Machine, Data and Learning

Assignment - 1

Satyam Viksit Pansari (2018101088)
Anishka Sachdeva (2018101112)

Problem Statement :

Calculate the bias and variance of your trained model.

Python libraries/modules used:

- **NumPy** - NumPy is a general-purpose array-processing package. It provides a high-performance multidimensional array object, and tools for working with these arrays.

- **Pickle** - The pickle module implements binary protocols for serializing and de-serializing a Python object structure. *"Pickling"* is the process whereby a Python object hierarchy is converted into a byte stream, and *"unpickling"* is the inverse operation, whereby a byte stream (from a binary file or bytes-like objects) is converted back into an object hierarchy.

- **Random** - Functions in the random module depend on a pseudo-random number generator function random().

- **Sklearn** - Scikit-learn is an open source Python library that implements a range of machine learning, pre-processing, cross-validation and visualization algorithms using a unified interface. It features various classification, regression and clustering algorithms.

- **Matplotlib** - Matplotlib is an amazing visualization library in Python for 2D plots of arrays. Matplotlib is a multi-platform data visualization library built on NumPy arrays.

- **Pandas** - Pandas is an open source library that allows you to perform data manipulation in Python. Pandas library is built on top of Numpy, meaning Pandas needs Numpy to operate. Pandas provide an easy way to create, manipulate and wrangle the data. Pandas is also an elegant solution for time series data.

For question 1,
Firstly opened the dataset file named “data.pkl” using the “**open**”.
Now loaded the opened dataset file using “**pickle**”.

Splitted the dataset into training set and testing set in the ratio 90:10 using “**train_test_split**” function from **sklearn.model_selection** library. Made “**shuffle**” parameter equal to true to randomly shuffle the dataset and divided and stored it into 10 equal parts using “**numpy.split**”.

Divided the testing set into two lists : One containing ‘x’ values and the other containing ‘y’ values.
Further each of the above lists is reshaped.

- **LinearRegression**

  LinearRegression fits a linear model with coefficients w = (w1, …, wp) to minimize the residual sum of squares between the observed targets in the dataset, and the targets predicted by the linear approximation.
  We are using sklearn's LinearRegression function for training the model.

- **PolynomialFeatures**

  Generate polynomial and interaction features. Generate a new feature matrix consisting of all polynomial combinations of the features with degree less than or equal to the specified degree. For example, if an input sample is two dimensional and of the form [a, b], the degree-2 polynomial features are [1, a, b, a^2, ab, b^2].
  Here though, we have only one feature that is the value ‘x’.

  For training models of higher degree, we use sklearn's PolynomialFeatures function.
  This comes under the preprocessing provided by sklearn
  Using this, we add x^2, x^3, ... features to the existing ones and then use LinearRegression on them.

- **make_pipeline**

  Construct a Pipeline from the given estimators.

This is a shorthand for the Pipeline constructor; it does not require, and does not permit, naming the estimators. Instead, their names will be set to the lowercase of their types automatically.

- **model.fit**

  Trains the model for a fixed number of epochs (iterations on a dataset).
  Here, the model.fit() takes the values of x and y from each of 10 training sets and trains the model. This is done for all the polynomial features from degree 1 to 9(preprocessing of x is being done via **PolynomialFeatures** as we had used **make_pipeline**).

- **model.predict**

  Given a trained model, predict the label of a new set of data. This method accepts one argument, the new data X_new (e.g. **model**. **predict**(X_new)), and returns the learned label for each object in the array.
  Here, the model.predict() takes the values of x from the testing set and predicts the model (values of y). This is done for all the polynomial features from degree 1 to 9.

