FOUNDATIONS OF DATA SCIENCE

Assignment Report
CS F320 (FoDS)

Submitted By:

Siddhant Panda	2020A7PS0264H
Dev Bansal	2020A7PS2051H
Kartikey Goel	2020A7PS2070H

Assignment 1-A: Regression without regularization

Task 1: Data Pre-processing

A function was written to normalize the feature variable using the formula: X'=(X-μ)/σ where μ represents the mean of feature value, and σ represents the standard deviation of feature values as shown below.

Normalization

```
Xbar = df['X'].mean()
Xstd = df['X'].std()
for i in range(len(df)):
    df.loc[i,"X"] = (df.iloc[i]["X"]-Xbar)/Xstd
```

✓ 0.1s

After this, we had to shuffle the dataset and create a random 80-20 split to aid in training and testing. We further split the train set and test set into X train, y train, X test and y test.

```
shuffle_df = df.sample(frac=1).reset_index(drop=True)
#defining size of training set
train_size = int(0.8 * len(df))
train_set = shuffle_df[:train_size]
test_set = shuffle_df[train_size:]
```

✓ 0.0s

With this, our first task has been completed.

Task 2: Polynomial Regression

We now apply gradient descent to develop models for polynomials of degree 1 to degree 9. We choose the degree of polynomial that gives us the least testing error as the best-fit classical polynomial regression model.

```python
def polynomial_regression(x_train, y_train, x_test, y_test, degree, iterations=500, l_rate = 0.01):
    train_error_list = []
    test_error_list = []
    weights = np.ones(degree+1)
    features_train = generate_features(x_train, degree)
    features_test = generate_features(x_test, degree)
    for i in range(iterations):
        y_train_pred = np.dot(features_train, weights)
        train_mse = MSE(y_train, y_train_pred)
        train_error_list.append(train_mse)

        y_test_pred = np.dot(features_test, weights)
        test_mse = MSE(y_test, y_test_pred)
        test_error_list.append(test_mse)

        error = y_train_pred - y_train
        gradients = np.dot(features_train.T, error)/len(y_train)
        weights = weights - l_rate*gradients
    return weights, train_error_list, test_error_list
```

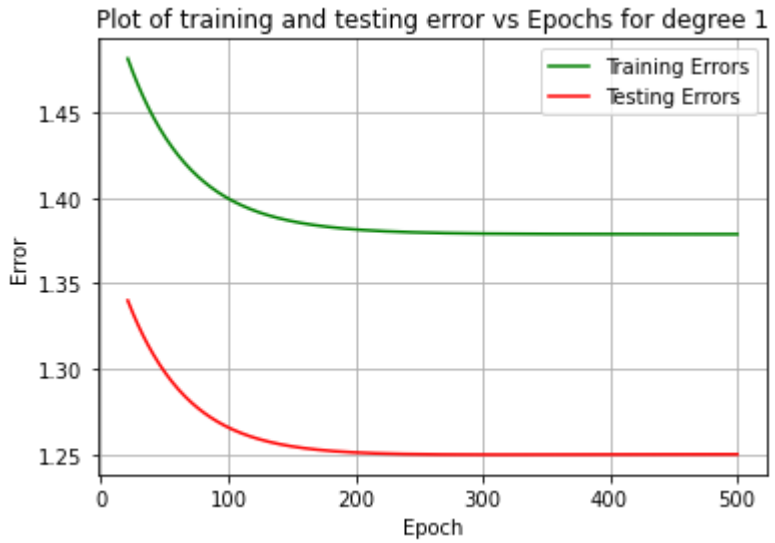
- First of all, we will generate the polynomial features corresponding to a degree of the polynomial which serves as the new feature vector.

- After that, in one iteration, this function is majorly performing the following things:

    1. Calculating the errors and performing gradient descent.
    2. Calculating the gradients and updating the weights.

Using batch gradient descent to converge to the optimal weights for polynomials of degree 1 to degree 9 we get the following graphs and results:

<u>Degree = 1:</u>

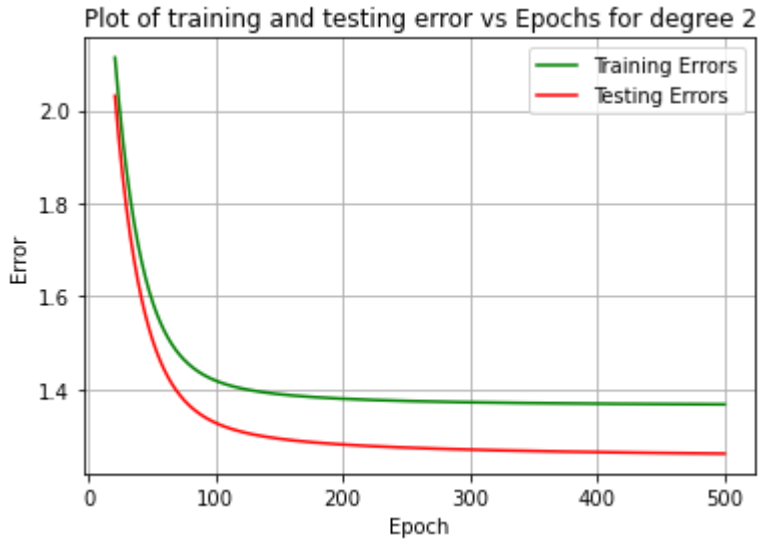


The optimal weights obtained after performing polynomial regression using gradient descent with a polynomial of degree 1 are:

X0 = 0.9546765355709251
X1 = 1.3859150921267822

- The training error obtained with polynomial of degree 1 is: 1.3787698347693769
- The testing error obtained with polynomial of degree 1 is: 1.250298758887331

Degree = 2:

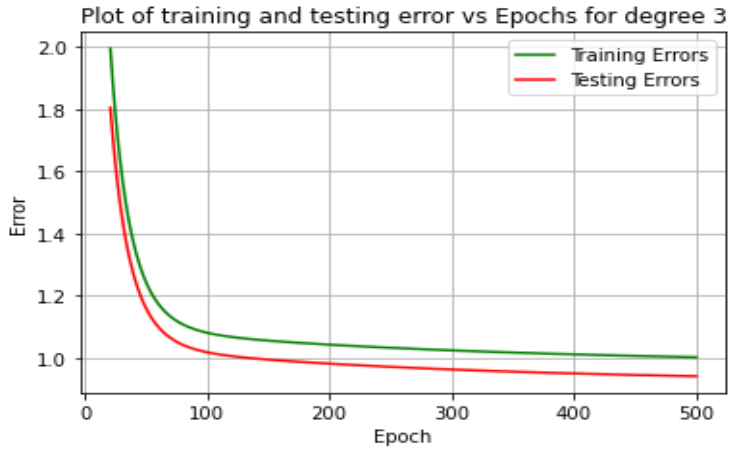


The optimal weights obtained after performing polynomial regression using gradient descent with a polynomial of degree 2 are:

X0 = 0.7735629561663651
X1 = 1.3843526205645305
X2 = 0.16192099749456706

- The training error obtained with polynomial of degree 2 is: 1.367689831273718
- The testing error obtained with polynomial of degree 2 is: 1.2616631400561245

Degree = 3:

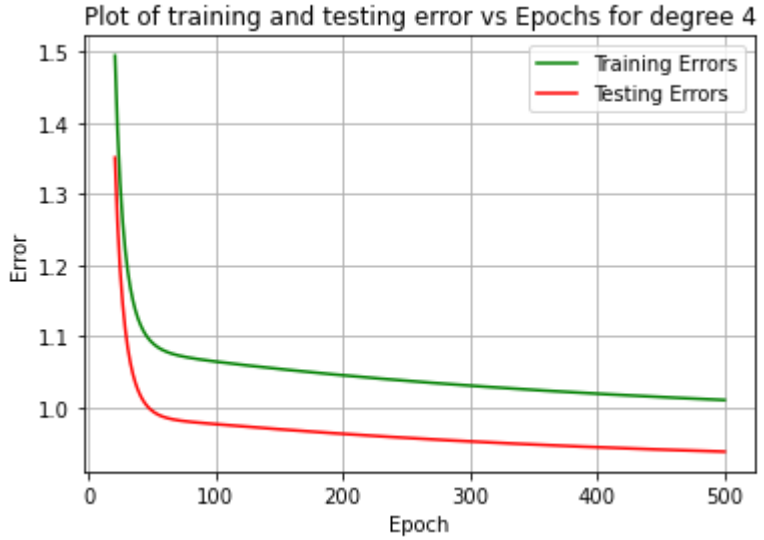


The optimal weights obtained after performing polynomial regression using gradient descent with a polynomial of degree 3 are:

X0 = 0.7739650184936157
X1 = 0.3543641311266618
X2 = 0.15604488430740346
X3 = 0.5969300964781201

- The training error obtained with polynomial of degree 3 is: 1.003098045186235
- The testing error obtained with polynomial of degree 3 is: 0.9420649822147191

Degree = 4:

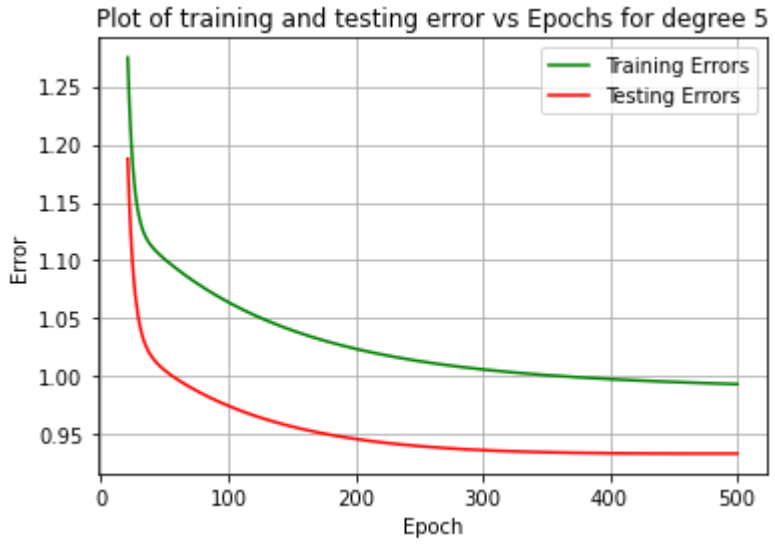


The optimal weights obtained after performing polynomial regression using gradient descent with a polynomial of degree 4 are:

X0 = 0.7270403204118115
X1 = 0.35562578799078687
X2 = 0.3651728739675003
X3 = 0.5966240611681146
X4 = -0.08254255325159551

- The training error obtained with polynomial of degree 4 is: 1.010649909544233
- The testing error obtained with polynomial of degree 4 is: 0.9381266532154271

Degree = 5:

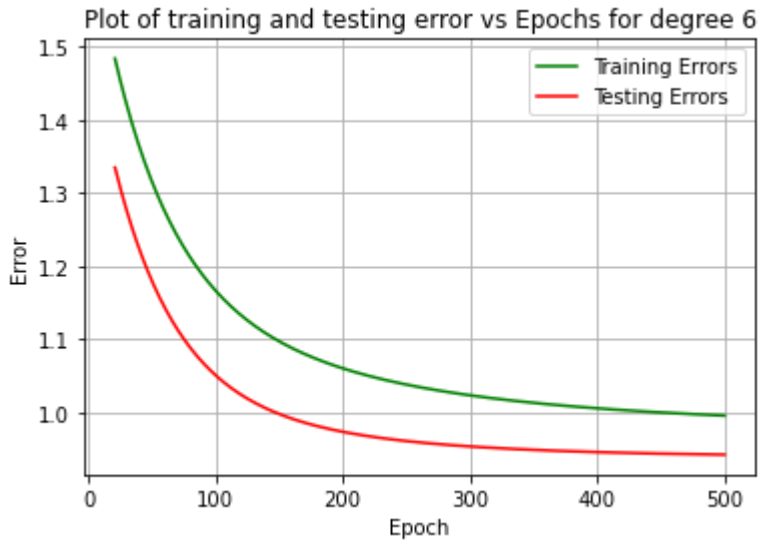


The optimal weights obtained after performing polynomial regression using gradient descent with a polynomial of degree 5 are:

X0 = 0.7260129554700817
X1 = 0.35846905158782266
X2 = 0.3630283928190786
X3 = 0.3503939385853993
X4 = -0.08153126765689343
X5 = 0.10755633374506245

- The training error obtained with polynomial of degree 5 is: 0.9932871715063325
- The testing error obtained with polynomial of degree 5 is: 0.9333674126132698

Degree = 6:

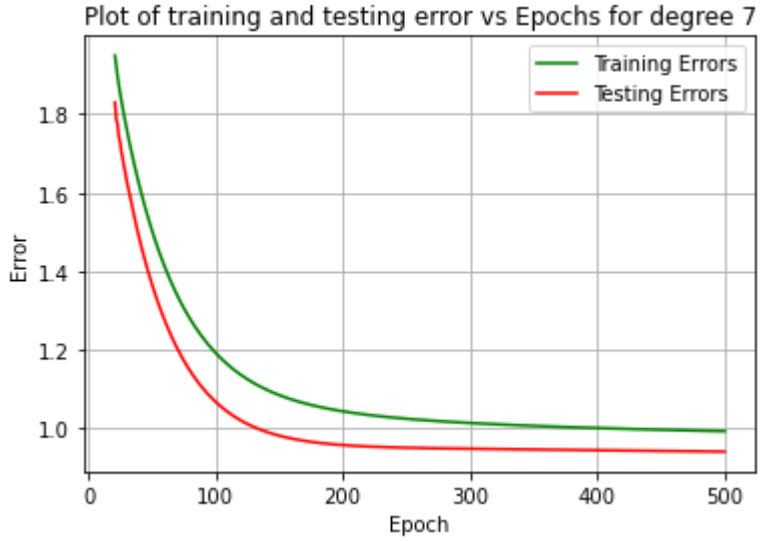


The optimal weights obtained after performing polynomial regression using gradient descent with a polynomial of degree 6 are:

X0 = 0.6927824374303828
X1 = 0.35923895314570947
X2 = 0.2661071485819632
X3 = 0.3524488536519513
X4 = 0.02454687191790374
X5 = 0.10644947867410141
X6 = -0.021942274322512972

- The training error obtained with polynomial of degree 6 is: 0.9953497599587396
- The testing error obtained with polynomial of degree 6 is: 0.9419307980455683

Degree = 7:

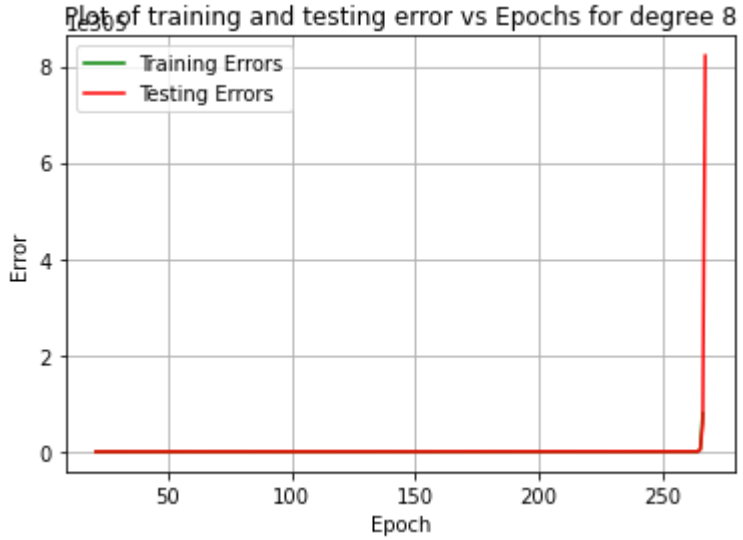


The optimal weights obtained after performing polynomial regression using gradient descent with a polynomial of degree 7 are:

X0 = 0.6921252797474017
X1 = 0.3299866425102023
X2 = 0.2659116552721364
X3 = 0.30916488035280865
X4 = 0.02466956568292743
X5 = 0.1587138386794249
X6 = -0.02194829054866524
X7 = -0.01171171886001146

- The training error obtained with polynomial of degree 7 is: 0.9925373383873187
- The testing error obtained with polynomial of degree 7 is: 0.9405064091920073

Degree = 8:

