# FYS-STK4155 – Applied data analysis and machine learning

**Project 1**

|  |  |
| --- | --- |
| **Name of author：** | **Huiying Zhang Laurent Fontaine** |
| **Topic ：** | **Project 1** |
| **Course ：** | **Applied data and Machine learning** |
| **Name of supervisor：** | **Morten** |
| **Submission date：** | **2019 年 10月07日** |

**Abstract**

Linear algebra can go a long way in finding patterns in data and is the basis of the more elaborate methods used in machine learning. Our study focused on the use of ordinary least squares (OLS), ridge and lasso regression to attempt to find the best fitting models to two sets of data, one generated from the Franke function, while the other is a real geographic dataset. Resampling techniques were used in combination with the abovementioned regression approaches. Our results suggest ridge regression produced the lowest mean squared error estimates while either bootstrapping and k-fold cross-validation yielded equivalent errors. Lasso regression produced slightly worse error estimates while OLS produced the highest error values. The effectiveness of the three methods were reflected in both datasets.

**Key words:**  Ordinary Least Squares (OLS) method; Ridge regression; Lasso regression; resampling techniques; Franke’s function; Bootstrap; k-fold cross-validation

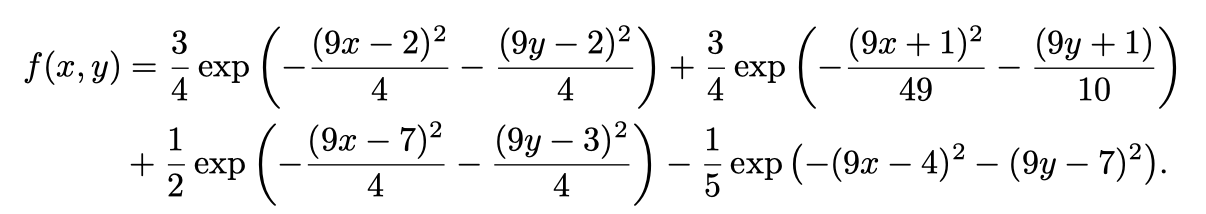
**1 1 Introduction**

* 1. **Problem-Regression analysis and resampling methods**

In this project, we will have two main aims as follows,

1. Evaluate Ordinary Least Squares (OLS), Ridge and Lasso regression techniques with or without resampling to perform regression on data we have generated ourselves.
2. Test on real data the regression libraries developed on the self-generated datasets and find the most accurate way to model the patterns in the form of an analytical function with the lowest mean square error as well as minimal over or underfitting.

In previous work, we have fit polynomials to one-dimensional functions. We now combine OLS, ridge and Lasso regression methods with resampling to ﬁt polynomials to Franke’s function, a speciﬁc two-dimensional function, which has been widely used when testing various interpolation and ﬁtting algorithms. Moreover, after having established the model and the method, the resampling techniques used are K-fold cross-validation and bootstraps in order to perform a proper assessment of our models. This study also focused on the so-called Bias-Variance trade oﬀ.

The Franke function is as follows，

**1.2 Problem restatement**

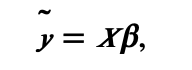
**Ordinary Least Squares on the Franke function with resampling.**

First of all, we have generated our own dataset for a function Franke Function f(x, y) with x, y ∈ [0, 1] and added a stochastic noise function N(x,y) to get a sum function as follows

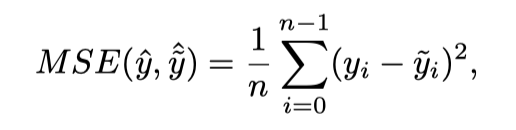
F(x,y) = f(x, y) + N(x,y)

Moreover, we use both matrix inversion (for small matrices) and singular value decomposition (SVD) for larger matrices using the numpy and scikit-learn libraries. A first analysis is performed with standard least square regression with ﬁfth degree polynomials in x and y. We have generated the corresponding design matrix ourselves.

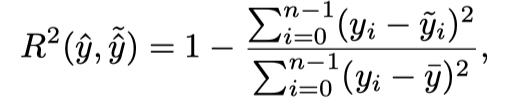
Finally, their variances are computed to find the conﬁdence intervals of the parameters β and get the predicted fitting function as follows



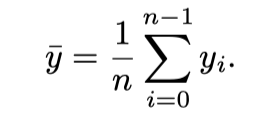
Then, evaluate the Mean Squared error (MSE)



and the score function.



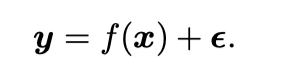
where the mean value of is defined as



**Resampling techniques and bias-variance tradeoﬀ**

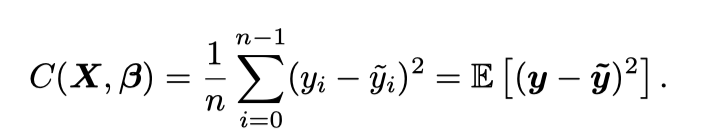
In order to be able to examine the bias variance tradeoff, it is necessary to use resampling techniques. To achieve this, the data is split in training and test sets by using the function for splitting training data provided by Scikit-Learn -- train\_test\_split. The OLS method is then used in combination as previously with the method of the k-fold cross-validation algorithm to fit the training data. This OLS with resampling allows to evaluate the MSE, bias and variance of the test and train data. It is then relevant to try to discern the order of the polynomial where MSE, bias and variance are minimal so as to have the best fitting function.

Assume that the true data is generated from a noisy model

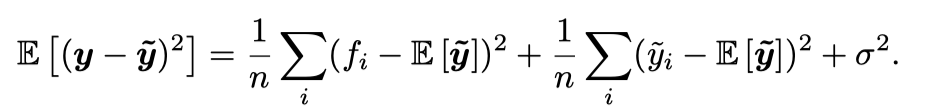


Here is normally distributed with mean zero and standard deviation . In our derivation of the ordinary least squares method, we deﬁned an approximation to the function f in terms of the parameters β and the design matrix X which embody our model, that is ˜y = Xβ.

The parameters β are in turn found by optimizing the mean squared error via the so-called cost function

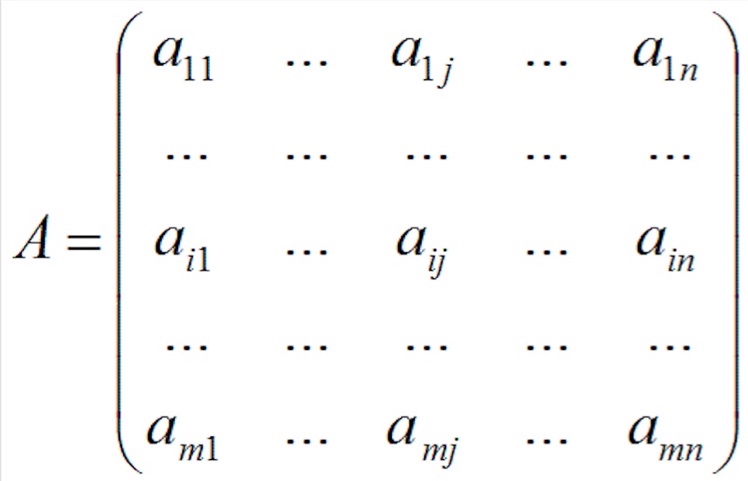


Which can be rewritten as



1. **Method**
   1. **Singular Value Decomposition (SVD)**

SVD, Singular value decomposition, is an important form of decomposition in linear algebra due to allowing to work around linear dependence. For example, in natural language processing, SVD can be used to classify news, and good results have been achieved. The core words in the news are expressed by a vector. Each news is expressed by a vector to form a matrix, which is decomposed by SVD. A large matrix A can be used to describe the correlation between these one million articles and half A million words. In this matrix, each row corresponds to an article, and each column corresponds to a word.



In the figure above, M=1,000,000, N=500,000. The element in row I, column j, is the weighted word frequency that the JTH word in the dictionary appears in article I (for example, TF/IDF). The reader may have noticed that this matrix is very large, with one million times half a million, which is 500 billion elements.

Singular value decomposition is the decomposition of a large matrix such as the one above into the multiplication of three small matrices, as shown in the figure below. The matrix in the example above can be decomposed into a million times a hundred matrix X, a hundred times a hundred matrix B, and a hundred times a hundred thousand matrix Y. The total number of elements in these three matrices is only 150 million, which is only one third of the original number. The corresponding amount of storage and calculation will be three orders of magnitude smaller.

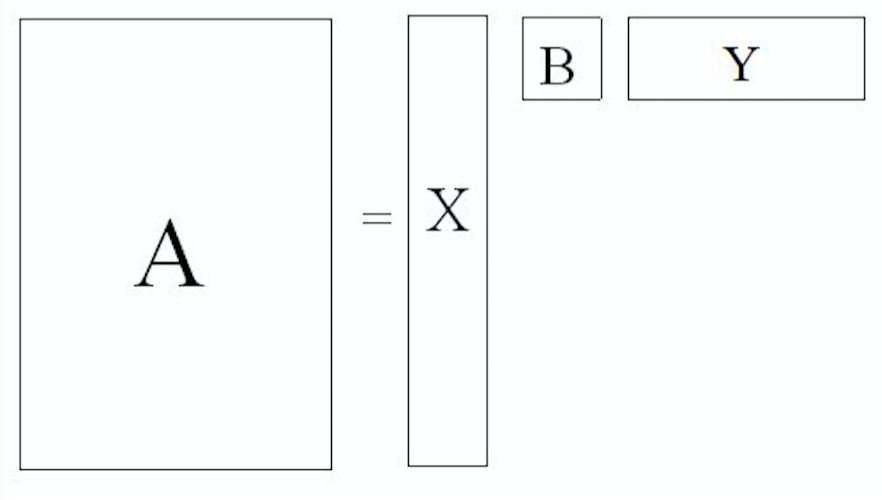


Fig. 2.1-1 The corresponding amount of storage and calculation A will be three orders of magnitude smaller (X, B, Y).

(Figure from <https://zhuanlan.zhihu.com/p/25801478>)

The three matrices have very clear physical implications:

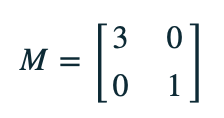
Each row in the first matrix X represents a class of words that are related in meaning, and each non-zero element represents the importance (or relevance) of each word in that class, the larger the number, the more relevant.

The second matrix B represents the correlation between the class word and the article. Therefore, it allows the classification of synonyms and articles at the same time by singular value decomposition of the incidence matrix A, beside the relevance of each type of article and each type of word.

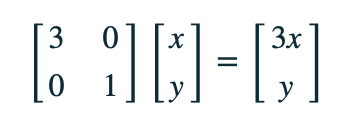
Each column in the third matrix Y represents an article of the same topic, with each element representing the correlation of each article in the article.

* + 1. **Eigenvalue decomposition**

It is of use to consider examples of linear transformation before moving to SVD. Taking a relatively special diagonal 2 X 2 matrix as example:

****

M is the transformation matrix of a point (x,y) on a two-dimensional plane through linear transformation to another point, as shown below,



The effect of the transformation is shown in the figure below. After the transformation, it is stretched 3 times along the X level without any change in the vertical direction.

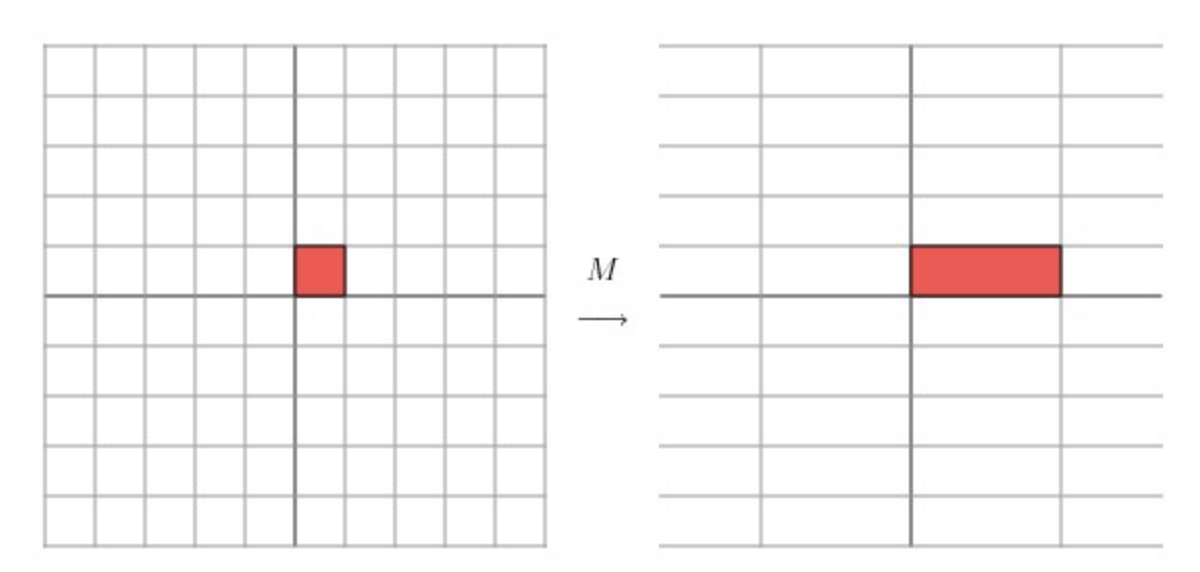
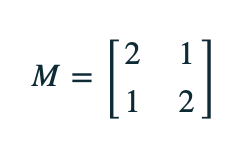


Fig. 2.1.1-1 The effect of the transformation of M

Now another matrix

****

The transformation effect generated by this matrix is shown in the following figure:

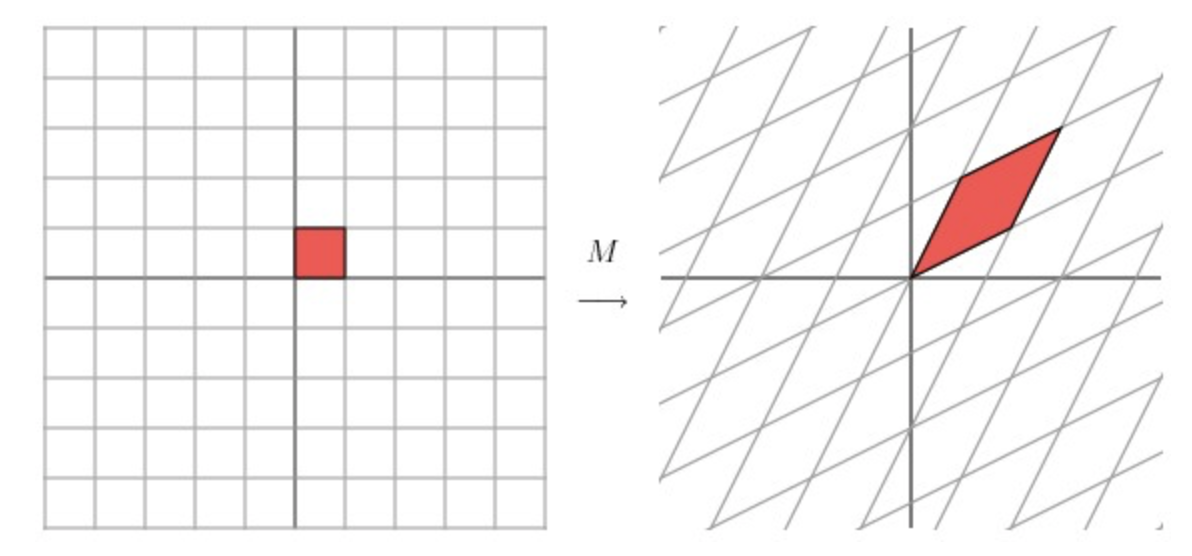


Fig. 2.1.1-2 The effect of the transformation of M

Identifying the transformation may be challenging as regards the angle of rotation or multiple of stretch, for example. Supposing the left plane is rotated by an angle of 45 degrees, and then perform the linear transformation of matrix M, the result is as follows:

