Q1

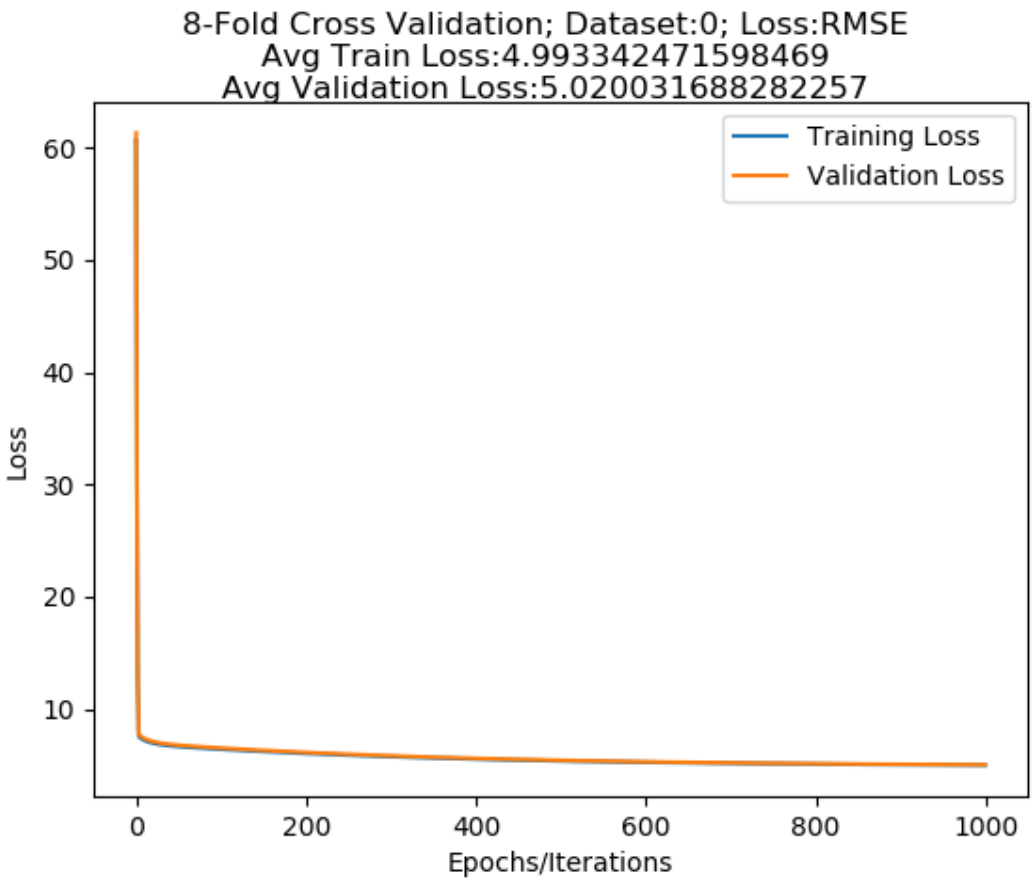
- Pre-processing Strategy:
    - **Dataset-0**: The abalone dataset has a sex field which contains values M, F or I. So, in order to make it compatible with the algorithm, I replaced it with 3 binary columns as Male (1 or 0), Female (1 or 0), and Infant (1 or 0). Other than that, there was no missing or ambiguous data entry, thus not much pre-processing was required.
    - **Dataset-1**: The video game dataset had quite a few field values missing. Since we were required to work with only three columns out of this dataset, I first extracted those columns only and deleted the rows with any value missing or set as TBD or NaN value. Due to this tbd value, one column was imported automatically as string type so it had to be converted to float data type.
- Trying K=5-10 and will select the one giving better results
    - Taking values bigger than 10 like e.g. k=20, are not reasonable from the computational perspective and will be likely to overfit.
    - Taking values smaller than 5 will increase the chances of underfitting.
    - In fact, this is the reason k=5 or 10 happen to be most widely used in the real-life scenarios.
    - K=5 will be marginally avoiding underfit, and K=10 will marginally avoid overfitting. Thus, it makes sense to choose 5<=k<=10.
    - Hence, I'll try 5<=k<=10 and choose the one that gives the minimum loss. This is done using the function called **findBestValueOfK()** defined in the file **Q1_plots.py**
- The following output is generated:

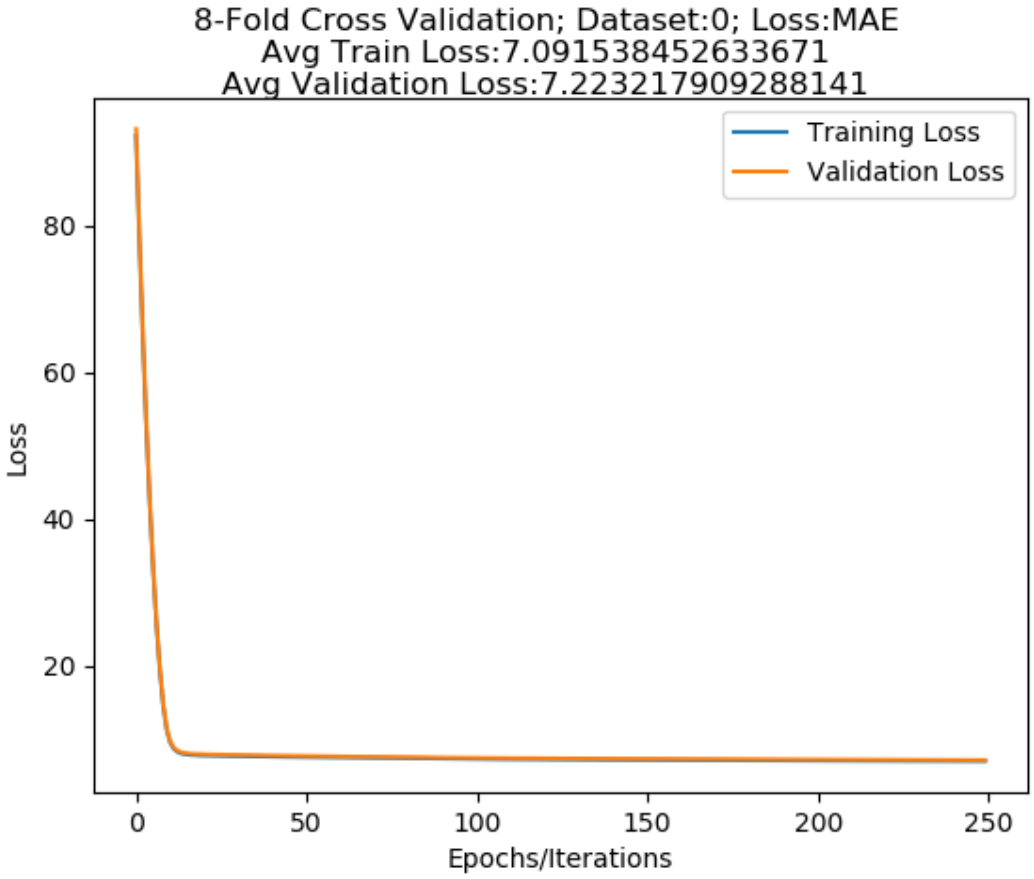
Order followed below: RMSE dataset0, MAE dataset0, RMSE dataset1, MAE dataset1
Values of K giving min loss | Calculated minimum loss for that K
Training: [8, 8, 8, 8] [4.993342471598469, 7.091538452633671, 2.873011858305148, 3.164295829079452]
Validation: [6, 6, 8, 8] [5.015898504358333, 7.1963450179239885, 3.619123535317868, 3.8957629170997303]

