CHAPTER 1

Abstract

In the world of data analysis, the quest for accurate models is a continual challenge. Central to this pursuit is the subtle balance between bias and variance, a core concept in predictive modeling. In this article, we explore this balance by examining three key regression techniques: Ordinary Least Squares (OLS), Ridge Regression, and Lasso Regression. Our goal extends beyond theory; we aim to uncover practical insights, particularly within the domain of geographical data analysis. Throughout this article, we investigating how OLS, Ridge Regression, and Lasso Regression handle geographical data. Through simulations, we explore the bias-variance trade-off and delve into the differences between OLS, Ridge Regression, and Lasso Regression in general.To enhance our understanding, we utilize Singular Value Decomposition (SVD). Our analysis of model complexity involves a detailed examination of OLS, Ridge, and Lasso, shedding light on their fundamental differences.

Contents

CHAPTER 2

Introduction

In this exploration, we delve into three fundamental regression techniques: Ordinary Least Squares (OLS), Ridge Regression, and Lasso Regression.We will We will begin with a foundational mathematical overview of each method, which will lay the groundwork for our subsequent analyses. In addition we will go through important parts of the theory of Min-Max scaling,,scaling,Moore Penrose Pseudo-inverse and different types of model assessment from the conventional train-test split to Bootstrap and Cross-validation.These concepts will be used to construct our simulation study.

In addition we will look at some of the important part of our implementation procedure,before diving into practical applications, we will consider a fundamental concept: the bias-variance tradeoff. This concept portrays the intricate balance between a model's ability to capture subtle data patterns and its vulnerability to fluctuations.

The last part of our theory will contain information on the metrics we will use,namely the Mean Squared Error(MSE) and the $R^2 - score$ to evaluate the model performances.

Transitioning from theory to application, we conduct experiments using geographical datasets. These experiments, allowing us to assess how OLS, Ridge, and Lasso Regression perform on geographical data,by testing it on a specific data set of a geographical area of Norway. Graphical representations,will give use the perspective on predictive accuracies and behaviors of the regression models.

Further we will have a discussion part, where we reflect on our key findings, with the help of presented theory,before summing up our journey with a conclusion.

Implementation: https://github.com/gituser1234566/some_coding/project1fys-stk4155

CHAPTER 3

Theory

3.1 Regression

Ordinary Linear Regression, Ridge Regression, and Lasso Regression are common techniques of linear regression. Regardless of the polynomial degree, it is termed linear since it is linear in the parameters. In this section, we will explore the nuances of these regression techniques, shedding light on their distinctive characteristics and use cases.

When modeling from data, our goal is to find a function that, when applied to our selected features, returns an output estimating the true response we are trying to model, denoted as Y. We aim to model Y through a function of features, i.e., $Y = f(X)$, where X is a list of features. Since our perception of reality is non-deterministic due to limited information, we assume the true relation as $Y = f(X) + \epsilon$, where ϵ represents random noise from some distribution. In linear regression, this noise is assumed to be Normally distributed with mean 0 and standard deviation σ. The true model for Linear Regression can be expressed as:

$$\epsilon \sim N(0, \sigma^2) \tag{3.1}$$
$$Y = X\beta + \epsilon \tag{3.2}$$

We can summarize what we can predict and what we cannot by calculating the expected value $E[Y]$ and the variance $Var[Y]$:

$$\epsilon \sim N(0, \sigma^2) \tag{3.3}$$

$$\begin{aligned} E[Y] &= E[f(X) + \epsilon] && (3.4)\\ &= E[f(X)] + E[\epsilon] && (3.5)\\ &= E[f(X)] + 0 && (3.6)\\ &= X\beta && (3.7) \end{aligned}$$

$$Var[Y] = Var[f(X) + \epsilon] \tag{3.8}$$

$$= Var[f(X)] + Var[\epsilon] \tag{3.9}$$
$$= 0 + Var[\epsilon] \tag{3.10}$$
$$= \sigma^2 \tag{3.11}$$

This leads to our main objective,which is to find a function $f(X)$ that, at each output point, is as close to Y as possible for every location in the space of X. This is achieved by minimizing the square of the $L2$-norm of the difference between Y and $f(X)$. This procedure is known as the least square method.

Objective Function:

$$L(\beta) = ||Y - X\beta||_2^2 \tag{3.12}$$

Minimization Estimator: To find the minimization estimator, we differentiate the objective function with respect to β and set it equal to zero:

$$\nabla L(\beta) = -X^T(Y - X\beta) = 0 \tag{3.13}$$
$$X^T(Y - X\beta) = \beta \tag{3.14}$$
$$X^TY - X^TX\beta = \beta \tag{3.15}$$
$$X^TX\beta = X^TY \tag{3.16}$$
$$(X^TX)\beta = X^TY \tag{3.17}$$

Solving for β gives the Ordinary Linear Regression minimization estimator:

$$\hat{\beta}_{\text{OLS}} = (X^TX)^{-1}X^TY \tag{3.18}$$

From the Gauss-Markov Theorem, we have that:

Theorem 3.1.1 (Gauss-Markov Theorem). *Under the assumptions of linearity, zero mean of errors, no perfect multicollinearity, homoscedasticity, no autocorrelation, and no endogeneity, the Ordinary Least Squares (OLS) estimator $\hat{\beta}_{OLS}$ is the Best Linear Unbiased Estimator of the parameters β among all linear unbiased estimators. In other words,*

$$Var(\hat{\beta}_{OLS}) \leq Var(\hat{\beta})$$

for any other linear unbiased estimator $\hat{\beta}$.

Why would we then consider other types of linear regression methods?.The motivation comes from the practicalities of applying the theory,where we do not have enough data to approximate the response for all its values.In addition we also have to take into account the random noise that links what we can model and the true response.Our main goal is not only to minimize the cost function for a single dataset but to find a model that is optimal for most datasets.

Ridge Regression is one such regressions method, which adds regularization to the β's by introducing an L2 norm penalty on the parameters in the regression equation. This penalty discourages overly complex models by shrinking the regression coefficients, especially those associated with less influential predictors, towards zero. Ridge Regression is useful when dealing with datasets containing a high number of predictors and multicollinearity. It achieves this by adding λ to every element in the diagonal of X^TX, resulting in a matrix that is invertible.

Lasso Regression, another regularization technique, employs an L1 norm penalty term. Unlike Ridge Regression, Lasso's L1 penalty encourages sparsity by setting certain regression coefficients to exactly zero. This feature makes it valuable in situations where feature selection is crucial. Lasso helps identify and retain the most relevant predictors, simplifying the model and enhancing interpretability.

In the next subsections, we will delve into the differences between these types of regressions. By understanding how these methods handle complexity and feature selection, we can make informed decisions about which technique best suits a given dataset or problem domain.

Differences Between OLS and Ridge Regression

In the previous section, we derived the least-squared solution for Ordinary Linear Regression. To compare Ordinary Least Squares (OLS) with the Ridge Regression solution, we follow a similar derivation for Ridge Regression.

Objective Function:

$$L(\beta) = ||Y - X\beta||_2^2 + \alpha||\beta||_2^2 \tag{3.19}$$

Minimization Estimator: Differentiating the objective function with respect to β and setting it to zero, we get:

$$\nabla L(\beta) = -X^T(Y - X\beta) + \alpha\beta = 0 \tag{3.20}$$

$$(X^TX + \alpha I)\beta = X^TY \tag{3.21}$$

Solving for β, the Ridge Regression minimization estimator is:

$$\hat{\beta}_{\text{ridge}} = (X^TX + \alpha I)^{-1}X^TY \tag{3.22}$$

When comparing Equation 3.15 with Equation 3.9, the only difference lies in the term λI. Adding λ to each diagonal element addresses issues like collinearity, aiding in the inversion of X^TX.

