REPORT: LINEAR REGRESSION

- Prince Singh Tomar (2019101021)
- Utkarsh Upadhyay (2019101010)

Task 1

Given a Linear Regression equation as: $y = b + \sum_{i=1}^{n} w_1 x_1$

LinearRegression().fit(x,y) is a function of the Class sklearn.linear_model.LinearRegression. It calculates the optimal values of the weights b and x_i ($\forall i \in [1,n]$) using the existing values of input (vector x which is a 2D array with n columns and each row represents one input) and output (y) from the train set.

These values are selected to minimize sum of squares between observed targets in the dataset and targets predicted by the linear approximation. It returns self, which is the an instance of the class Linear Regression.

Bias Varinace Table

Polynomial Function Degree	Bias	Varinace
1	820.1274029682196	23731.01464687495
2	811.5730310058455	70214.29003169124
3	67.20494330902764	57949.96115114641
4	77.24739410735778	70522.4381610456
5	76.98315532066735	81445.51638642991
6	75.46431283880787	90865.21613854464
7	81.89558694700833	100371.24692962335
8	83.87197593851022	132803.85460543976
9	83.224022338467	150769.81287848824
10	88.26058120838228	166940.40797911023
11	86.28055147245456	172618.76657839367
12	116.12440103896374	194131.77628783946
13	90.44937505320748	191979.63440157074
14	126.8661984242414	196947.464337216
15	161.12265463089085	200457.87006939476
16	166.63644820555805	202688.30748305662
17	232.54036599126349	210622.46210396715
18	233.89677351607446	209321.40413978443
19	304.1232994472372	222953.56650112747
20	302.1961271492279	221821.72018012352

Code:

```
from sklearn.preprocessing import PolynomialFeatures
import math

x = np.array(temp_x).reshape(-1,1)
y = np.array(temp_y)

temp_test_x2 = np.array(temp_test_x).reshape(-1,1)

mean_bias = []
squared_mean_bias = []
mean_variance = []
total_polynomials = 20
total_error = []
# irreduceble_error = []

for i in range(1,total_polynomials+1):        # Polynomial loop
    poly_bias = []
    val_mean = 0
    val_squared_mean = 0
    val_variance = 0
    temp_total_error = 0
    bias = [0 for k in range(1,total_polynomials+1)]
    for j in range(0,10):       # test case loop
        abc = np.array(resampled_x[j]).reshape(-1,1)
        poly_reg = PolynomialFeatures(degree=i)
        X_poly = poly_reg.fit_transform(abc)
        # poly_reg.fit(X_poly,y)
        # print("i : " + str(i) + " | j : "+ str(j))
        lin_reg = LinearRegression().fit(X_poly,resampled_y[j])
        # plt.plot(temp_test_x2, lin_reg.predict(poly_reg.fit_transform(temp_test_x2)),'o')
        predicted_value = lin_reg.predict(poly_reg.fit_transform(temp_test_x2))
        poly_bias.append(predicted_value)
    # finding bias below :
    for j in range(0,len(poly_bias[0])):
        temp = 0
        for k in range(0,10):
            temp += poly_bias[k][j]
        temp = temp/10
        for k in range(0,10):
            val_squared_mean +=  math.pow((temp - temp_test_y[j]),2)
            val_mean += abs(temp - temp_test_y[j])
    val_mean = val_mean/len(poly_bias[0])
    val_squared_mean = val_squared_mean/len(poly_bias[0])
    mean_bias.append(val_mean/10)
    squared_mean_bias.append(val_squared_mean/10)
```

```
    # variance finding down :
    for j in range(0,len(poly_bias[0])):
        temp = 0
        for k in range(0,10):
            temp += poly_bias[k][j]
        temp = temp/10
        for k in range(0,10):
            val_variance +=  math.pow((temp - poly_bias[k][j]),2)
    val_variance = val_variance/len(poly_bias[0])
    mean_variance.append(val_variance/10)
    # total error below :
    for j in range(0,len(temp_test_y)):
        for k in range(0,10):
            temp_total_error += math.pow(temp_test_y[j] - poly_bias[k][j],2)
    temp_total_error = temp_total_error/len(poly_bias[0])
    total_error.append(temp_total_error/10)

irreduceble_error = [(total_error[i] - squared_mean_bias[i] - mean_variance[i]) for i in range(0,total_polynomia
ls)]
print("\x1b[6;30;46m"+"Mean Bias : " + "\x1b[0m" + "\n",mean_bias)
print("\x1b[6;30;46m"+"Squared Mean Bias : " + "\x1b[0m" + "\n",squared_mean_bias)
print("\x1b[6;30;46m"+"Mean Variance : " + "\x1b[0m" + "\n",mean_variance)
print("\x1b[6;30;46m"+"Total Error : " + "\x1b[0m" + "\n",total_error)
print("\x1b[6;30;46m"+"Irreduceble Error : " + "\x1b[0m" + "\n",irreduceble_error)
```

Bias: Initially as the degree of hypothesis increases, the bias (difference between predicted value and correct value) decreases. This is because if the degree of the polynomial is too low, the hypothesis fails to capture important regularities in the training set resulting in a large difference between actual values and predicted values (this is called underfitting)

--- (for higher degrees).

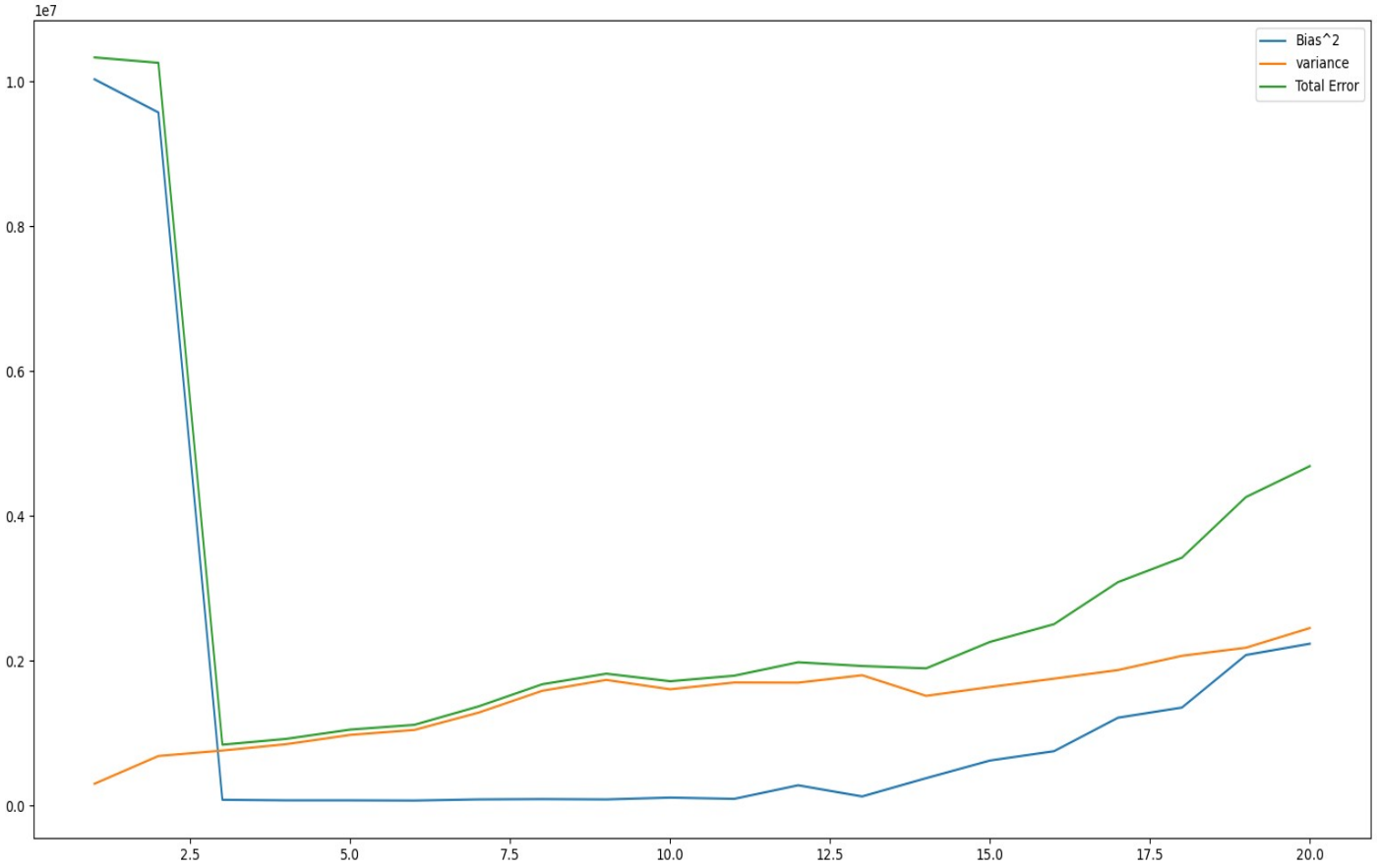
Variance: In general, variance increases with increase in degree of the polynomial function (hypothesis). This is because as the degree of the polynomial increases, the hypothesis becomes more flexible and more sensitive to minor variations in the training set. At higher degrees, the co-efficients of the hypothesis vary significantly in response to the fluctuations in the training set (noise), which results in an increase in variance (this is known as overfitting).

Irreducible Error

Polynomial Function Degree	Irreducible Error
1	-1.4915713109076023 e-09
2	3.2159732654690742 e-09
3	1.0186340659856796 e-10
4	-4.3655745685100555 e-11
5	1.4551915228366852 e-11
6	-7.275957614183426 e-11
7	1.6007106751203537 e-10
8	5.820766091346741 e-11
9	1.7462298274040222 e-10
10	1.4551915228366852 e-10
11	2.9103830456733704 e-11
12	-8.731149137020111 e-11
13	-3.2014213502407074 e-10
14	-1.1641532182693481 e-10
15	-6.693881005048752 e-10
16	8.731149137020111 e-11
17	3.2014213502407074 e-10
18	3.2014213502407074 e-10
19	3.2014213502407074 e-10
20	-8.440110832452774 e-10

Irreducible error: The irreducible error of the hypothesis remains more or less the same because (within ± 3.21 e-09), as the name suggests, it is irrelevant of the underlying model and has to do with the inherent noise in the problem. This noise can represent noise coming from data quality (e.g., inaccuracies in collection or reporting of data), from only approximate knowledge of the true function describing the real-life problem, from non-deterministic behavior of the underlying phenomenon and, in general, any type of noise that cannot be easily defined.

Bias^2 – Variance Graph



Code:

```
plt.plot([i for i in range(1,total_polynomials + 1)],squared_mean_bias,label='Bias^2')
plt.plot([i for i in range(1,total_polynomials + 1)],mean_variance,label='variance')
plt.plot([i for i in range(1,total_polynomials + 1)],total_error,label='Total Error')
plt.legend()
plt.show()
```

Observation:

From observation we can say that, when degree of the hypothesis is less than 3, the bias of the hypothesis is very high which results in a large difference between actual values and predicted values (this is known as underfitting).

For higher degrees, the bias is considerably low, however, the variance of hypothesis becomes significantly large because the hypothesis becomes more flexible and more sensitive to minor variations in the training set (this is known as overfitting).

From the above graph, we can say that the total error of the hypothesis is least when the degree is 3. This means that the data shows non-linearity to some extent.