# ML Project 1

Amanda Nathali NERIO AGUIRRE, Mojahed Sameer Moh'd ABUSHAWISH,
Nikhil MOZUMDAR, Snehankit PATTNAIK

University of Caen-Normandie, France

Erasmus Mundus Joint Master Degree in Nuclear Physics
Machine learning
Professor: Morten Hjorth-Jensenn

February 27, 2021

## Abstract

This report serves to present basic machine learning concepts such as ordinary least square regression, ridge regression, over-fitting and re-sampling techniques in supervised learning. The Franke function was studied using least squares and ridge regression. Re-sampling methods using bootstrap and cross-validation has also been covered. These topics are first presented for a two dimensional Franke function where the dataset has been produced. These methods have been applied to a real dataset namely the Boston housing data and the analysis of which has been discussed. A mean squared error and R2 score of 13.32 and 0.818, respectively is reported for ordinary least squares while for ridge regression the values are 12.7 and 0.827, respectively.

## 1 Introduction

Machine Learning (ML), statistics, and data science are fields that explain how to learn from and forecast data. ML and data science represent important tools in many modern fields of technology. ML is a sub-field of artificial intelligence with the objective to develop algorithms capable of making sense of data. At the heart of the techniques in ML, the essence is to make a suitable prediction and perform optimizations.

For making a prediction we select some measurable quantity **x** of the data structure we are examining and a model p(**x**|$\boldsymbol{\theta}$) that defines the probability of observing **x**. Estimation problems are related to the reliability of the model, while prediction problems such as ordinary least squares and ridge regression are related to the ability of the model to predict new insights. Currently a big data growth globally is demanding an increased processing power and memory. Computations that were inconceivable a few decades ago due to techniques such as deep learning, can now be readily performed on ordinary laptops and computers. ML methods are based on observational results and intuition unlike classical analysis.

ML researchers follow a common method to achieve models that are effective for predictive problems. The first step in the study is to randomly split the dataset D into two mutually exclusive groups, $D_{train}$ and $D_{test}$ to offer an unbiased estimation of the model's predictive accuracy. The model, then, tries to minimize the cost function $C(\boldsymbol{y}, f(\boldsymbol{X}; \boldsymbol{\theta}))$ [1] by including only the training data set. Finally, the

accuracy of the model is assessed by calculating the cost function using the test set $C$ ($\boldsymbol{y}_{test}$, $f(\boldsymbol{X}; \boldsymbol{\theta})$). The value of the cost functions of the best fit model on the training dataset is called the in-sample error and for the test data set it is known as out-of-sample error. ML problems include complex processes where we might not know the mathematical model that describes the data set. Therefore it is better to have multiple comparative models and the model that minimizes the out-of-sample error is selected as the better model.

## 1.1 Problem

The primary objective of this project is to present a modality for different regression approaches. First we try to fit polynomials to a two-dimensional function called the function of Franke (Eq.1). We then proceed to model different regression analysis with real dataset, the Boston Housing dataset. We would like to reduce the mean squared error and obtain the optimal model in order to predict the price of an unknown house given certain parameters, known as features. As this dataset has several features and not too many data samples, it is therefore necessary to pick the optimal number of parameters for our best model.

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

## 1.2 Linear regression

The concepts presented here can be found in more details in [1]. Let's suppose for a $n$ sample dataset $D = (y_i, \boldsymbol{x}^{(i)})_{i=1}^{n}$, where $\boldsymbol{x}^{(i)}$ is the vector of ith observation and $y_i$ is its respective scalar output. We consider $p$ as the number of features in any sample. Let $f$ be the actual function or model from which these samples $y_i = f(\boldsymbol{x}^{(i)}; \boldsymbol{w}_{true}) + \epsilon_i$ were produced , where $\boldsymbol{w}_{true} \in \mathrm{R}^p$ is a parameter vector, and $\epsilon_i$ is some noise with zero mean and finite variance. Using all samples we then create an $n \times p$ matrix, $\mathbf{X} \in \mathbf{R}^{n \times p}$ called the design matrix, with observations in the rows and $p$ features in the columns. As the function $f$ isn't directly defined to us, we assume its functional form. In linear regression, we assume $y_i = f(\boldsymbol{x}^{(i)}; \boldsymbol{w}_{true}) + \epsilon_i = \boldsymbol{w}_{true}^{\mathbf{T}} \boldsymbol{x}^{(i)} + \epsilon_i$ for some unknown but fixed $\boldsymbol{w}_{true} \in \mathrm{R}^p$. We hunt for a function $g$ that matches the data $D$ with $\hat{\boldsymbol{w}}$ parameters that can better fit the function $f$. When obtained we find a such that $g(\boldsymbol{x};\hat{\boldsymbol{w}})$ generates our best $f$ approximation, we will use this function $g$ to predict the output $y_0$ for a new data point $\boldsymbol{x}_0$.

### 1.2.1 Least squares regression

In *Ordinary least squares linear regression* (OLS) we try to optimize by minimizing the L2 norm of the difference between the true value $y_i$ and the predictor $g(\boldsymbol{x}^{(i)}; \boldsymbol{w}) = \boldsymbol{w}^{\mathbf{T}} \boldsymbol{x}^{(i)}$. The solution is denoted as:

$$\hat{\boldsymbol{w}}_{LS} = \underset{\boldsymbol{w} \in R^p}{arg\,min} \|\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y}\|^2 , \tag{2}$$

which upon differentiation results in:

$$\hat{\boldsymbol{w}}_{LS} = (\boldsymbol{X}^{\boldsymbol{T}}\boldsymbol{X})^{-1}\boldsymbol{X}^{\boldsymbol{T}}\boldsymbol{y} . \tag{3}$$

Once the solution to OLS is obtained, we compute $\hat{y}$ to predict the best fit as follows:

$$\hat{\boldsymbol{y}} = \boldsymbol{X}\hat{\boldsymbol{w}}_{LS} . \tag{4}$$

The average error can be written as:

$$|\bar{E}_{in} - \bar{E}_{out}| = 2\sigma^2 \frac{p}{n} , \tag{5}$$

where $\overline{E}_{in}$ and $\overline{E}_{out}$ are the average in-sample and out-of-sample error and $\sigma$ is the noise. This shows that for $p \gg n$, where the number of features are comparable to the number of sample data results in large errors. For better optimizations we use regularization, $L_2$ penalty called Ridge regression and $L_1$ penalty knows as LASSO.

