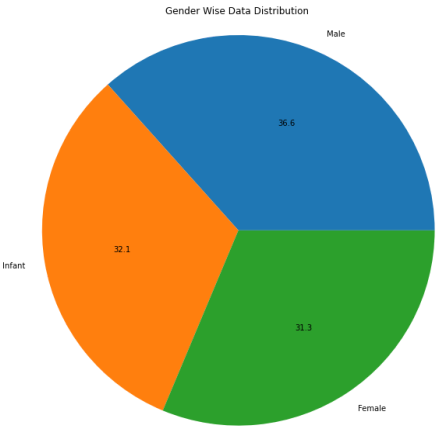
CSE 343 Machine Learning Assignment-1 Report

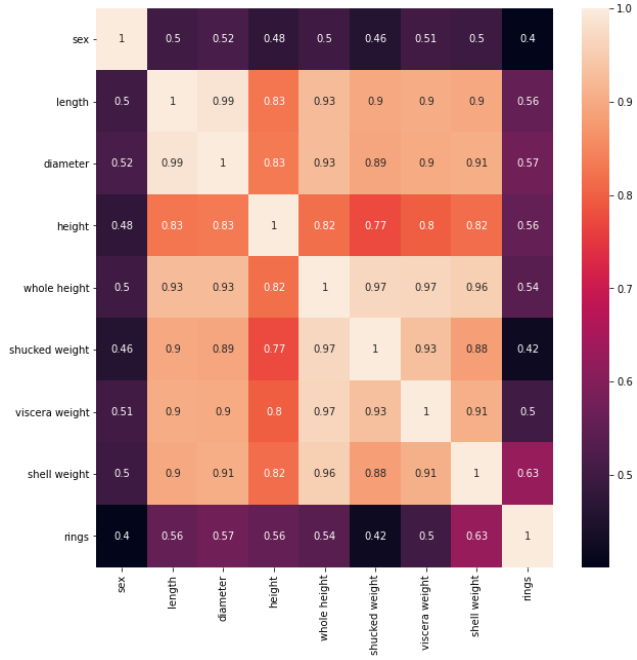
By: Hardik Garg, 2019040

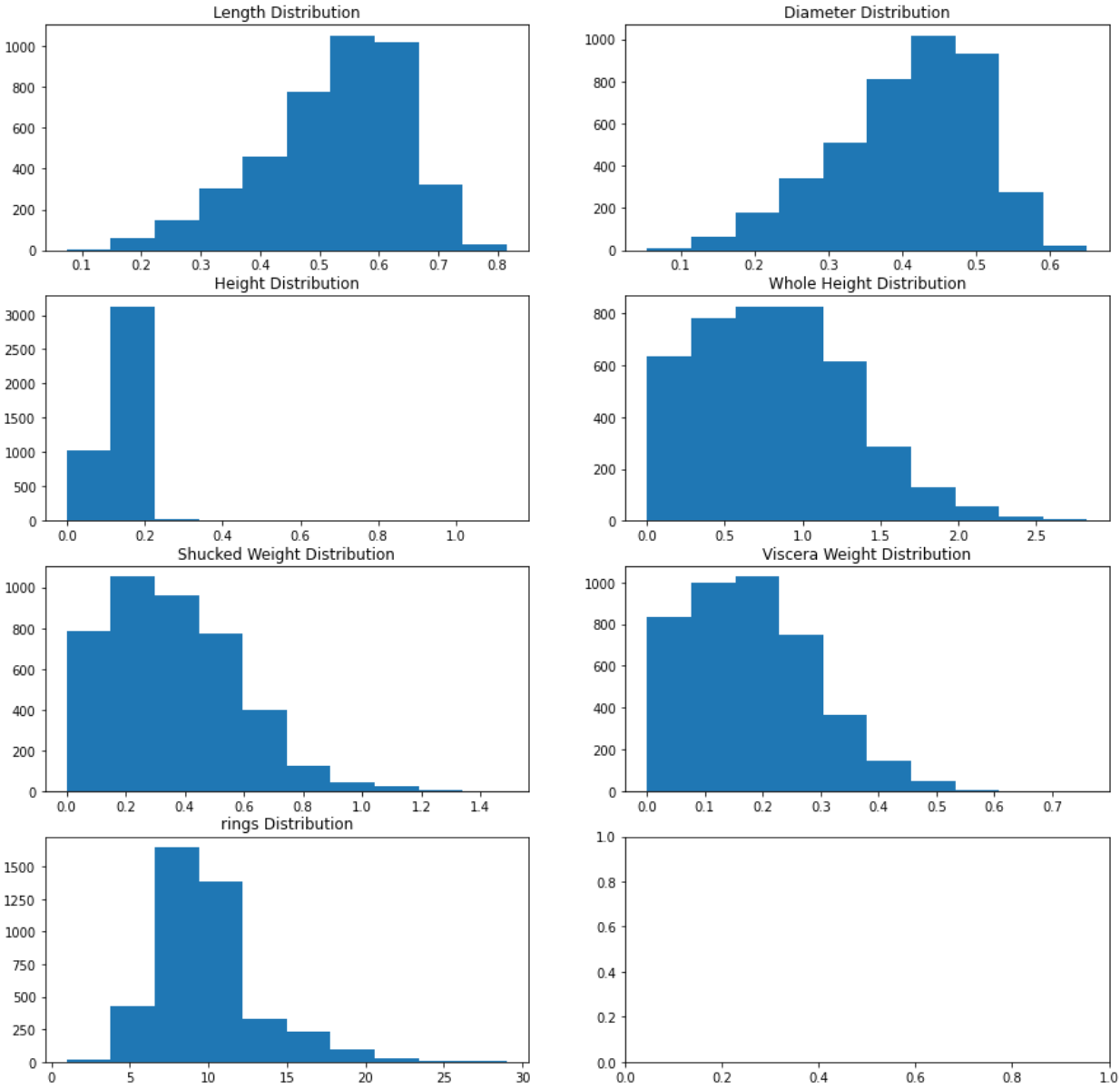
Q1

Preprocessing and EDA

1. Genders were mapped to 0,1,2 for ease of use with the model as numeric features
2. Data is already normalized so no need to further normalize it
3. Following graphs were plotted as part of EDA -
   i. Gender-wise data distribution
   ii. Relation of rings to all features - many of the features show a normal distribution trend
   iii. Correlation matrix - many features have a strong correlation (0.9 or so) with one another which indicates we can drop such features (in case we do feature engineering; not done for this question as the aim was to implement Linear Regression)







1.1

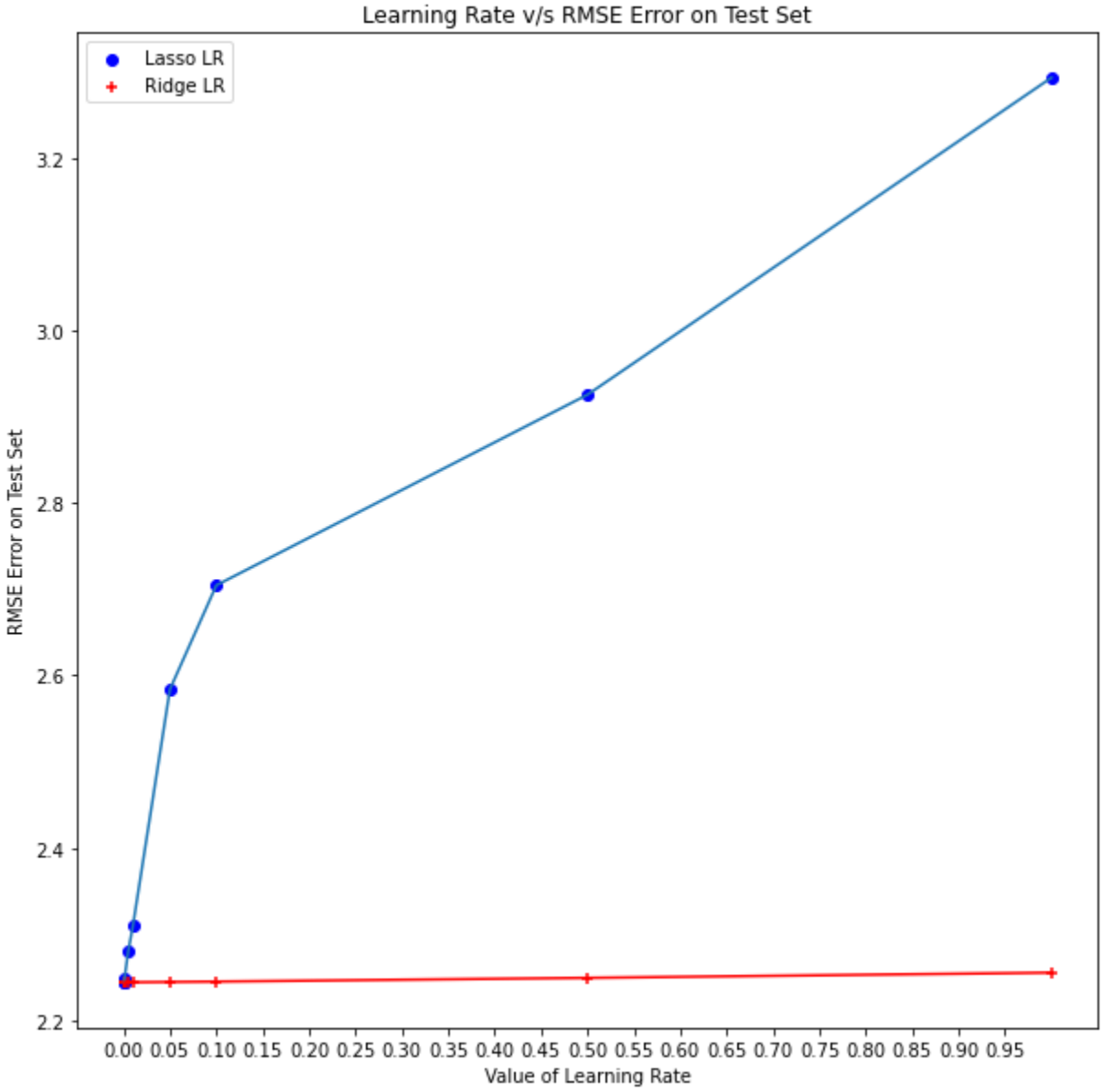
RMSE train: 2.4407432511343474

RMSE test: 2.4166977894032393

with learning rate = 0.25 and 200 iterations

1.2

(a) The graph for different learning rates with Ridge and Lasso Regressions is as follows -



On observation by eye, the best alpha values are as follows -

1. For Ridge LR = 0.05 or 0.5 (although all values are very close and differ in hundredths of place)

2. For Lasso LR = 0.0001 or 0.001 (1e-5, 1e-4, 1e-3 are all very close to zero and have nearly the same error)

Above values are observation by eye, for more clarity, we do grid search in the next part to verify our claims

Note: From theoretical knowledge we know that Learning Rate is directly proportional to loss. A very less learning rate will give us a very less loss, the only catch here is that as learning rate reduces, the time (or iterations) taken to converge will increase substantially, therefore, for practical applications we take a learning rate which has reasonable error and takes reasonable time to run. Theoretically, smaller value of learning rate is the best to minimise loss but more practical values are written above.

