Project 1:
Studying Classification and Regression problems in Machine Learning using the Ising model

Hoa Dinh, Jose Pablo Linares, Benedict Castro

February 20, 2020

Abstract

The project aims to study regression and classification problems in Machine Learning using the Ising model. First, values of the coupling constant for the energy of the one-dimensional case were determined using three linear regression methods, which are ordinary least square regression, Ridge regression, and LASSO regression. The results obtained agree well with those in Ref. [1]. Afterwards, a resampling method known as bootstrapping was included. In this setup, the bias-variance tradeoff was analysed and R^2 scores were calculated for two different sizes of training data. In the second part of the project, logistic regression was used to classify the phase of the two-dimensional Ising model. Here, another resampling technique was employed, which is cross-validation. Finally, the neural networks were used for both linear regression and logistic regression cases. Considering the model where 60% of data was trained, both linear regression and neural networks result in very high accuracy for test data, which is about 0.99. On the other hand, for a model with only 20% of training data, neural networks method is shown to have better results compared to logistic regression.

1 Introduction

A problem in Machine Learning usually composes of three ingredients: a data set $D = (\mathbf{X},\mathbf{y})$ which contains a matrix of of independent variables (design matrix, $\mathbf{X}$) and a vector of dependent variables (response variables, $\mathbf{y}$); a model $f(\mathbf{x};\theta)$, which is a function predicting an output from a vector of input variables $\mathbf{x}$ and parameters θ; and finally a cost function $C(\mathbf{y},\text{f}(\mathbf{x};\theta))$ that allows us to evaluate the performance of the model. In order to fit the model, we need to find values of parameters θ such that the cost function is a minimum [1].

When response variables are continuous and no prior knowledge on the form of the model function $f(.)$ is available, the relation between $\mathbf{X}$ and $\mathbf{y}$ is commonly assumed to be linear [2]:

$$\mathbf{y} = \mathbf{X}\beta + \epsilon = \tilde{y} + \epsilon, \tag{1}$$

where the approximation $\tilde{y} = \mathbf{X}\beta$, β is called the *regression parameter vector*, and ϵ is the *error vector* of our approximation. Here, our error vector and regression parameter vector are unknown.

In Ordinary Least Squares linear regression (OLS), optimal values of parameters β_i are obtained by minimizing the L_2 norm of the difference between the response y_i and the predictor $\tilde{y}_i$. Thus, the cost function can be written as

$$C_{OLS}(\beta) = \frac{1}{n}||\mathbf{y} - \mathbf{X}\beta||_2^2. \tag{2}$$

The optimal regression parameter vector is found to be

$$\beta_{OLS} = \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{y}. \tag{3}$$

This is well defined only if $\left(\mathbf{X}^T\mathbf{X}\right)^{-1}$ exists. To solve this singularity problem, Ridge regression and LASSO regression can be used, in which we introduce a *hyperparameter* denoted as λ. The cost functions in Ridge and LASSO regression methods are

$$C_{Ridge}(\mathbf{X},\beta) = \frac{1}{n}||\mathbf{y} - \mathbf{X}\beta||_2^2 + \lambda||\beta||_2^2. \tag{4}$$

$$C_{Lasso}(\mathbf{X},\beta) = \frac{1}{n}||\mathbf{y} - \mathbf{X}\beta||_2^2 + \lambda||\beta||_1. \tag{5}$$

When the form of outcomes is discrete, we need to consider classification problems. If there are two possible outcomes, we can apply logistic regression. In this case, by maximizing the probability of seeing the observed data, the cost function can be defined. As a result, the minimization of the cost function leads to a non-linear equation in the parameters β [3].

In this project, the Ising model was used to study Regression and Classification problems. Specifically, for the one-dimensional ($1D$) Ising model the energy coupling constants were determined using three linear regression methods mentioned above. Moreover, bias-variance analysis was considered using the bootstrap resampling method. Later on, logistic regression was applied in order to classify the phase for the two-dimensional ($2D$) Ising model. Theoretically, the critical temperature for the phase transition is approximately $\frac{kT_c}{J} \approx 2.26$, where k is the Boltzmann's constant. Below the critical temperature, the system is classified to be in a ferromagnetic phase, while above this value, the net magnetization is zero. The performance of the model was evaluated using accuracy scores. Furthermore, logistic regression was performed without and with cross validation. Finally, neural networks (NN) were included for both Regression and Classification cases.

The next section of our report is devoted to describe in details the methods and algorithms that have been used and developed. Afterwards, the results obtained are analysed and compared with those obtained in Ref. [1]. Lastly, the report is closed with the discussions and conclusions.

2 Methods and Algorithms

2.1 $1D$ Ising model

Determining coupling constants

Considering a $1D$ Ising model with nearest neighbor interaction, if there is no external field acting on the lattice, the energy of a given spin configuration can be written as

$$E[\hat{s}] = -J\sum_{i=1}^{L} s_i s_{i+1}, \tag{6}$$

in which L is the total number of spins (in our case $L = 40$), s_i is the spin of each site, s_i=±1, and J is a coupling constant which represents the interaction strength between nearest-neighbor spins.

The data were generated with $J = 1$. Our data set contains all spin configurations and the corresponding energies, $D = \{(S^i,\text{E}[S^i])\}$, where $i = 1, 2, 3, ..., n$. Here, n is the number of spin configurations.

Choosing a spin model with pairwise interaction between every spin pair, the energy for a spin configuration S^i for this model is

$$E_{model}[S^i] = -\sum_{j=1}^{L}\sum_{k=1}^{L} J_{jk} s_j^i s_k^i. \tag{7}$$

Our first task was to determine the coupling constant matrix $J_{j,k}$ using linear regression.
The above equation can be recast in the form of linear regression:

$$E^i_{model} \equiv \mathbf{X}^i\mathbf{J} = X^i_p J_p, \tag{8}$$

where $\mathbf{X} = \{s_j^i s_k^i\}_{j,k=1}^L$ and p={j,k}. Here, the index i runs over all samples in the data set, which is $n = 10000$ in this case. Moreover, as we have omitted the minus sign, we should expect $\mathbf{J}$ to have negative elements.

In order to obtain models that have good prediction potential, it is important to split a data set into training and test data sets. The models are fit using the training data, while the test data is used to judge the performance of the models. The best model is the one which minimizes the test error [1].

