ASSIGNMENT 1

Shivansh Anand & Abinash Maharana
Team 33
Machine Data and Learning

Question 1

1.1 INTRODUCTION

We are given a problem statement where we have to tabulate the biases and variances of the predicted functions to given training and test data set. We have been provided a dataset consisting of pairs (x i , y i). Then we were supposed to split the dataset intotraining and testing (90:10 split) sets .After that we were asked to divide the train set into 10 equal parts randomly, so that we get 10 different dataset to train our model. Further we have to train a linear classifier on each of the 10 train sets separately, so that we have 10 different classifiers or models. So now we have 10 different models or classifiers trained separately on 10 different training set, and then at last we should calculate the bias and variance. We need to repeat the above process for the following class of functions.

y = mx
y = ax2 + bx + c
y = ax4 + bx3 + cx2 + dx + e

And so on up till polynomial of degree 9. We were only supposed to use sklearn's linear_model.LinearRegression().fit() for finding the appropriate coefficients with the default parameters.

1.2 ALGORITHM DEFINITION

Linear Regression : Linear Regression is a machine learning algorithm based on supervised learning. It performs a regression task. Regression models a target prediction value based on independent variables. It is mostly used for finding out the relationship between variables and forecasting.Linear regression performs the task to predict a dependent variable value (y) based on a given independent variable (x). So, this regression technique finds out a linear relationship between x (input) and y(output).

Hypothesis Function for Linear Regression:

$$Y = \theta_1 + \theta_2 * X$$

While training the model we are given :

X : Input Training data (univariate – one input variable(parameter))

Y : Labels to data (supervised learning)

When training the model – it fits the best line to predict the value of y for a given value of x. The model gets the best regression fit line by finding the best θ 1 and θ 2 values.

θ1 : intercept

θ2 : coefficient of x

Once we find the best θ 1 and θ 2 values, we get the best fit line. So when we are finally using our model for prediction, it will predict the value of y for the input value of x.

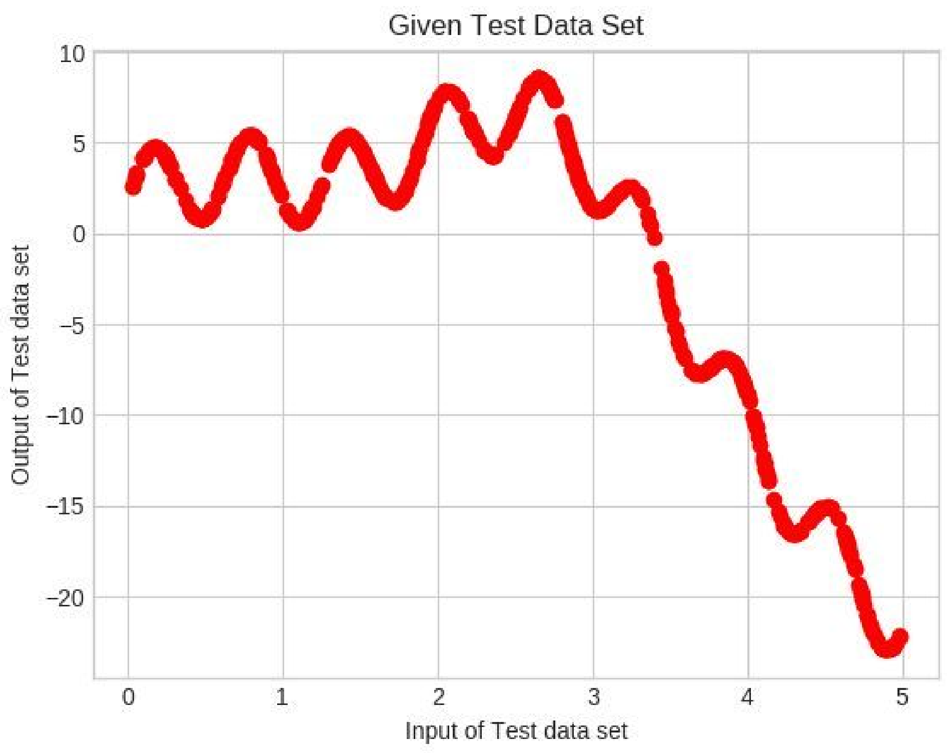
Cost Function (J) : By achieving the best-fit regression line, the model aims to predict y value such that the error difference between predicted value and true value is minimum. So, it is very important to update the θ 1 and θ 2 values, to reach the best value that minimize the error between predicted y value (pred) and true y value (y). Cost function(J) of Linear Regression is the Root Mean Squared Error (RMSE) between predicted y value (pred) and true y value (y).

