Project 2 - FYS-STK 4155

MACHINE LEARNING WITH ISING MODEL

November 12, 2018

Ingrid Utseth
Polina Dobrovolskaia
University of Oslo

Abstract

This project presents a study of both one- and two-dimensional Ising model data with different machine learning techniques.

By modeling one-dimensional Ising data we saw that linear regression presented the best model with R^2 score of 1.0 using Ordinary Least Squares, while regression with neural networks was a more demanding and less accurate in its performance.

For the classification of phase of the two-dimensional Ising model, however, neural networks with a log cross-entropy cost function gave us the best model with accuracy around 99.9%. Classification with logistic regression was not representative enough to be used on two-dimensional Ising data set. Accuracy here was at best 65.7%.

Contents

1 Introduction

Machine learning is a study and construction of algorithms that can learn and and make predictions on data. This makes such techniques an essential and powerful tool in data analysis. In this project we have studied, modeled and evaluated Ising data sets using supervised training with three machine learning techniques: linear regression, logistic regression and neural networks.

This project is organized as follows. First we begin with an introduction of our problem and define a way we will generate data for our first studies. In the next step we introduce and study linear regression on the self-generated one-dimensional data set. Here we will have a goal of finding the coupling constant, J, between spin states and energy, that illustrates our data in a best possible way. We evaluate the performance of our model with R^2 score, MSE, bias and variance.

Then, we will move our focus to binary classification of a two-dimensional Ising data set by the application of logistic regression and stochastic descent methods. Here we will apply use data that we got from our source article "A high bias low-variance introduction to Machine Learning for physicists" by Mehta et al. [9]

Moving on to neural networks, we will perform regression analysis of the one-dimensional Ising model with our own neural network script and code supported by a high-level neural networks API - `Keras`, followed by binary classification of a two-dimensional Ising model.

At the end we will compare and evaluate which method suits best for interpretation of one- and two-dimensional Ising data.

2 Theory

2.1 Producing Data for the Ising model (ID)

First we considered the one-dimensional Ising model.

$$E[\hat{s}] = -J \sum_{j=1}^{N} s_j s_{j+1}, \tag{1}$$

where $s_j = \pm 1$ represents spin variables. To generate the data, we set the coupling coefficient J to equal 1. Using python's `random_choice` function, we can generate a matrix with $L \times N$ spin variables and use these to calculate a set of points of forming pares of energy with its corresponding state, $(E[\hat{s}^i], \hat{s}^i)$.

Our first task is to train a linear regression model on the generated data set which represents the linear correspondence between X and energy, E coupled with a constant, J,

$$E = \hat{X} \cdot \hat{J} \tag{2}$$

where the matrix $\hat{X}$ presents two-body interactions between the spin variables and $\hat{J}$ is a matrix with non-local coupling strengths $\hat{J}_{jk}$.

2.2 Coupling constant with linear regression

In this part we will study one dimensional Ising model. We will use randomly generated data set with 10 000 data points which represents spin states together with their corresponding energies and use linear regression algorithms to predict the coupling strengths J_{jk}.

In our generated data set, all $J_{jk} = 1$. We used the three following regression methods: Ordinary Least Square (OLS), Ridge regression and the Lasso regression [2]. For OLS, singular value decomposition (SVD) is a necessary extra step in order to avoid singular matrices. We used the `linalg.pinv`-method from the `sklearn` python module. For the shrinkage methods, Ridge and the Lasso, we compared the mean squared error, MSE, and R^2-score for several shrinkage variables. Finally, we compared all the methods using the bootstrap algorithm to estimate bias and variance in addition to MSE and R^2.

2.3 Bootstrap method

One of the widely used re-sampling methods is the bootstrap method. Bootstrap can be used to find an estimate of a parameter from a limited data set by repeatedly re-sampling the original data set (i.e. randomly selecting values from the original data set until we have formed a data set of the same size) and taking the mean of the parameters calculated from each data set.

2.3.1 Implementation of bootstrap method

We extract 20 percent of our data that we will validate our model with, and call it **test data**. The rest, 80 % of the data, will be collection basis of data for our training. This is the training basis that we will re-sample `n_bootstrap` times using `sklearn.utils.resample`. The parameter `n_bootstrap` also specifies number of training sessions we employ on our model, each time with a re-sampled combination of training data.

2.4 Logistic regression

Using logistic regression, we want to classify the phase (ordered or disordered) of a sample lattice of $L \times L = 40 \times 40$ spins in two dimensions. We represent the lattices as flattened arrays of 1600 elements. The data set used consists of labeled 10000 samples for 16 temperatures between 0.25 and 4.0 (in units of energy).

We are only interested in positive ($y_i = 1$) or negative ($y_i = 0$) answers. To get a probability representation of whether the data point y_i belongs to a category $0, 1$, we use the sigmoid function [4],

$$f(\hat{X}_i\hat{\omega}) = \frac{1}{1 - \exp(\hat{X}_i\hat{\omega})} \tag{3}$$

Our objective is to find optimal parameters $\hat{\omega}$ that best classifies each object (in this case the spin lattices). In logistic regression, the optimal weights minimizes the cost function,

$$C(\hat{X}\hat{\omega}) = -\sum_{i=1}^{n} \Big(y_i(\hat{X}_i\hat{\omega}) - \log(1 + \exp(\hat{X}_i\hat{\omega}\Big), \tag{4}$$

where $\hat{X}_i$ is the vector containing the spins for sample i. To add a constant ω_0 to the model, we added a column of ones to $\hat{X}$.

