CS7.301 **13th Feb, 2020**

MDL Assignment-1

Vikrant Dewangan, Roll No.- 2018111024
Mohammad Nomaan Qureshi, Roll No.- 2018111027

Contents

1 Question 1

1.1 Problem Statement

Given a data set of 5000 data points, train a linear classifier and calculate the bias and variance of the trained model for different functions upto degree 9, always ensuring a 90:10 split between training and testing data.

1.2 Algorithm Implementation

The following steps are followed to obtain the final set of values -

1. First the given data in pickle format is loaded and uniformly shuffled.

```
#================================================================#
# Loading the Data file #
#================================================================#
file = open('Q1_data/data.pkl', 'rb')
data = pickle.load(file)
print("Loaded Data Successfully!!!")
#================================================================#


#================================================================#
# Resampling Data #
# First randomly Shuffle data into 10 equal parts and
    then divide them into 10
equal parts.
#================================================================#
np.random.shuffle(data)
```

2. Preparing Train and Test data set

```
test = data[:len(data)//10]
train = data[len(data)//10:]

random.shuffle(train)
train_y = np.asarray([[i[1]] for i in train])
test_y = np.asarray([[i[1]] for i in test])

train_x = np.asarray([np.asarray([i[0]**j for j in
    range(1, 10)]) for i in
train])
test_x = np.asarray([np.asarray([i[0]**j for j in
    range(1, 10)]) for i in
```

```
test])
```

As specified in the problem statement, the first 10 parts of data are prepared as training and the last part as test data set. After which, the data is structured in a proper format using numpy arrays.

3. Training 10 different models using 10 different functions

```
coef = []        # Coefficient Obtained by Regression #
intercept = []   # Intercept Obtained by Regression #
err, var, bas = [], [], []
for j in range(1, 10):
prediction = np.zeros((len(test_x), 1))
pred_array = []
variance = 0
mse = []
for i in range(10) :
if j == 1 :
fit_intercept = False
else :
fit_intercept = True
reg = LinearRegression(fit_intercept=fit_intercept,
    normalize =
True).fit(train_x[i*len(train_x)//10:(i+1)*len(train_x)//10,
    :j],
train_y[i*len(train_x)//10:(i+1)*len(train_x)//10])
coef.append(reg.coef_)
pred = reg.predict(test_x[:,:j])
pred_array.append(pred)
prediction = prediction + pred
```

The function **LinearRegression** is used to fit the model and the corresponding part of the [train_x,train_y] array is given as parameters to the function. The array *coef* signifies the coefficients obtained in the linear regression fit every time it is called. Thus, by appending it to the array, each time, a function of order j is used to train the model.

4. Calculating mean variance and bias for each function

```
pred_array = np.asarray(pred_array)
prediction = prediction/10
diff = pred_array - prediction
bias = np.mean((prediction - test_y))**2
```

```
variance = np.mean(diff**2)
err.append(np.mean(mse))
var.append(variance)
bas.append(bias)
```

Here are what all of the variables signify -

- *pred* represents the predicted value $\hat{f}(x)$
- *prediction* is the sum of all predictions done by the model. Used to calculate $E\big[\hat{f}(x)\big]$

$$E\big[\hat{f}(x)\big] = \frac{\sum_{j=1}^{10} \hat{f}(x)}{10}$$

- *bias* is calculated as mean of *prediction* and *test_y* arrays. In other words,

$$\begin{aligned} Bias^2 &= \left(E\left[E\big[\hat{f}(x)\big] - f(x) \right] \right)^2 \\ &= \left(E\big[\hat{f}(x)\big] - f(x) \right)^2 \end{aligned}$$

- *variance* is calculated as the mean of prediction values $\hat{f}(x)$ and average prediction value $E\big[\hat{f}(x)\big]$

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

- *mse* is calculated by the function *mean_squared_error* function and can be given by the formula

