MDL-Assignment1

February 6, 2021

1 Tasks

1.1 Linear Regression

Linear regression is a ML algorithm that attempts to model the relationship between two variables by fitting a linear equation to observed data. Linear regression attempts to draw a straight line that will best minimize the residual sum of squares between the observed responses in the dataset, and the responses predicted by the linear approximation.It fits a linear model to minimize the residual sum of squares between the observed targets in the dataset, and the targets predicted by the linear approximation.

LinearRegression().fit() is a function of the Class sklearn.linear_model.LinearRegression.

The method LinearRegression().fit() fits the model to the training database during the training part of the process.This algorithm gives us required coefficients which are necessary to predict the output for the test dataset. In other words,LinearRegression().fit() fits the model. It returns self, which is the variable model itself.

1.2 Calculating Bias and Variance

This question aims at exploring and calculating the bias and variance of different models.

Bias is the difference between the average prediction of our model and the actual value we are trying to predict. A model with a high bias pays little attention to the training data and oversimplifies the function it is trying to predict. Thus it ends up with a high error on both the training and test data.

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

Variance is the variability of a model prediction for a given data point. It captures how much the model predictions for a given point vary between different realizations of the model.

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

Here, we train 20 different functions, each of which is trained separately on 10 different datasets. The 10 realizations of each model are used to calculate the bias and variance of that model on each point of a test set.

[1]:
```
# importing required libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
import pickle
import operator
```

1.2.1 Dataset Preprocessing

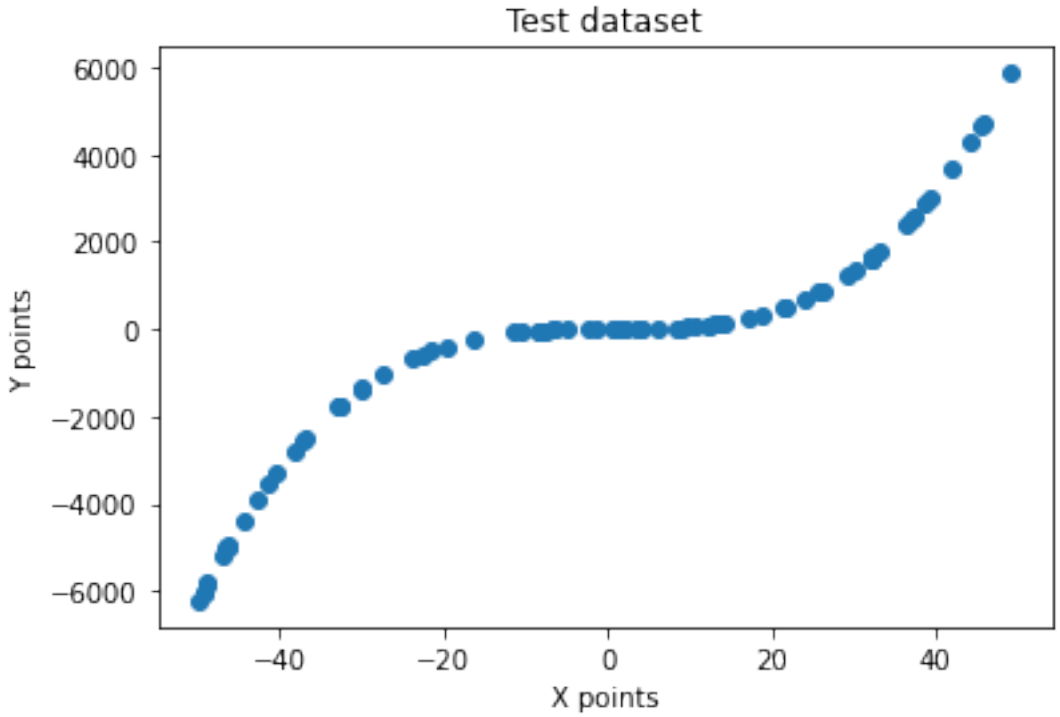
In this section, we load the dataset, ensuring that it is in a form that can be used for training the model.

