MDL Assignment-1 Report

LinearRegression().fit()

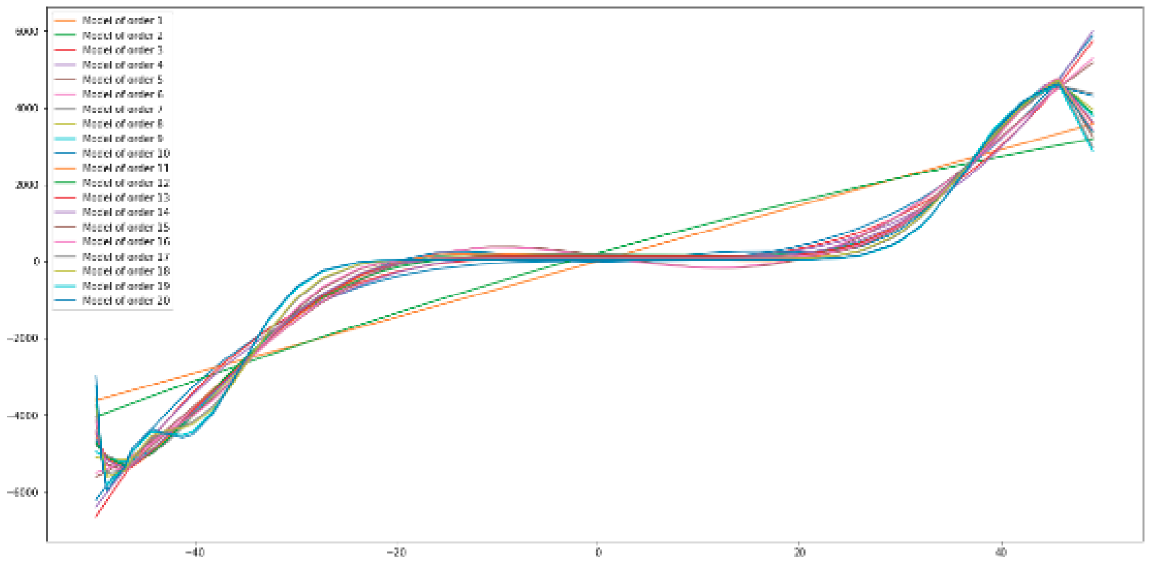
The fit() function serves to set the coefficients for each of the parameters and to set the intercept in an attempt to give us a 'line' (linear combination of input parameters) that best models our training data. We can easily find the coefficients and the intercept by running the following line of code:

```
print(model.coef_, model.intercept_)
"""
Returns the following: [[  0. -16.56603015 -0.1811364 0.05844285]] [33.79437874]
The above was for a polynomial regression model of degree 3.
"""
```

The output of the above function gives us the equivalent of the polynomial:

$$f(x) = 0.05844285x_3 - 0.1811364x_2 - 16.56603015x_1 + 33.79437874$$

In this case calling fit() set the above weights and intercept values, and stored them in the linear model. The set of x_i represents the input parameters that are given to the linear regression model. The fit function seeks to create the model (line in our case) of best fit by minimising the total square error of the training data by gradient descent. LinearRegression doesn't have support for polynomial regression by default and hence, we adapt it to polynomials by using the PolynomialFeatures class that converts a singular input into a vector with powers of the input (ex: converts x into $[1, x, x^2, x^3]$ for order 3). Linear Regression now perceives each of these inputs as separate linear parameters, and obtains weights (in the example stated, $x_3 = x^3, x_2 = x^2, x_1 = x$) by gradient descent. Therefore, PolynomialFeatures followed by linear regression is equivalent to polynomial regression.



