Bias-Variance-Tradeoff

Calculation of the bias and variance after training two models.

Introduction

The **prediction error** for any machine learning algorithm can be broken down into three parts:

Bias error

Bias is the difference between the average prediction of our model and the correct value which we are trying to predict.

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

Variance error

Variance is the variability of a model prediction for a given data point.

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

Irreducible error

Irreducible error cannot be reduced regardless of what algorithm is used. It arises due to the chosen framing of the problem and may be caused by factors like unknown variables that influence the mapping of the input variables to the output variables.

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

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

$$Err(x) = \text{Bias}^2 + \text{Variance} + \text{Irreducible Error}$$

Bias-Variance Trade-Off

Bias–variance problem is the conflict in trying to simultaneously minimize these two sources of error that prevent supervised learning algorithms from generalizing beyond their training set.

If the model is too simple and has very few parameters then it may have high bias and low variance. The model is unable to capture the underlying pattern of the data. This is known as ***underfitting***.

It may occur when we

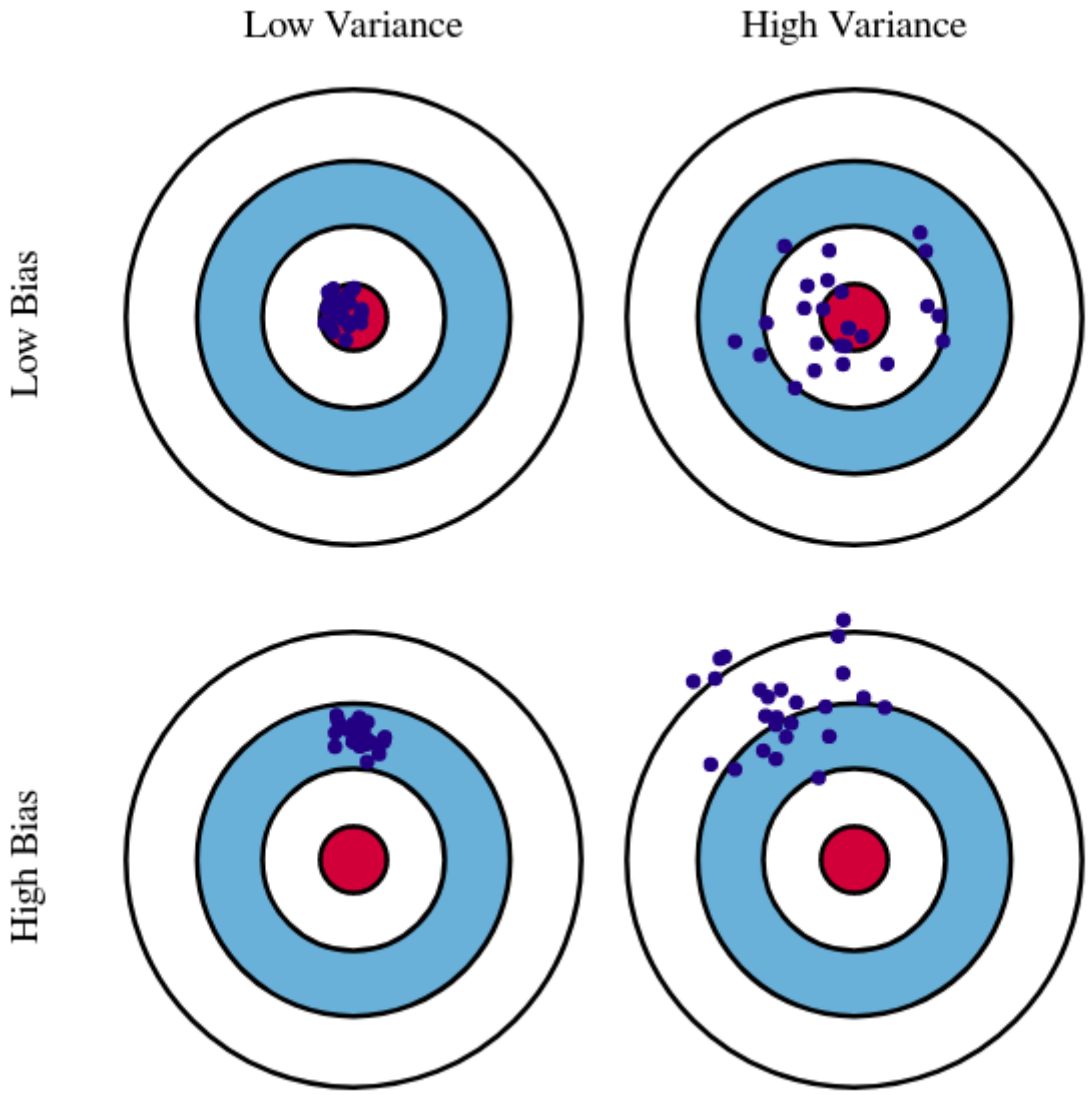
- Use less amount of data

- Try to build a linear model with a nonlinear data

On the other hand if the model has large number of parameters then it's going to have high variance and low bias. The model captures the noise along with the underlying pattern in data. This is known as ***overfitting***.

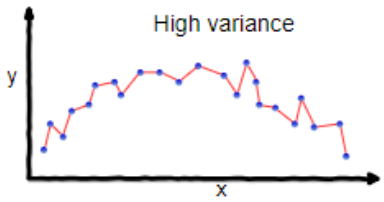
It may occur when we

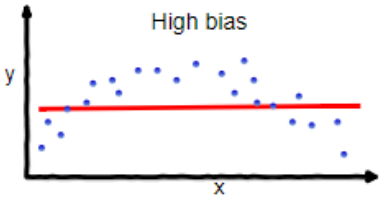
- Use a small dataset and the model satisfies these datapoints exactly
- Use an unnecessarily complex algorithm

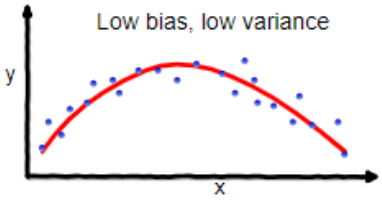


This tradeoff in complexity is why there is a tradeoff between bias and variance. An algorithm can't be more complex and less complex at the same time.

