FYS-STK4155 - Applied data analysis and machine learning

Project 1

Even M. Nordhagen

September 19, 2018

- Github repository containing programs and results: `https://github.com/evenmn/FYS-STK4155`

Abstract

Do not forget to be specific

Contents

1 Introduction

Should write some motivating words about how much regression is used in different fields etc..

2 Theory

2.1 Regression

A few general words about regression

2.1.1 Ordinary Least Square (OLS)

Suppose we have a set of points $\{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$, and we want to fit a p'th order polynomial to them. The most intuitive way would be to find coefficients $\vec{\beta}$ which minimize the error in

$$\begin{aligned} y_1 &= \beta_0 x_1^0 + \beta_1 x_1^1 + \ldots + \beta_p x_1^p + \varepsilon_1 \\ y_2 &= \beta_0 x_2^0 + \beta_1 x_2^1 + \ldots + \beta_p x_2^p + \varepsilon_2 \\ &\vdots \\ y_n &= \beta_0 x_n^0 + \beta_1 x_n^1 + \ldots + \beta_p x_n^p + \varepsilon_n, \end{aligned}$$

which for OLS is defines as

$$\text{MSE} = \sum_i \varepsilon_i^2 \tag{1}$$

NEED TO REWRITE THIS + COST FUNCTION

Standard cost function

$$Q(\vec{\beta}) = \sum_{i=0}^{n-1} (y_i - \tilde{y}_i)^2 = (\vec{y} - \hat{X}\vec{\beta})^T (\vec{y} - \hat{X}\vec{\beta}) \tag{2}$$

Instead of dealing with a set of equations, we can apply linear algebra. One can easily see that the equations above correspond to

$$\vec{y} = \hat{X}^T \vec{\beta} + \vec{\varepsilon}, \tag{3}$$

with

$$\hat{X} = \begin{pmatrix} x_1^0 & x_1^1 & x_1^2 & \ldots & x_1^p \\ x_2^0 & x_2^1 & x_2^2 & \ldots & x_2^p \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_n^0 & x_n^1 & x_n^2 & \ldots & x_n^p \end{pmatrix} \tag{4}$$

and

$$\vec{\beta} = (\beta_0, \beta_1, \ldots, \beta_p). \tag{5}$$

For a nonsingular matrix $\hat{X}$ (but not necessary symmetric) we can find the optimal $\vec{\beta}$ by solving

$$\vec{\beta} = (\hat{X}^T\hat{X})^{-1}\hat{X}^T\vec{y}, \tag{6}$$

which again corresponds to minimizing the cost function,

$$\vec{\beta} = \text{argmin}_{\vec{\beta}}\Big\{ \sum_{i=1}^{n} \Big(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j\Big)^2 \Big\}. \tag{7}$$

CONFIDENCE INTERVAL of $\hat{\beta}$: $\text{Var}(\hat{\beta})$.

This works perfectly when all rows in $\hat{X}$ are linearly independent, but this will generally not be the case for large data sets. If we are not able to diagonalize the matrix, we will not be able to calculate $(\hat{X}^T\hat{X})^{-1}$, so we need to do something smart.

Fortunately there is a simple trick we can do to make all matrices diagonalizable; we can add a diagonal matrix to the initial matrix.

2.1.2 Ridge regression

Ridge regression is a widely used method that can handle singularities in matrices. The idea is to modify the standard cost function by adding a small term,

$$Q^{\text{ridge}}(\vec{\beta}) = \sum_{i=1}^{N}(y_i - \tilde{y}_i)^2 + \lambda||\vec{\beta}||_2^2, \tag{8}$$

where λ is the so-called *penalty* and $||\vec{v}||_2$ is defined as

$$||\vec{v}||_2 = \sqrt{\vec{v}^T\vec{v}} = \Big(\sum_{i=1}^{N} v_i^2\Big)^{1/2}. \tag{9}$$

This will eliminate the singularity problem.