To understand the differences between OLS and Ridge Regression, we perform Singular Value Decomposition (SVD) on X and substitute it into Equations 3.9 and 3.15.

$$X(X^TX)^{-1}X^Ty = \left(U\Sigma V^T\right)\left(\left(U\Sigma V^T\right)^T\left(U\Sigma V^T\right) + \lambda I\right)^{-1}\left(U\Sigma V^T\right)^T y \tag{3.23}$$

Solving, we get:

$$X\hat{\beta}_{\text{ridge}} = U\Sigma\left(\Sigma^T\Sigma + \lambda I\right)^{-1}\Sigma^T U^T y \tag{3.24}$$

$$= \sum_{j=1}^{p} \mathbf{u}_j^T \mathbf{u}_j \left(\frac{\sigma_j^2}{\sigma_j^2 + \lambda}\right) y \tag{3.25}$$

From the derivation in Section 6.1,SVD decomposing the OLS Regression estimate gives the following result:

$$X(X^T X)^{-1} X^T y == UU^T y = \sum_{j=1}^{p} u_j u_j^T y \tag{3.26}$$

In Equation 3.24, $U^T y$ represents y coordinates with respect to orthogonal bases of X. Comparing Equations 3.26 with 3.25, Ridge Regression shrinks the orthogonal basis vectors corresponding to features in X based on their singular values, effectively handling directions with small variance.By performing singular value decomposition on the sample covariance matrix $\frac{X^T X}{n} = V^T \Sigma^2 V$, it becomes evident that the variance varies depending on the magnitude of Σ^2. Specifically, when $\Sigma_{i,i}^2$ is larger, the variance in $\frac{x_{i,i} x_{i,i}}{n}$ is higher. Ridge regression effectively reduces the influence of dimensions in the column space of X characterized by small singular values, essentially shrinking the orthogonal basis in directions with limited variance.

For orthogonal design matrices ($X^T X = I$), the comparison between Ridge and OLS is illuminating. In this case:

$$\beta^{\text{OLS}} = X^T y \tag{3.27}$$

$$\beta^{\text{Ridge}} = (I + \lambda I)^{-1} X^T y = (1 + \lambda)^{-1} \beta^{\text{OLS}} \tag{3.28}$$

Differences Between Ridge Regression and Lasso

When the number of features equals the number of observations ($p = n$) and the design matrix X becomes orthogonal ($X = I$), the distinctions between Ridge and Lasso become evident.

Ridge Regression Objective Function:

$$C(\beta) = \sum_i (y_i - x_i\beta_i)^2 + \lambda \sum_i \beta_i^2 \tag{3.29}$$

In Ridge regression, the derivative ($\frac{d}{d\beta}C(\beta) = 0$) scales each y_i term by $\frac{1}{1+\lambda}$.

Lasso Regression Objective Function:

$$C(\beta) = \sum_i (y_i - x_i\beta_i)^2 + \lambda \sum_i |\beta_i| \tag{3.30}$$

The derivative ($\frac{d}{d\beta}C(\beta) = 0$) results in a piecewise solution for $\beta^{\hat{\text{Lasso}}}$ based on λ:

$$\hat{\beta^{\text{Lasso}}} = \begin{cases} y_i - \frac{\lambda}{2}, & \text{if } y < \frac{\lambda}{2} \\ 0, & \text{if } y = \frac{\lambda}{2} \\ y_i + \frac{\lambda}{2}, & \text{if } y > \frac{\lambda}{2} \end{cases}$$

Lasso regression enforces strict sparsity, meaning it can force certain parameters to be precisely zero based on the chosen λ. In contrast, Ridge regression gradually reduces coefficients towards zero in a theoretical sense but does not enforce strict sparsity.

Bias and Variance Comparison

An important concept when it comes to model assessment of notion of an unbiased estimate. An estimate $\hat{\theta}$ is considered unbiased if $\theta = E[\theta]$, which is analogous to stating that $\theta - E[\theta] = 0$.

We can now check whether OLS estimate and Ridge estimate is unbiased by checking whether $X\beta_{OLS} = E[X\hat{\beta_{OLS}}]$, we examine the expectations:

$$\begin{aligned} E[\hat{\beta}^{\text{OLS}}] &= E[(X^TX)^{-1}X^Ty] && (3.31) \\ &= (X^TX)^{-1}X^TE[y] && (3.32) \\ &= \beta && (3.33) \end{aligned}$$

For Ridge regression, confirm whether $E[X\hat{\beta}^{\text{Ridge}}] = \beta$:

$$E[X\hat{\beta}^{\text{Ridge}}] = (XX^TX + \lambda I_{pp})^{-1}X^TX\beta \tag{3.34}$$

We see that the ridge estimate is biased.
Further we can compute the variance for OLS estimate:

$$\text{Var}[X\hat{\beta}^{\text{OLS}}] = \sigma^2(X^TX)^{-1} \tag{3.35}$$

While the variance for Ridge estimate is:

$$\text{Var}[X\hat{\beta}^{\text{Ridge}}] = \sigma^2(X^TX + \lambda I_{pp})^{-1}X^TX((X^TX + \lambda I_{pp})^{-1})^T \tag{3.36}$$

For $\lambda > 0$, it holds true that:

$$\text{Var}[X\hat{\beta}^{\text{Ridge}}] < \text{Var}[X\hat{\beta}^{\text{OLS}}] \tag{3.37}$$

This calculation demonstrates that by choosing a biased estimate, we can reduce the sum of $Bias^2$ and $Variance$, highlighting the advantage of introducing the $L2$ penalty term in minimization under specific circumstances.

Illustration of OLS Regression, Ridge Regression, and Lasso Regression

In this illustration, we fit a 10th order polynomial using OLS Regression, Ridge, and Lasso to the data. The effect of the tuning parameter becomes evident.

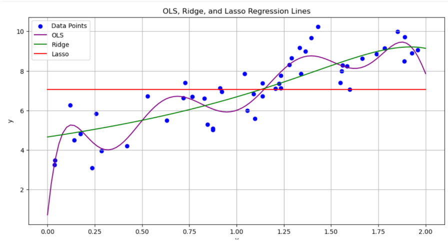


Figure 3.1: Tuning parameter $\lambda = 10$

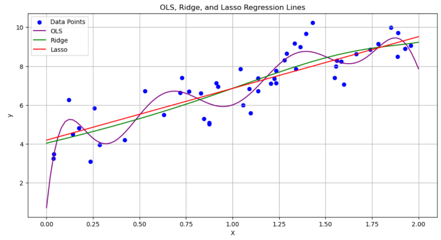


Figure 3.2: Tuning parameter $\lambda = 0.1$

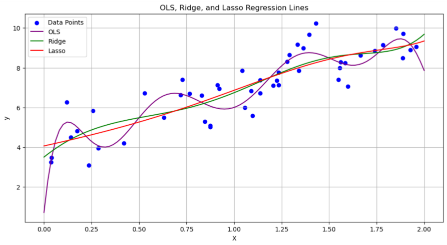


Figure 3.3: Tuning parameter $\lambda = 0.0001$

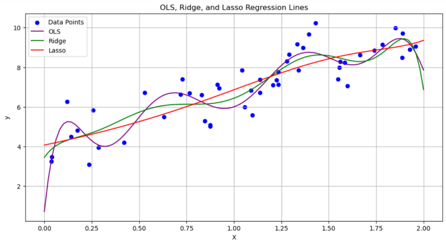


Figure 3.4: Tuning parameter $\lambda = 0.0000001$

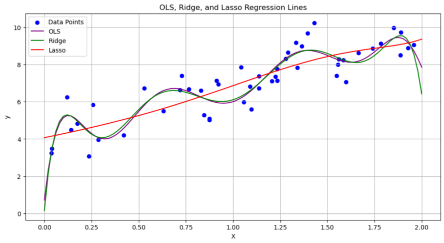


Figure 3.5: Tuning parameter $\lambda = 0.0000000001$

Here, when $\lambda = 10$, Lasso shrinks all parameters to zero, while Ridge does not. As λ decreases, the impact of λ on penalization becomes more apparent. In theory, as $\lambda \to 0$, no penalization occurs, and as $\lambda \to \infty$, all parameters are heavily penalized. However, even small λ values lead to considerable penalization, especially for Lasso, which uses the $L1$ norm.

