MACHINE LEARNING: REGRESSION ANALYSIS WITH RESAMPLING METHODS

SHAFAQ NAZ SHEIKH
HTTPS://GITHUB.UIO.NO/SHAFAQNS/FYS-STK4155/TREE/MASTER/PROJECT1

Abstract. In this article we study three types of regression methods, ordinary least squares(OLS), ridge regression and Lasso regression. We implement the ordinary least squares and ridge methods ourselves, while the lasso is done by scikit's built in function in python. We apply these methods on two data sets, one set is generated from the Franke function, and the other set is terrain data from a region in Norway. We study the bias-variance trade off, which is a phenomenon that occurs in essentially all machine learning algorithms. In addition we look at re sampling methods, namely bootstrap and cross validation.

Date: 5th October 2020.

1. Introduction

Machine learning(ML) has become an important part of modern science, it involves the development of computer algorithms that learn how to describe and make predictions from a given set of data. This allows us to extract important information about correlations about physical processes, and the underlying patterns in large data sets. Regression is one of the algorithms that can be used for making predictions from data sets, linear regression is a supervised learning algorithm, this means that we have a set of variables, let us denote these as inputs, these are already measured. They have some influence on one or more outputs. For each example the goal is to use the inputs to predict the values of the outputs, this is called supervised learning. The other kind of learning is known as unsupervised learning, this deals with finding patterns and relationship in data sets without any prior knowledge of the system.

The main goal is to learn the behaviour of data using a model, in this case the model is the linear regression. We need data to construct prediction rules, often a large set is required. Let us assume that we have available a set of measurements (x_i, y_i) for $i = 1, ..., n$, this is known as the training data. In this article the model we use is linear regression, namely, ordinary least squares(OLS), ridge regression, and lasso regression. We present the necessary theory and background information in the next section.

2. Theory of Regression

The earliest form of regression was the method of least squares, which was published by Adrien-Marie Legendre in 1805, and by Gauss in 1809. They both tried it as a means of finding a rough approximation to a set of points for the prediction of planetary movement. The term *"regression"* was first coined by Francis Galton in the ninteenth century to describe a bological phenomenon. Decades later, the method is still used today due to their simplicity and implementation.

A linear regression model assumes that the regression function $E(Y|X)$ is linear in the inputs $X_1, X_2, ..., X_p$. Consider a data set $\mathcal{L}$ consisting of the data $\mathbf{X}_{\mathcal{L}} = (x_i, y_i), i = 0, .., n-1$. Now let us assume that the true data is generated from a noisy model

$$\mathbf{z} = f(\mathbf{x}, \mathbf{y}) + \boldsymbol{\epsilon}$$

where $\mathbf{z}$ is the observation vector, the function f is the model. Here $\boldsymbol{\epsilon}$ is normally distributed with mean zero and variance σ^2. The epsilons are added noise, we add noise to the data to make our model as close to reality as possible. This has to do with how data is collected in real life, it is possible to have some outliers, to train the model well, we add noise. This can be written in matrix notation as

$$\mathbf{z} = \boldsymbol{X\beta} + \boldsymbol{\epsilon}$$

where $\boldsymbol{X}$ is known as the design matrix, since we want to fit a polynomial of two variables of degree d, the design matrix looks like,

$$\begin{pmatrix} 1 & x_1 & y_1 & x_1^2 & x_1y_1 & \dots & x_1^d & x_1^{d-1}y_1 & \dots & y_1^d \\ 1 & x_2 & y_2 & x_2^2 & x_2y_2 & \dots & x_2^d & x_2^{d-1}y_2 & \dots & y_2^d \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 1 & x_n & y_n & x_n^2 & x_ny_n & \dots & x_n^d & x_n^{d-1}y_n & \dots & y_n^d \end{pmatrix}$$

