MDL Assignment 02

Task 2

Task 2.1 : Linear regression

LinearRegression().fit() :-

Linear Regression is a ML algorithm that attempts to model the relationship between two variables by fitting a linear equation to observed data. Linear regression attempts to draw a straight line that will best minimise the residual sum of squares between the observed responses in the dataset, and the responses predicted by the linear approximation . it _fits a linear model to minimise the residual sum of squares between the observed targets in the dataset, and the targets predicted by the linear approximation.

LinearRegression().fit() is a function of the Class sklearn.linear_model.LinearRegression.

in short A linear regression model represents a linear equation which has a certain set of inputs (x) and predicted outputs (y) for those inputs.

.fit() function finds the optimal values of the intercepts where the arguments are the existing inputs and outputs. An estimator is fit onto the model to predict the outputs of the unseen inputs. A classifier is an estimator instance which is fitted to the model and its purpose is to learn from the model. The .fit() function fits any instance of the Linear Regression.

y = B0 + B1(x)

Here, B0 is the Regression coefficient/intercept

x = independent variable

The intercept is the expected mean value of Y when all X (predictor of the regression equation) equals zero.

Task 2.2 : Gradient descent

Gradient descent :-

Gradient descent is an iterative optimisation algorithm used to find the minimum of a cost function, also known as the objective function, by adjusting the values of the model's coefficients. In the context of linear regression with one independent variable and one dependent variable, the goal of gradient descent is to find the coefficient of the independent variable that minimises the sum of the squared differences between the predicted and actual values of the dependent variable.

The basic idea behind gradient descent is to update the coefficients in the opposite direction of the gradient of the cost function with respect to the coefficients. The gradient of the cost function is simply the derivative of the cost function with respect to each coefficient.

Here are the steps of gradient descent for linear regression with one independent variable and one dependent variable:

1. Initialize the coefficients to some arbitrary values.
2. Calculate the predicted values of the dependent variable using the current values of the coefficients and the independent variable.
3. Calculate the cost function, which is the mean squared error (MSE) between the predicted and actual values of the dependent variable.
4. Calculate the gradient of the cost function with respect to each coefficient.
5. Update the coefficients by subtracting the gradient of the cost function multiplied by a learning rate from the current values of the coefficients.
6. Repeat steps 2 to 5 until the cost function is minimized or a maximum number of iterations is reached.

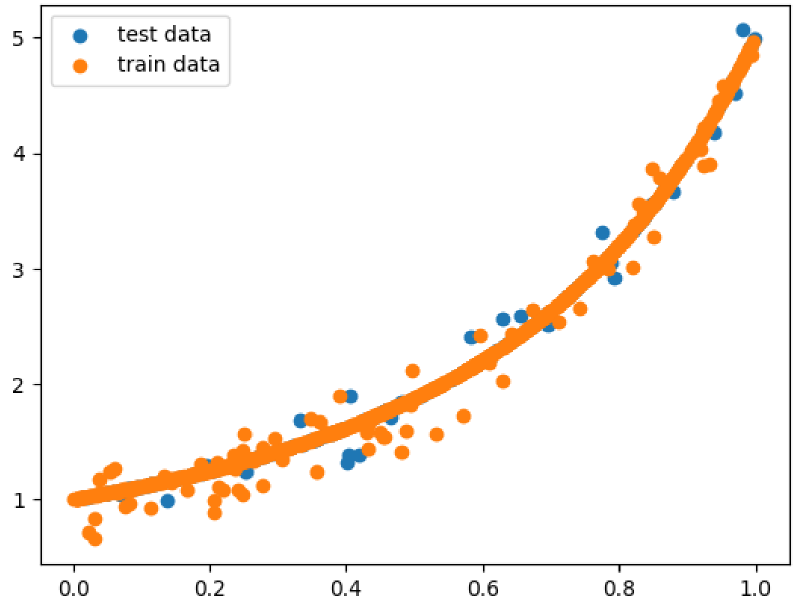
Task 2.3.2 : Task

Plot of bias and variance :

```
#libraries imported
import pickle
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from tabulate import tabulate
```

