Regularized Least Squares

Stella Vannier, Gabe Wallon, Connor Wilson

APPM 4600 Final Project

August 19, 2024

Abstract

Regularization Methods (also known as Shrinkage Methods) are used to strike a balance between model bias and variance, and address computational issues that arise when using ordinary least squares (OLS) using data with certain properties. We provide background on OLS, and the circumstances which cause it to suffer from numerical instability. The Ridge, Tikhonov, LASSO, and Elastic Net Regularization Methods prevent over-fitting by penalizing the magnitude of model coefficients, thereby shrinking model coefficients towards zero. LASSO can also be used for feature selection, as it may shrink coefficients to zero, rendering them out of the model. Elastic Net regularization combines Ridge and LASSO to allow for more variable control. We provide derivations for each method and then demonstrate their utility in the context of predicting housing prices when constrained to a limited quantity of data - except for Tikhonov, which would produce the same results as Ridge in this case. The large quantity of predictor variables available in this data set and the presence of multicollinearity emphasize the importance of regularization. We manually implement OLS and Ridge in Python, as well as the K-Fold Cross Validation method for choosing model parameters. However, we use functions from the scikit-learn Python library to corroborate our results and implement LASSO and Elastic Net. When constraining the amount of data at our disposal, simulating a real-world constraint, we find that our house-price prediction model adjusted R-squared accuracy metric can be increased by 32 percentage points after leveraging Ridge regularization. We conclude by comparing the different methods' performance.

1 Introduction

1.1 Regression and Ordinary Least Squares

Regression is a ubiquitous technique used to draw insights from data. We can use available data to build a regression model, which learns, or is trained, from the given data. The regression model is then able to make predictions about future data from what it has learned. Ordinary Least Squares (OLS) regression is the most fundamental version of this technique. OLS assumes the relationship between the predictor variables (the known independent variables), and the response variable (the dependent variable we are trying to predict) is linear.

1.2 Derivation of Ordinary Least Squares

Consider an over-determined linear system, such that $\mathbf{Ax} = \mathbf{b} + \mathbf{e}$, where there is some error, $\mathbf{e}$, in the solution. This leads to the linear model no longer having an exact solution. However, the solution can be approximated by minimizing the sum of the squared errors and solving for the coefficient vector. The solution of OLS will be derived here, step by step, and then stated throughout the rest of the paper. *Note:* $\mathbf{A}$ *is* $n \times m$, $\mathbf{x}$ *is* $m \times 1$, $\mathbf{e}$ *and* $\mathbf{b}$ *are* $n \times 1$.
The equation above can be rearranged to isolate the error term on the right-hand side, then the square of the 2-norm is applied to both sides, creating the sum of the squared errors on the right-hand side of the equation.

$$\|\mathbf{Ax} - \mathbf{b}\|_2^2 = \|\mathbf{e}\|_2^2 \tag{1}$$

Using the definition of the 2-norm, $\|\mathbf{v}\|_2^2 = \mathbf{v}^T\mathbf{v}$, equation (1) can be rewritten as:

$$(\mathbf{Ax} - \mathbf{b})^T(\mathbf{Ax} - \mathbf{b}) = \mathbf{e}^T\mathbf{e} \tag{2}$$

Using properties of the transpose, namely $(\mathbf{Ax})^T = \mathbf{x}^T\mathbf{A}^T$, equation (2) can be expanded as follows.

$$\mathbf{x}^T\mathbf{A}^T\mathbf{Ax} - \mathbf{b}^T\mathbf{Ax} - \mathbf{x}^T\mathbf{A}^T\mathbf{b} + \mathbf{b}^T\mathbf{b} = \mathbf{e}^T\mathbf{e} \tag{3}$$

In order to minimize the sum of the squared errors, the partial derivative with respect to $\mathbf{x}$ is applied on both sides.

$$\frac{\partial}{\partial\mathbf{x}}(\mathbf{x}^T\mathbf{A}^T\mathbf{Ax} - \mathbf{b}^T\mathbf{Ax} - \mathbf{x}^T\mathbf{A}^T\mathbf{b} + \mathbf{b}^T\mathbf{b}) = \frac{\partial}{\partial\mathbf{x}}\mathbf{e}^T\mathbf{e} \tag{4}$$

On the left-hand side, $\mathbf{b}^T\mathbf{Ax}$ and $\mathbf{x}^T\mathbf{A}^T\mathbf{b}$ become scalars of equal magnitude, therefore they can be combined. On the right-hand side, since $\mathbf{e}$ is $m \times 1$ vector, $\mathbf{e}^T\mathbf{e}$ becomes a scalar, therefore taking the partial derivative with respect to $\mathbf{x}$ will set it equal to zero.

$$\frac{\partial}{\partial\mathbf{x}}(\mathbf{x}^T\mathbf{A}^T\mathbf{Ax} - 2\mathbf{x}^T\mathbf{A}^T\mathbf{b} + \mathbf{b}^T\mathbf{b}) = 0 \tag{5}$$

After applying the partial derivative in equation (5) and utilizing matrix calculus, $\mathbf{x}^T\mathbf{A}^T\mathbf{Ax}$ is treated as a quadratic in $\mathbf{x}$, such that the partial derivative equal to $2\mathbf{A}^T\mathbf{Ax}$, the partial derivative of $\mathbf{x}^T$ with respect to $\mathbf{x}$ is $\mathbf{1}$. Therefore, the partial derivative of equation (5) is:

$$2\mathbf{A}^T\mathbf{Ax} - 2\mathbf{A}^T\mathbf{b} = 0 \tag{6}$$

The next step is to add $\mathbf{A}^T\mathbf{b}$ to both sides to isolate the $\mathbf{x}$ term.

$$\mathbf{A}^T\mathbf{Ax} = \mathbf{A}^T\mathbf{b} \tag{7}$$

Lastly, multiply both sides by $(\mathbf{A}^T\mathbf{A})^{-1}$ in order to solve for $\mathbf{x}$

$$\mathbf{x}^* = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{b} \tag{8}$$

Thus, the approximate solution to $\mathbf{x}^*$ is $(\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{b}$ where $(\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T$ is the least squares estimator.

1.3 Motivation for Regularization Methods

In using OLS with certain data, the models produced can be at risk of "over-fitting", i.e., generalizing noise in the data as broader trends, which it will use to make predictions. This over-fit property of a model is often caused by training a model on a data set insufficient in size, i.e. not having enough data[15]. Over-fitting can also be caused by the inclusion of large numbers of predictor variables in the model[15]. Furthermore, a large number of predictor variables will likely increase the chances of two or more of the predictor variables to be correlated with one another, what we call "collinearity". Data with high collinearity produce nearly linearly dependent columns in the design/data matrix (used in the computation of model coefficients), resulting in a nearly singular matrix. If the design matrix $\mathbf{A}$ is nearly singular, $\mathbf{A}^T\mathbf{A}$, referred to as the "Gram matrix", will also be nearly singular. This can be seen by observing the Singular Value Decomposition (SVD) of $\mathbf{A}$. Suppose the design matrix has the following SVD: $\mathbf{A} = \mathbf{U\Sigma V}^T$. Then it can be shown that $\mathbf{A}^T\mathbf{A}$ has the following SVD:

$$\begin{aligned}
\mathbf{A}^T\mathbf{A} &= (\mathbf{U\Sigma V}^T)^T(\mathbf{U\Sigma V}^T) \\
&= \mathbf{V\Sigma}^T\mathbf{U}^T\mathbf{U\Sigma V}^T \\
&= \mathbf{V\Sigma}^T\mathbf{I\Sigma V}^T \\
&= \mathbf{V\Sigma\Sigma V}^T \\
&= \mathbf{V\Sigma}^2\mathbf{V}^T
\end{aligned}$$

Since the data matrix $\mathbf{A}$ is nearly singular, the ratio between the largest and the smallest of its singular value will be large. $\mathbf{A}^T\mathbf{A}$, whose singular values are the squares of the singular values of $\mathbf{A}$, will therefore also have a large ratio between its largest and smallest singular values, and consequently be nearly singular. The condition number of $\mathbf{A}^T\mathbf{A}$ is precisely this ratio. Computing the model coefficients involves inverting this Gram matrix, and numerically inverting a matrix with a large condition number is unstable[12]. We illustrate this instability in the next section by introducing small perturbations into the data used to create our model. We observe how slightly changing the design/data matrix $\mathbf{A}$, will cause dramatic discrepancies between the resulting model's coefficients compared to the those of the unperturbed model.

Thus, models built from training data with a large number of predictor variables available relative to the number of observations or data points are highly susceptible to the issues that arise from the OLS procedure.

It also may be possible that our data has more predictor variables than data points. In this case, the OLS problem becomes "ill-posed", or a unique solution to the equation does not exist. In this case, there would be infinitely many models which perfectly fit the training data. Another way we can classify a problem as "ill-posed" is if the singular values of $\mathbf{A}$ decrease to exactly zero. In this case, the condition number of this matrix is infinite, and the matrix is non-invertible.

In these cases, where OLS falls short, there is a need to reformulate the method with the goal of building more accurate models achieving numerical stability. This motivates the regularization methods Tikhonov (a.k.a Ridge), LASSO, and Elastic Net.

1.4 Real World Application

To highlight the pitfalls and successes of each regularization method, we will apply the methods to a data set containing information about housing prices in King County, Washington[7]. Buying property is widely considered one of the best investments one can make. One can see how it would be valuable to gain an understanding of the causality relationships between house attributes and price. For example, say you are thinking about buying a house, and you are interested in the value of your property rising over time and are deciding between two neighborhoods. Knowing how attributes of the neighborhood (e.g., crime rate, proximity to schools or business district, air quality) could cause the property value to rise or decline and to what extent would be valuable for you to make the smartest choice. However, one can imagine when it would be beneficial for a model to predict house prices as accurately as possible, at the cost of increased model complexity and less reliable coefficient estimates in the inference sense. For example, you could verify the quoted price from a real estate agent, and feel confident it accurately represents the property's value. Although both inference and prediction accuracy are desirable in this context, we choose to focus on model accuracy.

We now perform some preliminary explorations of our data. Figure 1 shows the distributions of all the variables; descriptions of all the variables can be found in the appendix.

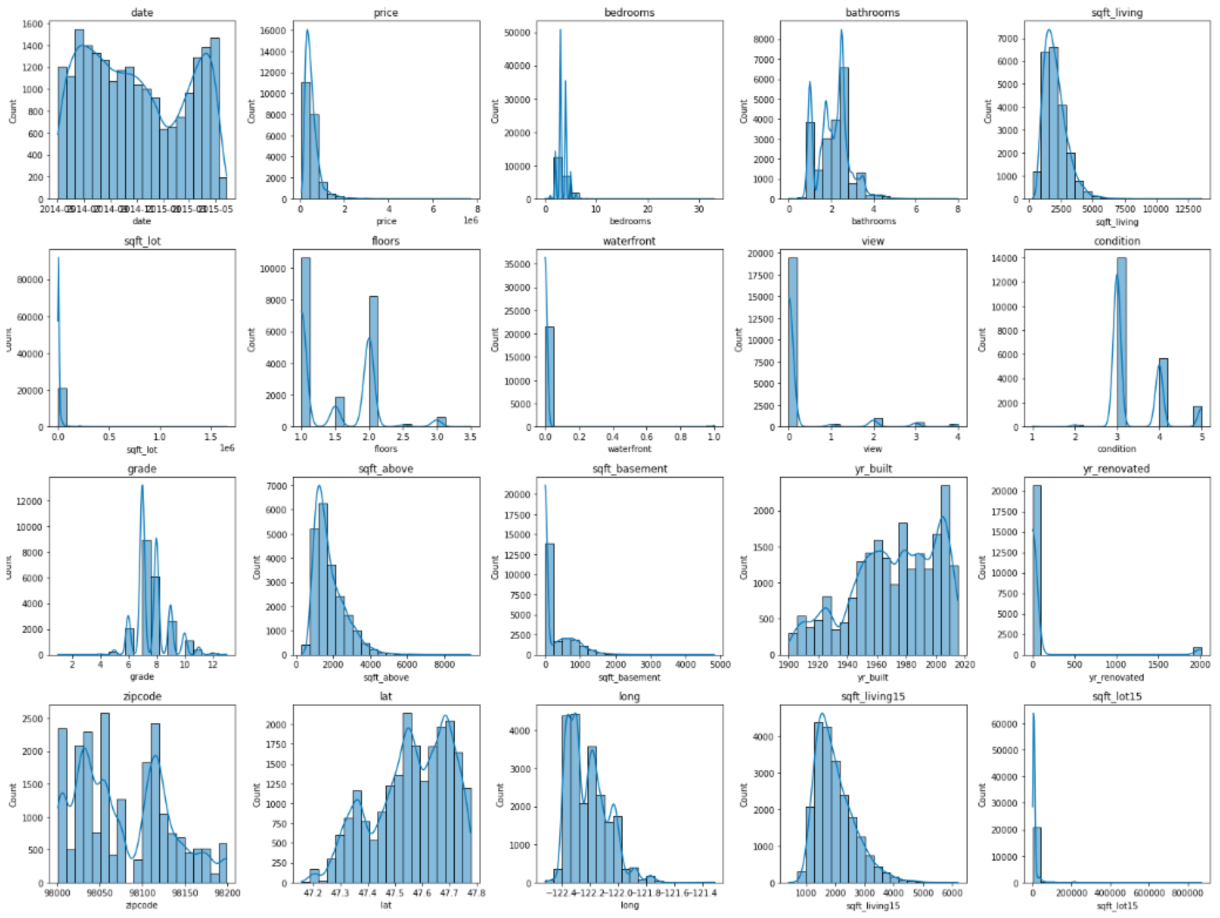


Figure 1: Distributions of the housing data variables

From these plots, we learn some basic information about our data; for example, we notice that the majority of the houses in this area were built within the last 50 years. We can also spot outliers in the data; such as, it appears that we have a house in our data with a suspiciously large number of bedrooms.

