# ML Assignment- 1 Report

-Daksh Thapar

2018137

***Q1- Linear Regression***

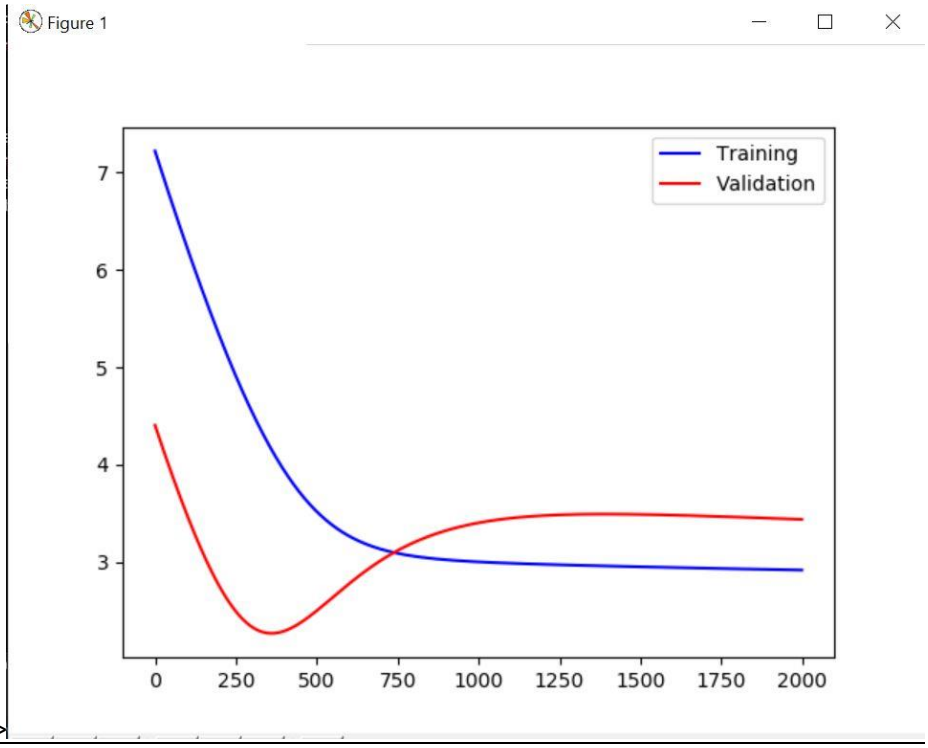
1. Performed K Fold Cross Validation on my set. I varied my K from 3 to 10 one by one, and observed that for K=10, my average error values for both RMSE and MAE come out to be less. Higher the K, lower are the costs but more in the computational inefficiency. I can even take a higher value of K, but it doesn't improve my error values considerably and ends up taking even more time. It's a trade- off between minimal gains on error and computational inefficiency, as taking K=10 is the most common practice for practical purposes and gives me a good improvement on error with respect to lower Ks, I have picked this value

2. In the code, I have made 2 gradient descent functions, 1 for RMSE and 1 for MAE; 1 for each because the derivatives of cost functions are different for both of them. I have not called these functions explicitly in the test file, but they are called in other functions' calls; i.e. to fit a model on my dataset, my fit functions calls gradient descent which takes Xtrain and ytrain sets and has a pre initialised theta parameter matrix and in every epoch, and is optimised in every epoch.

***<u>ANALYSIS</u>***

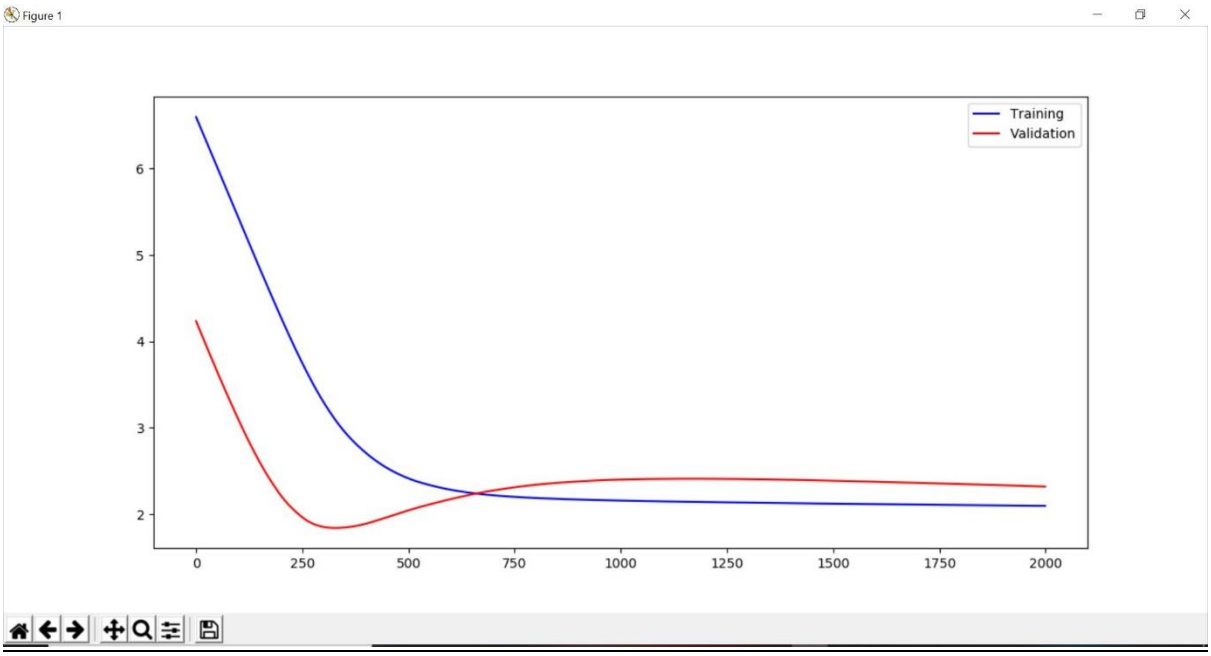
1. Training loss vs Iterations, compared to Validation loss vs iterations on the same plot

