Project 2 FYS-STK4155 Autumn 2018

Jon Audun Baar

November 12, 2018

Abstract

In this project we compared regression analysis and neural networks as methods for developing models for both continuos and categorical data. We used data generated by the Ising model in both cases. First we showed that linear regression is prefferable over deep networks in our special case when it comes to learning efficiency. With a large amount of data they both created good models however. We then compared logistic regression and deep neural networks on a classification problem and showed that the neural net by far is better than regression in this case due to the non-linear classification boundaries of our problem.

1 Introduction

The concept of machine learning have gained a hughe popularity boost over the last couple of years. And this is no wonder: the techniques have a wide range of applications and can be a major asset if you know when to use what.

When to use what is exactly what we're going to have a brief peek into in this project. We aim to evaluate the performance of regression analysis and deep neural networks in two different cases. First we compare linear regression and neural networks in the case of a continuous output, then we move on to comparing logistic regression and neural nets in the case of a classification problem. In both cases we use the much studied Ising model to generate our datasets.

2 Method

In order to present the methods used, we first need to establish some terminology and notation. In this text we will use both machine learning terminology and more statistical language. Hence we talk about our independent variables both as *input* and *predictors*, and our dependent variables as both *output* and *response*. We assume that we are given a dataset consisting of $N \in \mathbb{N}$ samples, where each sample consists of $p \in \mathbb{N}$ input variables and one output. We then denote by x_{ij} the j-th predictor of the i-th sample and by t_i the response of the i-th sample. Further we partition the dataset into training data consisting of N_t samples, and test data consisting of N_v samples. Of course we then have $N_t + N_v = N$.

When applying both regression analysis and neural networks the basic concept is the same: You want to fit a model to your trainingdata by minimizing some sort of error. The model and minimization methods however are different.

2.1 Regression analysis

2.1.1 Linear regression

The concept of linear regression should by now be known from [1]. The only difference in this project is that we now have $p = L$ predictors instead of 2 and a fixed model instead of several different models of different complexity. Since our model is given by

$$E = \sum_{i,j=1}^{L} \boldsymbol{J}_{i,j} x_i x_j \tag{1}$$

where $\boldsymbol{J}_{i,j}$ is the coupling constant from spin i to spin j, our design matrix is now given by

$$X = \begin{bmatrix} x_{1,1}x_{1,1} & \dots & x_{1,1}x_{1,L} & \dots & x_{1,L}x_{1,1} & \dots & x_{1,L}x_{1,L} \\ x_{2,1}x_{2,1} & \dots & x_{2,1}x_{2,L} & \dots & x_{2,L}x_{2,1} & \dots & x_{2,L}x_{2,L} \\ \vdots & & \vdots & & & & \vdots \\ x_{N,1}x_{N,1} & \dots & x_{N,1}x_{N,L} & \dots & x_{N,L}x_{N,1} & \dots & x_{N,L}x_{N,L} \end{bmatrix} \tag{2}$$

Apart from this little twist the procedure is exactly the same as in [1].

2.1.2 Logistic regression

When the output of a dataset is categorical, logistic regression is a possible method for developing a model for the relationship between input and output. The model to fit then gives us the probabilities for a datapoint to be of any of the given categories. In our case we have only two categories and our model is then given by

$$p(\boldsymbol{x}_i; \beta_0, \boldsymbol{\beta}) = \frac{1}{1 + \exp(-\beta_0 - \boldsymbol{x}_i \boldsymbol{\beta})} \tag{3}$$

where $p_1(\boldsymbol{x}_i; \beta_0, \boldsymbol{\beta}) := p(\boldsymbol{x}_i; \beta_0, \boldsymbol{\beta})$ then is the probability for datapoint i to be of category 1, and $p_2(\boldsymbol{x}_i; \beta_0, \boldsymbol{\beta}) := 1 - p(\boldsymbol{x}_i; \beta_0, \boldsymbol{\beta})$ is the probability for the same datapoint to be of category 2. Here $\boldsymbol{x}_i \in \mathbb{R}^p$ is the input of datapoint i given as a row vector. $\beta_0 \in \mathbb{R}$ and $\beta \in \mathbb{R}^p$ (column vector) are the coefficients/weights of the model which we are to estimate.

If we assume that the targets of the outputs of the model is $\{t_i\}_{i=1}^N$, the cost function is then given by

$$C(\beta_0, \boldsymbol{\beta}) = \sum_{i=1}^{N} t_i \log(p(\boldsymbol{x}_i; \beta_0, \boldsymbol{\beta})) + (1 - t_i) \log(1 - p(\boldsymbol{x}_i; \beta_0, \boldsymbol{\beta})) \tag{4}$$

where log is the natural logarithm.