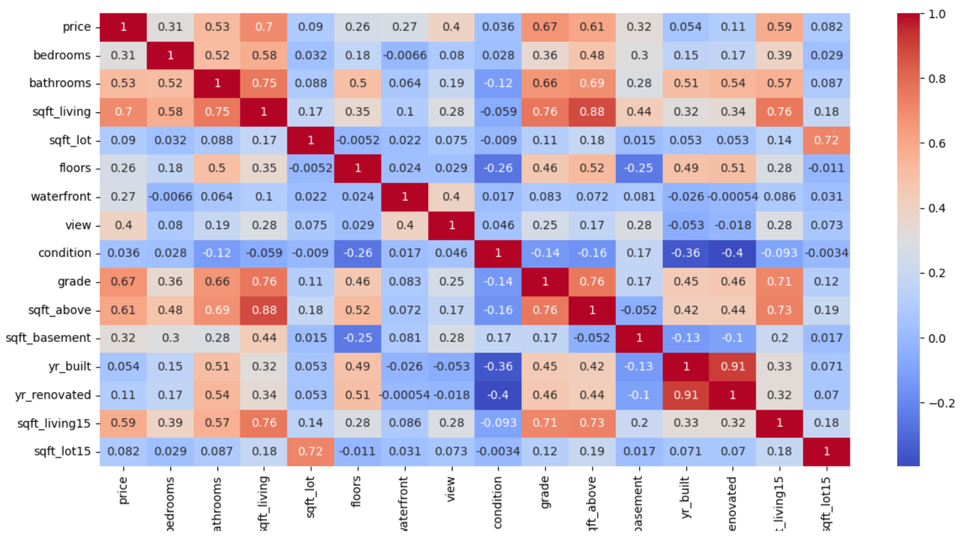


Figure 2: Correlations between numeric variables

Figure 2 displays a correlation matrix for our numeric variables. The strong correlation between 'yr_built' and 'yr_renovated' suggests that we do not need to add them both to our model. Indeed, when pre-processing the data (see Pre-Processing section in appendix), in order to consolidate the two categories of information, we changed the 'yr_renovated' values to equal the 'yr_built' for houses that never underwent a renovation. Thus the 'yr_renovated' column now holds information about the most recent time construction or renovation was performed on the house. We will keep this in mind and expect to see the regularization methods dramatically shrink the coefficient for one of these variables. This matrix offers other interesting takeaways, e.g., our data suggests that the location of a house on a waterfront does not strongly correlate to an increase in its price. However, correlations can also exist between three or more predictor variables, which we cannot observe from this correlation matrix. This is referred to as "multicollinearity".

We will use multiple linear regression to predict the prices of the houses in this data set. We justify this choice by observing approximate linear relations between the response and each of the predictor variables. Two such relations are depicted in figure 3.

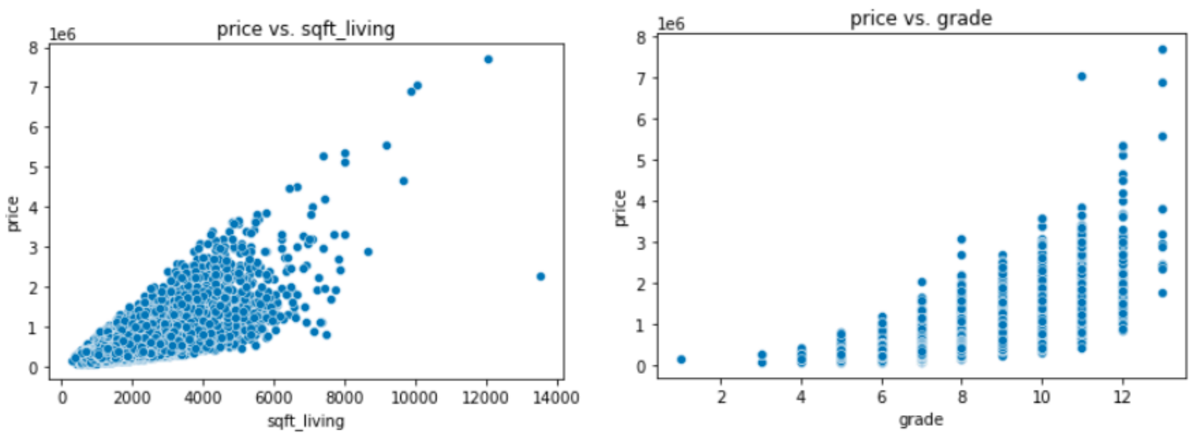
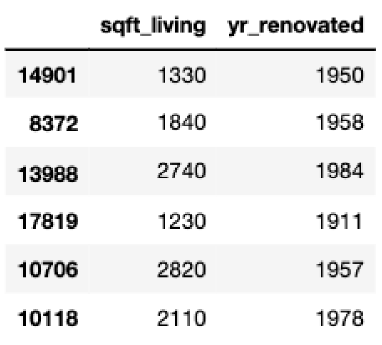


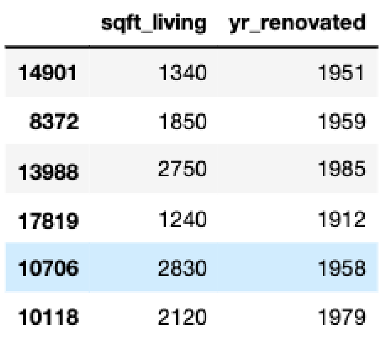
Figure 3: Correlations between price and predictor variables

We are interested in learning which attributes of a house affect its selling price. We believe this application will be an illuminating example of the usefulness of regularization methods because many variables can influence the price of a house. This large quantity of predictors increases the likelihood of over-fitting and thus would likely allow one to see a great increase in model performance with regularization methods. Furthermore, several of these predictor variables could have high collinearity, e.g., the square footage of the indoor living space, and the indoor living space above ground completely determine the square footage of the house's basement. This is an example of a column in our data set that is linearly dependent on two others. This multicollinearity can lessen the ability of the model to generalize and cause numerical instability[12].

To illustrate this numerical instability, let's observe the condition number of the product of the transpose of our design matrix with itself, $\mathbf{A}^T\mathbf{A}$, when we use the following features as predictor variables: 'bedrooms', 'bathrooms', 'sqft_living', 'sqft_lot', 'floors', 'waterfront', 'view', 'condition', 'grade', 'sqft_above', 'sqft_basement', 'yr_built', 'yr_renovated', 'sqft_living15', 'sqft_lot15', and 'zipcode'. We compute $\text{cond}(\mathbf{A}^T\mathbf{A}) \approx 1.34 \cdot 10^{23}$, certainly larger than the reciprocal of our machine precision (on the order of 10^{16}). Now let's make some minor changes to our training data, and observe if these minor changes in input, will result in large changes in the coefficients created by the OLS method, our output. We choose to train our model on 80% of our data, thus our model is learning from 17,290 data points. Of these 17,290 observations, we randomly choose 20, roughly 0.1%, and add 10 to the 'sqft_living' variable, and 1 to the 'yr_renovated' variable. Figure 4 shows snapshots of the data before and after making these minor adjustments.



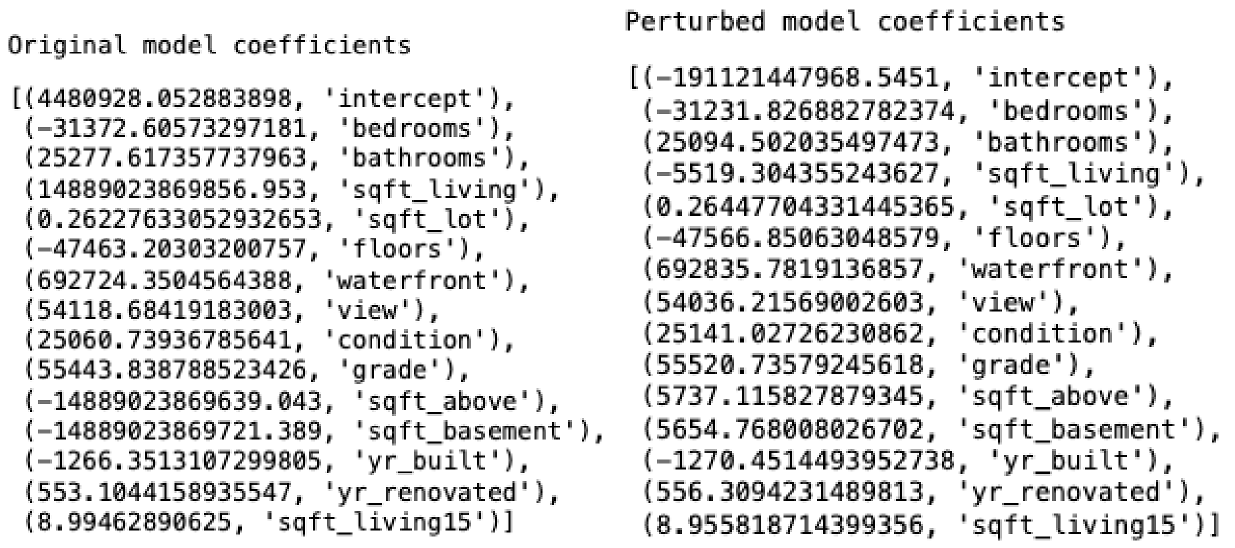
	sqft_living	yr_renovated
14901	1330	1950
8372	1840	1958
13988	2740	1984
17819	1230	1911
10706	2820	1957
10118	2110	1978



	sqft_living	yr_renovated
14901	1340	1951
8372	1850	1959
13988	2750	1985
17819	1240	1912
10706	2830	1958
10118	2120	1979

Figure 4: Before and after making minor changes to data

We use this new data to build an OLS model, and compare the coefficients to the OLS model from the original data. Indeed, we observe large discrepancies between the two sets of coefficients, displayed in figure 5.



```
Original model coefficients

[(4480928.052883898, 'intercept'),
 (-31372.60573297181, 'bedrooms'),
 (25277.617357737963, 'bathrooms'),
 (14889023869856.953, 'sqft_living'),
 (0.26227633052932653, 'sqft_lot'),
 (-47463.20303200757, 'floors'),
 (692724.3504564388, 'waterfront'),
 (54118.68419183003, 'view'),
 (25060.73936785641, 'condition'),
 (55443.838788523426, 'grade'),
 (-14889023869639.043, 'sqft_above'),
 (-14889023869721.389, 'sqft_basement'),
 (-1266.3513107299805, 'yr_built'),
 (553.1044158935547, 'yr_renovated'),
 (8.99462890625, 'sqft_living15')]
```

```
Perturbed model coefficients

[(-191121447968.5451, 'intercept'),
 (-31231.826882782374, 'bedrooms'),
 (25094.502035497473, 'bathrooms'),
 (-5519.304355243627, 'sqft_living'),
 (0.26447704331445365, 'sqft_lot'),
 (-47566.85063048579, 'floors'),
 (692835.7819136857, 'waterfront'),
 (54036.21569002603, 'view'),
 (25141.02726230862, 'condition'),
 (55520.73579245618, 'grade'),
 (5737.115827879345, 'sqft_above'),
 (5654.768008026702, 'sqft_basement'),
 (-1270.4514493952738, 'yr_built'),
 (556.3094231489813, 'yr_renovated'),
 (8.955818714399356, 'sqft_living15')]
```

Figure 5: Illustration of the effect of a large condition number. The coefficients from the original and the perturbed model are compared and displayed as (coefficient, variable name).

The original model predicts that, while holding all other predictor variables constant, with every 1 square foot increase in indoor living space, we should expect the price of the house to increase by $1.49 \cdot 10^{13}$ dollars. The model made from the perturbed data predicts that, while holding all other predictor variables constant, for every 1 square foot increase in indoor living space, we should expect the price of the house to decrease by 5,519 dollars. The multicollinearity in our data and the resulting large condition number of our data matrix cause these fluctuations in coefficient values[10].

