CSL603 Machine Learning
Lab - 2

Abhishek Chowdhry 2015csb1002@iitrpr.ac.in
Pratik Chhajer 2015csb1025@iitrpr.ac.in

September 22, 2017

Contents

1 Linear Ridge Regression

1.1 Introduction

In this lab we implemented **Linear Regression** to predict the age of Abalone (a type of snail). We used ridge regression to train data.

1.2 Pre-processing

First data was read from file named *linregdata*. There were total of 9 columns separated by comma. All the features were stored in 2d matrix named INPUT_X and corresponding y's were stored in 1d matrix named INPUT_Y.

For creating a new sample each time, first the whole 2d matrix X which contains all the features was shuffled(row's were shuffled) accordingly 1d matrix Y was also shuffled. Then let say we want f fraction of data to be training data then first f*Total_Rows were selected as trainging data and rest of the rows were selected as test data.

Initially data was standardized by using whole dataset including training and test dataset. But later on only training dataset was used for the purpose of standardization.

Further procedure of standardization was as follow: first mean and standard deviation was calculated for each feature, then each feature's value was subtracted by that feature's mean and this difference is divided by standard deviation of that feature.

1.3 Training

For training first we calculate weight 1d matrix named W using

$$W = (X^T X + \lambda I)^{-1} X^T Y$$

For calculating W we implemented a function named ***mylinridgereg(X, Y,lamda)***.

1.4 Testing

For testing we calculate mean squared error by finding expect y's by implementing function named ***mylinridgeregeval(X, W)*** which basically returns a 1d matrix containing predicted y's.

1.5 Parameters & Their Dependencies

We have two parameters here, namely λ and **fraction**. While performing experiment λ took values from [0.0,5.0] with increment of 0.05. While performing experiment fraction took values from [0.1,0.9] with increment of 0.1. Note here fraction = 0.1 represents 10 percent of dataset was taken as training. For each value of λ and **fraction** training and test error was calculated 100 times and then their average was considered.

1.6 Error Calculation

Mean squared error for each λ and **fraction** was calculated. There is a file named **error_data** which contains 4 columns, first column contains λ, second column contains **fraction** and third column contains training error corresponding to that λ and fraction and fourth column contains test error corresponding to that λ and fraction.

1.7 Observations & Conclusions

Mean squared error for each λ and **fraction** was calculated. There is a file named **error_data** which contains 4 columns, first column contains λ, second column contains **fraction** and third column contains training error corresponding to that λ and fraction and fourth column contains test error corresponding to that λ and fraction.

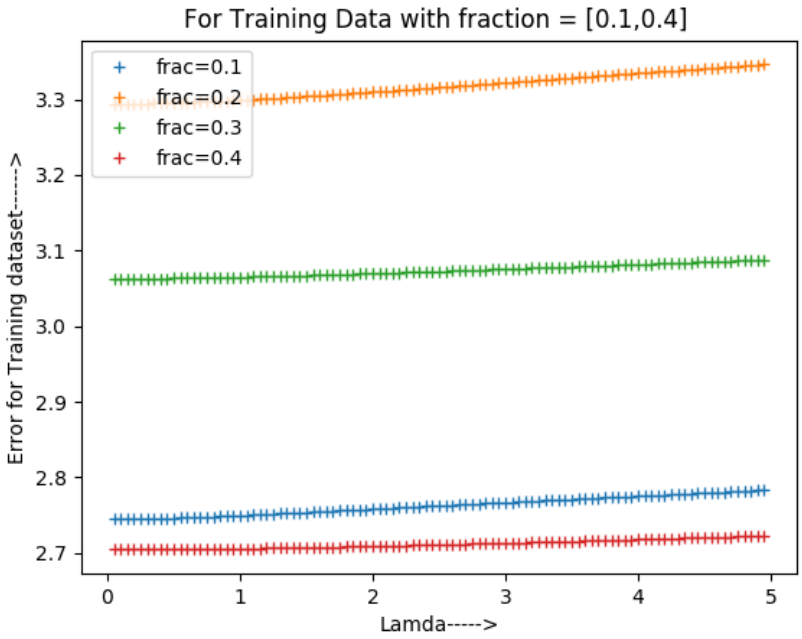
So we first observed effect of **fraction** and λ. We plot λ vs mean squared error for a given fraction.

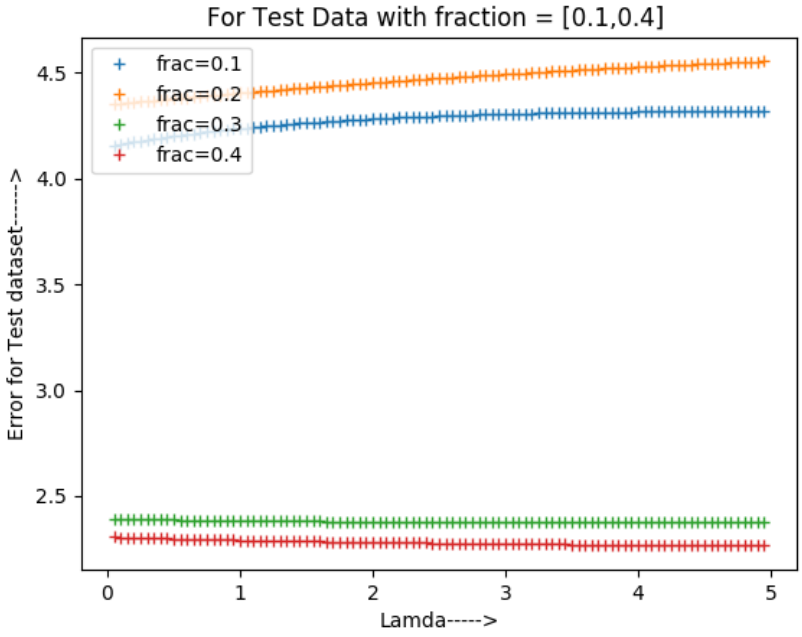
For clarity purpose we plot $fraction = 0.1, 0.2, 0.3, 0.4$ in one plot and $fraction = 0.5, 0.6, 0.7, 0.8, 0.9$ in second plot.

