Application of ML in investigating ferromagnetic transitions

Ashish Panigrahi and S. Gauthameshwar

National Institute of Science Education and Research

1D Ising Model

The problem:

We have the spin configuration of a 1D lattice. We also have the energy corresponding to that configuration.

Ex :

Lattice : [1 , 1, - 1 , 1 ,- 1 , 1 ,- 1 , 1 , 1 ,- 1 ,- 1 ,- 1, 1 ,- 1 , 1 , 1 , 1, - 1, - 1 ,- 1]

Energy label: 8 units

We are also given a scenario where one atom's spin is affected by another atom's spin (2-body interaction). The Energy of such an interacting system is given by:

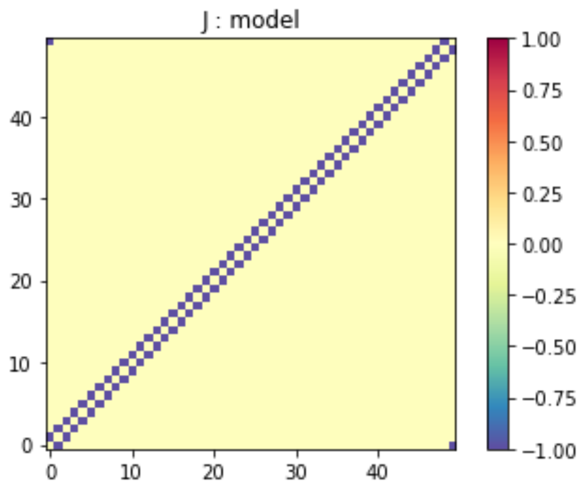
$H = \sum_{i,j} J_{\text{ij}}\, \sigma_{\text{ij}} = \sum_{i,j} J_{\text{ij}}\, S_i\, S_j$

S_i is the spin of ith atom and i and j sum over all the atoms in lattice.

Question : What is the extent of interaction of our atoms with each other?

If we have a lattice where all the atoms talk only to their first neighbours, we have :

$$J_{\text{ij}} = \begin{cases} -1 & \text{if } j = i-1 \text{ or } j = i+1 \\ 0 & \text{else} \end{cases}$$



`J matrix for nearest neighbour interaction. Lattice length : 50 atoms`

Data generation:

- The lattice data is generated using random function to assign the spin at each lattice point. Length of lattice $d = 50$ atoms. We assume our temperature is very high. So we don’t invoke any statistical physics here.
- The energy of all these lattice are calculated using the nearest neighbour J matrix assuming periodic boundaries. (the neighbours of 1st atom is 2nd atom and the last atom)

Training a model to find J matrix using the above data

Linear regression approach

Model parameters: $x_{\text{ij}} = S_i\, S_j$, $w_{\text{ij}} = J_{\text{ij}}$

Predict: $E_p{}^{(n)} = J_{\text{ij}}.x_{\text{ij}}{}^{(n)}$ such that:

$\theta = \frac{1}{N} \sum_{n=1}^{N} \left(E_p{}^{(n)} - E_{\text{label}}{}^{(n)}\right)^2$ is minimised.

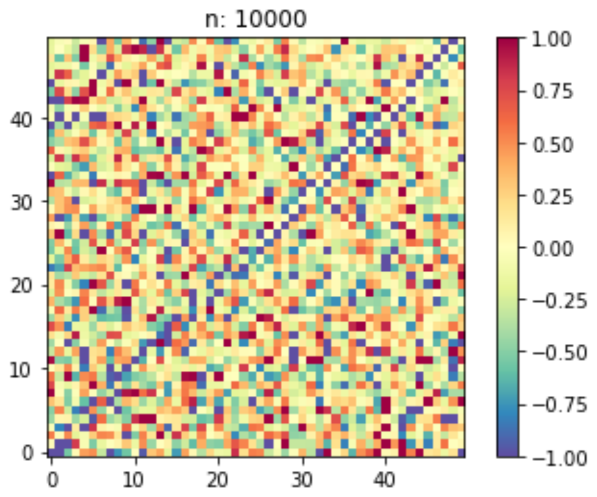
Results:

Run 1:

n: 10000. Train-Test: 7000 - 3000

Training score (R^2): 100.0 /100

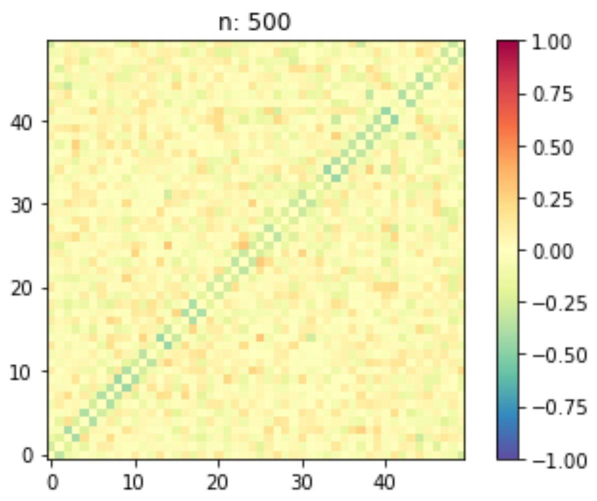
Testing score (R^2): 100.0 /100 (God!)

