ELL409-Assignment1
Linear Models for Regression and Classification

E Spandana - 2020EE10489

Ekansh Singh 2020EE10490

Rashee Agarwal 2020EE10539

18/09/2022

Contents

1 Multivariate Linear Regression

1.1 Introduction

The simplest linear model for regression is one that involves a linear combination of fixed non linear functions of input variables which is of the form

$$y(\mathbf{x}, \mathbf{w}) = \sum_{j=0}^{M-1} w_j \phi(\mathbf{x}) + \epsilon = \mathbf{w}^T \phi(\mathbf{x}) + \epsilon$$

where $\mathbf{x} = (x_1, ...x_D)^T$, $\mathbf{w} = (w_0, ...w_{M-1})^T$, $\phi = \phi_0, ...\phi_{M-1}^T$ where $\phi_j(\mathbf{x})$ are known as basis functions and ϵ is the the noise which is normally distributed ($\epsilon \sim N(0, \sum)$) for the complete dataset.

1.2 Understanding the data

- Correlation between two variables is the measure of how one variable moves in relation to other variable i.e. it is the degree of relationship between them given by:

$$\text{Correlation coefficient} = \frac{Covariance(x,y)}{std\ dev(x) * std\ dev(y)}$$

- The coefficient was calculated using the inbuilt function *corr* and then represented it as a heatmap which is present in the *seaborn* library

- We can see in the correlation matrix that the values for x1 with x1, x2 and x3 are positive and are very close to 1 which shows that if one of these factor increases then the others would also increase and vice versa.

	x1	x2	x3	x4	x5	x6	t
x1	1.000000	0.983218	0.971908	0.847482	0.475974	-0.996590	0.970183
x2	0.983218	1.000000	0.973034	0.847281	0.476185	-0.983578	0.971919
x3	0.971908	0.973034	1.000000	0.851716	0.480625	-0.971250	0.998455
x4	0.847482	0.847281	0.851716	1.000000	0.409703	-0.845230	0.878992
x5	0.475974	0.476185	0.480625	0.409703	1.000000	-0.476806	0.479483
x6	-0.996590	-0.983578	-0.971250	-0.845230	-0.476806	1.000000	-0.969157
t	0.970183	0.971919	0.998455	0.878992	0.479483	-0.969157	1.000000

Figure 1: Correlation Matrix

Figure 2: Heatmap for the correlation coefficients

- Similarly we see for other variables but the magnitude could vary showing the corresponding dependence of variables on each other.
- For x6 and x4 we can see that the value is negative which shows that on increasing one of the variable the other would decrease and vice versa.

1.3 Effect of batch size

We implemented least square based regression without regularization using Batch Gradient Descent (algorithm which minimizes the cost function using each example in the training data set and hence giving better results but taking much more time) and Stochastic Gradient Descent (a variation of Batch Gradient Descent just with a random example set and not the complete data set and hence it takes less time but doesn't always give the most accurate results)

```
[  21873.03169781   33058.17395797  334815.67831527 6075312.61804075
 2151377.86016112   91786.65624897  107003.21611866]
```

Figure 3: Weights for Stochastic Gradient Descent

```
[  17144.04261223   34820.84503211  303298.92039848 6074936.44406016
 2281391.17120344  100242.89783789  112788.96845392]
```