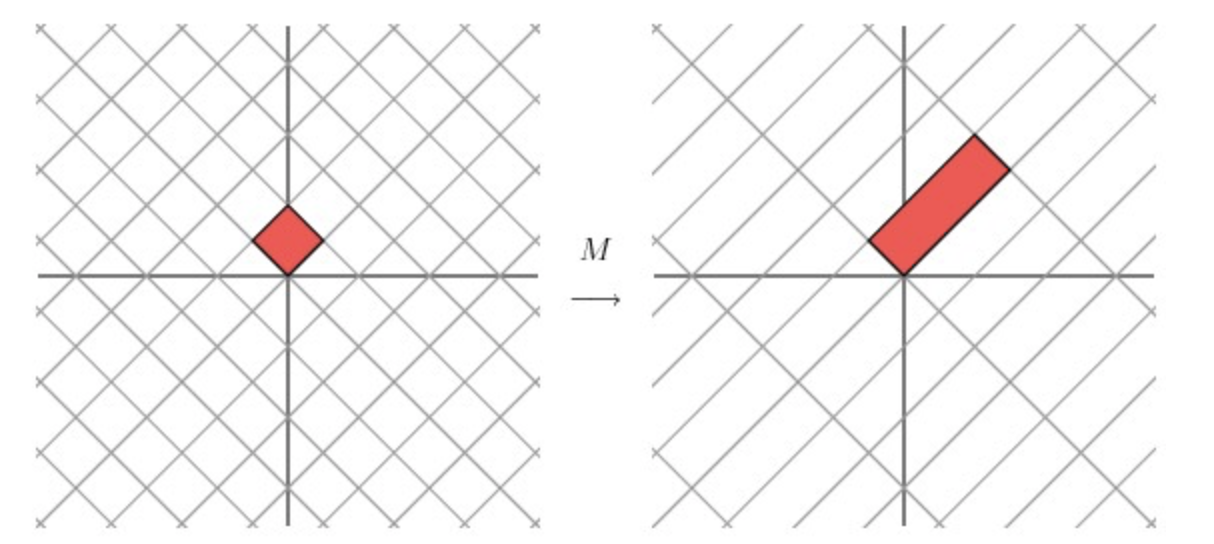
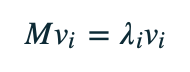


Fig. 2.1.1-3 Transformation of M and rotation of the left plane by an angle of 45 degrees.

After the transformation of M linear, the latter M matrix has the same effect as the diagonal matrix M above, which is to stretch the grid by 3 times in one direction. M here is a special case, because it's symmetric. What is not special is the relatively high occurrence of non-symmetric, non-square matrices in practical applications. As shown in the figure above, for a symmetric matrix M of 2 X 2, the grid is rotated by plane by a certain angle, and the transformation effect of M is to stretch the transformation on two dimensions.

To express it mathematically, given a symmetric matrix M, an orthogonal vector Vi, satisfying MVi, is the stretching transformation along Vi direction, and the equation is as follows:

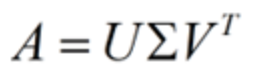


λi is the scale. M stretched the vector Vi. Vi is the eigenvector of the matrix M, or λi is the eigenvalue of the matrix M. The eigenvectors of the symmetric matrix M have an important property in that they are orthogonal to each other.

Eigenvalue decomposition is a useful matrix feature extraction method, but it is only for square, as, in the real world, most matrices are not square.

* + 1. **Singular Value Decomposition (SVD)**

Thus, if we want to describe the important features of such common matrix, Singular value decomposition is the best choice. Singular value decomposition is a decomposition method that can be applied to any matrix



Assume that A is a N X M matrix, then U get A phalanx of N X N (the inside of the vector are orthogonal, the inside of the U vector are called the left singular vectors), Σ is an N X M matrix (in addition to the diagonal elements are 0, on the diagonal elements called singular value), V 'transpose (V) is A matrix of N \* N, the inside of the vector are orthogonal, the inside of the V vector called right singular vectors), from the pictures to reflect the size of A few multiplication of matrix can get the image below:

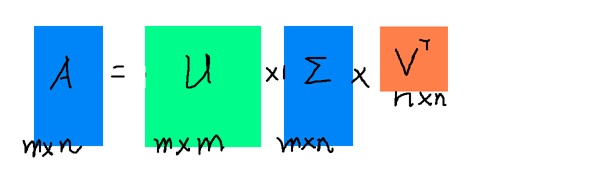


Fig. 2.1.2-1 figure of how the SVD formula consist of

Each row in the first matrix X represents a class of words that are related in meaning, and each non-zero element represents the importance (or relevance) of each word in that class, the larger the number, the more relevant. Each column in the final matrix Y represents a category of articles on the same topic, with each element representing the correlation of each article in that category. The intermediate matrix represents the correlation between the class word and the article ray. Therefore, we only need to perform singular value decomposition on the incidence matrix A, and w can complete the classification of synonyms and articles at the same time. (also get the relevance of each type of article and each type of word).

* 1. **Ordinary Least Squares**

In statistics, ordinary least squares (OLS) is a linear least squares method used to estimate unknown parameters in linear regression models. OLS uses the least square principle to select the parameters of a linear function of a set of explanatory variables: minimize the sum of squares of the residuals observed between the dependent variable (the value we want to predict) and the predictive variable in a given dataset.

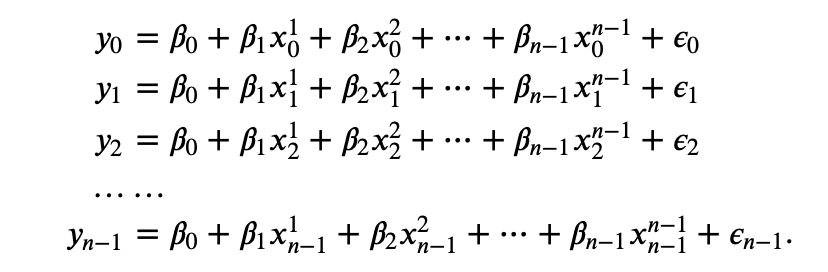
Regression modeling is often used to deal with the description of the sampling distribution of a given random variable y, which is called the dependent, the outcome or the response variable and how it varies as function of another variable or a set of such variables 𝑥=[,,…,, which is called the independent variable, predictor variable or explanatory variable.

A regression model aims at finding a likelihood function (𝑦|𝑥) that is the conditional distribution for 𝑦 with a given 𝑥. The estimation of (𝑦|𝑥) is made using a data set with

* 𝑛 cases 𝑖=0,1,2,…,𝑛−1
* Response (target, dependent or outcome) variable with 𝑖=0,1,2,…,𝑛−1
* p so-called explanatory (independent or predictor) variables =[,,…,] with 𝑖=0,1,2,…,𝑛−1and explanatory variables running from 00 to 𝑝−1p−1. See below for more explicit examples.

The objective of the analysis is to get the best fit relationship function between  and .

For the linear relationship between and , we have the set of equations as follows,



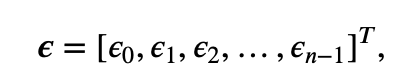
Defining the vectors (Response (target, dependent or outcome) variable)



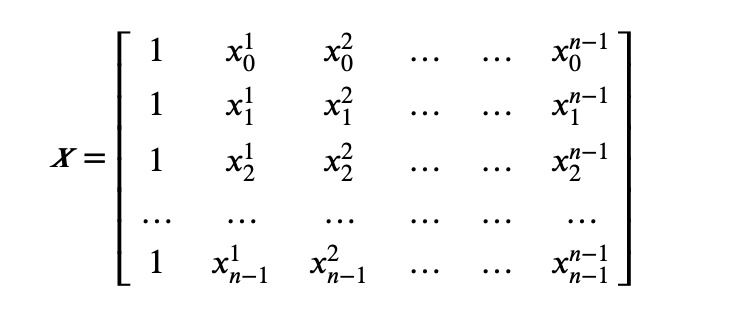
And the vector of parameters beta



The error vector



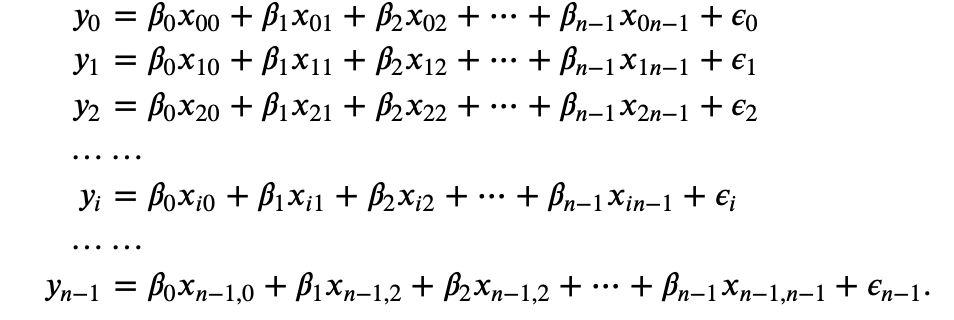
and the design matrix



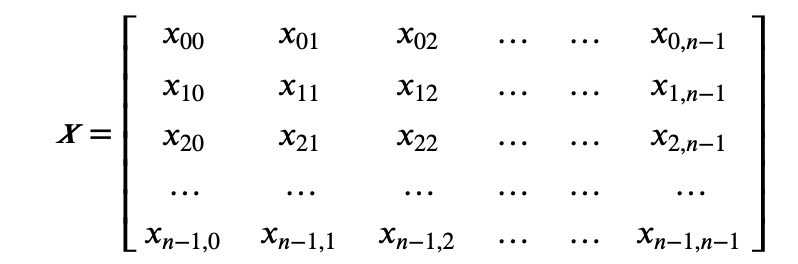
we can rewrite our equations as



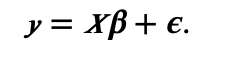
And then generalize the equations to



Redefine in turn the matrix X as

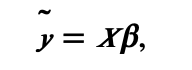


and without loss of generality we rewrite again our equations as

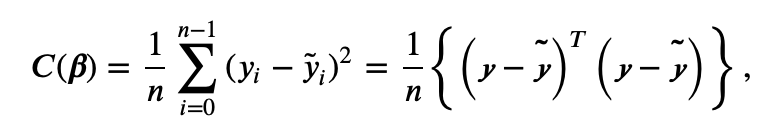


The vector is known while the error vector ϵ and the parameter vector β are unknow. Now we replace the ϵ to

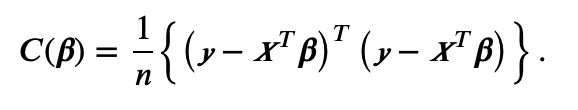
With the above we use the design matrix to define the approximation 𝑦̃ via the unknown quantity 𝜷 as



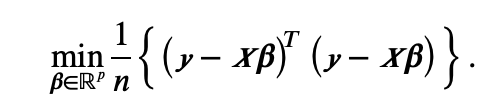
And now we want to find an optimal parameters  without solving the above linear algebra problem, so we define a so-called cost function to calculate the difference between the values  (real data) and the parameterized values 𝑦̃𝑖 (predicted data)



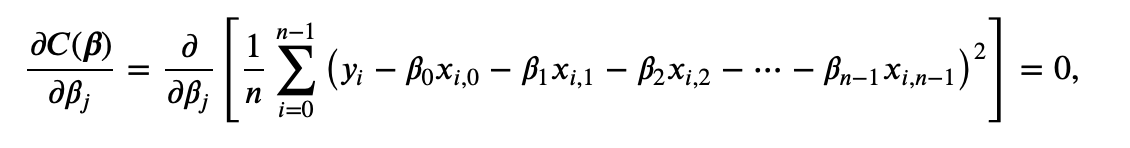
or using the matrix X and in a more compact matrix-vector notation as



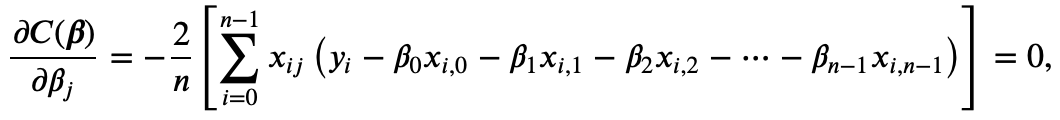
And then, what we need to do now is to find out the parameters which could get the minimum difference of (𝜷)),



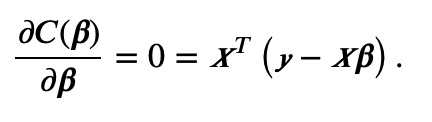
That means we will require



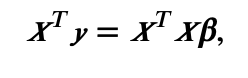
which results in



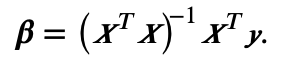
or in a matrix-vector form as



Combine the above equation we could have



and if the matrix  is invertible we have the solution



* 1. **Resampling techniques**

Resampling methods are widely used when the training data set and the test data set are not large enough. They involve repeatedly drawing samples from a training set and reﬁtting a model of interest on each sample in order to obtain additional information about the ﬁtted model.

Two resampling methods are often used in Machine Learning analyses,

1. The bootstrap method
2. and Cross-Validation
   * 1. **Bootstrap method**

The Bootstrap method is a very useful statistical estimation method. Bootstrap is a class of non-parametric Monte Carlo method. Its essence is to re-sample the observation information and then make statistical inference on the overall distribution characteristics. This method makes full use of the given observation information, does not need other assumptions of the model and adds new observations, and has the characteristics of robustness and high efficiency. Bootstrap has many advantages. First, Bootstrap avoids the problem of sample reduction caused by cross-validation by re-sampling. Second, Bootstrap can also be used to create randomness in data.

The Bootstrap process:

1. In the original sample, a certain number of new samples (for example, 100) will be sampled by re-sampling. Re-sampling means that there is a return sampling, that is, a data can be sampled repeatedly for more than one time.

