Project 1
Franke Function and Boston Housing Data

Casuga, Carlisle Aurabelle; Conde Villatoro, Daniel Eduardo; González-Miret Zaragoza, Luis; Langelund, Samuel; Muñoz Méndez, Jesús Eduardo

Abstract

This project is a first approach to machine learning techniques in order to explore the limits and applications of ordinary linear regression (OLS) and Ridge regression. To achieve this goal, two resampling methods, bootstrapping and cross-validation, were implemented to visualize the bias-variance trade-off and to compute the confidence intervals for the model parameters. The two datasets we explored, one corresponding to the Franke function and the other corresponding to the Boston housing data from sklearn where the median prices were selected as the target variable. For both cases, a optimization of the polynomial degree and hyperparameters of the fitting models was done.

I. Problem and methods used

A. Outline

We explored two scenarios where we can apply machine learning techniques that show explicitly the effect of model complexity and shrinkage parameters on the bias-variance trade-off. For the first part, we apply these techniques on the grid $[0,1] \times [0,1]$ and target values given by the Franke function. In the second, we use the Boston housing dataset from sklearn. The main difference between the two is that we have access and knowledge of the true function for the first scenario, whereas the true function of Boston housing prices is completely unknown.

The Franke function is defined as [1]:

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

For this part of the project, the introduction of techniques is progressive, meaning that cross-validation, for example, isn't used until it is introduced at the end. The results we obtained are

- the mean square error and variance score for different degree polynomials with fits with no scaling, no resampling and no splitting of the data,
- a comparison of the training and test set errors,
- a discussion on the bias-variance trade-off both by demonstrating the relationship between the errors, the bias, and variance; and explicitly looking at the trade-offs in the model by applying bootstrapping,
- an exploration of number of folds in cross-section and comparison to bootstrapping as a resampling technique,
- the determination of confidence intervals of fit parameters of OLS with the covariance matrix, bootstrapping and cross-validation, and
- application of Ridge regression

For the second part, we apply the resampling techniques and compare ordinary linear regression to Ridge regression on the famous data-set of Boston Housing Prices. For this, we consider the variable 'MEDV' as the target, and choose four of the other variables as features of the model on our discretion according to the correlation matrix. This time, we apply wholly the resampling techniques to obtain measures of the test error with confidence intervals. The code used for this project can be found in this git repository.

II. Franke function analysis

A. Naive OLS fits (no splitting, no scaling, no resampling)

The relevance of the Franke function is that it is commonly used to test machine learning models. So, to begin our analysis, we calculated target data from the Franke function with some noise in order to create a dataset. As a first approach, we computed polynomial fits up to ninth degree. We used no splitting, no scaling and no resampling;

simply nine polynomial fits. For this, we used the PolynomialFeatures() function from the Scikit-Learn library. Figure 1 shows the data from the Franke function with some transparency, and the different fits as the solid surfaces. We see naturally the surfaces getting sharper with higher degree polynomials, and also the MSE (mean squared error) and R^2 score going monotonically down and up, respectively.

