Project 1: Regression and Resampling Methods

Owen Huff and Ilya Berezin

October 7, 2022

FYS-STK 4155 - Applied Data Analysis and Machine Learning

University of Oslo

Abstract

In this project we present a theoretical overview and practicum of linear regression. We first describe the theory behind the different regression methods - ordinary least squares, ridge, and lasso regression - and then describe data resampling methods - bootstrapping and cross validation - which provide information necessary for better model selection. These models and resampling techniques are then applied on data generated by a 2D analytical function (the Franke function). Our results demonstrate several important concepts, such as the negative influence of noisy data, and the bias variance tradeoff inherent to model selection. Finally we test our model development workflows on real topography data from Iowa and Pennsylvania.

Reproducibility

The code used for this report can be accessed at:
https://github.com/orhuff/FYS-STK-4155-Project1-Regression

Introduction

The goal of regression is to produce a model that can return accurate predictions of continuous function values, based on inputs of data from one or more explanatory variables. Regression is applied frequently in science, engineering, economics, and practically any analytical field because of its power with simplicity. As an example, suppose you wanted to build a model to predict someone's income based on variables such as their age, years of work experience, and other quantities. Because you want to predict a continuous value (income), and the relationship between the variables likely isn't exceedingly complex, linear or polynomial regression is probably the best approach.

Let's say you collect data on income based on P explanatory variables from N people. This data can be thought of as a vector $\boldsymbol{y}$ which is N entries long. In an ideal world there would be a function $f(x)$ that perfectly describes this data - however in actuality there will almost always be error. Thus we can think of the data as a sum of the ideal function and an irreducible error vector ϵ. For simplicity, we will consider error to follow a Gaussian distribution with zero mean and a variance of σ^2.

$$\boldsymbol{y} = f(\boldsymbol{x}) + \boldsymbol{\epsilon} \tag{1}$$

In linear (or polynomial) regression our goal is to build a model $\tilde{\boldsymbol{y}}$ that best approximates the data $\boldsymbol{y}$. To build such a model we set up a linear algebra problem in which the model is equal to the product of a design matrix $\boldsymbol{X}$ and adjustable model parameter values $\boldsymbol{\beta}$. The design matrix consists of all the observations for each explanatory variable, and is therefore of dimension NxP. The model parameters are the weights by which to adjust and multiply the data in order to yield predictions - the most familiar example of these are the slope and intercept of a linear equation.

$$\tilde{\boldsymbol{y}} = \boldsymbol{X}\boldsymbol{\beta} \tag{2}$$

The optimization of the model parameters $\boldsymbol{\beta}$ is at the heart of regression (and machine learning in general) and requires selecting a metric by which to evaluate the quality of the model. This metric is called the cost function or loss function. Different cost functions define different regression models - ordinary least squares, ridge, and lasso - and are described next in the Theory section.

Theory

Ordinary Least Squares Regression

The most basic form of regression is that of ordinary least squares (OLS). In OLS the cost function by which to optimize the model parameters $\boldsymbol{\beta}$ is called the mean squared error (MSE). The MSE measures the mean difference in error between the data $\boldsymbol{y}$ and the model $\tilde{\boldsymbol{y}}$, and is defined as such:

$$MSE(\boldsymbol{y}, \tilde{\boldsymbol{y}}) = \frac{1}{n}\sum_{i=0}^{n-1}(y_i - \tilde{y}_i)^2 \tag{3}$$

We want to find the model parameters $\boldsymbol{\beta}$ that result in this quantity reaching a minimum value, because that will result in a model with the lowest possible error. Thus we can re-express the MSE cost function in matrix form (using the knowledge that $\tilde{\boldsymbol{y}} = \boldsymbol{X}\boldsymbol{\beta}$):

$$C(\boldsymbol{X}, \boldsymbol{\beta}) = \frac{1}{n}[(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta})^T(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta})] = \frac{1}{n}\left\|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}\right\|_2^2 \tag{4}$$

And then set its derivative with respect to $\boldsymbol{\beta}$ equal to zero:

$$\frac{\partial C}{\partial \beta} = \frac{\partial}{\partial \beta}(\frac{1}{n}[(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta})^T(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta})]) = 0 \tag{5}$$

Solving this equation for β yields:

$$\boldsymbol{\beta} = \left(\boldsymbol{X}^T\boldsymbol{X}\right)^{-1}\boldsymbol{X}^T\boldsymbol{y} \tag{6}$$

Therefore the beauty of OLS is that we have an expression that can be solved analytically for the optimal parameters β. The only sometimes-problematic aspect of this expression is if the matrix $\boldsymbol{X}^T\boldsymbol{X}$ is not analytically invertible, though this is not a large concern since pseudo-inverse methods such as singular value decomposition (SVD) can still be applied to return an inverse. Once β is calculated using this expression, the matrix multiplication of $\boldsymbol{X}\boldsymbol{\beta}$ then yields the predicted model values.

Because we have analytical expressions for the quantities in OLS, it is also relatively easy to derive analytical expressions for their statistical quantities, such as the expected value and variance. These following proofs assume again that our data is described by a function with added irreducible Gaussian noise of mean zero and variance σ^2:

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

Proof 1:

$$\mathbb{E}[y_i] = \boldsymbol{X_{i,*}}\beta,$$

$$\mathbb{E}[y_i] = \mathbb{E}[f(x_i) + \epsilon_i] = \mathbb{E}[\boldsymbol{X_{i,*}}\boldsymbol{\beta} + \epsilon_i] = \mathbb{E}[\boldsymbol{X_{i,*}}\boldsymbol{\beta}] + \mathbb{E}[\epsilon_i]$$
$$= \mathbb{E}[\sum_j x_{ij}\beta_j] + \mathbb{E}[\epsilon_i] = \sum_j x_{ij}\beta_j = \boldsymbol{X_{i,*}}\beta,$$

Proof 2:

$$Var(y_i) = \sigma^2$$

$$Var(y_i) = \mathbb{E}[(y_i - \mathbb{E}(y_i))^2] = \mathbb{E}[y_i^2] - \mathbb{E}[y_i]^2$$
$$= \mathbb{E}[(\boldsymbol{X_{i,*}}\beta + \epsilon_i)^2] - (\boldsymbol{X_{i,*}}\beta)^2$$
$$= \mathbb{E}[(\boldsymbol{X_{i,*}}\beta)^2 + 2\epsilon_i\boldsymbol{X_{i,*}}\beta + \epsilon_i^2] - (\boldsymbol{X_{i,*}}\beta)^2$$
$$= (\boldsymbol{X_{i,*}}\beta)^2 + 2\mathbb{E}[\epsilon_i]\boldsymbol{X_{i,*}}\beta + \mathbb{E}[\epsilon_i^2] - (\boldsymbol{X_{i,*}}\beta)^2$$
$$= \mathbb{E}[\epsilon_i^2] = Var(\epsilon_i) = \sigma^2$$

Proof 3:

$$\mathbb{E}[\hat{\boldsymbol{\beta}}] = \boldsymbol{\beta}$$

$$\mathbb{E}[\hat{\boldsymbol{\beta}}] = \mathbb{E}[(\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{Y}] = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\mathbb{E}[\boldsymbol{Y}]$$
$$= (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{X}\boldsymbol{\beta} = \boldsymbol{\beta}$$

Proof 4:

$$Var(\hat{\boldsymbol{\beta}}) = \sigma^2(\boldsymbol{X}^T\boldsymbol{X})^{-1}$$

$$Var(\hat{\boldsymbol{\beta}}) = \mathbb{E}[(\boldsymbol{\beta} - \mathbb{E}[\boldsymbol{\beta}])(\boldsymbol{\beta} - \mathbb{E}[\boldsymbol{\beta}])^T]$$
$$= \mathbb{E}[((\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{Y} - \boldsymbol{\beta})((\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{Y} - \boldsymbol{\beta})^T]$$
$$= (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\mathbb{E}[\boldsymbol{Y}^T\boldsymbol{Y}]\boldsymbol{X}(\boldsymbol{X}^T\boldsymbol{X})^{-1} - \boldsymbol{\beta}\boldsymbol{\beta}^T$$
$$= (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T(\boldsymbol{X}\boldsymbol{\beta}\boldsymbol{\beta}^T\boldsymbol{X}^T + \sigma^2)\boldsymbol{X}(\boldsymbol{X}^T\boldsymbol{X})^{-1} - \boldsymbol{\beta}\boldsymbol{\beta}^T$$
$$= \boldsymbol{\beta}\boldsymbol{\beta}^T + \sigma^2(\boldsymbol{X}^T\boldsymbol{X})^{-1} - \boldsymbol{\beta}\boldsymbol{\beta}^T$$
$$= \sigma^2(\boldsymbol{X}^T\boldsymbol{X})^{-1}$$

Ridge Regression

Ridge regression is a regularization modification to ordinary least squares that can be implemented to reduce overfitting. Ordinary least squares estimates can often be biased because of a large number of correlated input variables in relation to the number of data observations, causing multicollinearity. Ridge regression attempts to fix this by adding bias through a regularization term. The main difference between OLS and ridge regression is just this regularization term, which is a usually-small value λ that is multiplied by the 2-norm of the β parameters and then added to the typical OLS cost function:

$$C(\boldsymbol{X}, \boldsymbol{\beta}) = \frac{1}{n}\left\|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}\right\|_2^2 + \lambda\left\|\boldsymbol{\beta}\right\|_2^2 \tag{7}$$

The effect of the regularization term is to penalize β parameters that take too large of values. When lambda goes to zero the cost function is the same as OLS - therefore a larger value of λ means stronger regularization. The optimal parameters β can be shown to be the following, which is used in the practical calculations in our project:

$$\hat{\boldsymbol{\beta}}_{Ridge} = (\boldsymbol{X}^T\boldsymbol{X} + \lambda\boldsymbol{I})^{-1}\boldsymbol{X}^T\boldsymbol{y} \tag{8}$$