The minimum of the cost function can be found by gradient descent methods. The weights are updated according to the derivative of the cost function:

$$\frac{\partial C}{\partial \hat{\omega}} = -\hat{X}^T(\hat{y} - \hat{p}), \tag{5}$$

where $\hat{p}$ is a vector of fitted probabilities. The basic idea is that by following the gradient of a convex function, we will eventually reach the minimum of the function.

The first method we implemented was the Standard Gradient Descent Algorithm. In every step i we update the weights using the gradient of the cost function with all the samples in the training data set and a learning rate η:

$$\hat{\omega}_i = \hat{\omega}_{i-1} - \eta \cdot \Delta C(\hat{X}\hat{\omega}) \tag{6}$$

After a predetermined number of iterations, the gradients hopefully converges to zero and we can classify objects using the sigmoid function with a $threshold = 0.5$.

When dealing with larger datasets, the Standard Gradient Descent Algorithm can be quite slow, as it calculates the gradient using all the data points in the training set. It is also sensitive to the learning rate η. A small learning rate results in many iterations needed for convergence, but if the learning rate is too large, there will be no convergence at all.

To improve the logistic regression, we implemented Stochastic Gradient Descent (SGD). This method introduces a random element, which will hopefully aleviate the issues with the learning rate, and since it only calculates the gradient for a single sample each iteration, it is also faster than Standard Gradient Descent. The SGD builds on the idea that the cost function can be calculated as the sum of n data points, meaning that the gradient is also a sum of n data points. For a predetermined number of iterations, so-called epochs, we calculate the gradient for n random data points and update the weights:

$$\hat{\omega}_i = \hat{\omega}_{i-1} - \eta \sum_{i}^{n} c_i(\hat{X}_i\hat{\omega}) \tag{7}$$

It is possible to use n/M mini-batches of size M or rather than n mini-batches of size 1.

For our SDG implementation, we chose to use a fluctuating learning rate which depends on the epoch number i and the current mini-batch iteration m. In the beginning, we would like to take larger steps, but after some time we are presumably close to convergence and prefer a smaller learning rate.

$$\eta_j = \frac{t_0}{t_1 + (iM + m)} \tag{8}$$

t_1 and t_0 are predetermined constants, m is the current batch number. The accuracy of the model can be calculated using the following equation:

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

where I is the indicator function returning 1 if the prediction y_i is equal target t_i and 0 otherwise.

2.5 Prediction by Neural Network

2.5.1 Regression with Neural Network

In this section we will examine regression analysis by the use of neural networks. According to the Universal approximation theorem, a feed forward neural network can be a model for any input data. [1, 9] The neural network is therefore a powerful modelling method that has ability to learn complex and complicated data and thus present non-linear functional mappings between inputs and outputs. A fully-connected neural network is a processing system that is composed of one or more layers with a number of interconnected elements that we refer to as nodes or neurons. The input flows into first layer where it is firstly modified with weight and bias parameters at each node and then processed through an activation function. For regression analysis the last layer produces output values that we wish to match our target values. We are able to train our network thus by minimizing the error between the output values (prediction) and the target values.[1]

Our network will be organized into 2 layers; one hidden layer and one output layer with different activation functions in each layer. The weights will be initialized with random numbers between 0 and 1. Bias will initially be set to a small number or even to zero.

The output from a layer l, the input to layer $l+1$, is denoted a_i^l and is calculated as follows:

$$a_i^l = f^l(x_i \cdot w_i + b_i) = f^l(z_i), \tag{10}$$

where f^l is the activation function in layer l.

Having calculated the final output $a_i^L = y_i$ in the feed-forward step, we update the weights and biases in each layer in an attempt to minimize the cost function:

$$C(\hat{W}^L) = \frac{1}{2}\sum_{i=1}^{N}(y_i - t_i)^2, \tag{11}$$

In this section we will model a representation one dimensional Ising model. From now on, y_i represents our predicted energies and t_i is our target, the true energies defined by the equation (1). To minimize the cost function, we need to find the zero point of its derivative [1],

$$\frac{\partial C\hat{W}^L}{\partial W_{jk}^L} = (y_i - t_i)\frac{\partial \hat{y_j}^L}{\partial W_{jk}^L} \tag{12}$$

The derivative is dependent on the activation function.

The data in the final layer is not processed through any activation function. Thus we are interested in direct comparing of the values that our network develop with the target values.

$$\hat{y} = \hat{a}^L = \hat{a}^{L-1}\hat{W}^L + b^L \tag{13}$$

and it's derivation with respect to the weights is simply a. To update the weights and the biases in the outer layer, we use:

$$\hat{W}^L = \hat{W}^L - \eta(\hat{a}^{L-1})^T(\hat{y} - \hat{t}) \tag{14}$$

Depending on the choice of the activation function in the inner layers the weights and biases will thus be updated in the following way,

$$w^l_{jk} = w^l_{jk} - \eta \delta^l_j a^{l-1}_k \tag{15}$$

$$b^l_{jk} = b^l_j - \eta \delta^l_j \tag{16}$$

For the regression analysis of one dimensional Ising model have we decided to apply Exponential Linear Unit function as activation function for our inner/hidden layers. The weights and the biases in the inner layers are updated according to the derivation of the Exponential Linear Unit (ELU) function 18.

$$ELU(x) = \begin{cases} \alpha(e^x - 1), & \text{if } x \leq 0 \\ x, & \text{otherwise} \end{cases} \tag{17}$$

$$ELU'(x) = \begin{cases} ELU(x, \alpha) + \alpha, & \text{if } x \leq 0 \\ 1, & \text{otherwise} \end{cases} \tag{18}$$

According to the theory [1] the weights and the bias are updated with the error in calculated in the concecutive layer and the input values from the previous layer, as see in equations (15) and (16).

