#########################
Homework 4 | ML
Group Submission
Members:
Yasser Parambathkandy - (G01294910)
Indranil Pal - (G01235186)
Date: 12/2/2022

CS580 - Homework 4 | Regression

1) Linear Regression

a) Dataset - 2D-noisy-lin.txt
Loss - 0.10759283
Theta - [[2.93987438], [2.04156149], [-0.43683838]]
Plot:

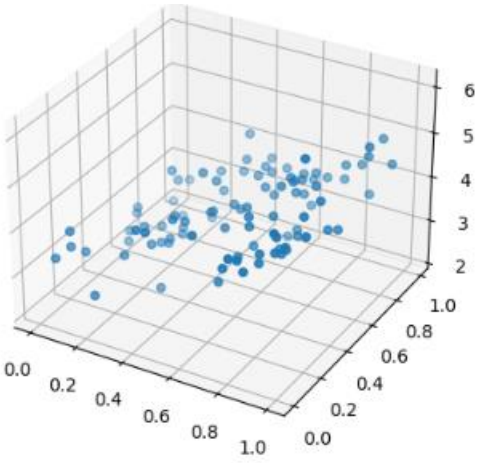


Figure 2: 2D-noisy-lin.txt data plot

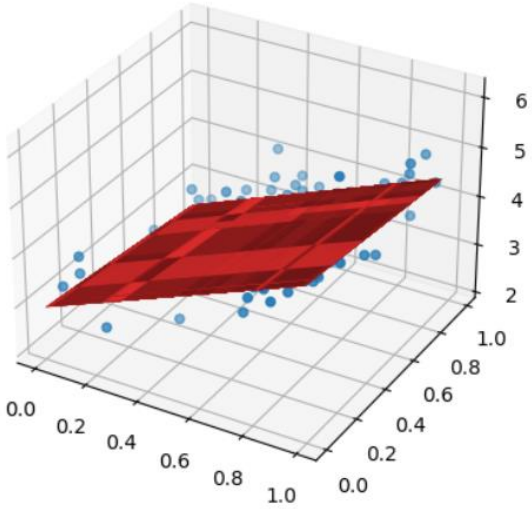


Figure 1: 2D-noisy-lin.txt linear regression plot

Dataset - 1D-no-noise-lin.txt
Loss – 0 (exact value - 2.92277086e-33)
Theta – [[0, 0.5]] (exact value - [[-4.681111291435602e-17], [0.5000000000000001]])
Plot:

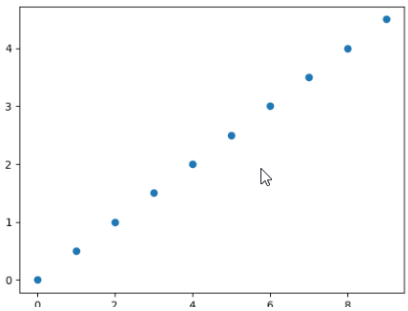


Figure 4: 1D-no-noise-lin.txt data plot

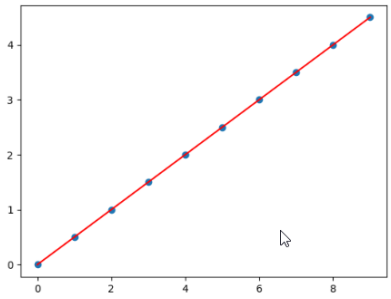


Figure 3: 1D-no-noise-lin.txt linear regression plot

b) When a feature is duplicated, the closed form solution on both datasets error out. This is because adding an identical column causes the X^TX matrix to be singular and the matrix inverse operation on it fails. When column is duplicated, it makes X^TX matrix have n-1 pivot points which makes it not an invertible matrix.

c) The same issue doesn't happen when a row is duplicated. The resulting theta value for 2D-noisy-lin.txt is [[2.97845216], [2.02338386], [-0.47754892]] and loss is [0.10906624]. The duplicate row has affected the thetas to be different from original. During loss calculation, there will be double the loss for this data point and linear regression will change thetas to minimize loss at the duplicated data.

d) There is no matrix inversion operation in gradient descent, so the issue encountered by closed form with duplicate feature doesn't occur in gradient descent.
With 1D-no-noise-lin.txt data, the default alpha produced a large loss value (2.98280724e+275) but by reducing the alpha to 0.03, the loss was almost zero at 500 iterations like the original dataset. Adding the additional feature brought the points together and hence the need to lower alpha for a slower descent. In the 2D-noisy-lin.txt, gradient descent produced results like its original data set.
For the duplicate row, gradient descent behaves like closed form, where it altered thetas to minimize the loss change due to the duplicate row. This was observed in both datasets.