(b) Best model coefficients found using Grid Search are as follows -
Best alpha value for Ridge LR: 0.5
Best alpha value for Lasso LR: 0.001

```
Ridge Regression model coefficients:
[  0.34996087   2.23425656   8.0648696    8.4862047    7.92621112
 -18.15111758  -8.73940904   9.5098849    0.        ]
Lasso Regression model coefficients:
[0.01396169 0.         0.         0.         0.         0.
 0.         0.         0.        ]
Self Implemented model coefficeints:
[ 0.52995808  3.13446505  2.89332357  1.6309801   2.59116313 -4.65847809
 -0.19015229  4.50163776  4.8445348 ]
```

Comparison with part (a) - Both values are very close to what was predicted in the previous part from the graph. Grid search gives more practical learning rates which also take into account the rate of convergence. Also the value of model coefficients is very different for all the 3 models.

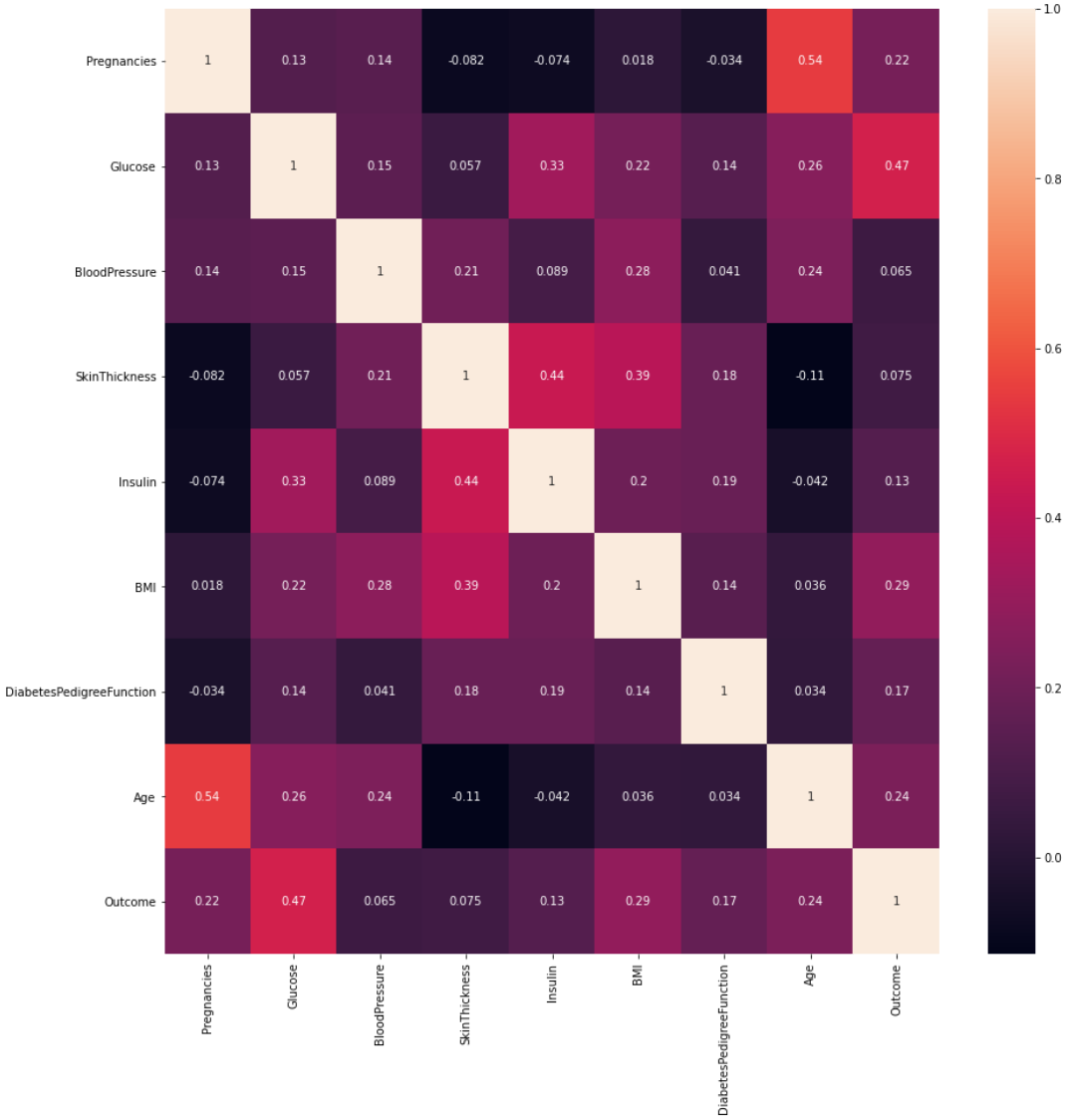
Also note that RMSE for Ridge LR increases sharply on increasing alpha whereas RMSE remains more or less the same for Lasso LR on increasing alpha. Also, the best alpha values can differ depending on the range of learning rates the Grid Search is performed upon. Here, the range of learning rates is - [1e-5,5e-5,1e-4,5e-4,1e-3,5e-3,1e-2,5e-2,1e-1,5e-1,1]

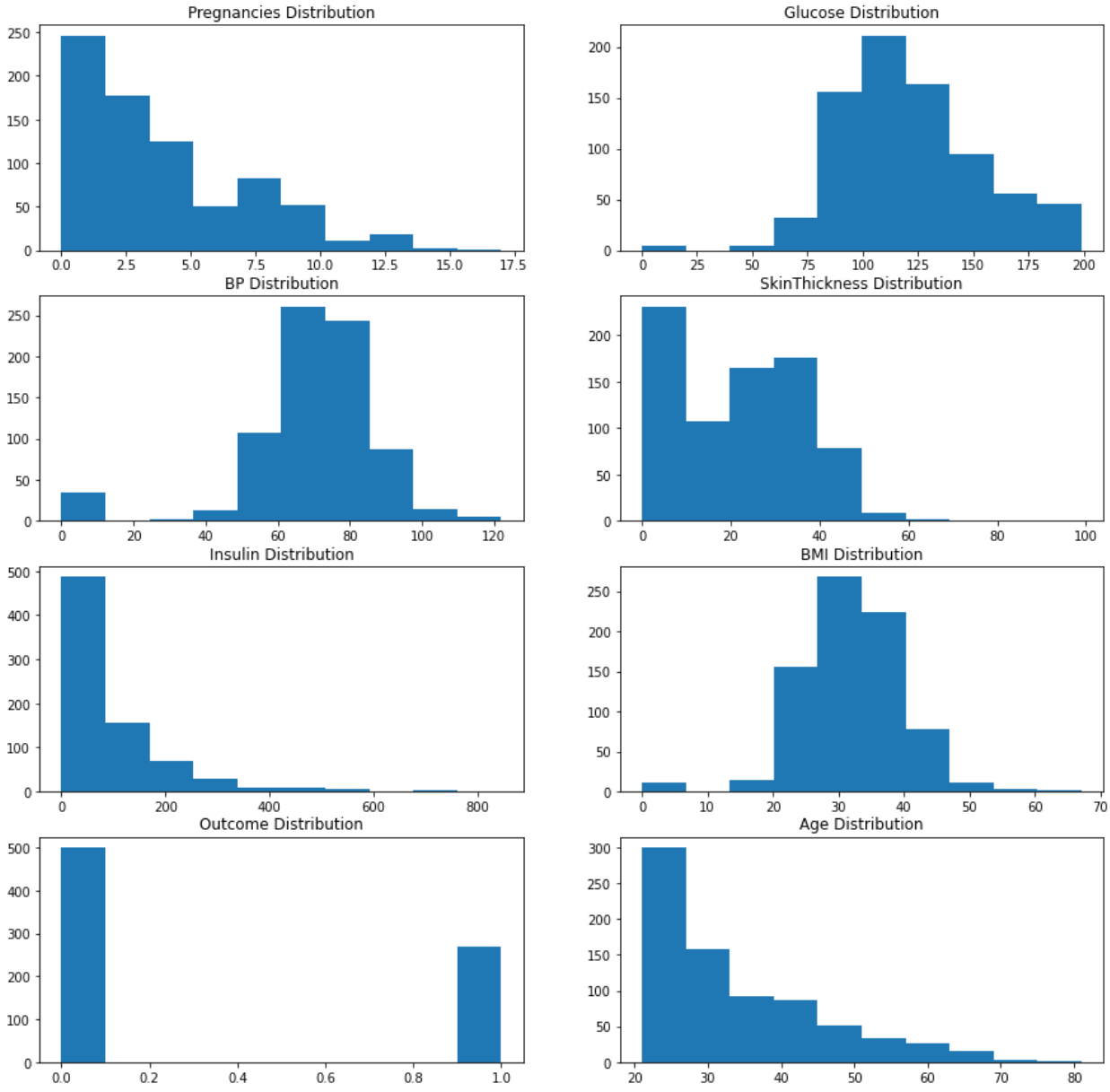
Q2

Preprocessing and EDA

1. Data was normalized using Standard Scaler
2. Following plots were made -
   i. Plots between every feature and final outcome
   ii. Correlation matrix - in this we observe that there is no strong correlation between any of the features which is also an indicator to use logistic regression because of its nonlinear sigmoid function. Also, features like skin thickness and age follow a normal distribution which is a good sign.