Figure 4: Weights for Batch Gradient Descent

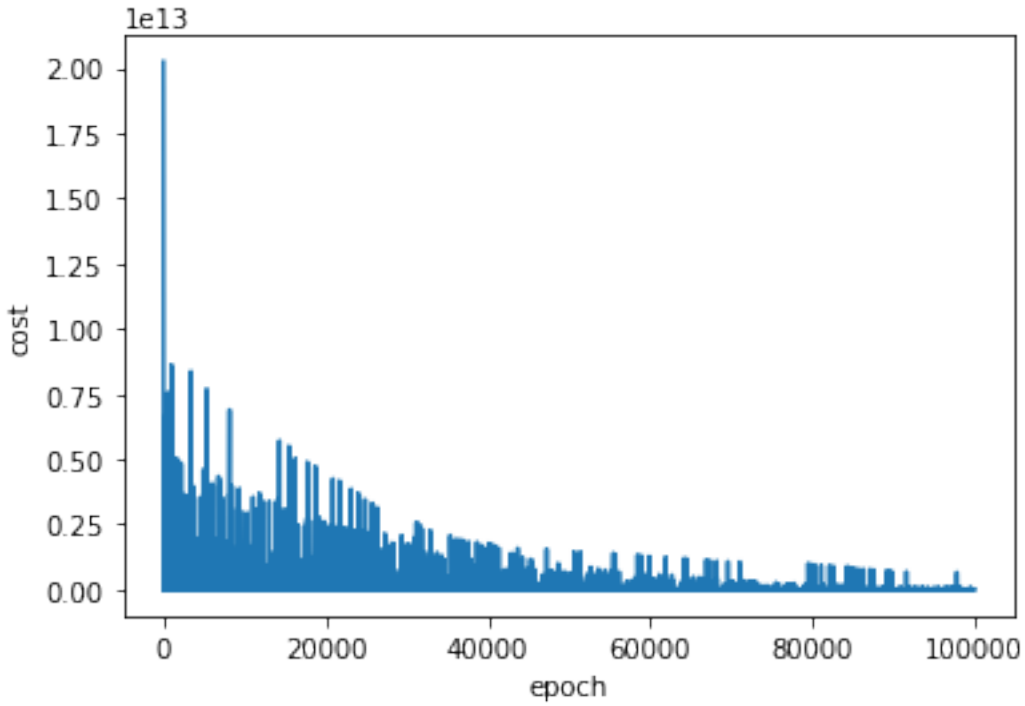


Figure 5: Cost Function plot for Stochastic Gradient Descent

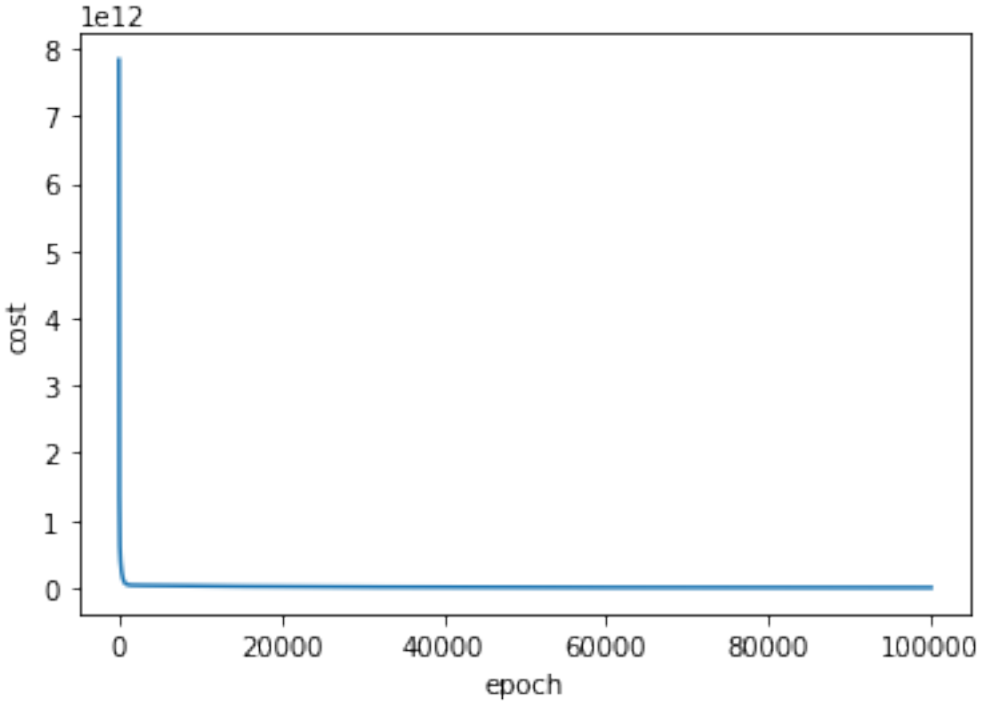


Figure 6: Cost Function plot for Batch Gradient Descent

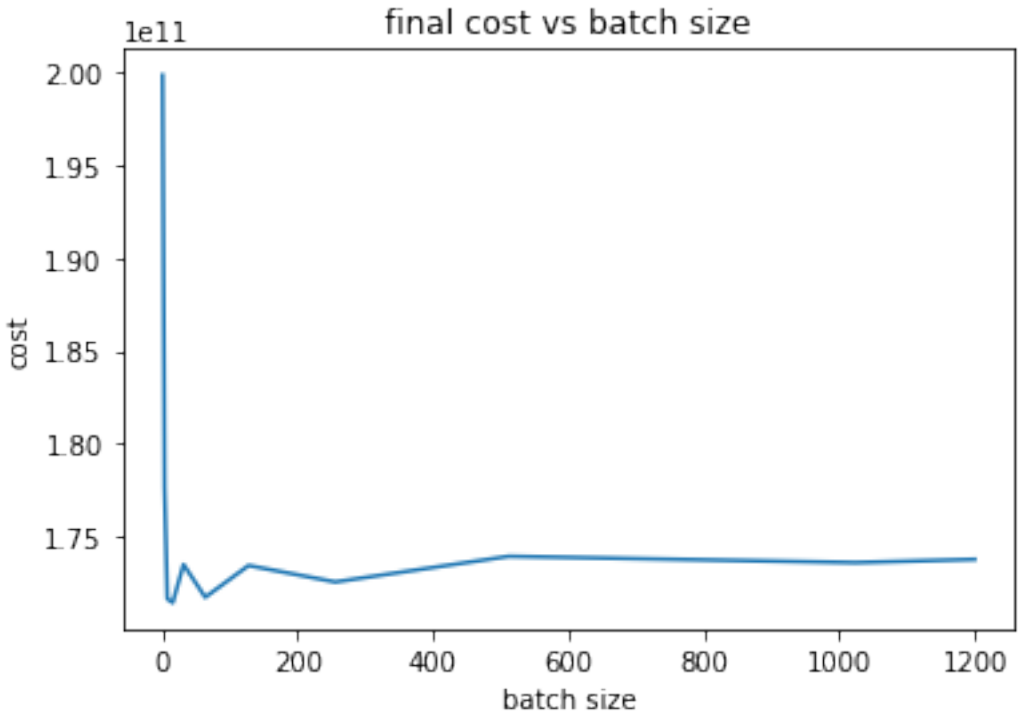


Figure 7: Cost vs Batch Size Plot for Some Batch Sizes

We observe that the plot of final cost vs batch size attains a peak. This can be explained

from the fact that we update parameters on the basis of each dataset point. As we increase the batch size cost increases, since now more number of points are being considered for updating values as compared to SGD. This happens because batch gradient descent takes into account the effect of all the dataset points and then updates the parameters. Thus, both the factors leads to a peak in the cost graph.

1.4 Effect of Regularization

Regularization is a common technique used to prevent over fitting in a model and here we achieved it using Lasso and Ridge regressions.

1.4.1 Lasso Regression

Lasso regression is used not only for reducing over fitting but also for feature selection because this technique can lead to features being neglected for prediction. We have the extra penalty factor λ multiplied with the sum of magnitudes of the weights.

$$\tilde{E}(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}(y(x_n, \mathbf{w}) - t_n)^2 + \lambda \sum_{j=0}^{M}|w_j|$$

The value of $\lambda = 500$
and the mean loss is 2.55.

```
array([-168187.0483427 ,  150505.4374521 ,  320137.12914442,
       6035121.88486337, 2623663.66567461,  434601.98508278,
        287169.25106022])
```

Figure 8: Weights for Lasso Regression

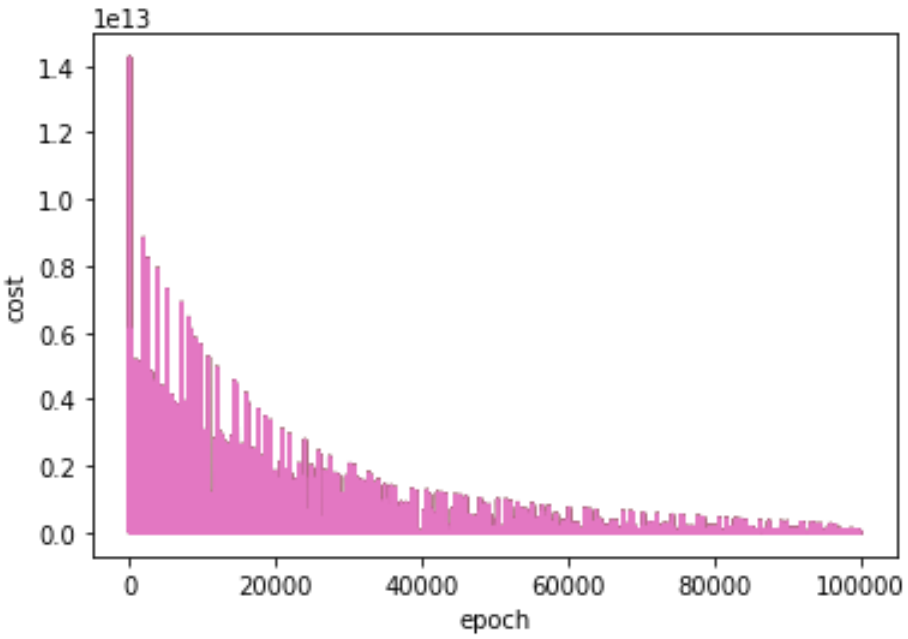


Figure 9: Cost function plot for Lasso Regression

1.4.2 Ridge Regression

In ridge regression just as in lasso regression the weights are being altered by adding penalty term, but here we multiply it with the sum of squares of the weights to avoid over fitting. Consequently, here we have a weights dependent term in the gradient of parameters. Higher the value of penalty term, the reduced are the chances of over fitting, but consequently there is a slight decrease in the accuracy.

$$\tilde{E}(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}(y(x_n, \mathbf{w}) - t_n)^2 + \lambda \sum_{j=0}^{M} w_j^2$$

The value of $\lambda = 0.001$
and the mean loss is 211.82.

```
array([ 736313.09202346, 1530664.91001822, 1574968.70135949,
       3156118.04044134,  756566.85372727,  146554.28639668,
       -743039.11312349])
```

Figure 10: Weights for Ridge Regression

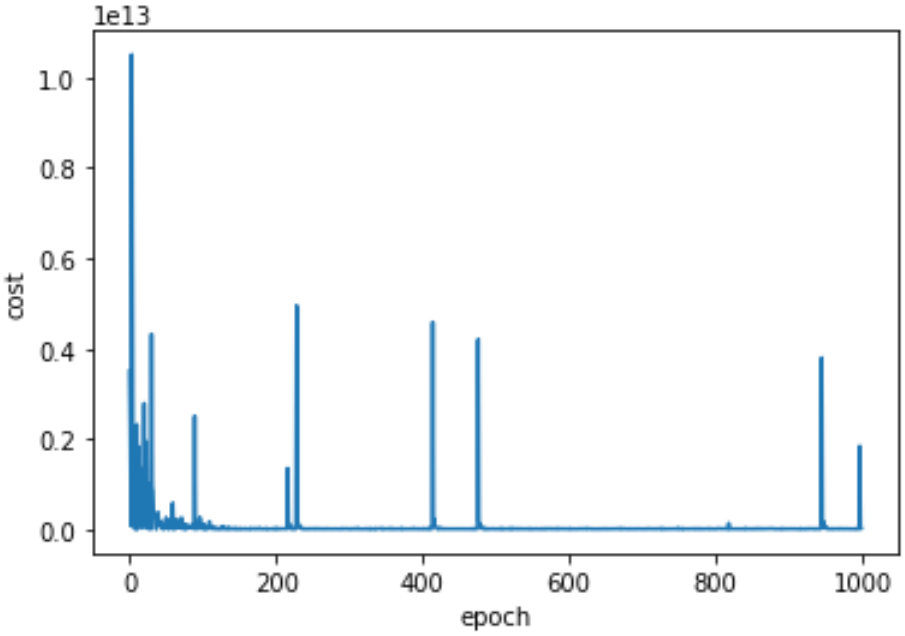


Figure 11: Cost function plot for Ridge Regression

answer the question

1.5 Feature Engineering

Approach : In this problem, after experimenting with different cost functions, we could see that the cost function:

$$\tilde{E}(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}(y(x_n, \mathbf{w}) - t_n)^2 + \lambda \sum_{j=0}^{M}|w_j|^{\frac{3}{2}}$$