2. Calculate the statistics we need to estimate based on the generated new samples.

3. Repeat the above steps n times (usually n>1000 times)

4. Finally, we can calculate the mean and variance of the estimator (without paying attention to the final specific value, which is related to the original sample):

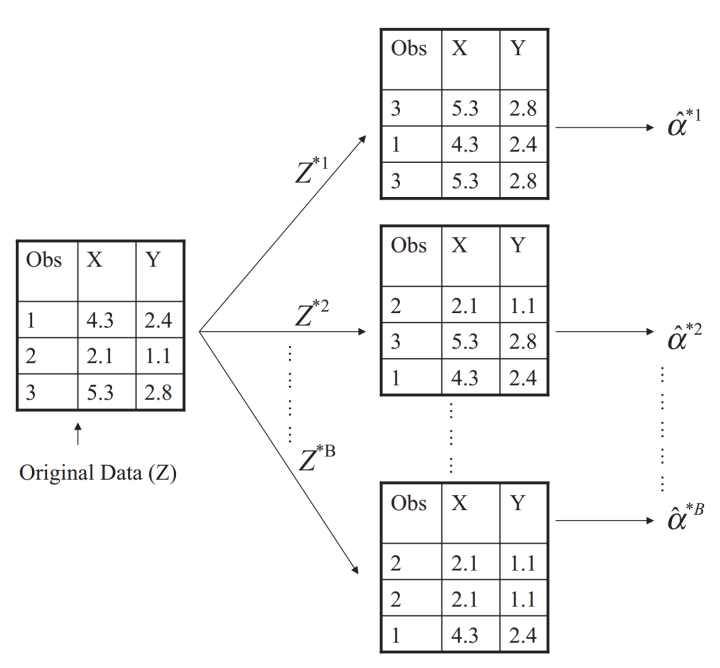


Fig. 2.3.1-1Figure of the Bootstrap process

Figure from 《An Introduction to Statistical Learning with Applications in R》

In essence, Bootstrap method is to repeat the estimation process thousands of times for tens of thousands of times, so as to obtain thousands or even tens of thousands of estimates. Therefore, by using more than one estimate, we can estimate other statistics other than a mean value, such as standard deviation and median.

* + 1. **K-fold Cross-Validation**

Generally, the labeled dataset used for both training and validation is not large enough which means that the model and parameter we try to fit the real data will depend heavily on how we divide our training and test sets. As shown in the figure, on the right is the test MSE from ten different training sets and test sets (using ten different partition methods). It is intuitive that under different partitioning methods, the difference of test MSE is very large, and the corresponding optimal degree is also different. So, if our training set and test set partition method is not good enough, it is likely that we cannot choose the best model and parameter.

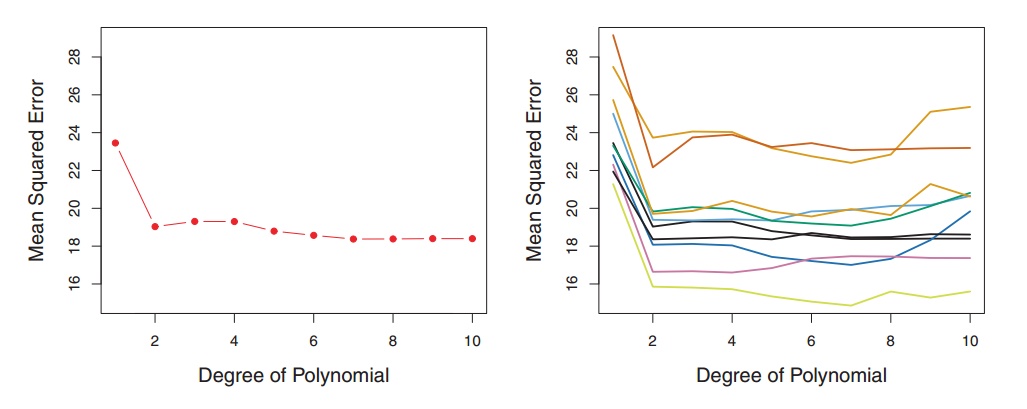


Fig. 2.3.2-1 Figure (left) of the test MSE from one training sets and test sets

Figure (right) of the test MSE from ten different training sets and test sets (using ten different partition methods).

Figure taken from《An Introduction to Statistical Learning with Applications in R》

The solution is to divide the original labeled dataset into K datasets.

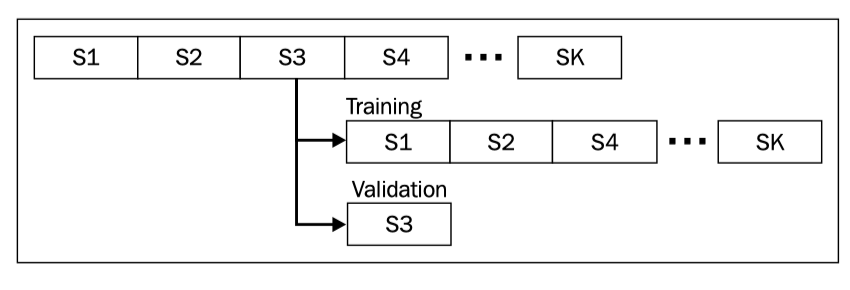
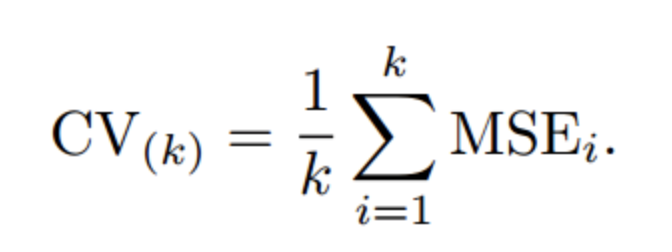


Fig. 2.3.2-2 Figure of the k-fold theory (how it divides into k groups and how it works)

Figure derive from《Scala for Machine Learning》

And then we take one dataset at a time as the only element of the test set, and the other k-1 datasets as the training set for the training model and the parameters. The result is that we end up training k models and getting an MSE each time. Finally, we calculate the average of all these MSE as follows



That is the K-fold cross validation method.

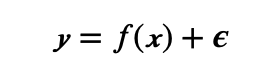
* 1. **Bias-variance tradeoﬀ.**

Generally, excluding human error, people encounter three sources of error: random error, deviation and variance.

The first is the random error. The random error is caused by the noise of the data itself, which is inevitable. It is generally believed that the random error obeys gaussian distribution. Consider a dataset   consisting of the

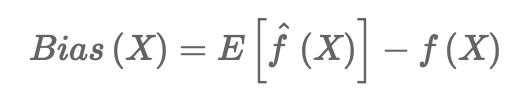


We assume that the true data is f, then we have

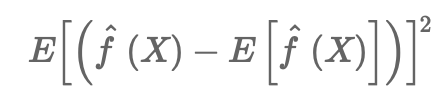


 ϵ is the noise which distributed with mean zero and standard deviation .

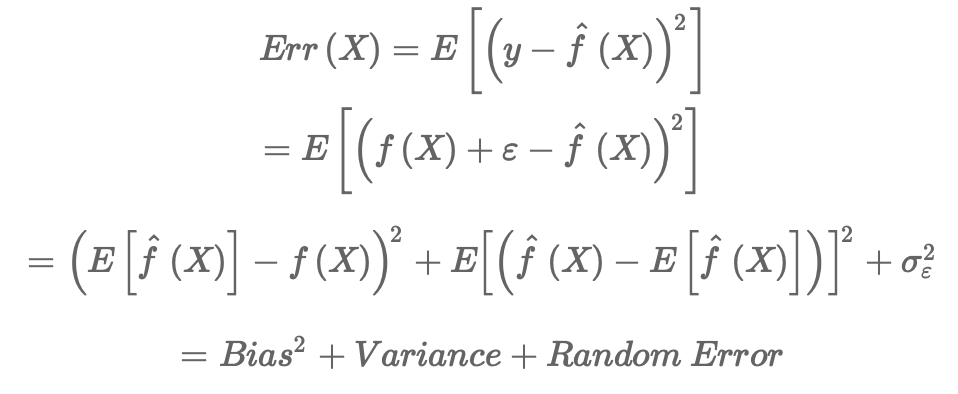
Bias describes the difference between the expectation of the result fitted by learning and the actual result



Variance describes the instability of the result obtained by learning to fit,



Taking mean square error as an example, the following conclusions can be drawn:



Suppose we now have a set of training data and need to train a model (gradient-based learning). At the beginning of training, bias was large because our model had not yet had time to start learning, which means it was far from the "real model". However, variance is small at this time because training data has not yet had an impact on the model, so applying the model to "different" training data sets will not make a big difference.

But as the training went on, Bias got smaller because our models got smarter, learned more about the "real model" and got closer to the real value. But if we practice for too long, variance can become very large, because we learn not only about real models, but also about specific training sets (subsets of real data) that we use. However, some features and noises between different possible training data sets (subsets of real data) are inconsistent, which results in that we cannot achieve good results on many other data sets, which is also known as overfitting.

Considering that the model error is the sum of deviation and variance, we can draw such a picture.

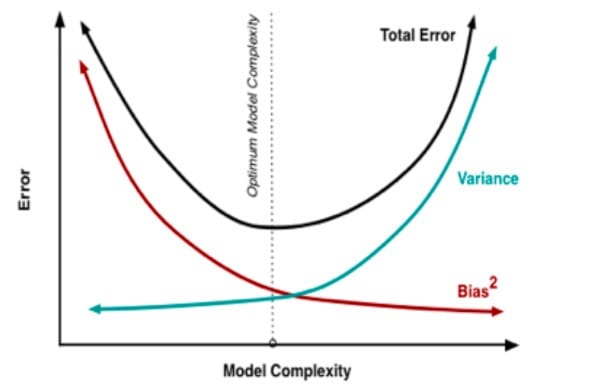
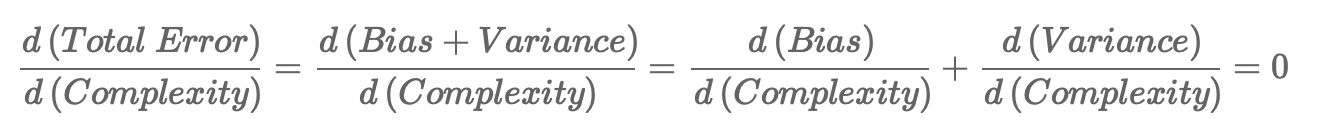


Fig. 2.4-1 Figure of the Variance, Bias, and Total error varies with the model complexcity

The optimal position in the figure is actually the inflection point of the Total Error curve. We know that the inflection point of a continuous function means that the value of the first derivative is 0.



This formula gives a mathematical description of finding the optimal equilibrium point. If the complexity of the model is less than the equilibrium point, the deviation of the model will be high, and the model tends to underfit. If the complexity of the model is greater than the equilibrium point, the variance of the model will be too high, and the model tends to overfit.

Despite the above mathematical statement, in the real world, it is sometimes difficult to calculate the deviation and variance of the model. Therefore, we need to judge the fitting state of the model through external performance: underfitting or overfitting.

Similarly, in the limited training data set, increasing the complexity of the model means that the model will reduce the errors in the training set as much as possible. Therefore, if the complexity of the model is increased continuously in the training set, the error in the training set will always decrease.

We divide the data into three parts: training data set, validation data set and test data set.

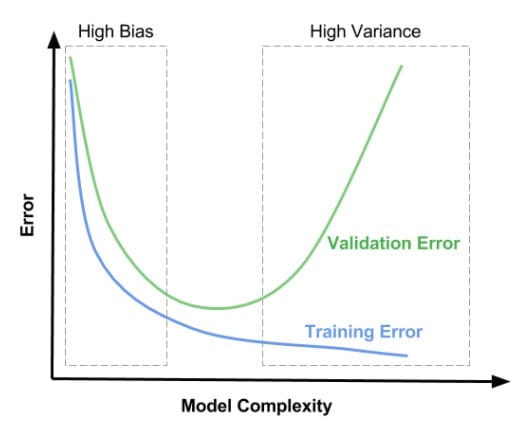


Fig. 2.4-2 Figure of the training data error and validation data error varies with the model complexity

In the left area of the figure above, the error between the training set and the verification set is very high, and the deviation in this area is relatively high. In the right region, the error on the validation set is very high, but the deviation on the training set is very low, and the variance of this region is relatively high. We want to get an optimal equilibrium point in the middle region.

Therefore, the high deviation (underfitting) has the following two characteristics:

1) high error in training set

2) the error of verification set is about the same as that of training set

High variance (overfitting)

1) the training set error is low

2) very high verification set error

* 1. **Ridge Regression**

When using OLS method to calculate the parameters of the linear regression model, if the design matrix X presents multicollinearity, then OLS method is very sensitive to the noise in the input variables, and its solution will be extremely unstable. A different approach is required to solve this issue.

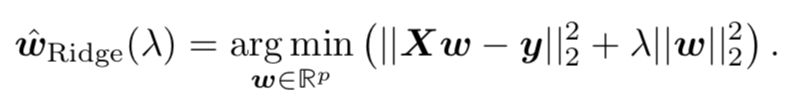
When the design matrix X is multicollinear, least squares method on the numerical values of parameters w is very large, and the general linear regression model is y = x, obviously, because w on the numerical value is very big, so, if there is a slight change in the input variable X, the reaction on the output will also become very big; this is the reason why it is sensitive to the overall noise input variables. If we can limit the growth of the parameter w, so that w does not become too large, then the model will be very sensitive to the input w. The sensitivity of the noise in the w will be reduced. That is the key point of Ridge Regression and Lasso Regression.

To limit the model parameter w value, add a penalty term on the original objective function of the model, this process is called Regularization:

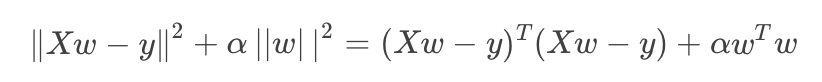
1.If the penalty term is of the parameter, the norm is the Ridge Regression.

2.If the penalty term is of the parameter, the norm is Lasso Regression.

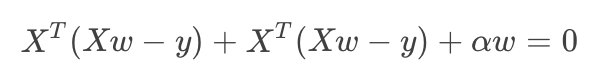
Add to the original cost function. The cost function of the penalty term of norm is as follows:

****

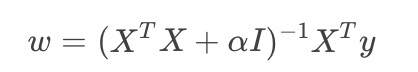
The larger the λ value of, the more obvious the role of the regular term and the punishment term; The smaller the λ value of, the less useful the regular term. In extreme cases, when λ =0, it's the same as the original cost function, if λ is equal to infinity, the cost function only has the regular term, and the result of its minimization must be w=0. We can rewrite it as



And then the derivative with respect to w

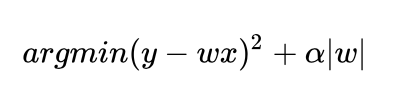


Solve this, we get the solution as follows,



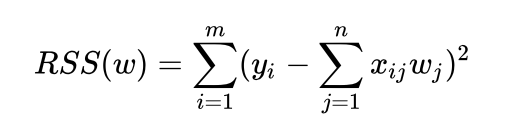
* 1. **Lasso Regression**

As mentioned previously, Lasso Regression is the same aim as the Ridge regression. The most obviously difference between these two is that the penalty is different. If the penalty term is of the parameter, the norm is Lasso Regression.