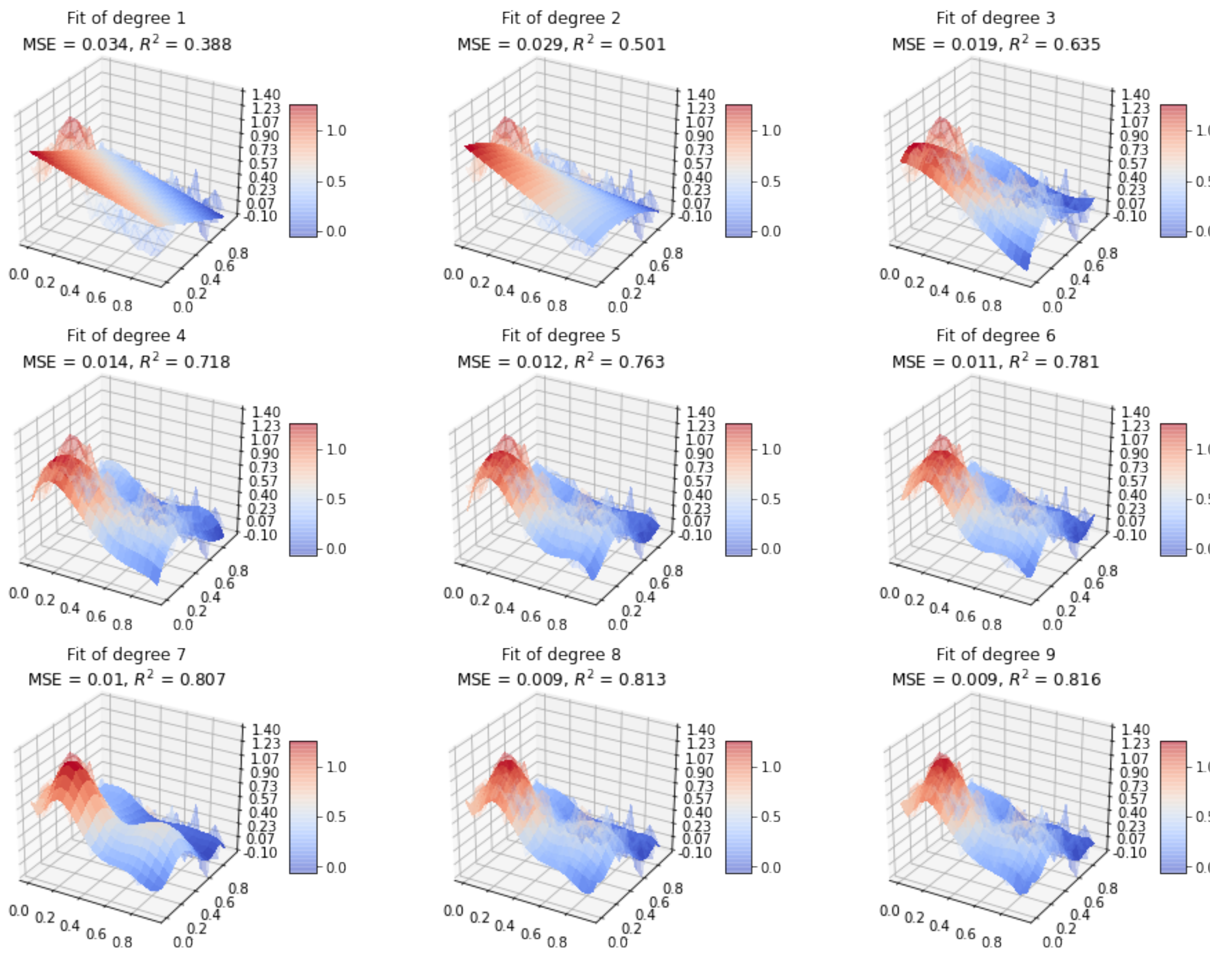


Figure 1: Franke function fits from degree = 1 up to degree = 9. The solid color surface is the fit, and the more transparent and more wavy surface is the Franke function values with some noise taken as training data

The graphs in Figure 1 show how the fits with no more machine learning application would look like, and we seem to see no downside to continuing rising the value for the polynomial degree. However, the next section shows that, after about a polynomial degree of six, the fits do not really improve beyond getting better at describing the training data, as seen explicitly from this Figure.

B. *Training and test set errors*

After computing the previous simple fits, we introduce scaling and splitting of the data. We used the train-test-split of sklearn with a test set size of 20% and used the StandardScaler() from sklearn to do the scaling. The standard scaling sets the mean value to zero and then scales the values so that the standard deviation is one; this procedure ensures that all features have the same relevance on the fit regardless of step size of whichever machine learning algorithm we apply next. In the context of polynomial fits with the design matrix, this means that higher order features will seem to be more spread out, whereas the linear terms will remain evenly spaced as set from the beginning.

Splitting the data allows us to test a crucial aspect of our model: its predictive power on fresh unseen data. We train on 80% of the data and use the resulting model on the unseen 20% of the data to measure how well it performs. A crucial aspect of the process of fitting that this procedure allows us to evaluate is overfit. This may

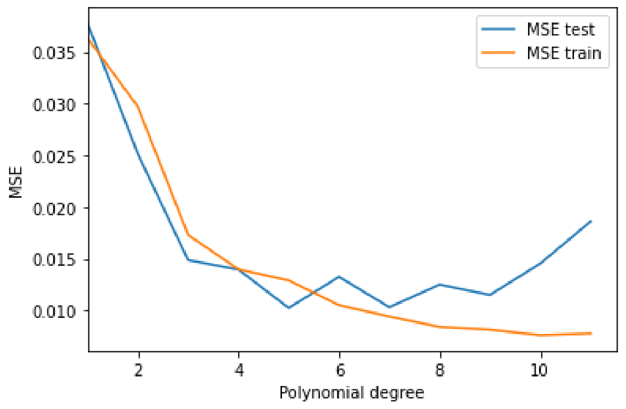


Figure 2: Training and test sets errors for polynomial degrees from 1 to 11

happen when some model with too many parameters adjusts itself so much that it reproduces the noise of the data and not the general trend. This means that new data with a different set of noise values will be badly reproduced by the overfitting model. After splitting and scaling the data of the previous section, we obtained Figure 2, which shows that the error on the training set continues to go down for higher degrees, but the test error eventually goes up as the effect of overfitting becomes more prevalent.

To visualize the relationship between noise and overfitting, we repeat the procedure for six different scales of noise. This is shown in Figure 12 of the Appendix, and the main takeaway is that as the noise term rises, the test error curve is more sensitive to the effects of overfitting at lower and lower polynomial degree values, whereas the training set error curve generally goes down in all cases.

C. *Bias-variance trade-off with bootstrapping*

The bias and variance trade-off is at the heart of many problems in machine learning. It follows from our appreciation of overfitting in the previous section: a method that fits itself too well to the noise of the data has high variance. A more loose method might not have this problem, but it describes the data more poorly, which we identify as high bias. If we assume the data to take the form $\boldsymbol{y} = \boldsymbol{f} + \boldsymbol{\epsilon}$, with $\boldsymbol{f} = f(\mathrm{x})$ an exact function depending on the predictors $\boldsymbol{x}$ and $\boldsymbol{\epsilon}$ a random noise so that $\mathbb{E}[\boldsymbol{f}] = \boldsymbol{f}$, $\mathbb{E}[\boldsymbol{\epsilon}] = 0$ and $Var[\boldsymbol{\epsilon}] = \mathbb{E}[\boldsymbol{\epsilon}^2] = \sigma^2$, then the error of a model with prediction $\tilde{\boldsymbol{y}}$ can be mathematically separated as

$$\begin{aligned}
\mathbb{E}\left[(\boldsymbol{y}-\tilde{\boldsymbol{y}})^2\right] &= \mathbb{E}\left[(\boldsymbol{f}+\boldsymbol{\epsilon}-\mathbb{E}[\tilde{\boldsymbol{y}}]+\mathbb{E}[\tilde{\boldsymbol{y}}]-\tilde{\boldsymbol{y}})^2\right] \\
&= \mathbb{E}[\big(\boldsymbol{f}-\mathbb{E}[\tilde{\boldsymbol{y}}])^2] + \mathbb{E}[(\boldsymbol{\epsilon}+\mathbb{E}[\tilde{\boldsymbol{y}}]-\tilde{\boldsymbol{y}})^2]] + \mathbb{E}[2(\boldsymbol{f}-\mathbb{E}[\tilde{\boldsymbol{y}}])(\boldsymbol{\epsilon}+\mathbb{E}[\tilde{\boldsymbol{y}}]-\tilde{\boldsymbol{y}})]] \\
&= \mathbb{E}[(\boldsymbol{f}-\mathbb{E}[(\tilde{\boldsymbol{y}}])^2] + \mathbb{E}[\boldsymbol{\epsilon}^2] + \mathbb{E}[(\mathbb{E}[\tilde{\boldsymbol{y}}]-\tilde{\boldsymbol{y}})^2] + 2\cancelto{0}{\mathbb{E}[\boldsymbol{\epsilon}]}\mathbb{E}[(\mathbb{E}[\tilde{\boldsymbol{y}}]-\boldsymbol{y})^2] \\
&+ 2(\boldsymbol{f}-\mathbb{E}[\tilde{\boldsymbol{y}}])(\cancelto{0}{\mathbb{E}[\boldsymbol{\epsilon}]+\mathbb{E}[\tilde{\boldsymbol{y}}]-\mathbb{E}[\tilde{\boldsymbol{y}}]}) \\
&= \frac{1}{n}\sum_i (f_i-\mathbb{E}[\tilde{\boldsymbol{y}}])^2 + \frac{1}{n}\sum_i (\tilde{y}_i-\mathbb{E}[\tilde{\boldsymbol{y}}])^2 + \sigma^2 = \text{Bias} + \text{Var}(\tilde{\boldsymbol{y}}) + \sigma^2.
\end{aligned} \tag{2}$$