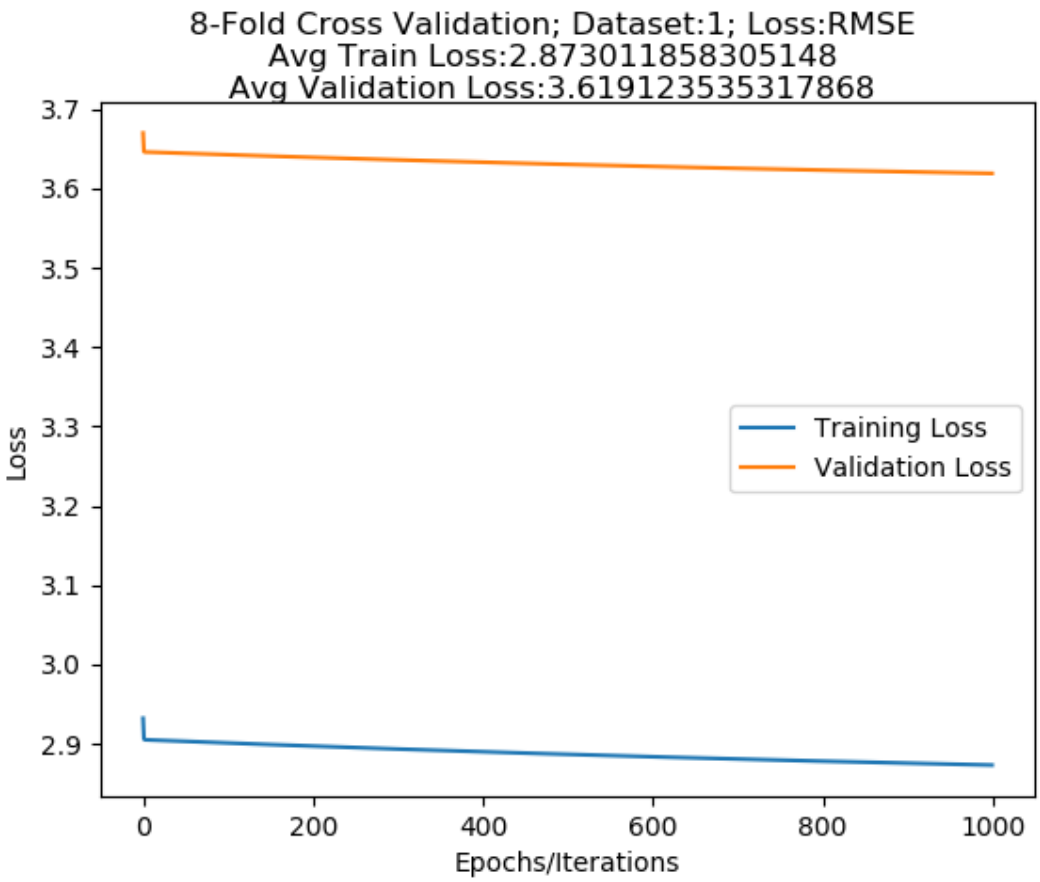
WHY I SELECTED K=8?

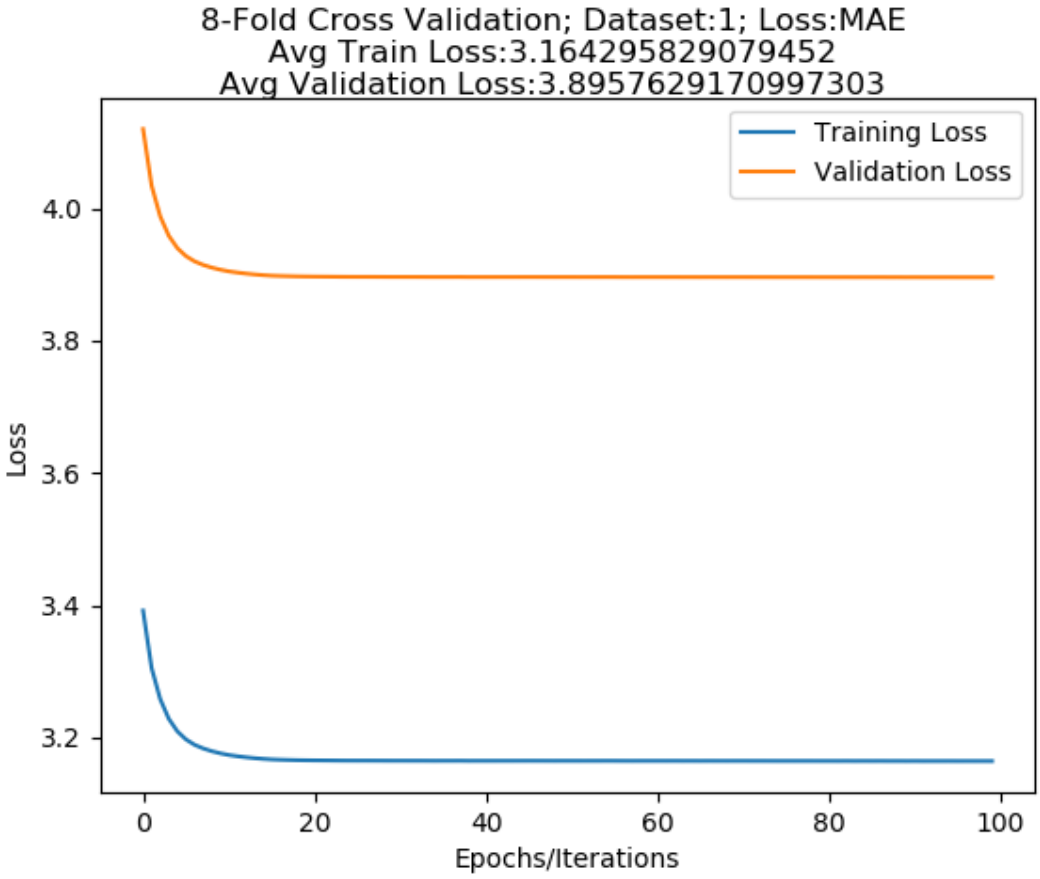
Following plots show MSE Loss (Performance) Vs Epochs for K=8

- It is evident from the above results that choosing **K=8** could be a good idea since it is giving the best performance in most of the cases. Hence, I select K=8 for the plots.
- The plots below can be regenerated by running **Q1_plots.py** provided in the project folder with **to_plot="MSE"** (inside MyLinearRegression.fit method definition).
- The following plots use RMSE/MAE as loss function but MSE loss for performance as well as the plot.
- ***Kindly note that these are performance curves for K=8 and not the answer to Q1(a) and (b)***

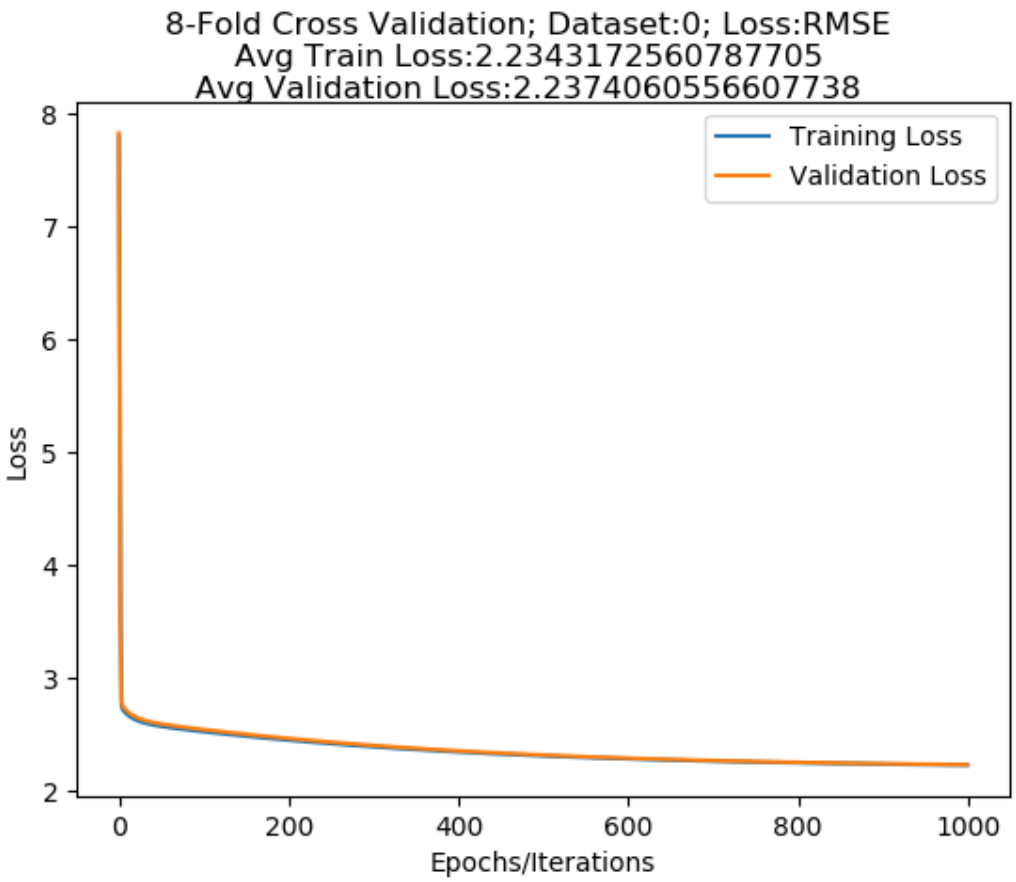

8-Fold Cross Validation; Dataset:0; Loss:RMSE
Avg Train Loss:4.993342471598469
Avg Validation Loss:5.020031688282257
Training Loss
Validation Loss
Loss
Epochs/Iterations
60
50
40
30
20
10
0
200
400
600
800
1000