At this point, it is important to recall that the goal of our model development is more geared toward making accurate predictions rather than inferring causal relationships from the model coefficients. The inclusion of collinear variables in the model can lead to less intuitive coefficients; however, including these variables will also likely increase the predictive power of the model. We would like for our coefficients to be somewhat intuitive but we are not too concerned if our model resembles a "black box". It is not intuitive that such slight changes in data would cause massive changes in the relationships between the variables, as observed in Figure 5. This would lead us to be less confident in inferring any causal relationships. We hope that applying the regularization methods will help to address this problem. However, we understand that the primary goal of our model is to make accurate predictions, even if it means sacrificing inference power.

Now, we evaluate the accuracy of our model using the R-squared metric. R-squared measures the percent of the variability in the dependent variable that can be explained by the independent variables in the model. We also use the Adjusted R-squared metric, which differs from R-squared in that it takes into account the total number of predictor variables and penalizes the inclusion of predictors with little explanatory power. Larger values correspond to models with stronger predictive power. We calculate the adjusted R-squared of our OLS model to be approximately 0.818. This score implies that our model can explain about 82% of the variability in the prices of houses. This is a strong score, suggesting that although the multicollinearity is creating instability in the calculation of coefficients, due to the large volume of data, OLS is not creating an over-fit model. Upon choosing a smaller subset of our data, we can observe that the accuracy of our model decreases. When subsetting the data to only 500 observations, we see our accuracy drop to roughly 0.45 (see the Subsetting Data code in the appendix).

1.5 sklearn

Scikit-learn or sklearn is a python library that includes algorithms for most regularization methods, including ridge regression, LASSO, and elastic net. There are also cross validation functions specific to each method. These functions take all of the heavy lifting off of the user. The cross validation functions take in a list of possible parameter values, and preform the cross validation to find the optimal value. We will use this library throughout our analysis.

1.6 K-Fold Cross Validation

K-Fold cross-validation is a common method for deciding the optimal value for a parameter in a regression model, although that is not its only function. The steps of K-fold cross-validation are outlined here:

1. Reserve a random portion of the data for the purpose of evaluating the accuracy of the final model. This data is called the "holdout set" and should never be used to train the model. The remaining data will be used to train the model and is referred to as the "training set".
2. Randomly partition the training set into K folds, or disjoint sets.
3. Select a parameter value γ to be used in the subsequent steps.
4. Use K-1 of these training set folds to train a model, with parameter value γ. Test this model on the remaining training set fold, compute a measurement of the models' accuracy, and store this metric. Repeat this step K times, so by the end, K models have been created and tested, and you have stored K performance metrics for the respective models.
5. Take the average of the performance metrics for the K models corresponding to the parameter value γ and store this average performance metric.
6. Select the next parameter value γ you are interested in, and repeat steps 4 and 5 for this parameter value.
7. Identify the parameter value γ with the strongest average performance metric as the optimal value for this parameter, γ_0.
8. Use the entire training set to train a model using this optimal parameter value γ_0 and test the model's accuracy using the holdout set.

There are functions in the scikit-learn machine learning Python library to either fully perform k-fold cross-validation, or to aid in a manual implementation. We used one of these library functions in our otherwise manual implementation (see code in appendix).

2 Ridge Regression

Ridge regression is a linear regression technique that extends the idea of ordinary least squares by adding a bias, also known as, a penalization term. The penalization term acts as a regulator to the slope of the best-fit line, meaning as the penalization term increases, the slope of the line decreases. This happens because the penalization term shrinks all coefficients proportional to the coefficients' magnitude. That is, larger coefficients get penalized more than smaller coefficients. Introducing the bias leads to a bias-variance trade-off. The penalization term controls the bias-variance trade-off; if the penalization term is small, it allows the model to fit the data more closely, which could result in a higher variance. However, increasing the penalization term could reduce the variance. We use variance here as it is used in machine learning jargon, meaning the difference in model accuracy observed when the model is used to make predictions on new sets of data. Hence, reducing the models' variance reduces the extent to which it is over-fit to the training data. This penalty term serves to reduce the instability of computation, by decreasing the condition number of the $\mathbf{A}^T\mathbf{A}$ matrix that arises in the computation of model coefficients. Cross-validation techniques are used to determine the optimal value for the penalty term. This is accomplished by testing multiple penalty values that result in the best balance between bias and variance. The range of penalty terms from which you test should be large at first, then narrowed iteratively[10].

2.1 Numerical Derivation

Ridge regression uses OLS with an additional penalization term $\gamma\|\mathbf{x}\|_2^2$. Similarly to OLS the goal of Ridge regression is to find an approximate solution by minimizing the error. Beginning with the original system with with the penalization term.

$$
\begin{aligned}
&\|\mathbf{Ax}-\mathbf{b}\|_2^2+\gamma\|\mathbf{x}\|_2^2 = \|\mathbf{e}\|_2^2\\
&(\mathbf{Ax}-\mathbf{b})^T(\mathbf{Ax}-\mathbf{b})+\gamma\mathbf{x}^T\mathbf{x} = \mathbf{e}^T\mathbf{e}\\
&\frac{\partial}{\partial\mathbf{x}}(\mathbf{x}^T\mathbf{A}^T\mathbf{Ax}-2\mathbf{x}^T\mathbf{A}^T\mathbf{b}+\mathbf{b}^T\mathbf{b}+\gamma\mathbf{x}^T\mathbf{x}) = \frac{\partial}{\partial\mathbf{x}}\mathbf{e}^T\mathbf{e}\\
&2\mathbf{A}^T\mathbf{Ax}-2\mathbf{A}^T\mathbf{b}+2\gamma\mathbf{x} = 0\\
&\mathbf{A}^T\mathbf{Ax}+\gamma\mathbf{xI} = \mathbf{A}^T\mathbf{b}\\
&\mathbf{x}(\mathbf{A}^T\mathbf{A}+\gamma\mathbf{I}) = \mathbf{A}^T\mathbf{b}\\
&\mathbf{x}^* = (\mathbf{A}^T\mathbf{A}+\gamma\mathbf{I})^{-1}\mathbf{A}^T\mathbf{b}
\end{aligned}
$$

Similar to OLS, where $(\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T$ is the least squares estimator, $(\mathbf{A}^T\mathbf{A}+\gamma\mathbf{I})^{-1}\mathbf{A}^T$ is the Ridge estimator, E_{Ridge}. Therefore, the approximate solution is $\mathbf{x}^* = E_{Ridge}\mathbf{b}$. As we will show below, this adjustment to the OLS estimator aids in the stability of calculating model coefficients. If $\mathbf{A}^T\mathbf{A}$ is an ill-conditioned matrix with a large condition number, $\mathbf{A}^T\mathbf{A}+\gamma\mathbf{I}$ will have a smaller condition number, thus increase numerical stability. Due to this penalization term in regularization methods, when the columns of your data matrix $\mathbf{A}$ vary in scale, the resulting coefficients may be penalized in an "unfair" fashion. The penalization of coefficients corresponding to variables measured on a small scale (such as the number of bedrooms in a house), will be influenced by the scaling of your other variables. Thus, if another one of your variables takes values in the hundreds or thousands (such as square feet of a house), the magnitude of the bedrooms' coefficient penalization will be too large. Therefore, it is considered best practice to standardize the predictor variables before fitting any model. This ensures that the final model will not depend on the scale on which the predictors are measured[10].

2.2 Application 1: Numerical Example

To demonstrate the increased stability when using Ridge regression, we have provided a small example using the simple linear function, $y = 3x + 2$. We randomly chose 20 points along the line ranging from $x = 0$ to $x = 5$, added Gaussian noise, and applied Ridge regression on 10 points to obtain an estimated function. The other 10 points are used to test the model, referred to as the validation set. The point of using noisy data to test the model is to simulate a real-world scenario where the true relationship of the given data is unknown. To do this, we calculated the sum of squared errors (SSE) on the validation set. The SSE is precisely what the regression model is built to minimize between its predictions and the training data and is frequently used to evaluate how accurate a model is able to make predictions on the validation set. We performed k-fold cross-validation by repeating this process with increasing values of gamma, leading us to find the optimal value. For details, see the cross-validation section above. Figure 6 shows the 10 points used to train the model, and the resulting function.

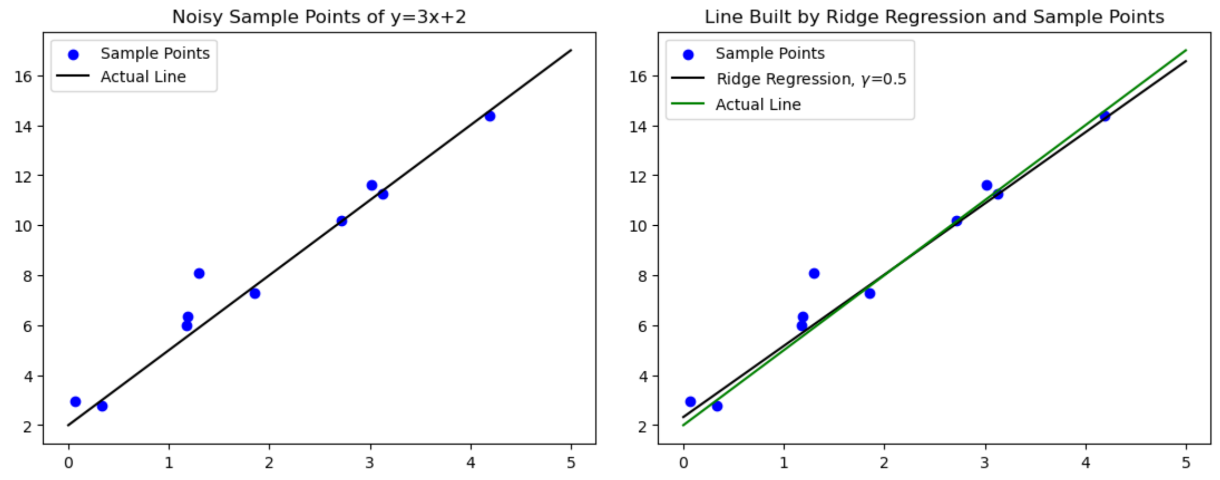


Figure 6: Noisy sample points of the line $y = 3x + 2$ and Ridge approximation

With these points and cross-validation, we found the optimal value of gamma to be, $\gamma = 0.5$. This lead to $\beta_0 = 2.32508$ and $\beta_1 = 2.84811$, thus resulting in the function $y = 2.32508 + 2.84811x$. To test the accuracy of this approximation, we have plotted the error between our model and the true function, $y = 3x + 2$.

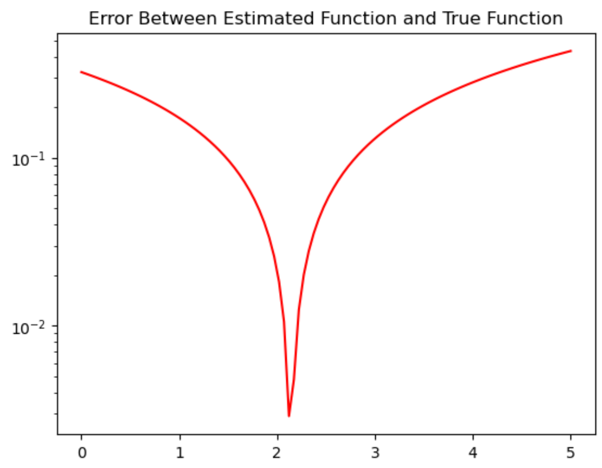


Figure 7: Error between Ridge approximation and true function

During our repeated tests with different samples of points, we found that even slight changes in the sample points led to different optimal choices for gamma. For example, with the following sample of points, the best value for gamma in the range $(0, 1)$ was $\gamma = 1.0$, and the resulting function was $y = 0.82430 + 3.48641x$.

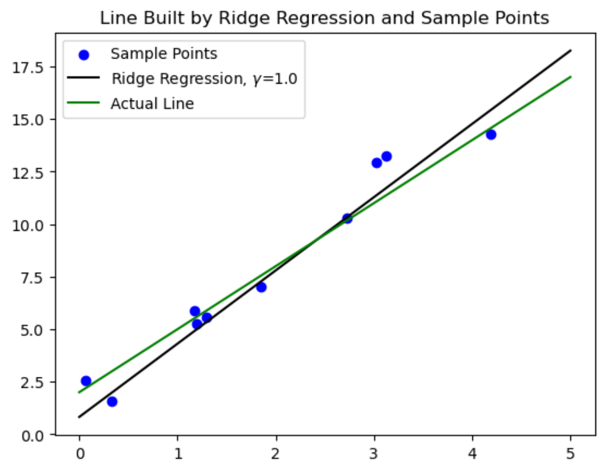


Figure 8: Ridge approximation on separate sample points

For very noisy points, the optimal value of gamma is usually large. This is due to the fact that the sum of squared errors is large, causing the optimal value for the penalization term to increase. To further understand the abilities of Ridge regression, we completed a similar process with the function $y = x^2$. For this example, we created a 5th-order polynomial and used points of x within the interval $(-2, 2)$. After cross-validation, the optimal value of gamma was found to be $\gamma = 0.6$, resulting in the function,

$$y = -0.048x^5 + 0.167x^4 + 0.103x^3 + 0.605x^2 - 0.208x + 0.112$$

Below we have provided the estimated function and the points used in Ridge regression, as well as the error between our model and the true function, $y = x^2$.