   The graphs plotted are shown below -


Pregnancies 1 0.13 0.14 -0.082 -0.074 0.018 -0.034 0.54 0.22
Glucose 0.13 1 0.15 0.057 0.33 0.22 0.14 0.26 0.47
BloodPressure 0.14 0.15 1 0.21 0.089 0.28 0.041 0.24 0.065
SkinThickness -0.082 0.057 0.21 1 0.44 0.39 0.18 -0.11 0.075
Insulin -0.074 0.33 0.089 0.44 1 0.2 0.19 -0.042 0.13
BMI 0.018 0.22 0.28 0.39 0.2 1 0.14 0.036 0.29
DiabetesPedigreeFunction -0.034 0.14 0.041 0.18 0.19 0.14 1 0.034 0.17
Age 0.54 0.26 0.24 -0.11 -0.042 0.036 0.034 1 0.24
Outcome 0.22 0.47 0.065 0.075 0.13 0.29 0.17 0.24 1
Pregnancies
Glucose
BloodPressure
SkinThickness
Insulin
BMI
DiabetesPedigreeFunction
Age
Outcome
1.0
0.8
0.6
0.4
0.2
0.0



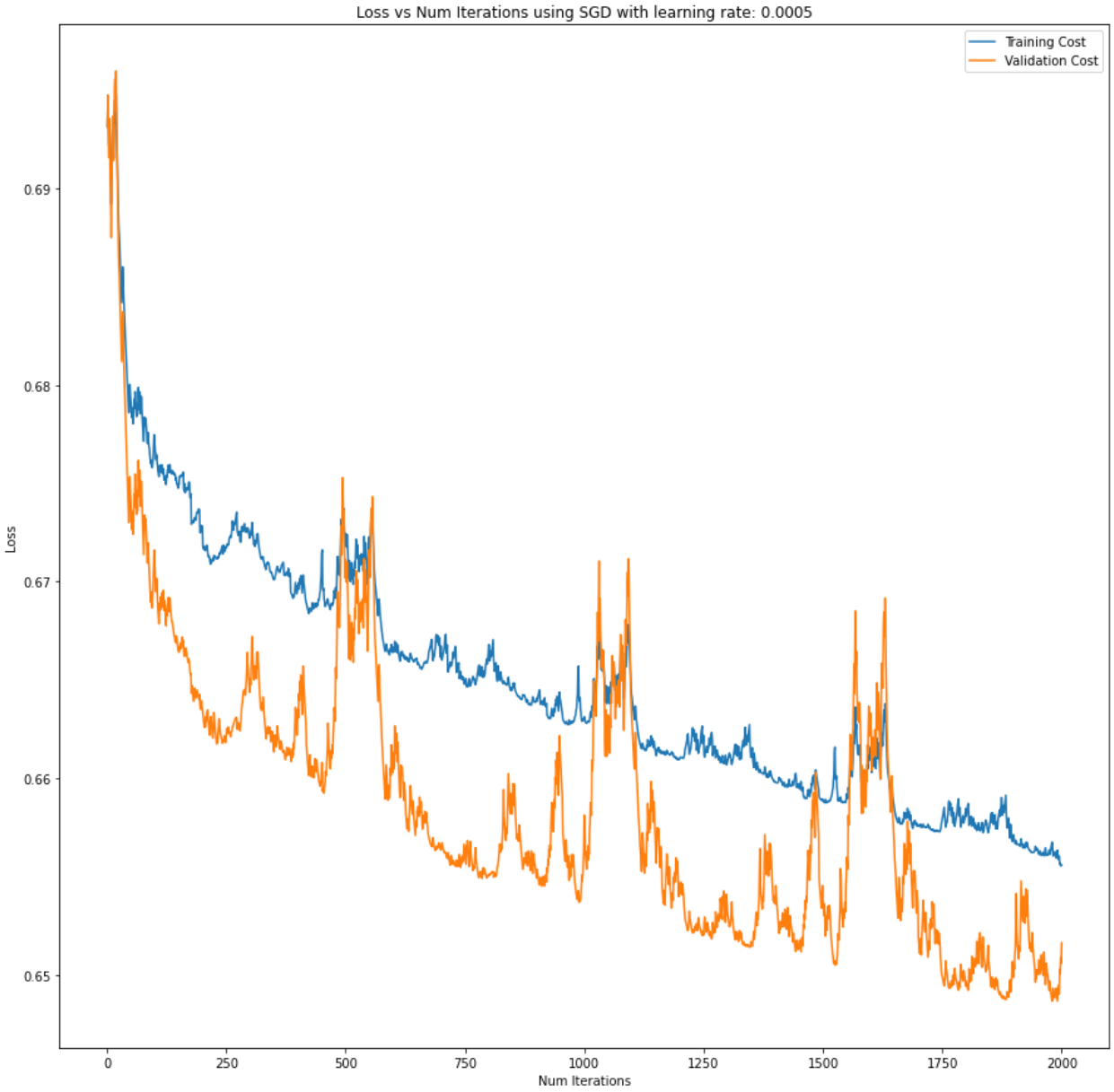
2.1

(a)

Note: For the loss vs iteration plots, a limited number of iterations were considered because beyond that, SGD plot became very noisy and resembled a shaded region which led to no useful inferences.

Parameters used: SGD - alpha = 0.0005, BGD - alpha = 0.00025

The loss vs iteration plots are as follows -

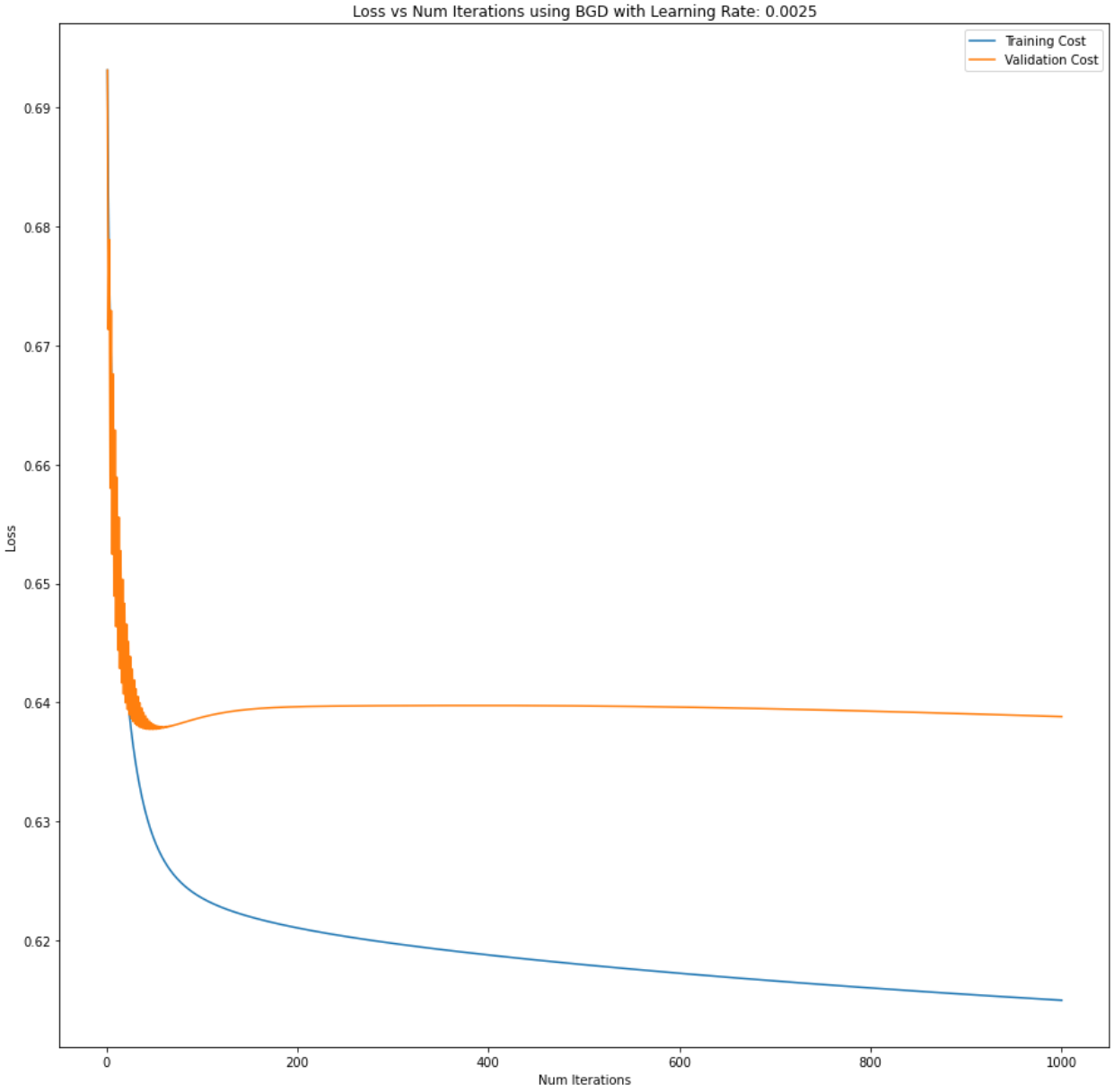


Observation - For SGD, we observe a noisy convergence for both train and validation sets. This is because the gradient is updated for every training point so the curve reaches a minimum value gradually but with a lot of fluctuations. The final scores are as follows -

Train Score: 0.6871508379888268

Val Score: 0.6688311688311688

Test Score: 0.7012987012987013



Observation - For BGD, we observe a smooth curve for both training and validation sets. This is because gradient is updated once after iterating over the entire training set. The final scores are as follows -

i. Train Score: 0.6815642458100558

ii. Val Score: 0.6818181818181818

iii. Test Score: 0.7272727272727273

(b)