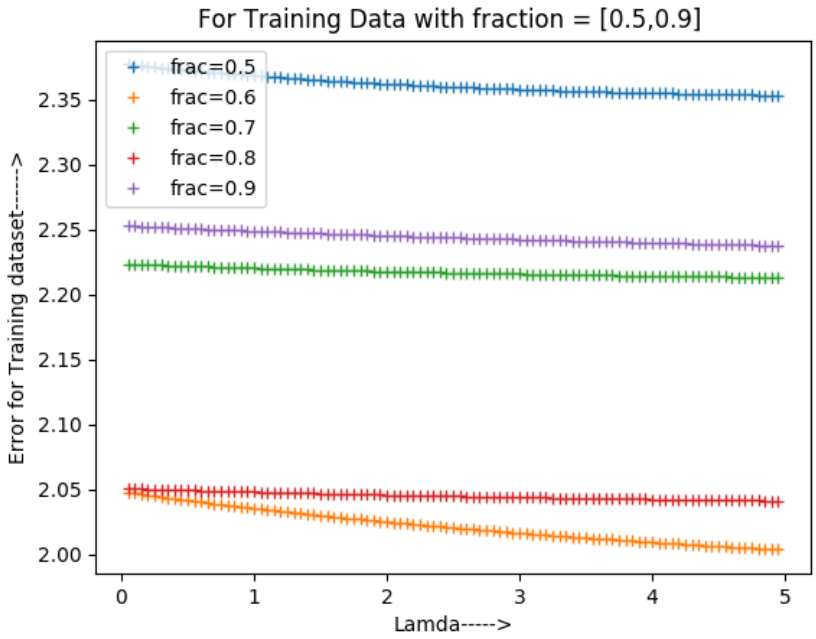
We plot different graphs for training error and test error.

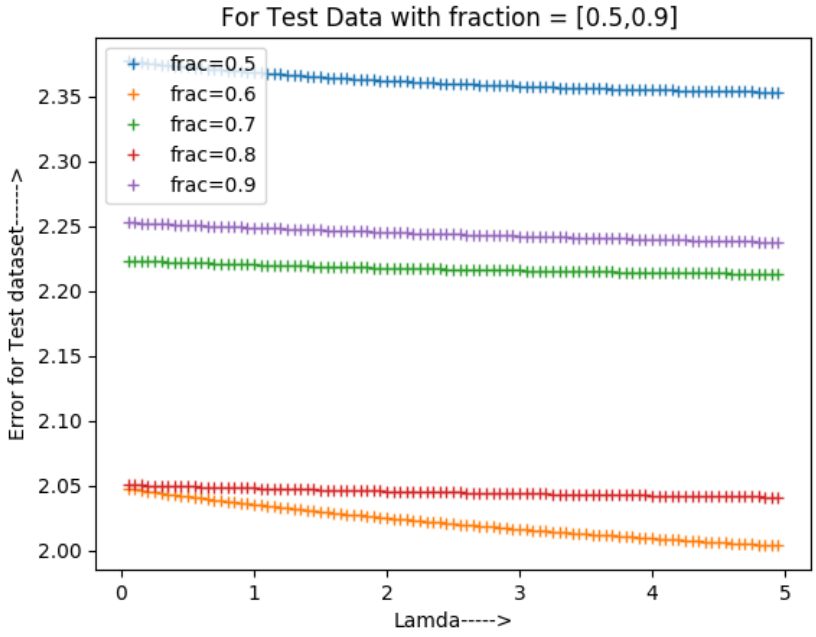
Observing below four graphs it can be seen that:

- For a given fraction, if we increase λ from 0 then upto a certain point mean squared error decreases then a minima is reached and after that minima again mean squared error start increasing.
- Values of mean squared error don't differ significantly ie.. if we change λ slightly then mean squared error also differ slightly.


For Training Data with fraction = [0.1,0.4]
frac=0.1
frac=0.2
frac=0.3
frac=0.4
Error for Training dataset----->
Lamda----->
3.3
3.2
3.1
3.0
2.9
2.8
2.7
0
1
2
3
4
5


For Test Data with fraction = [0.1,0.4]
frac=0.1
frac=0.2
frac=0.3
frac=0.4
Error for Test dataset----->
Lamda----->
4.5
4.0
3.5
3.0
2.5
0
1
2
3
4
5