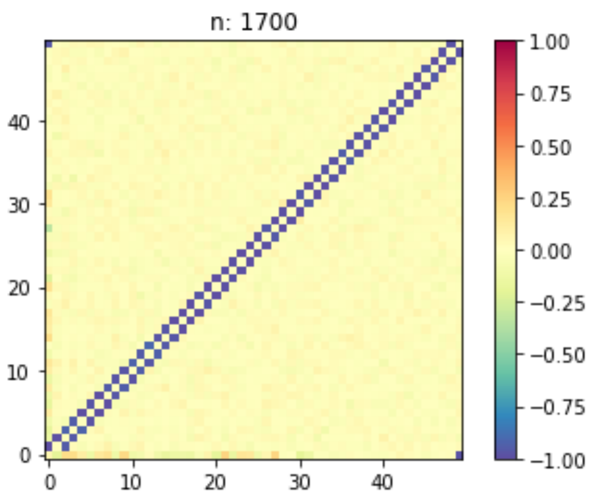


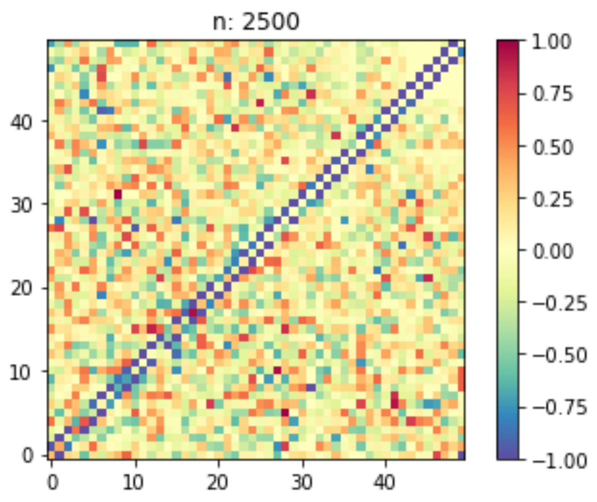
Hol up!

Taking a step back to understand what just happened..

Primary suspect: data size n is too large and our model was led to overfitting.
Running LSQ on three different data sizes:





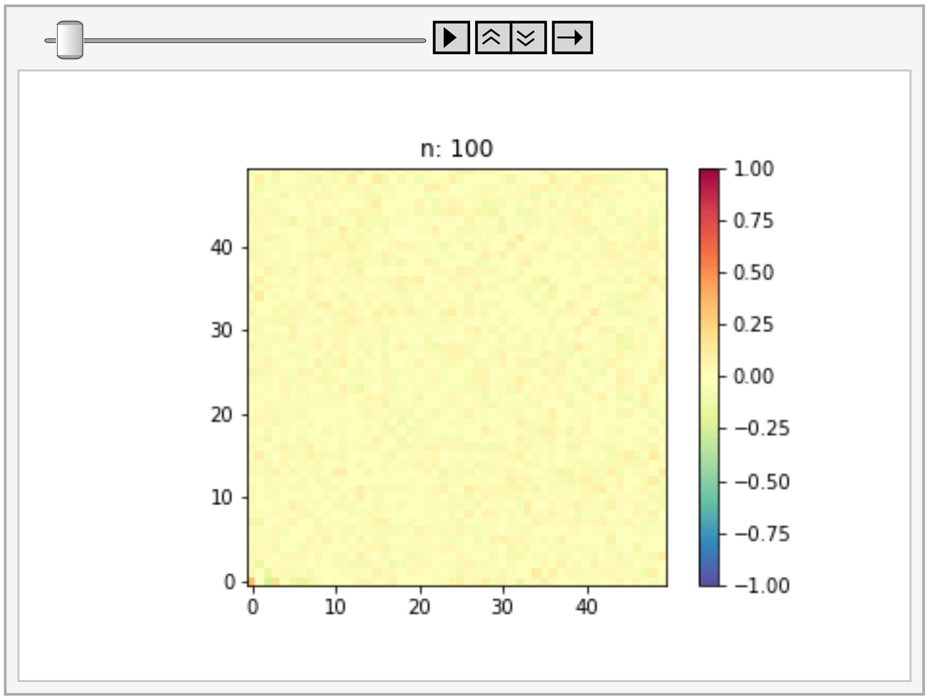


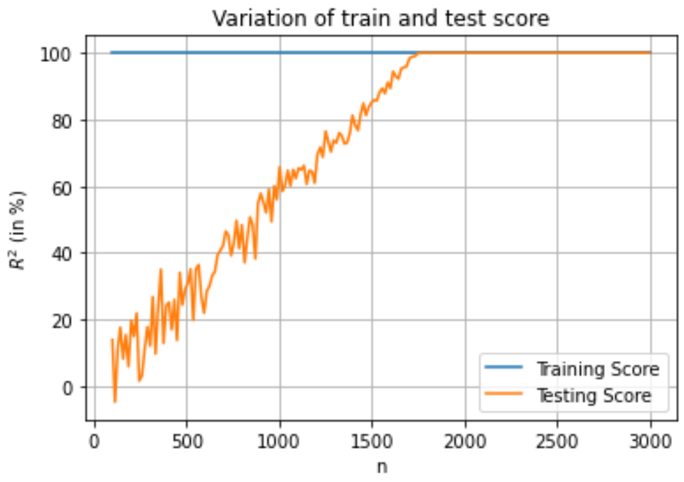
n : 500
train - test : 350 - 150
Training score : 100.0/100
/100
Testing score : 28.62/100
/100

n : 1700
train-test: 1050 - 450
Training score: 100.0/100
Testing score: 96.01/100

n: 2500
train-test: 3500 - 1500
Training score: 100.0
Testing score: 100.0

Capturing the effect of n on our LSQ model:





After a certain number of dataset, there is no distinction between our train and test data since they all are randomly generated!
Consider these two cases:

- Randomly generate n data and split it into train and test 70-30
- Randomly create $\frac{7n}{10}$ data in train and $\frac{3n}{10}$ data in test. The statistical properties of train and test in both these cases are similar for large enough n. This explains the reason why both our train and test give 100% score despite the model being nowhere close to the actual J.