This gave an error = 10.871168178263233 which was better than previous models.

```
[-2.39430564e+03  4.30435358e+03  2.93370296e+05  6.14384122e+06
  2.21924732e+06  7.74582404e+04  1.32861863e+05]
```

Figure 12: Weights for Improved Regularization

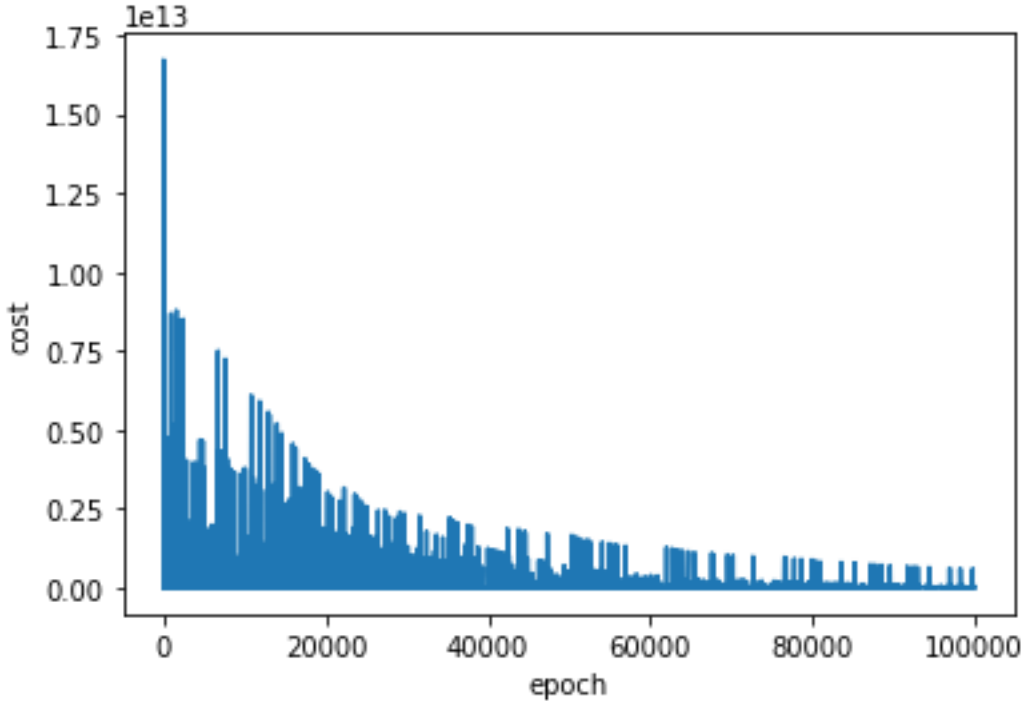


Figure 13: Cost Plot

1.6 Effect on Sample Distribution

We can observe that as the batch size increases the model trains in lesser number of epochs and the error seems to decrease as the batch size increases.This can be seen from the last graph we plotted between loss and batch size.

