Exploring the hyperspace of Machine Learning parameters

Eirik Ramsli Hauge, Joakim Kalsnes, Hans Mathias Mamen Vege

November 13, 2018

Abstract

NaN

1 Introduction

With the rapid expansion of computer power, data science is becomming an ever bigger part of modern society. Among the methods used to process data, some of the most popular are regression and classification. Through this paper we will look closely at the use of these methods when applied to data from the Ising model for spins. First of, we will use linear, ridge and lasso regression to estimate the coupling constant for the one-dimensional Ising model. Secondly, we move on to logistic regression and determining the phase of a spin matrix created by the two-dimensional Ising model. The latter problem is a classification problem and we will look further into it by creating a neural net. To learn more about the properties of a neural net, we will start by using the one-dimensional Ising model to find the optimal weights and biases in a regression case. Then we will apply what we learned by repeating the same classification as the logistic case, but this time we will train a neural net with a cross-entropy cost function to perform the task.

Our work is heavily inspired by the work of Metha et al. [3], and therefore it is only natural to compare our results to theirs. Lastly, we will give a detailed and in-depth analyzis of the algorithms used and how our home-made algorithms compares to those of Metha et al. and similar calculations done by standard libraries such as scikit-learn or TensorFlow.

2 Theory

2.1 The Ising Model

The Ising model is a way of modeling phase transitions at finite temperatures of magnetic systems. When modeling, we set up a chain or a lattice of particles and allow them to have either spin up or spin down. From this, one can sample energy and magnetization, and measure several quantities such as the heat capacity or magnetic susceptibility. Our focus will be on prediction the energy coupling constant for a 1D lattice, and the phase of a 2D-lattice.

Periodic boundary conditions is given in both cases, such that $j = N = 0$, where j is the lattice site and N is the lattice size.

2.1.1 1-dimensional Ising model

Energy for a 1 dimensional Ising model is given as

$$E = -J \sum_{j=1}^{N} s_j s_{j+1} \tag{1}$$

where the N is number of particles(or lattice size). Our goal will be to predict J, but in order to do so, we must recast the problem as linear regression problem.

We begin by labeling each site as coupled with J.

$$E_{\text{model}}[\boldsymbol{s}^i] = -\sum_{j=1}^{N}\sum_{k=1}^{N} J_{j,k} s_j^i s_k^i, \tag{2}$$

where i is the index over lattice configurations. The coupling strength $J_{j,k}$ can now be cast as a matrix, and we end up with

$$E_{\text{model}}^i \equiv \boldsymbol{X}^i \cdot \boldsymbol{J}, \tag{3}$$

where $\boldsymbol{X}^i$ is the design matrix consisting of all two-body interactions $\{s_j^i s_k^i\}_{j,k=1}^N$, and $\boldsymbol{J}$ the weight matrix we wish to find later using machine learning techniques.

2.1.2 2-dimensional Ising model

The 2D Ising model has its energy stated as,

$$E = -J \sum_{<kl>}^{N} s_k s_l, \tag{4}$$

where $< kl >$ indicates a sum over the nearest neighbors. That is, written out,

$$E = -J \sum_{i,j}^{N} 2 s_{i,j}(s_{i+1,j} + s_{i-1,j} + s_{i,j+1} + s_{i,j-1}), \tag{5}$$

where we have used the symmetry that $s_i s_{i+1,j} = s_{i+1,j} s_{i,j}$. J as as before a coupling constant, but will not be of our main focus when studying the 2D Ising model. This time, we will focus on its property of exhibiting phase transitions. Below a critical temperature of $T_C \approx 2.269$ found analytically by [5], the lattice will exhibit a ordered state, one in which the spins is *locked* or *frozen* into place. Above T_C the lattice exhibit a disordered phase, as the spins will be fluctuating randomly.

We will investigate the classification of states below $T < 2.0$ and $T > 2.5$. The phase for states between we will dub as being in a *critical phase*.

2.2 Logistic regression

In project 1 we used linear regression to predict a continuous output from a set of inputs (reference project1). We used ordinary least squares regression (OLS), Ridge regression and Lasso regression, where the two latter impose a penalty to the OLS. In this project we will

reuse the ideas and code of project 1, but we will also use neural networks to predict continuous variables. In addition we study situations where the outcome is discrete rather than continuous. This is a classification problem, and we will use logistic regression to model the probabilities of the classes.