Min-Max Scaling

In any model prediction problem the task of finding an appropriate scaling function that could be applied to unscaled data is important.Our simulation study will focus on geographical landscape data, which often have variations of height data for different coordinates often leading to significant differences in values. Such disparities can create challenges, especially when the dataset exhibits high variance. ’

Min-Max scaling transforms features lying in the interval $[a, b]$, to $[0, 1]$,while preserving inter-variable relationships.The Min-Max Scaler is an appropriate scaling approach when the relative relationship of the response instances is relevant for the prediction.One of the reasons for using this scaling is to prevent potential numerical overflow, pivotal for precise calculations.In our case we are dealing will deal with polynomial features,If the initial features have large values,will lead to even higher values for the other columns in the design matrix depending on the complexity.Using Min-Max scaling could in this situation lead to underflow.It is therefore important to be aware about the potential trade-off’s,and choose the type of scaling depending on the data-set at hand.

Centering the Data

Centering the data has a crucial role as a preprocessing step, especially when dealing with regularization techniques like Ridge and Lasso regression. The challenge arises from the inclusion of the intercept term in these regression methods, leading to uneven penalization. Unlike other coefficients that represent features, the intercept typically remains unperturbed as it signifies the mean response when all features are zero. This unique treatment introduces distortion, complicating the accurate assessment of feature significance.

This uneven penalization undermines the model’s interpretability, particularly concerning features correlated with the intercept, which may receive disproportionately lower penalties, potentially inflating their coefficients. To counter this, centering the data prior to applying regularization ensures that the intercept now represents the mean of the response variable. This approach rectifies the problem of uneven penalization, enabling regularization techniques to work uniformly across all features. Consequently, it facilitates a fair evaluation of feature importance, enhancing clarity and precision in interpreting the model's results.

Moore's Penrose Pseudo-Inverse

In the closed form solution of OLS Regression and Ridge Regression we will need to perform matrix inversion.In cases where the matrix is singular we need an estimate of the closest inverse to the singular matrix.The approximation techneque we will use is the Moore's Penrose Pseudo-Inverse.The Moore's Penrose Pseudo-Inverse offering a robust method for computing an approximate inverse of singular matrices. The process initiates with the Singular Value Decomposition (SVD) of the original matrix A, where $A = U\Sigma V^T$. Here, U and V represent orthogonal matrices, while Σ is a diagonal matrix housing the singular values of A.

The pseudo-inverse A^+ is then calculated based on the components obtained from SVD. If A is square and invertible ($m = n$), A^+ aligns with the standard matrix inverse ($A^+ = A^{-1}$). For cases where A has more rows than columns ($m > n$), A^+ follows $A^+ = V\Sigma^+U^T$, utilizing Σ^+, (where Σ^+ is the part of Σ with positive eigenvalues), as the pseudo-inverse of Σ. Similarly, if A possesses more columns than rows ($m < n$), A^+ is computed as $A^+ = U\Sigma^+V^T$. The Moore's Penrose Pseudo-Inverse proves invaluable in solving linear systems of equations and addressing situations where a true inverse might not exist.

Model Assessment

One of the fundamental practices in machine learning is to assess how well a model generalizes to unseen data. To achieve this, the dataset is typically split into training and test sets.

One common problem called overfitting, arises when a model becomes too attuned to the intricacies of the training data, capturing noise instead of genuine patterns. To detect overfitting, evaluating the model on a separate test set is crucial. An overfitted model excels on training data but falters on the test data due to its inability to generalize well. Conversely, an underfit model oversimplifies the data, leading to high training errors and poor performance, highlighting the significance of choosing appropriate algorithms and model complexities.

Maintaining the integrity of test results necessitates keeping the test set distinct from the training set. If the same data were used for both training and testing, the model might "memorize" specific examples, resulting in an overly optimistic evaluation. However, this approach provides only a suboptimal approximation of the generalized prediction error ,i.e the expected test error. More robust methods, such as Bootstrap and Cross-validation, offer enhanced insights into an algorithm's performance.

Bootstrap

In the domain of algorithm assessment, Bootstrap estimation emerges as a robust technique to compute the generalized prediction error. By generating diverse resampled datasets through random sampling with replacement, Bootstrap provides a holistic view of an algorithm's performance.

Multiple resampled datasets are created, and the algorithm is applied to each. These estimations are then averaged, resulting in a more stable and reliable evaluation. Bootstrap estimation is invaluable for understanding how an algorithm performs under varying conditions and data subsets, ensuring its real-world adaptability.

Cross-validation

Cross-validation is another technique which is intended to calculate the generalized prediction error.

The process involves dividing the available dataset into K subsets or folds. In each iteration, one of these folds is set aside as the validation set, while the model is trained on the remaining $K-1$ folds. This training-validation cycle is repeated K times, ensuring that every data point is used for both training and validation. The model's performance is assessed using various metrics, such as accuracy or mean squared error, on each validation set. The obtained metrics are then averaged over all iterations to provide a comprehensive evaluation score for the model. Cross-validation is particularly beneficial in scenarios with limited data, ensuring a robust estimation of the model's generalizability.

Evaluation metrics

(R^2) and the Mean Squared Error (MSE) is our chosen metrics for assessing the efficacy of regression models. R^2 measures the extent to which the model can explain the variability observed in the dependent variable. Ranging between 0 and 1, an R^2 value closer to 1 indicates a model that accurately captures the data's variance, while a value closer to 0 suggests a poor fit, implying that the model does not explain much of the variability. It is calculated as $R^2 = 1 - \frac{SSR}{SST}$, where SSR represents the sum of squared residuals , and SST represents the sum of squared total (the total variability with respect to the mean of the dependent variable).

On the other hand, MSE quantifies the average squared difference between the model's predictions and the actual values, formally defined as:

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

where n is the number of data points, y_i is the actual value, and $\hat{y}_i$ is the predicted value for data point i. A lower MSE indicates a better fit, as it reflects smaller errors between predicted and actual values. The relationship between R^2 and MSE could be understood as: as R^2 approaches 1, indicating a model that explains most of the variance, the residuals, which contribute to MSE, tend to be smaller. Models should be evaluated holistically, considering both R^2 and MSE, along with other metrics, to ensure accurate and meaningful interpretation of the regression results.

3.2 Bias-variance tradeoff and Model Assessment

In the context of model assessment, the bias-variance tradeoff provides valuable insights into the performance of machine learning models. The tradeoff involves decomposing the mean squared error ($E[(y - \tilde{y})^2]$) into its constituent components, which are essential for evaluating the model's predictive capabilities. Here, y represents a new data point from the true distribution, and $\tilde{y}$ represents the model's prediction.

The mean squared error can be expressed as the double expectation of the squared difference between the actual (y) and predicted ($\tilde{y}$) values.The inner expectation is conditioned with a specific training set and the outer expectation takes the expectation variyng over all training sets:

$$E[(y - \tilde{y})^2] = E[E[(y - \tilde{y})^2|x]] \tag{3.38}$$

By looking at this decomposition we can understand what the method of bootstrap and cross validation is trying to estimate.When we continue our decomposition we get an even better intuition of the generalized prediction error.From 6.2 we get:

$$E[(y - \tilde{y})^2] = bias(\tilde{y})^2 + var(\tilde{y}) + \sigma_y^2 \tag{3.39}$$

This equation indicates that the overall error is the sum of three fundamental components: $bias(\tilde{y})^2$, $var(\tilde{y})$, and σ_y^2. Here's how these components align with the context of model assessment:

1. Bias ($bias(\tilde{y})^2$): Bias measures how close the model's predictions ($\tilde{y}$) are to the true response y. In the context of model assessment, bias reflects how well the model approximates the underlying relationships in the training data.

