Algorithm Description and Modelling

As per the given problem statement, we have implemented the gradient descent and stochastic gradient descent and also applied the provided regularization to the function.

The initial step is data-pre-processing (where we have attempted to split the data into 80:20 (80% training and 20% testing)) and it is done using normalization and then calculating individual power coefficients and standardizing it.

Now for the gradient descent part, we have the following functions:

1. Concate\_ones -> This function simply adds a column of all 1’s at the beginning of the X.
2. Cost -> This function calculates the value of the cost of the function with calculated weights at each iteration.
3. Squared error -> This function calculates the final error at the end of the model making and it is calculating the final training and testing error.
4. Gradient Descent -> This is the heart of the program, and it is used to calculate the gradient descent and we are storing the cost history to trace the training error with all iterations.
5. make model -> This is the final function which considers the whole computation, where training and testing split takes place and final training and testing error is calculated.

After that we have plotted the training error and testing error against the degrees of polynomial that is training and testing error against flexibility of the polynomial.

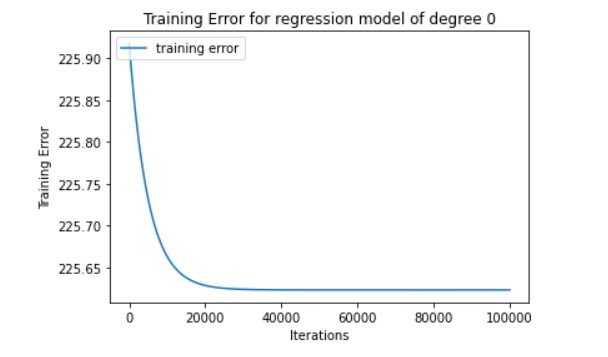
It is evident from the below provided graph that training error decreases as the flexibility increases and testing error initially decreases up to a minimum and then again increases as the flexibility increases.

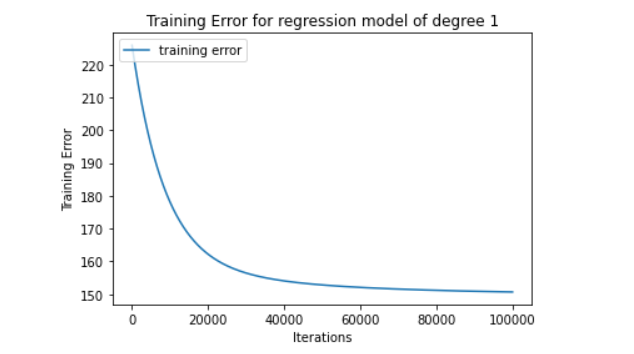
For the gradient descent we have polynomial of degree 1 as the most optimal solution.