In this project, the fitting was conducted using scikit-learn. The algorithm for determining the coupling constant matrix $\mathbf{J}$ using OLS, Ridge, and LASSO regression methods is described as follows:

```
Define input as:
    n <- number of spin configurations
    L <- number of sites
    states <- Independent data # spin states
    energies <- Dependent data # Ising energy
    lambdas <- hyperparameter #(Ridge and Lasso only)
Using numpy to create lambdas:
    lambdas = np.logspace(-4, 5, 10)
Generate states as a matrix of (-1) and (1) with dimension of (n, L)
Generate energies from states with J=-1 for two nearest spins
Create the design matrix X using a function f that couple all the spin pairs:
    X <- f(x)
Split data in test and training data using train_test_split in scikit-learn
with any desired ratio:
    X_train, energies_train <- training data
    X_test, energies_test <- test data
for _lambda in range(len(lambdas)):
    Compute fit coefficients using scikit-learn:
        clf <- sklearn.linear_model. modelused().fit(X_train, energies_train)
        # modelused <- LinearRegression()/ Ridge()/ Lasso()
        coef <- clf.coef_
    Obtain the coupling constant matrices by reshaping coefficients:
        J <- coef.reshape((L,L))
```

Bias-variance analysis

Assume that the true data is generated with a normally distributed noise with zero mean value and variance σ^2 .

$$\mathbf{y} = f(\mathbf{x}) + \epsilon. \tag{9}$$

The test data error is expressed as

$$Error = E[(\mathbf{y} - \tilde{y})^2], \tag{10}$$

where $\tilde{y} = \mathbf{X}\beta$ as we have defined before.
This equation can be decomposed as

$$Error = E[(\mathbf{y} - \tilde{y})^2] = E(\mathbf{y} - E[\tilde{y}])^2 + \frac{1}{n}\sum_i (\tilde{y}_i - E[\tilde{y}])^2 + \sigma^2. \tag{11}$$

The first term is called the bias

$$Bias^2 = E(\mathbf{y} - E[\tilde{y}])^2. \tag{12}$$

The bias expresses the difference between the mean value of our prediction and the true value.
The second term is called the variance

$$Variance = \frac{1}{n}\sum_i (\tilde{y}_i - E[\tilde{y}]). \tag{13}$$

It tells us the fluctuation of our prediction due to the finite-sample effects.
The testing error can be written as

$$Error = Bias^2 + Variance + Noise. \tag{14}$$

Therefore, in order to minimize the testing error, our model needs to results in low variance and low bias (see figure 1).

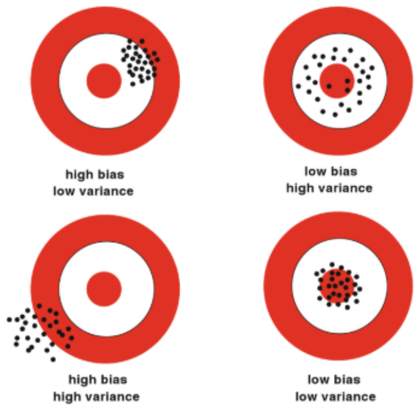

Figure 1: Drawing of the relation between the prediction power of a model (black dots) and bias-variance. The prediction is closest to the true values (red circle in the center) if both bias and variance are low (figure taken from Ref. [4]).

In this problem, we made the bias-variance analysis using bootstrap resampling. In modern statistics, resampling methods play a crucial role. The main idea of these methods is to draw repeatedly different samples from a training data set, and the fits are then done for each new sample. This approach may allows us to obtain information which would not be available from fitting the model only once using the original training sample [5]. Bootstrapping is a nonparametric approach to statistical inference that substitutes computation for more traditional distributional assumptions and asymptotic results [6]. The aim of the bootstrap is to estimate the sampling distribution function of estimator $\hat{\theta}$ by the relative frequency of $\hat{\theta}$ [7]. The main steps of bootstrap [8]:

1. Draw with replacement n numbers for the observed variables $\mathbf{x} = (x_1, \cdots, x_n)$.
2. Define a vector $\mathbf{x}^*$ containing the values which were drawn from $\mathbf{x}$.
3. Using the vector $\mathbf{x}^*$ compute $\hat{\theta}^*$ by evaluating $\hat{\theta}$ under the observations $\mathbf{x}^*$.
4. Repeat this process k times.

Creating the data set using the same algorithm as before, the algorithm for bias-variance analysis using bootstrap can be added as follows:

```
boots <- number of bootstraps
for _lambda in range(len(lambdas)):
    for j in range(boots):
        Resample data:
            X_, energies_ <- resample(X_train, energies_train)
        Fit data for each new sample using scikit-learn:
            clf <- sklearn.linear_model.modelused.fit(X_, energies_)
        Compute prediction data corresponding to each sample as
             energies_pred_test <- clf.predict(X_test)
             energies_pred_train <- clf.predict(X_)
        Obtain R2 score, test error, bias, and variance for each iteration:
            R2_train <- R2 score of training data
            R2_test <- R2 score of test data
            error <- mean squared error of test data
            Bias <- bias of test data
            Variance <- variance of test data
    Compute the average value of R2 score, error, bias, and variance over all
    iteration of bootstraps
```

Setting up neural networks

Artificial neural networks are computing systems which can learn to perform tasks by considering examples, generally without being programmed with any task-specific rules. Such systems consist of layers of connected neurons, or nodes or units. The feed-forward neural network (FFNN) was the first and simplest type of artificial neural networks, in which the information is always fed forward through the layers. A network contains of three or more layers (an input layer, one or more hidden layers, and an output layer) which composes of neurons that have non-linear activation functions (figure 2). Such networks are often called multilayer perceptrons (MLPs). An activation function in FFNN need to have the following properties: non-constant, bounded, monotonically increasing, and continuous. The typical activation functions are logistic Sigmoid and hyperbolic tangent [9]. In our problem, logistic Sigmoid was used.