### 1.2.2 Ridge regression

Like OLS we needed to solve Eq.2, for Ridge regression we have the following equation:

$$\hat{\boldsymbol{w}}_{Ridge}(\lambda) = \underset{\boldsymbol{w}\in R^p}{arg\,min}(\|\boldsymbol{Xw} - \boldsymbol{y}\|_2^2 + \lambda\|\boldsymbol{w}\|_2^2) , \tag{6}$$

where the additional term is the $L_2$ norm of the optimizing parameter.

The design matrix for Ridge regression is obtained by differentiation of Eq.6 to give Eq.7.

$$\hat{\boldsymbol{w}}_{Ridge}(\lambda) = (\boldsymbol{X^T X} + \lambda I_{p\times p})^{-1}\boldsymbol{X^T y} . \tag{7}$$

In order to compare the fit between OLS and Ridge regression, we perform a singular value decomposition (SVD) on $\boldsymbol{X}$. It is given as:

$$\boldsymbol{X} = \boldsymbol{UDV^T} , \tag{8}$$

where $\boldsymbol{U} \in R^{n\times p}$ & $\boldsymbol{V} \in R^{p\times p}$ are orthogonal matrices. $\boldsymbol{D}$ is a diagonal matrix $\boldsymbol{D} \in R^{p\times p}$. Using the previous equation in Eq.7 gives the Ridge estimator as:

$$\hat{\boldsymbol{w}}_{Ridge} = (\boldsymbol{V}(\boldsymbol{D}^2 + \lambda I)^{-1}\boldsymbol{DU^T y} , \tag{9}$$

which means

$$\hat{\boldsymbol{y}}_{Ridge} = \boldsymbol{X}\hat{\boldsymbol{w}}_{Ridge} = \boldsymbol{UD}(\boldsymbol{D}^2 + \lambda I)^{-1}\boldsymbol{DU^T y} . \tag{10}$$

This shows that Ridge regression performs a shrinkage. A more detailed explanation can be found in [2].

### 1.2.3 LASSO regression

This $L_1$ regularization penalty is knows as the "least absolute shrinkage and selection operator" and it is defined as:

$$\hat{\boldsymbol{w}}_{LASSO}(\lambda) = \underset{\boldsymbol{w}\in R^p}{arg\,min}\|\boldsymbol{Xw} - \boldsymbol{y}\|_2^2 + \lambda\|\boldsymbol{w}\|_1 , \tag{11}$$

in both Ridge and LASSO regression there is a parameter $\lambda$ known as hyper-parameter. A more detailed explanation can be found in [1].

## 1.3 Bias-variance tradeoff

For example let's consider the least-squares regression to construct a predictor $f(\boldsymbol{x};\, \hat{\boldsymbol{\theta}})$ that provides our model's estimation for a new data point $\boldsymbol{x}$. For any model evaluation we try to maximize $R^2$ score to 1 and it is defined as:

$$R^2 = 1 - \frac{\sum_{i=1}^{n} |y_i^{true} - y_i^{predict}|^2}{\sum_{i=1}^{n} |y_i^{true} - \frac{1}{n}\sum_{i=1}^{n} y_i^{predict}|^2} \tag{12}$$

By minimizing a cost function defined by the mean squared error, our estimator is defined as:

$$C(\boldsymbol{y}, f(\boldsymbol{X}; \boldsymbol{\theta})) = \sum_i (y_i - f(\boldsymbol{x_i}; \boldsymbol{\theta}))^2 = \mathbb{E}[(\boldsymbol{y} - \tilde{\boldsymbol{y}})^2] \, , \tag{13}$$

where the mean squared error (MSE) is defined as:

$$MSE(\boldsymbol{y}, \tilde{\boldsymbol{y}}) = \sum_i (y_i - f(\boldsymbol{x_i}; \boldsymbol{\theta}))^2 = \frac{1}{n} \sum_{i=0}^{n-1} (y_i - \tilde{y}_i)^2. \tag{14}$$

The estimates of the parameters for a given dataset $\boldsymbol{D}$ is defined as:

$$\hat{\boldsymbol{\theta}}_D = arg \min_{\theta} C(\boldsymbol{y}, f(\boldsymbol{X}; \boldsymbol{\theta})) \, , \tag{15}$$

The error in the cost function changes with each dataset $\boldsymbol{D}_k = (\boldsymbol{y_k}, \boldsymbol{X_k})$. As there are several possible datasets, we compute the expectation value for all datasets $\mathbb{E}_D$. Equation 15 can be expressed as:

$$\begin{aligned} \mathbb{E}[(\boldsymbol{y} - \tilde{\boldsymbol{y}})^2] &= \mathbb{E}[(\boldsymbol{y} - \mathbb{E}[\tilde{\boldsymbol{y}}] + \mathbb{E}[\tilde{\boldsymbol{y}}] - \tilde{\boldsymbol{y}})^2] \\ &= \mathbb{E}[\boldsymbol{y} - \mathbb{E}[\tilde{\boldsymbol{y}}]^2] + \mathbb{E}[\tilde{\boldsymbol{y}} - \mathbb{E}[\tilde{\boldsymbol{y}}]^2] + 2\mathbb{E}[\boldsymbol{y} - \mathbb{E}[\tilde{\boldsymbol{y}}][\tilde{\boldsymbol{y}}] - \mathbb{E}[\tilde{\boldsymbol{y}}]] \\ &= (\boldsymbol{y} - \mathbb{E}[\tilde{\boldsymbol{y}}])^2 + \mathbb{E}[[\tilde{\boldsymbol{y}} - \mathbb{E}[\tilde{\boldsymbol{y}}]]^2] \end{aligned} \tag{16}$$

From Eq.16 bias is the difference between the true value and the expectation value of our estimator and it is defined as:

$$Bias^2 = \sum_i (\boldsymbol{y} - \mathbb{E}[\tilde{\boldsymbol{y}}])^2, \tag{17}$$

and the variance is the measure of the fluctuations of our estimator because of the finite sample size and it is defined as:

$$Var = \sum_i \mathbb{E}[[\tilde{\boldsymbol{y}} - \mathbb{E}[\tilde{\boldsymbol{y}}]]^2]. \tag{18}$$

The out-of-sample can now be defined as:

$$E_{out} = Bias^2 + Var + Noise. \tag{19}$$

The bias-variance tradeoff helps to choose the appropriate model complexity. A more detailed description can be found [3].