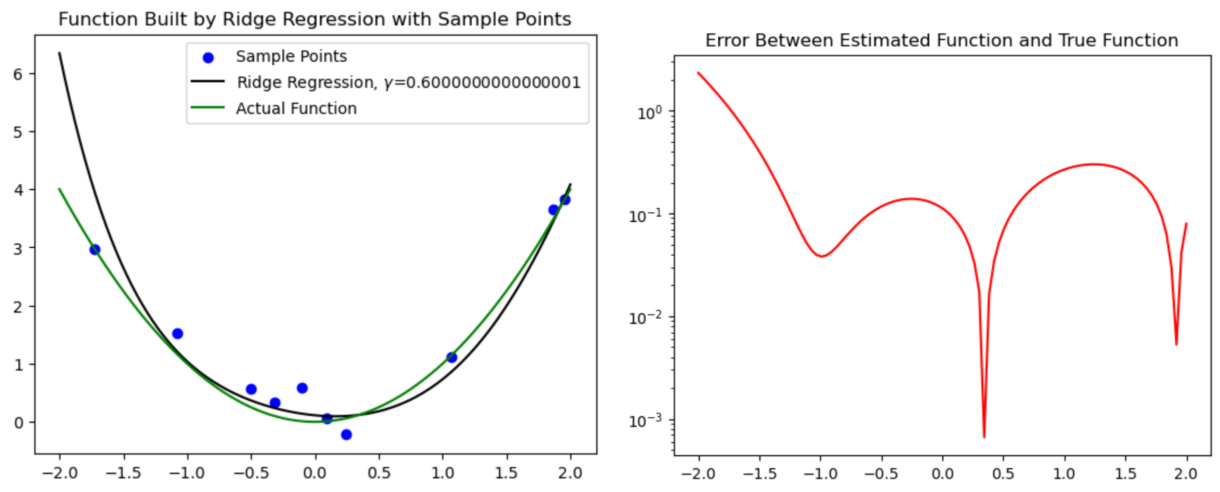
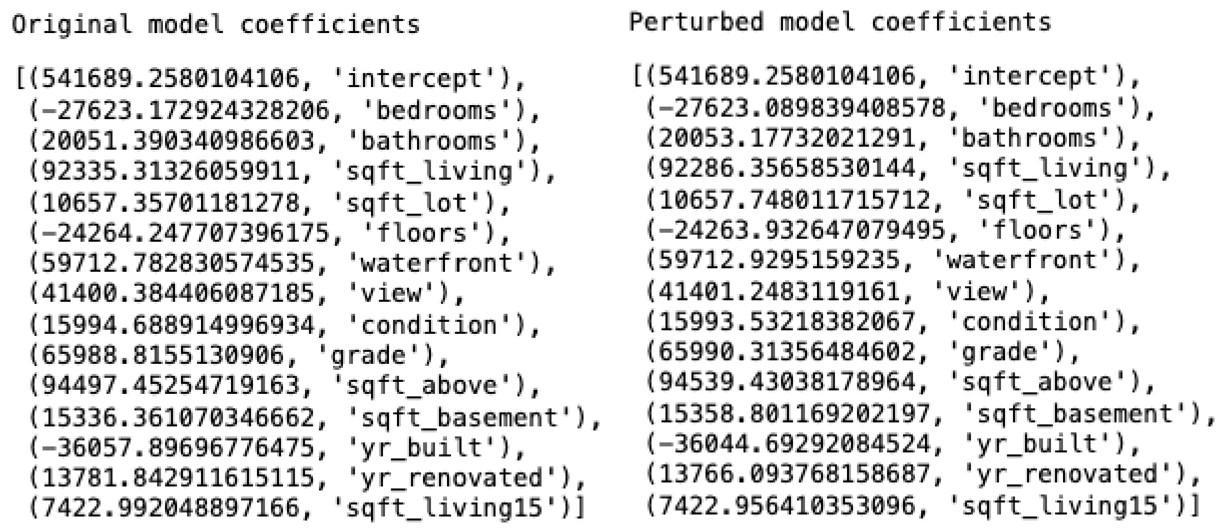


Figure 9: Ridge approximation of $y = x^2$ and error plot

2.3 Application 2: House Price Predictions

We now apply Ridge regression to create a house price prediction model using our King County Washington housing data set. We will only use a 500 observation subset of our data to help illustrate the utility of the Ridge method compared to OLS.

We perform 10-fold cross-validation to determine the optimal parameter value γ and use the R-squared metric for the evaluation of each model. We find the optimal value of γ to be approximately 123.44, while using the following features as predictor variables: 'bedrooms', 'bathrooms', 'sqft_living', 'sqft_lot', 'floors', 'waterfront', 'view', 'condition', 'grade', 'sqft_above', 'sqft_basement', 'yr_built', 'yr_renovated', 'sqft_living15', 'sqft_lot15', and 'zipcode'. We compute $\text{cond}(\mathbf{A}^T\mathbf{A}+123.44\mathbf{I}) \approx 4.00 \cdot 10^{11}$, where $\mathbf{A}$ is our data matrix consisting of all of our original 21613 observations. This condition number is smaller than the reciprocal of our machine precision (on the order of 10^{16}), and much smaller than the condition number of our Gram matrix used in the OLS calculations. This decrease in condition number results in more stability in the Gram matrix used in the calculations of the model coefficients. We can demonstrate this increased stability by repeating the experiment from section 1.4: perturbing 20 of our 17290 observations, creating a new model using this perturbed data, and comparing the resulting coefficients to those of our original model. We introduce the same perturbations as displayed in Figure 4 into a copy of our training data, then train two Ridge models on each of the training sets. Figure 10 displays the coefficients of the resulting models.



```
Original model coefficients

[(541689.2580104106, 'intercept'),
 (-27623.172924328206, 'bedrooms'),
 (20051.390340986603, 'bathrooms'),
 (92335.31326059911, 'sqft_living'),
 (10657.35701181278, 'sqft_lot'),
 (-24264.247707396175, 'floors'),
 (59712.782830574535, 'waterfront'),
 (41400.384406087185, 'view'),
 (15994.688914996934, 'condition'),
 (65988.8155130906, 'grade'),
 (94497.45254719163, 'sqft_above'),
 (15336.361070346662, 'sqft_basement'),
 (-36057.89696776475, 'yr_built'),
 (13781.842911615115, 'yr_renovated'),
 (7422.992048897166, 'sqft_living15')]
```

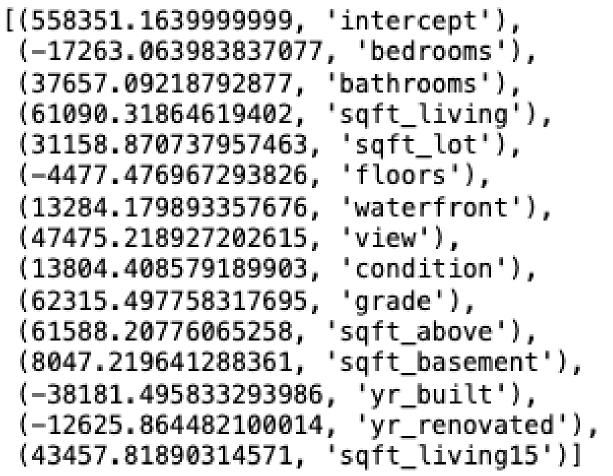
```
Perturbed model coefficients

[(541689.2580104106, 'intercept'),
 (-27623.089839408578, 'bedrooms'),
 (20053.17732021291, 'bathrooms'),
 (92286.35658530144, 'sqft_living'),
 (10657.748011715712, 'sqft_lot'),
 (-24263.932647079495, 'floors'),
 (59712.9295159235, 'waterfront'),
 (41401.2483119161, 'view'),
 (15993.53218382067, 'condition'),
 (65990.31356484602, 'grade'),
 (94539.43038178964, 'sqft_above'),
 (15358.801169202197, 'sqft_basement'),
 (-36044.69292084524, 'yr_built'),
 (13766.093768158687, 'yr_renovated'),
 (7422.956410353096, 'sqft_living15')]
```

Figure 10: Coefficients from the original and the perturbed Ridge models are compared, displayed as (coefficient, variable name).

As expected, the smaller condition number used in the Ridge procedure results in higher stability. The slight perturbations introduced in the input, did not result in large perturbations in the output. As we can see, the two models above have very similar coefficients. Figure 11 displays some of the coefficients and the predictor variables from our final model of the form:

$$\text{price} = \beta_0 + \beta_1\text{bedrooms} + \beta_2\text{bathrooms} + \ldots + \beta_{15}\text{sqft\_lot15} + \ldots + \varepsilon_i$$



```
[(558351.1639999999, 'intercept'),
 (-17263.063983837077, 'bedrooms'),
 (37657.09218792877, 'bathrooms'),
 (61090.31864619402, 'sqft_living'),
 (31158.870737957463, 'sqft_lot'),
 (-4477.476967293826, 'floors'),
 (13284.179893357676, 'waterfront'),
 (47475.218927202615, 'view'),
 (13804.408579189903, 'condition'),
 (62315.497758317695, 'grade'),
 (61588.20776065258, 'sqft_above'),
 (8047.219641288361, 'sqft_basement'),
 (-38181.495833293986, 'yr_built'),
 (-12625.864482100014, 'yr_renovated'),
 (43457.81890314571, 'sqft_living15')]
```

Figure 11: Ridge house price prediction model coefficients and corresponding variables

The coefficients are more stable, which should allow us to feel more confident in inferring causal relationships between the predictor variables and the response. However, there is still collinearity affecting their values, thus we still cannot make these connections. As we can observe in Figure 11, some of the coefficients suggest relationships between the predictor and response, which seem unintuitive, e.g., that as the number of bedrooms in a house increases, all else held constant, the value of the house would decrease. The next regularization technique, LASSO, will allow us to completely remove irrelevant predictors from the model, which will hopefully help the model become more interpretable.

Recall that the OLS model adjusted R-squared metric was approximately 0.453 when trained on the 500 data point subset. The adjusted R-squared metric of our Ridge model trained on the same data is 0.777. This illustrates how the introduction of bias into the model has led to a major increase in predictive power and generalizability.

3 Tikhonov's Regularization

3.1 Motivation

We now consider a generalized form of Ridge regression. Suppose we wanted to find an approximate solution to a linear system of equations $\mathbf{Ax} \approx \mathbf{b}$ where the data is contaminated with noise to the point where the problem is no longer well-posed. Instances of such problems arise when summing a Fourier series with imprecisely given coefficients, or when discretizing a Fredholm integral equation with the second derivative operator as a kernel[6]. Such problems require Tikhonov regularization for a solution to be found. A benefit of using Tikhonov to solve such problems is that it can enforce the smoothness of the solution vector by approximating first or second derivative information. Consequently, this method is often deployed when discretizing inverse problems[6]. If the solution obtained by Tikhonov is expected to have certain properties, which may be the case if solving a problem in a physics context, then the weight matrix $\mathbf{\Gamma}$ (see derivation below), can be chosen to meet this specific goal[5]. Our housing price prediction data set is not contaminated by noise. The prices of each house and the values of the response variable are fixed dollar amounts, and measurements of the independent variables, such as square feet or year built, we assume to be precise. Further, we do not have any expectations for the solution, as we do not expect to draw causal inferences from the model coefficients. Thus, we would choose our weight matrix $\mathbf{\Gamma}$ to be a constant multiplied by the identity, specifically $\sigma\mathbf{I}$ (see derivation below), and would end up minimizing the same quantity as in Ridge. Thus, for the sake of avoiding redundancy, we omit applying Tikhonov to our house price prediction problem.