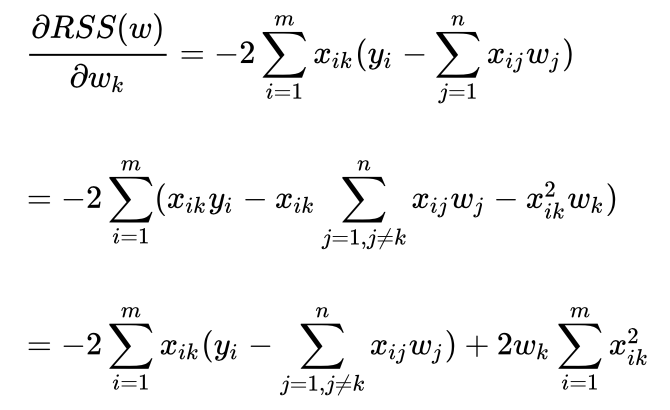


Although the penalty function is only slightly changed, compared with ridge regression, which can be directly obtained by matrix operation, the calculation of LASSO becomes relatively complicated. Since the penalty term contains absolute value, the derivative of this function is continuous and not smooth, so it is impossible to differentiate and use gradient descent optimization. Now, we are going to solve the two parts of LASSO cost function respectively:

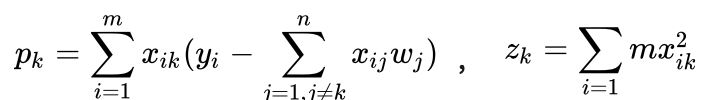
1) RSS



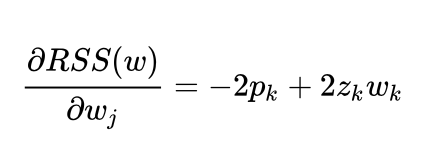
Then the derivation as follows,



We want it seems much simple, so we make

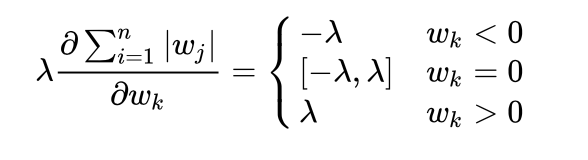


Then we have,

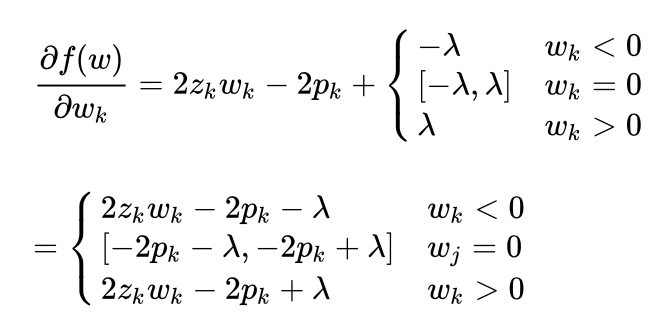


2) regular term

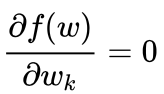
For the derivative of the penalty term, we need to use subgradient,



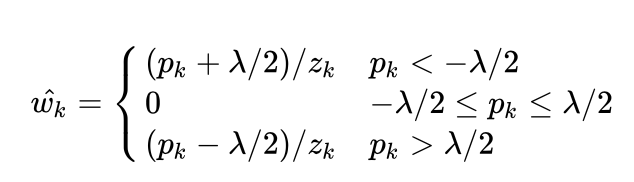
The whole derivative is as follows,



Then we will require



So, we could get:



According to the above formula, we can select one dimension for optimization every time and then get the optimal regression coefficient.

**3 The Model**

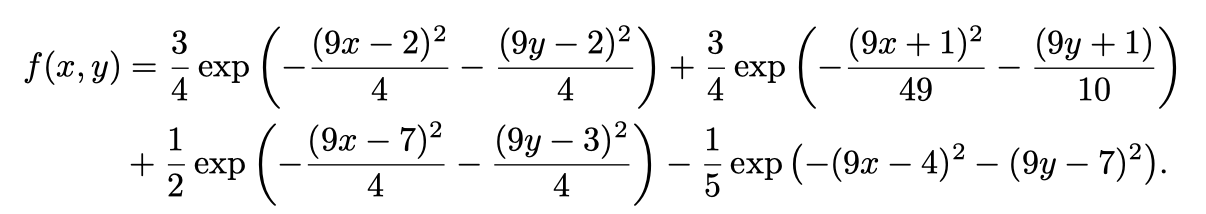
**3.1 Generating the data**

Depend on the computational requirements, we used two data sets.

1. For demanding computations, we create a small design matrix (20 X 20).
2. For the rest amount of calculation, we used a larger design matrix (40 X 40).

In the later part, we would use data 1 and data 2 to represent these two matrices respectively.

**3.2 Ordinary Least Square on the Franke function with resampling.**

As mentioned previously, the Franke function as follows

At the beginning, we set our own original function F(x, y) consisting of the function Franke Function f(x, y) with x, y ∈ [0, 1] and a stochastic noise function N(x, y)

F(x, y) = f(x, y) + N(x, y)

Then we create our design matrix X for a polynomial up to degree of 5

'1',

"x", "y",

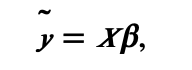
"x^2", "xy", "y^2",

"x^3", "x^2y", "xy^2", "y^3",

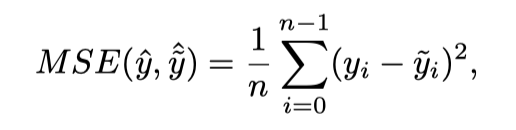
"x^4", "x^3y", "x^2y^2", "xy^3", "y^4",

"x^5", "x^4y", "x^3y^2", "x^2y^3", "xy^4", "y^5"

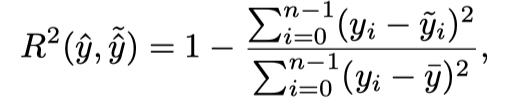
Using singular value decomposition (for big matrix) from numpy as well as our own, an analysis was performed with standard least squares regression.Then, we compute their variances to find the conﬁdence intervals of the parameters β and get the predict fitting function we want as follows



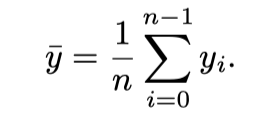
And then, evaluate the Mean Squared error (MSE)



and the score function.



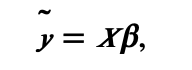
where we have deﬁned the mean value of as



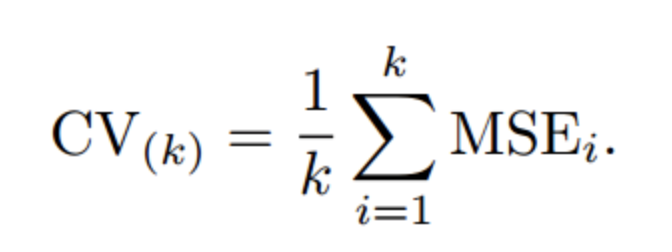
**3.3 Resampling techniques**

On the basement of the part a), we use the k-fold cross-validation to solve the problem of small amount of data.

First of all, we use the same predict fitting function in part a), using SVD the same:



We divide the original labeled dataset into 5 datasets. And then we take one dataset at a time as the only element of the test set, and the other 4 datasets as the training set for the training model and the parameters. The result is that we end up training 5 models and getting an MSE each time. Finally, we calculate the average of all these MSE as follows



And then draw the picture, observe how the MSE changes with the λ.

**3.4 Bias-variance tradeoﬀ.**

Based on previously performed OLS, we use the same predict fitting function with k-fold cross-validation and bootstrap.

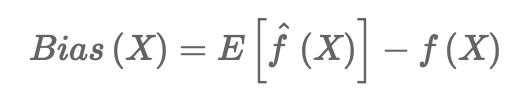
1. In the original sample, a certain number of new samples (40 X 40 or 20 X 20) will be sampled by re-sampling.

2. Calculate the statistics we need to estimate error, bias and variance based on the generated new samples.

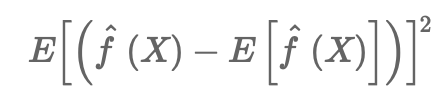
3. Repeat the above steps n times.

4. Finally, we can calculate the mean and variance of the estimator.

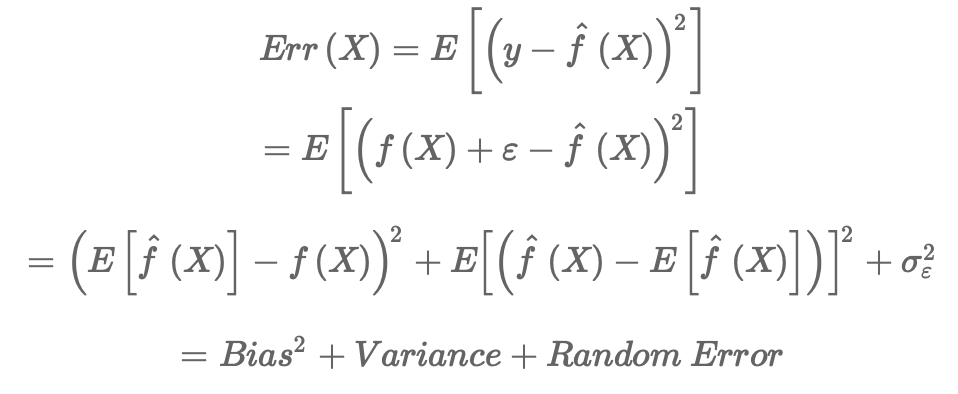
Then we calculate the bias of the one with resampling method, which describes the difference between the expectation of the result fitted by learning and the actual result



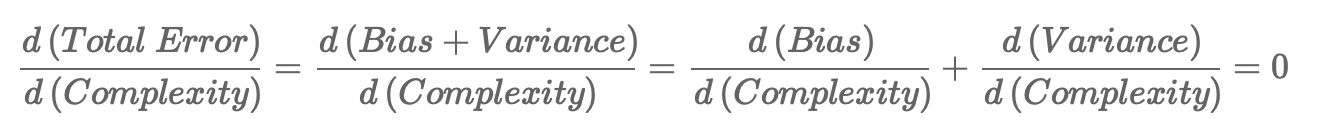
Calculate the ariance of the one with resampling method, which describes the instability of the result obtained by learning to fit,



calculate mean square error of the one with resampling method:



Considering that the model error is the sum of deviation and variance, the optimal position in the figure is actually the inflection point of the Total Error curve. We know that the inflection point of a continuous function means that the value of the first derivative is 0.

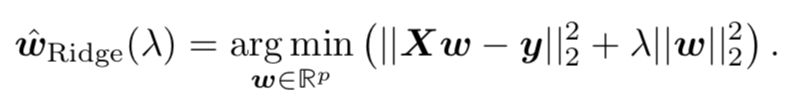


Use this formula gives a mathematical description of finding the optimal equilibrium point. If the complexity of the model is less than the equilibrium point, the deviation of the model will be high, and the model tends to underfit. If the complexity of the model is greater than the equilibrium point, the variance of the model will be too high, and the model tends to overfit.

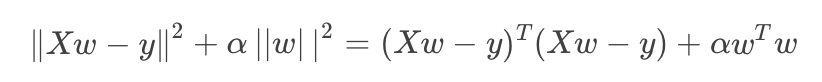
Then we plot MSE, bias and variance of an OSL model against test data of the Franke function, with bootstrap resampling and the picture of training and sample errors as functions of model complexity.

**3.5** **Ridge Regression on the Franke function with resampling.**

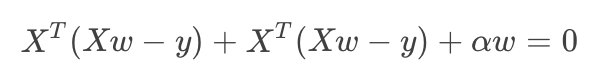
On the basis of previous analysis with OLS, we now use Ridge regression. Add to the original cost function. The cost function of the penalty term of norm is as follows:

****

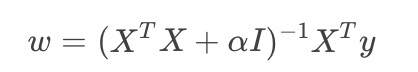
The larger the λ value of, the more obvious the role of the regular term and the punishment term; the smaller the λ value, the less useful the regular term. In extreme cases, when λ = 0, it's the same as the original cost function, if λ is equal to infinity, the cost function only has the regular term, and the result of its minimization must be w=0. We can rewrite it as



And then the derivative with respect to w



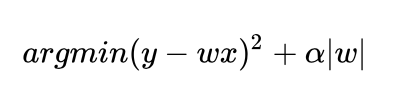
Solve this, we get the solution as follows,



And then we plot estimated MSE for ridge regression of Franke function data on a polynomial of degree 5 using K-fold cross-validation to see how MSE varies with the λ. We also studied the effect of the resampling method on the value of λ yielding the lowest error.

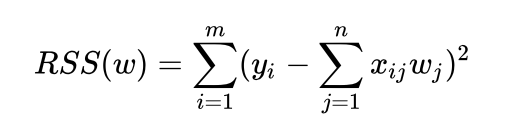
**3.6 Lasso Regression on the Franke function with resampling.**

As a complement to the previous regression techniques, we used Lasso regression. Add to the original cost function. The cost function of the penalty term of norm is as follows:

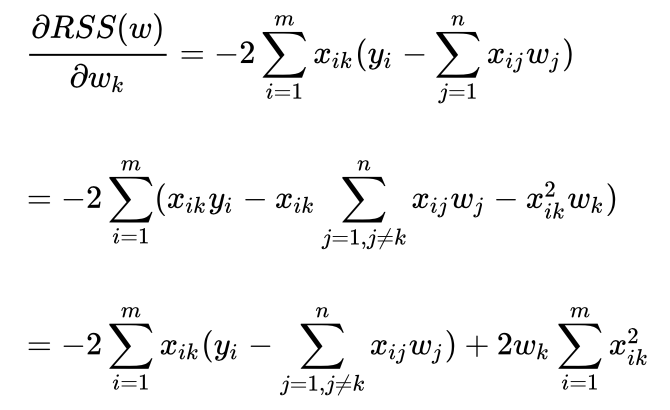


Now, we are going to solve the two parts of LASSO cost function respectively:

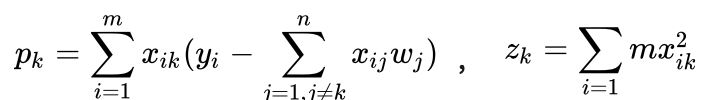
1) RSS



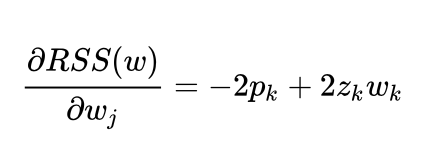
Then the derivation as follows,



We want it seems much simple, so we make

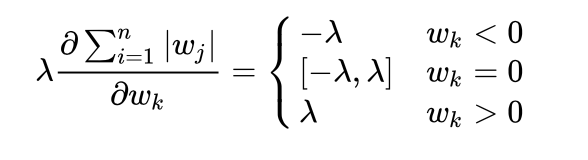


Then we have,

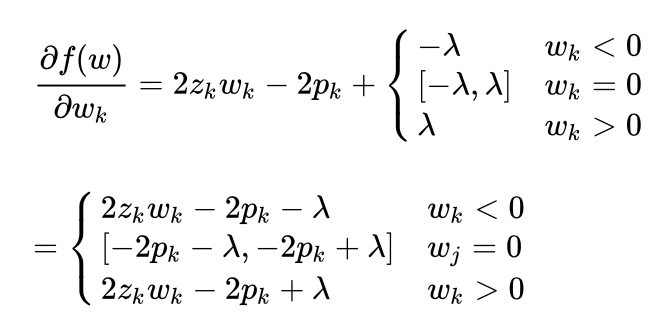


2) regular term

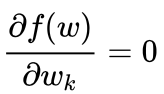
For the derivative of the penalty term, we need to use subgradient,



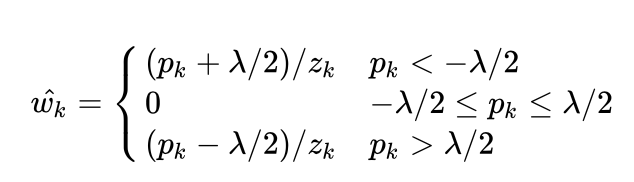
The whole derivative is as follows,



Then we will require



So, we could get:



According to the above formula, we can select one dimension for optimization every time and then get the optimal regression coefficient. Then plot the bias-variance tradeoff as along a comparison of training and test errors. As both Lasso and ridge depend on the λ hyperparameter, we compared its effects visually by plotting at once the mse for test and training error estimates for both methods.

