MDL ASSIGNMENT-1 REPORT

Team 95

Mihir Bani, 2019113003

Amul Agrawal, 2019101113

Task 1 : LinearRegression().fit()

LinearRegression is a model that fits a one degree function onto the given data.

To fit the model we call LinearRegression().fit($\mathbf{X}, \mathbf{y}$) where $\mathbf{X}$ is the matrix whose rows are the input attributes and $\mathbf{y}$ is the target vector.

the ith row of $\mathbf{X}$ are the attributes whose target value is ith element of $\mathbf{y}$. And $\hat{\mathbf{y}}$ is the prediction of the model.

The model finds the perfect linear fit by minimizing the mean squared error by performing gradient descent. The error which has to be minimized is defined as:

Algebraically

$$\mathbf{E} = \frac{1}{\mathbf{N}} \sum_{i=1}^{\mathbf{N}} (\mathbf{y}_i - \hat{\mathbf{y}}_i)^2$$

In matrix notation,

$$\mathbf{E} = \frac{(\mathbf{y} - (\mathbf{W}^T\mathbf{X} + \mathbf{b}))^T(\mathbf{y} - (\mathbf{W}^T\mathbf{X} + \mathbf{b}))}{\mathbf{N}}$$

$\mathbf{N}$ is number of data points.

$\mathbf{W}$ is the current set of weights. It is the matrix that contains the coefficients of the polynomial (the training data).

$\mathbf{E}$ is the mean squared error function. Can also be called the **Cost Function**.

$\mathbf{b}$ is the bias(different from the Bias which we calculate later). (coefficient of x^0)

$\therefore \mathbf{W^TX} + \mathbf{b}$ is matrix for **predicted_y**

Linear Regression minimizes this error using **gradient descent.**

WHAT IS GRADIENT DESCENT ?

> " A gradient measures how much the output of a function changes if you change the inputs a little bit." - Lex Fridman (MIT)

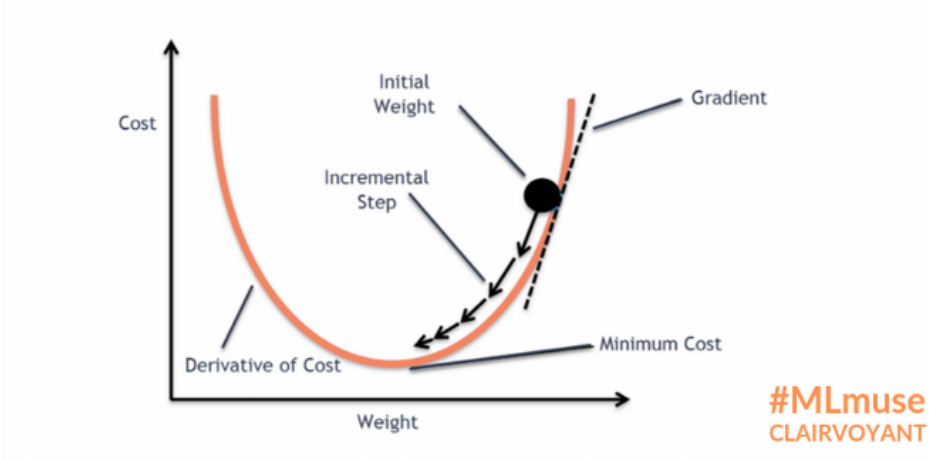
A gradient simply measures the change in all weights with regard to the change in error. We can also think of a gradient as the slope of a function. The higher the gradient, the steeper the slope and the faster a model can learn. But if the slope is zero, the model stops learning. In mathematical terms, a gradient is a partial derivative with respect to its inputs.

HOW GRADIENT DESCENT WORKS ?