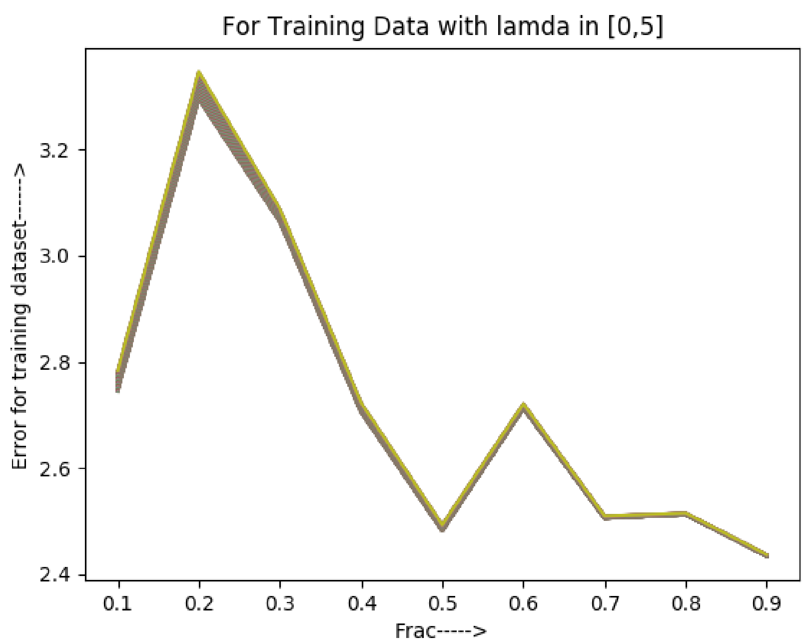

For Training Data with fraction = [0.5,0.9]
frac=0.5
frac=0.6
frac=0.7
frac=0.8
frac=0.9
Error for Training dataset----->
Lamda----->
2.35
2.30
2.25
2.20
2.15
2.10
2.05
2.00
0
1
2
3
4
5

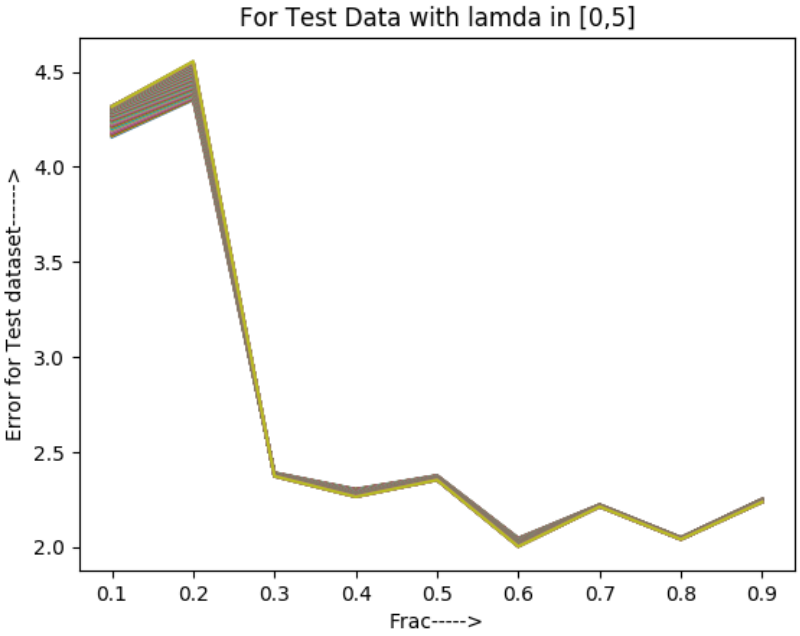

For Test Data with fraction = [0.5,0.9]
frac=0.5
frac=0.6
frac=0.7
frac=0.8
frac=0.9
Error for Test dataset----->
Lamda----->
2.35
2.30
2.25
2.20
2.15
2.10
2.05
2.00
0
1
2
3
4
5

We plot fraction vs mean squared error for a given λ.
We plot for $\lambda = 0.00,0.05,0.10,.....,4.95$.
We plot different graphs for training error and test error.

Observing below two graphs it can be seen that:

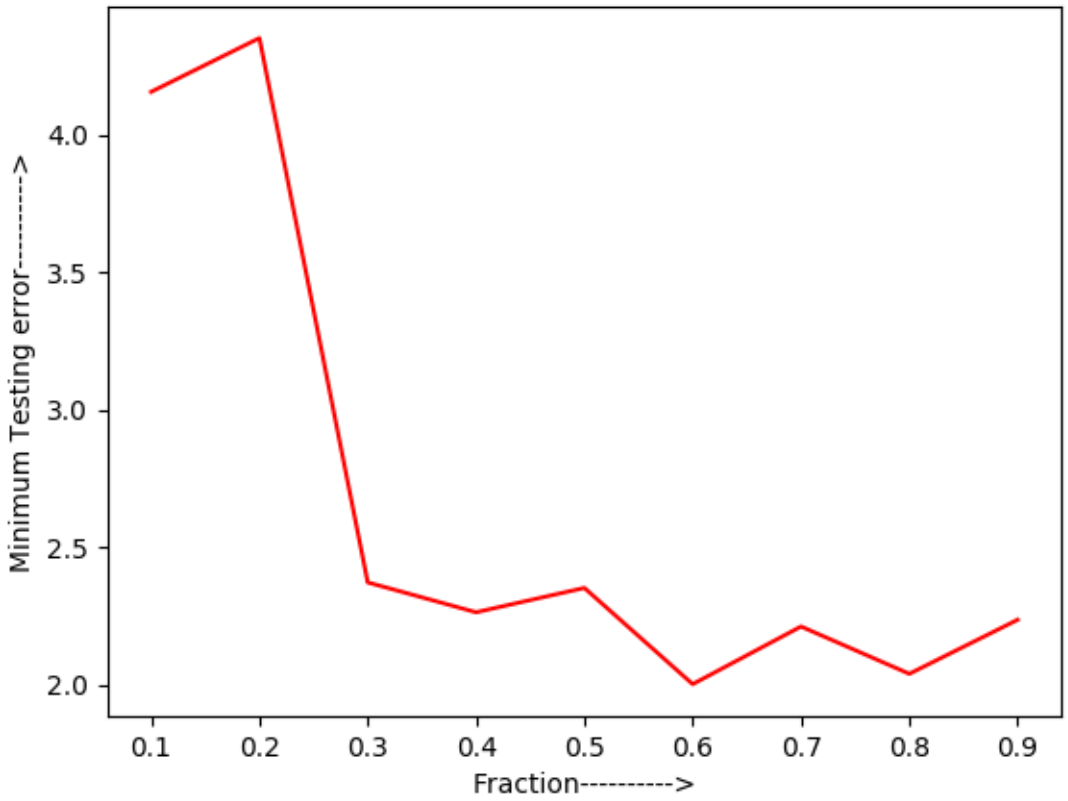
- With increase in fraction of training dataset the mean squared error is decreasing.
- Traininng error is saturating little faster.
- Difference between mean squared error for different values of λ doesn't vary much.

