Machine Data and Learning

Assignment 2

- Romica Raisinghani (2021101053)

2.1 Task 1: Linear Regression

Write a brief about what function the method LinearRegression().fit() performs.

Linear Regression models the relationship between a dependent variable and one or more independent variables by fitting a linear equation to the observed data.

`LinearRegression().fit()` is a function of scikit-learn library in Python. It is a method which is used to fit a linear regression model to a supplied dataset.It finds best values of the regression coefficients y minimising the sum of squares of errors which is the sum of squares of the difference between the expected value (by the trained model) and the actual value(provided ibn the input dataset).

To train the model on a given dataset, the fit() function is utilised. It uses two arguments x and y where x is a 2-D array(matrix) of features and y is a 1-D array(vector) of corresponding target values.

The `fit` method returns the fitted `LinearRegression` object, which can be used to make predictions on new data using the `predict` method.

This is implemented in the following way:

```
from sklearn.linear_model import LinearRegression

# Creating an instance of the class LinearRegression()
model = LinearRegression()

# Fit the model to the data
model.fit(X, y)

# Use the model to make predictions on the new test data
predictions = model.predict(X_new)
```

The `LinearRegression()` method generates an instance of the scikit-learn library's LinearRegression class, which may be used to build a linear regression model and this model is stored in `model`. The `fit()` function is then invoked on the LinearRegression class object (model), using the input features (X) and target variable (y) as arguments.

2.2 Task 2: Gradient Descent

Explain how gradient descent works to find the coefficients. For simplicity, take the case where there is one independent variable and one dependent variable.

In case where there is one independent variable and one dependent variable, suppose for example:

$$y = m * x + c$$

where y is dependent variable and x is independent variable. The objective of the gradient descent algorithm it to find the optimal values of the regression parameters(m and c) such that the cost of the function is minimised. Cost of the function is basically the average of sum of squared errors where a squared error is nothing but the square of the difference between the actual value and the predicted value given by the regression line y=m*x+c

This is achieved by updating the parameter values in each iteration until we get minimum cost function. The iterations are stopped based on certain criterion such as reaching a certain number of iterations or when the change in the cost function becomes smaller than a threshold value.

Explaining the above process mathematically for the regression line y=m*x+c :

Cost equation is given by:

$$f(m, c) = \frac{1}{2n} \sum_{i=1}^{n} (y_{actual} - y_{predicted})^2$$

(denominator is 2*n and not just n for mathematical convenience)

where:

y_actual = actual values for different values of x

y_predicted = values predicted by the regression line or y_predicted=m*x+c

Hence, the cost function becomes:

$$f(m,c) = \frac{1}{2n}\sum_{i=1}^{n}(y_{actual} - (mx + c))^2$$

Next we find the partial derivative to the cost function with respect to m and c and we get:

$$\frac{\partial f(m,c)}{\partial m} = \frac{1}{2n} \cdot \frac{\partial}{\partial m}\left[\sum_{i=1}^{n}((y_i - (mx_i + c))^2\right]$$

$$=> \frac{\partial f(m,c)}{\partial m} = \frac{1}{n} \cdot \sum_{i=1}^{n}(y_i - (mx_i + c)) \cdot (-x_i)$$

Similarly,

$$\frac{\partial f(m,c)}{\partial c} = \frac{1}{2n} \cdot \sum_{i=1}^{n}(y_i - (mx_i + c))$$

$$=> \frac{\partial f(m,c)}{\partial c} = \frac{1}{n} \cdot \sum_{i=1}^{n}(y_i - (mx_i + c))$$

The update rules for m and c after each iteration are given by:

$$m = m - \alpha \cdot \frac{1}{n} \cdot \sum_{i=1}^{n}((mx_i + c) - y_i) \cdot x_i$$

$$c = c - \alpha \cdot \frac{1}{n} \cdot \sum_{i=1}^{n}(y_i - (mx_i + c))$$

where α is the learning rate, a parameter that controls the step size taken in the direction of the negative gradient.

By entering in the values of the independent variable and computing the associated predicted value of the dependent variable, the optimal parameter values may be utilised to make predictions on fresh data.