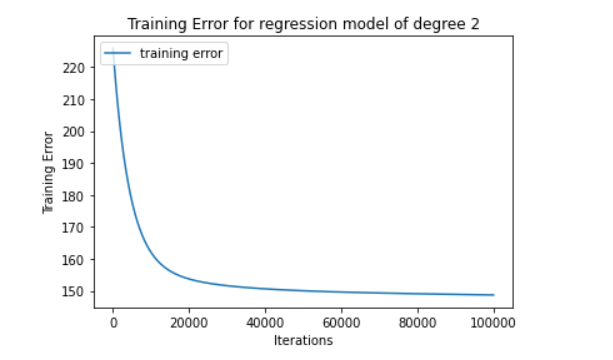
Table for Training and Testing Errors

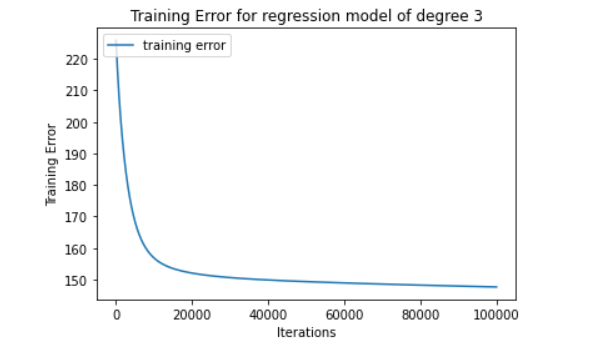
Degree Training error Testing error

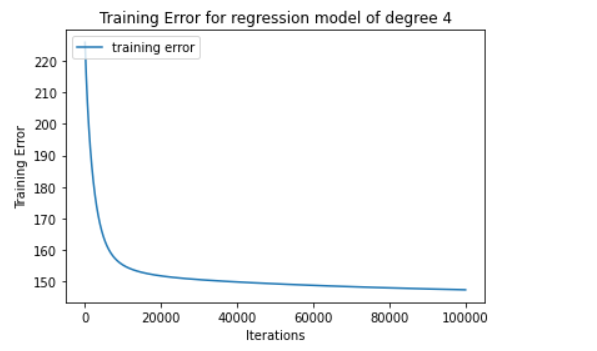
|  |  |  |
| --- | --- | --- |
| Zero | **0.5174846060607371** | **0.43405094826484564** |
| One | **0.34561239074600414** | **0.2864429421755414** |
| Two | **0.3410940479794503** | **0.29959921457066857** |
| Three | **0.3386945689449989** | **0.3217977065837518** |
| Four | **0.33813732272027075** | **0.3499431140708986** |
| Five | **0.3387868226817027** | **0.3801014412840353** |
| Six | **0.3394274869044149** | **0.40623300159760317** |
| Seven | **0.3393355382111092** | **0.4230920663344977** |
| Eight | **0.3384453835858003** | **0.4283115566989572** |
| Nine | **0.33704397866588015** | **0.4227906201663382** |

