**Theoretical background: Linear regression, logistic regression and Neural Network**

**Linear regression**

**We start by defining our training set as . The linear regression model is defined as**

**where is the noise term and is the parameter we want to estimate in order to fit our model to the data. By expanding our feature-space, we will instead define our linear model (1) as**

**where is the noise term, is a -dimensional vector and is a -dimensional matrix, which gives our design matrix**

**In order to estimate the parameters, we need to define a cost function for the model. In the case of linear regression, we** can write the cost function, including regularization, as follows (Hastie et al, 2001)

which gives the Ridge estimation of the **parameters**

leading to the estimated solution

In the case of singularity (of the matrix ) as is the case for the Ising model, Ridge estimation works as it applies a small value to the diagonal elements of matrix , making it non-singular. Another approach is to use singular value decomposition, both for OLS () and Ridge regression. The OLS estimator is rewritten in terms of SVD as follows (van Wieringen, 2015)

and for the Ridge estimator in terms of SVD (van Wieringen ,2015)

**An algorithm for computing the Ridge estimation is provided in Appendix A. Note that for computing OLS we can use the results from but setting** , which gives the estimation given in (5).

**Classification and logistic regression**

The classification problem resembles the regression problem, except that the target values we want to predict in classification are discrete variables (categories), which could take a binary or multiclass form. In binary classification, which we have studied in this report, the target value takes only two values, typically and , or and . The ultimate goal in classification, given a training set, is to predict the target value by some input , typically via some non-linear function , also known as the logistic function. The logistic function takes the form

where is defined as

where are the parameter we want to estimate. Note that the model in equation 8 is similar to the model defined in linear regression, only that this model is passed through a non-linear function (7), constraining the values between and. The function as defined above, is commonly known as the sigmoid function. As with linear regression, in order to estimate the parameters in **we need to define a cost function for the model.** Because of minimization of the cost function in logistic regression leads to non-linear equation for the estimators (Jensen, 2018); the estimation problem needs to be solved by some iterative method for minimizing the cost function. For a given training example the cost function for logistic regression including regularization is defined as (Ng, 2018)

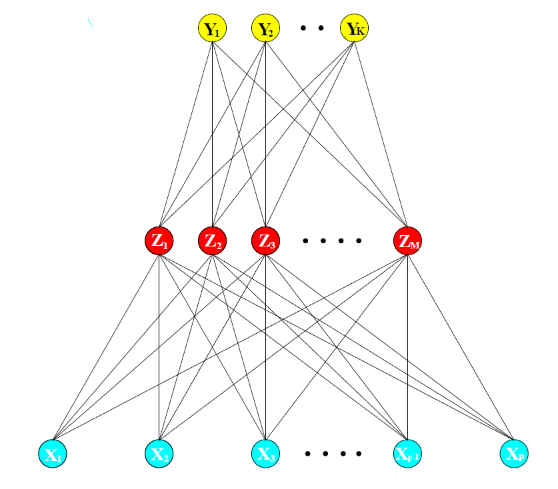
where the second term is the regularization. We can rewrite it in vectorized form by including all training examples

We take the partial derivatives wrt the estimators

Finally, we update the estimators by gradient methods, such as gradient descent. The gradient descent optimization, including regularization, is defined as

**where**  parameter controls the step-length of each gradient descent iteration, which is also known as the learning rate. **The main idea behind gradient descent, is to adjust the estimators in the direction where the gradient of the cost function is steepest, which ensures a stepwise migration towards a local (or global) minimum. An algorithm for computing the logistic cost function and gradient optimization, including**  penalty **is provided in Appendix A. Note that for computing OLS we can set** .

**Neural network**

The building blocks of a neural network is build up by an input layer, hidden layer(s) and an output layer. All layers consists of neurons, which are connected from the former layer via matrix–vector multiplication and non-linear activation functions. The forward computation from one layer to the next is known as forward propagation. In order to explain the forward propagation, we start with defining our training data as, where a single input vector as presented in Figure XXXXXXX. Feeding the network with training examples, gives the input matrix as

