Report : Assignment 1

Name: Shaunak Pal
Roll No.: 2018098

Problem 1) Linear Regression

Value of K Justification (K = 4)

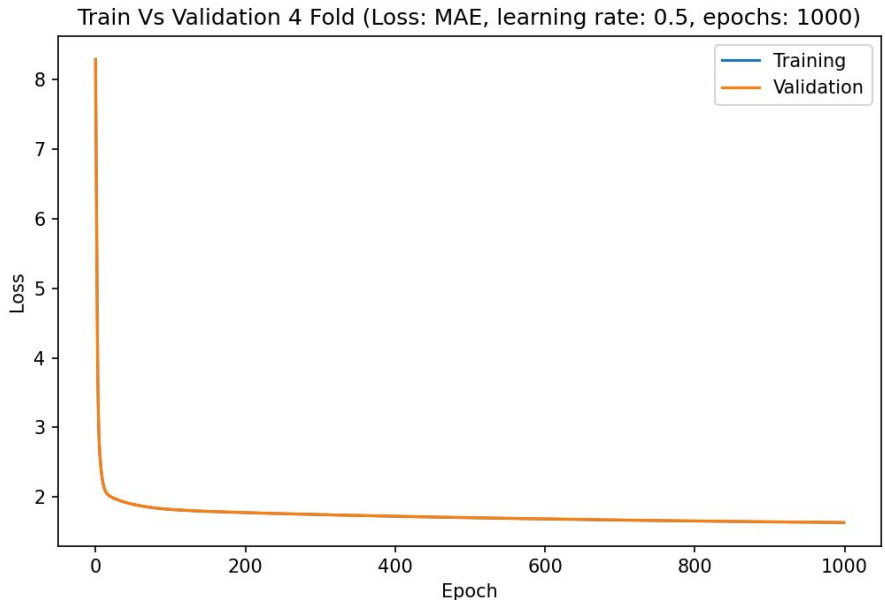
I have K as 4 since the dataset is not really big and does not require as many folds. The model is able to learn well in 1 fold itself

Preprocessing Strategy

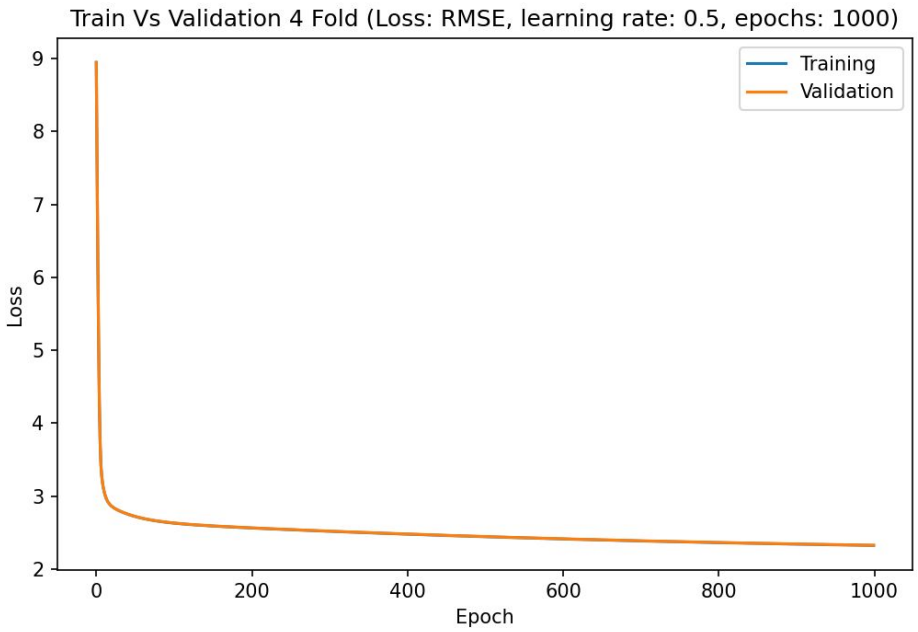
- Since more than half of the rows of the video games dataset has null values it is better to remove it than to replace it with mean or something else since we are predicting almost half of the dataset ourselves which can induce a lot of bias into the dataset
- After cleaning the data I have shuffled them so that the order doesn't affect our model in any way

a) Abalone Dataset (1)