To fit a polynomial of degree d, the vector $\boldsymbol{\beta}$ will have p features, or entries, where p is given by $p = \frac{(d+1)(d+2)}{2}$, so $\boldsymbol{X}$ is of size $n^2 \times p$ and z is of size $n^2 \times 1$ where n is the number of data points we have. We can then define an approximation to the function f in terms of the parameters $\boldsymbol{\beta}$ and the design matrix $\boldsymbol{X}$ which embody our model, that is

$$\hat{\mathbf{z}} = \boldsymbol{X\beta}$$

The parameters $\boldsymbol{\beta}$ are in turn found by optimizing mean squared error, via the so called cost function, $\boldsymbol{C(X,\beta)}$, the cost function is defined by :

$$\boldsymbol{C(X,\beta)} = \frac{1}{n}\sum_{i=0}^{n-1}(z_i - \tilde{z}_i)^2$$

The cost function can be written in matrix form as

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

2.1. **Ordinary least squares(OLS).**

In the OLS method we seek to minimize the cost function (1), that is, we get an optimization problem,

$$min_{\beta\in\mathbb{R}^{\prime}}\left\{\frac{1}{n}(\boldsymbol{z} - \boldsymbol{X\beta})^T(\boldsymbol{z} - \boldsymbol{X\beta})\right\} \tag{2}$$

This can be written using the L_2 norm as $\parallel \boldsymbol{Z} - \boldsymbol{X\beta} \parallel_2^2$, to find the minimum we differentiate with respect to β and set it equal to zero

$$= (\boldsymbol{z} - \boldsymbol{X\beta})^T(\boldsymbol{z} - \boldsymbol{X\beta}) = \boldsymbol{z^Tz} - \boldsymbol{\beta^TX^Tz} - \boldsymbol{z^TX\beta} + \boldsymbol{\beta^TX^TX\beta}$$

Here $\boldsymbol{(\beta^TX^Tz)^T = z^TX\beta}$ has the dimensionality 1x1, so it is a scalar and equal to its own transpose, hence $\boldsymbol{(\beta^TX^Tz)^T = z^TX\beta}$, the quantity to minimize becomes, $\boldsymbol{z^Tz - 2\beta^TX^Tz + \beta^TX^TX\beta}$, we differentiate this to obtian,

$$\begin{aligned}\frac{\partial}{\partial\beta}\left(\boldsymbol{z^Tz - 2\beta^TX^Tz + \beta^TX^TX\beta}\right) &= \boldsymbol{-X^Tz + (X^TX)\beta} \\ &= 0 \\ &\Leftrightarrow \hat{\beta}^{\boldsymbol{OLS}} = \boldsymbol{(X^TX)^{-1}X^Tz}\end{aligned}$$

Assuming $\boldsymbol{X^TX}$ is invertible, we have the solution $\hat{\boldsymbol{\beta}}$ .The "hat" on β means that this is the optimal beta, this is the beta that minimizes the cost function.

2.2. **Ridge regression.**

Ridge regression shrinks the regression coefficients by imposing a penalty on their size. The ridge coefficients minimize the same cost function (1), but now we introduce a coefficient, λ, which is known as the penalty parameter or regularization parameter. The optimization problem can be written in norm form as

$$\parallel \boldsymbol{z} - \boldsymbol{X\beta} \parallel_2^2 + \lambda \parallel \boldsymbol{\beta} \parallel_2^2 \tag{3}$$

subject to the constraint

$$\parallel \boldsymbol{\beta} \parallel_2^2 = \sum_{i=0}^{p-1} \boldsymbol{\beta_i^2} \leq s$$

here we have $\lambda \geq 0$. Similarly to the case of ordinary least squares, we differentiate (3) and set it equal to zero, the optimal solution is then given as

$$\hat{\boldsymbol{\beta}}^{\boldsymbol{Ridge}} = \boldsymbol{(X^TX + \lambda I)^{-1}X^Tz}$$

where $\boldsymbol{I}$ is the $p \times p$ identity matrix. The ridge regression solution is again a linear function of $\boldsymbol{z}$. The solution adds a positive constant to the diagnal before inversion, this makes the matrix $\boldsymbol{(X^TX + \lambda I)}$ non singular even if $\boldsymbol{(X^TX)}$ is singular. This was the main motivation for ridge regression when it was first introduced in statistics.