# 2 Methods & Implementations

In this section, we describe the methodology used to perform the regression analysis on the Boston Housing data. The codes developed for the analysis is based on the use of Python Functionalities, especially libraries from Pandas for simple implementation of the data-set and SciKit-Learn which has many optimized methods for fast and simple application of the regression analysis.

## 2.1 The Franke function

Since we are dealing with multi-variable functions, we start with the simplest system of two-dimensional function called the Franke's Function. The first step is to define the function as given in Eq.1 and to generate the relevant data-set. Python has an in-built method for handling Franke's Function using which we write the definition as shown below where, we have added a stochastic noise following normal distribution.

```
#Produce the data points according to Franke function and add error to the
    points
def FrankeFunction(n_pts):
    global x
    global y
    x = np.arange(0, 1, 1/math.sqrt(n_pts))
    y = np.arange(0, 1, 1/math.sqrt(n_pts))
    x, y = np.meshgrid(x,y)
    term1 = 0.75*np.exp(-(0.25*(9*x-2)**2) - 0.25*((9*y-2)**2))
    term2 = 0.75*np.exp(-((9*x+1)**2)/49.0 - 0.1*(9*y+1))
    term3 = 0.5*np.exp(-(9*x-7)**2/4.0 - 0.25*((9*y-3)**2))
    term4 = -0.2*np.exp(-(9*x-4)**2 - (9*y-7)**2)
    return term1 + term2 + term3 + term4 + np.random.normal(0, 0.1,x.shape)
z = FrankeFunction(400)
```

### 2.1.1 Ordinary least squares regression

We now perform a least square regression on the defined Franke's Function with polynomials in the variables x and y up to fifth order using functionalities of Scikit-Learn to split the data into train and test and to define the design matrix.

```
def polyfit(train,test,target,target_test,degree,model):
    #Produce the design matrix
    poly = PolynomialFeatures(degree)
    X1_train=poly.fit_transform(train)
    X1_test=poly.fit_transform(test)
    #Fitting the model
    model.fit(X1_train,target)
    ypredictR = model.predict(X1_train)
    ypredictRtest = model.predict(X1_test)
    return ypredictR,ypredictRtest
```

We obtained the fit parameters on the the training data using based on Eq.3 and computed the cost function [1] namely the Mean Squared Error (MSE) Eq.5 and the R2 Score Eq.12 for the test and train set of data. An interesting test is to check for the variation of the MSE as a function of the complexity of the fit as shown in Fig.1.

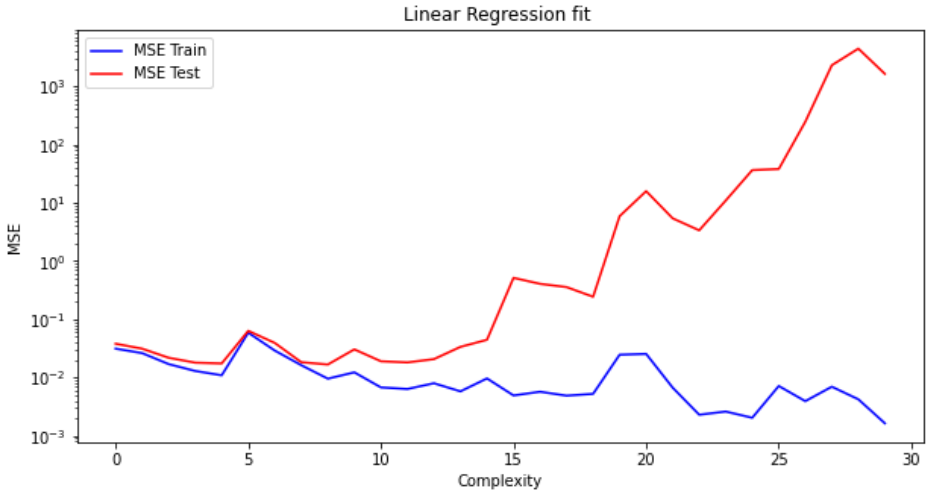


*Figure 1: Variation of the MSE as a function of the complexity of the fit for a least squares regression fit for Test(red) and Train (blue) data.*

It was clearly observed that at a certain complexity the MSE of the training set diverged from that of

the test set and we were at the point of overfitting. Fig.2 shows the value along with the confidence intervals of the fit parameters $\beta$ wherein the statistical significance of the parameters can be inferred form.

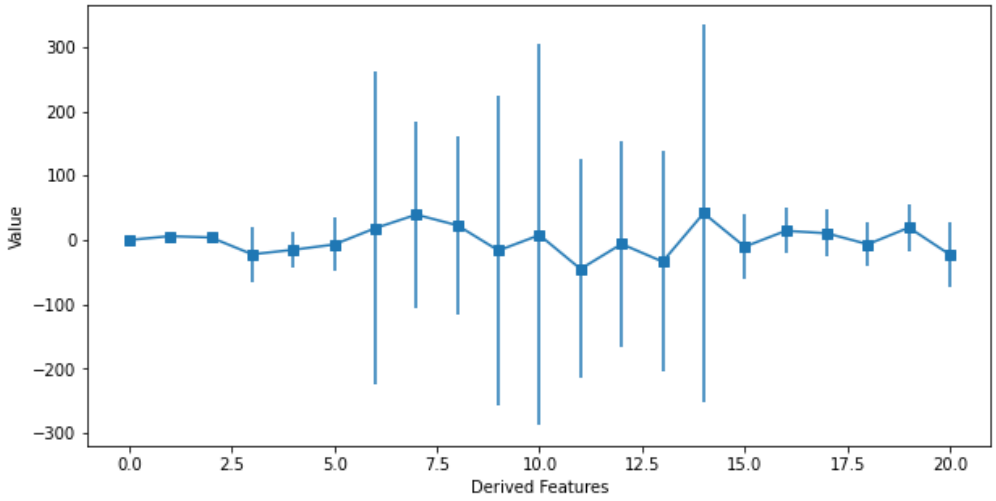


*Figure 2: Value along with the coverage factor of the derived features of the fit for a polynomial of order 5.*