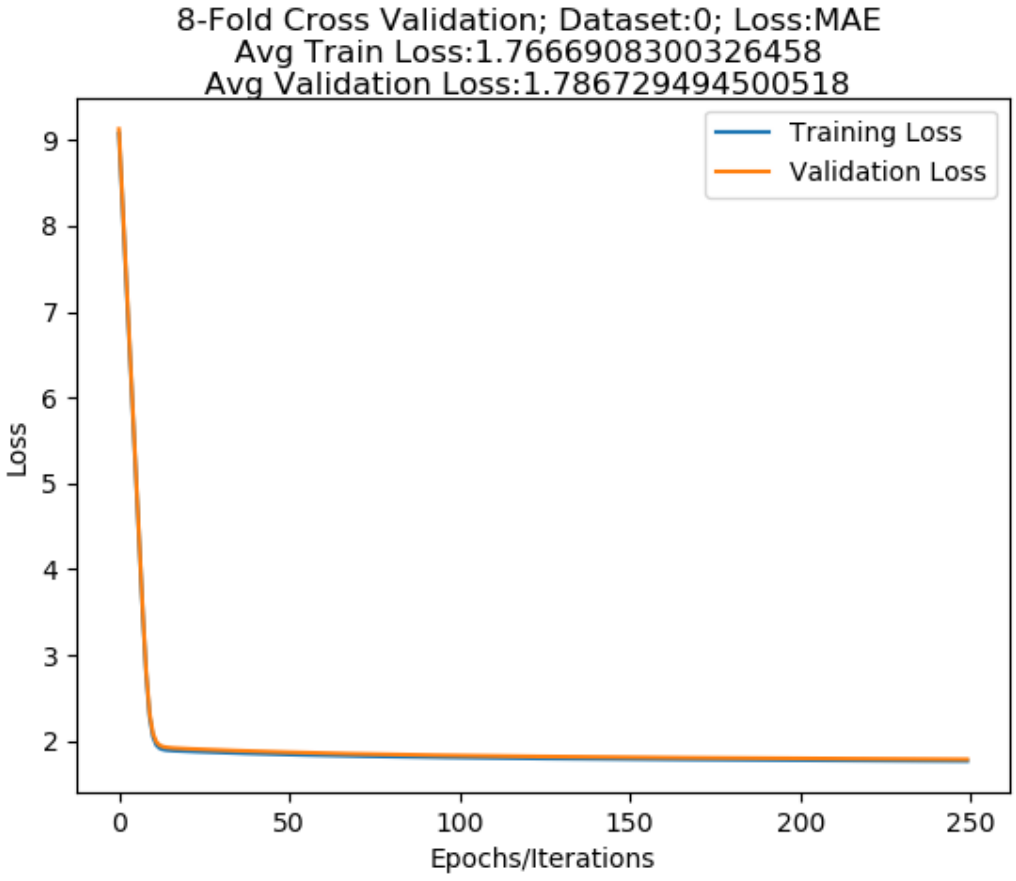

8-Fold Cross Validation; Dataset:0; Loss:MAE
Avg Train Loss:7.091538452633671
Avg Validation Loss:7.223217909288141
Training Loss
Validation Loss
Loss
Epochs/Iterations
80
60
40
20
0
50
100
150
200
250

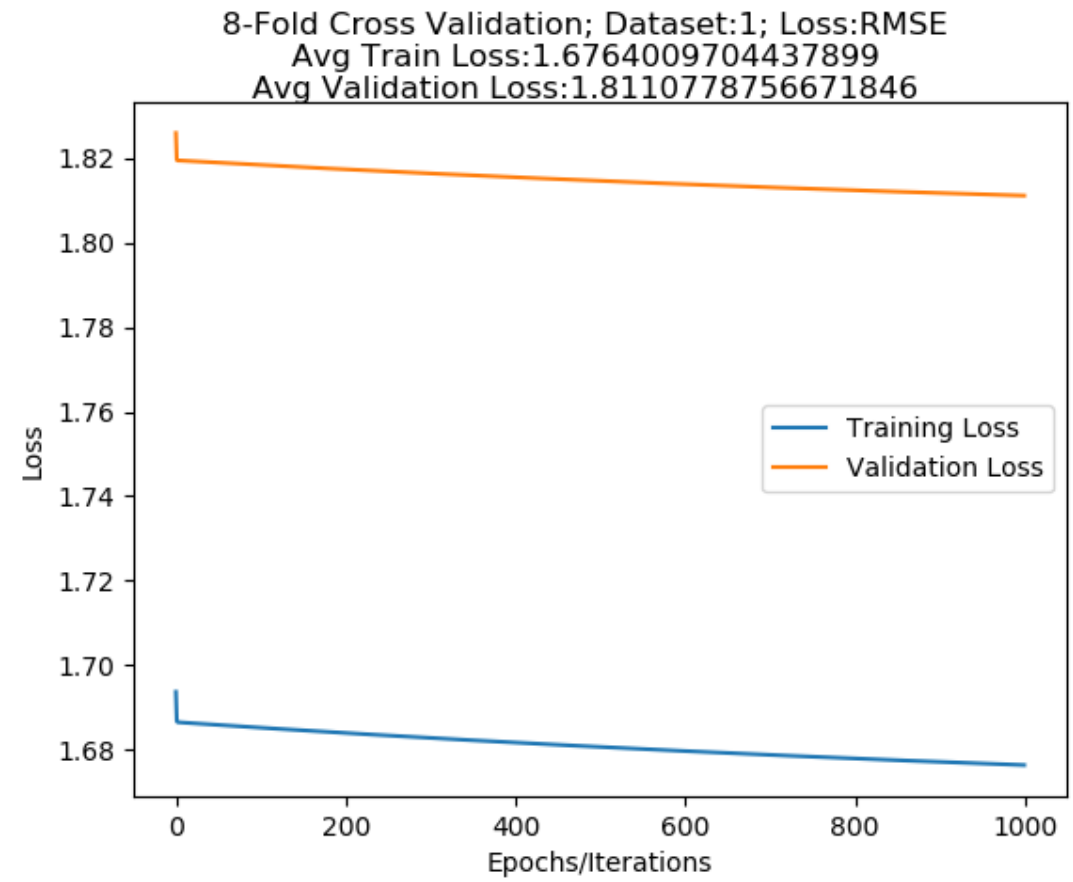

8-Fold Cross Validation; Dataset:1; Loss:RMSE
Avg Train Loss:2.873011858305148
Avg Validation Loss:3.619123535317868
Training Loss
Validation Loss
Loss
Epochs/Iterations
3.7
3.6
3.5
3.4
3.3
3.2
3.1
3.0
2.9
0
200
400
600
800
1000

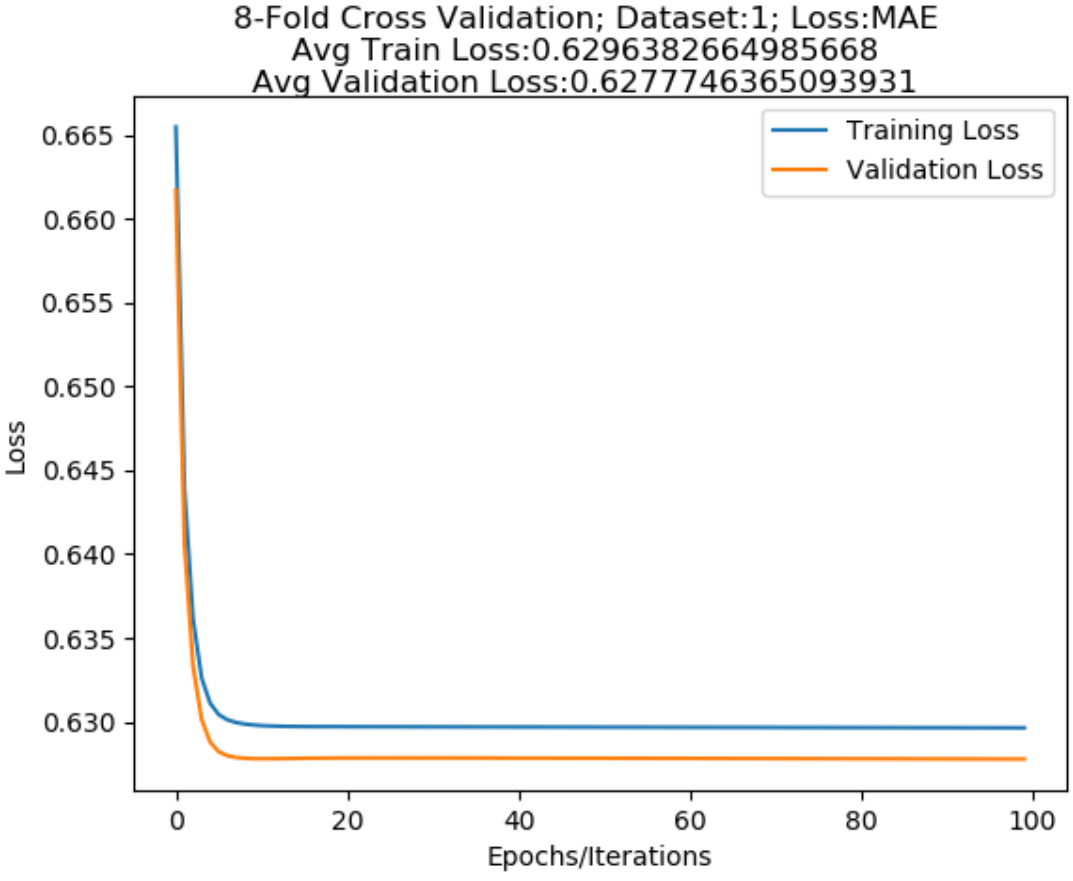

8-Fold Cross Validation; Dataset:1; Loss:MAE
Avg Train Loss:3.164295829079452
Avg Validation Loss:3.895762917099730 3
Training Loss
Validation Loss
Loss
Epochs/Iterations
4.0
3.8
3.6
3.4
3.2
0
20
40
60
80
100