Further we find the optimal $\vec{\beta}$ values by minimizing the function

$$\vec{\beta}^{\text{ridge}} = \text{argmin}_{\vec{\beta}}\Big\{ \sum_{i=1}^{n} \Big(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j\Big)^2 + \lambda\sum_{j=1}^{p}\beta_j^2 \Big\}, \tag{10}$$

or we could simply solve the equation

$$\vec{\beta}^{\text{ridge}} = (\hat{X}^T\hat{X} + \lambda I)^{-1}\hat{X}^T\vec{\beta}. \tag{11}$$

In the latter equation we can easily see why this solves our problem.

2.1.3 Lasso regression

The idea behind Lasso regression is similar to the idea behind Ridge regression, and they differ only by the exponent factor in the last term. The modified cost function now writes

$$Q^{\text{lasso}}(\vec{\beta}) = \sum_{i=1}^{N}(y_i - \tilde{y}_i)^2 + \lambda||\vec{\beta}||_2, \tag{12}$$

and to find the optimal coefficients $\vec{\beta}$, we need to minimize

$$\vec{\beta}^{\text{lasso}} = \text{argmin}_{\vec{\beta}}\left\{ \sum_{i=1}^{n}\left(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j\right)^2 + \lambda\sum_{j=1}^{p}\beta_j \right\}. \tag{13}$$

2.1.4 General form

We can generalize the models above to a minimization problem where we have a q in the last exponent,

$$\vec{\beta}^{q} = \text{argmin}_{\vec{\beta}}\left\{ \sum_{i=1}^{n}\left(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j\right)^2 + \lambda\sum_{j=1}^{p}\beta_j^q \right\}, \tag{14}$$

such that $q = 2$ corresponds to the Ridge method and $q = 1$ is associated with Lasso regression. It can also be interesting to try other q-values.

2.2 Higher order regression

We can use the same approach as above when dealing with regression of higher order, since the problem is to fit a function to points, no matter how many components they have. We will first take a look at how we can fit a 2D polynomial to some terrain data, before we briefly describe how to fit a function of arbitrary order to points.

2.2.1 Terrain

A set of data points $\{(x_1, y_1, z_1), (x_2, y_2, z_2), \ldots, (x_n, y_n, z_n)\}$ gives some coordinates in space, which for instance can describe the terrain. The system of linear equations to solve can then be lined up as

$$\begin{aligned}
z_1 &= \beta_0 x_1^0 y_1^0 + \beta_1 x_1^1 y_1^0 + \beta_2 x_1^0 y_1^1 + \ldots + \beta_p x_1^p y_1^p + \varepsilon_1 \\
z_2 &= \beta_0 x_2^0 y_2^0 + \beta_1 x_2^1 y_2^0 + \beta_2 x_2^0 y_2^1 + \ldots + \beta_p x_2^p y_2^p + \varepsilon_2 \\
&\vdots \\
z_n &= \beta_0 x_n^0 y_n^0 + \beta_1 x_n^1 y_n^0 + \beta_2 x_n^0 y_n^1 + \ldots + \beta_p x_n^p y_n^p + \varepsilon_n,
\end{aligned}$$

when fitting a polynomial of p'th order. In general the order associated with x-direction do not need to be the same as the order associated with y-direction.

We can set up a similar equation as for the first order case, presented in equation (3), but we will now (typically) have $\vec{z}$ on the left hand side and $\hat{X}$ will contain both x- and y-coordinates:

$$\vec{z} = \hat{X}^T \vec{\beta}. \tag{15}$$

The coordinate matrix $\hat{X}$ will now look like

$$\hat{X} = \begin{pmatrix} x_1^0 y_1^0 & x_1^1 y_1^0 & x_1^0 y_1^1 & x_1^1 y_1^1 & \dots & x_1^p y_1^p \\ x_2^0 y_2^0 & x_2^1 y_2^0 & x_2^0 y_2^1 & x_2^1 y_2^1 & \dots & x_2^p y_2^p \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ x_n^0 y_n^0 & x_n^1 y_n^0 & x_n^0 y_n^1 & x_n^1 y_n^1 & \dots & x_n^p y_n^p \end{pmatrix} \tag{16}$$

and we can again use the OLS method to find $\vec{\beta}$, i.e, calculating

$$\vec{\beta} = (\hat{X}^T \hat{X})^{-1} \hat{X}^T \vec{z}. \tag{17}$$

Similarly we can use ridge and lasso regression in the same way as when fitting 1D polynomials.