**DATASET1**



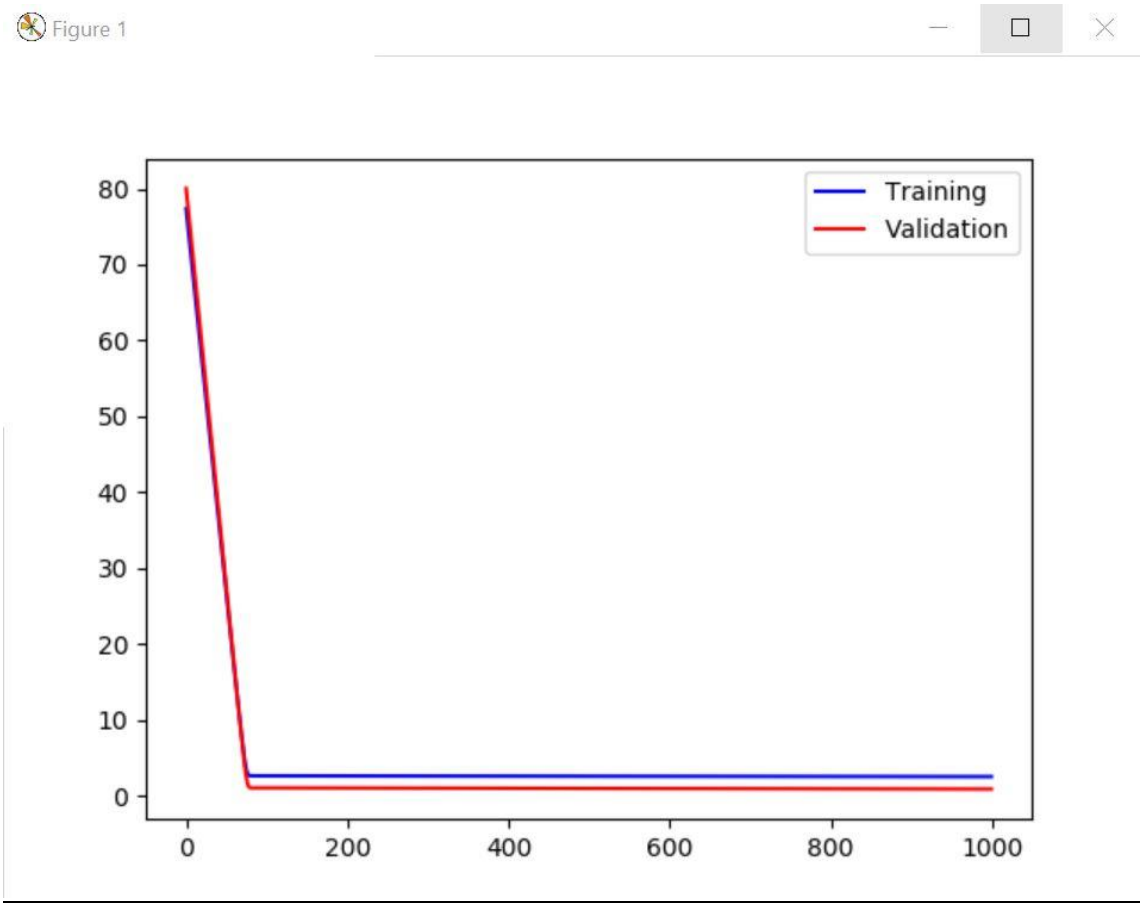

<u>For RMSE-></u>

For MAE->

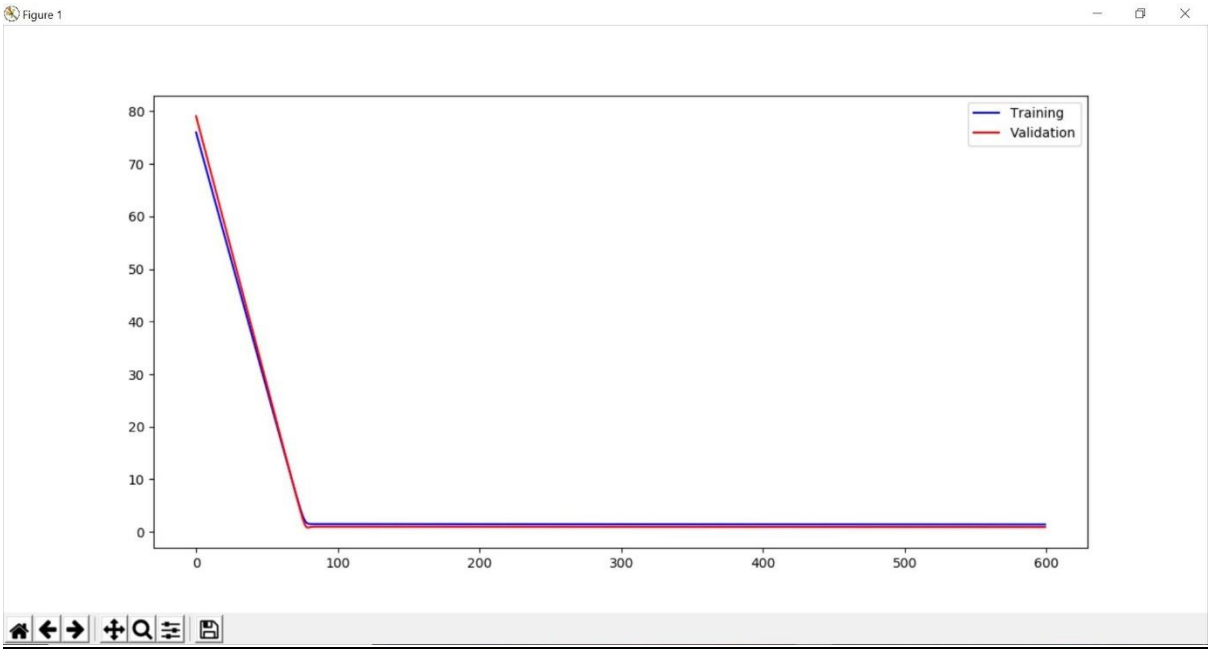



**DATASET2**

For RMSE->



For MAE->




2. RMSE vs MAE for best fold

**DATASET1**

*RMSE for folds*= [array([3.24663288]), array([4.45718749]), array([2.3930512]), array([1.82890927]), array([2.01973999]), array([3.90482775]), array([2.15061725]), array([2.75076387]), array([2.43852167]), array([2.5379686])]

*MAE for folds*= [array([2.62532847]), array([3.71073025]), array([1.49118541]), array([1.09860481]), array([1.19301562]), array([3.22577275]), array([1.29113192]), array([2.08444674]), array([1.58367272]), array([1.73235557])]

RMSE average= [2.772822]
MAE average= [2.00362443]
RMSE best value= [1.82890927]
MAE best value= [1.09860481]

**DATASET2**

*RMSE for folds*= [array([6.68628372]), array([1.15837753]), array([0.98181236]), array([1.09494295]), array([1.23079075]), array([1.38853299]), array([1.48162978]), array([1.632853]), array([1.67060213]), array([1.55983201])]

*MAE for folds*= [array([4.50370222]), array([1.23150835]), array([0.87030642]), array([0.74535192]), array([0.80332444]), array([0.93378243]), array([1.00565845]), array([1.13660364]), array([1.16899416]), array([1.02802103])]

RMSE average= [1.88856572]
MAE average= [1.34272531]
RMSE best value= [0.98181236]
MAE best value= [0.74535192]

Here my value of K is 10, so my code performs 10 different splits on the dataset, and for every split it finds out the MAE or RMSE (depends on function argument) value and saves it into an instance attribute for every split (hence 10 values as above) along with their corresponding optimised Theta parameters.

So I find out my minimum error out of the list of 10 for both MAE and RMSE, i.e. the best values and only for these best values each I save that corresponding theta to my instance attribute and only this theta parameter matrix will be used not to predict values for test set.

3. For Dataset1,
   RMSE average= [2.772822]
   MAE average= [2.00362443]

   For Dataset2,
   RMSE average= [1.88856572]
   MAE average= [1.34272531]

   For both datasets, we can observe that MAE (averaged for folds) has a smaller value as compared to RMSE (averaged for folds).

   Now to compare MAE and RMSE -
   **DATASET1**
   *RMSE for folds*= [array([3.24663288]), array([4.45718749]), array([2.3930512]), array([1.82890927]), array([2.01973999]), array([3.90482775]), array([2.15061725]), array([2.75076387]), array([2.43852167]), array([2.5379686])]

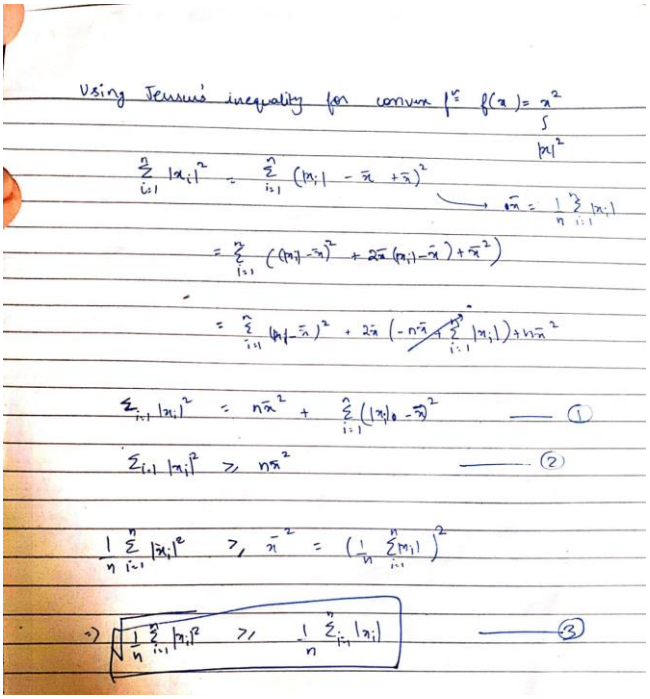
   *MAE for folds*= [array([2.62532847]), array([3.71073025]), array([1.49118541]), array([1.09860481]), array([1.19301562]), array([3.22577275]), array([1.29113192]), array([2.08444674]), array([1.58367272]), array([1.73235557])]

   **DATASET2**
   *RMSE for folds*= [array([6.68628372]), array([1.15837753]), array([0.98181236]), array([1.09494295]), array([1.23079075]), array([1.38853299]), array([1.48162978]), array([1.632853]), array([1.67060213]), array([1.55983201])]

   *MAE for folds*= [array([4.50370222]), array([1.23150835]), array([0.87030642]), array([0.74535192]), array([0.80332444]), array([0.93378243]), array([1.00565845]), array([1.13660364]), array([1.16899416]), array([1.02802103])]

Here we see that for Dataset 1,errors for folds are pretty comparable for al folds in a list, while in Dataset 2 we are finding global scales and in some folds the errors are huge, showing there are some major outliers in Dataset 2 (can also be seen from the standard dev value in df.describe()) and not so many in Dataset 1. RMSE has the benefit of penalizing large errors more so will be less appropriate in this case as we are comparing global sales, so I feel MAE is a better option for both datasets that have some outliers.

4. In general, RMSE>=MAE
   This can be proved using Jensen's inequality using convex function $x^2$




LHS=RMSE. RHS=MAE
(Both squared here)

Since the errors are squared before they are averaged, RMSE gives a relatively high weight to large errors. This means the RMSE is most useful when large errors are particularly undesirable.

The RMSE will always be larger or equal to the MAE; as can be seen from the above identity LHS=RHS when variance over set of (x0,x1…) is 0. This shows that greater the variance in individual errors in our dataset, more is the difference between RMSE and MAE. For MAE=RMSE, ideally the variance over the sample should be 0, i.e. all errors are of the same magnitude. For the given dataset, the individual losses are definitely not equal and the variance over the same is greater than 0, so RMSE will give a higher value than MAE as can be observed by my test, hence I will pick MAE to find errors for these datasets given the explanation.

From equation 1, RMSE=MAE for variance term of errors to be 0.
From equation 2 and 3, RMSE>=MAE

If RMSE= MAE for my model, I will probably pick RMSE. MAE is harder to optimize and does not have a closed form solution because it is a non-differentiable piecewise function, as it involves an absolute value. For this reason, MAE is computationally more expensive, as we can't solve it in terms of matrix mathematics and most rely on approximations.

5. I have created a function for Normal form of linear regression for best fold of my Dataset 1. As observed from the previous parts, we observe MAE is the better loss function and it gives smaller error values so we choose that to find Training loss and validation loss for the best fold.

   Call K fold for splitting into Training and validation Sets and picking up the best split and finding MAE and RMSE losses-

   MAE Training loss for best fold= [1.64372872]
   MAE Validation loss for best fold= [1.2960103865195918]

---

***Q2- Logistic Regression***

- For EDA I have performed the following steps

```
Pairwise correlation of features, using standard correlation coefficient
          variance  skewness  curtosis   entropy     class
variance  1.000000  0.264026 -0.380850  0.276817 -0.724843
skewness  0.264026  1.000000 -0.786895 -0.526321 -0.444688
curtosis -0.380850 -0.786895  1.000000  0.318841  0.155883
entropy   0.276817 -0.526321  0.318841  1.000000 -0.023424
class    -0.724843 -0.444688  0.155883 -0.023424  1.000000
```

  Above is the correlation matrix for the features, a nxn grid

- For variance and class, there is high negative correlation; increase of variance means lesser class label value
- For variance and kurtosis, there is high negative correlation; increase of variance means lesser curtosis
- 

```
First 5 rows
   variance  skewness  curtosis  entropy  class
0   3.62160    8.6661   -2.8073 -0.44699      0
1   4.54590    8.1674   -2.4586 -1.46210      0
2   3.86600   -2.6383    1.9242  0.10645      0
3   3.45660    9.5228   -4.0112 -3.59440      0
4   0.32924   -4.4552    4.5718 -0.98880      0
```

```
First 5 rows
   variance  skewness  curtosis  entropy  class
0   3.62160    8.6661   -2.8073 -0.44699      0
1   4.54590    8.1674   -2.4586 -1.46210      0
2   3.86600   -2.6383    1.9242  0.10645      0
3   3.45660    9.5228   -4.0112 -3.59440      0
4   0.32924   -4.4552    4.5718 -0.98880      0
```

```
Data Info
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1372 entries, 0 to 1371
Data columns (total 5 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   variance  1372 non-null   float64
 1   skewness  1372 non-null   float64
 2   curtosis  1372 non-null   float64
 3   entropy   1372 non-null   float64
 4   class     1372 non-null   int64
dtypes: float64(4), int64(1)
memory usage: 53.7 KB
None
```

```
Presence of null values
variance    0
skewness    0
curtosis    0
entropy     0
class       0
dtype: int64
```