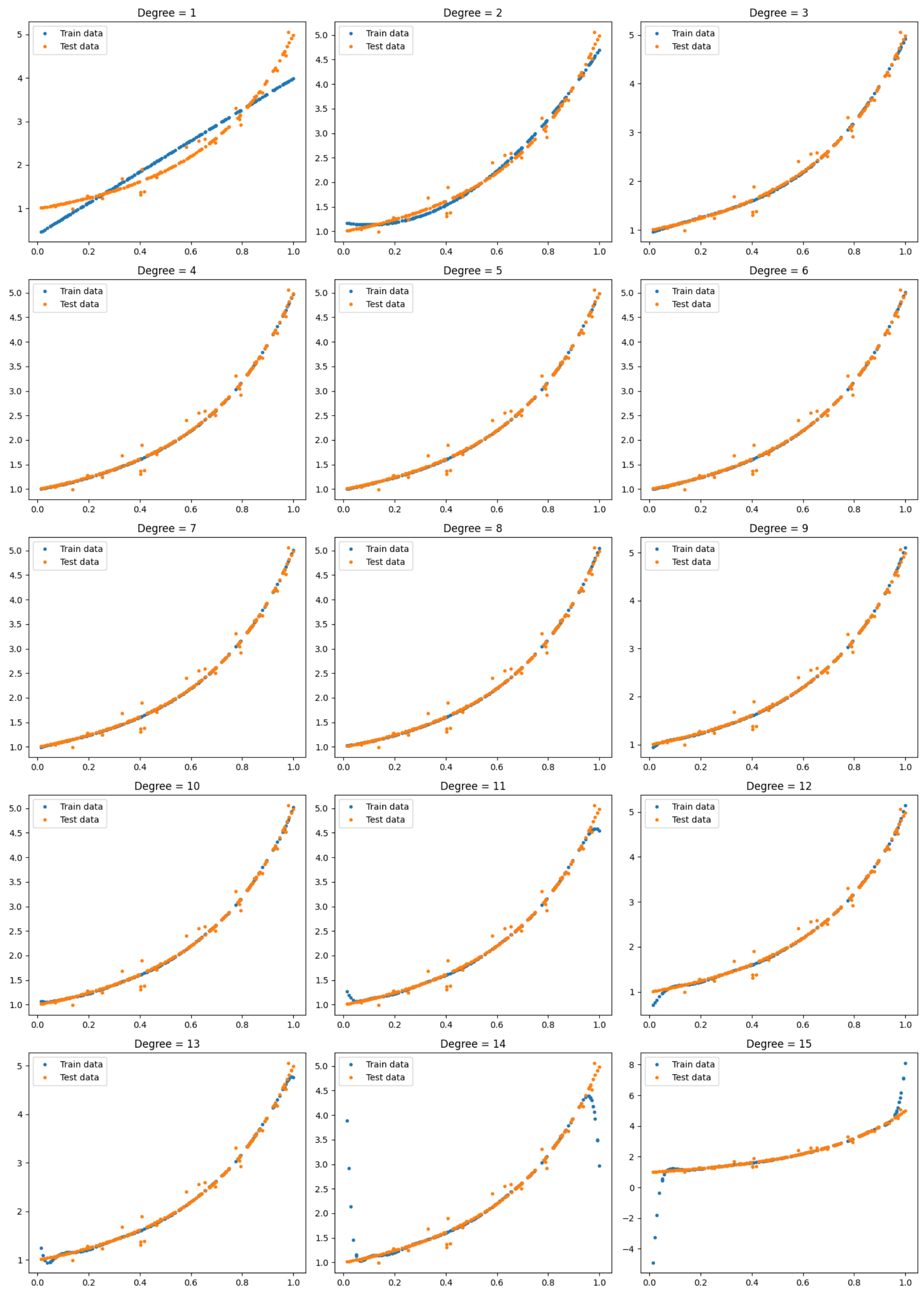
```
#opened both the files and split the train data in 20 equal datasets
num_splits = 20
```

```
total_degree = 15

with open('train_dataset.pickle' , 'rb') as td :
    train_data = pickle.load(td)


with open ('test_dataset.pickle' , 'rb') as td :
    test_data = pickle.load(td)

np.random.shuffle(train_data)
train_data_splits = np.array_split(train_data, num_splits)
```



```
# calculating bias and variance for each degree after converting the array into 2D array
for dist in range(0 , 20):
        x_deg_train = x_train[dist]
        y_deg_train = y_train[dist]
        model = LinearRegression().fit(PolynomialFeatures(degree=deg).fit_transform(x_deg_train), y_deg_train)
        pred = model.predict(PolynomialFeatures(degree=deg).fit_transform(x_test))
        pred_dist.append(pred)
k = np.mean(abs(np.mean(pred_dist , axis=0) - y_test))
bias.append(k)
variance.append(np.mean(np.var(pred_dist , axis=0)))
```