(a) Training Loss Vs Epochs and Validation Loss Vs Epochs Curves (8-Fold)
(b) Best RMSE/MAE values achieved can be seen within the plots.


8-Fold Cross Validation; Dataset:0; Loss:RMSE
Avg Train Loss:2.2343172560787705
Avg Validation Loss:2.2374060556607738
Training Loss
Validation Loss
Loss
Epochs/Iterations
0
200
400
600
800
1000
2
3
4
5
6
7
8


8-Fold Cross Validation; Dataset:0; Loss:MAE
Avg Train Loss:1.7666908300326458
Avg Validation Loss:1.786729494500518
Training Loss
Validation Loss
Loss
Epochs/Iterations
0
50
100
150
200
250
2
3
4
5
6
7
8
9


8-Fold Cross Validation; Dataset:1; Loss:RMSE
Avg Train Loss:1.6764009704437899
Avg Validation Loss:1.8110778756671846
Loss
Epochs/Iterations
Training Loss
Validation Loss
1.82
1.80
1.78
1.76
1.74
1.72
1.70
1.68
0
200
400
600
800
1000


8-Fold Cross Validation; Dataset:1; Loss:MAE
Avg Train Loss:0.6296382664985668
Avg Validation Loss:0.6277746365093931
Loss
Epochs/Iterations
Training Loss
Validation Loss
0.665
0.660
0.655
0.650
0.645
0.640
0.635
0.630
0
20
40
60
80
100

(c) Which loss gives better performance (MSE as performance metric)

- Dataset-0
  - Training Performance (MSE Loss):
    - Using RMSE loss function: 4.993342471598469
    - Using MAE loss function: 7.091538452633671
    - Winner: RMSE
  - Validation Performance (MSE Loss):
    - Using RMSE loss function: 5.015898504358333
    - Using MAE loss function: 7.1963450179239885
    - Winner: RMSE
  - Evident from above results that **RMSE performs better**, although MAE happens to converge in lower number of epochs but it gives more loss, possibly because of outliers.
- Dataset-1
  - Training Performance (MSE Loss):
    - Using RMSE loss function: 2.873011858305148
    - Using MAE loss function: 3.164295829079452
    - Winner: RMSE
  - Validation Performance (MSE Loss):
    - Using RMSE loss function: 3.619123535317868
    - Using MAE loss function: 3.8957629170997303
    - Winner: RMSE
  - Evident from above results that **RMSE performs better**, although MAE happens to converge in lower number of epochs but it gives more loss, possibly because of outliers.
- Thus, **RMSE performs better in both** the datasets.

(d) Relation between RMSE and MAE. Conditions under which they give similar values. Which loss to prefer in such case and why?

- Relation between RMSE and MAE: RMSE>=MAE
  - RMSE is always greater than MAE due to the difference in the way an error is penalized in both these loss functions. MAE on one hand, considers that an error value of 2e is twice as bad as error value of e. While on the other hand, for RMSE an error value of 2e is more than twice as bad as the error value e. While MAE is used where errors are supposed to be penalized in linear proportion to their magnitude, RMSE is used when a large error needs to be penalized even more.
- Condition under which they give similar values
  - RMSE=MAE iff all the errors are equal. Thus, if the variance of errors is very small, RMSE and MAE will give similar values and in fact exactly equal values if the variance is 0.
- Which loss to prefer in such a case and why?
  - Since both the functions give similar values, the choice has to be made by some other factors. MAE is non differentiable at 0, while RMSE is completely differentiable

in its entire domain and range. Thus, it will be easier to work with gradient descent and so, I would recommend to go with RMSE.

(e) Comparison: Normal form vs Gradient descent

Avg optimal parameters generated using normal form: (Starting with theta_0 upto theta_n)

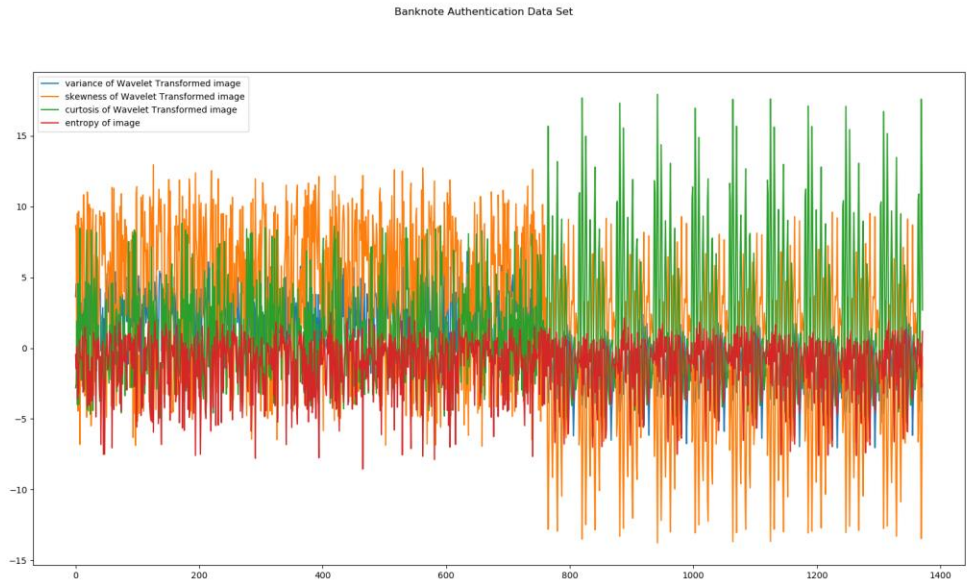
[2.60518621, 1.21255122, 1.14531168, 0.24732331, -1.32172352, 11.72514876, 14.48874417, 9.15614238, -19.46937474, -11.82536839, 7.73017629]

- 8-fold using the normal form optimal parameters on dataset-0
  - Avg training performance (MSE Loss): 4.778654712666407
  - Avg validation performance (MSE Loss): 4.936081182774652
- 8-fold using RMSE loss based gradient descent on dataset-0
  - Avg training performance (MSE Loss): 4.993342471598469
  - Avg validation performance (MSE Loss): 5.015898504358333
- Clearly Normal Form is the winner although there is a very small difference.

Q2

- Analyse class distributions and comment on the feature values

```
PS C:\Users\Dushyant Panchal\Desktop\2018033_HW1> & "C:/Users/Dushyant P
                0            1            2            3            4
count  1372.000000  1372.000000  1372.000000  1372.000000  1372.000000
mean      0.433735     1.922353     1.397627    -1.191657     0.444606
std       2.842763     5.869047     4.310030     2.101013     0.497103
min      -7.042100   -13.773100    -5.286100    -8.548200     0.000000
25%      -1.773000    -1.708200    -1.574975    -2.413450     0.000000
50%       0.496180     2.319650     0.616630    -0.586650     0.000000
75%       2.821475     6.814625     3.179250     0.394810     1.000000
max       6.824800    12.951600    17.927400     2.449500     1.000000
Labelled 0: 762
Labelled 1: 610
```