Lasso Regression

Lasso regression is a different regularization modification to OLS that can be used to reduce overfitting. Instead of the weighted 2-norm of the β parameters being added to the cost function, the weighted 1-norm (or absolute value) is used. LASSO is actually an acronym which stands for Least Absolute Shrinkage and Selection Operator. Its effect is to "shrink" the parameters β that are are too large and cause overfitting of the the model. The modified cost function for Lasso regression is therefore:

$$C(\boldsymbol{X}, \boldsymbol{\beta}) = \frac{1}{n}\left\|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}\right\|_2^2 + \alpha\left\|\boldsymbol{\beta}\right\|_1 \tag{9}$$

We use α for the regularization parameter in order to avoid confusion between lasso and ridge regression in our implementation and results. The optimal parameters β can be shown to equal the following:

$$\hat{\boldsymbol{\beta}}_i^{\text{Lasso}} = \begin{cases} y_i - \frac{\lambda}{2} & \text{if} \quad y_i > \frac{\alpha}{2} \\ y_i + \frac{\alpha}{2} & \text{if} \quad y_i < -\frac{\alpha}{2} \\ 0 & \text{if} \quad |y_i| \leq \frac{\lambda}{2} \end{cases} \tag{10}$$

Lasso regression has a certain potential advantage over the OLS and ridge regression schemes. This advantage comes from the fact that the L1-norm is more robust to outliers than the L2-norm. Thus in a dataset with many extreme values, OLS or ridge regression may overfit to these datapoints, which would be undesirable as they are by definition unlikely to be in the testing dataset.

Conceptually, the difference between lasso and ridge regression can be visualized as constraint regions in Figure 1. For ridge regression the constraint region for β (in 2D) is a circle, while for lasso it is a diamond. If the β value is too high and exceeds these regions, then it is penalized and shrunken to zero. When the number of features increases, the dimensionality of this constraint region also increases, and over-represented β values causing overfitting may be penalized. It is also important to note that both ridge and lasso regression assume scaled data, because otherwise improper weight in the regularization may be given to certain features. Thus this motivates the scaling of our data in the ridge and lasso implementation sections.

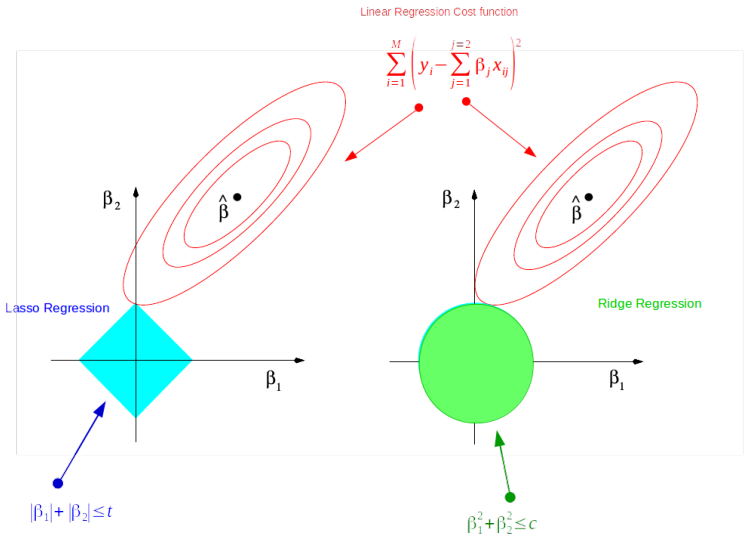

Figure 1: Dimension reduction of feature space with LASSO (figure from James et al., 2013; modified in Bhattacharyya, 2018)

Data Splitting and Resampling

We have described the analytical expressions used to calculate model parameters $\boldsymbol{\beta}$ for different regression methods. However, it is also important to understand methods that help in selecting a good model. In practice when training a machine learning model, it is important to split the data into a training set and testing set. The testing set is usually between 20-30% of the entire dataset. The model is then trained using X and y values (in the case of linear regression) from the training set, and predictions are made using the product of the testing set X values and the determined parameters β. These predictions are compared with the true y values, using a metric such as the MSE or R^2 score (defined below) over all the points in order

to evaluate the effectiveness of the model.

$$R^2(\boldsymbol{y}, \tilde{\boldsymbol{y}}) = 1 - \frac{\sum_{i=0}^{n-1}(y_i - \tilde{y}_i)^2}{\sum_{i=0}^{n-1}(y_i - \bar{y})^2} \tag{11}$$

However, this practice of data splitting has limitations if it is done alone. For instance, what if an overabundance of outlier data is contained in either the training or testing set? In this case the testing of the model may yield different results than are actually the case, and this could hinder the selection of the best model. This motivates the need for resampling techniques, whereby models can be trained over multiple resamplings of the dataset in order to find the best overall model.

One of the most common resampling techniques is the bootstrap method. In this method, one extracts samples with replacement repeatedly from the whole dataset, and calculates desired quantities from the samples. Then statistical properties of the dataset, such as the mean and variance, can be more accurately determined by investigating the behavior across all the samples that were drawn. This is a directly related to the central limit theorem, which says that if you repeatedly resample from normally-distributed data with replacement, then the sample means themselves will be normally-distributed. In statistics, the bootstrap method is usually applied to make more accurate statistical inferences about a population.

However in machine learning, we can also apply the method towards achieving the optimal hyperparameter selection for a model. This is accomplished by bootstrapping the data splitting operation described earlier. Essentially, we can iterate over the model hyperparameter we want (such as the polynomial order for OLS or λ value for ridge regression) and then for each hyperparameter value perform bootstrapping. For each bootstrap iteration we resample (with replacement) the data in order to obtain a training dataset, and then use the remaining data as the testing dataset. The temporary model resulting from this bootstrap iteration and data sampling can then be evaluated using an accuracy metric of choice, like the MSE or R2 value. Then we can average across all accuracy results to find how well, on average, the model with the given hyperparameter values performs. Thus with bootstrapping you can be more confident that a particular model is superior, if it performs better than other model configurations over many resamplings.

Another popular resampling technique is called k-fold cross validation. This technique involves splitting the dataset into k equal groups (or folds) and then using one group as the testing dataset, and the other k-1 groups as the training dataset. After training a model with this configuration, the groups are cycled so that a different group is the testing dataset and the remaining groups are used for training. This repeats k-1 times so that each group gets a turn being the testing dataset. During each iteration, an evaluation of the model performance can be made and then averaged over all k folds following completion, to get a more accurate estimation of the strength of a particular model configuration. Typically the number of folds used is from 5-10, so as to have a reasonable relative size between the testing and training datasets.

The Bias Variance Tradeoff

In the model selection process, one of the overarching concepts that can be observed is the bias variance tradeoff. In order to understand this concept we go back to the mean squared error cost function for ordinary least squares. Through the following Proof 5, this cost function can be re-expressed as the sum of three terms: the (square of the) bias, the variance of the irreducible noise of the data, and the variance of the model.

Proof 5:

$$\frac{1}{n}\sum_{i=0}^{n-1}(y_i-\tilde{y}_i)^2 = Bias[\tilde{\boldsymbol{y}}]^2+\sigma^2+Var[\tilde{\boldsymbol{y}}]$$

$$C(\boldsymbol{X},\boldsymbol{\beta}) = \frac{1}{n}\sum_{i=0}^{n-1}(y_i-\tilde{y}_i)^2 = \mathbb{E}[(\boldsymbol{y}-\tilde{\boldsymbol{y}})^2]$$

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

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

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

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

$$= Bias[\tilde{\boldsymbol{y}}]^2+Var[\epsilon]+Var[\tilde{\boldsymbol{y}}]$$

$$= Bias[\tilde{\boldsymbol{y}}]^2+\sigma^2+Var[\tilde{\boldsymbol{y}}]$$

The bias is defined as the difference between the mean prediction of the model and the truth. The conceptual meaning of bias can be thought of in terms of throwing many darts at a dartboard. If the average distance of the darts from the bullseye is low, then the bias is low. And the variance has the intuitive meaning of the degree of spread of the darts. These meanings translate to real plots of data in terms of the concepts of underfitting and overfitting. Underfitting is when the model has lower variance (spread) but higher bias (far from the target) - thus it can be thought of as a very simple fit of a curve with low polynomial order that does not contain enough complexity to correctly model the data. Overfitting is the opposite, where the model has higher variance and lower bias - this can be thought of as a needlessly-complex fit of a curve with high polynomial order that follows every little variation. Underfitted models can usually be easily detected because they will just return incorrect prediction values. Overfitted models may perform well on the current training and testing dataset, but are not likely to generalize well to make predictions on unseen data.

The bias variance tradeoff is therefore the concept that you want to strike a balance between these two effects, selecting a model that has a low bias and low variance. One can plot the bias and variance of a model, along with its overall error, in order to see this concept. Figure 2 from James et al., 2013 demonstrates this, where they plot the bias, variance, and the MSE (the bias + variance + irreducible noise) as a function of the model flexibility/complexity. At

low model complexity, the bias is high and the variance is low, while at high model complexity, the bias approaches zero and the variance increases significantly. The best model to select is therefore in the middle of these extremes, at a moderate model complexity, where the total error is minimized. This can be seen as the minima of the MSE curve. We will reproduce a similar curve to this in the Results section of the report.