> We need to find the right/good balance without overfitting and underfitting the data.







overfitting **underfitting** **Good balance**

Getting Started

Unzip the folder, run the scripts and view the saved tables and graphs for question 1 and 2 in the current directory.

```
$ unzip 65_assgn1.zip
$ cd 65_assgn1
$ pip install -r requirements.txt
$ python3 q1.py  # Graph and table will be displayed and saved in current
directory
$ eog graph1.png
$ eog table1.png
$ python3 q2.py  #Graph will be displayed and saved in the current
directory
$ eog graph2.png
$ eog table2.png
```

Alternately,

```
$ unzip 65_assgn1.zip
$ cd 65_assgn1
$ make all
$ make seeOne  # Data visualization for question 1
$ make seeTwo  # Data visualization for question 2
$ make clean
```

Libraries Used

```
# For data handling
import pickle
import numpy as np
import pandas as pd
# To generate training and testing data splits
from sklearn.model_selection import train_test_split
# For data visualization
```

```
import matplotlib.pyplot as plot
# To generate polynomial data and train the models
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
```

Question 1

Data Resampling

- 5000 entries of the form (x_i , y_i) are present are loaded using the `pickle` library, in numpy format.

```
file = open('Assignment/Q1_data/data.pkl', 'rb')
data =  pickle.load(file)
file.close()
```

- 90:10 split of data as training set and testing set is done once using the `train_test_split` from `sklearn`.

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.1)
```

- The training data is then reshuffled using the `np.random.shuffle` function.

```
training_data = np.column_stack((X_train, Y_train))
np.random.shuffle(training_data)
X_train = training_data[:, 0]
X_train = X_train[:, np.newaxis]
Y_train = training_data[:, 1]
```

- The training data is then split into 10 subsets using the `np.array_split` funtion. This is done because inorder to calculate the variance we need multiple realisations of the model. An easy way to achieve this is to randomly divide the training set (if feasible) into multiple subsets (here, `10`), so that we have `10` different realisations of model.

```
X_train_split = np.array_split(X_train, 10)
Y_train_split = np.array_split(Y_train, 10)
```

- `PolynomialFeatures(i)` is used to raise `x` data points to powers from 0 to `i` to generate data for polynomials of degree `i`.

Bias - Variance Tradeoff

- We have repeated the entire model building process `10` times for each polynomial.

- The outer loop chooses the polynomial and inner loop chooses 1 of the `10` training sets.

Calculating variance

- The variance is how much the predictions for a given point vary between different realizations of the model. This is calculated and averaged out for each polynomial as follows,

```
    np.mean(np.var(poly_prediction, axis = 0))
```

- Here, `poly_prediction` is a [10, 500] matrix containing the predicted outputs using the 10 models for each data point.
- `axis = 0` specificies that we traverse the matrix column-wise to obtain different model predictions for a single test data point.

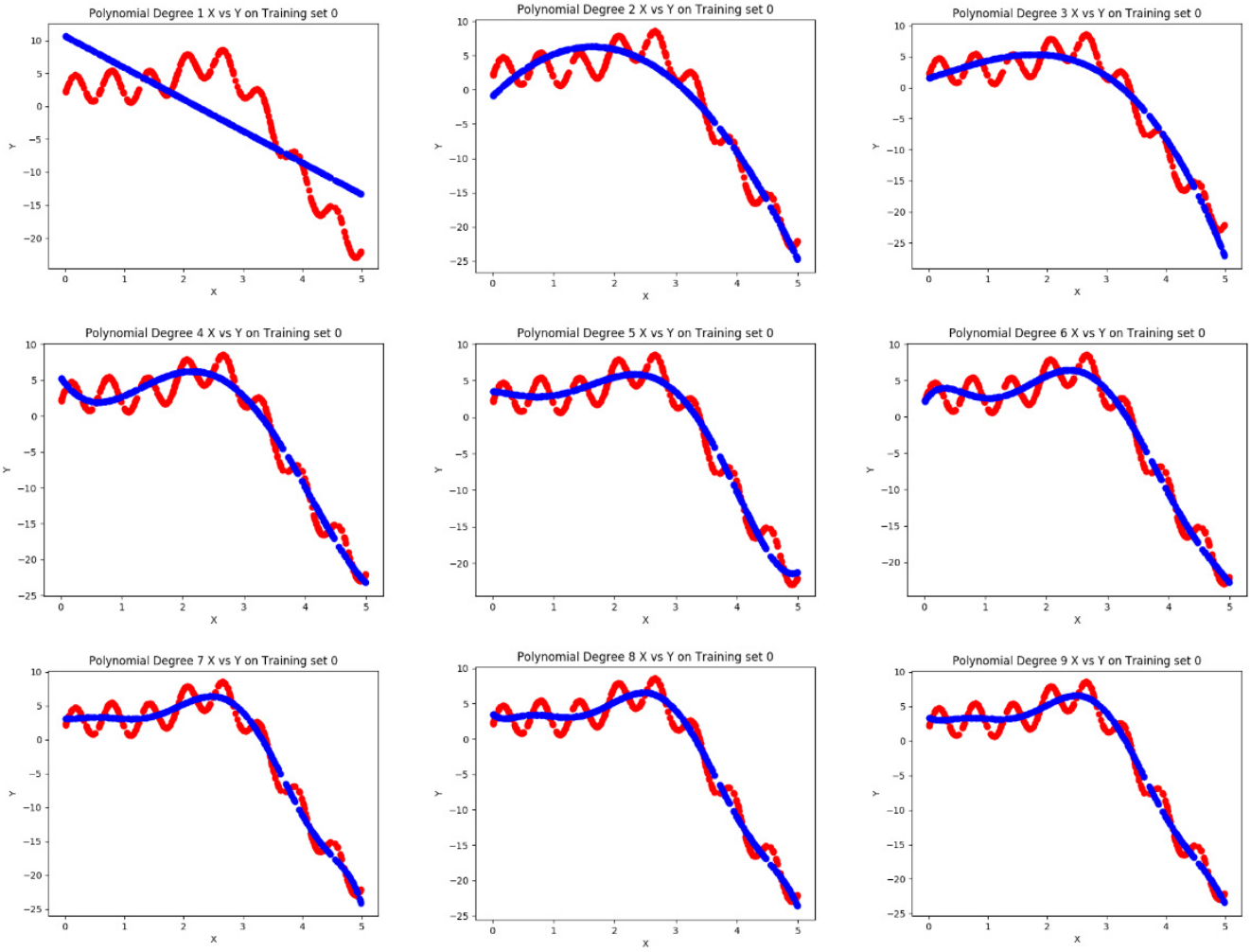
Calculating bias

We subtract the Y_test values with the predicted values of X_test on each model. We further average it out for each polynomial.