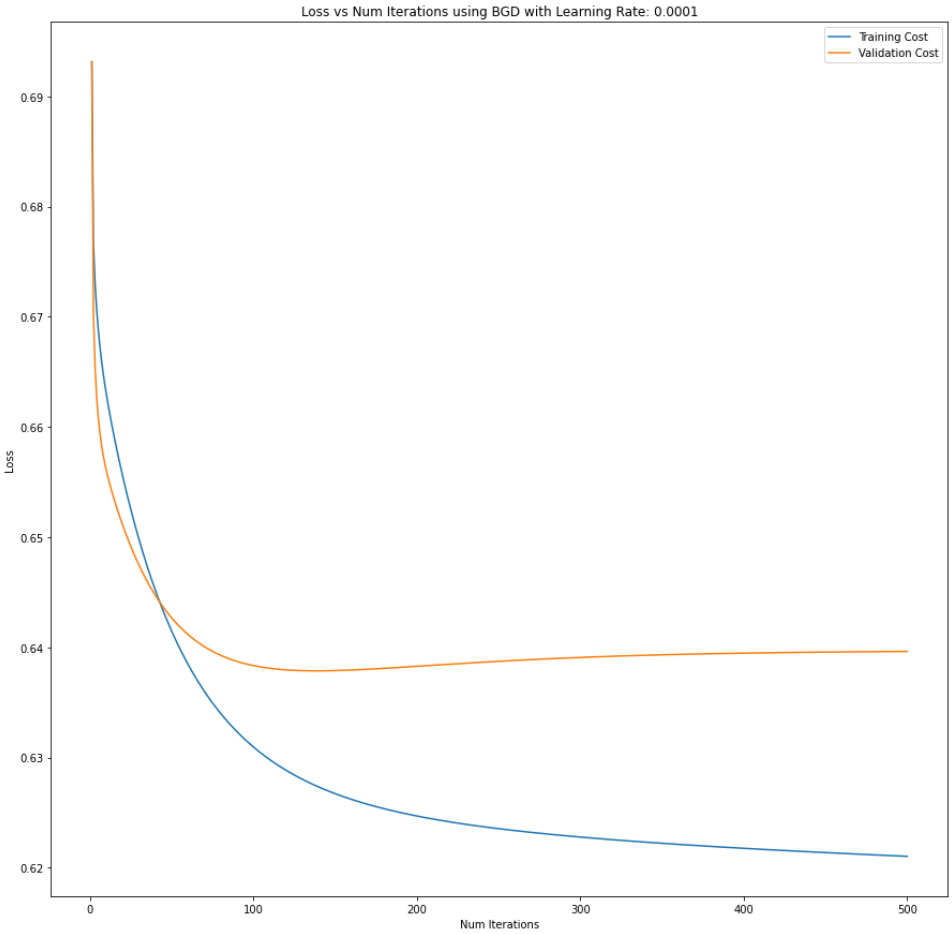
For BGD

1. **LR=0.0001**

   Train Score: 0.6741154562383612

   Val Score: 0.6688311688311688

   Test Score: 0.7532467532467533



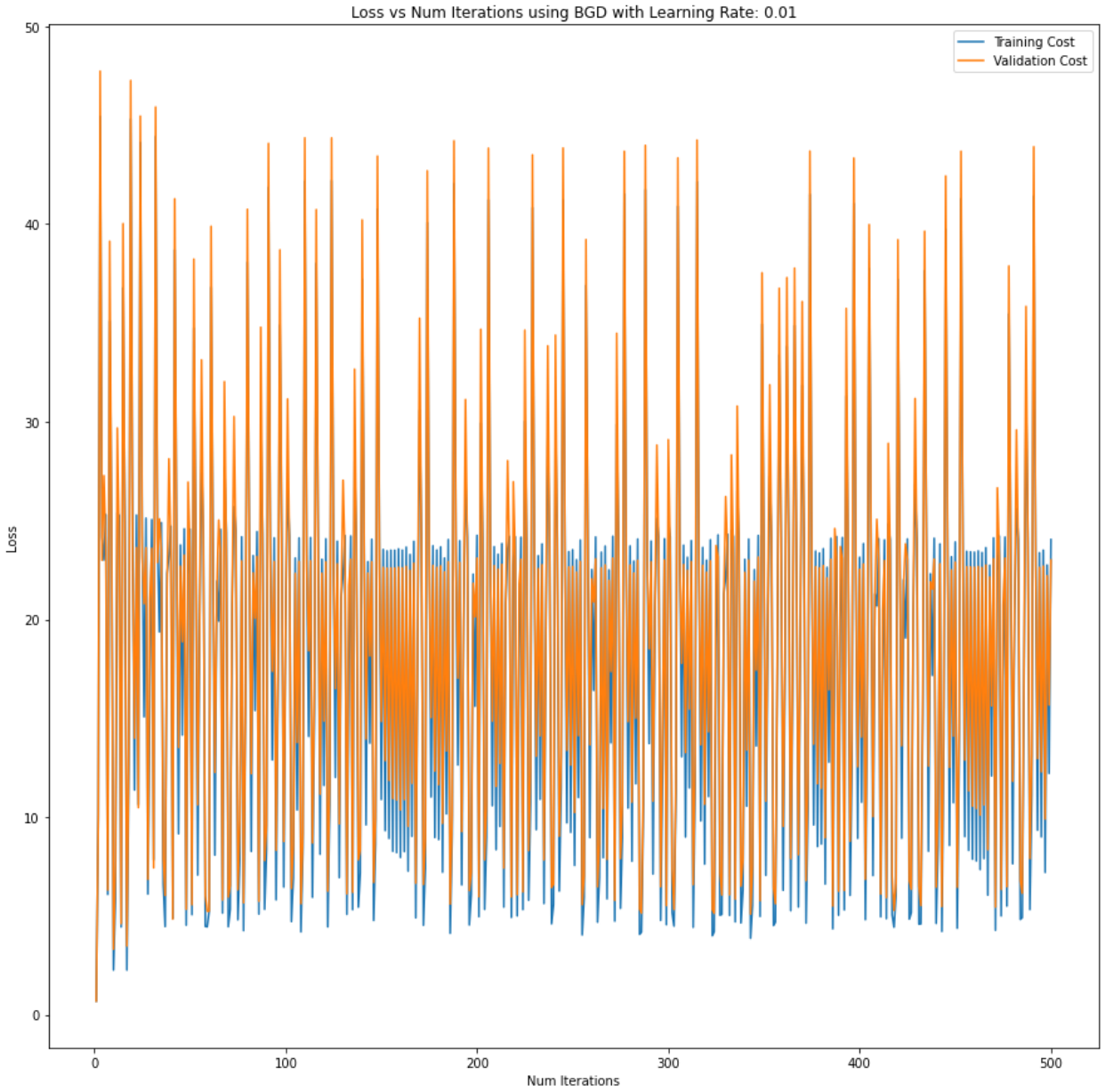
Observation - smooth loss vs iterations curve. Train, Val, Test scores are comparable with each other.

2. **LR = 0.01**

Train Score: 0.6945996275605214

Val Score: 0.6883116883116883

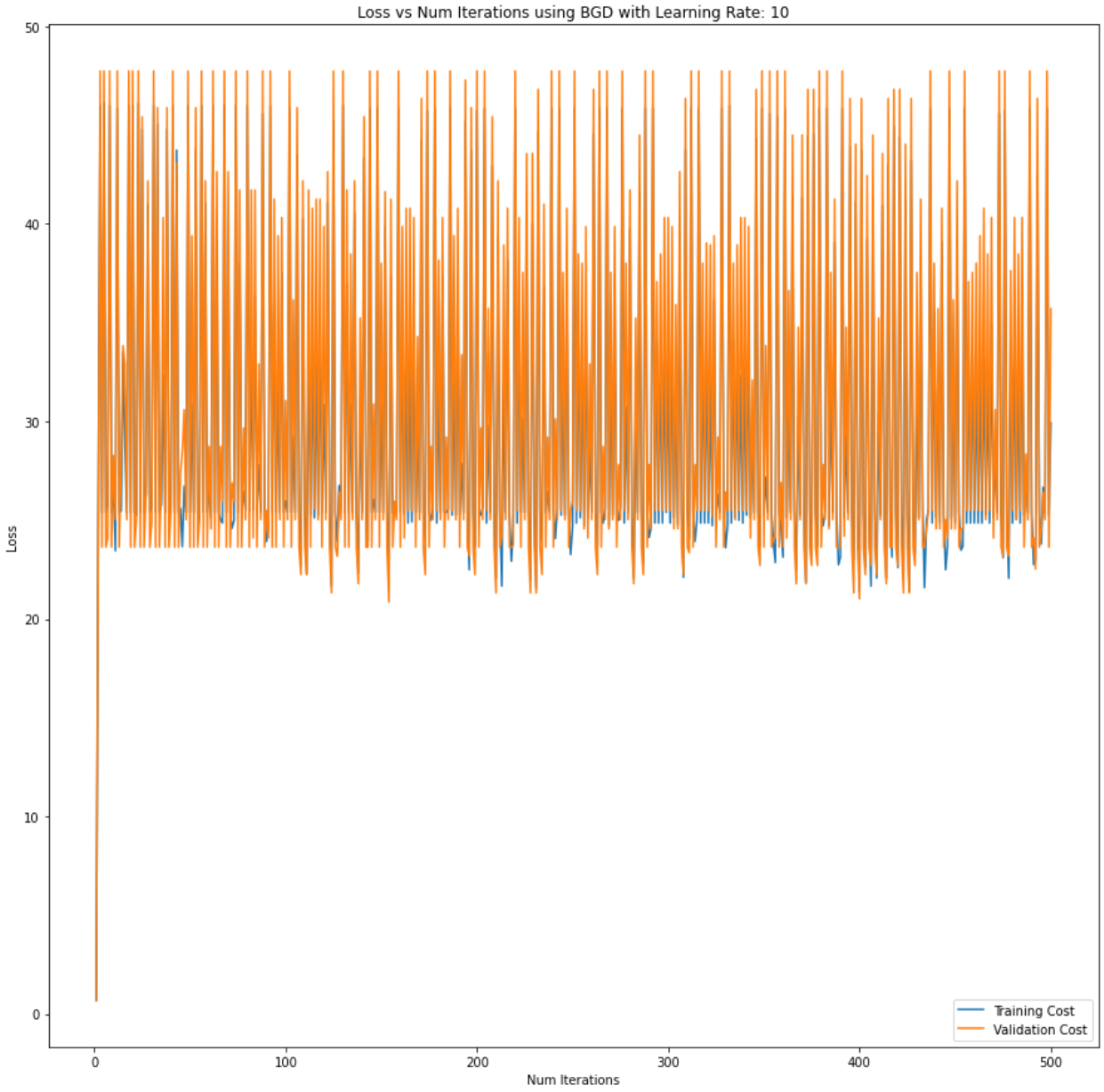
Test Score: 0.7012987012987013



Observation - A noise curve, indicating that gradient changes too rapidly. This is because of a high learning rate. Also observe the dropping Test score. Train score increased because of overfitting.