2. Variance ($var(\tilde{y})$): Variance quantifies the sensitivity of the model to changes in input. High variance indicates that small fluctuations in the training data leading to significant changes in predictions. While low bias models tend to have high variance.

3. Randomness (σ_y^2): This term represents the inherent randomness in the response variable. In practical scenarios, real-world data often contains unpredictable elements. In the context of model assessment, randomness highlights the challenges of accurately modeling natural variability.

When evaluating a model's performance, the goal is to find the optimal balance between bias and variance. Lowering bias typically increases variance and vice versa. The tradeoff is essential because overly complex models may fit the training data perfectly (low bias) but fail to generalize to new, unseen data (high variance).

When splitting the data into test and training set we compute the test error, representing a point estimate of the generalized prediction error $(E[E[(Y - \tilde{Y})^2|X]])$. However, when data is limited, traditional methods might not provide reliable estimates.

This limitation leads to the introduction of robust model assessment techniques discussed above like Bootstrap and Cross-validation. These methods offer more accurate approximations of the generalized prediction error by simulating various datasets, ensuring the model's performance is thoroughly evaluated across diverse scenarios.

3.3 Implementation

Our implementation of regression fitting will be illustrated with the standard setup outlined in Algorithm 1. The pseudocode is formulated using an approximation of scikit-learn syntax.

Algorithm 1. Prediction algorithm

```
Min-Max Scaling:

- Apply Min-Max scaling to features $X$:

  from sklearn.preprocessing import MinMaxScaler
  scaler_X = MinMaxScaler()
  X_scaled = scaler_X.fit_transform(X)

- Apply Min-Max scaling to target values $y$:

  from sklearn.preprocessing import MinMaxScaler
  scaler_y = MinMaxScaler()
  y_scaled = scaler_y.fit_transform(y.reshape(-1, 1)).flatten()

- Split Data:
  $X\_train, X\_test, y\_train, y\_test \leftarrow$ train_test_split$(X, y, test\_size = 0.2)$
  Loop over Polynomial Degrees:
  for $degree$ in $[1, 2, \ldots,$ max_degree$]$ :
  for $\lambda$ in $[tuning_parameter\lambda]$ :

-       Transform Features:
  $X\_train\_poly \leftarrow$ PolynomialFeatures$(degree, include\ \ bias = False, Center = True).fit\_transform(X\_train)$
  $X\_test\_poly \leftarrow$ PolynomialFeatures$(degree, include\ \ bias = False, Center = True).transform(X\_test)$
  $y\_train = y\_trainmean\_y\_train$

-       Train Linear Regression Model:
  $model \leftarrow$ Ridge$(\alpha = \lambda, fit_intercept = True)$
  $model$.fit$(X\_train\_poly, y\_train)$

-       Make Predictions:
  $predictions \leftarrow model$.predict$(X\_test\_poly)$
  Calculate Mean Squared Error:
  $error \leftarrow$ mean_squared_error$(y\_test, predictions)$
```

We start by Min-Max-scaling for the reasons explained in the Min-Max sectiona above.Next,we use the `PolynomialFeatures` function generates a design matrix corresponding to the polynomial degree. In this step, we exclude the intercept and center each column in both our train and test design matrices by using the mean of the corresponding column from the train design matrix Additionally, we center y_{train} by subtracting its mean. The Ridge model is fitted using the explicit solution:

$$X_{\text{train}}(X_{\text{train}}^T X_{\text{train}}^T + \lambda I)^{-1} X_{\text{train}}^T y$$

For predictions on the test data, we employ the `model.predict` function. In scikit-learn, setting `fit_intercept=True` in $Ridge(fit_intercept = True)$ automatically adds the mean of y_{train}, allowing predictions on uncentered data. Although removing the intercept has no numerical implication for OLS Regression, as discussed in the data centering section, excluding the intercept is advantageous when fitting models using Lasso and Ridge techniques.Finally,we do our evaluation with chosen metrics.

Algorithm 2. Cross-Validation:

```
• Split data into $K$ folds.
• Initialize total train MSE and total test MSE to 0.
• For each fold:
    – Separate current fold as test set, rest as train set.
    – Train the model on the train set.
    – Predict on train set and calculate train MSE.
    – Predict on test set and calculate test MSE.
    – Add train MSE to total train MSE.
    – Add test MSE to total test MSE.
• Calculate average train MSE and average test MSE.
Return average train MSE, average test MSE
```

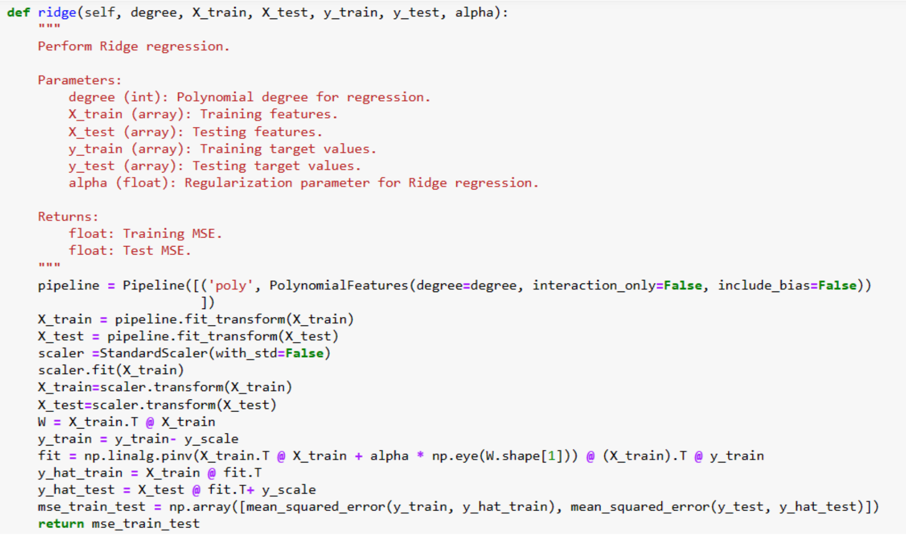
Algorithm 3. Bootstrap:

```
• Initialize total train MSE and total test MSE to 0.
• For $n$ iterations:
    – Sample data with replacement to create a bootstrap sample.
    – Split bootstrap sample into train and test sets.
    – Train the model on the train set.
    – Predict on train set and calculate train MSE.
    – Predict on test set and calculate test MSE.
    – Add train MSE to total train MSE.
    – Add test MSE to total test MSE.
• Calculate average train MSE and average test MSE.
Return average train MSE, average test MSE.
```

Algorithm 2 is a pseudo code of our implementation of Bootstrap while Algorithm 3 is the implementation for Cross Validation.

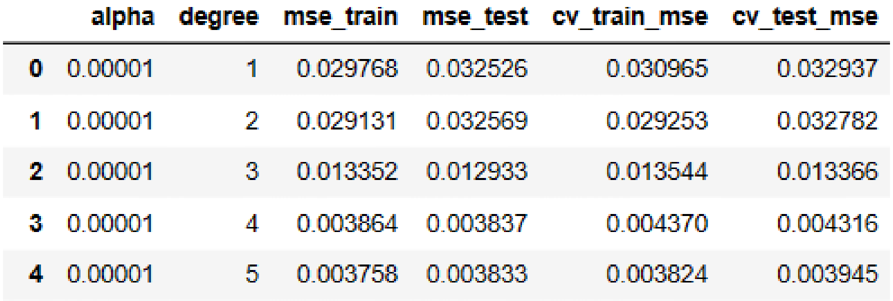
Skitlearn vs own implementation

Implementation of code in two different is a nice way to see if the code is doing what it is intended to do.The most important part of our implementation is the part of the code that estimate the predicted model.We therefore include the two different implementations in order to give some confidence for our implementation.