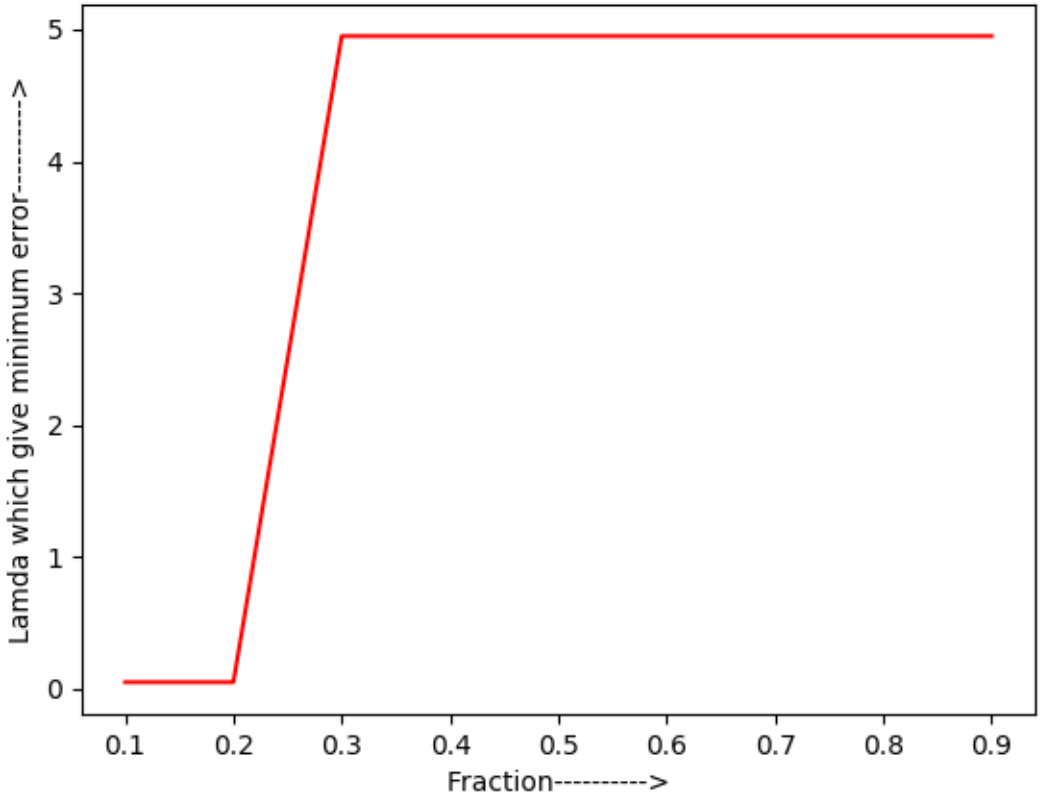



For Test Data with lamda in [0,5]
Error for Test dataset----->
Frac----->
4.5
4.0
3.5
3.0
2.5
2.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9

Here we observed the minimum mean squared error for given fraction ie.. we pick that λ which gives minimum mean squared error for that particular fraction.

We plot fraction vs minimum mean squared error and fraction vs corresponding λ.



