

MDL ASSIGNMENT 1

Q1. CALCULATING BIAS AND VARIANCE

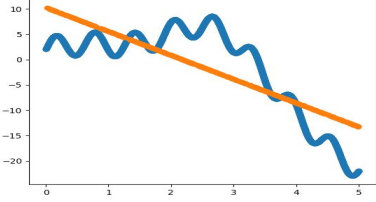
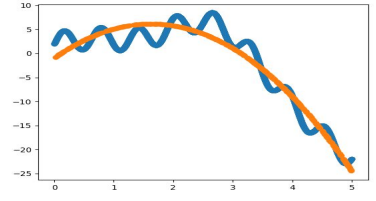
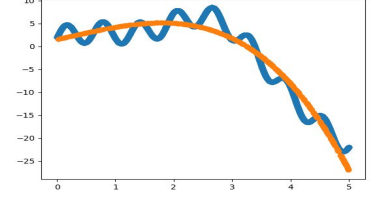
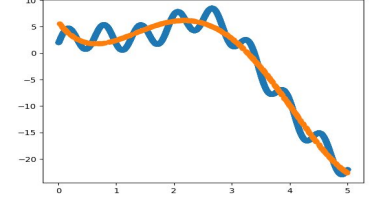
ALGORITHM IMPLEMENTATION

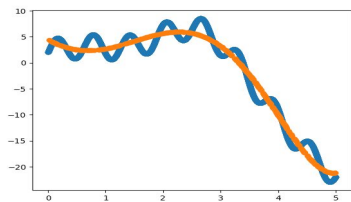
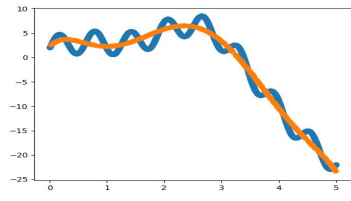
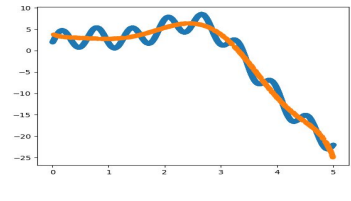
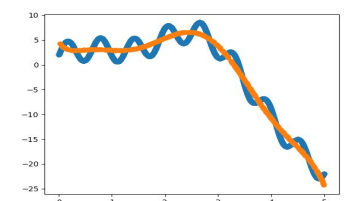
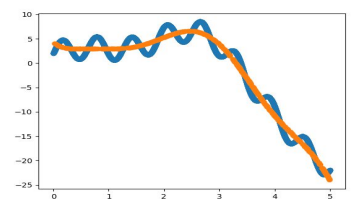
1. Load the data to the program using `pickle.load()` function.
2. Splitting data into Training and Test data sets using `train_test_split()` function in the ratio 90:10.
3. Split the training data further into 10 subsets of equal size and separate the x and y values correspondingly.
4. Initialize `final_bias` and `final_variance` vectors.
5. Iterate through degrees from 1 to 9 for fitting various class of functions and initialize bias and variance vectors with zeros.
6. Iterate through the 10 subsets of training data to get 10 models for each class of function.
7. Evaluate the list of powers of x values of all the data points in the training data set and test data set using `polyFeature.fit_transform()`.
8. Use `LinearRegression().fit()` to fit the model for the data.
9. Get `y_predicted` for the model obtained from test data.
10. Add the `y_predicted` values to bias vector, and $(y_predicted)^2$ to variance vector (these vectors don't indicate bias and variance at this point of time).
11. Take the average of the bias and variance vectors for 10 models.
12. Current variance vector represents $E[y'^2]$ and bias vector represents $E[y']$, where y' represents predicted value of a data point in test data set.
13. Calculate Variance as $E[y'^2] - (E[y'])^2$ and bias as $E[y'] - y$, where y' represents mean predicted value of a data point in test data set for 10 models and y represents true value of a data point in test data set.

14. Append the mean variance and mean square of bias of 500 data points in test data set to final_bias and final_variance vectors respectively for all class of functions.
15. Represent the final output in a tabulated format using PrettyTable library.