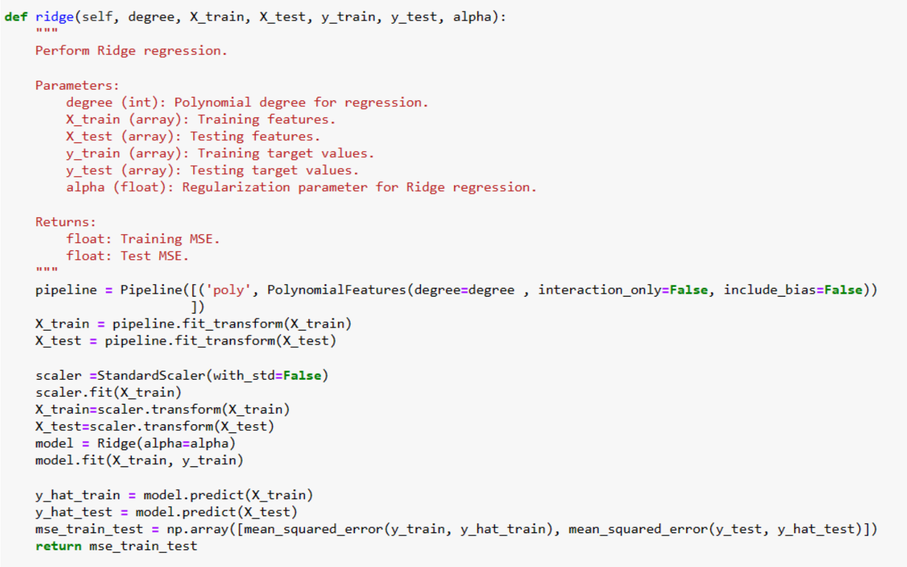
```
def ridge(self, degree, X_train, X_test, y_train, y_test, alpha):
    """
    Perform Ridge regression.

    Parameters:
        degree (int): Polynomial degree for regression.
        X_train (array): Training features.
        X_test (array): Testing features.
        y_train (array): Training target values.
        y_test (array): Testing target values.
        alpha (float): Regularization parameter for Ridge regression.

    Returns:
        float: Training MSE.
        float: Test MSE.
    """
    pipeline = Pipeline([('poly', PolynomialFeatures(degree=degree, interaction_only=False, include_bias=False))
                         ])
    X_train = pipeline.fit_transform(X_train)
    X_test = pipeline.fit_transform(X_test)
    scaler =StandardScaler(with_std=False)
    scaler.fit(X_train)
    X_train=scaler.transform(X_train)
    X_test=scaler.transform(X_test)
    W = X_train.T @ X_train
    y_train = y_train- y_scale
    fit = np.linalg.pinv(X_train.T @ X_train + alpha * np.eye(W.shape[1])) @ (X_train).T @ y_train
    y_hat_train = X_train @ fit.T
    y_hat_test = X_test @ fit.T+ y_scale
    mse_train_test = np.array([mean_squared_error(y_train, y_hat_train), mean_squared_error(y_test, y_hat_test)])
    return mse_train_test
```

Figure 3.6



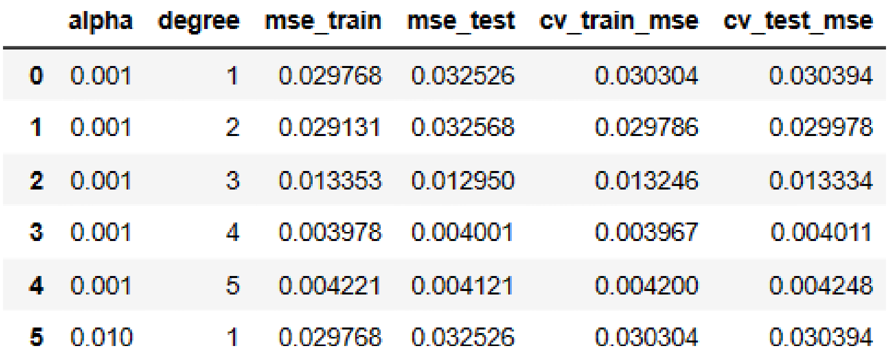
	alpha	degree	mse_train	mse_test	cv_train_mse	cv_test_mse
0	0.00001	1	0.029768	0.032526	0.030965	0.032937
1	0.00001	2	0.029131	0.032569	0.029253	0.032782
2	0.00001	3	0.013352	0.012933	0.013544	0.013366
3	0.00001	4	0.003864	0.003837	0.004370	0.004316
4	0.00001	5	0.003758	0.003833	0.003824	0.003945

Figure 3.7: Result for code fitting and predicting without skitlearn.Fist column in table shows tuning parameter alpha,second column shows polynomial complexity,third column shows training MSE and the fourth shows test MSE



```
def ridge(self, degree, X_train, X_test, y_train, y_test, alpha):
    """
    Perform Ridge regression.

    Parameters:
        degree (int): Polynomial degree for regression.
        X_train (array): Training features.
        X_test (array): Testing features.
        y_train (array): Training target values.
        y_test (array): Testing target values.
        alpha (float): Regularization parameter for Ridge regression.

    Returns:
        float: Training MSE.
        float: Test MSE.
    """
    pipeline = Pipeline([('poly', PolynomialFeatures(degree=degree , interaction_only=False, include_bias=False))
                        ])
    X_train = pipeline.fit_transform(X_train)
    X_test = pipeline.fit_transform(X_test)

    scaler =StandardScaler(with_std=False)
    scaler.fit(X_train)
    X_train=scaler.transform(X_train)
    X_test=scaler.transform(X_test)
    model = Ridge(alpha=alpha)
    model.fit(X_train, y_train)

    y_hat_train = model.predict(X_train)
    y_hat_test = model.predict(X_test)
    mse_train_test = np.array([mean_squared_error(y_train, y_hat_train), mean_squared_error(y_test, y_hat_test)])
    return mse_train_test
```

Figure 3.8: Code using skitlearn for fitting and predicting



	alpha	degree	mse_train	mse_test	cv_train_mse	cv_test_mse
0	0.001	1	0.029768	0.032526	0.030304	0.030394
1	0.001	2	0.029131	0.032568	0.029786	0.029978
2	0.001	3	0.013353	0.012950	0.013246	0.013334
3	0.001	4	0.003978	0.004001	0.003967	0.004011
4	0.001	5	0.004221	0.004121	0.004200	0.004248
5	0.010	1	0.029768	0.032526	0.030304	0.030394

Figure 3.9: Result for code fitting and predicting with skitlearn.Fist column in table shows tuning parameter alpha,second column shows polynomial complexity,third column shows training MSE and the fourth shows test MSE

CHAPTER 4

Simulation study

In this simulation study, we will use the theory above to do predictive modeling on geographical data sourced from one location in Norway.The Data set contains heights for given (x,y) coordinate pairs. Our exploration begins with the foundational models of Linear Regression, Lasso, and Ridge, forming the basis of our analysis.We will start by Min Max scaling the data together with there (x,y) pairs,before manually fitting OLS and Ridge using there explicit parameter solution and using Skitlearn for fitting Lasso.We use Skitlearn for lasso,since we do not have any nice explicit solution for Lasso. Lasso uses coordinate descent to approximate its parameter solution.We will have not introduced this method since the main perpose of the article is to get a high-level view on the differences between the regression methods,but it is worth mentioning. The analysis will be focused on varying the polynomial regression complexity up to degree five.For Lasso and Ridge regression, an additional layer of complexity is introduced by tuning the regularization parameter (lambda), allowing us to strike a balance between model complexity and sparsity. Through evaluation, of the Mean Squared Error (MSE) for both training and test datasets. Additionally, will plot the Bias-Variance tradeoff for OLS Regression, examining MSE, bias, and variance as functions of polynomial degree to see the relationship between model complexity and prediction accuracy for the dataset.

In addition we employ 5-fold, 10-fold cross-validation and Bootstrap. These techniques will give us a sense of how well our models does across diverse subsets of the data, providing insights into their adaptability to unseen observations.