2) Gradient Descent

a) Output of (loss, theta) for data set 1D-no-noise-lin.txt

Iteration: 1 , Loss: [3.5625] , Theta: [[0.], [0.]]
Iteration: 2 , Loss: [0.75738281] , Theta: [[0.1125], [0.7125]]
Iteration: 3 , Loss: [0.16152349] , Theta: [[0.0590625], [0.384375]]
Iteration: 4 , Loss: [0.03493767] , Theta: [[0.082125], [0.53585156]]
Iteration: 5 , Loss: [0.0080317] , Theta: [[0.06995215], [0.46628496]]
Iteration: 6 , Loss: [0.00229944] , Theta: [[0.07404042], [0.49858966]]
Iteration: 7 , Loss: [0.0010652] , Theta: [[0.07065573], [0.4839403]]
Iteration: 8 , Loss: [0.00078686] , Theta: [[0.07073638], [0.49092783]]
Iteration: 9 , Loss: [0.00071202] , Theta: [[0.0692408], [0.48793999]]
Iteration: 10 , Loss: [0.00068084] , Theta: [[0.06849226], [0.48954633]]

b) Run with default parameters for 1D-no-noise-lin.txt:

	Gradient Descent	Closed Form
Loss	[6.0173026e-10]	[1.01689101e-32]
Theta	[[6.44700512e-05], [4.99989719e-01]]	[[-1.66533454e-16], [5.00000000e-01]]

The difference is seen to be *insignificant* between the two solutions for data set 1D-no-noise-lin.txt

Run with default parameters for 2D-noisy-lin.txt

	Gradient Descent	Closed Form
Loss	[0.11078752]	[0.10759283]
Theta	[[2.75594278], [2.09675389], [-0.16343704]]	[[2.93987438], [2.04156149], [-0.43683838]]

The gradient descent loss is little higher, and the thetas are not far off from the closed form values. The gradient descent was stopped at 500 iterations and more iterations should minimize the loss and get the thetas same or better than the closed form solution.

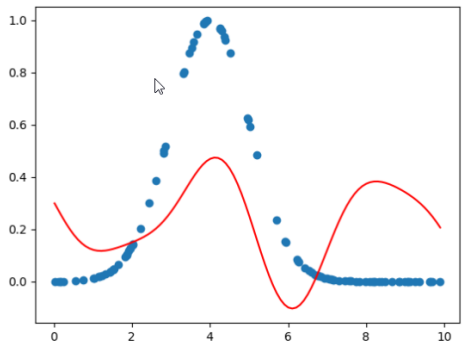
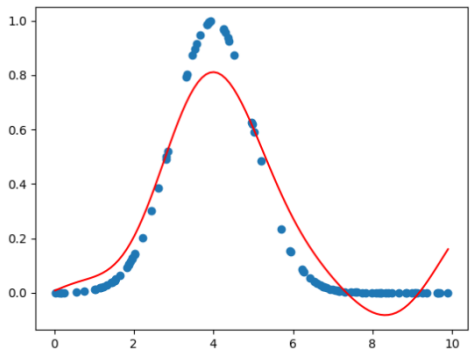
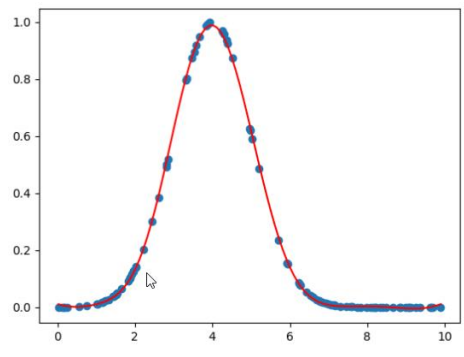
c) Values of learning rate and number of iterations for 1D-no-noise-lin.txt:
Alpha=[0.01, 0.05], Num_iters=[>=500]
The above set of learning rates and iterations produce the same results.
Alpha=[>0.06], Num_iters=[5, >=500]
As soon as the learning rate is more than 0.06, the losses and thetas are different.
For the 1D data, when the learning rate is less than 0.06 and the number of iterations is greater than 5, the answers are the same but when the learning rate goes above 0.06 deviations are noticed. This is because a smaller learning rate will need more iterations to converge than a higher learning rate. A high learning rate will overshoot the minima and skip the minima point. On the other hand, with a low number of iterations, gradient values will stop at a value before the global minima.

Values of learning rate and number of iterations for 2D-noisy-lin.txt:
Alpha=[0.1, 1.2], Num_iters=[>=600]
The above set of learning rates and iterations produce the same results.
Alpha=[>1.3], Num_iters=[>5]
As soon as the learning rate is more than 1.3, the losses and thetas are different.
The results of 2D data are similar in nature to that of the 1D data. The 2D data needed more iterations to converge at the global minima. This could be due to the slope between the training samples being smaller. A smaller learning rate will converge very slowly in this case. It is also noticed in the step history that when the learning rate is large, the theta values alternate between positive and negative, increasing the loss with each iteration.