In our analysis, we'll approximate the values f_i by the values from the dataset y_i, and use bootstrapping as a resampling technique to compute the variation of the models. We again use a 20% split of the data and scale with the StrandardScaler() function. Additionally, the parameter values of each bootstrap procedure were stored for their use in section E.

In Figure 3 it is shown the values of bias, variance and total error for 10 polynomial fits with ascending degree value. We see in this Figure the expected trends: variance goes up monotonically as the complexity of the model goes up, and bias goes down as the complexity allows for a better description of the data. The total error has local minimums at 4 and 9, but the phrase "local minimum" should be taken with a grain of salt since reruns of this code don't give the same conclusion, but just general minima in the [3,9] interval.

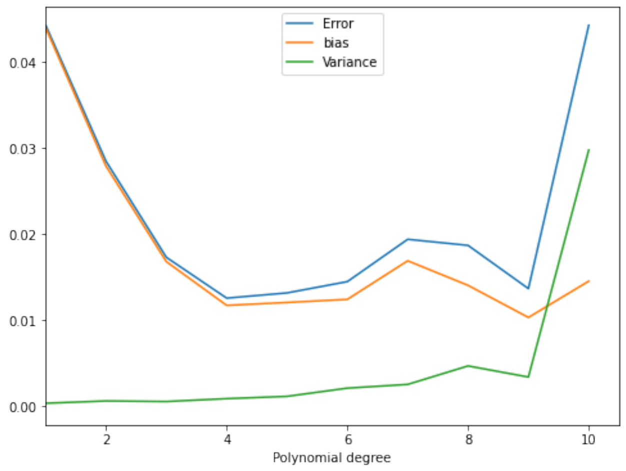


Figure 3: Total error in blue, bias in orange and variance in green for different polynomial degrees. The variance goes up monotonically while the bias goes down initially as the complexity of the model goes up.

D. *K-Fold cross-validation*

One obvious problem of splitting the data in a test set and a training set is that we're not using all of the data to construct the model. One strategy to alleviate this is to split the data in many different ways, construct the models for each, and average the results. There are many ways to do this, as many permutations of splitting the data there are, but the most commonly used is the K-Fold cross validation. With this method, the data is split in k subsets. Each run will use one of the subsets as the test set and the rest as the training set, consecutively, until each subgroup has been used as the test set once. The advantages of this method is that it is both more economical than using more permutations, and is also representative because all of the data is used for testing once.

Cross validation has to be performed carefully, since the premise of the training and test split is that the test data is supposed to be fresh unseen data. The k-fold split, therefore, has to be performed before any other machine learning technique that morphs the data in any way. For example, scaling of the data has to be performed for each individual subgroup, otherwise, information about the values of the test sets is 'seen' by the scaling before it is split away. For this reason, and also because the features in this analysis are equivalently scaled from the start, we chose not to scale the data in this section. We summarize the steps in the following:

1) The sample is divided into k subsets of roughly equal size.
2) The subsets are randomly ordered,
3) One of the sets is picked as the test set and the $k-1$ others as the training set,
4) The model is trained and scored with the training and test sets of each fold respectively,
5) Results are averaged for all the runs

The results of the training are the parameter values, the variance score (R^2) and the mean square error of the test set. We are estimating the error of each of these with the standard deviation: this is a safeguard against reaching false conclusions about predictability of the models because cross validation is prone to both overestimating and underestimating the test error. This high variability of the MSE in cross validation is one of its main disadvantages, but at least we're able to construct confidence intervals with the results of this technique.

We first explore the effect of varying the number of folds in the cross validation procedure. We choose a polynomial degree of 5, and obtain the results shown in Figure 4. The vertical axis is on a logarithmic scale, and the error bars are displayed with the formula:

$$\delta z = \frac{1}{\log_{10} e}\frac{\delta y}{y}, \tag{3}$$

where y is the original value, δy its uncertainty and δz is the error bar size in the logarithmic scale. The first value is 0.3186 ± 0.4879 and the last value is 0.0147 ± 0.0068. The main takeaway from this graph is that rising the number of folds seems to simply lower the error value, improving the model. However, the improvement is only stark at the beginning, so many often simply use 10 folds, since further increase in the number of subsets doesn't

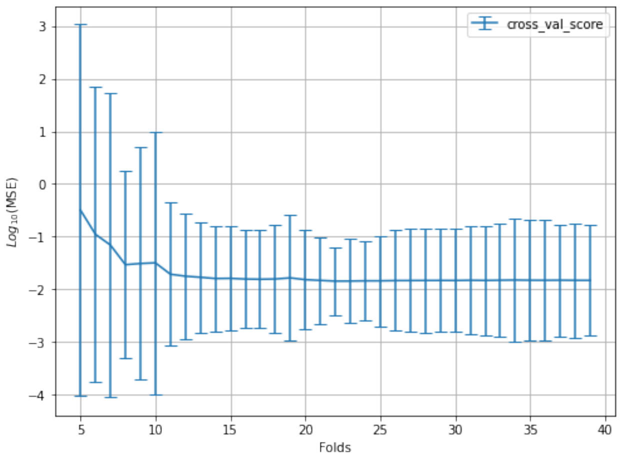


Figure 4: Test error dependence on number of k-folds.

seem to improve the results by much. Furthermore, we see here in effect the fact that the MSE estimates have high variance, shown by the large error bars. This puts in evidence that, after 10 folds, we can't really say there is an improvement for the MSE error but merely that, after 10 folds, we have minimized the variance of this estimation.