First I am plotting all the 15 modals corresponding to the polynomial . Here I printed the graph between test data and train data . As we observe the graph for the first two modal is **underfitting** because they have very low variance and the last two graphs are **overfitting** as they have very high variance. The graph of degree 4 and 5 seems to be best fitting because the test data is completely overlapping the train data and hence can be categorised as a **bestfitting**. rest all lies between overfitting and underfitting .


Degree = 1
Degree = 2
Degree = 3
Degree = 4
Degree = 5
Degree = 6
Degree = 7
Degree = 8
Degree = 9
Degree = 10
Degree = 11
Degree = 12
Degree = 13
Degree = 14
Degree = 15
Train data
Test data

Bias

Bias represents the difference between the true values of the target variable and the predicted values of the model .

Variance

Variance represents the variability in the predicted values of the model due to changes in the training data,

The value of bias and variance we got in any random set is this . We are plotting the graph till the degree 10 as stated . Now observing this graph we see that

Degree	Bias	Variance
1	0.2691677	0.0084612
2	0.0859972	0.0018732
3	0.0332592	0.0005684
4	0.0253346	0.0006752
5	0.0262888	0.0008730
6	0.0262826	0.0013679
7	0.0272958	0.0033248
8	0.0304686	0.0114704
9	0.0322122	0.0339811
10	0.0338691	0.0587522
11	0.0469649	0.1306532
12	0.0349644	0.2922423
13	0.0524450	0.6927972
14	0.2233833	6.6920933
15	0.1292089	12.9850574

Our Bias - Variance values are above and the graph is plotted below. Now , we have the plots for the bias and variance of each model, we can analyse and compare their performance.

As we can see that initially the bias has a very high value but as the polynomial power increases its value start decreasing till degree 05 and after that the slope is up down and variance also have a decreasing curve till degree 03 and after that the value start increasing .

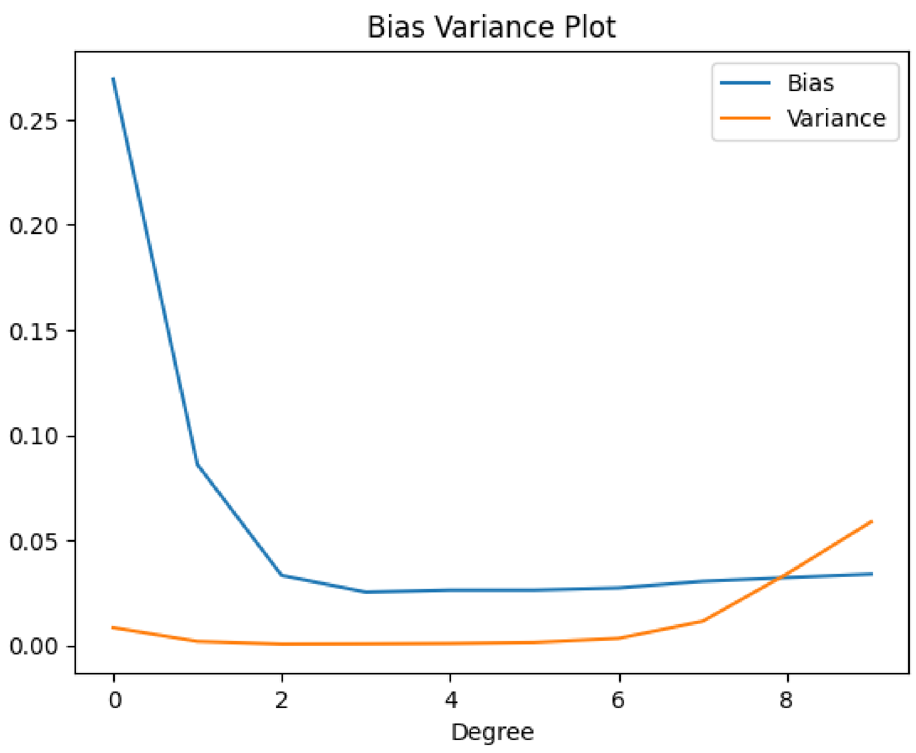
So as we know for

Overfitting : high variance and low bias .

Underfitting : low variance and high bias.

bestfitting : both should be moderate.

And from the above table we can just predict that the degree 4,5,6 are coming in the bestfit category.



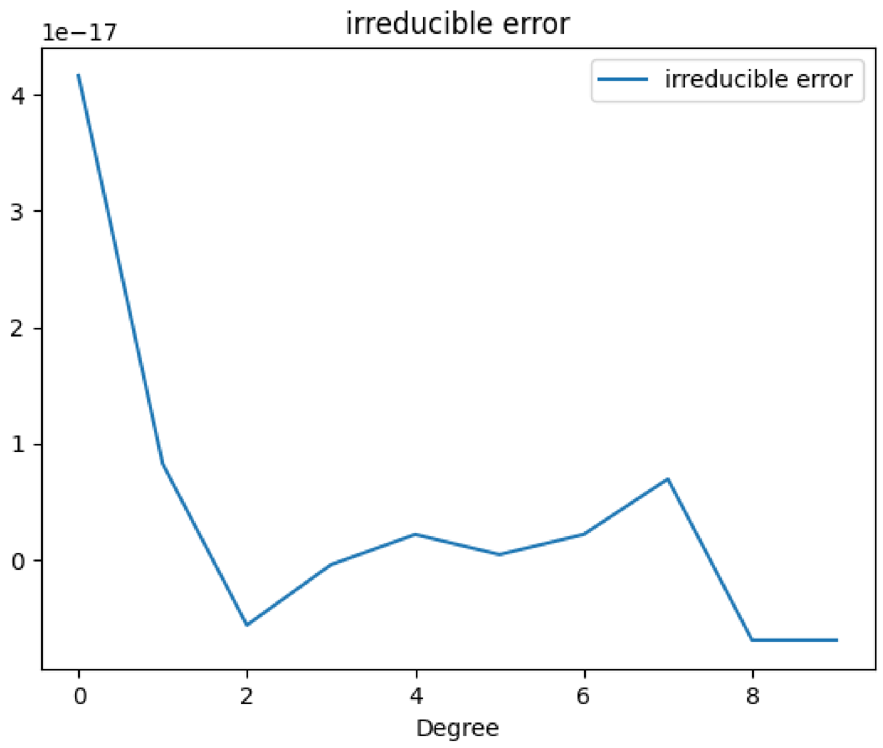
2.4 Task 4 : Calculating the irreducible error

```
# Calculate the irreducible error for each of the models

irreducible_error = np.array(error) - np.array(bias2) - np.array(variance)
print("Irreducible errors :" ,  irreducible_error)
```

Irreducible error is a type of error that cannot be reduced or eliminated by improving the model. It represents the inherent noise or randomness in the data that cannot be explained by the model. This type of error is typically caused by factors outside of the model, such as measurement error or natural variability in the data.