[2]:
```
# loading train and test file
with open('./train.pkl','rb') as f:
    Training_data = pickle.load(f)
with open('./test.pkl','rb') as f:
    Test_data = pickle.load(f)
```

[3]:
```
# shuffling training data for random distribution of dataset
np.random.shuffle(Training_data)

#splitting training data into 10 equal sets
Train_data = np.array_split(Training_data,10)

#spliting x dataset from training set
X_test = np.hsplit(Test_data,2)[0]

#splitting y dataset from training set
Y_test = np.hsplit(Test_data,2)[1]

ax = operator.itemgetter(0)
Sorted_array = sorted(zip(X_test,Y_test), key = ax)
X_test, Y_test = zip(*Sorted_array)

#splitting x and y points in training data set
X_train = []
Y_train = []
for i in range(10):
    X_train.append(np.hsplit(Train_data[i],2)[0])
    Y_train.append(np.hsplit(Train_data[i],2)[1])

Degree = []
for i in range(1,21):
    Degree.append(i)
```

```
# declaring array which store bias, bias_sqr, variance, irreducible error, mean␣
 ↪square error for degree i+1(i is the index of the arrays)
model_bias = []
model_bias_sqr = []
model_variance = []
model_irreducible_err = []
model_mean_sqr_err = []
Y_predicted_test_set = []
```

[4]:
```
# displaying the test data set
plt.scatter(X_test,Y_test)
plt.xlabel('X points')
plt.ylabel('Y points')
plt.title('Test dataset')
plt.show()
```



1.2.2 Training

In this section, we train the models on the training dataset.

In the following cell, we train each model (i.e. polynomial of a particular degree) 10 times, once on each partition. We run an outer loop ->

for i in range(1,21): create_polynomial_regression_model(i)

which calls a function create_polynomial_regression_model(i) for calculating bias,variance,MSE,irreducible error for each degree i (for 20 degree's).In that function we are calling another loop which train 10 models for that function class on 10 different dataset which we have splitted earlier in X_train and Y_train. then we use preprocessing.PolynomialFeatures() to extend our dataset according to the degree of plolynomial. Later on we train our model using all the 10 trianing set in the for loop and store the predicted value for training model for test set in Y_predicted array (i.e. array in array)

For calculating bias, some basic arithimatic is done according to formula of bias, variance.We got predicted value of every test data point with 10 models we trained for each degree. To calculate the bias we took average of all the 10 output values for each degree and applied bias formula on it. similarly we calculate bias square for each point and then took the average of all 800 points to find final bias.

For variance we calculated mean of squared output and square of means output for every data input point and then subtracted them to calculate variance.

For mean square error i.e.

$$E[(f - \hat{f})^2] = E[f^2] + E[\hat{f}^2] - 2.E[f].E[\hat{f}]$$

we calculated mean square error for each input point and then took the mean of that for points for average mean.

For irreducible error we calculated irreducible error for each 800 input points and then took mean of that 800 points to calculate irreducible error for each function model class