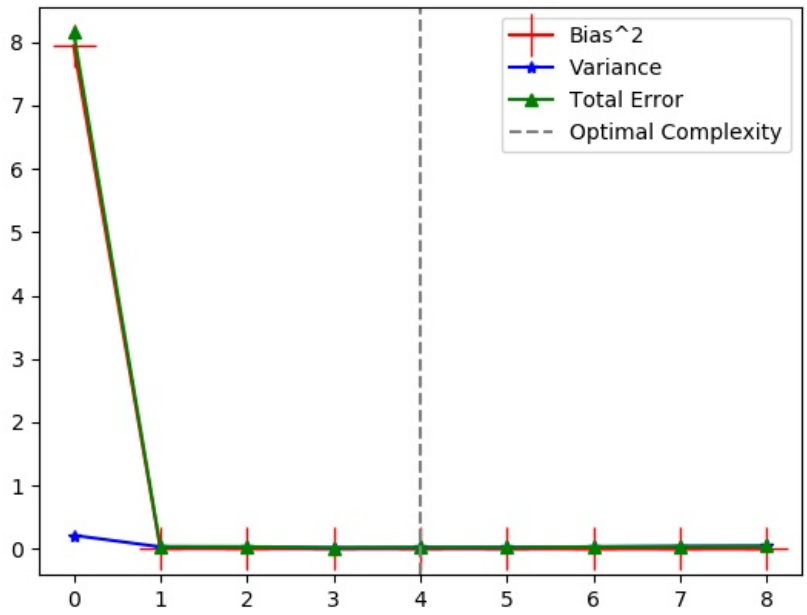
$$M.S.E = E\big[\big(\hat{f}(x) - f(x)\big)^2\big] + \sigma^2$$

  where σ^2 is the variance of the noise with mean(μ) = 0

Thus, for each polynomial, mean bias and variance are calculated and tabulated.

1.3 Results

The following is the result of the data shuffling.



In tabular version -

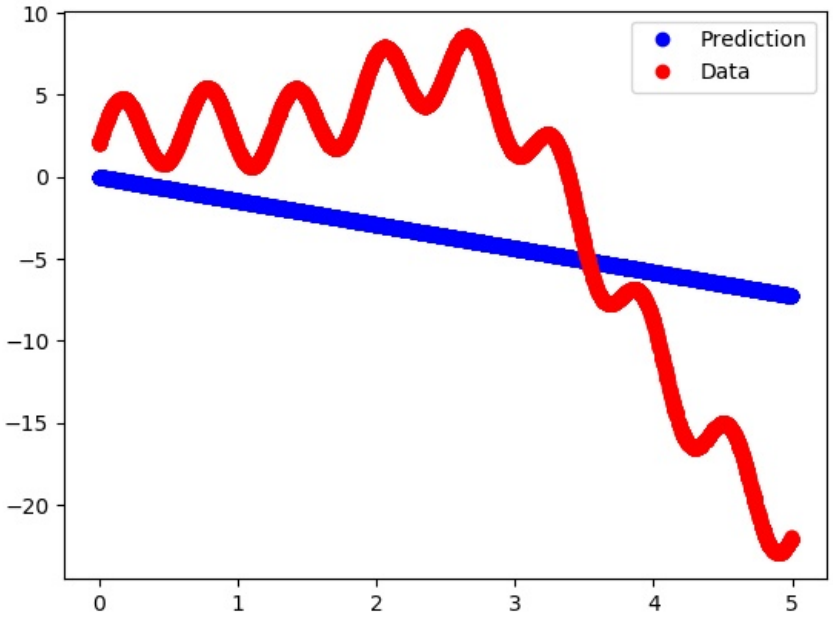
Model Complexity	**Bias**	**Variance**	**Total Error**
1	5.46952589256	0.205035064745	5.67456095731
2	0.00946157459989	0.0311687828902	0.0406303574901
3	0.00132121430958	0.0637597311871	0.0650809454967
4	0.00668651415434	0.0335039036798	0.0401904178342
5	0.00335006723856	0.0362704492444	0.039620516483
6	0.000368344916076	0.0418653802428	0.0422337251589
7	0.000666255490301	0.0538722377555	0.0545384932458
8	0.000572732337839	0.0649197944417	0.0654925267795
9	0.000691939933171	0.0624076730169	0.06309961295

Note that the *pred_array* is an array containing the values and thus the bias and variance are already calculated in a ***vectorized*** manner on numpy arrays.