The input vectors are fed forward to all neurons in the next layer (hidden layer in Figure XXXXXXX). The non-linear transform from the input layer to next, is defined by the equation

where the function is the non-linear activation function we compute before feeding forward to the next hidden layer, or output layer. The activation is commonly denoted for any layer as. The matrix are the weights and are the biases in layer (with neurons), and defines the parameters which we want to learn (estimate). There exist many kinds of activation functions. Ng and Katanforoosh (2018) lists a few examples of the most familiar activation functions in the Deep learning course notes

The forward propagation algorithm is described in more detail as

1. Set. Note that in these examples, inputs are column vectors.
2. Compute the first argument
3. Compute the activation function for the first hidden layer
4. Compute the argument in the next layers
5. Compute the activation function

An algorithm for computing the forward propagation with two hidden layers is presented in Appendix A. In order to train the weights and biases, we first need to define a cost function for the output and define its partial derivatives wrt the weights and biases. In our example, we will use the cost function as described in the theory part for logistic regression. The cost function for the output layer, including regularization, is defined as

The weights in each layer are updated using the same gradient method as explained for logistic regression, i.e. gradient descent. Updating the weights and biases for any layer is done by computing the gradient descent optimization

and

We take the partial derivative wrt the weights in the output layer

The error for the output layer is defined as . The error in any layer and partial derivative of the cost function generalized as

and

where is the Hadamard product, which imply an element-wise multiplication between the arguments. This result comes from the partial derivatives wrt the weights in any of the former layers. By introducing the chain-rule to the results derived in the forward propagation algorithm, we see that the partial derivative wrt gives (excluding regularization)

The scheme for estimating the errors in all layers and neurons, and respective gradients for weights and biases, is known as backpropagation. The task of the backpropagation algorithm is to determine the change to the weights and biases in terms of what relative proportions causes the most decrease to the cost. By following the course notes of Ng and Katanforoosh (2018) we can use the derivations above and the results from the forward propagation algorithm, to do the backward propagation algorithm. This is described in more detail as

1. Perform forward propagation (as explained), for all layers.
2. Compute the error in the output layer .
3. For layer

compute the error:

1. Compute the partial derivatives wrt weights and biases using the following results

Weights:

Biases:

An algorithm for computing the backward propagation with two hidden layers is presented in Appendix A. **In order to update the weights and biases we need to implement an optimization scheme, such as the gradient descent algorithm, which ensure a stepwise migration towards a local minimum.** There are basically three approaches of gradient descent method; batch gradient descent, which takes the entire dataset and computes the gradient of the cost function; mini-batch gradient descent, which takes mini-batches of size ; stochastic gradient descent, which is a special case of mini-batch gradient descent, with . Implementation of one mini-batch gradient descent is described in more detail as

1. Set and.
2. For batch size
3. Perform backward propagation to compute

Weights:

Biases:

1. Store gradients

Weights:

Biases:

1. Update the weights

Weights:

Biases:

Repeat the steps above in order to reduce the cost function. An algorithm for gradient descent is presented in Appendix A.

In order to evaluate the model performance, i.e. how well it predict the training data, we turn towards metrics such as MSE and the score. The MSE is defined as

and the score is defined as

where  **is the true model, is the predicted model and is the mean of the true model. However, for model evaluation in machine learning problems, the prediction error (MSE) measured on training data is not a good measure for model assessment (Hastie et al, 2001). There exists a couple of statistical tools, which provide model assessment in machine learning problems, known as resampling methods. The bootstrap method and the K-fold cross validation are** the most common for model assessment in machine learning problems (James et al, 2013).

**Suppose now that we split the jointly distributed set of measurements into a training set and an independent test set. The basic idea in bootstrapping is to randomly draw samples from your training set with replacement (i.e. the same sample can be drawn more than once), let’s say times, and refit the model for each bootstrap sample, and then apply the same set of predictors to test set. For bootstrap, the expected prediction error function reads (Hastie et al, 2001)**

where , i.e. the squared error between the model and the predicted model . **Assessing the model performance is done by computing the average of the expected prediction error on both the training set and the test set at each point. The purpose of splitting the data into a training set and an independent test set, is that the prediction error for the training set tends to decrease using higher model complexity, but for the test set, the prediction error tends to have a turning point where the prediction error increases. As prediction increases for the test set using higher model complexity, the model is overfitting, i.e. the training model is not able to generalize the predictions. This is an inevitable consequence, and is related to the variance-bias tradeoff of the prediction, which we can derive from expected prediction error**

where we have used the following relationships

The first term is the irreducible error, which contain the variance of the input data. Algorithms for computing the MSE and score is provided in Appendix A. The bootstrap method was implemented using functions provided by Scikit learn, to split the model and evaluate the model performance. We also tried the OLS on the Franke’s function with k-fold cross-validation, but we focused on the bootstrap method for all numerical tests.