- **Method of calculation of bias and variance:**

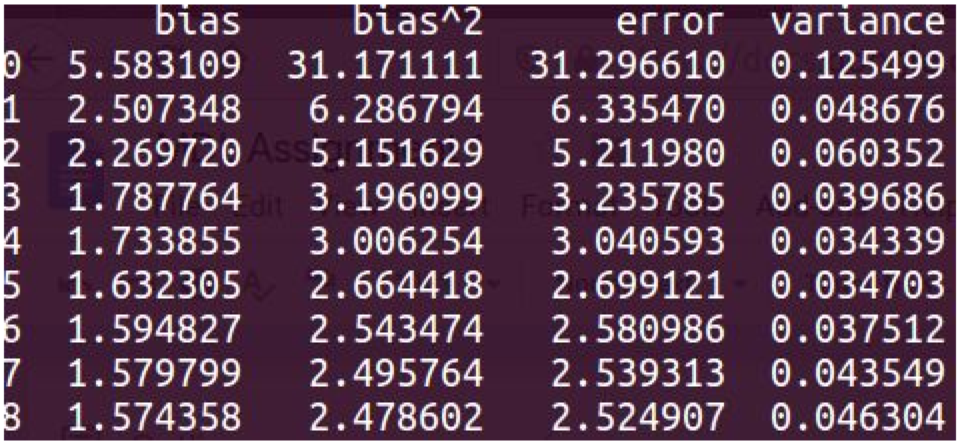
  For calculating bias, we accumulate the predictions of each model per degree. Then, we take the mean along column axis to get the average prediction across all models for that degree.

  Then, the formula bias^2 = (E[f\hat(x)] - f(x))^2 is used.

To calculate the variance, we used np.var

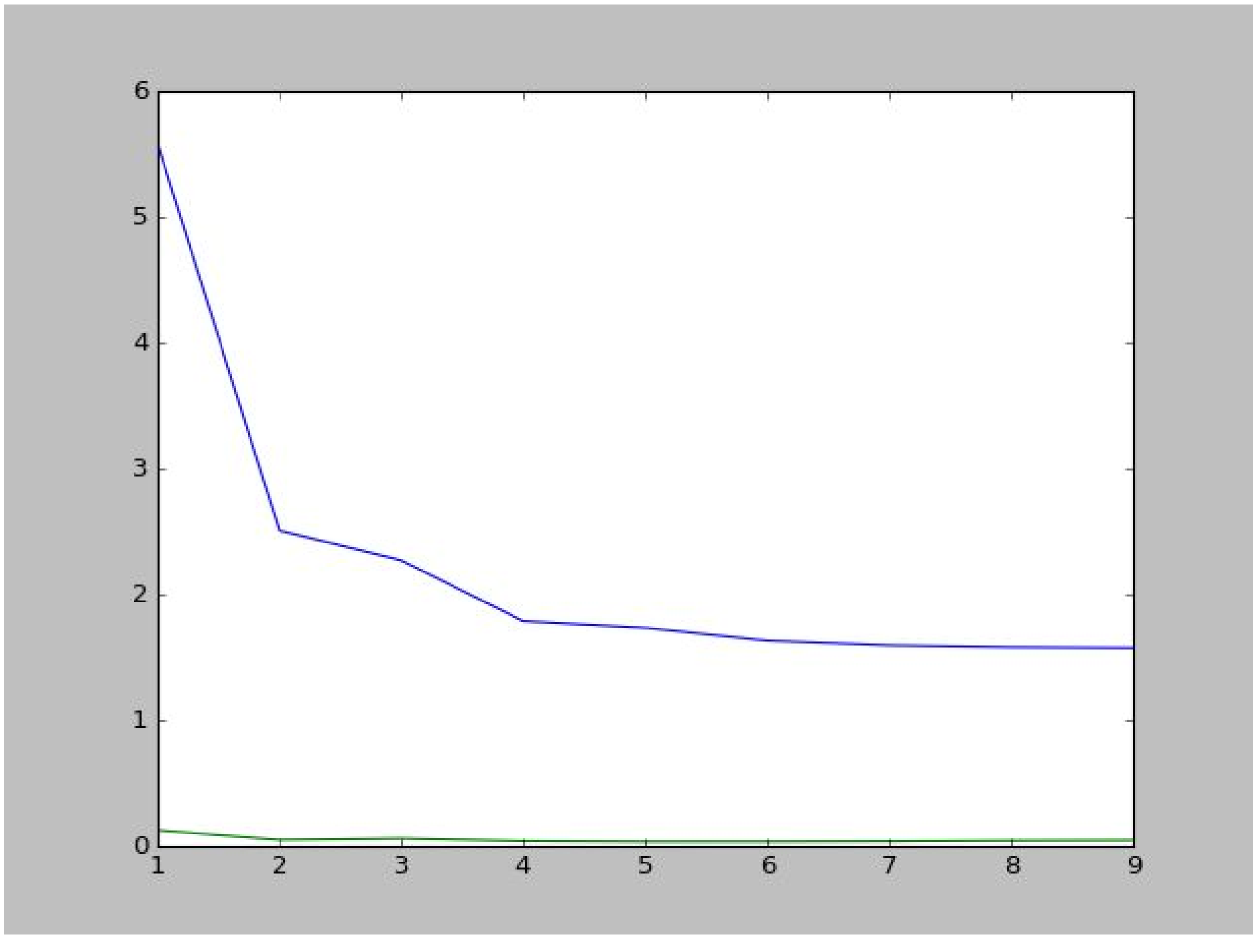
Question 1:

- **Values of bias and variance:**



```
       bias     bias^2      error  variance
0  5.583109  31.171111  31.296610  0.125499
1  2.507348   6.286794   6.335470  0.048676
2  2.269720   5.151629   5.211980  0.060352
3  1.787764   3.196099   3.235785  0.039686
4  1.733855   3.006254   3.040593  0.034339
5  1.632305   2.664418   2.699121  0.034703
6  1.594827   2.543474   2.580986  0.037512
7  1.579799   2.495764   2.539313  0.043549
8  1.574358   2.478602   2.524907  0.046304
```

- **Plot of the bias and variance:**

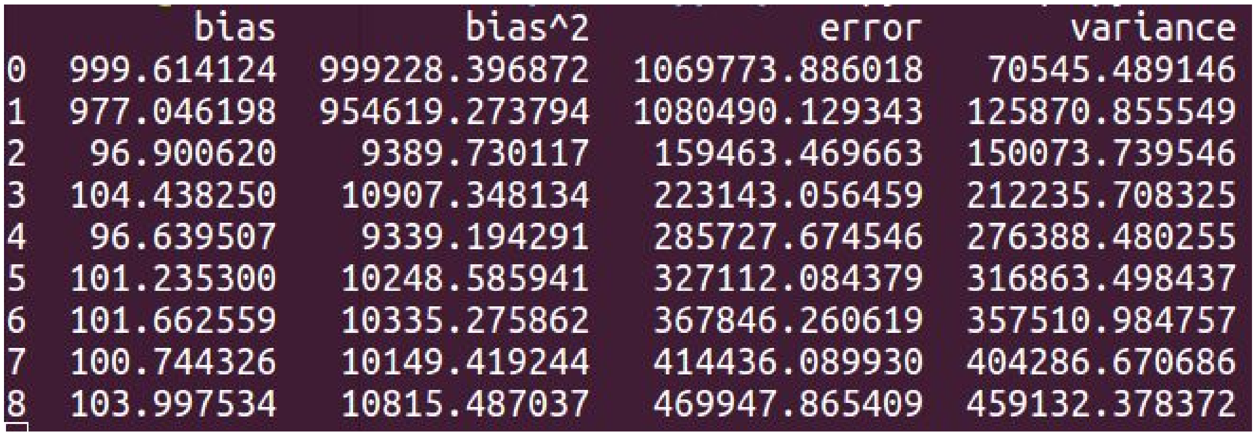


Let us analyse the graph:

- We notice that the bias consistently decreases: this is consistent with the idea that the model will keep fitting better and better with every degree i.e. as the degree increases, the model fits better.
- We notice that variance decreases till degree 5, and then again increases. This is because the given polynomial has roughly 3 inflection points. Therefore, it resembles a 5-degree polynomial with hills and valleys.