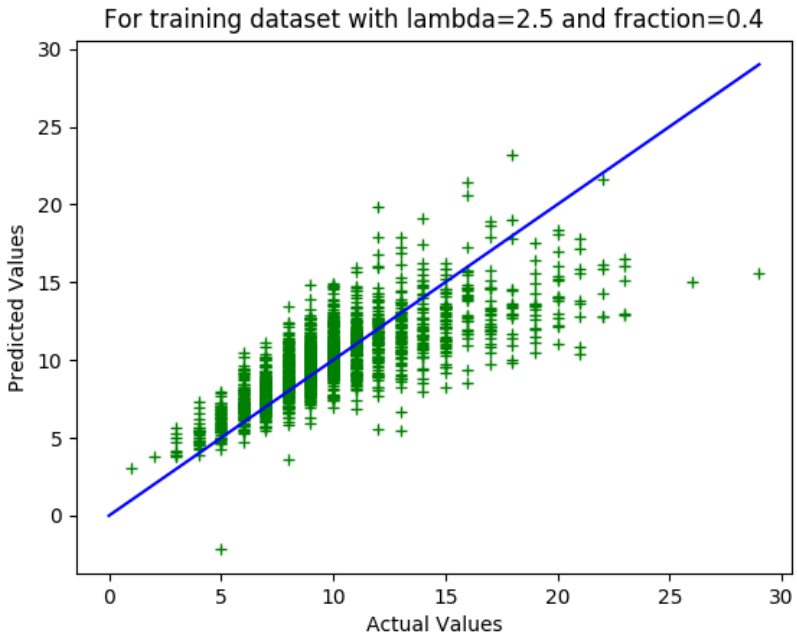


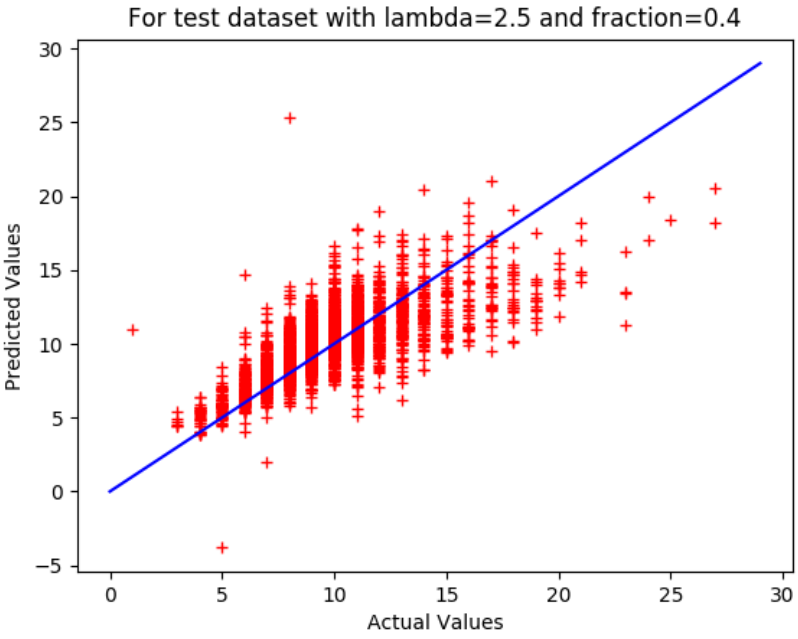
From above two graphs following observations were made:

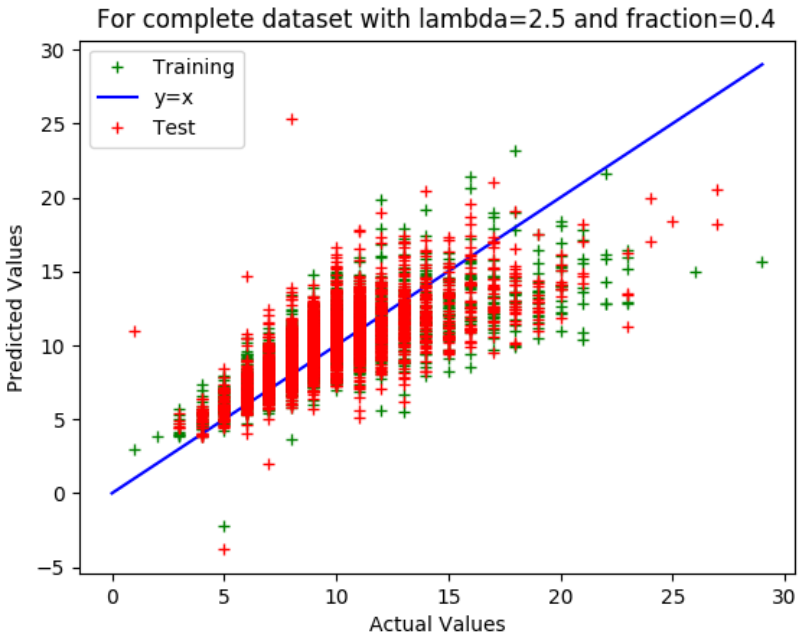
- As fraction of training data increases minimum mean squared error decreases.
- There are few inconsistencies in above statement which can explained with the help of over-fitting.
- For most of the fraction $\lambda = 4.95$ is giving minimum mean squared error, this means that minima is lying after $\lambda = 4.95$.

After observing all the above graphs for mean squared errors, it can be easily observed that for λ close to 2.5 we are getting minimum mean squared error. Also for fraction = 0.4 we see the minimum mean squared error. So we choose $\lambda = 2.5$ and fraction = 0.4 and then plot the actual values and predicted values.

We plot three graphs one for training dataset, one for test dataset and one both combined. To get more insights we also plotted y=x line along with each of these three graphs.


For training dataset with lambda=2.5 and fraction=0.4
Predicted Values
Actual Values
30
25
20
15
10
5
0
0
5
10
15
20
25
30


For test dataset with lambda=2.5 and fraction=0.4
Predicted Values
Actual Values
30
25
20
15
10
5
0
−5
0
5
10
15
20
25
30



Following observations can be made after seeing the above three graphs:

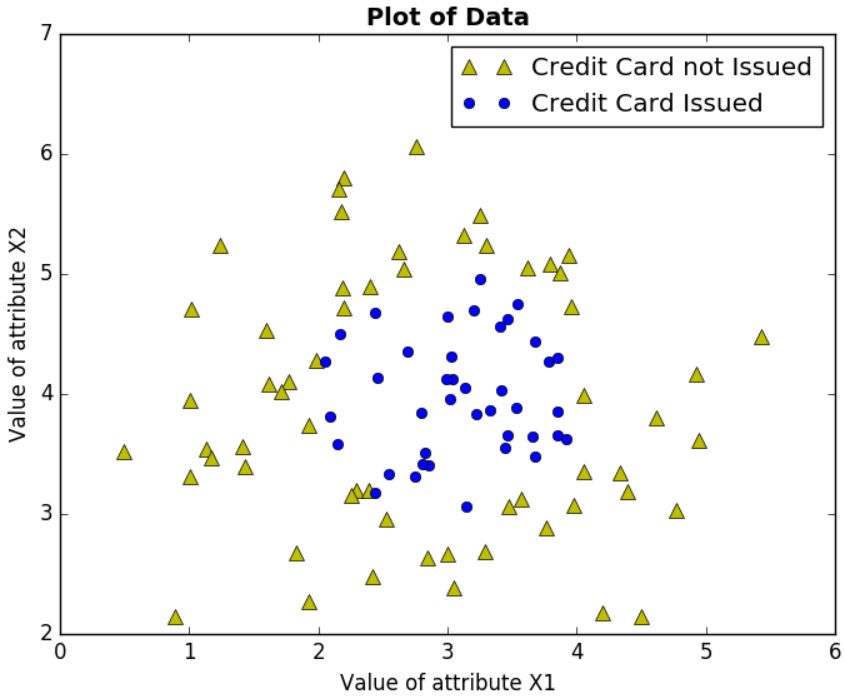
- Training dataset has more point clustered closer to line $y = x$ as expected
- The density of points clustered closer to line $y = x$ is very high.
- There are noise points but number of noise points seems to be small relative to total dataset which contains 4000+ points.