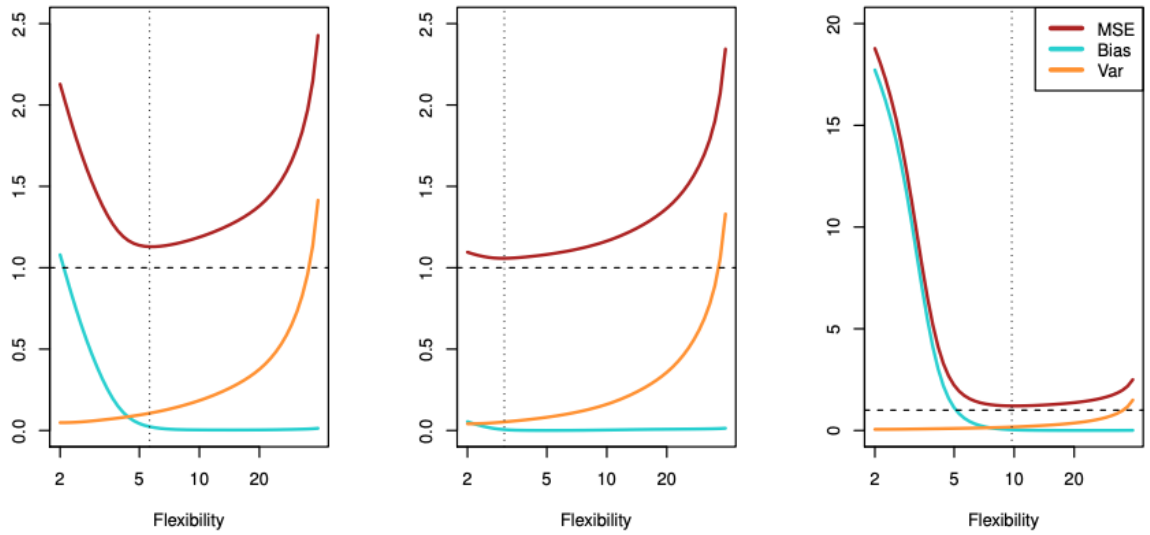

Figure 2: Model error, bias, and variance plotted as a function of model flexibility/complexity, illustrating the bias variance tradeoff (James et al., 2013)

The Franke Function

For our method of implementation to test the various regression methods and their properties, we will use a function called the Franke function (Franke, 1979). This function is two-dimensional and is frequently used to test spatial regression and interpolation problems. It is a weighted sum of four exponential terms and is defined below. An example surface plot of the Franke function calculated for 100 x and y points in the range [0,1] is shown in Figure 3. By having an analytical function with which to test, we can more easily see the influence of added noise and other factors before moving to real data.

$$\begin{aligned} f(x,y) = &\frac{3}{4}\exp\left(-\frac{(9x-2)^2}{4} - \frac{(9y-2)^2}{4}\right) + \frac{3}{4}\exp\left(-\frac{(9x+1)^2}{49} - \frac{(9y+1)}{10}\right) \\ &+\frac{1}{2}\exp\left(-\frac{(9x-7)^2}{4} - \frac{(9y-3)^2}{4}\right) - \frac{1}{5}\exp\left(-(9x-4)^2 - (9y-7)^2\right) \end{aligned} \tag{12}$$

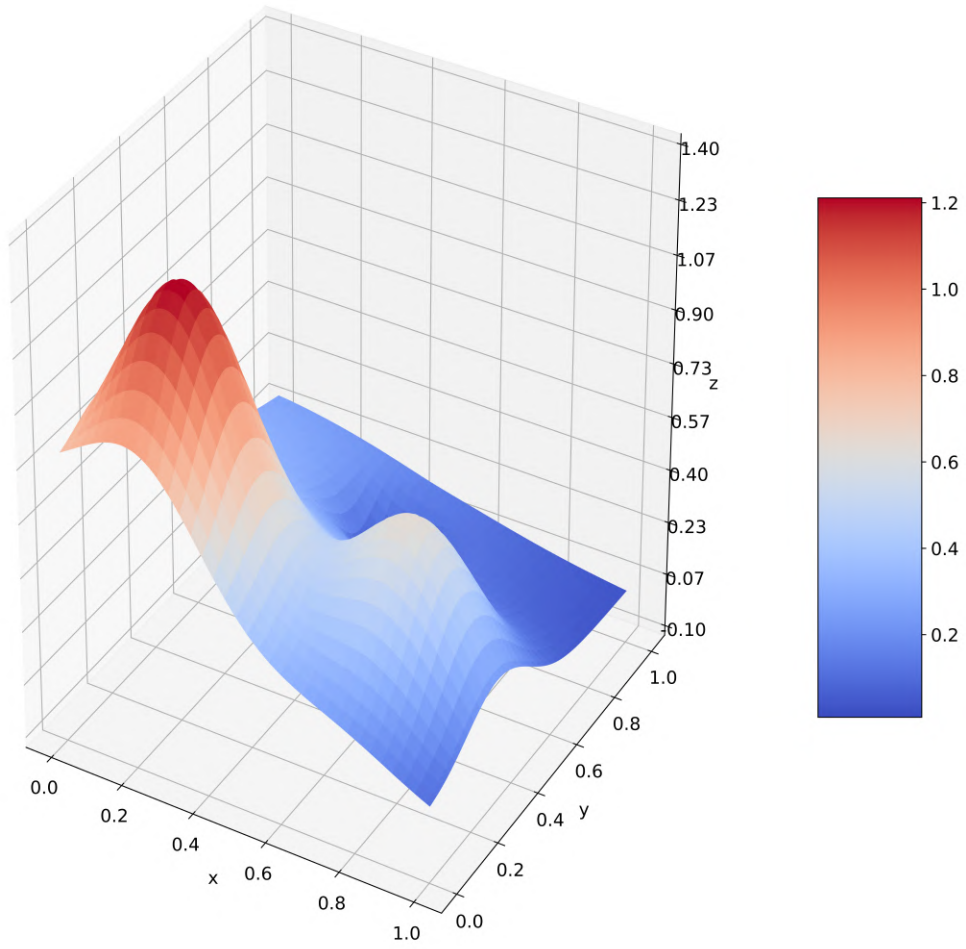

Figure 3: Surface plot of the Franke function calculated for 100 evenly-spaced x and y values in the range [0,1].

Therefore the formulation of the linear algebra problem we want to solve (similar to equation (2)) is as follows, where the model $\tilde{\boldsymbol{z}}$ is a multiplication of the design matrix for an mth degree polynomial of x and y, and the parameters $\boldsymbol{\beta}$:

$$
\begin{aligned}
\tilde{\boldsymbol{z}} &= \boldsymbol{X}\boldsymbol{\beta} \\
&\tilde{z}_0 = \beta_0 + \beta_1 x_0 + \beta_2 y_0 + \beta_3 x_0^2 + \beta_4 x_0 y_0 + \beta_5 y_0^2 + \ldots + \beta_{p-1} y_0^m, \\
&\tilde{z}_1 = \beta_0 + \beta_1 x_1 + \beta_2 y_1 + \beta_3 x_1^2 + \beta_4 x_1 y_1 + \beta_5 y_1^2 + \ldots + \beta_{p-1} y_1^m, \\
&\quad \vdots \\
&\tilde{z}_n = \beta_0 + \beta_1 x_n + \beta_2 y_n + \beta_3 x_n^2 + \beta_4 x_n y_n + \beta_5 y_n^2 + \ldots + \beta_{p-1} y_n^m,
\end{aligned}
\tag{13}
$$

Results and Discussion

OLS Regression on the Franke Function

We begin by implementing the ordinary least squares algorithm on the Franke function for different model complexities. The workflow consists of first generating an x,y grid of evenly-spaced points in the range [0,1], and calculating the Franke function values z from these x,y points. Optionally, we can include Gaussian noise by adding it to these z values. Then we iterate over polynomial order and do the following for each order:

1. add gaussian noise of mean zero and standard deviation 0.1 to the z data
2. generate the design matrix of the given order
3. perform a train-test split of X and z in the proportion 80-20%
4. scale the data by subtracting the mean of the training datasets
5. calculate the beta parameters using equation 6
6. calculate the beta parameters variance using the relation in Proof 4
7. make the z predictions for train and test data
8. calculate the mean squared error (equation 3) and R2 values (equation 11) for the train and test data

This overall structure of this workflow is largely similar for the implementation of other regression algorithms later in this report, with the main change being a resampling loop.

Our first test consists of an iteration over polynomial order from 1 to 15, where we also investigate the impact of added Gaussian noise and data scaling. We use 100 data points (evenly-spaced in the range [0,1]) to calculate the Franke function z values, and then add Gaussian noise of mean 0 and standard deviation 0.1. The results are presented in Figure 4.

We can see that, as an overall trend, the mean squared error decreases and the R2 value increases with increasing polynomial order. However, there is a point of diminishing returns at around order 5, after which the error decreases almost insignificantly in an asymptotical fashion. Furthermore, we notice that the run with noise added results in higher error and lower R2 value, which is intuitive because the noise is making the data more difficult for the model to fit. In the case of noise added, the train error tends to be lower than the test error, which is a pattern we see consistently throughout these regression tests. This is because the model has been trained on this set of data, while it has not been previously exposed to the testing data.

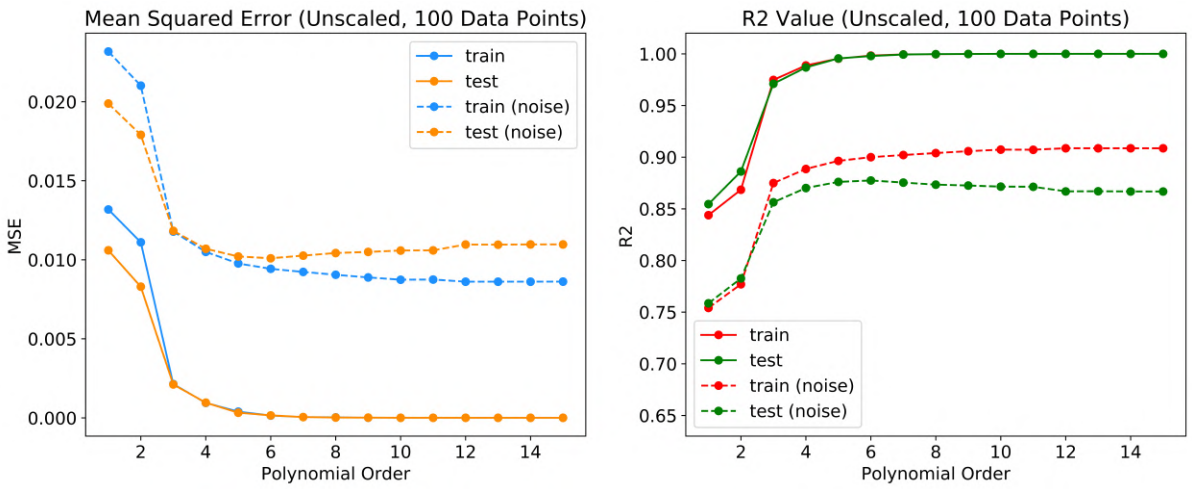

Figure 4: MSE and R2 values for OLS for polynomial order 1-15, using unscaled data.

