Machine Learning

Gradient Descent

Feb 15th, 2019

OVERVIEW

The Whole assignment is to generate a dataset over a function following sin(2▯x) and add noise over it following the gaussian distribution, then to fit a curve with the help of Gradient Descent with different scenarios varying the number of data set samples and with the different degree of the hypothesis function.

GOALS

1. Synthetic data generation and simple curve fitting
2. Visualization of the dataset and the fitted curves
3. Experimenting with larger training set
4. Experimenting with cost functions

How to go about the source code :

There are different scripts each for the goals written above, each file is to be interpreted by python version : Python 3.6.5 :: Anaconda, Inc. The script for each part will gonna create the files which contains the data points taken into the processing and finding the parameter/weight associated with the best fitting curve to that data set taken into the consideration. Along with that each script file will gonna create some plots referencing some concerning some result parameters or factors.

Goal 1 : Synthetic data generation and simple curve fitting

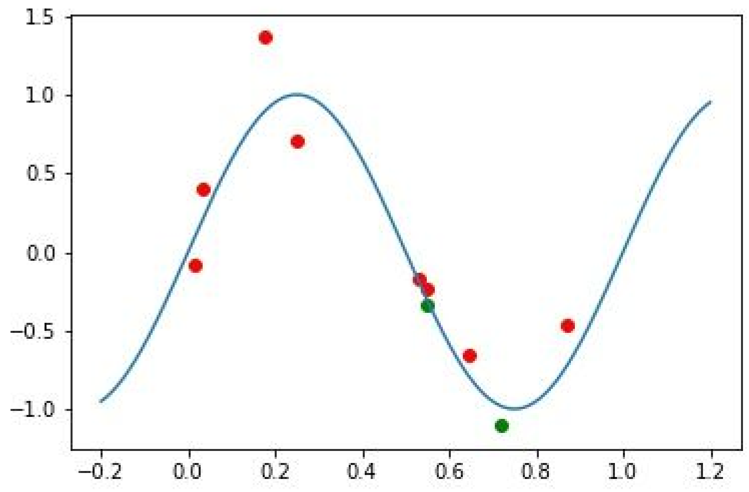
'part1.py' when interpreted with the python it will gonna generate a random dataset following the sinusoidal function as mentioned in the PS, along with the Gaussian Distribution following noise. That data set will gonna save in the file 'random_dataset.csv'

Then it will gonna take random 80% of that random data and this 80% will gonna taken as training dataset for the gradient algorithm. While the other random 20% will gonna be taken for the test dataset.

Finally the gradient descent is applied over that training dataset with different number of degrees of the hypothesis function varying from 1 to 9. The weights of all those hypothessis function haveing degree from 1 to 9 is written in the files `weight_n.csv` where n is the degree of the hypothesis function.

RED ONE ARE OF TRAINING SET AND GREEN ONE ARE FOR THE TEST DATASET

The dataset is as follows :



The following are the values of the weights that are found from the gradient descent for different values of n varying from 1 to n.