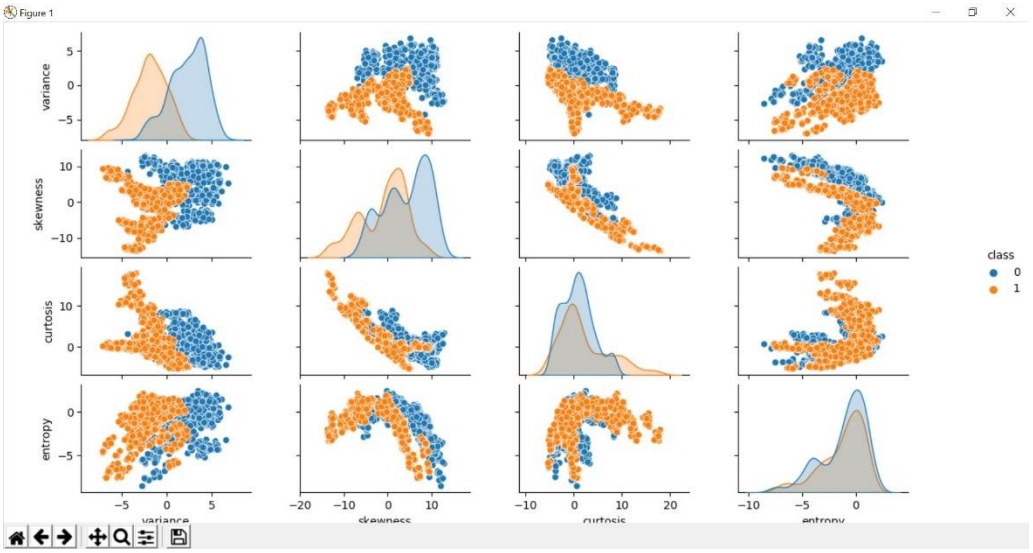
```
Data Description
          variance     skewness     curtosis      entropy        class
count  1372.000000  1372.000000  1372.000000  1372.000000  1372.000000
mean      0.433735     1.922353     1.397627    -1.191657     0.444606
std       2.842763     5.869047     4.310030     2.101013     0.497103
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
```

Data description gives us a brief idea about mathematical measures like mean std and count for all features

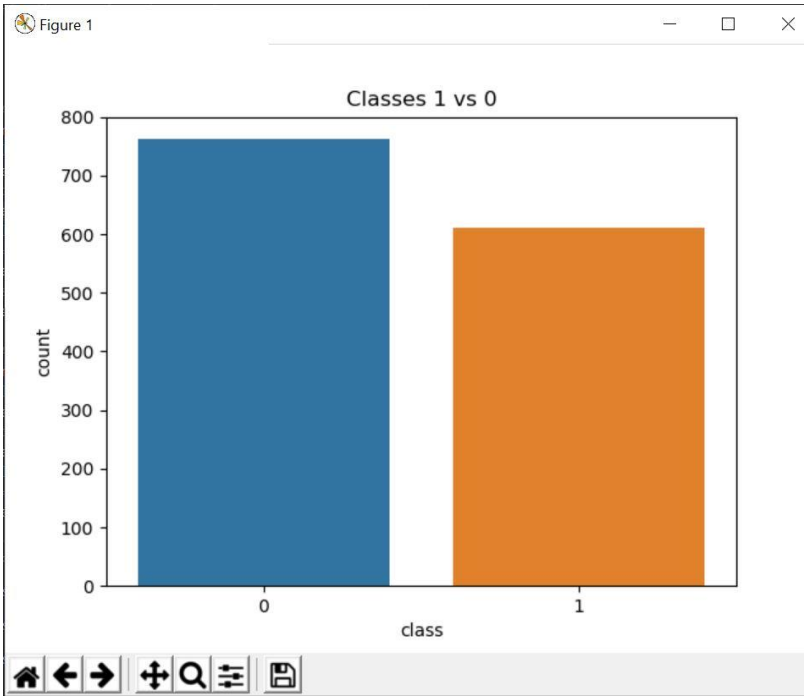
```
Class 0 vs Class 1
0    762
1    610
Name: class, dtype: int64
```

Gives us an idea of the frequencies of class labels

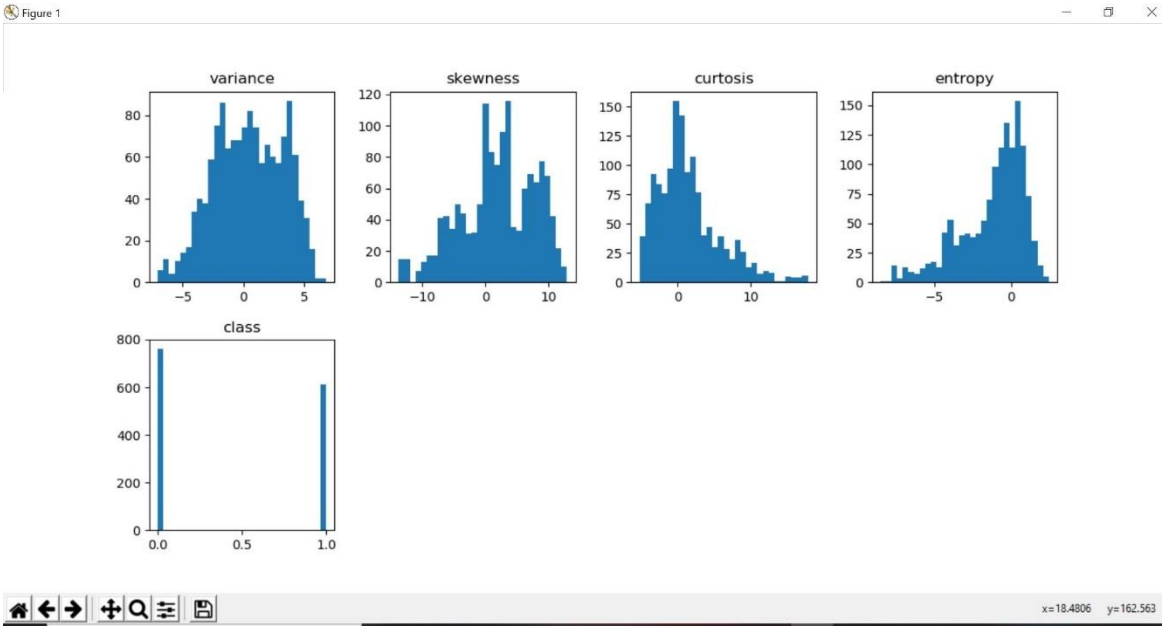
Pairplots between attributes given based on class labels