[5]:
```
#function for training data and
def create_polynomial_regression_model(degree):
    # 10X(elements in training set) array output of each test set for all 10␣
 ↪models
    Y_predicted = []

    poly_features = PolynomialFeatures(degree=degree)
    for i in range(10):
        # transforms the existing features to higher degree features.
        X_train_poly = poly_features.fit_transform(X_train[i])

        poly_model = LinearRegression()
        poly_model.fit(X_train_poly, Y_train[i])

        Y_test_predict = poly_model.predict(poly_features.fit_transform(X_test))
#         ax = operator.itemgetter(0)
#         Sorted_array = sorted(zip(X_test,Y_test_predict), ax)
#         X_test, Y_test_predict = zip(*Sorted_array)
        Y_predicted.append(Y_test_predict)
        if(i==1):
            Y_predicted_test_set.append(Y_test_predict)

#         plt.scatter(X_test,Y_test)
```

```
#         plt.plot(X_test, Y_test_predict)
#         plt.show()


    Y_predicted_mean = np.mean(Y_predicted,axis=0)
    bias = np.absolute(np.subtract(Y_predicted_mean, Y_test))
    bias_sqr = np.power(bias,2)
    model_bias.append(np.mean(bias))
    model_bias_sqr.append(np.mean(bias_sqr))
    Y_predicted_square = np.square(Y_predicted)
    Y_predicted_square_mean = np.mean(Y_predicted_square,axis=0)
    Y_predicted_mean_square = np.square(Y_predicted_mean)
    Variance = np.subtract(Y_predicted_square_mean , Y_predicted_mean_square)
    model_variance.append(np.mean(Variance))
    Y_test_square = np.square(Y_test)
    Y_2ab_factor = np.multiply(Y_test,Y_predicted_mean)
    Y_2ab_factor = np.multiply(Y_2ab_factor,-2)
    Y_irreducible_error = np.add(Y_test_square,Y_predicted_square_mean)
    Y_irreducible_error = np.add(Y_irreducible_error,Y_2ab_factor)
    model_mean_sqr_err.append(np.mean(Y_irreducible_error))
    Y_irreducible_error = np.subtract(Y_irreducible_error,bias_sqr)
    Y_irreducible_error = np.subtract(Y_irreducible_error,Variance)
    model_irreducible_err.append(np.mean(Y_irreducible_error))
```

[6]:
```
for i in range(1,21):
    create_polynomial_regression_model(i)
# TASK - 2
df = pd.DataFrame({'Degree':Degree,'Bias':model_bias,'variance':model_variance})
df
```

[6]:

	Degree	Bias	variance
0	1	819.914667	11811.137260
1	2	810.613141	27773.962252
2	3	69.813380	39233.635665
3	4	75.366555	54289.570988
4	5	73.934734	67175.002418
5	6	74.005591	116778.111498
6	7	82.176965	122311.541357
7	8	83.910857	147409.228401
8	9	87.034822	176614.153881
9	10	89.041603	179625.864717
10	11	88.941849	204545.515384
11	12	111.353637	216073.110390
12	13	90.280716	212280.065411

```
13      14  121.075772  207507.072149
14      15  160.567903  216353.374074
15      16  163.016433  216878.077213
16      17  232.068079  225012.138282
17      18  233.052447  222560.485988
18      19  301.592652  230446.823503
19      20  301.152899  226588.578810
```

1.2.3 Analysis

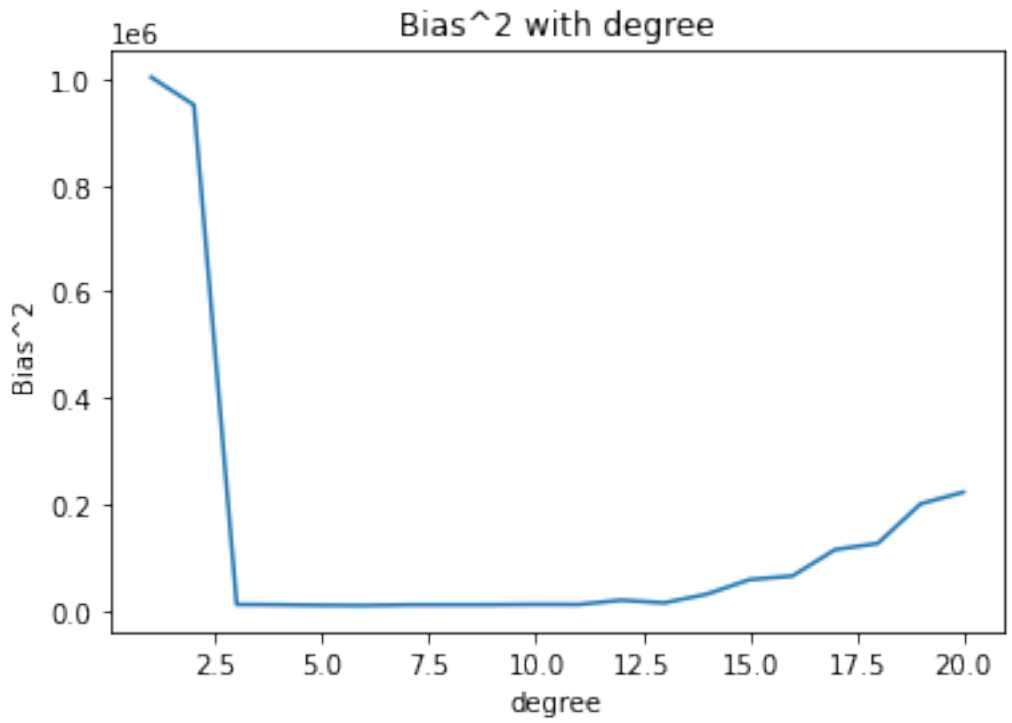
Now that we have the plots for the squared bias and variance of each model, we can analyse and compare their performance.