We also test the results of scaling our data in Figure 5. In our scaling we subtract the mean of the X train values from the X train and X test datasets, and subtract the mean of the z train values from the z train and z test datasets. The reason for this method (as opposed to scaling by the mean of the whole data) is to avoid biasing the training data towards the mean of the test dataset, which it should not see. The results in terms of the error using the scaled data are largely the same as for unscaled data, because our data values are all very similar (approximately from 0 to 1). However, we choose to use scaled data for future OLS tests (and for ridge and lasso regression) because of several benefits. First, in the case of features which are significantly different from each other, such as measurements of different quantities or different units, scaling will help make it so that features arenât inappropriately weighted. Secondly, scaling removes the intercept column from the data and design matrix, thus simplifying calculations and allowing easier comparisons between model performance.

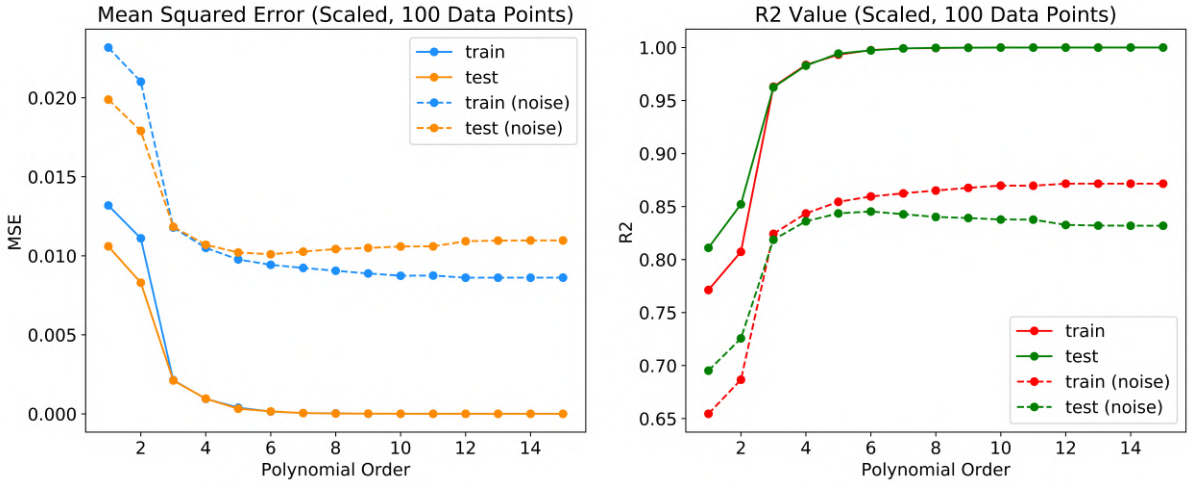

Figure 5: MSE and R2 values for OLS for polynomial order 1-15, using scaled data.

While doing this OLS regression test over polynomial order, we also calculated the variance of the determined parameters beta using the equation in Proof 4. By taking the square root of these variances, we have standard deviations which can be plotted as error bars on the beta values in order to show their uncertainties. This is shown in Figure 6. As we can see, the variance of parameters for higher order polynomials tends to increase significantly, especially approaching order 5. This indicates that a lower polynomial order such as 4 is likely better suited for building a model, especially considering that the improvement in error reduction is negligible as was seen in Figure 5.

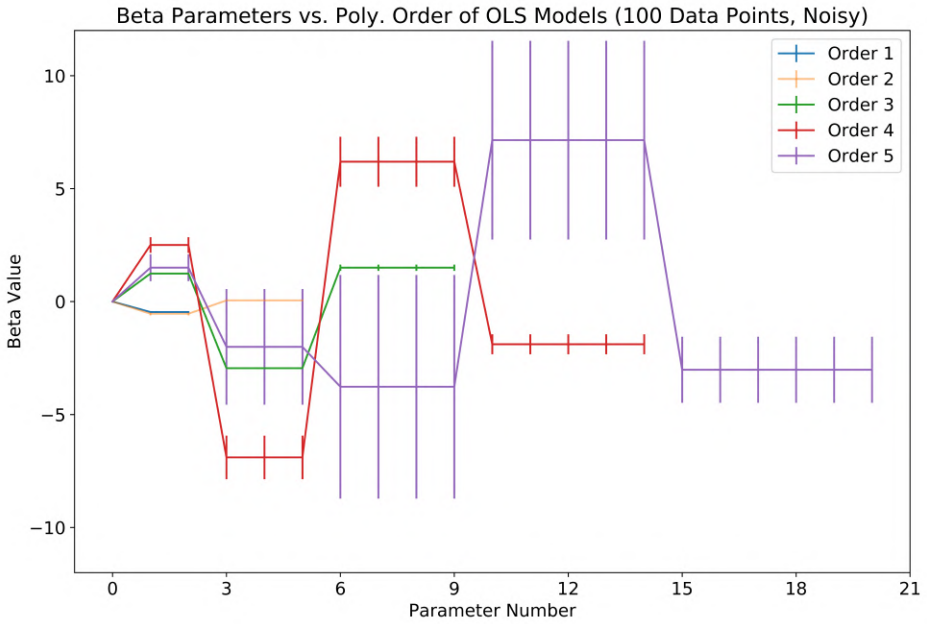

Figure 6: Beta parameter uncertainty plotted for OLS models of polynomial order 1-5.

Next we implement bootstrap resampling with OLS regression. The purpose of this resampling is to obtain more accurate overall behavior of the model. Using 100 bootstraps (re-samplings) we again calculate the mean squared error of OLS models with respect to polynomial degree, but now also calculate the bias and variance as defined in Proof 5. We can now see the bias variance tradeoff demonstrated in Figure 7. Note that we use 50 data points in the calculation in order to see a more pronounced effect of the bias variance tradeoff, as the normal 100 points results in models that are âtoo goodâ and donât show the effects we want to see, namely significant increases in error and variance. We also include Gaussian noise of mean 0 and standard deviation 1. At low polynomial order the MSE and bias are higher with a low variance, and the error decreases to a minimum around order 3 to 6. However as we go past that, the variance and error increase dramatically while the bias stays low. Thus this is the bias variance tradeoff, and the best model to select is in this minimum area from around order 3 to 6.

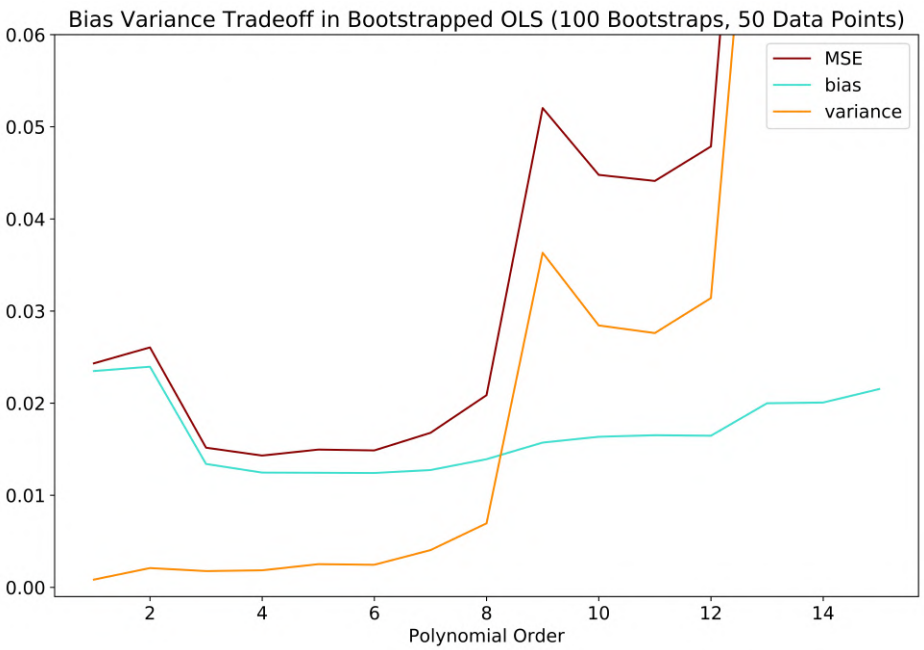

Figure 7: Bias variance tradeoff illustrated for OLS with bootstrap method.

In the next test we implement OLS with cross validation as the resampling technique. We again calculate the MSE as a function of polynomial order, but now also as a function of the number of folds in the cross validation resampling. Going from 5 to 10 folds, we find that the calculated error decreases significantly as the fold number increases, as can be seen in Figure 8. This is because as the number of folds in cross validation increases, the selected training folds are more representative of the entire dataset and thus the effect is to reduce the bias. However, going too high in fold number can increase the variance and would of course increase the computational time. Thus 10 folds is commonly used as a fold number, as it has empirically often yielded good results (Kohavi, 1995).

