# Contents

# 1. Image Reconstruction

## 1.1 Reconstructing Image after Singular Value Decomposition of a Grayscale Image

Firstly, the 8-bit coloured image is converted to 8-bit grayscale image. Each pixel of image can be consider as an element of a matrix in which image is the matrix. These pixels (or elements) can have any value between 0 and 255. In the grayscale 0 represents black colour and 255 represents white colour. Therefore, the 8-bit numbers from 0 to 255 in grayscale represents different shades of colour grey.

Secondly, the singular value decomposition is performed on the grayscale image which decomposes the given image matrix 'M' into product of three matrices. These three matrices are named as U, S and $V^T$.

$$M = U * S * V^T$$

Where, U = Left Singular Vectors of M

S = Diagonal Matrix of Singular Values of M

V = Right Singular Vectors of M

**Reconstruction of Image Matrix M' after choosing top N Singular values**

Create a new diagonal matrix S' which contains N singular values of matrix S. Similarly, create two more matrices U' and V' such that they contain only those columns which corresponds to singular values in S'.

From these U', S' and V', we try to construct a new image matrix M' and compare this new image M' to our original grayscale image M for different values of N.

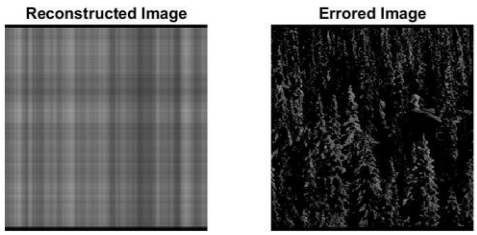
$$\text{i.e.} \quad M' = U' * S' * V'^T$$

We will reconstruct both Square and Rectangular Grayscale images.

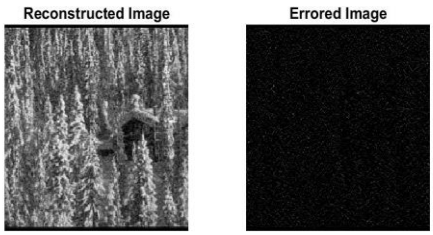
We will try to show both the reconstructed image and corresponding errored image for different values of N.

**N vs Root Mean Square Error (RMSE) plot**

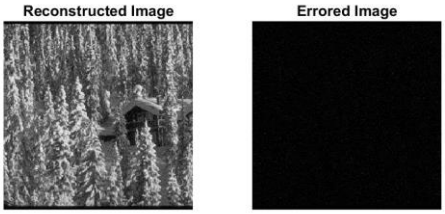
This plot will show the value of root mean square error between original and reconstructed image for different N values. This will help to indicate the difference in image obtained after reconstruction and the original one.



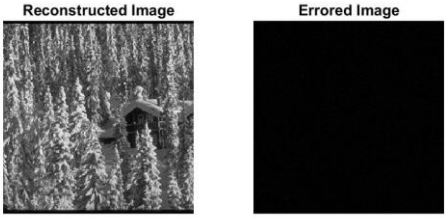

a) For N = 1




b) For N = 50

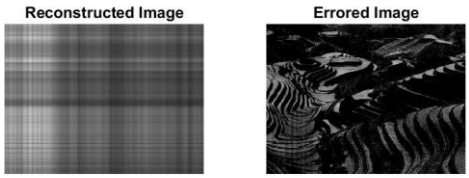



a) For N = 100

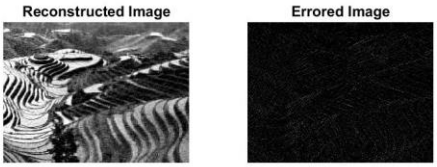



d) For N = 150

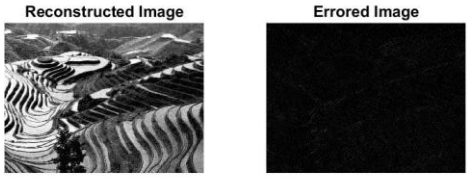
**Figure 1.1: Reconstructed Grayscale Square Images for different N's**



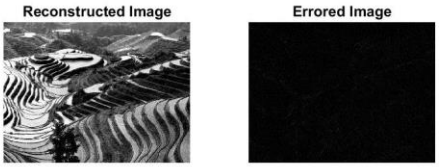

a) For N = 1




b) For N = 50




c) For N = 100




d) For N = 150

**Figure 1.2: Reconstructed Grayscale Rectangle Images for different N's**

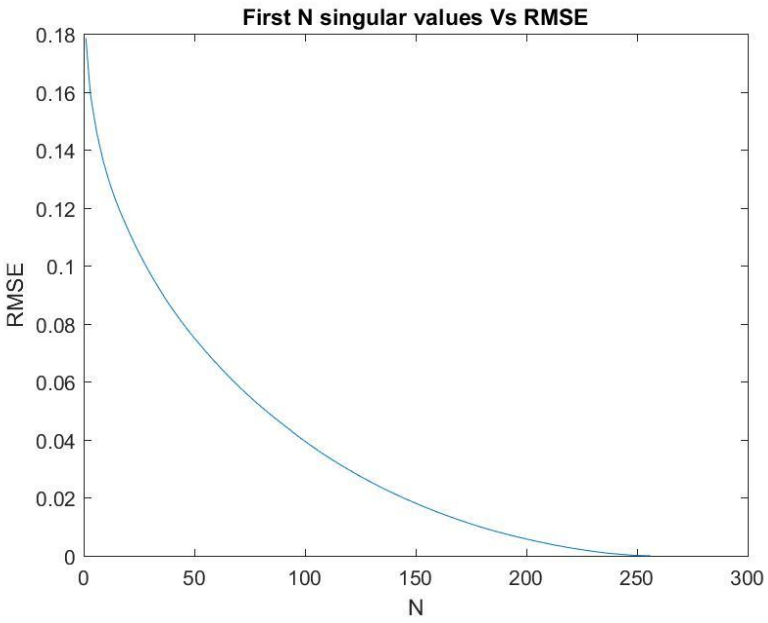



**Figure 1.3: Error Plot for Square Image**

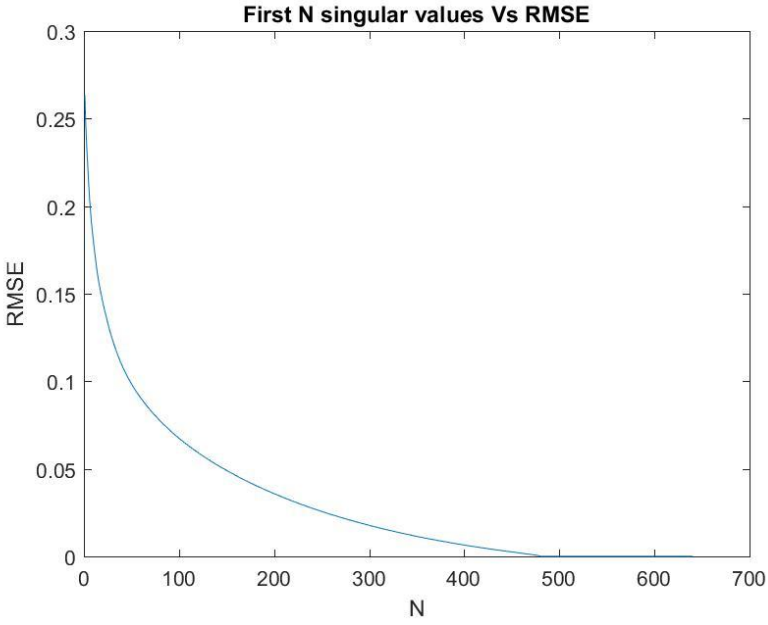



**Figure 1.4: Error Plot for Rectangular Image**

**Observations:**

1. The more the number of singular values we consider for reconstruction, the less is the error.

2. The higher value Singular values contribute much more than lower value Singular values for image reconstruction.

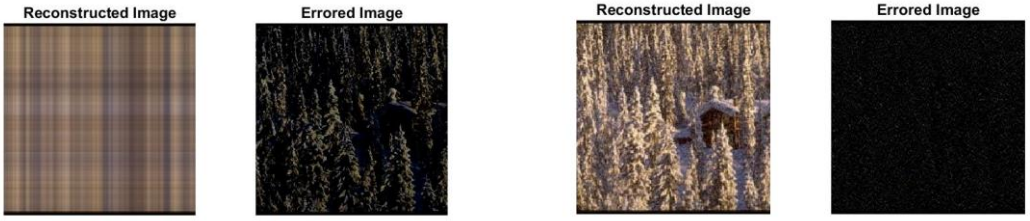
3. By using only these higher order singular values and omitting lower order singular values we can reconstruct the entire image with negligible error. Therefore, this technique can be used to compress images.

### 1.2 Reconstructing Image after Singular Value Decomposition of each colour band

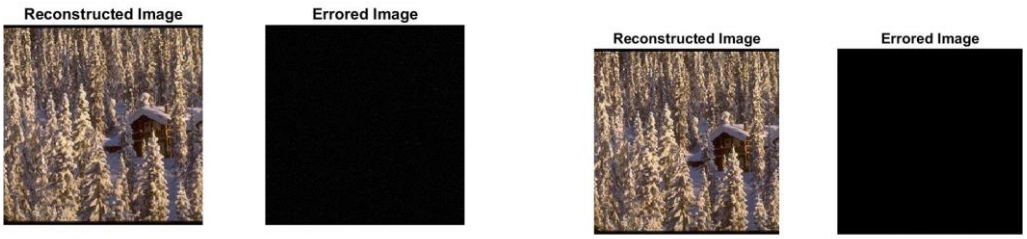
Firstly, separate the three colour bands Red, Green and Blue of original image in three different Matrices. Secondly, decompose these three matrices into product of three different matrices using singular value decomposition (explained in section 1.1) respectively.

Reconstruct three new Red, Green and Blue band matrices by selecting N singular values from original Red, Green and Blue sigma matrix respectively. Combine these three new obtained band to form a single matrix M'. This M' is nothing but our final reconstructed image.