2.5.2 Classification with Neural Networks

Much like in section 2.3, we will attempt to classify the phase of a sample 40×40 lattice of two-dimensional spins. This time, we will do the classification with a neural network.

The basic idea is similar to section 2.4. The feed forward process is much the same as previously. The biases, on the other hand, seemed to be a very sensitive parameter, thus we decided to ignore to get more precise classification. As will be presented in the results and often stated by the other neural network scholars is that a positive bias term will make the the node value shoot up to the extreme, while a negative terms affected in the opposite direction.

In our classification algorithm, we struggled with the bias being too large. Since we are in general dealing with probabilities, even a small bias can have a immense influence on the output from a node. In this fashion, we decided to remove all biases and run the feed forward algorithm as follows,

$$\hat{a}^l = f(\hat{a}^{l-1}\hat{W}) = f(z^l), \tag{19}$$

where $\hat{a}^l$ is the output from layer l and f is the activation function in that layer.

In the outer layer, we used the softmax function to calculate the probabilities. This function has the advantage that the output probabilities will always sum up to 1. [8]

$$a^L_j = \frac{e^{z^L_j}}{\sum_{k=1}^{K} e^{z^L_k}} \tag{20}$$

In the denominator, we sum up over all the output neurons, whose number correspond to the number of classes (2 in our case). Then we use a simple threshold for our final predicted values.

We experimented with two different activation functions in the single hidden layer we used. The sigmoid function and it's derivative: [6]

$$f(x) = \frac{1}{1+e^{-x}} \tag{21}$$

$$f'(x) = f(x)(1-f(x)) \tag{22}$$

and the Scaled Exponential Unit function (SELU) with it's derivative [7]:

$$f(x,\lambda,\alpha) = \lambda \begin{cases} \alpha(e^x - 1), & \text{if } x \leq 0 \\ x, & \text{otherwise} \end{cases} \tag{23}$$

$$f'(x,\lambda,\alpha) = \begin{cases} f(x,\lambda,\alpha) + \lambda\alpha, & \text{if } x \leq 0 \\ \lambda, & \text{otherwise} \end{cases} \tag{24}$$

Both equations will only return positive values. However, sigmoid has a known problem with vanishing gradients; for small inputs it will return values close to 0 which makes learning slow.[7] For positive inputs, SELU has a constant gradient λ that should speed up the learning process. SELU was chosen rather than the very similiar ELU function (SELU with $\lambda = 1$) because a gradient $= 1$ was simply too large.

The cost function used was the log cross-entropy classification cross function.

$$C = -\frac{1}{n}\sum_{i=1}^{n}(ylog(a) + (1-y)log(1-a)) \tag{25}$$

Substituting $a = f(z^{L-1}) = a^{L-1}$ into this expression and taking the derivative with regards to the weight $\hat{w_j}$ [8]:

$$\frac{\partial C}{\partial w_j} = -\frac{1}{n}\sum_{i=1}^{n}(\frac{y}{f(z)} - \frac{(1-y)}{(1-f(z))})f'(z)a_j, \tag{26}$$

This can be simplified into a rather elegant gradient used to update the weights

$$\frac{\partial C}{\partial w_j} = -\frac{1}{n}\sum_{i=1}^{n} a_j^{L-1}(a^L - y) = -\frac{1}{n}\sum_{i=1}^{n} a^{L-1}\delta^L, \tag{27}$$

where δ^L is the error in layer L.

3 Results and Discussion

3.1 Linear regression on generated data

One of the striking observations we encountered was that the one-dimensional Ising model is superbly represented by linear regression. An evident reason is a linear dependence between spin states and the its corresponding energy, which allows us to set initial linear dependence as a feature for our linear predictor function.

We generated a noiseless data set based on the linear Ising model according to equation 1 with $J = 1$.

MSE	6.05655e-28
R^2	1.00000
Bias	3.32791e-29
Variance	5.73692e-28

Table 1: Modeling characteristics of one-dimensional Ising model with OLS and bootstrap re-sampling method.

As illustrated i table 1 linear regression with Ordinary Least Squares, the MSE, bias and variance of our trained model is very small, while R^2 - score is 1.0.

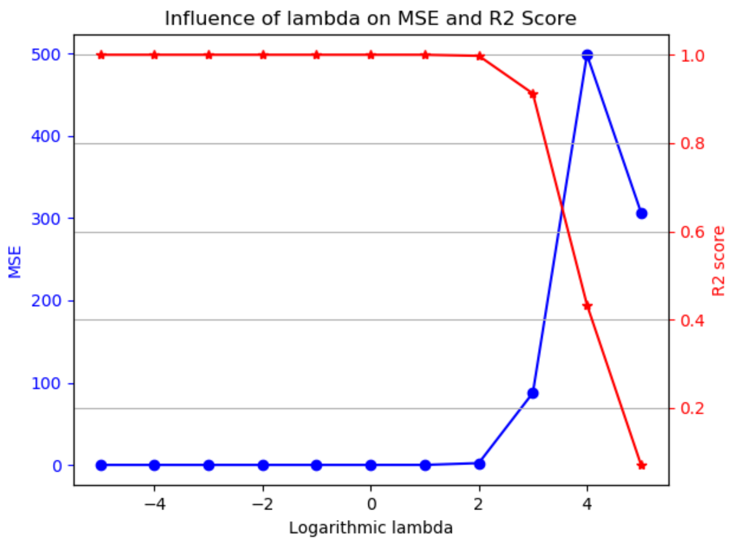

Figure 1: Ridge regression for one-dimensional Ising model. Mean Square Errors and R^2-score plotted against the logarithmic λ.