n = 1

0.57714227919003

-1.23516547631705

n = 2

0.6045747595522

-1.20789287780145

-0.16283159744651

n = 3

0.659160983698

-1.403979066771

-0.497657025913

0.627489991703

n = 4

0.7261951975424

-1.505999948347

-0.848331016587

0.1994907135875

1.080246776890

n = 5

0.7414310786975

-1.308030714319

-1.178014089815

-0.264467655381

0.604385231820

1.315792212471

n = 6

0.760773521793736

-1.21632986454226

-1.3452510938209

-0.568688679456

0.2454260537736

0.9479860485045

1.53534436335169

n = 7

0.77337983810832
5

-1.10366879708448

-1.46715922578063

-0.82077102745796

-0.0619851540552

0.628135031107859

1.22587279170049

1.73848820470228

n = 8

0.78164954823086

-0.984299114678318

-1.55941381761207

-1.03431995591671

-0.329114131162891

0.34696521700073

0.95202691425194

1.4826374062039

1.9446301809095

n = 9

0.78725927222078

-0.86568817489038

-1.6321098427489

-1.2191281586622

-0.56471740146973

0.096856808356

0.707241204570718

1.25321806190536

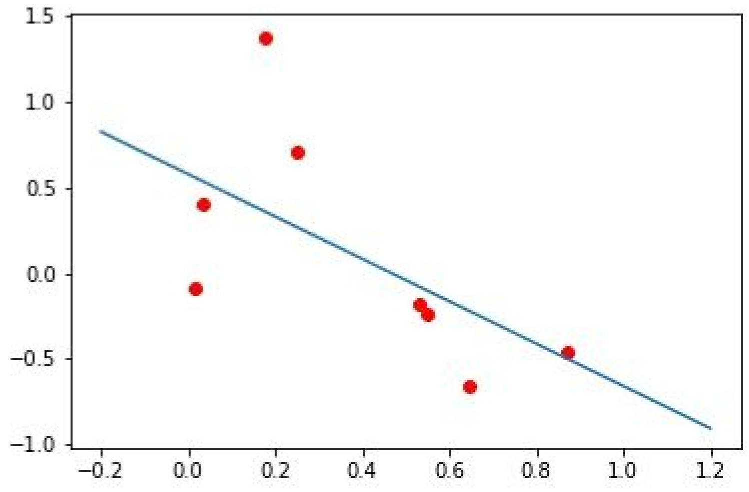
1.73501326693204

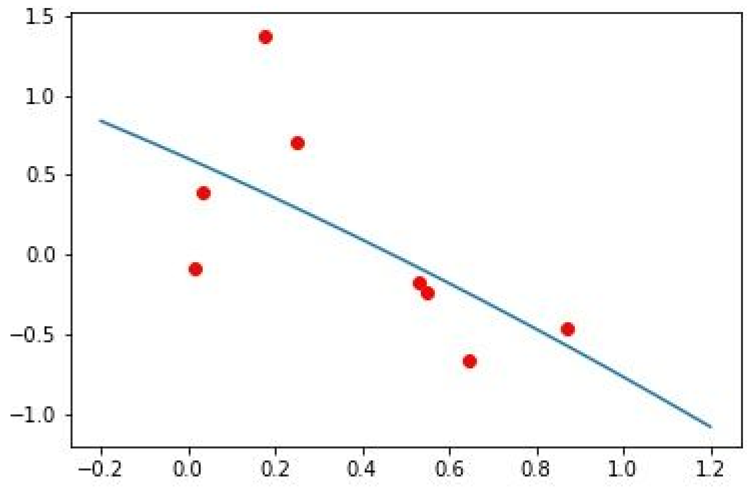
2.15728865789429

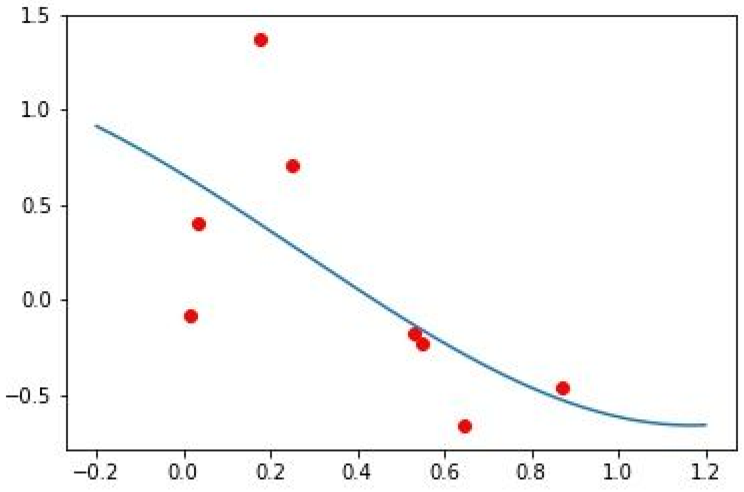
Goal 2 : Visualization of the dataset and fitted curve

‘part2.py’ when interpreted with the python it will gonna generate the plot of the dataset shoing training data points in red color and test set data points in green color, along with that it generates all the plot having name "fitted_curve_n.jpeg" which contains the fitted curve against the the dataset that we have taken into consideration to fit the curve on. And the last plot will be the train error and test error against the degree of the hypothessis function varting from 1 to 9, named “train_test_error.jpeg”