We will now show the different M' (reconstructed image) we obtained for different values of N with their Error Images.




a) For N = 1 b) For N = 50



c) For N = 110 d) For N = 215

**Figure 1.5: Reconstructed RGB Square Images for different N's**


a) For N = 1 b) For N = 50


c) For N = 100 d) For N = 157

**Figure 1.6: Reconstructed RGB Rectangle Images for different N's**

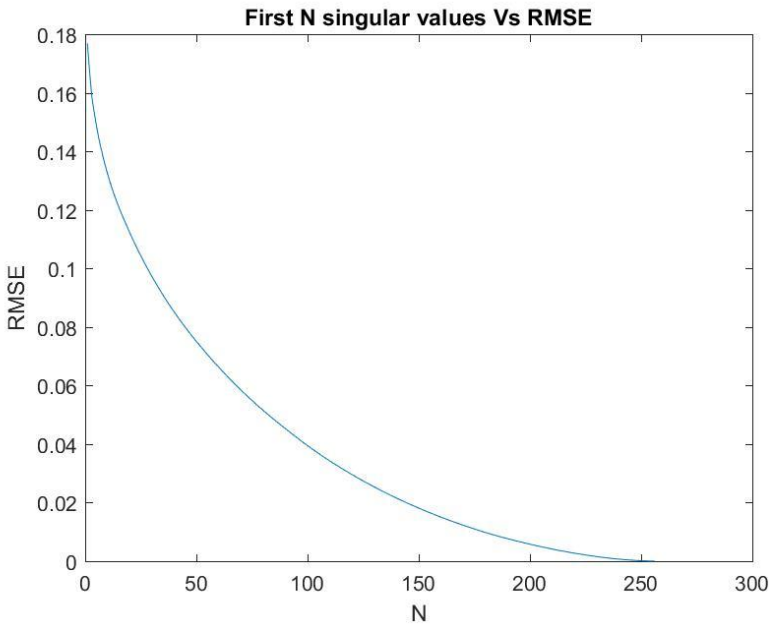



**Figure 1.7: Error Plot for Square RGB Image**

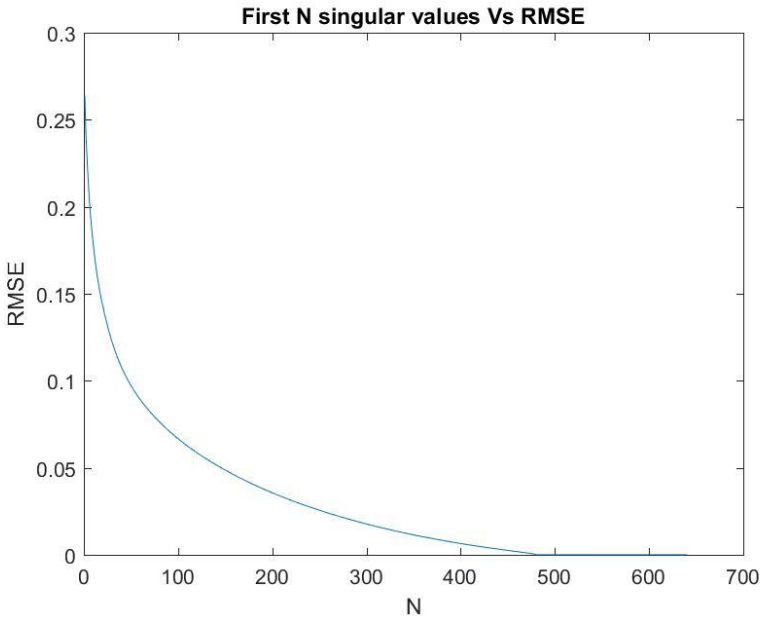



**Figure 1.8: Error Plot for Rectangular RGB Image**

**Observations:**

1. The more the number of singular values we consider for reconstruction, the less is the error.

2. The higher value Singular values contribute much more than lower value Singular values for image reconstruction.

3. By using only these higher order singular values and omitting lower order singular values we can reconstruct the entire image with negligible error. Therefore, this technique can be used to compress images.

### 1.3 Reconstructing Image after Singular Value Decomposition of Image formed after concatenating each colour band in RGB order

Separate each Colour band Red, Green and Blue in three different matrices. Each of these matrices has 8-bit numbers as entry.

Concatenate these 8-bit numbers to form 24-bit numbers. Create a 32-bit Matrix and store these 24-bit numbers in it. This will result in a 32-bit grayscale image matrix.

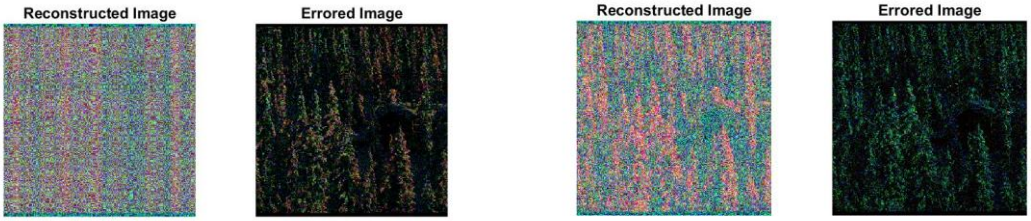
Perform singular value decomposition on this 32-bit grayscale matrix. Choose top N singular values and reconstruct a new 32-bit grayscale matrix.

Separate out last 8-bits of this newly obtained 32-bit matrix and put them in a Blue band of our final image matrix M'.

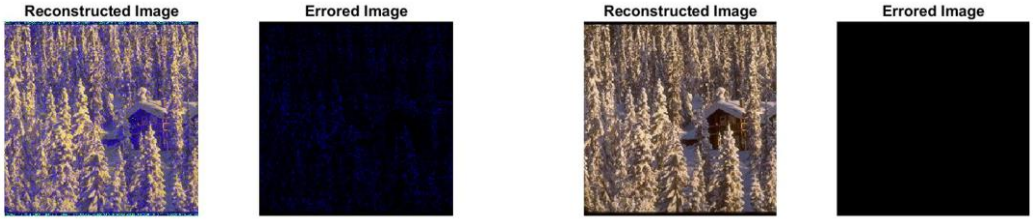
Then, Separate out next 8-bits of this newly obtained 32-bit matrix and put them in a Green band of our final image matrix M'.

Similarly, Separate out nest 8-bits of this newly obtained 32-bit matrix and put them in a Red band of our final image matrix M'.

We will now show the different M' (reconstructed image) we obtained for different values of N with their Error Images.



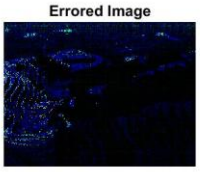

a) For N = 1 b) For N = 150



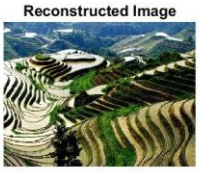

c) For N = 254 d) For N = 256

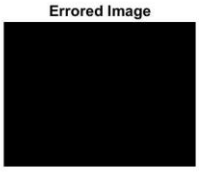
**Figure 1.9: Reconstructed RGB Concatenated Square Images for different N's**

a) For N = 1


b) For N = 300




c) For N = 479

d) For N = 480

**Figure 1.10: Reconstructed RGB Concatenated Rectangle Images for different N's**

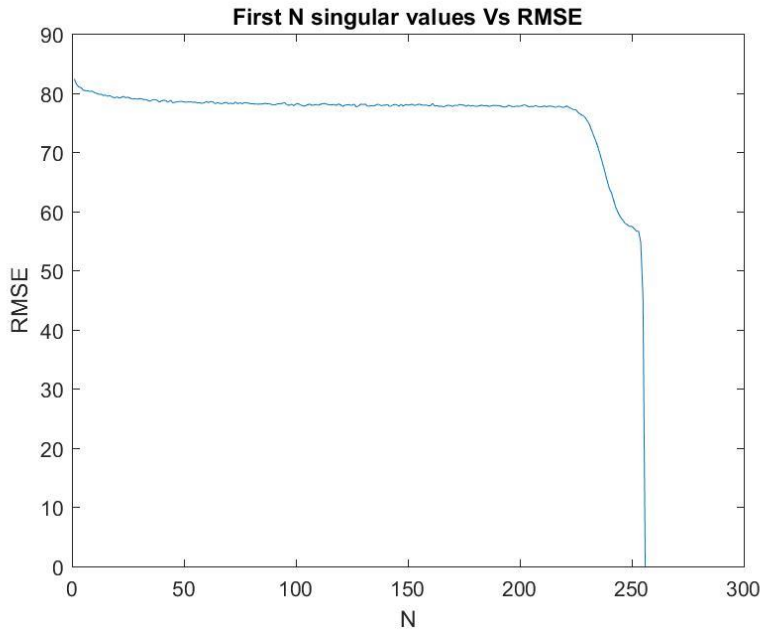



**Figure 1.11: Error Plot for Square RGB concatenated Image**

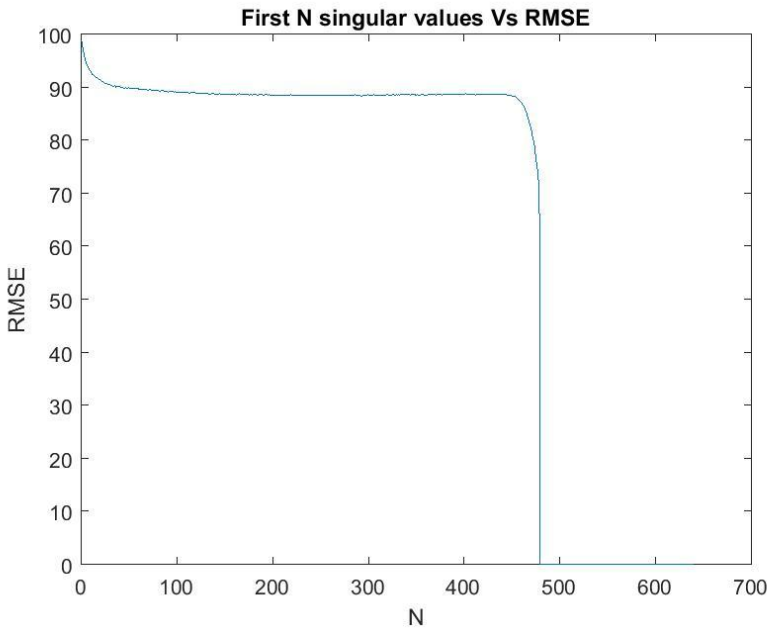



**Figure 1.12: Error Plot for Rectangle RGB concatenated Image**

**Observations:**

1. The error between original and reconstructed square image remain approx. const. till N = 250. After that a sudden reduction in error is observed leading to zero error at N = 256.

2. The error between original and reconstructed rectangle image remain approx. const. till N = 479. After that a sudden reduction in error is observed leading to zero error at N = 480.

### 1.4 Eigen Value Decomposition (EVD)

The Eigen value decomposition is performed on the image matrix which decomposes the given image matrix 'M' into product of three matrices. These three matrices are named as U, S and $U^{-1}$.

$$M = U * S * U^{-1}$$

Where, U = Eigen Vectors of M

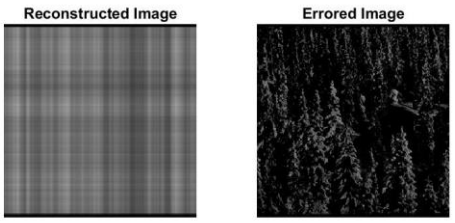
S = Diagonal Matrix of Eigen Values of M

$U^{-1}$ = Inverse of Eigen Vectors of M

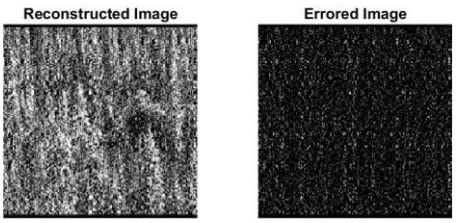
We will do same thing in EVD as we have done in SVD. The only difference is of the way we decompose a matrix in both methods.

We will construct images by considering different eigen values and plot an error curve between all the top N eigen value images and the original image.

