Machine, Data and Learning - Assignment 1

Team#54
Rishabh Daga (2018101015)
Anubhav Sharma (2018114007)

Question 1

Task:

We were provided with a dataset consisting of pairs (x, y).

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

This loads the data into the program using pickle module and creates a numpy array ‘data’.

```
train, test = train_test_split(data, test_size = 0.1)
```

This divides the dataset in training and testing set with 90:10 ratio.

```
Training_parts = numpy.array(numpy.array_split(train, 10))
```

This splits the training set into 10 equal parts.

Now, there are two for loops in the code. First for loop is for training model on different degree polynomials. Second for loop trains model on ten different datasets for a particular class of function.

```
p = PolynomialFeatures(degree + 1)
```

sklearn's PolynomialFeatures is used to generate polynomial and interaction features.

```
X_train_polynomial = p.fit_transform(X_train)
X_test_polynomial = p.fit_transform(X_test)
```

sklearn's fit_transform is used for initial fitting of parameters on the training set and returns transformed features.

This fits the Linear Model to minimize the cost function.

```
regr = LinearRegression()
regr.fit(X_train_polynomial,y_train)
```

To find how our model works on testing data, model.predict is used on testing data.

Underfitting happens when a model is unable to capture the underlying pattern of the data. These models usually have high bias and low variance. It happens when we

have very less amount of data to build an accurate model or when we try to build a linear model with nonlinear data.

Overfitting happens when a model captures the noise along with the underlying pattern in data. It happens when we train our model a lot over noisy dataset. These models have low bias and high variance.

An optimal balance of bias and variance would never overfit or underfit the model.
Bias is how far are the predicted values are from the actual values. If the average predicted values are far off from the actual values then the bias is high. High bias causes algorithm to miss relevant relationship between input and output variable
Variance occurs when the model performs good on the trained dataset but does not do so well on a dataset that it is not trained on, like a test dataset or validation dataset. Variance tells us how scattered are the predicted values from the actual values. High variance causes overfitting that implies that the algorithm models random noise present in the training data. We generally want to minimize both bias and variance i.e build a model which not only fits the training data well but also generalizes well on test/validation data.

For calculating BIAS

```
bias_array=numpy.square(list(map(sub,y_test,expectation_output))).reshape(-1,1)
bias_square = numpy.mean(bias_array)
total_bias.append(bias_square)
bias = bias_square**(0.5)
```

For calculating VARIANCE

```
for w in range(test_data_size):
     for k in range(len(training_parts)):
          variance_array[w] += (pow((arr_of_outputs[k][w][0] - expectation_output[w]),2))

variance = numpy.mean(variance_array)
variance /= len(training_parts)
total_variance.append(variance)
```

Values:

```
➜  34_assgn1 python3 Q1.py
Enter the number of maximum degree:9
Degree = 1 , BIAS^2 = 30.918942849396213 , BIAS = 5.5604804513096004 , VARIANCE = 0.10351281203071963
Degree = 2 , BIAS^2 = 5.969578100830015 , BIAS = 2.4432720071310143 , VARIANCE = 0.056216138111964256
Degree = 3 , BIAS^2 = 5.191539174416581 , BIAS = 2.278494936227988 , VARIANCE = 0.047014246974138495
Degree = 4 , BIAS^2 = 3.189606414971096 , BIAS = 1.7859469238953032 , VARIANCE = 0.028416741677304024
Degree = 5 , BIAS^2 = 3.019242803315131 , BIAS = 1.737596847175757 , VARIANCE = 0.032260905427834476
Degree = 6 , BIAS^2 = 2.6462318162031324 , BIAS = 1.626724259425405 , VARIANCE = 0.03622319471736105
Degree = 7 , BIAS^2 = 2.524654784064682 , BIAS = 1.5889162294043957 , VARIANCE = 0.04048537717430268
Degree = 8 , BIAS^2 = 2.4910368442279824 , BIAS = 1.578301886277775 , VARIANCE = 0.041611175350221914
Degree = 9 , BIAS^2 = 2.472160902930168 , BIAS = 1.572310689059312 , VARIANCE = 0.041354472678364625
```