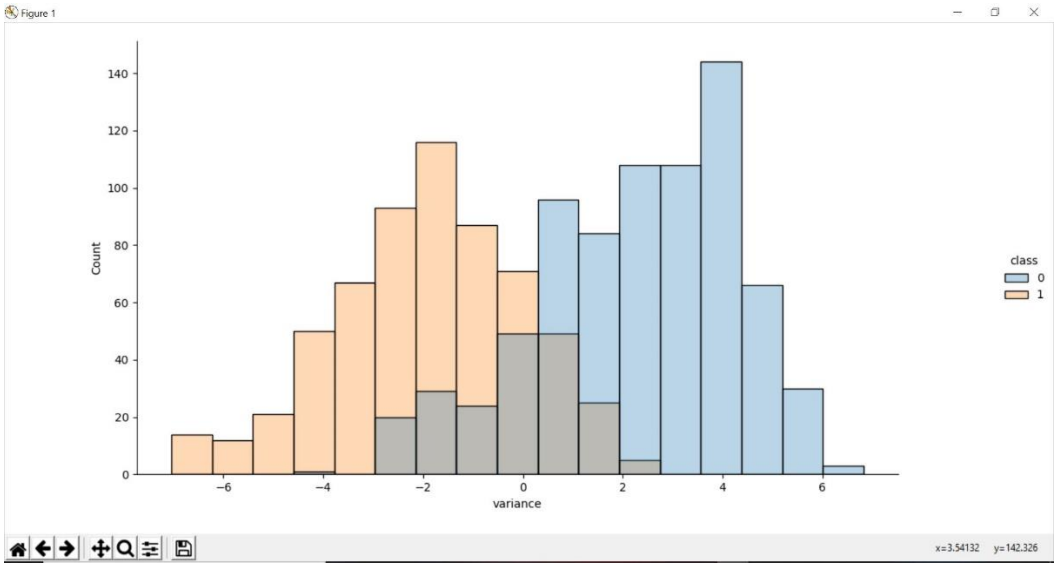
Class 0 vs Class 1




Attribute Histogram

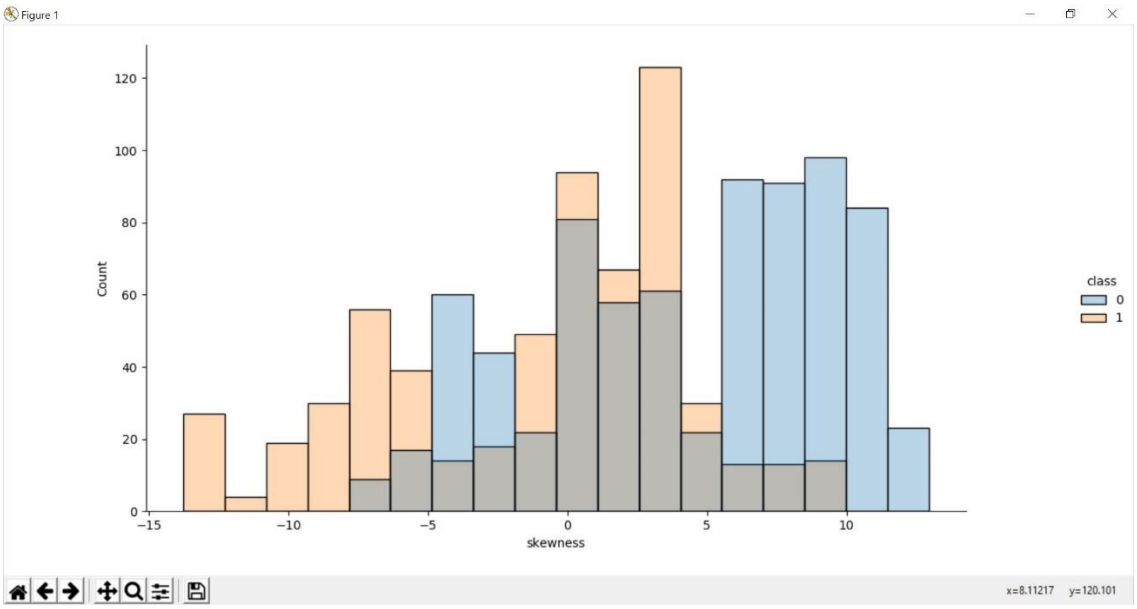


Variance vs Class label histograms



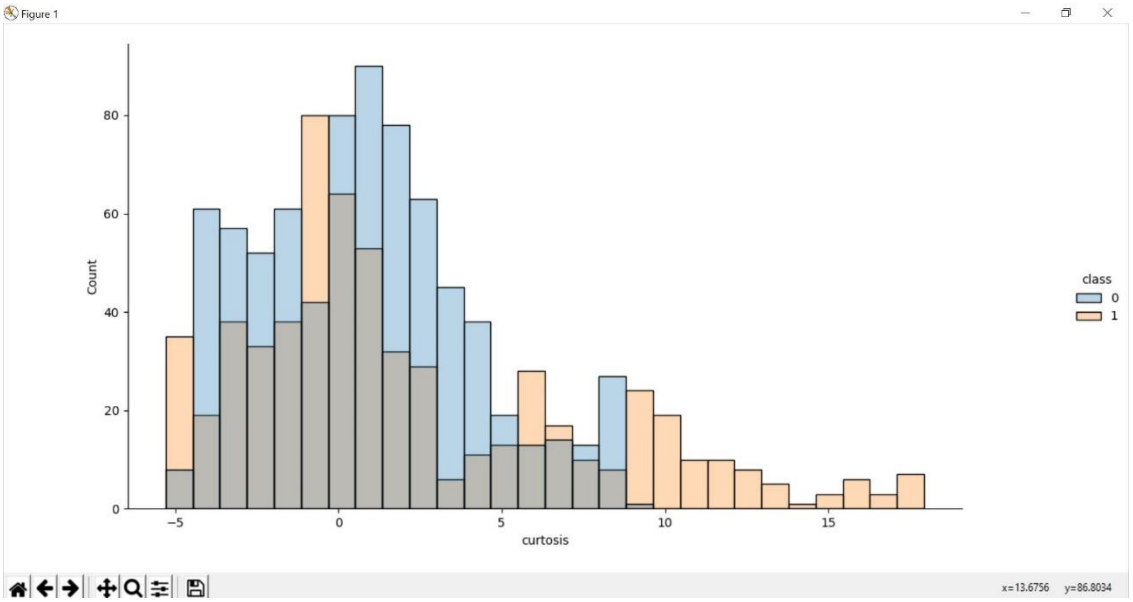

Using this histogram we can observe that for variance<1 we have class labels 1 predominantly and for variance>1 we have class labels 0 predominantly. This graph gives us a really good idea of the clusters formed and hence we can come to this conclusion that Variance is an important feature to classify.

Skewness vs Class label histograms



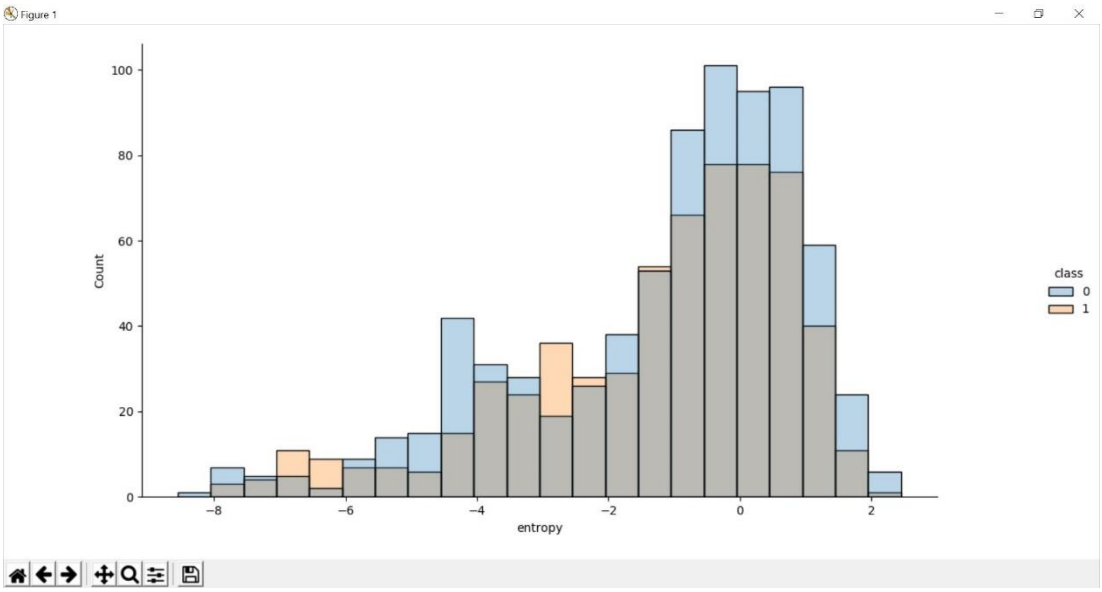

Here we observe for class label 0, skewness >5 or (<-1 and >-5) (more probability of occurrence) and the rest for class label 1 (more probability of occurrence). It's still an important factor to classify our data into these 2 classes, as it creates a segregation between the 2, though not as good as variance

Curtosis vs Class label histograms




For curtosis<9, class label occurrence is predominantly 0, and for curtosis>9, class label 1 has more occurrences and for >10 we only have label 1 values. This is a nice factor to classify dataset into 2 classes.

Entropy vs Class label histograms




For entropy>-6, class label 1 is more probable than 0 if we look at the occurrences but not by much for every column, so entropy is not a great factor to classify for our dataset, as class 0 and 1 label values vary over the entire entropy range.

## Split Dataset into 7:1:2

For training set 70%

For validation set 10%

For testing set 20%

Called a function to split data into these 3 categories after pre-processing the data.

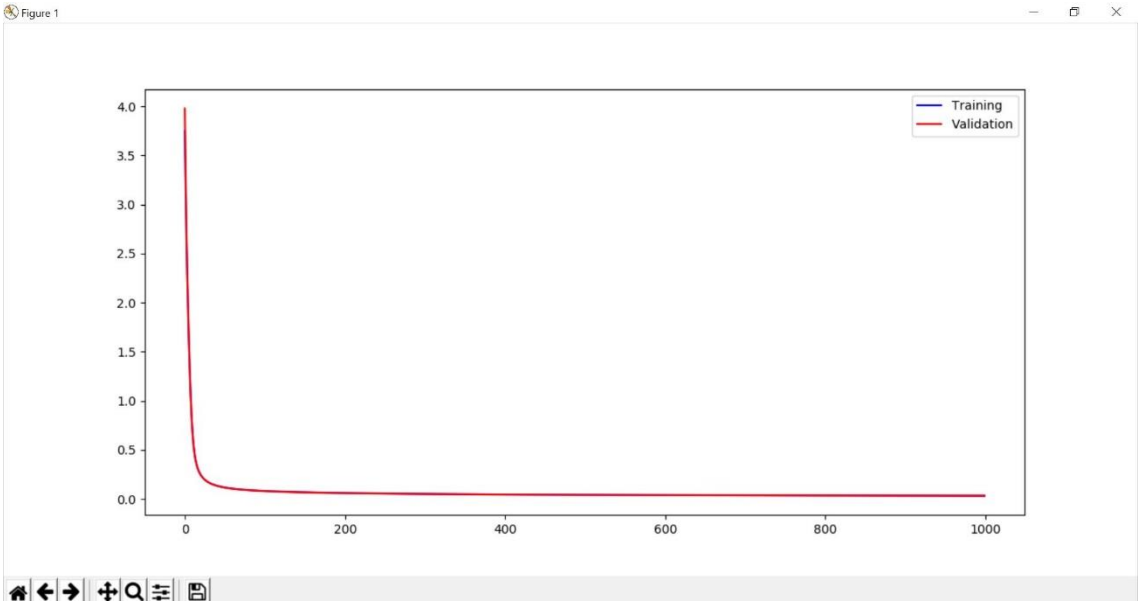
P.T.O.

***Q2- Logistic Regression***

## Batch Gradient Descent

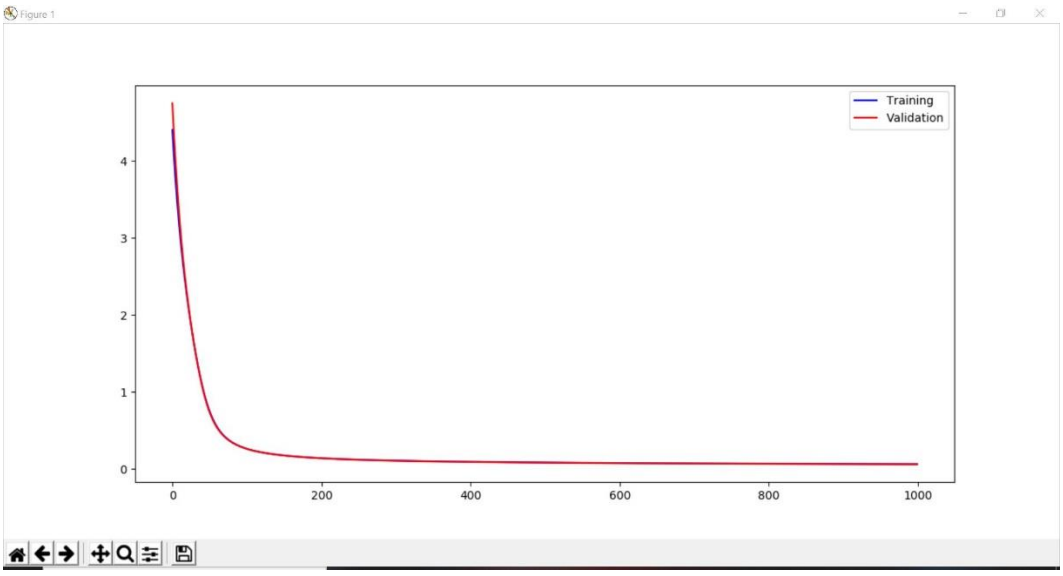
1. Best alpha= 0.05, iterations= 1000

   Training set
   Accuracy= 99.16579770594369
   Testing set
   Accuracy= 97.82608695652173