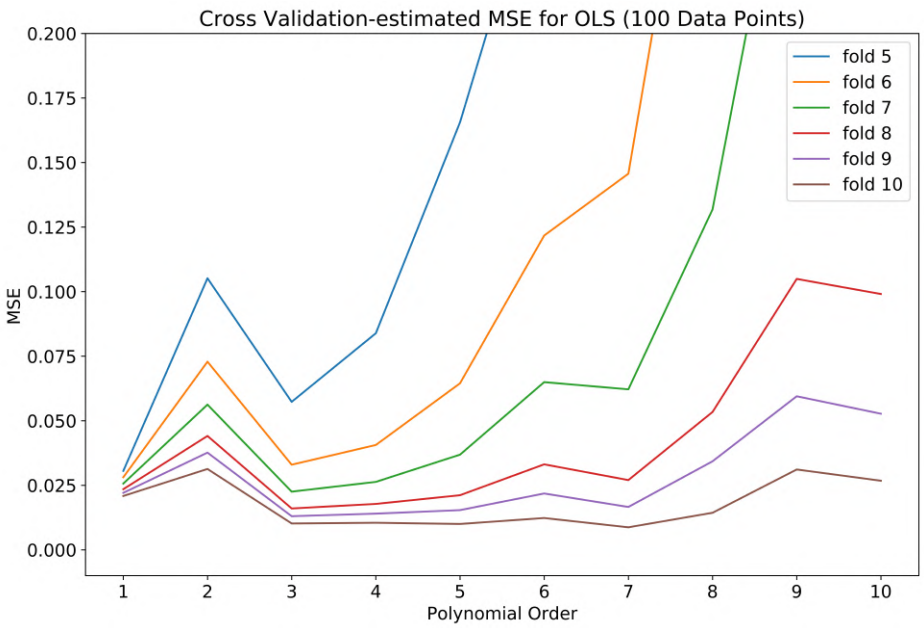

Figure 8: Cross validation for OLS implemented for fold number 5-10, with MSE calculated for models of polynomial order 1-10.

Ridge Regression on the Franke Function

After analysis of OLS model behaviour, we now apply the ridge regression method. We do this in a similar workflow described in the OLS section, except we also iterate over values of the λ regularization hyperparameter to investigate which values lead to the lowest model error. Figure 9 shows our testing results for a grid search over 20 lambda values, evenly spaced in base 10 log space, from 1e-4 to 1, with the other grid search parameter being the usual polynomial order search from 1-5. Our lowest MSE achieved was 0.011 which corresponds to 5^{th} polynomial degree and λ equal to 1e-4.

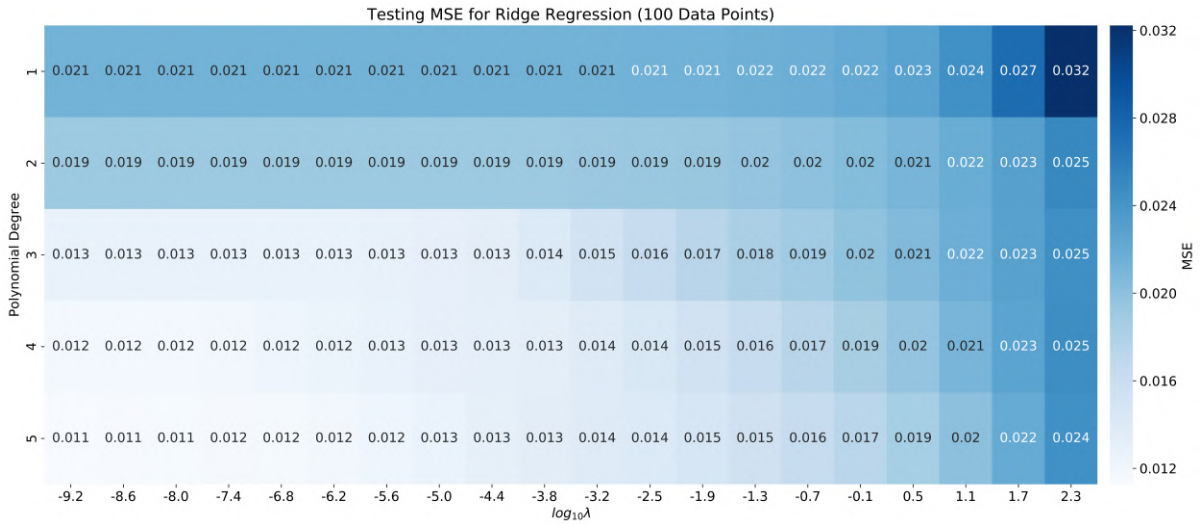

Figure 9: Mean squared error (testing) for the Ridge regression for different values of λ and polynomial degree

As we can see the lowest values of MSE tend to cluster towards the lower left corner of a matrix representing this grid search. This indicates that the regularization required to obtain the best model is not that strong, and this best ridge regression model may be very similar to an OLS model. We also calculate the MSE values for the training dataset, which are predictably lower (as the model is seeing them in the training) and show a more idealized pattern. These are shown in Figure 10. For the sake of brevity, from now on in this report we only show the testing MSE values. However, the training MSE values for all our regression tests are available in the accompanying appendix to this report.

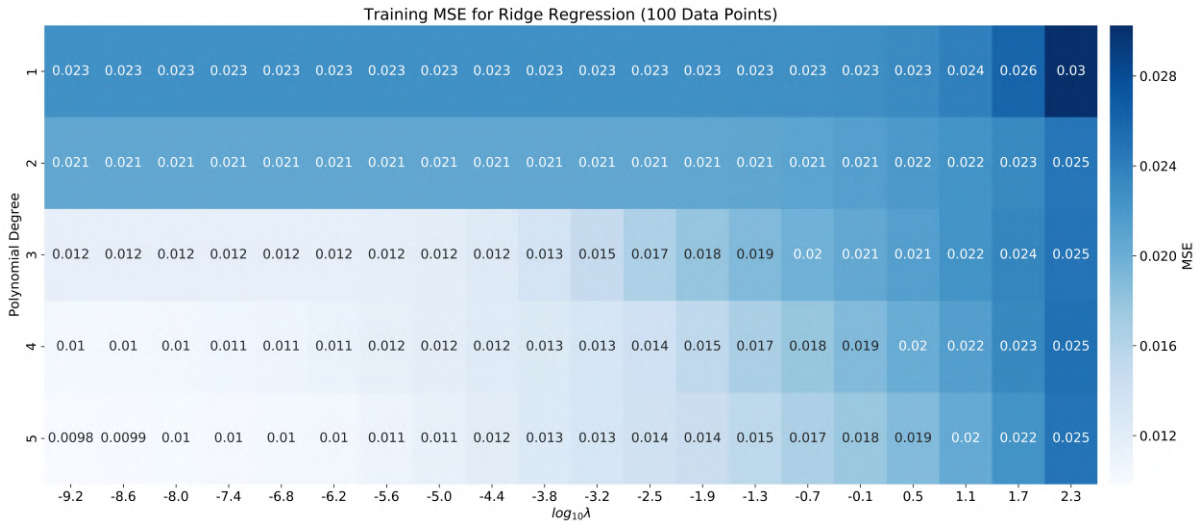

Figure 10: Mean squared error (training) for the Ridge regression for different values of λ and polynomial degree

In order to get a more accurate estimation of the overall behavior of our ridge models, we implement bootstrapping in a similar fashion similar to our work on OLS. Figure 11 shows the MSE resulting from 100 bootstrap resamplings, doing the same grid search over lambda and polynomial order as in the preceding example. Similar results were achieved in our bootstrapping example, with the lowest MSE of 0.012 corresponding to the 5th polynomial degree and λ equal to 1e-4.

Figure 11: Mean squared error for the Ridge regression for different values of λ and polynomial degree, with resampling of the dataset using the bootstrap method.

In Figure 12 we show the MSE, bias, and variance with respect to the polynomial degree for predefined λ = 1e-4 and 4.3e-2. We can see an overall expected trend where the MSE and variance increase dramatically with the model complexity, while the bias tends to level off. We can also see that increasing the lambda parameter reduces the variance, because it is reducing overfitting. However in our case this increased regularization does not correspond to improvement in the MSE, as it increases for higher lambda values. This again indicates that a lower-lambda regularization, close to OLS, is working the best.

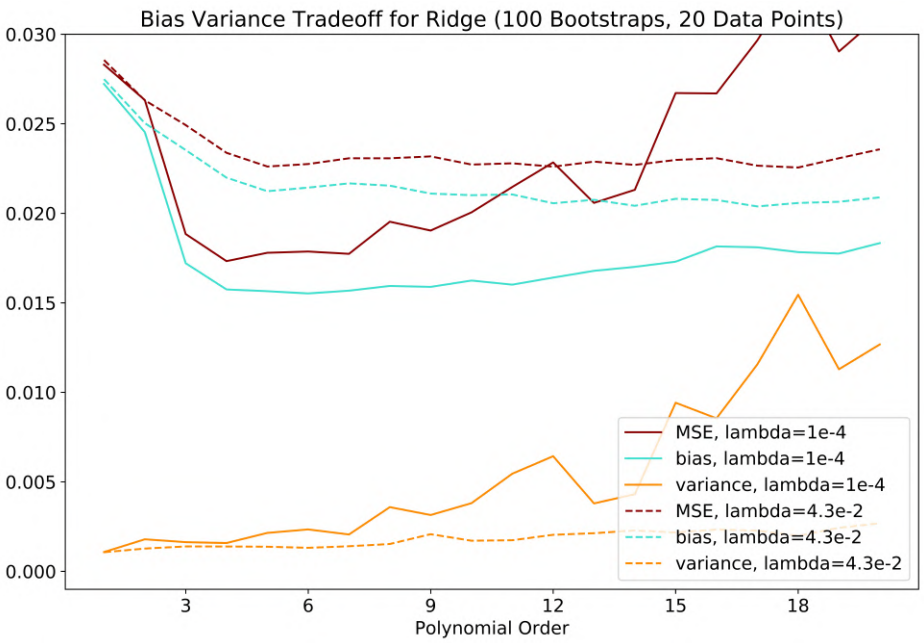

Figure 12: Bias-variance tradeoff and MSE of Ridge regression for different polynomial degree for two different lambda values, 1e-4 and 4.3e-2.