Implementing regularisation to avoid an overfitting model

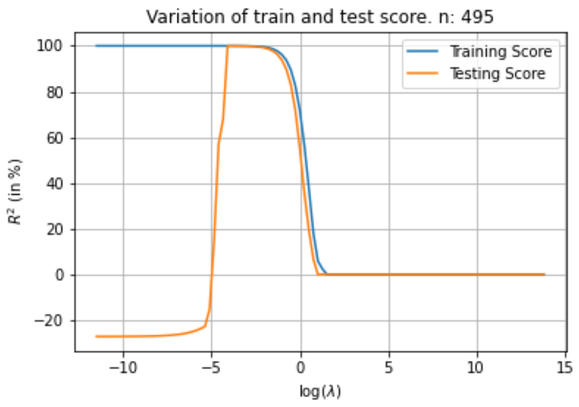
Lasso Regression

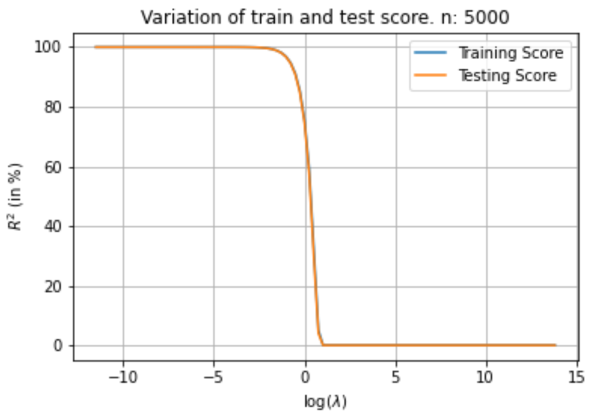
Model parameters: $x_{\mathrm{ij}} = S_i\, S_j$, $w_{\mathrm{ij}} = J_{\mathrm{ij}}$

Predict: $E_p{}^{(n)} = J_{\mathrm{ij}}.x_{\mathrm{ij}}{}^{(n)}$ such that:

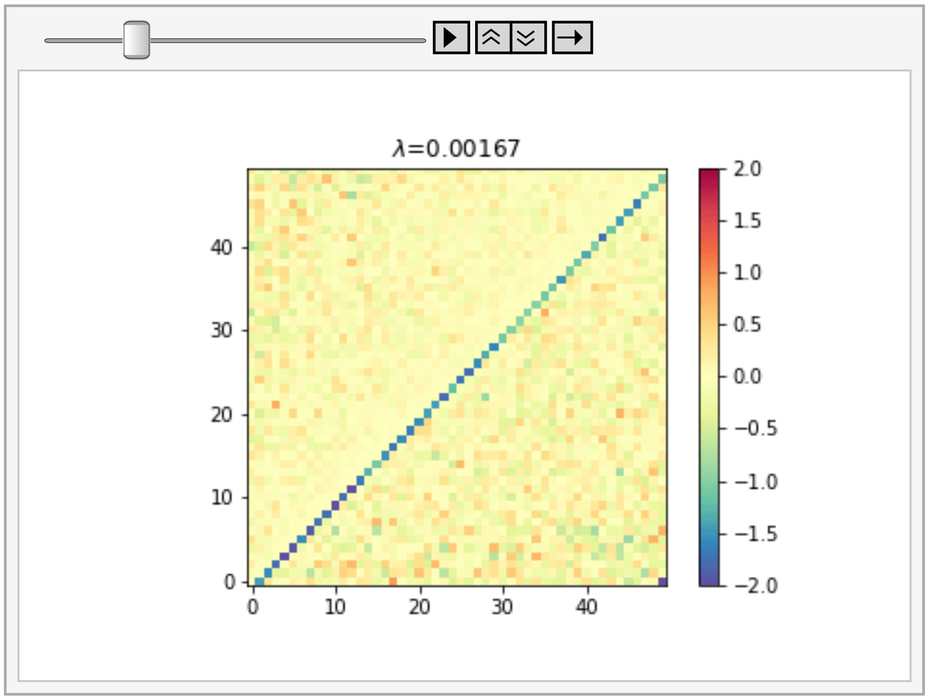
$\theta = \lambda \,||\, J \,||\, + \frac{1}{N} \sum_{n=1}^{N} \left(E_p{}^{(n)} - E_{\mathrm{label}}{}^{(n)}\right)^2$ is minimised.

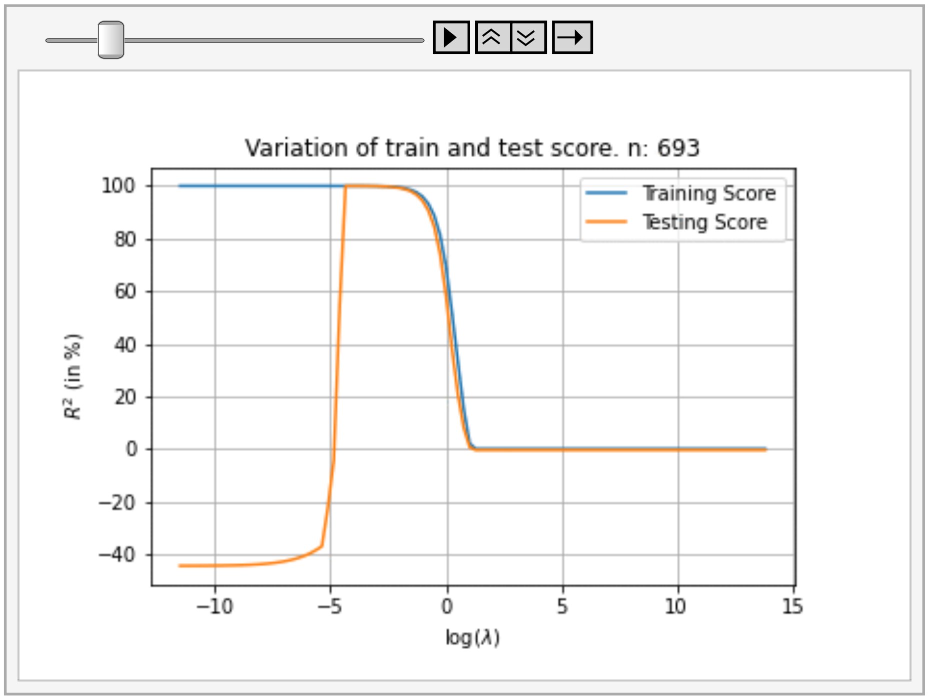
A regularisation parameter λ is introduced which, on properly fixing gives a model that has a balanced fitting.



