MDL Assignment 1

Yarava Lakshmi Madhuri, 2018101116
Tejaswini Yeleswarapu, 201456240

Question1

Part1: Importing Libraries

We started by importing all the necessary libraries required for this question as shown in the code below. **train_test_split() method** Split arrays or matrices into random train and test subsets. **NumPy** is an open source Python package for scientific computing, supports large, multidimensional arrays and matrices. **matplotlib**. **pyplot** is a collection of command style functions that make **matplotlib** work like MATLAB.

```
import pickle
import numpy as np
import operator
from operator import itemgetter
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt
from sklearn.preprocessing import PolynomialFeatures
import pandas as pd
```

Part2: Loading the Data

We then loaded the data using the pickle.load function and and started plotting the basic structure of the problem.
The process of loading a pickled file data.pkl into a python program . And ‘rb’ denotes r for read mode and b for byte mode.

```
infile = open("data.pkl",'rb')
```

```
new_dict = pickle.load(infile)
```

Part3: Segregating the Data

We segregated the given data into training data and testing data and stored then x,y coordinates in two different lists.The data is split into test data of size 1 and train data of size 9 of num=10 and gets shuffled and stored into X_train and X_test using **train_test_split() function.**

```
X_train, X_test = train_test_split(new_dict, train_size=0.9,test_size=0.1,andom_state=1)
y_train=(list(map(itemgetter(1),X_train)))
y_test=(list(map(itemgetter(1),X_test)))
X_train=list(map(itemgetter(0),X_train))
X_test=list(map(itemgetter(0),X_test))
```

Part4: Splitting the Data

We then split the data into 10 equal training sets as given in the question:-

```
final_bias=[]
final_var=[]
final_norm=[]
my_sets=np.array_split(X_train,10)
your_sets=np.array_split(y_train,10)
```

Part5: Creating Linear Classifiers

LinearRegression fits a linear model with coefficients w = (w1, …, wp) to minimize the residual sum of squares between the observed targets in the dataset, and the targets predicted by the linear approximation.

We created linear classifiers (using linear regression) for polynomials of degree upto 9 for one data set. We then fit our test data following which the predicted data is stored. This needs to be run ten times for the ten different training sets and here mean() is a statistics module function that used to calculate the average of numbers.

Train all 10 degrees on all 10 sets and average bias and variance over all 10 sets for each degree.

```
for ds in range(10):
        xtr = np.array([my_sets[ds]]).T
        ytr = np.reshape(your_sets[ds],(-1,1))
        yts=np.reshape(y_test,(-1,1))
        xts=np.reshape(X_test,(-1,1))
        poly_here=PolynomialFeatures(degree=degree)
        my_poly=poly_here.fit_transform(xtr)
        madhuri=LinearRegression().fit(my_poly,ytr)
        xts=poly_here.fit_transform(xts)
        pred=madhuri.predict(xts)
        temp.append(pred)
```

Part6: Storing Predicted Values

As mentioned above, the predicted values were stored in a separate array:=

```
        preds = []
        for i in range(9):
        preds.append(temp[i].T[0])
        temp = np.array(preds)
```

Part6: Analysis of Bias and Variance

In the penultimate step, we calculated bias and variance based on predicted value set and original result of test data. In a range of 0-500 we will calculate the (bias)^2 and we add bias value to tot_bias. Append() function is used for appending the bias_total value to final bias list and variance total value to final variance list

Bias is calculated using the following formula:-

Bias = mean(predicted-result of test data)^2

We used the default function to calculate variance.

```
        tot_bias = 0
```

```
    tot_var=0
    for k in range(500):
        tot_bias += np.mean(((temp.T[k]-y_test[k])**2))
        tot_var += (np.mean(np.var(preds,axis=0)))
        tot_bias/=500
        tot_var/=500
        final_bias.append(tot_bias)
        final_var.append((tot_var))
```