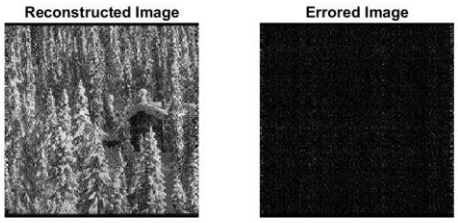
### 1.5 Reconstructing Image after Eigen Value Decomposition of a Grayscale Image



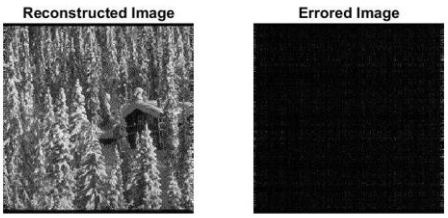

a ) For N = 1




b) For N = 50

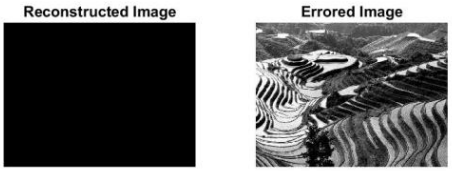



c) For N = 140

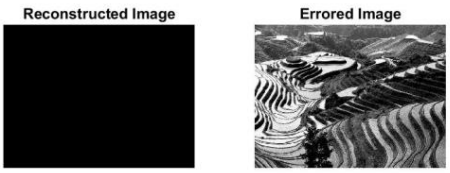



d) For N = 200

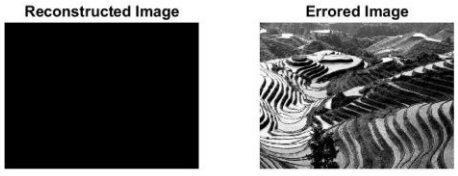
**Figure 1.13: Reconstructed Grayscale Square Images for different N's**



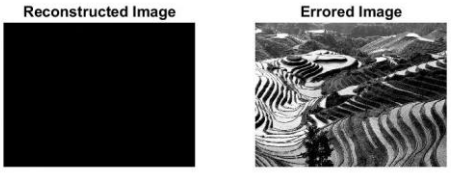

a) For N = 1




b) For N = 70




c) For N = 110




d) For N = 210

**Figure 1.14: Reconstructed Grayscale Rectangle Images for different N's**

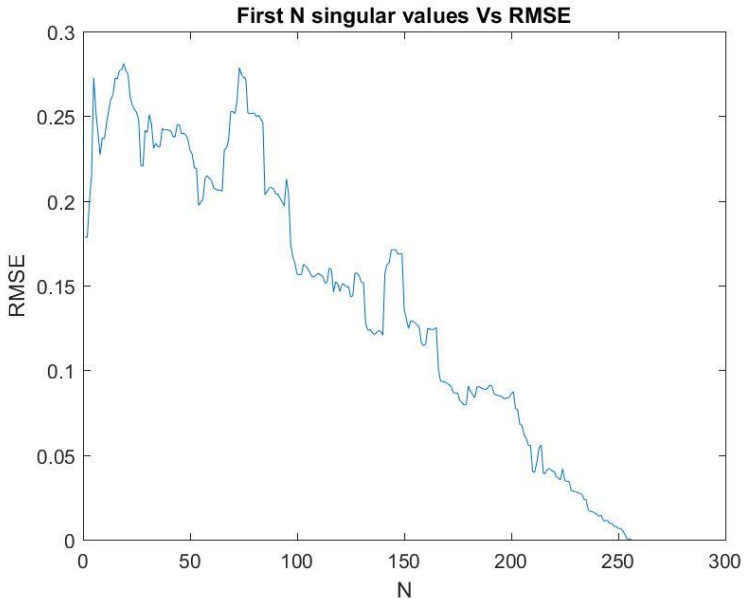


**Figure 1.15: Error Plot for Square Image**

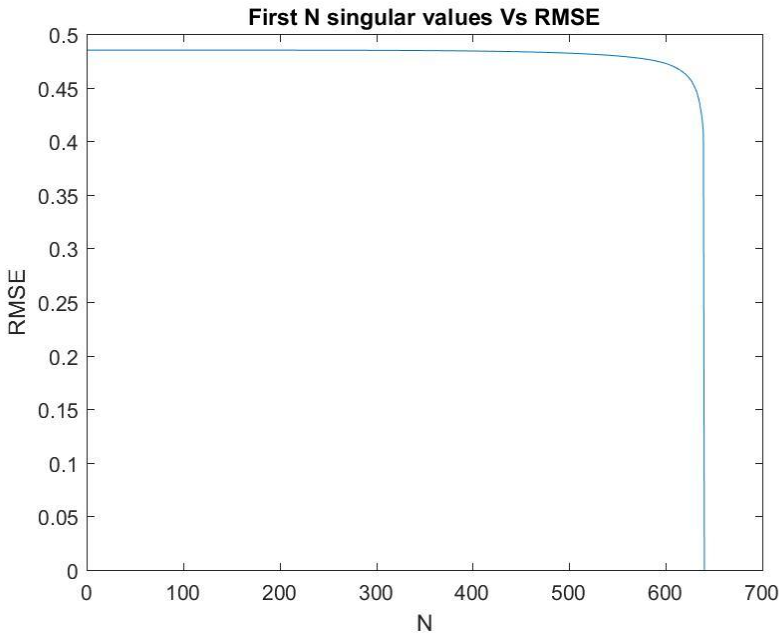
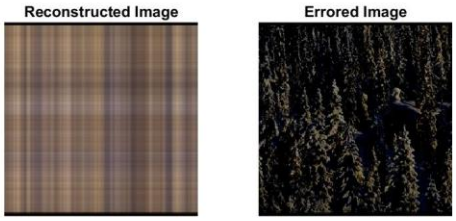
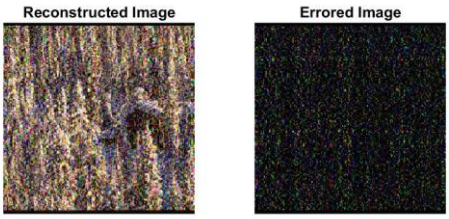



**Figure 1.16: Error Plot for Rectangle Image**

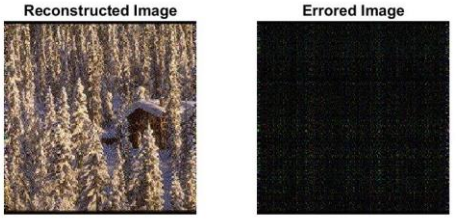
## 1.6 Reconstructing Image after Eigen Value Decomposition of each color band



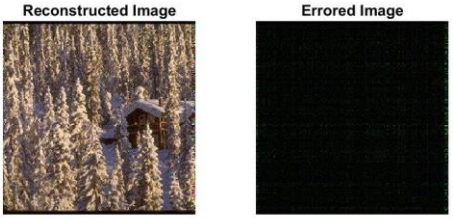

a) For N = 1




b) For N = 50

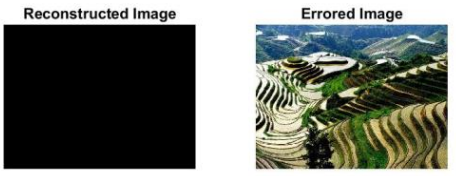



c) For N = 150

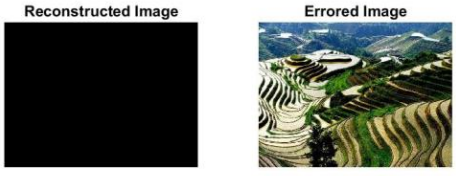



d) For N = 200

**Figure 1.17: Reconstructed RGB Square Images for different N's**




a) For N = 1




b) For N = 50


c) For N = 100 d) For N = 150

**Figure 1.18: Reconstructed RGB Rectangle Images for different N's**

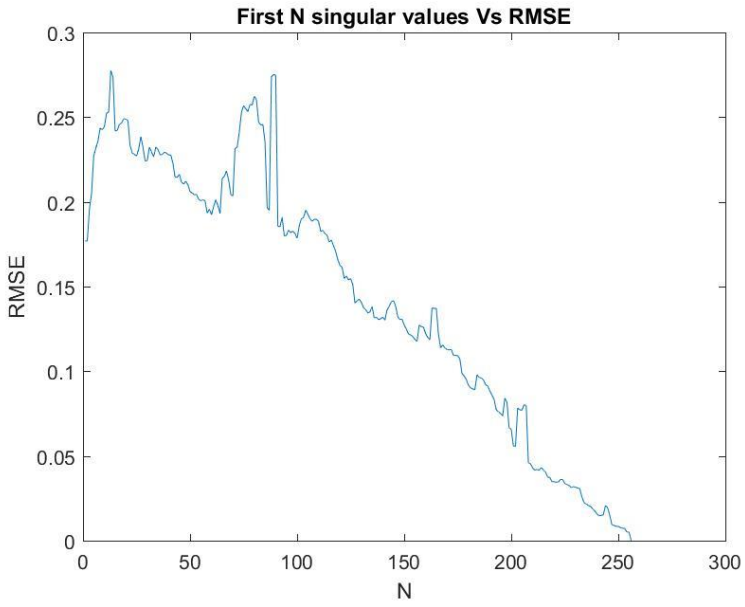



**Figure 1.19: Error plot for Square Image**

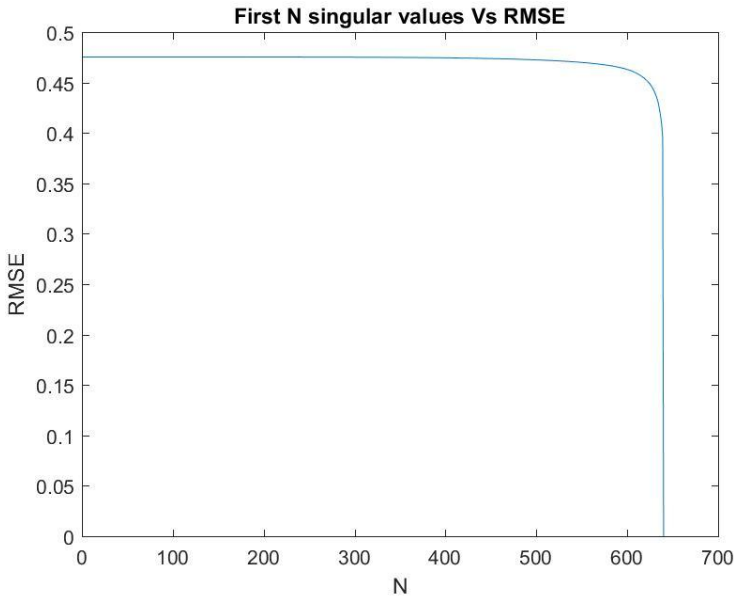
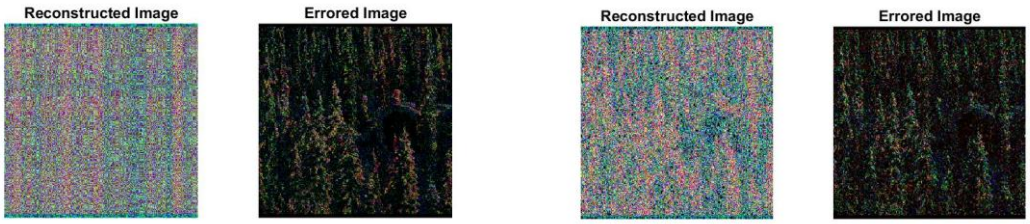
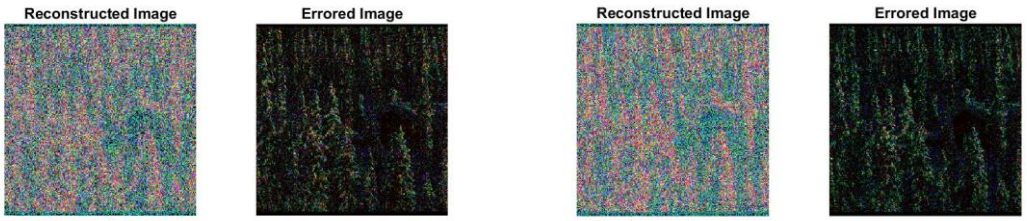