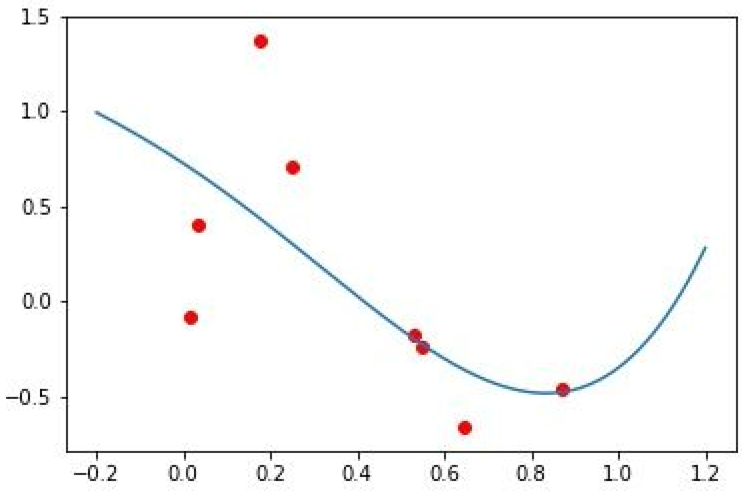
The following are the results :



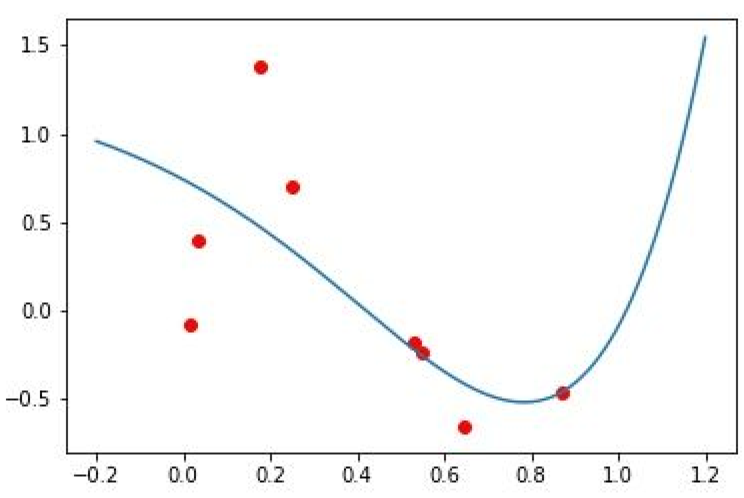




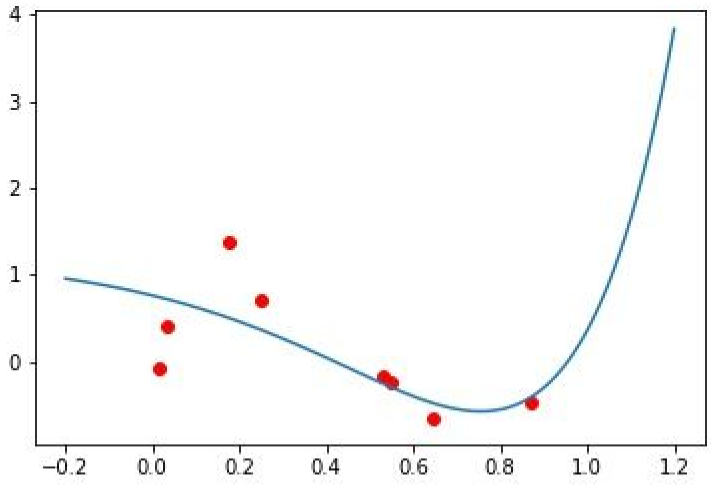
1.5
1.0
0.5
0.0
−0.5
−0.2
0.0
0.2
0.4
0.6
0.8
1.0
1.2