**3.7** **Introducing real data.**

We use the data from the Github -- (SRTM\_data\_Norway\_1.tif)

**3.8 OLS, Ridge and Lasso regression with resampling on the Norwegian spatial data**

The analyses performed on the Franke function datasets have been repeated on the Norwegian spatial data. Evaluate the results and compare all these three regression methods to the type of data we generate.

**4 Results and analysis**

**4.1 Ordinary Least Squares on the Franke function with resampling.**

The 3-D figures of F(x,y) is as follows,

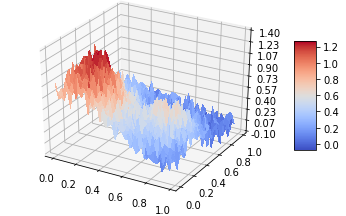


Fig. 4.1-1 Franke function surface generated with noise.

Then we calculate the Mean Squared error (MSE) with noise (function F(x, y))and without noise (function f(x, y))and the score with noise ( function F(x, y) ) and without noise(function f(x, y)) as follows：

|  |  |
| --- | --- |
| Mean squared error (own SVD code, no noise -- f(x, y)): | 0.00 |
| score (own SVD code, no noise -- f(x, y)): | 0.97 |
| Mean squared error (own SVD code, with noise -- F(x, y)): | 0.01 |
| score (own SVD code, with noise -- F(x, y)): | 0.87 |

Table 4.1-1 The Mean Squared error (MSE) with noise (function F(x, y))and without noise (function f(x, y))and the score with noise ( function F(x, y) ) and without noise(function f(x, y))

From the table we could see

1. Whether we add noise or not, MSE doesn’t vary much (below 0.01), which means that OLS method of Franke function data on a polynomial of degree 5 fit the data well no matter we add noise or not.
2. There’s big difference between score ( = 0.97) with noise and score ( = 0.87) without noise, which means that OLS method of Franke function data on a polynomial of degree 5 fit data well when there’s no noise.

**4.2 Resampling techniques**

Then, we get the result of OLS of Franke function data on a polynomial of degree 5 using own K-fold cross-validation as follows.

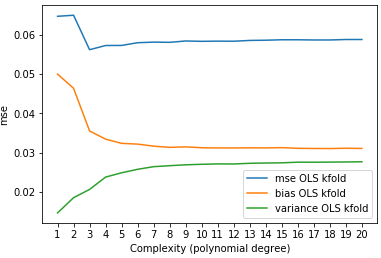


Fig. 4.2-1 Estimated MSE for OLS of Franke function data on a polynomial of degree 5 using own K-fold cross-validation code (K-fold).

From the figure, we can see the error is lowest for polynomial degree 3. The meeting of the curves is not seen on this range of 20 degrees however. Besides, the code works as the sum of bias and variance equals the error. The variance of the noise adds a third error term but here it is not present on a noticeable magnitude. It can happen on very large matrices as we have experienced.

**4.3 Bias-variance tradeoﬀ.**

As opposed to the result for k-fold cross-validation for OLS, the bootstrap method yielded somewhat superposed curves suggesting a possible best fit on the polynomial of degree 7 where the error and bias are minimal.

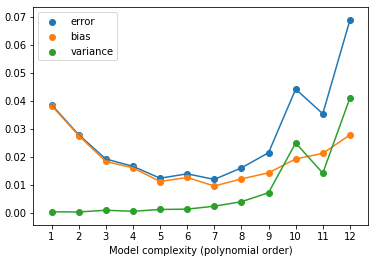


Fig. 4.3-1 MSE, bias and variance of an OSL model against test data of the Franke function, with bootstrap resampling.

From the figure we could see, at the beginning, bias was large while the variance is small (so-called Underfitting). As the training went on (with the increase of the model complexity -- polynomial order), bias and error got smaller and became much closer to the real value and the variance didn’t vary much (small increase). However, after passed a specific point, variance start to become very large, which results in that we cannot achieve good results on many other data sets and that is the so-called overfitting.

Therefore, the high bias (underfitting) has the following two characteristics:

1) high error in training set

2) the error of verification set is about the same as that of training set

High variance (overfitting) has the following two characteristics:

1) the training set error is low

2) very high verification set error

Which (all of these 4 characteristics) we have mentioned previously

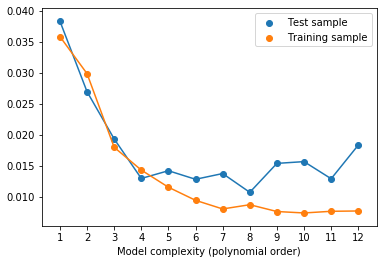


Fig. 4.3-2 Test and training error for OLS without resampling for the Franke function.

From the figure, we could see that both the errors of Test sample and Training sample were large (so-called underfitting) but they are almost the same at the beginning. Overfitting appears to begin on the 8th degree in figure 4.3-2. As the training went on (with the increase of the model complexity -- polynomial order), the errors of test sample and training sample got smaller. However, after a specific point, the errors of test sample start to become large while the errors of Training sample became much smaller and tend to be stable, which results in that we cannot achieve good results on many other data sets and that is the so-called overfitting. There is no benefit of fitting on higher degrees since the model is fitting the noise.

**4.4** **Ridge Regression on the Franke function with resampling.**

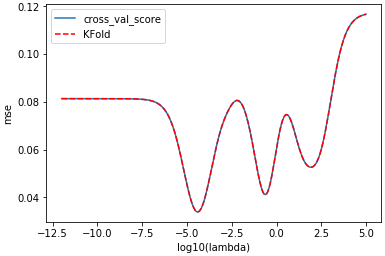
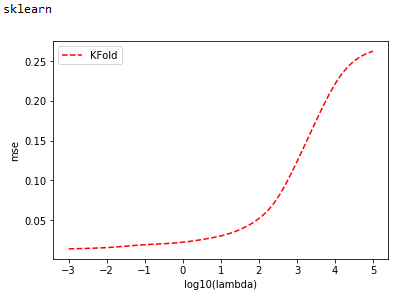
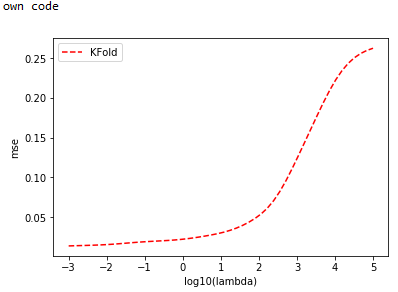


Fig. 4.4-1 Estimated MSE for ridge regression of Franke function data on a polynomial of degree 5 using K-fold cross-validation. Own k-fold as well as sci-kit learn cross-validation score function are used and result in matching curves.

From the figure, we could easily see that the smallest point of MSE for ridge regression of Franke function data on a polynomial of degree 5 using K-fold cross-validation is when λ close to . It must be noted data was not shuffled. The curve is otherwise quite different as can be seen in figure 4.4-2 for example.

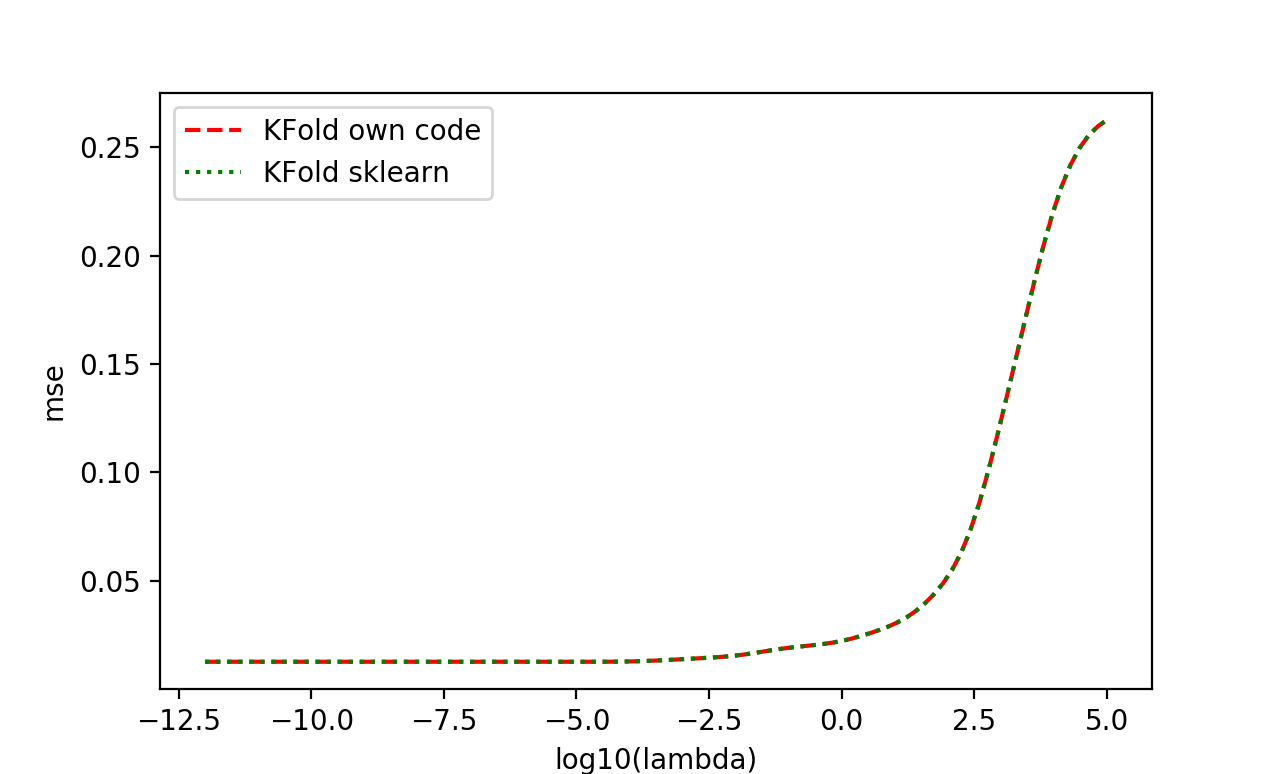


Fig. 4.4-2 Estimated MSE for ridge regression of Franke function test data on a polynomial of degree 5 using K-fold cross-validation (with our own code – the first figure, SciKit Learn’s ridge function – the second figure and combine them in the same picture).

The curves are the same while generated using SciKit Learn’s ridge function and own code for ridge regression.

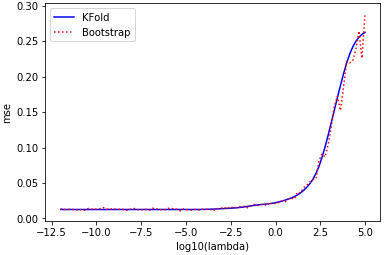


Fig. 4.4-3 Estimated MSE for ridge regression of Franke function test data on a polynomial of degree 5 using K-fold cross-validation and Bootstrap.

The curves are almost the same while generated using K-fold cross-validation and Bootstrap for ridge regression.

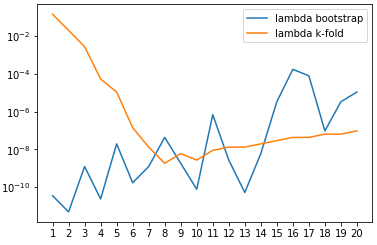


Fig. 4.4-4 λ for ridge regression of Franke function yielding the lowest MSE on test data polynomials of degrees 1-20 using K-fold cross-validation and Bootstrap.

λ of bootstrap sharply decreases at the beginning while the curve of the λ of k-fold cross validation increase. After passing a specific point, λ of bootstrap start to slight increase and λ of k-fold cross validation increase the same.

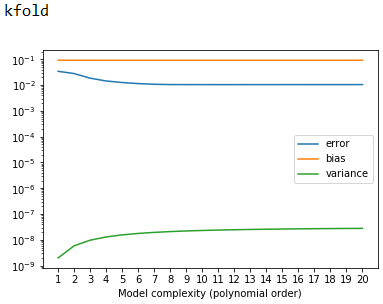


Fig. 4.4-5 Estimated MSE, bias and variance for ridge regression of Franke function training data on polynomials of degrees 1-20 using bootstrap cross-validation with our own code.

Variance have a slight increase at the beginning while error have a slight decrease at the beginning. And then both Variance and error keep stable. Bias always keep stable.

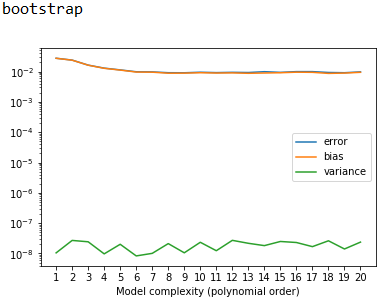


Fig. 4.4-6 Estimated MSE, bias and variance for ridge regression of Franke function training data on polynomials of degrees 1-20 using bootstrap cross-validation with our own code.

The curves of the bias and error are almost the same. And error, bias and variance keep stable.

Depending on the shuffle parameter of the K-fold function being set to True or False, the highest values of lambda providing the lowest MSE are quite close (Table 4.4-1). For example, the lambda and MSE are 69% and 63 % higher when shuffle is set to false when performing ridge regression with k-fold cross-validation on a polynomial degree of 5.

|  |  |  |
| --- | --- | --- |
| Shuffle | True | False |
| lambda | 3.95543624473e-05 | 1.2194734367e-05 |
| MSE | 0.0339472244285 | 0.0126313256466 |

Table 4.4-1 Highest lambda values yielding the lowest MSE for ridge regression with polynomial degree 5 depending on the use of shuffling in k-fold cross-validation.

**4.5 Lasso Regression on the Franke function with resampling.**

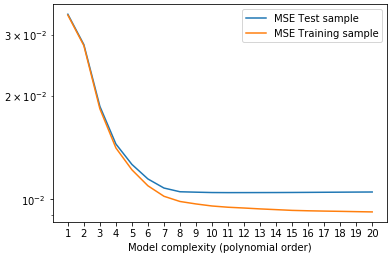


Fig. 4.5-1 Estimated MSE for Lasso regression of Franke function test and training data on polynomials of degrees 1-20 using K-fold cross-validation.

From the figure, we could see that the MSE of test sample and training sample are large at the beginning. After passing a specific point, the test sample keep stay while the training sample keep decrease. They follow the expected pattern of overfitting past a certain point for the training set, although the test error does not start to increase drastically in this range of polynomial degrees.