2.3. **Lasso regression.**

Lasso stands for least absolute shrinkage and selection operator. The lasso is a shrinkage method, just like the ridge method but with subtle differences. The optimization problem can be written in norm form as,

$$\parallel \boldsymbol{z} - \boldsymbol{X\beta} \parallel_2^2 + \lambda \parallel \boldsymbol{\beta} \parallel_1 \tag{4}$$

subject to the constraint

$$\parallel \boldsymbol{\beta} \parallel_1 = \sum_{i=0}^{p-1} |\boldsymbol{\beta_i}| \leq t$$

Notice the similarity to the ridge regression problem (3), the L_2 ridge penalty $\sum_i^p \boldsymbol{\beta_i^2}$ is replaced by the L_1 lasso penalty $\sum_i^p |\boldsymbol{\beta_i}|$. This is again a constraied optimization problem, however lasso penalty makes the solutions nonlinear in the $\boldsymbol{z_i}$, and a quadratic programming algorithm is used to compute them. We cannot find an analytical solution for $\hat{\boldsymbol{\beta}}^{\boldsymbol{Lasso}}$

3. Data analysis

Now that we have described the three regression methods, we will apply them to our data set. The data set that have is called the training data, we divide our training data into two sets, *training data* and *test data.* This is an important step in machine learning algorithms. This is because we will use the training data, to train our model, which means that we will do the regression on the training data first. To check the accuracy of out model, we then apply the model to the test data. The model has already seen the training data, so it is easy to have an almost perfect fit to it, but the real challenge occurs when we apply the model to our test data. There is no set rule on how to divide our data, a general rule of thumb is to have approximately 80% training and 20& test data. This leads to the discussion of model slelection, and the bias-variance tradeoff. To measure the accuracy of our model, we introduce risk metrics.

3.1. **Risk metrics.** 1. The Mean Squared Error(MSE), is a risk metric corresponding to the expected value of the squared (quadratic) error or lost function, it is defined as

$$MSE(\hat{y}, \tilde{y}) = \frac{1}{n} \sum_{i=0}^{n-1} (y_i - \tilde{y}_i)^2$$

Ideally we would like to have a MSE of zero, the smaller the value, the better the fit.

2. The R^2 score function, if $\hat{y_i}$ is the predicted value of the $i-th$ sample and y_i is the corresponding true value, then the R^2 score is defined as

$$R^2(\hat{y}, \tilde{y}) = 1 - \frac{\sum_{i=0}^{n-1}(y_i - \tilde{y}_i)^2}{\sum_{i=0}^{n-1}(y_i - \bar{y})^2},$$

where we define the mean value of $\hat{y}$ as, $\bar{y}$

$$\bar{y} = \frac{1}{n} \sum_{i=0}^{n-1} y_i.$$

Ideally we would want the R^2 score to be as close to one as possible. A negative score implies that the fitting is innaccurate. As we saw earlier, in

order to get the best approximation of our model to our training data, we minimize the cost function, now we can see tat the cost function was the mean squared error, therefore the optimal solution $\hat{\boldsymbol{z}}$ is that which minimises the MSE of the real data and the data predicted by our model. The predicted data is then $\tilde{\boldsymbol{z}} = \boldsymbol{X\beta}$, we would typically measure the error on the test data.

3.2. **Resampling Techniques.** In order to improve the prediction capabilities our our model, we introduce resampling techniques. Resampling methods can help us assess our model, without having to collect new data. Resampling involves repeatedly drawing samples from a training set and refitting our model of interest on each sample, this will help us gain additional information about the fitted model. Usually the aim of our model is to make predictions on new data, resampling allows us to see how our model would perform on data that it has not yet trained on.