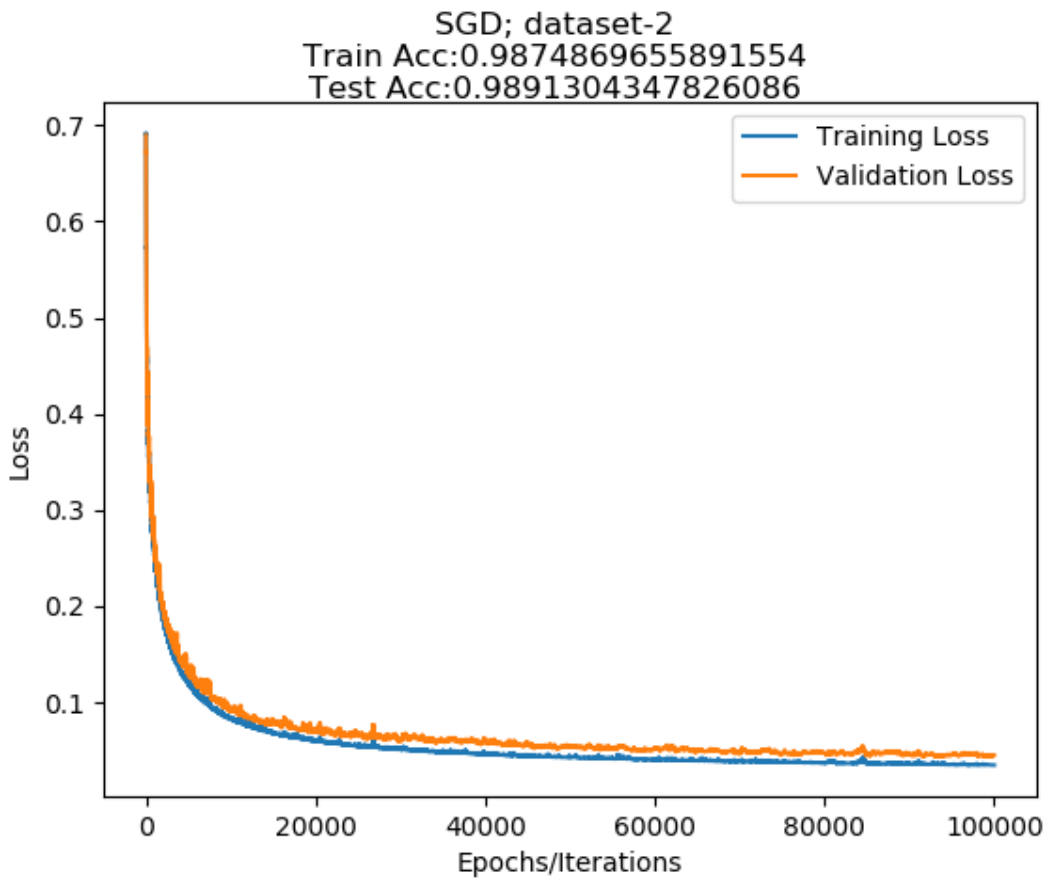
- From the above plot and the table, we notice the following:
    - There is no missing data as evident from the count for every column.
    - The dataset is very well organised in the sense that all the class 0 entries are together followed by class 1 entries altogether.
    - The class distribution of the dataset is pretty good i.e. 762:610 for class 0:class 1.
    - The data points in the initial half of the rows are pretty close in the plot while in the second half they are sparser. This could relate to the fact that first 762 rows correspond to class 0 and next 610 to class 1, thus showing a visual relationship between the features and the class.
    - The variance for attributes 1 and 2 (orange and green plots) are almost double than the other features evident from the data and visible in the plot as well.
    - Probably, attributes 1 and 2 (orange and green plots) will be the major contributors in distinguishing between class 0 and class 1 rows of the dataset.

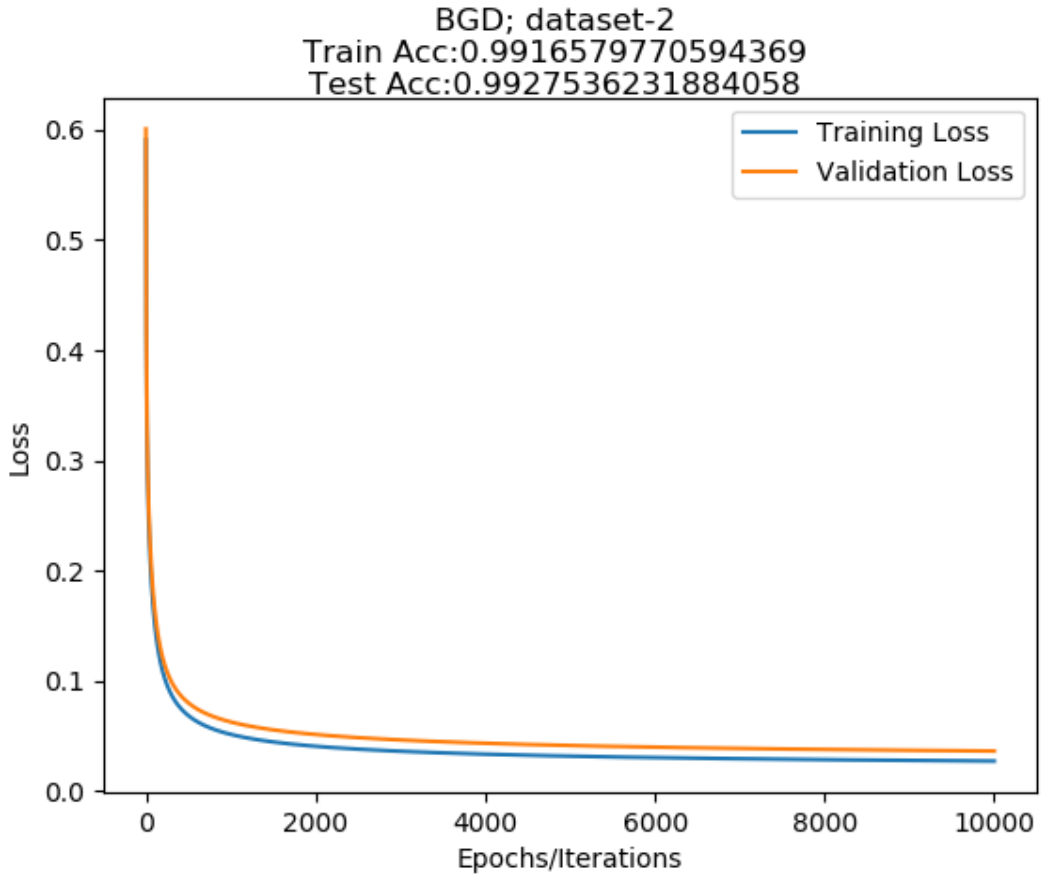
- Following table gives comparison between SGD and BGD when using appropriate hyperparameters. Output can be regenerated by running the file called **Q2_plots.py**

INFO	Stochastic GD	Batch GD
ALPHA	0.1	3
# EPOCHS	1,00,000	10,000

FINAL LOSS	Train: 0.03491175058646329 Test: 0.044790543039862664	Train: 0.027114366163377944 Test: 0.036221282175763175
ACCURACY	Train: 0.9874869655891554 Test: 0.9891304347826086	Train: 0.9916579770594369 Test: 0.9927536231884058
THETA	2.76703032e+00 -1.50399366e+01 -1.53504097e+01 -2.55420930e+01 1.96167612e-03	3.41808419 -19.82069471 -20.85730246 -35.05515438 -0.04542097

