Classification and Regression

from linear and logistic regression to neural networks

Elisabeth Strøm

November 8, 2019

Contents

1 Introduction

In this project we test three different linear regression methods: The Ordinary Least Squares (OLS), Ridge regression, and Lasso regression.

This project consists of two parts. First we perform a logistic regression on credit card data taken from ... TAIWAN. Next we create a Feed Forward Neural Network, and perform the logistic regression one more time.

In the next part, we perform a linear regression analysis using our Neural Network on the Franke function (Franke, 1979), and compare our output with that found in our project 1 report..

In Section 2, we go through the theory behind the methods we will use, and also show how we will implement them numerically.

Our results can be seen in Section 3, and we will discuss them in Section 4. Finally, we sum up our conclusions in Section 5.

2 Method

2.1 Logistic Regression

2.1.1 Gradient descent

Gradient descent and stochastic gradient descent.

2.2 Linear Regression

Short, OLS, Ridge and Lasso?

2.3 Neural Network

General idea,

2.3.1 Feed Forward

2.3.2 Backward propagation

2.3.3 Gradients

Gradients, logistic regression: COST:sigmoid, Activation: sigmoid, output: softmax, linear regression: COST: MSE, activation: sigmoid, output: linear (f(x)=x),

2.4 Ordinary Least Squares regression

We start with the simplest of the regression methods: The Ordinary Least Square.

The idea behind this method, is to fit a polynomial, $\tilde{\boldsymbol{y}}$, to our, usually, noisy data $\boldsymbol{y}$, and try to model the underlying, true function f. In the case of one predictor $\boldsymbol{x}$,

$$y_i \approx \tilde{y}_i + \epsilon_i = \sum_{j=0}^{m-1} \beta_j \boldsymbol{x}_i^j + \epsilon_i. \tag{1}$$

So the elements of the design matrix $\boldsymbol{X}$ (Equation **??**) are now $x_{i,j} = x_i^j$.

In our project, we try to recreate the Franke function, which has two independent parameters, $x, y \in [0, 1]$,

$$\begin{aligned} f(x,y) = &\frac{3}{4}\exp\left(-\frac{(9x-2)^2}{4} - \frac{(9y-2)^2}{4}\right) + \frac{3}{4}\exp\left(-\frac{(9x+1)^2}{49} - \frac{(9y+1)}{10}\right) \\ &+ \frac{1}{2}\exp\left(-\frac{(9x-7)^2}{4} - \frac{(9y-3)^2}{4}\right) - \frac{1}{5}\exp\left(-(9x-4)^2 - (9y-7)^2\right). \end{aligned} \tag{2}$$

Our estimate of $f(\boldsymbol{x}, \boldsymbol{y})$, is denoted as $\tilde{\boldsymbol{z}}$, and our noisy data we perform an analysis on, is then $z_i = f(\boldsymbol{x}_i, \boldsymbol{y}_i) + \epsilon_i$, where $\epsilon_i \sim \mathcal{N}(0, 1)$. The elements of the design matrix will then also assume a slightly different shape, where the order of the polynomial $\tilde{z}_i$ is a result of the polynomial order of both $\boldsymbol{x}_i$ and $\boldsymbol{y}_i$ multiplied together. We will get back to how to implement this in Section 2.5.

The $\boldsymbol{\beta}$ coefficients are calculated from

$$\boldsymbol{\beta}^{\text{OLS}} = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{z}, \tag{3}$$

for the $\tilde{\boldsymbol{z}}$, calculated from $\boldsymbol{X}$ (Equation **??**) for different polynomial orders, that minimizes the Mean Squared Error, or the predictive error.

The variance of $\boldsymbol{\beta}$ is,

$$\text{Var}(\boldsymbol{\beta}) = \sigma^2 \cdot \text{diag}(\boldsymbol{X}^T\boldsymbol{X}^{-1}), \tag{4}$$

where σ^2 is still the variance of the noise $\boldsymbol{\epsilon}$.

2.5 Implementation

The first thing we do is create the $\boldsymbol{x}$ and $\boldsymbol{y}$. We make two arrays with evenly spaced points, for a total of a 100 each, between 0 and 1. Then we use the `numpy` command `meshgrid(x,y)` to create a meshgrid. $\boldsymbol{x}$ and $\boldsymbol{y}$ are then used to construct the Franke function $\boldsymbol{z} = f(\boldsymbol{x}, \boldsymbol{y})$. Next we flatten the arrays, and copy $\boldsymbol{z}$ and add the normally distributed noise $\boldsymbol{\epsilon}$. We know have the true function, `z1_true` and the noisy function `z1_noise`,

```
z = FrankeFunction(x, y)
z1_true = np.ravel(z)
# adding noise
z1_noise = np.ravel(z) + np.random.normal(0, stddev, size=z1_true.shape)
```

Next thing to do is create the design matrix $\boldsymbol{X}$. It is made so that the polynomial degree increases with the columns. Say we have a 2nd order polynomial, the first row then looks like

$$x_{0,*} = [1, x_0, y_0, x_0^2, x_0y_0, y_0^2].$$

Now, finding $\tilde{\boldsymbol{z}}$ is straight forward. Note that we will train our algorithm on the noisy data $\boldsymbol{z}^{\text{noise}}$, but assert the fit by calculating MSE with the true Franke function $\boldsymbol{z}^{\text{true}}$.