Gradient Descent: To update θ 1 and θ 2 values in order to reduce Cost function (minimizingRMSE value) and achieving the best fit line the model uses Gradient Descent. The idea is to start with random θ 1 and θ 2 values and then iteratively updating the values, reaching minimum cost.

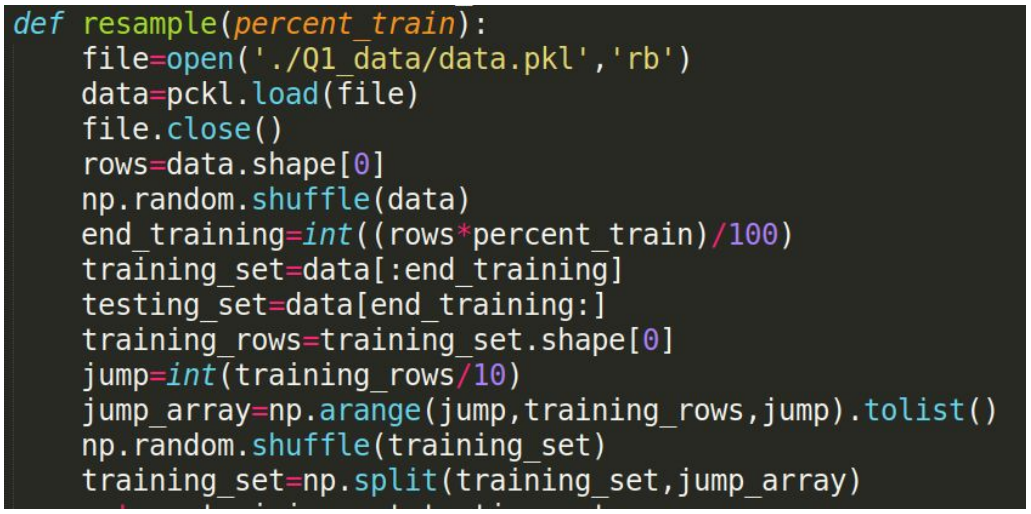
1.3 EXPERIMENTAL EVALUATIONS

Methodology: Following are steps to proceed to given problem statement and to get answer.

- ***Loading the Data :*** Loaded the data using pickle.load.Then using random shuffle and then divided into test and train data.