Figure 1 portrays the influence of the λ-value in linear Rigde regression study on the MSE and R^2-score for one-dimensional Ising model. As λ increases from 10^2 to 10^3 we observe a sharp decline in the R^2-score. This is an expected behaviour, due the fact that employed λ-regularizer no longer serves as effective, constraining the magnitude of the parameter vector learned from the data. [9]

After electing a favorable λ candidate, $\lambda = 10^2$, we continue to explore modeling of one-dimensional Ising model with Rigde regression. Being presented with extensive amount of generated data, we chose to run 20 rounds of re-sampling for training. In table 2 we see the statistical metrics from our modeling with Ridge regression. Both MSE and bias are of order of magnitude 10^{-3}, which means that overall error of our predictions is small. Variance is of order of magnitude 10^{-4}, which again points to the fact that or model with one feature is indeed a good representation of our data set.

Similarly, we investigated the influence of different regularization penalties, α values, on the Lasso regression fit. In figure 2 we see the that in the interval

MSE	0.00275
R^2	0.99925
Bias	0.00251
Variance	0.00024

Table 2: Modeling characteristics of one-dimensional Ising model with Ridge regression with $\lambda = 10^2$ and bootstrap re-sampling method.

from 10^{-2} to 10^1 MSE grows quickly from it minimum to it maximum, while R^2 score shrinks in the same interval. The two meet in roughly in the middle, $\alpha = 10^{-1}$. It follows that the Lasso regression gives a better representation of our data with an α value lower than the optimal λ value for Ridge regression.

We selected an optimal $\alpha = 10^{-1}$ and performed a 20 rounds with bootstrap re-sampling with a goal to achieve a best model representation for our data sample

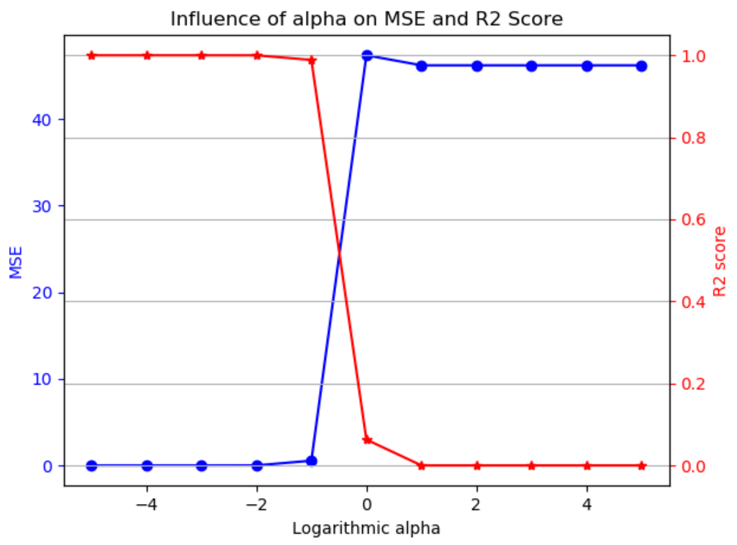

Figure 2: Mean Square Errors and R^2-score plotted vs the logarithmic α.

MSE	0.40998
R^2	0.98966
Bias	0.40812
Variance	0.00186

Table 3: Modeling characteristics of one-dimensional Ising model with Lasso regression with $\alpha = 10^{-1}$. and bootstrap re-sampling method.

In the table 3 we see MSE, R^2 score, bias and variance for Lasso regression model with $\alpha = 10^{-1}$ for our generated data for one-dimensional Ising model. The R^2 score is in the vicinity of one, while MSE of around 0.41 which indicates that that mean squared error for the best Lasso regression model has some fit discrepancy.

For our studies with linear regression we generated data sets consisting of pairs of energy and a corresponding state. The basis for this was an linear Ising model equation (1), which states that there is a linear correspondence between states and their energy, coupled with coupling constant J. Thus, the choice of the polynomial features as simply linear for the three regression studies was justified with the fact that we already knew the expression between input (states) and output (energies) values.

In this way it was reasonable for us to get such a good performance factors for all three regression methods.

Comparing our results to the results of Mehta et al. [10], we see that we have better R^2 score for our test data. The reason for this may lie in the fact that Mehta scholars used less amount of data for training than we did. On the other hand, we have approached the same optimal λ-value ($\lambda = 10^2$) for Ridge Regression and α-value ($\alpha = 10^{-1}$) for Lasso. In overall, linear regression techniques produce models with exceptionally good performance, as well as the code runs very fast.

3.2 Logistic Regression

For reasons of computational speed, we used 10% of the data set from Mehta et al.[12]. The samples were randomly chosen using sklearn's `train_test_split`-function. Using the same function, the data set was further split into 80% training data ($n = 10400$ samples) and 20% test data.

Initially, we ran a standard gradient descent algorithm on the training data with learning rates $\eta = \{10^{-5}, 10^{-6}, 10^{-7}\}$ and `max_iter`=1000. The weights were initialized using numpy's `random.randn`-function. The results is displayed in table 4, with accuracy calculated according to equation 9.

