Polynomial Curve Fitting

***Abstract*—This document contains theory involved in curve fitting.**

1 Objective

The objective is to fit best line for the polynomial curve using regularization.

2 Generate Dataset

Create a sinusoidal function of the form

$$y = A\sin 2\pi x + n(t) \tag{2.0.1}$$

n(t) is the random noise that is included in the training set. This set consists of N samples of input data i.e. x expressed as shown below

$$x = \left(x_1, x_2, .., x_N\right)^T \tag{2.0.2}$$

which give the corresponding values of y denoted as

$$y = \left(y_1, y_2, .., y_N\right)^T \tag{2.0.3}$$

The Fig 0 is generated by random values of x_n for

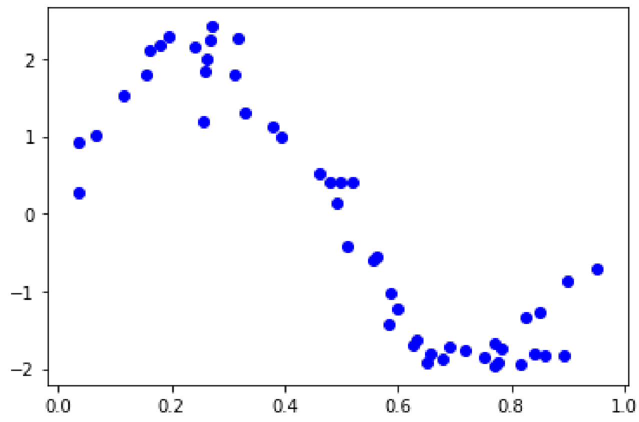

Fig. 0: Sinusoidal Dataset with added noise

n =1,2,..,N. where N=50 in the range [0,1].

The corresponding values of y were generated from the Eq (2.0.1).The first term $A\sin 2\pi x$ is computed directly and then random noise samples having a normal(Gaussian) distribution are added inorder to get the corresponding values of y.

```
#Generate the sine curve
import numpy as np
import matplotlib.pyplot as plt

N = 50
np.random.seed(20)
x = np.sort(np.random.rand(N,1),axis=0)
noise = np.random.normal(0,0.3,size=(N,1))
A = 2.5
y = A*np.sin(2*np.pi*x) + noise

plt.scatter(x,y,c='b',marker='o',label='Data with
    noise')
plt.xlabel('x');plt.ylabel('y')
```

The following code generates the input matrix F

```
sk_poly_deg=3
poly_feature = PolynomialFeatures(degree=
    sk_poly_deg,include_bias=False)
F = poly_feature.fit_transform(x)
```

The generated matrix would look like

$$\mathbf{F} = \begin{pmatrix} 1 & x_0 & x_0^2 & \dots & x_0^{N-1} \\ 1 & x_1 & x_1^2 & \dots & x_1^{N-1} \\ 1 & x_2 & x_2^2 & \dots & x_2^{N-1} \\ \vdots & & \vdots & & \vdots \\ 1 & \dots & \dots & \dots & x_N^{N-1} \end{pmatrix} \tag{2.0.4}$$

3 Polynomial Curve Fitting

The goal is to find the best line that fits into the pattern of the training data shown in the graph. We shall fit the data using a polynomial function of the form,

$$y\left(w, x\right) = \sum_{j=0}^{M} w_j x^j \tag{3.0.1}$$

$$\tag{3.0.2}$$

M is the order of the polynomial The polynomial coefficient are collectively denoted by the vector

w.The proposed vector **w** of the model referring to Eq (2.0.4) is given by

$$\hat{\mathbf{w}} = \left(\mathbf{F}^T\mathbf{F}\right)^{-1}\mathbf{F}^T y \tag{3.0.3}$$

4 Bias- Variance Tradeoff

Further, in this section we need to find suitable value for regularization parameter λ Once we are given the conditional distribution $p(t|x)$. A popular loss function in regression problems is squared loss function. Note that

Squared loss function(decision theory) != sum of squares error function (ML)

The optimal prediction for the squared loss function is given by the conditional expectation h(x)

$$h(x) = E[t|x] = \int tp(t|x)dt \tag{4.0.1}$$

To determine the conditional distribution, we need to use more sophisticated techniques such as regularization. The expected square loss takes the form

$$E[L] = \int (y(x) - h(x))^2 p(x)dx + \int (h(x) - t)^2 p(x,t)dxdt \tag{4.0.2}$$

The second term in Eq (4.0.2) is due to the noise in the data and represents the minimum achievable value of the expected loss. In the first term, we seek a solution for y(x) so that this term becomes minimum.