We have used a noisy response to compute the results for all illustrated in the figure below.

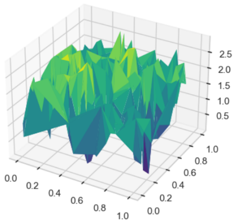


Figure 4.1: Landscape Data

4.1 Bias variance trade for Geographical Data

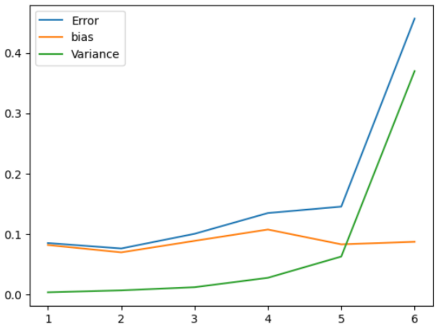
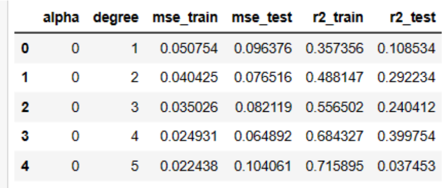


Figure 4.2: Bias-Variance data for Landscape data

4.2 Graphical Comparison of MSE and R^2



	alpha	degree	mse_train	mse_test	r2_train	r2_test
0	0	1	0.050754	0.096376	0.357356	0.108534
1	0	2	0.040425	0.076516	0.488147	0.292234
2	0	3	0.035026	0.082119	0.556502	0.240412
3	0	4	0.024931	0.064892	0.684327	0.399754
4	0	5	0.022438	0.104061	0.715895	0.037453

Figure 4.3: Table where Fist column in table shows tuning parameter alpha,second column shows polynomial complexity,third column shows training MSE and the fourth shows test MSE ,fourth column shows $R^2 - score$ for the training set and th fifth columns show $R^- score$ for the training set.

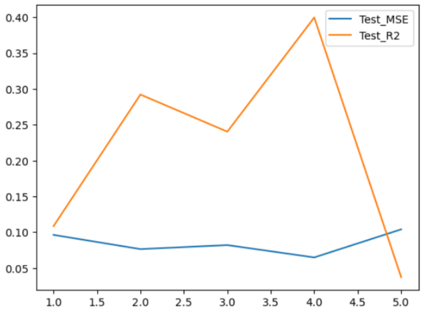
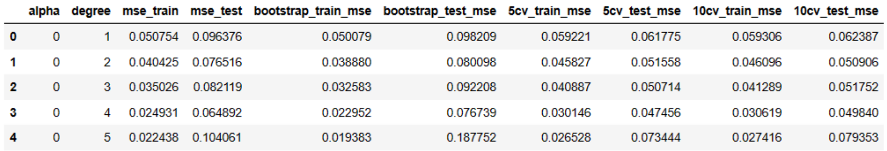


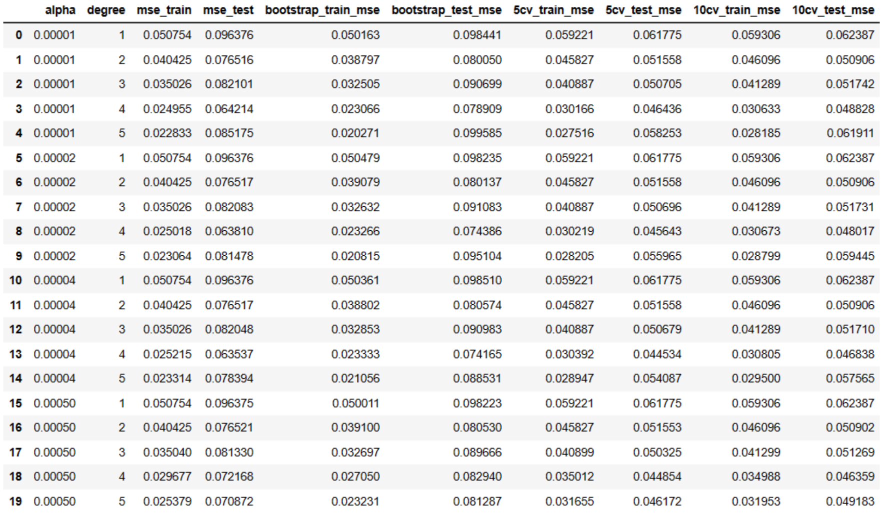
Figure 4.4: Plot showing R^2 variyng complexity in orange and the MSE variyng complexity in blue.X axis shos the polynomial complexity.

4.3 MSE results for OLS,Ridge,Lasso



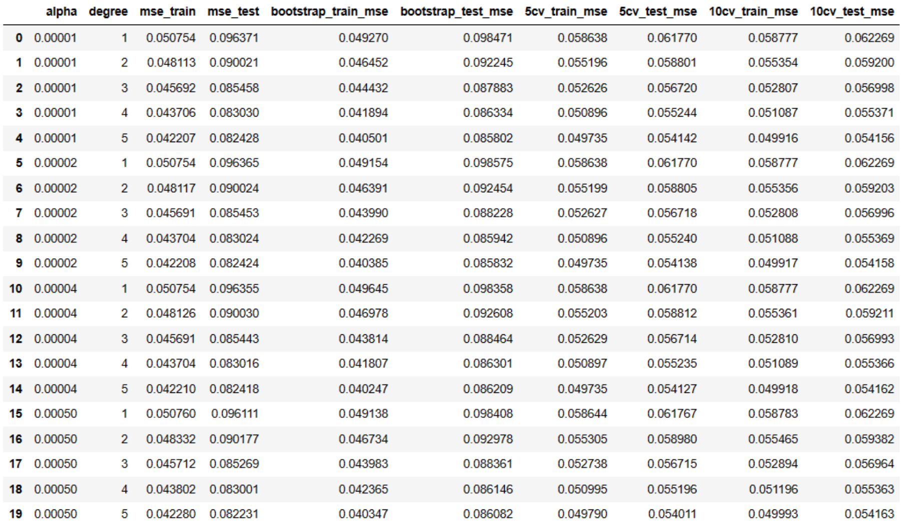
	alpha	degree	mse_train	mse_test	bootstrap_train_mse	bootstrap_test_mse	5cv_train_mse	5cv_test_mse	10cv_train_mse	10cv_test_mse
0	0	1	0.050754	0.096376	0.050079	0.098209	0.059221	0.061775	0.059306	0.062387
1	0	2	0.040425	0.076516	0.038880	0.080098	0.045827	0.051558	0.046096	0.050906
2	0	3	0.035026	0.082119	0.032583	0.092208	0.040887	0.050714	0.041289	0.051752
3	0	4	0.024931	0.064892	0.022952	0.076739	0.030146	0.047456	0.030619	0.049840
4	0	5	0.022438	0.104061	0.019383	0.187752	0.026528	0.073444	0.027416	0.079353