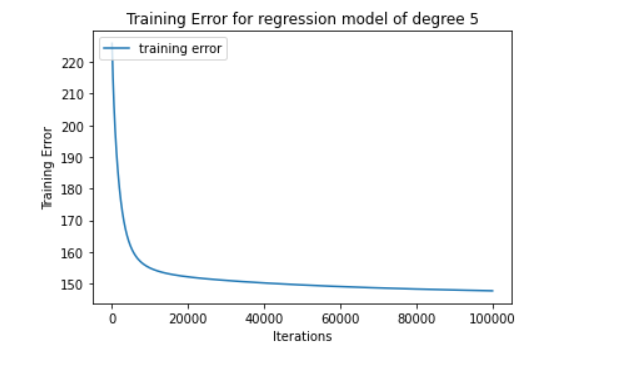


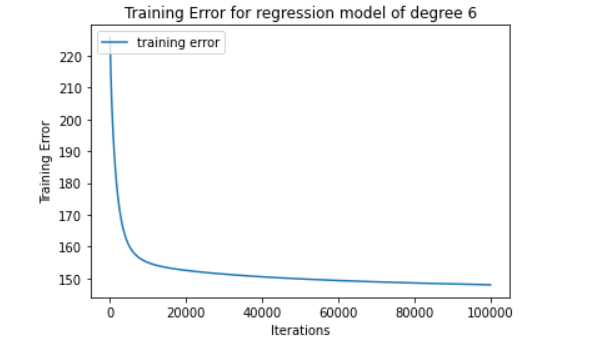


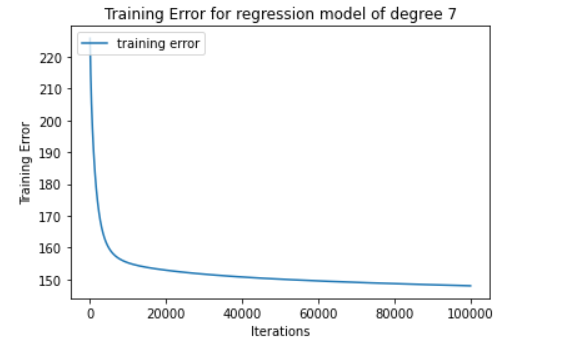


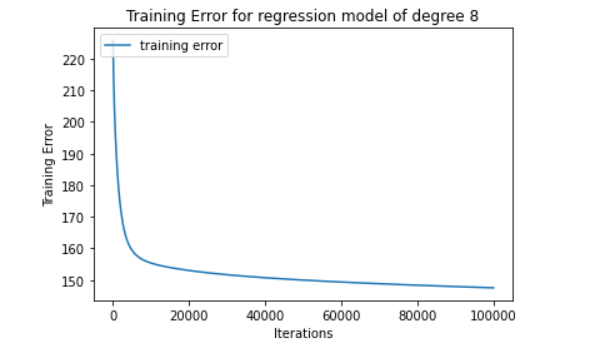


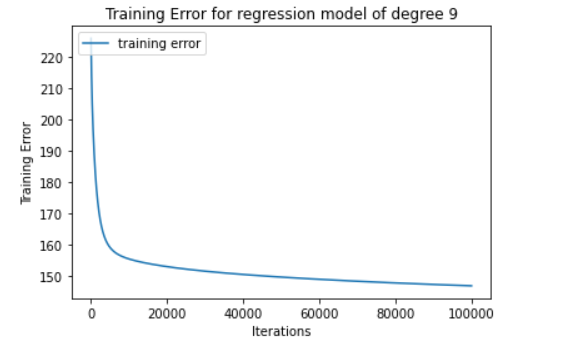


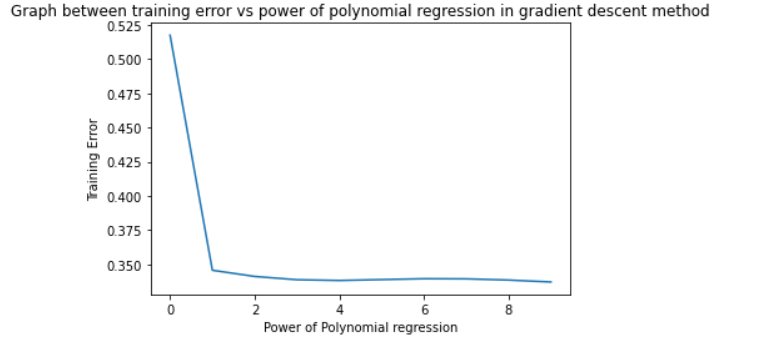


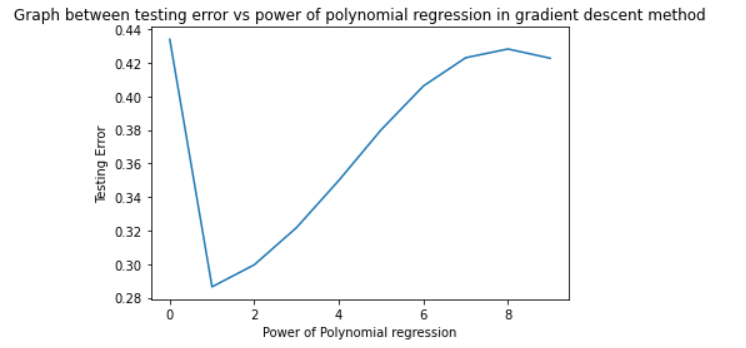


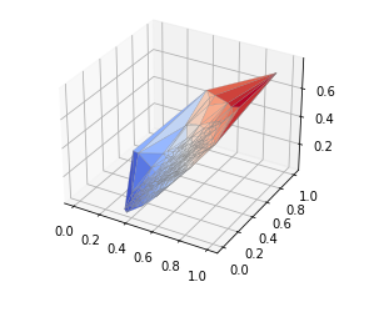












Stochastic gradient descent

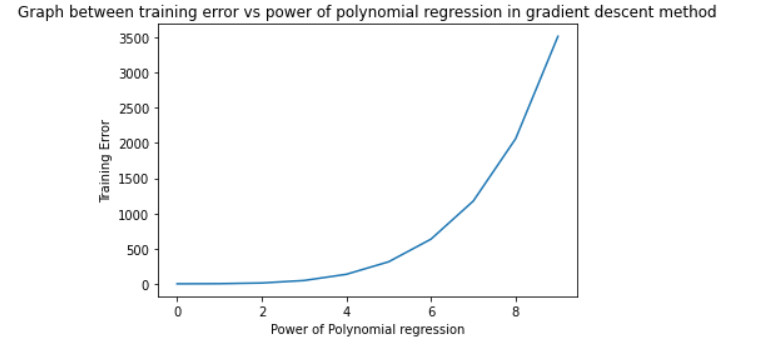
The modelling of the algorithm is like gradient descent and its application is same as gradient descent.

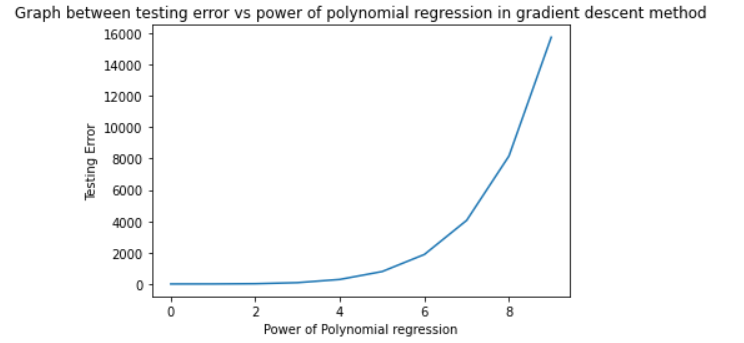
The best fit polynomial comes for the polynomial of degree 0. But there’s a possibility it can vary as in stochastic gradient descent we don’t have anything fixed, we take random data points and then apply the gradient descent.

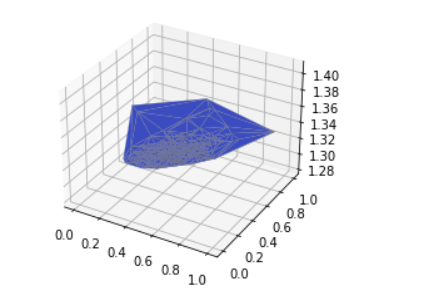
The following graphs are plotted for training and testing error against flexibility.

And for the provided and calculated weights we have calculated and plotted the surface plot, which is provided below.

|  |  |  |
| --- | --- | --- |
| **Degree** | **Training Error** | **Testing Error** |
| **Zero** | **0.930691669041752** | **0.8136089995996366** |
| **One** | **2.7813111521252596** | **2.7553226013830328** |
| **Two** | **12.591900633735824** | **18.11786980160741** |
| **Three** | **47.85455649290409** | **85.93266531021527** |
| **Four** | **135.63280140256703** | **290.5827526214352** |
| **Five** | **314.69956778646537** | **796.3387580761696** |
| **Six** | **635.2307206907583** | **1885.7696102165303** |
| **Seven** | **1178.728054970677** | **4067.3510450425038** |
| **Eight** | **2062.986139205537** | **8174.724421876133** |
| **Nine** | **3505.711238342265** | **15717.768583994433** |







Comparative analysis of the optimal regularized function

The comparative analysis is carried out for different values of the lambda and powers and then calculated the best fit polynomial and its regularized part.

We have come to a result that for the value of lambda = 0.001 we have the same value of training and testing error that means that on imposing a penalty that is regularization term we have decreased the testing error and increased the training error and at the value of lambda where we get the same training and testing error, we have the perfect lambda.

For our code snippet output for different values of lambda and different values of power, we got that no matter what the value of power is for lambda=0.001 we get training and testing error to be same and minimum.

|  |  |  |
| --- | --- | --- |
| **Q(power)** | **Training Error** | **Testing Error** |
| **0.5** | **90.54703288** | **90.47034991** |
| **1** | **90.52400873** | **90.47035742** |
| **2** | **90.49690731** | **90.47036192** |
| **4** | **90.47712043** | **90.47035522** |