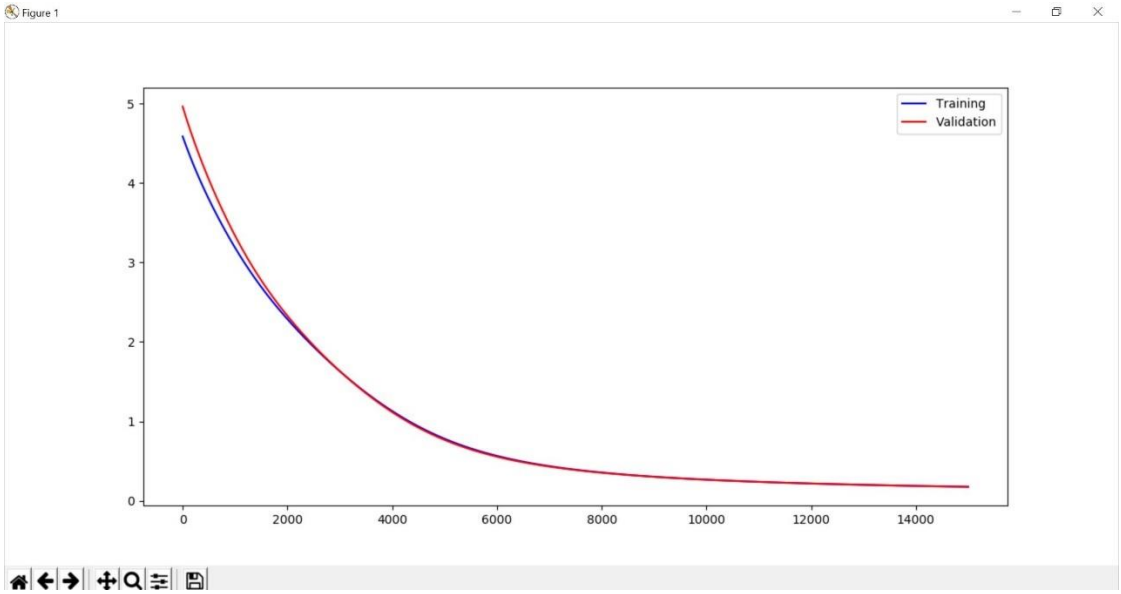

2. Alpha=0.01, iterations=1000

   Training set
   Accuracy= 98.95724713242961
   Testing set
   Accuracy= 97.10144927536231



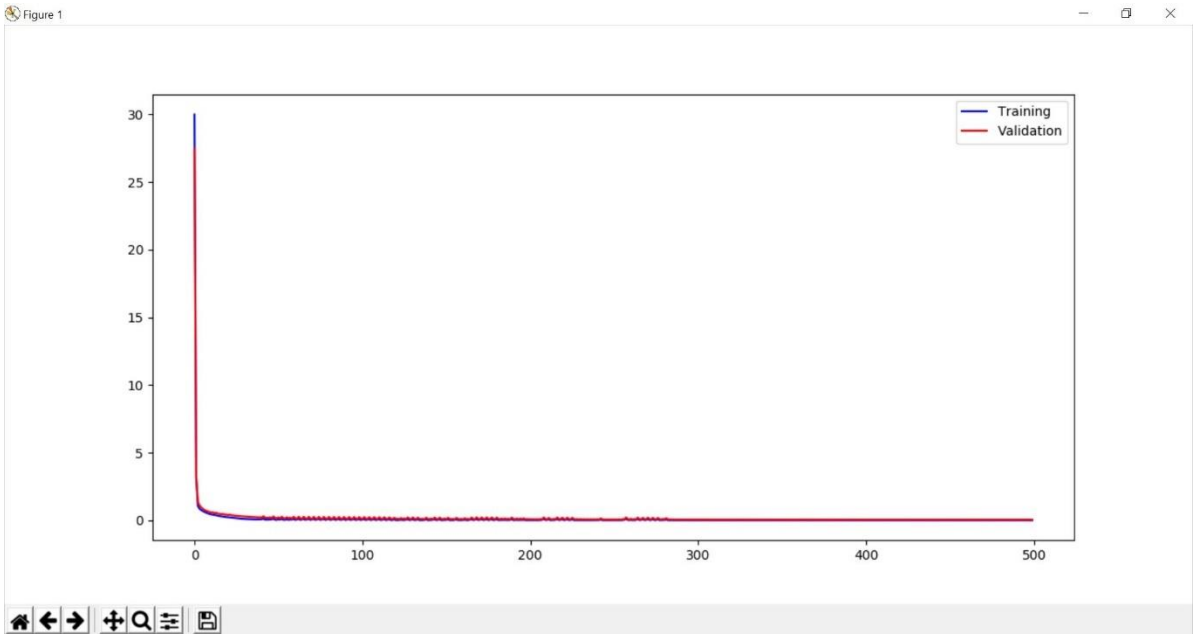
3. alpha=0.0001, iterations=15000

   Training set
   Accuracy= 95.82898852971846
   Testing set
   Accuracy= 96.3768115942029




4. alpha=10 iterations=500

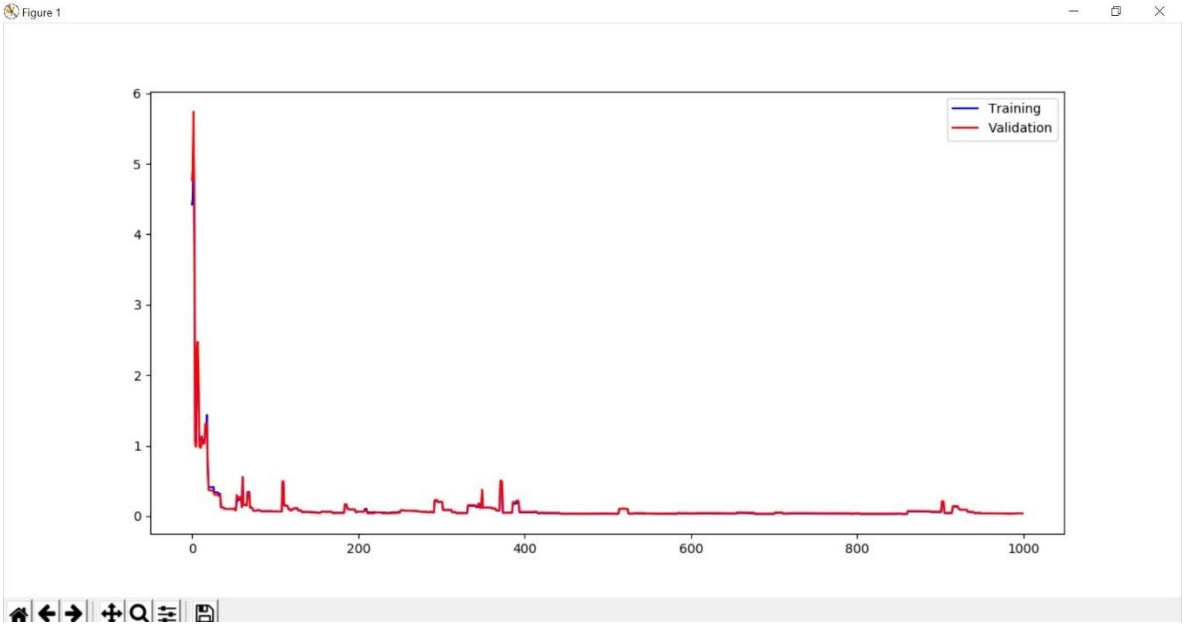
   Training set
   Accuracy= 99.79144942648593
   Testing set
   Accuracy= 97.62608695652173



## Stochastic Gradient Descent

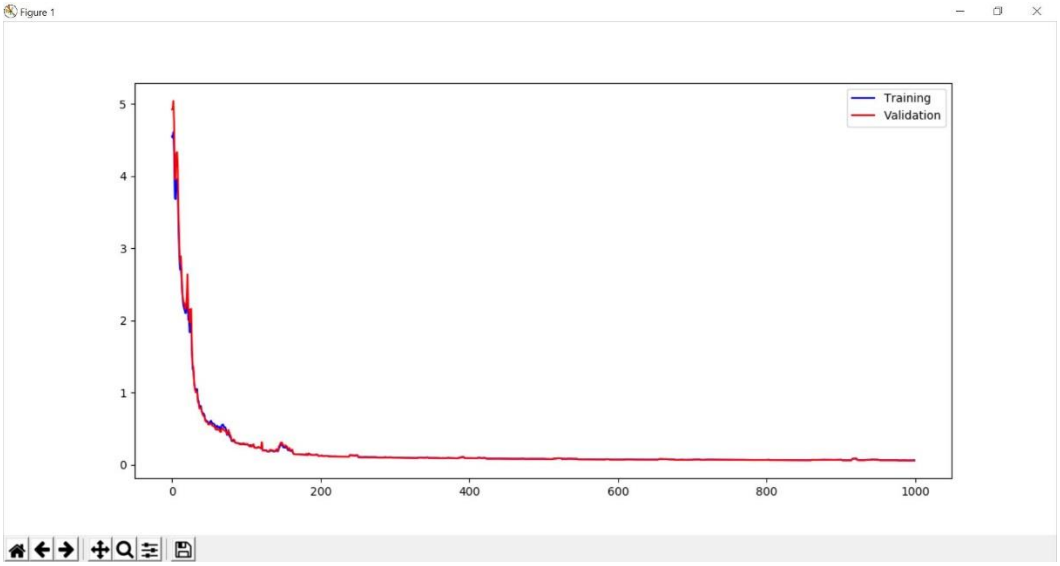
1. Best alpha= 0.05, iterations= 1000

   Training set
   Accuracy=  98.95724713242961
   Testing set
   Accuracy=  97.10144927536231