2.5.1 k-fold Cross-Validation

We implement the k-fold Cross-Validation. Here our data is shuffled and split into $k = 5$ equal parts. 1/5 is used as test data, wile the rest is training data. We then perform a regression analysis k times, where we change which is the test set every time. For each analysis, we estimate the MSE, and then, finally we take the mean of the k error estimates. MSE is calculated both for the test, and training set.

2.5.2 Standardizing the data

We standardize the datasets in the following way: For the training set $\boldsymbol{X}^{\mathbf{train}}$, we subtract the mean of each column from that column, and then we divide each column with the standard deviation of each column,

$$\boldsymbol{X}_{*,j}^{\text{train,c}} = \frac{\boldsymbol{X}_{*,j}^{\text{train}} - E[\boldsymbol{X}_{*,j}^{\text{train}}]}{\text{STD}(\boldsymbol{X}_{*,j}^{\text{train}})}, \qquad \text{for } j = 1, 2, \ldots, m-1. \tag{5}$$

For z, we subtract the mean only, $\boldsymbol{z}^{\text{train,c}} = \boldsymbol{z}^{\text{train,c}} - \text{E}[\boldsymbol{z}^{\text{train,c}}]$

We apply these same transformations to the test set, that is,

$$\boldsymbol{X}_{*,j}^{\text{test,c}} = \frac{\boldsymbol{X}_{*,j}^{\text{test}} - E[\boldsymbol{X}_{*,j}^{\text{train}}]}{\text{STD}(\boldsymbol{X}_{*,j}^{\text{train}})}, \qquad \text{for } j = 1, 2, \ldots, m-1. \tag{6}$$

and $\boldsymbol{z}^{\text{test,c}} = \boldsymbol{z}^{\text{test,c}} - \text{E}[\boldsymbol{z}^{\text{train,c}}]$. The columns now have a mean equal to 0, and a standard deviation equal to 1.

We add the mean again when calculating $\tilde{\boldsymbol{z}}$, that is

$$\tilde{\boldsymbol{z}}^{train} = \boldsymbol{z}^{train,c} + \text{E}[\boldsymbol{z}^{\text{train,c}}] \tag{7}$$

$$\tilde{\boldsymbol{z}}^{test} = \boldsymbol{z}^{test,c} + \text{E}[\boldsymbol{z}^{\text{train,c}}]. \tag{8}$$

Standardizing the data is not that important for the OLS method, but rather for Ridge and Lasso, where we have the penalty λ. λ puts constraints on the β-coefficients, but the coefficients size depends on the magnitude of each variable. It is therefore important to get the data on the same scales, so that no variable is penalized unfairly, and to be able to see which factors truly contribute the most to $\tilde{\boldsymbol{z}}$. Also, we no longer have an intercept, but it can be interpreted as being the $\text{E}[\boldsymbol{z}^{\text{train,c}}]$.

2.5.3 Linear regression

We first run the three regression analyses on the Franke Function data. Given that we know the true function already, we will train our algorithm with the noisy data, but calculate the MSE using the true function.

Next we move onto the terrain data, where we used the files found together with the project instructions. This is a quite large dataset, so we chose a small cutout, just the first 1000 pixels in x and y direction. We down-sampled this image, by only including every 30th pixel in either direction.

For the OLS, we calculate MSE for several polynomial degrees, to find which one offers the best fit.

For Ridge and Lasso, we must also find the best fit λ parameter that together with the right polynomial, minimizes the MSE.

The result of our analyses can be found in Section 3.

3 Results

Here we show the results of our regression analyses on the Franke function, and on the terrain data.

3.1 Cancer data

Logistic regression and NN.

3.2 The Franke function

Linear regression and NN

4 Discussion

5 Conclusion

All code can be found at https://github.com/sefthus/FYS-STK4155/tree/master/project2

References

Richard Franke. A Critical Comparison of Some Methods for Interpolation of Scattered Data, 1979. URL https://calhoun.nps.edu/handle/10945/35052.

Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *Elements of Statistical Learning*. Springer-Verlag New York, 2 edition, 2009. ISBN 978-0-387-84857-0. URL https://web.stanford.edu/~hastie/ElemStatLearn/.