Thinking of gradient descent as hiking down to the bottom of a valley. The equation below describes what gradient descent does: b is the next position of our climber, while a represents the current position. The minus sign refers to the minimization part of gradient descent. The γ in the middle is learning rate and the gradient term ($\nabla f(a)$) is the gradient of the function $f(a)$ which tells the direction of the steepest descent.

$$\mathbf{b} = \mathbf{a} - \gamma \nabla \mathbf{f(a)}$$

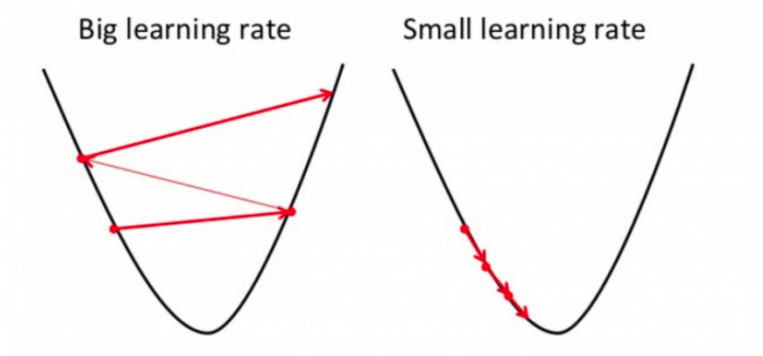
So this formula basically tells us the next position we need to go, which is the direction of the steepest descent. And this way we can minimize the Error function



IMPORTANCE OF THE LEARNING RATE

How big the steps are gradient descent takes into the direction of the local minimum are determined by the learning rate, which figures out how fast or slow we will move towards the optimal weights.

For gradient descent to reach the local minimum we must set the learning rate to an appropriate value, which is neither too low nor too high. This is important because if the steps it takes are too big, it may not reach the local minimum because it bounces back and forth between the convex function of gradient descent (see left image below). If we set the learning rate to a very small value, gradient descent will eventually reach the local minimum but that may take a while (see the right image).



So, the learning rate should never be too high or too low for this reason. You can check if you're learning rate is doing well by plotting it on a graph.

Task 2 : Finding the Bias and Variance

2.1 Loading

We first loaded the data from pkl file into the notebook and stored it in Numpy Arrays.

```
# loading from the train.pkl file
file = open('./train.pkl','rb')
train_data = pickle.load(file)
file.close()

# loading from the test.pkl file
file = open('./test.pkl','rb')
test_data = pickle.load(file)
file.close()
```

Then in the training set, divided it into 10 different parts. Also stored the X and Y values separately in Numpy array

```
# dividing into 10 training sets and storing in train_X and train_Y as numpy array
train_X = []
train_Y = []
len_train = int(train_data.shape[0]/10)
for i in range(10):
    train_X.append(train_data[i*len_train:(i+1)*len_train,0])
    train_Y.append(train_data[i*len_train:(i+1)*len_train,1])
train_X = np.array(train_X)
train_Y = np.array(train_Y)
```

2.2 Training and Testing

We train each model (i.e. polynomial of a particular degree) 20 times, once on each partition. The performance metrics of the model are computed by calculating its bias and variance on each data point (in the test set) across the 20 realizations of the model. The mean bias (or variance) of the model is then the mean of the bias (or variance) values calculated for each data point (in the test set).

The outer loop iterates over the polynomials from degree 1 to 20, and the inner loop iterates over the 10 different partitions of training set. A model is trained on one of the partitions of the training set, and then tested on the test set. The predictions of the model are then stored. After all the 10 parts are trained and tested, we take the mean of predictions and find the bias for a

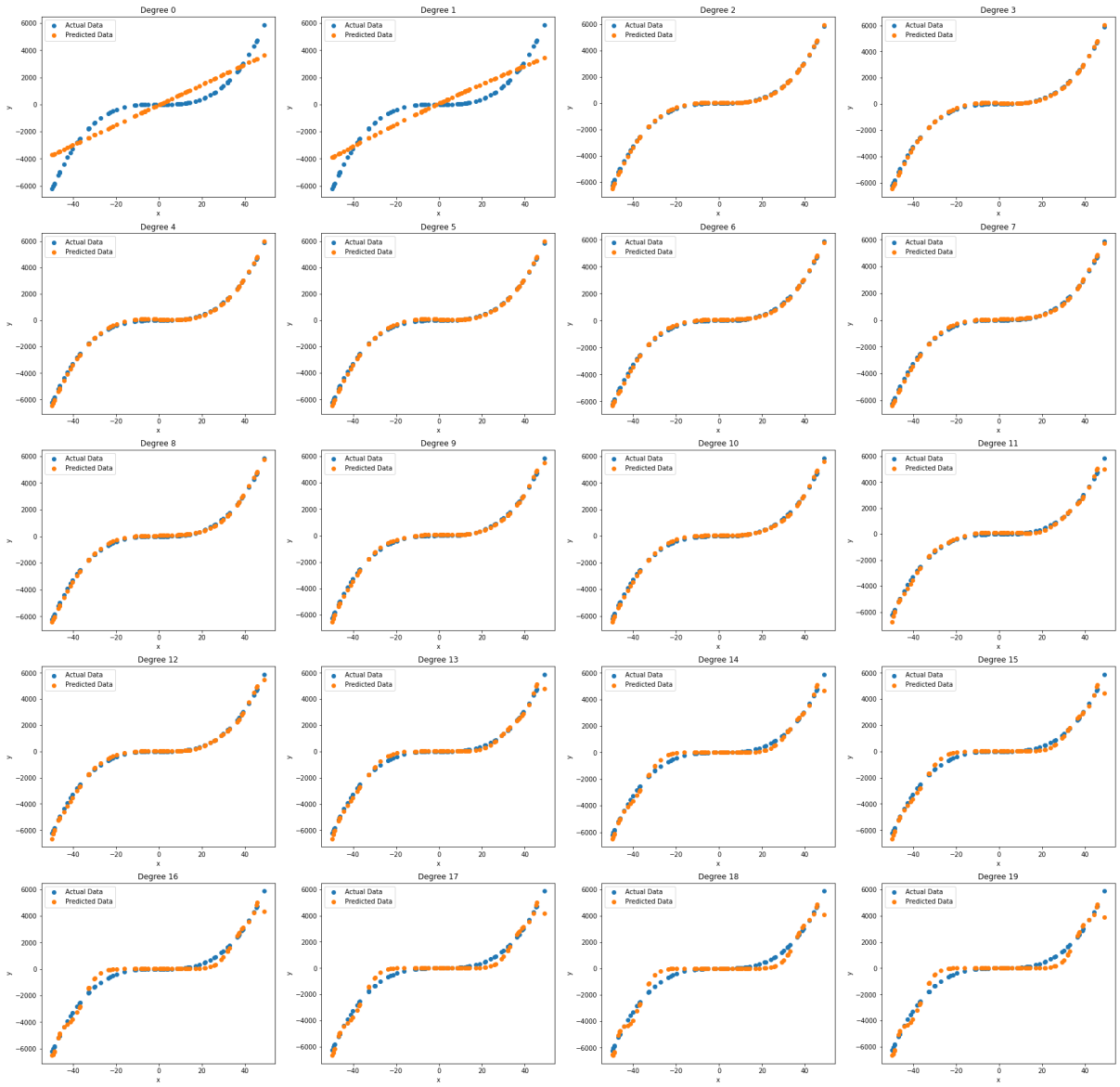
single test point and then take the mean over all test points, similarly variance is calculated.

This process is repeated for each of the 20 different degrees (models).

```
degrees = np.arange(1,21)

# loop over every polynomial from degree 1 to 20
for degree in degrees:
    # list for each degree
    predicted = []
    error = []

    # setting the polynomial of degree = degree
    poly = PolynomialFeatures(degree = degree)
    deg_test_X = poly.fit_transform(test_X.reshape(-1,1))

    # looping over all the 10 training sets, training the model and finding predictions
    for i in range(10):
        # setting the training data into polynomial type
        curr_train_X = poly.fit_transform(train_X[i].reshape(-1,1))

        # getting the regressor(training model)
        regressor = linear_model.LinearRegression()
        regressor.fit(curr_train_X,train_Y[i])

        # finding the predictions from the model
        deg_prediction = regressor.predict(deg_test_X)

        # storing the values
        predicted.append(deg_prediction)
        error.append(mean_squared_error(test_Y,deg_prediction))


    # calculating the required quantities
    error = np.array(error)
    predicted = np.array(predicted).T
    mean_pred = np.mean(predicted,1)

    # finding the bias of a particular test point from all models and taking the absolute value
    bias_diff = np.abs(mean_pred - test_Y)

    # taking the mean across all test points
    mean_bias = np.mean(bias_diff)

    # taking the mean of bias_sq across all test points
    mean_bias_sq = np.mean(bias_diff**2)

    # updating the final lists
    mse.append(np.mean(error))
    bias_sq.append(mean_bias_sq)
    bias.append(mean_bias)
    variance.append(np.mean((predicted.T - mean_pred.T)**2))
    all_predictions.append(mean_pred)
```

Plots:

These plots show how the model trained (averaged over the 10 partitions) perform on the given Test dataset



We can see that the best fit is in the case of degrees ranging from 3 to 10. The test data resembles a cubic curve.

Then for degree greater than 10, the model again gives incorrect predictions because of **overfitting**.

For the model with degree 1 or 2 the model performs very poorly, and this shows **underfitting**.

2.3 Tabulation of required quantities

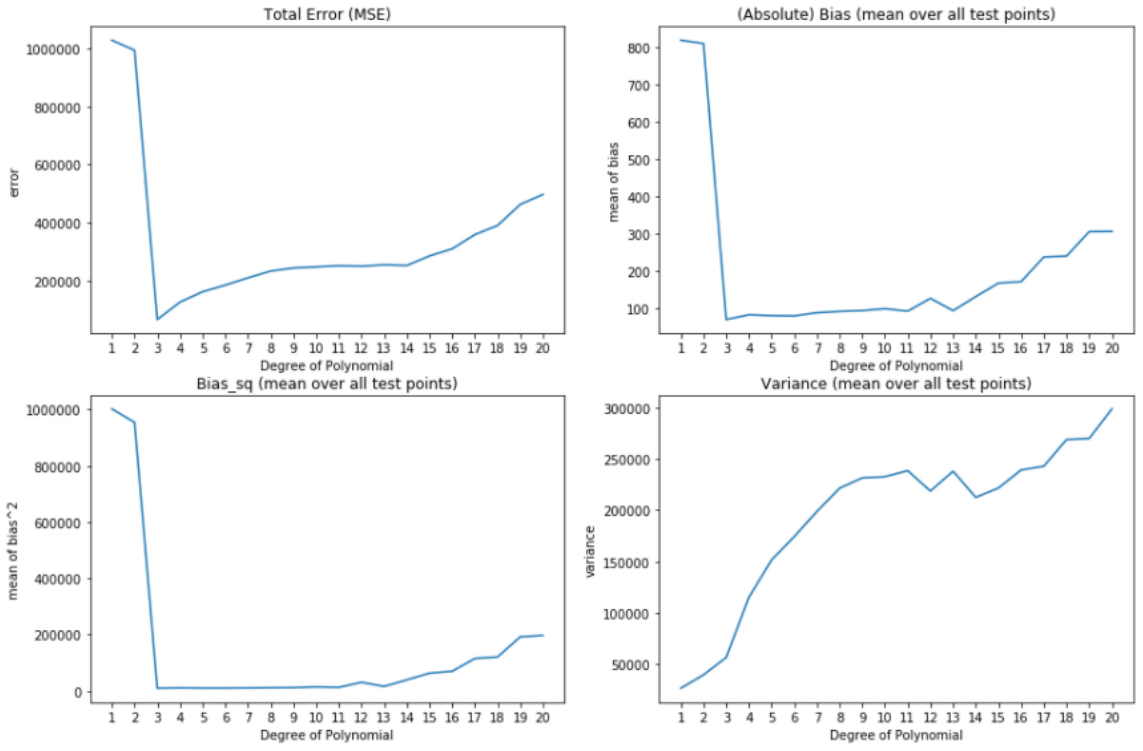
Total Error: calculated by taking mean over error of all test points.

Bias: Bias is found for a single point, then took the mean of absolute value of the bias for all test points.

Squared Bias: Bias is found for a single point, then took the mean of square of the bias for all test points.

Variance: Variance is found for a single point, then took the mean of the variance for all test points.