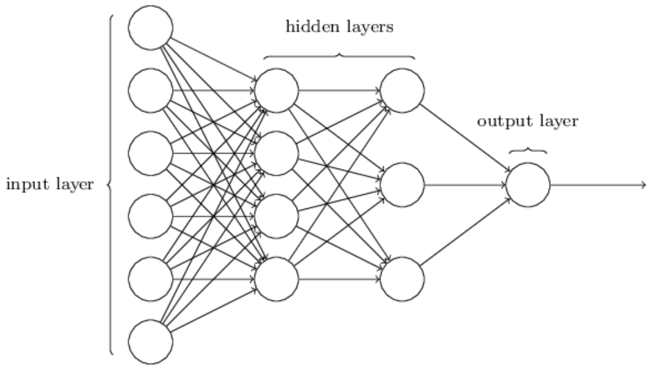

Figure 2: Four-layer network containing two hidden layers (figure taken from [10]).

Given an activation function at $l-th$ layer f^l, the output at a node in this layer is

$$y_i^l = f^l(u_i^l) = f^l\left(\sum_{j=1}^{N_{l-1}} w_{ij}^l y_j^{l-1} + b_i^l\right), \tag{15}$$

in which the index j runs over all nodes in the preceding layer. ω_{ij} and b_i are called *weight* and *bias*, respectively. We need to find weights such that the error of our model is as small as possible. This can be solved using *back propagation algorithm.*

The cost function is defined as

$$C(\omega, \mathbf{b}) = \frac{1}{2}\sum_{i=1}^{n}(y_i - t_i)^2, \tag{16}$$

where t_i are targets, and y_i are the outputs of the network.

Minimization of the cost function can be done using the following steps:

1. Initialize the NN with random value of weights ω_{ij}^l and biases b_i^l.

2. Propagate forwards to find outputs y_i and compute the corresponding $C(\omega, \mathbf{b})$.

3. Use back propagation algorithm to find optimal ω_{ij}^l and b_i^l.

4. Repeat steps 2 and 3 until $\frac{\partial C}{\partial \omega} = 0$ and $\frac{\partial C}{\partial \mathbf{b}} = 0$.

In optimizing the cost function, *gradient descent* is one of the most common used method classes. This method involves a parameter called *learning rate* η, which controls how big a step we take towards the minimum. Moreover, in order to constrain the size of the weights so that they do not grow out of control, a term with regularization parameter λ is added to the cost function [9].

In this project, we did not develop the code based on all steps mentioned above. Instead, we used the available function in scikit-learn. In particular, *MLPRegressor* was used as the outputs are continuous. Moreover, the accuracy for training data and test data were obtained as a function of η and λ. The optimal values of η and λ are those which correspond the best accuracy for test data. From this, the optimal weights and biases were found from $coefs\_$ in scikit-learn.

Again, we repeated the algorithm that we used to create the data, and added the following part:

```
epochs <- number of epochs
n_hidden_neurons <- number of neurons in the hidden layer
Create an array of learning rates using numpy:
    eta_vals = np.logspace(-5, 1, 7)
Create an array of hyperparameters using numpy:
    lmbd_vals = np.logspace(-5, 1, 7)
for i, eta in enumerate(eta_vals):
    for j, lmbd in enumerate(lmbd_vals):
        Fit the model with neural network using MLPRegressor in scikit-learn:
               dnn = sklearn.neural_network.MLPRegressor(hidden_layer_sizes=(
                        n_hiden_neurons),  activation='logistic', alpha=lmbd,
                        learning_rate_init=eta,max_iter=epochs, solver='adam')
               dnn.fit(X_train, energies_train)
        Compute accuracy for training data
        Compute accuracy for test data
```

2.2 $2D$ Ising Model

As we mentioned before, if outcomes are not continuous but have the form of discrete variables, we cannot use linear regression. In the case of the $2D$ Ising model, we need to identify phase (e.g. odered/disordered). This is a classification problem with binary outcomes. Therefore, we can use the logistic regression method.

In logistic regression, the probability that a data point $\mathbf{x}_i$ belongs to $y_i = \{0, 1\}$ is written as [1]

$$\begin{aligned} P(y_i = 1|\mathbf{x}_i, \hat{\beta}) &= \frac{1}{1 + e^{-\mathbf{x}_i^T \hat{\beta}}}, \\ P(y_i = 0|\mathbf{x}_i, \hat{\beta}) &= 1 - P(y_i = 1|\mathbf{x}_i, \hat{\beta}). \end{aligned} \tag{17}$$

In the above equations, $\hat{\beta}$ are the weights learned from the data.

The cost function is defined using Maximum Likelihood Estimation (MLE) [1], and is expressed as

$$C(\hat{\beta}) = \sum_{i=1}^{n} \left(y_i \log p(y_i = 1|\mathbf{x}_i, \hat{\beta}) + (1 - y_i) \log \left[1 - p(y_i = 1|\mathbf{x}_i, \hat{\beta})) \right] \right). \tag{18}$$

The right hand side of the above equation is called *cross entropy*.

In this project, we dealt with a $2D$ square lattice with $L \times L$ sites, where $L = 40$. The phases that had to be classified are ordered ($T < T_C$) and disordered ($T > T_C$). Firstly, the data was divided into ordered, disordered, and critical data. The ordered and disordered data were split into training and test data. Afterwards, the model was fit using scikit-learn. Finally, accuracy scores for training data, test data, and critical data were computed. The algorithm for classifying the phase of $2D$ Ising model is described as follows:

```
Define data as:
    X <- Design matrix #spin configurations
    Y <- Labels #ordered or disordered
Separate data as
    X_ordered, Y_ordered <- data with T < 2.0
    X_critical, Y_critical <- data with 2.0 =< T =< 2.5
    X_disordered, Y_disordered <- data with T > 2.5
Split ordered with disordered data in test and training data
    X_train, Y_train <- training data
    X_test, Y_test <- test data
Scale the data using StandardScaler
Define values of $\lambda$ to be used using numpy
    lmbdas = np.logspace(-5,5,11)
for lmbda in enumerate(lmbdas):
    Compute fit coefficients using scikit-learn
            clf <- sklearn.lineal_model.Logistic_Regression(C = 1.0/lmbda)
                  .fit(X_train,Y_train)
    Use this coefficients to calculate accuracies for each value of lambda
            train_accuracy[lmbda]=clf.score(X_train,Y_train)
            test_accuracy[lmbda]=clf.score(X_test,Y_test)
            critical_accuracy[lmbda]=clf.score(X_critical,Y_critical)
```

When this algorithm was implemented, there were a couple of issues that had to be addressed. First, the available hardware had issues running the data. This resulted in a limitation on the size of the training sample, where 20% of the data was used for training instead of a more ideal 80%, even when using one fourth of the data Mehta *et al.* utilized for training, the accuracies obtained were similar. Another problem that was encountered was that the computers would 'freeze' until some of the memory was cleared at the end of the code. Therefore, since the runtime of the code is of about one hour, debugging became very time-consuming.