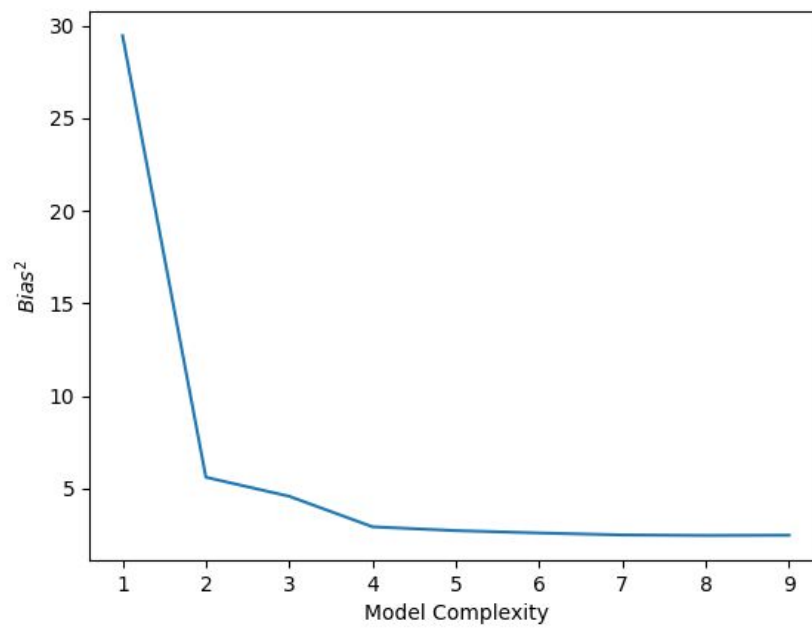
RESULTS (TABLE AND GRAPHS)

BIAS², VARIANCE TABLE

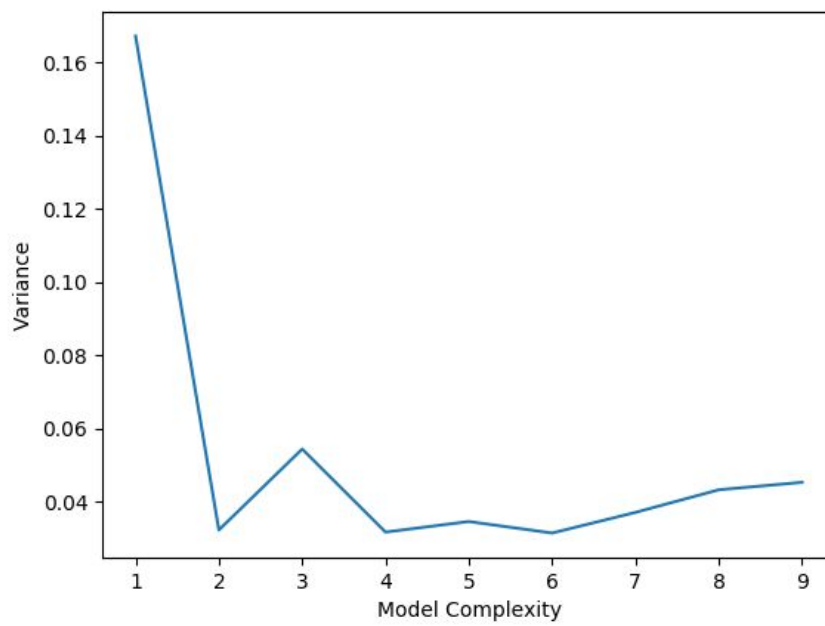
DEGREE	BIAS ²	VARIANCE	PLOT
1.	27.8443135868	0.130444286192	
2.	5.95576990667	0.0268281578917	
3.	5.09812336312	0.0239156262764	
4.	3.23393384006	0.0244378096568	

5.	3.04211539061	0.0298540866011	
6.	2.567069912	0.0305966356117	
7.	2.40088688413	0.0365438926044	
8.	2.37473818804	0.0459852421735	
9.	2.36033919601	0.0486677610207	

Bias² vs Model Complexity Graph



Variance vs Model Complexity Graph



OBSERVATIONS

1. For degree = 1, the bias² and variance is high which indicates **UNDERFITTING** of model to data given.
2. For degree = 2, we see a steep decrease in bias. It is clear that as the models move from linear to quadratic which improves the fit, it can learn better and minimize the total error.
3. As the complexity of the polynomial used for fitting increases the overall bias decreases. However, the variance increases due to the **BIAS-VARIANCE TRADE-OFF**.
4. Model complexities between 4 and 6 are a good fit for the given data as observed from the **BIAS-VARIANCE TRADE-OFF** graph.
5. In **OVERFITTING**, the model performs poorly in the test dataset. This is due to the increased complexity of the model and hence over generalization of the knowledge learned from the training data onto the test data set. However, the bias is very low as the model predicts values accurately.