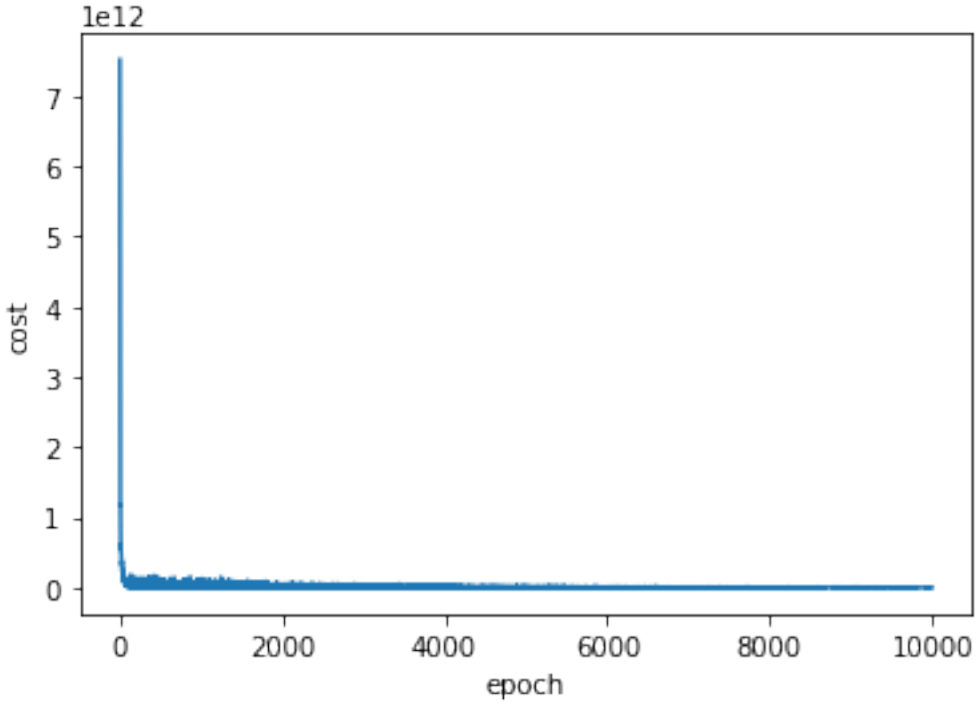


Figure 14: Cost Function for 10% Data Set

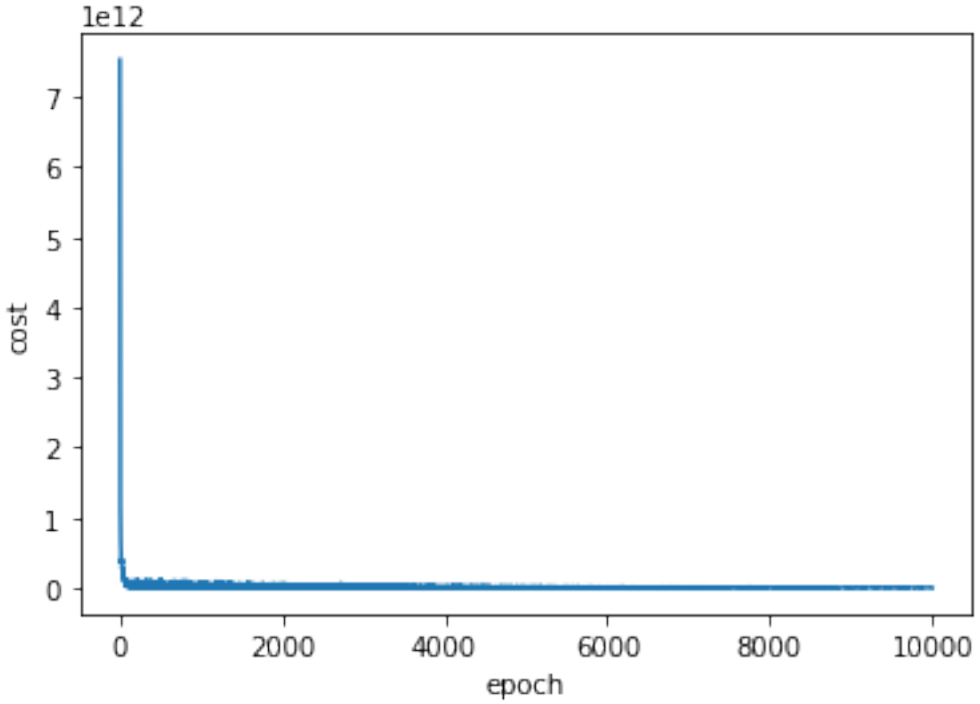


Figure 15: Cost Function for 20% Data Set

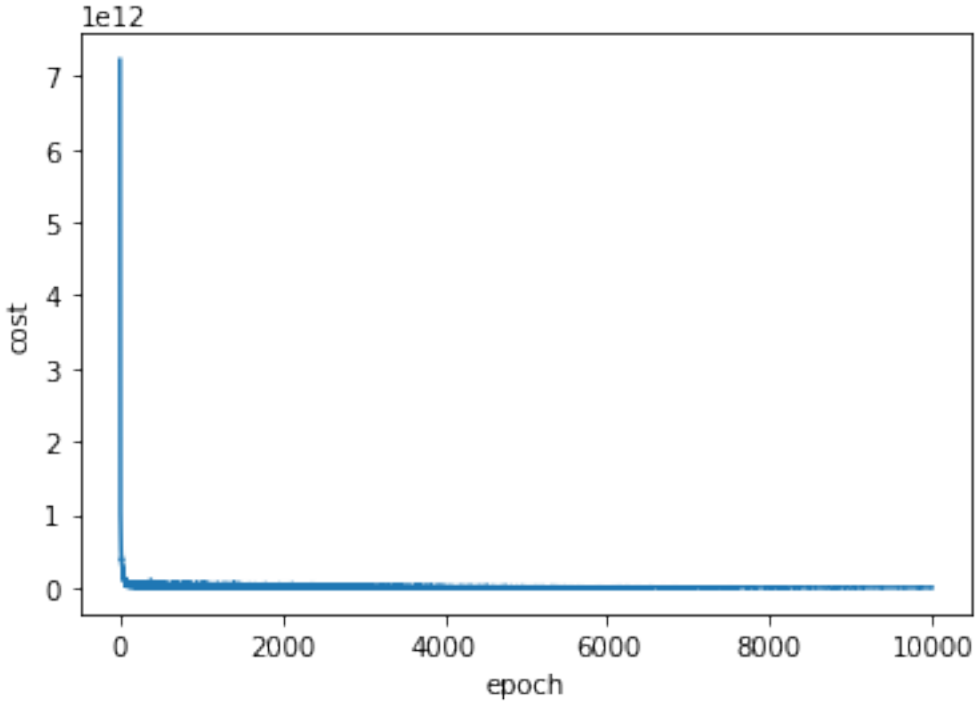


Figure 16: Cost Function for 30% Data Set

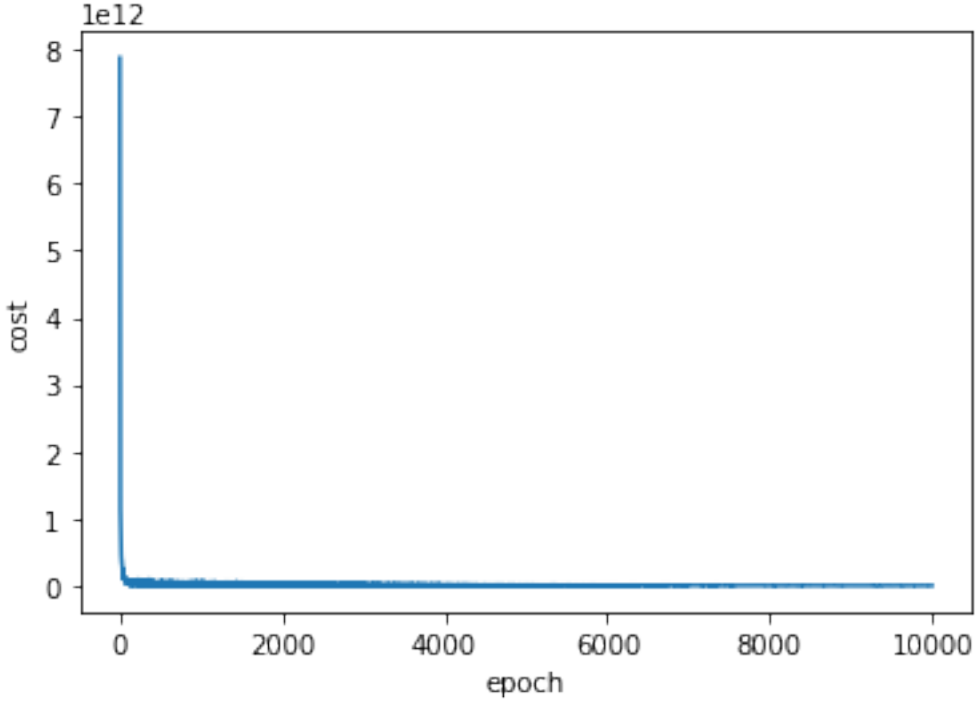


Figure 17: Cost Function for 40% Data Set

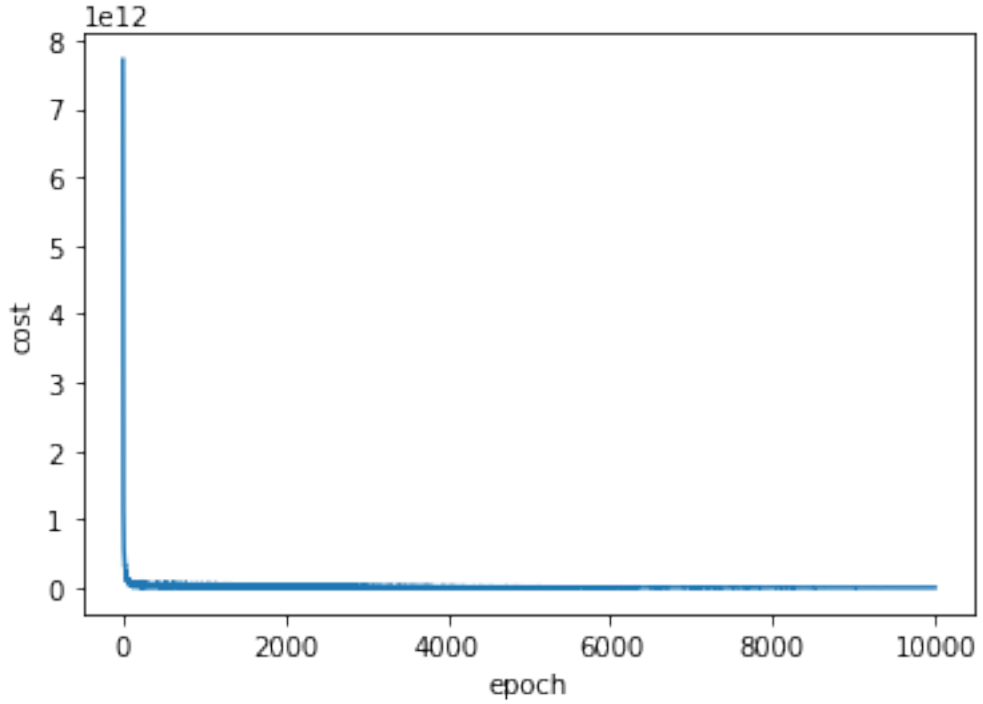


Figure 18: Cost Function for 50% Data Set

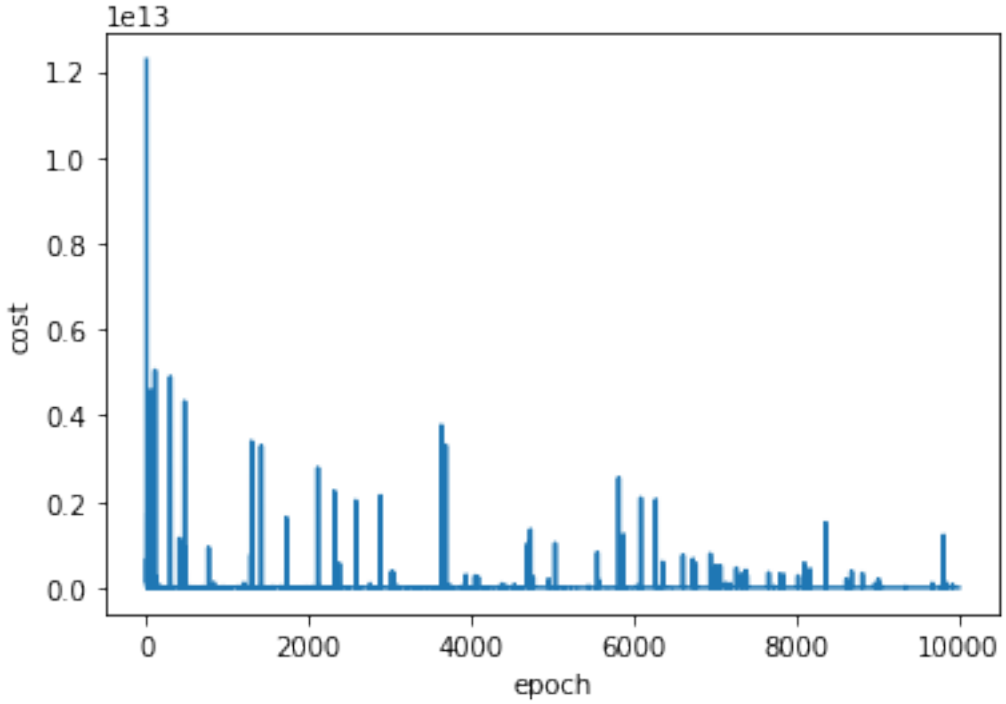


Figure 19: Cost Function for Complete Data Set

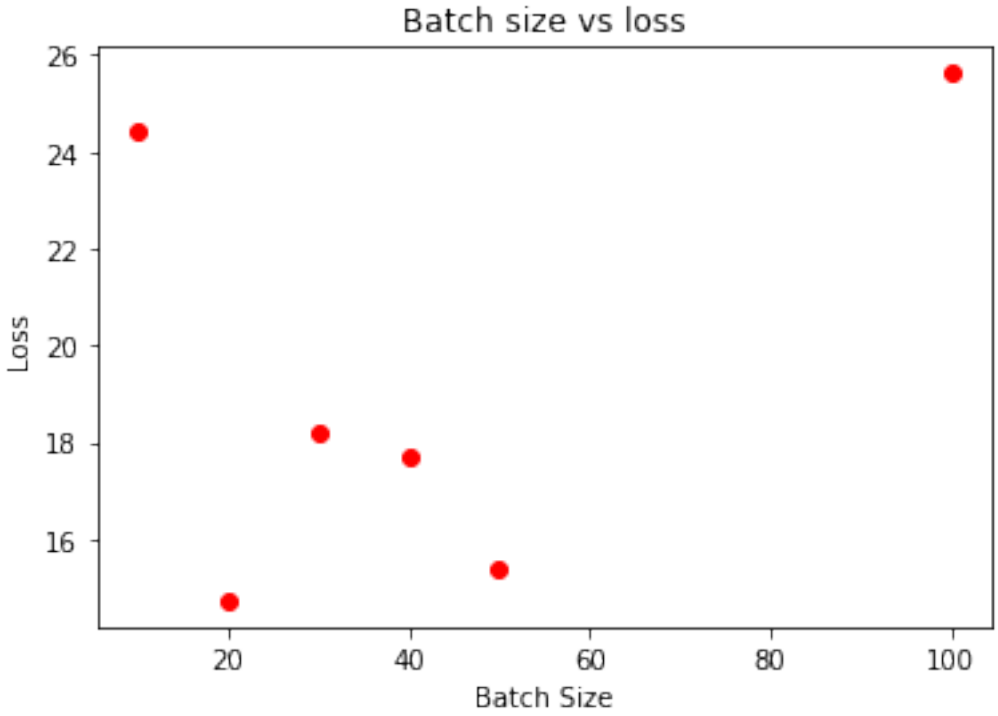


Figure 20: Batch size vs Loss Plot

1.7 Variance in Noise

The variance in the noise is 5.6046. Here the noise is the error that we are getting in predicting the values. So considering noise as error, we get

$$\text{Variance of noise} = \text{E}[(error)^2] \text{ - } (E[error])^2$$

1.8 Optimal Weights

Approach : We saw that Lasso Regression gave the least error among all the models we made. So training the data set on Lasso Regression model with learning rate = 0.01 and penality = 0.1, we get the following weights :

```
array([  15981.72548346,   25732.04420205,  335711.19529584,
