frac{1}{n}\sum_{x}\left[y\ln a + (1-y)\ln(1-a)\right]$$

2.4.2 Gradient descent and minimizing cost functions

The fundamental idea of the gradient descent is that a function $F(\vec{x})$ decreases fastest if one moves $\vec{x}$ in the direction of the negative gradient:

$$\partial f(X) = \nabla_X f(X)\partial X$$

where ∇_X is the gradient of f with respect to X, i.e.

$$\nabla_X \equiv \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \ldots, \frac{\partial f}{\partial x_m}\right)$$

This idea stems from a well known linear algebraic relation, the inner product between two vectors of fixed lengths:

$$\vec{w}^T\vec{v} = |w||v|\cos\theta$$

This product can be shown to be at its maximum when the two vectors are aligned (i.e. when $\theta = 0$), and consequently at its minimum when the two vectors point in opposite directions.
In summary, finding the minimum of our scalar function requires us to find a turning point — i.e. a point where the gradient will be 0. As most of the cost functions desirable to apply for neural networks doesn't have closed form solutions, it is often not possible to simply solve $\nabla_X f(X) = 0$.

Iterative solutions

When closed form solutions are unavailable we begin with an initial "guess" for the optimal X and refine it iteratively until the correct value is obtained — One should here however note the "risk" of discovering mere local minima. For *squared error loss*

$$Loss = \frac{1}{T}\sum_i div(Y_i, d_i)$$

Minimize wrt $\{w_{ij}^l, b_j^l\}$ intialize all weights and biases $\{w_{i,j}^k\}$ For every layer k, for all i,j update $w_{i,j}^k = w_{i,j}^k - \eta\frac{\partial Loss}{\partial w_{i,j}^k}$ until loss has converged

One starts of with an initial guess LOL for the minimum value of F, and compute new approximations according to

- Initialize x^0, $k = 0$
- While $||\nabla_x f(x^k)|| > \epsilon$:

  $$x^{k+1} = x^k - \eta^k\nabla_x f(x^k)^T$$

  where η^k is the "step size" (learning rate), and ϵ is some threshold "Taking a step against the derivative"

2.4.3 Implementing backpropagation

Backpropagation was first introduced in a famous 1986 paper by David Rumelhart, Geoffrey Hinton, and Ronald Williams. The paper describes several neural networks where backpropagation works far faster than earlier approaches, making it possible to use neural nets to solve problems which had previously been insoluble. Backpropagation is now the main workhorse of learning in neural networks.

Assumptions for employing backpropagation

To successfully apply backpropagation we need to make two assumptions regarding the cost function:

1. **The cost function can be written as an average of the cost functions of the individual training samples:**

$$C = \frac{1}{n}\sum_{i}^{n} C_i$$

   Backpropagation let us compute the partial derivatives $\frac{\partial C_i}{\partial \vec{w}}$ and $\frac{\partial C_i}{\partial b}$ for a single training sample. We then need to ”recover” $\frac{\partial C}{\partial \vec{w}}$ and $\frac{\partial C}{\partial b}$ by averaging over the training samples — if this is not possible due to the cost function we choose, backpropagation will not give good approximations to the optimum parameter values.

2. **Cost can be written as a function of the outputs from the neural network**

The expression tells us how quickly the cost changes when we change the weights and biases

w^l_{jk} denotes the weight for the connection from the k^{th} neuron in the $(l-1)^{th}$ layer to the j^{th} neuron in the l^{th} layer. b^l_j for bias from j^{th} neuron in the l^{th} layer a^l_j activation of the j^{th} neuron in the l^{th} layer $a^l_j = \sigma\left(\sum_k w^l_{jk} a^{l-1}_k + b^l_j\right)$ W^l weight matrix for each layer l — entries the weight connecting to the l^{th} layer of neurons, every entry in the j^{th} row and k^{th} column is w^l_{jk}. $\vec{b^l}$ bias vector — components are just b^l_j, one for each neuron in the l^{th} layer $\vec{a^l}$ whose components are the activations a^l_j $a^l = \sigma(W^l \vec{a^{l-1}} + \vec{b^l})$

$z^l \equiv w^l a^{l-1} + b^l$ intermediate quantity — *weighted input to the neurons in layer l*

partial derivatives $\frac{\partial C}{\partial w}$ $\frac{\partial C}{\partial b}$