- Here are the plots: -

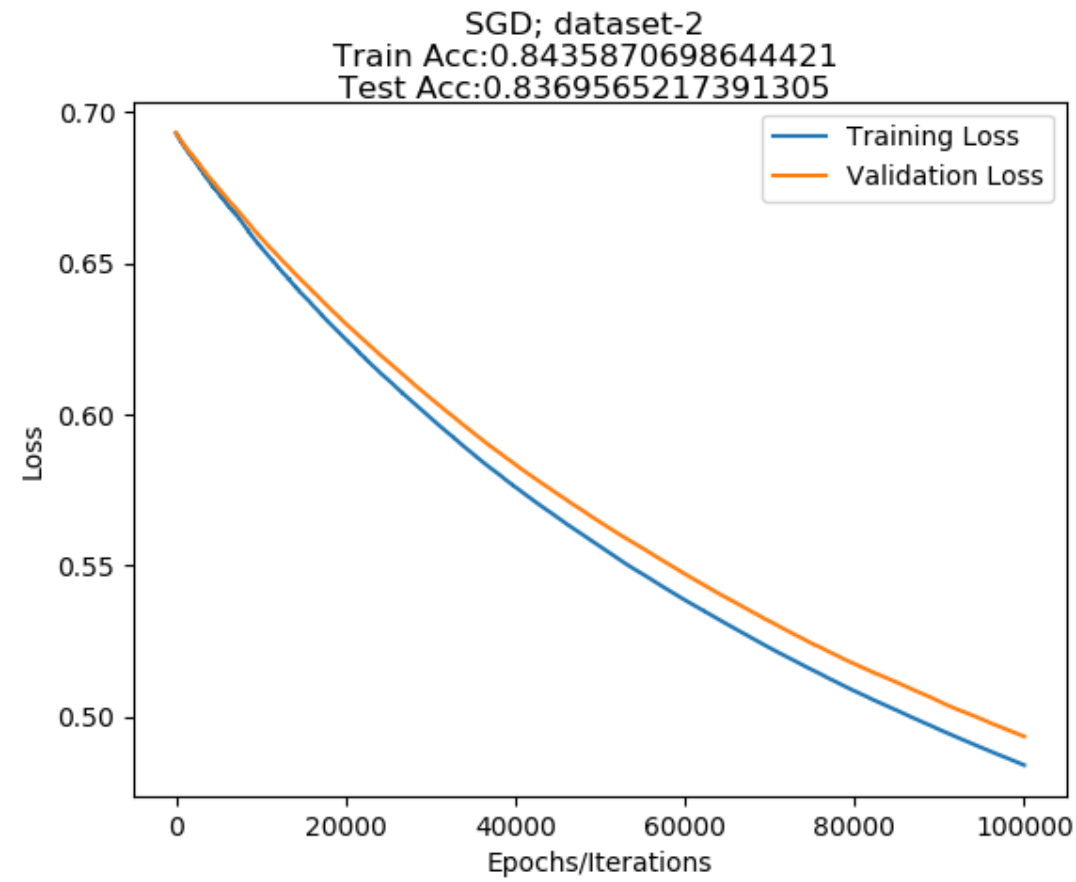

SGD; dataset-2
Train Acc:0.9874869655891554
Test Acc:0.9891304347826086
Training Loss
Validation Loss
Loss
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
20000
40000
60000
80000
100000
Epochs/Iterations

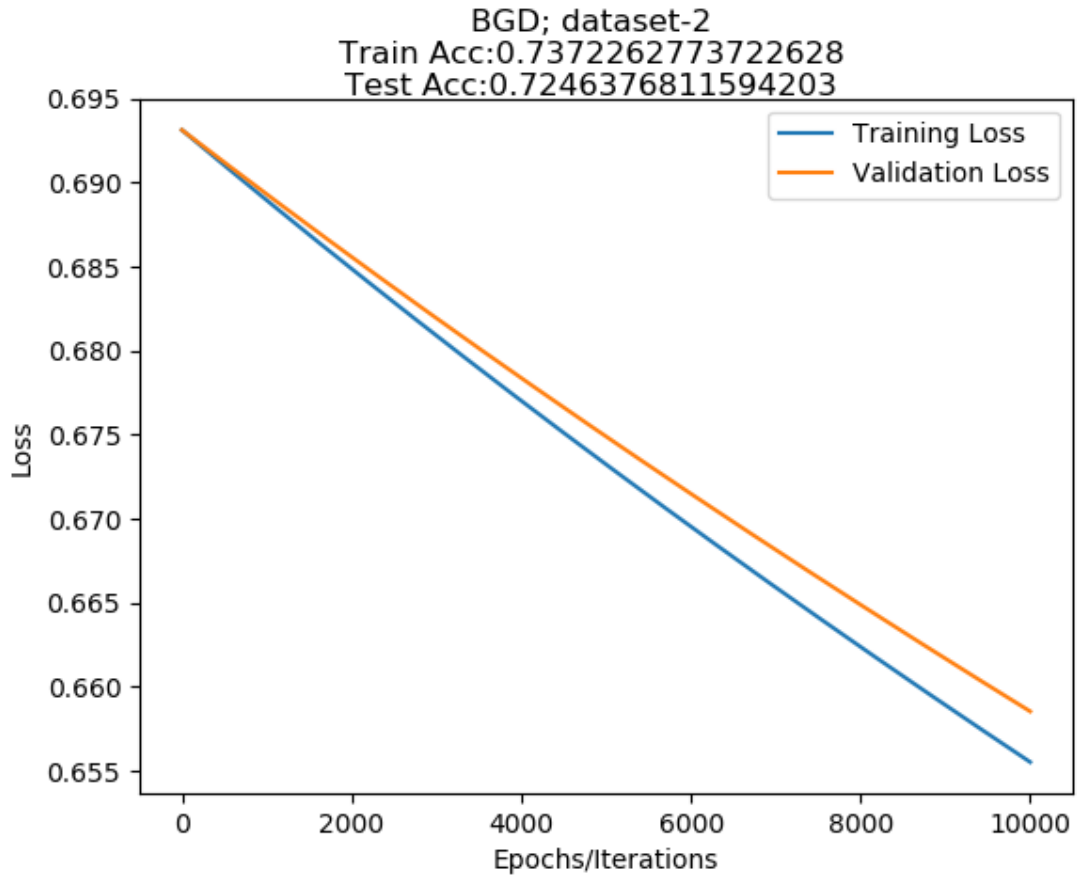

BGD; dataset-2
Train Acc:0.9916579770594369
Test Acc:0.9927536231884058
Training Loss
Validation Loss
Loss
0.6
0.5
0.4
0.3
0.2
0.1
0.0
0
2000
4000
6000
8000
10000
Epochs/Iterations

- Now trying alpha = 0.0001, 0.01, 10 keeping the #epochs same as in above table.

    - Alpha = 0.0001, Epochs = 1,00,000 (SGD) and 10,000 (BGD)

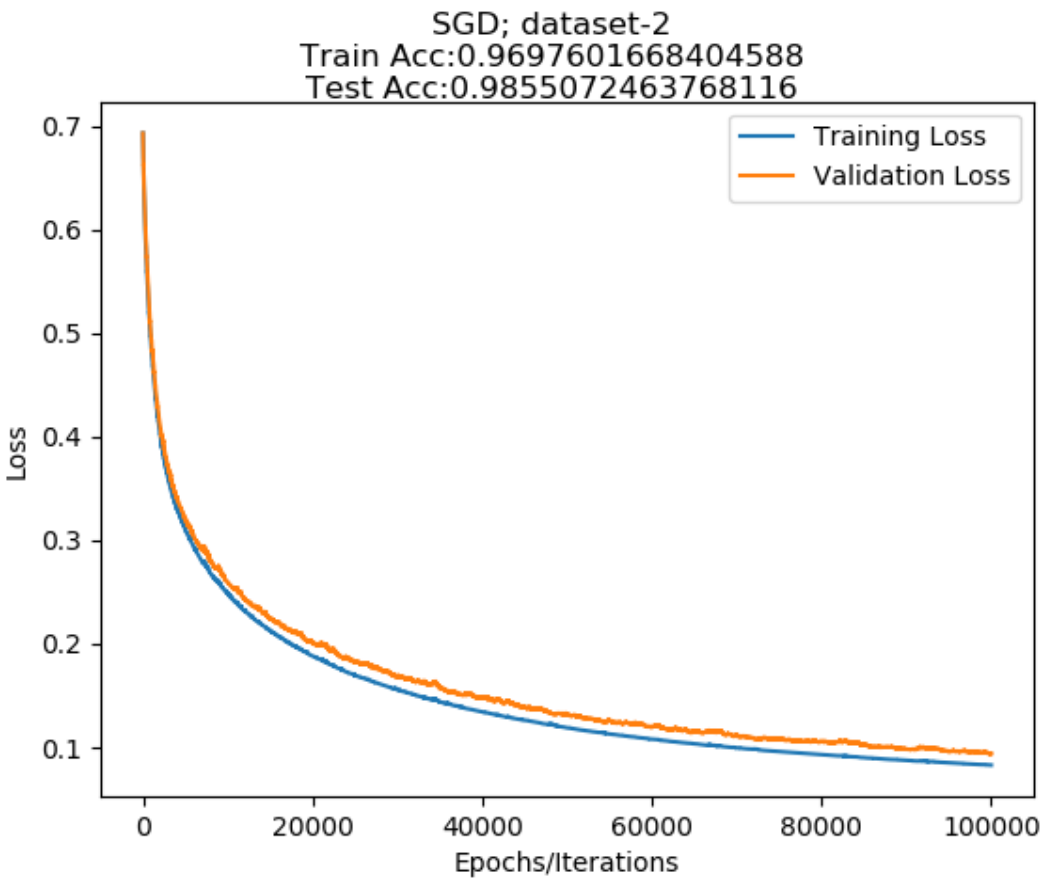
INFO	Stochastic GD	Batch GD
FINAL LOSS	Train: 0.48396411912260046 Test: 0.4934209335828096	Train: 0.6555337736747782 Test: 0.6585494118797457
ACCURACY	Train: 0.8435870698644421 Test: 0.8369565217391305	Train: 0.7372262773722628 Test: 0.7246376811594203