2.3 Task 3: Calculating Bias and Variance

The following formulas have been used:

`BIAS:` It is the difference between the average of predicted value and the true value

$$Bias = E[\hat{f}(x)] - f(x)$$

`VARIANCE:` It is the average of the square of predicted value and the expected predicted value

$$Variance = E[(\hat{f}(x) - E[\hat{f}(x)])^2]$$

`MEAN SQUARED ERROR:` It is the average of the square of predicted value and the predicted value

$$MeanSquaredError = E[(f(x) - \hat{f}(x))^2]$$

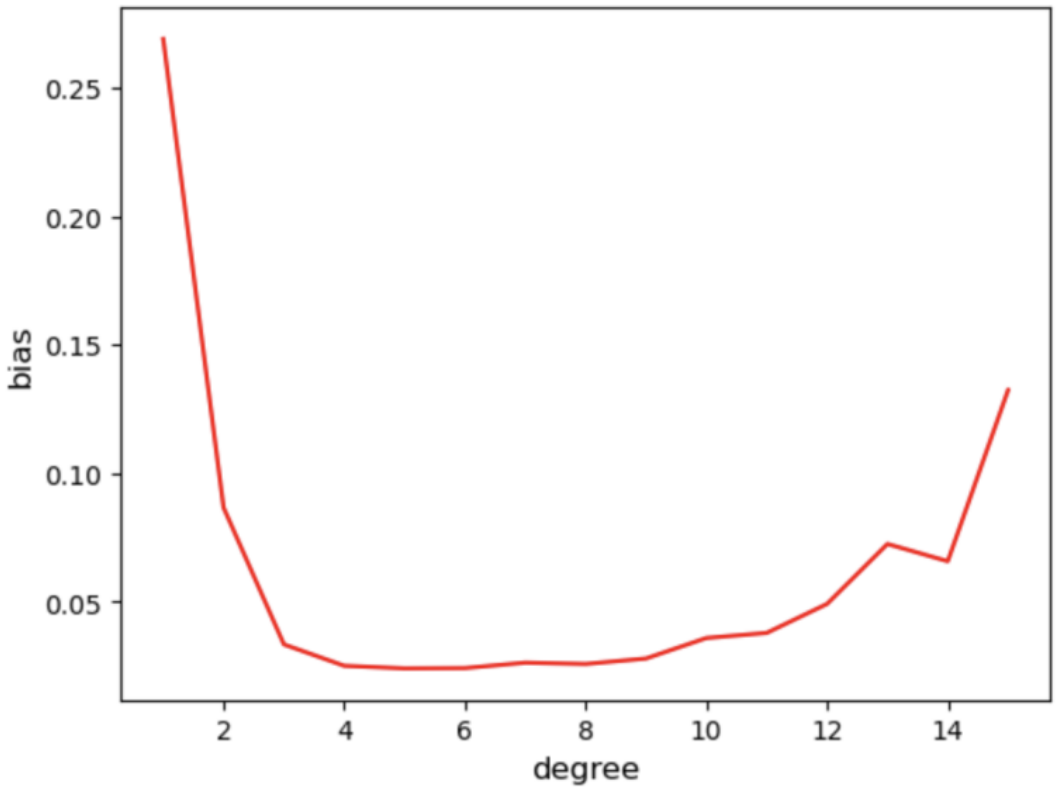
`IRREDUCIBLE ERROR:` It is the error that cannot be reduced by creating good models. It is a measure of the amount of noise in the data

$$IrreducibleError = MeanSquaredError - (Bias^2 + Variance)$$

Following is the tabulation of bias, variance and mean squared error for degrees ranging from 1 to 15:

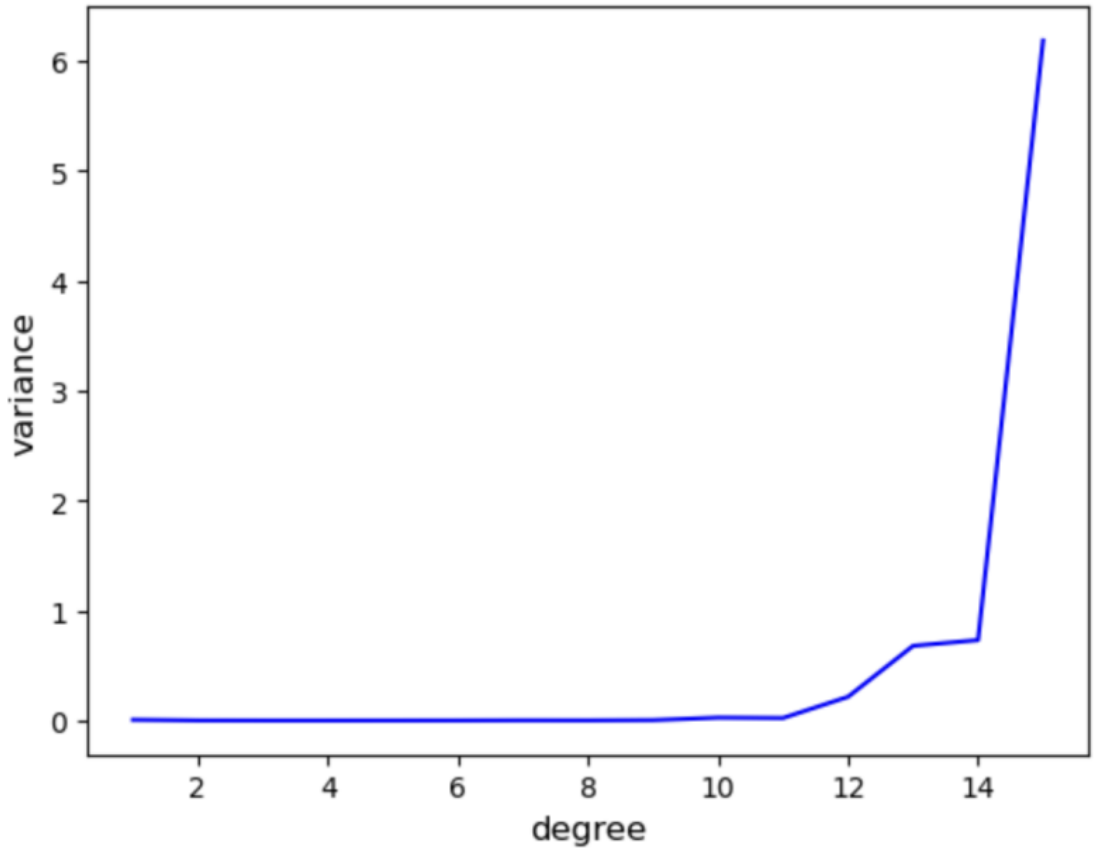
Degree	Bias	Variance	mse
1	0.269019	0.00727275	0.121834
2	0.0865393	0.00119742	0.0133201
3	0.0332713	0.00057629	0.00527597
4	0.0249476	0.000545964	0.00480833
5	0.0239383	0.000771227	0.0050005
6	0.0240628	0.0011307	0.00533878
7	0.0261704	0.00214634	0.00642266
8	0.0256659	0.00153915	0.00585038
9	0.0277896	0.00405257	0.00859177
10	0.035769	0.0269729	0.0331655
11	0.037752	0.0233748	0.0299142
12	0.0490429	0.215954	0.228452
13	0.0723846	0.679166	0.716561
14	0.0656619	0.731559	0.783631
15	0.132334	6.18169	6.46348

Below is the graph which shows how the bias varies with the degree of the polynomial:



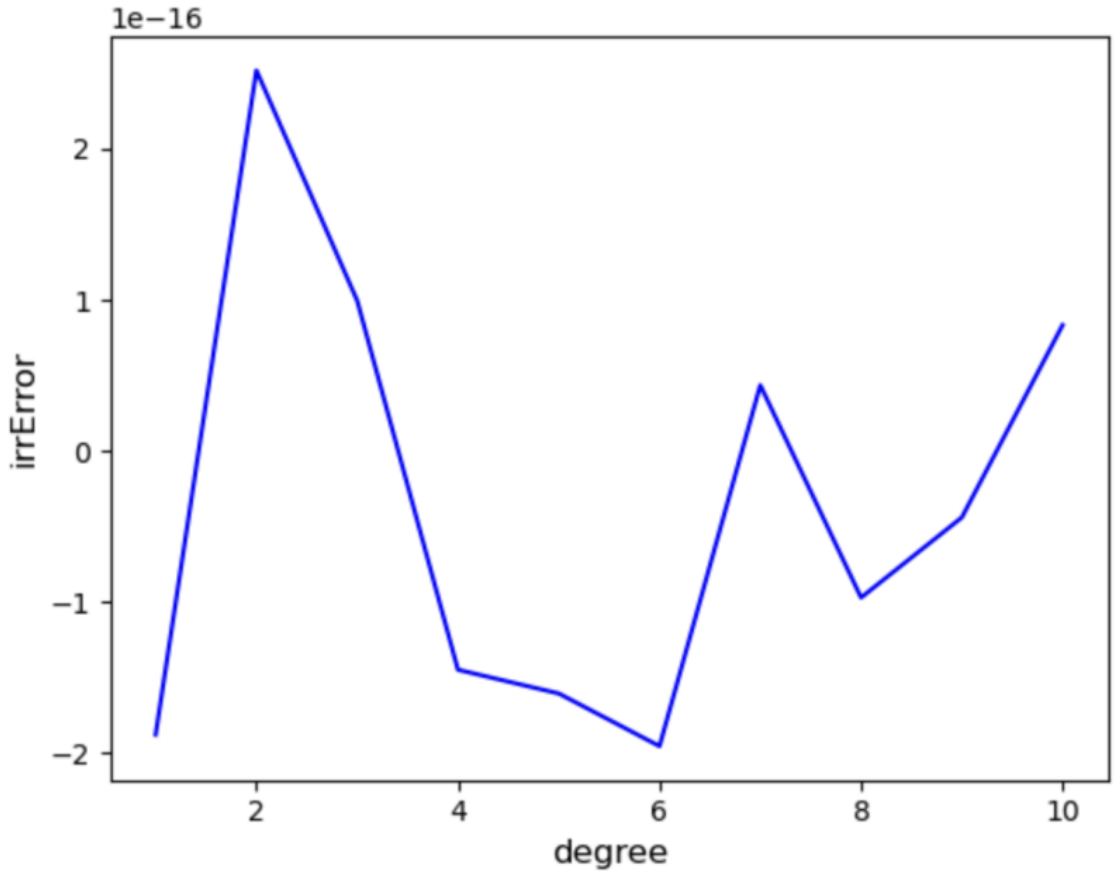
The models for degrees less than or equal to 3 were `underfitting` the provided data set, which led to an unusually high bias. As a result, the predicted values of the models differ greatly from the actual value of the training data set. The bias has very high values for degrees less than or equal to 3.
From this point on, the bias diminishes as the polynomial's degree rises, but from degree 9 onward, the bias grows.

Below is the graph which shows variance vs the degree of the polynomial:



Data spread is measured using variance. A larger variance number indicates that there is a greater fluctuation between the anticipated value and the dataset's actual value.
According to the figure, variance normally rises as polynomial degree grows because higher degree polynomials produce more complicated polynomial terms, which results in greater variation.

Below is the graph as well as the tabulation for Irreducible Error vs the degree of the polynomial:

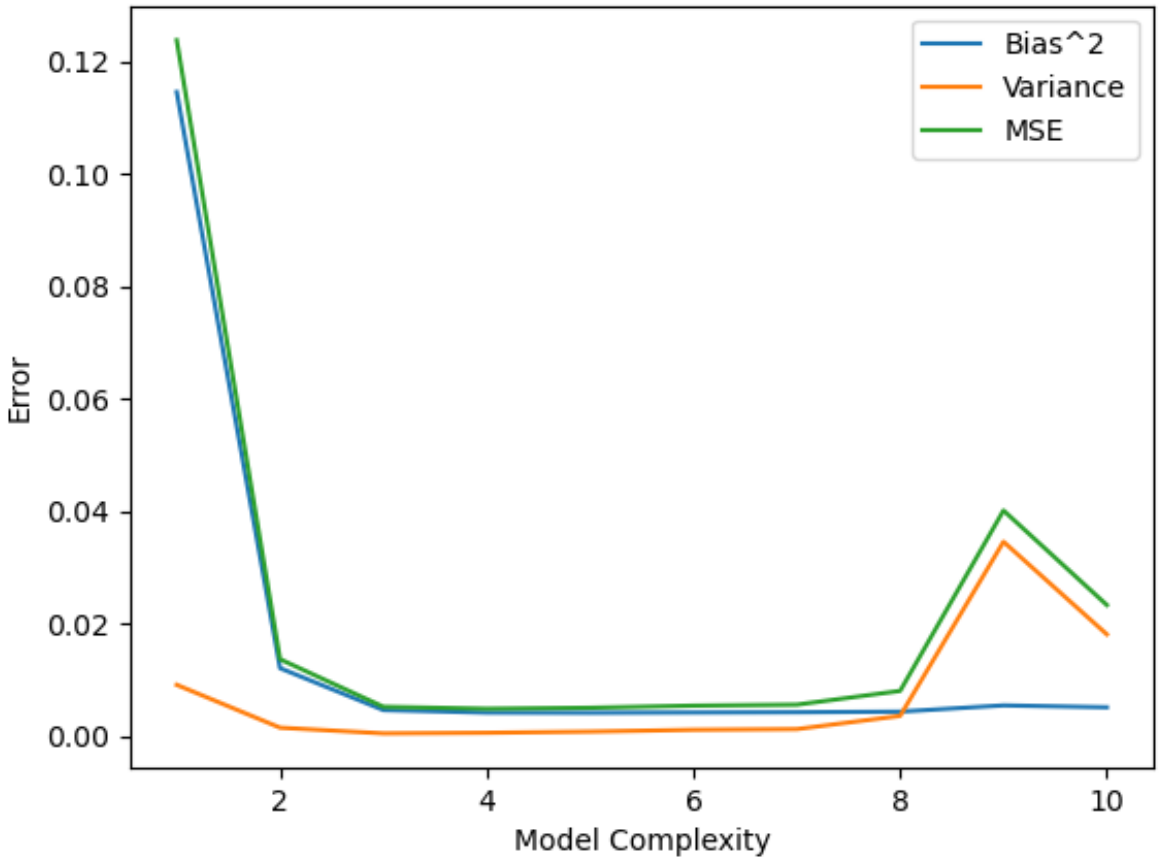

1e−16
2
1
0
−1
−2
irrError
2
4
6
8
10
degree

Degree	Irreducible Error
1	-1.88217e-16
2	2.51969e-16
3	9.94213e-17
4	-1.45175e-16
5	-1.60896e-16
6	-1.95807e-16
7	4.33681e-17
8	-9.73614e-17
9	-4.42354e-17
10	8.32667e-17
11	-6.93889e-18
12	-8.32667e-17
13	1.11022e-16
14	3.33067e-16
15	0

The `irreducible error` is on the order of 10^(-16), which is quite modest. Because the numbers are so tiny, there is likely very little noise in the data.
Several results are negative with really small values, which suggests that rounding mistakes rather than true dataset noise are responsible for some of this problem.

Why or why not the irreducible error changes as we vary the class function?

Since irreducible error is a characteristic of the dataset and not of the models, the values are in the same range for all models and are essentially the same for all models.



The bias values are rather large for degrees below 3, indicating that the models are `underfitting` the data; however, for degrees 3 and higher, the bias suddenly decreases to a much lower value and practically remains constant.

After the bias reaches degree 8, it becomes more pronounced, indicating that the models have `overfitted` the data, producing significant bias and variation.

The MSE climbs almost monotonously for degrees three and higher after beginning at a high value, falling abruptly for degrees three and below.

The lowest MSE shows that the bias and variance in this model are in their optimal equilibrium at degree=3. This model thus provides the dataset with the greatest fit.
The optimal model is produced when degree=3. The dataset has the cubic form y=ax3+bx2+cx+d, where (a,b,c) are the set of coefficients and d is the bias term for the linear model, according to analysis.

3 Bonus

Charge on the capacitor (dependent variable) is a function of time(independent variable) and varies exponentially according to the following equation:

$$Q = CV_0e^{-\frac{t}{RC}}$$

where

$$V_0 = 5Volts$$

Since, the function is exponential one, we cannot directly perform linear regression. Hence, we first take logarithm on both sides to convert the given equation in the form of y=mx+c

$$Q = CV_0e^{-\frac{t}{RC}}$$

$$Q/V_0 = Ce^{-\frac{t}{RC}}$$

$$ln(Q/V_0) = ln(Ce^{-\frac{t}{RC}})$$

$$ln(Q/V_0) = -\frac{t}{RC} + lnC$$

Comparing this with y=mx+c we get:

$$y = ln(Q/V_0), x = t, m = -\frac{1}{RC}, c = lnC$$

Here y is the dependent variable and x is the independent variable.