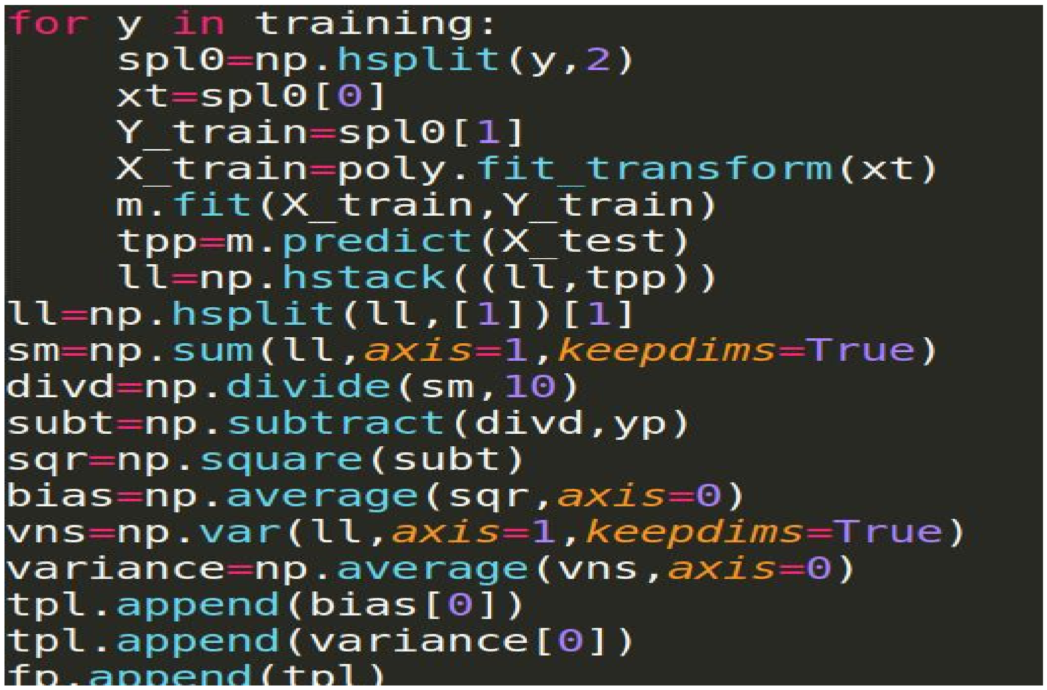


- ***Building the Model :***



```
def resample(percent_train):
    file=open('./Q1_data/data.pkl','rb')
    data=pckl.load(file)
    file.close()
    rows=data.shape[0]
    np.random.shuffle(data)
    end_training=int((rows*percent_train)/100)
    training_set=data[:end_training]
    testing_set=data[end_training:]
    training_rows=training_set.shape[0]
    jump=int(training_rows/10)
    jump_array=np.arange(jump,training_rows,jump).tolist()
    np.random.shuffle(training_set)
    training_set=np.split(training_set,jump_array)
```

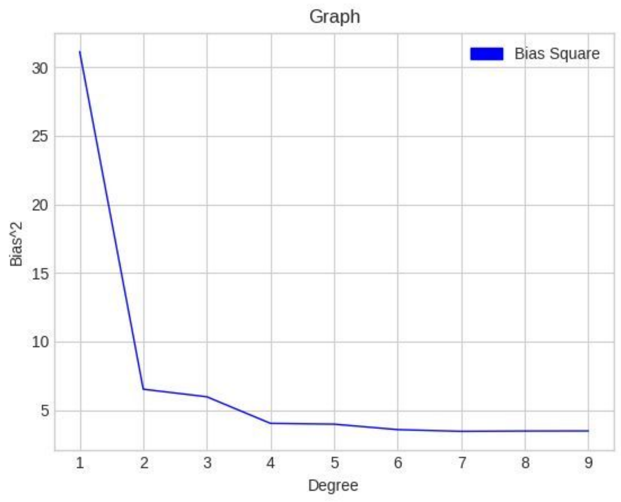
For bias variance following :