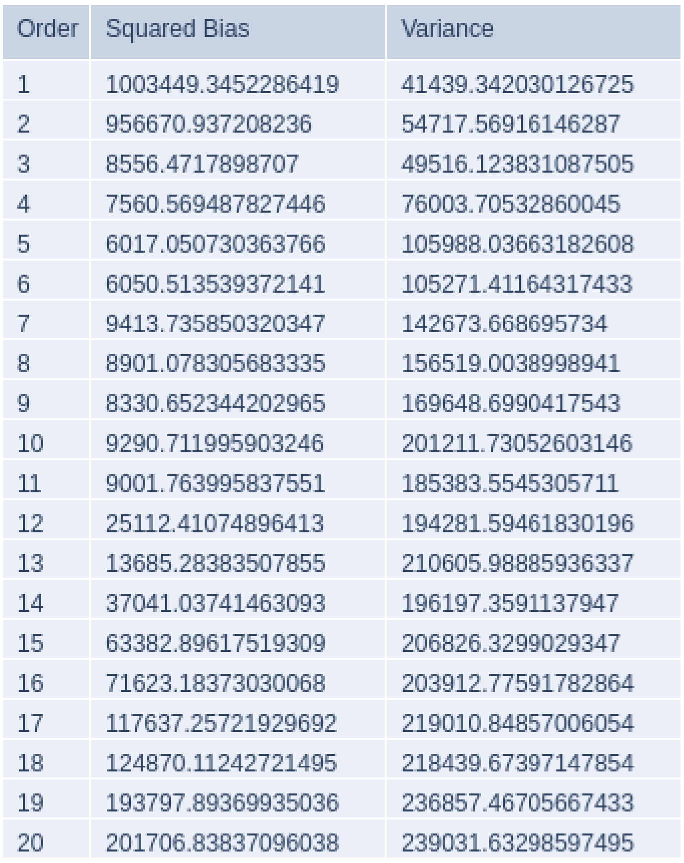
One estimator from each polynomial class that is created after calling fit() on a realisation of the training dataset

Squared-Bias vs Variance

To calculate the bias^2 variance and mean squared error, we take the mean over all the estimators of a single polynomial class $\hat{f}(x)$ and calculate the values of bias^2, variance and mean squared error for each point on each estimator with respect to $\hat{f}(x)$ and then take the mean over all these points to obtain single values.

As the order changes, squared bias decreases and variance increases. Bias (actually squared bias but written as bias for simplicity) is a measure of the simplicity of our model. Since it is a measure of how far our average estimator value is from the actual function output (test data y) it tells us if our model is 'biased' towards being simple and doesn't fit the data well. As we increase the degree of the polynomial that we use for our estimator, bias will decrease as the model will fit the data better (and become more complex).

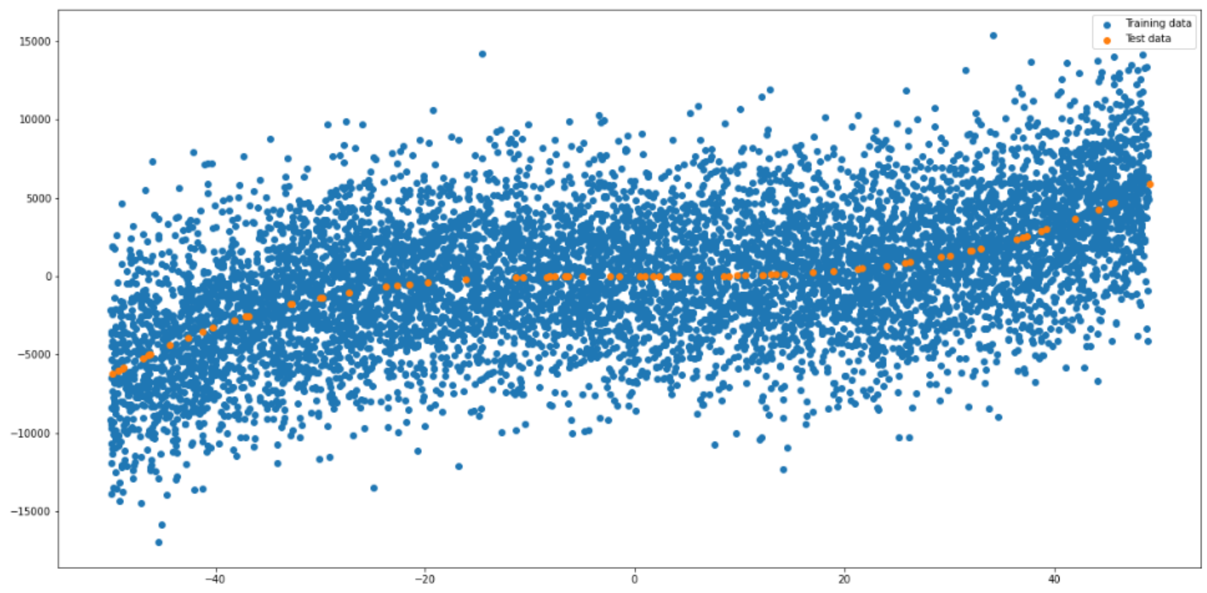
Variance is a measure of how far a model based on a particular realisation of the training data differs from the average estimator of that polynomial order. For lower order polynomials, all estimators will be centred (not vary much from each other) and hence will have low variance. As the model becomes more complex, the realisation of the training data has a bigger impact and hence variance increases.