       6061795.85999801, 2267760.62380787,  110379.0539289 ,
        109436.98739551])
```

Figure 21: Weights for the applied model

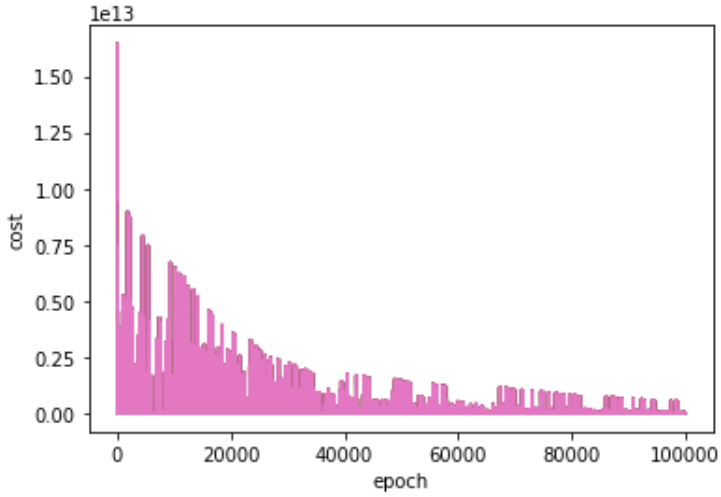


Figure 22: Cost Plot

2 Multivariate Logistic Regression

2.1 Introduction

The difference between regression and logistic classification is that later involves prediction output to be a finite set of values, here being classification of images as either 2 or 9. The cost function for classification is the below mentioned logistic loss. Also activation function is used to further separate the output into two categories. Apart from it, the basic model structure remains the same as regression, but here the efficiency of model can be estimated by comparing number of correct predictions.

$$\tilde{E}(\mathbf{w}) = \frac{-1}{N} \sum (y_n log \hat{y_n} + (1 - y_n) log(1 - \hat{y_n}))$$

The loss function

where $y_n = \frac{1}{1+e^{-z}}$, $\hat{y_n}$ is the predicted value and the initial term in the error is the cost function

2.2 Logistic Regression using Stochastic Gradient Descent

Using the loss function provided, we get the following results for the classification algorithm using Stochastic Gradient Descent:

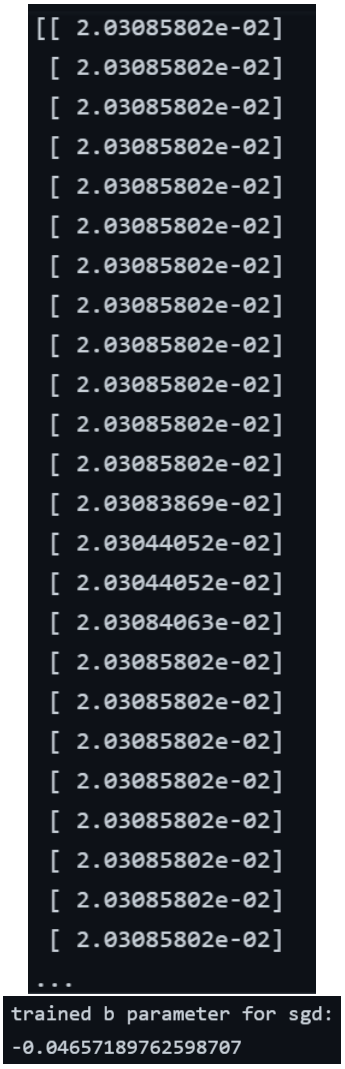


Figure 23: Weights in SGD applied in the given data set

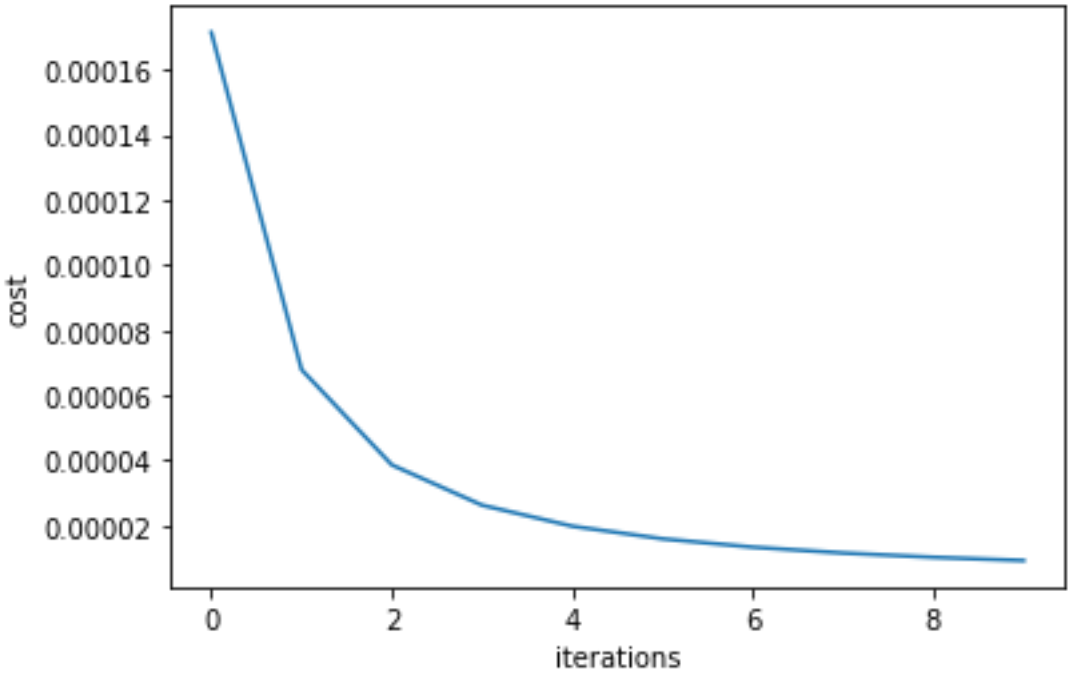


Figure 24: Plot for Cost in case of SGD

```
train accuracy: 99.32808398950131 %
test accuracy: 98.7825356842989 %
```

Figure 25: Training and Testing Data Set Accuracy for SGD

2.3 Logistic Regression using Batch Gradient Descent

Using the loss function provided, we get the following results for the regression algorithm using Batch Gradient Descent:

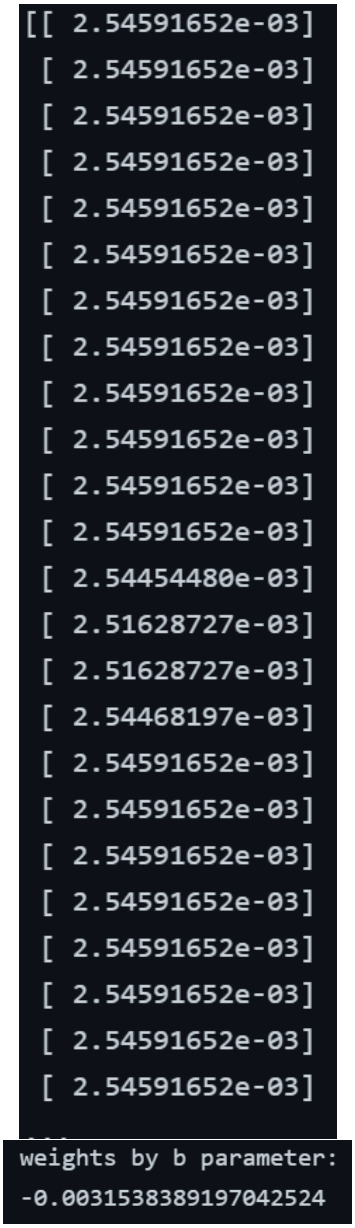


Figure 26: Weights in BGD applied in the given data set

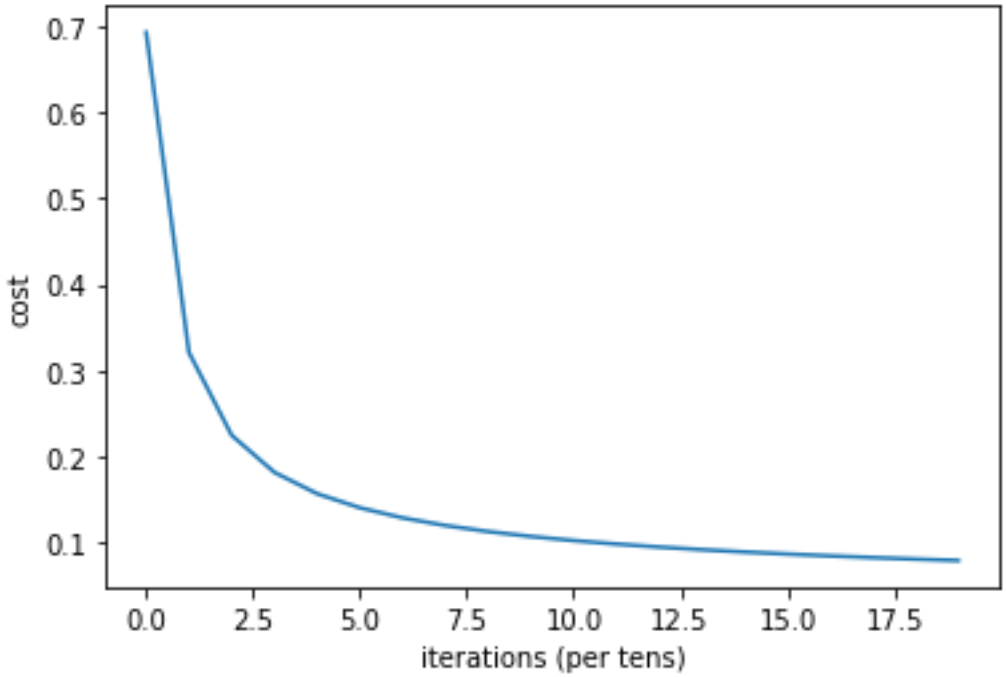


Figure 27: Plot for Cost in case of BGD

```
train accuracy: 97.91076115485565 %
test accuracy: 97.81696053736356 %
```

Figure 28: Training and Testing Data Set Accuracy for BGD

2.4 Effect of batch size

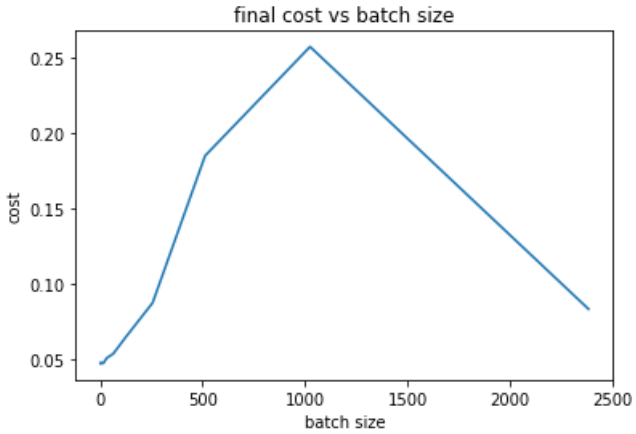


Figure 29: Cost vs Batch Size Plot for SGD

Conclusion : We observe that the plot of final cost vs batch size attains a peak. This can be explained from the fact that we attained maximum accuracy in the case of stochastic gradient descent i.e. 98.78%, where we update parameters on the basis of each dataset point. As we increase the batch size cost increases, since now more number of points are being considered for updating values as compared to SGD. But then we attain a peak and finally attain an accuracy of 97.8% in case of batch gradient descent. This happens because batch gradient descent takes into account the effect of all the dataset points and then updates the parameters. Thus, both the factors leads to a peak in the cost graph.

2.5 Effect of Regularization

Regularization is a common technique used to prevent over fitting in a model and we will achieve the same using Lasso and Ridge regularization in case of Logistic Regression just as we did in case of Linear Regression in the previous section.

