Machine Learning

FYS-STK 4155

Project 2

Kari Eriksen

Abstract

Contents

1 Introduction

In this project we will deal with different methods within the field of machine learning to solve different types of problems, such as classification and regression. We will be looking at the Ising model in both 1D and 2D. We begin with estimating the coupling constant J for the 1 dimensional case using linear regression methods and evaluating the calculations with bootstrap. In 2 dimension the Ising model experiances a phase transition when there is a change in temperature. This phase shift we can define as a classification problem and try to train the different models to identify these. We will be using both logistic regression and a neural network for this purpose.

2 Theory

2.1 Ising model

The Ising model is a mathematical model named after Ernst Ising and it describes a system of spins in a lattice where the energy of the system can be found through eq. 1. J represents the coupling constant, s_k and s_l are spins with value either $+1$ or -1 (up or don), N is the total number of spins in the system and B is the external magnetic field which in this project is zero. We therefore look at a system with energy equal to eq. 2. The spins themself represents magnetic dipole moments and the lattice allow each spin to interact with its neighbors, indicated by the symbol $< kl >$ in the sum.

$$E = -J \sum_{<kl>}^{N} s_k s_l - B \sum_{k}^{N} s_k \tag{1}$$

$$E = -J \sum_{<kl>}^{N} s_k s_l \tag{2}$$

In 1 dimension there is no phase transition and we will use regression in order to determine the coupling constant between the spins in the system.
In 2 dimensions or higher however the system goes through a phase transition with change in temperature. Below the critical temperature, $T_C \approx 2.269$, the system is what we will call an ordered state. The spins tend to be aligned which causes a net magnetization of the system, also described as a ferromagnet. Above the critical temperature the coupling constant is smaller than zero and the spins interact in an antiferromagnetic way. We wish to train our model to classify a configurations of spins as an ordered or disordered system. To do so we begin with the 1D Ising model with nearest-neighbor interactions, eq. 3.

$$E[s] = -J \sum_{j=1}^{L} s_j s_{j+1} \tag{3}$$

Now we have L number of spins in a one dimensional array. If we where to have a data set $i = 1, ..., n$ of different configurations on the form $\{(E[s^i], s^i)\}$ we could use this in training a linear model to find the coupling constant. That way we could use much of the code from project 1. To do so we use the all-to-all Ising model and notice that the energy is linear in J.

$$E_{model}[s^i] = -\sum_{j=1}^{L}\sum_{k=1}^{L} J_{j,k} s_j^i s_k^i \tag{4}$$

We can now recast this problem to a linear regression model, rewriting all two-body interactions $\{s_j^i s_k^i\}_{j,k}^L$ as a vector $\mathbf{X}^i$.

$$E_{model}^i \equiv \mathbf{X}^i \cdot \mathbf{J} \tag{5}$$

This looks much like the design matrix in project

3 Methods

3.1 Linear Regression

Linear regression is a method in statistics that predicts the response of one or several explanatory variables. It assumes a linear relationship between the dependent and independent variables. At its simplest form we could try to find the stright line between two points. This equation is fairly simple.

$$\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x + \hat{\epsilon}$$

Here $\hat{y}$ is a dependent variable, the outcome, x is an independent variable, or the predictor, and $\hat{\beta}_0$ and $\hat{\beta}_1$ the intercept and slope respectively. ϵ is the error in our prediction. The solution for $\hat{\beta}_0$ and $\hat{\beta}_1$ in this problem is best found with least square and is also fairly easy. Calculating the mean over both variables ($\bar{x}$ and $\bar{y}$) we can find the parameters that give the prediction that differs the least from the exact solution.

$$\beta_1 = \frac{\sum^n (x_i - \bar{x})(y_i - \bar{y})}{\sum^n (x_i - \bar{x})^2}$$

$$\beta_0 = \bar{y} - \beta_1 \bar{x}$$

If we have several predictors we can extend our problem to a more general case.