3. **LR = 10**

Train Score: 0.6461824953445066
Val Score: 0.6493506493506493
Test Score: 0.6623376623376623



Observation - Again a very high learning rate due to which the loss plot is very noisy. Also note the dropping Train, Test, Val scores. In case of high learning rates, the gradient changes very rapidly from very small to very large values and hence the fluctuation.

For SGD

1. **LR = 0.0001**

   Train Score: 0.6685288640595903

   Val Score: 0.6818181818181818

   Test Score: 0.7532467532467533

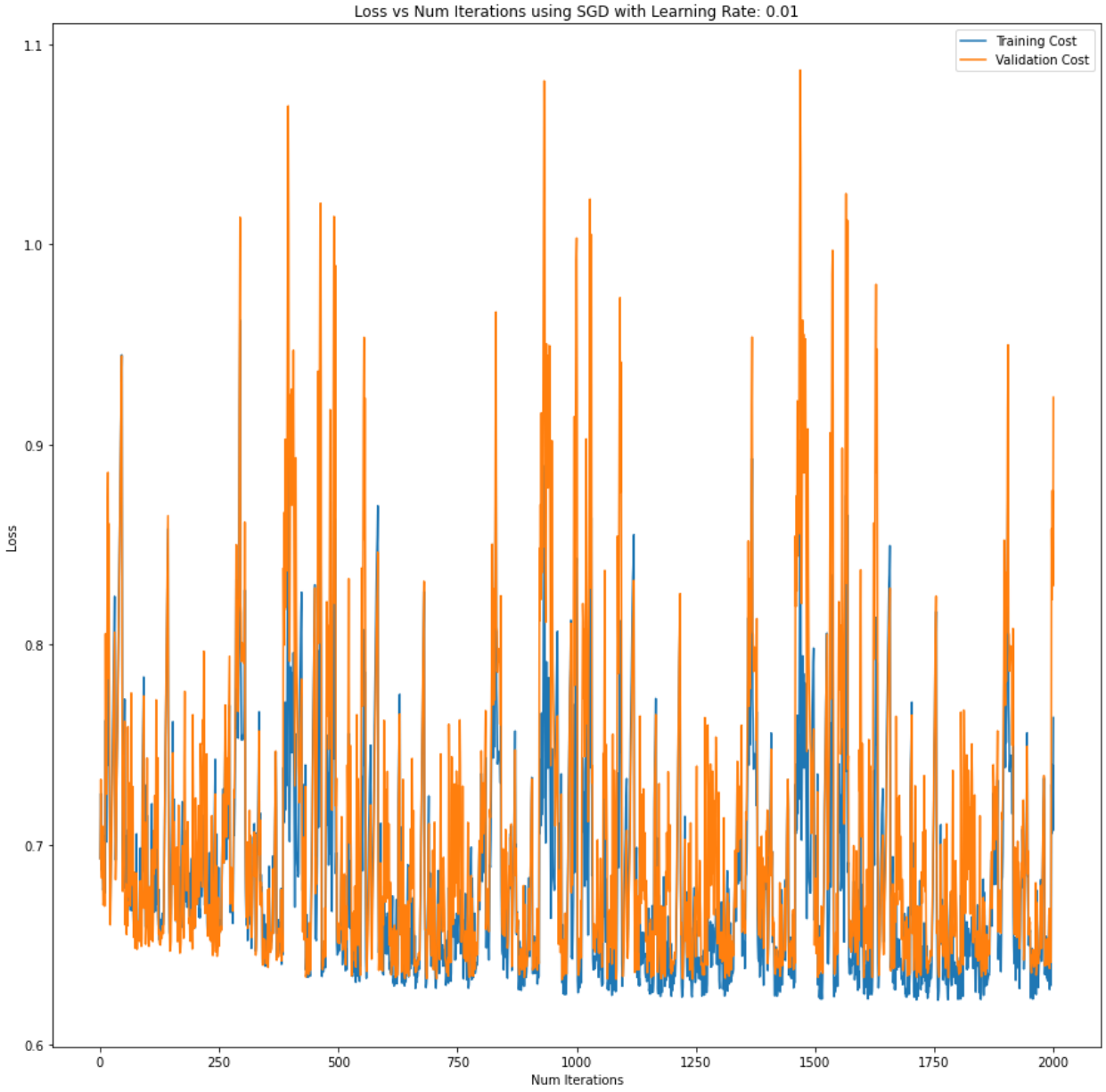
Observation - The graph is a typical noisy SGD curve, indicating 0.0001 to be an ideal choice for Learning Rate.

2. **LR = 0.01**

Train Score: 0.6554934823091247

Val Score: 0.6363636363636364

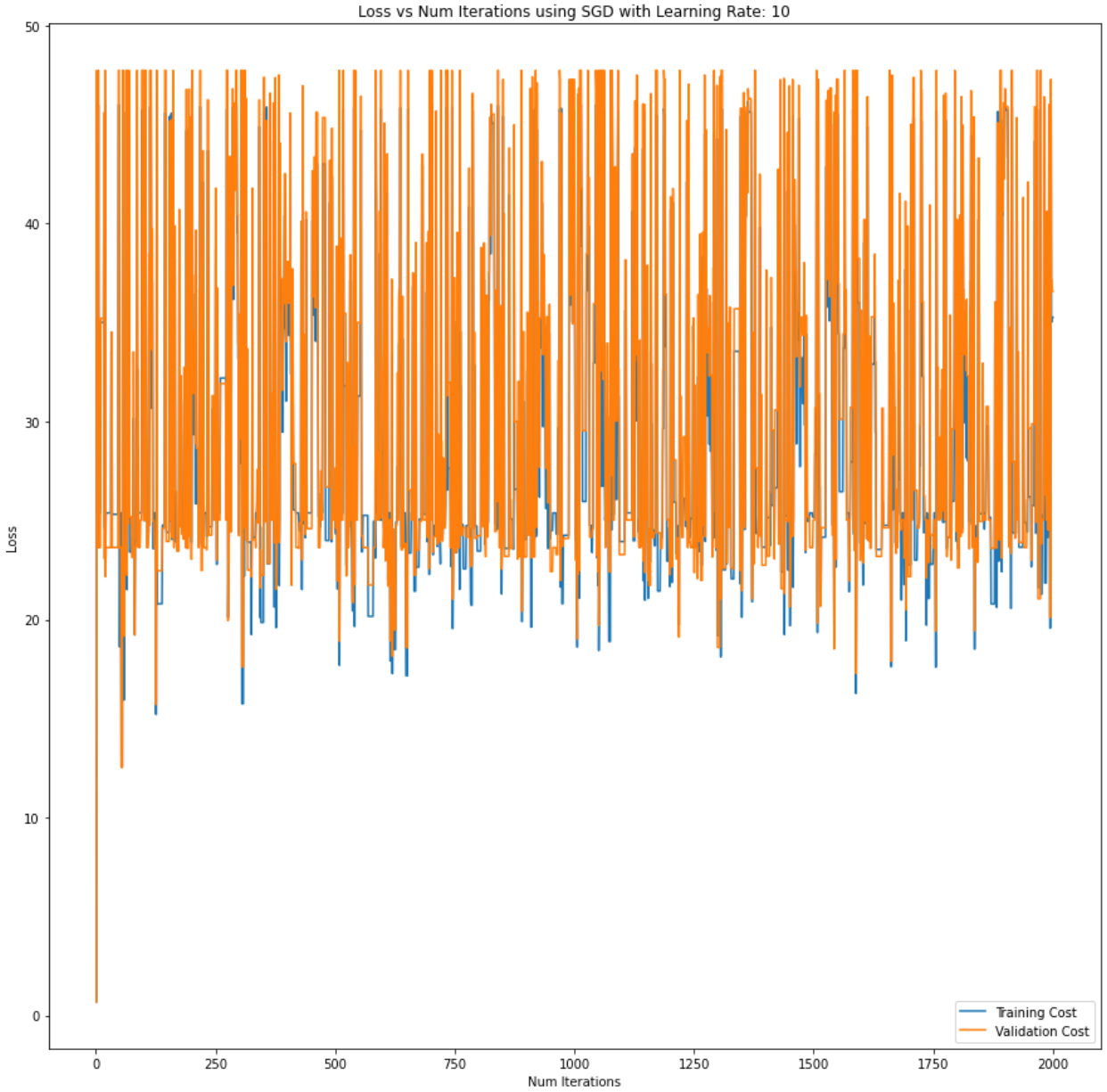
Test Score: 0.6883116883116883



Observation - A very noisy curve because of a high learning rate, leading to rapid changes in gradient. Also note the dropping train test val scores.

3. **LR = 10**

Train Score: 0.6666666666666666
Val Score: 0.6493506493506493
Test Score: 0.6493506493506493



Observation - Very noisy curve with higher loss than before. Note the dropped train test val scores compared to last time.