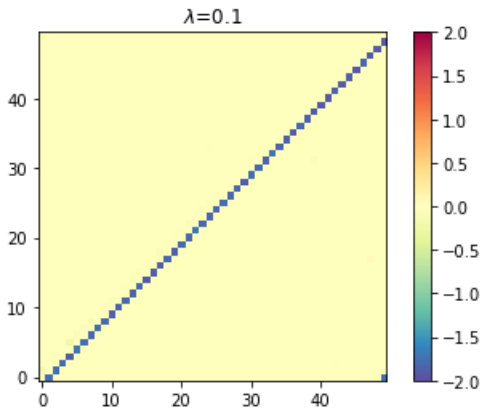


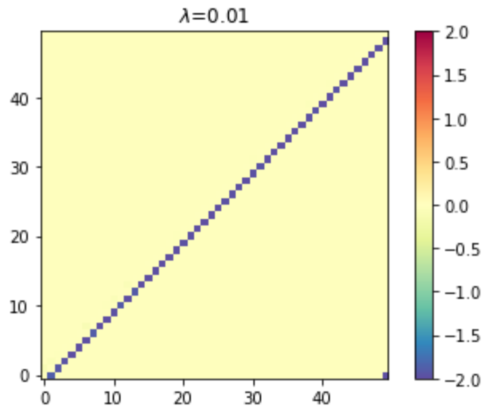
- For smaller datasets, Lasso has sweet spots that give score as good as the training set.
- For larger datasets, the train and test sets give the same statistical results as expected.

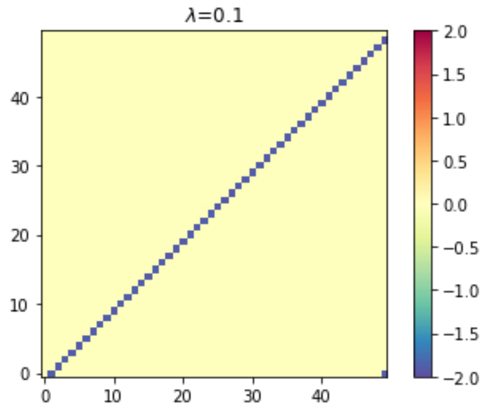




Verifying we get the same initial model at these sweet spots without any dependence on the data size:







λ : 0.1	λ: 0.01	λ: 0.1
train - test : 350 - 150	train-test: 1050 - 450	train-test: 3500 - 1500
Training score : 99.71/100	Training score: 99.99 /100	Training score: 99.76/100
Testing score : 99.53/100	Testing score: 99.99/100	Testing score: 99.75 /100

Ridge Regression

Model parameters: $x_{\text{ij}} = S_i\, S_j$, $w_{\text{ij}} = J_{\text{ij}}$

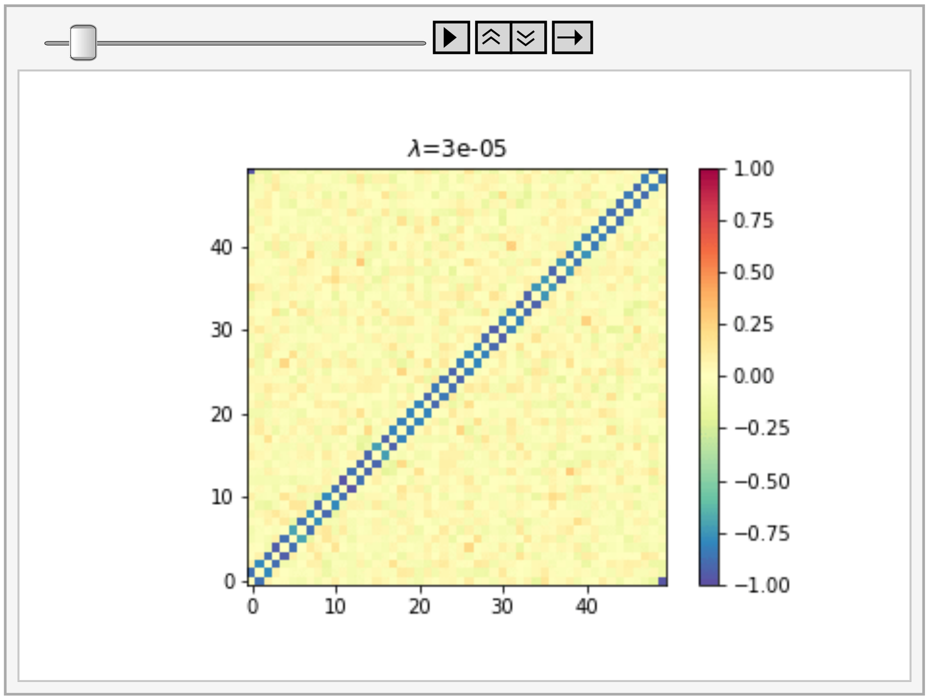
Predict: $E_p{}^{(n)} = J_{\text{ij}}.x_{\text{ij}}{}^{(n)}$ such that:

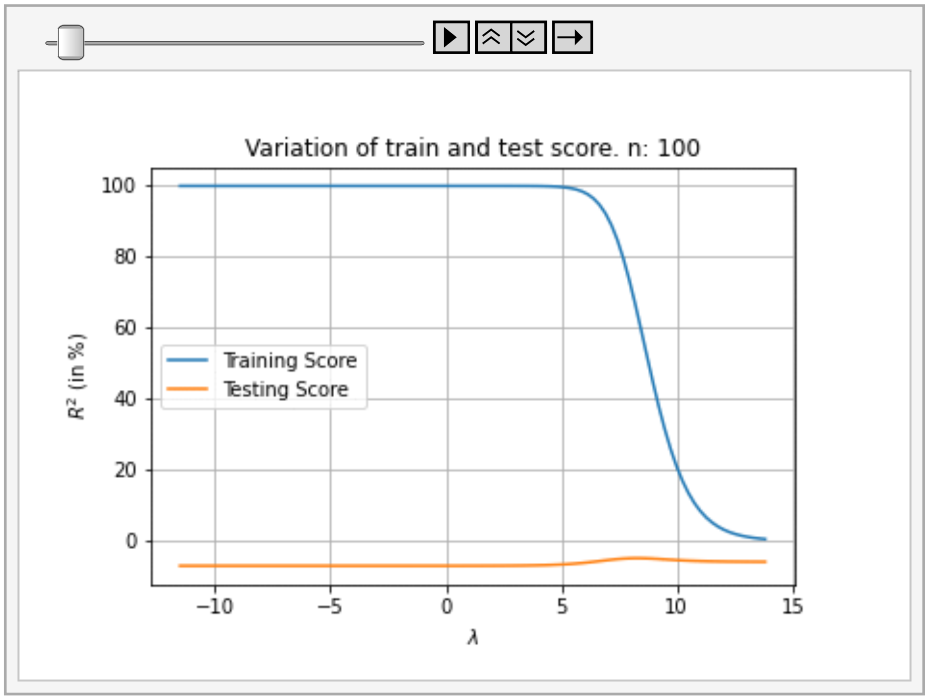
$\theta = \lambda \,||\, J \,||^2 + \frac{1}{N}\sum_{n=1}^{N}\left(E_p{}^{(n)} - E_{\text{label}}{}^{(n)}\right)^2$ is minimised.

A regularisation parameter λ is introduced which, on properly fixing gives a model that has a balanced fitting.

- Unlike Lasso, Ridge is just as bad as LSQ for small datasets. There are no sweet spots like we had in Lasso and the model simply mimics LSQ for small λ.

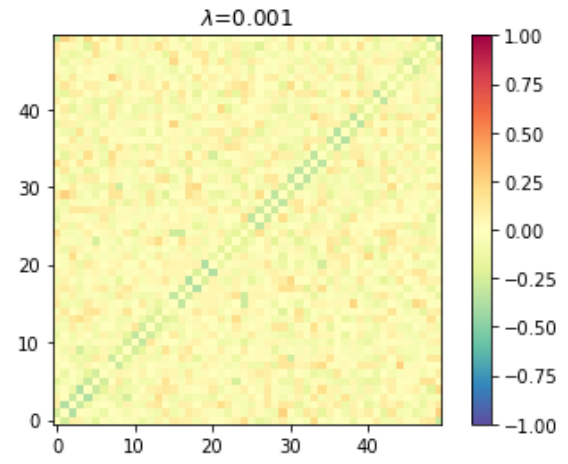
- For larger datasets, the train and test sets give the same statistical results as expected and it predicts the model accurately without the issue of overfitting!

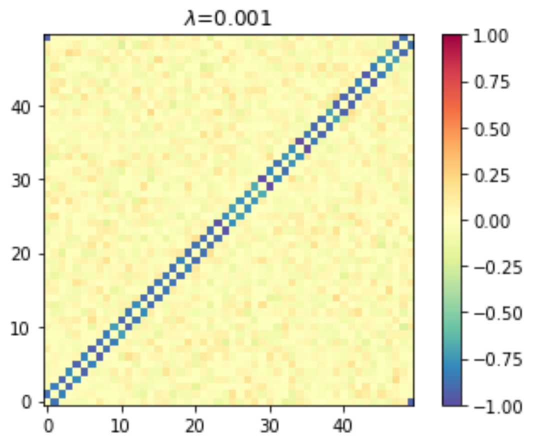


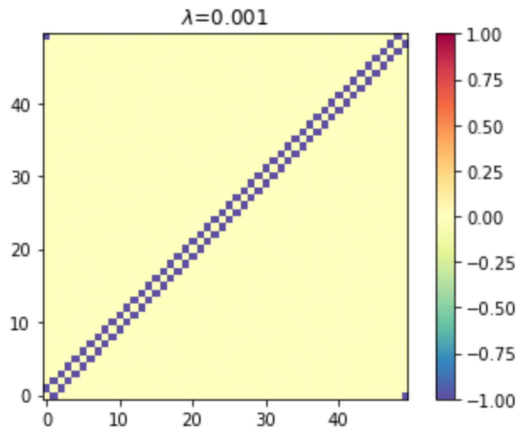


In[∘]:=
```
Import[
 "G:\\Sem 7 (labs+online)\\ML\\Ising model\\Ridge reg_1D\\score variation ridge.gif",
 "Animation"]
```

J matrix for different data size:







λ : 0.001
train - test : 350 - 150
Training score : 99.99/100
Testing score : 17.14/100

λ: 0.001
train-test: 1050 - 450
Training score: 99.99 /100
Testing score: 85.24/100

λ: 0.001
train-test: 3500 - 1500
Training score: 99.99/100
Testing score: 99.99/100

Side note on Lasso vs Ridge

Why do we get different results for Ridge and Lasso despite them being very similar?

- Both Ridge and Lasso don't like large values of ||J|| as it makes our data more sensitive to errors. They try to prefer J with smaller "slope" than that with large one and hence, prefer low variance over low bias. Thus, they regularise the overfitting issue efficiently.
- But the optimisation of Lasso and ridge w.r.t LSQ is different. It can be mathematically shown that :

$J_{\text{ridge}} = \frac{J_{\text{lsq}}}{1+\lambda}$

Thus, ridge always scales the J by the regularisation parameter. Larger the λ, more flat our "slope" becomes. But even for large λ, J only approaches towards 0 asymptotically (close but never equal to 0).