Order	Squared Bias	Variance
1	1003449.3452286419	41439.342030126725
2	956670.937208236	54717.56916146287
3	8556.4717898707	49516.123831087505
4	7560.569487827446	76003.70532860045
5	6017.050730363766	105988.03663182608
6	6050.513539372141	105271.41164317433
7	9413.735850320347	142673.668695734
8	8901.078305683335	156519.0038998941
9	8330.652344202965	169648.6990417543
10	9290.711995903246	201211.73052603146
11	9001.763995837551	185383.5545305711
12	25112.41074896413	194281.59461830196
13	13685.28383507855	210605.98885936337
14	37041.03741463093	196197.3591137947
15	63382.89617519309	206826.3299029347
16	71623.18373030068	203912.77591782864
17	117637.25721929692	219010.84857006054
18	124870.11242721495	218439.67397147854
19	193797.89369935036	236857.46705667433
20	201706.83837096038	239031.63298597495

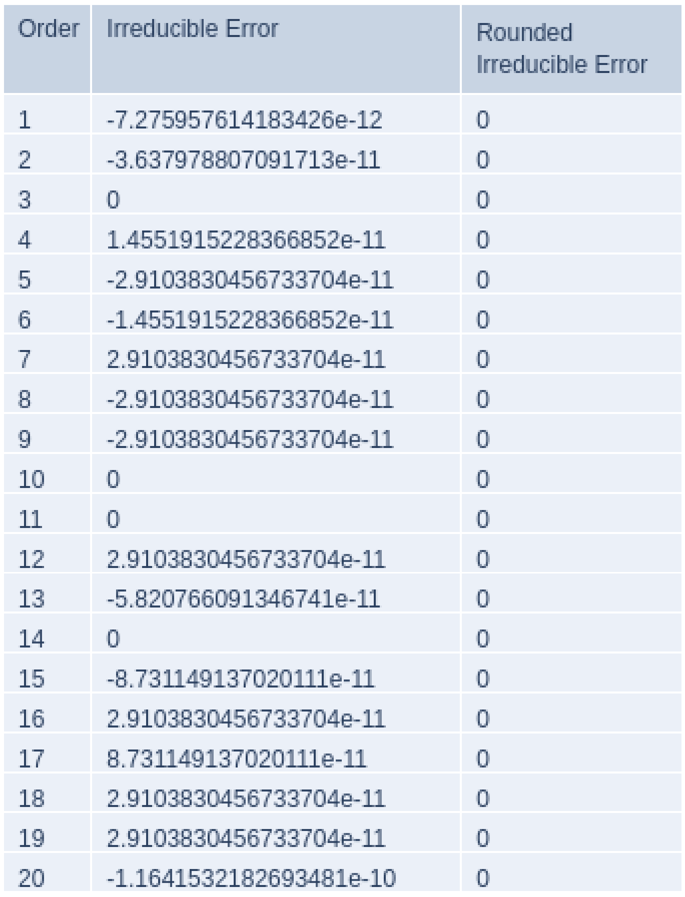
The strange rise in bias as the order gets higher is due to inherent shortcomings in PolynomialFeatures and LinearRegression. Double precision arithmetic gives us 16 digits of precision which isn't nearly enough to account for the matrix inversions and other calculations required. This means that our weight set will have noise, making it faulty and susceptible to errors. PolynomialFeatures and linear regression is only reliable till around order 6-7 and after that starts to break. It may also be a result of heavy overfitting of higher order polynomials, and hence the average model may vary even further from the test data (the model overfits the training data and hence misses the mark on the testing data).