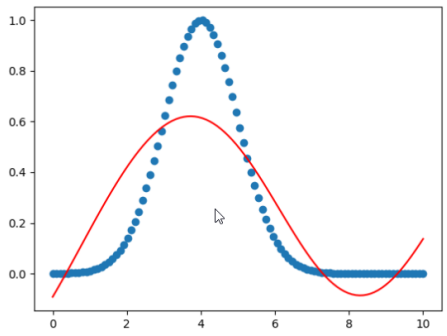
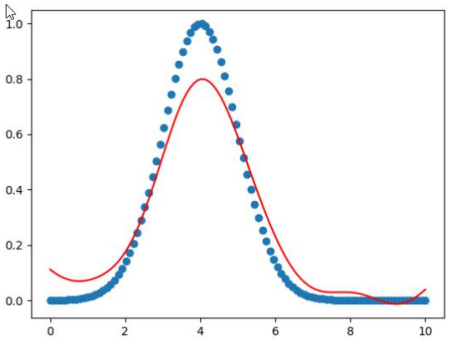
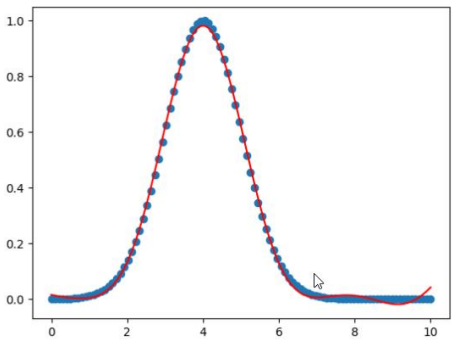
3) Random Fourier Features:

In all of the below datasets, the perfect fit for a large value of K could be achieved only by altering the alpha and iteration values of gradient descent. The values if changed from default have been noted in the tables below.

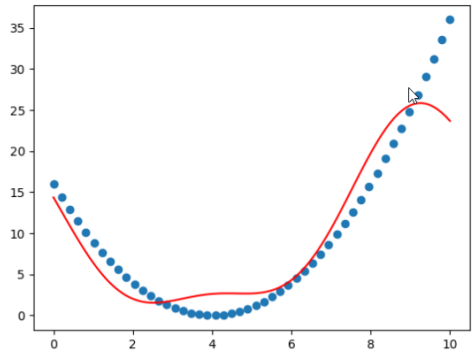
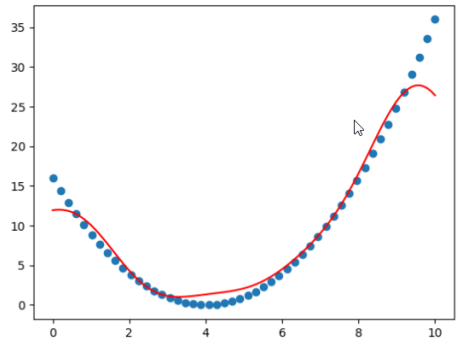
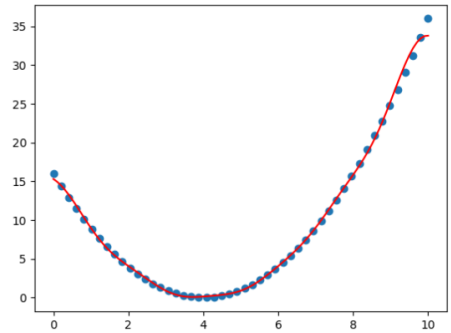
a) Dataset - 1D-exp-samp.txt

Small K = 2 | Underfit	Medium K = 50	Large K = 1000 Perfect Fit (with alpha=0.2, iteration=1000)
		

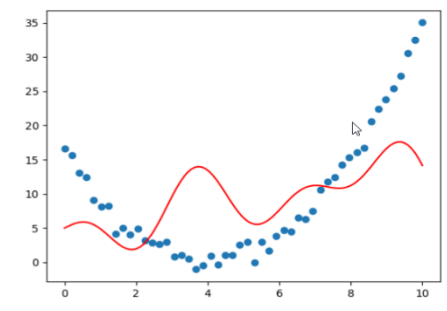
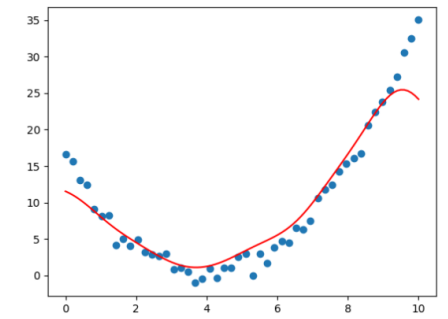
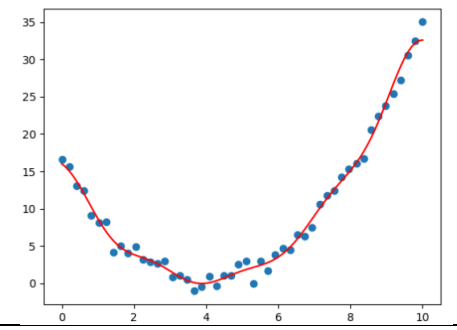
Dataset - 1D-exp-uni.txt:

Small K = 5 | Underfit	Medium K = 50	Large K = 1000 Perfect Fit (with alpha=0.2, iteration=1000)
		

Dataset - 1D-quad-uni.txt:

Small K = 5 | Underfit	Medium K = 100	Large K = 1000 Perfect Fit (with alpha=0.3, iteration=1000)
		

Dataset - 1D-quad-uni-noise.txt:

Small K = 5 | Underfit	Medium K = 100	Large K = 5000 Perfect Fit (with alpha=0.3, iteration=1000)
		

b)