3.2 Derivation

As mentioned above, Tikhonov regularization is a generalized version of Ridge. Therefore, Tikhonov can be rewritten from Ridge. Starting with Ridge, $\|\mathbf{Ax}-\mathbf{b}\|_2^2+\gamma\|\mathbf{x}\|_2^2 = \|\mathbf{e}\|_2^2$, we can consider a new variable $\sigma = \sqrt{\gamma}$. Ridge can be rewritten with the form: $\|\mathbf{Ax}-\mathbf{b}\|_2^2+\|\sigma\mathbf{Ix}\|_2^2 = \|\mathbf{e}\|_2^2$. Tikhonov considers a more general form $\|\mathbf{Ax}-\mathbf{b}\|_2^2+\|\mathbf{\Gamma x}\|_2^2 = \|\mathbf{e}\|_2^2$, where $\mathbf{\Gamma}$ is a weight matrix, and is often recast as $\|\mathbf{Ax}-\mathbf{b}\|_2^2+\lambda^2\|\mathbf{Dx}\|_2^2 = \|\mathbf{e}\|_2^2$ where $\mathbf{D}$ is a derivative matrix. Since first and second derivative information is unknown, it can be approximated by finite difference techniques, i.e., Taylor expanding around each point in $\mathbf{x}$. The Tikhonov estimator can be determined using the same method that determined the Ridge estimator. Some steps will be omitted here since this requires the same algebra used in least squares and Ridge derivation. Since λ^2 is an arbitrary value, setting $\lambda^2 = \gamma$ does not change the derivation.

$$\begin{aligned}
&\|\mathbf{Ax}-\mathbf{b}\|_2^2+\gamma\|\mathbf{Dx}\|_2^2 = \|\mathbf{e}\|_2^2\\
&\mathbf{x}^T\mathbf{A}^T\mathbf{Ax}-2\mathbf{x}^T\mathbf{A}^T\mathbf{b}+\mathbf{b}^T\mathbf{b}+\gamma\mathbf{x}^T\mathbf{D}^T\mathbf{Dx}=\mathbf{e}^T\mathbf{e}\\
&\frac{\partial}{\partial\mathbf{x}}(\mathbf{x}^T\mathbf{A}^T\mathbf{Ax}-2\mathbf{x}^T\mathbf{A}^T\mathbf{b}+\mathbf{b}^T\mathbf{b}+\gamma\mathbf{x}^T\mathbf{D}^T\mathbf{Dx})=\frac{\partial}{\partial\mathbf{x}}\mathbf{e}^T\mathbf{e}\\
&\mathbf{x}(\mathbf{A}^T\mathbf{A}+\gamma\mathbf{D}^T\mathbf{D})=\mathbf{A}^T\mathbf{b}\\
&\mathbf{x}^*=(\mathbf{A}^T\mathbf{A}+\gamma\mathbf{D}^T\mathbf{D})^{-1}\mathbf{A}^T\mathbf{b}
\end{aligned}$$

The Tikhonov estimator is defined as $(\mathbf{A}^T\mathbf{A}+\gamma\mathbf{D}^T\mathbf{D})^{-1}\mathbf{A}^T$ where $\gamma=\lambda^2$.

To determine $\mathbf{D}$, different finite difference techniques can be applied to approximate derivatives at each point in $\mathbf{x}$. Here, one derivative matrix will be derived, namely, the first derivative, centered finite difference matrix. Two more derivative matrices will be stated, yet not derived because all derivations follow the same pattern. Assume there is a point x_n in the interval needed to approximate the function $y(x_n)$; a Taylor expansion can be created for $y(x_n)$ evaluated at x_{n+1} and again for $y(x_n)$ evaluated at x_{n-1}, where the step size between x_n and $x_{n\pm1}$ is $h=1$.

Solving for $y(x_{n+1})$

$$\begin{aligned}
y(x_{n+1})&=y(x_n)+y'(x_n)(x_{n+1}-x_n)+\frac{1}{2}y''(x_n)(x_{n+1}-x_n)^2+\cdots\\
y(x_{n+1})&=y(x_n)+y'(x_n)h+\frac{1}{2}y''(x_n)h^2+\mathcal{O}(h^3)\\
y'(x_n)&=\frac{1}{h}[y(x_{n+1})-y(x_n)-\frac{1}{2}y''(x_n)h^2-\mathcal{O}(h^3)]
\end{aligned}$$

Solving for $y(x_{n-1})$

$$\begin{aligned}
y(x_{n-1})&=y(x_n)+y'(x_n)(x_{n-1}-x_n)+\frac{1}{2}y''(x_n)(x_{n-1}-x_n)^2+\cdots\\
y(x_{n-1})&=y(x_n)-y'(x_n)h+\frac{1}{2}y''(x_n)h^2-\mathcal{O}(h^3)\\
y'(x_n)&=\frac{1}{h}[-y(x_{n-1})+y(x_n)-\frac{1}{2}y''(x_n)h^2+\mathcal{O}(h^3)]
\end{aligned}$$

Now that both equations are in the form $y'(x_n)=\cdots$, the system of equations can be used to solve for $y'(x_n)$ in terms of $y(x_{n+1})$ and $y(x_{n-1})$.

Solving the system of equations

$$\begin{aligned}
&y'(x_n)=\frac{1}{h}[y(x_{n+1})-y(x_n)-\frac{1}{2}y''(x_n)h^2-\mathcal{O}(h^3)]\\
+\quad &y'(x_n)=\frac{1}{h}[-y(x_{n-1})+y(x_n)-\frac{1}{2}y''(x_n)h^2+\mathcal{O}(h^3)]\\
&y'(x_n)=\frac{1}{2}[y(x_{n+1})-y(x_{n-1})] \qquad \text{Recall } h=1
\end{aligned}$$

Thus, the $\mathbf{D}$ matrix with first derivative, centered finite difference approximation is:

$$\mathbf{D}=\begin{bmatrix}-\frac{1}{2} & 0 & \frac{1}{2} & 0 & \cdots & 0\\ 0 & -\frac{1}{2} & 0 & \frac{1}{2} & 0 & \vdots\\ \vdots & 0 & \ddots & \ddots & \ddots & 0\\ 0 & \cdots & 0 & -\frac{1}{2} & 0 & \frac{1}{2}\end{bmatrix}$$

where $\mathbf{D}$ is a $(n-2)\times n$ matrix. As stated above, the next two $\mathbf{D}$ matrices will be claimed but not derived.

First derivative, forward finite difference approximation

$$\mathbf{D} = \begin{bmatrix} -\frac{3}{2} & 2 & -\frac{1}{2} & 0 & \cdots & 0 \\ 0 & -\frac{3}{2} & 2 & -\frac{1}{2} & 0 & \vdots \\ \vdots & 0 & \ddots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & -\frac{3}{2} & 2 & -\frac{1}{2} \end{bmatrix}$$

Second derivative, centered finite difference approximation

$$\mathbf{D} = \begin{bmatrix} 1 & -2 & 1 & 0 & \cdots & 0 \\ 0 & 1 & -2 & 1 & 0 & \vdots \\ \vdots & 0 & \ddots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & 1 & -2 & 1 \end{bmatrix}$$

The use of a first-order finite difference matrix creates a new vector, $\mathbf{Dx}$, where each element is the difference between adjacent coefficients. If some elements are large, it means there is a rapid change between coefficients. Therefore, by applying the penalization term, these rapid changes will be shrunk, enforcing smoothness. The application of a second-order finite difference matrix not only penalizes sharp changes in adjacent points but rapid changes in the slope as well, thus encouraging smoothness and smooth changes between coefficients. Therefore, the selection of which finite difference matrix to use depends on the data and how much smoothing is required.

3.3 Numerical Example

To illustrate the usefulness of this method, consider data sampled from the curve $y = \sin x + \sin 5x$, with Gaussian noise added. Figure 12 shows the data we are working with.

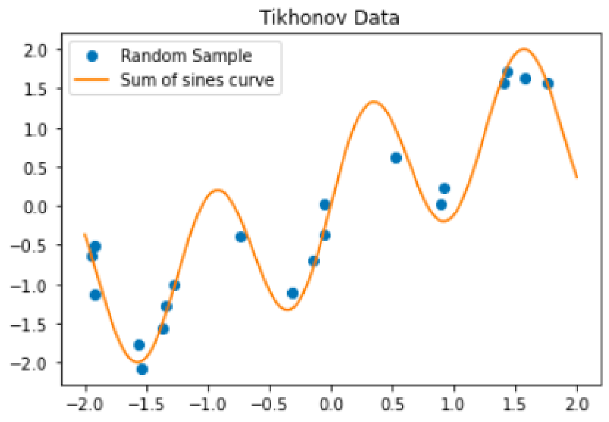


Figure 12: Tikhonov numerical example data

We perform K-fold cross-validation to determine the optimal value for the penalty parameter γ, using a 5th-order polynomial regression with SSE as the performance metric. We obtain an optimal penalty parameter $\gamma = 5.0$. Figure 13 shows the final model regressing through the data, on top of the distribution from which the data were sampled. We also see a plot depicting the error between the regression line and the target function/distribution.

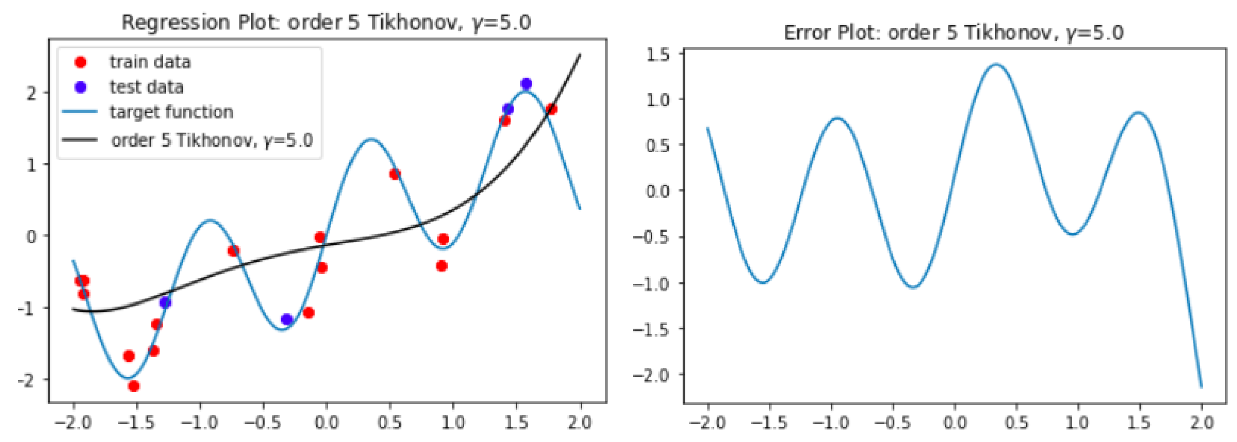


Figure 13: Tikhonov regression curve and error plot

We compare these results with the OLS 5th order polynomial regression line, shown in figure 14.

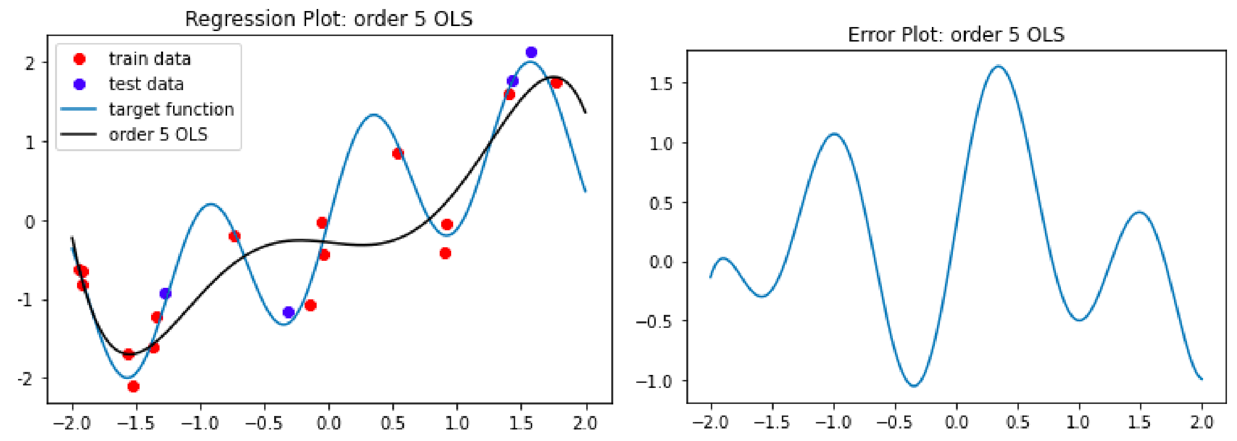


Figure 14: OLS regression curve and error plot

One can notice how the differences in regression curves appear between the two methods. With a penalty term of 5, the Tikhonov regression curve appears to have fewer fluctuations than the OLS curve. We also observe that the SSE for the Tikhonov regression, approximately 2.25, is larger than the SSE for the OLS regression, approximately 1.44. This example contradicts the bias-variance trade-off. The bias induced by the penalty term acts to shrink the model coefficients in magnitude, however, because of this shrinkage, the Tikhonov model should be better equipped to make predictions on new, "never before seen" data. However, we see that the OLS model performed better on this new data. We believe this is due to the willingness of OLS to produce coefficients with large magnitudes, paired with this specific data sample, which is relatively small, coming from this specific distribution, with oscillatory behavior. We also expect the norm of the coefficients vector produced by Tikhonov to be smaller in magnitude than that of OLS, and indeed, we observe this to be true. The norm of the coefficients vector produced by Tikhonov is roughly 0.40, while that produced by OLS is 1.17.

Using the other finite difference methods to calculate the derivative operator matrix produces the results shown in the figures. We can see these methods produce less accurate regression curves.