**Figure 1.20: Error plot for Rectangle Image**

## 1.7 Reconstructing Image after Eigen Value Decomposition of Image formed after concatenating each color band in RGB order




a) For N = 1 b) For N = 50




c) For N = 100 d) For N = 150

**Figure 1.21: Reconstructed RGB Concatenated Square Images for different N's**


a) For N = 1 b) For N = 50


c) For N = 100 d) For N = 150

**Figure 1.22: Reconstructed RGB Concatenated Rectangle Images for different N's**

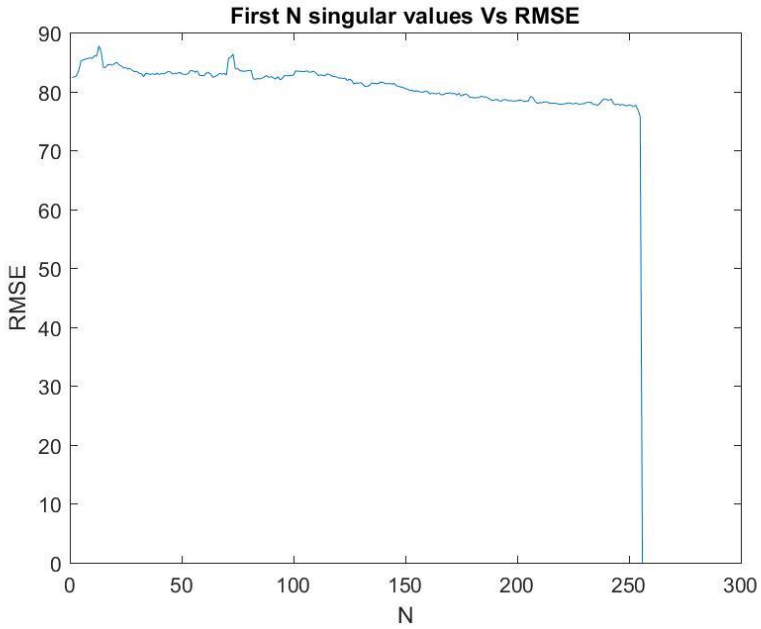



**Figure 1.23: Error plot for Square Image**

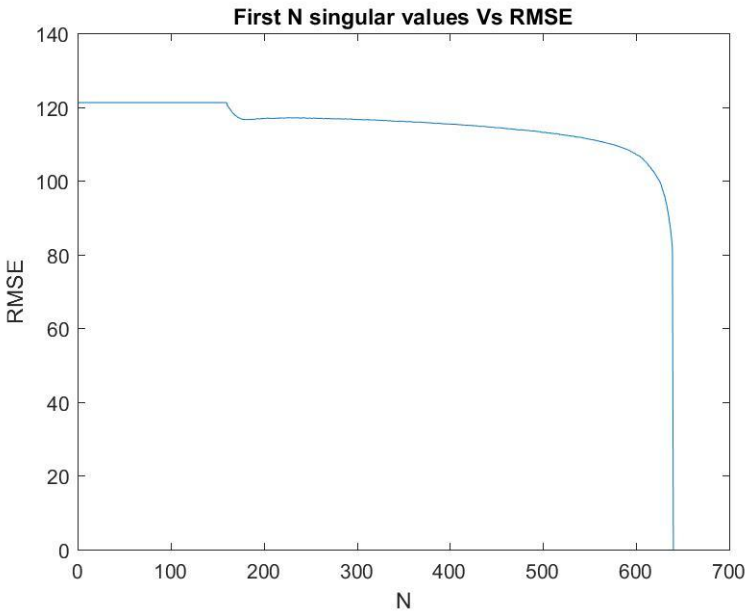



**Figure 1.24: Error plot for Rectangle Image**

# 2 Polynomial Regression

In this problem we will start with simple linear regression for one dimensional data and we will perform polynomial regression on one dimensional data, two dimensional data and k-dimensional data. We will solve regression problems using ordinary least squares method. Linear Regression problems we will try to fit a line for the given data.

Let's start with simple linear regression for one-dimensional data. A simple linear regression line is in the form of $\boldsymbol{Y = w_0 + w_1 X}$, where **X** is the feature of the data and **Y** is the dependent variable, $w_0$ and $w_1$ are constants. We will find out the values of $w_0$ and $w_1$ by minimizing the sum of squares of the error.

### 2.1) Dealing with one-dimensional data

Degree of the Polynomial = 1

Divide the given data into three parts as train data to train the model, validation data to validating the model, test data to test the model. To train model we are using least squares method. For validating the data we are taking measure as $\mathbf{R^2}$. $\mathbf{R^2}$ =1 means we are having perfect model, $\mathbf{R^2}$ = 0 means we predicted the average of the target variable. $\mathbf{R^2}$ is negative means our model is worse than the mean of the target variable.

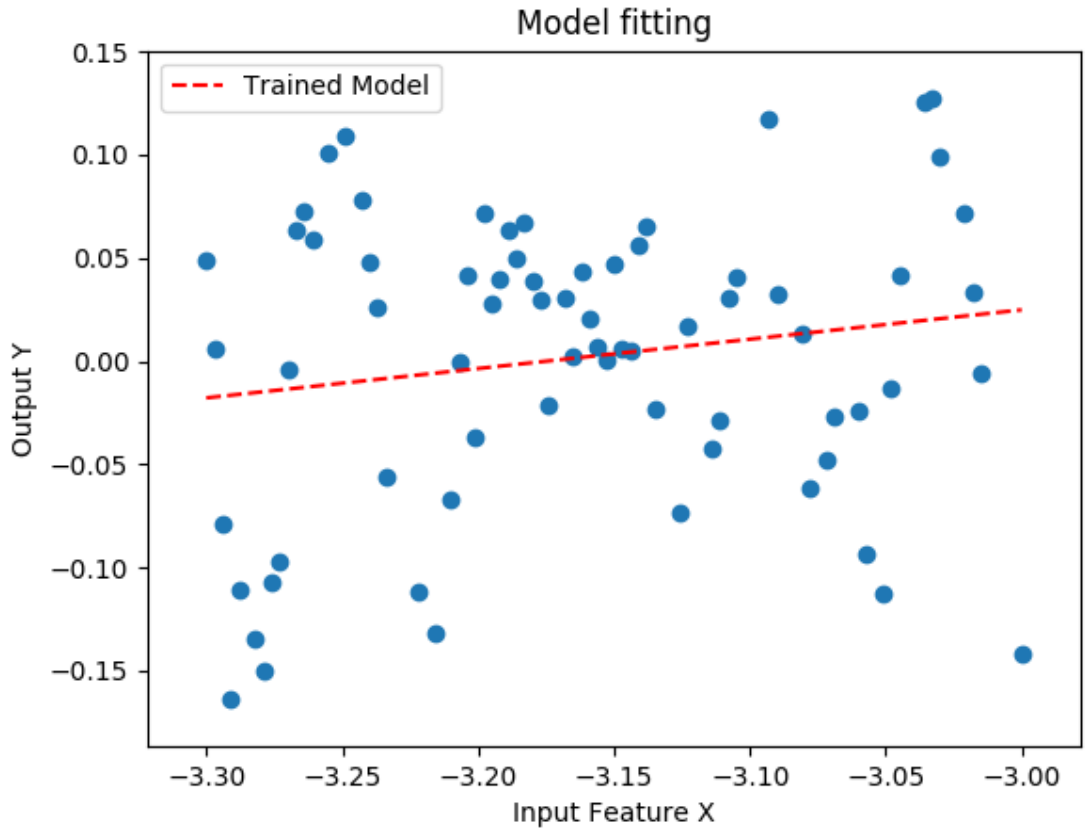


**Fig 2.1 One-Dimensional data Model Polynomial Degree 1**

The blue dots in the Fig 2.1 indicates the corresponding y values to the x values. The dashed line in the Fig 2.1 is our model. The blue dots which are on the circles contributes the zero error. The dots above the red line contributes positive error, the dots below the circles contributes negative error. We are taking sum of the squares of the error, because sometimes dot above the red line contributes positive error value of 5, dot below the red line contributes the error value -5.If we add the errors without squaring we will end up with error as zero. What we want is for any target not equal to prediction a positive contribution of error.

Below are the few results after training the data.

R-squared for validation data is: 0.0285660399962. Our model is predicting the data which is the mean of the target variable.

After training our data, we will try to predict the y variable for the given input X feature vector. Below is the graph which shows the experimental results of the prediction.

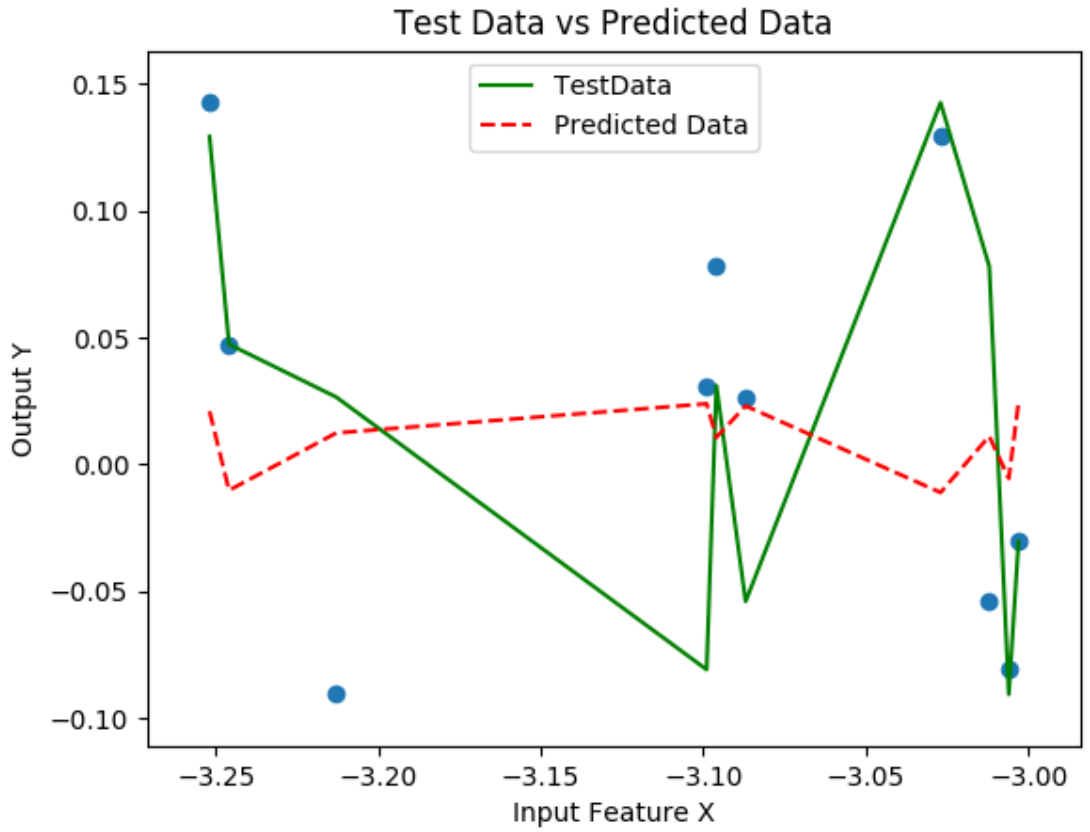



**Fig 2.2 Test data VS Predicted Data for One Dimensional Polynomial Degree 1**

Clearly figure shows our model is not predicting correctly for unseen test data.