3.2.1. *The bootstrap.* Bootstrap is resampling with replacement. Suppose we have a set of training data $\boldsymbol{Z} = (z_1, z_2, ..., z_N)$, where $z_i = (x_i, y_i)$. The basic algorithm is,

- Randomly draw a new sample with replacement from the training data, draw samples the same size as the original training set, i.e, we have $z_1^* = (z_1^*, z_2^*, ..., z_n^*)$
- compute some estimate function, $\hat{\theta} = g((z_1^*, z_2^*, ..., z_n^*)$ thsi can be any quantiy, like the MSE, relative error, R2 score etc.
- This is done B times, producing B data sets.
- Compute the standard deviation,

$$std = \sigma = \sqrt{\frac{1}{B}\sum_{j=1}^{B}(\hat{\theta}_j - \bar{\theta})^2}$$

where the quantity $\bar{\theta}$ is given by

$$\bar{\theta} = \frac{1}{b}\sum_{j=1}^{B}\hat{\theta}_j$$

3.2.2. *Cross validation.* The algorithm of focus will be the k-fold cross validation, this is the simplest and most widely used method for estimating prediction error in cross validation. K-fold cross validation uses part of the available data to fit the model, and a different part to test it. Unlike the bootstrap, the k-fold CV has no overlap of data sets. The basic algorithm is,

- First split the data into K roughly equally sized part,i.e, divide the data into K equally sized subsamples.
- for the kth part of the data, we fit the model on the rest of the k-1 parts of the data. This means that the kth part is to be considered as the "test data", while the rest k-1 parts are to be considered as "training data"
- Fit the model on the k-1 parts of the data, and calculate the prediction error on the kth test data.
- Do this for $k = 1, 2, ..., K$ and combine the K estimates to find the total prediction error.

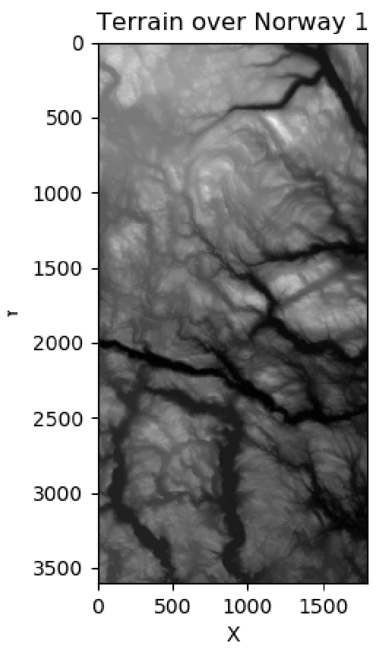
In this algorithm, each of the k subsamples have played the role of test data once, and training data K times. There is no set rule for chosing K, typically K is chosen to be between 5 and 10.

3.3. **The bias-variance tradeoff.** Now we are in a good position to discuss the bias variance trade off. Denote the training set by $\mathcal{T}$ with N samples, then we can calculate the expected prediction error at a point x_i for our model, we average over all samples of size N, since the problem is deterministic, this is the mean squared error(MSE) for estimating the function $f(\cdot)$ at a specific data point. The MSE can be written as

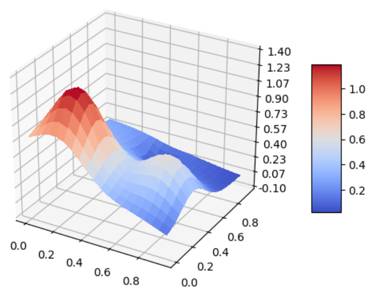
$$MSE = \mathbb{E}[(\boldsymbol{y} - \tilde{\boldsymbol{y}})^2]$$

Here the expected value $\mathbb{E}$ is the sample value. We can rewrite this as,

$$\mathbb{E}[(\boldsymbol{y} - \tilde{\boldsymbol{y}})^2] = \frac{1}{n}\sum_i(\boldsymbol{f_i} - \mathbb{E}[\tilde{\boldsymbol{y}}])^{\boldsymbol{2}} + \frac{1}{n}\sum_i(\tilde{\boldsymbol{y}}_{\boldsymbol{i}} - \mathbb{E}[\tilde{\boldsymbol{y}}])^{\boldsymbol{2}} + \sigma^2$$



(A) Terrain data



(B) Franke's function

FIGURE 1. Data sets in this article, terrain data and Franke's function.