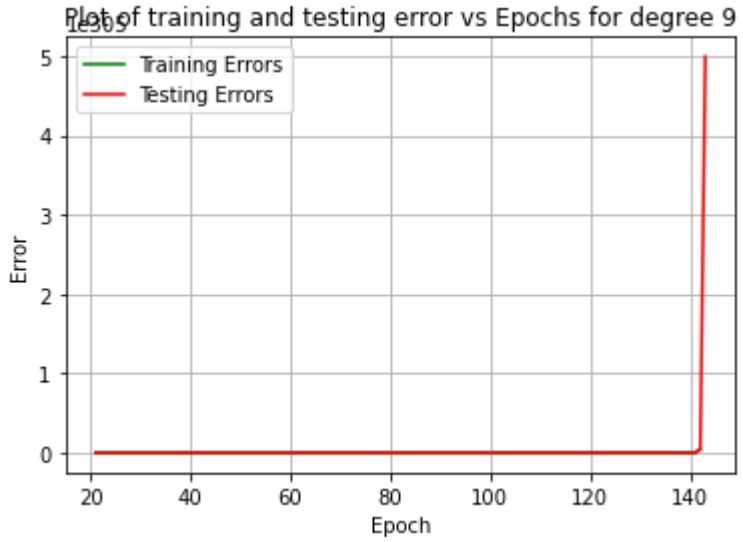


The optimal weights obtained after performing polynomial regression using gradient descent with a polynomial of degree 8 are:

X0 = 2.0522977534669365e+283
X1 = 5.21609943184956e+281
X2 = 5.00579624359764e+283
X3 = 1.138237805924196e+282
X4 = 1.2680054860394975e+284
X5 = 2.2278474625887816e+282
X6 = 3.293686437483717e+284
X7 = 3.587220230685359e+282
X8 = 8.713363332425446e+284

- The training error obtained with polynomial of degree 8 is: **NaN**
- The testing error obtained with polynomial of degree 8 is: **NaN**

Degree = 9:



The optimal weights obtained after performing polynomial regression using gradient descent with a polynomial of degree 9 are:

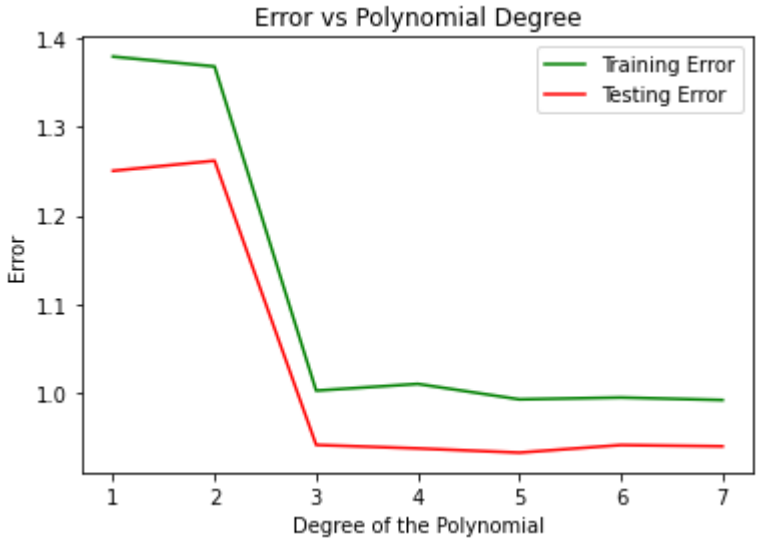
X0 = nan
X1 = nan
X2 = nan
X3 = nan
X4 = nan
X5 = nan
X6 = nan
X7 = nan
X8 = nan
X9 = nan

- The training error obtained with polynomial of degree 9 is: **NaN**
- The testing error obtained with polynomial of degree 9 is: **NaN**

Observation:
In the case of polynomials with degrees 8 and 9, the model's weight parameters experience significant escalation due to the **absence of regularization**. Consequently, these models exhibit pronounced signs of **overfitting,** leading to the observation that weight values approach infinity or NaN values.

Task 3: Graph Plotting

a. Plot 1- Final Training and Testing Errors v/s degree of polynomial

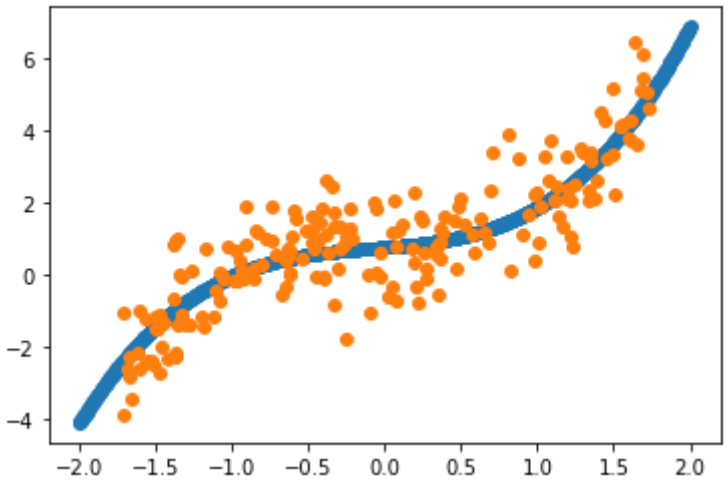


A notable reduction in both training and testing errors becomes apparent when the polynomial degree is elevated to 3. However, as we continue to increase the polynomial degree beyond this point, there is no substantial reduction in either training or testing errors. Additionally, it is evident that the models exhibit signs of overfitting when polynomial degrees reach 8 and 9.
Consequently, we can infer that the **most optimal** model for fitting the provided dataset is **a polynomial of degree 3**.

b. Shown in Task 2

c. The best polynomial fitted curve on the data points:

As inferred from the "Final Training and Testing Errors v/s degree of polynomial" plot, we will choose a polynomial of degree 3 as the best polynomial fitted curve.



Task 4: Comparative Analysis

1.For polynomials through degree 1 to 3, the errors steadily decrease, after which it remains fairly constant throughout till degree 7.

2. **At degree 3, there is a noticeable reduction in both training and testing errors, and the model achieves relatively low errors.** This may indicate an optimal trade-off between model complexity and fit to the data.

3. However, when the polynomial degree reaches 8, the weights explode to extremely large values (approaching infinity), resulting in infinite training and testing errors. This indicates severe overfitting, where the model fits the training data perfectly but fails to generalize to new data. This suggests that increasing the degree beyond 7 doesn't provide significant benefits in terms of reducing testing error and may lead to overfitting.

Problem : Now, the question comes to what degree to choose among degree 3, 4, 5, 6 and 7 as all of them give almost the same kinds of error values :

Solution : We choose a degree 3 model, because it makes sense to use a **less complex model** - which in turn makes sure that our model is more generalizable rather than extremely specific to our data.

Assignment 1-B: Polynomial Regression with regularization

Task 1: Data Pre-processing

A function was written to normalize the feature variable using the formula: X'=(X-μ)/σ where μ represents the mean of feature value, and σ represents the standard deviation of feature values as shown below.

Normalization

```
Xbar = df['X'].mean()
Xstd = df['X'].std()
for i in range(len(df)):
    df.loc[i,"X"] = (df.iloc[i]["X"]-Xbar)/Xstd
```

✓ 0.1s

We were told to replace NaN/null values with the mean of the existing values of the corresponding feature vector but we didn't see any null value in the dataset and that's why we will skip this.

```
null_counts = df.isna().sum()
null_counts
```

```
Height    0
Width     0
Weight    0
dtype: int64
```

After this, we had to shuffle the dataset and create a random 80-20 split to aid in training and testing. We further split the train set and test set into X train, y train, X test and y test.

```
shuffle_df = df.sample(frac=1).reset_index(drop=True)
#defining size of training set
train_size = int(0.8 * len(df))
train_set = shuffle_df[:train_size]
test_set = shuffle_df[train_size:]
```

✓ 0.0s

Task 2: Polynomial Regression

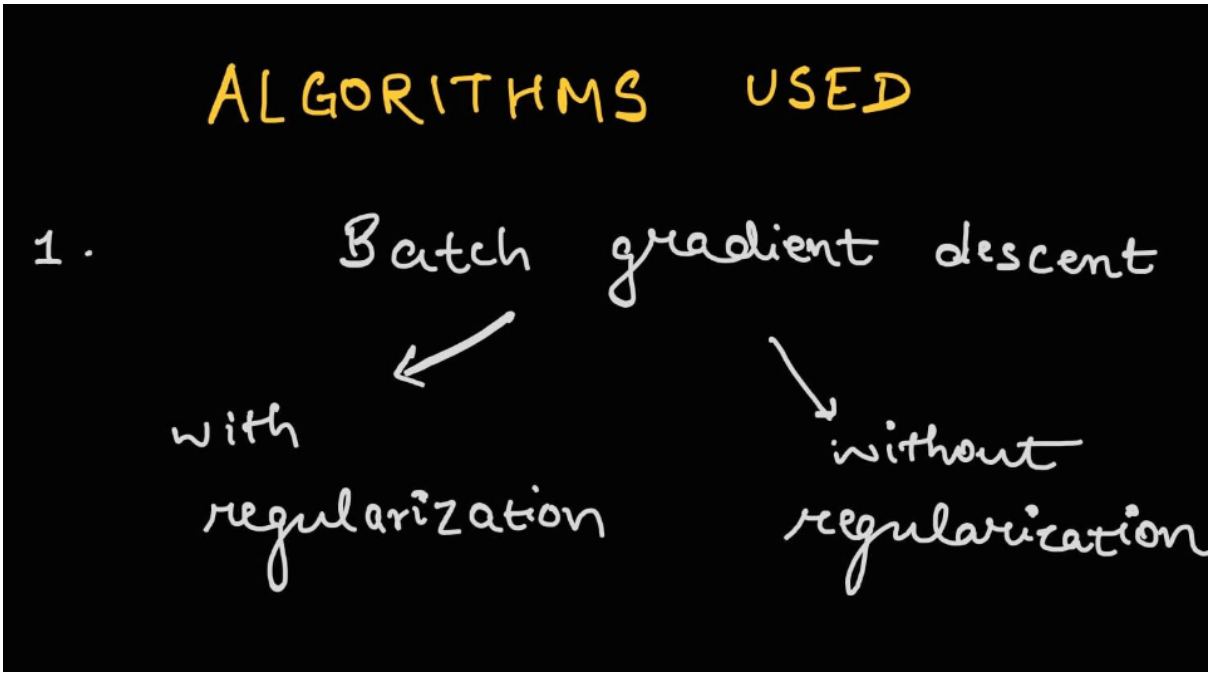
A brief description of the model that we have developed is given below:

- We have created separate functions which perform the following tasks:

• Generate the polynomial features given the degree of the polynomial and the data frame.
• Predict the output given the input features x1 and x2.
• Determine the cost or loss given the features, weights (parameters) and the observed values of the target variable.
• Helper function that determines the regularization correction or penalty given the norm and the weights along with the lambda value.
• Batch gradient descent algorithm to converge to the optimal weights or parameters.
• Stochastic gradient descent algorithm to converge to the optimal weights or parameters.
• Batch gradient descent algorithm incorporated with regularization where the type of norm is taken as input along with the lambda value.
• Stochastic gradient descent algorithm incorporated with regularization where the type of norm is taken as input along with the lambda value.

All these tasks performed in the order in which they have been written gives us our required model.
We will explain each step as and when we come across that step.

Algorithms used:

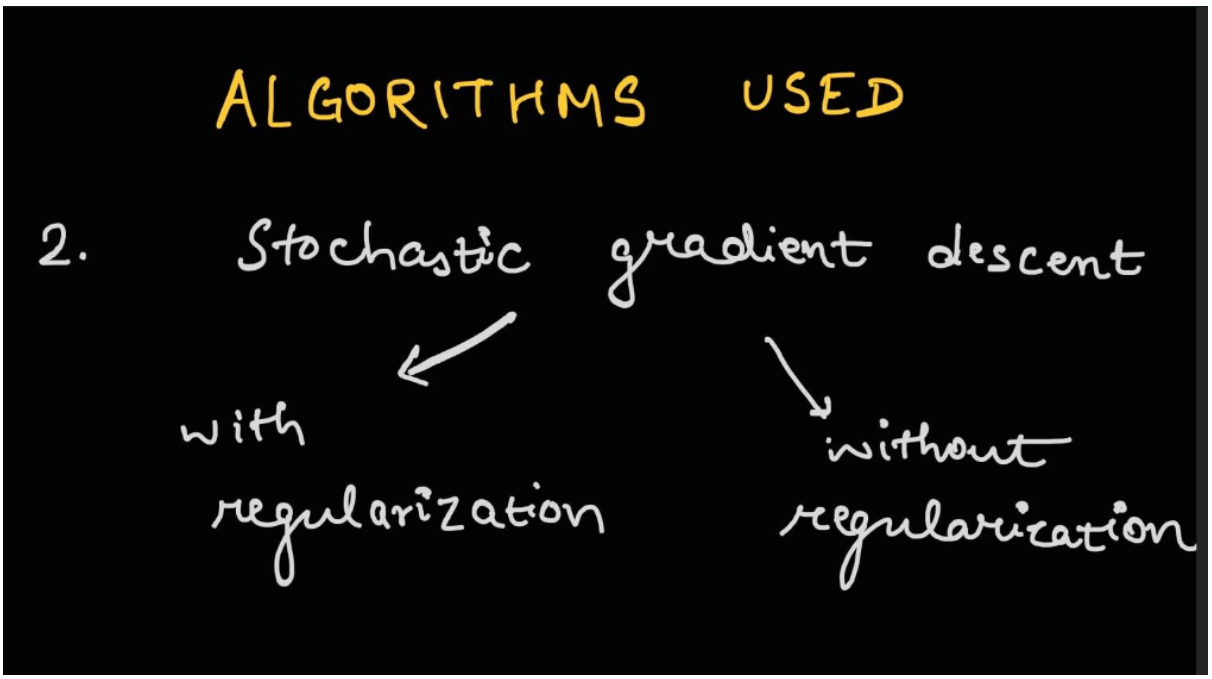


Batch gradient descent algorithm:

The batch gradient descent algorithm computes gradients by considering all data points from the dataset in each iteration. Consequently, it converges to the optimal weights at a relatively slow pace.

Batch gradient descent algorithm with regularization:

In the context of the batch gradient descent algorithm with regularization, the weight updates involve the incorporation of a regularization penalty term. To accomplish this, we initially compute the gradient using either gradient descent or stochastic gradient descent. Subsequently, we augment the gradient with the regularization penalty term, and finally, we subtract the entire augmented term from the weights to obtain the updated weights. It is essential to note that during the regularization process, the bias term is not subject to regularization.



Stochastic gradient descent algorithm:

The stochastic gradient descent (SGD) algorithm computes gradients by utilizing only a single randomly chosen data point from the dataset in each iteration. As a result, it converges to the optimal weights significantly faster than the batch gradient descent algorithm.

Stochastic gradient descent algorithm with regularization:

Stochastic Gradient Descent (SGD) with regularization is an iterative optimization algorithm used to train machine learning models. It combines the benefits of SGD's faster convergence with regularization techniques to prevent overfitting. In each iteration, a random data point (or mini-batch) is selected from the training set. The algorithm calculates the gradient of the loss function with respect to model parameters, adds a regularization term (such as L1 or L2) to the gradient, and then updates the model's weights. The regularization term encourages smaller parameter values, helping to control model complexity and improve generalization. By incorporating regularization, SGD with regularization strikes a balance between fitting the training data and avoiding excessive model complexity.

Regularization Penalty algorithm:

Regularization

```
def regularization_penalty(params, q, lamda):
    if q == 0.5:
        return np.reciprocal(np.sqrt(abs(params))) / 2
    elif q == 1:
        sign = np.sign(params)
        return lamda * sign
    elif q == 2:
        return lamda * params
    elif q == 4:
        return 2 * lamda * (params ** 2)
```

The above function gives the value of regularization penalty given different values of q and lambda along with the current weights.
Here, *q is the regularization norm and lambda - the regularization parameter*.

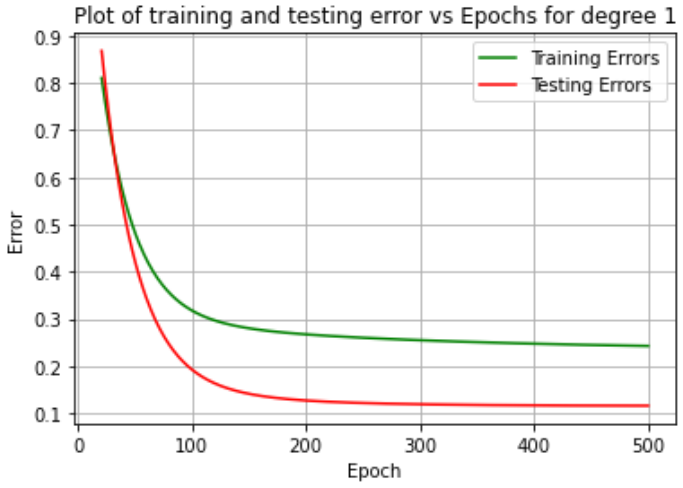
Task 2 (Part A):

We now apply gradient descent and stochastic gradient descent to develop models for polynomials of degree 1 to degree 9. We choose the degree of polynomial that gives us the least testing error as the best-fit classical polynomial regression model.