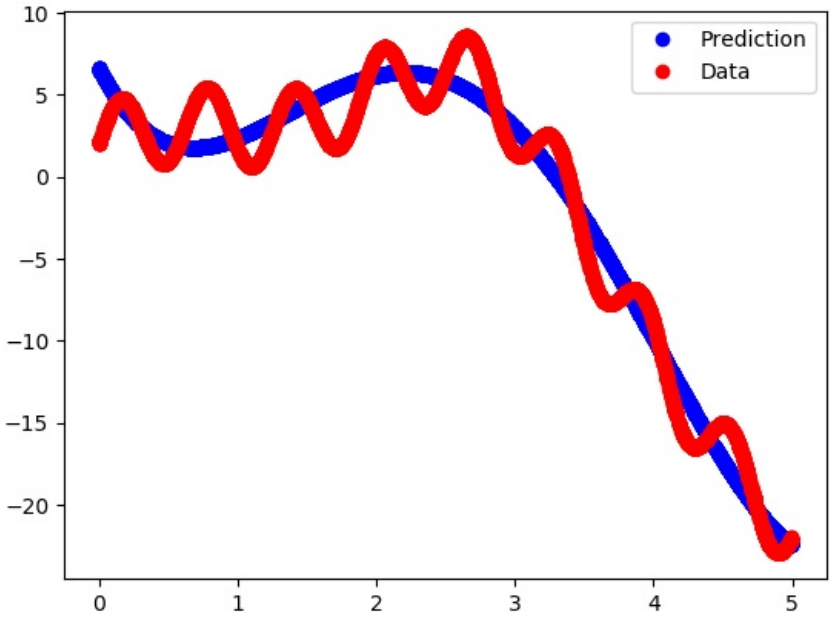
1.4 Inferences

Following can be inferred from the data -

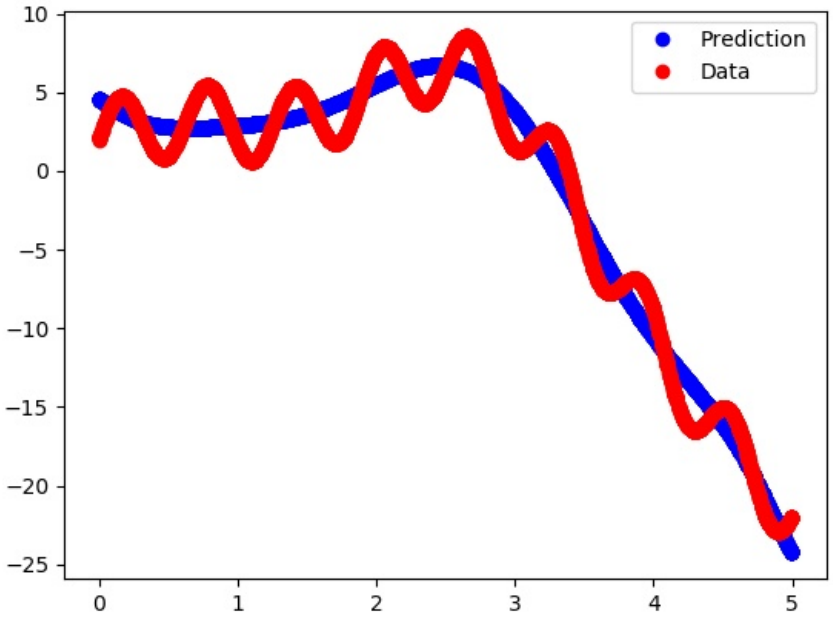
- Initially, bias seems to be very high for a naively complex model (i = 1). It signifies that the trained model seems to underfit with the test data as well as the trained data. As we could see the plot below, the model is not only failing on data both training and testing. The variance seems to be low initially as it consistently. This is an example of **Underfit** model.



- As we increase the complexity, bias drops drastically and the model is able to almost accurately predict the trend. However, due to increase in parameters, a small change in test data results in very high change in the predictive model. For us, we achieve the optimal complexity at order of 4.



- Further ,if we increase the complexity more, we get a model highly **Overfit**. In that, it is involving a very high variance. Any small changes in the test data results in the parameters getting severely changed. Although the model involves very low bias, we don't choose it because of the high total error present in the model due to high variance.



2 Question 2

2.1 Problem Statement

Given 20 subsets of training data containing 400 samples each, train 20 models on each subset of data and observe the bias and variance on testing data. Comment on the fit of the model by plotting the bias-variance graph.

2.2 Algorithm Implementation

1. Loading the data file

```
#=============================================================================#
# Loading the Data file #
#=============================================================================#
file = open('Q2_data/Fx_test.pkl', 'rb')
test_y = pickle.load(file)
file.close()
file = open('Q2_data/X_test.pkl', 'rb')
test_x = pickle.load(file)
```

```
file = open('Q2_data/X_train.pkl', 'rb')
train_x = pickle.load(file)
file = open('Q2_data/Y_train.pkl', 'rb')
train_y = pickle.load(file)
print("Loaded Data Successfully!!!")
#================================================================================#
```

2. Training a polynomial model on given data.

```
for i in range(10) :
   prediction = np.zeros((80, 1))
   pred_array = []
   variance = 0
   mse = []
   for j in range(20) :
      poly = PolynomialFeatures(degree = i)
      subTr_x = np.reshape(train_x[j], (400, 1))
      subTr_y = train_y[j]
      subTr_x = poly.fit_transform(subTr_x)
      reg = LinearRegression().fit(subTr_x, subTr_y)
      subTs_x = test_x
      subTs_x = poly.fit_transform(subTs_x)
      pred = np.reshape(reg.predict(subTs_x), (80, 1))
      pred_array.append(pred)
      prediction = prediction + pred
```

We have used the **PolynomialFeatures()** function which generates a feature matrix consisting of all polynomial combinations of the features with degree less or equal to specified degree. In each iteration, $subTr\_x$ is train the model via the **LinearRegression().fit()** function. After this, the model is then fitted onto the test data using the **poly.fit_transform()** function.

3. Finding the mean variance and bias.

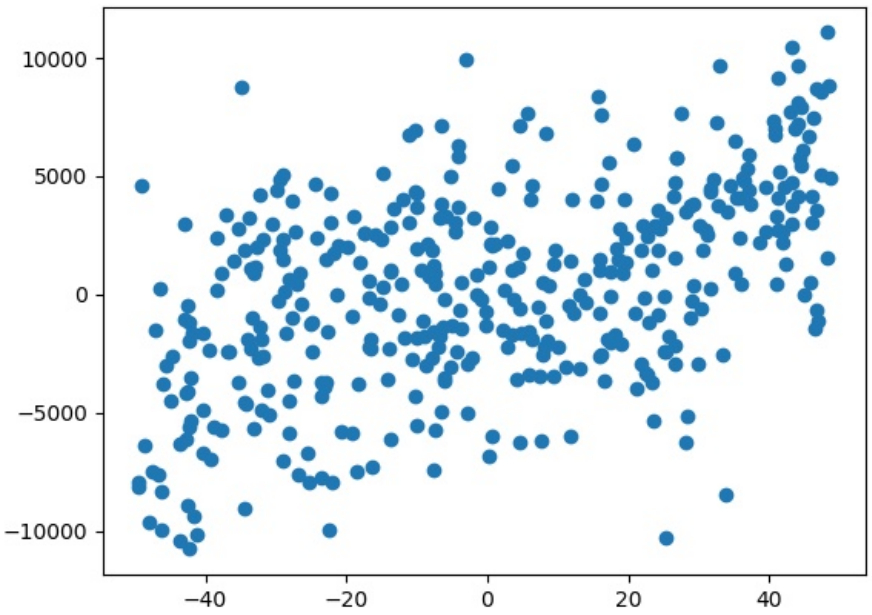
```
pred_array = np.asarray(pred_array)
prediction = prediction/20
diff = pred_array - prediction
bias = (np.mean((prediction - test_y))**2)
variance = np.mean(diff**2)
```

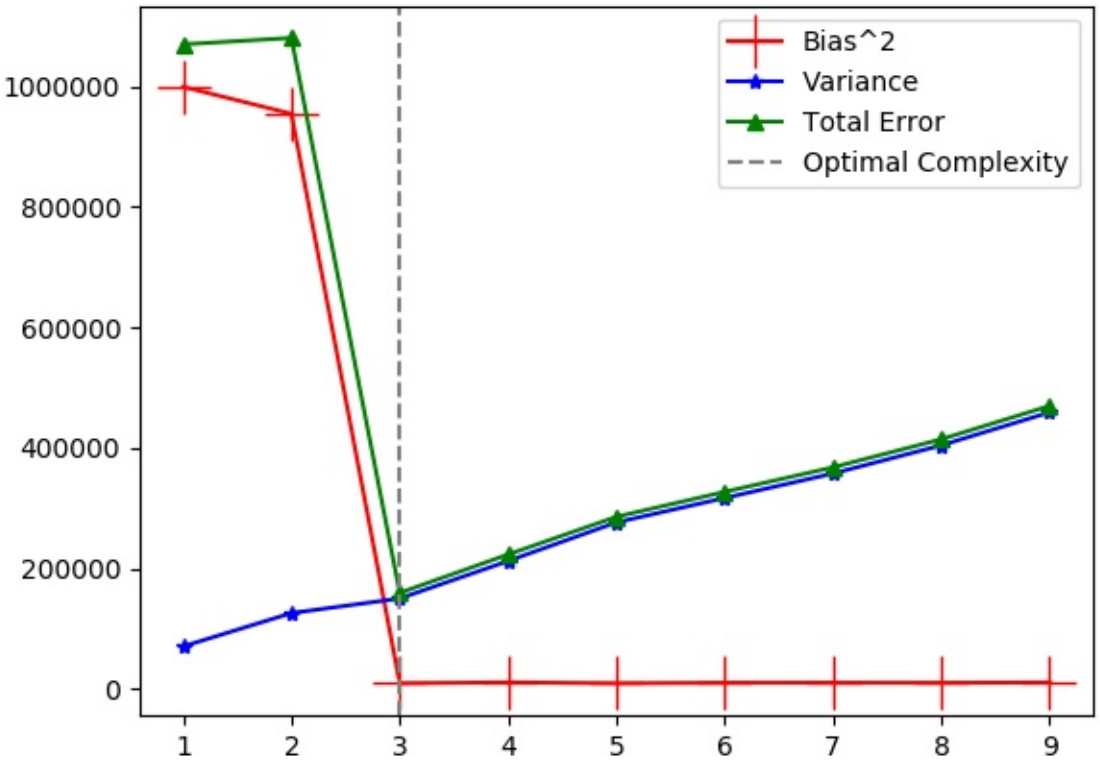
Bias^2 and variance are found out as per the formulae mentioned above.

Note that the *pred_array* is an array containing the values and thus the bias and variance are already calculated in a ***vectorized*** manner on numpy arrays.

2.3 Results

Here is the scatter plot of the given data -





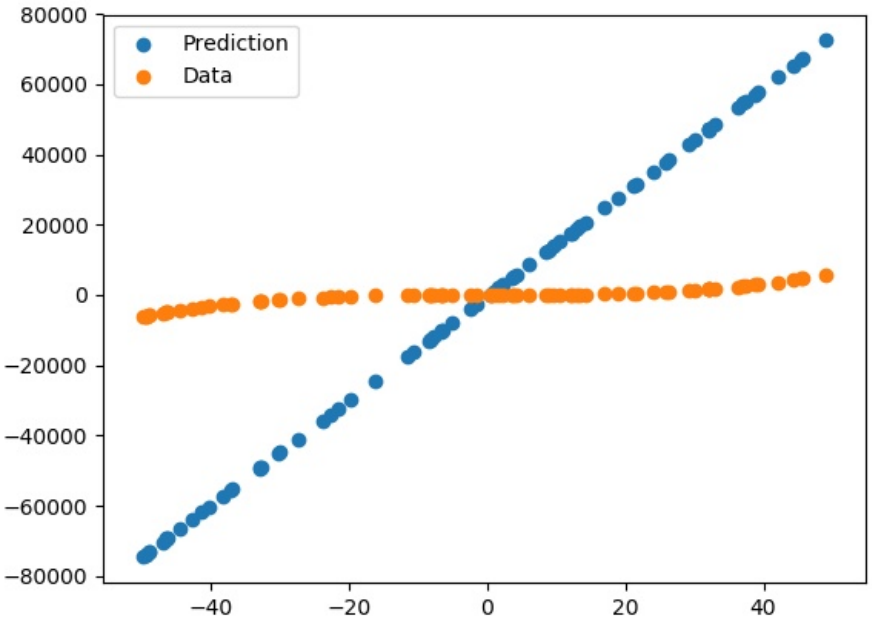
Here is the tabulated version of the following data -

Model Complexity	**Bias**	**Variance**	**Total Error**
1	999228.396872	70545.4891458	1069773.88602
2	954619.273794	125870.855549	1080490.12934
3	9389.73011679	150073.739546	159463.469663
4	10907.348134	212235.708326	223143.05646
5	9339.19428314	276388.480184	285727.674467
6	10248.586398	316863.497736	327112.084133
7	10335.2794411	357511.103478	367846.382919
8	10149.4951806	404289.174765	414438.669946
9	10715.8079055	459113.935504	469829.74341