Q2. BIAS VARIANCE TRADEOFF

ALGORITHM IMPLEMENTATION

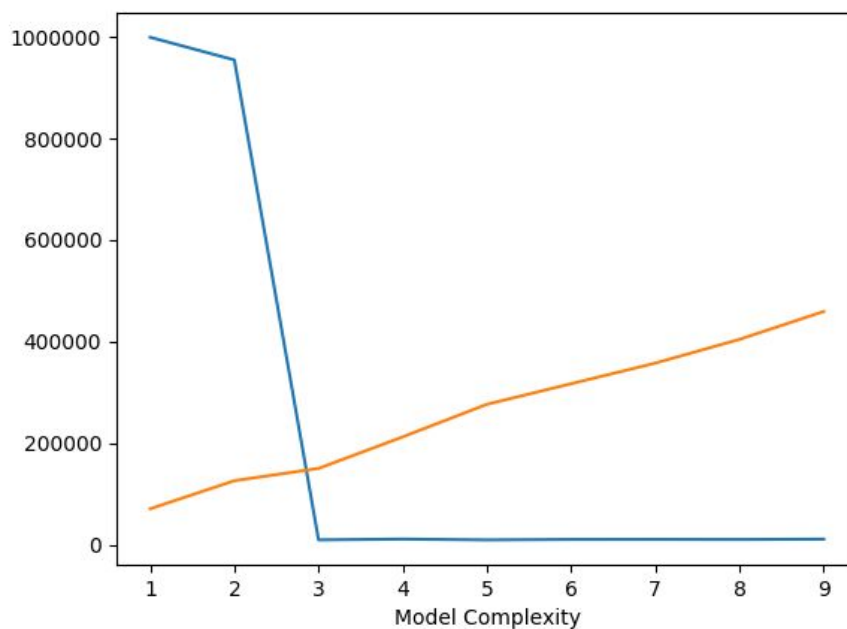
1. Load the data to the program using `pickle.load()` function. The data is already split into 20 subsets of training data set containing 400 data points each and test data set containing 80 data points.
2. Initialize `final_bias` and `final_variance` vectors.
3. Iterate through degrees from 1 to 9 for fitting various class of functions and initialize bias and variance vectors with zeros.
4. Iterate through the 20 subsets of training data to get 20 models for each class of function.
5. Evaluate the list of powers of x values of all the data points in the training data set and test data set using `polyFeature.fit_transform()`.
6. Use `LinearRegression().fit()` to fit the model for the data.
7. Get `y_predicted` from the model obtained for test data.
8. Add the `y_predicted` values to bias vector, and $(y_predicted)^2$ to variance vector.
9. Take the average of the bias and variance vector for 20 models.
10. Current variance vector represents $E[y'^2]$ and bias vector represents $E[y']$, where y' represents predicted value of a data point in test data set.
11. Calculate Variance as $E[y'^2] - (E[y'])^2$ and bias as $E[y'] - y$; where y' represents mean predicted value of a data point in test data set for 20 models and y represents true value of a data point in test data set.
12. Append the mean variance and mean square of bias of all 80 data points in test data set to `final_bias` and `final_variance` vectors respectively for all class of functions.
13. Represent the final output in a tabulated format using `PrettyTable` library.
14. Plot graph of **BIAS**² and **VARIANCE** using `matplotlib` library.

RESULTS (TABLE AND GRAPHS)

BIAS², VARIANCE TABLE

DEGREE	BIAS ²	VARIANCE
1	999228.3968719237	70545.48914575038
2	954619.273794425	125870.85554877351
3	9389.730116791214	150073.739546477
4	10907.34813407133	212235.70832526078
5	9339.194291326017	276388.4802547405
6	10248.585941147872	316863.4984374904
7	10335.2758616491	357510.9847573546
8	10149.419243937262	404286.6706857862
9	10815.487036574234	459132.378372487

BIAS - VARIANCE TRADE-OFF GRAPH



OBSERVATIONS

1. For functions of complexities 1 and 2, the bias is found to be very huge. This simply results from the fact that we are using linear/quadratic models (which is a simple and a less complex) to fit the given data. However, the variance increases from linear to quadratic. The model performs poorly in both training and test sets.

This is **UNDER-FITTING**.

2. For complexity 3, there is a good balance between bias and variance which results in minimum total error. The graphs of bias² and variance intersect near 3. We can conclude that the Optimal Model Complexity lies close to 3.

This is a **GOOD-FIT**.

3. For complexities greater than 3, the graph shows low bias and high variance. This is evident from the fact that as we build more complex models, the prediction mechanism is more efficient leading to lower bias. However, the model performs poorly in the test set as it over generalizes the information gained from the training data.

This is **OVER-FITTING**.

4. From the bias-variance plot obtained, we can conclude that the data fits very well for a third-degree polynomial. The plot of bias² and variance indicate near complexity 3. The minima of the error function exists for degree 3 polynomial. Hence the data is understood to be a good fit for a 3rd degree polynomial.
5. The graph is a visual depiction of the **BIAS-VARIANCE TRADE-OFF**.