Observation: It can be seen that the bias decreases as the complexity of the polynomial increases hence, easy fit and low error. Variance initially shows a slight increase as the degree of the polynomial increases. It was high initially since the model was too simple(straight line only one configurable parameter). So, slight variations in data meant variation in slope which will change output for all. When our polynomial is too simple and has very few parameters then a high bias and low variance is observed. On the other hand, if our model has a large number of parameters then high variance and low bias is observed. Clearly for a given training set when the degree is low, hypothesis underfits the data and there is a high bias error. As the degree increases, fit gets better.(does not imply a good generalization) When the degree of the polynomial is low, error is high. This error decreases as the degree of polynomial increases and we tend to get a better fit. However error will again increase as the degree of polynomial increases more.

Question 2
:

Details of Algorithm Implementation:

1. We load the 20*400 sized (20 sets of 400 samples of (x,y) data) training data using pickle.load into the training_parts_X, training_parts_y variables and the testing data of size 80 (80 pairs of (x,y)) into X_test and y_test respectively. We vectorise the above using reshape(-1,1) before operating on them, for faster computation and ease.

```
training_parts_X=pickle.load(open('../Q2_data/X_train.pkl', 'rb'))
training_parts_y=pickle.load(open('../Q2_data/Y_train.pkl', 'rb'))
X_test=pickle.load(open('../Q2_data/X_test.pkl','rb')).reshape(-1,1)
y_test=pickle.load(open('../Q2_data/Fx_test.pkl','rb')).reshape(-1,1)
```

2. Then, we iterate i over 1 to 9, for each polynomial degree and have the nested for-loop, with i iterating over each of the 20 training sets (each time getting the ith set from training_parts_X, training_parts_y and storing them into X_train & y_train respectively)

3. We store the features (1, x, x^2, ... so on until x^i) for each point in X_train and X_test in X_train_polynomial and X_test_polynomial respectively.

```
p = PolynomialFeatures(degree + 1)
X_train_polynomial = p.fit_transform(X_train)
X_test_polynomial = p.fit_transform(X_test)
```

4. From sklearn.linear_model, we use the LinearRegression function to model X_train_polynomial on y_train (training set outcomes) and using the theta (optimal coefficients) gained from it, predict the outcomes of the testing set and training set as well, to find the error.

```
regr = LinearRegression()
regr.fit(X_train_polynomial,y_train)
```

We store the predictions for the testing set from the ith training set in output.

```
output = regr.predict(X_test_polynomial).reshape(-1,1)
```

5. We then calculate the bias and variance

```
arr_of_outputs = numpy.asarray(list_of_outputs)
expectation_output=(numpy.array(output_total)/len(training_parts_X)).reshape(-1,1)
```