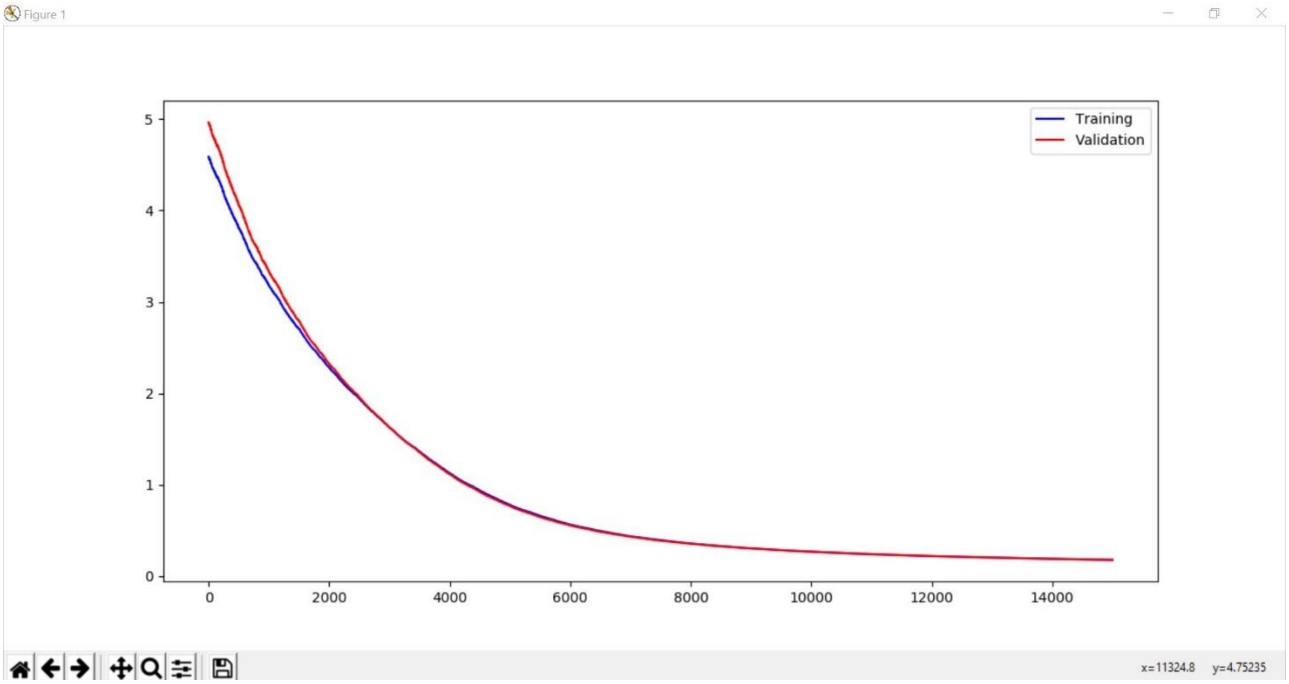

2. alpha=0.01 iterations=1000

   Training set
   Accuracy=  98.6444212721585
   Testing set
   Accuracy=  96.3768115942029



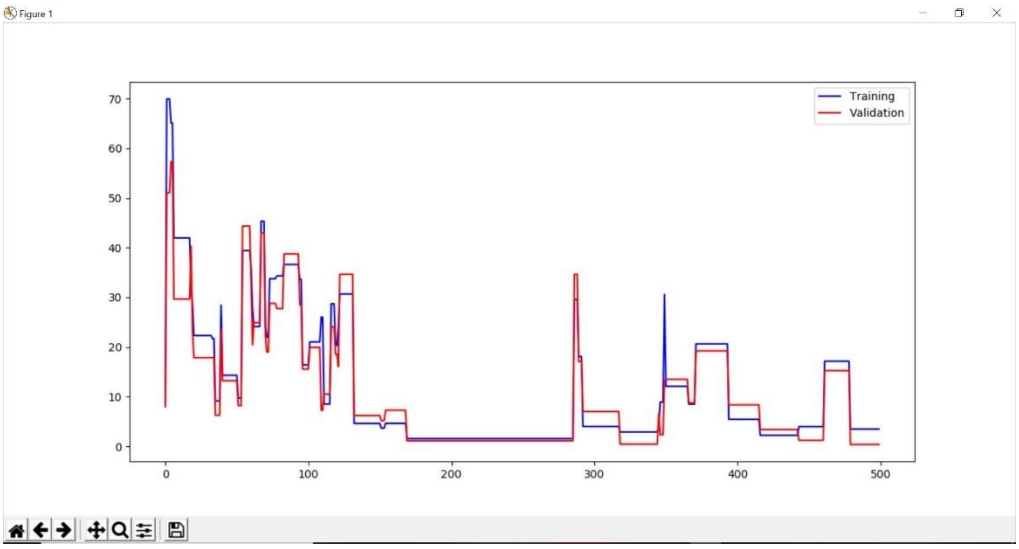
3. alpha=0.0001 iterations=15000

   Training set
   Accuracy= 95.82898852971846
   Testing set
   Accuracy= 96.3768115942029




4. alpha=10 iterations=500

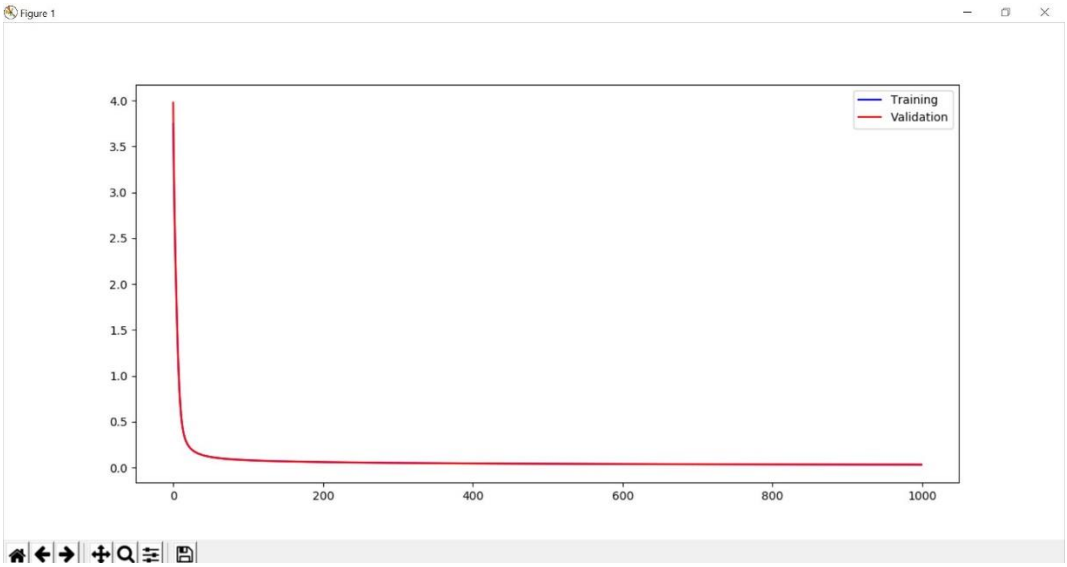
   Training set
   Accuracy= 97.08029197080292
   Testing set
   Accuracy= 94.92753623188406