So we can clearly conclude that to predict the correct output we need some more input features. So we will try to move to polynomial regression.

The equation for polynomial regression in single dimensional data is

$$y = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + \ldots + a_k x^k$$

Clearly now we have more input features and one output variable. Output variable value depends now on many input variables.

Now we will try to increase our degree of polynomial and train our model with polynomial Degree 250.

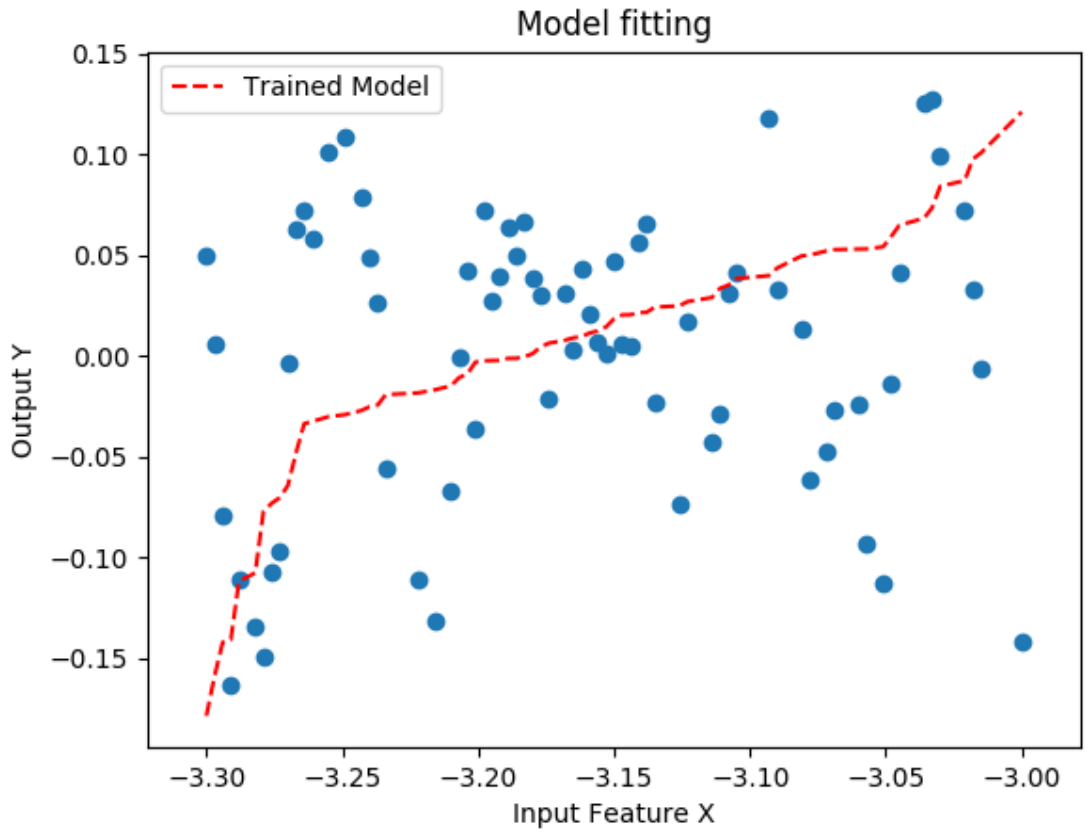



**Fig 2.3 One-Dimensional data Model Polynomial Degree 250**

After training the data we got the r-squared for training data is 0.679830910575**.** It's is little bit closer to the 1.Our model predicts correctly 67% of the time.

Below is the graph showing the results of test data vs train data

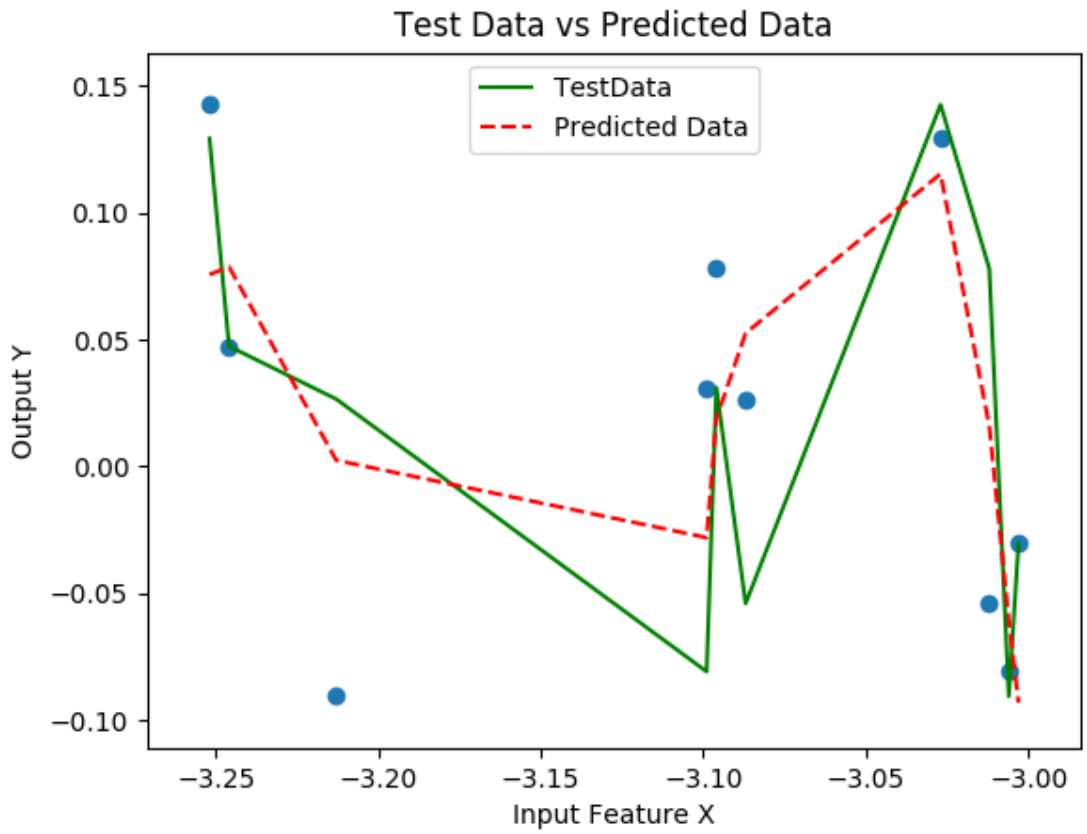



**Fig 2.4 Test data vs Predicted Data for one-dimensional data polynomial degree 250**

We got R-squared value as 0.679830910575, and square error while validating the data as 0.0312276153855. We will try to achieve R-squared value equal to one. So now we will increase the degree of the polynomial to 260 and try to train our model. Interestingly what happened is we got R-squared value 0.461522647292 and squared error in validation as 0.0431836444369.Our R squared value is decreased now and error values is increases compared to previous degree polynomial. This case is called **overfitting**. In overfitting what is happening is we are training the data too much. If we train the too much it can't able to predict the values correctly.

Below is the graph Fig 2.5 test data vs predicted data which shows overfitting. Clearly we can see our model is predicting the values which are higher than the original output. In this case we will stop training our data or else we can apply **RIDGE Regression** to train our data. In ridge regression we will generalize the data.

Now for the same polynomial degree 260 apply ridge regression taking lambda generalization coefficient as one. R squared value is 0.685136435214.After performing the ridge regression R-squared value is increased. It means we got better model now after applying the ridge regression when we encountered the problem of overfitting.

Now again increase the polynomial degree and try to perform Ridge regression on same data. Take polynomial degree as 270 and apply ridge regression. The value of r-squared is 0.244519055156.It decreased suddenly. Which means we can't train our data beyond the polynomial degree 270.