$$\hat{y} = f(X) = \beta_0 + \sum_{j=1}^{p} X_j \beta_j$$

Now X is a vector containing all predictors, $X^\top = \{X_0, X_1, X_2, X_3, ..., X_p\}$, β_0 is the intercept and β_j is a vector keeping all coefficients for each predictor, the parameters we are searching for. $\hat{y}$ is the predicted values of $y = f(X)$. Moving β_0 to the $\beta -$ vector and adding an extra column with 1's to the design matrix $X^\top$ we can reduce the problem to vector form and get the following. We will make use of this notation when finding solutions using least square etc.

$$\hat{y} = \hat{X}\hat{\beta} + \epsilon \tag{6}$$

3.1.1 Ordinary Least Square

The least square method selects the parameters β so that residual sum of squares (RSS) is minimized.

$$\text{RSS}(\beta) = \sum_{i=1}^{N}(y_i - x_i^\top \beta)^2$$

y_i is still the independent variable, and $x_i^\top \beta$ represents the prediction of outcome given the calculated parameter β. And the difference between these variables squared gives us the RSS of the parameter β. β is a vector $p+1$ long, the number of features (plus the intercept) in the design matrix.

This can be expressed in matrix notation, using eq. 6. To find an expression for the β-parameter we look for the minimum of the RSS, meaning we take its derivative wrt. β.

$$\text{RSS}(\beta) = (\mathbf{y} - \mathbf{X}\beta)^\top(\mathbf{y} - \mathbf{X}\beta) \tag{7}$$

$$\frac{\partial \text{RSS}(\beta)}{\partial \beta} = 0 = -2\mathbf{X}^\top(\mathbf{y} - \mathbf{X}\beta)$$

$$\mathbf{X}^\top\mathbf{y} - \mathbf{X}^\top\mathbf{X}\beta = 0$$

$$\hat{\beta} = (\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top\mathbf{y} \tag{8}$$

This is the expresion we use in the ordinary least square-method in order to find the optimal β-values. This method depends on the propertie $\mathbf{X}^\top\mathbf{X}$ being postive definit in order to be able to calculate its inverse. In case it is not we must use other method.

3.1.2 Ridge Regression

As mentioned in the section above we may come across problems where the columns in X are not linear independent, often an issue for problems in high dimesions. Then the coefficients in β are not uniquely defined through least square. This was the motivation for what would be the Ridge regression, an ad hoc solution to the singularity of $\mathbf{X}^\top\mathbf{X}$ introduced by Hoerl and Kennard (1970). They suggested adding a tuning parameter λ, i.e. a penalty to the sizes of the coefficients.

$$\mathbf{X}^\top\mathbf{X} \;\rightarrow\; \mathbf{X}^\top\mathbf{X} + \lambda\mathbf{I}$$

By doing the replacement above we are able to calculate the inverse and can again find the expression for β but this time through minimizing the penalized RSS. The solution for β is eq. 8, which is now dependent on the parameter λ.

$$\text{PRSS}(\beta) = (\mathbf{y} - \mathbf{X}\beta)^\top(\mathbf{y} - \mathbf{X}\beta) + \lambda||\beta||^2$$

$$\frac{\partial \text{PRSS}(\beta)}{\partial \beta} = 0 = -2\mathbf{X}^\top(\mathbf{y} - \mathbf{X}\beta) + 2\lambda\beta$$

$$\hat{\beta}(\lambda) = (\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^\top\mathbf{y}$$

$\mathbf{I}$ is the identity matrix, a $p \times p$ matrix, and $\lambda \in [0, \infty]$. The tuning parameter λ determines the regularization of the problem and different λ will give different solution to the regression problem. Our task will be to find an optimal parameter for our case.

$$\hat{\beta}^{ridge} = \text{argmin}_\beta \left\{ \sum_{i=1}^{N}(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{p} \beta_j^2 \right\} \tag{9}$$

We can see from eq. 9 that the method assumes our design matrix is centered, the intercept does not depend on the tuning parameter. β_0 is instead found by calculating the mean of y.

$$\beta_0 = \bar{y} = \frac{1}{N}\sum_{i}^{N} y_i \tag{10}$$

3.1.3 Lasso Regression

In 1996 Tibshirani suggested a new penalty, the Lasso. Similar to ridge regression but the difference lies in the last part.

$$\hat{\beta}^{lasso} = \text{argmin}_\beta \left\{ \sum_{i=1}^{N}(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{p} |\beta_j| \right\}$$

As for the ridge regression the intercept is given by the mean of y.

