Machine Data Learning

Team 65
Anvita Katipelly 2019115009
Stella Sravanthi 2019101101

Task-1: **Linear Regression**

Write a brief about what function does the method, LinearRegression().fit() performs.

Regression analysis is a modeling technique that predicts the relation between a dependent and independent variable. Linear Regression is the most mosted regression technique. Linear Regression, as mentioned, is the supervised learning algorithm. Basically, the goal of a regression model is to build a mathematical equation that defines y(dependent variable) as a function of the x(independent variable) where y is a Target/outcome variable and x is an input variable. Linear Regression fits a linear model with coefficient to minimize the residual sum of square between observed targets and targets predicted by linear approximation.
In a whole regression is about predicting a continuous quantity output.
Simple regression model helps to estimate the relationship between two quantitative variables.

X ———— Y

So that given an unknown data through linear regression continuous-valued output is predicted.

Y=W1*X+b
where,
- Y =Predicted value/Target Value
- X =Input
- W1 =Gradient/slope/Weight
- b =Bias

Now, this straight line equation gives optimized results when it(train data) gets the best fit which is possible on adjusting the slope (W1) and Bias (b).

From the Machine Learning library scikit-Learn, we import the LinearRegression class into our code as:

```
from sklearn.linear_model import LinearRegression
```

Which will be used for various linear and polynomial regressions to make predicts for out given training data set.

We then create an instance of the LinearRegression class as:

reg = LinearRegression( )

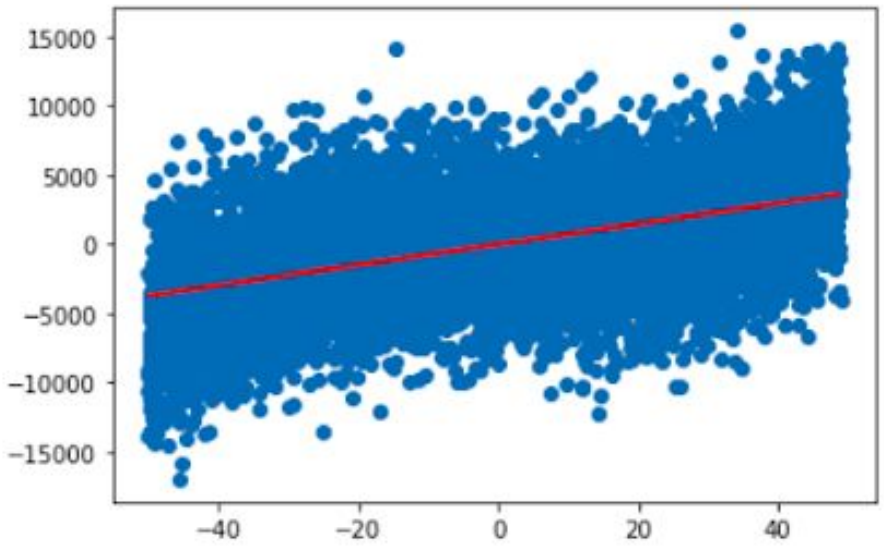
This helps create an instance in the reg variable for our existing data.
In order to call the model we created, we first used ***LinearRegression.fit()*** as:

reg.fit(x_poly,y_split[j])

It has two parameters, fit(x,y) which it fits into the model which follows the slope-intercept equation

$$Y=W1*X+b$$

It tries to give an optimal solution for our equation by estimating approximate values of weight(W1) and bias(b) to fit it into the straight line equation, by training the model with a fixed number of iterations on the supplied dataset. It takes the values of x_poly, which is a generated matrix for polynomials from degrees 1 to 20(both inclusive) and y_split, which is split into 10 data training sets.



For the given train dataset consisting of X,Y, the blue dots plotted represent the (X,Y) and LinearRegression().fit(X,Y) fitted the X,Y into a linear equation by returning the most appropriate values of intercept and coefficient for the given data. The red line represents the linear equation returned.

Task-2: Calculating Bias and Variance

The given two datasets Train and Test data.The data is loaded using pickle library that is pickle.load function.Further the train dataset into 10 different models of equal size.Then to calculate bia and variance of data set partitions for polynomial models where the corresponding degree is from 1 to 20.