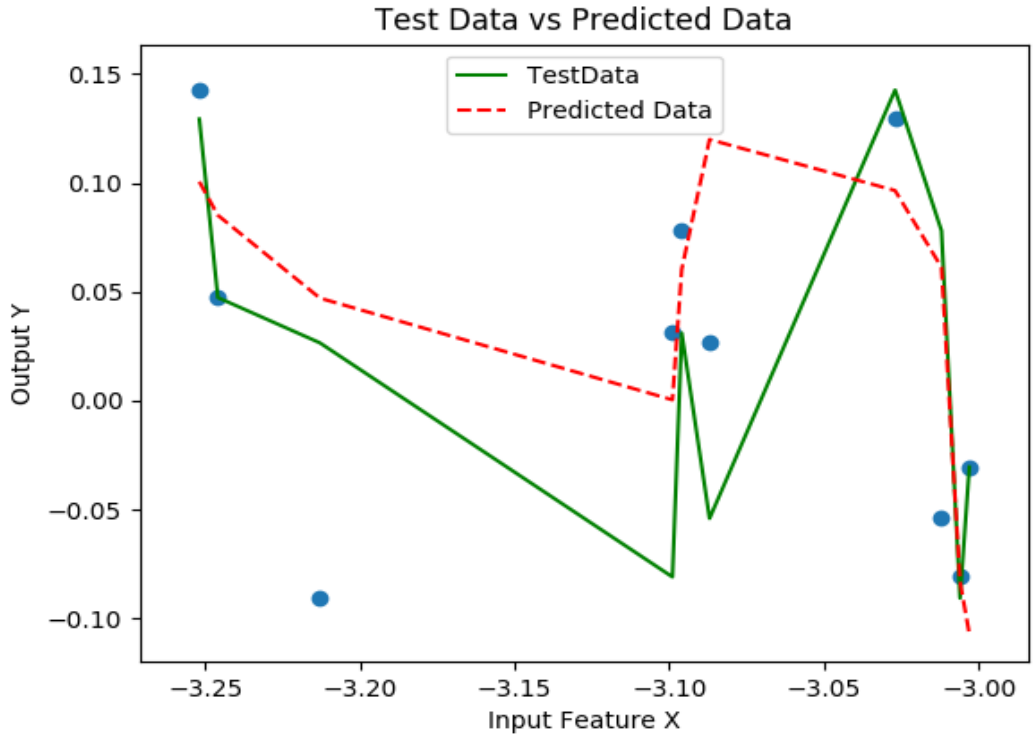



**Fig 2.5: Overfitting result in one dimensional data polynomial degree 260**

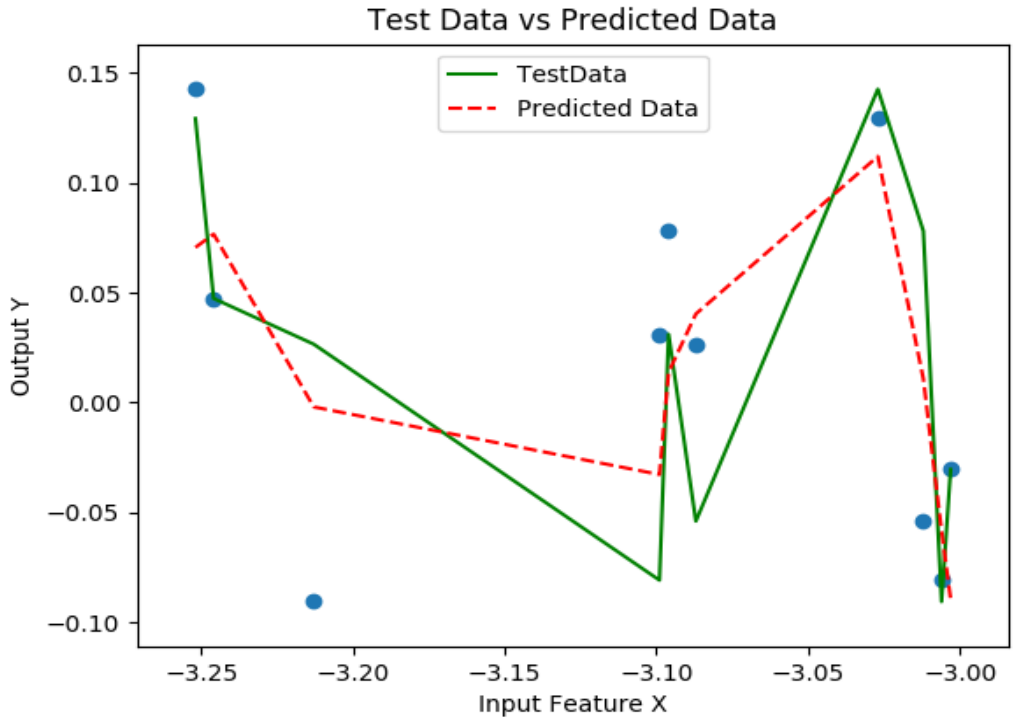


**Fig 2.6 Ridge Regression with Polynomial Degree 260**

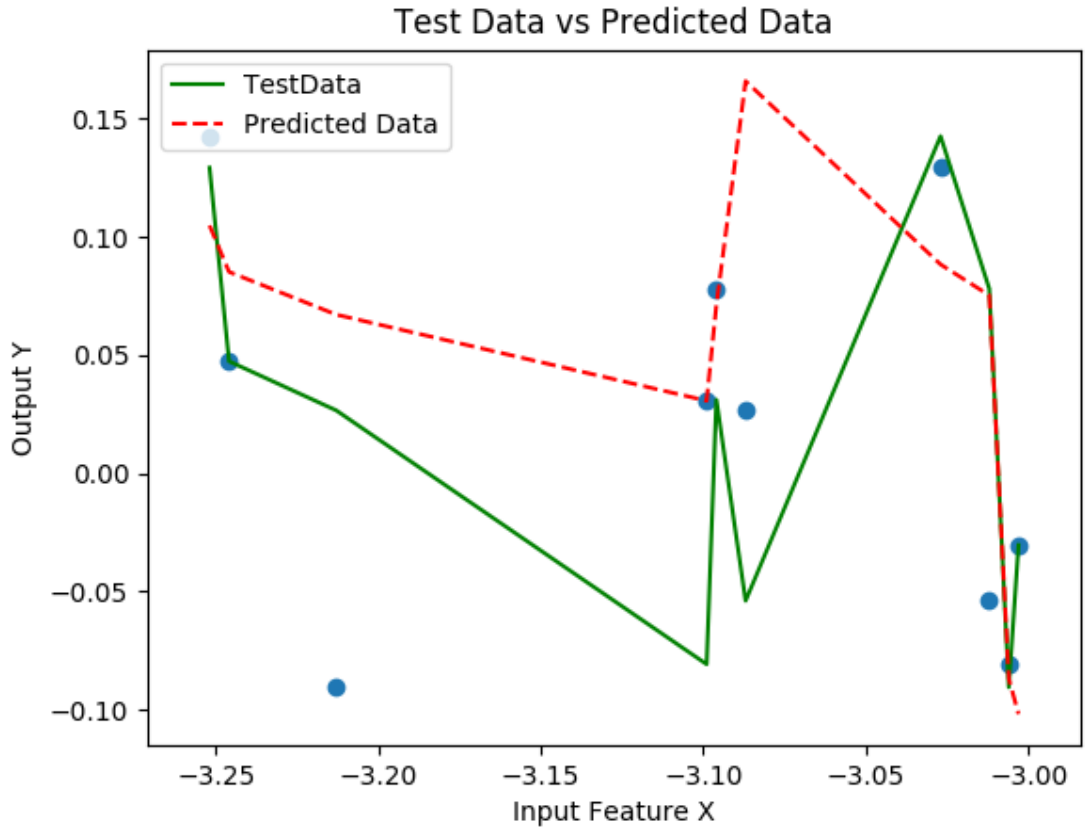



**Fig 2.7 Ridge Regression with Polynomial Degree 270**

In simple Linear Regression for one-dimensional data, the output variable is depending on single feature. But in the real world output variables are depends on many features. Now we will move on to the two Dimensional data.

### 2.2) Dealing with two-dimensional data

In two-dimensional data, output variable is depends on two feature vectors. The equation for the two-dimensional data is $\mathbf{y=a_0+a_1x_1+a_2x_2}$. Where $a_0,a_1,a_2$ are constants, which we call them as weights , $x_1,x_2$ are features which output variable y is depends on.

First simple Linear Regression we will perform on two dimensional data. Below are the few observations.

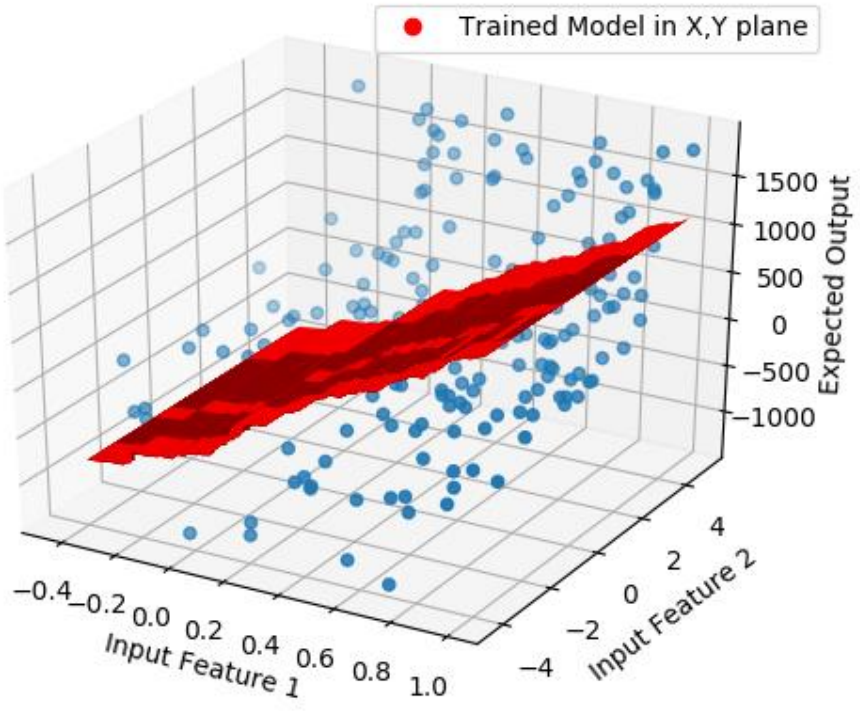



**Fig 2.8 Trained Linear model two Dimensional Data**

In Fig 2.8 graph we have taken two input features $\mathbf{x_1}$ and $\mathbf{x_2}$ and trained our model. R-squared value for the training data is 0.802788144836. The r-squared for validation data is: 0.806502563963.Sum of squares of the error in validation data is 2226661.53548.Error in simple regression is very high.

Below are the graphs related to test data vs predicted data and trained model in two-dimensional data polynomial degree 1.

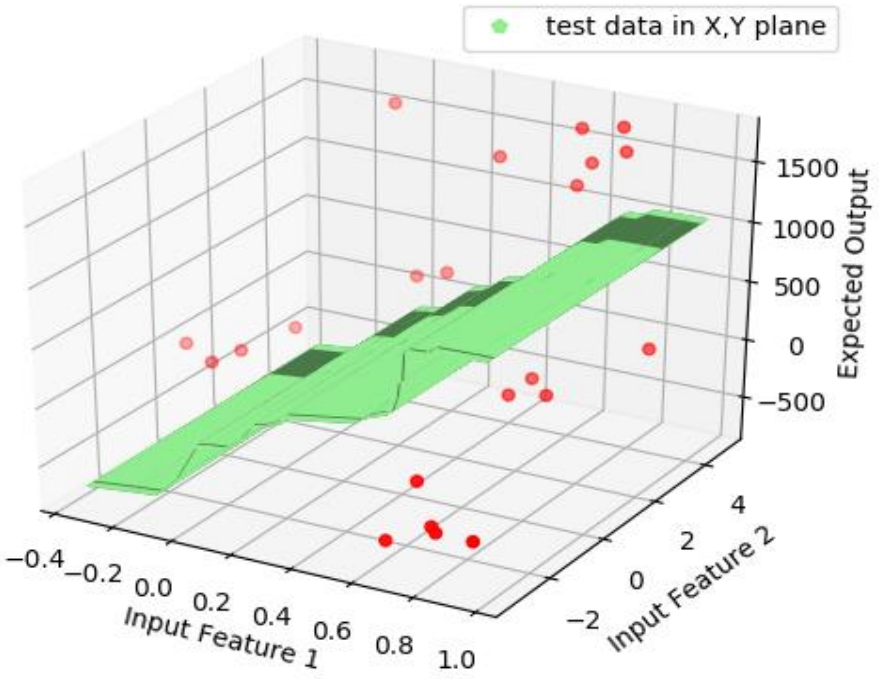



**Fig 2.9 Model with Test Data in Two Dimensional**

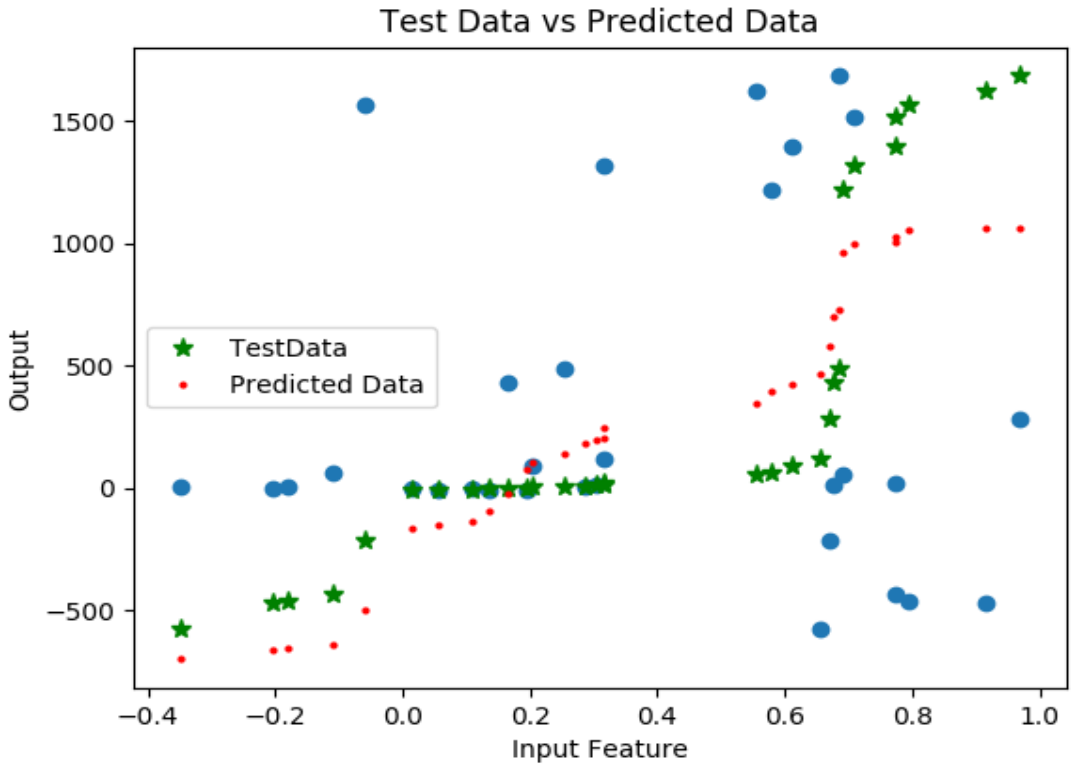



**Fig 2.10 Comparison between Test Data and Predict Data Two Dimensional**

You can see from the Fig 2.10 our model is not predicting the data accurately. Now we will increase our degree of the polynomial to 3 and train our model. The equation for the multivariate polynomial regression with degree 2 is $\mathbf{y=a_0+a_1x_1+a_2x_2+a_3x_1^2+a_4x_1x_2+a_5x^2}$.Below are the results of training our model.

R-squared for train data is 0.999999996836

R-squared for test data is 0.999999997604

R-squared for validation data is 0.999999997717

Sum of squared error for validation data is 0.0262686826954

In all three cases r-squared is pretty close. Sum of squared error is small but we will try to achieve r-squared as 1.So increase our polynomial degree to 5 and train our model again.

R-squared for training data is 1.0

R-squared for test data is: 1.0

R-squared for validation data is: 1.0

Sum of squared error in validation is 1.12874071493e-08

Our Trained model is predicting the data accurately. Observe the Fig 2.13 comparison of test data and predict data.

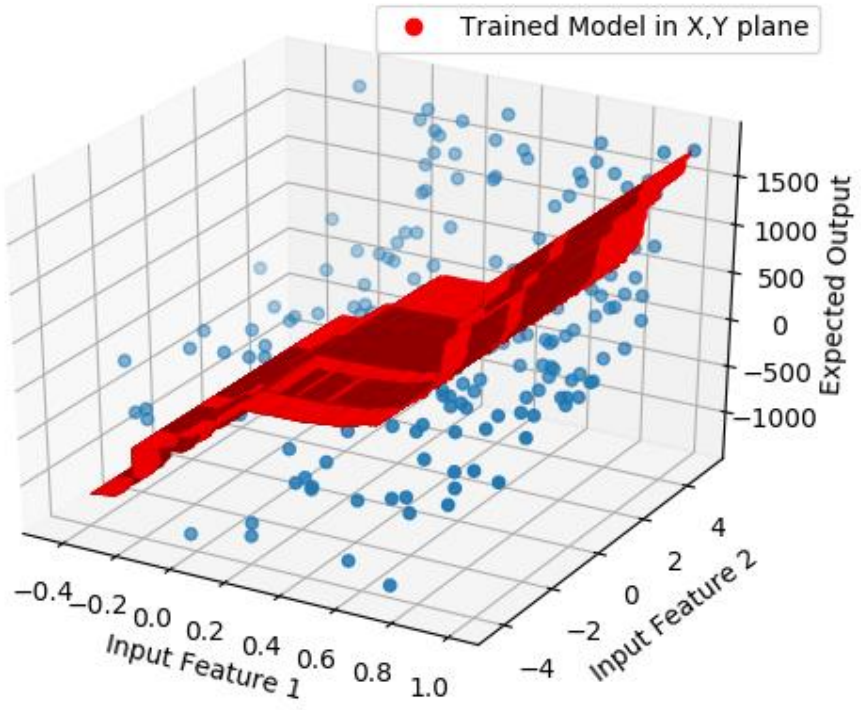



**Fig 2.11 Trained model in Two Dimensional Data Polynomial Degree 5**

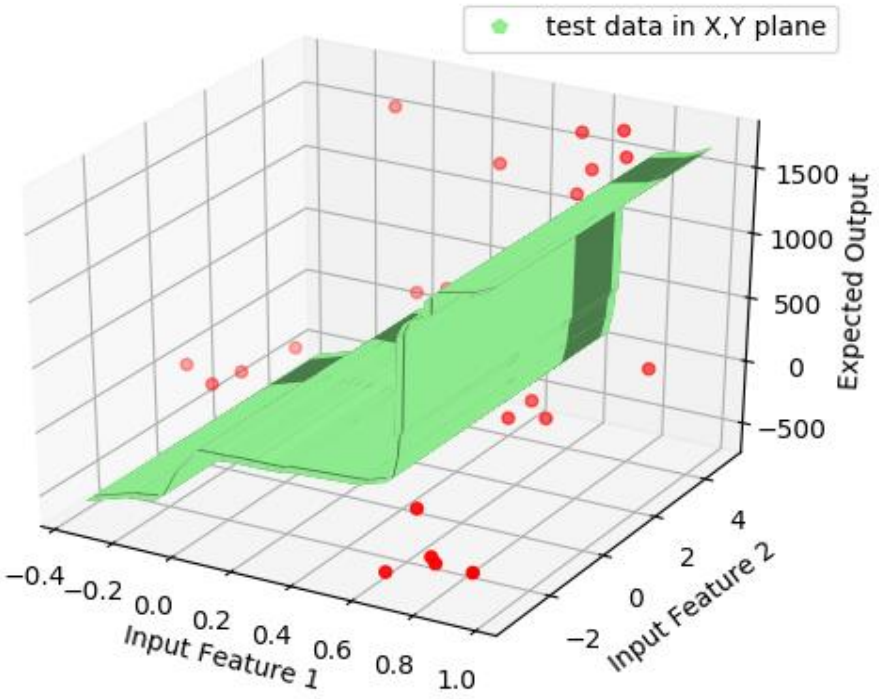



**Fig 2.12 Test Model in Two Dimensional Data Degree 5**

Observe the Fig 2.13 Stars indicates the Test data, Dots Indicates the Predicted Data. For each feature value our model is predicting accurately

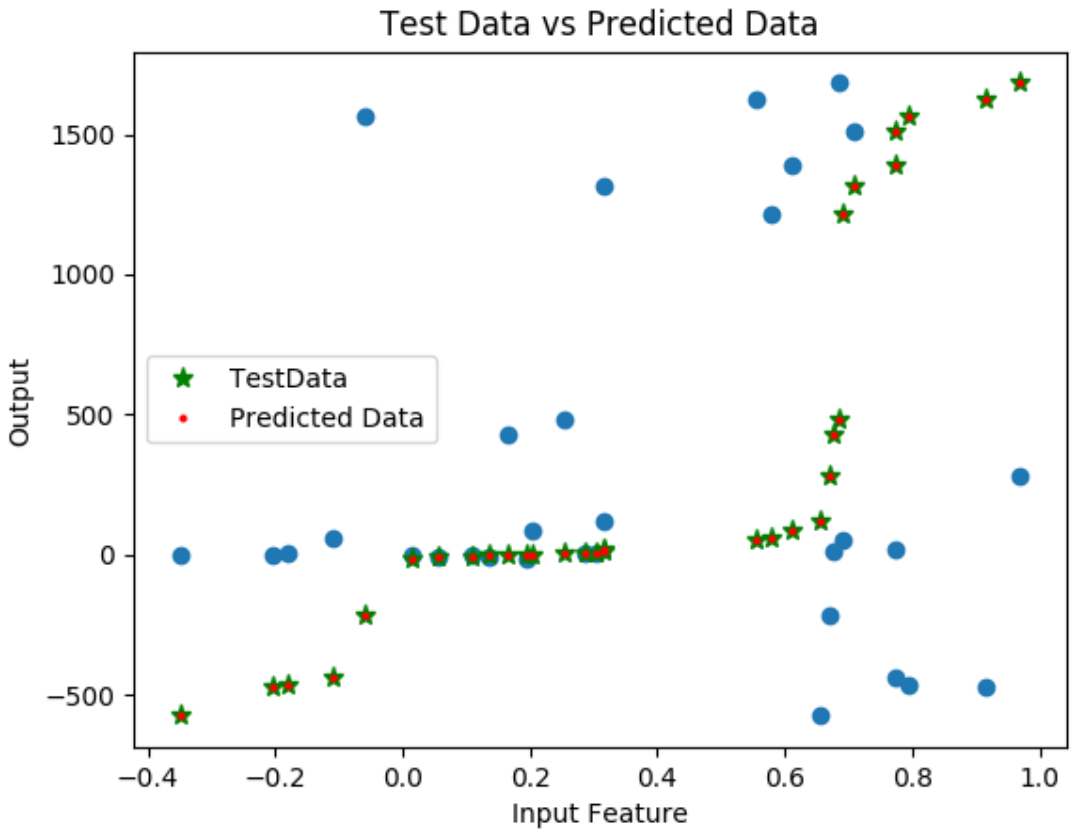



**Fig 2.13 Comparison between Train data and Test Data**

Below is the Results of Ridge Regression of Two Dimensional Data Polynomial Degree 5 with lambda = 1.

R-squared for training data is 0.999999968519

R-squared for test data is 0.999999941795

R-squared for validation data is 0.999999979665

Sum of squared error in validation data is 0.23400564721

Compared to the Polynomial Regression, in Ridge Regression with same degree and with lambda =1 the error is increased and the r-squared value is decreased very less.

### 2.3) Dealing with K-Dimensional Data

Output variable depends on many input features. It's unable to visualize for humans beyond 3-Dimenson so we will take 2 input features and plot graphs.

Below are simple Linear Regression results for K-Dimensional Data first degree Polynomial

R-squared for training data is 0.875109499846

R-squared for test data is 0.880208466941

R-squared for validation data is 0.888351045582

Sum of squared error in validation is 0.576694021395

We got r-squared value near to 1, we will try to train our model with greater degree.

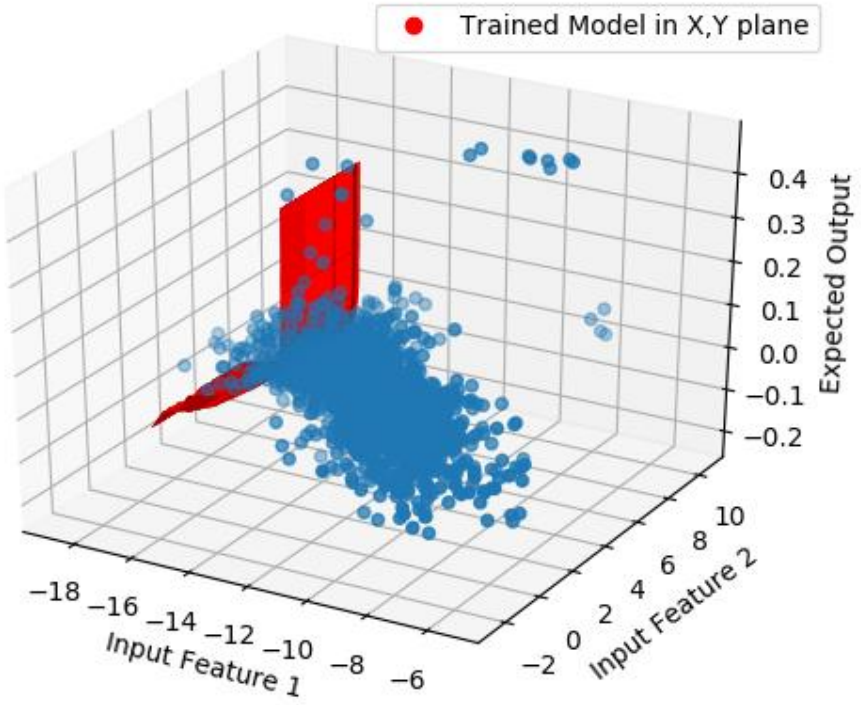



**Fig 2.14 Trained Model in K-Dimensional Data Polynomial Degree 1.**

Observe the Fig 2.14, Model was not that much accurate, because our output data is depends on many feature vectors not only on two feature vectors.

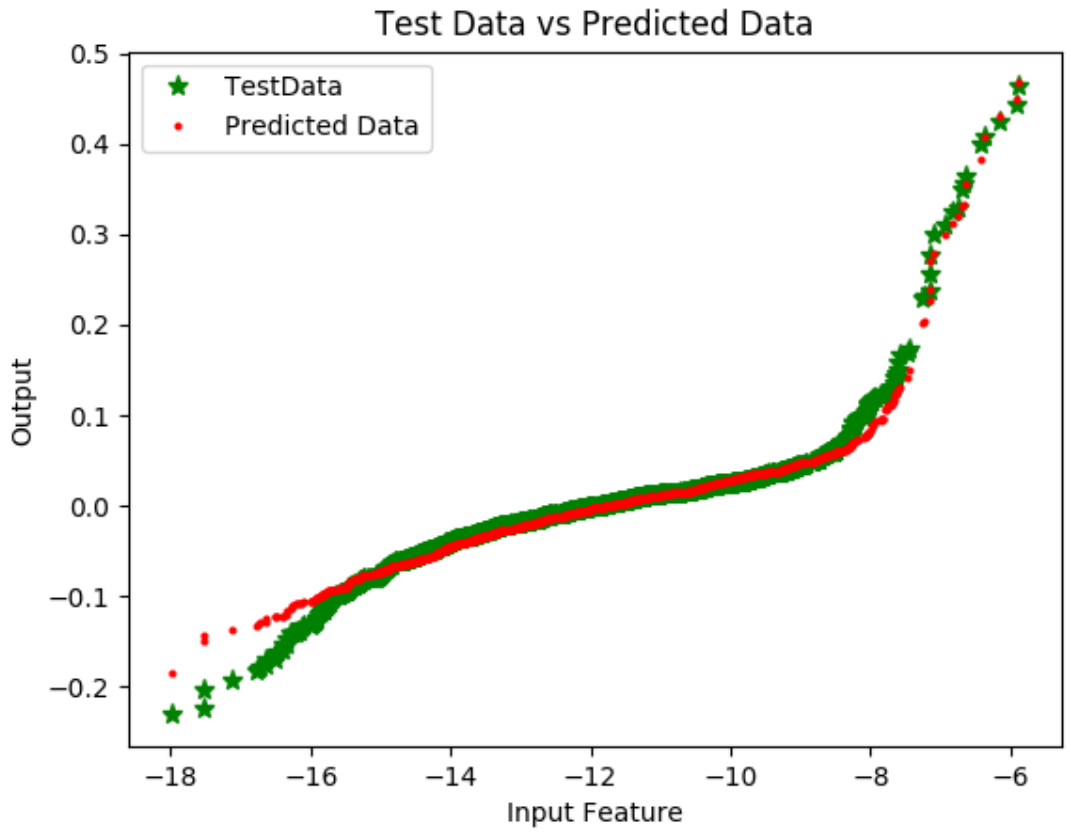



**Fig 2.15 Predicted Data vs Tested Data in K-dimension with Degree 1**

In Fig 2.15 at some input Feature values our model is not predicting correctly. Below are the results of K-dimensional data with degree Polynomial 2.

R-squared for training data is 0.944259480606

R-squared for test data is 0.942291376693

R-squared for validation data is 0.947613985719

Sum of squared error in validation data is 0.270586512861

R-squared value is close to 1.Now observe the graphs.

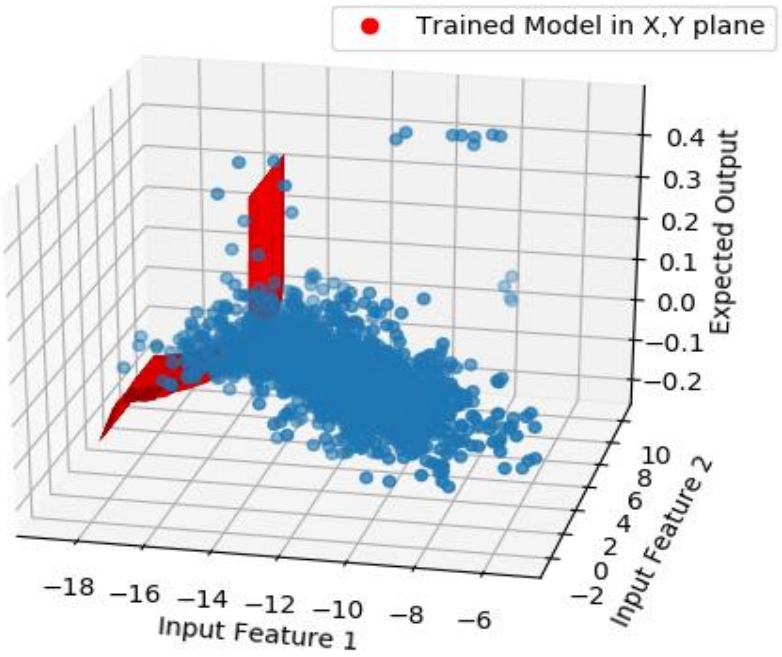



**Fig 2.16 Trained Model K-Dimensional Data Polynomial Degree 2**

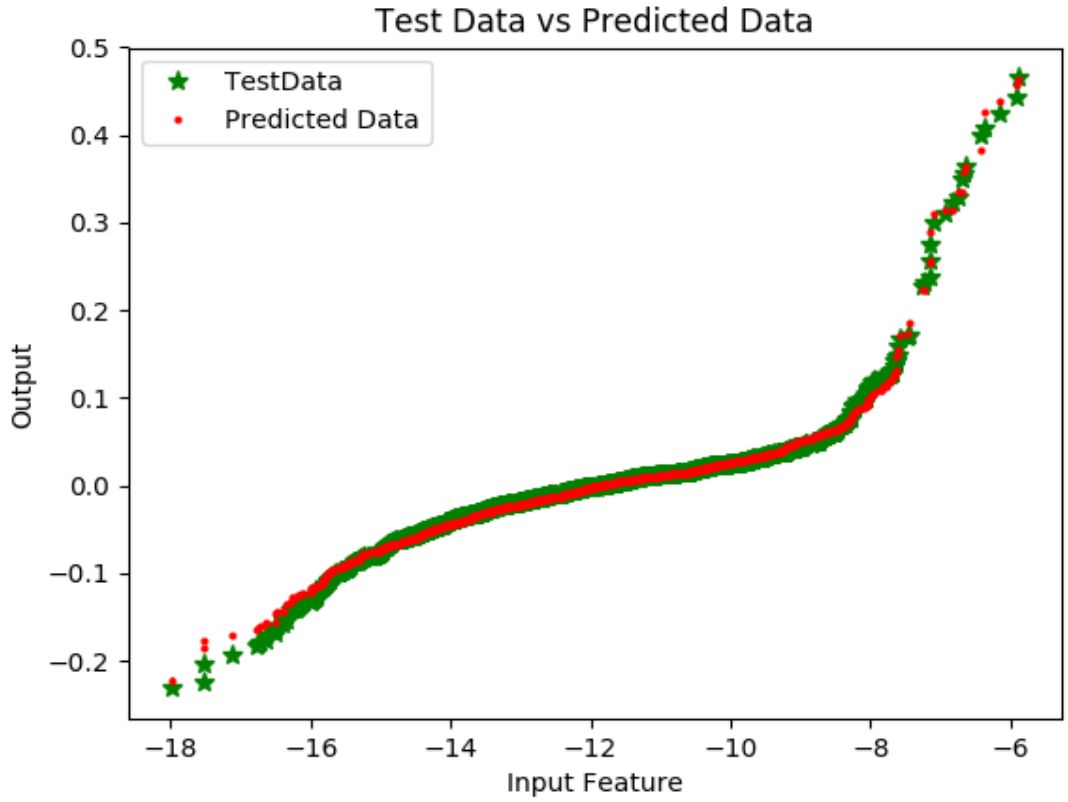



**Fig 2.17 Comparison of Tested Data vs Predicted Data K-Dimensional Polynomial Degree 2**

Now try to increase the degree of polynomial and train our model. It's showing out input Feature matrix is Singular Matrix. So now perform Ridge Regression. These are the observations of Ridge Regression with lambda values 0.1 and 1 for K-Dimensional Data with Polynomial Degree 3.

Lambda = 0.1

R-squared for training data is 0.959114748963

R-squared for test data is 0.956109916293

R-squared for validation data is 0.959144931205

Sum of squared error for validation data is 0.21102637316

Lambda = 1

R-squared for training data is 0.957548235294

R-squared for test data is 0.953723991911

R-squared for validation data is 0.958159875023

Sum of squared error for validation data is 0.216114428068

As we can see from above two results, as we increases lambda value error is increasing and r-squared value is decreasing.