η	Accuracy
$\eta = 10^{-3}$	0.49731
$\eta = 10^{-4}$	0.50923
$\eta = 10^{-5}$	0.50346
$\eta = 10^{-6}$	0.49231

Table 4: Accuracy of standard gradient descent method for classification of two-dimensional Ising model

Overall the results are very poor - random guesses would have given us much the same results. Wishing to improve the results, we used stochastic gradient descent with an adaptive learning rate (equation 8). Number of epochs `n_epochs` was set to 100, and in each epoch we update the weights with a randomly chosen sample $M = n$ times, where n is the total number of samples.

Once again, the results in table 5 are not satisfactory to truly predict future samples. It should also be noted by looking at the accuracy for every 10 epochs, we could observe that the models hardly improved through the iterations; the accuracy had no steady increase even when we increased the number of iterations or epochs.

t_0	t_1	Accuracy
$t_0 = 5$	$t_1 = 50$	0.50731
$t_0 = 5$	$t_1 = 100$	0.49423
$t_0 = 5$	$t_1 = 250$	0.50423
$t_0 = 5$	$t_1 = 500$	0.51077
$t_0 = 5$	$t_1 = 1000$	0.48077
$t_0 = 5$	$t_1 = 10000$	0.50423
$t_0 = 5$	$t_1 = 100000$	0.50654

Table 5: Accuracy of stochastic gradient descent method for classification of two-dimensional Ising model.

Curious to see if a constant would improve the model, we added a column of ones to the initial data. The weight ω_0 belonging to this column will be updated along with the other weights and hopefully help the model in the right direction. We repeated the calculations with this modification.

η	Accuracy
$\eta = 10^{-3}$	0.40846
$\eta = 10^{-4}$	0.43923
$\eta = 10^{-5}$	0.48577
$\eta = 10^{-6}$	0.63231
$\eta = 10^{-7}$	0.35885

Table 6: Accuracy of standard gradient descent method with a constant ω_0 for classification of two-dimensional Ising model.

Table 6 shows that the learning rates $\eta = \{10^{-3}, 10^{-4}, 10^{-5}\}$ are certainly too high. This was supported by our observations of the accuracy developing throughout the iterations decreased rather than showing any tendencies of increase. $\eta = 10^{-5}$ improves the result drastically. Further changes in the learning rate did not give us any prospering results; the weights were changing too slowly.

The model with one constant weight seemed promising, thus we repeated the calculations for stochastic gradient descent and various adaptive learning rates.

t_0	t_1	Accuracy
$t_0 = 5$	$t_1 = 50$	0.63654
$t_0 = 5$	$t_1 = 100$	0.64962
$t_0 = 5$	$t_1 = 250$	0.65654
$t_0 = 5$	$t_1 = 500$	0.64269
$t_0 = 5$	$t_1 = 1000$	0.64000
$t_0 = 5$	$t_1 = 10000$	0.66885

Table 7: Accuracy of stochastic gradient descent method with a constant ω_0 for classification of two-dimensional Ising model.

As seen in table 7, this algorithm gave us better results, more or less independent of t_1. From observing the accuracy development as the script runs, it seems like it converges quite quickly and 100 epochs are unnecessary. Figure 3 shows how the model with $t_1 = 10000$ converges after less than 20 epochs. It seems that increasing the number of epochs will not significantly increase the accuracy of the classification.

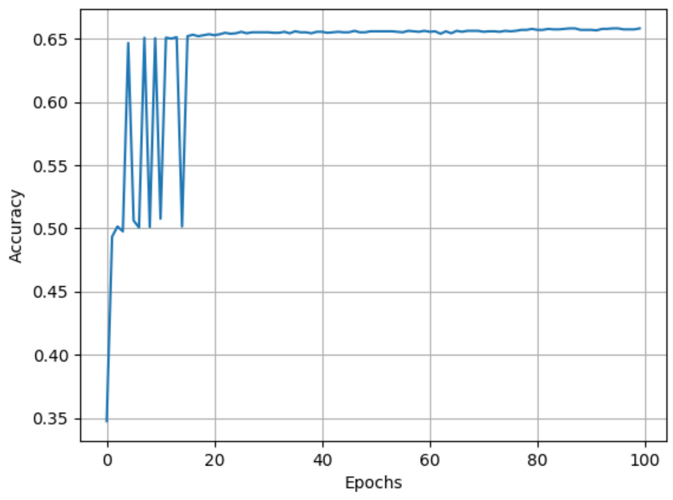

Figure 3: Classification accuracy of logistic models against number of epochs.

Even the best accuracy reached, around 65%, is not good enough to confidently classify future samples; the margin of error is too large. We only used a small part of the data sets in these models, for reasons of computational speed. It is possible that increasing the number of input samples would increase the accuracy. Using batches with a larger size than 1 could also have helped decrease the runtime and possibly increase the accuracy.

Mehta et al. found [13] that the best achieved stochastic gradient descent accuracy for test data was around 68%. This supports the results achieved with our own code. In Mehta's studies `sklearn`'s `linear_model.LogisticRegression` was used, which a good benchmark to assess that this performance is limited in interpretation of our data. The logistic regression should not be a primary method for classification of the two-dimensional Ising model data. However, if we used a larger data, which results in a much longer computational time, we would presumably produce a better acuracy. This would be a task for our future studies.