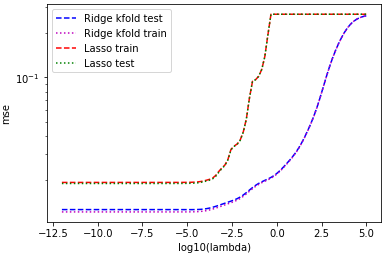


Fig. 4.5-2 Estimated MSE for ridge regression and Lasso Regression of Franke function test data and training data on a polynomial of degree 5 using K-fold cross-validation.

The patterns are similar for ridge and Lasso regression in relation to lambda, where the training errors are lower than test and Lasso presenting a higher error for the same lambda values.

**4.6** **Real data.**

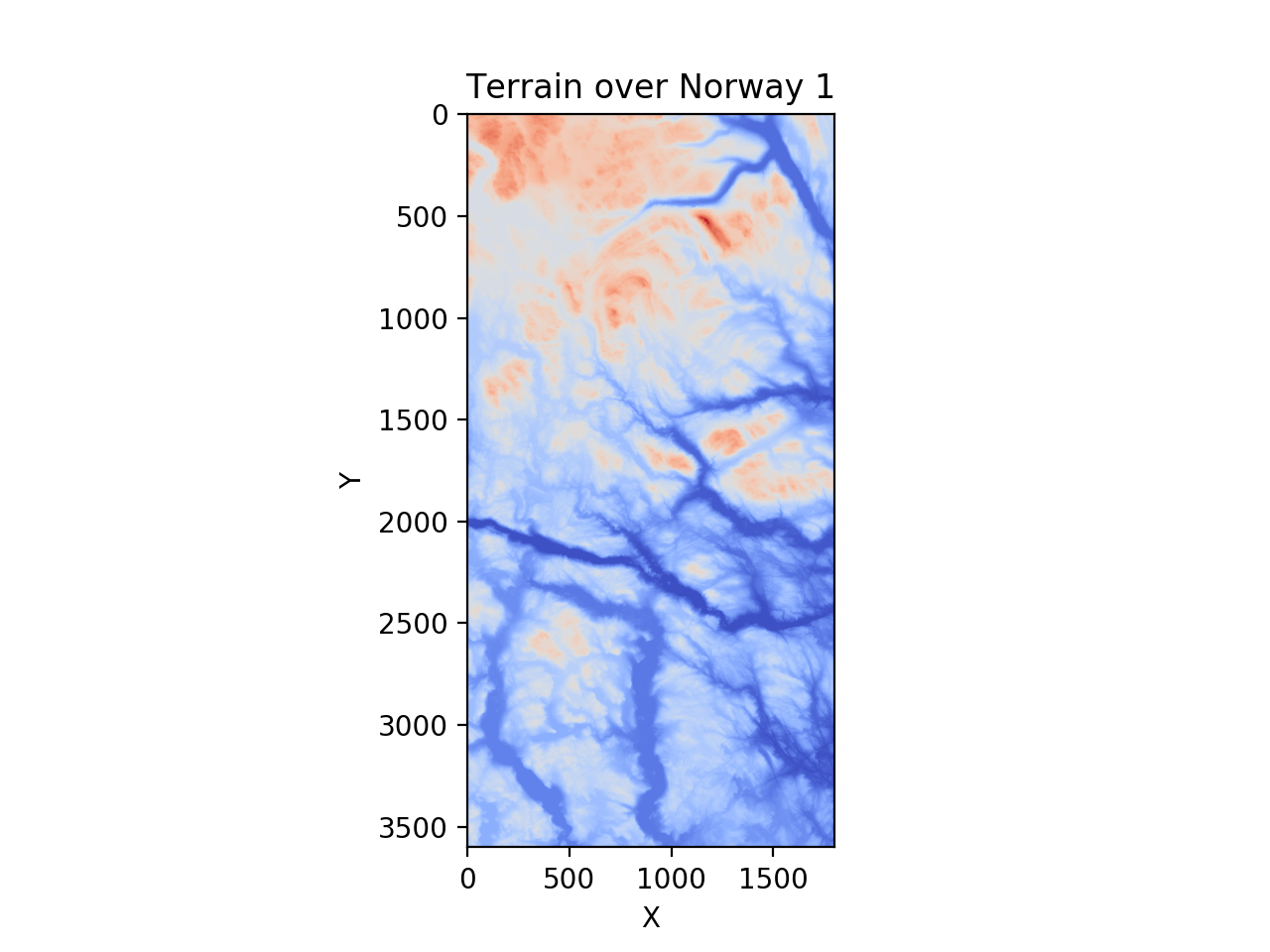
**4.6.1 Ordinary Least Square with resampling.** 

Fig. 4.6.1-1 The 3-D figures of Terrain over Norway 1

Then we calculate the Mean Squared error (MSE) with noise (function F(x, y))and without noise (function f(x, y))and the score with noise ( function F(x, y) ) and without noise(function f(x, y)) as follows：

|  |  |
| --- | --- |
| Mean squared error (own SVD code, no noise -- f(x, y)): | 0.00 |
| score (own SVD code, no noise -- f(x, y)): | 0.97 |
| Mean squared error (own SVD code, with noise -- F(x, y)): | 0.01 |
| score (own SVD code, with noise -- F(x, y)): | 0.87 |

Table 4.1-1 The Mean Squared error (MSE) with noise (function F(x, y))and without noise (function f(x, y))and the score with noise ( function F(x, y) ) and without noise(function f(x, y))

From the table we could see，

1. Whether we add noise or not, MSE doesn’t vary much (below 0.01), which means that OLS method of Franke function data on a polynomial of degree 5 fit the data well no matter we add noise or not.
2. There’s big difference between score ( = 0.97) with noise and score ( = 0.87) without noise, which means that OLS method of Franke function data on a polynomial of degree 5 fit data well when there’s no noise.

**4.6.2 Resampling techniques**

Then, we get the result of OLS of Terrain over Norway 1 data on a polynomial of degree 5 using own K-fold cross-validation as follows.

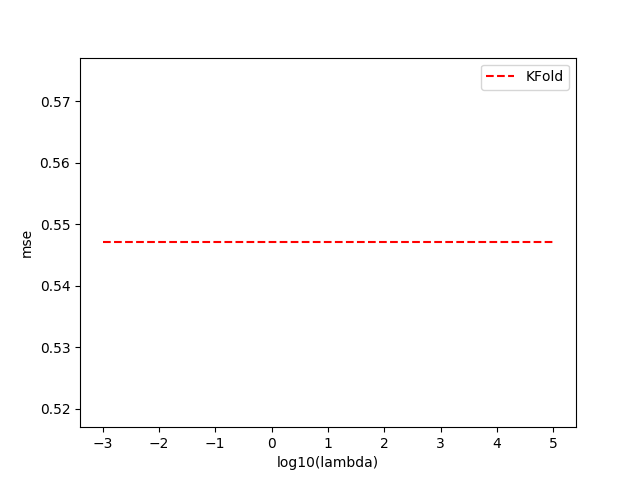


Fig. 4.6.2-1 Estimated MSE for OLS of Terrain over Norway 1 data on a polynomial of degree 5 using own K-fold cross-validation code (K-fold).

From the figure, we find an interesting trend that there’s no change for the MSE with the lambda. The reason why this happens is that there’s no penalty term in OLS method (λ is the independent variable to MSE in OLS method).

**4.6.3 Bias-variance tradeoﬀ.**

From the code of part c, we get the MSE, bias and variance of an OSL model of degree 5 against test data of the Terrain over Norway 1, with bootstrap resampling and MSE, bias and variances of an OSL model of degree 5 against training and test data of the Terrain over Norway 1, without resampling respectively as follows.

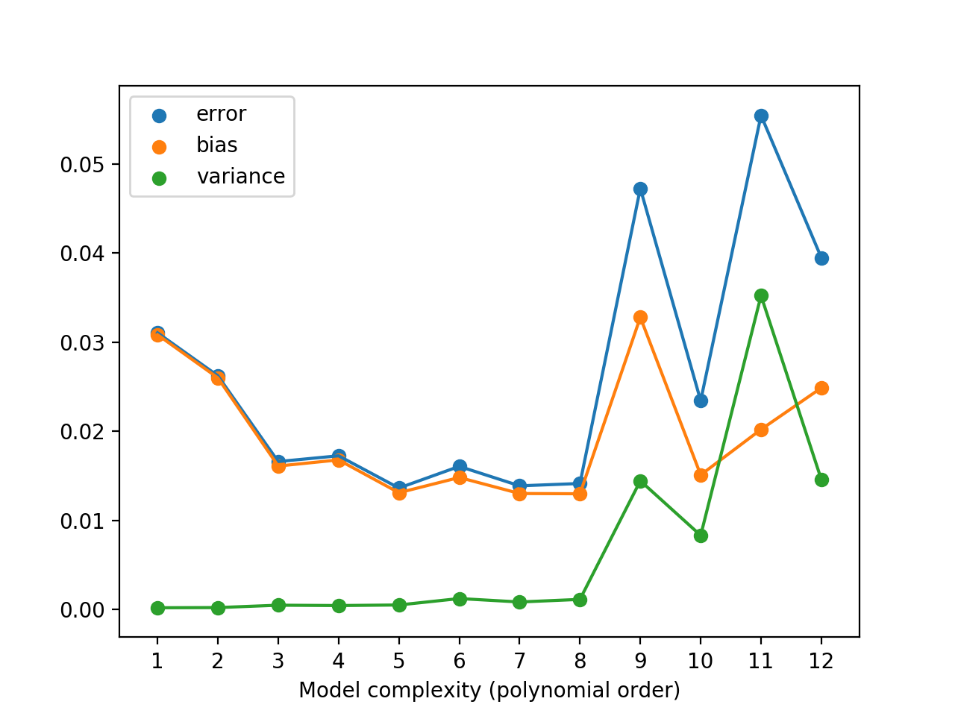
****

Fig. 4.6.3-1 MSE, bias and variance of an OSL model against test data of the Terrain over Norway 1, with bootstrap resampling.

From the figure we could see, at the beginning, Bias was large while the variance is small (so-called Underfitting) With the training went on (with the increase of the Model complexity -- polynomial order), Bias and error got smaller and became much closer to the real value and the variance didn’t vary much (small increase). However, after passed a specific point (degree of 8), variance start to become very large, which results in that we cannot achieve good results on many other data sets and that is the so-called Overfitting. At the same time, Bias and error both become much large.

Therefore, the high bias (underfitting) has the following two characteristics:

1) high error in training set

2) low variance in training set

High variance (overfitting) has the following two characteristics:

1) high error in training set

2) high variance in training set

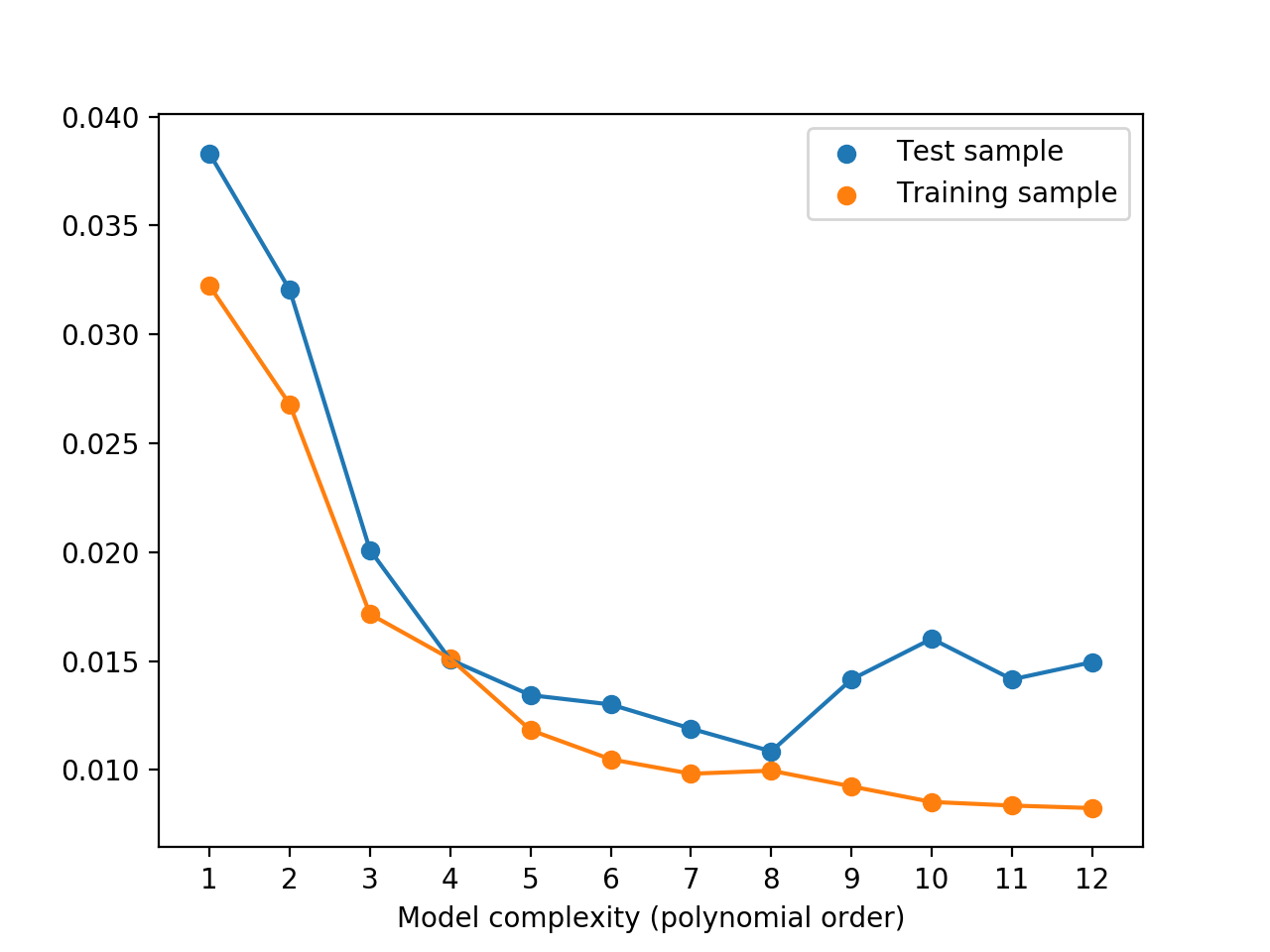


Fig. 4.6.3-2 MSE, bias and variances of an OSL model against training and test data of the Terrain over Norway 1.

From the figure, we could see that both the errors of Test sample and Training sample were large (so-called Underfitting) but they are almost the same at the beginning. With the training went on (with the increase of the Model complexity -- polynomial order), the errors of Test sample and Training sample got smaller. However, after passed a specific point, the errors of Test sample start to become large while the errors of Training sample became much smaller and tend to be stable, which results in that we cannot achieve good results on many other data sets and that is the so-called Overfitting.