We have broken down the MSE into two components, squared bias, and variance, plus an irreducible error as a result of the noise. Such a decmposition is known as the bias-variance decomposition.

The bias is given by

$$\frac{1}{n}\sum_i(\boldsymbol{f_i} - \mathbb{E}[\boldsymbol{\tilde{y}}])^{\mathbf{2}} \tag{5}$$

The variance is given by

$$\frac{1}{n}\sum_i(\boldsymbol{\tilde{y}_i} - \mathbb{E}[\boldsymbol{\tilde{y}}])^{\mathbf{2}} \tag{6}$$

To see how this expression is derived from the MSE, take a look at the appendix.

4. RESULTS

The data sets that we will study can be seen in figure 1. An important thing to note is that the data here has been scaled. Scaling of the data can have a large effect on the quality of the final solution. It is best to standardise all inputs to have mean zero and standard deviation one. This ensures that all inputs are treated equally and fit into a smaller range. I have used python's function standardscalar[1] to scale the data, this function standardizes features by removing the mean and scaling to unit variance.

[1]https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html

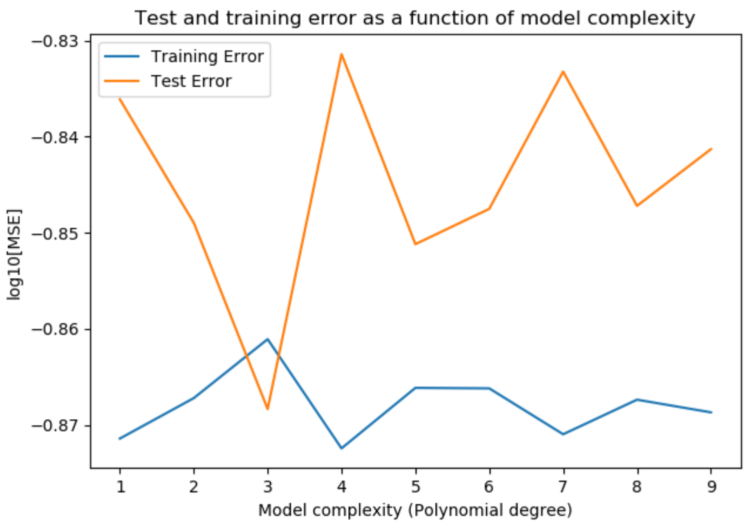
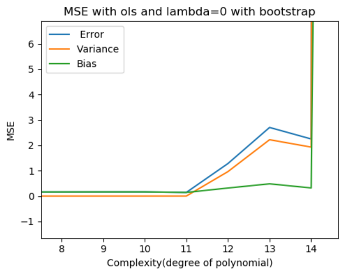
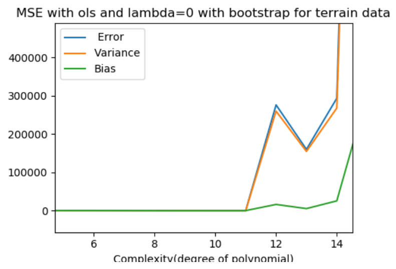


FIGURE 2. Test and training error as a function of model complexity for the Franke function, generated with n=6 samples, with the maximum polynomial degree equal to ten and with $\sigma^2 = 0.01$



(A) OLS regression on the Franke function



(B) OLS regression on the terrain data

FIGURE 3. Bias-Variance pots for the two data sets. This is produced with the bootstrap with a sample size of 100, with a maximum polynomial degree of 15 and with 10 bootstraps

4.1. **Bias Variance analysis for the data.** Let us first study the test and training error as a function of model complexity.