```
    bias = abs(np.mean(poly_prediction, axis = 0) - Y_test) # 500 bias
values - takes mean of all same polynomial models at a test point
```

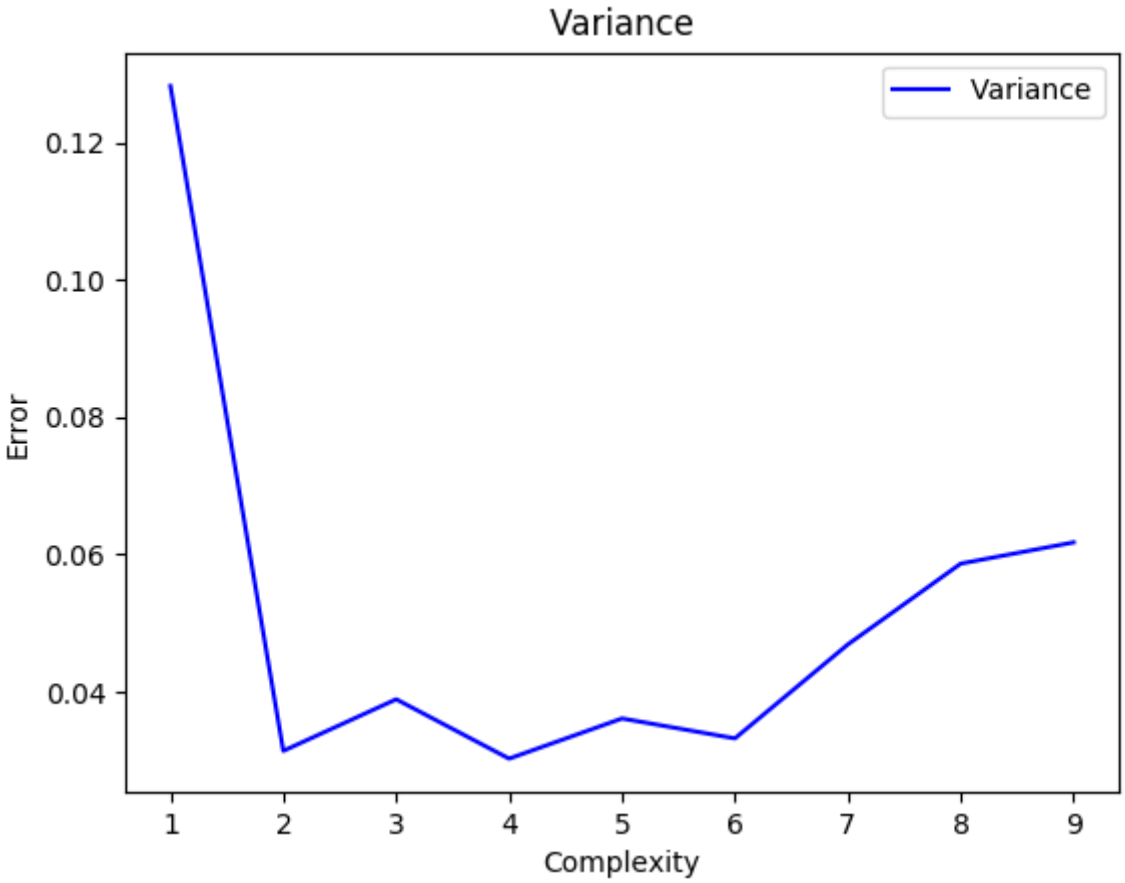
- Here, `Y_test` represents the correct output value for the input data point.

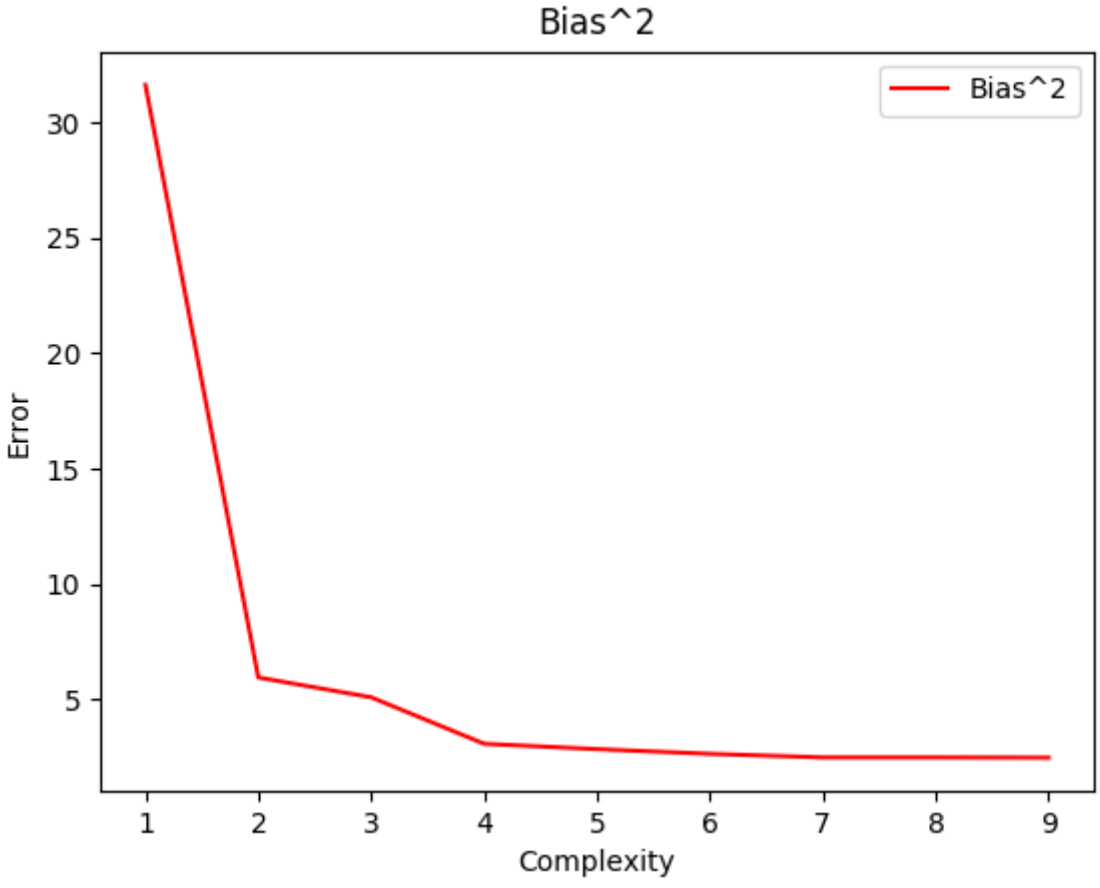
Listed are the plots for the models obtained using the `LinearRegression()` function using the first randomly generated test set for the respective polynomials.

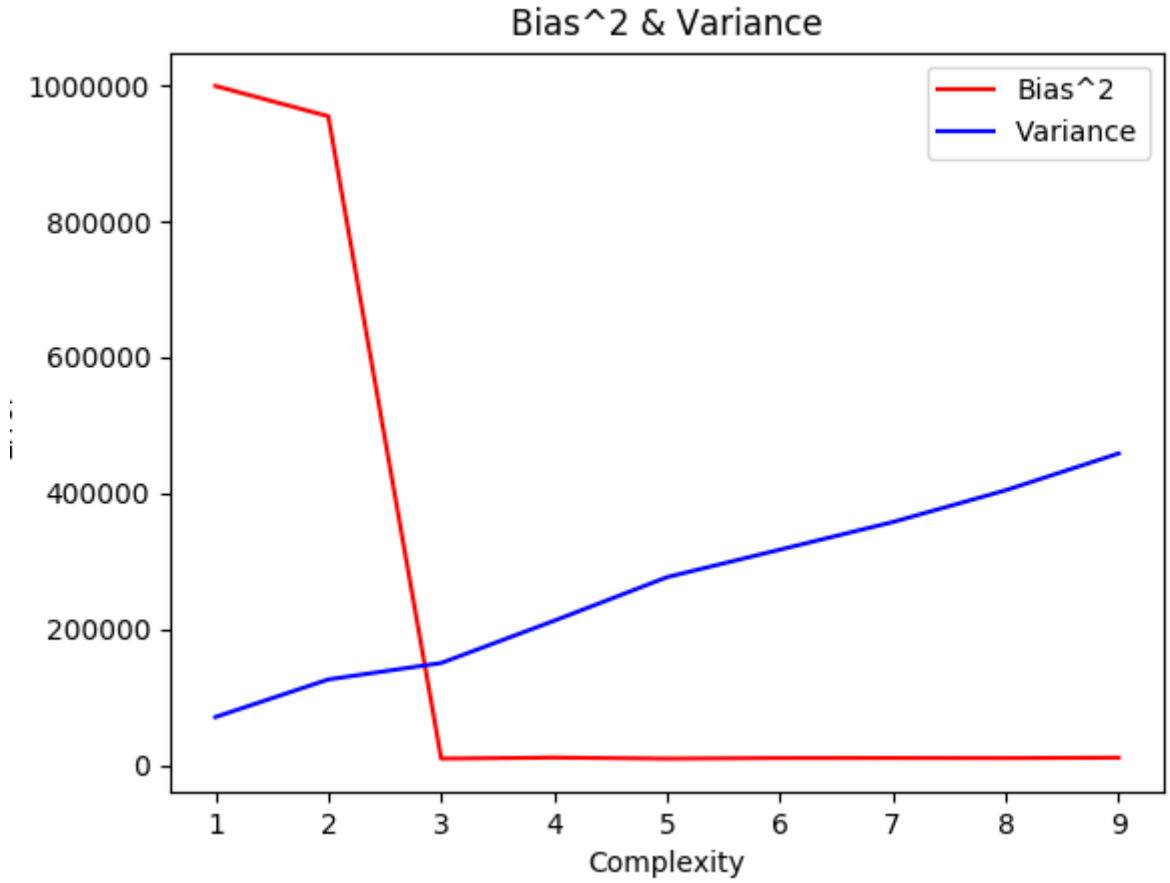


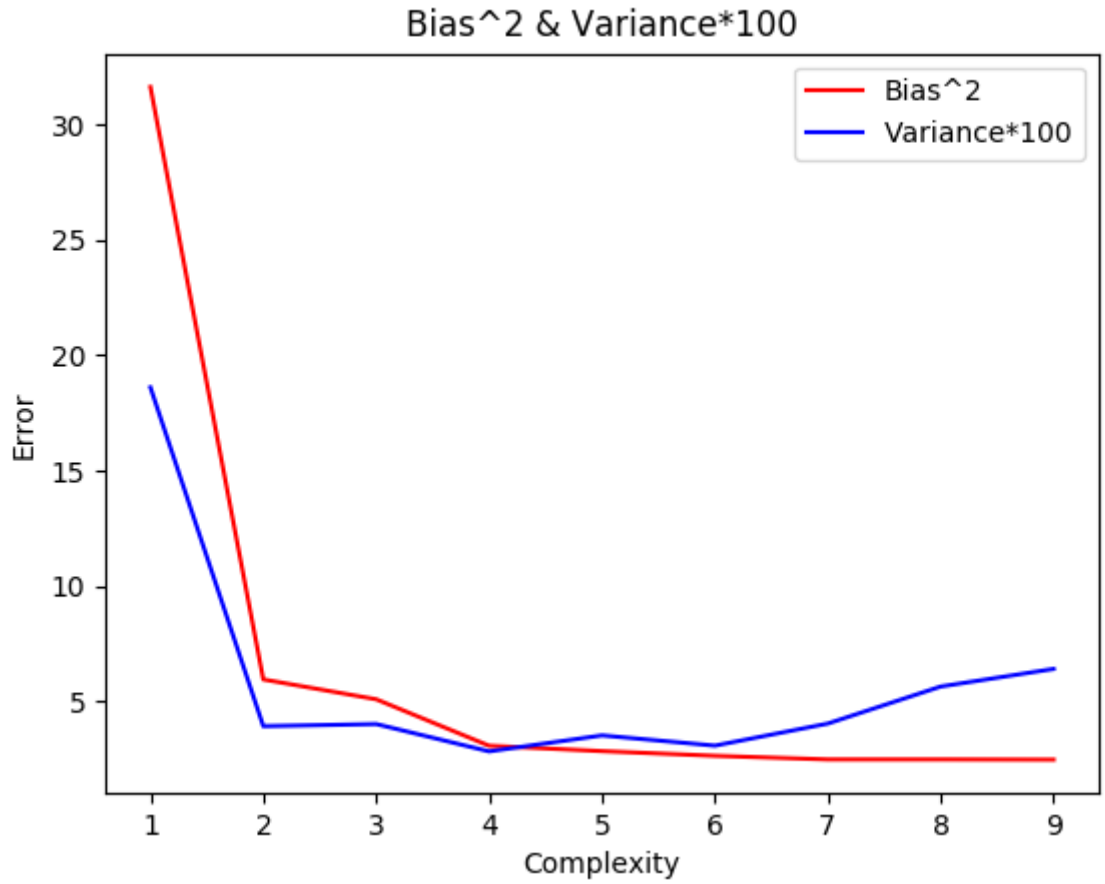
Bias & Variance

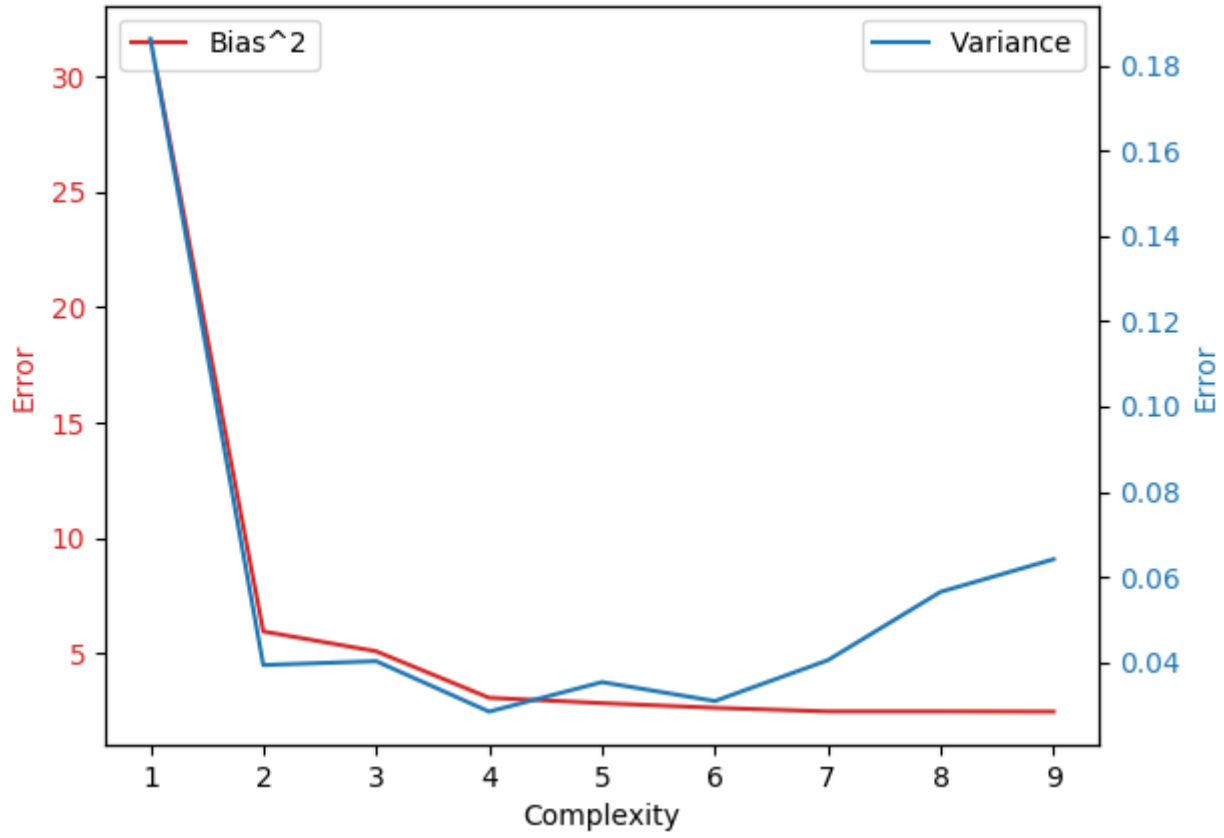
Degree	Bias	Bias^2	Variance
1.0	5.623981568730672	31.629168685422307	0.18616847445109744
2.0	2.441961412014718	5.963175537768915	0.039393099199810364
3.0	2.259114876644666	5.1036000258772445	0.04030186532825462
4.0	1.7579000241868794	3.0902124950362313	0.028468139050103574
5.0	1.6905340527685588	2.8579053835700887	0.03539104325959967
6.0	1.6292798776280797	2.6545529196437703	0.03094858317097518