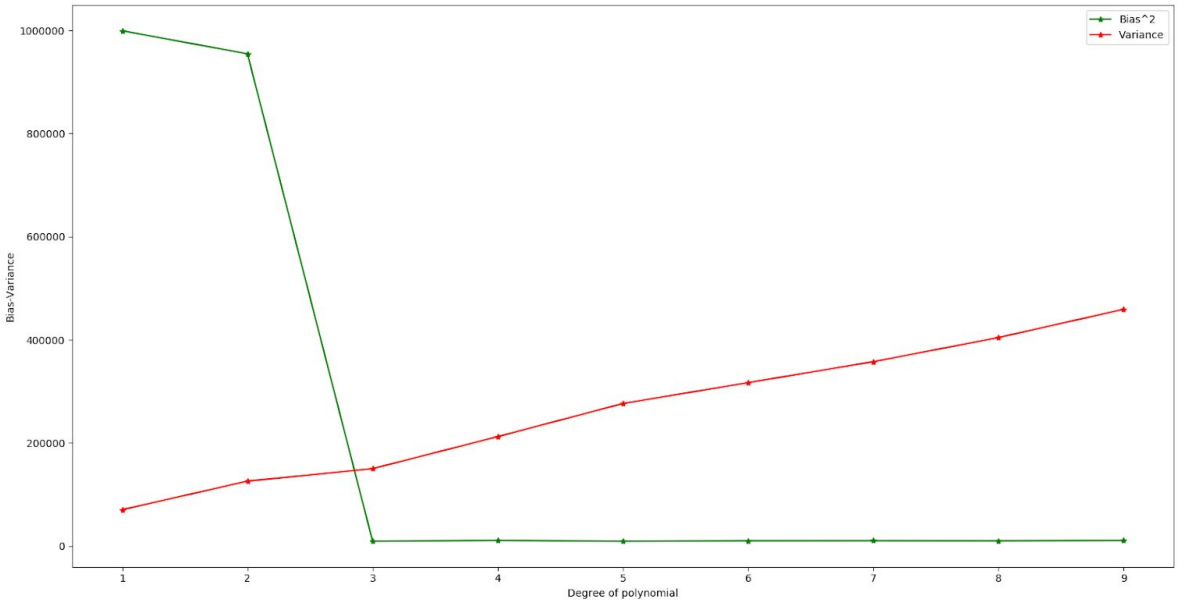
```
bias_array=numpy.square(list(map(sub,y_test,expectation_output))).reshape(-1,1)
bias_square=numpy.mean(bias_array)
total_bias.append(bias_square)
bias = bias_square**(0.5)
```

```
for w in range(test_data_size):
     for k in range(len(training_parts_X)):
variance_array[w]+=(pow((arr_of_outputs[k][w][0]-expectation_output[w]),2))
variance=numpy.mean(variance_array)
variance /= len(training_parts_X)
total_variance.append(variance)
```

6. Finally, we plot the bias-variance values over the degrees of polynomials and tabulate the values

```
→  34_assgn1 python3 Q2.py
Enter the number of maximum degree:9
Degree = 1 , BIAS^2 = 999228.3968719237 , BIAS = 999.6141239858127 , VARIANCE = 70545.48914575044
Degree = 2 , BIAS^2 = 954619.273794425 , BIAS = 977.0461983931082 , VARIANCE = 125870.85554877334
Degree = 3 , BIAS^2 = 9389.730116791214 , BIAS = 96.9006197957021 , VARIANCE = 150073.7395464768
Degree = 4 , BIAS^2 = 10907.34813407133 , BIAS = 104.43825033995604 , VARIANCE = 212235.70832526154
Degree = 5 , BIAS^2 = 9339.194291326017 , BIAS = 96.63950688681113 , VARIANCE = 276388.48025474057
Degree = 6 , BIAS^2 = 10248.585941147872 , BIAS = 101.23529987681111 , VARIANCE = 316863.49843748985
Degree = 7 , BIAS^2 = 10335.2758616491 , BIAS = 101.66255879943756 , VARIANCE = 357510.9847573547
Degree = 8 , BIAS^2 = 10149.419243937262 , BIAS = 100.74432611287477 , VARIANCE = 404286.670685786
Degree = 9 , BIAS^2 = 10815.487036574234 , BIAS = 103.99753380044275 , VARIANCE = 459132.3783724864
```

Bias-Variance v/s Polynomial
Degree:



Comment on Bias & Variance vs Degree : The bias decreases until n=3 (showing that as the complexity increases, it generalises lesser and lesser) and then stagnates at 0 till the end. The sharp drop from n=2 to n=3 shows that the data is actually a 3-degree polynomial thereby making bias = 0, and on continuing to increase the degree, it only overfits, to match it more and more, keep the bias at 0, and increasing the variance.

Whereas variance on the other hand, a measure of the spread of the predicted data, goes on increasing with n (i.e with increase in complexity of the model), which shows that the noise in the training data is significant.

Conclusion:

By the bias-variance tradeoff curve, we figure that the best model for the data is a polynomial of degree n = 3.