The next step was to include a resampling method, which is cross-validation (CV). CV is used to structure the data splitting in order to avoid the unbalanced effect on either model building or prediction evaluation due to random split. For CV with k folds, CV splits the data set into k groups. Then for each group, it decides which one to use for test data, and the rest is used to train. The model is then fit with training data, and the corresponding evaluation score is recorded [3].

The algorithm with CV is

```
    Define data as:
    X <- Design matrix #spin configurations
    Y <- Labels #ordered or disordered
Separate data as
    X_ordered, Y_ordered <- data with T < 2.0
    X_critical, Y_critical <- data with 2.0 =< T =< 2.5
    X_disordered, Y_disordered <- data with T > 2.5

Scale the data using StandardScaler
Define values of $\lambda$ to be used using numpy
    lmbdas = np.logspace(-5,5,11)
We define the number of folds as
    Kfold <- number of folds
for lmbda in enumerate(lmbdas):
    for k in range(Kfold):
        Select training and test data from the ordered and disordered sets
            X_train <- X_ordered[k] + X_disordered[k]
            Y_train <- Y_ordered[k] + Y_disordered[k]
```

```
                X_test <- X_ordered[i!=k] + X_disordered [i=!k]
                Y_test <- Y_ordered[i!=k] + Y_disordered [i=!k]
                # here k is symbolic, it means we are taking the subset k
                # i!=k refers to all sets that are not for training
            Compute fit coefficients using scikit-learn
                clf <- sklearn.lineal_model.Logistic_Regression(C = 1.0/lmbda)
                       .fit(X_train,Y_train)
            Use this coefficients to calculate accuracies for each value of lambda
                train_accuracy[lmbda][k]=clf.score(X_train,Y_train)
                test_accuracy[lmbda][k]=clf.score(X_test,Y_test)
                critical_accuracy[lmbda][k]=clf.score(X_critical,Y_critical)
        Find the accuracies by doing a mean calculation
            train_accuracy_CV[lambda] = mean(test_accuracy[lmbda])
            test_accuracy_CV[lmbda] = mean(test_accuracy[lmbda])
            critical_accuracy[lmbda] = mean(critical_accuracy[lmbda])
        Calculate the variance of the accuracies
            train_var_CV[lambda] = variance(test_accuracy[lmbda])
            test_var_CV[lmbda] = variance(test_accuracy[lmbda])
            critical_var_CV[lmbda] = variance(critical_accuracy[lmbda])
```

Here, there were some challenges to overcome. Similar as when no CV was done, memory errors appeared if we tried to do a proper resampling. Therefore, 20% of data was used for training instead of 80%. Another issue was that the CV couldn't be run using the scikit learn libraries, else memory errors would be shown. Therefore, it was coded from scratch. The only thing that had to be taken special care of was to not take a sample counting only ordered or disordered arrays, so the splits were done first on the disordered and ordered data for separate and then joined together. As it will be discussed in the following section, CV isn't needed for this case. On the other hand, the code runs more smoothly when doing the split 'by hand', as the computers wouldn't freeze as much. However, since it is so slow (around one hour and a half), debugging was time-consuming.

Finally, NN was included using *MLPClassifier* in scikit-learn. The accuracy scores for training data, test data, and critical data were computed and compared with those obtained using logistic regession. The algorithm is as follows:

```
Define data as:
    X <- Design matrix #spin configurations
    Y <- Labels #ordered or disordered
Separate data as
    X_ordered, Y_ordered <- data with T < 2.0
    X_critical, Y_critical <- data with 2.0 =< T =< 2.5
    X_disordered, Y_disordered <- data with T > 2.5
Split ordered with disordered data in test and training data
    X_train, Y_train <- training data
    X_test, Y_test <- test data
Scale the data using StandardScaler
Define values of $\lambda$ to be used using numpy
    lmbd_vals = np.logspace(-5, 1, 7)
Define values of $\eta$ to be used using numpy
    eta_vals = np.logspace(-5, 1, 7)
Define parameters for the neural network
    n_hidden_neurons <- number of neurons
    epochs <- number of epochs
for eta in enumerate(eta_vals):
    for lmbd in enumerate(lmbd_vals):
        Define the method
            dnn <- sklearn.neural_network.MLPClassifier(hidden_layer_sizes=
                    (n_hidden_neurons), activation='logistic', alpha=lmbd,
                    learning_rate_init=eta, max_iter=epochs, solver='adam')
        Fit the data
            dnn.fit(X_train, Y_train)
        Obtain the accuracy scores
            accuracy[lambda][eta] <- dnn.score(X_test, Y_test)
```

In contrast with the previous method, the available hardware runs this algorithm and returns the results in about one hour, which is very similar to the runtime of the logistic regression. However, the computers used did not freeze when doing the calculations.

3 Results and Discussions

3.1 Determination of the $1D$ Ising model coupling constant using linear regression methods

Coupling constant

The first result obtained is the coupling constants of pairwise interaction for all spin pairs in a 1D lattice with the number of spins chosen to be $L = 40$. The coupling constant matrices using OLS, Rigde, and LASSO regression methods are shown in figure 3. The result is shown with four values of λ chosen to be the same as those in Ref. [1] so that it is convenient to compare with their results.

Since OLS regression method does not involve λ, $J_{i,j}$ does not change with λ. On the other hand, coupling constants in Ridge and LASSO regressions vary for different values of λ as we should expect from Eq. 4 and Eq. 5. As mentioned in section 2.1, our data were generated with $J_{i,j} = J = 1$, where we only considered the interaction between nearest-neighbor spins s_j and s_{j+1}. However, when computing $\mathbf{J}$ using linear regression, we took into account all interaction pairs s_i and s_j, in which both indices run over all number of sites. Therefore, we did not only include $J_{i,i+1}$ but also $J_{i,i-1}$. As we can see in figure 3, $J_{i,j} \approx -0.5$ for interaction between two nearest-neighboring pair in OLS and Ridge, which means that there is a symmetry of coupling constant . However, this symmetry is not respected in LASSO regression, and we get $J_{i,i+1} \approx -1$ for $\lambda = 10^{-2}$.