Intermediate quantity δ^l_j *error in the j^{th} neuron in the l^{th} layer* instead of outputting σz^l_j $\sigma(z^l_j + \Delta z^l_j)$ This change propagates through later layers in the network, finally causing the overall cost to change by an amount $\frac{\partial C}{\partial z^l_j}\Delta z^l_j$

$$\delta^l_j \equiv \frac{\partial C}{\partial z^l_j}$$

The fundamental equations of backpropagation

Backpropagation is based around four fundamental equations:

$$\delta^L_j = \frac{\partial C}{\partial a^L_j}\sigma'(z^L_j). \tag{BP1}$$

$$\delta^L = (a^L - y) \odot \sigma'(z^L). \tag{BP2}$$

$$\frac{\partial C}{\partial b^l_j} = \delta^l_j. \tag{BP3}$$

$$\frac{\partial C}{\partial w^l_{jk}} = a^{l-1}_k \delta^l_j. \tag{BP4}$$

An equation for the error in the output layer

$$\delta^L_j = \frac{\partial C}{\partial a^L_j}\sigma'(z^L_j)$$

1. **Input x:** Set the corresponding activation a^l for the input layer
2. **Feed forward:** For each $l = 2, 3, \ldots, L$ compute $z^l = w^l a^{l-1} + b^l$, and $a^l = \sigma(z^l)$
3. **Output error σ^L:** Compute the vector $\sigma^L = \nabla_a C \odot \sigma'(z^L)$
4. **Backpropagate the error:** For each $l = L-1, L-2, \ldots, 2$ compute $\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l)$
5. **Output:** The gradient of the cost function is given by $\frac{\partial C}{\partial w^l_{jk}} = a^{l-1}_k \delta^l_j$ and $\frac{\partial C}{\partial b^l_j} = \delta^l_j$

We compute the error vectors δ^l backward, starting from the final layer — hence *backpropagation.* the backward movement is a consequence of the fact that the cost is a function of outputs from the network. To understand how the cost varies with earlier weights and biases we need to repeatedly apply the chain rule, working backward through the layers to obtain usable expressions. In practice, it's common to combine backpropagation with a learning algorithm such as stochastic gradient descent, in which we compute the gradient for many training examples. In particular, given a mini-batch of m training examples, the following algorithm applies a gradient descent learning step based on that mini-batch

1. Input the set of training samples
2. Repeat step 1-4 of the back propagation algorithm for each training sample x, that is:
   (a) **Input x:** Set the corresponding input activation $a^{(x,l)}$
   (b) **Feed forward:** For each $l = 2, 3, \ldots, L$ compute $z^{(x,l)} = w^l a^{(x,l-1)} + b^l$, and $a^{(x,l)} = \sigma(z^{(x,l)})$

(c) **Output error** $\sigma^{(x,L)}$**:** Compute the vector $\sigma^{(x,L)} = \nabla_a C_x \odot \sigma'(z^{(x,L)})$

(d) **Backpropagate the error:** For each $l = L-1, L-2, \ldots, 2$ compute $\delta^{(x,l)} = ((w^{l+1})^T \delta^{(x,l+1)}) \odot \sigma'(z^{(x,l)})$

3. **Gradient descent:** For each $l = L, L-1, \ldots, 2$ update the weights and biases according to the following rules:

   - $w^l \rightarrow w^l - \frac{n}{m} \sum_x^n \delta^{(x,l-1)} (a^{(x,l-1)})^T$
   - $b^l \rightarrow b^l - \frac{n}{m} \sum_x^n \delta^{(x,l-1)}$

Of course, to implement stochastic gradient descent in practice you also need an outer loop generating mini-batches of training examples, and an outer loop stepping through multiple epochs of training

3 Datasets

3.1 Data generated by the Franke Function

For the regression part of the implementation we again use the data generated by the Franke Function from Project 1[1]. We here chose to use the non-noisy version, as one of our goals was to study whether or not the ANN actually was able to produce feature engineering similar to what we implemented in Project 1 using polynomials and interaction terms. Thus,

3.2 Default of credit card clients

In the classification part of the project we will study a cross sectional dataset containing credit card clients who either have, or have not defaulted. The time horizon is one month from the last observation of the proposed explanatory variable, until the observation of the dependent variable. The dataset were published

4 Application and results

4.1 Scoring metrics for classification problems

To properly assess the results from applying our models to the datasets discussed in section 3 we use the following metrics, as well as R^2 for the regression problem. For a discussion on R^2, see our report for Project 1[1].

Accuracy score
The accuracy score for binary classification problems are given by

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

where t_i is the guessed target, y_i I is the indicator function, equaling 1 if $t_i = y_i$ and equaling 0 otherwise. As we can see, a perfect classifier will have an accuracy score of 1, while a classifier that predicts every target wrong will have an accuracy score of 0. It also follows that a classifier that is correct exactly half of the time will have an accuracy score of 0.5, making interpretation easy.