<u>Using Gradient Descent:</u>

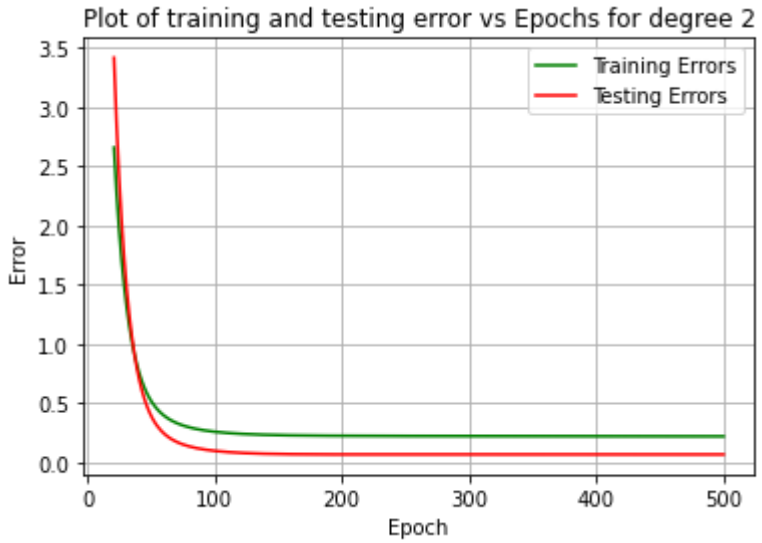
9 different models with degrees varying from 1 to 9 are made using gradient descent algo, the results for which are shown below:

Degree=1:



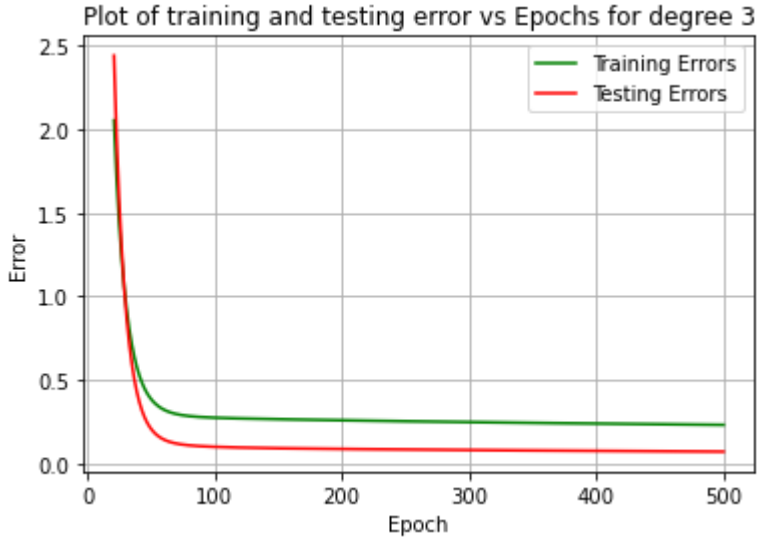
- The training error obtained with polynomial of degree 1 is: 0.24303645819103983
- The testing error obtained with polynomial of degree 1 is: 0.11668684611979932

Degree=2:



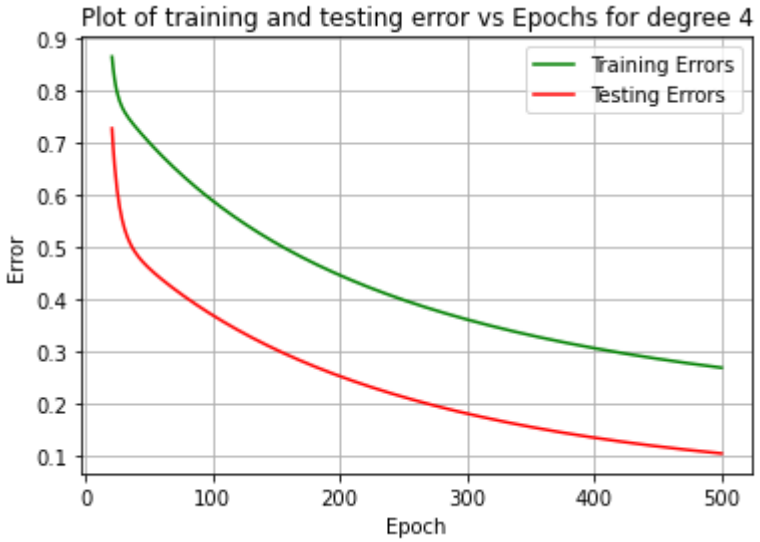
- The training error obtained with polynomial of degree 2 is: 0.21866121441524144
- The testing error obtained with polynomial of degree 2 is: 0.06546380198463866

Degree=3:



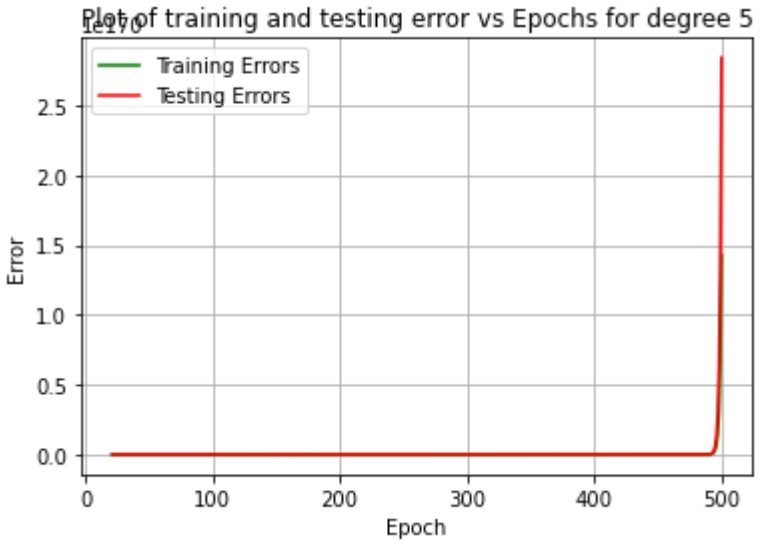
- The training error obtained with polynomial of degree 3 is: 0.23179779576868123
- The testing error obtained with polynomial of degree 3 is: 0.07154564061150342

Degree=4:



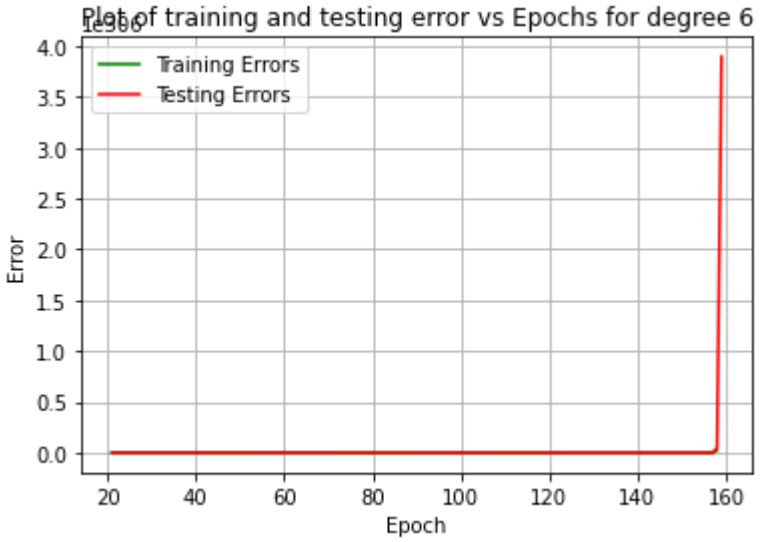
- The training error obtained with polynomial of degree 4 is: 0.2678425260862806
- The testing error obtained with polynomial of degree 4 is: 0.10317603524627356

Degree=5:



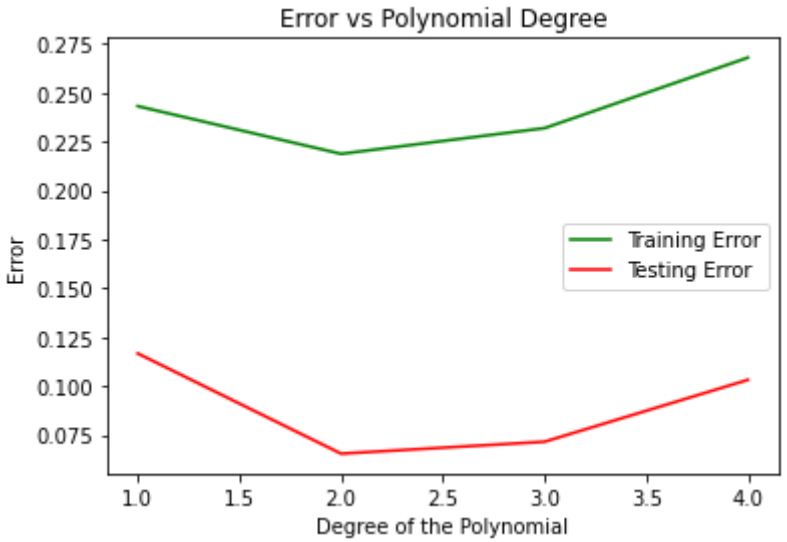
- The training error obtained with polynomial of degree 5 is: 1.4252087256479473e+170
- The testing error obtained with polynomial of degree 5 is: 2.842691389798922e+170

Degree=6:



- The training error obtained with polynomial of degree 6 is: **NaN**
- The testing error obtained with polynomial of degree 6 is: **NaN**

To find the degree of the polynomial which best fits the given data, we will now construct an "Error v/s polynomial degree" graph.



Here, we can see that:

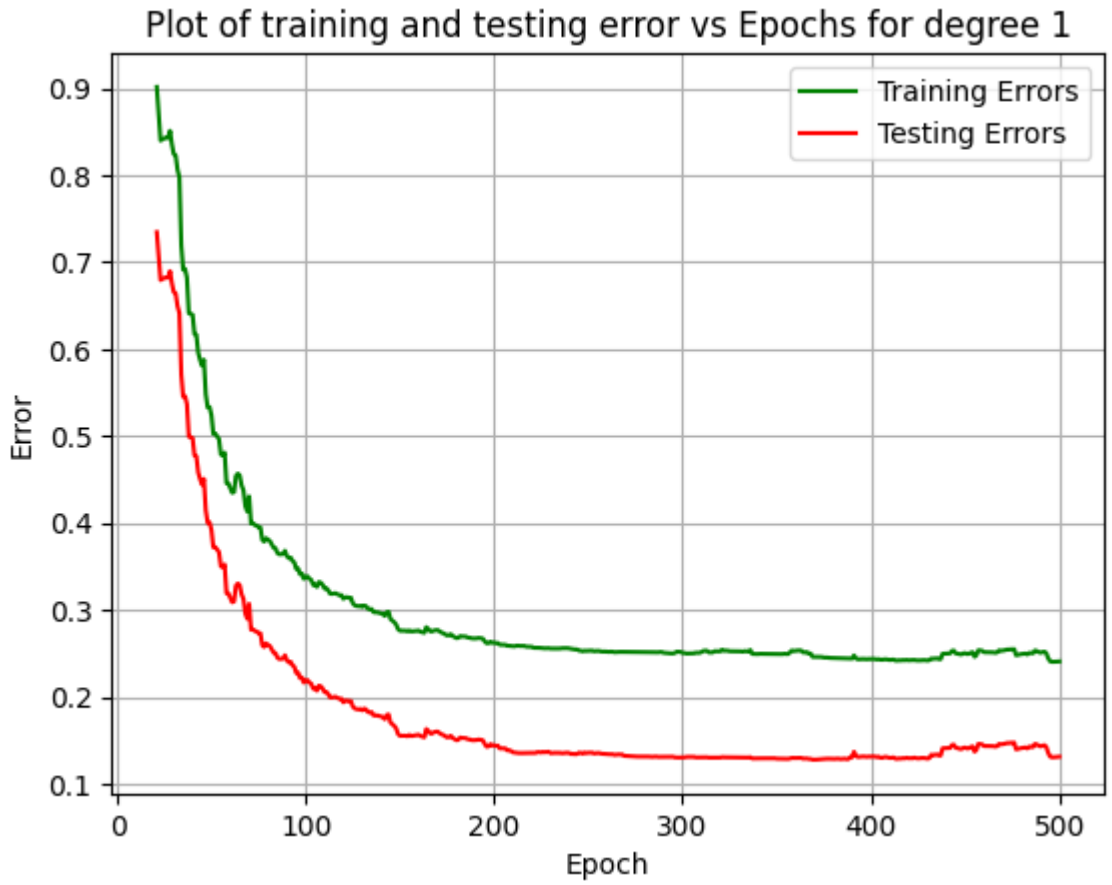
- Polynomials of degree >4 are overfitting. (the value of their weights has blown up)

- Polynomial with degree 2 is giving the least error hence an increase in the degree of the polynomial is resulting in an increase in the training error indicating that a **degree 2 polynomial is a right fit** for the given dataset.

Using Stochastic Descent:

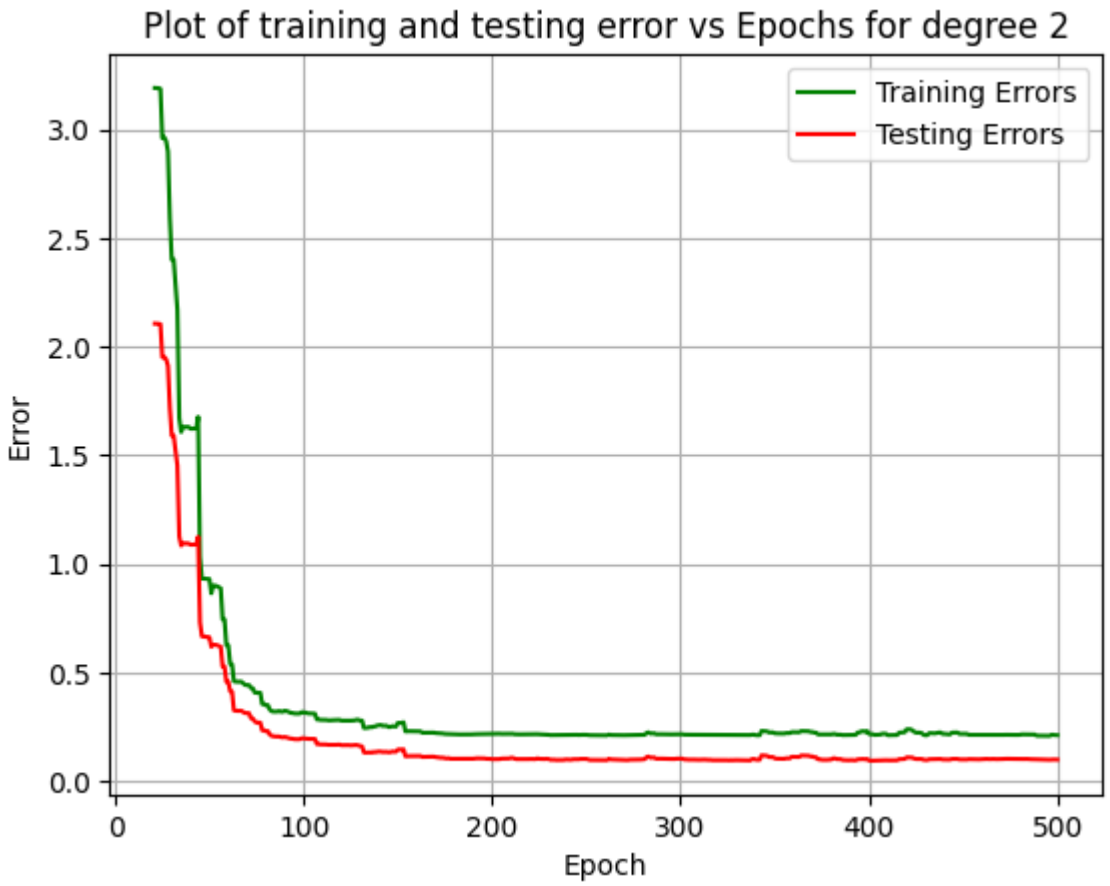
9 different models with degrees varying from 1 to 9 are made using stochastic descent algorithm, the results for which are shown below:

Degree=1:



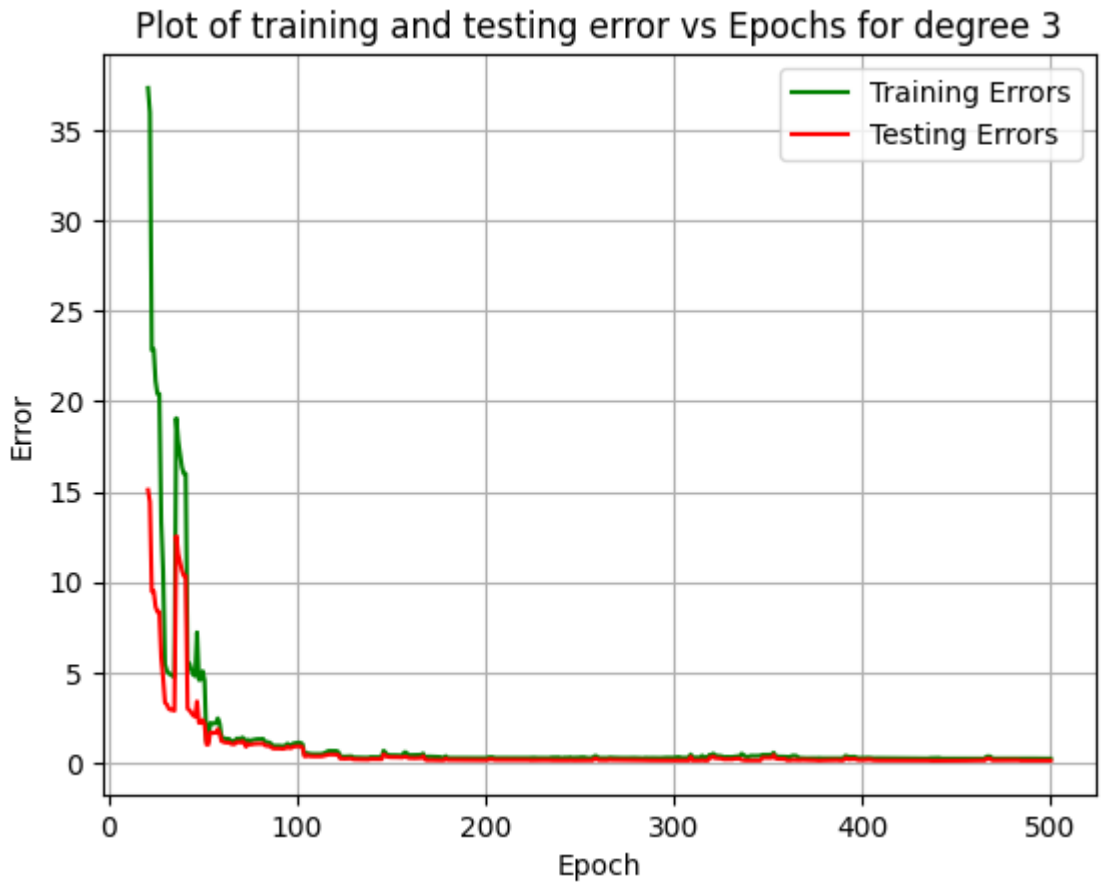
- The training error obtained with polynomial of degree 1 is: 0.24067507520239412
- The testing error obtained with polynomial of degree 1 is: 0.1310685079849066

Degree=2:



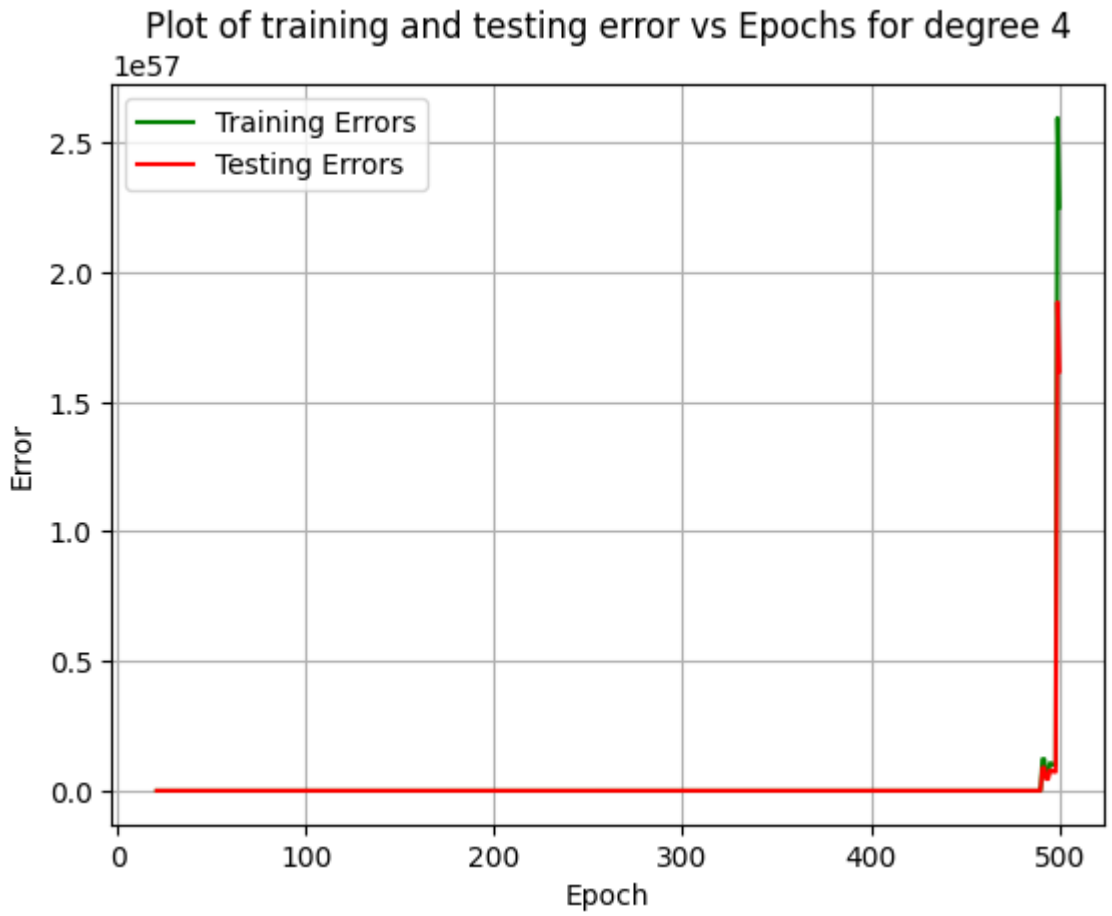
- The training error obtained with polynomial of degree 2 is: 0.2127148668585026
- The testing error obtained with polynomial of degree 2 is: 0.10026623905861128

Degree=3:



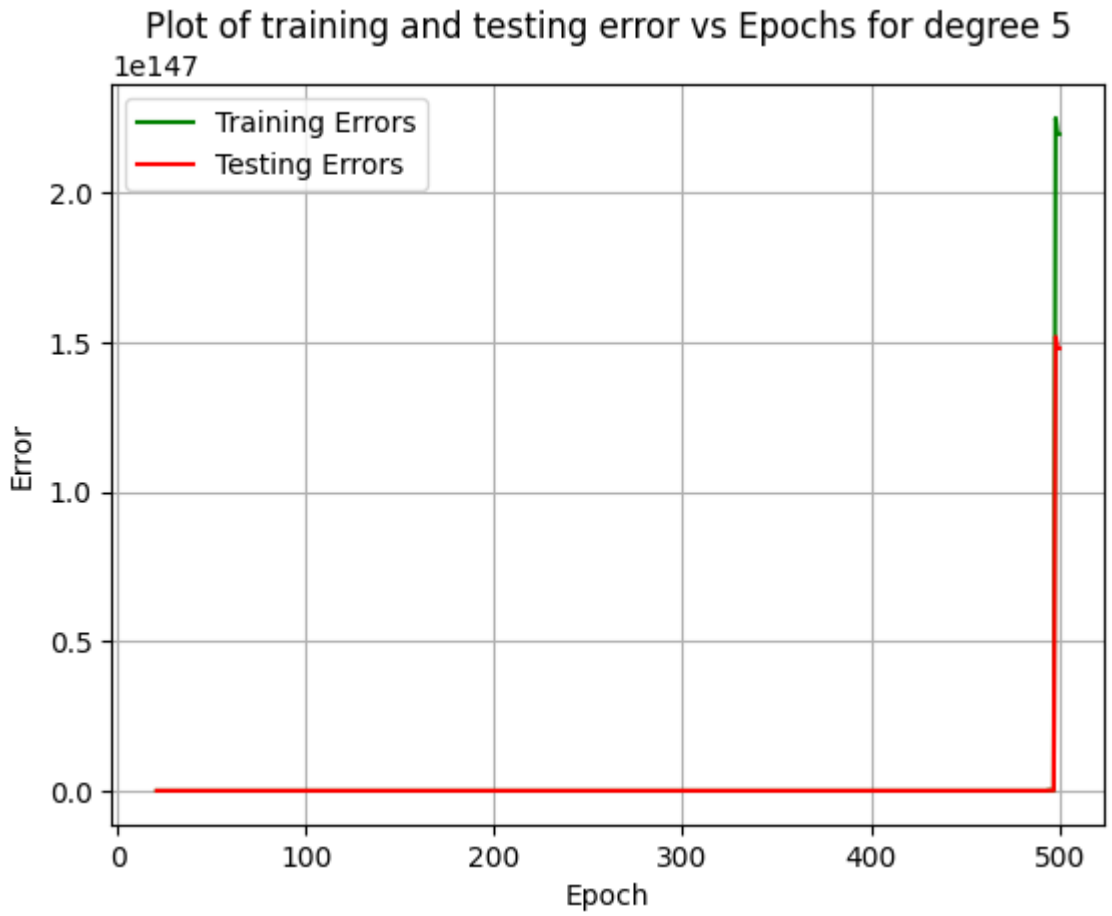
- The training error obtained with polynomial of degree 3 is: 0.2319287123893553
- The testing error obtained with polynomial of degree 3 is: 0.12469911486466233

Degree=4:



- The training error obtained with polynomial of degree 4 is: 2.249894520376113e+57
- The testing error obtained with polynomial of degree 4 is: 1.6174604253849276e+57

Degree=5:

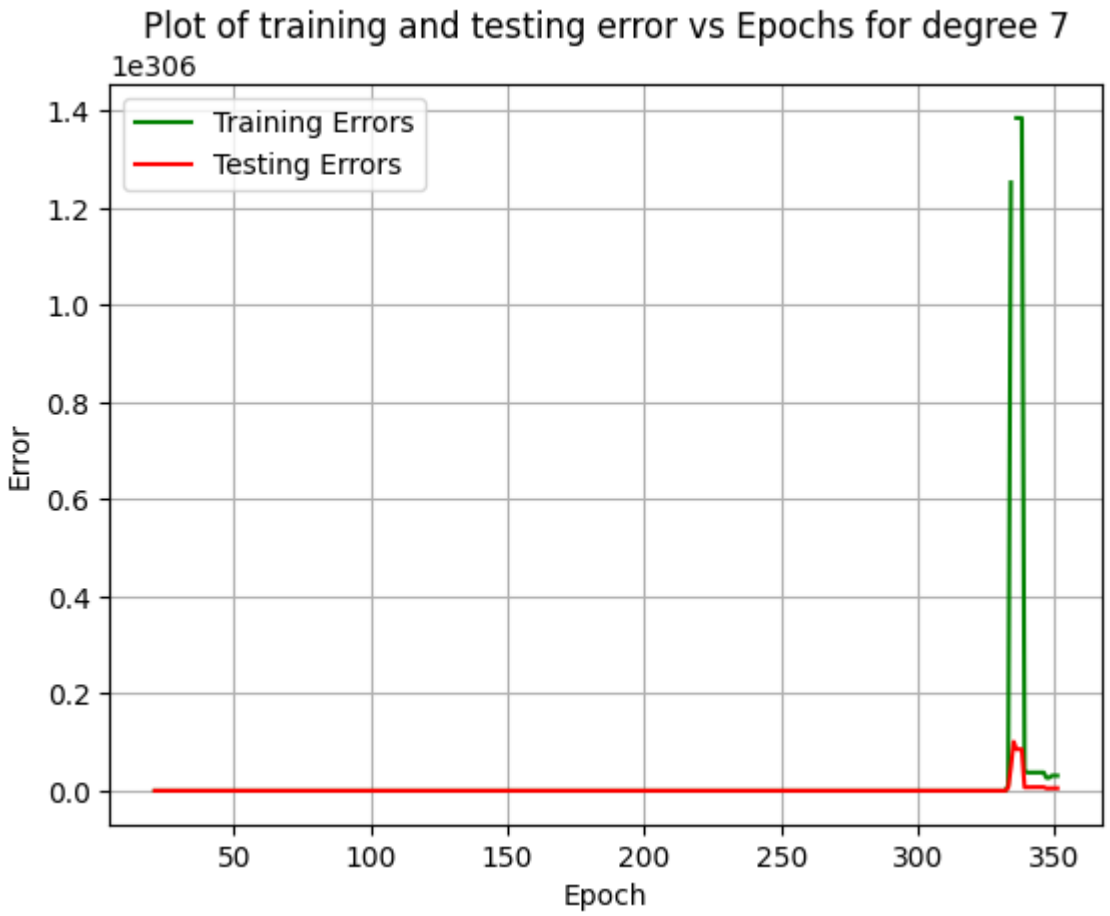


- The training error obtained with polynomial of degree 5 is: 2.1954830606969218e+147
- The testing error obtained with polynomial of degree 5 is: 1.4788872927909303e+147

Degree=6:

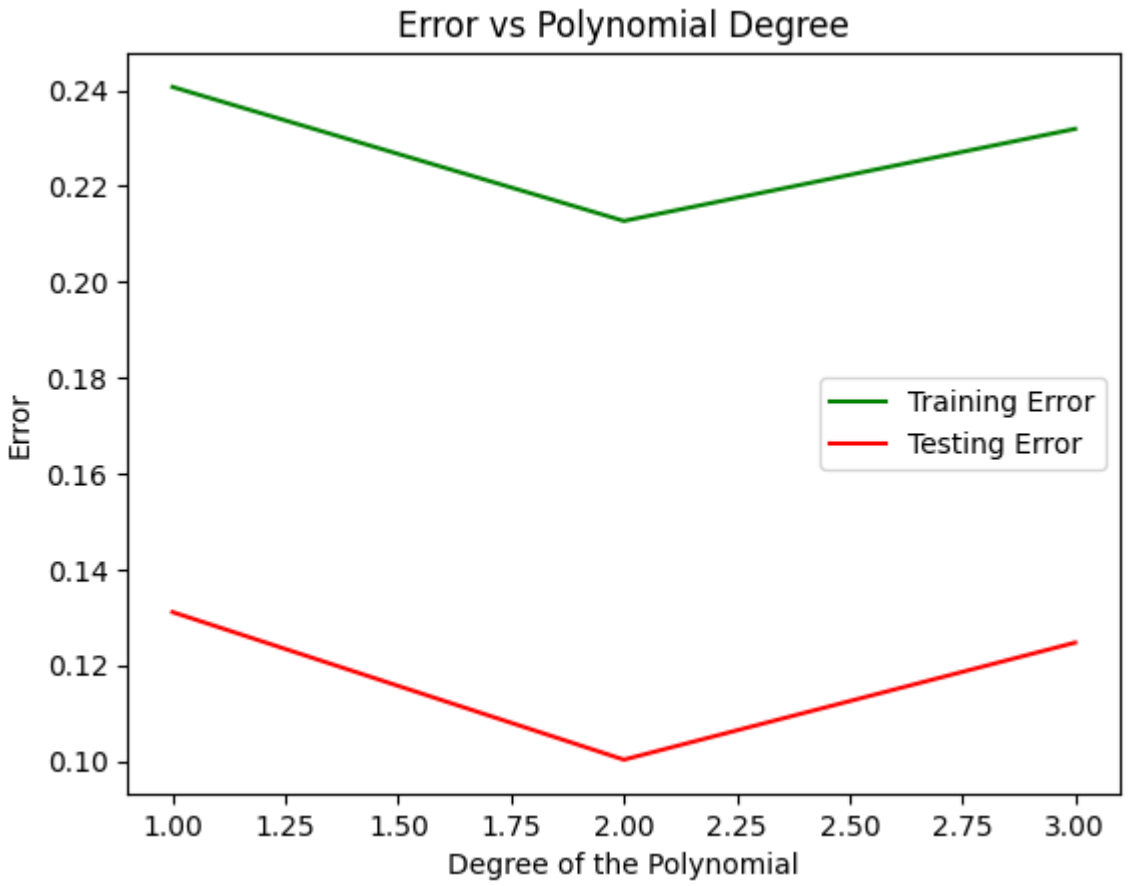
- The training error obtained with polynomial of degree 6 is: inf
- The testing error obtained with polynomial of degree 6 is: 1.4862782407925357e+306

Degree=7:


Plot of training and testing error vs Epochs for degree 7
1e306
Training Errors
Testing Errors
Error
Epoch
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
50
100
150
200
250
300
350

The training error obtained with polynomial of degree 7 is: inf
The testing error obtained with polynomial of degree 7 is: inf

To find the degree of the polynomial which best fits the given data, we will now construct an "Error v/s polynomial degree" graph.



Here, we can see that:

- Polynomials of degree >3 are overfitting. (the value of their weights has blown up)
- Polynomial with degree 2 is giving the least error hence an increase in the degree of the polynomial is resulting in an increase in the training error indicating that a **degree 2 polynomial is a right fit** for the given dataset.

NOTE :

- Though degree 3 polynomial gives even lesser errors than degree 2, we still go ahead to choose degree 2 polynomial because we notice that in degree 3 the training and testing errors both approach very close to 0, this implies that :
    - The model is overfitting.
    - It is not very generalizable, hence might result in our model to perform worse for a completely unrelated data point.

