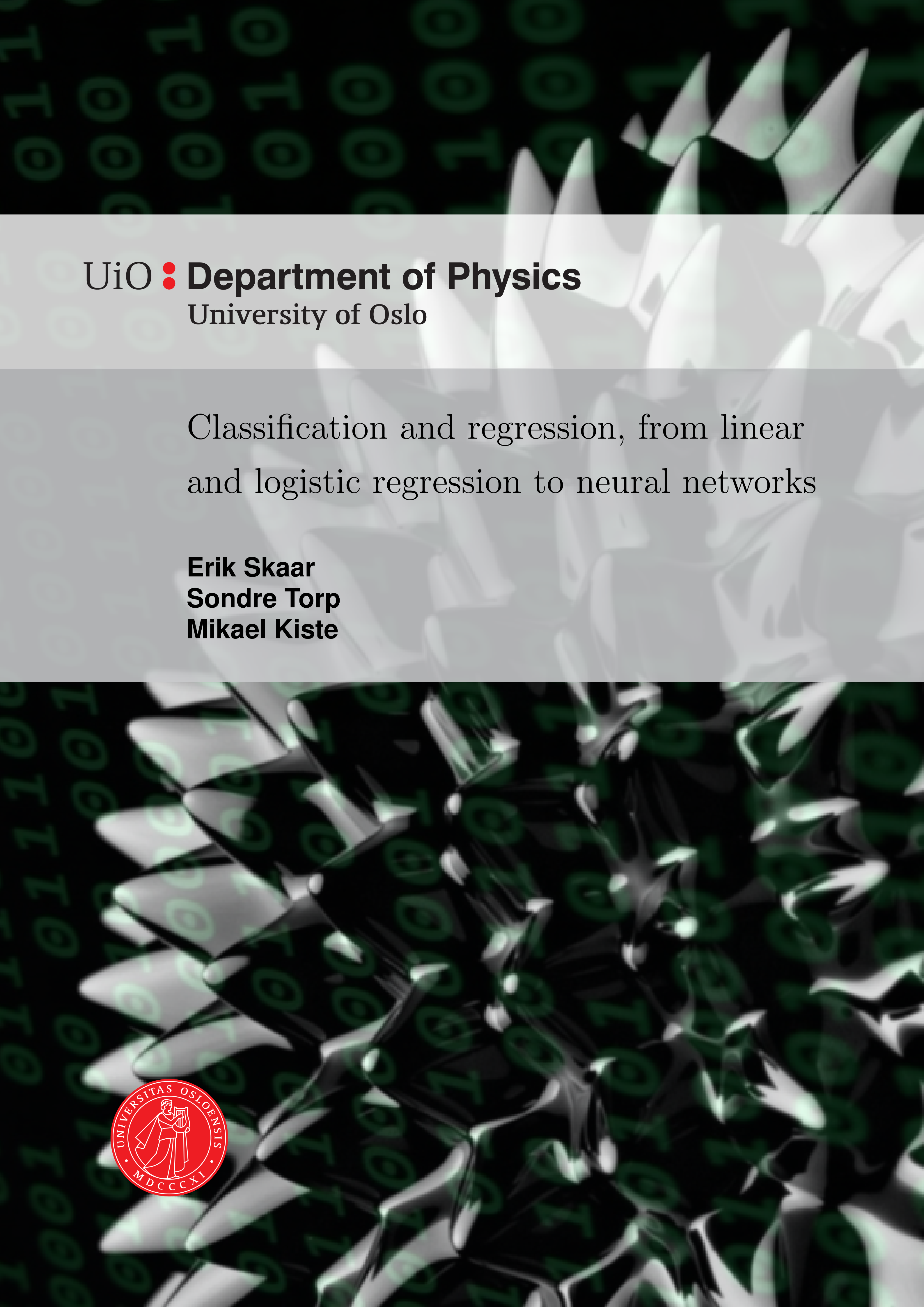

UiO : **Department of Physics**
University of Oslo

Classification and regression, from linear and logistic regression to neural networks