3.3 Neural Networks

3.3.1 Regression with Neural Networks

The most tedious task was setting up our neural network for regression analysis and to choose a reliable activation function. After some testing with the sigmoid function (common activation function used in neural networks) without getting convergence of error, we settled on the ELU-function. The small α-parameter seemed to help prevent the error from "exploding".

First, we randomly divided our data set with $n = 10000$ samples into training data (80 %) and testing data (the remaining 20 %). We used one hidden layer with $k = 1600$ nodes (equal to the number of features in the data) and the ELU function with $\alpha = 10^{-3}$ as the activation function. In our outer layer we had no activation function. We initialized the weights in the outer layer with numpy's `random.randn`-function, which returns random from the standard normal distribution, and the weights in the hidden layer were all set to 1 (attempts to run the script with random hidden weights did not converge). The biases in the hidden and the outer layer were set to 0.1.

To find a suitable learning rate η, we used a batch size = 1000 and calculate the MSE and R^2-score after 20 epochs. The results in table 8 indicates that $\eta = 10^{-6}$ is a good choice. We used this learning rate in all subsequent calculations.

Learning rate	**MSE**	$\mathbf{R}^2$
$\eta = 10^{-5}$	22455.60000	-564.00000
$\eta = 10^{-6}$	45.90000	-0.20000
$\eta = 10^{-7}$	89.40000	-1.20000

Table 8: MSE and R^2-score after 20 epochs for different learning rates.

In order to hopefully speed up the calculations, we updated the weights using batches of various size B, 200 epochs and n/B iterations in each epoch. We calculated the MSE and R^2 score using the test data.

Batch size	**MSE**	$\mathbf{R}^2$
1000	11.66139	0.70660
500	11.61617	0.70774
100	11.57950	0.70867

Table 9: MSE and R^2-score after 200 epochs.

The computation of these number were slow and the result in table 4 is significantly worse than the regression methods we used in section 3.1.

Observing the MSE for each epoch, we were led to believe that the network was still converging after 200 epochs. In an attempt to improve the results, we increased the number of epochs to 200. As the batch size did not seem to significantly affect the results, we used the largest batch size = 1000 and thus fewer iterations in each epoch. Figure 4 displays the MSE for every 50 epochs. It is clear that the network converges slowly but surely, and is still converging even after 500 epochs. The final results are displayed in table 10.

MSE	2.48306
$\mathbf{R}^2$	0.93753

Table 10: MSE and R^2 score after 500 epochs.

While still not as good as the results we got using linear regression methods in section 3.1, the results have improved. We predict that if we increased the

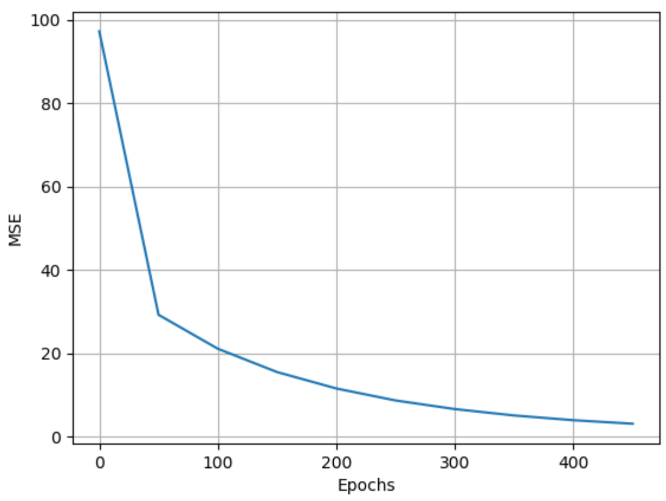

Figure 4: MSE of the model against number of epochs.

number of epochs, we would eventually get results similar to the linear regression methods results. This is unfortunately not feasible in this context, as the network is too slow. A way to possibly speed up the convergence rate could be to use an adaptive learning rate, similar to equation 8.

There are many options for activation functions that could be used in the hidden layer, and it is possible that we would have gotten a better result replacing ELU with something else. We do believe using a single hidden layer was a good choice. We know that we can get good results using linear regression. More layers makes the model more non-linear, and we could end up overfitting the model. The calculations would also be slower. The way the weights and biases are initialized also affects the results. For the future studies it would be interesting to explore how they affect the rate of convergence and the final result.

3.3.2 Keras - Regression with neural networks

In this section we will employ deep learning library that enables to implement neural network both for regression and classification. Here we will look at the neural network study for regression analysis.

As seen from table 11 we are able to get quite good R^2 score for regression models obtained with Keras. However, it is accompanied by poor MSE, bias and variance values, which makes the models not as representative of our data as we wish. During simulation of data we tested varying number of epochs, training batches and number for bootstrap re-sampling. The time to run the code with Keras is somewhat faster than our own code. We also easily got quite good R^2, while MSE, bias and variance seem to be not satisfactory to properly characterize our one-dimensional Ising data.

All in all, it is possible to get somewhat good regression model with neural

epochs	50	500	5	20
bootstrap	10	1	10	100
batches	1000	1000	2000	100
MSE	74.74112	75.30031	80.89519	18.732670
R^2	0.97347	0.98318	0.96332	0.99711
Bias	74.37185	75.30031	79.704997	6.00561
Variance	0.36927	0.00001	1.19020	0.45331

Table 11: Analysis of one-dimensional Ising model with Keras neural networks by regression. **Epochs** represents number of learning iterations. **Bootstrap** represents number of re-sampling sesssions per epoch. **Batches** represents number of batches used for training models.

network. However it is more complicated than it needs to be. Thus, for linear models it is best to simply employ linear regression, which in this case was both faster and more accurate.

3.3.3 Neural Networks Classification

Similar to section 3.3.1, we used one hidden layer with 1600 nodes. Softmax and SELU with $\lambda = 10^{-3}$ and $\alpha = 10^{-3}$ were our activation functions in the outer and hidden layer respectively. Since the data originally only had a single value y_i for each output, we processed the data to match two output values per sample. The weights in both layers were initialized using numpy's `random.randn`-function and as discussed previously we used no biases.

To determine a suitable learning rate η, we used a batch size = 1000 and evaluated the accuracy after 20 epochs. The results are displayed in table 12.

Learning rate	**Accuracy**
$\eta = 10^{-1}$	0.72577
$\eta = 10^{-2}$	0.99885
$\eta = 10^{-3}$	0.99769
$\eta = 10^{-4}$	0.86962

Table 12: Accuracy after 20 epochs for different learning rates

$\eta = 10^{-2}$ produced a very good result, much better than logistic regression with gradient descent in section 3.2. It also converges fast, as seen in figure 5. The accuracy was calculated according to equation 9.

We were curious how SELU compared to another common activation function: the sigmoid function. We let sigmoid be the activation function in the hidden layer, used batch size = 1000 and ran through 20 epochs for various learning rates.

Table 13 shows that the results are worse than with the SELU function for every learning rate we tried and figure 6 shows that the network is more unstable, with the accuracy jumping widely between the iterations. It seems that SELU, with a small λ, is a better choice for activation function.

Although Mehta et al. [11] also use the softmax activation function in the outer layer, they use at most 1000 neurons in the hidden layer and the Rectified

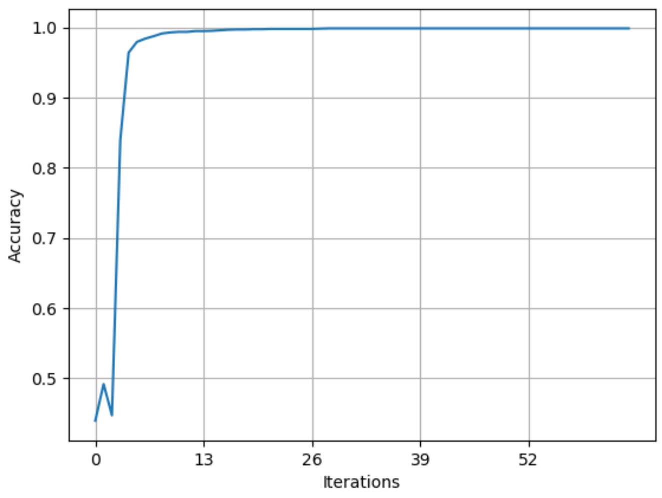

Figure 5: Accuracy of the model against number of iterations, with 5 epochs.

Learning rate	Accuracy
$\eta = 10^{-4}$	0.62000
$\eta = 10^{-5}$	0.66038
$\eta = 10^{-6}$	0.65500
$\eta = 10^{-7}$	0.65346

Table 13: Accuracy after 20 epochs for different learning rates

Linear Unit (RELU) function as activation function in the hidden layer. Their network is therefore not really comparable to ours. However, their best result with accuracy=99.9% is very close to our best result. With its fast convergence and high accuracy, our neural classification network is superior to our previous attempts at classification with logistic regression and gradient descent in section 3.2.

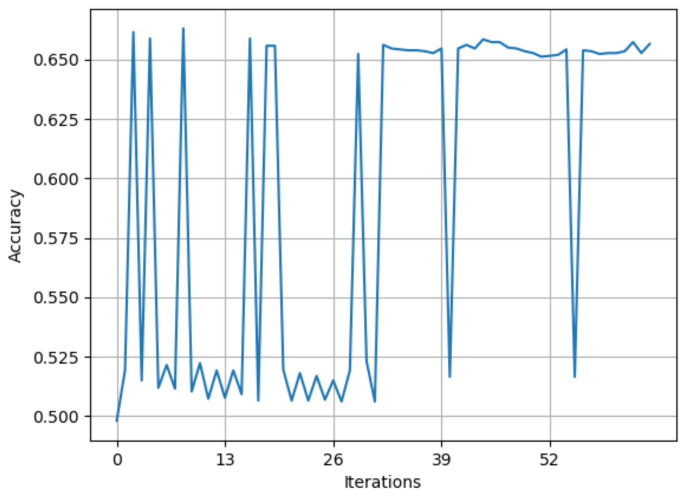

Figure 6: Accuracy of the model against number of iterations, with 5 epochs.

4 Conclusion

The primary aim of this project was to apply, understand and assess the performance of different modelling methods and techniques. This study considers linear regression methods, logistic regression and neural networks. For sample training we have utilized bootstrap method.

We performed studies with two types of data sets,

- one-dimensional Ising model (self-generated)
- two-dimensional Ising model (Mehta et al.)

The data sets above were studied with regression analysis and classification analysis, respectively. For regression we used Ordinary Least Squares, Ridge regression, Lasso regression and regression with neural networks. All three linear regression techniques gave good representation of our data, nonetheless a clear performance winner was the model obtained with OLS regression (with R^2 score of 1.0 and MSE, bias and variance roughly of order of magnutude 10^{-29}). Since we already were aware of the linear dependence between spin states and energies, we set up our features for modelling in that way when training.