It is stated that in general LASSO tends to give sparse solutions [1]. Hence, this leads to many zero elements in the coupling constant matrix. This can be understood by considering figure 4. The blue concentric ovals are the contours representing the least squares regressors, while the red shaded regions represent the constraint functions: $|\omega_1| + |\omega_1| \leq t$ (LASSO) and $\omega_1^2 + \omega_1^2 \leq t$ (Ridge), where t is a fixed and non negative regularization strength, and ω_i are components of the regression parameter. The intersection between the contour and the shaded region is the solution for the optimization problem. As the L_1 regularizer of LASSO has sharp protrusions along the axes, the intersection is more likely to occur at the vertex. As a result, the regression parameter vector tends to be sparse [1] .

Furthermore, another important feature that can be observed in figure 3 is that the optimal value of λ for LASSO regression is $\lambda = 10^{-2}$. The performance of Ridge regression does not change significantly with $\lambda = 10^{-4}, \lambda = 10^{-2}$, and $\lambda = 1$. When λ gets large, both Ridge and LASSO regressions give poor results .

The results we obtain above for the coupling constant agree well with those obtained in figure 17 of Ref. [1].

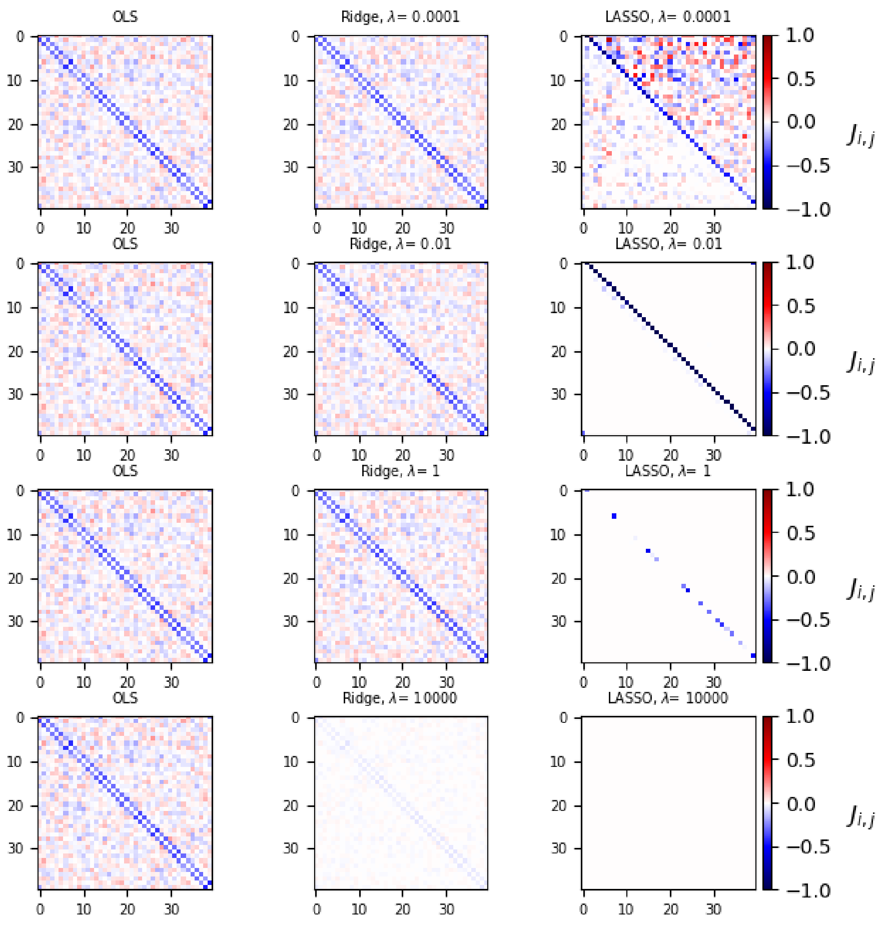

Figure 3: Coupling constants for 1*D Ising model using OLS, Ridge, and LASSO Regression methods for different values of hyperparameters* λ*. The coupling constant matrices have dimension of* $L \times L$*, where L=40.*

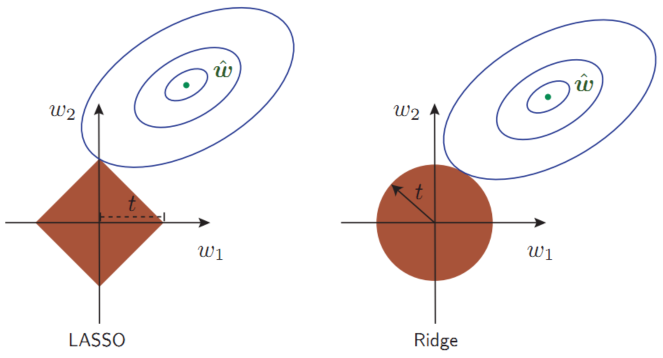

Figure 4: Illustration of LASSO (left) and Ridge regression (right). The blue concentric ovals are the contours representing the least squares regressors, and the red shaded regions represent the constraint functions. (figure taken from [11])

Bias-Variance analysis

After including bootstrap resampling, the bias-variance analysis was performed, and the R^2 scores were computed for OLS, Ridge, and LASSO regression methods.

First, we split the data set such that the size of training data set is small. To be more specific, we chose 4% of the data to be the training data. As we have $n = 10000$ spin configurations, the number of states in the training data is $n_{train} = 400$. This size of training data is the same as the one used to fit the model in Ref. [1]. In this case, the bias-variance tradeoff for the $1D$ Ising model is shown in the top panel of figure 5.