In this project one we will actually deal with terrain data. Firstly we implement some regression tools which fit a 2D function to points in space. To verify the implementation, we pick points from a known function such that we know the result. The specific verification function used is the Franke Function, which looks like

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

and is a smooth 2D function as one can see in figure (3).

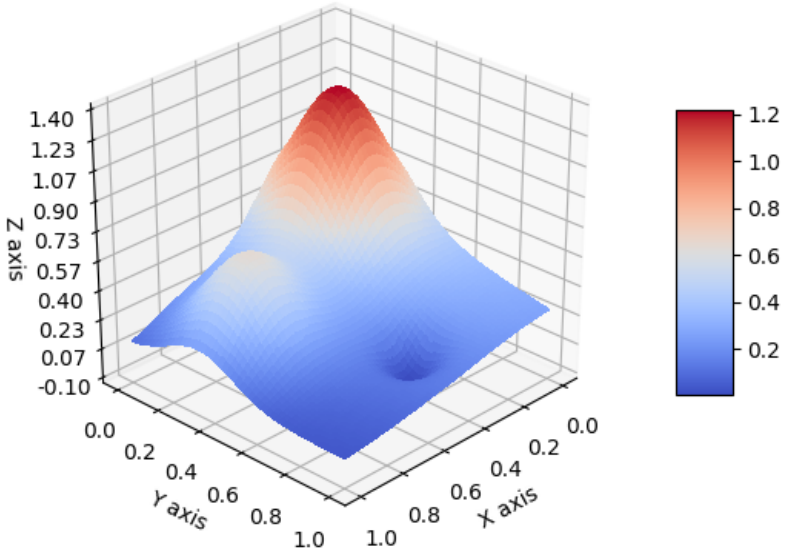


Figure 1: The Franke function in the interval $x \in [0, 1]$, $y \in [0, 1]$, $z \in [0, 1.2]$.

Secondly we use the verified regression tools to fit a polynomial to real terrain data. NEED TO ADD A SENTENCE OR TWO HERE

2.2.2 Higher order

There should be no surprise that we can extend the theory above to even higher orders. Although we stick to 2D regression in this project, I add this section for completeness.

2.3 Error analysis

When dealing with data sets, we always want to know how much the data points vary from each other, in other words we want to calculate the variance of the data. Often we are studying multiple data sets at the same time, ...

Sample variance:

$$\sigma_x^2 = \frac{1}{N}\sum_{i=1}^{N}(x_i - \bar{\mu})^2 \tag{18}$$

Sample mean:

$$\bar{\mu} = \sum_{i=1}^{N} x_i \tag{19}$$

Sample mean and distribution mean are not necessarly the same, could point it our by show an example on both (perhaps by using Franke function?). Mean value in one data set α:

$$\bar{\mu}_\alpha = \frac{1}{N} \sum_{k=1}^{N} X_{\alpha,k} \tag{20}$$

Total sample mean of M data sets:

$$\bar{\mu}_M = \frac{1}{M} \sum_{\alpha=1}^{M} \bar{\mu}_\alpha = \frac{1}{MN} \sum_{\alpha=1}^{M} \sum_{k=1}^{N} X_{\alpha,k} \tag{21}$$

NEED TO DECIDE WHETHER I USE MU OR X.

The total variance will then be a sum over the single set variances plus the cross terms:

$$\sigma_M^2 = \frac{\sigma^2}{N} + \frac{2}{MN^2} \sum_{\alpha=1}^{M} \sum_{k<l}^{N} (X_{\alpha,k} - \bar{\mu}_M)(X_{\alpha,l} - \bar{\mu}_M). \tag{22}$$

The last term is called the covariance, and is a measure on how much the data set are related. If they are totally independent, the covariance is zero. For large data sets this term is computational expensive to calculate, and we will rather use resampling techniques to estimate the variance. A few resampling techniques are presented in the Method section.

Cost function (loss function)

Different methods to estimate error:

- Absolute error
- Relative error
- Mean square error (MSE)
- R^2 score function

3 Methods

3.1 Resampling techniques

A resampling technique is.. There are plenty of resampling techniques, and we have already went through several of them in this course:

- Validation set approaches
- Leave one out validation
- Jackknife resampling
- K-fold validation
- Bootstrap method
- Blocking method.

For this particular project we have been focusing on the bootstrap and the k-fold validation methods, so here I will cover them only

3.1.1 Bootstrap method

3.1.2 K-fold validation method

3.2 Singular Value Decomposition (SVD)

3.3 Minimization methods

When the interaction term is excluded, we know which α that corresponds to the energy minimum, and it is in principle no need to try different α's. However, sometimes we have no idea where to search for the minimum point, and we need to try various α values to determine the lowest energy. If we do not know where to start searching, this can be a time consuming activity. Would it not be nice if the program could do this for us?

In fact there are multiple techniques for doing this, where the most complicated ones obviously also are the best. Anyway, in this project we will have good initial guesses, and are therefore not in need for the most fancy algorithms.

3.3.1 Gradient Descent

Perhaps the simplest and most intuitive method for finding the minimum is the gradient descent method (GD), which reads

$$\alpha^+ = \alpha - \eta \cdot \frac{d\langle E(\alpha)\rangle}{d\alpha}. \tag{23}$$

where α^+ is the updated α and η is a step size. The idea is that one finds the gradient of the energy with respect to a certain α, and moves in the direction which minimizes the energy. This is repeated until one has found an energy minimum, where the energy minimum is defined as either where $\frac{d\langle E(\alpha)\rangle}{d\alpha}$ is smaller than a given

tolerance, or the energy fluctuates around a value are smaller than a tolerance, and thus changes minimally.

To implement equation 23, we need an expression for the derivative of E with respect to alpha:

$$\bar{E_\alpha} = \frac{d\langle E(\alpha)\rangle}{d\alpha}. \tag{24}$$

By using the expression for the expectation value for the energy $\langle E(\alpha)\rangle$ in equation 25

$$\langle E(\alpha)\rangle = \frac{\langle \psi_T(\alpha)|H|\psi_T(\alpha)\rangle}{\langle \psi_T(\alpha)|\psi_T(\alpha)\rangle} \tag{25}$$

and applying the chain rule of differentiation, it can be shown that equation 24 is equal to equation 26

$$\bar{E_\alpha} = 2\left[\langle E_L(\alpha)\frac{\bar{\psi_\alpha}}{\psi_\alpha}\rangle - \langle E_L(\alpha)\rangle\langle\frac{\bar{\psi_\alpha}}{\psi_\alpha}\rangle\right] \tag{26}$$

where

$$\bar{\psi_\alpha} = \frac{d\psi(\alpha)}{d\alpha}. \tag{27}$$

The algorithm of this minimization method is thus as follows:

```
for(max number of iterations with minimizing)

        do M Monte Carlo cycles
        calculate E and dE/dalpha

        Check if dE/dalpha < eps or alpha fluctuation over the last 5 steps is <
            ↪ eps


                if yes, print optimal alpha and break loop
                if no, continue to next iteration
```

4 Code

I will briefly describe the code. Written in Python.

4.1 Code structure

To keep the code neat and clean we decided to write object oriented code, although we do not have that many functions. The entire code used in this project consists

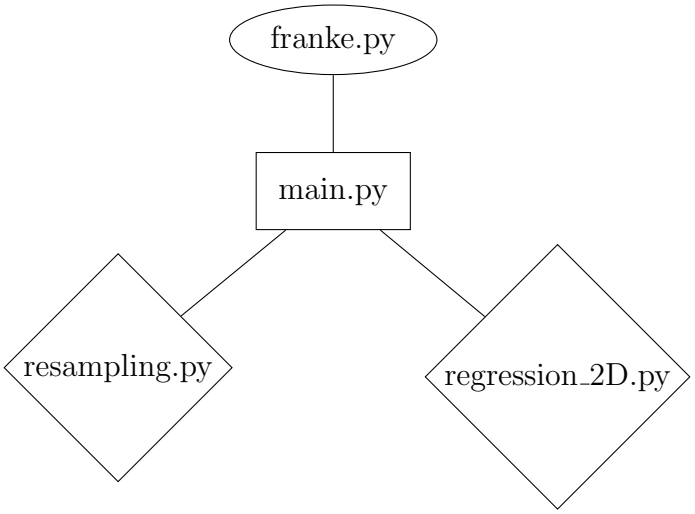


Figure 2: Code structure

of a main.py which call the other functions, a Franke function (franke.py), a class for 2D regression (regression_2D.py) and a class containing resampling techniques (resampling.py). See figure (2) for code structure.

4.2 Implementation and optimization

To get a code which provides good performance, we need to be thoughtful when implementing it. The loops are often bottlenecks are often, so by replacing loops with vector operations we can save a lot of time. An example on this, is when we calculate the mean square error,

$$\text{MSE}(\vec{y}) = \sum_i \left(y_i - \beta_0 - \sum_j X_{ij}\beta_j \right)^2, \tag{28}$$

where $\vec{y}$ and $\vec{\beta}$ are vectors and $\hat{X}$ is a matrix. For implementing this directly, we a double loops, which can be slow for large systems. A better solution would be to exploit the linear algebra properties of vectors:

$$\text{MSE}(\vec{y}) = (\vec{y} - \hat{X}^T\vec{\beta})^T \cdot (\vec{y} - \hat{X}^T\vec{\beta}) \tag{29}$$

which is usually much faster.

5 Results

5.1 Franke function

First we will take a look at how good our linear regression is...

5.1.1 Visualization of graphes

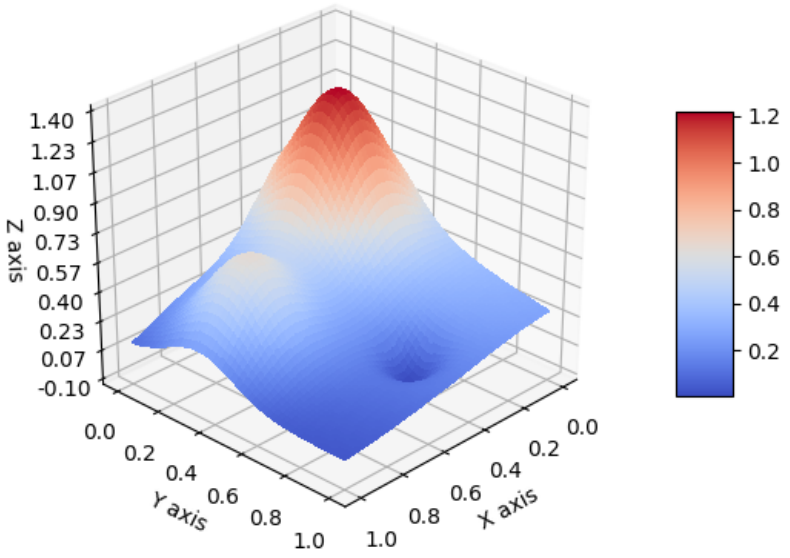
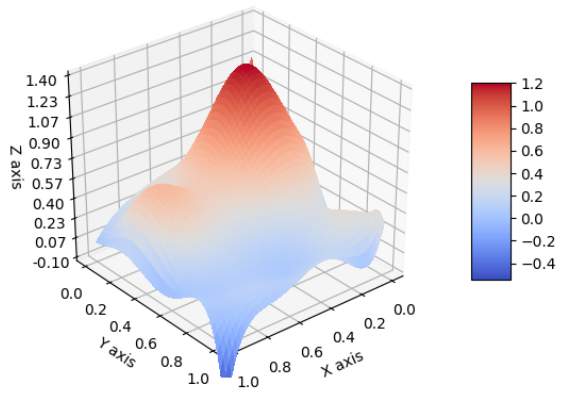
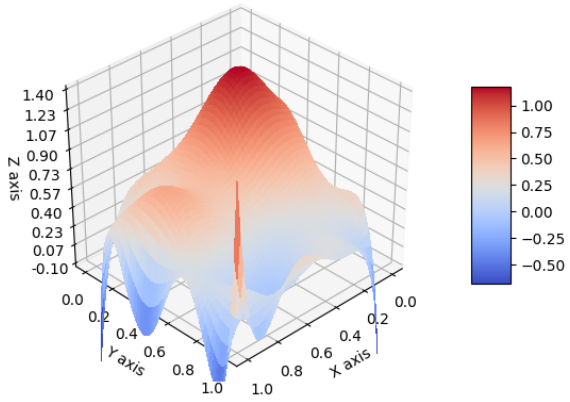


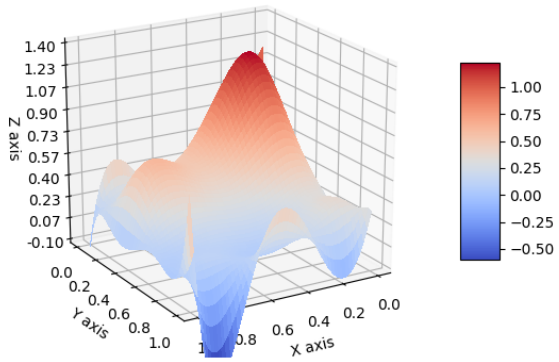
Figure 3: The Franke function in the interval $x \in [0, 1]$, $y \in [0, 1]$, $z \in [0, 1.2]$.



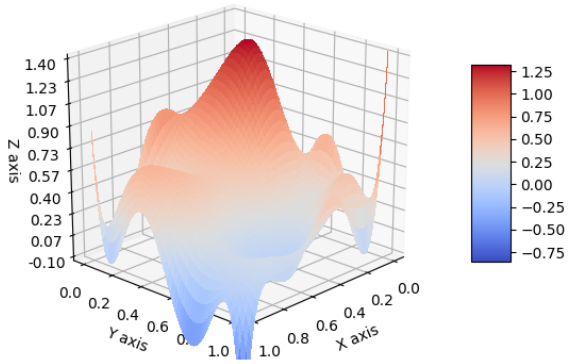
(a) OLS without noise



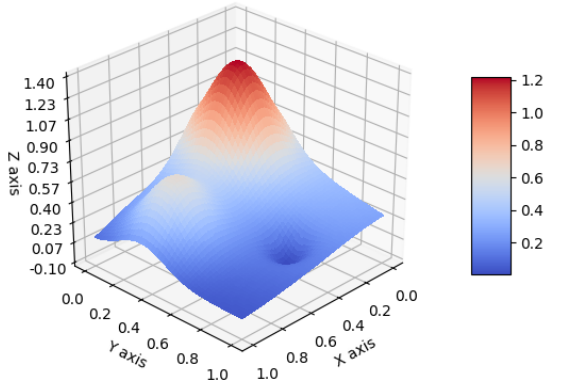
(b) OLS with noise



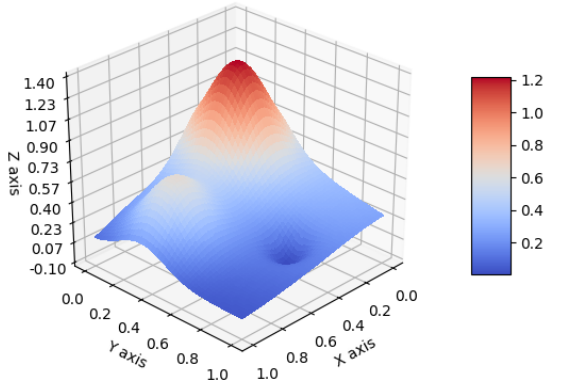
(c) Ridge without noise



(d) Ridge with noise



(e) Lasso without noise



(f) Lasso with noise

Figure 4: Fitted polynomial by OLS, Ridge and Lasso with and without noise. For these plots, we used a low penalty of $\lambda = 1e - 15$.