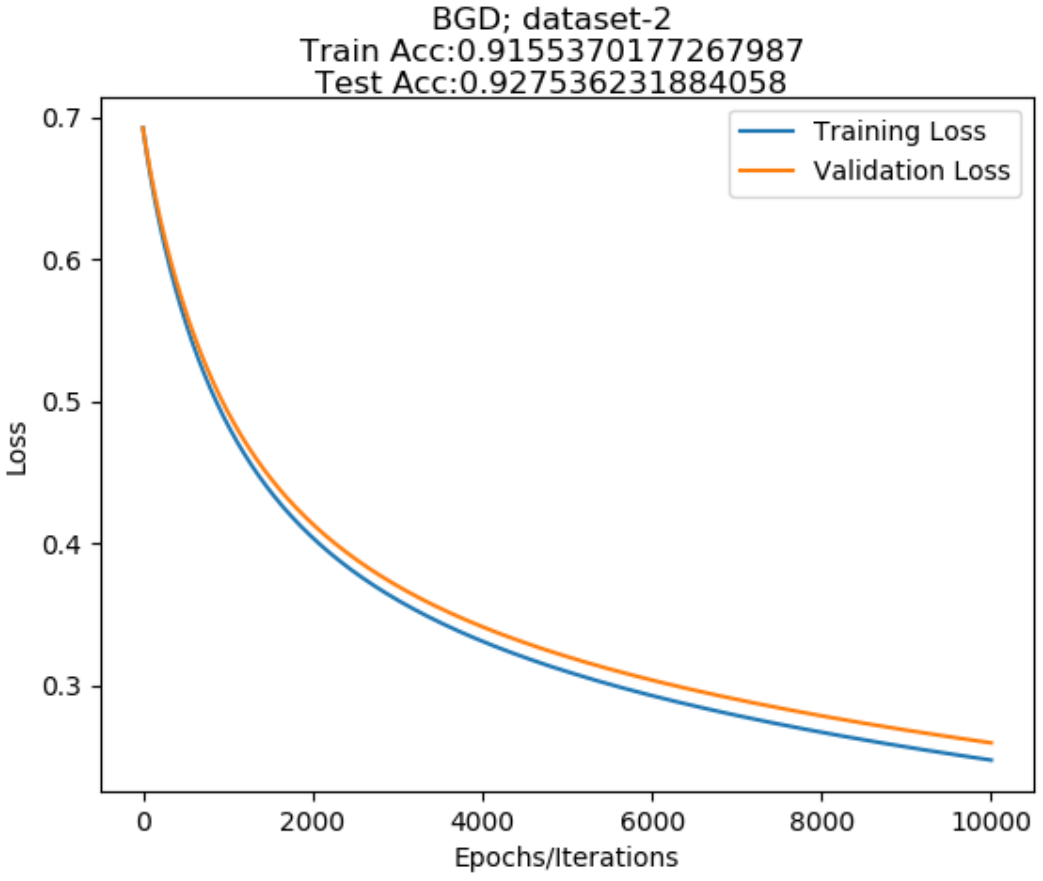

SGD; dataset-2
Train Acc:0.8435870698644421
Test Acc:0.8369565217391305
Training Loss
Validation Loss
Loss
0.70
0.65
0.60
0.55
0.50
0
20000
40000
60000
80000
100000
Epochs/Iterations


BGD; dataset-2
Train Acc:0.7372262773722628
Test Acc:0.7246376811594203
Training Loss
Validation Loss
Loss
0.695
0.690
0.685
0.680
0.675
0.670
0.665
0.660
0.655
0
2000
4000
6000
8000
10000
Epochs/Iterations

- Alpha = 0.01, Epochs = 1,00,000 (SGD) and 10,000 (BGD)

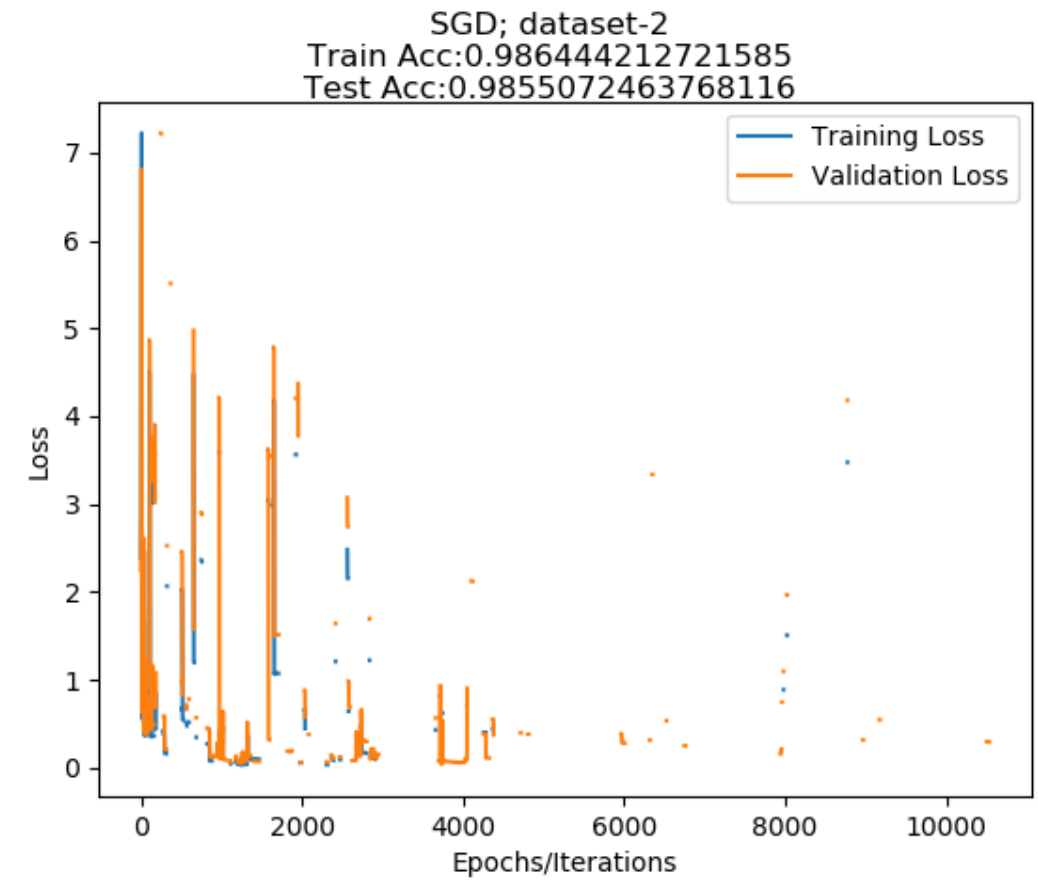
INFO	Stochastic GD	Batch GD
FINAL LOSS	Train: 0.08305080571842972 Test: 0.09402818675023615	Train: 0.24741458375157013 Test: 0.25952663165205475
ACCURACY	Train: 0.9697601668404588 Test: 0.9855072463768116	Train: 0.9155370177267987 Test: 0.927536231884058