The following are major steps taken to train the model for every dataset among all 10 training datasets:

1. Using PolynomialFeatures() and fit_transformation() method from sklearn.preprocessing class generates transformed versions of trained data.
2. Also apply this polynomial transformation to the test dataset using fit_transform() method itself.
3. Now, using LinearRegression() and fit() method from sklearn.linear_model class the train set is trained and best fit into function as (x,y) pairs input and output from train set.
4. Now, Predict the output values for the test set x_test_poly for each 10 training model and store the output on appending to list named poly.

```
x_poly=PolynomialFeatures(i, include_bias=False).fit_transform(x_split[j])
x_test_poly=PolynomialFeatures(i, include_bias=False).fit_transform(x_test)
reg=LinearRegression()
reg.fit(x_poly,y_split[j])
poly.append(reg.predict(x_test_poly))
#w+=1
w.append((numpy.mean(((poly)-y_test)**2)))
```

Bias is calculated using the formula:

$$Bias^2 = (E[\hat{f}(x)] - f(x))^2$$

```
bias.append(numpy.mean(numpy.abs(bias_square-y_test)))
```

Where,

bias_square here refers **`(numpy.mean(poly,axis=0))`**

Variance is calculated using the formula:

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

```
variance.append(numpy.mean(numpy.var(poly,axis=0)))
```

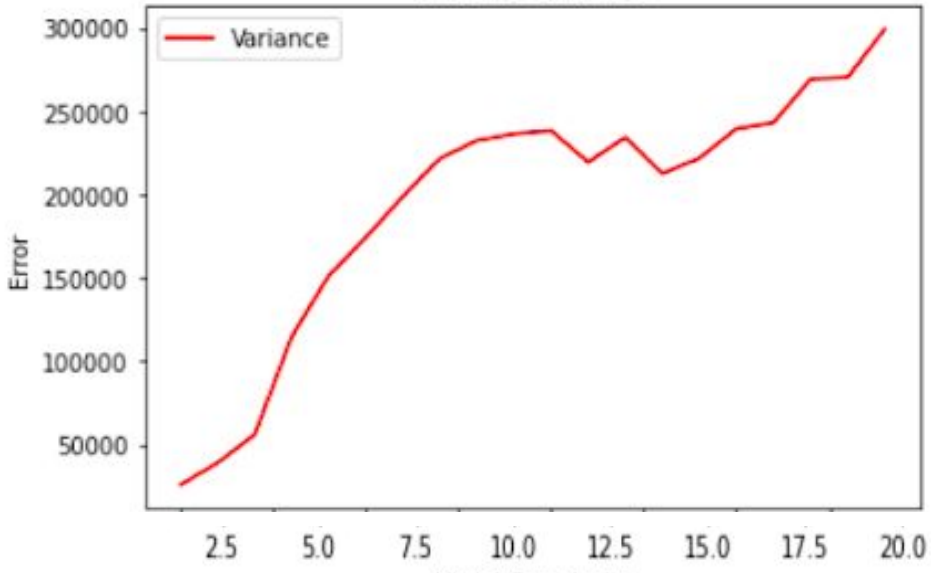
Table for bias and corresponding variance values:

Degree	Variance	Bias
1	25999.093010	819.717436
2	39105.833813	810.763391
3	56095.893210	68.510851
4	114907.291529	81.339711
5	151434.027857	78.958414
6	174226.744550	78.364801
7	198849.684616	86.916241
8	221551.968136	90.327445
9	232378.905956	92.450149
10	236185.285585	93.509377
11	238090.836956	91.163645
12	219356.681407	125.170874
13	234171.303968	93.628111
14	212545.070291	130.174210
15	221715.121438	166.459281
16	239355.637249	170.417979
17	242992.950957	236.714480
18	269050.631916	239.127012
19	270101.241167	304.866074
20	299029.012048	305.459406

Observations:
As the degree of the polynomial increases, the variance steadily increases uptill degree 11 and then dips a little at 12 and 14 degrees and then increases until 20.
With the increase in the degrees of the polynomial, the complexity and number of features increases. As the features increase, the model's variance increases which indicates that the