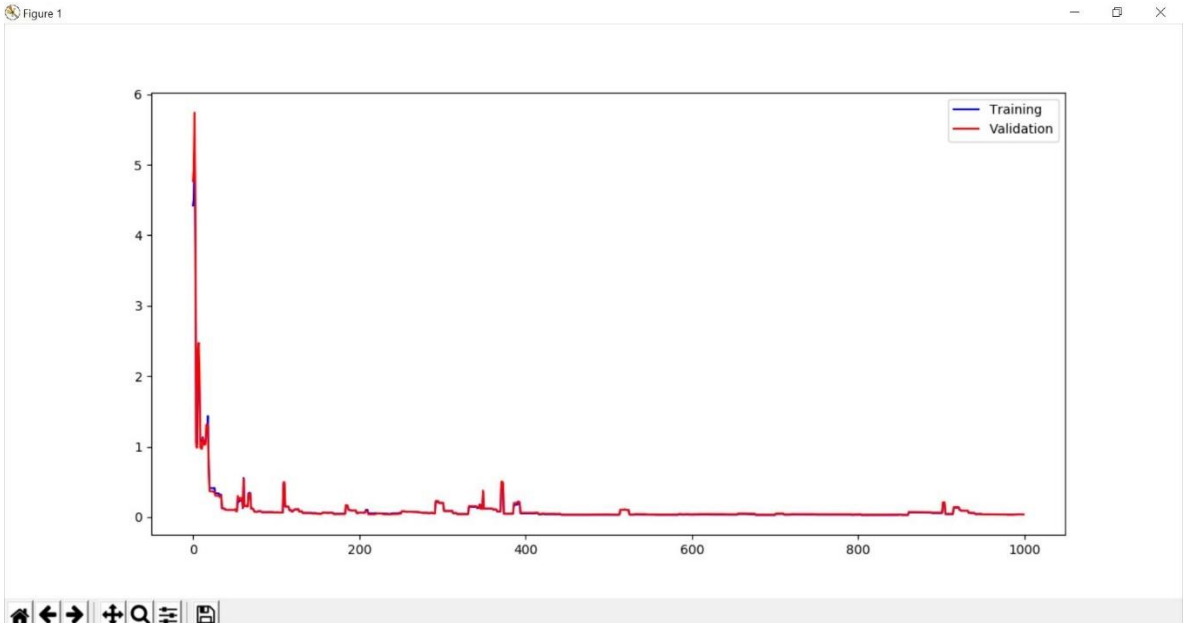
## For SGD vs BGD

1. Alpha=0.05, iterations=1000

BGD



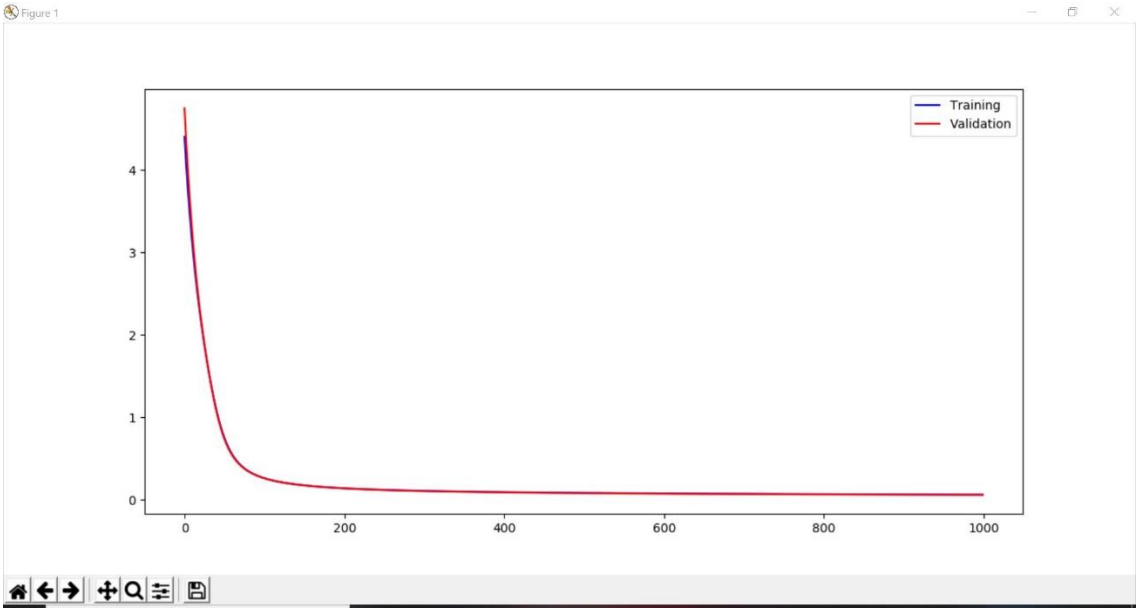

SGD



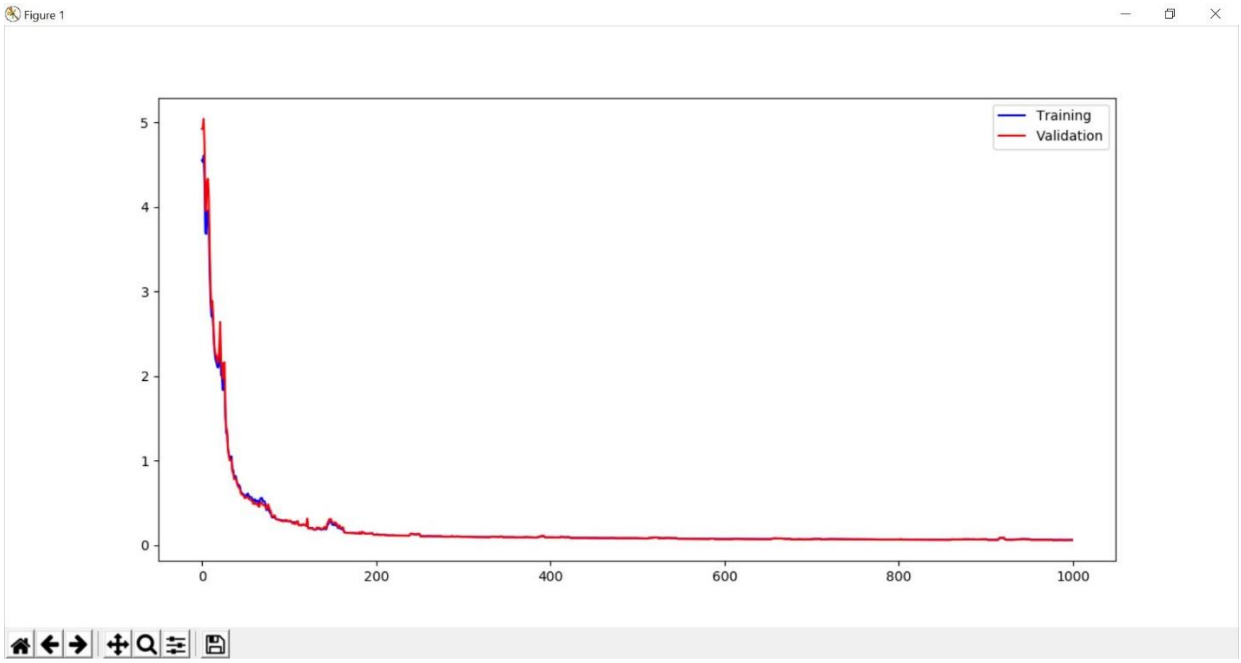

Graph stabilises and becomes parallel to x axis earlier in the case of BGD and is very smooth too, whereas in SGD we see some disturbances and random peaks and it tends to reduce to the threshold value in more epochs, and has a little poorer accuracy as compared to BGD.

2. Alpha=0.01, iterations=1000

BGD



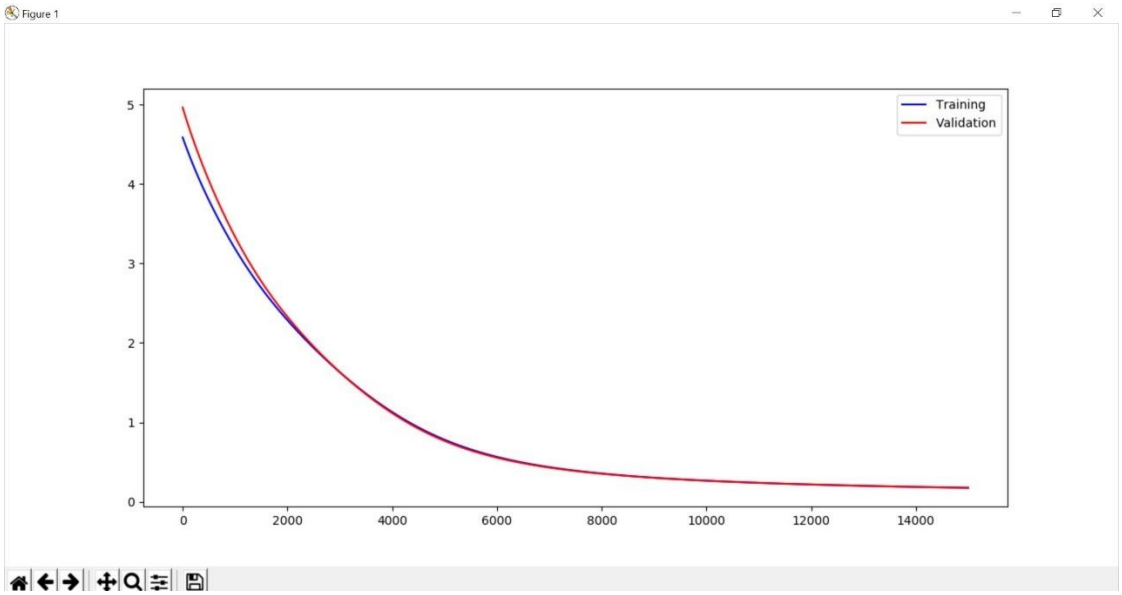

SGD



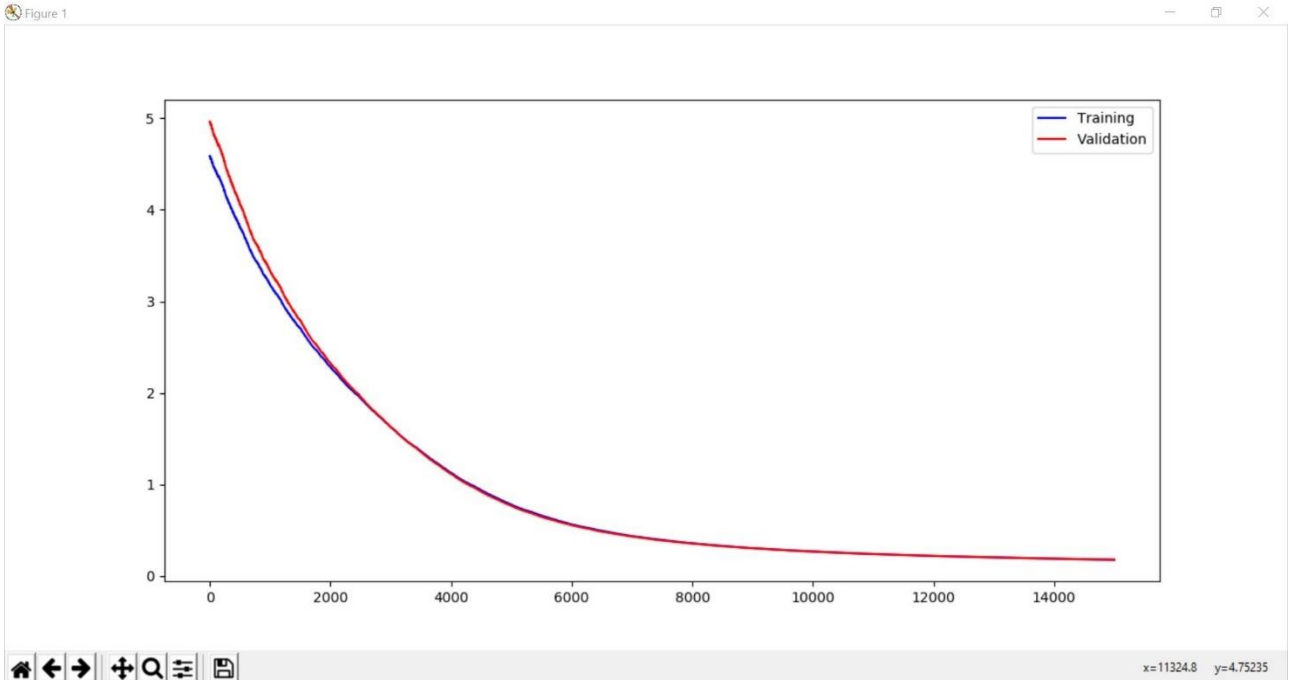

Graph for BGD is smoother again with some disturbances for SGD and for these hyperparameters reports comparable accuracies. On observation and zooming, curve for BGD converges earlier than SGD.