It is important to understand model prediction errors, bias and variance help us in doing that. Bias are the simplifying assumptions made by a model in order to make the target function easier to learn. Variance is the amount that the estimate of the target function will change if different training data was used. The bias term would most likely increase as the degree,d increases. The variance term would typically decrease as the inverse of the degree, d. So as the degree, d varies, we have a bias variance tradeoff. This tradeoff is not clearly visible in figure 3, this may because the maximum degree I used is 15, maybe increasing it would eventually show the theoretical results. From figure 3a, I can see that the ideal model would be to fit a polynomial of degree 11, as this is where the bias and variance are approximately zero. Figure 3b shows similar results.

Typically we would like to choose our model complexity to trade bias off with variance in such a way to minimize the test error.In this case model complexity refers to the degree of the polynomial we are trying to fit. As model complexity increases, the training error tends to decrease, that is, whenever we try to fit the data harder. However, with too much fitting, the model adapts itself too well to the training data, and will not generalize well. This means it will not be able to adapt itself well to the test data, which leads to a high test error. As model complexity increases, the model will capture the noise as well as the rest of the data too well, this is known as *overfitting.* Such models have low bias and high variance. In contrast, if the model is not complex enough, it will not be able to capture the data points well, which will lead to inaccurate results. This is known as *underfitting.* Such models have high bias and low variance. Figure 2 shows that as model complexity increases, the training error is lower than the test error. It seems like a polynomial of degree 3 is ideal, as this is where the test error is lower than the training error.

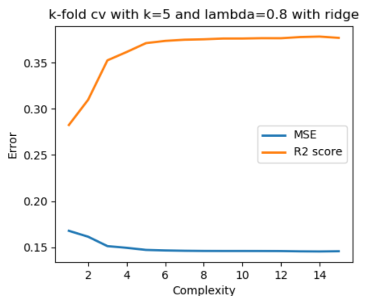
4.2. **Regression analysis.** In this section we present some result of the regression methods with the resampling. We discuss Ridge and Lasso regression in particular. These methods are shrinkage methods. This means they involve retaining a subset of the predictors and discarding the rest. Shrinkage methods are more continuous, and do not suffer as much from high variability. By varying the parameter λ we can reduce the number of degrees of freedom, and we can shrink the role of specific parameters.

In ridge regression, λ is use to minimize the effects of some features, while lasso will simply set the values of some features to zero. This can help the model in overfitting and underftting. Resampling will also improve the quality of our model.

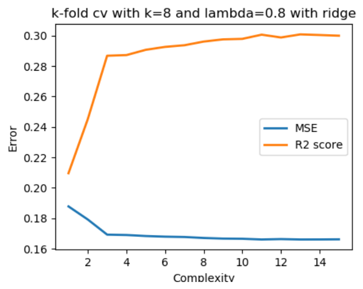
com

Figure 4 shows k-fold cross validation performed on the data generated from the franke function. The big question here is how to choose K? The figure shows results for three values of K. The MSE does decrease as K increases, and the R^2 score increases towards one, which is where we would want it to be. In figure 4, the difference of varying K is not dramatic, we can see that as K increases, the plotted MSE and R2 score become smoother, and tend to flatten out more. So as K increases, we would expect better results.

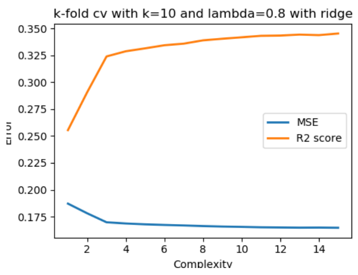
When it comes to lasso regression, figure 5 shows the effects of lasso with k-fold cross validation on the two data sets. Here I have chosen K=10. For the franke function, I can see that the R2 score is negative. This means that something has definitely wrong in my method of implementation, as a negative R2 score means that the results are unreliable. On the other hand, the terrain data shows an R2 score of zero, which is much better than having