Irreducible Error = MSE - (Bias^2 + Variance)

Degree	Irreducible error
1	4.16333634e-17
2	8.23993651e-18
3	-5.63785130e-18
4	4.33680869e-19
5	2.16840434e-18
6	4.33680869e-19
7	2.16840434e-18
8	6.93889390e-18
9	6.93889390e-18
10	6.93889390e-18
11	0.00000000e+00
12	-5.55111512e-17
13	1.11022302e-16
14	8.88178420e-16

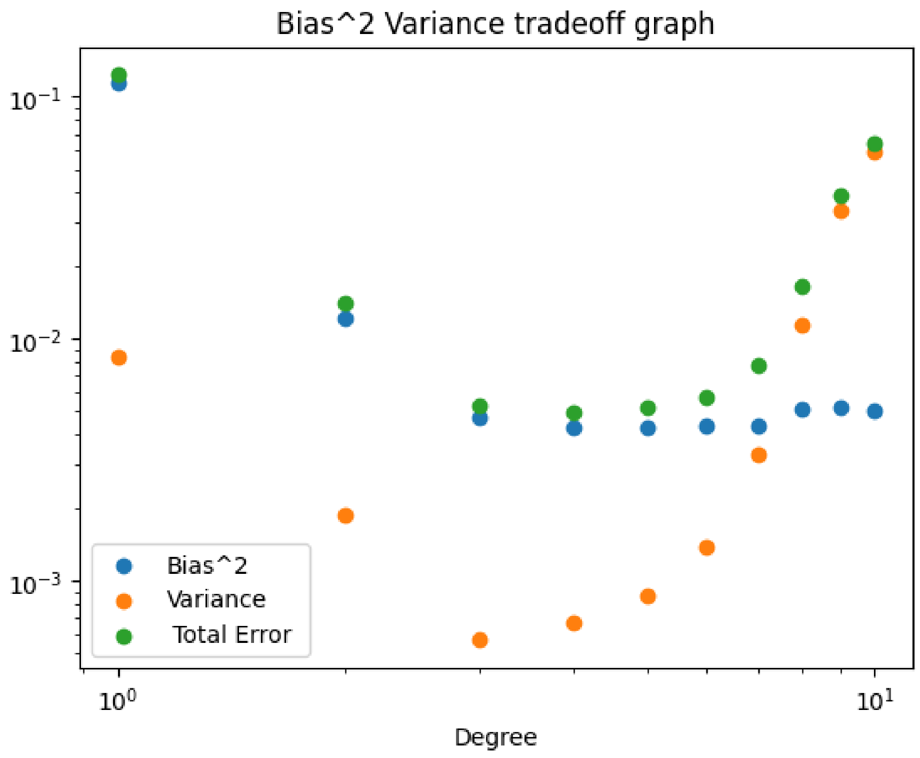


The error has the range of - 8.881784197001252e-16 to 4.163336342344337e-17.

So from this we can see that the error is changing in the range of 10^-17 . which is very close to zero . So the irreducible error is actually independent of the modal and it only depend on the data . and because of this the value of irreducible is not changing for different modal (changing very small value near to 0).

2.5 Task 5 : Plotting Bias^2 vs variance

```
plt.scatter(range(1,11),bias2[:10], label='Bias^2')
plt.scatter(range(1,11),variance[:10], label='Variance')
plt.scatter(range(1,11),error[:10], label=' Total Error ')
plt.xscale('log')
plt.yscale('log')
plt.xlabel('Degree')
plt.title('Bias^2 Variance tradeoff graph')
plt.legend()
plt.show()
```



In the above plot we have printed the data till polynomial for 10 degree .