3. Alpha=0.0001, iterations=15000

BGD



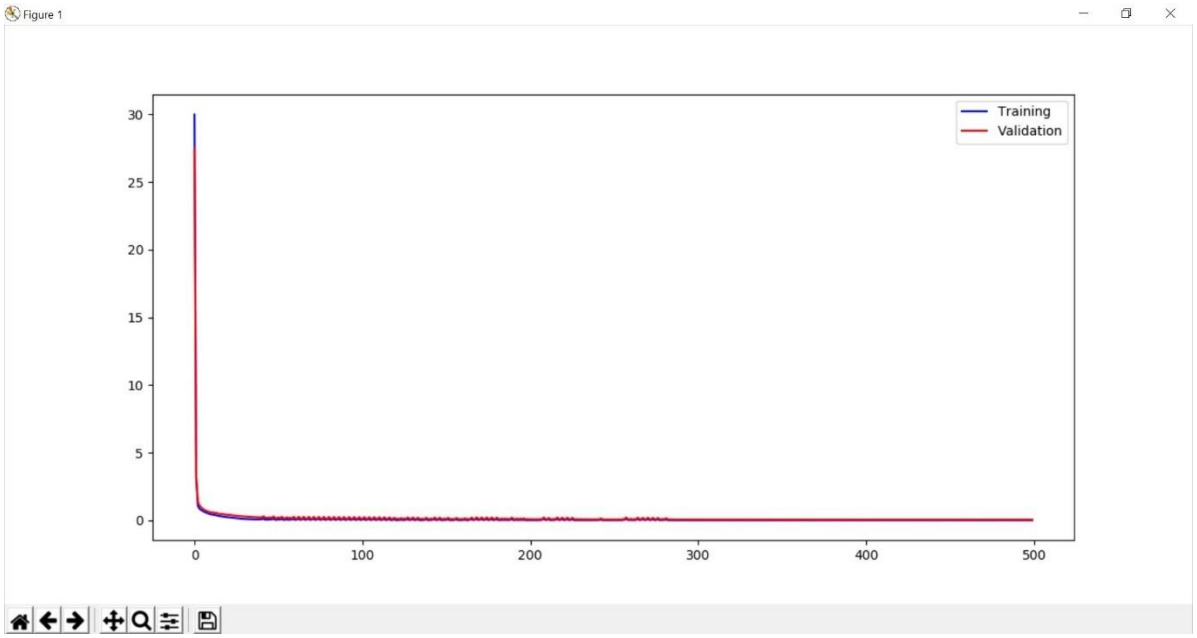

SGD



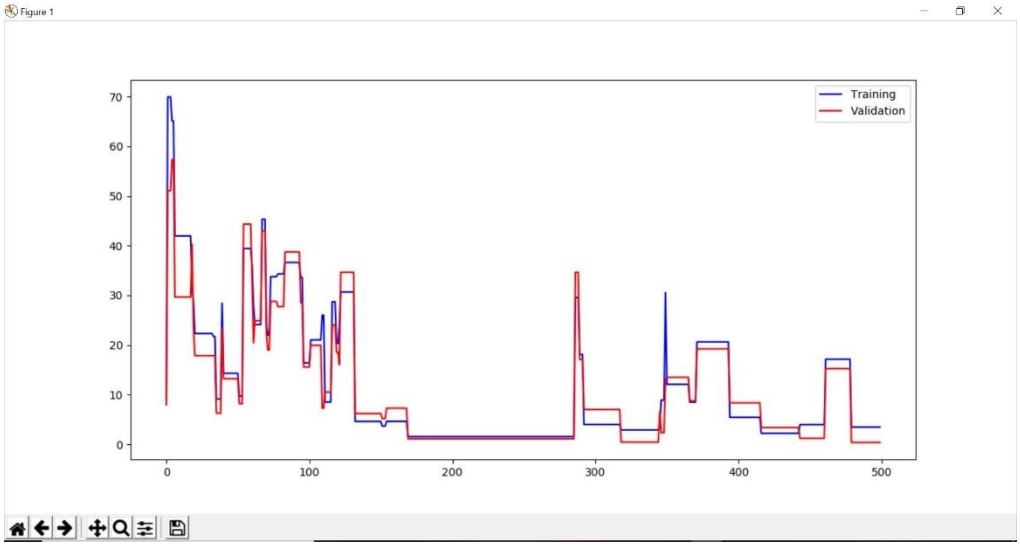

Graph for BGD is smoother again with some minute disturbances for SGD this time and for these hyperparameters reports comparable accuracies. On observation and zooming, curve for BGD converges earlier than SGD.

4. Alpha=10, iterations=500

BGD




SGD




Graph for BGD is smoother and because of high alpha it reaches minima very quickly and stabilises and oscillates a little in the minima because of alpha being too large. Meanwhile for SGD there is a recurring problem whereas sigmoid function yields maths overflow error, so to deal with that I have clipped my value to a threshold so when it exceeds that value I clip it to that value and give a certain penalty as cost. Since the alpha is too large for this case, it tends to move back and forth with respect to the minima and yields us a graph and a lot of spikes and troughs. Also we can clearly see BGD converges quicker than SGD.

3. Using sklearn's Logistic Regression implementation on my training and test set-

   Accuracy for Training Set: 99.27007299270073
   Accuracy for Test Set: 98.18840579710145

   Also used SGD classifier for part d) for the best hyperparameters taken in SGD (alpha=0.05, iterations=1000)

   *Sklearn's performance-*
   Accuracy for Training Set: 99.27007299270073
   Accuracy for Test Set: 98.18840579710145

   *SGD performance-*

   Training set Accuracy= 98.95724713242961
   Testing set Accuracy= 97.10144927536231

   We can clearly observe Sklearn's methods perform better and gives more accurate results.

**Q3, Q4, Q5 handwritten**
**P.T.O**

Q3. According to question, we are using mean square error for logistic regression model

i.e. cost f$^n$ $f(x) = (y-\hat{y})^2$ for one sample

($\hat{y}$ → predicted value)

$y \rightarrow$ true value

$\hat{y} \rightarrow \frac{1}{1+e^{-ax}}$, i.e. sigmoid f$^n$

$$f(x) = (y-\hat{y})^2$$

$$f'(x) = \frac{df}{d\theta} = \frac{df}{d\hat{y}} \cdot \frac{d\hat{y}}{d\theta}$$

$$f'(x) = -2(y-\hat{y})\,\hat{y}(1-\hat{y})\,x$$

Case 1

Now ATQ, let's say $y = 1$ and $\hat{y} \approx 0$

$$\Rightarrow \lim_{\hat{y}\to 0} f'(x) = \lim_{\hat{y}\to 0} (-2(y-\hat{y})\hat{y}(1-\hat{y})x) = 0 \quad \text{— (1)}$$

$$= \lim_{\hat{y}\to 0} -2y(0)x = 0$$

Case 2:

Let's take $y = 0$, $\hat{y} \approx 1$

$$\Rightarrow \lim_{\hat{y}\to 1} f'(x) = \lim_{\hat{y}\to 1} (-2(y-\hat{y})\hat{y}(1-\hat{y})x)$$

$$= 0 \quad \text{— (2)}$$

i.e. gradient of gradient f$^n$ is 0.

However now if we analyse this,

$\frac{\partial J}{\partial \theta} = 0$ for MSE for logistic region for this data point of misclassification.

i.e. for predicting wrong values $\frac{\partial J}{\partial \theta} = 0$ —①

⇒ in ~~the~~ ~~Hard~~ the gradient descent iteration,

$$\theta_j := \theta_j - \alpha \frac{\partial J}{\partial \theta}$$

from ①

$\theta_j := \theta_j$

i.e. for misclassification, ideally we should give harsh penalty to our model and minimise $\theta$ for this scenario, we don't subtract the gradient term (now 0) from theta, so our algorithm is not optimised well for this situation and is not going to learn our model effectively.

Meanwhile for our cross entropy loss f$^n$:

$$\text{Cost} = -\left[y \log(p) + (1-y)\log(1-p)\right]$$

Let's say $y = 1$

$\hat{y} \approx 0$ ⇒

$\text{cost} = -y \log(\approx 0)$ : $= -y \log(\text{no. very close to } 0)$

$= -y(\text{no. very close to } -\infty)$

$=$ no. closer to infinity

i.e. for misclassification example, the cross entropy loss function gives a penalty tending to $\infty$ (for perfect misclassification), which is great for our algorithm because it penalizes misclassification and for perfect classification (let's say $y = 1$, $\hat{y} \approx 1$)

$$\begin{aligned} \text{Cost} &= y \log p \\ &= 1 \log(\approx 1) \\ &= 1 \times (\approx 0) \\ &= \approx 0 \end{aligned}$$

it doesn't penalize our algorithm and takes it to the optimum parameters. Hence, cross entropy loss is a better option.

Q4. 33 rows from our dataset

X Matrix → $\begin{bmatrix} 1 & x1 & x2 \\ \vdots & \vdots & \vdots \end{bmatrix}$ (x1 = price, x2 = age)

Y matrix $\begin{bmatrix} 0/1 \\ \vdots \end{bmatrix}$

- Perform 7 : 1 : 2 split for Train validation and Test

a) • Fit my model for logistic regression (log likelihood)

$\beta_0 = -4.363$
$\beta_1 = 0.064$
$\beta_2 = 0.520$

b) response $f = \frac{1}{1+e^{-\beta_0 + \beta_1 x_1 + \beta_2 x_2}}$

c) for $\exp(\beta 1) = e^{0.064} = 1.066609$
$\exp(\beta 2) = e^{0.520} = 1.682027$ times

The term $\exp(\beta 1)$ is the odds ratio of "extent to which disease has occurred %", i.e. for unit increase in %, odds for disease occurrence increases by 1.066 times.
The term $\exp(\beta 2)$ is the odds ratio for "age of patient", for every increase in year, odds for diseases occurence increases by 1.682 times

d) for given q,
% = 75%
age = 2 years

$\Rightarrow T_q = \beta_0 + \beta_1(75) + \beta_2(2)$

$T_q = -4.363 + (0.064)(75) + (0.520)(2)$

$T = 10.4765$ 1.4938465

To find out probability,

$$h(\sigma) = \frac{1}{1+e^{-\tau}} = \frac{1}{1+e^{-1.4938965}}$$

probability for recurrence in 5 years = 0.81666 or 81.6%.

Q5.

$$y_i = \beta_1 + \beta_2 x_{2i} + \beta_3 x_{3i} + \cdots \beta_{ik} x_{ki} + \varepsilon_i$$

$$Y = X\beta + \varepsilon$$

$$Y = \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix} \quad X = \begin{pmatrix} 1 & x_{21} & \cdots & x_{k1} \\ \vdots & & & \vdots \\ 1 & x_{2n} & \cdots & x_{kn} \end{pmatrix} \quad \beta = \begin{pmatrix} \beta_1 \\ \vdots \\ \beta_k \end{pmatrix} \quad \varepsilon = \begin{pmatrix} \varepsilon_1 \\ \vdots \\ \varepsilon_n \end{pmatrix}$$

for matrix expression of squared error losses,

$$\varepsilon = Y - X\beta$$

$$\begin{bmatrix} \varepsilon_1 & \varepsilon_2 & \varepsilon_3 & \cdots & \varepsilon_n \end{bmatrix} \times \begin{bmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \varepsilon_3 \\ \vdots \\ \varepsilon_n \end{bmatrix} = \begin{bmatrix} \varepsilon_1^2 & \varepsilon_2^2 & \varepsilon_3^2 \end{bmatrix}$$

$$\Rightarrow \varepsilon'\varepsilon = (y - X\beta)'(y - X\beta)$$

Transpose of scalar = scalar

$$= y'y - \beta' X' y - y' X \beta + \beta' X' X \beta$$

$$\boxed{\varepsilon'\varepsilon = y'y - 2\beta' X' y + \beta' X' X \beta} \quad \text{—— (1)}$$

To find $\beta^*$ that minimises squared errors loss →

differentiate eq$^n$ (1) wrt $\beta^*$

$$\frac{\partial(\varepsilon'\varepsilon)}{\partial\beta} = 0 - 2X'Y + 2X'X\beta^* = 0$$

$$\Rightarrow 2X'Y = 2X'X\beta^*$$

$$\boxed{\beta^* = (X'X)^{-1}X'Y}$$

↳ for this to exist,

$(X'X)^{-1}$ has to exist

i.e. $(X'X)^*$ needs to be ~~[illegible]~~ invertible

and non-singular