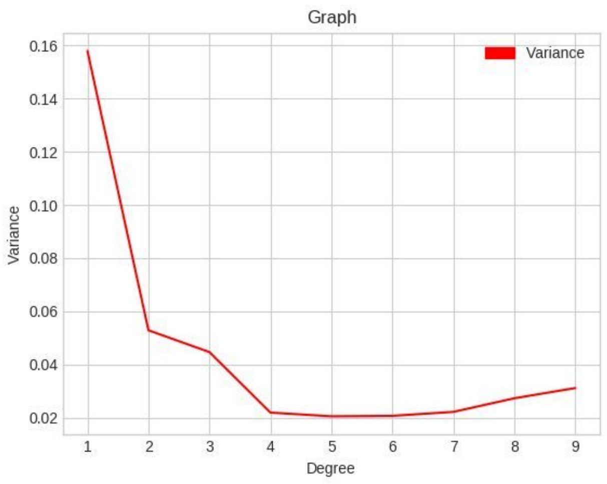


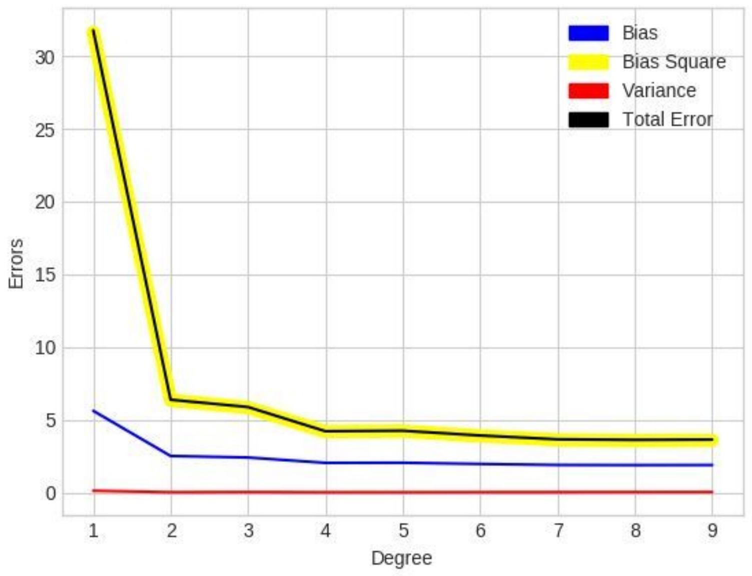
```
for y in training:
    spl0=np.hsplit(y,2)
    xt=spl0[0]
    Y_train=spl0[1]
    X_train=poly.fit_transform(xt)
    m.fit(X_train,Y_train)
    tpp=m.predict(X_test)
    ll=np.hstack((ll,tpp))
ll=np.hsplit(ll,[1])[1]
sm=np.sum(ll,axis=1,keepdims=True)
divd=np.divide(sm,10)
subt=np.subtract(divd,yp)
sqr=np.square(subt)
bias=np.average(sqr,axis=0)
vns=np.var(ll,axis=1,keepdims=True)
variance=np.average(vns,axis=0)
tpl.append(bias[0])
tpl.append(variance[0])
fp.append(tpl)
```

- ***Analysing Bias and Variance :***

 We are plotting Bias and Variance for each model and against model size to see if the trends are satisfied. We expect that * The Smaller Models will be High Bias and Low Variance since it has very little space to produce varying outputs, i.e. The function it learns will be simple. But bias is huge cause it didn't really learn a lot.
* The Bigger Models will be High Variance and Low Bias since it can overfit the data getting the mean almost perfectly equal, but have huge deviations due to learning too complex a function on different input.