Following the discussion in the previous section the next step was to look for the variation in the bias-variance tradeoff with application of Bootstrap resampling as a function of the complexity. This was done using the following methodology, to get the plot as shown in Fig.3.

```
#define the bootstrap function to fit the data with bootstraps resampling
def bootstrap(XY_train,XY_test,z_train,z_test,degree,alpha,n_bootstraps):
    y_pred = np.empty((z_test.shape[0], n_bootstraps))
    R=Ridge(alpha)
    #Define Model
    #R=LinearRegression()
    #Produce the design matrix
    poly = PolynomialFeatures(degree)
    X1_train=poly.fit_transform(XY_train)
    X1_test=poly.fit_transform(XY_test)
    #bootstraping the data points and fitting the model
    for j in range(n_bootstraps):
        x_, y_ = resample(X1_train, z_train)
        y_=y_.reshape(-1,1)
        y_pred[:, j] = R.fit(x_, y_).predict(X1_test).ravel()
    bootstraps=np.mean(y_pred, axis=1, keepdims=True)
    bootstraps=bootstraps.reshape(-1)
    #Saving the results of the bootstrap
    bias = np.mean( (z_test - bootstraps)**2 )
    variance = np.mean( np.var(y_pred, axis=1, keepdims=True) )
    return bias,variance
```

A k-fold cross validation [1] was implemented with the use of Scikit-learn's cross_val_score functionality. The data was properly scaled with StandardScaler which is found to be the one with the least MSE before the implementation of the method. The basic idea of cross validation is this: the data is reshuffled and separated into k groups one of those groups is taken as the test data while the rest are clumped together into the training data which is used to fit the model and calculate the cost function. This is repeated k-times with a different group serving as the test data each time and finally an average of the cost function is evaluated which gives the cross validation score.

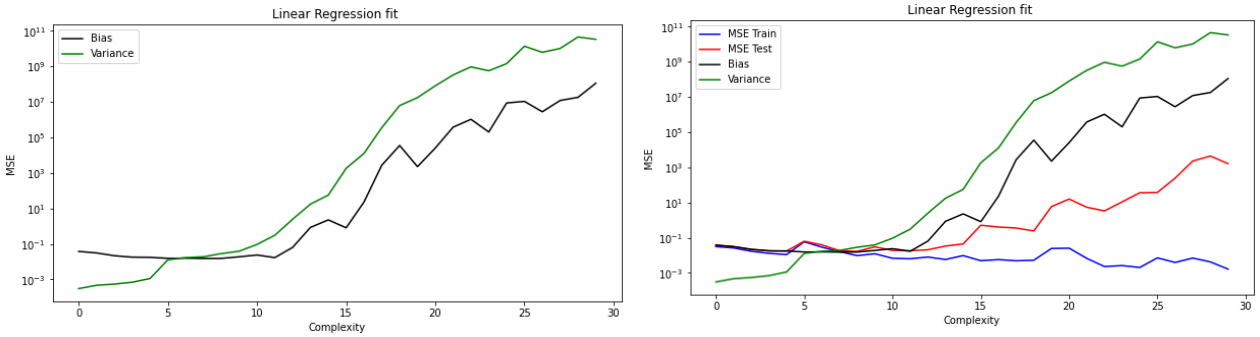


*Figure 3: Left) variation of the bias-variance as a function of the complexity of the fit. Right) the variation of the MSE values have been included for comparison.*

The trend of the cross validation score was similar to the one for normal MSE however it can be noted that it decreased with increasing number of folds used for the calculation.