Bias As the degree of the polynomial hypothesis increases, the function becomes more flexible, allowing it to better mould itself to fit the training dataset. Hence, the error on the test dataset also decreases. Because the bias (the average difference between the predicted and actual values) effectively captures this error, we see a steadily deceasing trend in the bias values with an increase in degree of the hypothesis function. The change, however, is not so marked after degree 3-4 which hints at the fact that the data is best modelled by a degree 3-4 hypothesis, and a further increase in degree of the hypothesis would be frivolous.

Variance Save for an initial anomaly, the variance of the models shows a general increase with an increase in the degree of the hypothesis. This is because as the degree of the polynomial increases and it becomes more flexible, it also becomes more susceptible to minor variations in the training dataset. Hence, each time the model is trained, the increased flexibility of the highed degree polynomials causes the coefficients to turn out significantly different due to differences in the training set. Hence, the high variance on the test dataset. The degree one hypothesis has an abnormally high variance because the lack of any flexibility at all causes the models to turn out vastly different from each other as the learning algorithm tries to fit them to points that vary slightly amongst the different training sets. Because the hypothesis lacks the flexibility to account for these variations, they cause the entire model to train drastically differently.

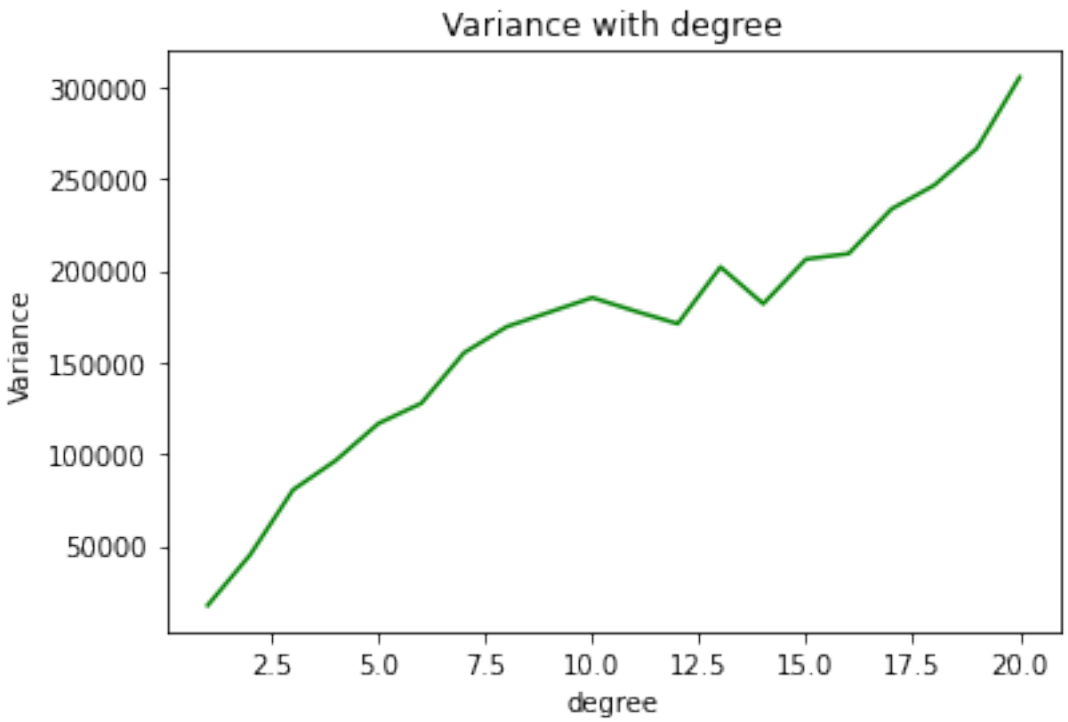
[7]:
```
plt.plot(Degree,model_bias_sqr)
plt.title('Bias^2 with degree')
plt.xlabel('degree')
plt.ylabel('Bias^2')
plt.plot()
```