Degree	Total Error	Bias (mean of |bias|)	Squared Bias (mean of bias^2)	Variance
1	1.02768e+06	819.717	1.00168e+06	25999.1
2	992942	810.763	953836	39105.8
3	65629.2	68.5109	9533.34	56095.9
4	125496	81.3397	10588.3	114907
5	161386	78.9584	9952.45	151434
6	184226	78.3648	9999.2	174227
7	209276	86.9159	10425.8	198850
8	232551	90.3294	10998.8	221552
9	243260	92.6333	11687.2	231573
10	246925	97.5357	14150.9	232774
11	251072	91.0452	12429.8	238642
12	249380	125.502	30664.9	218716
13	254194	92.3364	16081.9	238112
14	251776	130.174	39231	212545
15	284644	166.459	62928.3	221715
16	309180	170.418	69821.8	239358
17	358233	236.715	115240	242993
18	389631	239.125	120581	269051
19	462082	304.868	191980	270102
20	496276	305.438	197268	299008



Task 3: Irreducible Error

It is the noise that is present in the data on which we are trying to train, and which can not be corrected however we train our ML model.

This noise depends on the way the data was collected, involving many factors like the precision of the device which collects the data or etc. In a way, we can say that the training data we have is not completely identifying the outcomes. The equation for Y can be given as below. Where ϵ denotes the irreducible error. This error can have different distributions, mostly Normal Distribution with parameters σ^2.

$$Y = f(X) + \epsilon$$

For our case, It is given by the below equation, where σ^2 denotes the irreducible error

$$\mathrm{MSE} = \mathrm{E}_x \left\{ \mathrm{Bias}_D \left[\hat{f}(x; D)\right]^2 + \mathrm{Var}_D \left[\hat{f}(x; D)\right] \right\} + \sigma^2.$$

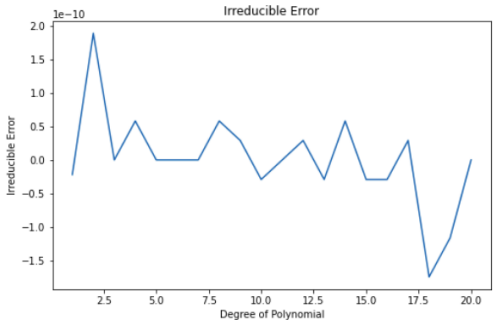
$$\sigma^2 = MSE - Bias^2 - Variance$$

```
irr_error = mse - bias_sq - variance
```

We take the mean over all 10 partitions and all the test points.

Tabulation of Irreducible Error along with Total Error and Squared Bias, Variance

Degree	Total Error	Squared Bias (mean of bias^2)	Variance	Irreducible Error
1	1.02768e+06	1.00168e+06	25999.1	-2.18279e-11
2	992942	953836	39105.8	1.89175e-10
3	65629.2	9533.34	56095.9	0
4	125496	10588.3	114907	5.82077e-11
5	161386	9952.45	151434	0
6	184226	9999.2	174227	0
7	209275	10425.9	198850	0
8	232550	10996.4	221553	5.82077e-11
9	244164	11559.5	232605	2.91038e-11
10	247531	14476.8	233054	-2.91038e-11
11	251122	12416.4	238706	0
12	249565	30551.3	219014	2.91038e-11
13	250340	16173.9	234166	-2.91038e-11
14	251776	39231.1	212545	5.82077e-11
15	284643	62928.4	221715	-2.91038e-11
16	309179	69821.5	239358	-2.91038e-11
17	358234	115239	242994	2.91038e-11
18	389630	120580	269050	-1.74623e-10
19	462081	191979	270102	-1.16415e-10
20	496312	197285	299027	0