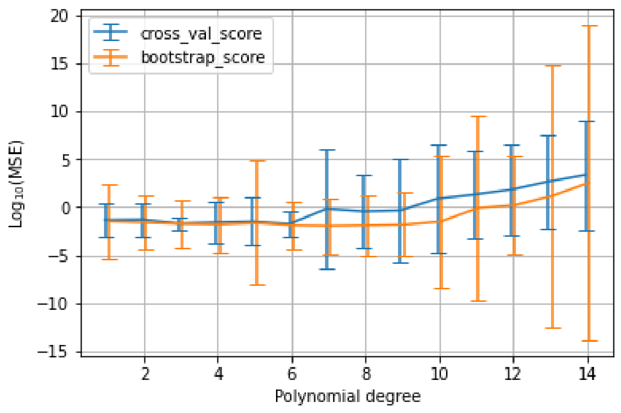


Figure 5: Test error dependence on number of k-folds. The blue plot shows the result with cross validation, with its first value being 0.32 ± 0.49 and the last value being 0.0147 ± 0.0068. The orange plot shows the results with a bootstrap resampling, slightly offset to the right of the blue one for clarity of the error bars.

Following this, we compute the MSE error (always for the test sets) given by the full cross validation procedure for different values of polynomial degree, and this is shown in Figure 5 together with the same value computed from doing bootstrapping only. This is also in logarithmic scale for clarity. In this graph, we see that more complex models give both high test error and also high variance of this test error according to the k-fold estimation. There seems to be a sweet-spot in between [3,6], consistent with our discussion in section B. The cross validation scores seem to be in agreement with the bootstrap scores for MSE. If anything, the MSE and its variation seems to be very similar in the range of interest in [3,6] for polynomial degree. Intuition would then say that in the future, it would be best to just do both at the same time, or relinquish one in favor of computing time with no much loss in quality of analysis.

E. Confidence intervals of parameter values

A measure of uncertainty for the parameter values is the square values of the diagonal entries of the covariance matrix. These values get higher for higher degree polynomials, since the number of features increases. These are shown in the blue plot in Figure 6.

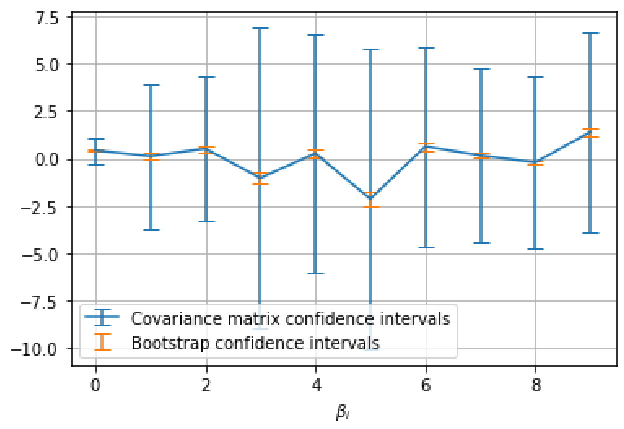


Figure 6: The parameter values of a 3 degree polynomial fit, with the error bars representing the confidence intervals for the 9 different parameters.

We see that with a 3 degree polynomial (which in previous sections we show that is the model close to an optimal degree value for test error), the confidence intervals of all parameters include the value zero, essentially showing no significant difference between this model with 45 parameters and a constant model where all of these parameters are zero. As we have shown in the test error measure, the models are indeed different. Since this isn't shown well in this measure, we also consider the confidence interval given by resampling techniques.

With bootstrapping, we see the comparatively much smaller error bars of the beta parameters, which are actually more precisely determined than the covariance matrix suggested. In this case, we think the confidence intervals given by the bootstrap resampling are more accurate since, when redoing the analysis with a different seed of random values, it didn't give parameters varying as in the blue confidence interval, but as in the orange one.

F. The λ hyperparameter in Ridge regression with resampling

In this section we introduce Ridge regression, replacing OLS. We can think of Ridge regression as the modification of the cost function that penalizes high (absolute) values of parameters in a measure proportional to a new hyperparameter λ. This counteracts overfitting in a way, because most complex models overcompensate with exorbitant parameters to fit around noise. The desired result of Ridge regression is then to reduce the test error due to overfitting.

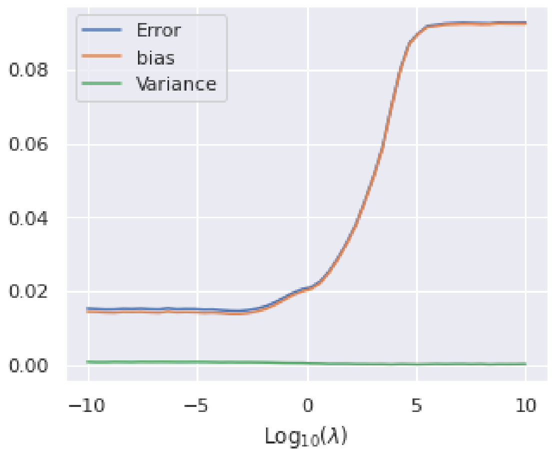
In the following analysis, we explore the dependence of the test error on the λ hyperparameter. We find, just as with polynomial degree, that certain values of λ are optimal for our data-set. We first see it when we recreate the bias-variance trade-off graph from Figure 3 but now with respect to the new hyperparameter, shown in the left panel of Figure 7. This time, we might interpret the horizontal axis as the lowering of model complexity, since shrinkage methods such as this suppress the effect of features. In this case, we do see the expected behaviour that variance goes down for less complex models, but bias goes up.