Area ratio
The area-ratio for a binary classification problem are given by

$$\text{Area ratio} = \frac{\text{Area between the model curve and the baseline curve}}{\text{Area between theoretical optimal curve and the baseline curve}}$$

where a higher area ratio indicates a better performing model.

Lift charts

4.2 Comparison

LR credit mot sci article

5 Conclusion

This paper examined various regression and classification techniques, applying Logistic Regression and Artificial Neural Networks on the datasets generated by the Franke-Function, and the Credit Card data provided in I-Cheng and Che-hui's paper[7].

5.1 Assessing the different models

OLS vs ANNs for regression
In terms of accuracy (estimated with test R^2) our ANN implementation performed similar to our OLS implementation from project 1[1]. The main differences can thus be summarized in

Logistic Regressions vs ANNs for classification

A logistic regression model specifies that an appropriate function of the estimated probability of an event is a linear function of the observed values of the available explanatory variables[?] . The model can produce a simple and fairly interpretable formula for classification, however it is typically harder to properly model non-linear and interactive effects between the observed explanatory variables.

Artificial neural networks can use non-linear equations to develop relationships between input and output variables through what is called a *learning process*. In our study we, as well as I-Cheng and Che-hui[?] , apply a back propagation feed-forward neural network. As we have well labeled data (i.e. customers who defaulted, and customers who did not default), we were able to use supervised learning. ANNs can handle non-linear and interactive effects of explanatory variables, but does not have an easily interpreted formula for classification.

5.2 Conclusion and final remarks

5.3 Feedback for future versions of this course

When starting this project, we expected to be able to use the feedback from Project 1 to improve upon our delivery from the last project. As we still haven't gotten any feedback, we were not able to do so. This is in our opinion very unfortunate. Additionally, we are under the impression that some students have received feedback from the first project in devilry, and thus have been able to take into account your remarks to improve their report for Project 2. If this is the case, scoring and feedback on this project (i.e. project 2) should reflect this difference in possibilities to improve upon ones report.

Table 1: Default of credit card clients attribute information

Explanatory variable	Explanation
X_1	Amount of the given credit (NT dollar): Individual consumer credit and family credit.
X_2	Gender (1 = male; 2 = female).
X_3	Education (1 = graduate school; 2 = university; 3 = high school; 4 = others).
X_4	Marital status (1 = married; 2 = single; 3 = others).
X_5	Age (year).
X_6 - X_{11}	(Monthly) Past payment status. Scale: -1 = pay duly; 1 = delay 1 month; 2 = delay 2 months
X_{12} - X_{17}	(Monthly) Amount of bill statement (NT dollar).
X_{18} - X_{23}	(Montly) Amount of previous payment (NT dollar).

6 Appendix

References

[1] Johannes A. Barstad and Olve Heitmann. Project1 github repository. https://github.com/Barstad/fys_stk3155/tree/master/PROJECT1.

[2] Johannes A. Barstad and Olve Heitmann. Project2 github repository. https://github.com/Barstad/fys_stk3155/tree/master/PROJECT2.

[3] Derek Bingham. Virtual library of simulation experiments: Test functions and datasets. franke's function. https://www.sfu.ca/~ssurjano/franke2d.html.

[4] M. Chou. Cash and credit card crisis in taiwan, 2006.

[5] Norway Department of Physics, University of Oslo. Project 2 assignment description. https://compphysics.github.io/MachineLearning/doc/Projects/2019/Project2/pdf/Project2.pdf.

[6] Trevor Hastie, Robert Tibshirani, and Jerome H. Friedman. *The Elements of Statistical Learning*. Springer, 2008.

[7] Che-hui Lien I-Cheng, Yeh. The comparisons of data mining techniques for the predictive accuracy of probability of default of credit card clients. https://bradzzz.gitbooks.io/ga-seattle-dsi/content/dsi/dsi_05_classification_databases/2.1-lesson/assets/datasets/DefaultCreditCardClients_yeh_2009.pdf.

[8] Wessel N. van Wieringen. Data analysis and machine learning lectures: Optimization and gradient methods. https://compphysics.github.io/MachineLearning/doc/pub/Splines/html/._Splines-bs000.html.

[9] Wessel N. van Wieringen. Data analysis and machine learning: Logistic regression. https://compphysics.github.io/MachineLearning/doc/pub/LogReg/html/LogReg.html.

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

[11] USGS. Earthexplorer. https://earthexplorer.usgs.gov/.

[12] USGS. Uci machine learning repository - default of credit card clients data set. https://earthexplorer.usgs.gov/.