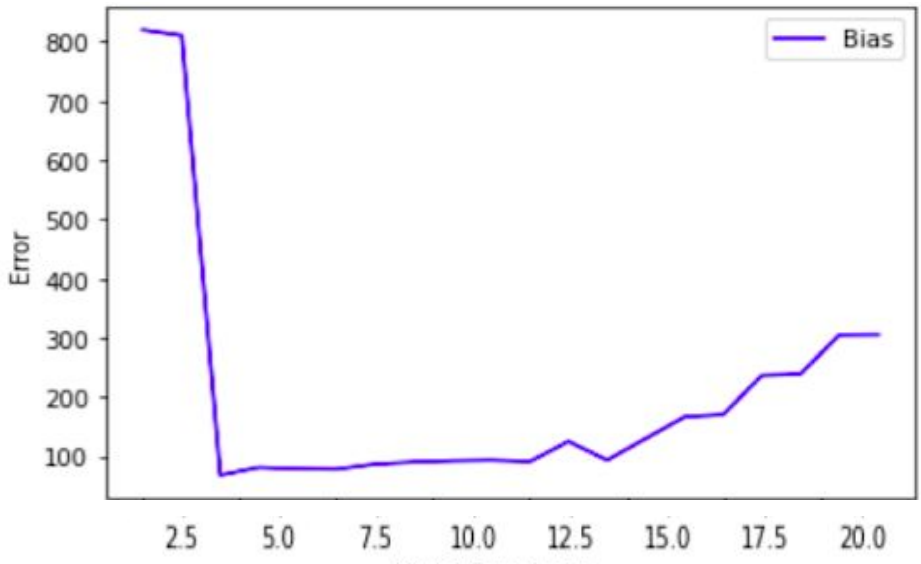
model does not generalize well. This is a result of the modelling of noise/unwanted distortion in the data which is not suited for deriving patterns. Hence the variability of model prediction seems to increase with increase in degree.

Variance Graph:



The bias of the model is highest at lowest degree polynomial and then sudden low in the bias in range 2-10 of degrees and then on the bias seems to increase. At the lowest degree polynomial, the high bias indicates less complexity of model due to less features and hence high bias due to oversimplification similarly in higher degrees after 10, the complexity increases and over complicated the model, thus these both cases are not ideal for generalization. The polynomials with degrees in range 3-11 seem ideal for generalization due to low comparative bias.

Bias Graph:



Task-3: **Measuring irreducible error**

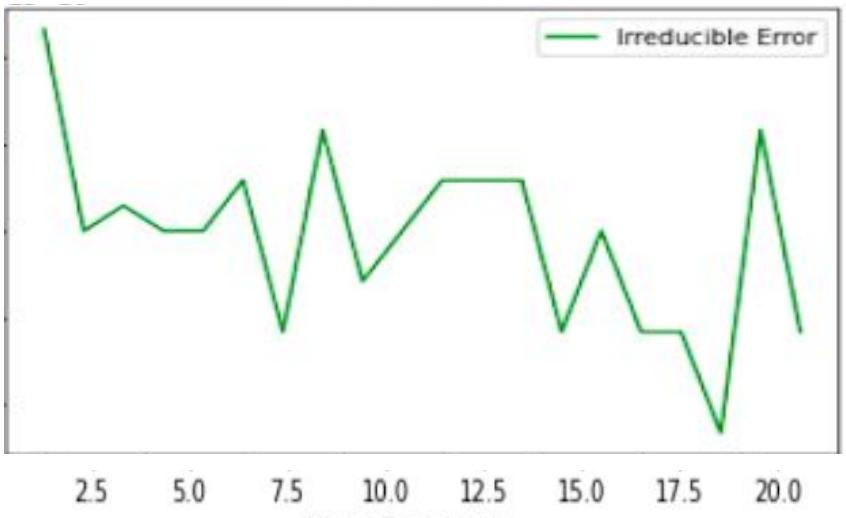
Table for variance, bias^2 and Irreducible Error

Degree	Variance	Bias^2	Irreducable Error
1	25999.093010	1.001683e+06	1.164153e-10
2	39105.833813	9.538361e+05	0.000000e+00
3	56095.893210	9.533344e+03	1.455192e-11
4	114907.291529	1.058830e+04	0.000000e+00
5	151434.027857	9.952445e+03	0.000000e+00
6	174226.744550	9.999198e+03	2.910383e-11
7	198849.684616	1.042593e+04	-5.820766e-11
8	221551.968136	1.099837e+04	5.820766e-11
9	232378.905956	1.160477e+04	-2.910383e-11
10	236185.285585	1.292357e+04	0.000000e+00
11	238090.836956	1.249108e+04	2.910383e-11
12	219356.681407	3.068093e+04	2.910383e-11
13	234171.303968	1.625411e+04	2.910383e-11
14	212545.070291	3.923089e+04	-5.820766e-11
15	221715.121438	6.292827e+04	0.000000e+00
16	239355.637249	6.982166e+04	-5.820766e-11
17	242992.950957	1.152399e+05	-5.820766e-11
18	269050.631916	1.205828e+05	-1.164153e-10
19	270101.241167	1.919786e+05	5.820766e-11
20	299029.012048	1.973065e+05	-5.820766e-11