As opposed to some cases of linear regression, there is no analytical solution to the problem of minimizing this cost function. We therefore have to resort to numerical methods to approximate a minimum. This will be discussed in section 2.3. Anyway we will need the gradient and Hessian of C in order to apply these methods. These are

$$\nabla C(\beta_0, \boldsymbol{\beta}) \; = \; \sum_{i=1}^{N} \boldsymbol{x}_i (t_i - p(\boldsymbol{x}_i; \beta_0, \boldsymbol{\beta})) \tag{5}$$

and

$$\frac{\partial^2 C(\beta_0, \boldsymbol{\beta})}{\partial \beta \partial \beta^T} \; = \; -\sum_{i=1}^{N} \boldsymbol{x}_i \boldsymbol{x_i}^T p(\boldsymbol{x}_i; \beta_0, \boldsymbol{\beta})(1 - p(\boldsymbol{x}_i; \beta_0, \boldsymbol{\beta})) \tag{6}$$

respectively.

2.2 Neural Networks

When dealing with neural networks, the overarching concept is still the same as when one does regression analysis: We have a model that we wish to fit to a dataset by minimizing some error. The difference is the model and the methods by which we minimze the error.

As the scope of this text is not to fully explain neural networks rather than to analyse it's performance in some special cases, we will simply establish the needed notation and assumptions. For a thorough explanation of neural networks consult [2].

We will use a sequential multilayered network with L layers as our model. Let the number of nodes in layer l be denoted by n^l and let $n^0 = p$. We then denote by $W^l \in \mathbb{R}^{n^l \times n^{l-1}}$ and $\boldsymbol{b}^l \in \mathbb{R}^{n^l}$ the weight matrix and bias vector of layer l respectively. Finally we denote by $f^l : \mathbb{R}^{n^l} \to \mathbb{R}^{n^l}$ the activation function of layer l. As we in our case have a binary classification problem, it is sufficient with one node in the last layer, and our model is then given by $h : \mathbb{R}^p \to [0, 1]$ defined by

$$h(x) \; = \; f^L(\dots f^2(W^2 f^1(W^1 \boldsymbol{x} + \boldsymbol{b}^1) + \boldsymbol{b}^2) + \dots + \boldsymbol{b}^L) \tag{7}$$

There are several activationfunctions: Sigmoid, relu, softmax, leaky relu, tanh and so on. Since we basicly are faced with a forest of parameters to tweak when working with neural nets, we've limited ourselves to try out just a few in this project.

2.3 Gradient methods

As we have now seen, there is no analytical solutuion to the problem of minimizing the cost function when dealing with both logistic regression and neural networks. For this reason we need to consider some numerical methods for approximating the minimium. All these methods have one weakness in common, they might end up finding local minima. However if the functions we are minimizing are convex, we are guaranteed to find the global minima. Luckily most of the functions we are interested in minimizing are convex. (If they are concave, the negative of the function is concave and we can maximize this instead.)

2.3.1 Gradient descent

The perhaps simplest approach to minimizing a real valued function f iteratively, is to set some starting point $\boldsymbol{x}_0$, compute the gradient ∇f in $\boldsymbol{x}_0$, and move $\eta > 0$ in the opposite direction of this gradient in order to obtain $\boldsymbol{x}_1 = \boldsymbol{x}_0 - \eta \nabla f(\boldsymbol{x}_0)$. Then one iterates by setting $\boldsymbol{x}_{i+1} = \boldsymbol{x}_i - \eta \nabla f(x_0)$. This is the framework of gradient descent. For small enough η this implies $f(\boldsymbol{x}_i) > f(\boldsymbol{x}_{i+1})$. The problem with this approach is to find out what η is small enough. If we choose η too small, it might take days for our algorithm to converge. If we on the contrary choose η to big, it might not converge at all.

2.3.2 Newton-Raphson

The Newton-Raphson method is a kind of gradient descent method, except we choose the learning rate in a more clever way. We thus start out by choosing a random starting point $\boldsymbol{x}_0$, just as in the case of gradient descent. The update step however is a bit different:

$$\boldsymbol{x}_{i+1} = \boldsymbol{x}_i - \frac{\partial^2 f(\boldsymbol{x}_i)}{\partial \boldsymbol{x}_i \partial \boldsymbol{x}_i^T} \frac{\partial f(\boldsymbol{x}_i)}{\partial \boldsymbol{x}_i} \tag{8}$$

The result is a method that converges way faster than normal gradient descent. The draw back is however the need to compute the Hessian of your objective function. For a thourough derivation of this method consult [4]

2.3.3 Stochastic gradient descent

To avoid overfitting when training neural networks, training on randomly picked batches of the training data for each iteration is a good regularization method. If we let b be our batch size, the first step of this method consists of dividing the training data into $N//b$ batches. One does this by picking from the N datapoints without replacement. The second step is then to perform a gradient descent as described in section 2.3.1 with a new batch for each iteration.

2.4 Performance measures

2.4.1 MSE, R2-score, Bias and Variance

For an explanation of the MSE, R2-score, bias and variance please consult [1].