7.0	1.580640554422373	2.4984245622846664	0.04051475126273183
8.0	1.580739404291244	2.498737064279037	0.05656014540942231
9.0	1.5777097081160691	2.4891679230836923	0.06419273408252091


Variance
Variance
0.12
0.10
0.08
0.06
0.04
Error
1
2
3
4
5
6
7
8
9
Complexity


Bias^2
Bias^2
30
25
20
15
10
5
Error
1
2
3
4
5
6
7
8
9
Complexity


Bias^2 & Variance
Bias^2
Variance
1000000
800000
600000
400000
200000
0
1
2
3
4
5
6
7
8
9
Complexity


Bias^2 & Variance*100
Bias^2
Variance*100
30
25
20
15
10
5
Error
1
2
3
4
5
6
7
8
9
Complexity



Observervations

The **bias decreases** and the **variance increases** as the *degree of the polynomial increases*.

That is, with increase in the number of features, or complexity (essentially increasing the degree of the model), the model becomes **more flexible**. It extracts more information from the training sets and fits the data well. The values it predicts are close to the actual values, giving lesser bias.

However, as it extracts more information, it also starts **memorizing the noise**. It does not generalize well and gives a lot of variation on data sets that it has not seen before, ie variance increases. Some (minor) bumps are observed in the variance graph but the overall trend is still increasing.

For lower degree polynomial models, we observe high bias and low variance. The model may not be able to perform well even on existing training data since the polynomial is so simple that it cannot capture the features of the training data. The variance is low since the model may not give accurate predictions but it does so consistently, generalizing well.

This is in accordance with the Bias-Variance tradeoff where an optimum model is reached when total error is minimised.

Question 2

Data Resampling

- 20 subsets of 400 entries of X datapoints each are loaded as the training set and 1 set of 80 entries of X datapoints is loaded as the testing set using the `pickle` library, in numpy format. Corresponding Y

sets are loaded as well.