Degree	Total Error
1	0.12294901834128924
2	0.013988293705386401
3	0.005310359150451271
4	0.004944353754885029
5	0.005181700625973363
6	0.005708505739573364
7	0.007708240042548874
8	0.016578919674093803
9	0.03919582304527014
10	0.06379186678383489
11	0.14381828882023068
12	0.29999036035920706
13	0.7098895198777245
14	7.333720948455142
15	13.140877189206185

1. **Underfitting :-** Underfitting can occur when the model is too constrained, or when the number of parameters or features is too small. This can cause the model to oversimplify the data, and ignore or miss important patterns or relationships.In the Bias-Variance tradeoff graph, overfitting occurs when the variance is low and the bias is high.

   When the degree of complexity is low (1-2), the model is too simple and cannot capture the underlying patterns in the data. This results in high bias and low variance, which indicates underfitting.

2. **Overfitting :-** Overfitting occurs when a machine learning model becomes too complex and starts to fit the training data too closely, to the point that it captures noise and irrelevant patterns in the data, resulting in poor generalization to new data.In the Bias-Variance tradeoff graph, overfitting occurs when the variance is high and the bias is low.

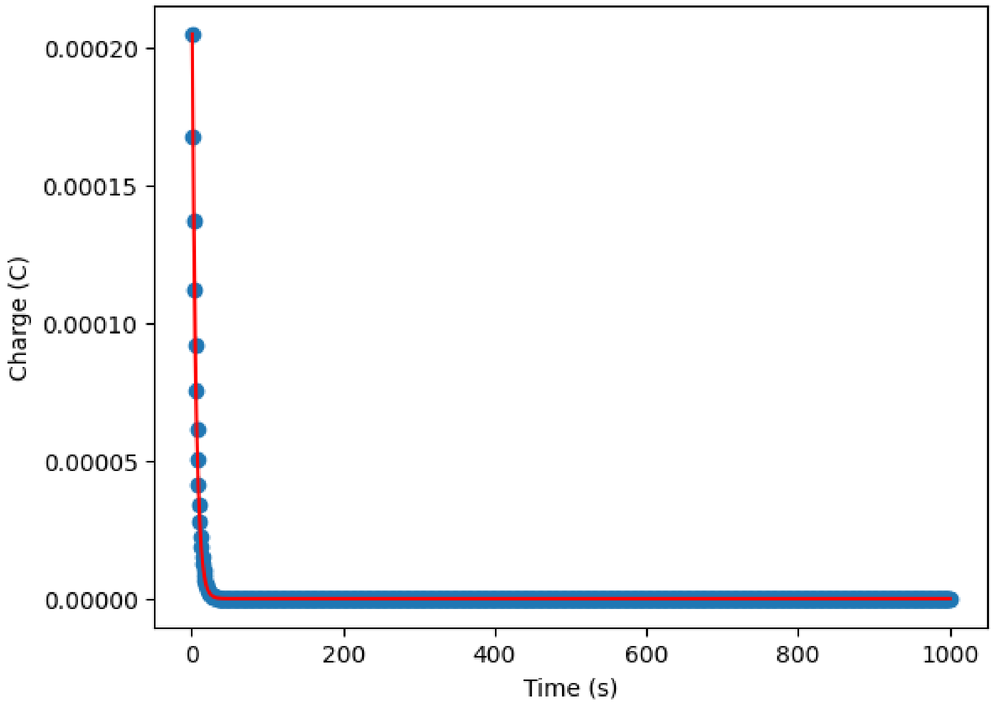
   As the degree of complexity increases (9-15), the model becomes more complex and can fit the data more closely. However, this results in high variance and low bias, which indicates overfitting.

3. **Optimal complexity / Best fit** : the optimal complexity of a model depends on the size and complexity of the dataset, the number of features, and the level of noise and variability in the data. As The Bias-Variance tradeoff plot shows , It is the point where the total error of the Total Error is at its minimum. And By this if we see the graph we get the point in the **degree 4** in the minimum and hence that is the best fit modal .

BONUS PART

Getting the value of capicitor and resistance as

Estimated capacitance (C) = [5.e-05] F
Estimated resistance (R) = [100000.00000001] Ω



Shivam Tiwari

2021101127

-