2.3 Logistic regression

Just like a linear regression model, a logistic regression model computes a weighted sum of the predictor variables, written in matrix notation as $\boldsymbol{X}^T\beta$. However, the logistic regression returns the logistic of the this weighted sum as the probabilities. For a classification problem with K classes, the model has the following form (Hastie et al. p.119),

$$\begin{aligned} log\frac{Pr(G=1|X=x)}{Pr(G=K|X=x)} &= \beta_{10} \ + \ \beta_1^T x \\ log\frac{Pr(G=2|X=x)}{Pr(G=K|X=x)} &= \beta_{20} \ + \ \beta_2^T x \\ &\vdots \\ log\frac{Pr(G=K-1|X=x)}{Pr(G=K|X=x)} &= \beta_{(K-1)0} \ + \ \beta_{K-1}^T x \end{aligned} \tag{6}$$

It is arbitrary which class is used in the denominator for the log-odds above. Taking the exponential on both sides and solving for $Pr(G=k|X=x)$ gives the following probabilities:

$$\begin{aligned} Pr(G=k|X=x) &= \frac{exp(\beta_{k0} \ + \ \beta_k^T x)}{1+\sum_{l=1}^{K-1} exp(\beta_{l0} \ + \ \beta_l^T x)}, \ k=1,...,K-1, \\ Pr(G=K|X=x) &= \frac{1}{1+\sum_{l=1}^{K-1} exp(\beta_{l0} \ + \ \beta_l^T x)}, \end{aligned} \tag{7}$$

and the probabilities sum to one. The output is then classified as the class with the highest probability.

2.3.1 Fitting logistic regression model

The usual way of fitting logistic regression models is by maximum likelihood. The log-likelihood for N observations is defined as:

$$l(\theta) \ = \ \sum_{i=1}^{N} logp_{g_i}(x_i;\theta), \tag{8}$$

where $p_k(x_i;\theta) \ = \ Pr(G=k|X=x_i;\theta)$ and $\theta \ = \ \{\beta_{10},\beta_1^T,.....,\beta_{(K-1)0},\beta_{K-1}^T\}$.

One very common classification problem is a situation with binary outcomes, either it happens or it does not. As we see from Equation 6 above, setting K=2 simplifies the model considerable, since there will now be only a single linear function. θ in Equation 8 will also be simplified: $\theta = \beta = \{\beta_{10},\beta_1^T\}$. The two-class case is what is used in this project, and the following discussion will assume the outcome has two classes.
We start by coding the two-class g_i with a 0/1 response y_i, where $y_i = 1$ when $g_i = 1$, and

$y_i = 0$ when $g_i = 2$. Next, we let $p_1(x;\theta) = p(x;\beta)$, and $p_2(x;\theta) = 1 - p(x;\beta)$. The log-likelihood can then be written

$$\begin{aligned} l(\beta) &= \sum_{i=1}^{N}\{y_i log p(x_i;\beta) + (1-y_i)log(1-p(x_i;\beta))\} \\ &= \sum_{i=1}^{N}\{y_i log\frac{p(x_i;\beta)}{1-p(x_i;\beta)} + log(1-p(x_i;\beta))\} \\ &= \sum_{i=1}^{N}\{y_i\beta^T x_i + log(1-\frac{1}{1+exp(-\beta^T x_i)}\} \\ &= \sum_{i=1}^{N}\{y_i\beta^T x_i + log(\frac{exp(1}{1+exp(\beta^T x_i)}\} \\ &= \sum_{i=1}^{N}\{y_i\beta^T x_i - log(1+exp(\beta^T x_i))\}. \end{aligned} \tag{9}$$

This is the equation we want to maximize to find the best fit. Following the approach in Géron's book (reference), we chose the equivalent approach of minimizing the following

$$J(\beta) = -\frac{1}{N}\sum_{i=1}^{N}\{y_i\beta^T x_i - log(1+exp(\beta^T x_i)).\} \tag{10}$$

This is just the negative of Equation 9, divided by the number of samples. This is our cost function, and dividing by the number of training samples finds the mean cost.