In the right panel of Figure 7, we show the test error with respect to the λ hyperparameter again, but for different number of k-folds. In general, we simply see again that a higher number of k-folds reduces error overall, and that the apparent best value for λ for k>7 is around 10^{-3}.

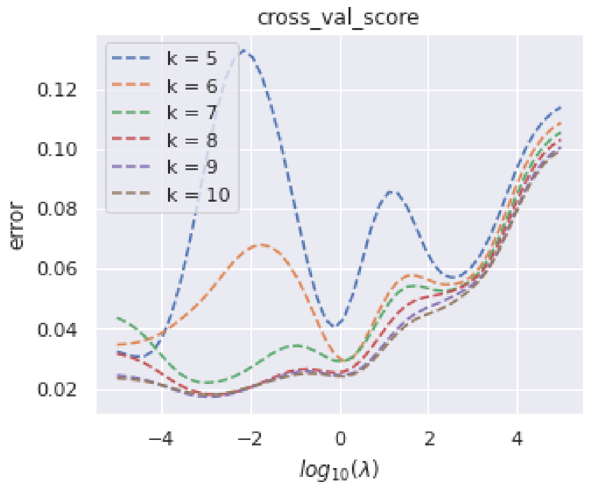
III. Boston housing data analysis

For this part of the project, we apply previously detailed machine learning techniques to a real data set. We compare two regression techniques, ordinary least squares and Ridge regression to predict the target variable. Features were selected according to the correlation matrix of the data and cherry picking the two features most correlated with our target. Same methods were applied to a four-feature design matrix.

The Boston housing data can be easily obtained through Scikit-Learn's library. It contains information collected by the U.S. Census Service concerning housing in the Boston Mass area with 506 samples with 13 features available. The target variable is MEDV and it contains the median value prices of owner - occupied homes in $1000's. We list the description of the predictors below.
The features/predictors are



(a)



(b)

Figure 7: The bias and variance trade off with respect to the λ hyperparameter visualization using bootstrapping and b) the cross-validation MSE test error also with respect to λ for different numbers of k-folds

1) CRIM: Per capita crime rate by town
2) ZN: Proportion of residential land zoned for lots over 25000 square feet
3) INDUS: Proportion of non-retail business acres per town
4) CHAS: Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)
5) NOX: Nitric oxide concentration (parts per 10 million)
6) RM: Average number of rooms per dwelling
7) AGE: Proportion of owner-occupied units built prior to 1940
8) DIS: Weighted distances to five Boston employment centers
9) RAD: Index of accessibility to radial highways
10) TAX: Full-value property tax rate per $10000
11) PTRATIO: pupil-teacher ratio per town
12) B: $1000(Bk - 0.63)^2$, where Bk is the proportion of [people of African American descent] by town
13) LSTAT: Percentage of lower status of the population

A. *Historical and Real-world Implications*

As advancements are made in technology and the science of data collection the way we monitor aspects of daily life have and will begin to change. Analyses like this one offer valuable insight to the values and pitfalls such methods have and how we can go about accounting for or eliminating them. The topic of housing as a factor of other variables is especially important as census data collected of inhabitants is often used to determine how city, state, and federal resources are distributed and allocated in the United States.

For example, the most recent census in the USA took place in 2020 and required over half a million people to go door to door to count its inhabitants. This incredible use of resources has led many lawmakers and politicians to question if a different method of collecting data may be more effective, primarily sampling. The issue begins when examining the constitution of the United States where some scholars argue that sampling may be unconstitutional. This stems from the fact that article 1 section 2 of the Constitution which decrees the terms for and by which a census will be carried out uses the wording *shall be determined by adding to the whole Number of Persons* to describe how the process will be carried out. The use of the word adding has led some to argue that this legally demands the census be conducted by counting each person. The driving argument against it, which in recent years has gained more traction, is that sampling is more accurate than a straight count and represents disenfranchised communities such as the homeless better. Additionally it is noted that the article is arcane and obsolete. Its true wording below echoes a dark time in US history where not all people were treated with the same unalienable rights and therefore it is important that it be viewed in the context of the society in which we now live.

...shall be determined by adding to the whole Number of free Persons, including those bound to Service for a Term of Years, and three fifths of all other Persons.[3]

Analyses like this one help show the benefits of switching to a more modern and advanced system which more accurately represents the facts as they are.

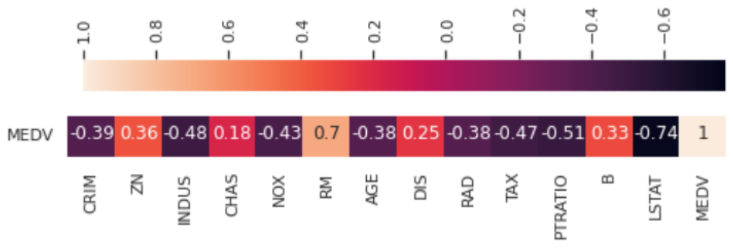


Figure 8: Visualization of dataset: Correlation values of Boston Housing dataset between the variable MEDV and all the rest in the dataset

B. Data and visualization and feature selection

The histogram of the MEDV data shows the median value to be roughly around \$20,000. To help us determine which features we could use, a correlation matrix (values shown in Figure 8), was calculated. We see that the predictors, LSTAT (low income and education) and RM (number of rooms) are the most correlated with MEDV. Additional good indicators are INDUS, TAX, NOX, and PTRATIO, all with a correlation above 0.5 with MEDV. Also, further inspection reveals that RAD (accessibility to radial highways) and TAX (tax rate) are highly correlated and one of these features can be chosen in place of the other. We ultimately included LSTAT, RM and also NOX and DIS. In particular, NOX was included because it might show the shrinkage method of Ridge regression at work by suppressing the parameters associated with this feature. Figure 13 in the Appendix shows the individual features plotted versus MEDV. The scatter plots further prove the high correlation of LSTAT and RM with the target.