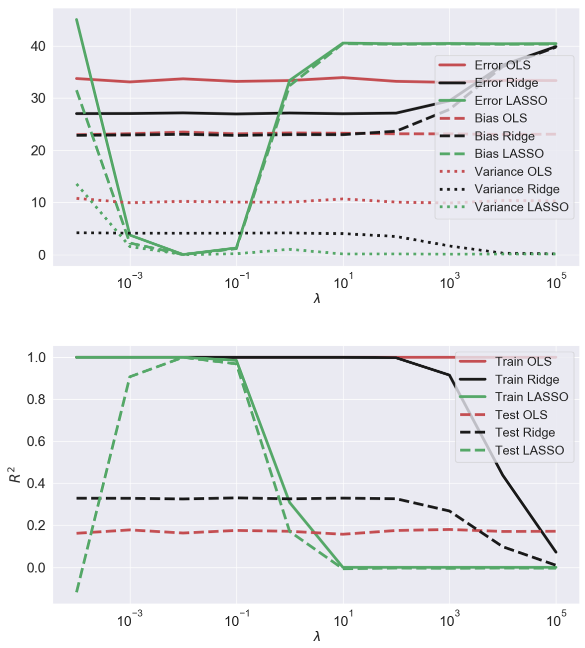

Figure 5: Bias variance tradeoff (top panel) and R^2 score for training data and test data (bottom panel) using OLS, Ridge, and Lasso regressions as a function of λ (4% data is trained).

The error, bias, and variance for OLS regression (red) are constant as they are independent of λ. For Ridge regression (black), these quantities remain constant as λ increases until it reaches the value of around 10^2. Moreover, as long as λ is not too large (less than 10^4), Ridge regression gives better result (smaller test error) than OLS regression. On the other hand, LASSO regression (green) performs very well for hyperparameter in the range $10^{-3} \leq \lambda \leq 10^{-1}$, and the smallest error is achieved at $\lambda \approx 10^{-2}$. This result agrees with what we have seen in the case of the coupling constant, that is at this optimal value of λ, the interaction is only between nearest spins.

Another quantity that can be used to evaluate our model is R^2 score.

$$R^2 = 1 - \frac{\sum_{i=1}^{n} |y_i^{true} - y_i^{pred}|^2}{\sum_{i=1}^{n} |y_i^{true} - E[y^{true}]|^2}. \tag{19}$$

The bottom panel of figure 5 shows R^2 scores for training data and test data for OLS, Ridge, and LASSO regressions as a function of hyperparameter λ. From this we can confirm again that the optimal value of λ for LASSO regression is 10^{-2} ($R^2 \approx 1$).

In Ref. [1], R^2 scores are also computed. Their R^2 scores for training data are the same as what we obtained. In addition, our R^2 scores for test data follow the same trends as those in [1]. In particular, $R^2(OLS)$ remains constant, $R^2(Ridge)$ decreases when value of λ gets large, while $R^2(LASSO)$ first increases and reaches a maximum at $\lambda = 10^{-2}$ and then decreases quickly. However, in [1] the model performs better as their R^2 scores for test data are higher than ours. This is because we used a very big size for test data (9600 spin configurations) while they used only 200 spin configurations for the test data .

Considering a bigger training data set (60%), the results are plotted in figure 6. We can see that the increase in the training data gives us better results. To be more specific, in the bias-variance tradeoff (top panel of figure 6), the $Error(OLS) \approx 0$, and $Error(Ridge) \approx 0$ until λ reaches value of about 10^4. After this value, $Error(Ridge)$ increases sharply. On the other hand, LASSO regression performs well only with $\lambda \leq 10^{-1}$. The R^2 score shown in bottom panel of figure 6 also leads us to the same conclusion about the performance of linear regression methods. Moreover, the variances for all three methods are both close to zero. This means that the errors and biases are almost the same.

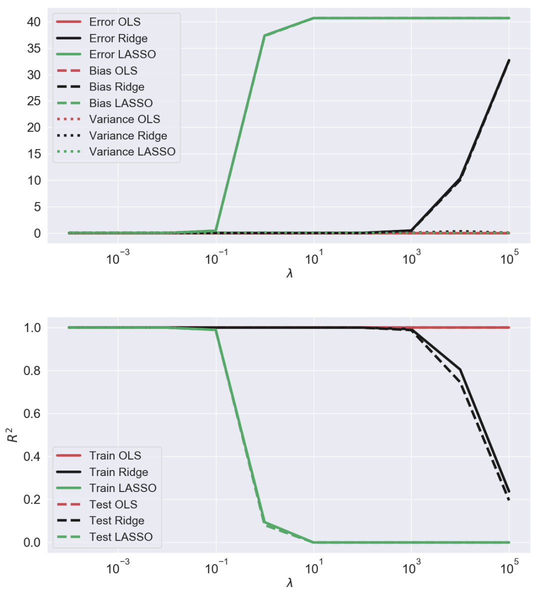

Figure 6: Bias variance tradeoff (top panel) and R^2 scores for training data and test data as a function of λ (60% data is trained).

3.2 Regression analysis of the $1D$ Ising model using neural networks

Using NN, the result obtained for test accuracy and training accuracy as a function of hyperparameter and learning rate is shown in figure 7. In this case, 60% of data was used to train. The NN was set up with one hidden layer which contains 50 neurons. Moreover, $\lambda, \eta \in \{10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 10^{0}, 10^{1}\}$. From this figure, we can conclude that the optimal values for learning rate is $\eta = 10^{-3}$, and for hyperparameter are $\lambda \leq 1$. Comparing this result to the ones using linear regression (figure 6), we can see that linear regression and NN both perform very well.

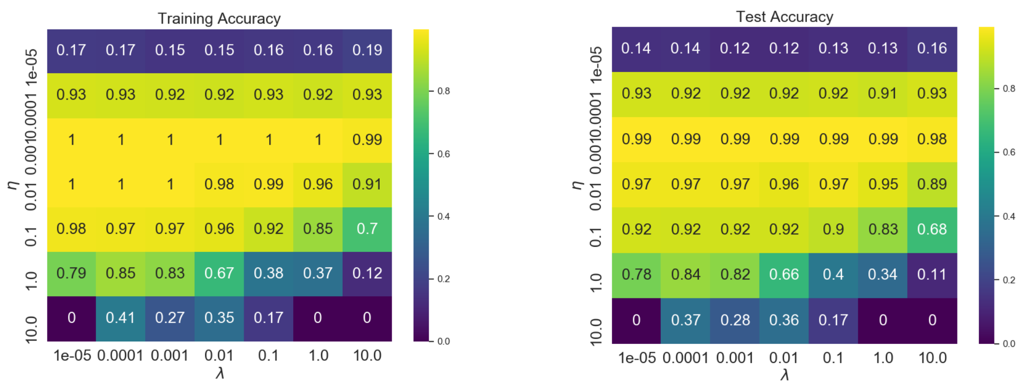

Figure 7: Training accuracy (left panel) and test accuracy (right panel) as a function of learning rate η and regularizer parameter λ (60% data is trained). Negative values are substituted by a 0.