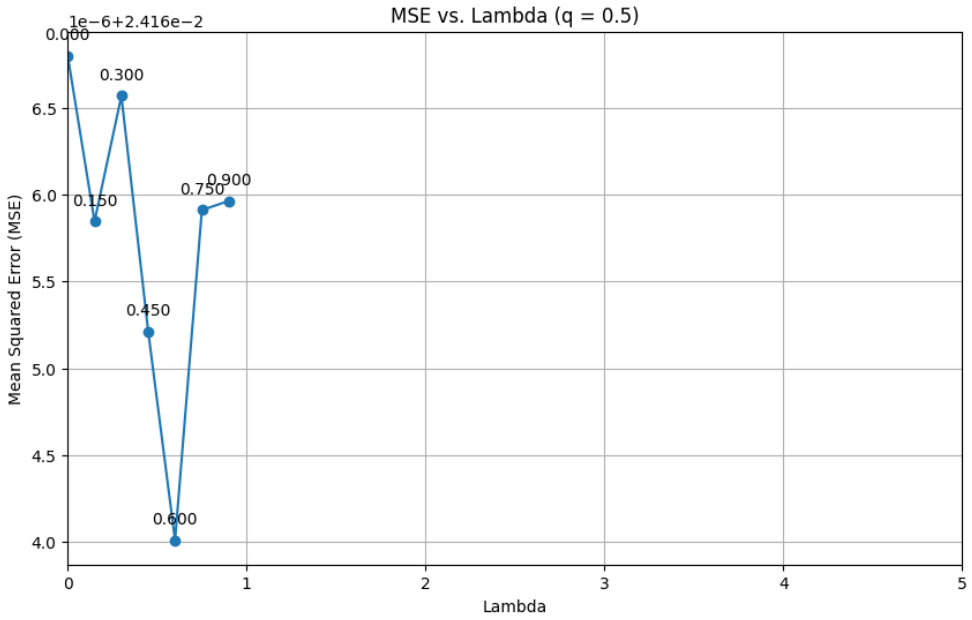
Task 2 (Part B):

For both Stochastic gradient descent and Batch gradient descent, the polynomial of degree 2 is the best fit for the given dataset. So, now we need to make 4 optimal models for degree=2 for q= 0.5,1,2,4 using each descent algorithm.

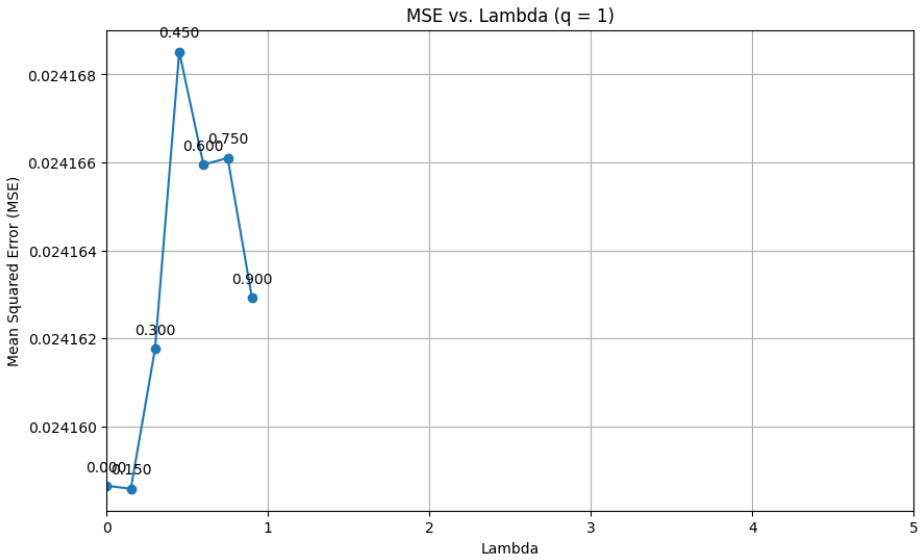
1. Batch gradient descent algorithm

*Let us focus on finding the optimal lambda for each q in **batch gradient** case :*

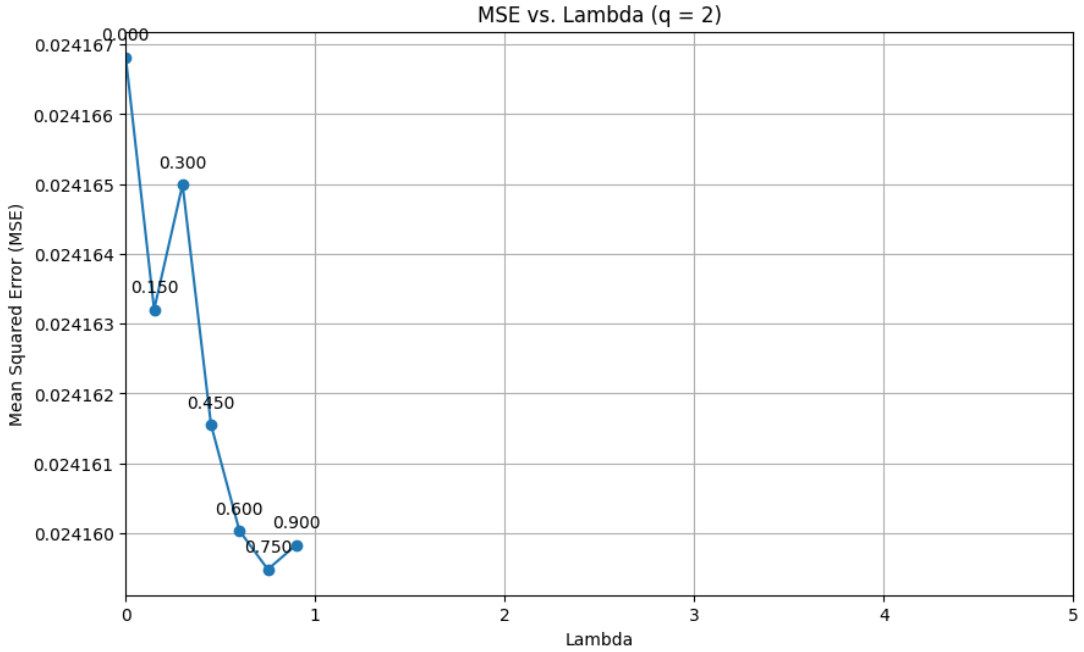
- **q = 0.5** → Best Lambda = 0.6



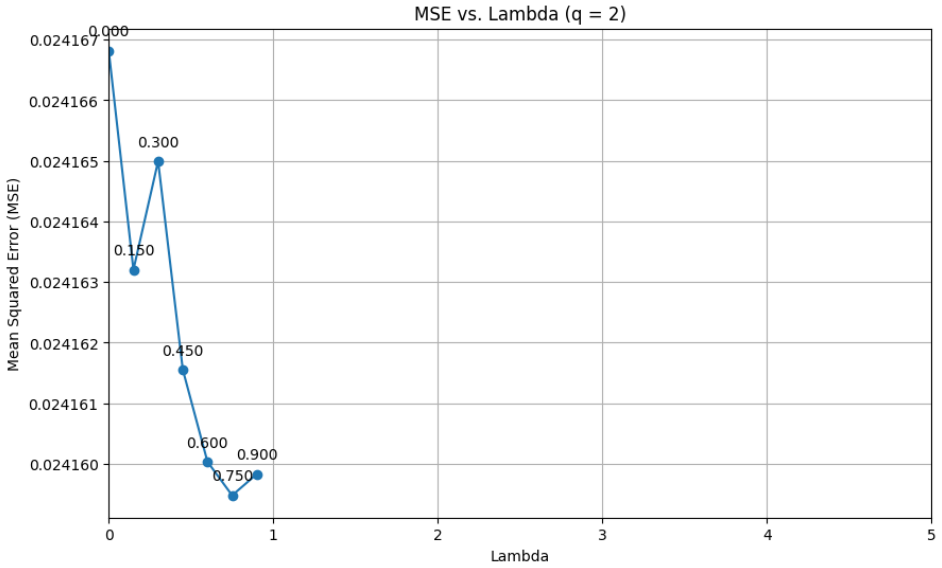
- **q = 1** → Best Lambda = 0.15



- **q = 2** → Best Lambda = 0.75

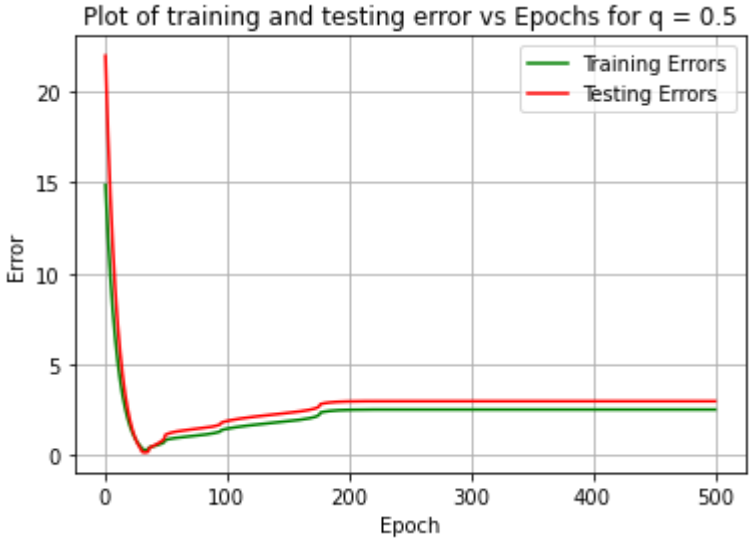


- **q = 4**→ Best Lambda = 0.75



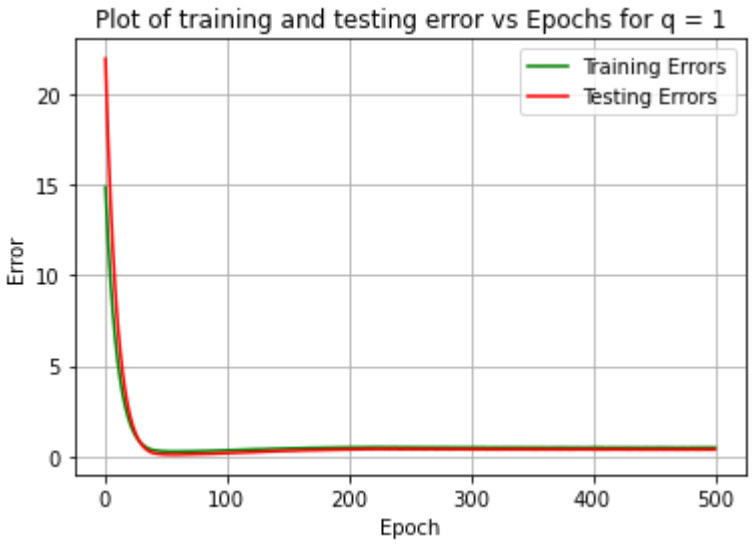
Now let us plot the graphs for each q with optimal lambda :

- **q=0.5:**



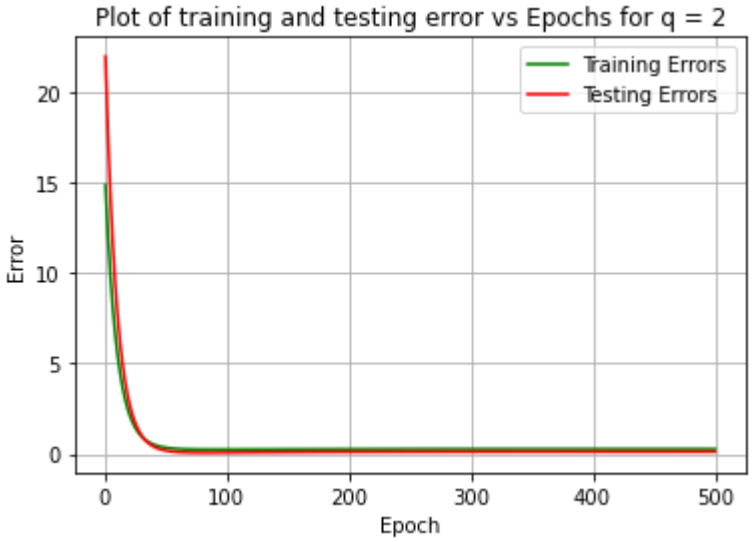
- The training error obtained is: 2.520688955495926
- The testing error obtained is: 2.9822496602827773

- **q=1:**



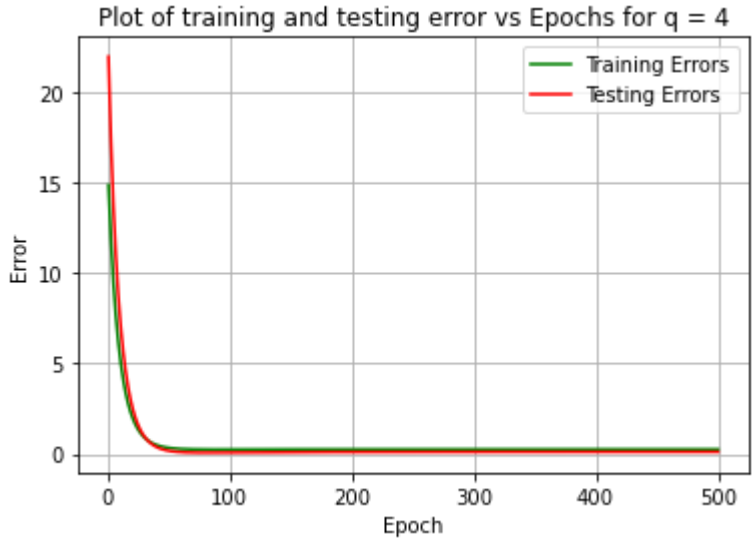
- The training error obtained is: 0.49578985922196234
- The testing error obtained is: 0.3875696719443395

- **q=2:**



- The training error obtained is: 0.269873626508414
- The testing error obtained is: 0.12427385186889041

- **q=4:**

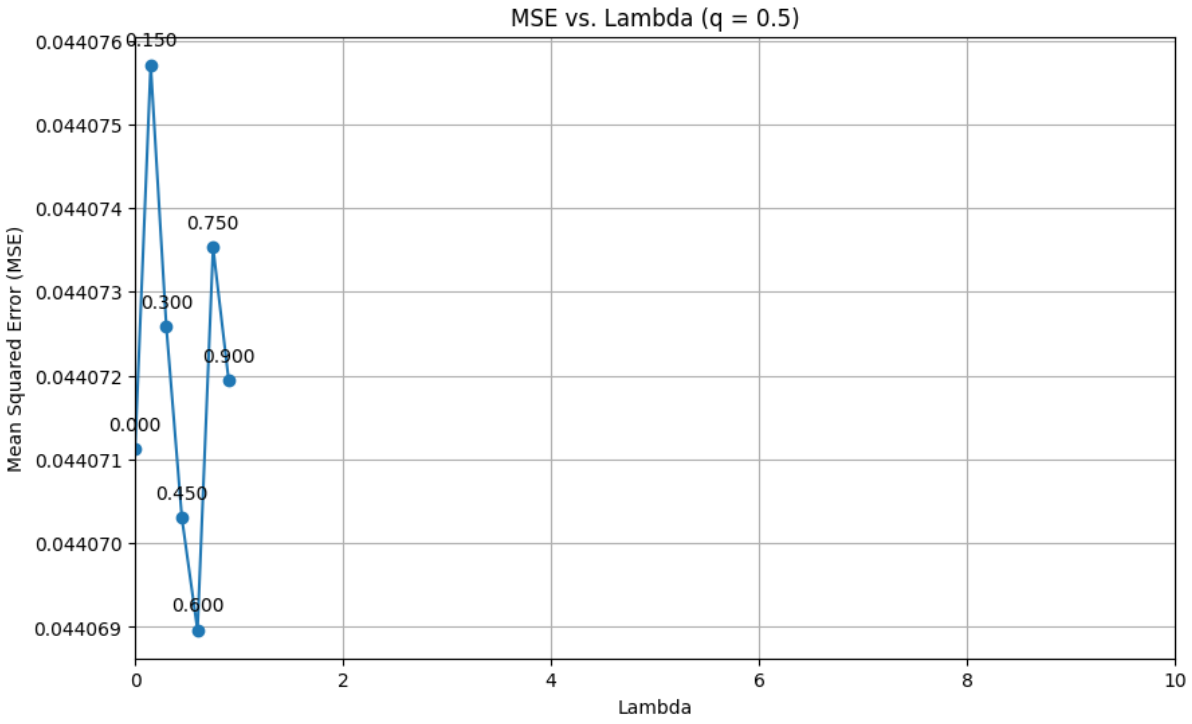


- The training error obtained is: 0.2559902039081671
- The testing error obtained is: 0.11536400130598212