The regression with neural networks was both time consuming and did not give as good a representation of our data as linear regression. This draws us to conclusion that neural networks are not always the best and simplest approach for modelling data.

For the classification of the two-dimensional Ising model, neural networks are superior in characterizing two-dimensional Ising data compared to logistic regression. For classification by neural networks, we obtained an accuracy of 99.9 %. This was supported by the findings of Mehta et al.[9]

For our future studies, we would be interested in examining how number of epochs, batches and neurons affect the performance of the neural network. Also, both for regression and classification problems we employed a fully connected neural network with only one hidden layer. We would have liked to look into how an increase of the number of layers with fewer neurons affect the performance of the model. Finally, we feel that further testing of the neural networks with other activation functions, for instance RELU utilized by Mehta et al [9], could yield better results in future studies.

5 Python code

https://github.com/ingridut/-FYS-STK4155-Project2

6 Appendix

6.1 Mathematical Formulas

6.1.1 Mean Squared Error

$$MSE(\hat{y}, \tilde{\hat{y}}) = \frac{1}{n}\sum_{i=0}^{n-1}(y_i - \tilde{y}_i)^2 \tag{28}$$

$\hat{y}$ is the true value, $\tilde{\hat{y}}$ is the predicted values.

6.1.2 R^2-score

$$R^2(\hat{y}, \hat{\hat{y}}) = 1 - \frac{\sum_{i=0}^{n-1}(y_i - \tilde{y}_i)^2}{\sum_{i=0}^{n-1}(y_i - \bar{y})^2} \tag{29}$$

$\hat{y}$ is the true value, $\tilde{\hat{y}}$ is the predicted values. $\bar{y}$ is the mean value of the true values $\hat{y}$.

6.1.3 Bias

$$bias = \frac{\sum_{i=0}^{N-1}(y_i - \bar{\tilde{y}}_i)^2}{N} \tag{30}$$

where $\bar{y}_i$ will be the average predicted value of y at point i.

6.1.4 Variance

$$bias = \frac{\sum_{i=0}^{N-1}(\tilde{y}_i - \bar{\tilde{y}}_i)^2}{N} \tag{31}$$

where $\bar{y}_i$ will be the average predicted value of y at point i.

Bibliography

[1] Hjorth-Jensen, M. 2018. *Data Analysis and Machine Learning: Neural networks, from the simple perceptron to deep learning and convolutional networks.* Available at:<`https://compphysics.github.io/MachineLearning/doc/pub/NeuralNet/html/NeuralNet.html`> [Accessed 30 October 2018]

[2] Dobrovolskaia, P, Utseth, I. 2018. *OLS, Ridge and Lasso Regression with Resampling*

[3] Hjorth-Jensen, M. 2018. *Data Analysis and Machine Learning: Linear Regression and more Advanced Regression Analysis.* Available at:<`https://compphysics.github.io/MachineLearning/doc/pub/Regression/pdf/Regression-minted.pdf`> [Accessed 10 October 2018]

[4] Hjorth-Jensen, M. 2018. *Data Analysis and Machine Learning: Logistic Regression.* Available at:<`https://compphysics.github.io/MachineLearning/doc/pub/LogReg/pdf/LogReg-minted.pdf`> [Accessed 17 October 2018]

[5] *ML Cheatsheet: Activation Functions.* Available at:<`https://ml-cheatsheet.readthedocs.io/en/latest/activation_functions.html`> [Accessed 2 November 2018]

[6] Chakraborty, A. *Derivative of the Sigmoid function* Available at:<`https://towardsdatascience.com/derivative-of-the-sigmoid-function-536880cf918e`> [Accessed 5 November 2018]

[7] Pedamonti, D. 2018. *Comparison of non-linear activation functions for deep neural networks on MNIST classification task* Available at:<`https://arxiv.org/pdf/1804.02763.pdf`> [Accessed 5 November 2018]

[8] Nilsen, M. 2018. *Neural Networks and Deep Learning* Available at:<`http://neuralnetworksanddeeplearning.com/index.html`> [Accessed 5 November 2018]

[9] Mehta, P., Wang, C., Day, A. G. R., and Richardson, C. *A high-bias, low-variance introduction to Machine Learning for physicists.*

[10] Mehta, P., Wang, C., Day, A. G. R., and Richardson, C. *Notebook 4: Linear Regression (Ising)* Available at:<`https://physics.bu.edu/~pankajm/ML-Notebooks/HTML/NB_CVI-linreg_ising.html`> [Accessed 5 November 2018]

[11] Mehta, P., Wang, C., Day, A. G. R., and Richardson, C. *Notebook 12: Identifying Phases in the 2D Ising Model with TensorFlow* Available at:<`https://physics.bu.edu/~pankajm/ML-Notebooks/HTML/NB_CIX-DNN_ising_TFlow.html`> [Accessed 7 November 2018]

[12] Mehta, P. *Data sets - Ising Model.* Available at:<`https://physics.bu.edu/~pankajm/ML-Review-Datasets/isingMC/`> [Accessed 15 October 2018]

[13] Mehta, P. *Notebook 6: Phases of the Ising Model with Logistic Regression.* Available at:<`https://physics.bu.edu/~pankajm/ML-Notebooks/HTML/NB_CVII-logreg_ising.html`> [Accessed 15 October 2018]

[14] B. Efron *THE 1977 RIETZ LECTURE. Bootstrap Methods: Another Look at the Jacknife.*