2.5.1 Lasso Regression

Lasso regression is used not only for reducing over fitting but also for feature selection because this technique can lead to features being neglected for prediction. We have the extra penalty factor multiplied with the sum of magnitudes of the weights in the cost function that leads to regularization of parameters. Consequently, we have another dependent term in the parameters gradients.

$$\tilde{E}(\mathbf{w}) = \frac{-1}{N}\sum(y_n log\hat{y_n} + (1-y_n)log(1-\hat{y_n})) + \lambda\sum_{j=0}^{M}|w_j|$$

where $y_n = \frac{1}{1+e^{-z}}$, $\hat{y_n}$ is the predicted value and the initial term in the error is the cost function

The value of λ =0.1

```
train accuracy: 97.61679790026247 %
test accuracy: 97.60705289672543 %
```

Figure 30: Training and Testing data set accuracy for Lasso Regression

```
[[ 2.10110676e-03]
 [ 2.10110676e-03]
 [ 2.10110676e-03]
 [ 2.10110676e-03]
 [ 2.10110676e-03]
 [ 2.10110676e-03]
 [ 2.10110676e-03]
 [ 2.10110676e-03]
 [ 2.10110676e-03]
 [ 2.10110676e-03]
 [ 2.10110676e-03]
 [ 2.10110676e-03]
 [ 2.09998851e-03]
 [ 2.07695255e-03]
 [ 2.07695255e-03]
 [ 2.10010034e-03]
 [ 2.10110676e-03]
 [ 2.10110676e-03]
 [ 2.10110676e-03]
 [ 2.10110676e-03]
 [ 2.10110676e-03]
 [ 2.10110676e-03]
 [ 2.10110676e-03]
 [ 2.10110676e-03]
...
trained b parameter for lasso regularization:
-0.0025092572860381385
```

Figure 31: Weights for Lasso Regression

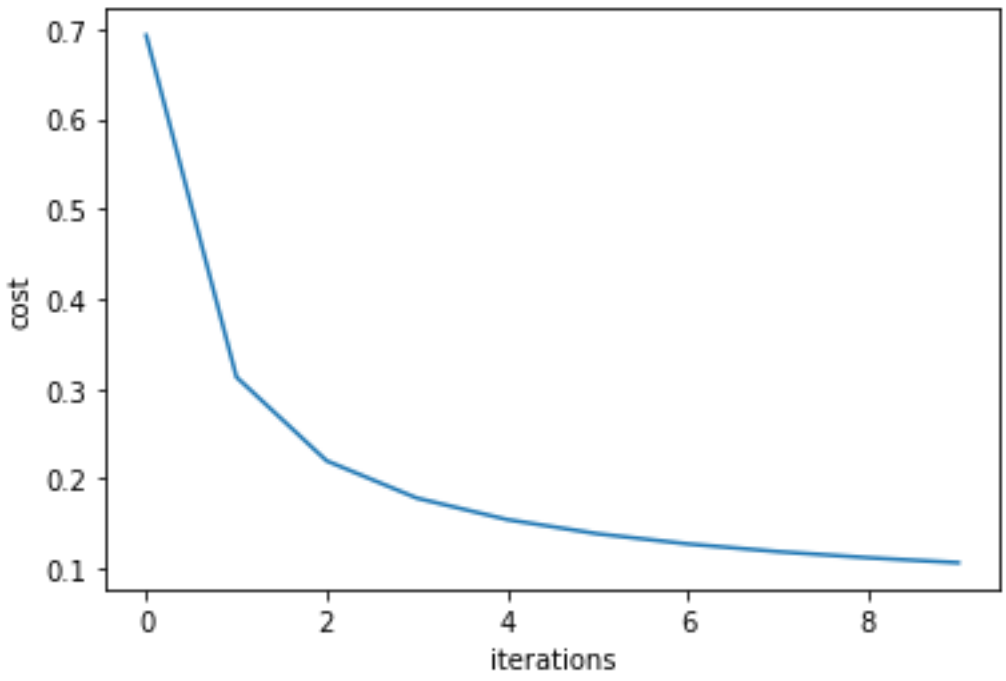


Figure 32: Cost Function plot for Lasso Regression

2.5.2 Ridge Regression

In ridge regression just as in lasso regression the weights are being altered by adding penalty term, but here we multiply it with the sum of squares of the weights to avoid over fitting. Consequently, here we have a weights dependent term in the gradient of parameters. Higher the value of penalty term, the reduced are the chances of over fitting, but consequently there is a slight decrease in the accuracy.

$$\tilde{E}(\mathbf{w}) = \frac{-1}{N}\sum(y_n log\hat{y_n} + (1-y_n)log(1-\hat{y_n})) + +\lambda\sum_{j=0}^{M} w_j^2$$

where $y_n = \frac{1}{1+e^{-z}}$, $\hat{y_n}$ is the predicted value and the initial term in the error is the cost function

The value of λ =0.1

```
train accuracy: 98.36220472440945 %
test accuracy: 98.36272040302266 %
```

Figure 33: Training and Testing data set accuracy for Ridge Regression

```
[[ 9.41806019e-03]
 [ 9.41806019e-03]
 [ 9.41806019e-03]
 [ 9.41806019e-03]
 [ 9.41806019e-03]
 [ 9.41806019e-03]
 [ 9.41806019e-03]
 [ 9.41806019e-03]
 [ 9.41806019e-03]
 [ 9.41806019e-03]
 [ 9.41806019e-03]
 [ 9.41806019e-03]
 [ 9.41447592e-03]
 [ 9.34063990e-03]
 [ 9.34063990e-03]
 [ 9.41483435e-03]
 [ 9.41806019e-03]
 [ 9.41806019e-03]
 [ 9.41806019e-03]
 [ 9.41806019e-03]
 [ 9.41806019e-03]
 [ 9.41806019e-03]
 [ 9.41806019e-03]
 [ 9.41806019e-03]
...
trained b parameter for ridge regularization:
-0.016788201761795747
```

Figure 34: Weights for Ridge Regression

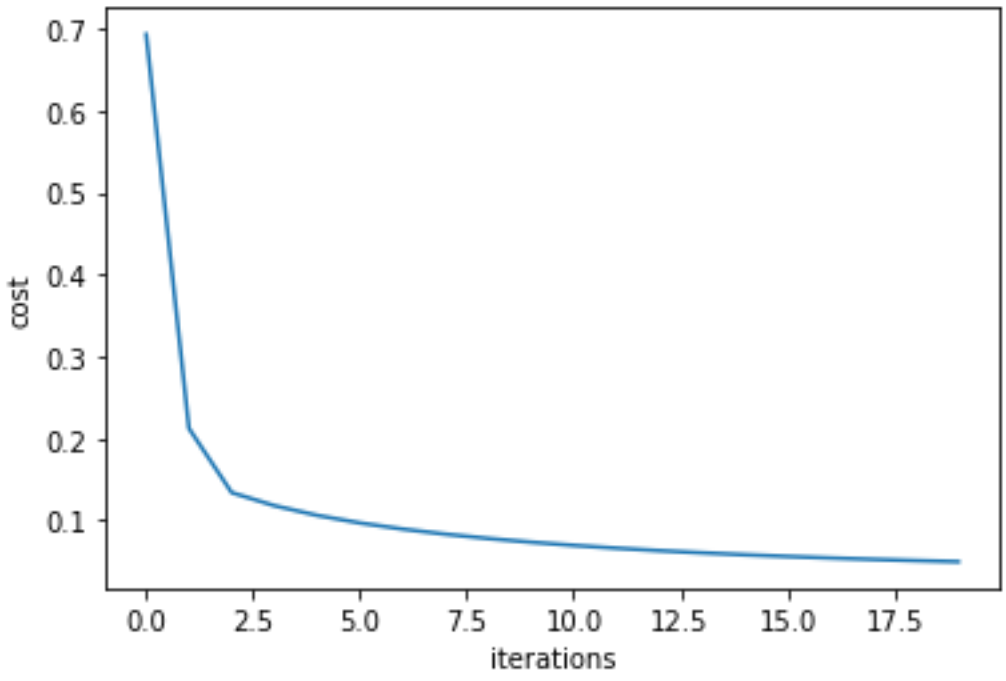


Figure 35: Cost Function plot for Ridge Regression

Conclusion : On the basis of values attained by weights parameters in case of lasso and ridge regularization,it can observed that the average numerical values of weights is higher (< 1) in case of ridge. This can be explained from the fact that the higher accuracy attained in case of ridge was supported by lower values of weights squared as compared to higher weights values in case of lasso.