**Results and discussion**

**Linear regression analysis**

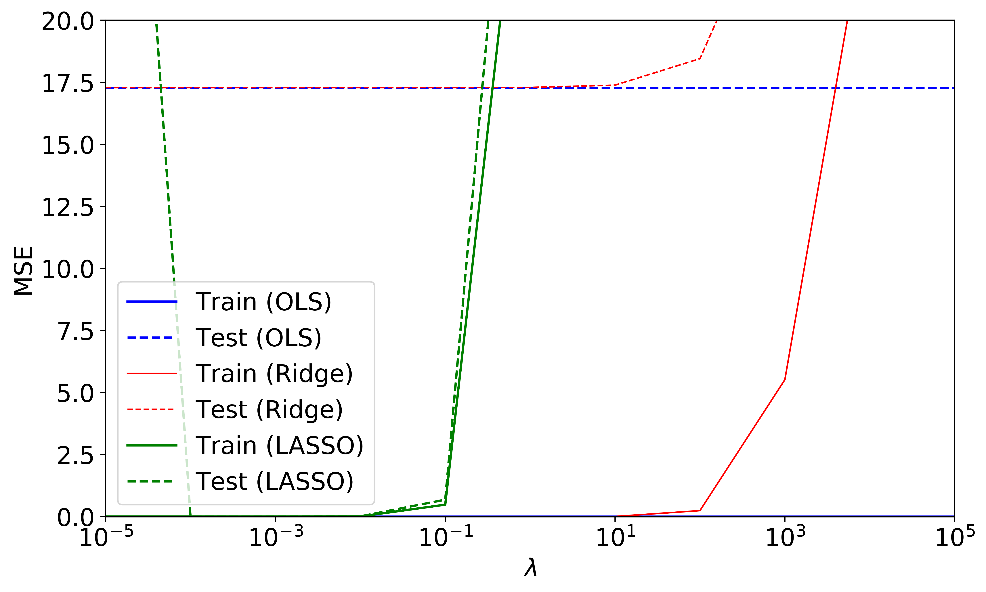
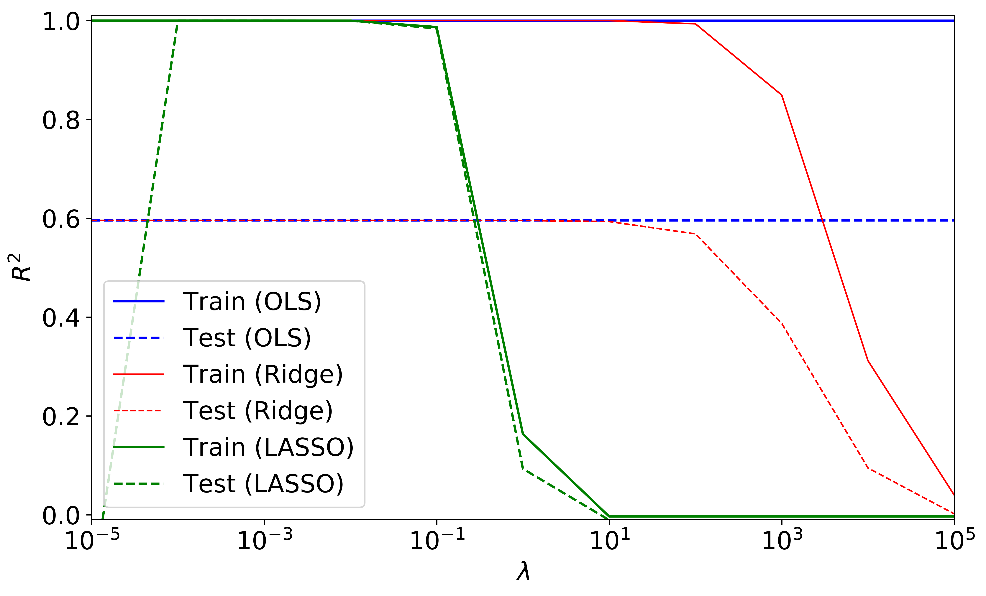
In order to study a binary system, i.e a system which is only defined by two values, such as or and , using both regression analysis and classification, we will turn our focus towards a commonly used model known as the Ising model. The Hamiltonian function for the one-dimensional Ising model reads (Mehta et al, 2018)

where , is the total number of spins, is known as the coupling constant. The coupling constant expresses the strength of the interaction between the adjacent spins. In physics the Hamiltonian is interpreted as state of energy, denoted , where in this context our aim is to learn the coupling constant from the training data . Before we start, we will rephrase the 1D equation (equation XXXX) to include every pairwise interaction between all spin variables. The equation then reads (Mehta et al, 2018)

where is now a matrix of coupling constants. In order to train the we will define equation XXX in a form which is more familiar from a linear regression point

Where

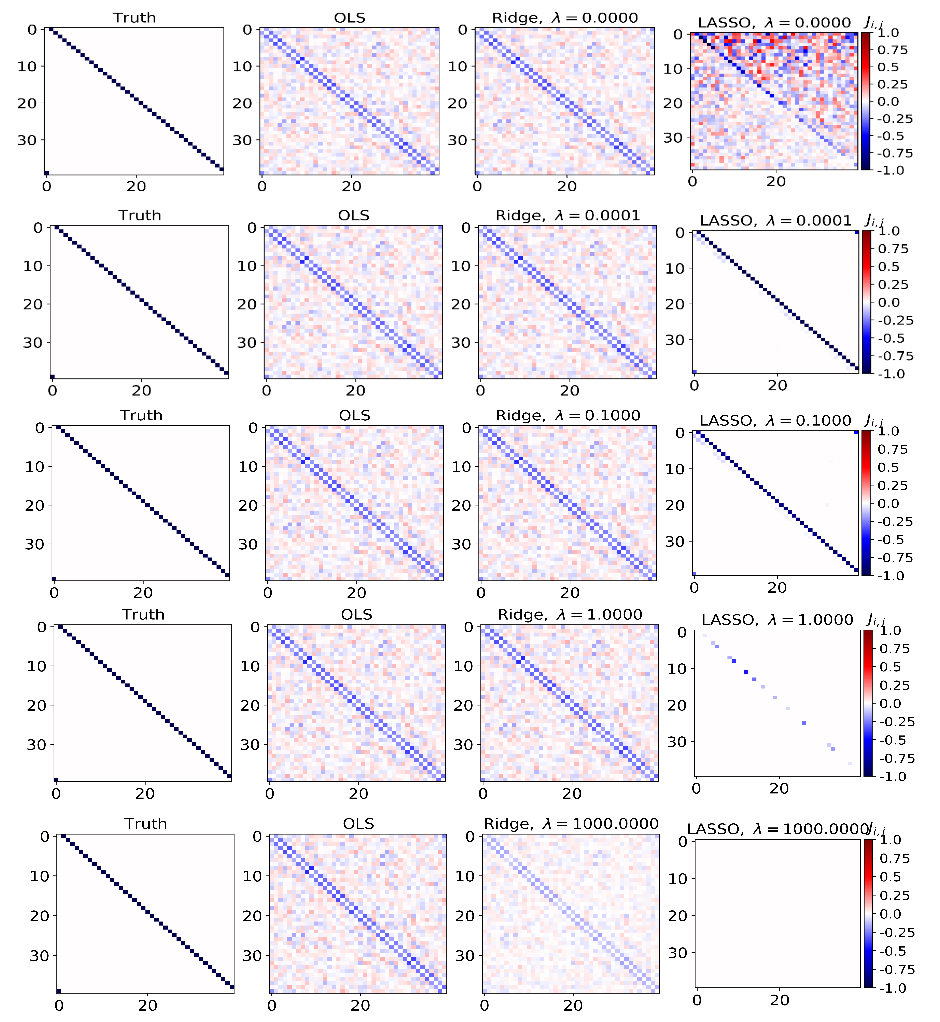
giving a -dimensional vector of the coupling constants corresponding to all the possible spininteractions given in the design matrix. The design matrix is now given by a -dimensional matrix

For the Ising model, the design matrix is singular, which force us to turn to other methods than the OLS estimator to provide a solution to the problem. In this case we will turn to SVD for both the OLS and Ridge estimators. The Lasso estimator is computed by an iterative approach using the scikit learn package. The training data was split into roughly 70/30 training- test set ratio. The results from the computed prediction error with no resampling technique, using OLS, Ridge and Lasso are presented in Figure XXXX. By considering the MSE and score for the test set, we observe that the Lasso estimation gives a better fit for the OLS and Ridge prediction error. This we believe is due to the sparsity of the coupling constant and the sparse nature of the Lasso estimator, which will therefore be favored over Ridge and OLS.

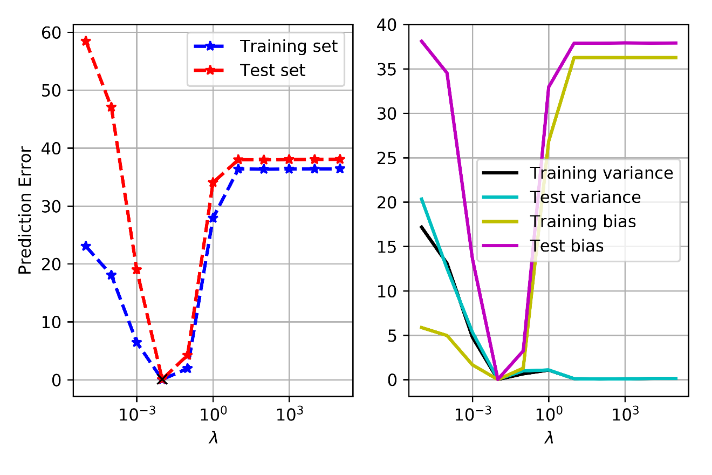
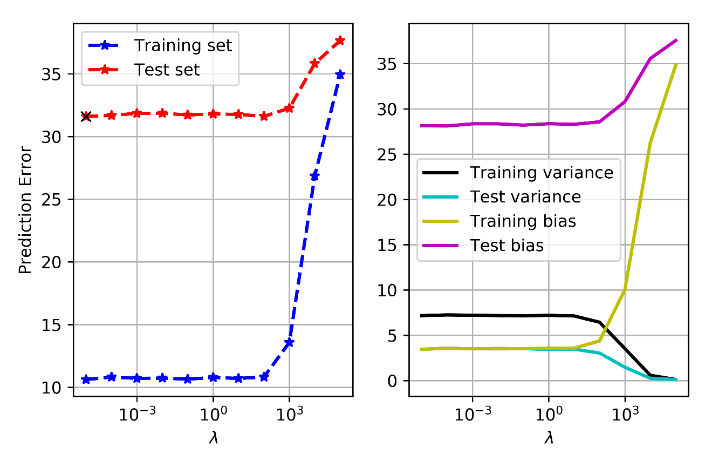
**Figure 1:** Computed prediction error (MSE) wrt model complexity using OLS, Ridge () and Lasso () estimation, and by applying SVD to avoid singularity problems. The prediction error from Lasso gives the overall best score, considering the test set, but shows overfitting issues at the lowest penalty values, .

The predicted estimators from OLS, Ridge and Lasso, by using a range of different values of are presented in Figure XXXXXXXXX, and we observe how the Lasso prediction resembles the ground truth very well. In order to better assess our model, we now turn to the bootstrap resampling technique, as explained in the theory section for model evaluation, and find the best-fit model based on the prediction error from the test set. In this example, the data was split using an 80/20 relationship on the training- and test set and used 200 bootstrap iteration. The MSE results and variance and bias decomposition from the Ridge and Lasso estimations using bootstrap resampling are displayed in Figure XXXX and Figure XXXXX respectively.

**Figure 2:** Computed score wrt model complexity using OLS, Ridge () and Lasso () estimation, and by applying SVD to avoid singularity problems. The score from Lasso gives the overall best score, considering the test set, but shows overfitting issues at the lowest penalty values, . We believe that the improved score for Lasso estimation is due to the sparse nature of the regularization, and since the coupling constants in this example is sparse, it will favor Lasso over Ridge and OLS.



**Figure 3:** Computed OLS, Ridge and Lasso estimators wrt model complexity. The Lasso estimator using a penalty parameter seems to give a good estimation. However, as we see in Figure XXXX, gives overfitting and high prediction error results.

****

**Figure 5:** Computed prediction error (MSE) (left) for training set and test set wrt to model complexity using Ridge. The training/test split was set to 80/20 and using 200 bootstrap iterations. The prediction error decreases as model complexity increases considering both the training and test set. However, its shows a turning point for, where prediction error increases again, due to overfitting. Considering the test set, the example shows a best fit for (black cross).

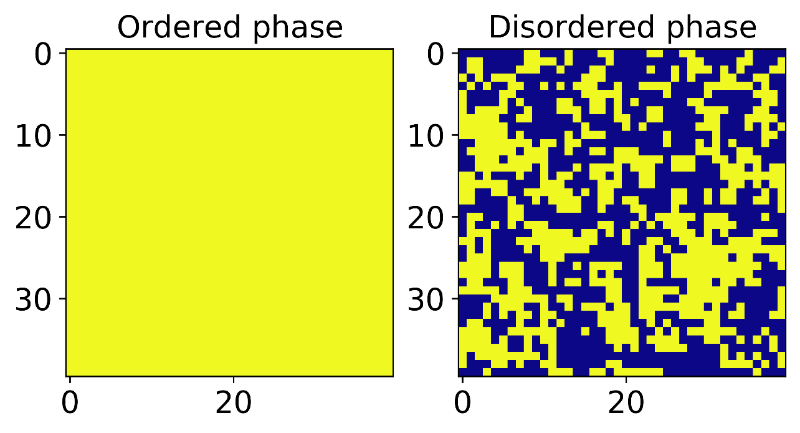
**Figure 4:** Computed prediction error (MSE) (left) for training set and test set wrt to model complexity using Ridge. The training/test split was set to 80/20 and using 200 bootstrap iterations. The prediction error decreases as model complexity increases considering both the training and test set. We see that variance increases as model complexity increases but is merely constant from penalty values . The bias shows similar trend but the bias decrease with model complexity. Considering the test set, the example shows a best fit for (black cross).

**Logistic regression classification**

We continue to study the binary system as in the regression analysis above, but now we turn our focus towards the 2D Ising model. The Hamiltonian function is somewhat different for the two-dimensional Ising model, and reads (Mehta et al, 2018)

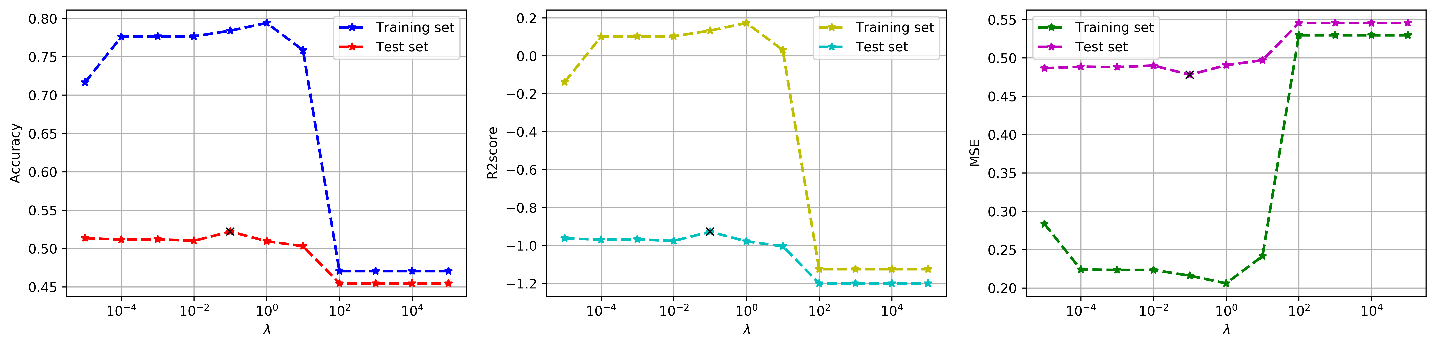
Which describes a 2D square lattice where indicates a summation over the nearest neighbors. Our ultimate goal is to train a logistic regression classifier to classify the magnetic phase. In our case this is a binary classification problem, since we deal with only ordered or disordered phases, as shown in Figure XXXXX. We are not going to classify the disordered phase at (or close to) the critical temperature. In this case, we will predict the magnetic phases by binary classification using as the disordered phase and as the ordered phase. The input data, consists of “images” defining the two-dimensional square lattice consisting of disordered and ordered magnetic phases. These “images” are merged and transformed to a design matrix where the individual “images” are structured as row-vectors

The target phases in are structured as a column vector

Figure XXXXX displays one ordered and one disordered phase, extracted from the design matrix. Before we train the classifier, the training data was separated into a training- and test set, with a train/test ratio of 0.8. As a first test we trained the logistic classifier using regularization and without resampling technique, such as bootstrap. In a second test round we assess the model using bootstrap resampling, using both and regularization. We will differentiate between the regularization parameters of and by and respectively. The regularization gives a different penalty term, as explained in the theory section, and leads to sparsity of the estimator as some of the parameters could be estimated as. This is in contrast to the estimators as for every parameter in .

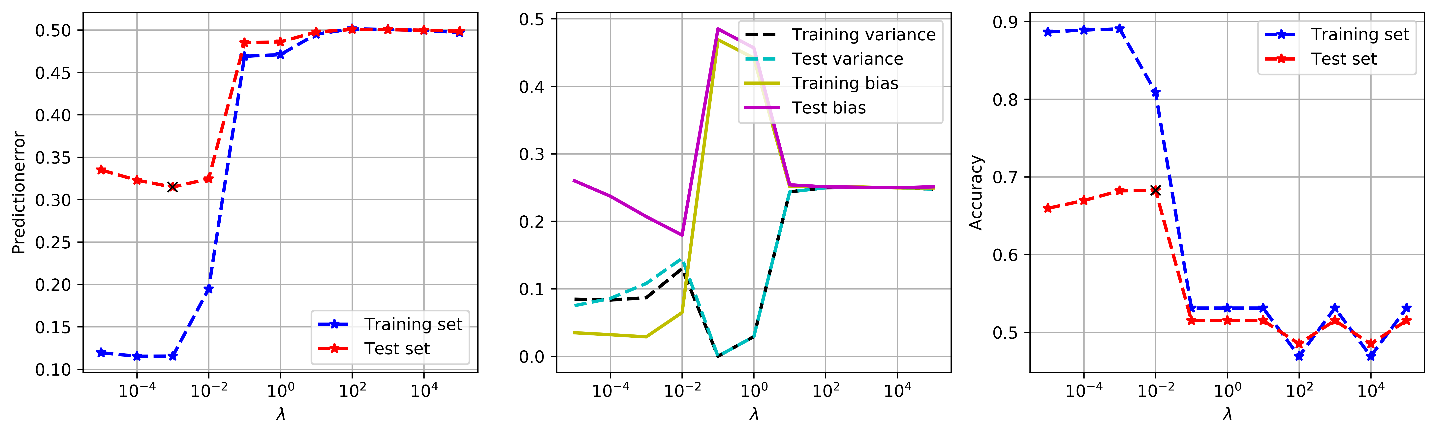
**Figure 5:** A single training example from the ordered magnetic phase and from the disordered magnetic phase, extracted from the training data.

We tested the logistic classifier on a small subset of the data, using 2000 training examples only regularization using different penalty values, 500 epochs, batch size of 50 and a learning rate. The results are presented in Figure XXXXXX. Our decision threshold for prediction was set to. The classifier accuracy in relation to the test data, show quite poor results, with a best fit of our model using a penalty parameter*.* We had hard time getting any good results on our test set using our own implementation; testing different learning rate, mini-batch sizes and epochs. There could exist many reasons for these results, like amount of training or finding an optimal training rate, batch size etc. However, another reason comes to mind is the characteristic of the model itself, and the regularization method used. The discrete characteristic of the Ising model, could possibly favor sparsity in its estimators. Here is where the attractiveness of the Lasso, or regularization comes in.