C. A first fit with simple linear regression

Starting with a design matrix featuring only LSTAT and RM, a simple linear regression resulted in a MSE and R^2 score of 31.81 and 0.63, respectively, when fitted against the training data. Meanwhile, for the test data the scores obtained were 26.42 and 0.70. The purpose of this fit is to show that a linear fit will give predictions that don't capture the patterns in the data. This is shown more explicitly in Figure 9. In here, the data points seem to have an underlying pattern not captured by the model.

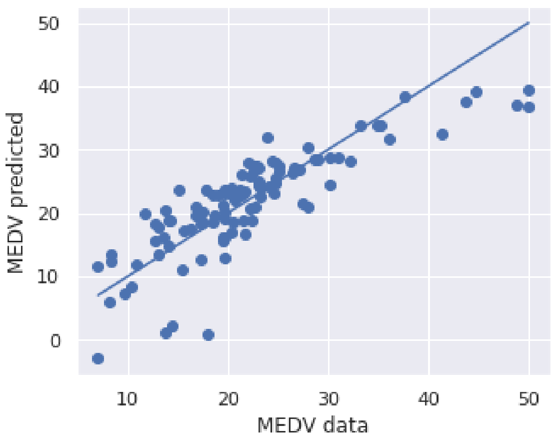


Figure 9: Predicted values for MEDV using a simple linear regression against the true values. The diagonal line $MEDV_{predicted} = MEDV_{data}$ is the idealized result.

D. *Bias-variance trade-off with OLS*

To further improve these scores, we increase the complexity of the models by calculating design matrices of higher polynomial degree. To decide which degree to work with, we look at the bias variance trade off graph in Figure 10 to determine the optimal polynomial function to fit. This was done for two resampling techniques, bootstrap and cross validation. These graphs feature the expected overfitting at higher complexity. With bootstrap resampling and a second order design matrix, the MSE is further reduced to a 23 ± 21.

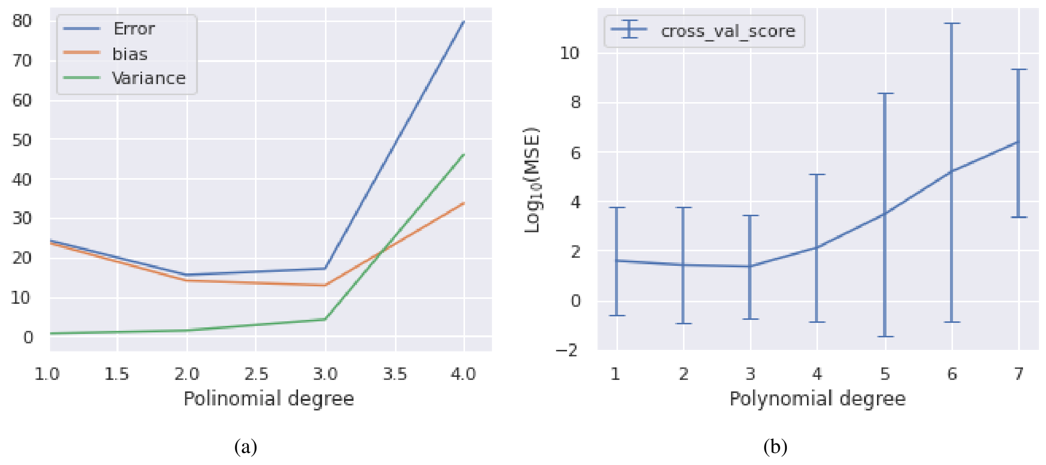


Figure 10: The bias and variance trade off visualization using bootstrapping and b) the cross-validation MSE test error on a logarithmic scale

E. *Search for the optimal λ hyperparameter with Ridge regression*

In the interest of a better model, we introduce another adjustable parameter: the hyperparameter λ through Ridge regression. We also include two more features into the model, "DIS" and "NOX". The purpose of introducing these is to put the shrinkage to the test, since "DIS" is a feature not noted to be highly correlated with "MEDV". Here we use the RidgeRegression() model from Scikit-Learn instead of LinearRegression(). Together with the polynomial degree, we have two adjustable parameters to optimize. For this, we explore this parameter space from degrees = [1,6] and $\lambda = [10^{-10}, 10^5]$. A visualization of this search is added to the Appendix in Figure 14.

Most models, except for the ones of degree 1 and 3, find a local minimum for lambda. We found that the best score for the model is given with a fit of degree 2, right on a local minimum that is deeper than any of the ones for degree 3 or 1. In the case of the ones of higher degree than 3, we can suspect that the shrinkage goes away with unwanted overfitting, and that results in a dramatic increase in error for small lambda. The minimum MSE error found among all models was 22 ± 20 with a degree 2 and lambda 2.56. This value is not much better than the one we had obtained with OLS. Additionally, 2.56 is a relatively big number for lambda, and we could explain its high value maybe because of the inclusion of the "DIS" feature. We had said before that this feature might not be the best candidate, so a high shrinkage might have taken place. However, we didn't explore this further.

F. *Predicted MEDV values from the final model*

Finally, in the Figure 11 we show the predictions by our final Ridge model against the real values. This is the result of using a slightly more complex model (degree 2 and lambda 2.56) than a simple linear fit with the parameters calculated from the previous section. After comparing this to the plot in Figure 9, we see a much better fit since the data scatters uniformly around the diagonal line.

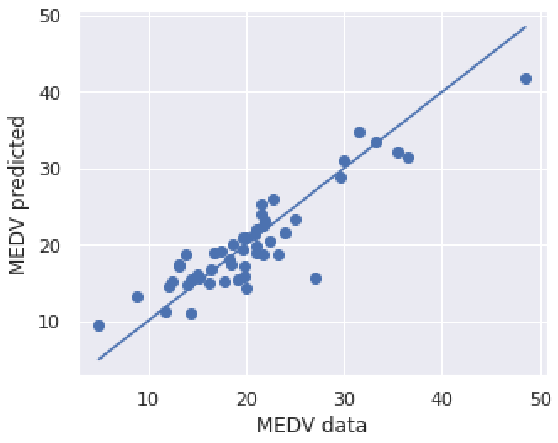


Figure 11: Predicted values for MEDV using the final Ridge model against the true values. The diagonal line $MEDV_{predicted} = MEDV_{data}$ is the idealized result.