Erik Skaar
Sondre Torp
Mikael Kiste

Contents

Abstract

1 Introduction

2 Theory

2.1 the Ising model

The Ising model describes a coupled system. Where only the nearest neighbor affect each other. In this report the Ising model will be applied to a two dimensional magnetic system. This will be a grid of spins, where each spin s_i can either have 1 or 0 as value. The total energy is expressed as:

$$E = -\sum_{<i,j>} J_{i,j} s_i s_j$$

Where the symbol $<$ kl $>$ indicates that we sum over nearest neighbors only. If we assume that each coupling has the same magnitude J, then the energy is expressed as:

$$E = -J \sum_{<i,j>} s_i s_j \tag{1}$$

2.1.1 Periodic boundary conditions

When working with a finite matrix we run into a problem with the boundaries. They are missing neighbours. We solve this by introducing periodic boundary conditions. This means that the right neighbour for S_n is assumed to take the value of S_1.

2.2 Statistical physics

2.2.1 the partition function

Boltzmann distribution is used as the probability distribution. Boltzmann distribution states the probability for E_i is proportional to $e^{-\beta E_i}$, where β is $\frac{1}{k_B T}$. k is the Boltzmann constant. For this to be a probability distribution, it needs to be normalized. To normalize the distribution divide the sum of probabilities by a constant Z:

$$1 = \frac{\sum_i e^{-\beta E_i}}{Z}$$
$$Z = \sum_i e^{-\beta E_i}$$

Z is called the partition function.

2.2.2 Calculation of values

The partition function is very useful. In combination with the Boltzmann distribution we get a expression for the probability.

$$P(E_i) = \frac{e^{-\beta E_i}}{Z}$$

For finding a mean value, one can simply make a sum over $P(E_i)$ multiplied by the value of interest. For instance the mean energy is given by:

$$\langle E \rangle = \sum_i E_i P(E_i)$$

Expressions for important expectation values can be derived such for the energy E, magnetic moment |M|, specific heat capacity C_v and the susceptibility χ. The expressions used in this report are listed below[**compphys**]:[1]

$$\langle E \rangle = \sum_i E_i P(E_i) \tag{2}$$

$$\langle |M| \rangle = \sum_i M_i P(E_i) \tag{3}$$

$$\langle C_V \rangle = \frac{1}{kT^2}\left(\langle E^2 \rangle - \langle E \rangle^2\right) \tag{4}$$

$$\langle \chi \rangle = \frac{1}{kT}\left(\langle M^2 \rangle - \langle |M| \rangle^2\right) \tag{5}$$

2.3 Phase transition

The two dimensional Ising model is able to predict a phase transition in the material. At a critical temperature T_C the quantities for the material will start to behave differently. For C_V and for χ the phase transition is a sharp peak when plotted against Temperature. For $|M|$ and E it can be seen, but only as a slight change in value.

A second order phase transition is characterized by a correlation length. For finite lattice the correlation length is equal to the length of the system. T_C can be obtain through scaling of the results from a finite system with a infinite system:

$$T_C(L) - T_C(L = \infty) = aL^{\frac{-1}{v}} \tag{6}$$

a is an unknown constant and v = 1. For finding a we use eq. 6 with two different L. Substract the expression with L_i by the expression with L_j and we get:

$$T_C(L_i) - T_C(L_j) = a\left(L_i^{\frac{-1}{v}} - L_j^{\frac{-1}{v}}\right)$$

$$a = \frac{T_C(L_i) - T_C(L_j)}{L_i^{\frac{-1}{v}} - L_j^{\frac{-1}{v}}} \tag{7}$$