**4.6.4 Ridge Regression with resampling.**

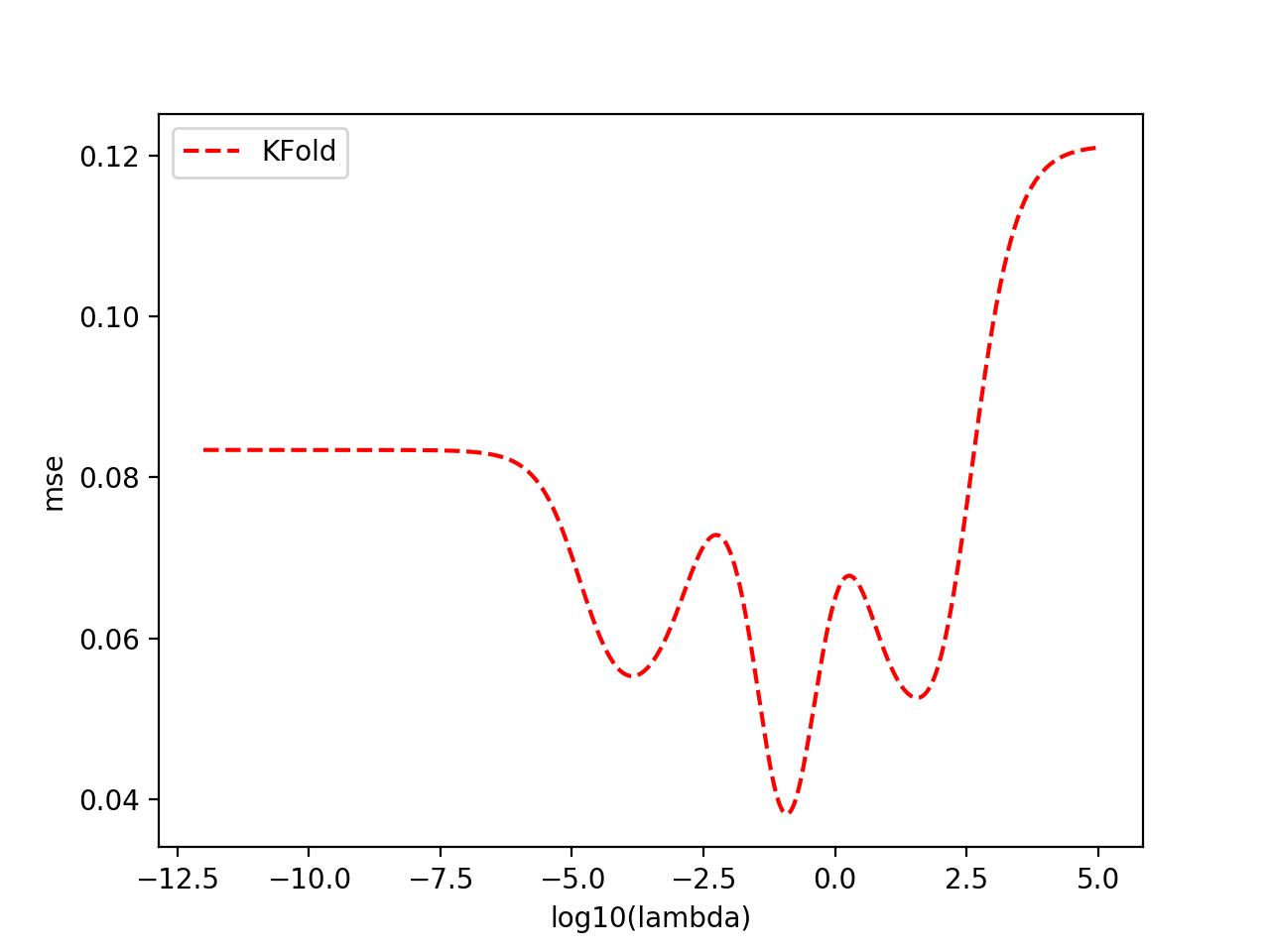


Fig. 4.6.4-1 Estimated MSE for ridge regression of the Terrain over Norway 1 data on a polynomial of degree 5 using K-fold cross-validation.

From the figure, we could easily see that the smallest point of MSE for ridge regression of Franke function data on a polynomial of degree 5 using K-fold cross-validation is when λ close to .

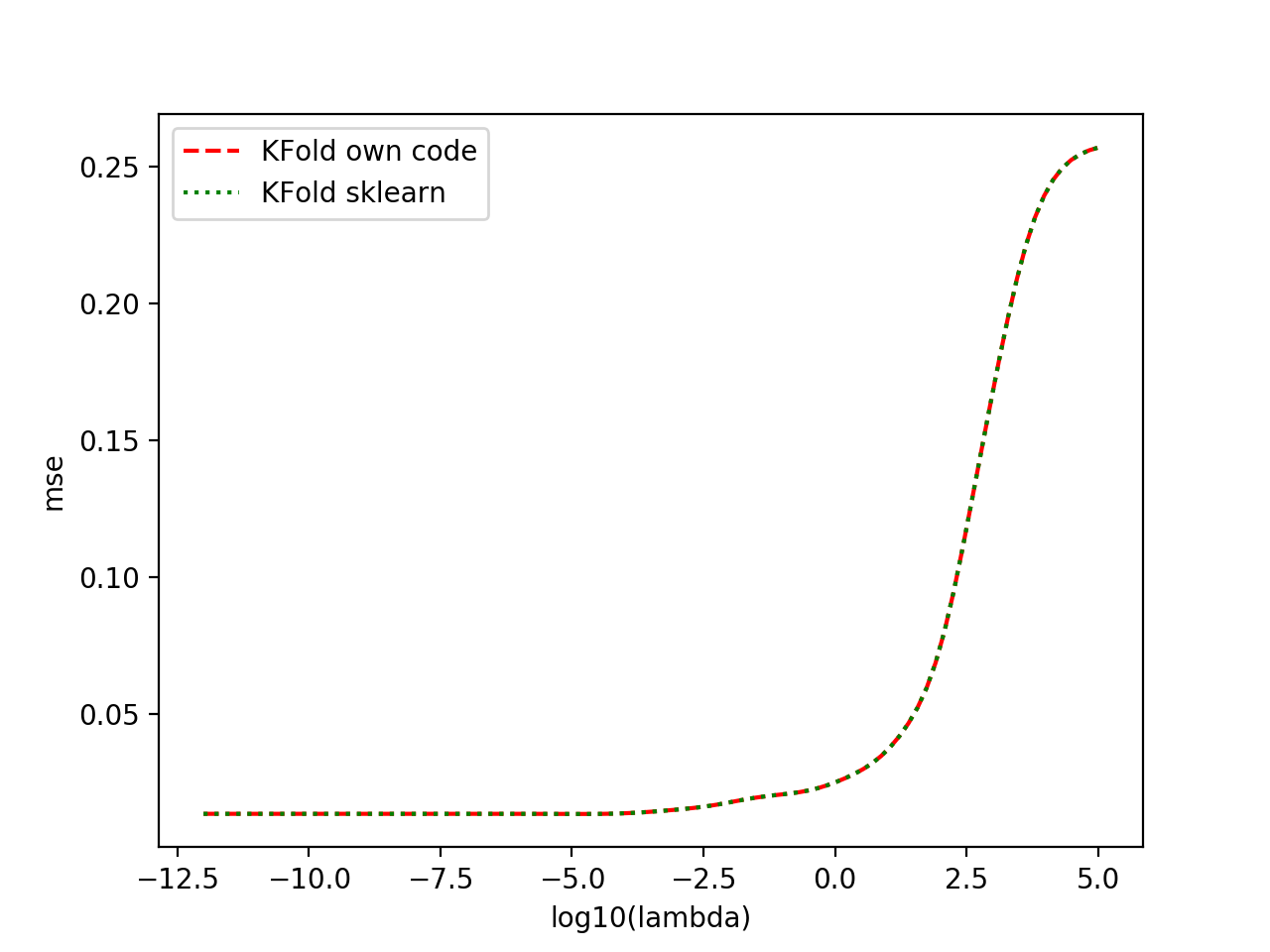


Fig. 4.6.4-2 Estimated MSE for ridge regression of the Terrain over Norway 1 test data on a polynomial of degree 5 using K-fold cross-validation (with our own code – the first figure, SciKit Learn’s ridge function – the second figure and combine them in the same picture).

The curves are the same while generated using SciKit Learn’s ridge function and own code for ridge regression.

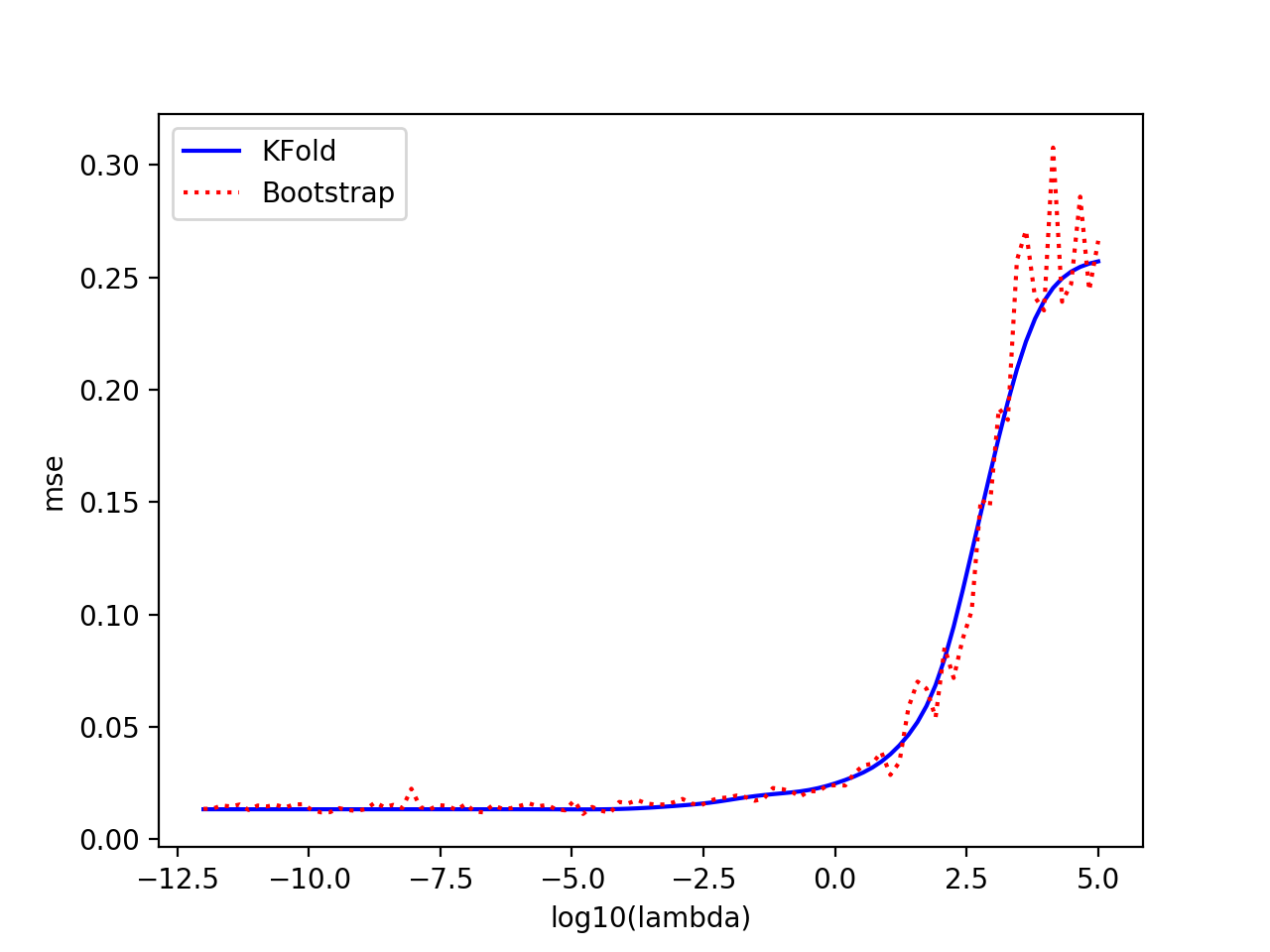


Fig. 4.6.4-3 Estimated MSE for ridge regression of the Terrain over Norway 1test data on a polynomial of degree 5 using K-fold cross-validation and Bootstrap.

The curves are almost the same while generated using K-fold cross-validation and Bootstrap for ridge regression.

**4.6.5 Lasso Regression with resampling.**

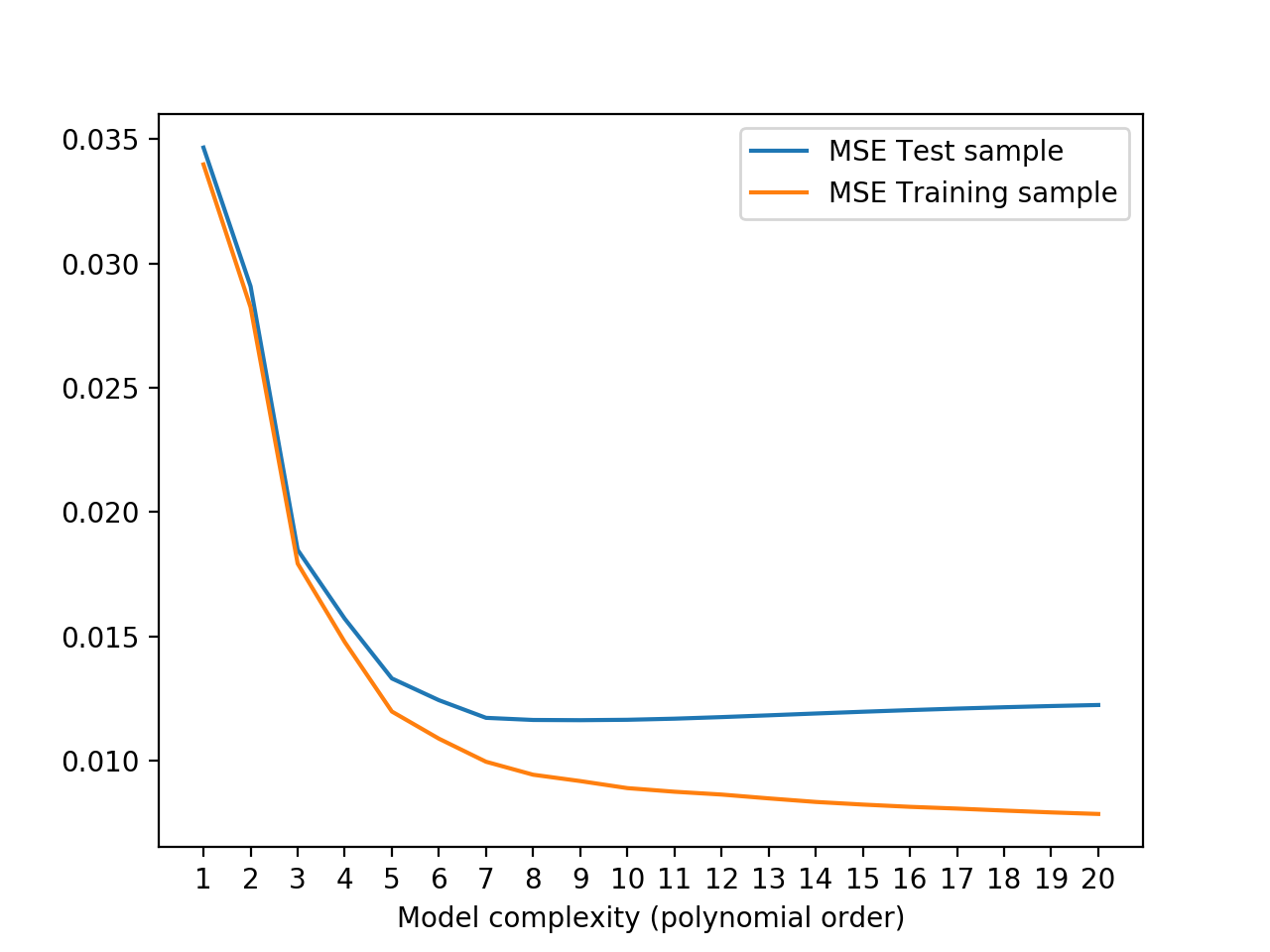


Fig. 4.6.5-1 Estimated MSE for Lasso regression of the Terrain over Norway 1 test data and training data on a polynomial of degree 5 using K-fold cross-validation and Bootstrap.