(A) $K = 5$

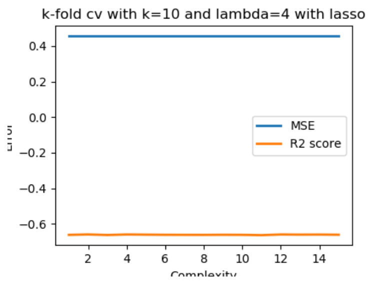


(B) $K = 8$

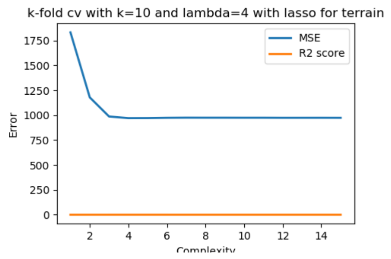


(C) $K = 10$

FIGURE 4. Ridge regression performed with K-fold cross validation on the franke function, for k=5,8 and 10. This is produced with a sample size of 100, and the maximum polynomial degree being 15 and $\lambda = 0.8$



(A) Results for franke function



(B) Results for terrain data

FIGURE 5. Lasso regression performed with K-fold cross validation with $K = 10$ on the franke function, and the terrain data. This is produced with a sample size of 100, and the maximum polynomial degree being 15 and $\lambda = 4$

a negative value. I can also see that the MSE decreases as the the model complexity increases.

Although resampling techniques help the overall results, the parameter λ plays an important role. Typically we would have to perform a grid search to find the optimal value of λ. Although I was unable to find the optimal value of λ, this can be done by for example, making a plot of λ vd the

MSE. We would choose some specific values of λ, in an interval, and then we would perform ridge or lasso regression with each of those values and find the MSE, after plotting λ vs MSE, the optimal value of λ would be the one which gives the lowest MSE.

5. Conclusion

The three regression methods are quite simple methods which help us see the important topics of machine learning. Ridge and Lasso regression are better methods than OLS, as the hyperparamter λ helps in minimizing the number of features, which in turn leads to better training by the model. In order to improve the results we could increase the smaple size, but this may not always be the case as most data is finite. Therefore resampling methods help to improve the results. No matter what model we choose, the bias-variance trade off will always be relevant. This represents the fundamental tension in machine learning, particularly supervised learning, between the complexity of a model and the amount of training data needed to train it. In order to minimize the expected test error, we need to select a statistical learning method that simultaneously achieves low variance and low bias. In practice, it is often useful to use a less-complex model with higher bias, that is a model whose asymptotic performance is worse than another model, because it is easier to train data less sensitive to sampling noise, which typically arises from having a finite-sized training data set.

6. References

- En.wikipedia.org. 2020. Bias–Variance Tradeoff. [online] Available at: `https://en.wikipedia.org/wiki/Bias%E2%80%93variance_tradeoff#In_regression` [Accessed 10 October 2020].
- Hastie, T., Tibshirani, R. and Friedman, J., 2004. The Elements Of Statistical Learning. New York: Springer, pp.chapter 2, 3, 7 and 10.
- Hjorth-Jensen, M., 2020. Overview Of Course Material: Data Analysis And Machine Learning (Weekly Schedule May Be Revised). [online] Compphysics.github.io. Available at: `https://compphysics.github.io/MachineLearning/doc/web/course.html` [Accessed 10 October 2020].
- Medium. 2020. Understanding The Bias-Variance Tradeoff. [online] Available at: `https://towardsdatascience.com/understanding-the-bias-variance-tra` [Accessed 10 October 2020].