2.6 Visualizations

The highest accuracy that we could obtain was in the case of stochastic gradient descent

```
train accuracy: 99.10761154855643 %
test accuracy: 98.7825356842989 %
```

Figure 36: Highest accuracy for training and test data set.

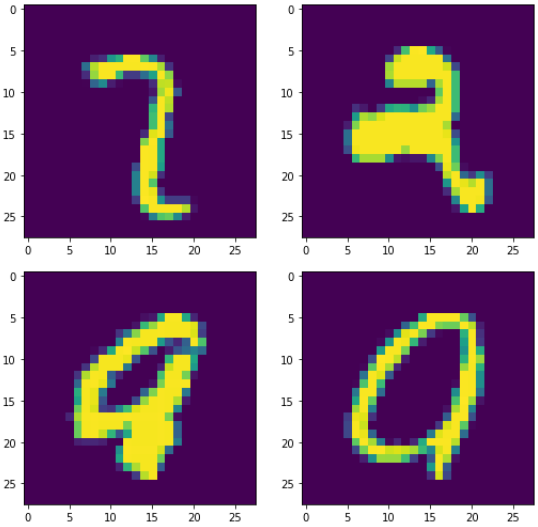
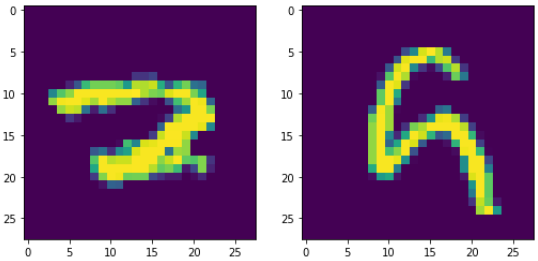


Figure 37: Data which was wrongly predicted in the training data



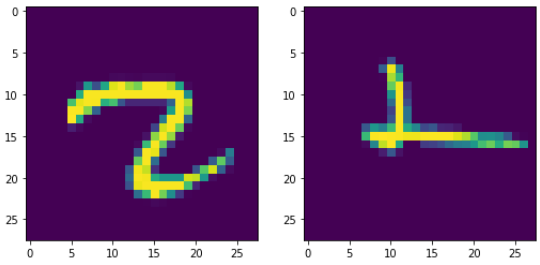


Figure 38: Data which was wrongly predicted in the test data

First set of images are the wrongly classified images in the training dataset and the second set of images are wrongly classified in test dataset. It can be inferred from these images that some of the numbers are so badly written that it might be confusing to infer then even through the eye. Other were the cases where the shape of written letter was very much similar to the complement number, thus model classified it wrongly. Some of the cases where the model wrongly classified could be where the given image didn't represent any of the number precisely. Thus, model might have attained any midway value thus, leading to an almost close prediction.

2.7 Optimal Weights

We saw in the first part of the question that the accuracy was very high using stochastic gradient descent.

Accuracy for stochastic gradient descent

```
d_sgd = model_sgd(X_train, Y_train, X_test,Y_test, 1,10, learning_rate = 0.005)
```

```
train accuracy: 99.32808398950131 %
test accuracy: 98.7825356842989 %
```

Figure 39: Accuracy for the first part of the question

So we decided to customise our model by increasing the number iterations as well as the learning rate. Increasing learning rate reduces the chance of under fitting and increasing number of iterations ensured that the weights were more closer to the minima.

```
d_sgd = model_sgd(X_train, Y_train, X_test,Y_test, 1,15, learning_rate = 0.01)
✓ 4.9s
train accuracy: 99.48556430446195 %
test accuracy: 99.03442485306465 %
```

Figure 40: Model with different parameters and increased accuracy

```
[[ 3.46596012e-02]
 [ 3.46596012e-02]
 [ 3.46596012e-02]
 [ 3.46596012e-02]
 [ 3.46596012e-02]
 [ 3.46596012e-02]
 [ 3.46596012e-02]
 [ 3.46596012e-02]
 [ 3.46596012e-02]
 [ 3.46596012e-02]
 [ 3.46596012e-02]
 [ 3.46592188e-02]
 [ 3.46513415e-02]
 [ 3.46513415e-02]
 [ 3.46592571e-02]
 [ 3.46596012e-02]
 [ 3.46596012e-02]
 [ 3.46596012e-02]
 [ 3.46596012e-02]
 [ 3.46596012e-02]
 [ 3.46596012e-02]
 [ 3.46596012e-02]
 [ 3.46596012e-02]
...
```

Figure 41: Weights for the new model with increased learning rate and number of iterations

3 BONUS Question

Approach : Initially, the concept of one hot encoding was used to convert the target data into the one hot matrix. The model trained remained the same, with the slight difference in the dimension of weights matrix, with it being corresponding to the classification of 10 digits. Also, finally one hot encoding was again converted into label encoding for better intuition and accuracy calculation process.
The decrease in accuracy in case of 10 digit classification as compared to binary classification can be attributed to the fact of higher dimension weight parameters. Also, the model took more time to attain the minima because of higher number of examples in dataset and higher dimensionality of parameters.

```
train accuracy: 87.55%
test accuracy: 87.25%
```

Figure 42: Accuracy in training and test data set

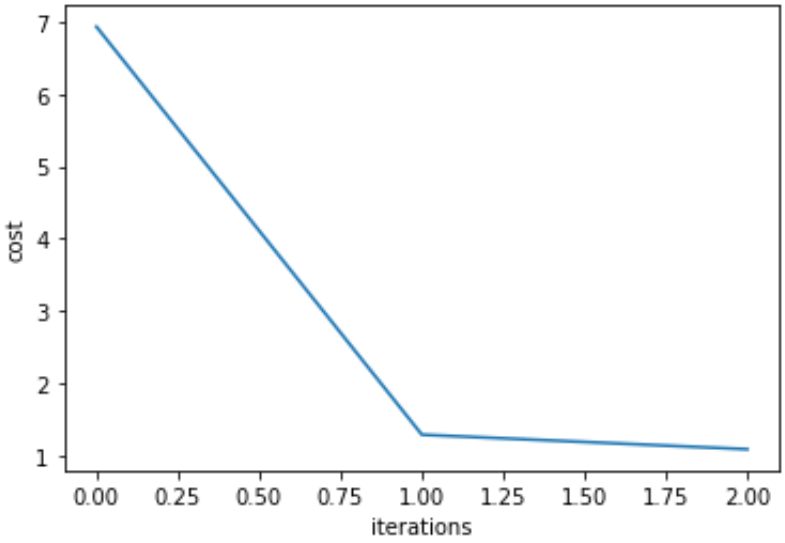


Figure 43: Cost function plot

$ConfusionMatrix$:Confusion matrix is another way of determining the accuracy of the model. The diagonal elements of the matrix determines the accurate positive or negative examples, whereas the non diagonal elements represent false positive or true negative predictions. A well trained model has higher value of diagonal elements as compared to the

non diagonal ones. The confusion matrix output in case of our model represent the same for all the 10 digit target values with it being in the from 0 to 9. It can be seen the diagonal elements are significantly higher than the non diagonal ones.

```
[[[10679   112]
  [   47  1162]]

 [[10446   147]
  [   58  1349]]

 [[10676   134]
  [  179  1011]]

 [[10548   185]
  [  204  1063]]

 [[10739   137]
  [  121  1003]]

 [[10770   143]
  [  317   770]]

 [[10703   136]
  [   72  1089]]

 [[10644   125]
  [  126  1105]]

 [[10584   228]
  [  208   980]]

 [[10681   183]
  [  198   938]]]
```

Figure 44: Confusion Matrix showing the performance of our model