(c)

1. BGD (alpha=0.00025)

```
Train Score: 0.6815642458100558 Val Score: 0.6818181818181818 Test Score: 0.7272727272727273
y_train
Confusion Matrix (TN,FP,TP,FN) clockwise from top-left:
 [[308  38]
 [133  58]]

Precision Recall F1-Score (1st row for label 0, 2nd row for label 1):
 [[0.6984127  0.89017341 0.78271919]
 [0.60416667 0.30366492 0.40418118]]

y_val
Confusion Matrix (TN,FP,TP,FN) clockwise from top-left:
 [[88 15]
 [34 17]]

Precision Recall F1-Score (1st row for label 0, 2nd row for label 1):
 [[0.72131148 0.85436893 0.78222222]
 [0.53125    0.33333333 0.40963855]]

y_test
Confusion Matrix (TN,FP,TP,FN) clockwise from top-left:
 [[44  7]
 [14 12]]

Precision Recall F1-Score (1st row for label 0, 2nd row for label 1):
 [[0.75862069 0.8627451  0.80733945]
 [0.63157895 0.46153846 0.53333333]]
```

2. BGD(alpha=0.0001)

```
Train Score: 0.6741154562383612 Val Score: 0.6688311688311688 Test Score: 0.7532467532467533
y_train
Confusion Matrix (TN,FP,TP,FN) clockwise from top-left:
 [[306  40]
 [135  56]]

Precision Recall F1-Score (1st row for label 0, 2nd row for label 1):
 [[0.69387755 0.88439306 0.77763659]
 [0.58333333 0.29319372 0.3902439 ]]

y_val
Confusion Matrix (TN,FP,TP,FN) clockwise from top-left:
 [[86 17]
 [34 17]]

Precision Recall F1-Score (1st row for label 0, 2nd row for label 1):
 [[0.71666667 0.83495146 0.77130045]
 [0.5        0.33333333 0.4       ]]

y_test
Confusion Matrix (TN,FP,TP,FN) clockwise from top-left:
 [[45  6]
 [13 13]]

Precision Recall F1-Score (1st row for label 0, 2nd row for label 1):
 [[0.77586207 0.88235294 0.82568807]
 [0.68421053 0.5        0.57777778]]
```

3. BGD(alpha=0.01)

```
Train Score: 0.6945996275605214 Val Score: 0.6883116883116883 Test Score: 0.7012987012987013
y_train
Confusion Matrix (TN,FP,TP,FN) clockwise from top-left:
 [[284  62]
 [102  89]]

Precision Recall F1-Score (1st row for label 0, 2nd row for label 1):
 [[0.7357513  0.82080925 0.77595628]
 [0.58940397 0.46596859 0.52046784]]

y_val
Confusion Matrix (TN,FP,TP,FN) clockwise from top-left:
 [[83 20]
 [28 23]]

Precision Recall F1-Score (1st row for label 0, 2nd row for label 1):
 [[0.74774775 0.80582524 0.77570093]
 [0.53488372 0.45098039 0.4893617 ]]

y_test
Confusion Matrix (TN,FP,TP,FN) clockwise from top-left:
 [[40 11]
 [12 14]]

Precision Recall F1-Score (1st row for label 0, 2nd row for label 1):
 [[0.76923077 0.78431373 0.77669903]
 [0.56       0.53846154 0.54901961]]
```

4. BGD(alpha=10)

```
Train Score: 0.6461824953445066 Val Score: 0.6493506493506493 Test Score: 0.6623376623376623
y_train
Confusion Matrix (TN,FP,TP,FN) clockwise from top-left:
 [[335  11]
 [179  12]]

Precision Recall F1-Score (1st row for label 0, 2nd row for label 1):
 [[0.65175097 0.96820809 0.77906977]
 [0.52173913 0.06282723 0.11214953]]

y_val
Confusion Matrix (TN,FP,TP,FN) clockwise from top-left:
 [[98  5]
 [49  2]]

Precision Recall F1-Score (1st row for label 0, 2nd row for label 1):
 [[0.66666667 0.95145631 0.784     ]
 [0.28571429 0.03921569 0.06896552]]

y_test
Confusion Matrix (TN,FP,TP,FN) clockwise from top-left:
 [[49  2]
 [24  2]]

Precision Recall F1-Score (1st row for label 0, 2nd row for label 1):
 [[0.67123288 0.96078431 0.79032258]
 [0.5        0.07692308 0.13333333]]
```

5. SGD (alpha=0.0005)

```
Train Score: 0.6871508379888268 Val Score: 0.6688311688311688 Test Score: 0.7012987012987013
y_train
Confusion Matrix (TN,FP,TP,FN) clockwise from top-left:
 [[288  58]
 [110  81]]

Precision Recall F1-Score (1st row for label 0, 2nd row for label 1):
 [[0.72361809 0.83236994 0.77419355]
 [0.58273381 0.42408377 0.49090909]]

y_val
Confusion Matrix (TN,FP,TP,FN) clockwise from top-left:
 [[80 23]
 [28 23]]

Precision Recall F1-Score (1st row for label 0, 2nd row for label 1):
 [[0.74074074 0.77669903 0.75829384]
 [0.5        0.45098039 0.4742268 ]]

y_test
Confusion Matrix (TN,FP,TP,FN) clockwise from top-left:
 [[39 12]
 [11 15]]

Precision Recall F1-Score (1st row for label 0, 2nd row for label 1):
 [[0.78       0.76470588 0.77227723]
 [0.55555556 0.57692308 0.56603774]]
```

6. SGD(alpha=0.0001)

```
Train Score: 0.6685288640595903 Val Score: 0.6818181818181818 Test Score: 0.7532467532467533
y_train
Confusion Matrix (TN,FP,TP,FN) clockwise from top-left:
 [[302  44]
 [134  57]]

Precision Recall F1-Score (1st row for label 0, 2nd row for label 1):
 [[0.69266055 0.87283237 0.77237852]
 [0.56435644 0.29842932 0.39041096]]

y_val
Confusion Matrix (TN,FP,TP,FN) clockwise from top-left:
 [[86 17]
 [32 19]]

Precision Recall F1-Score (1st row for label 0, 2nd row for label 1):
 [[0.72881356 0.83495146 0.77828054]
 [0.52777778 0.37254902 0.43678161]]

y_test
Confusion Matrix (TN,FP,TP,FN) clockwise from top-left:
 [[45  6]
 [13 13]]

Precision Recall F1-Score (1st row for label 0, 2nd row for label 1):
 [[0.77586207 0.88235294 0.82568807]
 [0.68421053 0.5        0.57777778]]
```

7. SGD(alpha=0.01)

```
Train Score: 0.6554934823091247 Val Score: 0.6363636363636364 Test Score: 0.6883116883116883
y_train
Confusion Matrix (TN,FP,TP,FN) clockwise from top-left:
 [[225 121]
 [ 64 127]]

Precision Recall F1-Score (1st row for label 0, 2nd row for label 1):
 [[0.77854671 0.65028902 0.70866142]
 [0.51209677 0.66492147 0.5785877 ]]

y_val
Confusion Matrix (TN,FP,TP,FN) clockwise from top-left:
 [[64 39]
 [17 34]]

Precision Recall F1-Score (1st row for label 0, 2nd row for label 1):
 [[0.79012346 0.62135922 0.69565217]
 [0.46575342 0.66666667 0.5483871 ]]

y_test
Confusion Matrix (TN,FP,TP,FN) clockwise from top-left:
 [[32 19]
 [ 5 21]]

Precision Recall F1-Score (1st row for label 0, 2nd row for label 1):
 [[0.86486486 0.62745098 0.72727273]
 [0.525      0.80769231 0.63636364]]
```

8. SGD(alpha=10)

```
Train Score: 0.6666666666666666 Val Score: 0.6493506493506493 Test Score: 0.6493506493506493
y_train
Confusion Matrix (TN,FP,TP,FN) clockwise from top-left:
 [[278  68]
 [111  80]]

Precision Recall F1-Score (1st row for label 0, 2nd row for label 1):
 [[0.71465296 0.80346821 0.75646259]
 [0.54054054 0.41884817 0.4719764 ]]

y_val
Confusion Matrix (TN,FP,TP,FN) clockwise from top-left:
 [[75 28]
 [26 25]]

Precision Recall F1-Score (1st row for label 0, 2nd row for label 1):
 [[0.74257426 0.72815534 0.73529412]
 [0.47169811 0.49019608 0.48076923]]

y_test
Confusion Matrix (TN,FP,TP,FN) clockwise from top-left:
 [[38 13]
 [14 12]]

Precision Recall F1-Score (1st row for label 0, 2nd row for label 1):
 [[0.73076923 0.74509804 0.73786408]
 [0.48       0.46153846 0.47058824]]
```

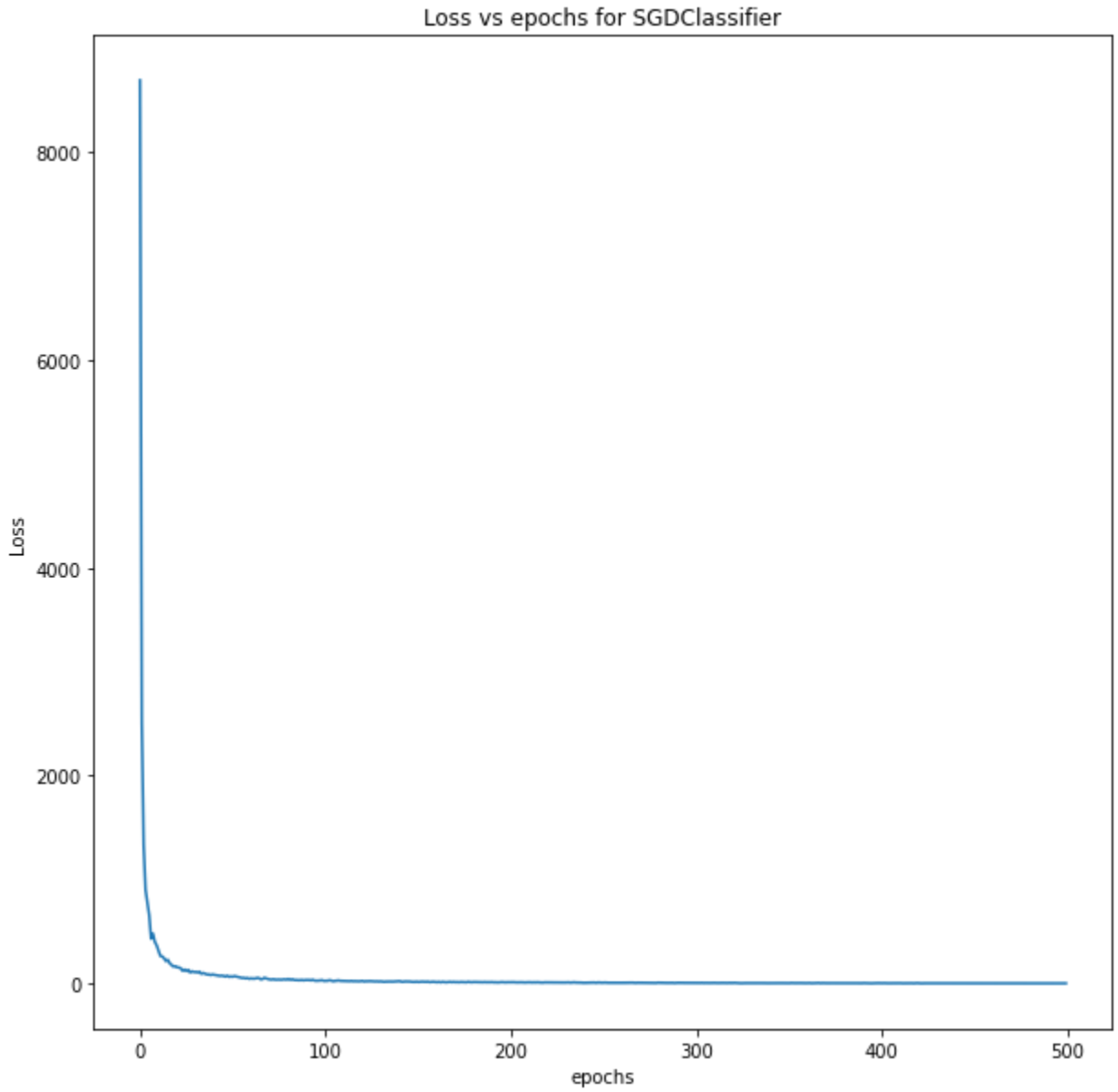
Observation and Analysis - In each of the cases (SGD and BGD), it is observed that -