We combine this with eq. 6 and we get an expression for $T_C(\infty)$:

$$T_C(L) - T_C(L = \infty) = aL^{\frac{-1}{v}}$$

$$T_C(L = \infty) = T_C(L) - \frac{T_C(L_i) - T_C(L_j)}{L_i^{\frac{-1}{v}} - L_j^{\frac{-1}{v}}} L^{\frac{-1}{v}} \tag{8}$$

[1]lecture note page 420

2.4 Potts model

2.5 Ordinary least squares

We want to get a specific solution to the equation

$$\hat{y} = \hat{X}\hat{\beta} + \hat{\epsilon} \tag{9}$$

Where $\hat{y}$ is a vector of our measured values, $\hat{X}$ is a matrix containing variables and determines how we want to fit our data, $\hat{\beta}$ is a vector of the parameters for our fit and $\hat{\epsilon}$ is a vector representing the error in our datapoints (often termed the residuals, representing how far off our prediction is from the measurements). The variables $\hat{y}$ and $\hat{X}$ are fixed and we want to choose parameters $\hat{\beta}$ in such a way that the errors $\hat{\epsilon}$ are minimized. An example might help clarify the situation

Lets say we have conducted an experiment where we have measured the position of a ball launched straight up into the air from a cannon. Neglecting air resistance we know that the analytical solution is on the form of a second order polynomial $x(t) = x_0 + v_0 t + at^2$, where x_0 and v_0 are the initial conditions for the position and velocity respectively. This analytical solution is our model, but the actual measured values could (and indeed probably would) differ from this, simply due to errors in the measurement or other effects coming into play that the model has not accounted for (like air resistance). In any case, if we measured the position n times our linear algebra problem could be stated like this:

$$\begin{bmatrix} x(t_0) \\ x(t_1) \\ \vdots \\ x(t_{n-1}) \\ x(t_n) \end{bmatrix} = \begin{bmatrix} (t_0)^0 & (t_0)^1 & (t_0)^2 \\ (t_1)^0 & (t_1)^1 & (t_1)^2 \\ \vdots & \vdots & \vdots \\ (t_{n-1})^0 & (t_{n-1})^1 & (t_{n-1})^2 \\ (t_n)^0 & (t_n)^1 & (t_n)^2 \end{bmatrix} \begin{bmatrix} x_0 \\ v_0 \\ a \end{bmatrix} + \begin{bmatrix} \epsilon_0 \\ \epsilon_1 \\ \vdots \\ \epsilon_{n-1} \\ \epsilon_n \end{bmatrix}$$

Or, again, stated through vectors and matrix notation

$$\hat{x} = \hat{T}\hat{\beta} + \hat{\epsilon}$$

With some of the variable names adjusted simply to better indicate the represented values we have in this problem. Essentially we want to determine the variables x_0, v_0 and a so that the error terms are minimized (the equation has solutions for all $\hat{\beta}$s, but most of those are horrible fits with huge error terms). Note that there are two dimensionalities coming into play here. n is the number of measurements and determines the length of the column vectors $\hat{x}$ and $\hat{\epsilon}$. In addition there is a second dimension that determines the number of columns in the matrix $\hat{T}$ and the length of the vector $\hat{\beta}$. This number, say m, indicates the complexity of our model. When doing a polynomial fit, $m - 1$ is the order of the polynomial we want to fit our data to. So in this case, where we want to fit a second degree polynomial, we have $m = 3$.

The example above gives an impression of what the variables in the linear algebra equation represents but is quite specific and we can generalize a bit. For instance, it is not necessary to fit a polynomial at all. By changing our $\hat{X}$ matrix we could fit to any orthogonal function that we would like. The number of datapoints can be anything we want as log as $m \leq n$.