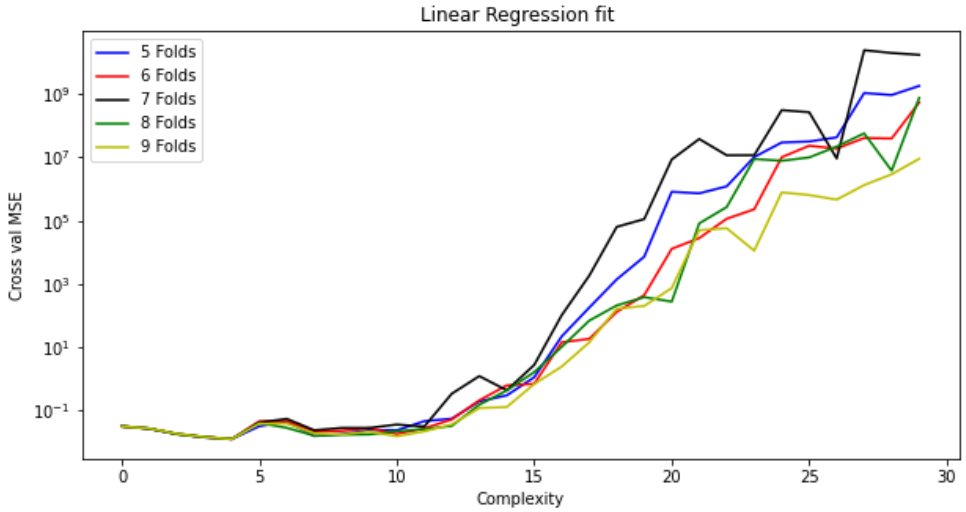


*Figure 4: Variation of cross-validation score with increasing number of folds.*

### 2.1.2 Ridge regression

The Scikit-Learn functionality was also used to perform a ridge regression as given by Eq.6. The MSE value was evaluated for different values of the hyper-parameter $\lambda$ for a polynomial order of 5 as shown in Fig.5. Bootstrap resampling was also implemented as before to identify the variation in the bias-variance tradeoff as a function of the hyper-parameter $\lambda$.

```
#Loop over hyperparameter values complexity set to 5
for i in range(tot):
    #calculate bias variance
    bias[i],variance[i] = bootstrap(XY_train,XY_test,z_train,z_test,degree=5,
    alpha=val[i],n_bootstraps=400)
    model=Ridge(val[i])
    ind[i]=i
    #calculate cost functions
    r2_train[i],mse_train[i],r2_test[i],mse_test[i]=score(XY_train,XY_test,
    z_train,z_test,degree=5,model=model)
    print(mse_test[i])
```

We can see that the bias was almost exactly the same as the MSE of the test data for all values of the Hyper-parameter, while the variance, though very little, decreased. This decrease in the variance

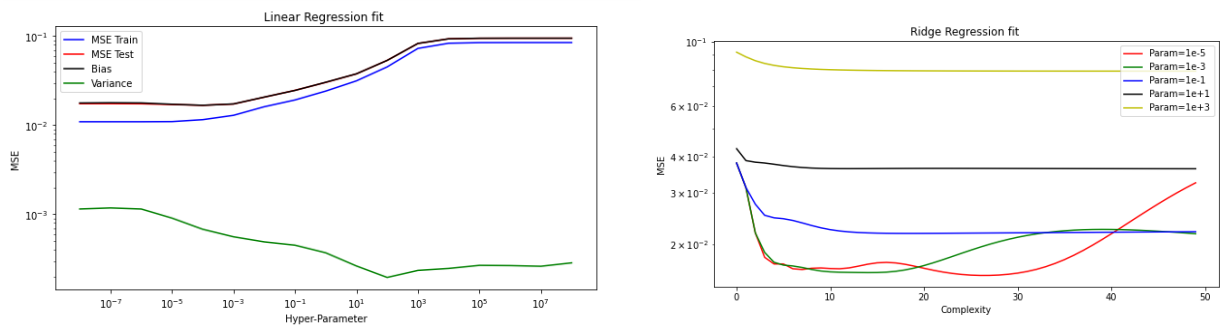


*Figure 5: Left) variation of the bias-variance and the test and train MSE as a function of the value of the Ridge Hyper-parameter. Right) the variation of the test MSE values as a function of the complexity for different values of the Hyper-parameter.*

pointed to greater applicability and flexibility of the fit. Furthermore from the figure it is clearly seen that the MSE of the test data increased with increasing value of the Hyper-parameter. Fig.6 Shows the change in the MSE for a given value of the Hyper-Parameter $\lambda = 0.001$ with the complexity of the fit.

This can be compared to Fig.3 (linear regression) where we see that though with the increase in the complexity the test data MSE increased, however this increase was not as drastic as the one for linear regression. Furthermore the values of the bias and variance are more stable.

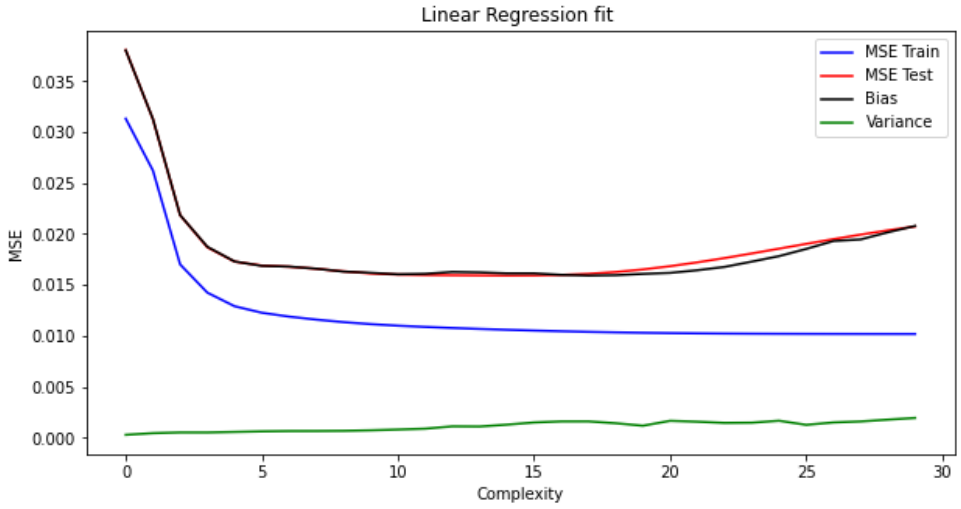


*Figure 6: The bias-variance and the test and train MSE for ridge regression as a function of the complexity of the fit for $\lambda = 0.001$.*

## 2.2 Boston housing

Now we focus on the dataset for the Boston Housing. It has 506 data points, 13 Features and 1 target which is the Median value of owner-occupied homes in USD 1000 (MEDV). The objective of this dataset is to predict the value of prices of the houses as a function of the list of features. This dataset is already included in the Scikit-Learn's library. We made use of Pandas here, which makes the handling of the column-based data simple as shown below.