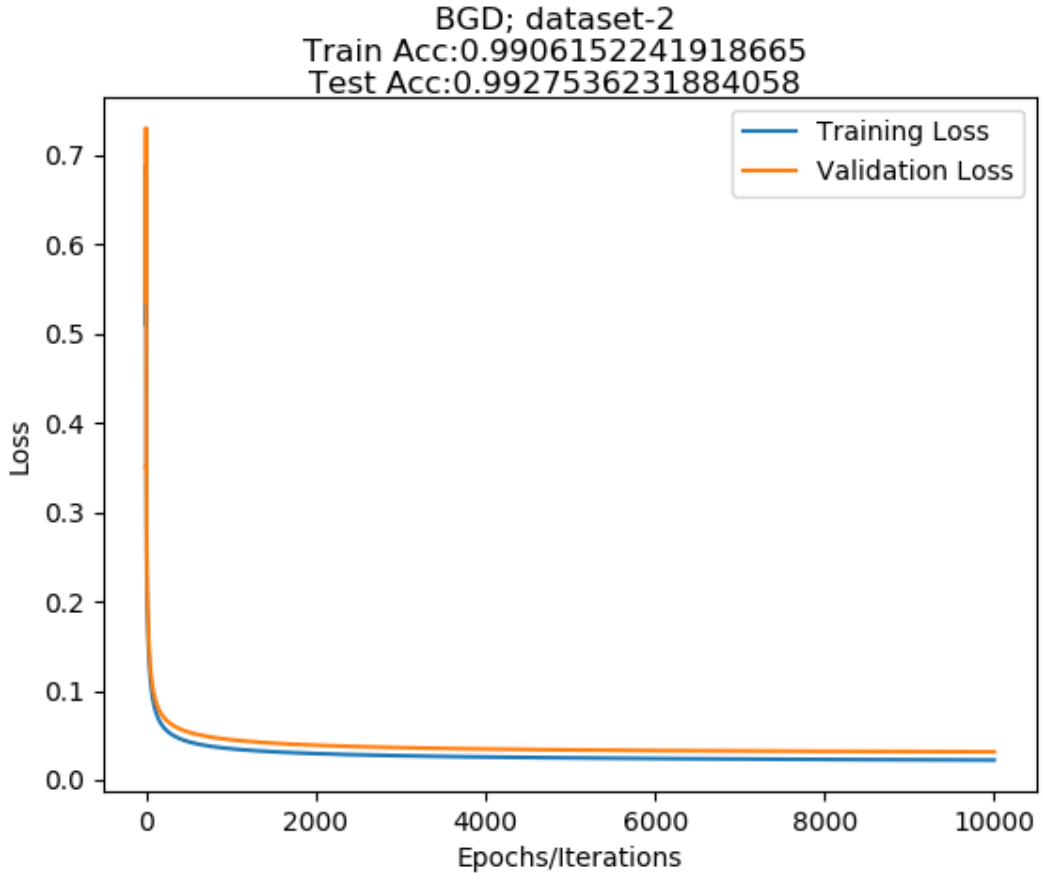

SGD; dataset-2
Train Acc:0.9697601668404588
Test Acc:0.9855072463768116
Training Loss
Validation Loss
Loss
Epochs/Iterations
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
20000
40000
60000
80000
100000


BGD; dataset-2
Train Acc:0.9155370177267987
Test Acc:0.927536231884058
Training Loss
Validation Loss
Loss
Epochs/Iterations
0.7
0.6
0.5
0.4
0.3
0
2000
4000
6000
8000
10000

- Alpha = 10, Epochs = 1,00,000 (SGD) and 10,000 (BGD)

INFO	Stochastic GD	Batch GD
FINAL LOSS	Train: NaN Test: Nan	Train: 0.02228580468983722 Test: 0.03152175781600083
ACCURACY	Train: 0.986444212721585 Test: 0.9855072463768116	Train: 0.9906152241918665 Test: 0.9927536231884058


SGD; dataset-2
Train Acc:0.986444212721585
Test Acc:0.9855072463768116
Training Loss
Validation Loss
Loss
Epochs/Iterations
0
1
2
3
4
5
6
7
0
2000
4000
6000
8000
10000


BGD; dataset-2
Train Acc:0.9906152241918665
Test Acc:0.9927536231884058
Training Loss
Validation Loss
Loss
Epochs/Iterations
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0
2000
4000
6000
8000
10000

- Clearly above plots show that BGD gives better performance in terms of non-noisy loss curves, even though BGD and SGD are close wrt final loss and prediction accuracy. But SGD converges faster even though taking a greater number of epochs, SGD converged comparatively faster than BGD.
- Sklearn implementation of the Logistic regression when applied to the same dataset gives surprising results.
    - Train accuracy: SkLearn<SGD<BGD
    - Test accuracy: SkLearn<SGD<BGD
    - Our model indeed performs better than the library probably for this specific dataset.

INFO	SGD (alpha=0.1) (epochs=100000)	BGD (alpha=3) (epochs=10000)	SkLearn
TRAIN ACCURACY	0.9874869655891554	0.9916579770594369	0.9801876955161627
TEST ACCURACY	0.9891304347826086	0.9927536231884058	0.9855072463768116

Q3

(3) MSE for Logistic Regression

$h_\theta(x) = g(\theta^T x)$

$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x_{(i)}) - y_{(i)})^2$

$\frac{\partial J(\theta)}{\partial \theta_j} = \frac{2}{m} \sum_{i=1}^{m} (h_\theta(x_{(i)}) - y_{(i)}) \cdot g'(\theta^T x_{(i)}) \cdot x_{(i)}^j$

Gradient for a point:

$\frac{\partial J(\theta)}{\partial \theta_j} = 2 (g(\theta^T x) - y)\, g(\theta^T x)[1 - g(\theta^T x)]\, x^j$

Now consider a wrong result:

Case 1: $y=1$, $g(\theta^T x) \to 0$

$\Rightarrow \frac{\partial J(\theta)}{\partial \theta_j} \to 0 \quad \forall j$

Case 2: $y=0$, $g(\theta^T x) \to 1$

$1 - g(\theta^T x) \to 0$

$\Rightarrow \frac{\partial J(\theta)}{\partial \theta_j} \to 0 \quad \forall j$

In either case, gradient $\to 0$, hence model will not be able to learn well since no updates will happen in SGD if this particular element gets selected.

*Model is not penalizing this data point.

Using cross entropy loss will not lead the gradient to zero in such cases because of a log term which will prevent it.

$\frac{\partial J(\theta)}{\partial \theta_j} = \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x_{(i)}) - y_{(i)})\, x_{(i)}^j$ (Cross Entropy Loss)

This will surely penalize wrong results.

Q4

- Estimate β0, β1, and β2 using MLE. (Found using our implementation of MyLogisticRegression, code used is available in a file called **Q4.py**)
    - β0 = -4.28312333
    - β1 = 0.06246534
    - β2 = 0.52165424

Q4) (a) $\beta_0 = -4.283$
$\beta_1 = +0.062$
$\beta_2 = +0.522$

(b) Response function

$$\hat{R}(x_i) = \frac{e^{\beta_0 + \beta_1 x_1^{(i)} + \beta_2 x_2^{(i)}}}{1 + e^{\beta_0 + \beta_1 x_1^{(i)} + \beta_2 x_2^{(i)}}}$$

Values of $\beta_0, \beta_1, \beta_2$ can be put from the result of (a) part

(c)
(i) $e^{\beta_1} = e^{0.062} = 1.064$
This number represents the change in the odds ($Y=1$) i.e. the odds of # children with recurrence to the # children without recurrence, per unit change in X_1, i.e. the extent (in %) to which the disease spread occurred.

(ii) $e^{\beta_2} = e^{0.522} = 1.685$
Similar to above, this represents the change in the odds ($Y=1$) per unit change in the age of the patient (X_2) in years.

(d) $P(Y=1 \mid X_1 = 75, X_2 = 2) = \frac{1}{1 + e^{-\beta_0 - \beta_1 X_1 - \beta_2 X_2}}$

Putting in the values, we get
$P = 0.8092$

- What is the estimated probability that a patient with 75% of disease spread and an age of 2 years will have a recurrence of disease in the next 5 years?
    - P = 0.80924092 (Predicted by the regression model)

Q5

(5) $Y_{n\times 1}$; $X_{n\times k}$; $\beta_{k\times 1}$; $\varepsilon_{n\times 1}$

Hypothesis : $\hat{Y} = X\beta + \varepsilon$

$J(\beta)$:= Mean Squared Loss

$$J(\beta) = \frac{1}{n}\sum_{i=1}^{n}\left(Y_{(i)} - \hat{Y}_{(i)}\right)^2$$

$$= \frac{1}{n}\left(Y - \hat{Y}\right)^T\left(Y - \hat{Y}\right)$$

$$nJ(\beta) = (Y - X\beta - \varepsilon)^T(Y - X\beta - \varepsilon)$$

$$= (Y^T - \beta^T X^T - \varepsilon^T)(Y - X\beta - \varepsilon)$$

$$= Y^TY - Y^TX\beta - Y^T\varepsilon - \beta^TX^TY + \beta^T(X^TX)\beta + \beta^T(X^T\varepsilon) - \varepsilon^TY + \varepsilon^TX\beta + \varepsilon^T\varepsilon$$

$Y^TX\beta \equiv (1\times n).(n\times k).(k\times 1) \equiv 1\times 1$

$\Rightarrow$ scalar $\Rightarrow (Y^TX\beta)^T \equiv (Y^TX\beta)^T$

$= \beta^TX^TY$

$\varepsilon^TX\beta \equiv (1\times n).(n\times k).(k\times 1) \equiv 1\times 1$

$\Rightarrow \varepsilon^TX\beta = (\varepsilon^TX\beta)^T = \beta^TX^T\varepsilon$

$$\therefore nJ(\beta) = Y^TY - 2\beta^TX^TY - Y^T\varepsilon + \beta^T(X^TX)\beta + 2\beta^T(X^T\varepsilon) - \varepsilon^TY + \varepsilon^T\varepsilon$$

#Differentiating :

$$\frac{n\partial J(\beta)}{\partial\beta} = 0 - 2X^TY - 0 + 2X^TX\beta + 2(X^T\varepsilon) - 0 + 0$$

#Minimize Error : $\partial J(\beta) = 0$

$\Rightarrow -2X^TY + 2X^TX\beta + 2X^T\varepsilon = 0$

$\Rightarrow X^TX\beta = X^T(Y - \varepsilon) \Rightarrow \boxed{\beta = (X^TX)^{-1}X^T(Y-}$