Derivation of the bias and variance components of the MSE. Firstly we write the MSE in vector notation. Given $y = f + \epsilon$, we have that $\epsilon \sim \mathcal{N}(0, \sigma^2)$. So $\mathbb{E}[\epsilon] = 0$ and $Var(\epsilon) = 0$

$$\mathbb{E}[(\boldsymbol{y} - \tilde{\boldsymbol{y}})^{\boldsymbol{2}}] = \mathbb{E}[(\boldsymbol{f} + \boldsymbol{\epsilon} - \tilde{\boldsymbol{y}})^{\boldsymbol{2}}]$$

By adding and subtracting $\mathbb{E}[\tilde{\boldsymbol{y}}]$ we get,

$$\mathbb{E}[(\boldsymbol{y} - \tilde{\boldsymbol{y}})^{\boldsymbol{2}}] = \mathbb{E}[(\boldsymbol{f} + \boldsymbol{\epsilon} - \tilde{\boldsymbol{y}} + \mathbb{E}[\tilde{\boldsymbol{y}}] - \mathbb{E}[\tilde{\boldsymbol{y}}])^{\boldsymbol{2}}]$$

Expanding the brackets and calculating further we get,

$$\begin{aligned} = \mathbb{E}[(\tilde{\boldsymbol{y}} - \mathbb{E}[\tilde{\boldsymbol{y}})^{\boldsymbol{2}}] + \mathbb{E}[\boldsymbol{\epsilon}^{\boldsymbol{2}}] + \mathbb{E}[(\mathbb{E}[\tilde{\boldsymbol{y}}] - \tilde{\boldsymbol{y}})^{\boldsymbol{2}}] + 2\mathbb{E}[(\boldsymbol{f} - \mathbb{E}[\tilde{\boldsymbol{y}}])\boldsymbol{\epsilon}] \\ +2\mathbb{E}[\boldsymbol{\epsilon}(\mathbb{E}[\tilde{\boldsymbol{y}}] - \tilde{\boldsymbol{y}}])] + 2\mathbb{E}[(\mathbb{E}[\tilde{\boldsymbol{y}}] - \tilde{\boldsymbol{y}})(\boldsymbol{f} - \mathbb{E}[\tilde{\boldsymbol{y}}])] \end{aligned}$$

$$\begin{aligned} = (\tilde{\boldsymbol{y}} - \mathbb{E}[\tilde{\boldsymbol{y}})^{\boldsymbol{2}} + \mathbb{E}[\boldsymbol{\epsilon}^{\boldsymbol{2}}] \\ +\mathbb{E}[(\mathbb{E}[\tilde{\boldsymbol{y}}] - \tilde{\boldsymbol{y}})^{\boldsymbol{2}}] + 2[(\boldsymbol{f} - \mathbb{E}[\tilde{\boldsymbol{y}}])\mathbb{E}[\boldsymbol{\epsilon}] \\ +2\mathbb{E}[\boldsymbol{\epsilon}]\mathbb{E}[(\mathbb{E}[\tilde{\boldsymbol{y}}] - \tilde{\boldsymbol{y}})] \\ +2\mathbb{E}[(\mathbb{E}[\tilde{\boldsymbol{y}}] - \tilde{\boldsymbol{y}}](\boldsymbol{f} - \mathbb{E}[\tilde{\boldsymbol{y}}]) \end{aligned}$$

This is equal to

$$= (\tilde{\boldsymbol{y}} - \mathbb{E}[\tilde{\boldsymbol{y}})^{\boldsymbol{2}} + \mathbb{E}[(\mathbb{E}[\tilde{\boldsymbol{y}}] - \tilde{\boldsymbol{y}})^{\boldsymbol{2}}] + \mathbb{E}[\boldsymbol{\epsilon}^{\boldsymbol{2}}]$$

Taking the sum over all is we get

$$\mathbb{E}[(\boldsymbol{y} - \tilde{\boldsymbol{y}})^2] = \frac{1}{n}\sum_i (\boldsymbol{f_i} - \mathbb{E}[\tilde{\boldsymbol{y}}])^{\boldsymbol{2}} + \frac{1}{n}\sum_i (\tilde{\boldsymbol{y}}_{\boldsymbol{i}} - \mathbb{E}[\tilde{\boldsymbol{y}}])^{\boldsymbol{2}} + \sigma^2$$