3.3 Determination of the $2D$ Ising model phase using logistic regression methods

As stated in section 1, the $2D$ Ising model has a critical point at $kT_C/J \approx 2.26$. We would like to train a model to distinguish between the ordered phase and the disordered phase. In contrast to the previous problem, this time the outcomes are discrete and not continuous, either ordered or disordered state. As we can see on figure 8, it is very easy to distinguish phases at temperatures not close to the critical point, however close to T_C it becomes very hard to classify.

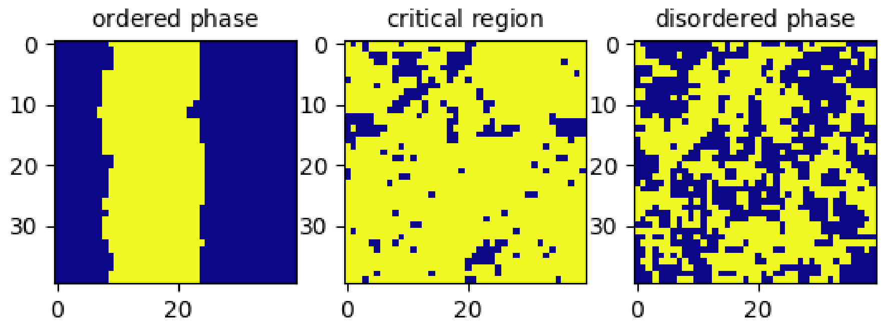

Figure 8: Samples of $2D$ Ising model for different phases and near the critical point. From left to right: ordered system at $T = 1.5$, near critical point system at $T = 2.25$ and disordered system at $T = 3.75$.

The model used to solve the classification problem was a logistic regression with an L_2 penalty that solves the optimization problem using an L-BFGS solver. The data sets of Mehta *et al.* [1] were used and were classified into three groups: ordered, near critical $(2.0 < T < 2.5)$ and disordered. Then, the model was trained using a sample of 20% of the total combined ordered and disordered data (26000 samples). This sample size was used due to memory limitations on the available hardware. After that, the accuracy of the model on the test data, the critical data and all the data were computed. Finally, the model was tested using different values of the hyperparameter λ. The values used were 10^i with $i = -5, -4, ..., 4, 5$. Afterwards, the whole process was repeated with an added cross-validation resampling technique.

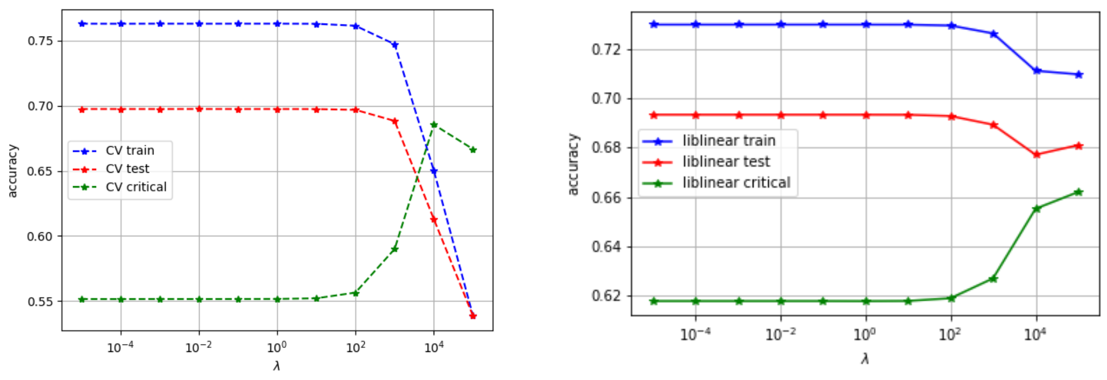

Figure 9: Left panel: accuracy of a logistic regression for the $2D$ Ising model as a function of an hyperparameter λ where the training data consists of 26000 samples, the test data consists of 134000 samples and the critical data consists of 30000 samples. The solver used for the logistic regression was L-BFGS and the penalty was of the L_2 type. Right panel: the results from Mehta et al. where the number of test and training samples are reverted.

As it can be seen on figure 9, the hyperparameter value is irrelevant unless it is greater than 10^2. For bigger values, the accuracy on the critical data improves but the test and training data worsen. This tells us that if we want to improve the accuracy on the near-critical regime, a big hyperparameter should be used, but to have a better overall accuracy then a small hyperparameter would be more appropriate. Initially, one would argue that this was just dependent on the sample taken for training. Because of this, the same regression was done using a cross-validation (CV) sampling technique where a 5-fold cross validation was chosen as this would allow us to have the same size of training data as the case without CV, which would be a $20\% - 80\%$ training-test split per iteration. By comparing figures 9 and 10, no considerable difference can be found. Due to this, the variance on the accuracy was plotted and it is astoundingly small (on the order of 10^{-4}). This means that the previous hypothesis is not valid, the improvement of the accuracy on the critical data at the cost of the accuracy in everything else seems to be something that happens. Maybe if a split of $80\% - 20\%$ of training-test data would allow to understand why this happens, moreover the hardware available is not able to handle such calculations. When comparing with Mehta *et al.* [1], they were able to do calculations with a $50\% - 50\%$ training-test split. The increased training data makes a considerable difference as it can be seen on figure 9, where the ratio of test and training is reversed. Even though they used a different solver, it should not impact the results as long as the solutions converge.