IV. Summary and Conclusions

We performed Ridge regression and OLS regression on data obtained from the Franke function and, on the other hand, on the Boston housing data with MEDV as the target value. We obtained models with parameter values that optimize the test error.

For both of our analysis, and to explore the effect of model parameters on the test error, we increased the complexity of the models by raising the polynomial degree of the design matrix in our fits, or we changed the shrinkage parameter in Ridge regression. The first of these gave us a clear idea of what overfitting does to our models, and how to find a compromise in the bias-variance trade-off. This compromise shows its best results in the comparison between Figure 9 and 11: the final predicted values are the result of increasing model complexity over the first predicted values, while not sacrificing much variance in the bias-variance trade-off since we'd chosen model parameters that minimized test error. Changing the shrinkage parameter shed light on the advantages of Ridge regression over OLS, however, in the Boston housing dataset we didn't find a significant improvement in the test error.

A difficulty we found while finding a compromise in the bias-variance trade-off, however, was the assertion itself of what the best test error is. The reason is that, after calculating the confidence intervals for test error, it was usually impossible to differentiate between values with any significant confidence. In accordance to what is mentioned in section 7.10.3 "Does Cross-Validation Really Work?" in [2], we found that the variance of the test error when computed in cross validation is high, and thus it was hard to assert which model parameters gave the best test error. Take Figure 10, for example: in the right panel wee see the test error for different model complexities with their corresponding standard deviations as error-bars, and even while spanning 1 through 7 polynomial degrees, all values are within each other confidence intervals.

The code used for this analysis can be found in this git repository.

References

[1] *Machine Learning and Data Analysis for Nuclear Physics, European ERASMUS+ Master of Science program. Hjorth - Jensen, M., Wold, K., Boumbouras Sønderland, P.D., 2021, URL: github.com/CompPhysics/MLErasmus*
[2] *"Elements of Statistical Learning: Data Mining, inference and Prediction". Friedmann, J., Hastie, T., Tibshirani, R. 2008*
[3] *"U.S. Constitution article 1, § 2". 1776*

Appendix A
Additional Figures

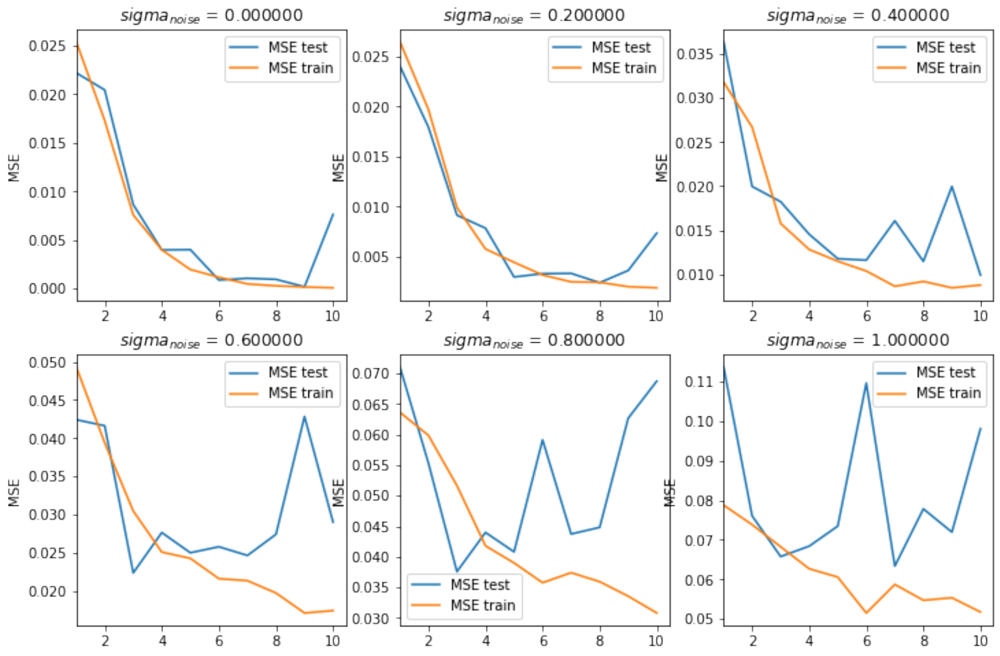


Figure 12: Training and test sets errors for polynomial degrees from 1 to 10 for 6 different values of noise for the original data