3.2 Singular value decomposition

As mentioned in 3.1.1 we are dealing with a design matrix that is singular meaning $\mathbf{X}^\top\mathbf{X}$ is not invertible. One way to solve this problem is with the use of singular value decomposition (SVD). We can rewrite $\mathbf{X}$ as the factorization of a $n \times p$ unitary matrix $\mathbf{U}$, a $p \times p$ diagonal matrix $\mathbf{\Sigma}$ and the conjugate transpose of a $p \times p$ unitary matrix $\mathbf{V}$.

$$\mathbf{X} = \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^\top$$

Now the inverse of the matrix product $\mathbf{X}^\top\mathbf{X}$ can be written as the inverse of the matrix product of the SVD.

The Moore-Penrose pseudoinverse is defined as $\mathbf{X}^+ = (\mathbf{X}^\top\mathbf{X})^{-1}$. Using the SVD we get eq. 11.

$$\mathbf{X}^+ = \mathbf{V}\boldsymbol{\Sigma}^+\mathbf{U}^\top \tag{11}$$

Now we can rewrite the solution of β in eq. 8 to the following.

$$\beta = \mathbf{V}\boldsymbol{\Sigma}^+\mathbf{U}^\top\mathbf{y}$$

3.3 Logistic regression

We now move on to the classification part of the project and look closer on logistic regression. Where regression methods focus on predicting continuous variables of a response a classification method look at outcomes in form of discrete variables (i.e. categories). Calling it regression is therefore some what misleading.
However, our aim is to make a classifier that can take in a number of spin-state congigurations and predict which categories these configurations belong to, the mentioned ordered and disordered phase.

A simple classifier that is easy to understand and has some similarities to the regression methods above is a linear classifier that categorizes according to sign function. So for values above a certain threshold it maps the output to a category and for values below it maps it to another. But the function itself takes in continuous data, eq. 12.

$$\mathbf{x}_i^\top\mathbf{w} + b_0 \equiv \mathbf{x}_i^\top\mathbf{w} \tag{12}$$

But instead of a sign function we could, given $\mathbf{x}_i$, calculate the probability of the outcome being in a certain category. To calculate this probability we use the sigmoid function, eq. 13, and the outsome is given as $y_i = \{0, 1\}$.

$$f(\mathbf{x}_i^\top\mathbf{w}) = \frac{1}{1 + e^{-\mathbf{x}_i^\top\mathbf{w}}} \tag{13}$$

$$P(y_i = 1|\mathbf{x}_i, \theta) = \frac{1}{1 + e^{-\mathbf{x}_i^\top\mathbf{w}}}$$

Since we are only dealing with a binary problem, the phase can either be ordered ($y_i = 1$), or disordered ($y_i = 0$) and we have that

$$P(y_i = 0|\mathbf{x}_i, \theta) = 1 - P(y_i = 1|\mathbf{x}_i, \theta).$$

To find the cost function of the logistic regression we we use the Maximum Likelihood Estimation (MLE). We want to maximize the probability of a configuration being in a category. This gives us the cross entropy 14.

$$C(\beta) = \sum_{i=1}^{N} -y_i \log(f(X_i^\top \beta) - (1 - y_i) \log[1 - f(X_i^\top \beta)]) \tag{14}$$