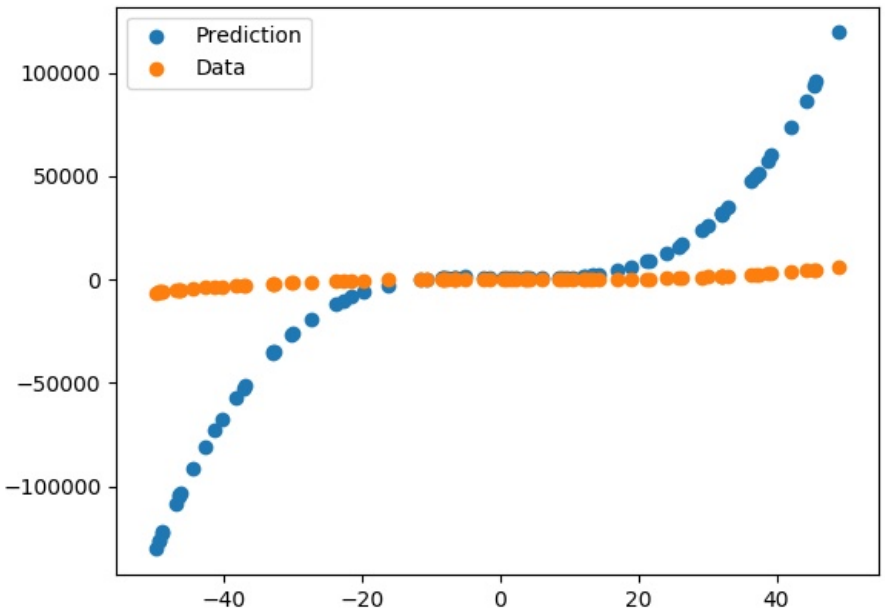
2.4 Inferences

Following can be inferred from the data -

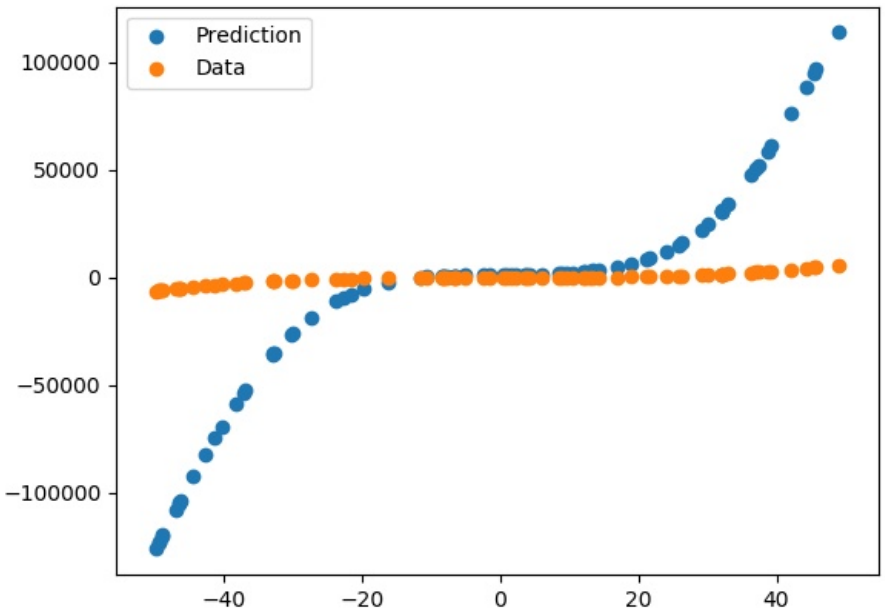
- Initially, bias seems to be very high for a naively complex model (i = 1). It signifies that the trained model seems to underfit with the test data as well as the trained data. As we could see the plot below, the model is not only failing on data both training and testing. The variance seems to be low initially as it consistently. This is an example of **Underfit** model.



- As we increase the complexity, bias drops drastically and the model is able to almost accurately predict the trend. However, due to increase in parameters, a small change in test data results in very high change in the predictive model. For us, we achieve the optimal complexity at order of 3.



- Further ,if we increase the complexity more, we get a model highly **Overfit**. In that, it is involving a very high variance. Any small changes in the test data results in the parameters getting severely changed. Although the model involves very low bias, we don't choose it because of the high total error present in the model due to high variance.



- Upon looking at the plot, we offer to comment the following on the data -

    1. The data has very high variance, in that it requires a model of complexity 3 to be fit.
    2. Also, the data is very noisy, since at particular points we obtain multiple values.