[7]:
```
[]
```



[8]:
```
plt.plot(Degree,model_variance,color="green")
plt.title('Variance with degree')
plt.xlabel('degree')
plt.ylabel('Variance')
plt.plot()
```

[8]:



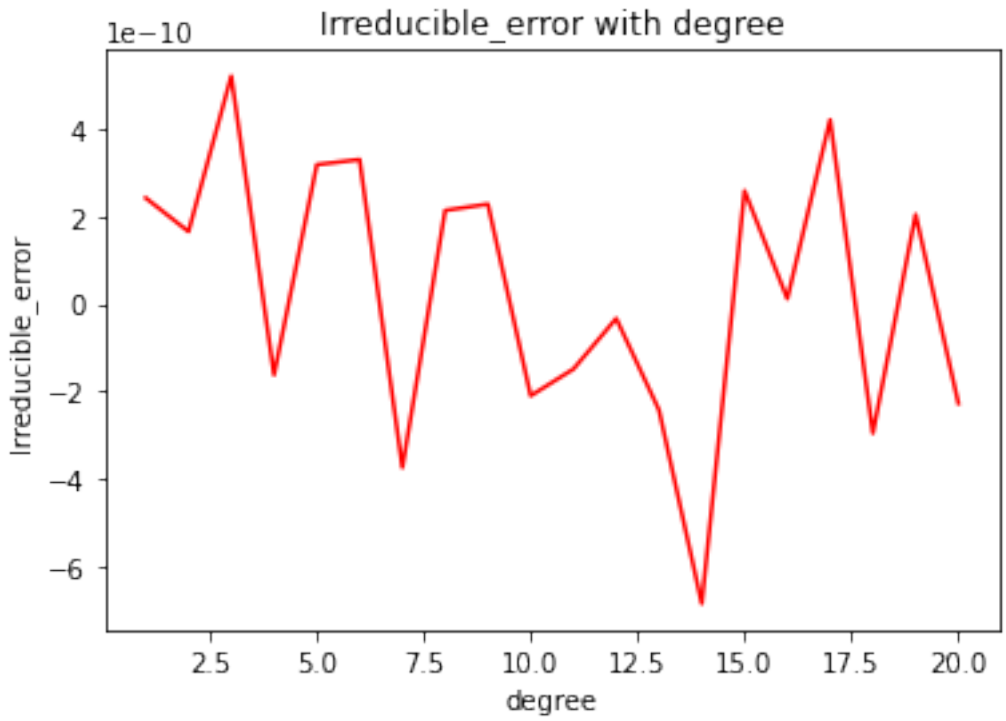
1.3 Calculating Irreducible Error

[9]:
```
#Task - 3
df = pd.DataFrame({'Degree': Degree,'Irreducible error':
 ↪model_irreducible_err,'MSE':model_mean_sqr_err})
print(df)
plt.plot(Degree,model_irreducible_err,color="red")
plt.xlabel('degree')
plt.ylabel('Irreducible_error')
plt.title('Irreducible_error with degree')
plt.plot()
```

```
   Degree  Irreducible error           MSE
0       1       1.968374e-10  1.014593e+06
1       2      -1.609806e-10  9.826625e+05
2       3       2.292836e-10  4.881808e+04
3       4       2.381853e-10  6.251271e+04
4       5       1.463718e-10  7.466793e+04
5       6      -3.753939e-11  1.245626e+05
6       7      -4.373987e-10  1.316602e+05
7       8      -4.758931e-11  1.568570e+05
8       9       3.670721e-10  1.864535e+05
9      10       3.957211e-10  1.906963e+05
```

```
10      11       1.480657e-10  2.156301e+05
11      12       1.757712e-10  2.385338e+05
12      13      -2.730758e-11  2.265362e+05
13      14      -3.431296e-10  2.407284e+05
14      15       3.514060e-10  2.751329e+05
15      16       9.081305e-11  2.814390e+05
16      17      -3.704827e-10  3.361613e+05
17      18       2.162096e-10  3.396285e+05
18      19       2.879915e-10  4.185167e+05
19      20       1.257831e-10  4.217095e+05
```