Irreducible error values



Training and test data scatterplot

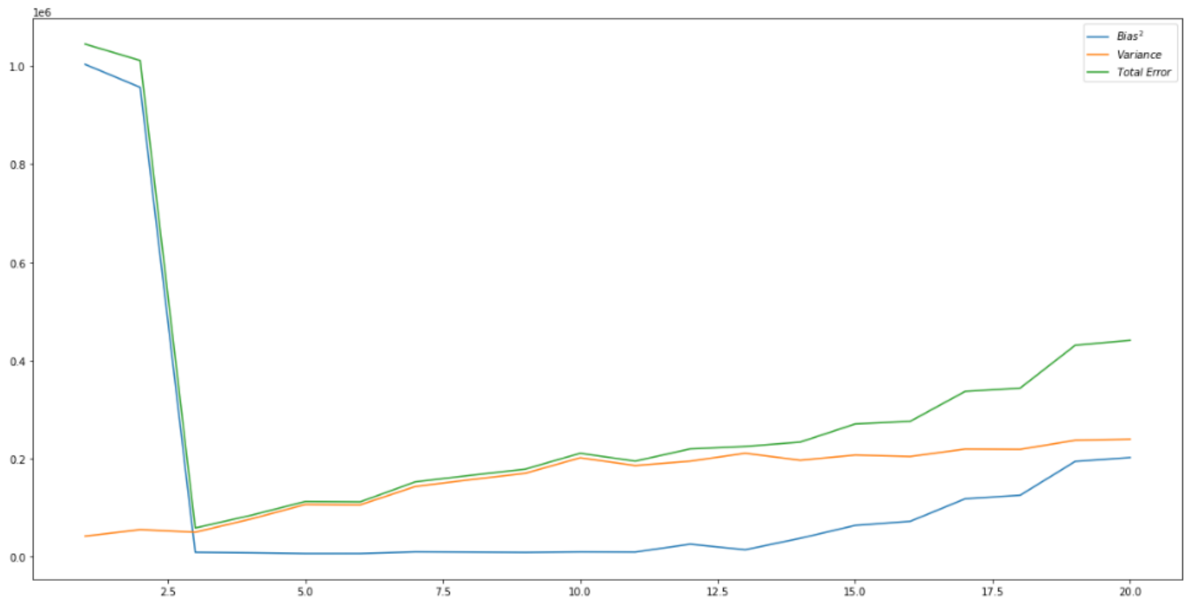
As we can see in the below image, the irreducible error values hardly change. Furthermore, they are very close to 0 (some are negative due to python precision errors). As we can see in the rounded error (rounded to 10^{-9}) the errors are for realistic purposes 0. The irreducible error does not change, as it is a property of the test data. We assume that noise is normally distributed, and our irreducible error $E_{irreducible} = \sigma^2$. This value will be consistent across realisations of the testing data, and across polynomial orders. Looking at the scatterplot above, we can see that the test data (in orange) seems to exactly fit a polynomial, which leads us to believe that there was minimal noise introduced into the test data. This explains why the observed irreducible error is constant, and very close to 0.