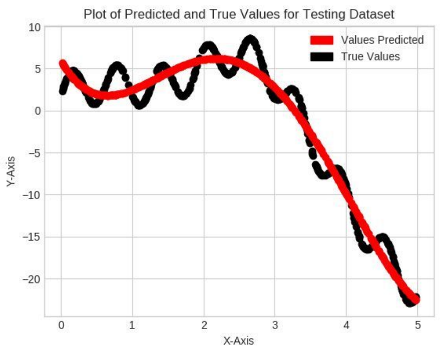


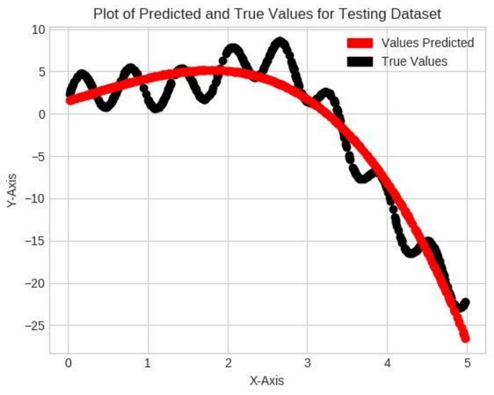


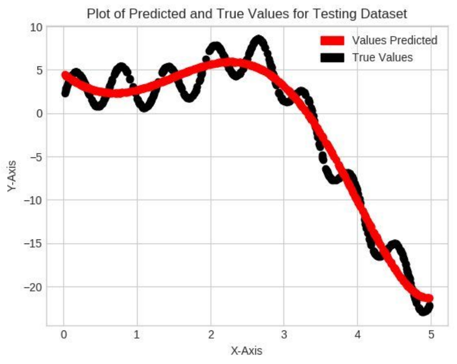


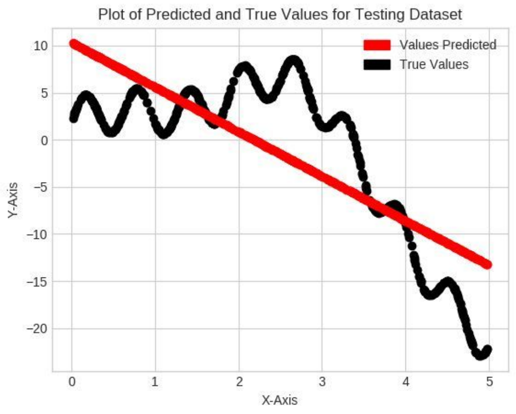
Since here in the given test data set, the value of variance is negligible with respect to Bias2 . Hence, We get almost coincident lines representing Total Error and Bias2.

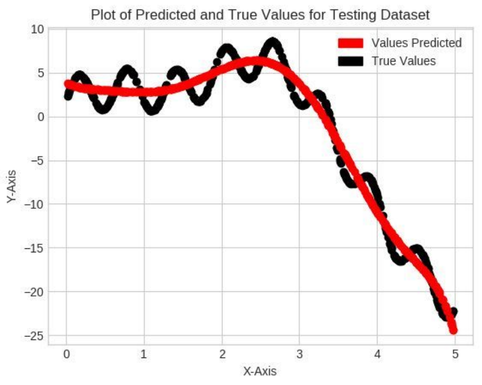
● **Check the Model's Predictions:** One final run to see how the models, on average are fitting the data. Here we have checked the models trained on 90 parts of data set i.e. whole training set rather than splitting training set into 10 sets. This is because if we check for all the parts of training sets then we will get 90 different models and that's very tedious to present on a report. The basic idea of training a model is the same.. The difference lies in the fact of splitting the training data set into 10 parts.

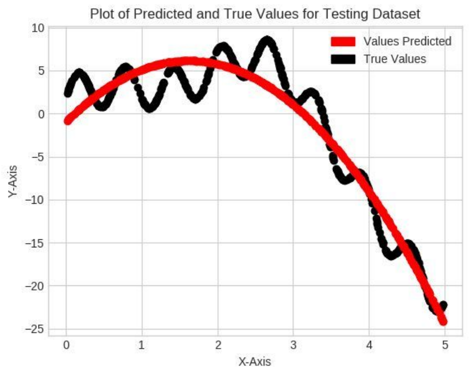

Plot of Predicted and True Values for Testing Dataset
Values Predicted
True Values
Y-Axis
X-Axis

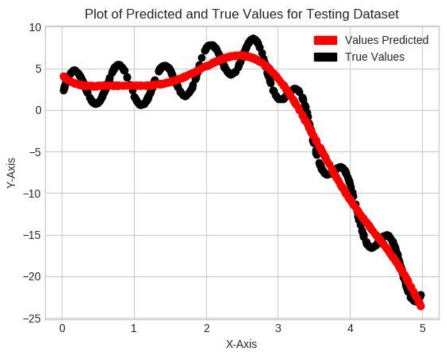

Plot of Predicted and True Values for Testing Dataset
Values Predicted
True Values
Y-Axis
X-Axis

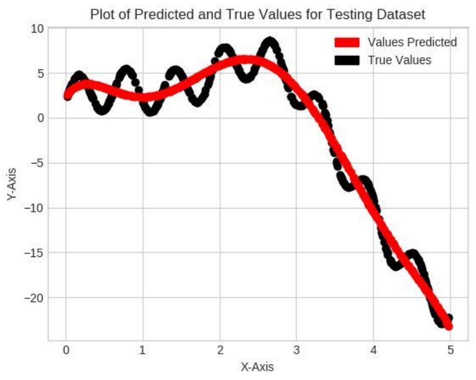

Plot of Predicted and True Values for Testing Dataset
Values Predicted
True Values
Y-Axis
X-Axis