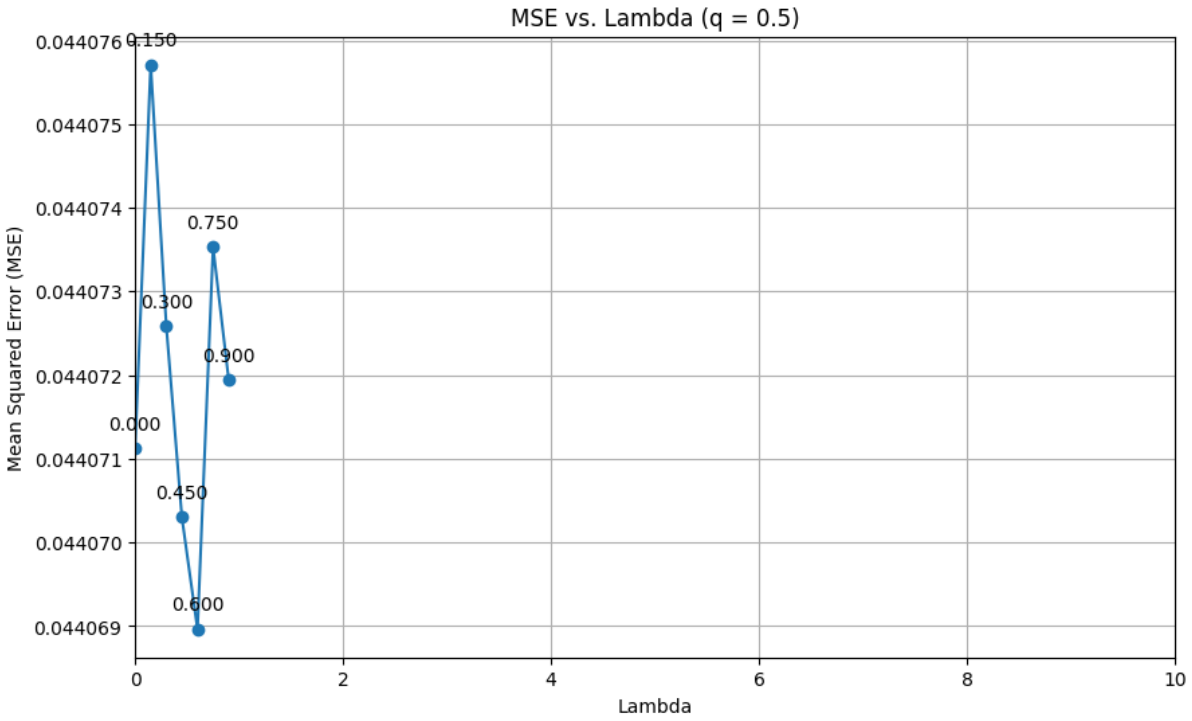
2. Stochastic gradient descent algorithm

*Let us find the best Lambda for various q values in **stochastic** case :*

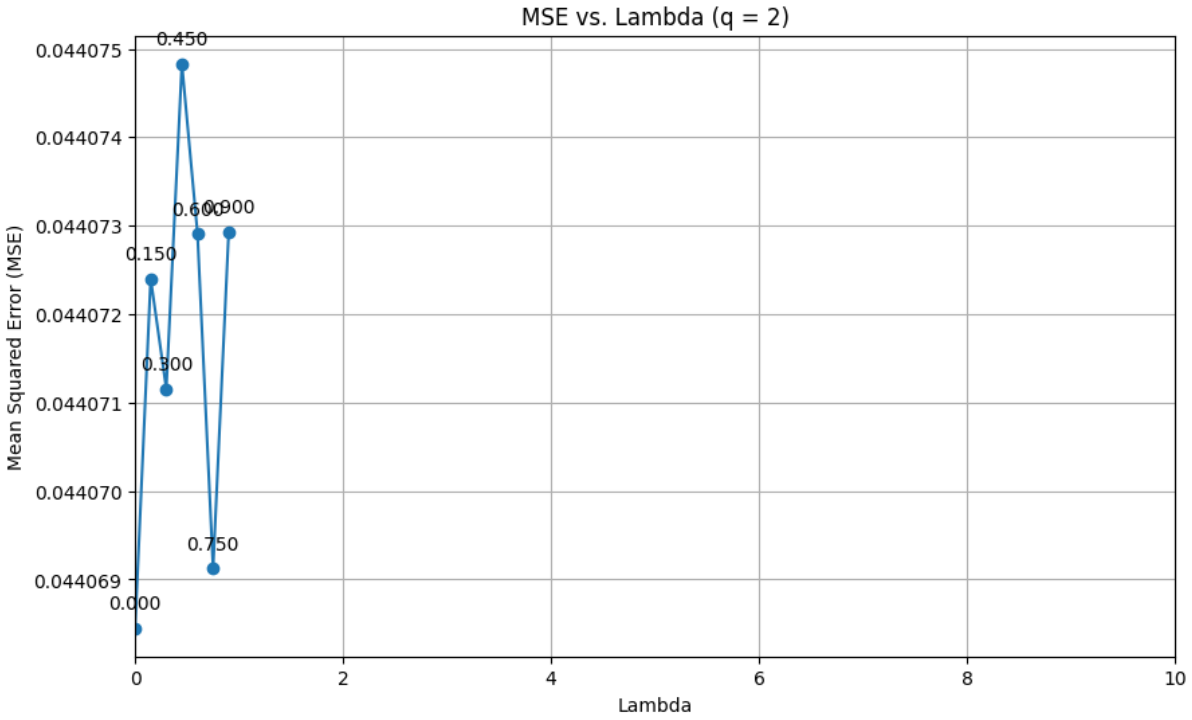
- **q = 0.5** → Best Lambda = 0.6



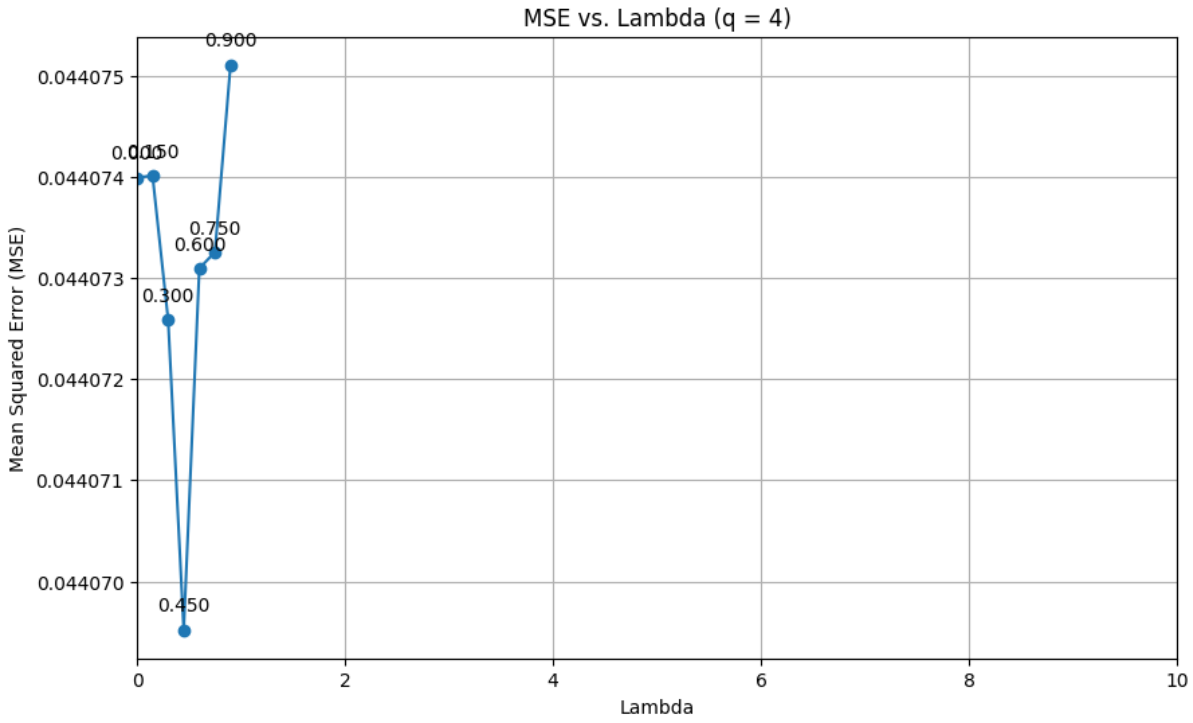
- **q = 1** → Best Lambda = 0.449



- **q = 2** → Best Lambda = 0

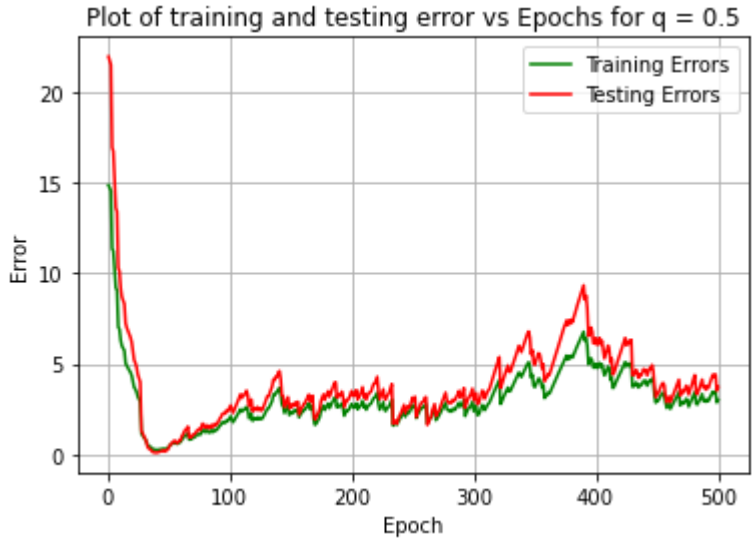


- **q = 4** → Best Lambda = 0.448



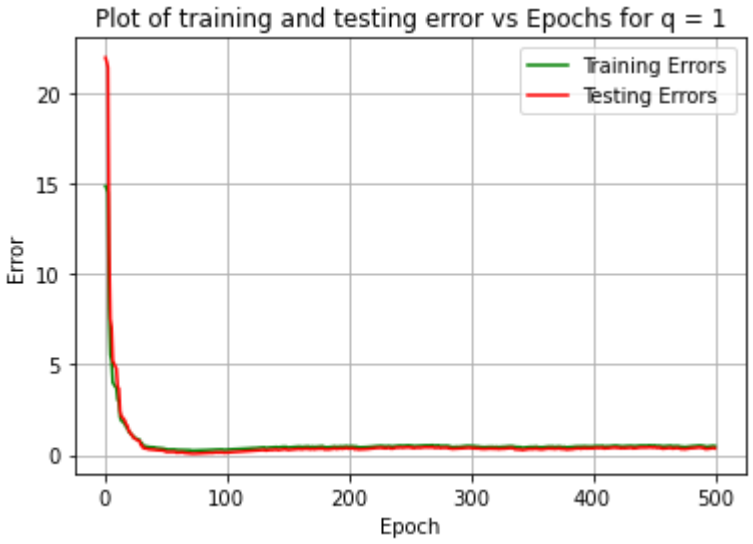
Now let us plot the errors of different q values at their best lambdas :

- **q=0.5:**



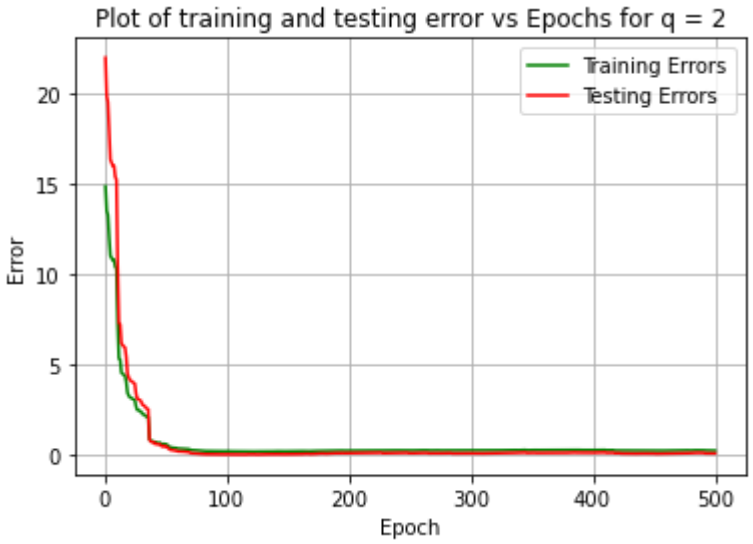
- The training error obtained is: 3.006334268070193
- The testing error obtained is: 3.7061445344541344

- **q=1:**



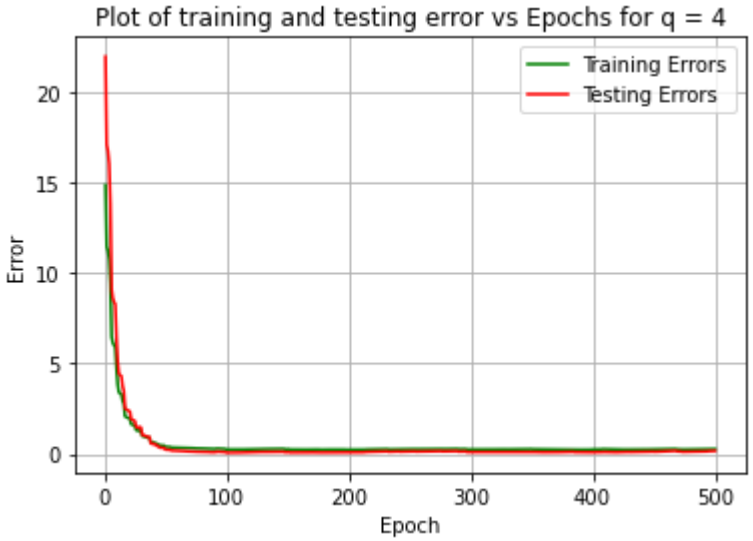
- The training error obtained is: 0.4905066837454878
- The testing error obtained is: 0.37125399464979864

- **q=2:**



- The training error obtained is: 0.27557595926101797
- The testing error obtained is: 0.12578613948407352

- **q=4:**



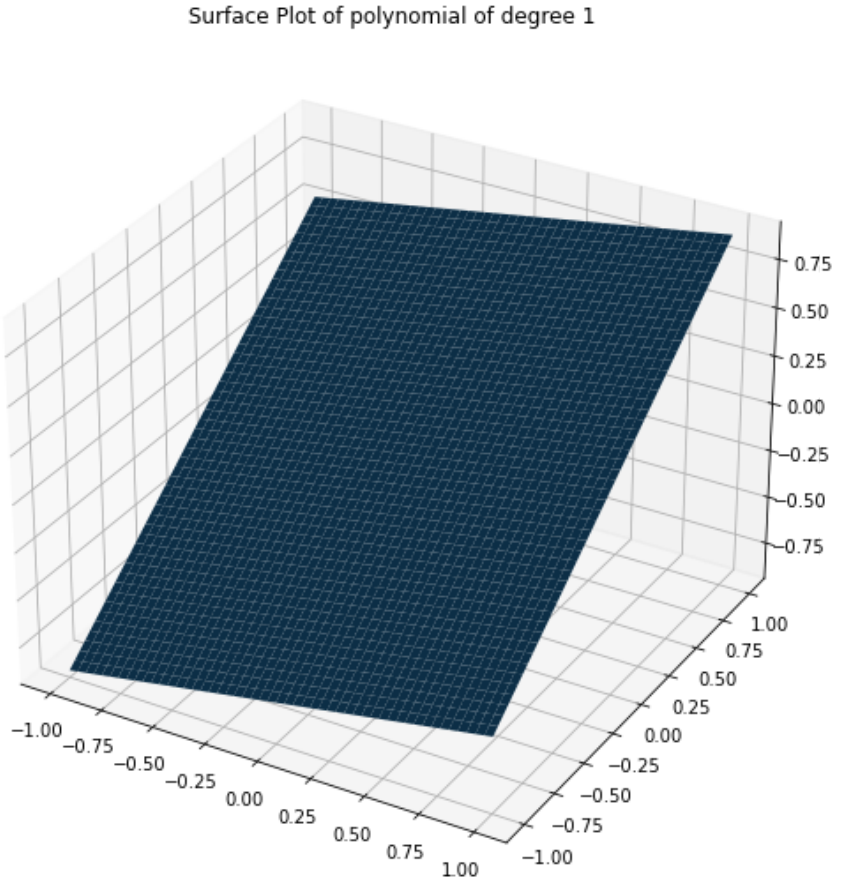
- The training error obtained is: 0.2744936649587803
- The testing error obtained is: 0.17471305300961082

Task 3: Graph Plotting

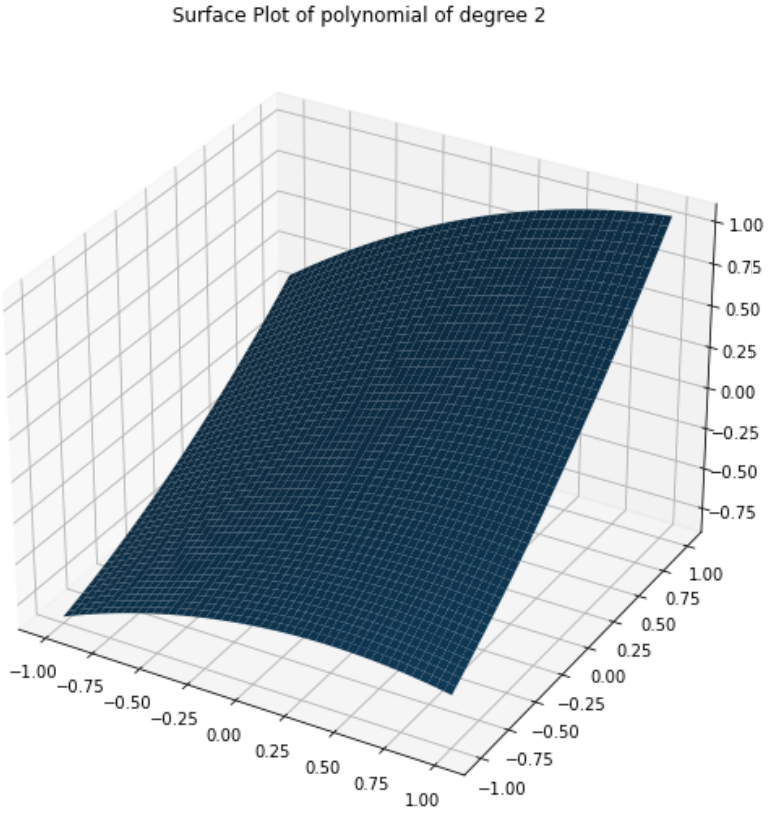
a. Surface plots for the nine polynomial regression models and the four optimal regularized linear regression models.