Next we implement k-fold cross validation resampling, where we set k = 10 based on our findings on OLS. We conduct a similar grid search over the lambda parameters and polynomial order, and the results are presented in Figure 13. Here we find that the lowest MSE achieved is 0.0055, corresponding to the 5th polynomial degree and λ equal to 1e-4.

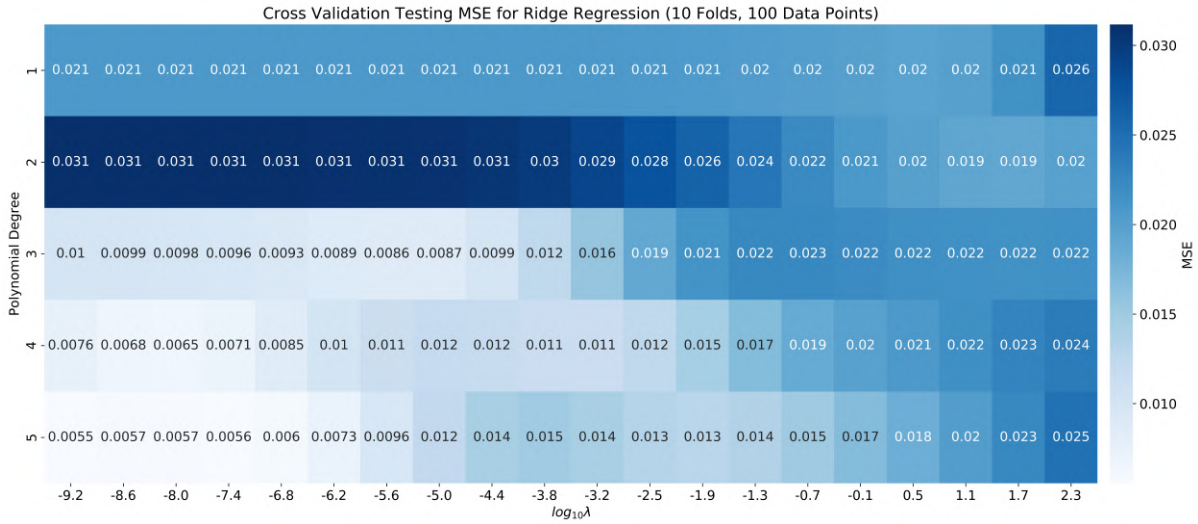

Figure 13: MSE for the Ridge regression for different values of λ and polynomial degree using cross validation technique

Comparing achieved results for the ridge regression using bootstrap resampling and cross

validation, we note that 10-fold cross validation yielded better results than 100-bootstrap resampling, with a best MSE of 0.0055 for the 5th polynomial degree and λ equal to 1e-4, as compared to 0.012 for bootstrapping.

Lasso Regression on the Franke Function

We next test lasso regression on the Franke function, in a similar manner to our tests using ridge regression. The regularizing α parameter represents an adjustable penalty for large β values, therefore we can therefore explore whether the MSE is dependant on it.

Figure 14 illustrates MSE for the LASSO regression on the Franke function for different values of α and polynomial degree using 100 data points. The lowest MSE achieved is 0.013 respective to the 3rd polynomial degree and α equal to 1e-4. Thus our results here are similar to ridge regression with in regards to the lowest regularization parameter yielding the best results, however the results are different in that the lasso model does better with a lower degree polynomial (3 as opposed to 5). This may mean that the lasso model can effectively fit the data with lower model complexity.

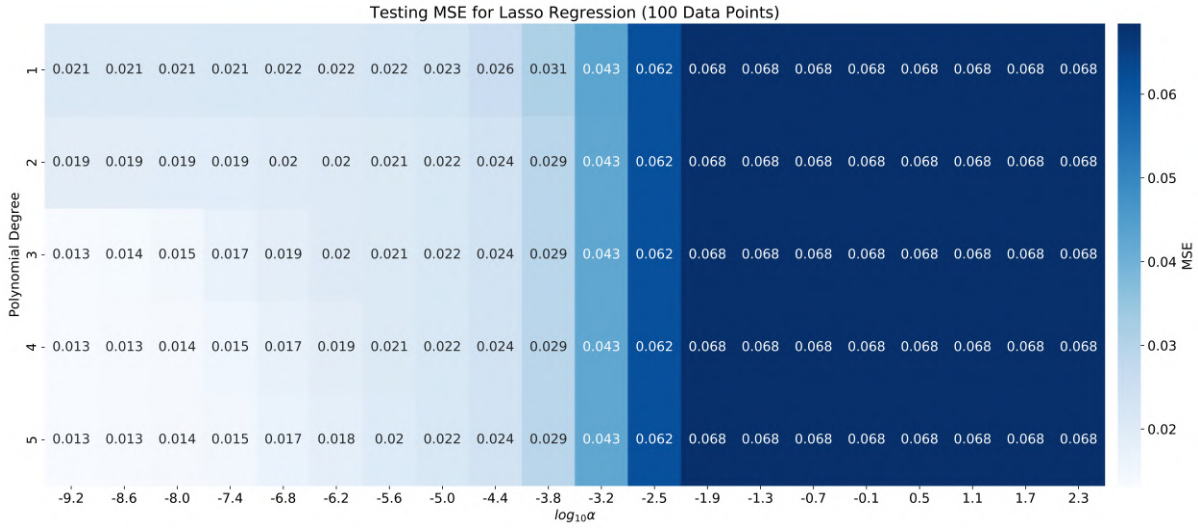

Figure 14: Mean squared error for the LASSO regression for different values of λ and polynomial degree

Figure 15 illustrates the MSE results for lasso regression, this time using 100-bootstrap resamplings. The lowest MSE achieved is 0.014 corresponding to the 3rd polynomial degree and α equal to 1e-4. Again these results are very similar to the ridge regression and OLS results.

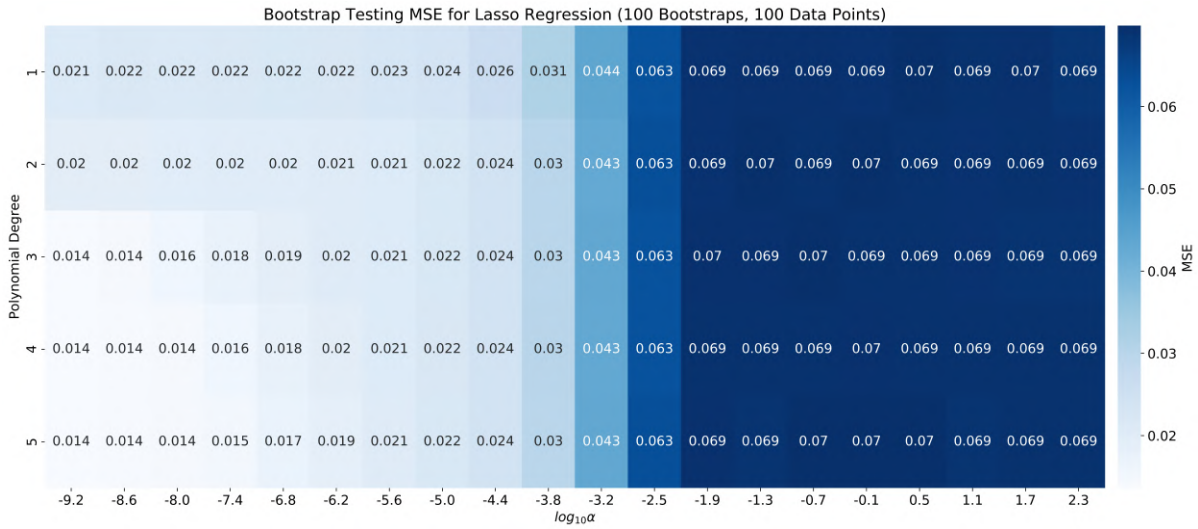

Figure 15: Mean squared error for the LASSO regression for different values of λ and polynomial degree using bootstrap method

Figure 16 illustrates the MSE, bias and variance in respect to the polynomial degree for predefined α = 1e-4 and 4.3e-2. It is important to note that we use a smaller number of data points (20 instead of 100) in order to more easily see the bias variance tradeoff, so the MSE values are slightly higher than for our earlier bootstrapping grid search. We can see the significant increase in the error and bias (with a small decrease in the variance) as we increase the value of the α parameter. We also can see that improvements start to level off past the 3rd order polynomial model complexity.

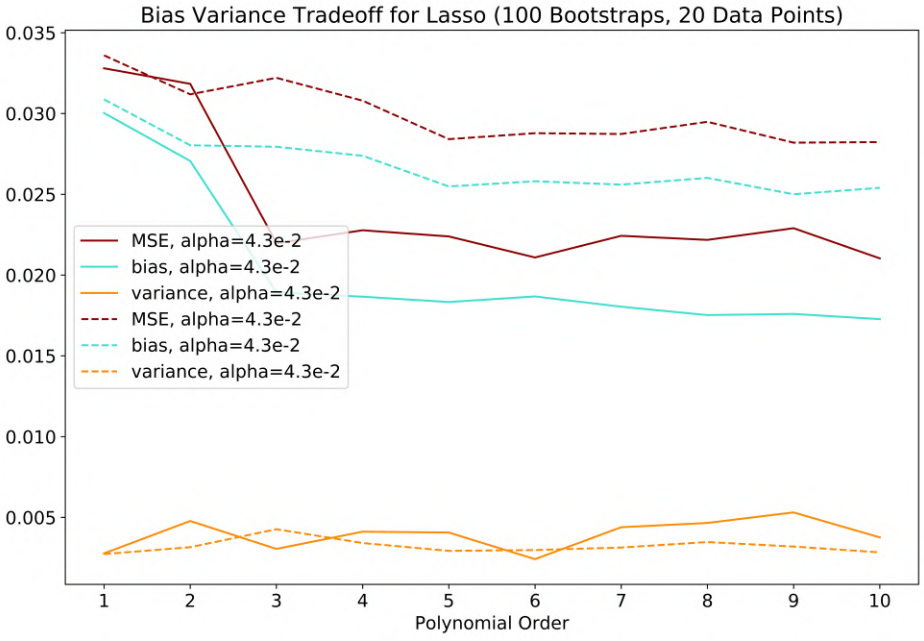

Figure 16: Bias variance tradeoff demonstrated for lasso regression.