Part7: Plotting Graphs

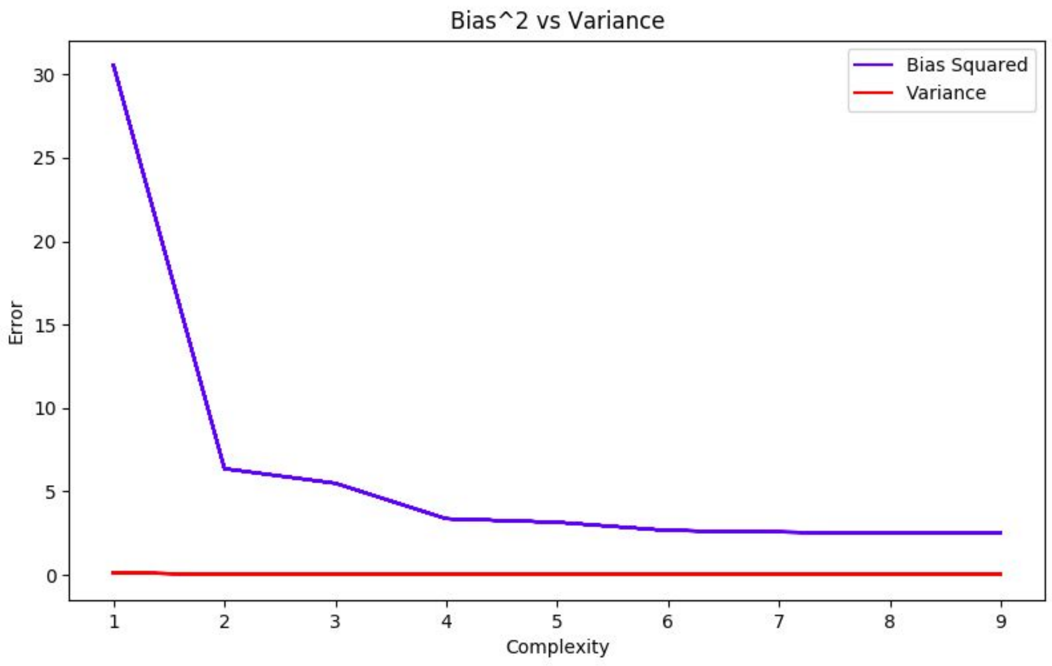
As a final step, this code was used to plot graphs:-
Firstly,created array of size 9 for plotting polynomial degree vs bias ,variance
And then plotting using library matplotlib.pyplot

The below code snippet is for plotting the graph (**number of parameters Vs Bias and number of parameters Vs Variance**)

```
arr=[]
for me in range(1,10):
    arr.append(me)

for i in range(9):
    print(final_bias[i],final_var[i])
    poly_here=PolynomialFeatures(degree=i,interaction_only=True)
    plt.plot(arr, final_bias, color="b")
    plt.plot(arr, final_var, color="b")
    plt.title('Bias^2 vs Variance')
    plt.legend(('Bias Squared', 'Variance'), loc='best')
    plt.xlabel("Complexity")
    plt.ylabel("Error")
plt.show()
```

Final Graph



Result:

	Bias	Variance
0	30.52531699797918	0.1306026903977567
1	6.367291195036185	0.04117468664328044
2	5.495460264196135	0.04096430789155517
3	3.368859059376354	0.02434818386726125
4	3.158469659647835	0.028303819594994218
5	2.6645134075270254	0.03004602215844085
6	2.5698825726279293	0.038986015652151776
7	2.5198457893092834	0.04224222975138506
8	2.501522412677898	0.04739726454329698

As shown by the above values, a higher degree results in the increase of features for data and hence bias will decrease(Overfitting). With respect to variance, data gets randomly split into different parts thereby causing an increase as we extract too much information from train data and perform well on train data but such models won't perform well on test data.Here , we use

Supervised Learning. To overcome under-fitting, we need to increase the **complexity** of the model.To generate a higher order equation we can add powers of the original features as new features.To convert the original features into their **higher order terms** we will use the **PolynomialFeatures class** provided by scikit-learn. Next, we train the model using **Linear Regression.**

Question2

Part1: Importing Libraries

We started by importing all the necessary libraries required for this question as shown in the below code:-

```
import pickle
import numpy as np
import operator
from operator import itemgetter
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt
from sklearn.preprocessing import PolynomialFeatures
```

Part2: Loading Data

The process of loading pickled files given into a python program ,and 'rb' denotes r for read mode and b for byte mode.As the training and test data has already been given, we just loaded those instead of splitting the data like in the first question:-

```
infile1 = open("X_train.pkl",'rb')
new_trx = pickle.load(infile1)

infile2 = open("X_test.pkl",'rb')
new_tsx = pickle.load(infile2)

infile3 = open("Fx_test.pkl",'rb')
new_tsy = pickle.load(infile3)

infile4 = open("Y_train.pkl",'rb')
```

```
new_try = pickle.load(infile4)
```

Part3: Prediction and Storage

As we have twenty equal parts of train data , we find the predicted values twenty times and store it in “temp”. We used polynomial features for linear classifiers upto degree 10 to achieve the same:-

```
for j in range(1,20,1):
 my_sets=np.reshape(new_trx[j],(-1,1))
 your_sets=np.reshape(new_try[j],(-1,1))
 dy=np.reshape(new_tsy,(-1,1))
 dx=np.reshape(new_tsx,(-1,1))

 #print(your_sets)
 poly_here=PolynomialFeatures(degree=i)
 my_poly=poly_here.fit_transform(my_sets)
 #print(my_poly)
 madhuri=LinearRegression().fit(my_poly,your_sets)
 dx=poly_here.fit_transform(dx)
 huhu=madhuri.predict(dx)
 temp.append(huhu)
```

