Assignment 1

Mukkiri Pravalika(2019101098)
MC Bhavana(2019101100)

Task 1

Linear Regression

It is a linear approach to modelling the relationship between a scalar response and one or more explanatory variables.The linear regression model takes given pairs of x and y values from the training data as arguments and fits them along a straight line or curve depending on the order of the x value projection.

LinearRegression().fit() does this in the following steps:

- ❏ Arbitrarily initializing a set of coefficients for these models.
- ❏ Calculating mean squared error for a predicted model
- ❏ The regressor now penalizes the existing model and consequently provides us a new set of coefficients.
- ❏ These two steps are repeated until the penalizing has little to no effect on the existing model

Expression:

LinearRegression.fit(X,Y)

Parameters:

X:Training data(array like space matrix)

Y: Target values. Will be cast to X's dtype if necessary(array like space matrix)

Task 2

1. Imported required Libraries as shown in snippet below.The pickle module is to convert a python object into a character stream. Numpy is the python package for computing and also largely used to support multidimensional matrices and arrays.matplot.pyplot is to plot the graph.

```
import pickle
import numpy as np
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
import math
```

2. Loading and Initializing the Data

Loaded the data using pickle model.'rb' is r(read mode) and b(byte mode).

```
open('train.pkl', 'rb') as f:
data = pickle.load(f)
with open('test.pkl', 'rb') as f:
data1 = pickle.load(f)

x_train=data[:,:-1]
y_train=data[:,1]
x_test=data1[:,:-1]
y_test=data1[:,1]
```

3. Dividing the data

Divided the train set into 10 equal parts randomly, so that we get 10 different train datasets to train the model.

```
x_cords_datasets=np.array(np.array_split(x_train,10))
y_cords_datasets=np.array(np.array_split(y_train,10))
```

4. Creating Linear classifiers

Linear Regression fits a linear model with coefficients to minimize the residual sum of squares between the observed targets in the dataset, and the targets predicted by the linear approximation.

The data set partitions are trained for polynomial models with complexity upto degree 20.Then,the predicted values for x_test is obtained for each of the 10 training data set partitions which is stored in a temp array and the mean() is that used to calculate the average of numbers.

```
for degree in range(1,21):
temp=[]
for ds in range(10):
poly_here=PolynomialFeatures(degree=degree, include_bias=False)
xtr=poly_here.fit_transform(x_cords_datasets[ds])
xts=poly_here.fit_transform(x_test)
reg=LinearRegression().fit(xtr,y_cords_datasets[ds])
temp.append(reg.predict(xts))
```

Calculating Bias and Variance

We calculated the bias and variance based on the predicted test set(temp) and true result of data.Bias is calculated by the formula *mean(temp-y_test(result of test data))*.Variance is calculated by using the function np.var and the bias and variance values were appended to respective arrays.

Bias

```
final_bias.append(abs(np.mean((np.mean(temp,axis=0)-y_test))))
```

Variance

```
final_var.append(np.mean(np.var(temp,axis=0)))
```

Results we get

Degree	Bias	Bias square	Variance
1	230.24621179020974	1001682.500648845	25999.0930099878
2	226.5165089990066	953836.1284241958	39105.83381326911
3	15.556092876570034	9533.344066292839	56095.89320974464
4	13.838667901460145	10588.298437385378	114907.29152938948
5	13.013400266603986	9952.445156729227	151434.0278565172
6	13.298331929468912	9999.197545907076	174226.74455039212
7	8.522307150149436	10425.925501349539	198849.68461579824
8	13.120786315648592	10998.3731384639	221551.96813647877
9	14.649893126305722	11604.768445899248	232378.9059556284
10	17.077789474084163	12923.573075222637	236185.28558500978
11	19.007224268818867	12491.078994915488	238090.8369555693
12	13.711023324907561	30680.926998748182	219356.68140677904
13	23.10235210218422	16254.107088694745	234171.30396785503
14	33.736621277578024	39230.89096329569	212545.0702908175
15	41.55332651357744	62928.27247914714	221715.12143846988
16	42.391751564015294	69821.66301116839	239355.63724936606
17	53.913162218852996	115239.94932996777	242992.95095726848

```
18    56.6174545615677   120582.79321116325   269050.63191597984
19    69.9142294883137   191978.57468342822   270101.2411667309
20    73.8943402057698   197306.4545325786   299029.01204823644
```

Task 3: Calculating Irreducible Error

Mean square Error

We calculated mean square error based on the formula E[(f (x) − f ˆ (x)) 2]. Here f(x) is y_test and f^(x) is predicted values . So irreducible error σ^2 becomes

np.mean((y_test-predicted_vales)**2).

```
     MSE
1   1027681.593658833
2   992941.9622374651
3   65629.23727603749
4   125495.58996677486
5   161386.47301324646
6   184225.94209629923
7   209275.6101171478
8   232550.34127494268
9   243983.67440152765
10   249108.85866023242
11   250581.9159504848
12   250037.60840552713
13   250425.4110565498
14   251775.96125411318
15   284643.3939176171
16   309177.3002605344
17   358232.9002872362
18   389633.4251271431
19   462079.81585015915
20   496335.46658081503
```

Irreducible Error

Irreducible error is the error that can't be reduced by creating good models. It is a measure the amount of noise in our data.

Irreducible error is

$\sigma 2$ = E[(f (x) – f^(x))2] -(Bias2 + Variance).

This is just calculated by using mean square error ,bias square and variance

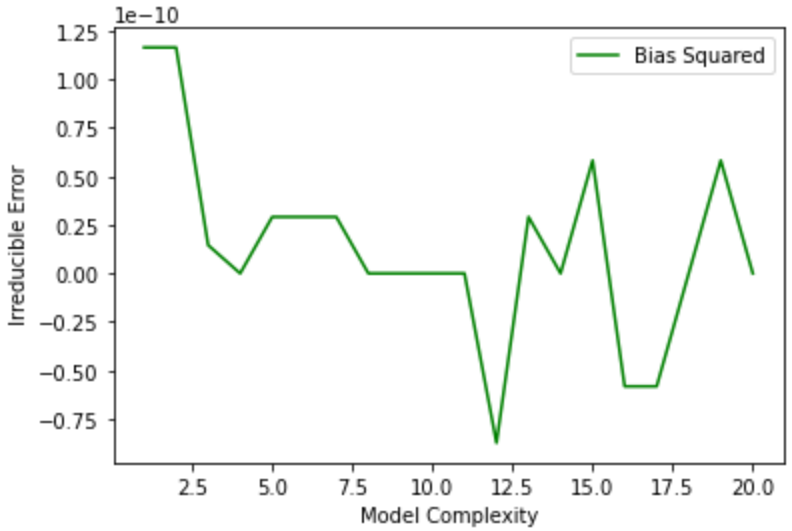
```
final_irrerror.append(final_error[degree-1]-(final_bias_square[degree-1]+final_var[degree-1]))
```

```
#tabulating error values
arr1=[i for i in range(1,21)]
print( "Mean square error", " IrreducibleError ")
for i in range(20):
print(i+1,' ',final_error[i],' ',final_irrerror[i])
```

Values we get

```
    Irreducible Error

1   1.1641532182693481e-10
2   1.1641532182693481e-10
3   1.4551915228366852e-11
4   0.0
5   2.9103830456733704e-11
6   2.9103830456733704e-11
7   2.9103830456733704e-11
8   0.0
9   0.0
10   0.0
11   0.0
12   -8.731149137020111e-11
13   2.9103830456733704e-11
14   0.0
15   5.820766091346741e-11
16   -5.820766091346741e-11
17   -5.820766091346741e-11
18   0.0
19   5.820766091346741e-11
20   0.0
```

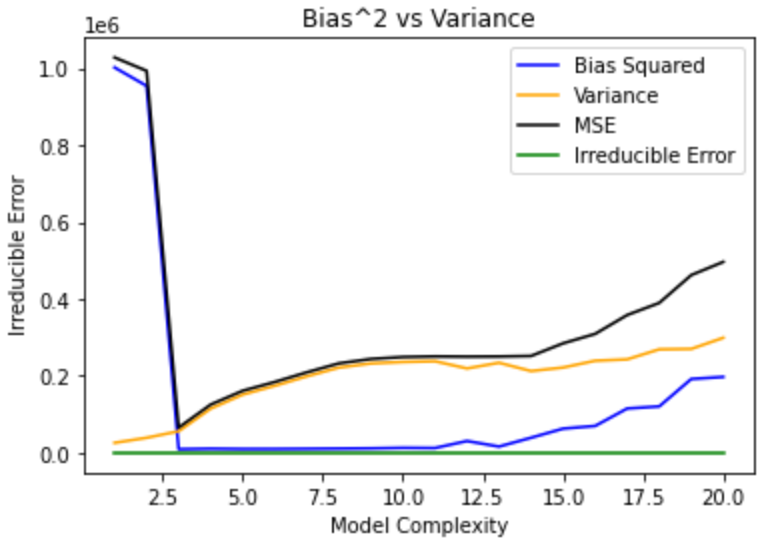


In the graph we can observe that the irreducible error shows no ready patterns.

Task 4

Plotting Bias 2 − Variance graph

Based on the calculated values bias,variance and total error . We will plot Bias ^2 vs variance graph.



Observation

- 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 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.

- The best fit possible (Minimum Total Error) is observed around a polynomial of degree 3
- Bias stays low from degree 3 and until around degree 10-11, but the variance increases steadily.
- Hence total error increases as well.

Polynomials of degree 1 & 2:

They are oversimplified and do not generalize the data well, leading to a high bias. Which is underfitting.

Polynomial of degree 3:

Here we can observe the lowest total error which gives best fitting of our data.

Polynomial of degree 4 and greater than 4:

- Here we can observe that Variance increases gradually.
- Also this is expected, since the no of features are high, it leads to overfitting of models over the trained data, and can be explained by regression models trying to fit the noise from the training data
- The flexibility provided by having more features leads to noise being captured which causes incorrect prediction for unseen data and there is relatively low bias as the more complex model is able to model the training set well.