To minimize this cost function we used gradient descent. Gradient descent measures the local gradient of the cost function, with regards to β in our case. Since the gradient goes in the direction of fastest increase, we will go in the opposite direction, i.e. negative gradient. We start by choosing random values for β (since our cost function is convex any choice should give correct results), calculate the gradient, update the β values, and do this iteratively until the algorithm converges to a minimum. The size of the steps is important, and is determined by the learning rate. If the learning rate is too small, we will need many iterations which is time consuming. However, if the learning rate is too high, we might overshoot and miss the minimum. One way to choose the learning rate is too let it depend on the size of the gradient. If the gradient is large, i.e a steep slope, the learning rate can be relatively high. When the gradient is small, the learning rate is also small.

Returning to the logistic regression problem, the derivative of the cost function is

$$\begin{aligned} \frac{\partial J(\beta)}{\partial\beta} &= -\frac{1}{N}\boldsymbol{X}^T(\boldsymbol{y}-\boldsymbol{p}) \\ &= \frac{1}{N}\boldsymbol{X}^T(\boldsymbol{p}-\boldsymbol{y}), \end{aligned} \tag{11}$$

where $\boldsymbol{X}$ is the $N \times (p+1)$ matrix of x_i values, $\boldsymbol{p}$ is the vector of fitted probabilities with ith element $p(x_i;\beta)$ and $\boldsymbol{y}$ is the vector of y_i values. The new β using gradient descent is then $\beta_{new} = \beta_{old} - \frac{\partial J(\beta)}{\partial\beta} lr$, where lr is the learning rate (step size). This is done iteratively until we reach the set max iterations or $\frac{\partial J(\beta)}{\partial\beta}$ is within a given tolerance of zero.

Like we introduced Lasso and Ridge regression to avoid overfitting in Project 1, we can add a penalty term to the cost function in Equation 10. In our project we used two different penalties: $L1 = \lambda|\beta|$ and $L2 = \lambda||\beta||^2$. When fitting the model we need to include the derivatives of the penalty term in Equation 11. The gradient with the penalty term is

$$\begin{aligned} \frac{\partial J(\beta)}{\partial \beta} &= \frac{1}{N}\boldsymbol{X}^T(\boldsymbol{p}-\boldsymbol{y}) \ + \lambda \cdot sign(\beta), \ for\ L1\ regularization \\ &or \\ \frac{\partial J(\beta)}{\partial \beta} &= \frac{1}{N}\boldsymbol{X}^T(\boldsymbol{p}-\boldsymbol{y}) \ + \lambda \cdot 2\beta, \ for\ L2\ regularization. \end{aligned} \tag{12}$$

2.4 Cost Functions

Some text if not it wont compile

2.5 Neural Networks

2.6 Neural Networks

Among the many methods developed for machine learning, neural networks, and especially deep neural networks, are among the most popular. Neural networks were suggested already in 1943 [2] and have had many renaissances since. Currently we are experiencing such a renaissance, but in contrast to earlier periods of resurfaced interest, we now have the computer power to use neural nets efficiently.

A neural net bases itself loosely upon the biological model of neurons communicating together in the brain. A neuron cell contains most of what a normal cell contains, but it also has a long tail called an axon and some antenna like extension called dendrites. The axon of one cell can extend quite far and attach to some of the dendrites of another neural cell. Thus, the biological neural net consists of neural cells receiving input through their dendrites from many other cells and sending output through one output [1][p. 257].

The computed neural networks works in a similar way. We construct "neurons" or "nodes" which are ordered in different layers where each neuron in one layer is connected to all neurons in the next layer. Initially, we start with an input layer which we feed information. Following this initial layer we have one or many hidden layers before we reach the output layer. A neural network with two or more hidden layers are called deep neural networks [1][p. 263]. Each neuron contains an activation function which determines the strength of the output. In the early days, a step function was used as the activation function. However, one has found that the use of a activation function with a gradient, such as the logistic function used in logistic regression, gives a better neural net. This is due to the fact that we now can apply gradient descent when optimizing the neural net which is discussed below.