From the figure, we could see that the MSE of test sample and training sample are large at the beginning. After passing a specific point, the test sample start to have a slight increase while the training sample keep decrease.

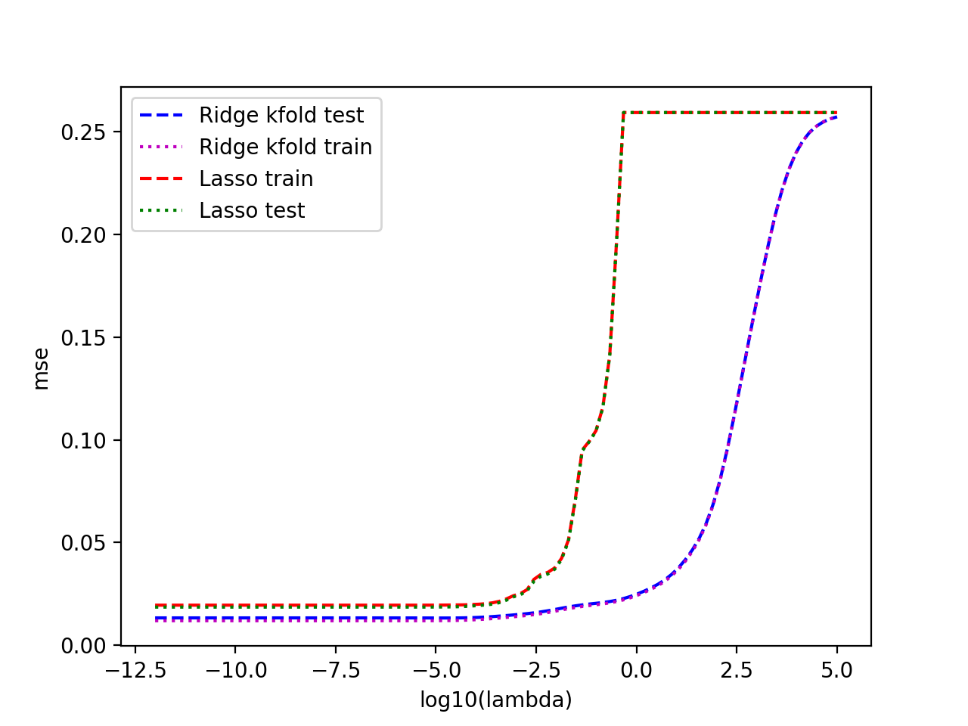


Fig. 4.6.5-2 Estimated MSE for ridge regression and Lasso Regression of the Terrain over Norway 1 test data and training data on a polynomial of degree 5 using K-fold cross-validation

**4.6.7 OLS, Ridge and Lasso regression with resampling.**

|  |  |  |  |
| --- | --- | --- | --- |
| Degree | MSE OLS | Bias OLS | Variance OLS |
| 1 | 0.034666 | 0.094812 | 0.00054 |
| 2 | 0.029088 | 0.094749 | 0.000595 |
| 3 | 0.018474 | 0.094389 | 0.000686 |
| 4 | 0.01571 | 0.094443 | 0.000717 |
| 5 | 0.013354 | 0.094503 | 0.000745 |
| 6 | 0.012475 | 0.094524 | 0.000749 |
| 7 | 0.011814 | 0.094434 | 0.000753 |
| 8 | 0.011804 | 0.094419 | 0.000765 |
| 9 | 0.011955 | 0.094452 | 0.000765 |
| 10 | 0.012702 | 0.094467 | 0.000776 |
| 11 | 0.013463 | 0.094472 | 0.000782 |
| 12 | 0.015958 | 0.094549 | 0.000806 |
| 13 | 0.015303 | 0.094514 | 0.000803 |
| 14 | 0.016646 | 0.094604 | 0.000818 |
| 15 | 0.017481 | 0.094584 | 0.000821 |
| 16 | 0.019875 | 0.094644 | 0.000849 |
| 17 | 0.026963 | 0.094823 | 0.00092 |
| 18 | 0.027888 | 0.094837 | 0.000931 |
| 19 | 0.024617 | 0.094722 | 0.000897 |

Table 4.6.7-1 for error, bias and variance by complexity for OLS

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Degree | MSE Ridge | Bias Ridge | Variance Ridge | lambda Ridge |
| 1 | 0.034663 | 0.094791 | 6.57E-10 | 1.45E-01 |
| 2 | 0.02908 | 0.094749 | 1.92E-09 | 4.43E-02 |
| 3 | 0.018473 | 0.094389 | 3.16E-09 | 5.72E-04 |
| 4 | 0.015703 | 0.09442 | 4.21E-09 | 5.34E-05 |
| 5 | 0.013302 | 0.09443 | 5.08E-09 | 1.63E-05 |
| 6 | 0.012426 | 0.09443 | 5.79E-09 | 4.64E-07 |
| 7 | 0.011716 | 0.094426 | 6.38E-09 | 1.96E-08 |
| 8 | 0.011631 | 0.094419 | 6.88E-09 | 1.83E-09 |
| 9 | 0.011622 | 0.094421 | 7.30E-09 | 1.32E-08 |
| 10 | 0.011638 | 0.094422 | 7.65E-09 | 6.43E-08 |
| 11 | 0.011682 | 0.094423 | 7.96E-09 | 1.42E-07 |
| 12 | 0.011746 | 0.094427 | 8.24E-09 | 3.13E-07 |
| 13 | 0.011817 | 0.09443 | 8.48E-09 | 4.64E-07 |
| 14 | 0.011891 | 0.094433 | 8.69E-09 | 6.89E-07 |
| 15 | 0.011961 | 0.094436 | 8.88E-09 | 1.02E-06 |
| 16 | 0.012027 | 0.094439 | 9.05E-09 | 1.52E-06 |
| 17 | 0.012088 | 0.094441 | 9.20E-09 | 2.26E-06 |
| 18 | 0.012143 | 0.094442 | 9.34E-09 | 3.35E-06 |
| 19 | 0.012191 | 0.094442 | 9.47E-09 | 4.98E-06 |
| 20 | 0.012233 | 0.094441 | 9.58E-09 | 7.39E-06 |

Table 4.6.7-2 for error, bias, variance and lambda by complexity for Ridge

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Degree | MSE Lasso | Bias Lasso | Variance Lasso | lambda Lasso |
| 1 | 0.034665 | 0.094783 | 0 | 1.18E-04 |
| 2 | 0.029078 | 0.094749 | 0 | 5.34E-05 |
| 3 | 0.020545 | 0.094508 | 0 | 1.18E-04 |
| 4 | 0.019493 | 0.094431 | 0 | 5.34E-05 |
| 5 | 0.019542 | 0.094451 | 0 | 1.02E-06 |
| 6 | 0.019127 | 0.094462 | 0 | 1.00E-12 |
| 7 | 0.018587 | 0.094459 | 0 | 1.00E-12 |
| 8 | 0.017968 | 0.09445 | 0 | 1.00E-12 |
| 9 | 0.017423 | 0.094438 | 0 | 1.00E-12 |
| 10 | 0.017088 | 0.094431 | 0 | 1.00E-12 |
| 11 | 0.01696 | 0.094428 | 0 | 1.00E-12 |
| 12 | 0.016968 | 0.094427 | 0 | 1.00E-12 |
| 13 | 0.017033 | 0.094427 | 0 | 6.89E-07 |
| 14 | 0.017106 | 0.094429 | 0 | 1.52E-06 |
| 15 | 0.017168 | 0.094431 | 0 | 1.52E-06 |
| 16 | 0.017216 | 0.094434 | 0 | 2.26E-06 |
| 17 | 0.017251 | 0.094437 | 0 | 2.26E-06 |
| 18 | 0.01727 | 0.094438 | 0 | 2.26E-06 |
| 19 | 0.017271 | 0.094439 | 0 | 2.26E-06 |
| 20 | 0.017267 | 0.09444 | 0 | 2.26E-06 |

Table 4.6.7-3 for error, bias, variance and lambda by complexity for Lasso

**4.7 OLS, Ridge and Lasso regression with resampling.**

|  |  |  |  |
| --- | --- | --- | --- |
| Degree | MSE OLS | Bias OLS | Variance OLS |
| 1 | 0.034666 | 0.094812 | 0.00054 |
| 2 | 0.029088 | 0.094749 | 0.000595 |
| 3 | 0.018474 | 0.094389 | 0.000686 |
| 4 | 0.01571 | 0.094443 | 0.000717 |
| 5 | 0.013354 | 0.094503 | 0.000745 |
| 6 | 0.012475 | 0.094524 | 0.000749 |
| 7 | 0.011814 | 0.094434 | 0.000753 |
| 8 | 0.011804 | 0.094419 | 0.000765 |
| 9 | 0.011955 | 0.094452 | 0.000765 |
| 10 | 0.012702 | 0.094467 | 0.000776 |
| 11 | 0.013463 | 0.094472 | 0.000782 |
| 12 | 0.015958 | 0.094549 | 0.000806 |
| 13 | 0.015303 | 0.094514 | 0.000803 |
| 14 | 0.016646 | 0.094604 | 0.000818 |
| 15 | 0.017481 | 0.094584 | 0.000821 |
| 16 | 0.019875 | 0.094644 | 0.000849 |
| 17 | 0.026963 | 0.094823 | 0.00092 |
| 18 | 0.027888 | 0.094837 | 0.000931 |
| 19 | 0.024617 | 0.094722 | 0.000897 |

Table 4.7-1 for error, bias and variance by complexity for OLS

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Degree | MSE Ridge | Bias Ridge | Variance Ridge | lambda Ridge |
| 1 | 0.034663 | 0.094791 | 6.57E-10 | 1.45E-01 |
| 2 | 0.02908 | 0.094749 | 1.92E-09 | 4.43E-02 |
| 3 | 0.018473 | 0.094389 | 3.16E-09 | 5.72E-04 |
| 4 | 0.015703 | 0.09442 | 4.21E-09 | 5.34E-05 |
| 5 | 0.013302 | 0.09443 | 5.08E-09 | 1.63E-05 |
| 6 | 0.012426 | 0.09443 | 5.79E-09 | 4.64E-07 |
| 7 | 0.011716 | 0.094426 | 6.38E-09 | 1.96E-08 |
| 8 | 0.011631 | 0.094419 | 6.88E-09 | 1.83E-09 |
| 9 | 0.011622 | 0.094421 | 7.30E-09 | 1.32E-08 |
| 10 | 0.011638 | 0.094422 | 7.65E-09 | 6.43E-08 |
| 11 | 0.011682 | 0.094423 | 7.96E-09 | 1.42E-07 |
| 12 | 0.011746 | 0.094427 | 8.24E-09 | 3.13E-07 |
| 13 | 0.011817 | 0.09443 | 8.48E-09 | 4.64E-07 |
| 14 | 0.011891 | 0.094433 | 8.69E-09 | 6.89E-07 |
| 15 | 0.011961 | 0.094436 | 8.88E-09 | 1.02E-06 |
| 16 | 0.012027 | 0.094439 | 9.05E-09 | 1.52E-06 |
| 17 | 0.012088 | 0.094441 | 9.20E-09 | 2.26E-06 |
| 18 | 0.012143 | 0.094442 | 9.34E-09 | 3.35E-06 |
| 19 | 0.012191 | 0.094442 | 9.47E-09 | 4.98E-06 |
| 20 | 0.012233 | 0.094441 | 9.58E-09 | 7.39E-06 |

Table 4.7-2 for error, bias, variance and lambda by complexity for Ridge

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Degree | MSE Lasso | Bias Lasso | Variance Lasso | lambda Lasso |
| 1 | 0.034665 | 0.094783 | 0 | 1.18E-04 |
| 2 | 0.029078 | 0.094749 | 0 | 5.34E-05 |
| 3 | 0.020545 | 0.094508 | 0 | 1.18E-04 |
| 4 | 0.019493 | 0.094431 | 0 | 5.34E-05 |
| 5 | 0.019542 | 0.094451 | 0 | 1.02E-06 |
| 6 | 0.019127 | 0.094462 | 0 | 1.00E-12 |
| 7 | 0.018587 | 0.094459 | 0 | 1.00E-12 |
| 8 | 0.017968 | 0.09445 | 0 | 1.00E-12 |
| 9 | 0.017423 | 0.094438 | 0 | 1.00E-12 |
| 10 | 0.017088 | 0.094431 | 0 | 1.00E-12 |
| 11 | 0.01696 | 0.094428 | 0 | 1.00E-12 |
| 12 | 0.016968 | 0.094427 | 0 | 1.00E-12 |
| 13 | 0.017033 | 0.094427 | 0 | 6.89E-07 |
| 14 | 0.017106 | 0.094429 | 0 | 1.52E-06 |
| 15 | 0.017168 | 0.094431 | 0 | 1.52E-06 |
| 16 | 0.017216 | 0.094434 | 0 | 2.26E-06 |
| 17 | 0.017251 | 0.094437 | 0 | 2.26E-06 |
| 18 | 0.01727 | 0.094438 | 0 | 2.26E-06 |
| 19 | 0.017271 | 0.094439 | 0 | 2.26E-06 |
| 20 | 0.017267 | 0.09444 | 0 | 2.26E-06 |

Table 4.7-3 for error, bias, variance and lambda by complexity for Lasso

**5 Strengths and Weaknesses**

Linear regression is generally suitable only for low dimensions, such as n=50, with five variables that do not yet exist with multicolinearity.

Ridge Regression could also create multicolinearity and add an L2 penalty term for ease of calculation. However, it doesn't shrink parameters to 0. So you can't do variable selection.

For LASSO regression, L1 penalty is not an analytic solution, but it can reduce some of the coefficients to 0. However, although LASSO can do variable selection, it is not consistent, and when n is very small, only n variables can be selected at most. And you can't do group selection.

**6.Reference**

[1] Jianhua Z. Huang. An Introduction to Statistical Learning: With Applications in R By Gareth James, Trevor Hastie, Robert Tibshirani, Daniela Witten[J]. Journal of Agricultural, Biological, and Environmental Statistics,2014,19(4).

**7.Appendix**

https://github.com/laurent0001/FYS-STK4155