With unlimited data and computational resources we could find the regression function h(x) to any accuracy, but practically we only have dataset D and finite number of points, So h(x) is not known exactly. When we model h(x) using a parametric function y(w,x) with paramter vector **w**, From Bayesian point of view, the uncertainity in the model is expressed through a posterior distribution over **w**.

A frequentist treatment makes a point estimate of **w** based on the data, the uncertainity of this estimate is expressed through the large dataset consisting finite number of points and each drawn independently from distribution $p(t|x)$.

For any data set D, we learn the algorithm and get a prediction function y(x:D). So, different datasets give different functions which in turn give different squared losses. Averaging over sets gives us the performance of the algorithm.

Consider the integrand of the first term in Eq (4.0.2), which becomes

$$(y(x:D) - h(x))^2 \tag{4.0.3}$$

for a particular data set which has to averaged over the ensemble of sets. Adding and subtracting $E[y(x;D)]$ to Eq (4.0.3)

$$(y(x:D) - E[y(x;D)] + E[y(x;D)] - h(x))^2 \tag{4.0.4}$$

Expanding,

$$\begin{aligned}(y(x;D) - E[y(x;D)] + E[y(x;D)] - h(x))^2 = \\ (y(x;D) - E[y(x;D)])^2 + (E[y(x;D)] - h(x))^2 + \\ 2\,(y(x;D) - E[y(x;D)])\,(E[y(x;D)] - h(x))^2)\end{aligned} \tag{4.0.5}$$

Take the expectation w.r.t D,

$$\begin{aligned}E[(y(x:D) - h(x))^2] = (E[y(x;D)] - h(x))^2 \\ + E_D[(y(x;D) - E_D[y(x;D)])^2]\end{aligned} \tag{4.0.6}$$

In Eq (4.0.6),

the first term refers to $(bias)^2$ - which represents the extent to which the average prediction over all datasets differ from desired regression and

second term refers to *variance* - which measures the extent to which the solutions for individual data sets vary around their average and hence measuring the extent to which $y(x;D)$ is sensitive to a particular data set

Here, we have considered single input value x. Now substituting this expanded eq in Eq (4.0.2), $expectedloss = (bias)^2 + variance + noise$

$$(bias)^2 = \int (E[y(x;D)] - h(x))^2 \tag{4.0.7}$$

$$variance = \int E_D[(y(x;D) - E_D[y(x;D)])^2] \tag{4.0.8}$$

$$noise = \int (h(x) - t)^2 p(x,t)dxdt \tag{4.0.9}$$

The ultimate goal is to minimize the expected loss which we have decomposed into $(bias)^2$, *variance* and constant noise term.

There is a trade off between bias and variance

A flexible model has low bias and high variance

A rigid model has high bias and low variance

The model with optimal predictive capability is the one with best balance.

Consider this example where we generate 100 datasets, each containing 50 data points independently from the sinusoidal curve $h(x) = sin(x)$. For each dataset, we fit the model by minimizing the regularized error function to give a prediction function.

It is observed that,

large λ, low variance but high bias and

small λ, low bias but high variance.

We have to choose to λ value such that the value of $(bias)^2 + variance$ is minimum.

Here are few plots of different polynomial degrees with different λ values.

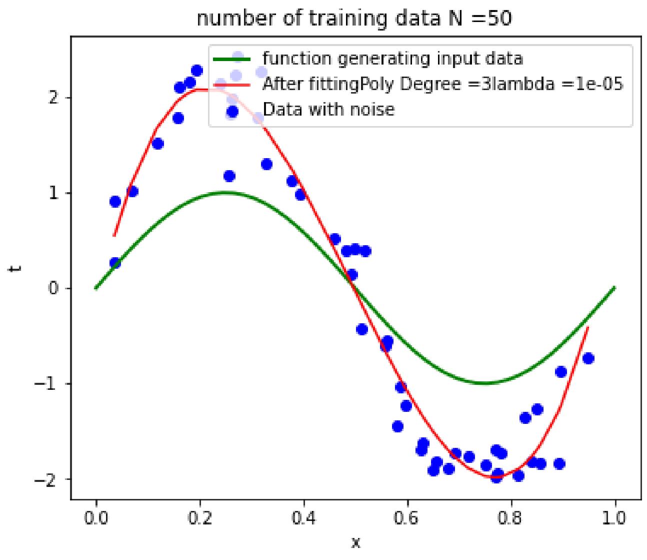

Fig. 0

In this example, we can also examine the bias - variance trade off quantitatively.