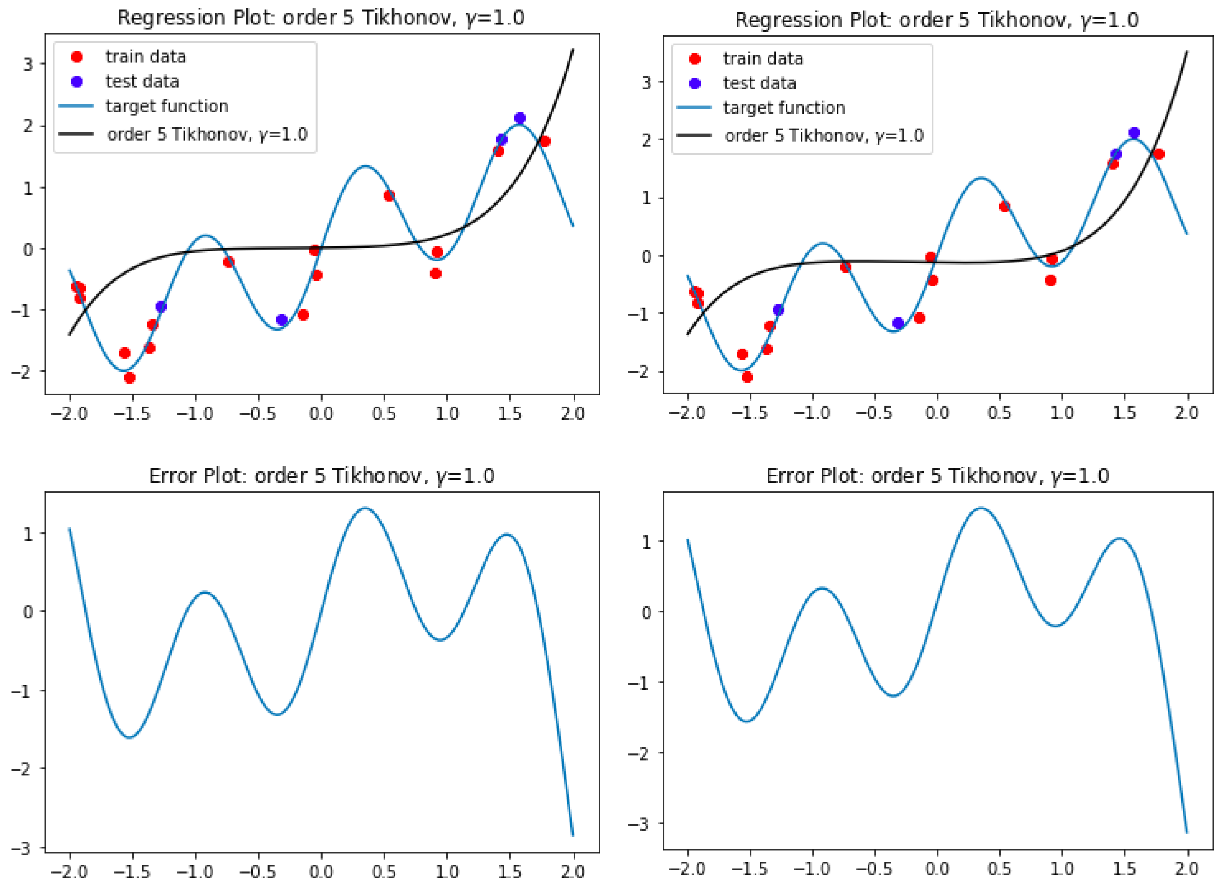


Figure 15: Tikhonov regressions using the first derivative, forward difference derivative operator approximation (left), and the second derivative, centered difference derivative operator approximation (right), with their respective error plots below.

4 LASSO

LASSO, or Least Absolute Shrinkage and Selection Operator regression, is similar to Ridge regression with one significant difference. Just as in Ridge regression, the goal of LASSO is to shrink the values of the coefficients towards zero to, in turn, minimize the sum of squared errors. While Ridge regression is subject to the penalization term $\gamma\|\mathbf{x}\|_2^2$, LASSO is subject to $\gamma\|\mathbf{x}\|_1$, using the 1-norm rather than the 2-norm. This slight but significant change gives LASSO the ability to reduce some coefficient values to zero, while Ridge can only bring coefficients close to zero. Having certain coefficients equal to zero makes for a sparse model that is easy to interpret because it omits redundant or unnecessary variables.

4.1 Derivation

Unlike Ridge regression and Tikhonov regularization, LASSO does not have a closed-form solution due to the l_1 penalization term. When minimizing LASSO, the solution will depend on the sign and value of the coefficients; therefore, LASSO must be minimized one term at a time. In matrix representation, LASSO takes the form $\|\mathbf{Ax} - \mathbf{b}\|_2^2 + \lambda\|\mathbf{x}\|_1 = \|\mathbf{e}\|_2^2$. Where $\|\mathbf{x}\|_1 = \Sigma_{i=1}^m |x_i|$. To minimize LASSO, consider minimizing the functions separately. Minimize the OLS function first, then the penalization term, and then combine them for a general solution for each coefficient. The minimization of OLS is derived in section 1.3

$$\frac{\partial}{\partial \mathbf{x}}(\text{OLS}) = 2\mathbf{A}^T\mathbf{Ax} - 2\mathbf{A}^T\mathbf{b}$$

The penalty term is minimized separately because taking the partial derivative of $|x|$ is invalid for $x = 0$. However, using the subgradient technique will allow for a piece-wise solution.

$$\frac{\partial}{\partial x_i}|\lambda x_i| = \begin{cases} -\lambda & \text{if } x_i < 0 \\ [-\lambda, \lambda] & \text{if } x_i = 0 \\ \lambda & \text{if } x_i > 0 \end{cases}$$

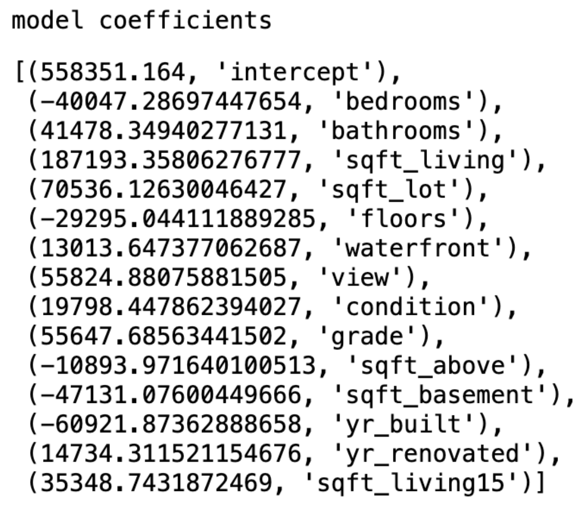
For the case of $x_i = 0$ the subgradient technique states that there are infinitely many derivatives; therefore, the solution contains all slopes between $-\lambda$ and $+\lambda$. The solution will be stated here, though the derivation is in the appendix[4].

$$x_i^* = \begin{cases} ((\mathbf{A}^T\mathbf{b})_i + \frac{\lambda}{2}\mathbf{I})/(\mathbf{A}^T\mathbf{A})_i & \text{if } (\mathbf{A}^T\mathbf{b})_i < \frac{-\lambda}{2} \\ 0 & \text{if } (\mathbf{A}^T\mathbf{b})_i \text{ in } [\frac{-\lambda}{2}, \frac{\lambda}{2}] \\ ((\mathbf{A}^T\mathbf{b})_i - \frac{\lambda}{2}\mathbf{I})/(\mathbf{A}^T\mathbf{A})_i & \text{if } (\mathbf{A}^T\mathbf{b})_i > \frac{\lambda}{2} \end{cases}$$

Feature selection can be seen in the solution with $x_i^* = 0$, which corresponds to features that are less important in the model. Since these features do not contribute much to the model, they are shrunk down by the penalty term. Depending on the value of λ, these less important features can be completely eliminated from the model.

4.2 Application: House Price Predictions

We now utilize the built-in LASSO function in the sklearn library to implement LASSO to create a housing price prediction model using the King County housing data. Again, for illustration, we use the same 500 observations from the previous analysis to show how LASSO is performs in comparison to Ridge and OLS. Similarly to Ridge, we use 10-fold cross-validation to calculate the optimal value for λ, using the R-squared metric on each evaluation. After the cross-validation, sklearn determines the optimal λ is approximately 11.11 and yields the following coefficients.



```
model coefficients

[(558351.164, 'intercept'),
 (-40047.28697447654, 'bedrooms'),
 (41478.34940277131, 'bathrooms'),
 (187193.35806276777, 'sqft_living'),
 (70536.12630046427, 'sqft_lot'),
 (-29295.044111889285, 'floors'),
 (13013.647377062687, 'waterfront'),
 (55824.88075881505, 'view'),
 (19798.447862394027, 'condition'),
 (55647.68563441502, 'grade'),
 (-10893.971640100513, 'sqft_above'),
 (-47131.07600449666, 'sqft_basement'),
 (-60921.87362888658, 'yr_built'),
 (14734.311521154676, 'yr_renovated'),
 (35348.7431872469, 'sqft_living15')]
```

Figure 16: LASSO house price prediction model coefficients and corresponding variables

While these coefficients do not tell us much on their own, we can compare the adjusted R-squared values given by this model to the previous models. The adjusted R-squared metric of the LASSO model is 0.752. While this is not an improvement over Ridge, it is similar to Ridge and, therefore, an improvement over OLS. The improvements over OLS further show that the introduction of bias into the model and applying the ability to eliminate irrelevant predictors improves the predictive power.

As discussed earlier, a larger penalization term, λ, leads to smaller variable coefficients and, in some cases of LASSO, coefficients of zero. Within the housing data, the optimal value of lambda was relatively small. Thus, none of the variable coefficients were zero. Yet, with a data set that contains a large number of redundant or useless predictors, we would see many variable coefficients being brought to zero. LASSO does not perform well when there is a group of variables that display a high correlation. LASSO cannot perform group selection, meaning it would pick one variable to represent the groups' effect on the response variable. This can lead to poor predictions of future data. LASSO performs best with a moderate level of collinearity but suffers when collinearity is high[8].

5 Elastic Net

5.1 Motivation

Elastic Net regression is another regularization method that combines the penalties of LASSO and Ridge regression. Ridge regression is useful for data sets where all predictor variables are helpful in the prediction, whereas LASSO is useful for data sets with useless or redundant predictor variables. However, there are drawbacks to Ridge and LASSO. Although Ridge regression effectively trades off between bias and variance, the use of the 2-norm requires all variables to remain in the model. Even though LASSO can eliminate variables from the model, it is limited to the size of the sample, meaning if there are p number of predictor variables and n is the number of variables in the sample. If p is larger than n, LASSO will choose n number of predictor variables and discard the rest[12]. This is an issue if there is more than one highly correlated variable in a model because LASSO will randomly select one and reject the rest. To overcome this, Elastic Net was created to combine the strengths of Ridge and LASSO and negate the weaknesses.

5.2 Derivation

Since Elastic Net method combines LASSO and Ridge regression, both penalty terms are tacked onto the OLS equation, giving two separate lambdas, λ_1 for LASSO and λ_2 for Ridge, which lends to a highly adaptive estimator. A combination of λ_1 and λ_2 plays to the strengths of both Ridge and LASSO. However, if $\lambda_1 = 0$, the estimator is equivalently Ridge regression. On the other hand, when $\lambda_2 = 0$, the estimator represents LASSO. Lastly, when $\lambda_1 = \lambda_2 = 0$ the model is back to OLS.
Elastic Net can be formulated as

$$\|\mathbf{Ax} - \mathbf{b}\|_2^2 + \lambda_1\|\mathbf{x}\|_1 + \lambda_2\|\mathbf{x}\|_2^2$$

This formula can be equivalently written in a way that includes only one λ and one mixing term, α, where $\lambda = \lambda_1 + \lambda_2$, and $\alpha = \frac{\lambda_1}{\lambda_1+\lambda_2}$[12].

$$\|\mathbf{Ax} - \mathbf{b}\|_2^2 + \lambda[\alpha\|\mathbf{x}\|_1 + (1-\alpha)\|\mathbf{x}\|_2^2]$$

This reformulation allows for the contributions of Ridge and LASSO to be percentage-based. Meaning if $\alpha = 0.75$, then 75% of the penalization goes to LASSO, and 25% goes to Ridge. As with all of the previous regularization methods, the objective is to create the Elastic Net estimator by minimizing Elastic Net. However, the Elastic Net estimator will not be derived because it combines the minimization of OLS, Ridge, and LASSO, shown in sections 1.2, 2.1, and 4.1, respectively.