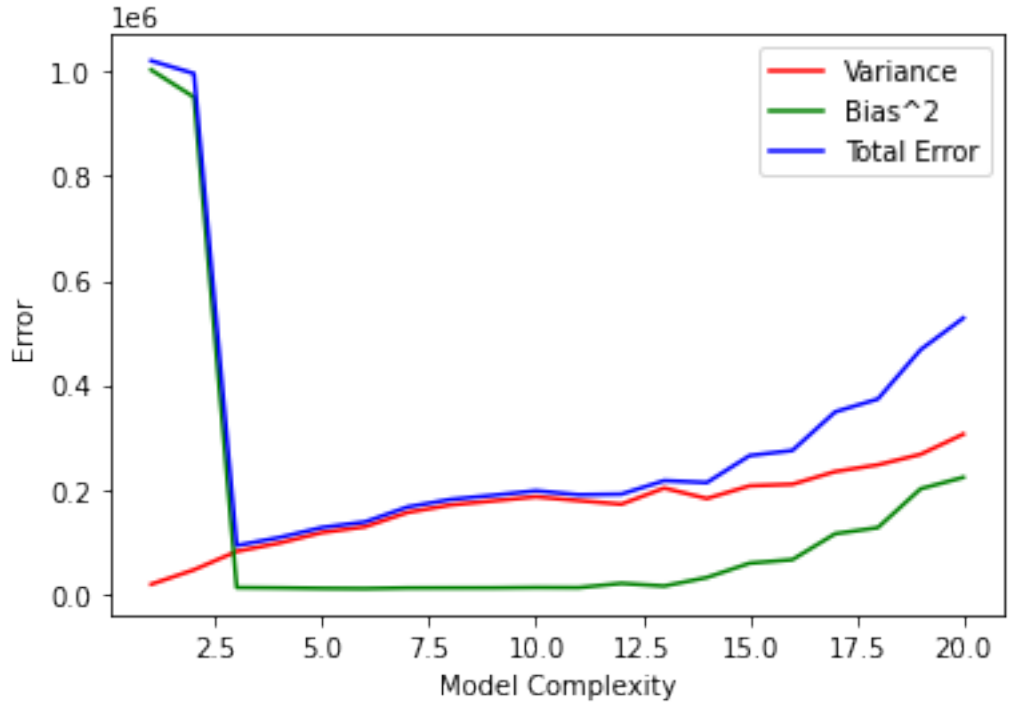
[9]: []



An irreducible error is an error that you get not because your model is not correct, but because of the noise in the data you are training or testing on. Hence, irreducible error doesn't change much with the model i.e our polynomial models from degree 1 to 20. The order of irreducible error is of 10^{-10} which is small and cannot be reducible.And the negative values of irreducible error are due to the floating-point precision error of the python interpreter.

1.4 Plotting Bias^2 – Variance graph

[10]:
```
# plt.plot(model_irreducible_err)

plt.plot(Degree, model_variance,color="red",label="Variance")
plt.plot(Degree, model_bias_sqr,color="green",label="Bias^2")
plt.plot(Degree, model_mean_sqr_err,color = "blue",label="Total Error")
plt.xlabel('Model Complexity')
plt.ylabel('Error')
plt.legend()
plt.show()
```



This plot shows the squared bias, variance and total error plotted on the same graph to visualize the BiasVariance Tradeoff. From the graphs, it is evident that an increase in degree beyond 5 does not bring much benefit in terms of bias, but just increases the variance for the worse. Because degree 3 has the least total error amongst degrees 3 to 5, it might be the best suited model for this case.