The average prediction is estimated from

$$\bar{y}(x) = \frac{1}{L}\sum_{l=1}^{L} y^{(l)}(x) \tag{4.0.10}$$

and the integrated $(bias)^2$ and variance is given by

$$(bias)^2 = \frac{1}{N}\sum_{n=1}^{N} (\bar{y}(x_n) - h(x_n))^2 \tag{4.0.11}$$

$$variance = \frac{1}{N}\sum_{n=1}^{N} \frac{1}{L}\sum_{l=1}^{L} \left(y^{(l)}(x_n) - \bar{y}(x_n)\right) \tag{4.0.12}$$

Python code:

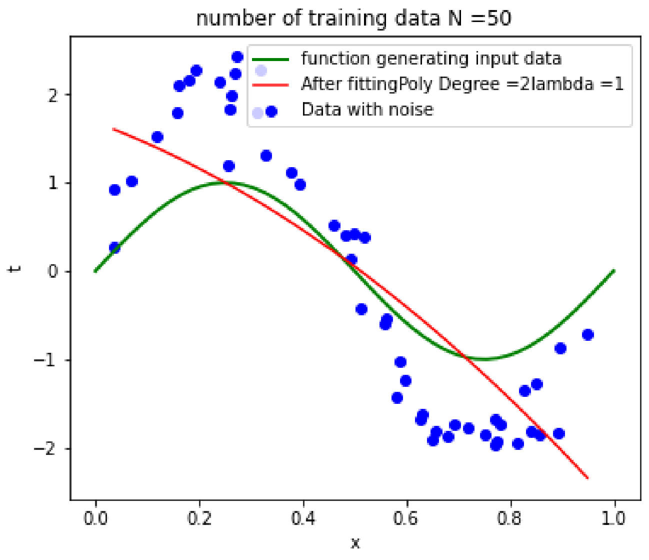

Fig. 0

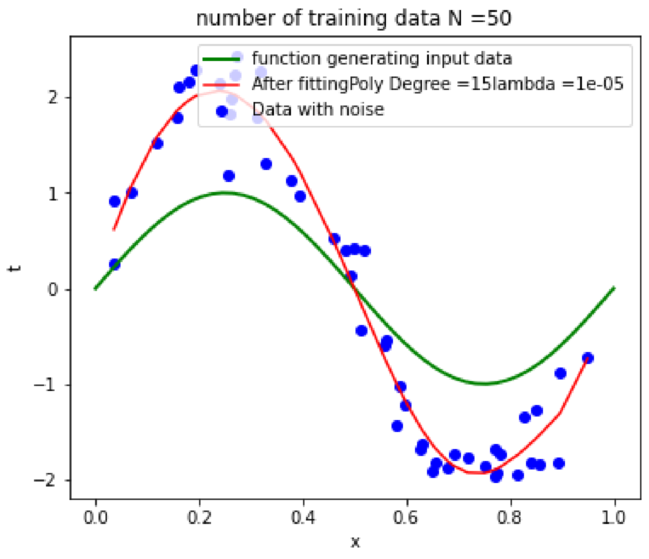

Fig. 0

https://github.com/Hrithikraj2/EE4015_IDP/blob/main/Assignment_5/Assignment_5.ipynb

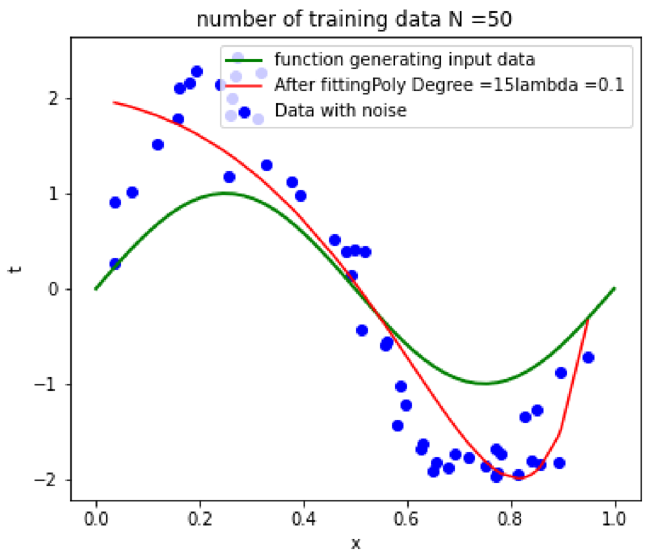
number of training data N =50
function generating input data
After fittingPoly Degree =15lambda =0.1
Data with noise
t
x
2
1
0
−1
−2
0.0
0.2
0.4
0.6
0.8
1.0

Fig. 0