Surface plots for the nine polynomial regression:

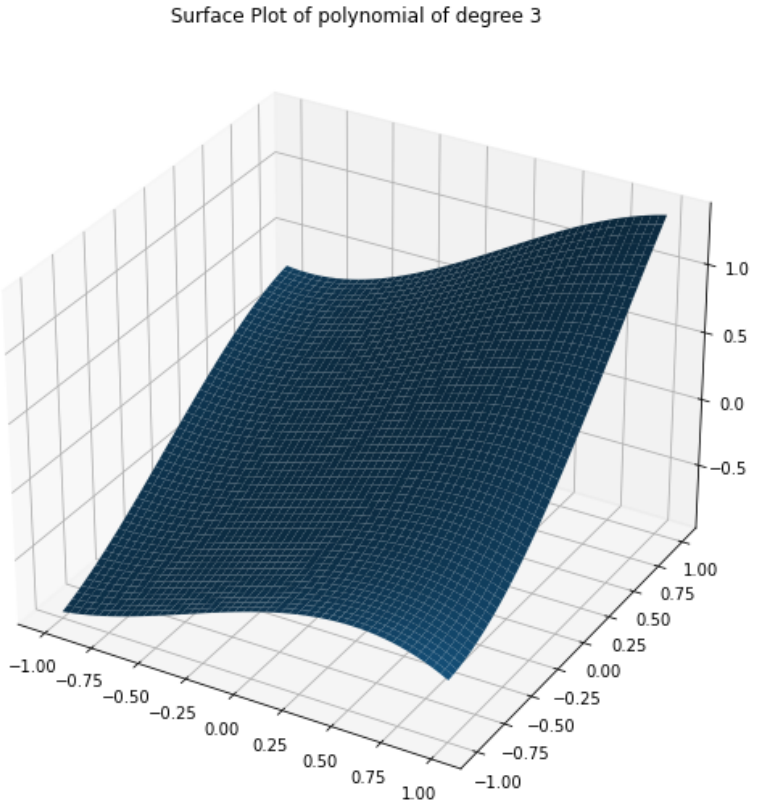
Degree=1:


Surface Plot of polynomial of degree 1
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00

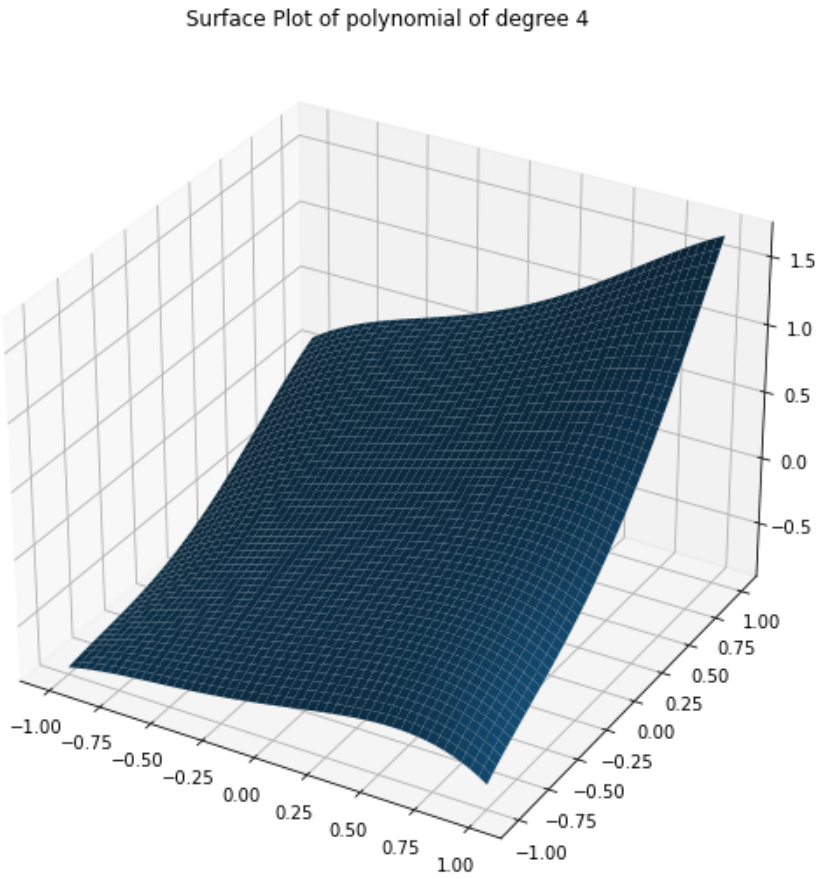
Degree=2:


Surface Plot of polynomial of degree 2
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00

Degree=3:


Surface Plot of polynomial of degree 3
1.0
0.5
0.0
−0.5
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00

Degree = 4:

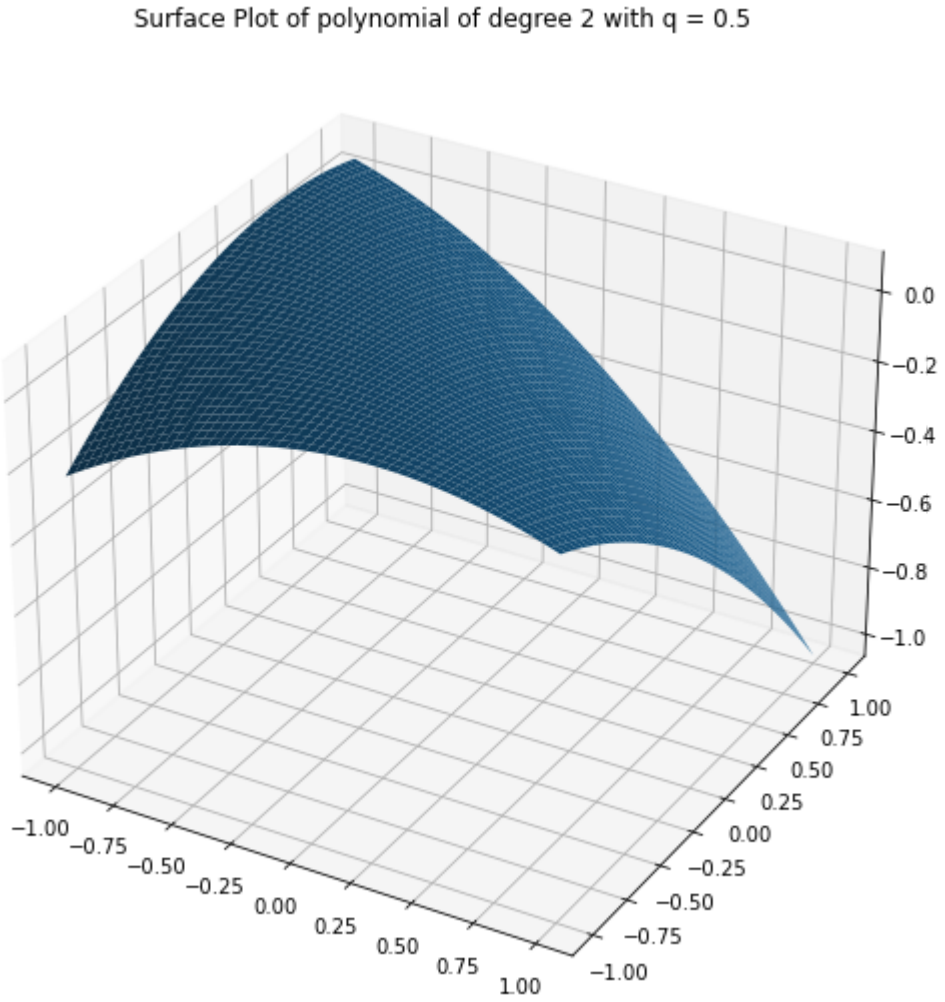


Since the models were overfitting for degree greater than 4, no surface plots were generated for degrees 5 to 9.

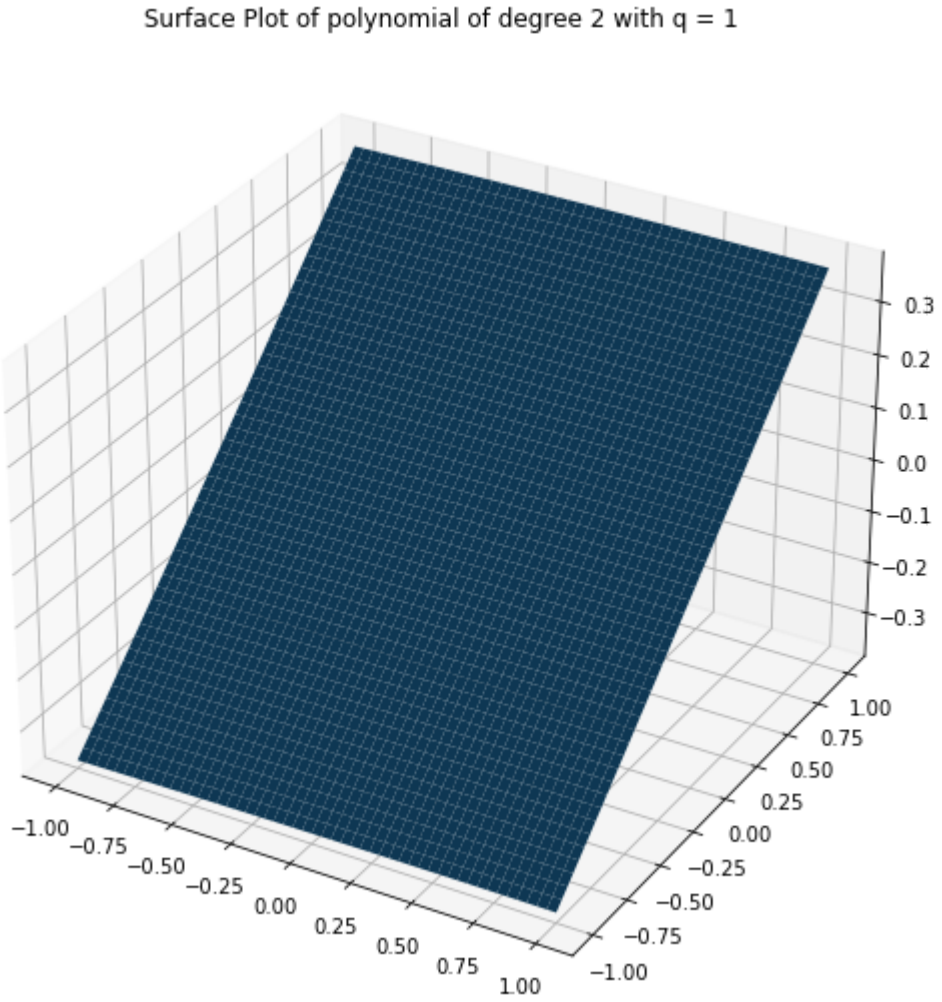
Four optimal regularized linear regression models:

1. Batch Gradient Descent algo:

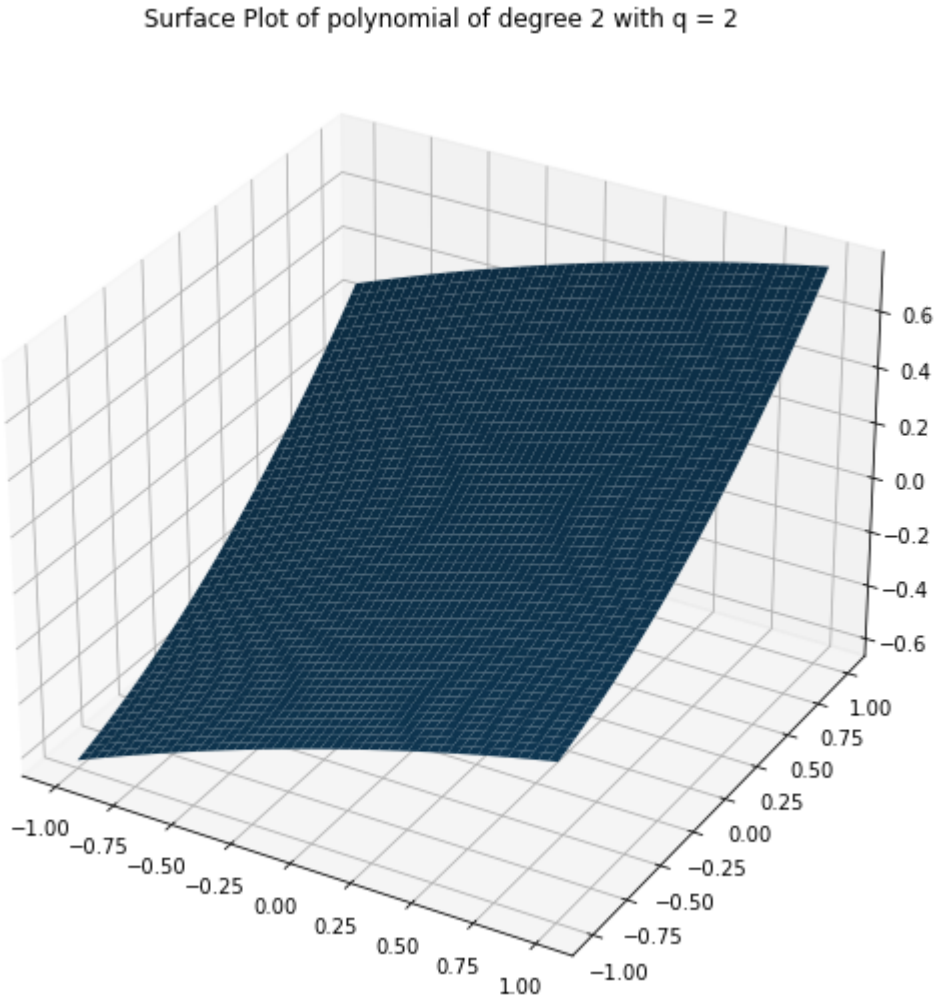
- q=0.5:


Surface Plot of polynomial of degree 2 with q = 0.5
0.0
−0.2
−0.4
−0.6
−0.8
−1.0
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00