This prediction problem was therefore a general regression case but here we have 13 features. Therefore, it became convenient to use SciKit-Learn's functionalities to make the design matrix. Note that

since we have a large number of features and relatively less number of data points our number of derived features will run up the number of data points with increasing complexity. Therefore, it was important here to identify the features, which are more relevant to the regression problem. One way was to look at the correlation matrix and identify the more relevant features (Fig.7). Furthermore, to remove multicollinearity we also identified features with high correlation among themselves and remove their contribution.

```
from sklearn.datasets import load_boston
boston_dataset = load_boston()
# Checking the containing elements
boston_dataset.keys()
boston = pd.DataFrame(boston_dataset.data, columns=boston_dataset.feature_names)
print(boston)
boston['MEDV'] = boston_dataset.target
# display(boston)
# Check for the number of missing artibutes for each feature
boston.info()
boston.isnull().sum()
# Check for features with low variance
boston.var()
```

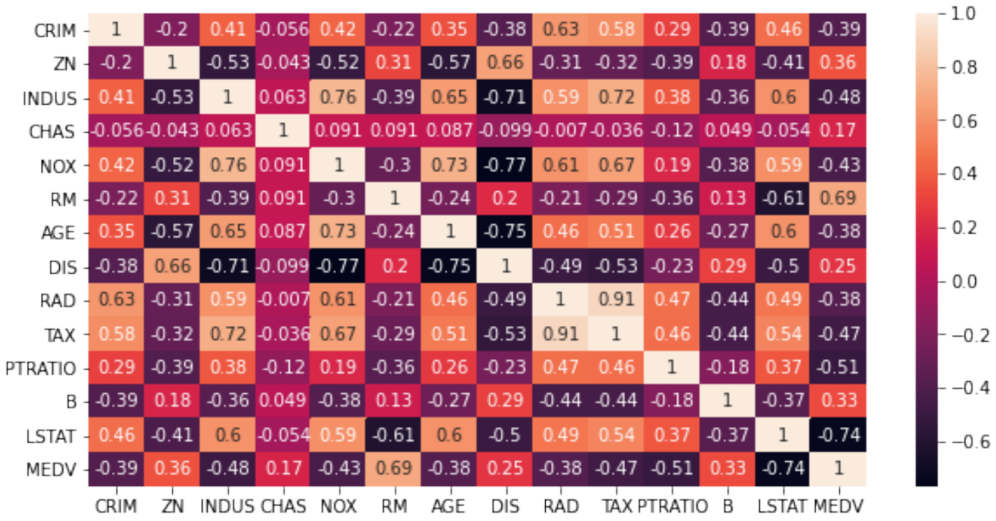


*Figure 7: Correlation matrix among different features of the Boston housing data.*

### 2.2.1 Least squares regression

We performed an ordinary least squares regression on the data with the use of SciKit-Learn's regression functionalities and obtained the cost function values for the Test data i.e. the MSE and the R2-Score for varying values of the complexity (cross validation was also performed, check code for further details).

```
maxdegree = 7
mse_arr=np.zeros(maxdegree)
index=np.zeros(maxdegree)
r2sc=np.zeros(maxdegree)
threshold = 0                    #selected all parameters
# loop over different order polynomials
for i in range(maxdegree):

   #Call the getCorrelatedFeatures function to display the filtered features
```

```
    correlation_value, feature = getCorrelatedFeature(correlation_matrix['MEDV'
    ], threshold)
    correlated_data = boston[correlation_value.index]
    #correlated_data = correlated_data.drop(labels = ['RAD'], axis = 1)
    #correlated_data = correlated_data.drop(labels = ['TAX'], axis = 1)
    degree = i+1
    index[i]=i+1
   #Call fit function again
    y_test, y_predict = get_y_predict(correlated_data,degree)
   #Calculate the cost function
    mse_arr[i]= mean_squared_error(y_test, y_predict)
    r2s=r2_score(y_test, y_predict)
    r2sc[i]=r2s
```

Where the fit function used is described below,

```
# provides a fit and the prediction for the chosen features
def get_y_predict(correlated_data,degree):
    #Drop target Label
    X = correlated_data.drop(labels = ['MEDV'], axis = 1).to_numpy()
    #Define Target
    y = correlated_data['MEDV'].to_numpy()
    print(X.shape,y.shape)

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2,
    random_state = 42)
    #scaler = RobustScaler()
    # scaler = MinMaxScaler()
    scaler = StandardScaler()                    #check scaling effect
    poly = PolynomialFeatures(degree,include_bias=False)
    #define Design Matrix
    X1_train=poly.fit_transform(X_train)
    X1_test=poly.fit_transform(X_test)
    #define Model and Fit
    model = LinearRegression()
    model.fit(X1_train, y_train)
    y_predictTest = model.predict(X1_test)
    return y_test, y_predictTest
```

For the current displayed example taking all parameter we got a minimum MSE of 14.26 at complexity equals to 2 corresponding to an R2 Score of 0.806 on the Test data as can be seen in Fig.8.

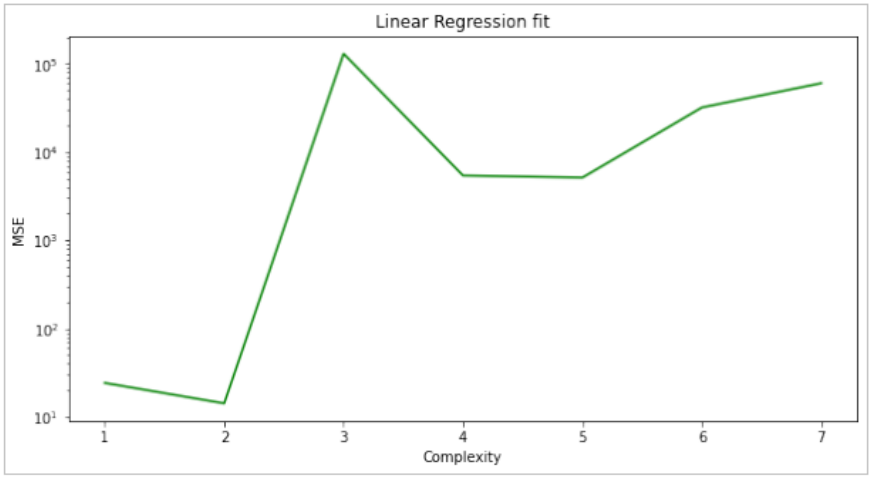


*Figure 8: Variation of MSE for least squares regression fit as a function of the complexity .*