Figure 4.5: MSE results for OLS Regression.The first column shows the tuning parameter,the second column shows the polynomial complexity,the third column shows the train MSE the corresponding tuning parameter and complexity of polynomial and the fourth column shows the test MSE for the corresponding tuning parameter and complexity of polynomial.Sixth and seven column shows the result for training and test MSE for Bootstrap.Column eight and ninth shows the training and test MSE for 5-fold-cv and column 10th and 11th show the training and test MSE for 10-fold-cv



	alpha	degree	mse_train	mse_test	bootstrap_train_mse	bootstrap_test_mse	5cv_train_mse	5cv_test_mse	10cv_train_mse	10cv_test_mse
0	0.00001	1	0.050754	0.096376	0.050163	0.098441	0.059221	0.061775	0.059306	0.062387
1	0.00001	2	0.040425	0.076516	0.038797	0.080050	0.045827	0.051558	0.046096	0.050906
2	0.00001	3	0.035026	0.082101	0.032505	0.090699	0.040887	0.050705	0.041289	0.051742
3	0.00001	4	0.024955	0.064214	0.023066	0.078909	0.030166	0.046436	0.030633	0.048828
4	0.00001	5	0.022833	0.085175	0.020271	0.099585	0.027516	0.058253	0.028185	0.061911
5	0.00002	1	0.050754	0.096376	0.050479	0.098235	0.059221	0.061775	0.059306	0.062387
6	0.00002	2	0.040425	0.076517	0.039079	0.080137	0.045827	0.051558	0.046096	0.050906
7	0.00002	3	0.035026	0.082083	0.032632	0.091083	0.040887	0.050696	0.041289	0.051731
8	0.00002	4	0.025018	0.063810	0.023266	0.074386	0.030219	0.045643	0.030673	0.048017
9	0.00002	5	0.023064	0.081478	0.020815	0.095104	0.028205	0.055965	0.028799	0.059445
10	0.00004	1	0.050754	0.096376	0.050361	0.098510	0.059221	0.061775	0.059306	0.062387
11	0.00004	2	0.040425	0.076517	0.038802	0.080574	0.045827	0.051558	0.046096	0.050906
12	0.00004	3	0.035026	0.082048	0.032853	0.090983	0.040887	0.050679	0.041289	0.051710
13	0.00004	4	0.025215	0.063537	0.023333	0.074165	0.030392	0.044534	0.030805	0.046838
14	0.00004	5	0.023314	0.078394	0.021056	0.088531	0.028947	0.054087	0.029500	0.057565
15	0.00050	1	0.050754	0.096375	0.050011	0.098223	0.059221	0.061775	0.059306	0.062387
16	0.00050	2	0.040425	0.076521	0.039100	0.080530	0.045827	0.051553	0.046096	0.050902
17	0.00050	3	0.035040	0.081330	0.032697	0.089666	0.040899	0.050325	0.041299	0.051269
18	0.00050	4	0.029677	0.072168	0.027050	0.082940	0.035012	0.044854	0.034988	0.046359
19	0.00050	5	0.025379	0.070872	0.023231	0.081287	0.031655	0.046172	0.031953	0.049183

Figure 4.6: MSE for Ridge Regression.The first column shows the tuning parameter,the second column shows the polynomial complexity,the third column shows the train MSE the corresponding tuning parameter and complexity of polynomial and the fourth column shows the test MSE for the corresponding tuning parameter and complexity of polynomial.Sixth and seven column shows the result for training and test MSE for Bootstrap.Column eight and ninth shows the training and test MSE for 5-fold-cv and column 10th and 11th show the training and test MSE for 10-fold-cv



	alpha	degree	mse_train	mse_test	bootstrap_train_mse	bootstrap_test_mse	5cv_train_mse	5cv_test_mse	10cv_train_mse	10cv_test_mse
0	0.00001	1	0.050754	0.096371	0.049270	0.098471	0.058638	0.061770	0.058777	0.062269
1	0.00001	2	0.048113	0.090021	0.046452	0.092245	0.055196	0.058801	0.055354	0.059200
2	0.00001	3	0.045692	0.085458	0.044432	0.087883	0.052626	0.056720	0.052807	0.056998
3	0.00001	4	0.043706	0.083030	0.041894	0.086334	0.050896	0.055244	0.051087	0.055371
4	0.00001	5	0.042207	0.082428	0.040501	0.085802	0.049735	0.054142	0.049916	0.054156
5	0.00002	1	0.050754	0.096365	0.049154	0.098575	0.058638	0.061770	0.058777	0.062269
6	0.00002	2	0.048117	0.090024	0.046391	0.092454	0.055199	0.058805	0.055356	0.059203
7	0.00002	3	0.045691	0.085453	0.043990	0.088228	0.052627	0.056718	0.052808	0.056996
8	0.00002	4	0.043704	0.083024	0.042269	0.085942	0.050896	0.055240	0.051088	0.055369
9	0.00002	5	0.042208	0.082424	0.040385	0.085832	0.049735	0.054138	0.049917	0.054158
10	0.00004	1	0.050754	0.096355	0.049645	0.098358	0.058638	0.061770	0.058777	0.062269
11	0.00004	2	0.048126	0.090030	0.046978	0.092608	0.055203	0.058812	0.055361	0.059211
12	0.00004	3	0.045691	0.085443	0.043814	0.088464	0.052629	0.056714	0.052810	0.056993
13	0.00004	4	0.043704	0.083016	0.041807	0.086301	0.050897	0.055235	0.051089	0.055366
14	0.00004	5	0.042210	0.082418	0.040247	0.086209	0.049735	0.054127	0.049918	0.054162
15	0.00050	1	0.050760	0.096111	0.049138	0.098408	0.058644	0.061767	0.058783	0.062269
16	0.00050	2	0.048332	0.090177	0.046734	0.092978	0.055305	0.058980	0.055465	0.059382
17	0.00050	3	0.045712	0.085269	0.043983	0.088361	0.052738	0.056715	0.052894	0.056964
18	0.00050	4	0.043802	0.083001	0.042365	0.086146	0.050995	0.055196	0.051196	0.055363
19	0.00050	5	0.042280	0.082231	0.040347	0.086082	0.049790	0.054011	0.049993	0.054163

Figure 4.7: MSE for Lasso Regression.The first column shows the tuning parameter,the second column shows the polynomial complexity,the third column shows the train MSE the corresponding tuning parameter and complexity of polynomial and the fourth column shows the test MSE for the corresponding tuning parameter and complexity of polynomial.Sixth and seven column shows the result for training and test MSE for Bootstrap.Column eight and ninth shows the training and test MSE for 5-fold-cv and column 10th and 11th show the training and test MSE for 10-fold-cv

CHAPTER 5

Discussion

Beginning with Figure 4.2, where we have plotted the squared bias, variance, and mean squared error (MSE), where we see that the bias remains relatively constant within the complexity range up to a 7th order polynomial. In contrast, the variance starts to increase after the 2nd degree, leading to a rise in the test MSE. This observation aligns with our expectation that the test MSE would increase as the model fits the training data more accurately. Our theoretical discussion, elaborated in the preceding sections, regarding the disparities between R^2 and MSE seems to be confirmed in Figure 4.4: as MSE increases, R^2 decreases, and vice versa. However, it's important to note that a high R^2 does not guarantee the absence of issues like overfitting, which could result in poor generalization to new data. Moreover, excessively high R^2 values might be achieved by overly complex models, a situation which might not be desirable.

Moving on to Figure 4.5, illustrating the results for Ordinary Least Squares (OLS) Regression, we observe that with increasing complexity, the training error decreases while the test error rises. Interestingly, the results from cross-validation and Bootstrap methods closely align with those obtained from splitting the data into training and test sets. When examining the outcomes of Ridge and Lasso Regression, we notice that the disparity between test and training MSE is narrower. This indicates that the impact of the penalization tuning parameters is noticeable, even with our choice of a grid involving small penalization values.Over al we see that both Ridge and lasso those better then OLS Regression.

CHAPTER 6

Conclution

In this report we have looked at the differences between regression methods OLS, Ridge and Lasso Regression.We started with some theory on the differences between the methods trough derivations of the parameters that minimizes the least squares for each method.We found out that Ridge which uses the L2 norm shrinks the parameters of the OLS estimate with $\frac{1}{1+\lambda}$,while Lasso can shrink parameters to zero.We saw that both Ridge and lasso results in biased parameters.We saw that by choosing biased parameters we could reduce the variance of the parameter,which could give us a model more general model that could do better new data.We this in our simulation study which confirmed the theory. In addition we looked at the Bias-Variance trade an the interpretation of its decomposition.We looked at methods We looked at the aspect of data preprocessing,before looking at the most important part of our implementation.Overall this has been a report that has been informative to write.