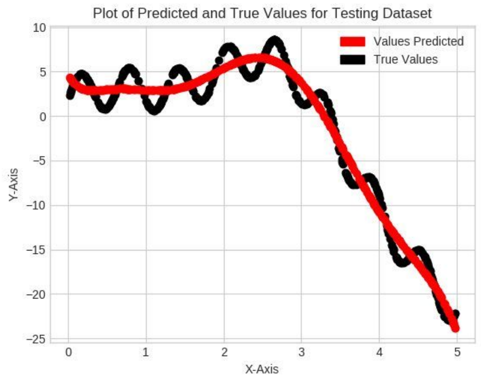

Plot of Predicted and True Values for Testing Dataset
Values Predicted
True Values
Y-Axis
X-Axis


Plot of Predicted and True Values for Testing Dataset
Values Predicted
True Values
Y-Axis
X-Axis


Plot of Predicted and True Values for Testing Dataset
Values Predicted
True Values
Y-Axis
X-Axis

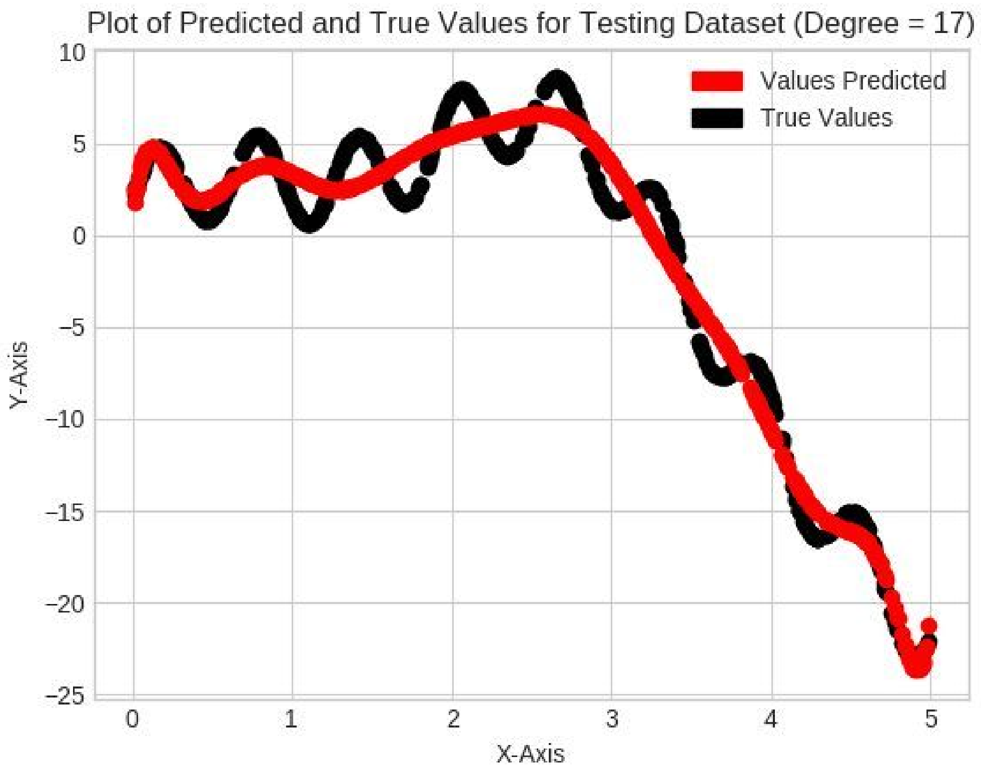






1.4 CONCLUSION

The Variance is bizarrely decreasing with the increase in complexity of the model. This is because of the way the dataset is structured. There is little to no noise in the data , all the data follows a very regular distribution, which is approximated by all the models almost equally well and in the same fashion. This would be very different if there was a lot of noise in the data. The Smallest Agent are both High Bias and High Variance as they can neither approximate well nor correctly, the approximations keep changing. The Wavy nature of the curve allows for multiple lines of best fit. We also see that after decreasing the variance increases a little bit . This is where the models are actually becoming powerful, and are overfitting a little. However, until we take a 17 degree model, we will not overfit too much.



Plot of Predicted and True Values for Testing Dataset (Degree = 17)
Values Predicted
True Values
Y-Axis
X-Axis
10
5
0
−5
−10
−15
−20
−25
0
1
2
3
4
5