1. As LR increases, accuracy/score decreases because of overshooting of gradients. This holds true for Train, Test, Val sets.
2. As LR increases, the imbalance in precision, recall and F-1 score increases. For example - instead of being 0.6 for both classes, the values are like 0.9 and 0.1 which indicates that the model is predicting most values as belonging to 1 which is not a good performance.

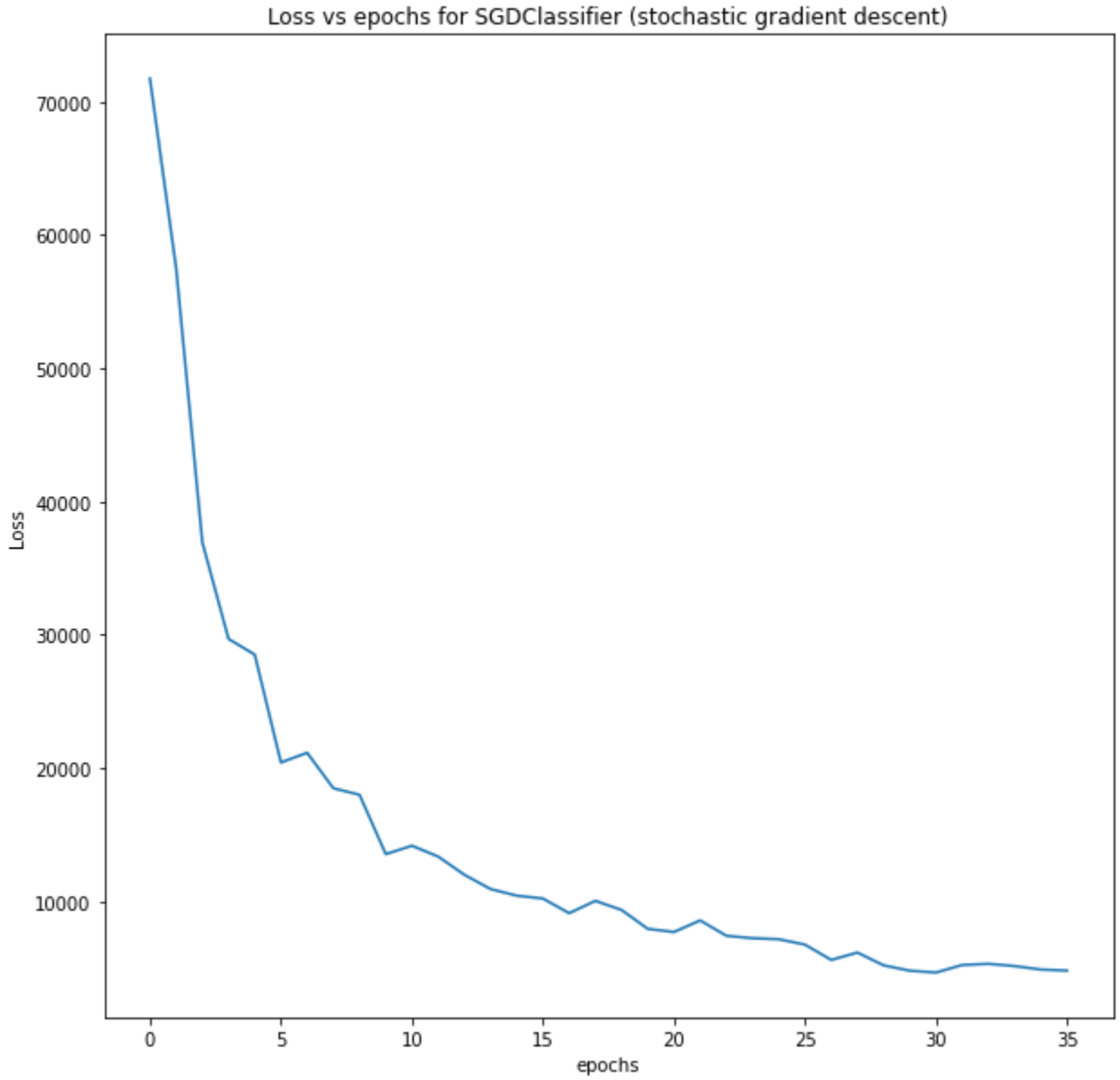
2.2

(a)

Loss plot using sklearn Batch Gradient Descent -



Loss using SGDClassifier



Comparison between Inbuilt and Self Implemented - Both the models show a decrease in the Loss. However, the inbuilt implementations showed a steadier decrease (smooth in BGD and noisy in SGD), but in case of inbuilt library, the decrease is more erratic. In inbuilt, there are also horizontal lines at places which indicates use of mini batch gradient descent.

(b) For self implemented, the model showed small amounts of decrease in losses even after 1000 iterations but the major decrease occurred in 500 iterations, after which the decrease was of the order 0.001; in the case if inbuilt library, as we observe from the graph, the loss becomes constant after around 35 iterations in case of stochastic gradient descent and 200 something for batch gradient descent

(c) Precision, Recall, F-1 score, accuracy and confusion matrix for inbuilt library -

1. Using Batch Gradient Descent

```
Train Score: 0.6461824953445066 Val Score: 0.6428571428571429 Test Score: 0.6363636363636364
y_train
Confusion Matrix (TN,FP,TP,FN) clockwise from top-left:
 [[199 147]
 [ 43 148]]

Precision Recall F1-Score (1st row for label 0, 2nd row for label 1):
 [[0.82231405 0.57514451 0.67687075]
 [0.50169492 0.77486911 0.6090535 ]]

y_val
Confusion Matrix (TN,FP,TP,FN) clockwise from top-left:
 [[62 41]
 [14 37]]

Precision Recall F1-Score (1st row for label 0, 2nd row for label 1):
 [[0.81578947 0.60194175 0.69273743]
 [0.47435897 0.7254902  0.57364341]]

y_test
Confusion Matrix (TN,FP,TP,FN) clockwise from top-left:
 [[28 23]
 [ 5 21]]

Precision Recall F1-Score (1st row for label 0, 2nd row for label 1):
 [[0.84848485 0.54901961 0.66666667]
 [0.47727273 0.80769231 0.6       ]]
```

2. Using SGDClassifier

```
Train Score: 0.6573556797020484 Val Score: 0.6298701298701299 Test Score: 0.7012987012987013
y_train
Confusion Matrix (TN,FP,TP,FN) clockwise from top-left:
 [[289  57]
 [127  64]]

Precision Recall F1-Score (1st row for label 0, 2nd row for label 1):
 [[0.69471154 0.83526012 0.75853018]
 [0.52892562 0.33507853 0.41025641]]

y_val
Confusion Matrix (TN,FP,TP,FN) clockwise from top-left:
 [[77 26]
 [31 20]]

Precision Recall F1-Score (1st row for label 0, 2nd row for label 1):
 [[0.71296296 0.74757282 0.72985782]
 [0.43478261 0.39215686 0.41237113]]

y_test
Confusion Matrix (TN,FP,TP,FN) clockwise from top-left:
 [[43  8]
 [15 11]]

Precision Recall F1-Score (1st row for label 0, 2nd row for label 1):
 [[0.74137931 0.84313725 0.78899083]
 [0.57894737 0.42307692 0.48888889]]
```

We can clearly observe that the inbuilt implementation performs better than the self implemented one in terms of accuracy/score and F-1 score by a significant margin (around 10% better). The precision and recall of inbuilt are slightly higher than self-implemented but they are still comparable. Possible reason for this is that inbuilt implementation uses advanced regularization algorithms such as *{'newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga'}* (from documentation) whereas in self-implementation, an approximation is being used to take care of overflows which arise in the absence of regularization.