A key part of the fitting is how to minimize the so called "cost-function". In our case we want to minimize the ϵ's. There are different ways of doing this, but perhaps the simplest one is to do an **Ordinary Least Squares** (OLS) fit. That is to say when taking the difference between our predicted values and the measured values (essentially being the residuals) we want the squared sum of the difference for each datapoint to be as low as possible. That is to say that we want to minimize the function

$$Q = \sum_{i=0}^{n-1} \epsilon_i^2 = \hat{\epsilon}^T \hat{\varepsilon} = \left(\hat{y} - \hat{X}\hat{\beta}\right)^T \left(\hat{y} - \hat{X}\hat{\beta}\right)$$

We can see that taking the squared sum of all the elements in $\hat{\epsilon}$ is the same as taking the inner product of the vector with itself, and that we can use equation 9 to develop the expression further. We are interested in finding the parameters $\hat{\beta}$ that leads to the minimization of the squared sum of the residuals. So now that we have a function of the squared sum of the residuals with $\hat{\beta}$ as a variable it is a simple matter of finding when the derivative of this function with respect to $\hat{\beta}$ is zero; as this will give us the minimum (it must, of course, be a minimum as only a quite specific $\hat{\beta}$ will give a good fit and other values, deviating from this, would only increase the residuals squared sum. The possibility of more than one minimum is not something we need to worry about at this point). We will here state the derivative without further explanation, suffice it to say that looking at the expanded indexed expression (i.e. not on vector form) one quickly comes to the conclusion that this must indeed be the correct derivative.

$$\begin{aligned} \frac{\partial Q(\hat{\beta}}{\partial \hat{\beta}} &= 0 = \hat{X}^T \left(\hat{y} - \hat{X}\hat{\beta}\right) \\ \hat{X}^T \hat{y} &= \hat{X}^T \hat{X} \beta \\ \\ \hat{\beta} &= \left(\hat{X}^T \hat{X}\right)^{-1} \hat{X}^T \hat{y} \end{aligned} \tag{10}$$

And by the magic of linear algebra we have arrived at an analytical solution for the parameters $\hat{\beta}$ we need for an OLS regression. It is, however, important to note that we assume that the matrix $\hat{X}$ here is invertible.

2.6 Evaluations of our models

2.6.1 Mean square error

The Mean Square Error (**MSE**) can give a measure of the quality of our estimator. It is defined as

$$MSE(\epsilon) = \frac{1}{n} \sum_{n}^{n-1} \epsilon^2 \tag{11}$$

As such it can be thought of as the average of the square of our residuals. We see that our OLS method minimizes the MSE of our predictor variables. Of course, as easily seen from the definition, the MSE can never be negative and lower values means that we have a better prediction (at zero there is a perfect fit).

2.6.2 R^2 score - The Coefficient of Variation

In regression validation the R^2 is the gold standard when it comes to measuring goodness of fit. In straight terms it is the proportion of the variance in the dependent variable that is predictable from the independent variable(S).[**coef**]

$$R^2 = 1 - \frac{\sum(y_i - \tilde{y}_i)^2}{\sum(y_i - \bar{y}_i)^2} \tag{12}$$

Where y_i are the indexed response variables (data we want to fit) and $\tilde{y}_i$ is the predictor variables from our model (so $\epsilon_i = y_i - \tilde{y}_i$). The average of the response variables is denoted $\bar{y}_i$. In our case it the second term can also be considered as the ratio of **MSE** to the variance (the $1/n$ factors kill each other in a fraction). Let's interpret the formula step by step to get an impression of how it differentiates a poor from a good fit. If the residual sum of squares (SS_{res}) is low we have a good fit. However, we should compare this to the spread of our response variables. After all, if the response variables are all nicely distributed close to the mean then getting a good SS_{res} is not that impressive. We therefore do a sort of normalization in the fraction, taking the scale of our data into consideration. In the simplest polynomial fit, using a zeroth order polynomial (just a constant), we see that our model would just be a constant function of the mean. The sums would be equal, returning unity on the fraction and the total R^2 score would be zero. In the other extreme, if our model fits perfectly, then SS_{res} would be zero and the R^2 score would be one. In this sense we have a span of possible R^2 scores between zero and one, from the baseline of the simplest model at zero and a perfect fit at one. The R^2 score is useful as a measure of how good our model is at predicting future samples.

2.7 Ridge

A problem with the **OLS** method through linear regression is that the matrix is not necessarily invertible. In this case it is basically impossible to model the data using linear regression [**morten-reg**]. When there are many columns in the $\hat{X}$ matrix it is less likely that the columns are all linearly independent, which is a requirement to get the inverse $(\hat{X}^T\hat{X})^{-1}$. Since the number of columns increase when the complexity of our fit increases (as mentioned earlier, if the polynomial we want to fit is of order n then the number of columns is $m = n + 1$) we understand that a more complex model decreases the likelihood of invertibility. Since there is no longer a unique solution to our problem, it is an example of an ill-posed problem that is either overdetermined with more equations than unknowns (oftentimes leading to no solution) or underdetermined with more unknowns than equations (resulting in many potential solutions). These situations correspond to an over-fitting or an under-fitting respectively. [**wiki-tikhonob**] A simple solution to this linear algebra problem is to add a diagonal matrix, $\lambda\hat{I}$, term to the matrix that is to be inverted. This shrinks the regression coefficients so that $\hat{\beta}$'s with smaller norm are preferred. In this way we superimpose a quality control variable on our solution space allowing us to arrive at a unique solution with (relatively) small variance. [**hastie**] In the end we only need to adjust our linear algebra equation slightly

$$\hat{\beta} \quad = \left(\hat{X}^T\hat{X} + \lambda\hat{I}\right)^{-1}\hat{X}^T\hat{y} \tag{13}$$

There is a caveat here when considering the coefficient determining the y-intercept. If this is included in the regularization the result would depend on the origin chosen for y.[**hastie**] This is a problem as we expect that adding a constant to the response variables should simply shift the predictor variables by the same amount, not change the shape of our fit. As an illustration, if we return to our previous cannon and ball experiment, we don't want the time at which we start the clock, be it at the moment of launch or a minute before, to lead to a completely different polynomial fit. To avoid this problem it is possible to center the inputs by simply estimating β_0 by the average $\bar{y}$ and doing a ridge regression by equation 13 with the remaining coefficients but replacing the elements in $\hat{X}$ with $x_{ij} - \bar{x}_j$.

2.8 Lasso

The Lasso (least absolute shrinkage and selection operator) method works in much the same way as Ridge in that it incorporates variable selection and regularization in the regression. But the constraints makes the solutions nonlinear and there is no closed form expression as in the ridge regression.[**hastie**] We therefore need to specify the workings of the method more explicitly

$$\hat{\beta} = \text{argmin} \sum_{i=1}^{N} \left(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j \right)^2 \tag{14}$$

Under the restriction

$$\sum_{j=1}^{p} |\beta_j| \leq t \tag{15}$$

At first glance this expression may seem foreign when given on this form but it is essentially the same as for Ridge with the only difference being the restriction in Ridge

$$\sum_{j=1}^{p} {\beta_j}^2 \leq t \tag{16}$$

We recognize equation 14 as a function that, when we exclude β_0 due to centering, finds the minimum of SS_{res} with an additional constraint that discards some solutions. In the case of Lasso the constraint makes sure that the L_1 norm is less than t while Ridge considers the L_2 norm (i.e. euclidian norm).

2.9 K-fold

K-folding is a cross validation technique that allows us to generalize the trends in our data set to an independent data set. In this way we can circumvent typical problems like over-fitting and selection bias.[**cross-valid**] The approach for the technique is pretty straight forward. Instead of doing a regression on the entire data set, we first segment it into k number of subsets of equal size (making sure to pick out the variables randomly before distributing them to the subsets). Now we choose one subset to be the 'control' or 'validation' set while the rest of the subsets are the training sets. We then perform the regression we want to use on the training set, arriving at some data fitting that is our prediction. From here it is a straight forward process to compare how well our predicted variables compare to the validation variables, for example through the R^2 score function. However, even though our subsets are picked randomly, the validation subset we used could potentially not be a representative selection of the entire set. Therefore we make sure to repeat the process k times, each time using a new subset as the validation subset. After all this is done we can simply calculate the average of the scores to get the predictive power of our model. As an added benefit, since we are doing the calculations anyway, we can use the average of our predictions as our final fit. Cross validation techniques are extremely useful when the gathering of new data is difficult or, sometimes, even impossible, as we are using the extra computational power at our disposal to squeeze the most amount of relevant information out of our precious data.
Combining cross validation with the penalty parameter t used in Ridge or Lasso (eq: 15 and 16) can make for an even better fit. Instead of trying to pick a good t from our intuition we can actually get a numerical measure of the predictive power of our model as a function of t by performing the cross validation method for a range of t's, choosing the penalty parameter that optimizes the predictive power of the model.[**morten-reg**]

2.10 Logistic regression

2.11 Neural networks

2.11.1 Tensor flow

2.11.2 Back progagation algorithm

3 Method

3.1 Metropolis algorithm

The metropolis algorithm only has a few steps. First, pick one site in the matrix of spins. This process need to be random. For that site, calculate the energy difference if the spin is flipped. Then the algorithm decide whether to flip the spin or not. This is decided based on the energy difference. If the difference is negative flip, then flip the spin. If not, then pick a random number between 0 and 1 and if this number is less then $e^{-\beta \Delta E}$ flip the spin. Else keep the spin. Finally update expectation values.

3.1.1 Precalculate

The energy difference is expressed as an exponential function. Exponential values are expensive to calculate. In two dimensions there is a finite number of energy differences. We can precalculated the exponentials. By calculating these in advance the program will run more efficient. It can be shown that the energy difference then is[**compphys**]:

$$\Delta E = 2Js_j \sum_{<k>} s_k$$

4 Implementation

The metropolis algorithm was implemented as discussed in section 3.1 in the programs called main-"...".cpp. Their is a few different versions of the main.cpp. The only difference is basicly how they write to file. All of the programs discussed in this section can be found at github.

These calculations are expensive in terms of FLOPs. That is why a parallelized version has been made. Not all the code needs to be parallelized. Most of the code in the parallelized version is the same as for the non-parallelized. Except each thread open a specific file for that thread and runs the metropolis algorithm for $\frac{1}{\text{nr. of temperatures}}$. What temperature that each thread calculate is determined by the rank of the thread. Since the rank is a unique number for each thread, all the temperatures calculated is unique from the other temperatures.

MPI is used since it is easy to implement and it does not have shared memory. The parallelized version should be (nr. of threads - 1) times faster then the normal version.

Table 1: The grids ran for 50'000 Monte Carlo cycles. The test ran on a macbook pro 13. It has a dual core CPU. Expected difference is 2.

Size	Normal	MPI	Expected difference	Actual difference
40x40	15.251s	5.991 s	2.000	2.546
60x60	33.923s	13.351 s	2.000	2.541
100x100	92.584s	36.245 s	2.000	2.554

The difference was higher, then expected. This is due to the Hyper-Threading technology in this CPU [2].

For size scaling we expect a time increased proportional to the size increase squared.

Table 2: The table shows how we expect the time to develop and how it actually it develops. There is a minor difference from expected and calculated and that comes from the fact that the program does more then just the algorithm and the fact that the algorithm has not been perfectly implemented.

Size	Expected time	Actual time	$\frac{T_i}{T_{40}}$
40x40	x	8.490 s	1.000
60x60	2.25x	19.350 s	2.279
100x100	6.25x	54.068 s	6.368

[2] Intel Hyper-Threading Technology

5 Result & Discussion

6 Conclusion