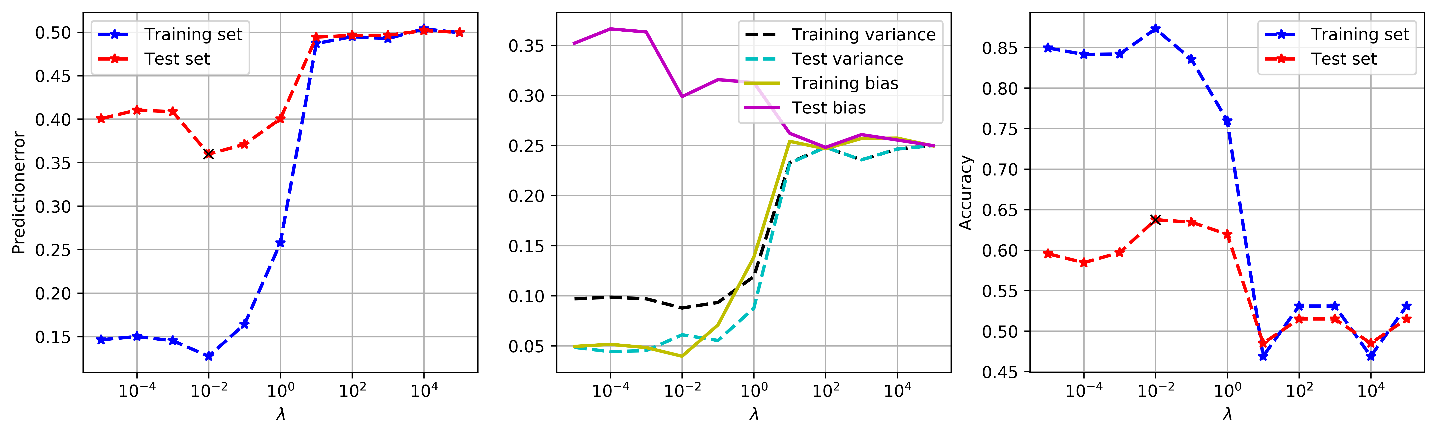


**Figure 6:** Computed accuracy score, score and prediction error (MSE) for training and test set, wrt to model complexity, using logistic regression with regularization. The training/test split was set to 80/20. The decision threshold for our prediction was set to. The prediction error decreases with increasing model complexity, but increases again at the highest model complexity. The prediction of the test set shows a best fit using. However, these classification results of a sub-sample (2000 training examples) of the two-dimensional Ising model. We trained the classifier using 500 epochs, a batch size of 50, a learning rate and and different values of the penalty parameterHowever, the results on the test set are quite poor.

In the second example, we utilized bootstrap resampling for model assessment. In this example we also utilized the scikit learn in-built function linear\_model.SGDClassifier with optimal learning rate. In this example we tested and compared the results from and regularization. The results from regularization and regularization are presented in Figure XXXX and Figure XXXX respectively.



**Figure 7:** Computed prediction error (left) and accuracy score (right) for training and test set, wrt to model complexity range, using logistic regression with regularization and bootstrap resampling. The decision threshold for our prediction was set to. From the bias-variance decomposition (middle) we see that variance and bias varies more wrt model complexity in these computations compared to regularization. The prediction of the test set shows a best fit using, same as with . However, we see an increase in model accuracy compared to , which could be related to the sparsity of the estimators.



**Figure 8:** Computed prediction error (left) and accuracy score (right) for training and test set, wrt to model complexity, using logistic regression with regularization and bootstrap resampling. The decision threshold for our prediction was set to. From the bias-variance decomposition (middle) we see that variance decreases and bias increases as model complexity increases, which is counter intuitive. However, we find no explanation for why this result. The prediction of the test set shows a best fit using. We also see a slight increase in model accuracy compared to the simpler example displayed in Figure XXXXXX.

**Neural network classification**

We now turn our focus over on neural network for the classification problem of the two-dimensional Ising model.

**References**

Hastie, T., Tibshirani, R., Friedman, J. (2001). *The Elements of Statistical Learning*. New York, NY, USA: Springer New York Inc.

Ng, A. and Katanforoosh, K. (2018). *CS229 Lecture Notes: Deep Learning,* <http://cs229.stanford.edu/notes/cs229-notes-deep_learning.pdf>.

Mehta, P., Bukov, M., Wang, C., Day, A., Richardson, C., Fisher, C. K., Schwab, D. J., (2018). *A high-bias, low-variance introduction to Machine Learning for physicists,* arXiv:1803.08823

van Wieringen W. N. (2015). *Lecture notes on ridge regression*, arXiv:1509.09169

Appendix A