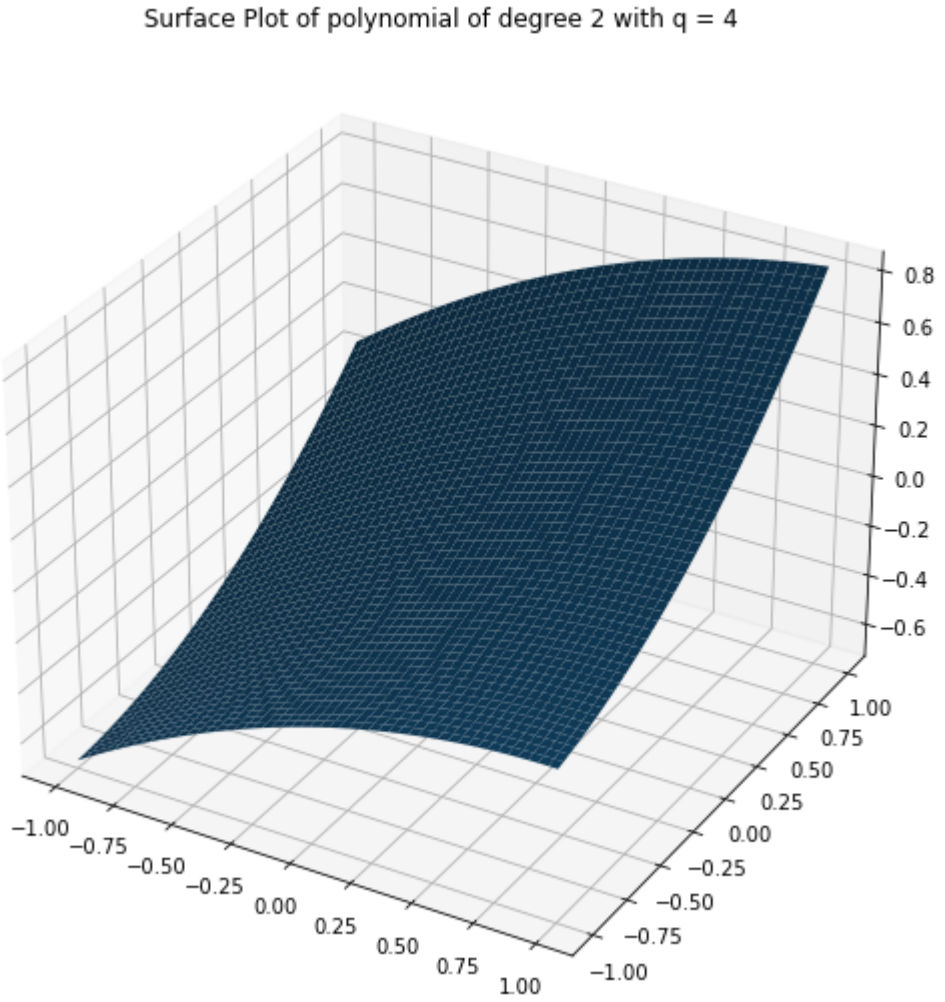
- q=1:


Surface Plot of polynomial of degree 2 with q = 1
0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00

- q=2:

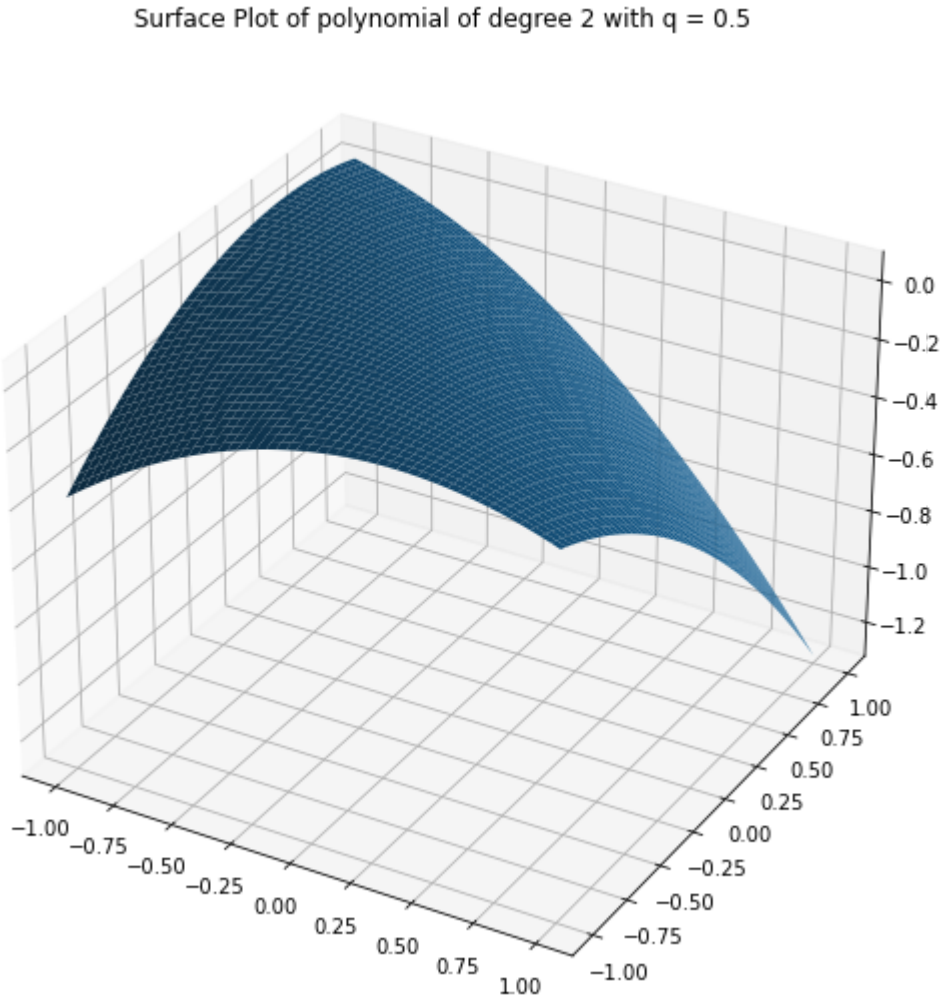

Surface Plot of polynomial of degree 2 with q = 2
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00

- q=4:

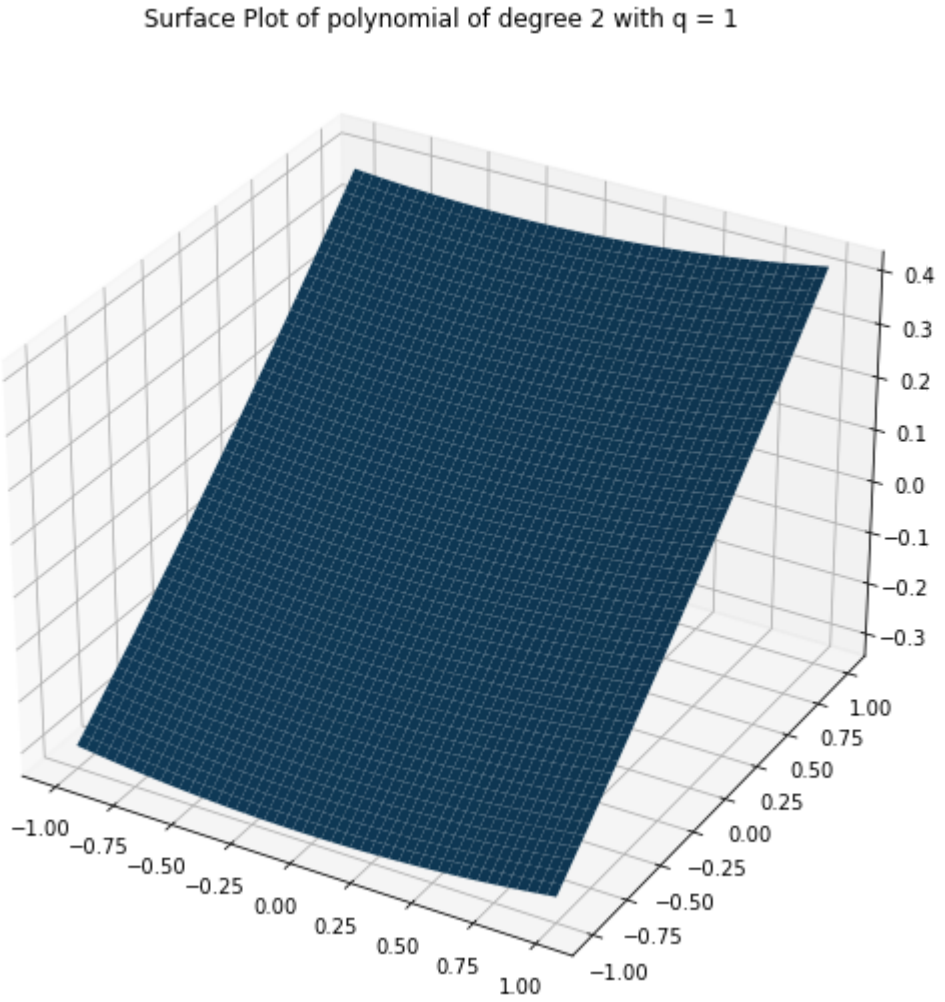


2. Stochastic Gradient Descent algo:

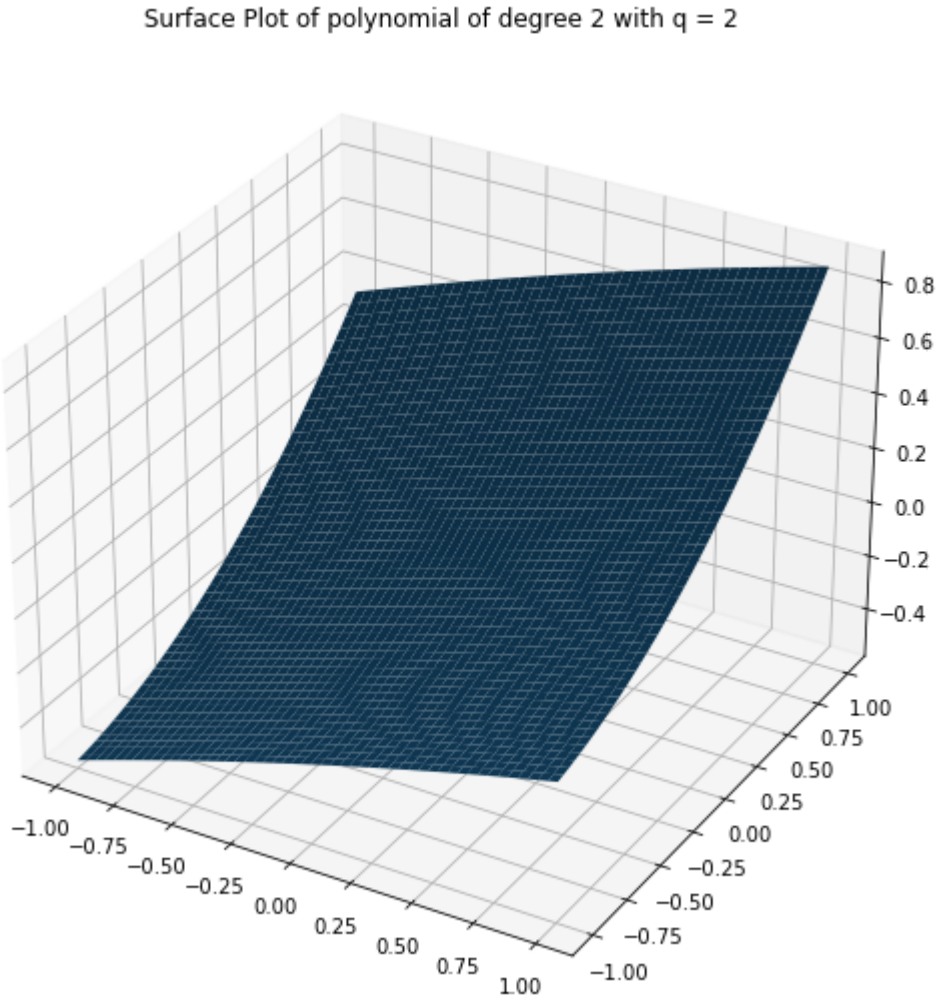
- q=0.5:


Surface Plot of polynomial of degree 2 with q = 0.5
0.0
−0.2
−0.4
−0.6
−0.8
−1.0
−1.2
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00

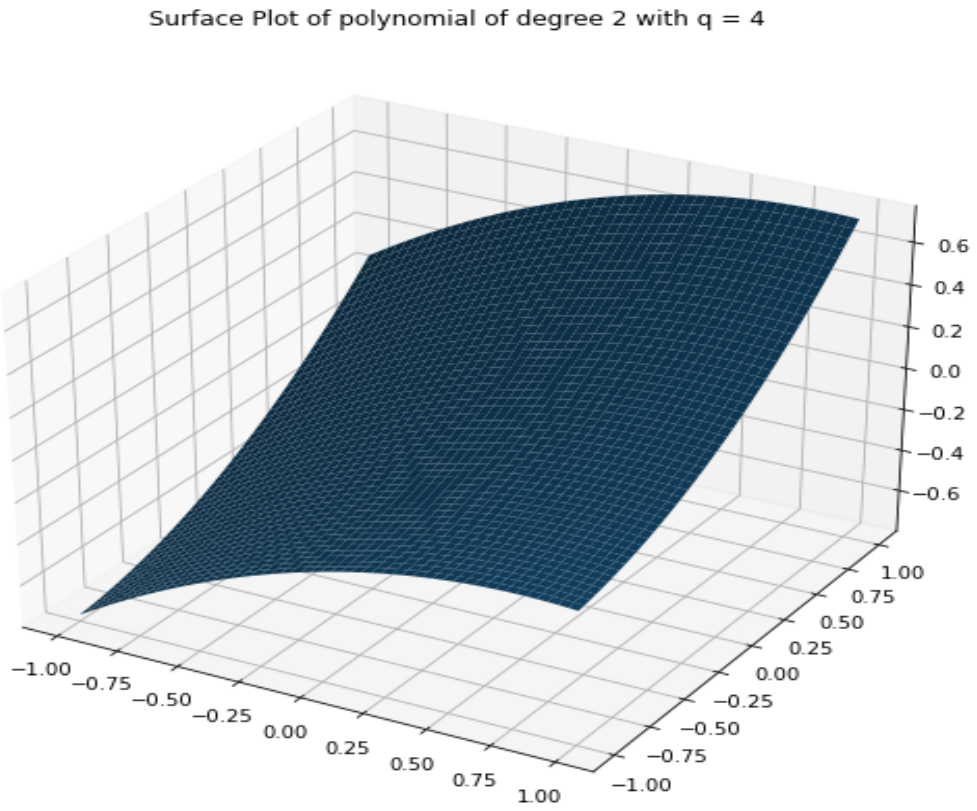
- q=1:


Surface Plot of polynomial of degree 2 with q = 1
0.4
0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00

- q=2:


Surface Plot of polynomial of degree 2 with q = 2
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00

- q=4:



Task 4: Comparative Analysis

Best-Fit classic polynomial regression model:

The training error and testing error vs polynomial degree for the best fit classic polynomial regression model suggests that a polynomial of **degree 2** is sufficient and optimal to describe the data as the training error and testing errors are the least for a polynomial regression model of **degree 2**. As we keep increasing the degree of the polynomial the training error is stable, but the testing error is observed to shoot up as observed in the graph of training error and testing error vs polynomial degree using both gradient descent and stochastic gradient descent.

Optimal regularized regression models:

The choice of "q" in regularization has a significant impact on the model's ability to handle overfitting.

q = 0.5:

When q is set to a small value, such as 0.5, the regularization term becomes less sensitive to large coefficients. Consequently, this results in a weaker regularization effect, as it does not strongly penalize large coefficients. The consequence of this weaker regularization is that it can lead to overfitting, allowing the model to have large and complex coefficients. This is clearly visible in plots with high variance.

Another very significant observation is that the training error is likely to be higher than the testing error, indicating overfitting.

q = 1:

In the case of regularization with q = 1, it corresponds to the standard L1 regularization, found in Lasso Regression. This form of regularization encourages coefficients to be small but not exactly zero, offering a balanced approach. It helps prevent overfitting by penalizing large coefficients while maintaining some degree of flexibility in the model. In this case of Lasso regression, q = 1 works to smooth out the impact of correlated features, reducing the potential for overfitting by distributing the importance among them more evenly. We typically see a more balanced relationship between training and test error. The training error is relatively low, and the test error is also relatively low. The model is learning to generalize from the training data while controlling overfitting.

q = 2:

A q value of 2, corresponding to standard Ridge Regression, represents a common and well-balanced choice for regularization. In Ridge Regression, the regularization term is formulated to balance the trade-off between overfitting and underfitting. This form of regularization discourages the coefficients from becoming too large, which is essential to prevent overfitting. As we can notice by increasing the value of q from 0.5, 1 to 2 there is a significant drop in the amount of overfitting in the model and a consequential dip in the test and train errors. Here both training and test errors are relatively low and show quite similar results as in the previous case.

q = 4:

In this case, the training error more or less remained the same but there is quite a notable increase in the testing error which shows that with "q" value like 4, the model experiences strong regularization. Training error remains relatively low, indicating that the model does not overfit the training data significantly. However, the test error may also remain low, but not as low as in cases where "q" is 1 or 2. The model is highly regularized, which restricts its flexibility, but it doesn't typically lead to significant underfitting. Instead, it results in a balanced model that generalizes reasonably well while maintaining simplicity.

In summary, the choice of "q" influences the trade-off between model complexity and generalization. A "q" value of 0.5 tends to lead to overfitting, with low training error but high test error. A "q" value of 1 and 2 results in a balanced relationship, providing a good trade-off between training and test errors. A "q" value of 4, while having reasonably low errors, also maintains a balance due to strong regularization. The optimal choice of "q" depends on the specific data characteristics and the desired trade-off between model complexity and generalization in your machine learning task.