Again for lasso regression we now implement k-fold cross validation, where k = 10 and the number of data points is 100. Figure 17 illustrates the MSE for different values of α and polynomial order using this technique. The lowest MSE achieved is 0.022 corresponding to the 3rd polynomial degree and α equal to 1e-4

Figure 17: Mean squared error for lasso regression on the Franke function for different values of α and polynomial degree using 10-fold cross validation resampling.

Overall, comparing achieved results for the regression models (OLS, ridge, and lasso) on the Franke function in Table 1, we note that the errors achieved are largely very similar on the order of 0.01. However there is a lower ridge regression cross validation result, and a higher lasso regression cross validation result, so this may indicate that ridge regression performs slightly better than OLS, which performs slightly better than lasso. Furthermore we note that the most common polynomial order for OLS and ridge is 5 while for lasso it is 3. However this is misleading because the errors for OLS and ridge at order 3 are very close to those at order 5 - there is not much improvement in increasing the model complexity as such. We also noticed that the regularization parameters that the grid search returned as best for ridge and lasso were very close to zero, meaning these models were very close to OLS anyway. Thus our overall recommendation and analysis is that either ridge regression with a small regularization parameter (1e-4) or simply OLS with order 3 or 4 is the best choice for implementing a model on the Franke function. In practice the user may choose to use ridge regression if they are seeking a small improvement in error at all costs, or alternatively use OLS if they value simplicity and easier interpretability.

Regression	Method	Polynomial degree	λ	MSE
OLS	Standard	5	0.0001	0.01
	Bootstrap	5	0.0001	0.011
	Cross validation (k = 10)	5	0.0001	0.01
Ridge	Standard	5	0.0001	0.011
	Bootstrap	5	0.0001	0.012
	Cross validation (k = 10)	5	0.0001	0.0055
Lasso	Standard	3	0.0001	0.013
	Bootstrap	3	0.0001	0.014
	Cross validation (k = 10)	3	0.0001	0.022

Table 1: Comparison of best model performance on the Franke function.

Regression on Real Topography Data

Using the knowledge we have gained from testing regression models on the Franke function, we now test our model workflows on real topography data. This data is from the Shuttle Radar Topography Mission (SRTM) and is provided by the U.S. Geological Survey (USGS, 2022). The first real dataset we select is from Iowa, which is a U.S. state that has mostly flat farmland. The second dataset we select is from Rothrock State Forest in the state of Pennsylvania, which is part of the Appalachian mountain range area and has significant variations in topography. We chose these two datasets in order to see how models would perform on a flat dataset that is easier to fit (Iowa) and a varying dataset that is harder to fit (Pennsylvania). The two datasets are shown in map view in Figure 18 and 3D view in Figure 19.

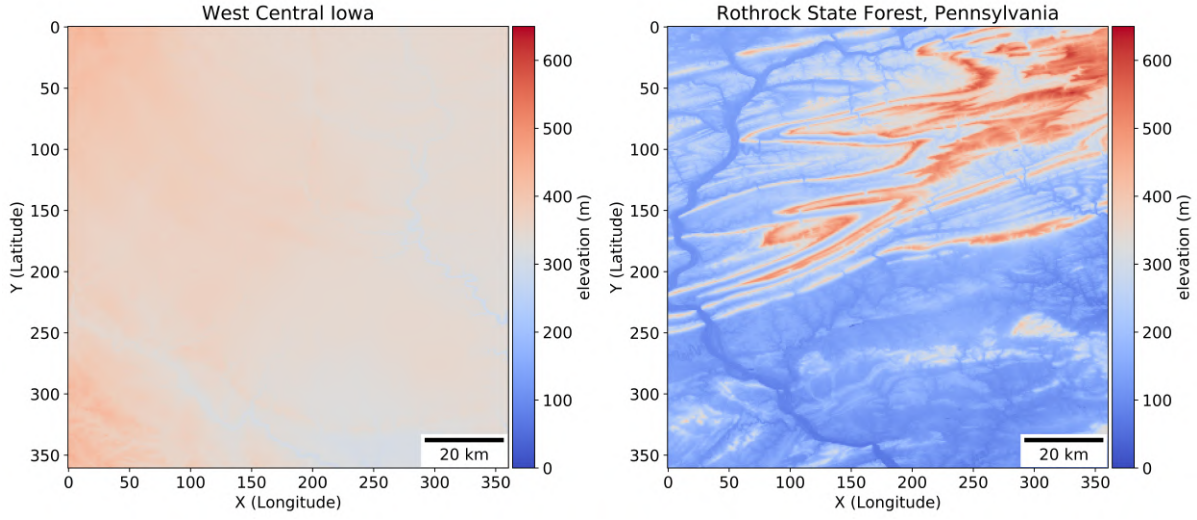

Figure 18: Topographic maps for two different locations in the United States.

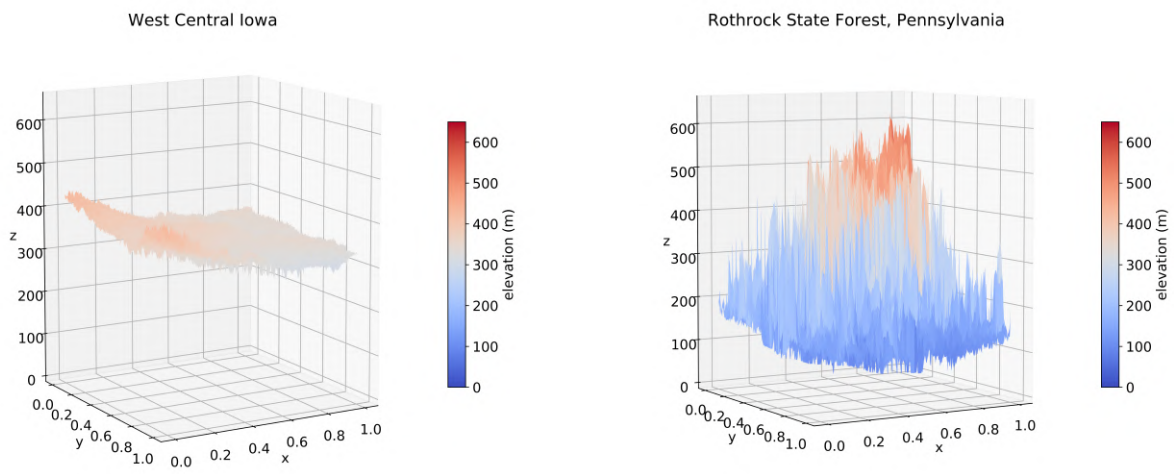

Figure 19: Topographic maps for two different locations in the United States.

We first conduct a test for OLS error as a function of polynomial order from 1-10, using 10-fold cross validation resampling. The results are presented in Figure 20. For the Iowa example, the errors are much lower because the data is flatter and easier to fit. However past order 8, the test error increases dramatically while the training error continues to decrease, indicating overfitting. In contrast the Pennsylvania data has significantly higher error since the topography is more varied and it is harder to fit a model. Furthermore the test error increases dramatically at only polynomial order 5, indicating extreme overfitting of the data.

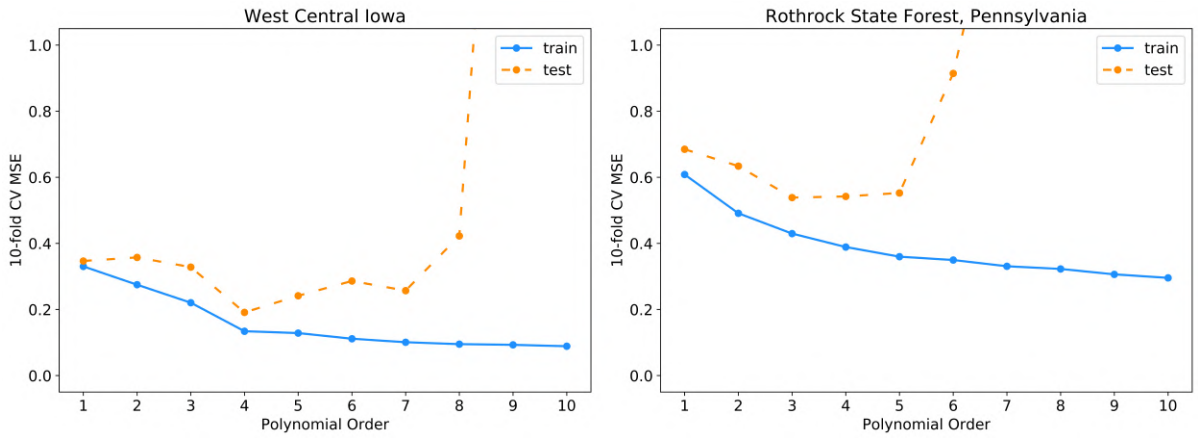

Figure 20: Real OLS CV.

Next we test ridge regression on the two real datasets, again with 10-fold cross validation resampling. We conduct a similar grid search in to that used on the Franke function, going over polynomial degree 1-5 and lambdas from 1e-4 to 1. The results are shown in Figure 21 for Iowa and Figure 22 for Pennsylvania. The lowest test MSE for the Iowa dataset is 0.17, and

is at a somewhat-unexpected location near the bottom center with a lambda value of 2.6e-1 - unlike in our idealized tests on the Franke function where the lowest possible lambda value of 1e-4 was returned. This may mean that the regularization has a bigger impact on this real data than for the analytic function, which would be intuitive since real data is more likely to have outliers.