```
file = open('Assignment/Q2_data/X_train.pkl', 'rb')
X_train =  pickle.load(file)
file.close()
file = open('Assignment/Q2_data/Y_train.pkl', 'rb')
Y_train =  pickle.load(file)
file.close()
file = open('Assignment/Q2_data/X_test.pkl', 'rb')
X_test =  pickle.load(file)
file.close()
file = open('Assignment/Q2_data/Fx_test.pkl', 'rb')
Y_test =  pickle.load(file)
file.close()
```

- The training data has 20 subsets. This is now used to calculate the variance by generating multiple realisations of the model.
- Now, the outer loop selects the degree of the polynomial and the inner loop chooses the training set and a model is generated for each traning set.

```
for i in range(1, 10): # Choosing polynomial power
    poly_prediction = []
    poly = PolynomialFeatures(i)
    X_test_poly = poly.fit_transform(X_test)

    for j in range(0, 20): # Choosing training set
        x_train = X_train[j]
        x_train = x_train[:,np.newaxis]

        X_poly = poly.fit_transform(x_train)
        linearRegressor.fit(X_poly, Y_train[j])  # Training model on subset

        X_test_poly = poly.fit_transform(X_test)
        poly_prediction.append(linearRegressor.predict(X_test_poly)) #
Predicting test set output on model
```

- The prediction of test set data on each model is appended to a list for that polynomial. This is used to calculate variance and bias further.

Bias - Variance Tradeoff

- We have repeated the entire model building process `20` times for each polynomial.

Calculating variance

The variance is how much the predictions for a given point vary between different realizations of the model. This is calculated and averaged out for each polynomial as follows,

```
np.mean(np.var(poly_prediction, axis = 0))
```

- Here, `poly_prediction` is a [20, 80] matrix containing the predicted outputs using the 20 models for each data point in the testing set of 80 points.
- `axis = 0` specificies that we traverse the matrix column-wise to obtain different model predictions for a single test data point.

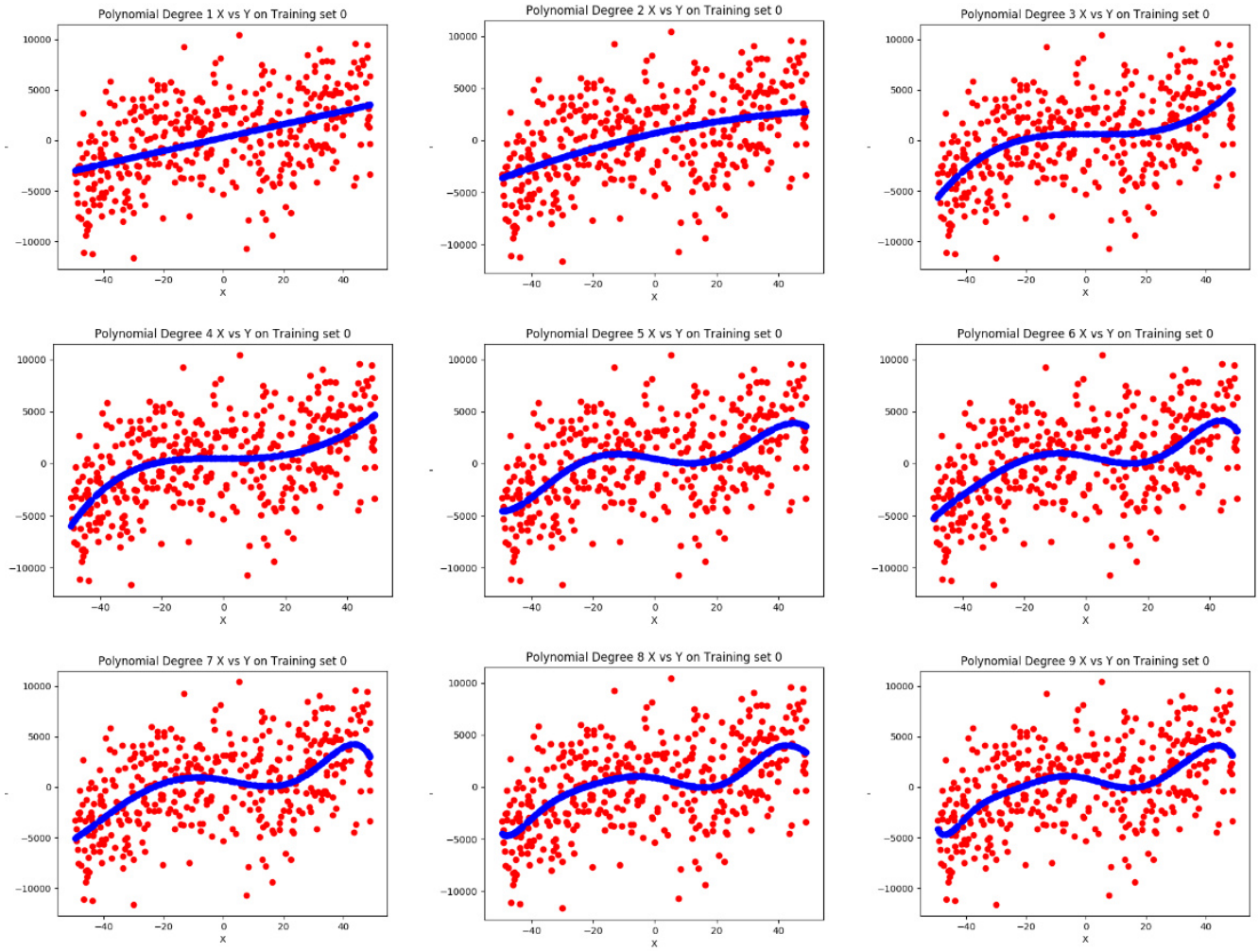
Calculating bias

This is calculated and averaged out for each polynomial as follows,