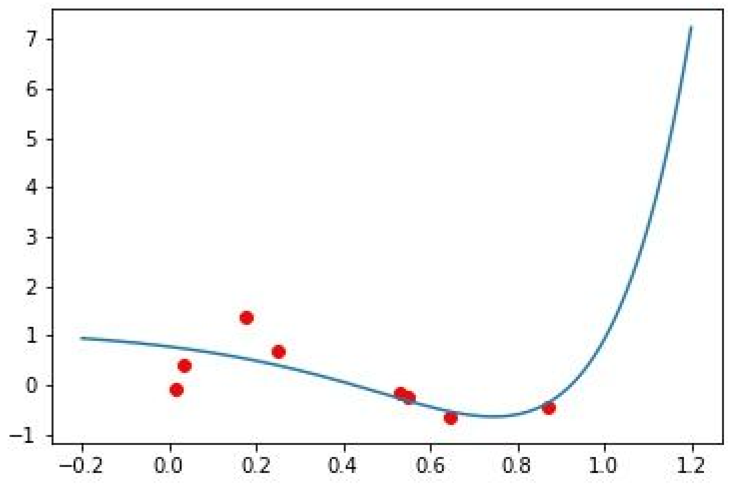


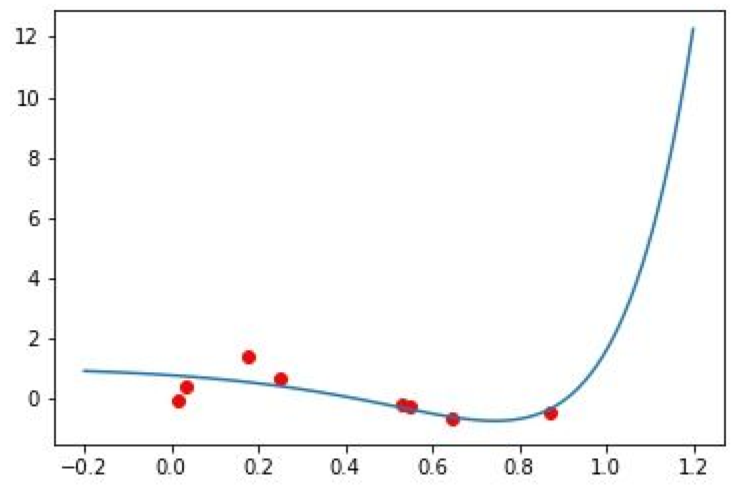
1.5
1.0
0.5
0.0
−0.5
−0.2
0.0
0.2
0.4
0.6
0.8
1.0
1.2

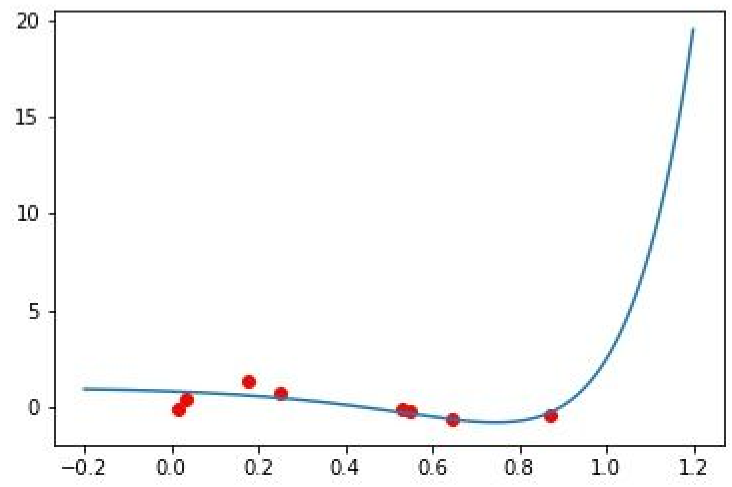


1.5
1.0
0.5
0.0
−0.5
−0.2
0.0
0.2
0.4
0.6
0.8
1.0
1.2


4
3
2
1
0
−0.2
0.0
0.2
0.4
0.6
0.8
1.0
1.2

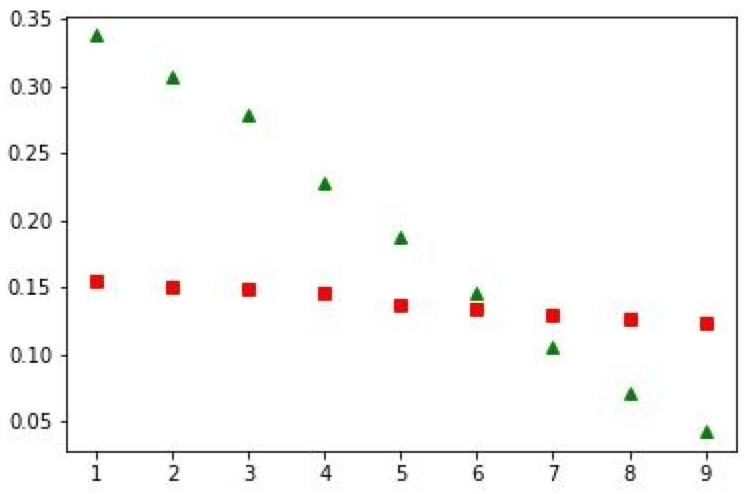

7
6
5
4
3
2
1
0
−1
−0.2
0.0
0.2
0.4
0.6
0.8
1.0
1.2





The test error and train errros plot is as follows :

Red one is for training error and green one is for the test error



For different run this plot is kinda vauge and sometimes even lower number of degree is proved better than anyone else that is because of the processing constraints and it saturating at the local minimum rather than Global minimum