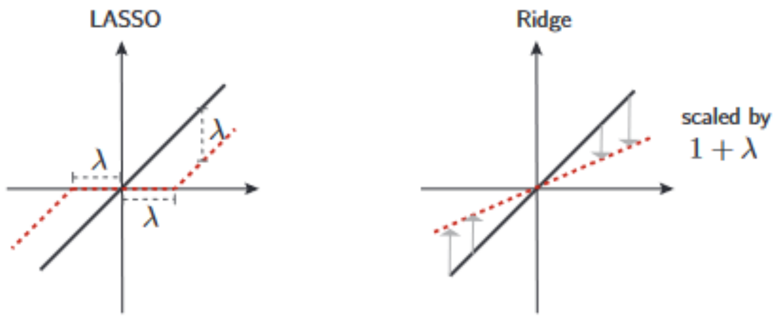


Fig. adapted from P. Mehta *et al* (2019)

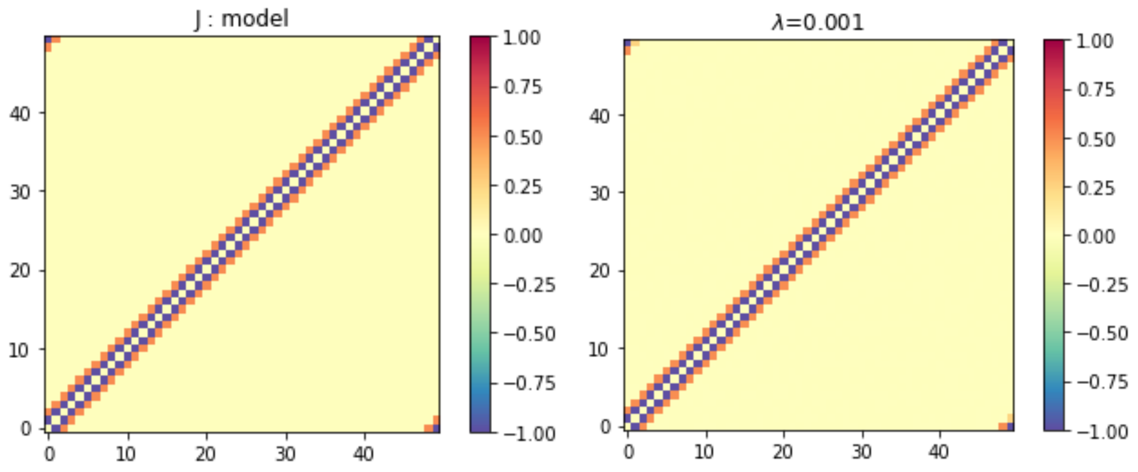
- Lasso, on the other hand doesn't care about asymptotically reducing J. As we increase λ, the gap b/w the J_{lsq} and J_{lasso} where $J = 0$ widens. This results in more and more unimportant parts of our data being set to 0 (which we don't see in ridge). Thus, for some specific λ, we have exactly the unnecessary data set to 0, and the prominent parts of our "slope" J intact. This is the reason we see sweet spots that perfectly regularise by making the unnecessary overfitting to 0 and only retain the original model. This is phenomenon is also referred to as 'sparse' regression.

Beyond Nearest neighbours:

We took instances where our J is an interacting matrix only with the neighbours. What if we have a next-nearest neighbour interaction also? Does our ML model predict their nature as well?

Ridge Regression for next nearest interaction 1D lattice:

- Generated Model: $J_{\text{ij}} = \begin{cases} -1 & \text{if } j = i-1 \text{ or } j = i+1 \\ 0.5 & \text{if } j = i-2 \text{ or } j = i+2 \\ 0 & \text{else} \end{cases}$



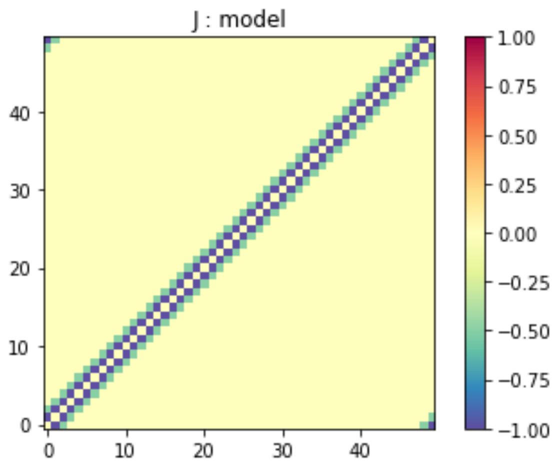
n: 5000

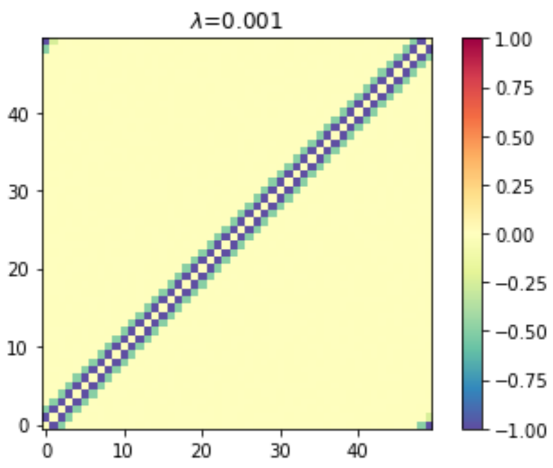
train-test: 3500 - 1500

Training score: 99.99/100

Testing score: 99.99 /100

- Generated Model: $J_{\text{ij}} = \begin{cases} -1 & \text{if } j = i-1 \text{ or } j = i+1 \\ -0.5 & \text{if } j = i-2 \text{ or } j = i+2 \\ 0 & \text{else} \end{cases}$





n: 5000

train-test: 3500 - 1500

Training score: 99.99/100

Testing score: 99.99 /100

Conclusion

- Very bad idea to use Linear Regression to train 1D lattice!
- If the data set is small, Lasso predicts the model accurately with regularisation $\lambda \sim 10^{-2} - 10^{-4}$
- If the data set is large, both Lasso and Ridge predicts the model accurately with regularisation $\lambda \sim 10^{-2} - 10^{-4}$
- Ridge works like magic for predicting interactions in our lattice beyond neighbouring atoms as well!