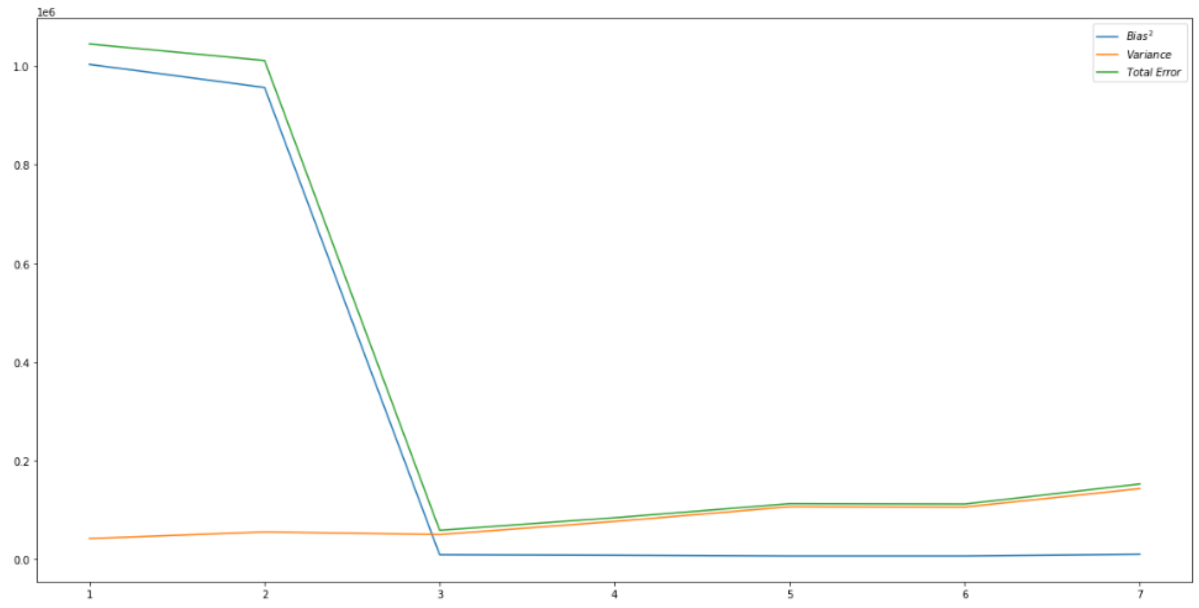
Order	Irreducible Error	Rounded Irreducible Error
1	-7.275957614183426e-12	0
2	-3.637978807091713e-11	0
3	0	0
4	1.4551915228366852e-11	0
5	-2.9103830456733704e-11	0
6	-1.4551915228366852e-11	0
7	2.9103830456733704e-11	0
8	-2.9103830456733704e-11	0
9	-2.9103830456733704e-11	0
10	0	0
11	0	0
12	2.9103830456733704e-11	0
13	-5.820766091346741e-11	0
14	0	0
15	-8.731149137020111e-11	0
16	2.9103830456733704e-11	0
17	8.731149137020111e-11	0
18	2.9103830456733704e-11	0
19	2.9103830456733704e-11	0
20	-1.1641532182693481e-10	0

Irreducible error for each polynomial degree

Bias-Variance Tradeoff



Graph of squared bias, variance, and total error with respect to polynomial order.



Same graph but for low orders only

From the above graphs, we can conclude the following:

- **High Bias, Low Variance -**
  When a model (polynomial order) has high bias and low variance, it 'underfits' the data. We can see this as models with low orders have high biases. This is because they tend to simplify the estimator that are very similar to each other and result in underfitting.
- **High Variance, Low Bias -**
  When a model (polynomial order) has high variance and low bias, it 'overfits' the data. We can see this as models with high orders have high variances. This is because they tend to create complex estimators that vary a lot from each other and result in overfitting.
- **Ideal model -**
  As we can see, the lowest error (mean squared) is observed at order 3. This is not only for this particular run of the training data split, order 3 gives the best results consistently. However orders from 3-6 have acceptable errors, and could all potentially be used to estimate the function.
- **Dataset -**
  The provided test set lacks noise, and hence has irreducible error very close to 0 (the total error graph is equal to the bias graph added to the variance graph). We can also reasonably conclude that the data given is of order three with respect to the input parameter, as the lowest error is

obtained when our order is three, and the shape of the graph also indicates that $y = ax^3 + bx^2 + cx + d$.