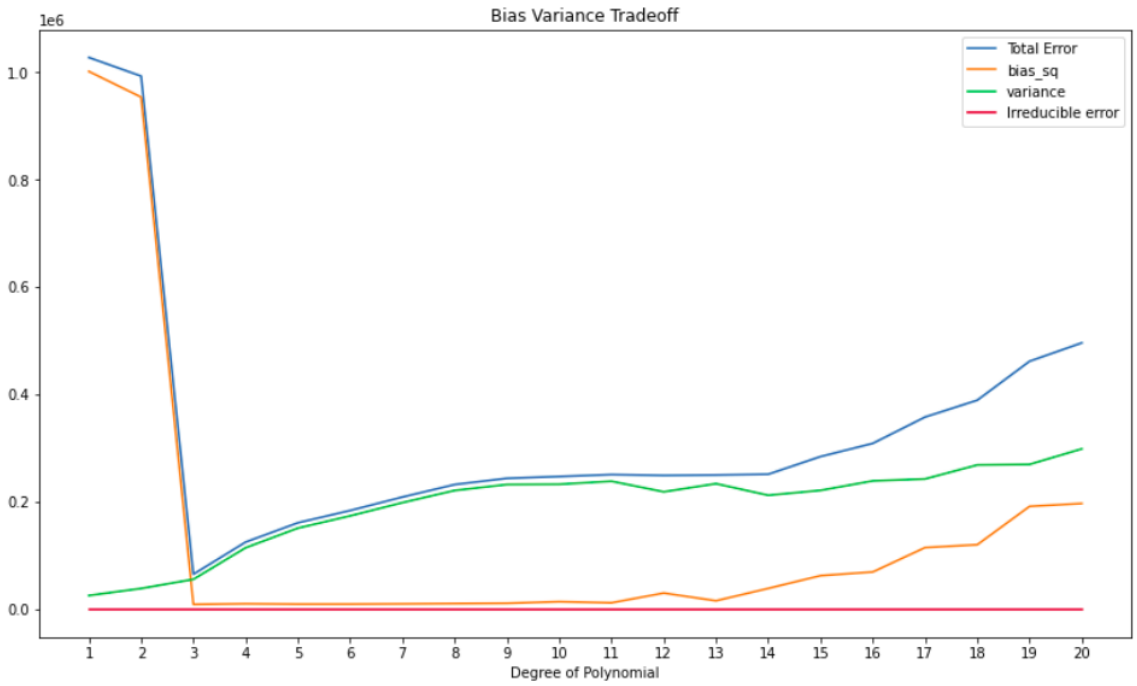


The value of irreducible error is very close to zero. The negative values are of the order 1e-11 come because of floating-point precision error.

The noise is a internal part of the training data and remains independent to the performance of the machine learning model.

So the noise is nearly zero in the dataset given.

Task 4 : Bias Variance Tradeoff



We can observe that the lowest Total Error happens when degree = 3.

Bias Square: As the degree increases, the bias first decreases because the model can very well predict the test data, but for degree > 11, it starts to increase. It is the case because the polynomial of degrees 3 to 10 are much closer to actual test data. The model with very high polynomial degree did seem to have a little more bias than lower degrees. This is because those models have overfitted to the training data so much (like memorized the values), so its not able to perform well on the test data (which is different from training data). Thus their bias on the test data starts to increase. Their bias on the training data will negligible but on the test data, it is not the case.

Variance: It increases with degree as the function becomes enough complex to fit the data set. Whereas Variance is very low for lower degrees and then starts to increase with the degrees. As the degree increases the model captures the extreme points of the data, and thus the predictions are more varied around the mean. This is the definition of variance, so the variance increases with higher complexity

Total Error: At lower complexity, the TE is very high due to Bias being very big, then it decreases and reaches a minimum at degree=3, after that, it starts to increase again as now the variance is increasing a lot even when the bias is very low. The slope of TE increases even more after degree>14, when even the bias starts to rise.

Irreducible error: It was expected to stay the same as the data is very structured, there is almost no irreducible error.

Overfitting: For degrees higher than 6, the Variance starts to shoot, while the Bias is very low. This shows that the model is picking even the small irregularities in the data. This is a case of Overfitting. For even higher degree like in the case of 18 to 20, the Variance and Bias are both very high. This tells that the model at this complexity has become just too sensitive that its not even able to keep the bias low on the test data. The model has nearly memorized every point on the training data, so its not able to perform well on the test data (which is different from training data).

Underfitting: For degree less than 3, the Variance is pretty low, but the Bias is very huge. This shows that the model is not able to predict the values at all. It has learned very little till now. This is the case of underfitting