To activate a neuron, it needs an input. This input is provided by all the neurons in previous layers through "wires" connecting the neurons (think of the axon to a dendrite). Each of these "wires" is weighted and all connections between one layer and the next is affected by a

bias term. Thus, the output of a neuron is given as [1][p. 260]

$$a = \sigma(\vec{w}^T \cdot \vec{x} + b) \tag{13}$$

where a is the output, $\vec{w}$ are the weights, $\vec{x}$ are the inputs and b is the bias term. Note that if we had used a stepfunction instead of the logit function in the equation above, the neuron would either give an output of 1 or nothing, i.e. 0. When we instead use the sigmoid function, the output can be in the range of 0 to 1.

Once all neurons have been calculated in a layer, we can move on to the next and continue until we reach our output layer. Each layer can have as many neurons as the user wants. Optimizing the number of neurons in each layer is an art and requires both experience and a bit of luck. The output layer needs one neuron for each class we wish to identify.

After initial calculation of the outer layer, you will most likely have an answer that is completely rubbish. It is clear that we have to optimize the neural net. As each neurons activation function is calculated using equation (13), we can see that we can optimize the weights between the neurons and the bias between the layers. This is done through a method called backwards propagation where one uses the cost function to identify the magnitude of the error (the cost) of a neural net and then one goes backwards through the neural net to update the weights and biases.

The basic procedure of backpropagation is this (heavily influenced by the work of Michael Nielsen [4]):

- Compute the output error vector for the final layer (L) given by

  $$\delta^L = \nabla_a C \odot \sigma'(\vec{w}^T \cdot \vec{x} + b)$$

  where C is the cost function and ∇_a is a vector who has components that are the partial derivatives $\frac{\partial C}{\partial a_j^L}$ where a_j is the j'th output found by using equation (13) for a.

- Go back through all the previous layers l = L-1, L-2, ..., 2 and compute

  $$\delta^l((\vec{w}^{l+1})^T \delta^{l+1}) \odot \sigma'(\vec{w}^T \cdot \vec{x} + b)$$

  this is where we backpropagate the error.

- Finally, find the gradient of the cost function for the two parameters we want to change, $\vec{w}$ and b by:

  $$\frac{\partial C}{\partial \vec{w}_{jk}^l} = a_k^{l-1} \delta_j^l$$
  $$\frac{\partial C}{\partial b_j^l} = \delta_j^l$$

  where k indicates the column of $\vec{w}^l$ as j indicates the row.

Once all layers have been adjusted through backpropagation, one can run through the whole network again and repeat the process. A common cost function may be the quadratic loss function given by:

$$C = \frac{1}{N}\sum_j (y_j - a_j^L)^2 \tag{14}$$

where N are the total numbers of outputs in j and y are the true answers.

3 Implementation

Code can be found on [6].

4 Results

Results for two differnt cases are being presented, one the one-dimensional Ising Model and another for the the two-dimensional Ising model.

4.1 $1D$ Ising model

4.2 $2D$ Ising model

5 Discussion

6 Conclusion

Appendices

Put stuff like Bootstrapping, kfold CV, OLS/Ridge/Lasso regression here.

A Refreshing linear regression

A.1 Ridge regression

A.2 Lasso regression

B Bootstrapping

C k-fold Cross-Validation

References

[1] Aurélien Géron. *Hands-on machine learning with Scikit-Learn and TensorFlow: concepts, tools, and techniques to build intelligent systems.* " O'Reilly Media, Inc.", 2017.

[2] Warren S. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, Dec 1943. ISSN 1522-9602. doi: 10.1007/BF02478259. URL `https://doi.org/10.1007/BF02478259`.

[3] P. Mehta, M. Bukov, C.-H. Wang, A. G. R. Day, C. Richardson, C. K. Fisher, and D. J. Schwab. A high-bias, low-variance introduction to Machine Learning for physicists. *ArXiv e-prints*, March 2018.

[4] Magnus Nielsen. How the backpropagation algorithm works. http://neuralnetworksanddeeplearning.com/chap2.html, October 2018.

[5] Lars Onsager. Crystal statistics. i. a two-dimensional model with an order-disorder transition. *Physical Review*, 65(3-4):117, 1944.

[6] Mathias Vege. Fysstk4155: Project 2 github repository. https://github.com/hmvege/FYSSTK4155-Project2, October 2018.