CHAPTER 7

The First Appendix

7.1 Derivation of OLS estimate and Ridge estimate using SVD

$$X(X^TX)^{-1}X^Ty = (U\Sigma V^T)((U\Sigma V^T)^T(U\Sigma V^T))^{-1}(U\Sigma V^T)^Ty \tag{7.1}$$

$$= (U\Sigma V^T)(V\Sigma^T\Sigma V^T)^{-1}(U\Sigma V^T)^Ty \tag{7.2}$$

$$= (U\Sigma V^T)(V\Sigma^T\Sigma V^T)^{-1}(U\Sigma V^T)^Ty \tag{7.3}$$

$$= U\Sigma V^TV(\Sigma^T\Sigma)^{-1}V^TV\Sigma^TU^Ty \tag{7.4}$$

$$= U\Sigma\Sigma^T(\Sigma\Sigma^T)^{-1}U^Ty \tag{7.5}$$

$$\tag{7.6}$$

$$= UU^Ty = \sum_{j=1}^{p} u_ju_j^Ty \tag{7.7}$$

$$\tag{7.8}$$

We want to do the same in the Ridge case

$$X\hat{\beta}_{\text{ridge}} = X(X^TX + \lambda I)^{-1}X^Ty \tag{7.9}$$

Substituting the expression for X:

$$X\hat{\beta}_{\text{ridge}} = U\Sigma V^T((U\Sigma V^T)^T(U\Sigma V^T) + \lambda I)^{-1}(U\Sigma V^T)^Ty \tag{7.10}$$

Since V is orthogonal, $V^T = V^{-1}$ and U is orthogonal, so $U^TU = I$:

$$X\hat{\beta}_{\text{ridge}} = U\Sigma V^T(V\Sigma^TU^TU\Sigma V^T + \lambda I)^{-1}V\Sigma^TU^Ty \tag{7.11}$$

$$X\hat{\beta}_{\text{ridge}} = U\Sigma V^T(V\Sigma^T\Sigma V^T + \lambda I)^{-1}V\Sigma^TU^Ty \tag{7.12}$$

$$X\hat{\beta}_{\text{ridge}} = U\Sigma V^T(\Sigma^T\Sigma + \lambda I)^{-1}V\Sigma^T U^T y \tag{7.13}$$

$$X\hat{\beta}_{\text{ridge}} = U\Sigma(\Sigma^T\Sigma + \lambda I)^{-1}\Sigma^T U^T y \tag{7.14}$$

$$= \sum_{j=1}^{p} \mathbf{u}_j^T \mathbf{u}_j \left(\frac{\sigma_j^2}{\sigma_j^2 + \lambda}\right) y \tag{7.15}$$

7.2 Bias-variance trade of derivation

$$E[(y - \tilde{y})^2] = E[E[(y - \tilde{y})^2|x]] \tag{7.16}$$

y here represents a new data point comming from the true dist.

we want to show that

$$E[(y - \tilde{y})^2|x] = bias(\tilde{y})^2 + var(\tilde{y}) + \sigma_y^2 \tag{7.17}$$

$$E[(y - \tilde{y})^2|x] = E[(y - E[y|x] + E[y|x] - \tilde{y})^2|x] \tag{7.18}$$

$$= E[(y - E[y|x])^2|x] + E[(E[y|x] - \tilde{y})^2|x] - 2E[y - E[y|x]|x]E[E[y|x] - \tilde{y}|x] \tag{7.19}$$

$E[y_i - E[\tilde{y}]|x] = 0$

we get:

$$E[(y - \tilde{y})^2|x] = E[(y - E[y|x])^2|x] + E[(E[y|x] - \tilde{y})^2|x] \tag{7.20}$$

we get:

$$E[(y - E[\tilde{y}])^2|x] = Var[y|x] + E[(\tilde{y} - E[y|x])^2|x] \tag{7.21}$$

Further:

$$E[(\tilde{y} - E[y|x])^2|x] = E[(\tilde{y} - E[\tilde{y}] + E[\tilde{y}] - E[y|x])^2|x] \tag{7.22}$$

$$E[(\tilde{y} - E[y|x])^2|x] = E[(\tilde{y} - E[\tilde{y}])^2|x] + E[(E[y|x] - E[\tilde{y}])^2|x] - 2E[y - E[y|x]|x]E[E[y|x] - \tilde{y}|x] \tag{7.23}$$

$$= E[(\tilde{y} - E[\tilde{y}])^2|x] + E[(E[y|x] - E[\tilde{y}])^2|x] \tag{7.24}$$

$$= Var[f\tilde{(x)}|x] + (f(x) - (E[f\tilde{(x)}]|x])^2 \tag{7.25}$$

$$= var(\tilde{y}) + bias(\tilde{y})^2 \tag{7.26}$$

$$E[E[(y-\tilde{y})^2|x]] = E[bias(\tilde{y})^2|x] + E[var(\tilde{y}|x) + E[\sigma_y^2] \tag{7.27}$$

$$E[(y-\tilde{y})^2] = bias(\tilde{y})^2 + var(\tilde{y}) + \sigma_y^2 \tag{7.28}$$

7.3 More detailed derivation of variance and expectation of the Ridge and OLS estimate

$$\begin{aligned}
E[X\hat{\beta}^{Ridge}] &= E[X(X^TX)^{-1}X^Ty] && (7.29)\\
&= E[X(X^TX)^{-1}X^Ty] && (7.30)\\
&= X(X^TX)^{-1}X^TE[y] && (7.31)\\
&= X\beta && (7.32)
\end{aligned}$$

For Ridge regression we only check whether $E[X\hat{\beta}^{Ridge}] = \beta$:

$$\begin{aligned}
E[X\hat{\beta}^{Ridge}] &= E[X(X^TX+\lambda I_{pp})^{-1}X^Ty] && (7.33)\\
&= E[X(X^TX+\lambda I_{pp})^{-1}X^Ty] && (7.34)\\
&= (XX^TX+\lambda I_{pp})^{-1}X^TE[y] && (7.35)\\
&= (XX^TX+\lambda I_{pp})^{-1}X^TX\beta && (7.36)
\end{aligned}$$

Which is not the case.

The variance OLS:

$$\begin{aligned}
Var[X\hat{\beta}^{Ridge}] &= Var[X(X^TX)^{-1}X^Ty] && (7.37)\\
&= ((X^TX)^{-1}X^T)((X^TX)^{-1}X^T)^TVar[y] && (7.38)\\
&= \sigma^2(X^TX)^{-1} && (7.39)
\end{aligned}$$

Variance of Ridge

$$\begin{aligned}
Var[X\hat{\beta}^{Ridge}] &= Var[X(X^TX+\lambda I_{pp})^{-1}X^Ty] && (7.40)\\
&= ((X^TX+\lambda I_{pp})^{-1}X^T)((X^TX+\lambda I_{pp})^{-1}X^T)^TVar[y] && (7.41)\\
&= (X^TX+\lambda I_{pp})^{-1}X^TX((X^TX+\lambda I_{pp})^{-1})^TVar[y] && (7.42)\\
&= Var[y](X^TX+\lambda I_{pp})^{-1}X^TX((X^TX+\lambda I_{pp})^{-1})^T && (7.43)\\
&= \sigma^2(X^TX+\lambda I_{pp})^{-1}X^TX((X^TX+\lambda I_{pp})^{-1})^T && (7.44)
\end{aligned}$$

For $\lambda > 0$ we have that:

$$Var[X\hat{\beta}^{Ridge}] < Var[X\hat{\beta}^{OLS}] \tag{7.45}$$

Bibliography

ref1 [GBC16] Goodfellow, I., Bengio, Y. and Courville, A. *Deep Learning.* http://www.deeplearningbook.org. MIT Press, 2016.

ref3 [Hjo23] Hjorth-Jensen, M. *Lecture Notes Fys-stk 4155.* https://compphysics.github.io/MachineLearning/doc/LectureNotes/_build/html/intro.html.. 2023.

ref2 [HTF01] Hastie, T., Tibshirani, R. and Friedman, J. *The Elements of Statistical Learning.* Springer Series in Statistics. New York, NY, USA: Springer New York Inc., 2001.