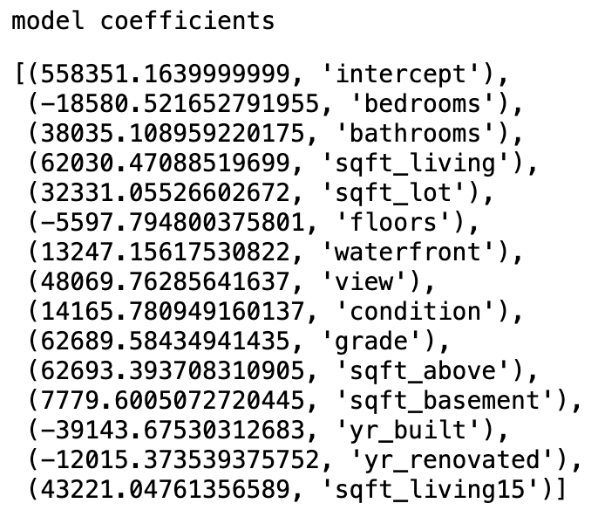
5.3 Application

Within the sklearn package, the implementation of Elastic Net follows the formula below

$$\frac{1}{2n}\|\mathbf{Ax} - \mathbf{b}\|_2^2 + \alpha[\text{l1\_ratio}\|\mathbf{x}\|_1 + \frac{\alpha}{2}(1 - \text{l1\_ratio})\|\mathbf{x}\|_2^2]$$

where α takes the place of λ in the derivation, and the l1_ratio represents α in the derivation. The terms $\frac{1}{2n}$ and $\frac{1}{2}$ are included for mathematical convenience. When minimizing the Elastic Net function, these terms cancel out nicely. The different notation between the derivation and sklearn lead to confusion between the two different α's, so for the rest of the explanation, only the notation from the derivation will be used. That is, α will denote the mixing parameter.

We now apply the built-in Elastic Net function from the sklearn library to the housing data for another housing price prediction. The method is similar to that of Ridge and LASSO where we use the 500 observations to demonstrate the performance. However, because Elastic Net uses a penalty term and a mixing parameter, we use 10-fold cross-validation to optimize both.



```
model coefficients

[(558351.1639999999, 'intercept'),
 (-18580.521652791955, 'bedrooms'),
 (38035.108959220175, 'bathrooms'),
 (62030.47088519699, 'sqft_living'),
 (32331.05526602672, 'sqft_lot'),
 (-5597.794800375801, 'floors'),
 (13247.15617530822, 'waterfront'),
 (48069.76285641637, 'view'),
 (14165.780949160137, 'condition'),
 (62689.58434941435, 'grade'),
 (62693.393708310905, 'sqft_above'),
 (7779.6005072720445, 'sqft_basement'),
 (-39143.67530312683, 'yr_built'),
 (-12015.373539375752, 'yr_renovated'),
 (43221.04761356589, 'sqft_living15')]
```

Figure 17: Elastic net house price prediction model coefficients and corresponding variables

After the cross-validation, sklearn found the best values for the penalty term and mixing parameter are, $\lambda = 22.22$ and $\alpha = 0.99$. This shows that for the housing data, sklearn heavily favors LASSO by suggesting the model uses 99% of LASSO and 1% of Ridge. After using the optimal values in the model, the adjusted R-squared metric for Elastic Net was found to be 0.778, a slight increase over Ridge and LASSO. Furthermore, this continues to show that using an adaptive estimator increases the predictions in the model.

6 Conclusion

Within the umbrella of regression, regularization methods allow for greater numerical stability and offer more favorable results depending on properties of the data. Ordinary least squares (OLS) is the most naive form of regression. It performs best with large data sets that have independence among the predictor variables and a constant variance[10]. Although the housing data set displayed a correlation between several predictor variables, OLS performed reasonably well due to the large number of observations. When we applied OLS to a subset of 500 observations, the performance decreased significantly, yielding an adjusted R-squared value of 0.45.

Ridge regression introduces a penalization term, λ, which helps against over-fit models. The penalization term decreases the condition number of the the Gram matrix $\mathbf{A}^T\mathbf{A}$, which in turn increases the numerical stability of the problem. Applying Ridge regression to our 500 observation sample of the housing data set gave us a model with increased stability compared to OLS. The Ridge model was not only more numerically stable, but yielded a greater adjusted R-squared value compared to that of OLS, i.e. it had greater prediction power. We were able to achieve and adjusted R-squared value of 0.77 with Ridge.

Next, we explored Tikhonov regularization, which is a general form of Ridge regression. Tikhonov can use a finite difference matrix instead of an identity matrix in the calculations. The implementation of the finite difference matrix aids in decreasing separation between adjacent points and sharp oscillations in the data, leading to increased smoothness in the solution. We show numerical examples comparing the performance of Tikhonov with OLS in a sinusoidal function. It was shown that the performance of Tikhonov varied with the different finite difference matrices used in the model. We believe this is due to the sinusoidal function not varying much in oscillation or separation between points. Since we do not desire smoothness in our housing model coefficients, we chose to not apply this method within this context.

The next method we explore is LASSO. LASSO regression is parallel to Ridge regression, except LASSO uses a l_1 penalization term rather than the l_2 penalization term used by Ridge. This change gives LASSO the ability to perform variable selection, allowing some coefficients to be rendered out of the model completely. In applying LASSO to our housing data sample, we saw an increase in performance over OLS, but not Ridge.

Elastic Net is a combination of Ridge and LASSO regression, using a penalization term, λ, and a "mixing" term α that distributes the amount of the penalization between Ridge (l_2) and LASSO (l_1). Since Elastic Net is a combination of the two regression methods stated, it performs well where the two are lacking. Ridge regression can lead to biased coefficients, which are influenced by the scaling of other variables. If there is a group of predictor variables that display high correlation, LASSO will randomly choose one of them to represent the group, which can lead to poor predictions[12]. That being said, Elastic Net performs well under these circumstances, specifically when the number of predictors is larger than the number of observations. Elastic Net performed the best in making predictions for our housing data. The optimal model of Elastic Net used a mixing term, or l1-ratio, of $\alpha = 0.99$, implying that the model heavily favored LASSO over Ridge. This is likely due to the moderate collinearity within the housing data.

Future work includes looking into other parameter tuning methods besides cross validation. Hyper-parameter tuning is a growing area of interest in machine learning that we would like to learn more about. Future work also includes exploring the Bayesian interpretations of the Lasso and Ridge regularization methods. This would allow us to see the statistical learning methods through the lens of this classical statistical theory.

References

[1] URL: https://aswani.ieor.berkeley.edu/teaching/SP15/265/lecture_notes/ieor265_lec6.pdf.

[2] URL: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.ElasticNet.html.

[3] Laura Dykes et al. URL: https://www.math.kent.edu/~reichel/publications/multiL.pdf.

[4] Emily Fox. URL: https://courses.cs.washington.edu/courses/cse446/17wi/slides/lasso.pdf.

[5] Martin Fuhry and Lothar Reichel. "A new Tikhonov regularization method". In: *Numerical Algorithms* 59.3 (2011), pp. 433–445. DOI: 10.1007/s11075-011-9498-x.

[6] Gene H. Golub, Per Christian Hansen, and Dianne P. O'Leary. "Tikhonov regularization and total least squares". In: *SIAM Journal on Matrix Analysis and Applications* 21.1 (1999), pp. 185–194. DOI: 10.1137/s0895479897326432.

[7] Harlfoxem. *House sales in King County, USA*. Aug. 2016. URL: https://www.kaggle.com/datasets/harlfoxem/housesalesprediction/data.

[8] Trevor Hastie, Jerome Friedman, and Robert Tisbshirani. *The elements of Statistical Learning: Data Mining, Inference, and prediction*. Springer, 2017.

[9] Thomas Huckle and Matous Sedlacek. "Data Based Regularization Matrices for the Tikhonov-Phillips Regularization". In: *PAMM* 12.1 (2012), pp. 643–644. DOI: https://doi.org/10.1002/pamm.201210310. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/pamm.201210310. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/pamm.201210310.

[10] Gareth James et al. *An introduction to statistical learning with applications in Python*. Springer International Publishing, 2023.

[11] Kashnitsky. *Topic 4. Linear models. part 1. OLS*. May 2021. URL: https://www.kaggle.com/code/kashnitsky/topic-4-linear-models-part-1-ols.

[12] Ryan W Lee, C. Fosco, and P. Protopapas. "Methods of regularization and their justifications". In: *Harvard IACS* (2019). DOI: https://harvard-iacs.github.io/2019-CS109A/a-section/a-section2/notes/A-sec2_Regularization.pdf.

[13] L E Melkumova and S Ya Shatskikh. *Comparing Ridge and lasso estimators for data analysis*. Sept. 2017. URL: https://www.sciencedirect.com/science/article/pii/S1877705817341474#section-cited-by.

[14] Andrew E. Yagle. URL: https://web.eecs.umich.edu/~aey/recent/regular.pdf.

[15] Xue Ying. "An overview of overfitting and its solutions". In: *Journal of Physics: Conference Series* 1168 (2019), p. 022022. DOI: 10.1088/1742-6596/1168/2/022022.

7 Appendix

7.1 Housing Introduction and OLS Code (Section 1.4)

7.1.1 Housing Data Variable Descriptions

date - Date of the home sale
price - Price of each home sold
bedrooms - Number of bedrooms
bathrooms - Number of bathrooms, where .5 accounts for a room with a toilet but no shower
sqft_living - Square footage of the apartments interior living space
sqft_lot - Square footage of the land space
floors - Number of floors
waterfront - A dummy variable for whether the apartment was overlooking the waterfront or not
view - An index from 0 to 4 of how good the view of the property was
condition - An index from 1 to 5 on the condition of the apartment
grade - An index from 1 to 13, where 1-3 falls short of building construction and design, 7 has an average level of construction and design, and 11-13 have a high quality level of construction and design.
sqft_above - The square footage of the interior housing space that is above ground level
sqft_basement - The square footage of the interior housing space that is below ground level
yr_built - The year the house was initially built
yr_renovated - The year of the house's last renovation
zipcode - What zipcode area the house is in
lat - Lattitude
long - Longitude
sqft_living15 - The square footage of interior housing living space for the nearest 15 neighbors
sqft_lot15 - The square footage of the land lots of the nearest 15 neighbors

7.1.2 Importing libraries

```python
# IMPORTS
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import math
import random
from numpy.linalg import inv
import seaborn as sns

from sklearn.linear_model import LinearRegression, Ridge, RidgeCV, LASSO,
   ElasticNet
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score
from sklearn.preprocessing import StandardScaler
```

7.1.3 Pre-Processing

```
# load data and change date column datatype
df = pd.read_csv("kc_house_data.csv", parse_dates=['date'])
pd.set_option('display.max_columns', None)

# remove the id column
df.drop("id" , axis=1 , inplace=True)

# make yr_renovated = yr_built if zero
mask = df.yr_renovated==0
df.loc[mask, 'yr_renovated'] = df.yr_built

# drop date of sale, latitude, logitude data
df = df.drop(columns=['date', 'lat', 'long'])

# create a dummy variable for zipcode
# change datatype to string
convert_dict = {'zipcode': str}
df = df.astype(convert_dict)
df = pd.get_dummies(df)

# change observation with 33 bedrooms to 3 as it was probably a typo
df.loc[df.bedrooms == 33, 'bedrooms'] = 3
```

7.1.4 EDA

```
# Variable Distributions (code chunk executed before the third step of
   pre-processing)
plt.figure(figsize=(20, 15))
for feature in df.columns.to_list():
    plt.subplot(4, 5, df.columns.to_list().index(feature) + 1)
    sns.histplot(data=df[feature], bins=20, kde=True)
    plt.title(feature)
plt.tight_layout()
plt.show()

# Correlation Matrix
newdf = df[df.columns[:17]]
plt.figure(figsize=(16,7))
sns.heatmap(newdf.corr(), cmap='coolwarm', annot=True)
plt.show()

# Correlations between predictors and response
features = ['sqft_living', 'grade']
for i in range(len(features)):
    sns.scatterplot(x=features[i], y='price', data=df)
    t = f"price vs. {features[i]}"
    plt.title(t)
    plt.show()
```

7.1.5 Condition number illustration: OLS

```
# Splitting the data
predictors = df[df.columns[1:]]
response = df['price']
A_train, A_test, b_train, b_test = train_test_split(predictors, response,
    train_size=.8, random_state=11)

# create a copy for perturbations
new_train = A_train.copy()
np.random.seed(11)
play_inds = np.random.randint(0,new_train.shape[0],20)
print("before perturbations")
print(new_train.iloc[play_inds[:6], [2,12]])
print()

# adding 10 sqft to 500 houses
new_train.iloc[play_inds, [2]] = new_train.iloc[play_inds, [2]] + 10
# adding 1 to yr_renovated 500 houses
new_train.iloc[play_inds, [12]] = new_train.iloc[play_inds, [12]] + 1
print("after perturbations")
print(new_train.iloc[play_inds[:6], [2,12]])
print()

new_OLSmodel = LinearRegression()
new_OLSmodel.fit(new_train, b_train)

OLSmodel = LinearRegression()
OLSmodel.fit(A_train, b_train)

var_names = ['intercept', 'bedrooms', 'bathrooms', 'sqft_living', 'sqft_lot',
       'floors', 'waterfront', 'view', 'condition', 'grade', 'sqft_above',
       'sqft_basement', 'yr_built', 'yr_renovated', 'sqft_living15',
       'sqft_lot15']
print('Perturbed model coefficients')
print(list(zip(([new_OLSmodel.intercept_] + list(new_OLSmodel.coef_))[:15],
    var_names)))
print()
print('Original model coefficients')
print(list(zip(([OLSmodel.intercept_] + list(OLSmodel.coef_))[:15], var_names)))
```