2.4.2 Accuracy score

For classification problems with multiple categories the performance measures we derived in [1] is not well defined. Therefore in order to measure the performance of our classification models, we define the following measure

$$\text{Accuracy} = 1/N \sum_{i=1}^{N} \chi_{\{t_i\}}(y_i) \tag{9}$$

where $\chi_{\{t_i\}}$ is the indicator function for the set $\{t_i\}$ and y_i is the class predicted by our model. In other words, the accuracy is the ratio of correct classifications over the total number of classifications.

2.5 The Ising model

The model we use to generate data for our experiments is the well known Ising model given by

$$E = -J \sum_{\langle kl \rangle}^{N} s_k s_l \tag{10}$$

where $\langle kl \rangle$ indicates that we sum only over neighbouring spins, $s_k = \pm 1$ are the spin values, N is the total number of spins and J is the coupling constant that expresses the strength of the interaction between neighbouring spins. For a more thorough explanation of this model consult [3].

3 Implementation

Implementation of the logsitic regression and neural networks was done with Python. We've chosen to implement each of the models as classes. The full code with test is located at https://github.com/jonabaa/Project2.git

4 Analysis of the methods

4.1 Estimating the coupling constant of the one-dimensional Ising model using linear regression

We investigated how well OLS-, Ridge- and Lasso regression were able to estimate the couplingconstants of the general Ising model given the spins $\boldsymbol{x}_i$ and the energies E_i from the onedimensional version of the model. We denote the coupling constants of the general Ising model by $\boldsymbol{J}$. None of the methods yielded useable models before N_t was larger than 200, hence we used $N_t = 400$ (and $N_v = 600$) in our further investigations.

As one can see from figure 1, Lasso regression with $10^{-3} < \lambda < 10^{-1}$ clearly outperforms the two other methods in estimating the coupling constants. This is further established by the plots in figure 2.

4.2 Estimating the energy of the one-dimensional Ising model using deep neural networks

We performed a grid search over the regularization parameter λ and the size of the training data set. This showed that neural networks can do very well predicting enrgies, with R2-scores exceeding 99%. The only draw back is the need for large amounts of datapoints for training.

4.3 Determining the phase of the two-dimensional Ising model using logistic regression

For as large datasets as my computer could handle, logistic regression performed poorly consistently. Mininmizing the cost function with Newton-Raphson tended to yield the fastest and best results, yet not good results. We managed to get an accuracy of roughly 70% for $N = 3000$.

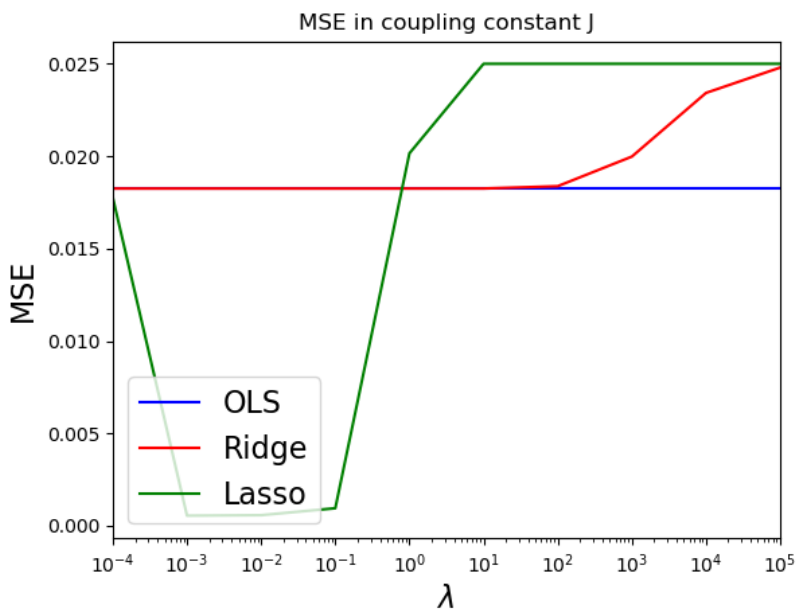

Figure 1: Comparing coupling constants

4.4 Determining the phase of the two-dimensional Ising model using deep neural networks

Classification seems to be some of the strengths of neural networks, at least in this case. We did a grid search over different regularization paramters and training data sizes and managed to make models that predicted with accuracys well over 97%, which must be considered to be very good.

5 Conclusion

The clearest result from this investigation would be that logistic regression don't perform well in the case of classifying wether a spin configuration is stable or unstable. Deep neural networks, on the contrary, performs the task of classification very well.

It is also pretty obvious from figure 2 that Lasso regressoion is the only method that yields good results in estimating the coupling constants in the Ising problem.

References

[1] Baar and Ravndal. Project 1 fys-stk4155 autumn 2018. *https://github.com/jonabaa/Project1/blob/master/project01.pdf*, pages 1–5, 2018.

[2] Hastie, Trevor, Tibshirani, Robert, Friedman, and Jerome. *The Elements of Statistical Learning.* Springer, 2009.

[3] Barry M. McCoy and Tai Tsun Wu. *The two-dimensional Ising model.* Harvard University Press, 1973.

[4] G. R. Walsh. *Methods of optimization.* John Wiley and sons, 1975.