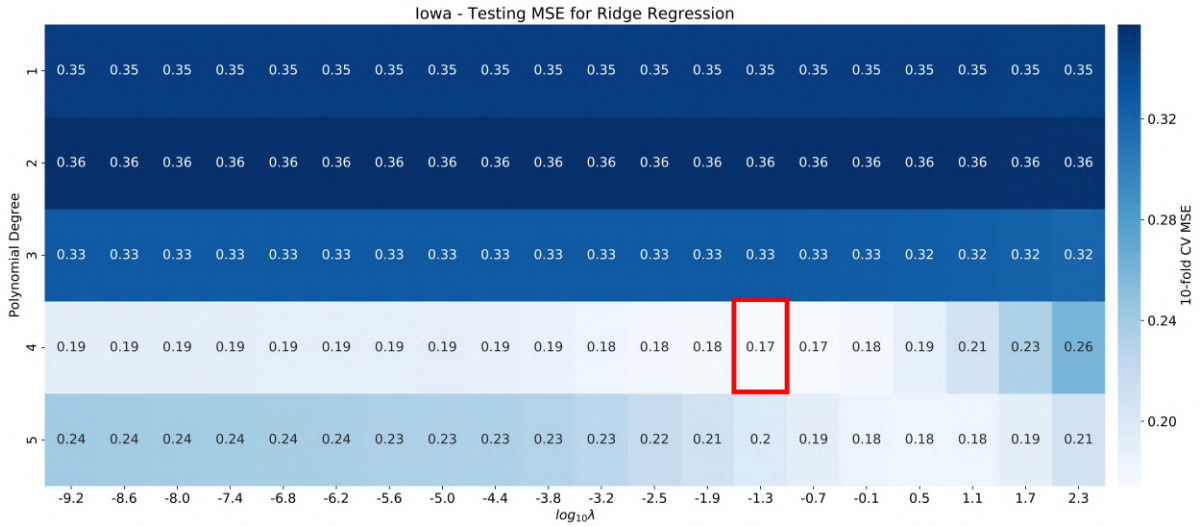

Figure 21: Iowa ridge test.

In the test on the Pennsylvania dataset, the MSEs are significantly higher, with the lowest being 0.5, again near the bottom center of the plot in Figure 22. The lowest MSE value corresponds to a model of order 5 and lambda value of 1.3e-2.

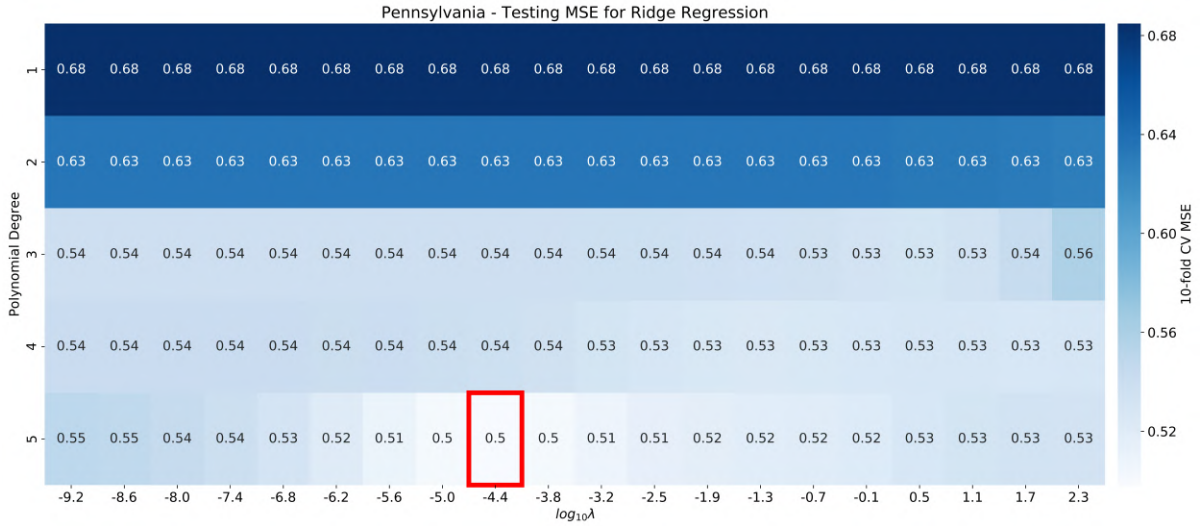

Figure 22: Pennsylvania ridge test.

Finally we test lasso regression (again with 10-fold CV for resampling) on both of the real

datasets. The results are shown in Figure 23 for Iowa and Figure 24 for Pennsylvania. The lowest MSE value for Iowa is 0.19 corresponding to order 5 and alpha 1.8e-4, and the lowest for Pennsylvania is 0.53 corresponding to the same order and regularization. Because this is the lowest possible alpha value in our grid search, we find that the regularization has less impact than in the case of ridge regression. Therefore it is likely that this model and result is relatively similar to OLS.

Figure 23: Iowa lasso test.

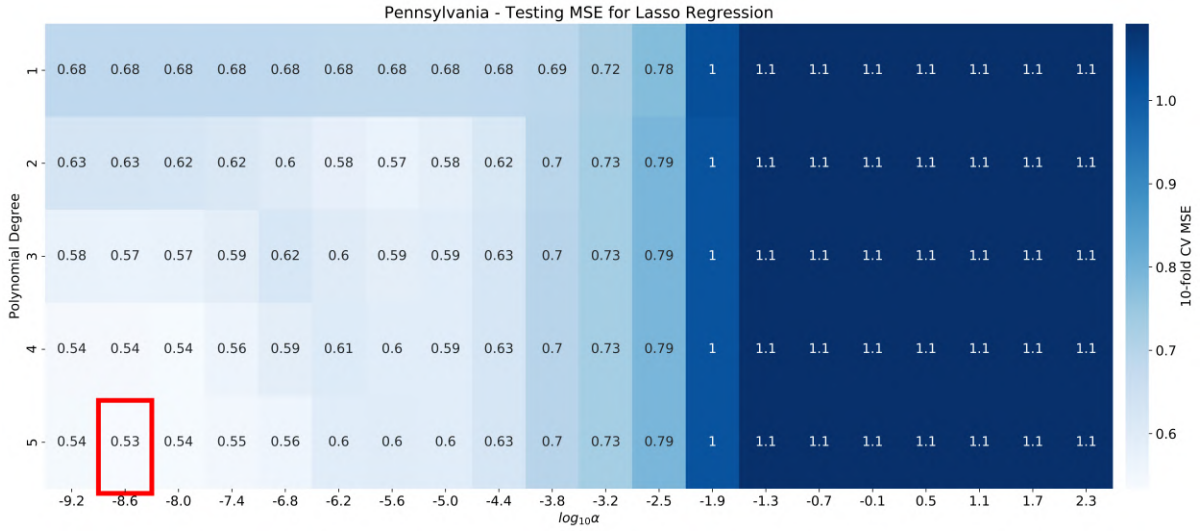

Figure 24: Pennsylvania lasso test.

Finally we compare the testing MSE results for our best models on the real data in Table 2. The results show that the ridge regression models of 5th order and relatively higher regularization parameters actually return the lowest error. Therefore it may be that the regularization is

having some small positive effect in the ridge regression, reducing the impact of large β parameters. However the result is again not much improved over low-order (3-4) ordinary least squares. And in terms of lasso regression, it returns essentially the same MSE values as for OLS. Therefore the choice for the best model will likely be informed by the needs of the user: do they need the lowest error at all costs? In this case they may choose the L2-regularized ridge regression to seek some small improvement. However if there are other practical considerations important to the user, such as the simplicity and interpretability of the model, then OLS may be the best choice as its results are almost as good.

	Iowa			Pennsylvania		
	OLS	Ridge	Lasso	OLS	Ridge	Lasso
Best Model Params	5th order	5th order, lambda = 2.6e-1	5th order, alpha = 1.8e-4	5th order	5th order, lambda = 1.3e-2	5th order, alpha = 1.8e-4
Best Test MSE	0.19	0.17	0.19	0.54	0.50	0.53

Table 2: Best models for real datasets.

Conclusions

Through this project we presented an overview of the three main regression methods: ordinary least squares, ridge, and lasso regression, and tested them on an analytic function and real topography data. We also implemented two resampling methods, the bootstrap and cross validation, in order to gain greater confidence about our model testing results. We found that low polynomial order-models (3-5) for ordinary least squares, or ridge regression with low values of the regularization parameter yielded the best results in terms of mean squared error on the testing data. In practice the user may choose a regularized ridge method in order to seek some small improvement in accuracy, or the standard OLS method to seek simplicity and efficiency, depending on their needs.

References

Bhattacharyya, S., 2018, Ridge and Lasso Regression: L1 and L2 Regularization, https://towardsdatascience.com/ridge-and-lasso-regression-a-complete-guide-with-python-scikit-learn-e20e34bcbf0b.

Efron, B. and R.J. Tibshirani, 1993, "An Introduction to the Bootstrap," Chapman and Hall, https://doi.org/10.1201/9780429246593.

Franke, R., 1979, "A Critical Comparison of Some Methods for Interpolation of Scattered Data," Naval Postgraduate School Reports, NPS-53-79-003, http://hdl.handle.net/10945/35052.

James, G., D. Witten, T. Hastie, and R. Tibshirani, 2013, "An Introduction to Statistical Learning: with Applications in R," Springer, https://doi.org/10.1007/978-1-4614-7138-7.

Kohavi, R., 1995, "A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection," Proceedings of the Fourteenth International Conference on Artificial Intelligence, https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.48.529.

Refaeilzadeh, P., L. Tang, and H. Liu, 2009, "Cross-Validation," in Encyclopedia of Database Systems, Springer, https://doi.org/10.1007/978-0-387-39940-9_565.

United States Geological Survey, 2022, Shuttle Radar Topography Mission (SRTM) Data, https://earthexplorer.usgs.gov/.

van Wieringen, W.N., 2021, "Lecture Notes on Ridge Regression," https://doi.org/10.48550/arXiv.1509.09169.