3.4 Neural Network

$$\Delta_j^l = \frac{\partial E}{\partial z_j^l} = \frac{\partial E}{\partial a_j^l} \sigma'(z_j^l) \tag{15}$$

$$\Delta_j^l = \frac{\partial E}{\partial z_j^l} = \frac{\partial E}{\partial b_j^l} \frac{\partial b_j^l}{\partial z_j^l} = \frac{\partial E}{\partial b_j^l} \tag{16}$$

$$\Delta_j^l = \frac{\partial E}{\partial z_j^l} = \sum_k \frac{\partial E}{\partial z_k^{l+1}} \frac{\partial z_j^{l+1}}{\partial z_j^l} \tag{17}$$

$$= \sum_k \Delta_k^{l+1} \frac{\partial z_k^{l+1}}{z_k^l} \tag{18}$$

$$= \left(\sum_k \Delta_k^{l+1} \beta_{kj}^{l+1} \right) \sigma'(z_j^l) \tag{19}$$

$$\frac{\partial E}{\partial \beta_{jk}^l} = \frac{\partial E}{\partial z_j^l} \frac{\partial z_j^l}{\partial \beta_{jk}^l} = \Delta_j^l a_k^{l+1} \tag{20}$$

3.5 Stochastic gradient descent

In this project we are dealing with different types of cost functions. Depending on the method we use the minimization of the first derivative of the cost function is done differently. In the case of linear regression there is no use of numerical optimization methods as there exists an analytical solution to the derivative of the RSS, 7. In the case of logistic regression and the neural network however, there are no closed form solutions to the cost functions being used. We therefore take use of two algorithms called gradient descent and stochastic gradient descent in order to opdate the parameters (weights).

We call the function we wish to minimize $E(\theta)$, as energy is the quantity we in most problems within physics are trying to minimize. In linear regression this is the MSE (RSS) and in logistic regression it is the cross entropy.

$$E(\theta) = \sum_{i=1}^{N} e_i(X_i, \theta) \tag{21}$$

$$\theta_{t+1} = \theta_t - \eta_t \nabla_\theta E(\theta_t) \tag{22}$$

In regular gradient descent the algorithm simply updates the parameter θ_{t+1} according to eq. 22. Since the cost function tells us something about how well the parameters work for our regression model we seek to minimize its gradient. Updateing the parameters with 22 causes the model to moves closer to a local minima. The η is called the learning rate and it decides how fast we want to move in the gradients direction. A to large learning rate may cause the model to diverge, but given an optimal value or close to one the gradient descent will converge towards the minima.

Usually the first parameters are given random variables.

Gradient descent has some drawbacks, one being that it can get stuck in one local minima and never reaching the correct one. Another is that it is sensitvive to initial conditions, so what values we give the parameters in the begining matters. It is also somewhat computational expensive. A solution to this is stochastic gradient descent.

$$\nabla_\theta E(\theta) = \sum_i^n \nabla_\theta e_i(\mathbf{x}_i, \theta) \longrightarrow \sum_{i \in B_k} \nabla_\theta e_i(\mathbf{x}_i, \theta) \tag{23}$$

$$\nabla_\theta E^{MB}(\theta) = \sum_{i \in B_k}^M \nabla_\theta e_i(\mathbf{x}_i, \theta) \tag{24}$$

$$\theta_{t+1} = \theta_\mathbf{t} - \eta_t \nabla_\theta E^{MB}(\theta) \tag{25}$$

Here we devide the data set into smaller minibatches of size M creating n/M batches. We denote these minibatches B_k running from $k = 1...n/M$. Now we solve the gradient descent for the new minibatches and update the parameters according to 25. Not do we only speed up the computational process, but it also introduces stochasticity.

4 Resampling Methods

4.1 Bootstrap

The bootstrap is a resampling method suggested by Efron in 1979. It tell us how well our regression models assess the problem at hand. To say something about this it is commen to calculate the statistical properties given in the section above, particularly the MSE. As we can see from eq. 26 this is a measurment consisting of three properties. The variance, the bias and the irreducible error.

In machine learning phenomena such as overfitting and underfitting are highly important to be aware of and is connected with the MSE. Our goal is to predict the outcome $\hat{y}$ given some observed data. To do so we split our data into a training set and a test set and train the model with the training data. Depending on how well we fit our model to the training data we may overfit or underfit. Overfitting means that we fit the data so well that we have made the model to close and dependent on the training data. On the other hand we can underfit, meaning that we miss many important features of the data. In both cases trying the model out on the test set we get bad results. This is what is known as the

bias-variance tradeoff.
The variance tells us how the predicted outcome differs from its mean, and a high variance will correspond to overfitting. The bias says how much difference there is between the models predicted value and the true value. A high bias corresponds to underfitting.
Since our predictor is a random variable, how we draw the data will effect the estimate of the response. In bootstrap we draw samples with replacement from our dataset (the training data) and fit the model with this. Then we test the model with the test data and calculate the errors. Doing this many times we can examine the behavoir of the fit and get a more accurate estimate of the errors.

5 Statistical Properties

$$\begin{aligned} \text{MSE} &= \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \qquad (26)\\ &= \frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - \bar{\hat{y}}_i)^2 + \frac{1}{N}\sum_{i=1}^{N}(y_i - \bar{\hat{y}}_i)^2 + \frac{1}{N}\sum_{i=1}^{N}(y_i - \bar{\hat{y}}_i)(\bar{\hat{y}}_i - \hat{y}) \qquad (27)\\ &= Var(\hat{y}) + Bias + \epsilon \qquad (28) \end{aligned}$$

$$\text{R}^2(y, \hat{y}) = 1 - \frac{\sum_n^{i=1}(y_i - \hat{y}_i)^2}{\sum_n^{i=1}(y_i - \bar{y})^2} \qquad (29)$$

$$\text{Accuracy} = \frac{\sum_{i=1}^{n} I(t_i = y_i)}{n} \qquad (30)$$

6 Implementation

All code material can be found at `https://github.com/KariEriksen/Machine-Learning/tree/master/Project%202/Code`. It should also be mentioned that when working with the programs the notebook of Metha,[6] has been widely used.

6.1 Part b: 1D Ising with linear regression

The data produced for this part is generated through code given in the notebook mentioned above. The function ising_energies produces the energy-term in 4 given the sampled states. This is our data. In the file ising_1D.py we call upon the Bootstrap class which runs the linear regression model for the number of given bootstrap iterations. The Bootstrap class calls the class My_Linear_Regression and given the method choosen; OLS, ridge or lasso it will solve the regression problem for as many times as given. In ising_1D.py it is possible to select different parts, plotting the matrix containing the coupling constant J given the different methods, or plotting the R^2 or MSE.

6.2 Part c: 2D Ising with logistic regression

6.3 Part d: 1D Ising with neural network

6.4 Part e: 2D Ising with neural network

7 Results

7.1 Part b: 1D Ising with linear regression

We are using linear regression on the sampled spins-states and corresponding energies in order to train the network estimating the coupling constant. For ordinary least square the result can be viewed in fig. 1. The first we see is that the diagonal elements are close to zero, and the upper and lower elemtents are around 0.5. The same can be observed for the ridge regression and it does not seem to make a difference what λ we use. If we look at fig. 7.1 however we can see that for λ higher than 100 the score decreases towards zero. And both for OLS and ridge the score is fairly low for all values.

Moving on to lasso regression the situation is much better. Fig. 5 show the coupling constant which seems to be very close to -1. The R-score confirm our guess. It is close to 1 for $\lambda = \{0.001, 0.1\}$ but drops when we get higher than 0.1. A plot of the bias-variance tradeoff can be seen in fig. 7.1.

7.2 Part c: 2D Ising with logistic regression

7.3 Part d: 1D Ising with neural network

7.4 Part e: 2D Ising with neural network

7.5 Part f: Critical evaluation

8 Discussion

9 Conclusion

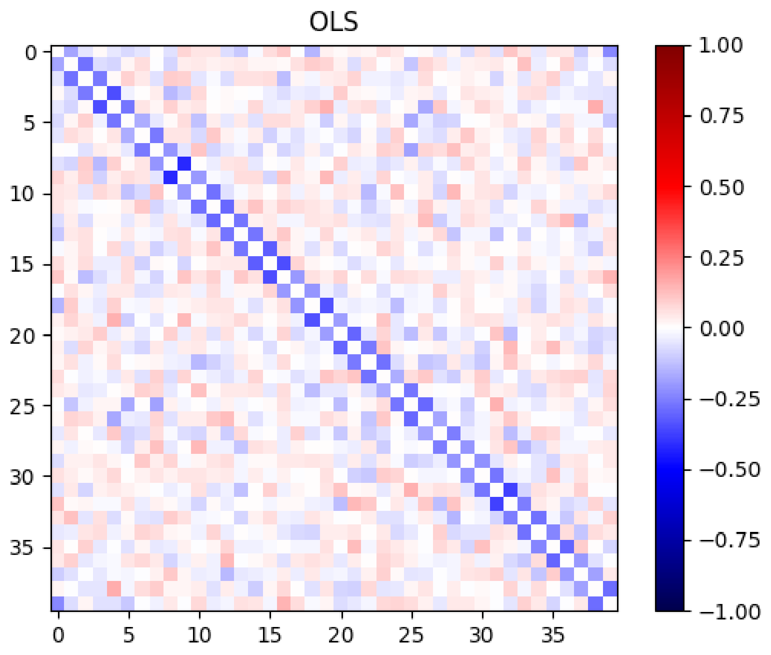

Figure 1: The figure shows the matrix J given by the ordinary least square method explained in 6.1. The model has been trained with a training set and tested with a separeted test set.

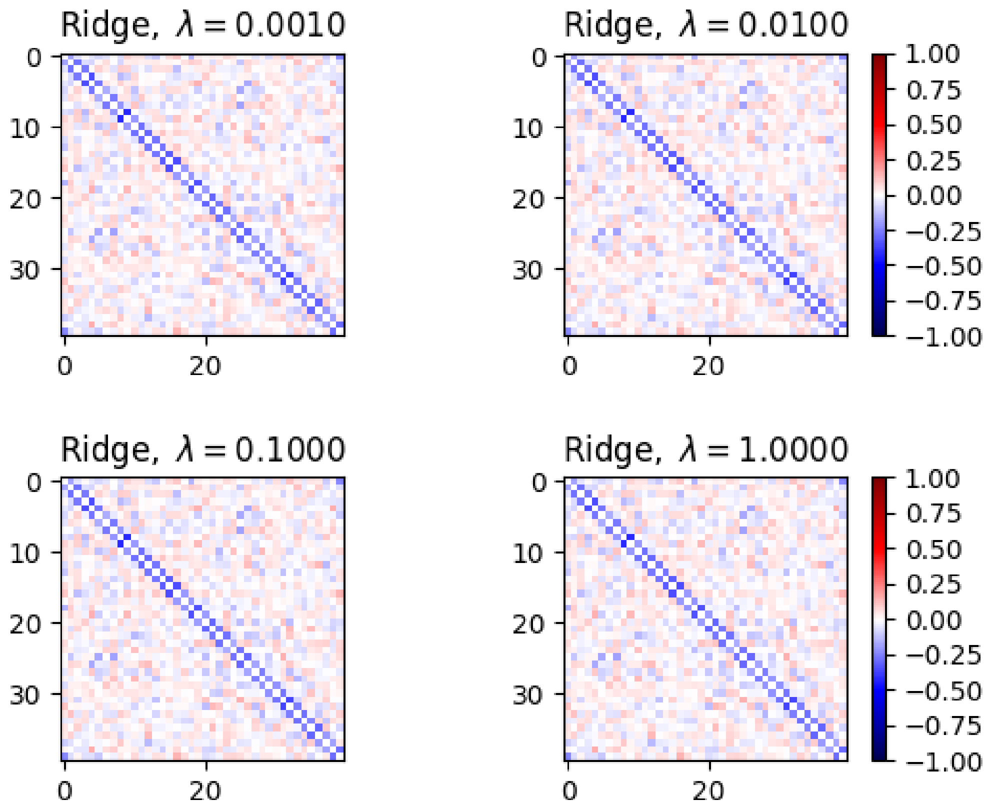

Figure 2: The matrix J with ridge regression method. We observe that changes in λ does not give a closer representation of the coupling constant.