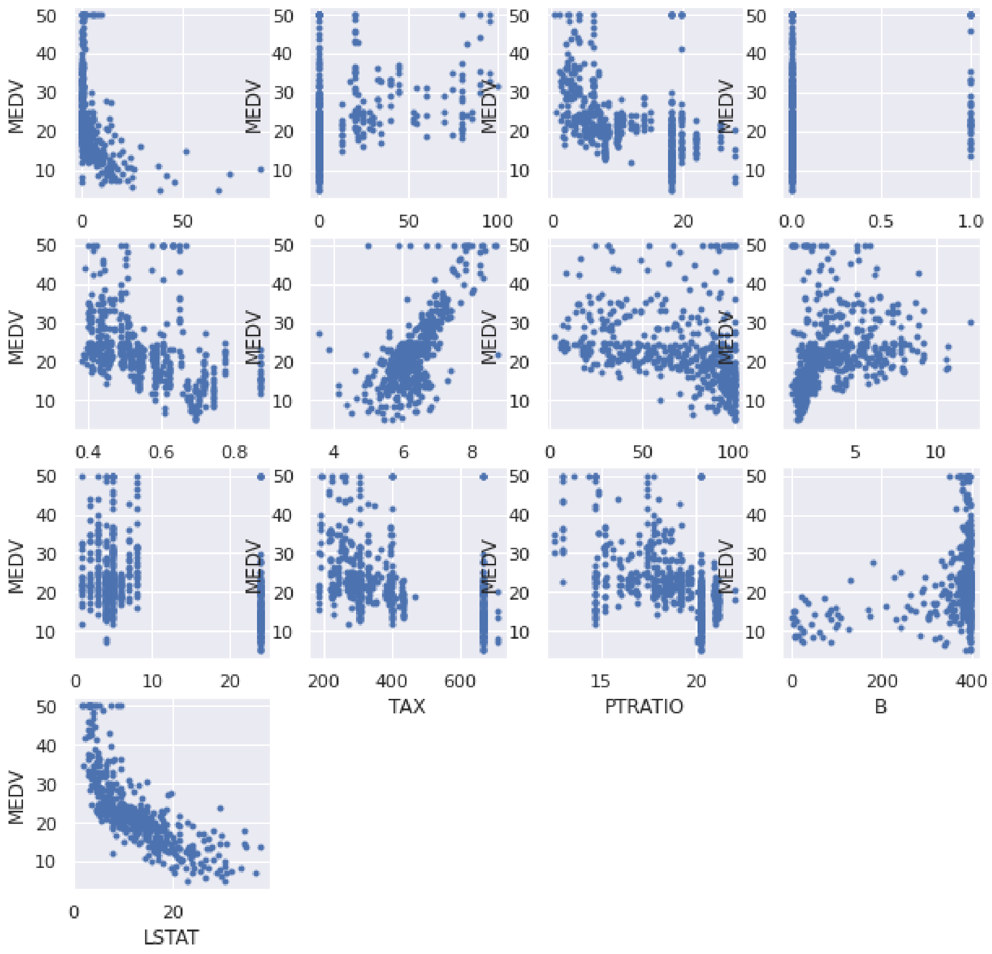


Figure 13: Dispersion plots between the selected feature "MEDV" among all features in the dataset. The selected features were "RM", "LSTAT", "NOX" and "DIS".

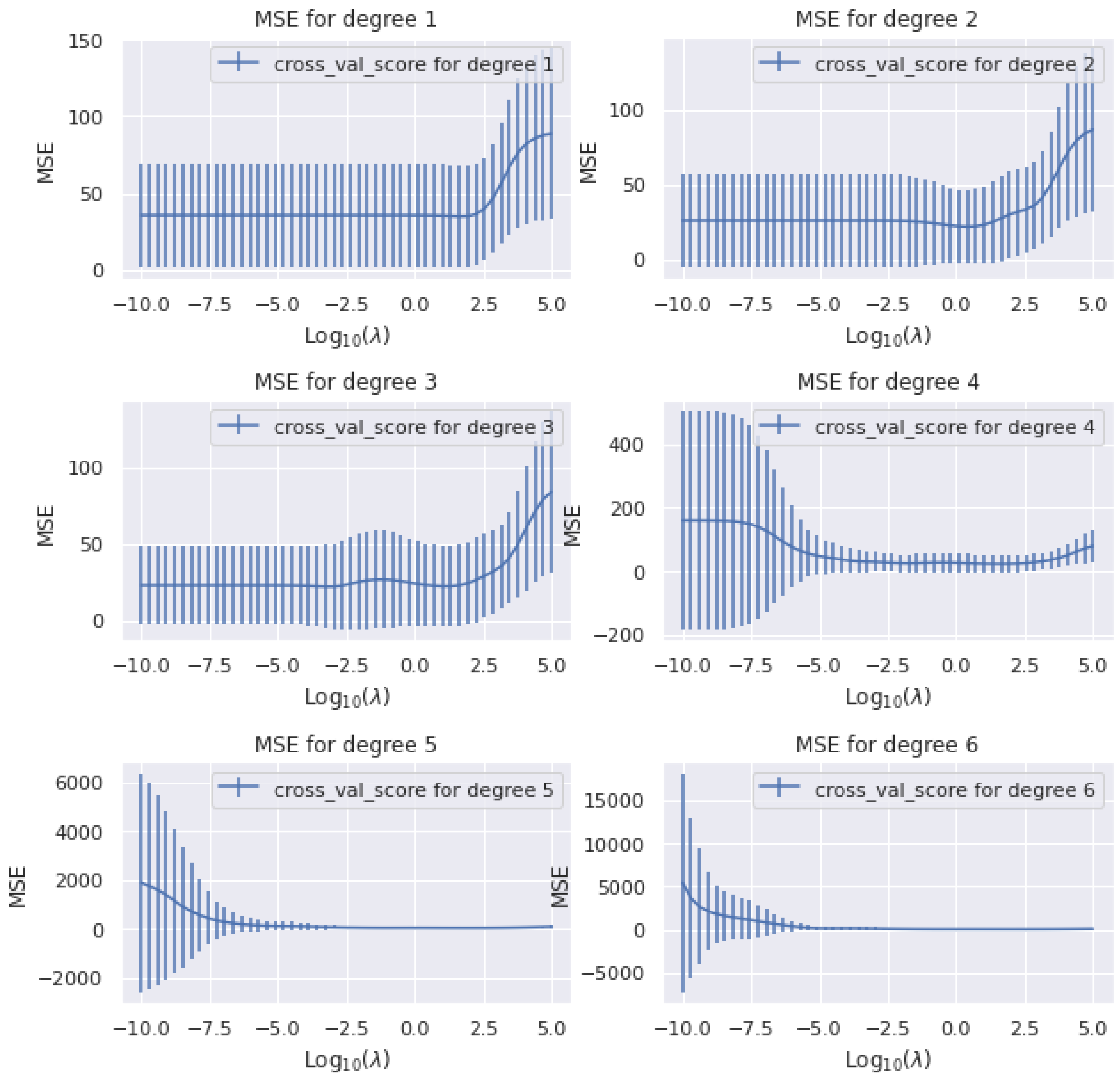


Figure 14: MSE test error with respect to the λ hyperparameter for different values of polynomial degree. In these graphs, the horizontal axis shows the 10 logarithm of the lambda values.