Part5: Calculating Bias and Variance

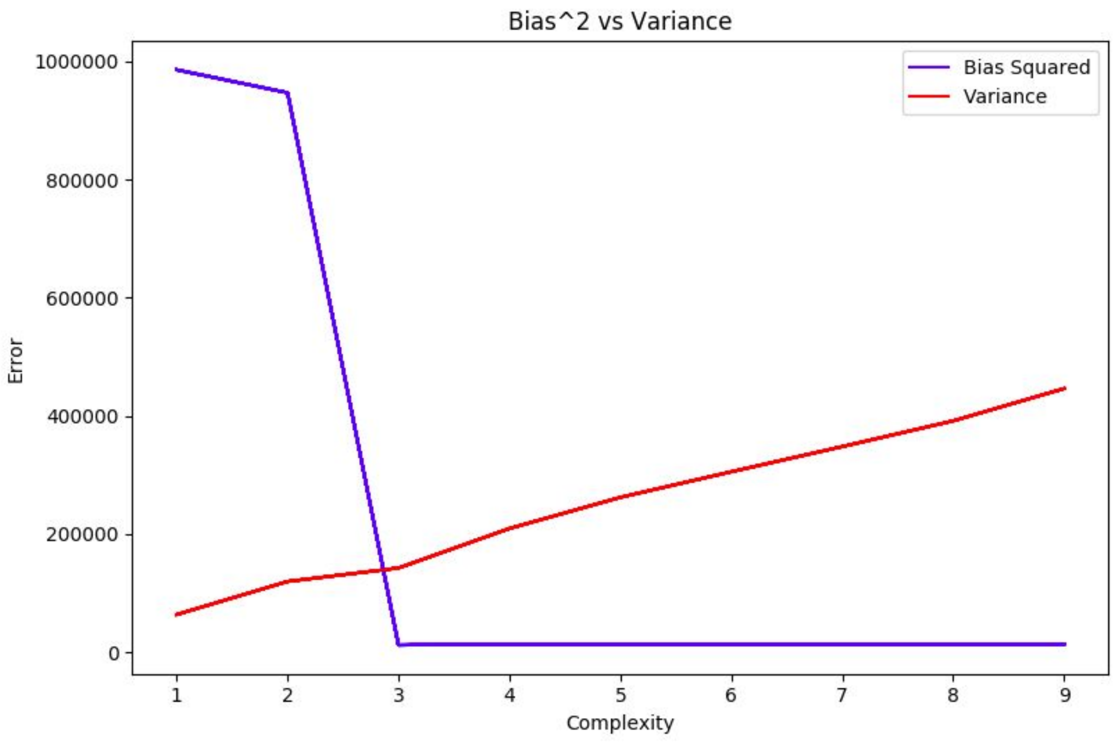
To calculate bias for every column in the predicted data, we use the below formula and take their mean and append it to the final array-

Bias = predicted[of that particular index]-y_test[values of that particular index]
Variance = E [(f_cap(x) − E[f_cap(x)])^2] ,
where f(x) represents the true value, f^(x) represents the predicted value

Variance for every index in predicted data is calculated in the same way.
Formula is ….

```
arr_bias=[]
arr_var=[]
for k in range(0,80,1):
        calc=(list(map(itemgetter(k),temp)))
        only=abs(np.mean(calc)-new_tsy[k])
        only=only*only
        arr_bias.append(only)
        var1=(np.mean(calc))
        var2=np.mean(abs((calc-var1)*(calc-var1)))
        arr_var.append(var2)
print(calc)
there=np.mean(arr_bias)
their=np.mean(arr_var)
final_bias.append(there)
final_var.append(their)
```

Final Graph:



Final Result:

	Bias	**Variance**
1	985789.3888662659	63435.85355557959
2	946940.8052755384	119456.44008512162
3	11727.50555783929	142300.6011466644
4	13102.458211946407	209149.26335241567
5	12937.703542769352	261899.93415413116
6	13272.971529393926	305256.71611296537
7	12578.098584781541	347817.39224963635
8	12944.393589338046	391366.0185978751
9	13775.09648565172	446082.507023257

Bias refers to the error due to the model's simplistic assumptions in fitting the data. A high bias means that the model is unable to capture the patterns in the data and this results in **under-fitting**.

Variance refers to the error due to the complex model trying to fit the data. High variance means the model passes through most of the data points and it results in **overfitting** the data.

To overcome under-fitting, we need to increase the **complexity** of the model.To generate a higher order equation we can add powers of the original features as new features.To convert the original features into their **higher order terms** we will use the **PolynomialFeatures class** provided by scikit-learn. Next, we train the model using **Linear Regression.**

Finally,Clearly for a given training set when the degree is low the hypothesis will underfit the data and there will be a high bias error. However when the degree of the polynomial is high then the fit will get better and better on the training set

This error will decrease as the degree of the polynomial increases as we will tend to get a better fit. However the error will again increase as higher degree polynomials that overfit the training set will be a poor fit for the cross validation set.

From the above data we can say that as the model complexity increases(polynomial degree to 20), the bias decreases and the variance increases.