Following the discussion on the identification of relevant features we observed how the cost function varied as a function of the number of features taken based on their correlation with the target (MEDV) and among themselves.

```
# the threshold value corresponds to the correlation value
threshold = 0
# Obtain predictions for different threshold
for i in range(8):
    k = i
    threshold = k/10

#   calls the getCorrelatedFeatures function to select the features with
    coreelation values above the threshold
    correlation_value, feature = getCorrelatedFeature(correlation_matrix['MEDV'
    ], threshold)
    correlated_data = boston[correlation_value.index]
    # high correlation among 'RAD' and 'TAX' check effect of removing
    # if 'RAD' in correlated_data.columns:
    #    correlated_data = correlated_data.drop(labels = ['RAD'], axis = 1)
    #if 'TAX' in correlated_data.columns:
    #    correlated_data = correlated_data.drop(labels = ['TAX'], axis = 1)
    degree = 2                                              #check for polynomial
    order 2
    y_test, y_predict = get_y_predict(correlated_data,degree)

#   calls the performance_metrics function to display the accuracy of the model

performance_metrics(correlated_data.columns.values, threshold,y_test, y_predict)
```

We can see in Fig.9 that the MSE varied greatly with the number of features taken. For a lower order polynomial we can take more features in the fit as compared to a higher order one. It was observed that the best MSE value for a polynomial of order 3 was obtained by taking 6 features while on the other hand for a polynomial of order 4 a comparable (but poorer) MSE value was obtained for 3 features.

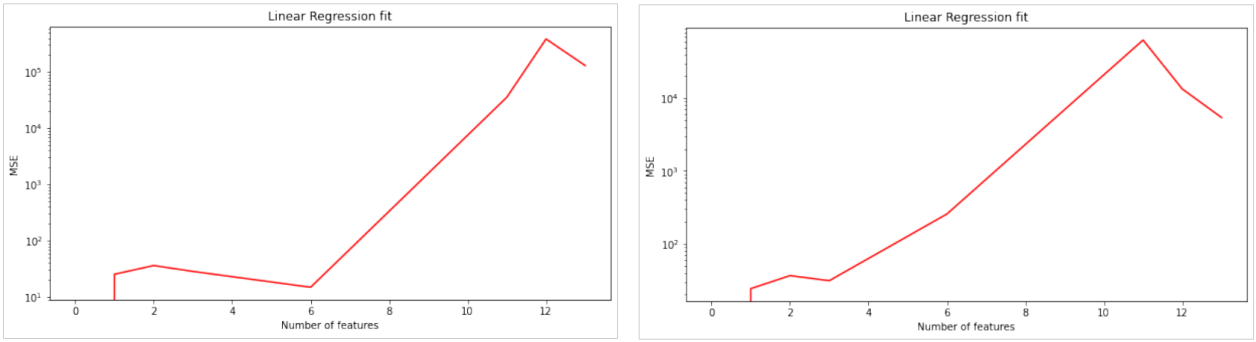


*Figure 9: Variation of MSE for least squares regression fit as a function of the number of features for (Left) polynomial order 3 (right) polynomial order 4.*

### 2.2.2 Ridge regression

Further analysis using Ridge regression was implemented in the fit function to see if our model produced improved results.

```
I = np.identity(len(X1_train.T @ X1_train))
beta = p.linalg.inv(X1_train.T @ X1_train + alpha*I) @ X1_train.T @ y_train
y_predictTest = X1_test @ beta
```

The hyper-parameter and the complexity was optimized by studying the variation in the MSE and R2-score values on the Test data as shown in Fig.10.

```
#Ridge regression study over different hyper parameter values
stop = 11
folds=5
start=3
tot=start+stop+1
val=np.logspace(-start,stop,num=tot)            #define range of hyper-parameter
    values
mse_arr=np.zeros(tot)
kf_arr=np.zeros(tot)
index=np.zeros(tot)
r2sc=np.zeros(tot)
threshold = 0.0        #All features taken
# loop over different hyperparameter values
for i in range(tot):
#     Call the getCorrelatedFeatures function to display the filtered features
    correlation_value, feature = getCorrelatedFeature(correlation_matrix['MEDV'
    ], threshold)
    correlated_data = boston[correlation_value.index]
    #correlated_data = correlated_data.drop(labels = ['RAD'], axis = 1)
    #correlated_data = correlated_data.drop(labels = ['TAX'], axis = 1)
    degree = 2       #Done for Second order Polynomial
    index[i]=val[i]
   #Call ridge regresion function
    y_test, y_predict,mse_kf = get_y_predictkf_ridge_d(correlated_data,degree,
    folds,val[i])
   #Calculate cost function
    mse_arr[i]=mean_squared_error(y_test, y_predict)
    kf_arr[i]=mse_kf
    r2s=r2_score(y_test, y_predict)
    r2sc[i]=r2s
```

In this example a minimum MSE equals to 12.78 was found for $\lambda = 10^3$ with R2 Score of 0.826.

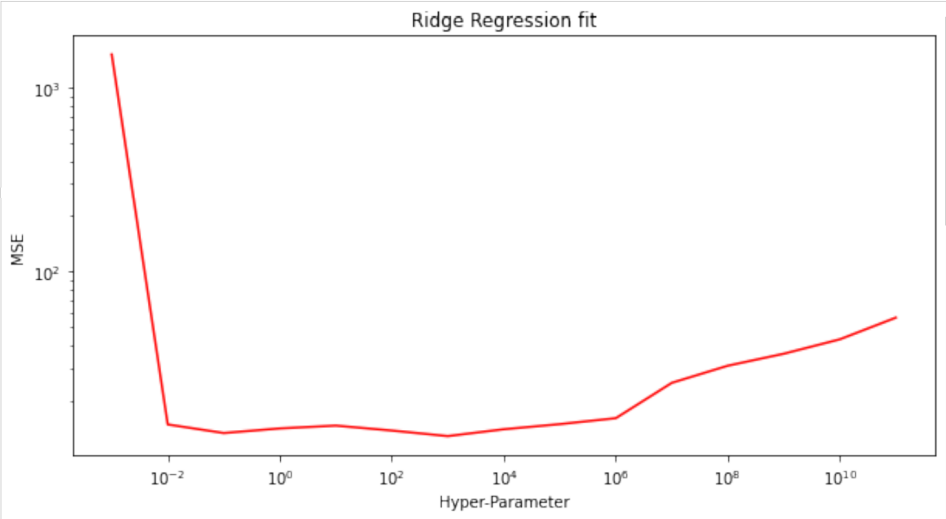


*Figure 10: Variation of MSE for Ridge regression as a function of the value of the Hyper parameters.*

Again, a study was done on the effect of removing features and how it affected the cost functions.