The Irreducible error is the error in the model which cannot be reduced by any good model or feature of the model. It is the noise present in the data inherently. A part of this error can also be because of the float point error in python, which explains the difference of 10^-10 magnitude in error among various class functions. For zero irreducible error, like incases of degree 1,3,4,9 etc they are the perfect models among the class functions. But generally irreducible error is termed as stochastic error inherent to real systems, which cannot be reduced or explained but only identify them. Hence we cannot attribute it to any part of data but to the system itself. Hence the positive or negative or zero irreducible error is only the uncertainty because of randomness in the system or noise in the data without any attribution to bias or variance.

$$\sigma^2 = MSE - (\text{Var} + \text{Bias}^2)$$

Irreducible Error Graph:



Task 4: Plotting Bias^2 − Variance graph

Bias^2 is calculated through

```
meanBiasSquare.append(numpy.mean((bias_square-y_test)**2))
```

That is (predicted value - actual value)^2

Total error is calculated through

Which is MSE (Mean Squared Error)

```
w.append((numpy.mean(((poly)-y_test)**2)))
```

Calculated for every degree of polynomial model where complexity range is from 1 to 20 and stored in w.

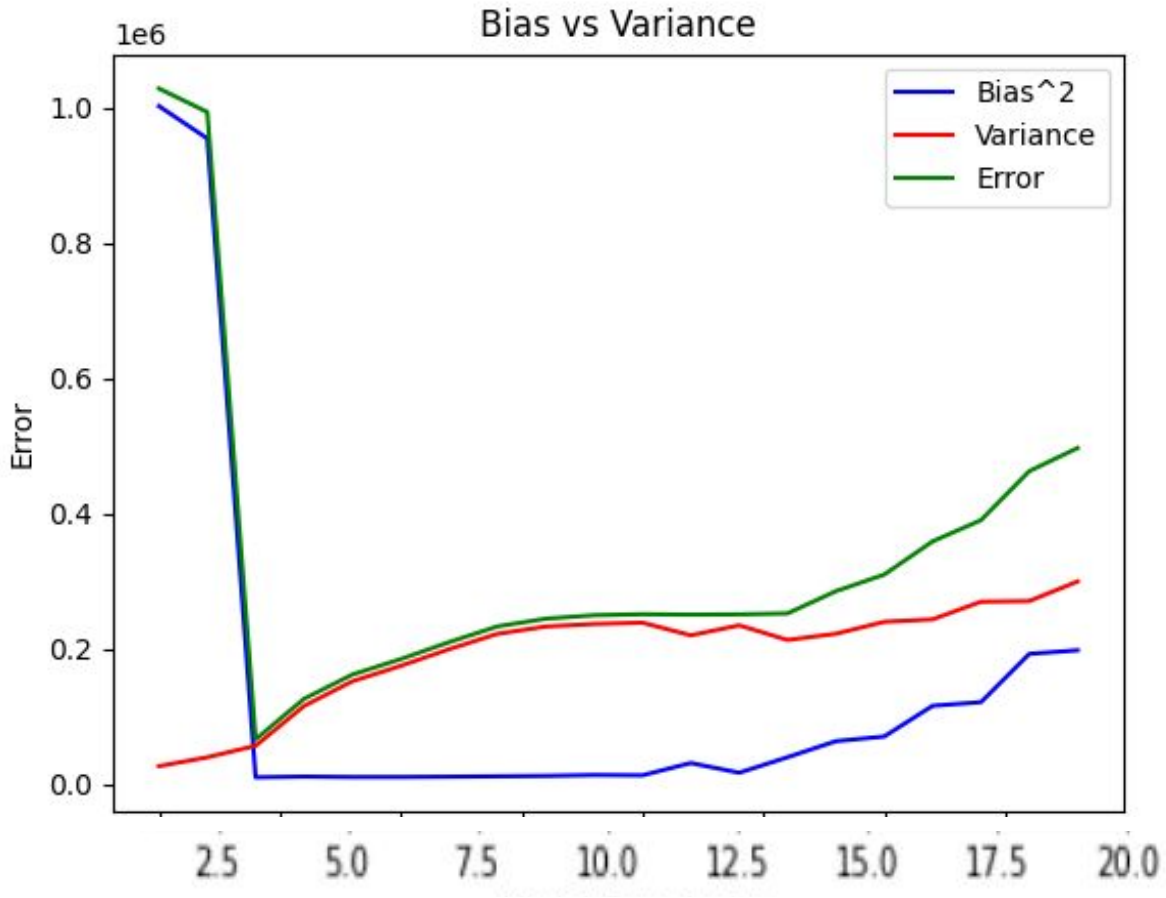
```
err.append(w[-1])
```

MSE of each degree from all 10 different models is stored as Total Error.

Table for Variance, Bias^2 and Total Error:

Degree	Variance	Bias^2	Total Error
1	25999.093010	1.001683e+06	1.027682e+06
2	39105.833813	9.538361e+05	9.929420e+05
3	56095.893210	9.533344e+03	6.562924e+04
4	114907.291529	1.058830e+04	1.254956e+05
5	151434.027857	9.952445e+03	1.613865e+05
6	174226.744550	9.999198e+03	1.842259e+05
7	198849.684616	1.042593e+04	2.092756e+05
8	221551.968136	1.099837e+04	2.325503e+05
9	232378.905956	1.160477e+04	2.439837e+05
10	236185.285585	1.292357e+04	2.491089e+05
11	238090.836956	1.249108e+04	2.505819e+05
12	219356.681407	3.068093e+04	2.500376e+05
13	234171.303968	1.625411e+04	2.504254e+05
14	212545.070291	3.923089e+04	2.517760e+05
15	221715.121438	6.292827e+04	2.846434e+05
16	239355.637249	6.982166e+04	3.091773e+05
17	242992.950957	1.152399e+05	3.582329e+05
18	269050.631916	1.205828e+05	3.896334e+05
19	270101.241167	1.919786e+05	4.620798e+05
20	299029.012048	1.973065e+05	4.963355e+05

Underfitting :
In supervised learning, the patterns are not derived from the given data, as the data for the model to build upon is very less. As a result these models are very “simple” to capture “complex” patterns in data like that of linear regression. Models with high bias and low variance are said to be underfit, because they have too few features and are less flexible in modeling.