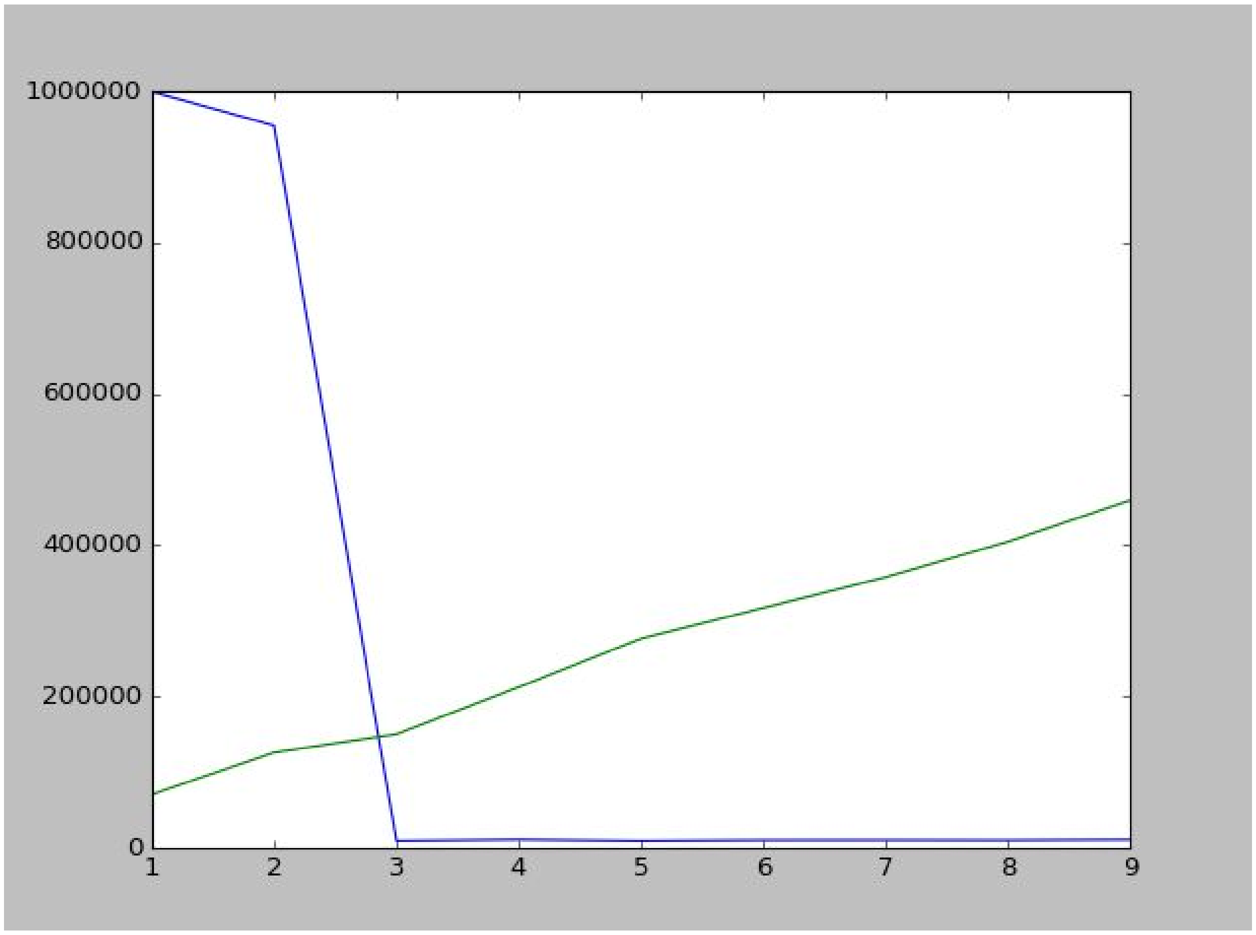
Question 2:

- **Values of bias and variance:**



```
         bias         bias^2           error        variance
0  999.614124  999228.396872  1069773.886018   70545.489146
1  977.046198  954619.273794  1080490.129343  125870.855549
2   96.900620    9389.730117   159463.469663  150073.739546
3  104.438250   10907.348134   223143.056459  212235.708325
4   96.639507    9339.194291   285727.674546  276388.480255
5  101.235300   10248.585941   327112.084379  316863.498437
6  101.662559   10335.275862   367846.260619  357510.984757
7  100.744326   10149.419244   414436.089930  404286.670686
8  103.997534   10815.487037   469947.865409  459132.378372
```

- **Plot of the bias and variance:**



Let us analyse the graph:

- The value of Bias^2 is very high for lower degrees (1,2) from which it is very clear that those models are underfitted. Then as the degree increases from 2 to 3 there is a drastic decrease in the value of bias^2. Then, as the degree increases the value of variance increases. Hence, clearly those models with higher degree are overfitted.
- From the graph, as there is a huge drop in bias^2 for degree 3, no such drastic changes are observed for other degree transitions and the value of variance is also not high, the data probably fits best for degree 3.

- Since we were given only a 3-degree polynomial, it fits perfectly for every degree after 3, and the variance and bias values are in the extremely low range of 10^{-25}.