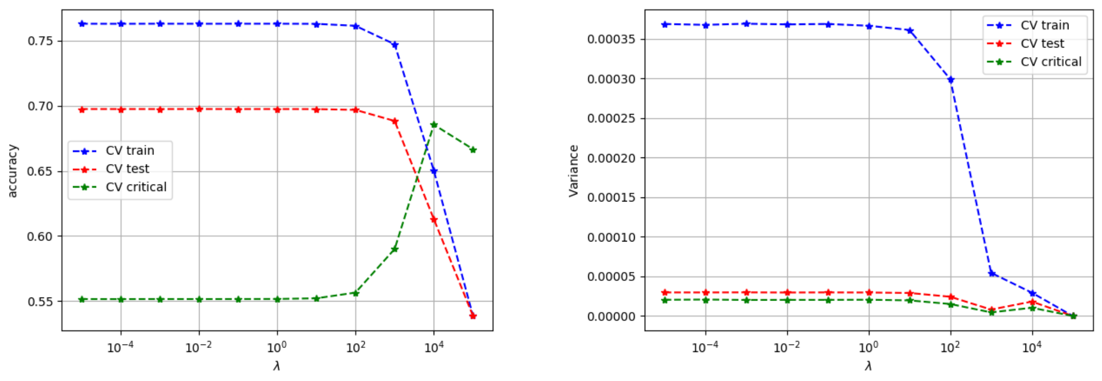

Figure 10: On the left accuracy of the logistic regression with cross validation for the $2D$ Ising model as a function of the hyperparameter λ where for each K-fold the training data consists of 26000 samples, the test data consists of 134000 samples and the critical data consists of 30000 samples. The solver used for the logistic regression was L-BFGS and the penalty was of the L_2 type. On the right, the variance of the results as a function of the hyperparameter λ

3.4 Determination of the $2D$ Ising model phase using neural networks

The same problem as the previous section is tackled using NN instead. The architecture used consisted of 1 hidden layer, 50 neurons per layer, 10 epochs and a sigmoid activation function. The data was split in the same fashion as the previous section (a 0.2 training - 0.8 test split). The algorithm was run for the values of hyperparameters λ equal to 10^i for $i = -5, -4..., 1$ and learning rates η equal to 10^j for $j = -5, -4, ..., 1$.

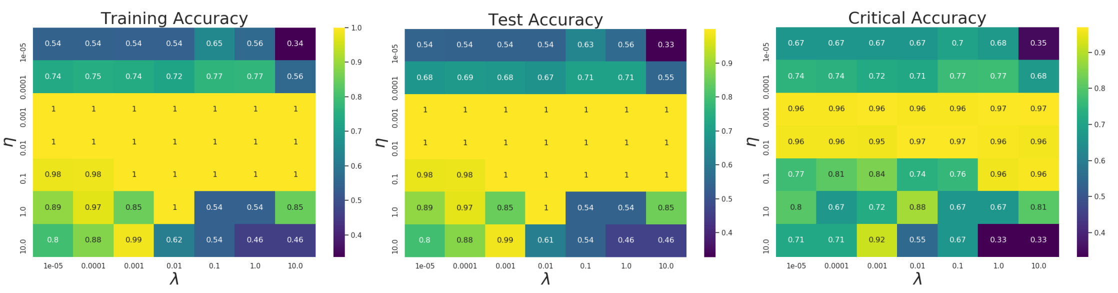

Figure 11: From left to right, accuracy for the training data, the test data and the data close to the critical point as a function of an L_2 hyperparameter λ and the learning rate η.

The results are astonishing, compared to the logistic regression approach where we obtained ≈ 0.6 accuracy for the critical data, whereas for values of $\eta = 10^{-3,-2}$ and any value of λ we get an accuracy greater or equal to 0.95, while getting a near perfect accuracy on the training and test data. Figure 11 shows that near perfect results may be achieved for many values on the hyperparameter and the learning rate. This allows to see how choosing the right parameters can make the model work better. These results show that for this problem, NN are vastly superior to a logistic regression. We compare this to the results of Mehta *et al.* [1] whose architecture consisted of 100 epochs, a variable amount of neurons, a variable learning rate, a ReLu activation functions and a stochastic gradient descent method to solve the NN. Although they didn't do an analysis on data near the critical point, a difference is noticeable since their best accuracy result on test data was 0.96. This shows that the activation function may play an important role in how the model performs.

4 Conclusions

For the $1D$ Ising model, the coupling constant for energy obtained is $J \approx -0.5$ for both Ridge and ordinary least square regression methods, while for LASSO regression method, the solution obtained is sparse. At the optimal value of hyperparamerer, $\lambda = 10^{-2}$, the LASSO regression gives $J \approx -1$. In analysing the bias-variance tradeoff and R^2 scores for training and test data, the prediction of the model is better when we used a larger training data set. Moreover, neural networks and the linear regression can both result in high accuracy scores for test data, which is about 0.99.

When solving the $2D$ Ising model classification problem, the size of the training data is also significant . Specifically, for less data it improves in the testing and training data but worsens for data near the critical point. In addition, using 26000 samples for training in the logistic regression of the $2D$ Ising model problem is enough to not worry about using resampling methods as the variance in accuracy is insignificant when using a cross-validation method. Finally, NN is far superior than using a logistic regression, as the accuracy for critical data can reach 0.96 using less resources than the logistic regression that only gets ≈ 0.6. The learning rate is more important than the hyperparameter value, for this problem.

References

[1] Pankaj Mehta, Marin Bukov, Ching-Hao Wang, Alexandre GR Day, Clint Richardson, Charles K Fisher, and David J Schwab. A high-bias, low-variance introduction to machine learning for physicists. *Physics reports*, 2019.

[2] Wessel N. van Wieringen. Lecture note on ridge regression. 2020.

[3] Morten Hjorth Jensen. Data analysis and machine learning: Logistic regression. 2020.

[4] Robert Kass, Uri Eden, and Emery Brown. *Analysis of Neural Data*. 2014.

[5] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An Introduction to Statistical Learning: with Applications in R*. 2013.

[6] John Fox. *Applied Regression Analysis and Generalized Linear Models*. 2016.

[7] Paul Lavrakas. *Encyclopedia of Survey Research Methods*. 2008.

[8] Morten Hjorth Jensen. Data analysis and machine learning: Linear regression and more advanced regression analysis. 2020.

[9] Morten Hjorth Jensen. Data analysis and machine learning: Neural networks, from the simple perceptron to deep learning. 2020.

[10] Michael Nielsen. *Neural Networks and Deep Learning*. 2019.

[11] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. *The elements of statistical learning*, volume 1. Springer series in statistics New York, 2001.