```
(np.mean(poly_prediction, axis = 0) - Y_test)**2
```

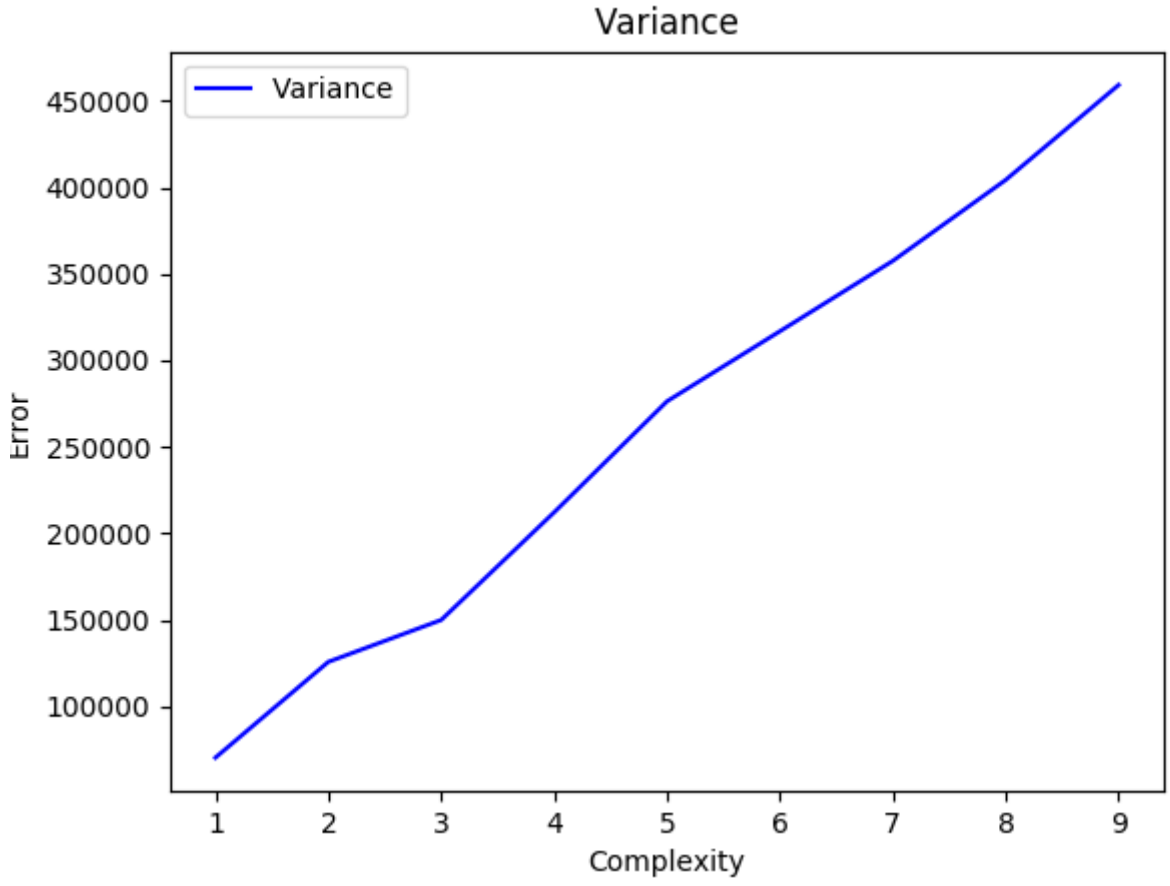
- Here, `Y_test` represents the correct output value for the input data point.

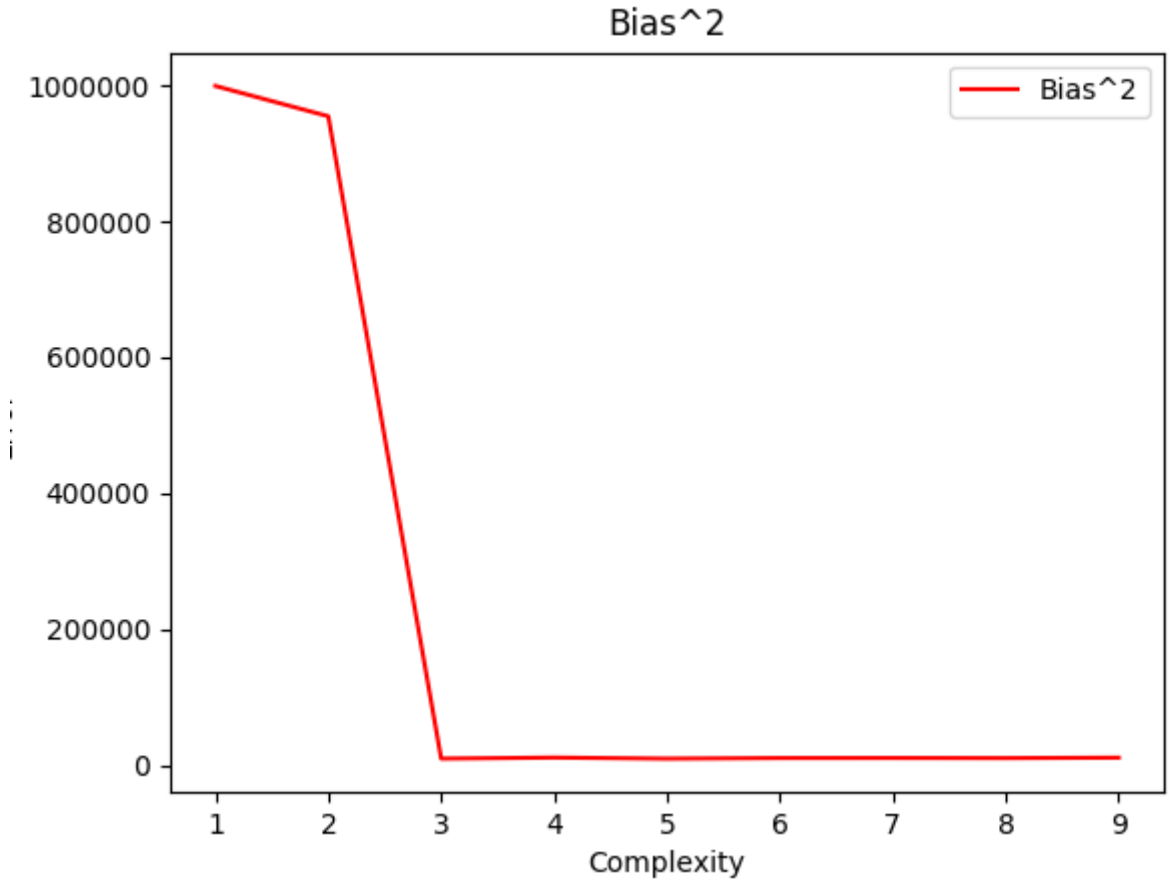
Listed are the plots for the models obtained using the `LinearRegression()` function using the first test set for the respective polynomials.

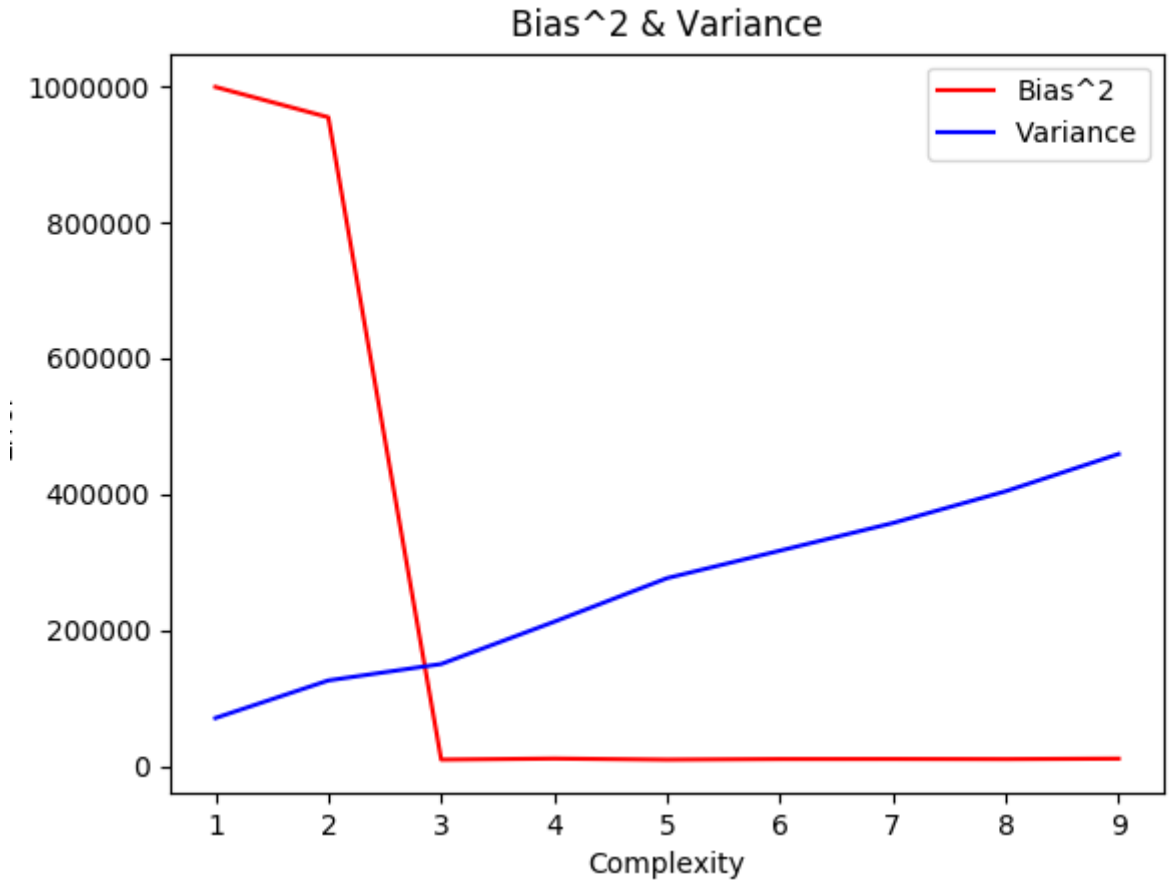


Bias^2 & Variance

Degree	Bias^2	Variance
1.0	999228.3968719237	70545.48914575034
2.0	954619.2737944251	125870.8555487738
3.0	9389.730116790513	150073.73954646458
4.0	10907.348133919644	212235.7083254309
5.0	9339.194280323269	276388.4802830072
6.0	10248.586258019084	316863.4993131382
7.0	10335.254180485183	357511.18901765486
8.0	10148.973246456477	404290.18163856165
9.0	10796.675000208099	458431.5382296077







Observervations

- The ***bias decreases*** and the ***variance increases*** steadily as the *degree of the polynomial increases*.
- That is, with increase in the number of features, or complexity (essentially increasing the degree of the model), the model becomes ***more flexible***. It extracts more information from the training sets and fits the data well. The values it predicts are close to the actual values, giving lesser bias.
- However, as it extracts more information, it also starts ***memorizing the noise***. It does not generalize well and gives a lot of variation on data sets that it has not seen before, ie variance increases.
- For lower degree polynomial models, we observe high bias and low variance. The model may not be able to perform well even on existing training data since the polynomial is so simple that it cannot capture the features of the training data. The variance is low since the model may not give accurate predictions but it does so consistently, generalizing well.
- This is in accordance with the Bias-Variance tradeoff where an optimum model is reached when total error is minimised.
- Here it can be see from the graph that the optimum complexity is 3, tnat is, the polynomial of degree 3 is the best model in terms of having the best Bias-Variance Tradeoff.