Overfitting:
In supervised learning, the patterns are not derived from the given data, as the data for the model to build upon is data as well as noise present in it. AS a result these models are more complex than the underlying data and hence end up modeling noise. Models with low bias and high variance are said to be overfit, because they have too many features and are more flexible in modelling.
So we need to find a right fit between the two extremes of underfitting and overfitting, with the goal of the algorithm being minimization of bias and variance for good prediction performance of the model.
But we know that:
Bias increases as variance decreases
Variance increases and bias decreases
Hence we need the bias-variance tradeoff to play between the two of them and to choose a way to configure them. The Bias-Variance tradeoff is useful in our model, which is a predictive machine learning model.



Observations:
From the graph above, we can tell that the general trend despite a few exceptions, is that

- Bias decreases uptil degree 10 and then increases
- Variance Increases till degree 10 and then decreases
- Total Error increases

For the model of lower degree:
There is a very high bias and the lowest variance of all, we can tell that the model is underfitted, and that this model has less features than required. The model is not performing well on the train data since the model has less features and is simple, hence cannot capture the features of training data. These models generalize well because the low variance is consistent, but the predictions made are not accurate.

For models of higher degree:
As the degree increases the complexity increases. These models generally have less bias and more variance, hence are overfit. The models start extracting more patterns from the training sets and hence fits the data well. Which is proved by the fact that the low and decreasing bias represents close prediction of actual values. Hence these models are said to be more flexible.

It is seen that the variance keeps on increasing because the model extracts more information than required, because of which it starts modeling the noise as well. These models do not generalize well because the variation keeps on increasing leading to more error.

The bias-variance trade-off tells us that the model with minimum total error is the ideal model of all. **Hence the model with degree 3 is the optimum model**, as the **error in prediction is least** for it.