2 Regularized Logistic Regression

In this question we will be implementing Regularized Logistic Regression implemented using Gradient Descent and Newton Raphson Methods in order to predict whether a credit card should be issued to an individual based on the values of two attributes of that individual. We will be determining whether the credit card application of an individual of an individual should be accepted or rejected. To learn the models we will be using the data set of the past applications of credit cards made by individuals along with their outcomes which is provided to us in a text file.

2.1 Plotting of Data

We read the data from the file and plotted it on a graph with the two attributes(X1 and X2) on the X and Y axis respectively. The plotted graph is given below:



2.2 Regularized Logistic Regression

In this section we implemented Regularized Logistic Regression using two different methods: Gradient Descent and Newton Raphson for classifying the credit card applications.i.e. Whether the credit card application of an individual should be accepted or not. We learned the models using a data set of previous applications along with their outcomes. We used two attributes(X1 and X2) of an individual to classify their applications.

2.2.1 Logistic Regression using Gradient Descent

We first implemented Regularized Logistic Regression using gradient descent. We used the equation given below for parameter update:

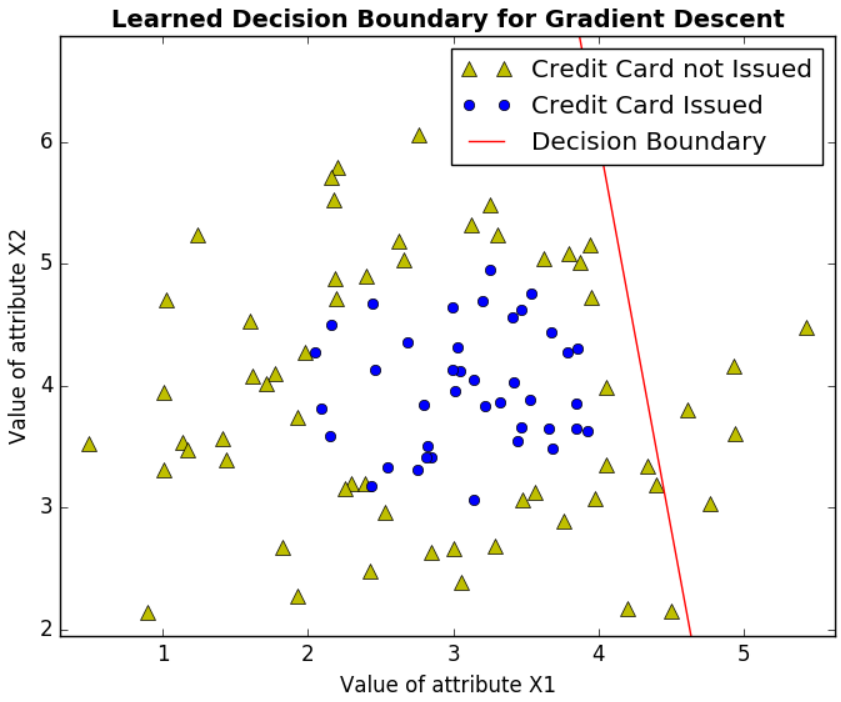
$$W_j = W_j - \alpha[\frac{1}{m}\sum_{i=1}^{m}(h_W(x^{(i)}) - y^{(i)})x_j^{(i)}) + \frac{\lambda}{m}W_j] \tag{1}$$

We used the above equation for updating each weight W_j at each iteration till we converged at the optimal solution. We ran gradient descent for several iterations using multiple parameters. Some observed results are given below.

Values of parameters used:
Learning rate(α): 0.1
Number of iterations: 8000
Regularization factor(λ): 1

The values obtained for W were:
w_0: -1.45707935
w_1: 0.2948165
w_2: 0.04599758

The linear decision boundary obtained by these parameters is shown in the graph given below:



2.2.2 Logistic Regression using Newton Raphson

In this part we implemented Regularized Logistic Regression using Newton Raphson method. We used the equation given below for parameter update:

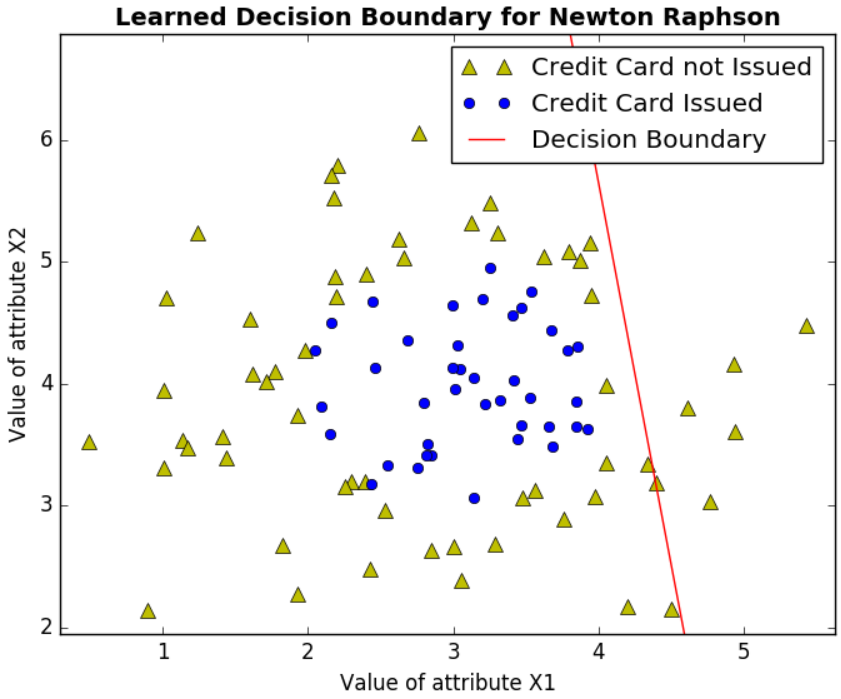
$$W_j = W_j - H^{-1}\frac{dJ(W)}{dW} \tag{2}$$

We used the above equation for updating each weight W_j at each iteration till we converged at the optimal solution. We ran newton raphson for several iterations using multiple parameters. Some observed results are given below.