# 3 Results and discussion

## 3.1 Franke function

Fig.11 shows the Fit (left) and the Actual data distribution (right) of the Franke's Function used in the current problem. The fit was based on linear regression.

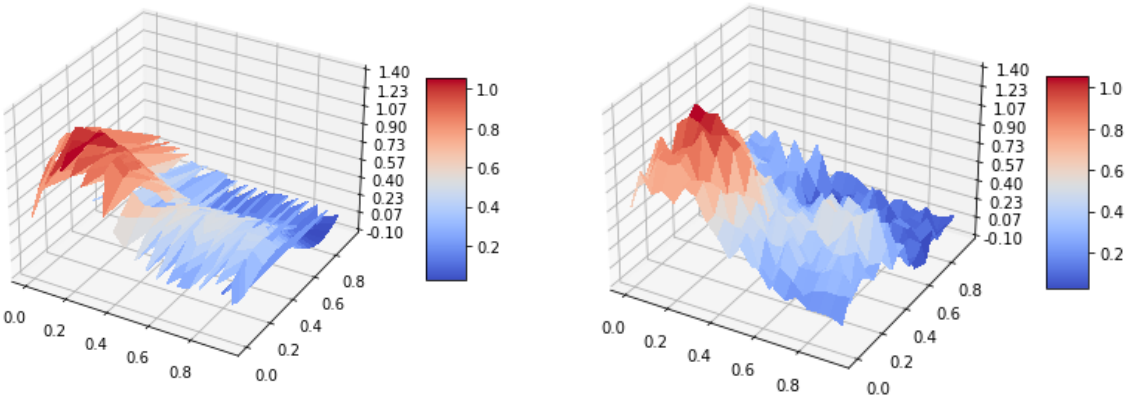


*Figure 11: Least squares regression fit of the Franke Function data right) actual distribution with added noise.*

For the case left) of least squares regression with the fit for a polynomial of order 5 we got the cost function values as,

| MSE Train | $R_2$ Score Train | MSE Test | $R_2$ Score Test |
|---|---|---|---|
| 0.0109 | 0.87 | 0.0174 | 0.816 |

This MSE value was compared to the one obtained via Cross Validation,

MSE cross Validation (8 Folds) = 0.0125.

The cross validation is a resampling technique based on the application of statistical tools, which gives an idea of how well the model performs. Such resampling techniques involves resampling and refitting the data over and over again to get to a cost function value much closer to the real value.

Using the Ridge regression in the same polynomial of order 5 with the value of the hyper-parameter $\lambda = 10^4$ we got the relevant cost function values as,

| MSE Train | $R_2$ Score Train | MSE Test | $R_2$ Score Test |
|---|---|---|---|
| 0.0115 | 0.8636 | 0.0167 | 0.824 |

Again we compared to the cross validation score of MSE cross Validation (8 Folds) = 0.013.

## 3.2 Boston housing data

In the least squares regression case the important variables to be considered were not only the complexity of the model but as discussed, also the number and type of features to be used for the fit. With regards to the linear regression of the Boston housing data we found that the most optimum fit was obtained for Complexity = 2 i.e a polynomial fit of order 2 using 12 features, removing the feature 'CHAS' which had the lowest correlation value with the target.

| MSE Test | $R_2$ Score Test |
|---|---|
| 13.34 | 0.818 |

For this system we got the minimum cost function values for the test data as,

In the ridge regression case we added another variable into the fitting conditions, namely the Hyper-parameter. We found that the optimum condition corresponded to using all 13 Features with the Hyper-parameter value of 600 and a polynomial order of 2. We got the minimized cost function values as,

| MSE Test | $R_2$ Score Test |
|---|---|
| 12.72 | 0.827 |

These results were compared to the values obtained in article [4].

Therefore, we see a slight improvement in the fit with the Ridge regression but with more features which follows from the fact that Ridge regression reduces the effect of the multi-collinearity as well as reduces the influence of the higher order derived terms.

Regarding multi-collinearity however, even though the features 'RAD' and 'TAX' had a correlation of 0.91 we observed that removing either of them from the list of features for the fit leaded to a deterioration of the cost function values. Nevertheless, it is also seen that the effect of removing some of the features was not huge and it was also understood that using 12 features was computationally expensive.

To give an example, using only 5 features in polynomial order 3 using linear regression we get an R2 Score of 0.784 with MSE of 15.86.

# 4 Conclusion

We therefore present in the current work the basic concepts applied to supervised learning through linear regression. This work serves to present these concepts in a form, which is pedagogical and intuitive by looking at how the model behaved under different conditions for the fit parameters.

For the Boston housing data we obtained a model which provided us an R2 Score of $\sim 0.83$ which value can be compared to the work previously done in reference [4]. It was, however, also understood that with much deeper analysis these values can be further improved upon. In addition to the regression analysis presented here it would be interesting to compare to results of regression done through application of other concepts such as Neural Networks.

# References


[1] T. Hastie, R. Tibshirani, and J. Friedman, *The elements of statistical learning: data mining, inference, and prediction*. Springer Science & Business Media, 2009.

[2] W. N. van Wieringen, "Lecture notes on ridge regression," 2020.

[3] P. Mehta, M. Bukov, C.-H. Wang, A. G. Day, C. Richardson, C. K. Fisher, and D. J. Schwab, "A high-bias, low-variance introduction to machine learning for physicists," *Physics reports*, vol. 810, pp. 1–124, 2019.

[4] D. Harrison Jr and D. L. Rubinfeld, "Hedonic housing prices and the demand for clean air," *Journal of environmental economics and management*, vol. 5, no. 1, pp. 81–102, 1978.