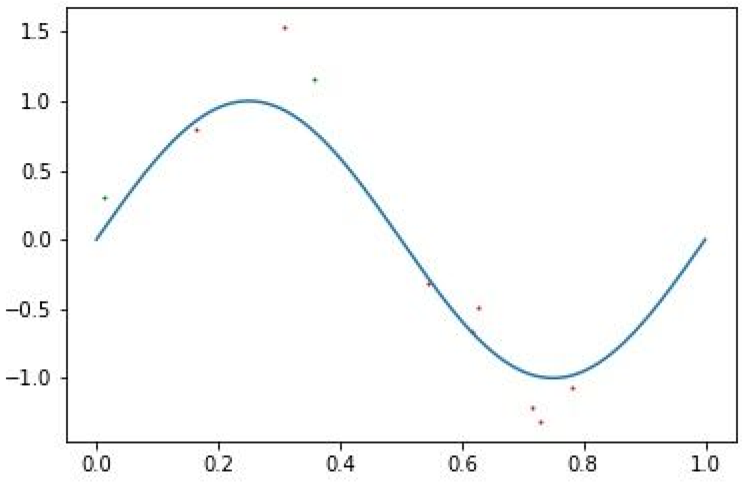
So rather taking any optimum value of n as best fit, I am gonna consider the average value of our experiment and take n = 6.

Goal 3 : Experimenting with larger training set

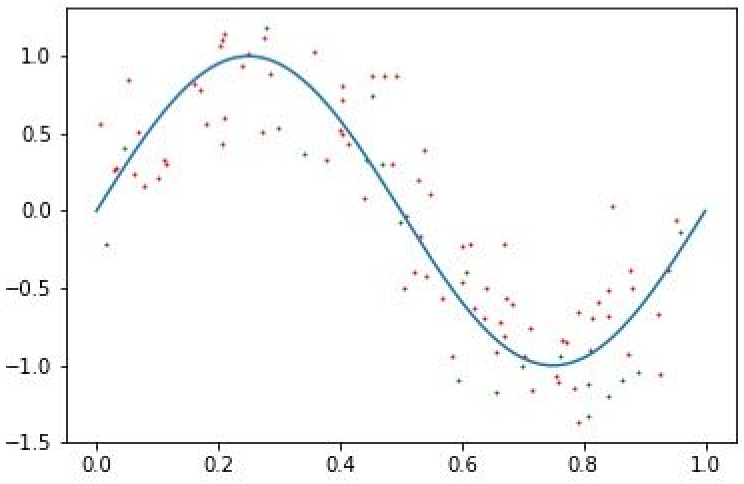
'part3.py' plot of the random dataset is saved as the csv file named random_dataset_part3_n.csv where the n is the number of the datapoints.

And the plot are saved in the file named as number_of_sample_n.jpeg where the n is the number of dtapoints.

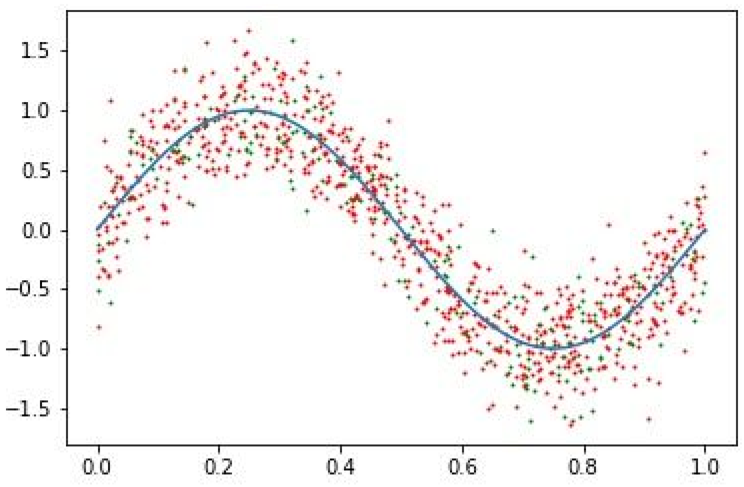
The data points are as follows :