Values of parameters used:
Learning rate(α): 0.1
Number of iterations: 8
Regularization factor(λ): 1

The values obtained for W were:
w_0: -1.51115535
w_1: 0.30819383
w_2: 0.04917364

The linear decision boundary obtained by these parameters is shown in the graph given below:



Observations

We generated 1000 test examples to compare the accuracy of the models learned from Regularized Logistic Regression through Gradient Descent and Newton Raphson. We also compared their accuracy on the training data set and the following results were observed:

Gradient Descent

Accuracy on Training Data set: 55%
Accuracy on Test Data set: 55%
Number of iterations taken to converge: about 8000

Newton Raphson

Accuracy on Training Data set: 54%
Accuracy on Test Data set: 55%
Number of iterations taken to converge: about 10

From the above observations, it is clearly visible that both Gradient Descent and Newton Raphson converge to the same model which can be seen by their accuracies on test and train data sets. Although the time taken by both the methods would be different but they eventually learn the same model.

Another thing which can be concluded from the above observations is that Newton Raphson converges to the optimal solution quickly as compared to Gradient descent method. This is clearly visible by comparing the number of iterations taken to converge for both of them.

So we can say that although both of them converge to the same solution, Newton Raphson converges quickly as compared to the Gradient Descent method.

2.3 Is the data linearly Separable?

By looking at the plot of the data in part one, it can be seen that the data is not linearly separable. There is no one straight line which can separate the data into two parts each having objects with the same label.

2.4 Feature Transformation

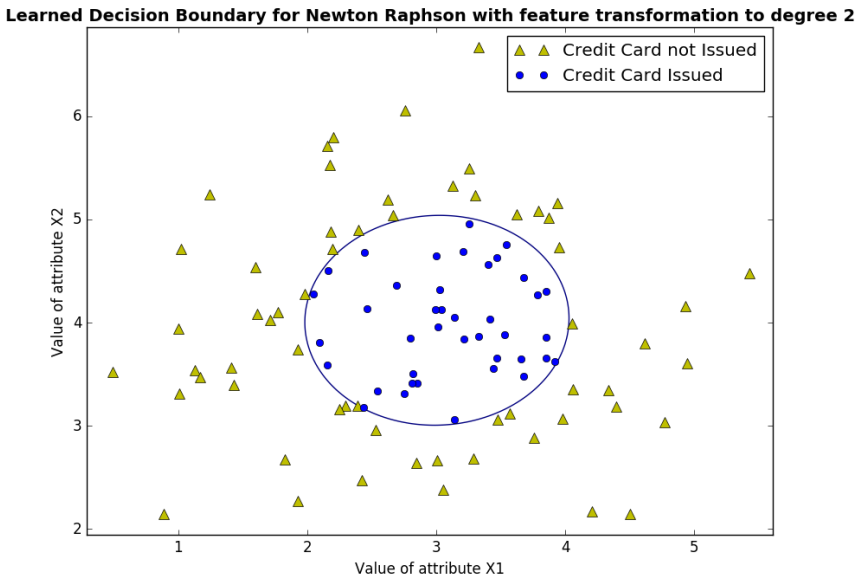
We saw in the previous part that the data given to us is not linearly separable. That is the reason we got such bad acuracies when we tried to separate the data using a linear decision boundary.
One method to improve this is by transforming the features to a higher dimensional space.i.e. Generating more features from the features given to us. We can do this by making polynomial terms of the features given to us(X1 and X2) to some higher degree terms.
This method of transforming the features to some higher dimension is called Feature Transformation. Once the features have been transformed to some higher dimensional space, then we can run Regularized Logistic Regression on these features to get a more accurate decision boundary. Here we used Regularized logistic regression with Newton Raphson method because it converges to the optimal solution quickly. The best fit came when the data was transformed to a space of degree 2. This is explained in the next part.

2.5 Best Degree of transformation

We tried various degrees of transformation and their plots along with their accuracies on test and training data sets. The results are given below.

2.5.1 Degree 2

The plot for Degree 2 is given below:

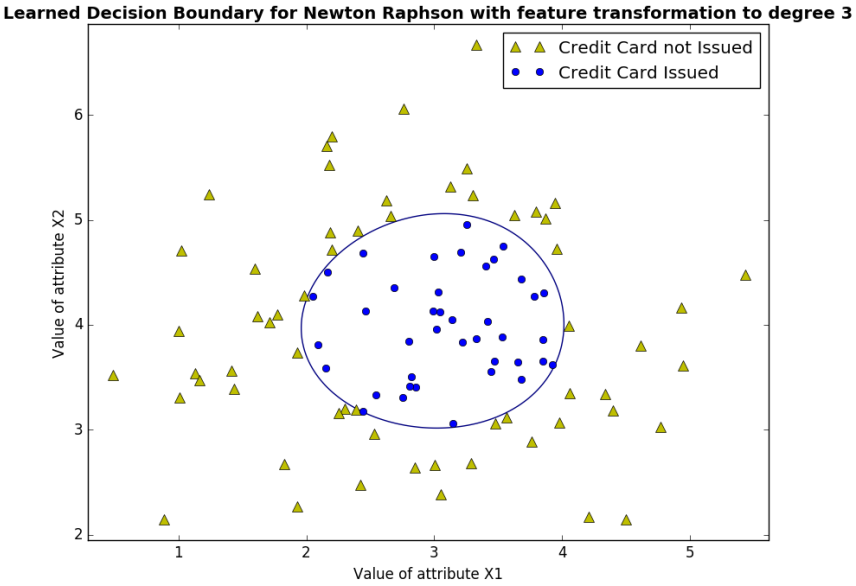


The accuracies of the learned model on different Data sets are given below:

Accuracy on the Training Data Set: 100%
Accuracy on the Test Data set: 99%

2.5.2 Degree 3

The plot for Degree 3 is given below:

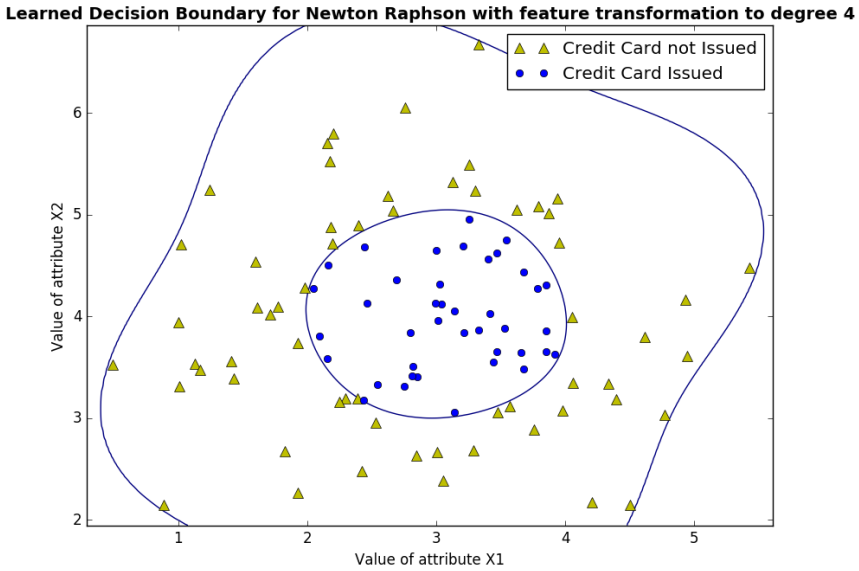


The accuracies of the learned model on different Data sets are given below:

Accuracy on the Training Data Set: 100%
Accuracy on the Test Data set: 100%

2.5.3 Degree 4

The plot for Degree 3 is given below:



The accuracies of the learned model on different Data sets are given below:

Accuracy on the Training Data Set: 100%
Accuracy on the Test Data set: 90%

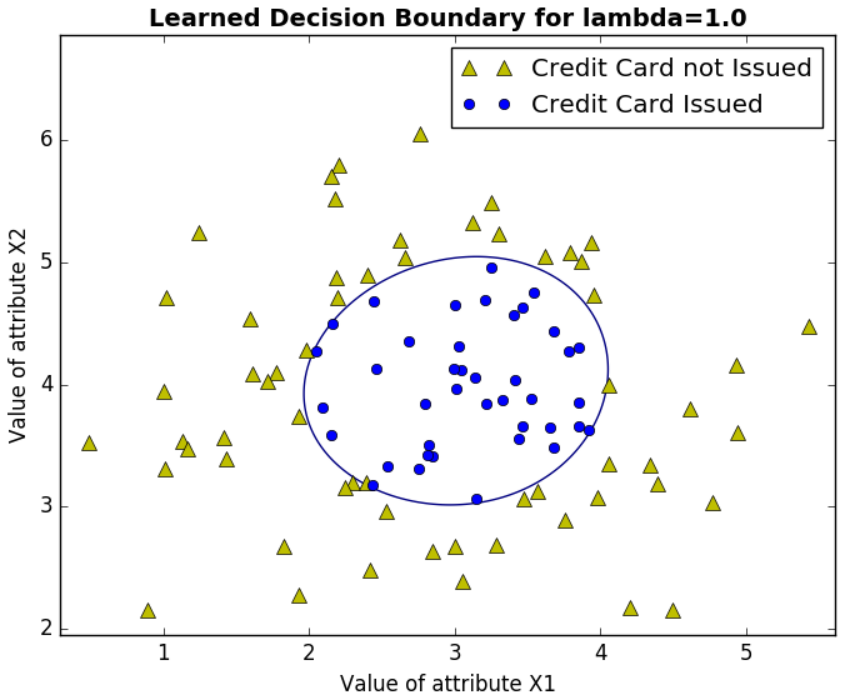
By looking at the above three plots and their accuracies on training and test data sets it can be said that the degree of transformation which gives the best separation between different classes and a hence the best accuracy is degree 3. As we increase the degree after that although the training accuracy remains almost 100% but the accuracy on the test data set decreases drastically. This happens mainly because of over fitting of data with higher degree polynomials. So although feature transformation to a higher degree may improve the accuracy of our learned model, it is also susceptible to over fitting.

2.6 Varying Regularization Parameter

Regularization reduces over fitting in training data by penalizing the solutions having higher degree terms. So by increasing the value Regularization Parameter(λ) we can reduce over fitting. But if the value of λ increases beyond a particular limit, under fitting of data begins to occur. This is shown in the plots given below.

2.7 Over fitting of Data

When we took λ=1, The classifier learned a model which gave zero training error on the training data set, but did not worked so well on unseen test data sets. The classifier in this case over fitted the data. This can be seen in the graph below:



2.8 Underfitting of Data

When we took λ=25, the classifier learned a model which gave very bad results not only on the training data set but also on the test data sets. The classifier in this case under fitted the data. This can be seen in the graph below: