Regression analysis on two dimensional Franke's function

Aram Salihi[1], Martine Tan [2] & Andras Filip Plaian [2]
[1]Department of Informatics,University of Oslo, N-0316 Oslo, Norway
Department of Mathematics[2], University of Oslo, N-0316 Oslo, Norway

October 6, 2019

Abstract

In this numerical study we have used three different linear regression methods to study the polynomial fit of the two dimensional Franke's function. The standard ordinary least squared (OLS), Ridge and LASSO regression has been used. Further we have used the cross validation to further assess if the model is correct.

Contents

1 Introduction

In todays science and industrial societies the number of data created is constantly growing. This makes it harder to analyze data when using

2 Theory

2.1 Franke's function

Franke's function is a two dimensional Gaussian function with two peaks. This function is widely used as a test function in interpolation and fitting problems.

$$f(x,y) = \frac{3}{4}\exp\left\{-\frac{(9x-2)^2}{4} - \frac{(9y-2)^2}{4}\right\} + \frac{3}{4}\exp\left\{-\frac{(9x+1)^2}{49} - \frac{(9y+1)^2}{10}\right\} \quad (1)$$

$$+\frac{3}{4}\exp\left\{-\frac{(9x-7)^2}{4} - \frac{(9y-3)^2}{4}\right\} + \frac{3}{4}\exp\left\{-(9x-4)^2 - (9y-7)^2\right\} \quad (2)$$

2.2 Linear regression and matrix notation

Consider an experiment with a set of datapoints $\mathbf{z} = \{z_0...z_{n-1}\}$ observed, Further we want to model these observations as a function of $\mathbf{x} = \{x_0...x_{n-1}\}$ and $\mathbf{y} = \{y_0...y_{n-1}\}$. Keep in mind that, when modeling datapoints one has to be perpared that error can arise in our approximation. We will now assume that our response variable $z(x, y)$ is a smooth function and thus can be parameterized as a two dimensional polynomial of x and y

$$z_i = z(x_i, y_i) = \tilde{z}_i + \epsilon_i \tag{3}$$

Where $\tilde{z}$ is the estimator of our response variable z. Expanding this, we will obtain a set of equation

$$\begin{aligned}
&z_0 = \epsilon_0 + \beta_0 + \beta_1 x_0 + \beta_2 y_0 + \beta_3 x_0^2 + \beta_4 y_0 x_0 + \beta_5 y_0^2 + ... + \beta_j x_0^k y_0^{k-d} \quad . \\
&. \\
&. \\
&. \\
&z_{n-1} = \epsilon_{n-1} + \beta_0 + \beta_1 x_{n-1} + \beta_2 y_{n-1} + \beta_3 x_{n-1}^2 + \beta_4 y_{n-1} x_{n-1} + \beta_5 y_{n-1}^2 + ... + \beta_j x_{n-1}^k y_{n-1}^{k-d}
\end{aligned}$$

Where βj is the unknown weights we wish to find in order to fit the datapoints z. In this case there exist in total $\sum_{t=0}^{d-1}(i+1)$ of β's, and k goes from $k = d, ..., 0$, where d is order of the polynomial we want to use. Notice that this set of linear equation can be expressed in vector-matrix notation on the form:

$$\mathbf{y} = X\boldsymbol{\beta} + \epsilon \tag{4}$$

Where X is a design matrix with dimension $(n-1) \times d$, which looks like

$$X = \begin{pmatrix}
1 & x_0 & y_0 & x_0^2 & x_0 y_0 & y_0^2 & . & . & x_0^k y_0^{k-d} \\
. & . & . & . & . & . & . & . & . \\
. & . & . & . & . & . & . & . & . \\
. & . & . & . & . & . & . & . & . \\
. & . & . & . & . & . & . & . & . \\
. & . & . & . & . & . & . & . & . \\
. & . & . & . & . & . & . & . & . \\
. & . & . & . & . & . & . & . & . \\
. & . & . & . & . & . & . & . & . \\
1 & x_{n-1} & y_{n-1} & x_{n-1}^2 & x_{n-1} y_{n-1} & y_{n-1}^2 & . & . & x_{n-1}^k y_{n-1}^{k-d}
\end{pmatrix} \tag{5}$$

2.3 Ordinary least square (OLS)

Quite often we will stumble on problems where the number of unknowns β are greater than observables (response variable). Due to this problem we cannot invert the design matrix (not a square matrix) to solve the matrix equation, we will therefore consider a cost function $C(\beta)$ (for the sake of simplicity we will assume that response and estimator variable are one dimensional)

$$C(\beta) = \frac{1}{n}\sum_{i=0}^{n}(z - \tilde{z})^2 = \frac{1}{n}(\mathbf{y} - \tilde{\mathbf{y}})^T(\mathbf{y} - \tilde{\mathbf{y}}) \tag{6}$$

where the estimator variable is $\tilde{\mathbf{z}} = X\boldsymbol{\beta}$. We now want to find $\boldsymbol{\beta}$ which minimizes the cost function. In order to this consider the following

$$\min_{\boldsymbol{\beta}\in\mathbb{R}^d} C(\boldsymbol{\beta}) = \min_{\boldsymbol{\beta}\in\mathbb{R}^d}\left\{\frac{1}{n}(\mathbf{y} - \tilde{\mathbf{y}})^T(\mathbf{y} - \tilde{\mathbf{y}})\right\} = 0 \tag{7}$$

For the sake of simplicity we can also express this as

$$\frac{\partial C(\beta)}{\partial \beta_j} = \frac{\partial}{\partial \beta_j}\left[\frac{1}{n}\sum_{i=0}^{n-1}\left(y_i - \beta_0 x_{i,0} - \beta_1 x_{i,1} - ..... - \beta_{n-1}x_{i,n-1}\right)^2\right] = 0 \tag{8}$$

Performing this partial derivative we will obtain the following

$$\frac{\partial C(\beta)}{\partial \beta_j} = -\frac{2}{n}\left[\frac{1}{n}\sum_{i=0}^{n-1} x_{ij}\left(y_i - \beta_0 x_{i,0} - \beta_1 x_{i,1} - ..... - \beta_{n-1}x_{i,n-1}\right)\right] = 0 \tag{9}$$

This can be beutifully expressed in matrix notation as

$$X^T\left(\mathbf{y} - X^T X\boldsymbol{\beta}\right) = 0 \tag{10}$$

Notice that that this equation can be simply be written as:

$$X^T X\boldsymbol{\beta} = X^T\mathbf{y} \tag{11}$$

This is the so called normal equation. Now notice that $X^T X$ is a square matrix. If this square matrix is a non-singular matrix, if and only if all the coloumn vectors are linearly independent there will exist a invertible matrix such that:

$$\boldsymbol{\beta} = (X^T X)^{-1} X^T\mathbf{y} \tag{12}$$

Quite often inverting a matrix is quite tedious and require alot more operations, but it do exist algorithm which solves this equation more efficient, such as SVD (singular value decompistion), Cholesky factorization or QR method. In our case we have used numpy's PINV function to invert this matrix, but this function uses SVD as algorithm. The beauty of this numpy function is that it handles problems with singularities perfectly. Consider a case where $X^T X$ contains singular values, in order to remove this values we will add some small value λ

$$X^T X \approx X^T X + \lambda\mathbb{I} \tag{13}$$

The numpy function will already take care of this by doing this. Keep in mind that OLS can be generalized to higher dimensions.

2.4 Ridge regression

.In comparison with the standard OLS method the Ridge regression will actually take care of the singularities if $X^T X$ is singular. Recall the cost function from the previous section (6)

$$C(\boldsymbol{\beta}) = \frac{1}{n}\left(\mathbf{y} - X\boldsymbol{\beta}\right)^T\left(\mathbf{y} - X\boldsymbol{\beta}\right) \tag{14}$$

Which is the MSE for the OLS method. From previous section we mentioned that that the matrix product $X^T X$ could be a singular matrix, thus not invertible. We therefore wish to add a regularization parameter λ by penalizing the cost function.

$$C(\boldsymbol{\beta}) = \frac{1}{n}\left(\mathbf{y} - X\boldsymbol{\beta}\right)^T\left(\mathbf{y} - X\boldsymbol{\beta}\right) + \lambda\boldsymbol{\beta}^T\boldsymbol{\beta} \tag{15}$$

We now wish to shrink the distance between the response and estimator thus finding a $\boldsymbol{\beta}$ which satisifies this condition. Consider the following

$$\min_{\boldsymbol{\beta}\in\mathbb{R}^d} C(\boldsymbol{\beta}) = \min_{\boldsymbol{\beta}\in\mathbb{R}^d}\frac{1}{n}||\mathbf{y} - X\boldsymbol{\beta}||_2^2 + \lambda||\beta||_2^2 \tag{16}$$

Solving this equation we can express the expression above as

$$X^T y = \beta^{ridge}(X^T X + n\lambda\mathbb{I}) \tag{17}$$

We will further define $n\lambda = \lambda$. Recall from previous that the regularization paramter took care of the singularities, thus $(X^T X + \lambda\mathbb{I})$ should be invertible since the new matrix should contain $n-1$ linearly independent coloumn vectors. This can be proven by using SVD decompistion on this matrix, a small proof is this is given in the appendix. Thus, the solution to $\boldsymbol{\beta}$

$$\beta^{ridge} = (X^T X + \lambda\mathbb{I})^{-1} X^T y \tag{18}$$

This is the famous ridge regression method. As previously mentioned, the invertion of matrix is quite tedious, and thus SVD decompistion is the most desired method for solving this type of equation. Further keep in mind, if the matrix $X^T X$ is non-singular (thus orthogonal), what ridge regression does is shriking ***skrivmer!***

2.5 LASSO regression

The idea behind LASSO is to shrink the data towards a central point. Thus shriking the β to a point where some of the coefficients are zero. What this does compared to ridge regression is that, not only does it reduce overfitting but also generating a more simpler model, due to fewer coefficients in the polynomial fit.

Mathematically, this is done by adding a λ term to minimized cost function. Instead of taking the l2 norm of $\boldsymbol{\beta}$ as in ridge regression (the normal euclidian norm), we are doing a l1 norm (taking the absolute value) on $\boldsymbol{\beta}$. Thus we want to minimize the following

$$\min_{\boldsymbol{\beta}\in\mathbb{R}^d} C(\boldsymbol{\beta}) = \min_{\boldsymbol{\beta}\in\mathbb{R}^d} \frac{1}{n}||\mathbf{y} - X\boldsymbol{\beta}||_2^2 + \lambda||\beta||_1 \tag{19}$$

Where the l1 norm is $||\beta||_1 = \sum_{i=0}^{n-1} |\beta_i|$. We now perfomed an absolute shrinkage to the coefficients we wish to find. ***skrivmer!***

3 Method/Implementation

In this following sections we will describes the implementation of the different linear regression used and the creation of the design matrix. Keep in mind that these are implementation in python3, used different libraries such as numpy, scipy and scikitlearn.

3.1 Design matrix

The dimensionality of the design matrix X is dependent on the polynomial order and number of datapoints from Franke's function.

3.2 OLS

consider