7.2 K-Fold Cross Validation Manual Implementation (Section 1.5)

```
# Cross validation
# reserve 10% of the data as a holdout, for testing final model
np.random.seed(11)
holdout_size = round(df.shape[0]*.1)
rand_inds = np.random.randint(0,df.shape[0],holdout_size)
holdout_data = df.iloc[rand_inds]

# remove holdout from data, will use this subset of data for cross-validation
train_folds = df.drop(list(rand_inds), axis=0, inplace=False)

# build K models for each gamma
gammas = np.linspace(0,1,11)
gamma_r2_lst = []
from sklearn.model_selection import KFold
kf = KFold(n_splits = 10, shuffle = True, random_state = 11)
for gamma in gammas:
    r2_lst = []
    for train_ind, test_ind in kf.split(train_folds):
#        print(sum(train_ind)) # can show that each training set is different
#        print(sorted(list(train_ind)+list(test_ind)) ==
    list(range(df.shape[0]))) # can show that the training and testing sets
    combine to equal the whole df

        train = df.iloc[train_ind,:]
        test = df.iloc[test_ind,:]

        b = train.price.to_numpy()
        A = train.drop(columns = 'price').to_numpy()
        coeffs = inv(A.transpose()@A + gamma * np.identity(86)) @ (A.transpose()
            @ b)

        test_prices = test.price.to_numpy()
        test_mat = test.drop(columns = 'price').to_numpy()
        price_preds = test_mat @ coeffs
        # Calculate the adjusted R-squared
        r_squared = 1 - (np.sum((test_prices - price_preds)**2) /
            np.sum((test_prices - np.mean(test_prices))**2))
        r2_lst.append(r_squared)

    avg_r2 = sum(r2_lst)/len(r2_lst)
    gamma_r2_lst.append((gamma,avg_r2))


# select optimal gamma corresponding to largest avg_adj_r2
min_tuple = max(gamma_r2_lst, key=lambda t: t[1])
opt_gamma = min_tuple[0]

# now that we have our optimal value of gamma, we can build a new model, using
    all of the training folds, and test
    # this optimal model on our holdout data to get an idea of its accuracy
train = train_folds
```

```
test = holdout_data
b = train.price.to_numpy()
A = train.drop(columns = 'price').to_numpy()
coeffs = inv(A.transpose()@A + opt_gamma * np.identity(86)) @ (A.transpose() @ b)
test_prices = test.price.to_numpy()
test_mat = test.drop(columns = 'price').to_numpy()
price_preds = test_mat @ coeffs
# Calculate the adjusted R-squared
r_squared = 1 - (np.sum((test_prices - price_preds)**2) / np.sum((test_prices -
    np.mean(test_prices))**2))
n = len(test_prices)
num_predictors = (len(coeffs)-1)
best_model_adj_r2 = 1 - ((1 - r_squared) * (n - 1) / (n - num_predictors - 1))
best_model_adj_r2
```

7.3 House Price Predictions (Section 2.3)

7.3.1 Condition number illustration: Ridge

```
# Splitting the data
predictors = df[df.columns[1:]]
response = df['price']
A_train, A_test, b_train, b_test = train_test_split(predictors, response,
    train_size=.8, random_state=11)

# create a copy for perturbations
new_train = A_train.copy()
np.random.seed(11)
play_inds = np.random.randint(0,new_train.shape[0],20)
print("before perturbations")
print(new_train.iloc[play_inds[:6], [2,12]])
print()

# adding 10 sqft to 500 houses
new_train.iloc[play_inds, [2]] = new_train.iloc[play_inds, [2]] + 10
# adding 1 to yr_renovated 500 houses
new_train.iloc[play_inds, [12]] = new_train.iloc[play_inds, [12]] + 1
print("after perturbations")
print(new_train.iloc[play_inds[:6], [2,12]])
print()

# standardize the data
scaler = StandardScaler()
scaler.fit(A_train)
stand_A_train= scaler.transform(A_train)

scaler2 = StandardScaler()
scaler2.fit(new_train)
stand_new_train= scaler2.transform(new_train)
```

```
alpha_range = np.linspace(118,125,10)

# create model with original data
Ridge_cv_model=RidgeCV(alphas=alpha_range, scoring='r2', cv=10)
Ridge_cv_model.fit(stand_A_train, b_train)

# create model with perturbed data
new_Ridge_cv_model=RidgeCV(alphas=alpha_range, scoring='r2', cv=10)
new_Ridge_cv_model.fit(stand_new_train, b_train)

var_names = ['intercept', 'bedrooms', 'bathrooms', 'sqft_living', 'sqft_lot',
      'floors', 'waterfront', 'view', 'condition', 'grade', 'sqft_above',
      'sqft_basement', 'yr_built', 'yr_renovated', 'sqft_living15',
      'sqft_lot15']
print('Original model coefficients')
print(list(zip(([Ridge_cv_model.intercept_] + list(Ridge_cv_model.coef_))[:15],
    var_names)))
print()
print('Perturbed model coefficients')
print(list(zip(([new_Ridge_cv_model.intercept_] +
    list(new_Ridge_cv_model.coef_))[:15], var_names)))
```

7.3.2 Calculating final model

```
# Splitting the data
predictors = df[subset_df.columns[1:]]
response = df['price']
A_train, A_test, b_train, b_test = train_test_split(predictors, response,
    train_size=.8, random_state=11)

# Selecting 1000 random observations
np.random.seed(11)
rand_inds = np.random.randint(0,A_train.shape[0],500)

# subsetting data
A_train_subset = A_train.iloc[rand_inds]
b_train_subset = b_train.iloc[rand_inds]
A_train = A_train_subset
b_train = b_train_subset

# standardize the data
scaler = StandardScaler()
scaler.fit(A_train)
stand_A_train= scaler.transform(A_train)
stand_A_test= scaler.transform(A_test)

# set range of values for penalty parameter
alpha_range = np.linspace(118,125,10)
```

```
# train model and evaluate
Ridge_cv_model=RidgeCV(alphas=alpha_range, scoring='r2', cv=10)
Ridge_cv_model.fit(stand_A_train, b_train)
y_preds = Ridge_cv_model.predict(stand_A_test)
Ridge_r2 = r2_score(b_test,y_preds)
print('Ridge Adj. R2:', adj_r2(A_test, Ridge_r2))
print()
print('Optimal gamma:', Ridge_cv_model.alpha_)
```

7.4 Tikhonov Numerical Example (Section 3.3) Code

```
# Line evaluation helper function
def line2(coeffs, x):
    summ = 0
    for i in range(len(coeffs)):
        summ += coeffs[i]*x**i
    return summ

# CROSS VALIDATION
# create a random sample from the distribution
np.random.seed(11)
f = lambda x: np.sin(x) + np.sin(5*x)
x = np.random.uniform(-2,2,20)
y = f(x)
noise = np.random.normal(0,.2,20)
noisy_y = y+noise
data = list(zip(x,noisy_y))

# partition data into 4 folds
random.seed(17)
random.shuffle(data)
folds = []
for i in range(5): # create 5 folds
    fold_i = []
    for j in range(4): #add 4 random data points from data to fold
        elt = random.choice(data)
        data.remove(elt) #remove element from list of data
        fold_i.append(elt)
    folds.append(fold_i) #add fold_i to fold list

holdout_fold = folds[-1]
folds = folds[:-1]

# test gamma values
gammas = np.linspace(0,10,11)
gamma_sse_lst = []
for gamma in gammas:
    sse_list = []
    # creates 4 models, for each test fold
    for i in range(4):
        x,y,z = folds[:i] + folds[i+1:]
        training = x+y+z
        testing = folds[i]
        trainx,trainy = zip(*training)
        testx,testy = zip(*testing)

        #generate coeffs and build regression
        n = len(training) # n is number of data points
        order = 5   # regression order

        # create A/design matrix
        A = np.ones((n,(order+1)))
        for i in range(n):
            for j in range((order+1)):
                A[i,j] = trainx[i]**j

        # create D matrix
        D = np.zeros(((order+1)-2,(order+1)))
        for i in range((order+1)-2):
            for j in range((order+1)):
                if i==j:
                    D[i,j] = -1/2
                if j==i+2:
                    D[i,j] = 1/2

        # create E matrix and model coeffs
        E = inv(A.transpose()@A + (gamma**2)*D.transpose()@D)@A.transpose()
        coeffs = E@trainy

        # calculate SSE
        yhat = np.zeros(len(testing))
        for i in range(len(testing)):
            yhat[i] = line2(coeffs,testx[i])
        sse = 0
        for j in range(len(testing)):
            sse += (yhat[j]-testy[j])**2

        # add sse to list
        sse_list.append(sse)

    avg_sse = sum(sse_list)/len(sse_list)
    gamma_sse_lst.append((gamma,avg_sse))

# select optimal gamma corresponding to smallest avg_sse
min_tuple = min(gamma_sse_lst, key=lambda t: t[1])
opt_gamma = min_tuple[0]
```

```
# now that we know the optimal gamma value, we create a new model using all of
    our training folds, and test its
    # accuracy on the holdout data (which wasn't used to train the model)
# test the accuracy of the final model on the holdout fold
w,x,y,z = folds
training = w+x+y+z
trainx,trainy = zip(*training)
testx,testy = zip(*holdout_fold)

#generate coeffs and build regression
n = len(training) # n is number of data points
order = 5   # regression order

# create A matrix
A = np.ones((n,(order+1)))
for i in range(n):
    for j in range((order+1)):
        A[i,j] = trainx[i]**j #populate A matrix

# create D matrix
D = np.zeros(((order+1)-2,(order+1)))
for i in range((order+1)-2):
    for j in range((order+1)):
        if i==j:
            D[i,j] = -1/2
        if j==i+2:
            D[i,j] = 1/2

# create E matrix and model coeffs
E = inv(A.transpose()@A + (gamma**2)*D.transpose()@D)@A.transpose()
coeffs = E@trainy

# calculate SSE for opt_model
yhat = np.zeros(len(holdout_fold))
for i in range(len(holdout_fold)):
    yhat[i] = line2(coeffs, testx[i])
final_sse = 0
for j in range(len(holdout_fold)):
    final_sse += (yhat[j]-testy[j])**2
```

7.5 Derivation of minimized LASSO

$$\frac{\partial}{\partial x_i}|x_i| = \begin{cases} -\lambda & \text{if } x_i < 0 \\ [-\lambda, \lambda] & \text{if } x_i = 0 \\ \lambda & \text{if } x_i > 0 \end{cases}$$

For the case of $x_i = 0$ the subgradient technique states that there are infinitely many derivatives; therefore, the solution contains all slopes between $-\lambda$ and $+\lambda$. Now, to combine the two solutions.

$$\frac{\partial}{\partial \mathbf{x_i}}(\text{LASSO}) = 2(\mathbf{A}^T\mathbf{A})_i\mathbf{x_i} - 2(\mathbf{A}^T\mathbf{b})_i + \begin{cases} -\lambda & \text{if } x_i < 0 \\ [-\lambda, \lambda] & \text{if } x_i = 0 \\ \lambda & \text{if } x_i > 0 \end{cases}$$

$$= \begin{cases} 2(\mathbf{A}^T\mathbf{A})_i\mathbf{x_i} - 2(\mathbf{A}^T\mathbf{b})_i - \lambda & \text{if } x_i < 0 \\ [-2(\mathbf{A}^T\mathbf{b})_i - \lambda, -2(\mathbf{A}^T\mathbf{b})_i + \lambda] & \text{if } x_i = 0 \\ 2(\mathbf{A}^T\mathbf{A})_i\mathbf{x_i} - 2(\mathbf{A}^T\mathbf{b})_i + \lambda & \text{if } x_i > 0 \end{cases}$$

Now, setting the subgradient equal to zero and solving for x_i

$$x_i^* = \begin{cases} ((\mathbf{A}^T\mathbf{b})_i + \frac{\lambda}{2}\mathbf{I})/(\mathbf{A}^T\mathbf{A})_i & \text{if } (\mathbf{A}^T\mathbf{b})_i < \frac{-\lambda}{2} \\ 0 & \text{if } (\mathbf{A}^T\mathbf{b})_i \text{ in } [\frac{-\lambda}{2}, \frac{\lambda}{2}] \\ ((\mathbf{A}^T\mathbf{b})_i - \frac{\lambda}{2}\mathbf{I})/(\mathbf{A}^T\mathbf{A})_i & \text{if } (\mathbf{A}^T\mathbf{b})_i > \frac{\lambda}{2} \end{cases}$$