With the help of `LinearRegression` class, we fit a linear regression model to the data. The `model` object contains the estimated coefficients and other information about the fitted model.

We calculate slope and intercept by model coefficients where `model.coef_[0][0]` gives us the slope of the line and `model.intercept_[0]` gives us the y-intercept of the line.

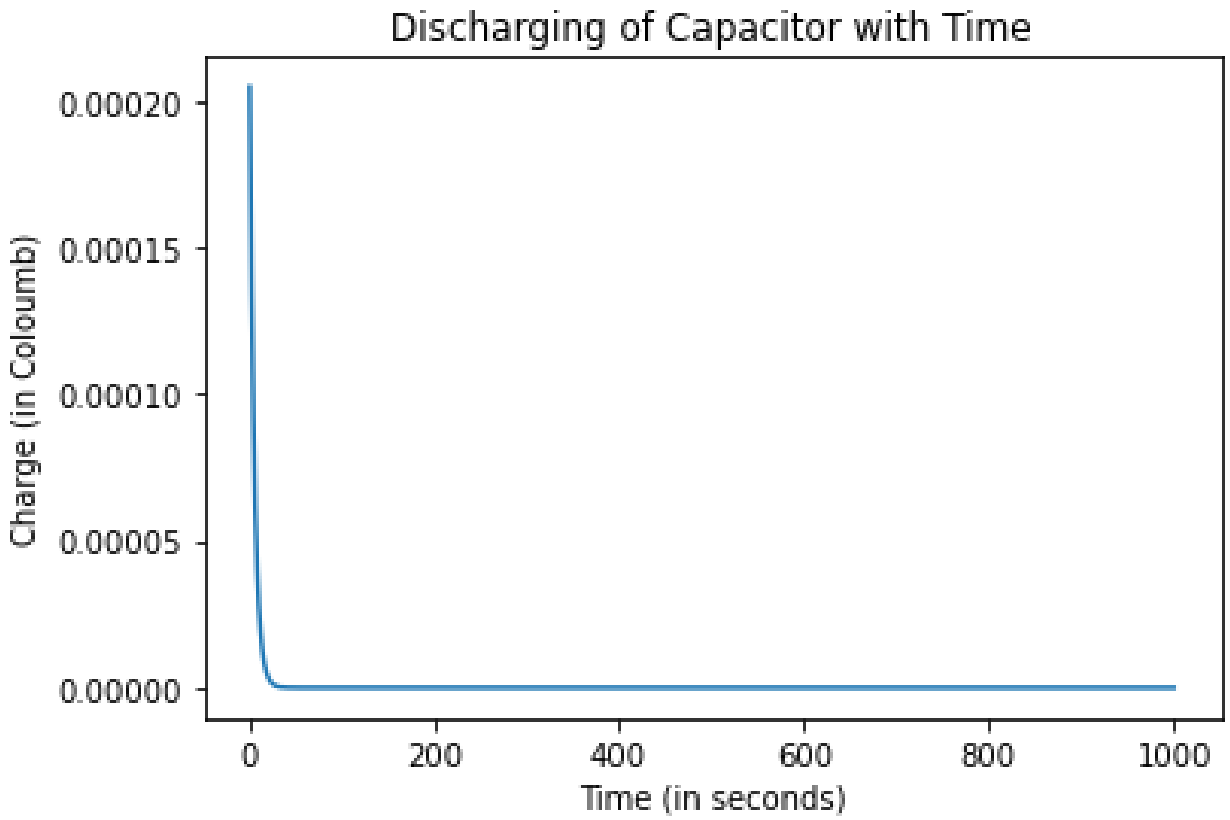
$$intercept = lnC$$

$$=> C = e^{\text{intercept}}$$

$$slope = -\frac{1}{RC}$$

$$=> R = -\frac{1}{C \cdot \text{slope}}$$

The following is the graph for Time vs Charge on the Capacitor :



The approximate values of Capacitance and Resistance calculated are as follows:

$$\text{Capacitance} = 5 \cdot 10^{-5} \text{farad}$$

$$\text{Resistance} = 10^5 \text{ohms}$$