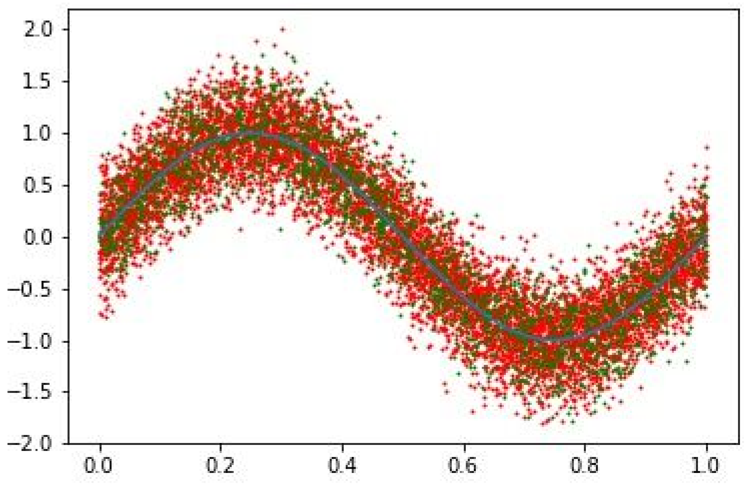
1.5
1.0
0.5
0.0
−0.5
−1.0
0.0
0.2
0.4
0.6
0.8
1.0



1.0
0.5
0.0
−0.5
−1.0
−1.5
0.0
0.2
0.4
0.6
0.8
1.0

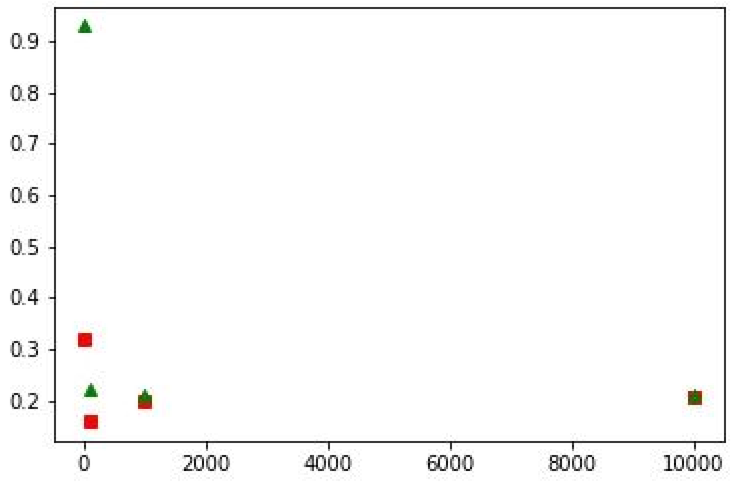


1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
0.0
0.2
0.4
0.6
0.8
1.0



2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
0.0
0.2
0.4
0.6
0.8
1.0

So the test and train error graph vs the number of training samples is as follows :

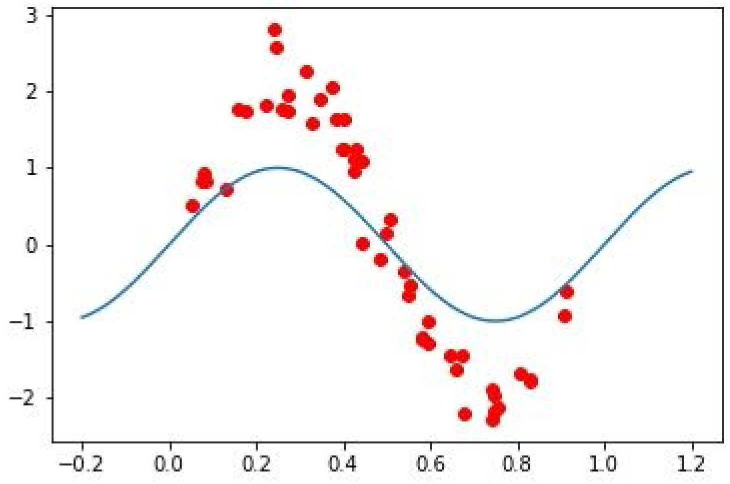


Which is named as train_test_error_large_dataset.jpeg file where red one are train error and green one are test error.

Goal 4 : Experimenting with Cost functions:

So in the final part, on interpreting the file part4.py, we will get the result like this ...

I have taken the random data as follows:



Which is stored in random_dataset_part4.csv file

And the values of the weights for the two cost function are as follows:

-0.17602454602786

-0.17602454602786

-0.17602454602786

-0.17602454602786

-0.17602454602786

-0.17602454602786

-0.17602454602786

And for the next cost function :

0.034681333844289

0.064499417260176

0.013839347375721

0.033298040958555

0.015180295383634

0.021339168941226

0.038575114599379