2D Ising Model

Introduction to the problem:

An interesting phenomenon gets manifested if we move from a 1D to 2D Ising lattice. We observe something called “Phase transition”. To put in simple words, phase transition is a phenomenon where if we cross a certain temperature T_c, our 2D lattice behaves very different in spin alignment of our lattice. Below T_c, our spin lattice has a tendency to form lumps of atoms that have the same spin alignment (also called Domains). This is leads to a physically observable magnetic property called Ferromagnetism. Our lattice retains some finite spin even without an external magnetic field **B.** But once our temperature crosses T_c, the spins get more dominated by the thermal agitation over the exchange energy stability. Here, our domains disappear and our lattice acquires net zero spin in the absence of **B**. Our Ferromagnet becomes a paramagnet beyond the critical phase transition temperature T_c. In this 2D Ising model, we wish to determine the critical phase temperature T_c using Machine Learning models.

Goal till Midway: Generate the 2D Ising model data using Monte Carlo methods

- Our goal is to generate 10,000 sample 2D lattices for each different temperatures T.
- We take 10 temperature instances from 1 to 10 SI.

The data generation for 2D ising model is not as simple as what we did for 1D ising model. We need to find lattice configurations which are extremely stable at the given temperature (using random spin assignments wont do the job!).

The Metropolis Algorithm

The goal is to find an equilibrium state for the two-dimensional spin lattice system at a particular temperature dictated by β. It is achieved as follows:

1. Begin with a random state μ for the lattice configuration.
2. Pick any random lattice site and flip the sign of the spin. We call this new state ν. What is the probability P($\mu \to \nu$) that we'll accept this new state?
3. If $E_\nu > E_\mu$, then we set $P(\nu \to \mu) = 1$ and thus by using the detailed balance equation we get $P(\mu \to \nu) = e^{-\beta(E_\nu - E_\mu)}$
4. If $E_\mu > E_\nu$, then we set $P(\mu \to \nu) = 1$ (i.e) with full probability, the spins gets flipped to a more stable configuration
5. Iterate step (1)-(4) as many times as required. Eventually we end up with an equilibrium state for large enough iterations

Some points to note in our algo:

- The convergence for a solution is guaranteed for large enough iterations. The relative error for the above Monte-Carlo goes as $\frac{\Delta E}{E} \sim \frac{1}{\sqrt{N}}$ for large iterations N.

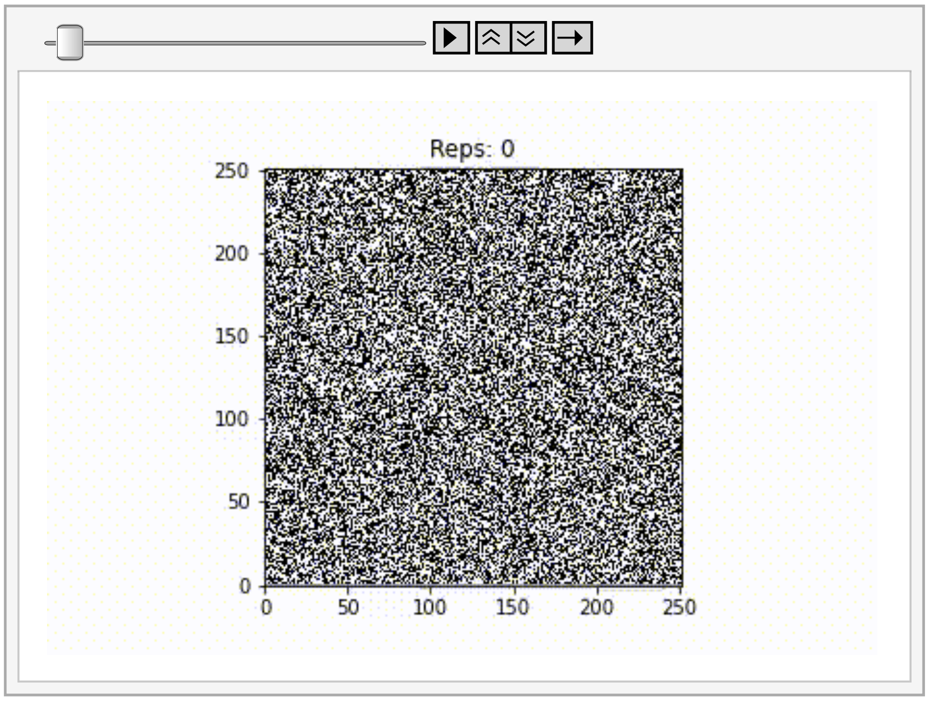
Seeing Metropolis algo in action

At first, the algo took 2 hrs to complete tabulating 10,000 Monte Carlo iterations for a 50x50 lattice (Extremely slow!). This was optimised by observing the fact that energy changes only upto the neighbours due to flipping one spin. Hence, only incremental energy was calculated for updating the spin (instead of calculating the entire lattice energy every time we encounter step (3) or (4)). Runtime: ~ 46 sec to complete 1,000,000 monte iterations for 200x200 lattice!

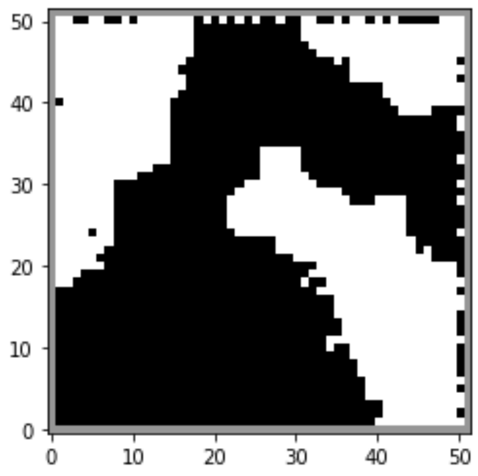
We optimised our code using Numba which gave us even better runtime ~ 0.26 secs

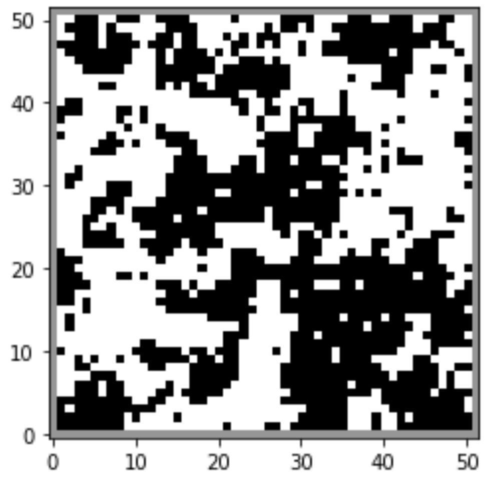
250x250 Spin Lattice at $\beta = 0.75$:

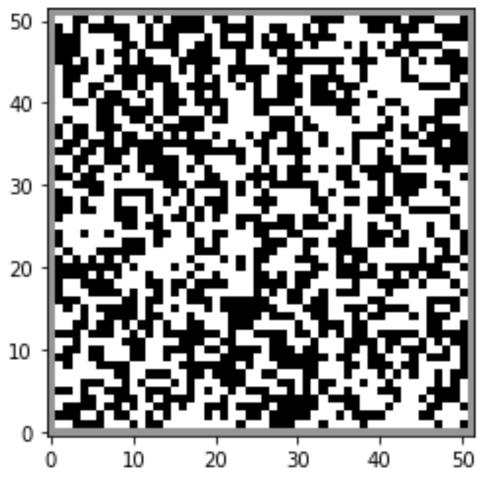
Out[ ]=



Instances of Ordered, Critical and Disordered phases after sufficient iterations:

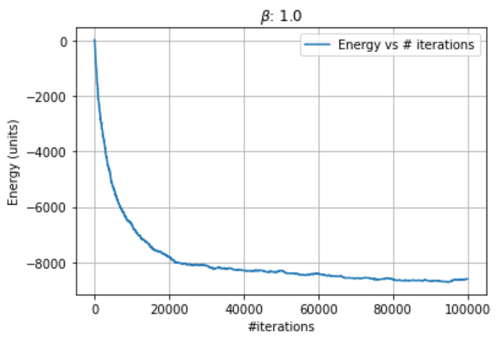


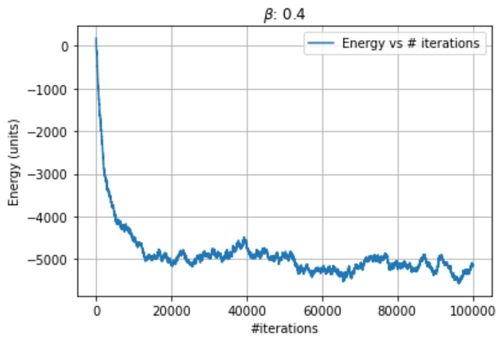


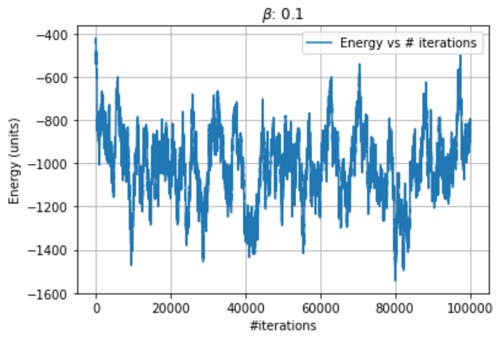


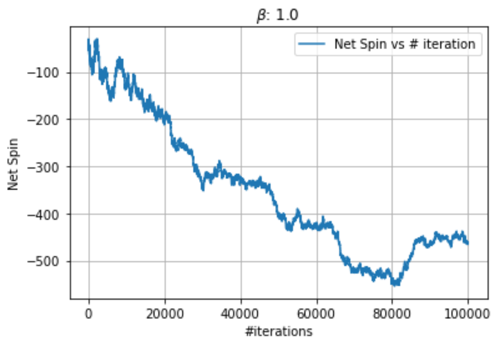
$\beta = 1$ ($T = 1$ SI) $\beta = 0.4$ ($T = 2.5$ SI) $\beta = 0.1$ ($T = 10$ SI)

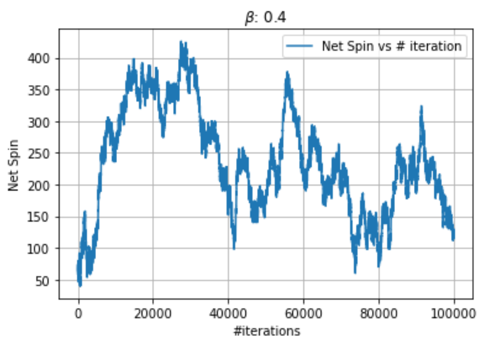
Reliability of our Data generation method

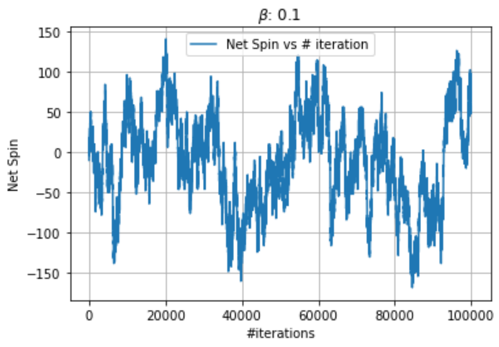












For ordered phases, the energy converges to a value up to a good accuracy. The energy, however gets random noises as we move from critical to disordered phase. This is solely due to the fact that our system gets chaotic. All these energy states almost have equal likelihood due to high T, and this does not correspond to the fact that our algo misbehaves at those cases. Spins usually do not converge since there are theoretically many spin configurations that might correspond to the same energy state.

So our generated data is quite reliable!

References:

1. Pankaj Mehta et al. “A high-bias, low-variance introduction to machine learning for physicists”. In: Physics reports 810 (2019), pp. 1-124
2. Beichl, Isabel, and Francis Sullivan. “The metropolis algorithm.” Computing in Science & Engineering 2.1 (2000): 65-69
3. Hastings, W.K. (1970). “Monte Carlo Sampling Methods Using Markov Chains and Their Applications”. 57 (1): 97–109
4. Deisenroth, Marc Peter, A. Aldo Faisal, and Cheng Soon Ong. Mathematics for machine learning. Cambridge University Press, 2020