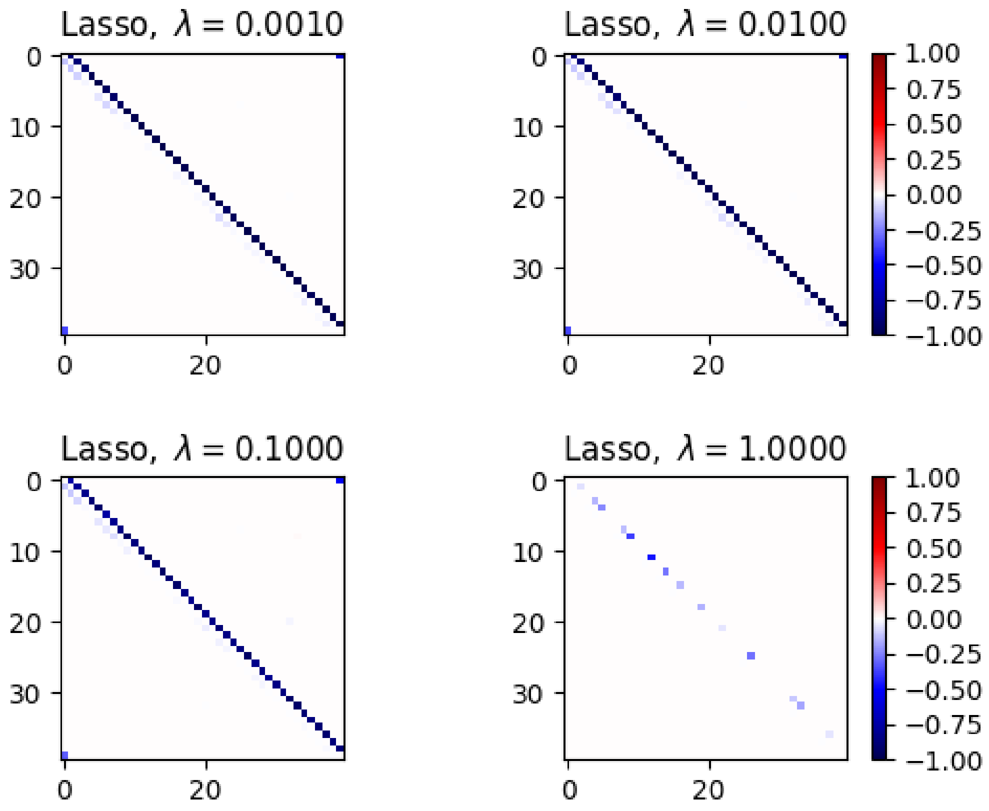

Figure 3: With lasso regression we get an enormous increase in the fit of the coupling constant. The method is able to fit the model very well for $\lambda = \{0.001, 0.1\}$.

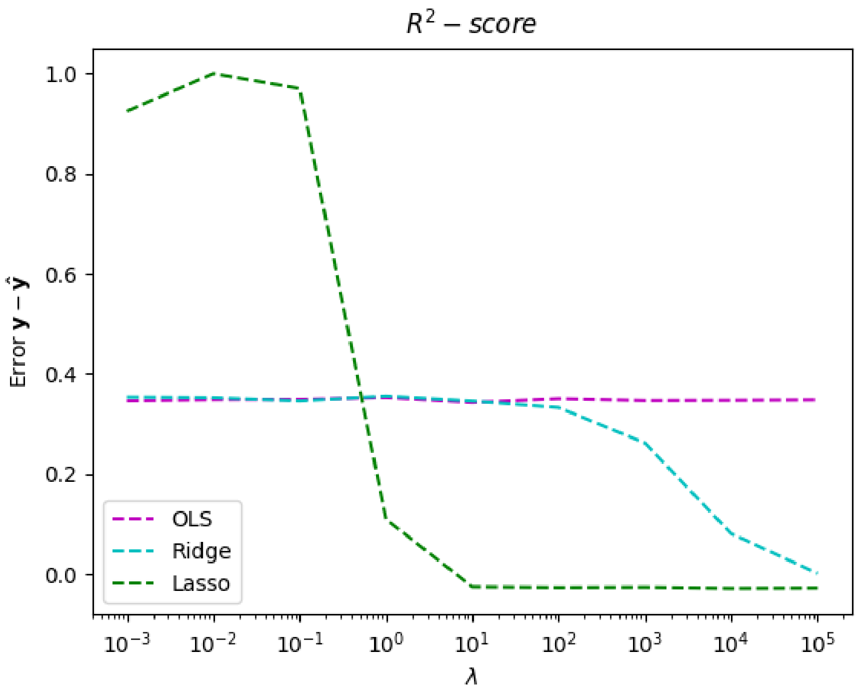

Figure 4: The R^2-score of the predicted output values, the energy, for all linear regression methods using bootstrap, with 100 iterations. The lasso is superior to the others with a score close to 1 for smaller λ.

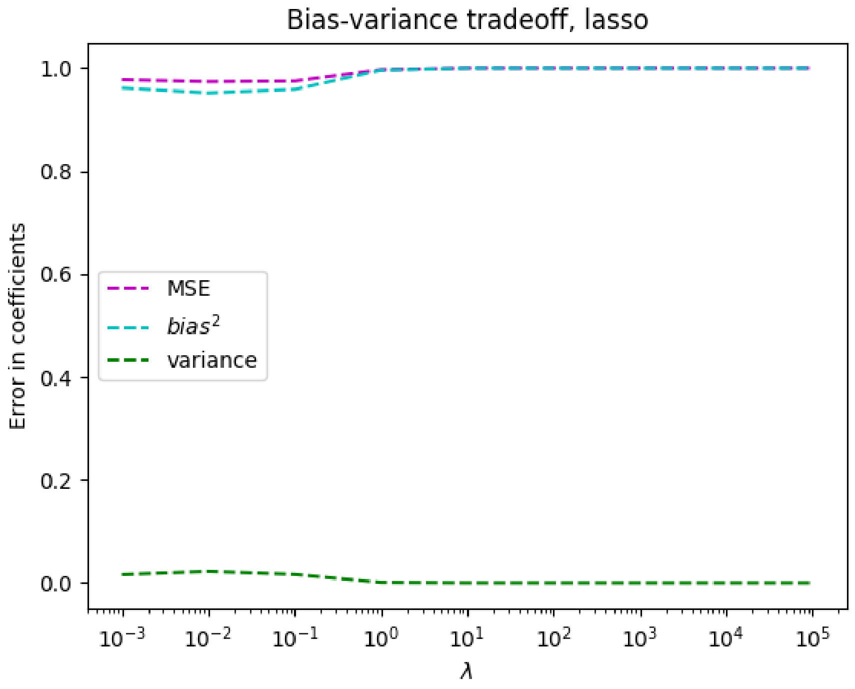

Figure 5: The bias-variance tradeoff for the coefficients with lasso regression using bootstrap resampling method with 100 iterations.

References

[1] Trevor Hastie, *The Elements of Statistical Learning*, Springer, New York, 2nd edition, 2009.

[2] Gareth James, *An Introduction to Statistical Learning*, Springer, New York, 2013.

[3] `https://compphysics.github.io/MachineLearning/doc/pub/Regression/html/._Regression-bs000.html`, 08/10-18.

[4] `https://ml.berkeley.edu/blog/2017/07/13/tutorial-4/`, 08/10-18.

[5] `https://arxiv.org/pdf/1803.08823.pdf,`, 08/10-18.

[6] `https://physics.bu.edu/~pankajm/MLnotebooks.html`, 08/10-18.

[7] `https://www.jstor.org/stable/pdf/2346178.pdf`, 08/10-18.

[8] `http://math.arizona.edu/~hzhang/math574m/Read/RidgeRegressionBiasedEstimationForNo`
`pdf`, 08/10-18.

[9] `http://statweb.stanford.edu/~tibs/sta305files/Rudyregularization.pdf`, 08/10-18.