Q3

Preprocessing and EDA

1. Only Trouser and Pullover Values were filtered from the dataset
2. Pixel values were rounded off to either 0 or 255 (binarization)
3. Correlation matrix between output and input features (frequency of 0 and 255) were plotted -

3.1 The classifier is implemented and explained in the jupyter notebook. The results obtained are as follows -

Train Score: 0.8714166666666666
Test Score: 0.8365

3.2 There are 12000 training + 2000 testing points = 14000 total points. The value chosen for the KFold cross validation is **k=4.** This is because the data has low variance and 4 is divisible by 14000 which makes perfect splits. Moreover, 3 was discussed as a reasonable value in lectures which performs well on large datasets, so in our case, we take 4 (to take care of the divisibility issues). Running with k=3 is also fine, in which case, one of the splits will have more data points. The results obtained are as follows -

Scores: [0.8537142857142858, 0.8565714285714285, 0.8568571428571429, 0.8531428571428571]
Avg Score: 0.8550714285714286

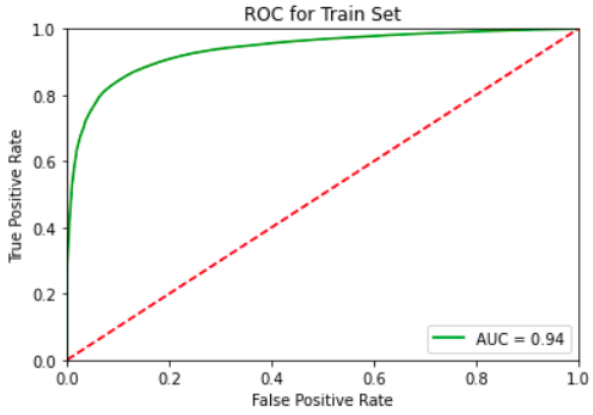
3.3

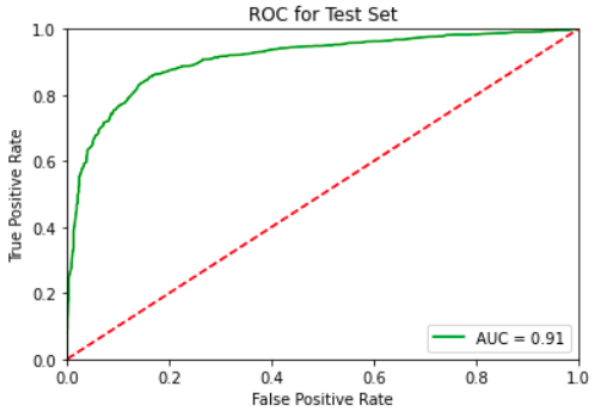
(a) and (c) The confusion matrices and precision, recall and F-1 scores for training and testing set (respectively) are as follows (order is "First Train, Second Test") -

```
Confusion Matrix (TN,FP,TP,FN) clockwise from top-left:
 [[5422  578]
 [ 965 5035]]

Precision Recall F1-Score (1st row for label 0, 2nd row for label 1):
 [[0.84891185 0.90366667 0.87543392]
 [0.89702476 0.83916667 0.86713166]]

Confusion Matrix (TN,FP,TP,FN) clockwise from top-left:
 [[876 124]
 [203 797]]

Precision Recall F1-Score (1st row for label 0, 2nd row for label 1):
 [[0.81186284 0.876      0.84271284]
 [0.86536374 0.797      0.82977616]]
```

The accuracy - **Train Score**: 0.8714166666666666 and **Test Score**: 0.8365

(b) The ROC curve is as follows (first for training, then for testing data) -





Observation - The ROC curves for both training and testing sets show a very high value of True Positive Rate (sensitivity) for a low False Positive Rate(specificity) (around 0.8) which indicates that a high performance threshold is achieved on the dataset using naive bayes. Area under curve (AUC) is directly proportional to how well the model can distinguish between true and false labels.

Q4

4.1 It is assumed that u_i is a random variable whose mean and variance are constant and known to us. Therefore, for a large number of samples, it will act as a constant only.

(a) I will add $\beta_2 G_i$ as a dummy variable to the equation, where G_i is 1 for males and 0 for females (or vice versa). Then I will train the model on the given dataset. If after training, β_2 is very close to zero (or exactly zero), I will conclude that gender has no effect on final output (because if β_2 is zero, then the final equation is same for both men and women). In case β_2 is not zero, then gender does play a role as we will end up with different intercepts for men and women.

$W_i = \beta_0+\beta_1 X_i+\beta_2 G_i+u_i$ is the final equation

(b) I will add a dummy variable once again, as per the equation below -

$W_i = \beta_0+(\beta_1+\beta_2 G_i)X_i+u_i$ is the final equation

Model will be trained on the given dataset. Reasoning is analogous to part (a). If β_2 is close to 0, then slope will be the same for both men and women, which is β_1. In case that is not so, we can conclude that gender does play a role in the slope (and hence in return to experience)

(c) I will modify the equation as follows -
$W_i = \beta_0+(|\beta_1|+\beta_2)X_i+u_i$ is the final equation.

The modulus ensures that the original coefficient is always positive (an upwards slope). The model will be trained on the given dataset. If after the training, $|\beta_1|-\beta_2 < 0$, then the slope will be downward one. In case it is zero, there will be no correlation between Wage and Experience and in the case that the difference is > 0, it indicates that there is indeed a positive correlation (upward slope)

4.2 L2 regularization adds a term equal to *(lamda*(sum of squares of all coefficients))* to the cost. The intuition behind this method is that it is used to prevent overfitting the data (or in other words, reduces the variance of the data). A low value of lambda leads to overfitting of training data while a high value of lambda leads to underfitting. As to why the coefficients are tried to be kept small is simple - they are part of the cost function and we need to minimize the cost, therefore, the coefficients are also tried to be kept as low (and accurate) as possible. In the absence of this L2 regularization term, two things happen - overfitting of training data and exploding of gradients to very high values. Adding L2 regularization terms puts in check both of these issues at once.

4.3

4.3.

Posterior Probability

$$P(\beta|y) = \frac{P(y|\beta) \cdot P(\beta)}{P(y)} \Rightarrow \text{estimate } \beta \text{ (parameters) given } y \text{ (data)}$$

We need to maximize this for all β's.

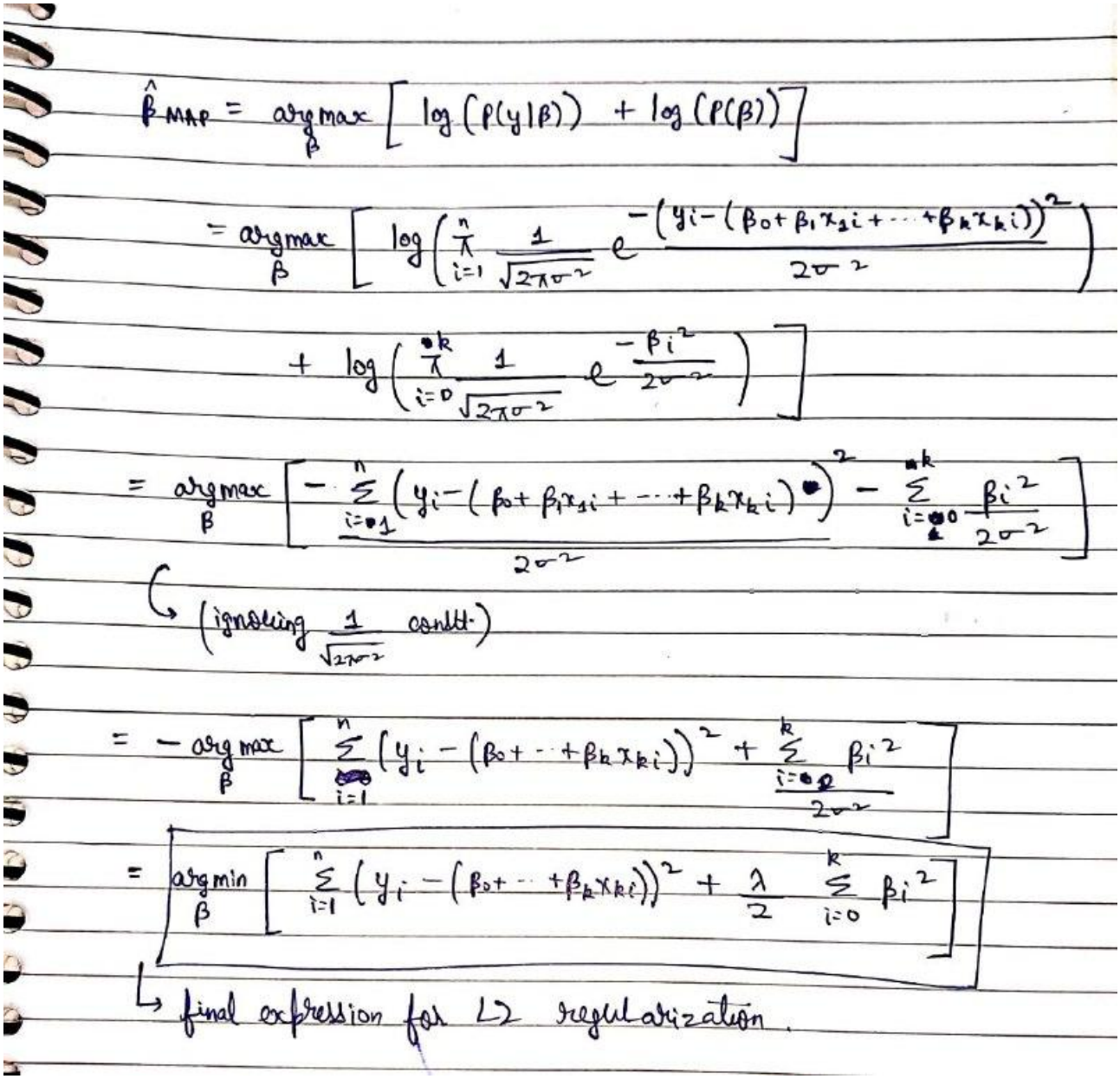
$$\therefore \hat{\beta}_{MAP} = \underset{\beta}{\operatorname{argmax}}\ P(\beta|y) \qquad (MAP = \text{max a posteriori}).$$

$$= \underset{\beta}{\operatorname{argmax}} \left(\frac{P(y|\beta) \cdot P(\beta)}{P(y)} \right)$$

Since this function is monotonically increasing, we can take log (inc. fn) and drop $P(y)$ since it's a Constt. (log is also taken to convert to Logits)

$$\therefore \boxed{\hat{\beta}_{MAP} = \underset{\beta}{\operatorname{argmax}} \left[\log(P(y|\beta)) + \log(P(\beta)) \right]}$$

Now, apply gaussian prior on all parameters and derive L2 regularization, (let gaussian be $\sim N(\mu, \sigma^2)$)



THE END (thank you for your time)