Mean Absolute Error

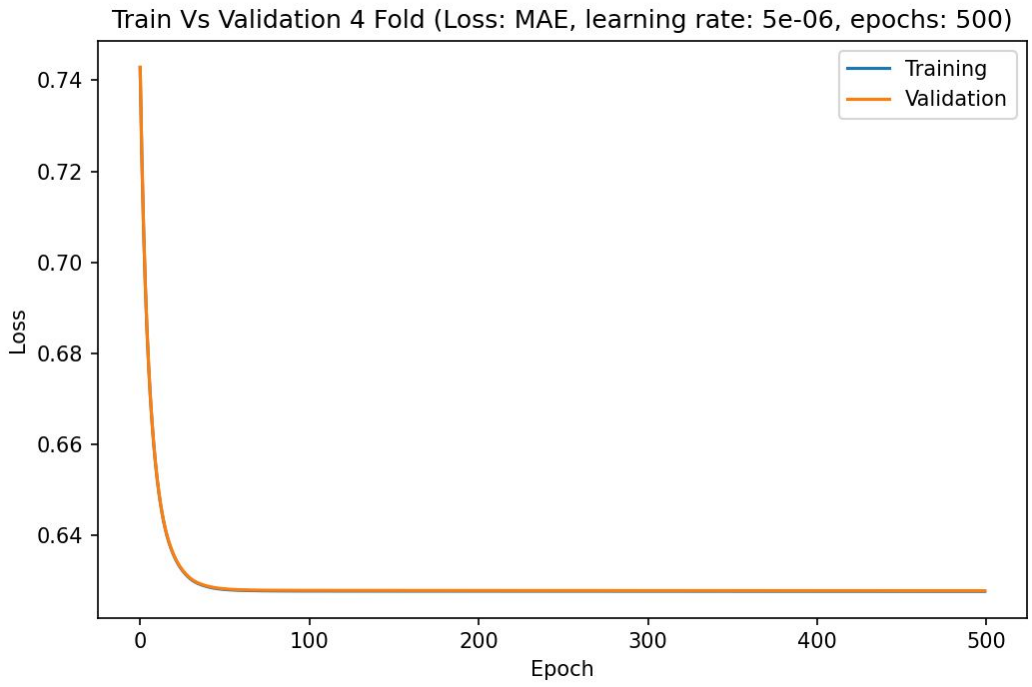


Root Mean Squared Error

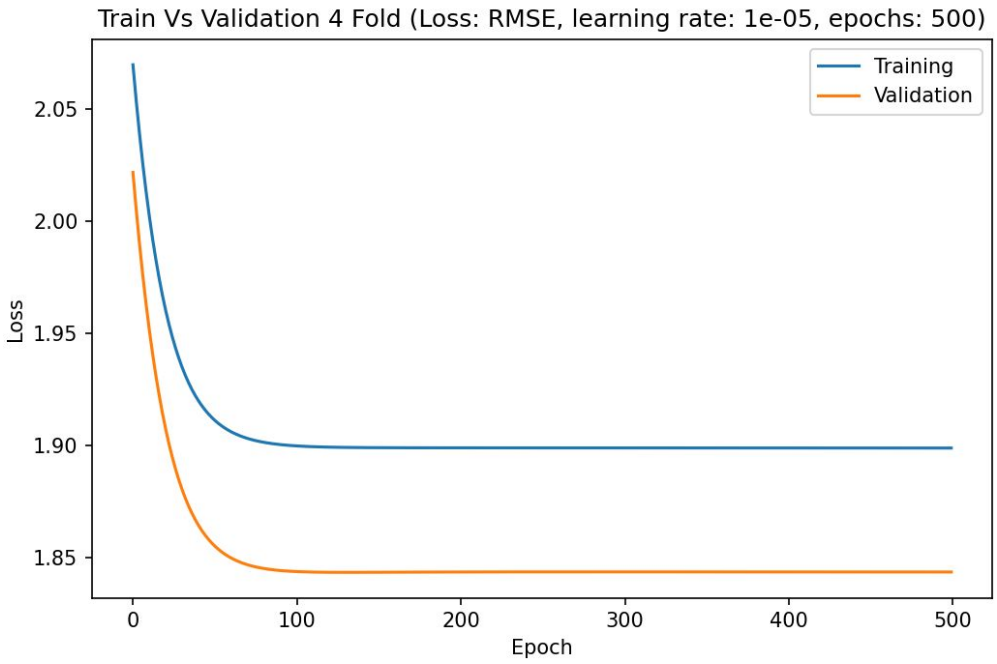


Video Game Dataset (2)

Mean Absolute Error

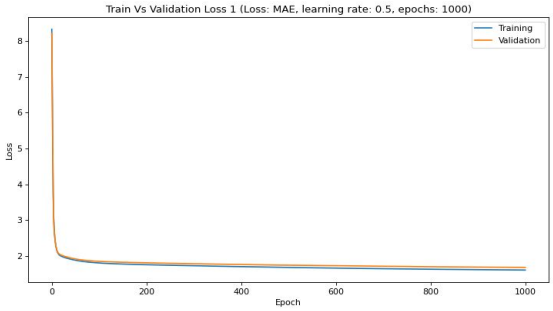


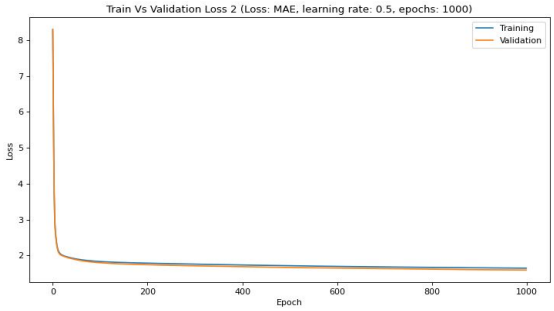
Root Mean Squared Error

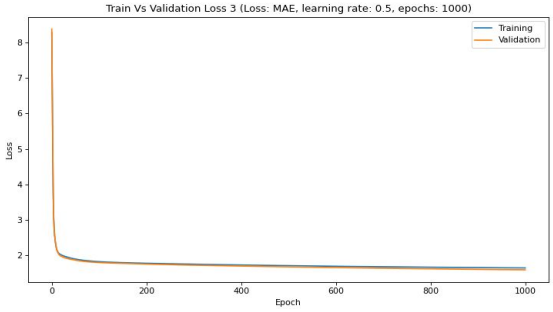


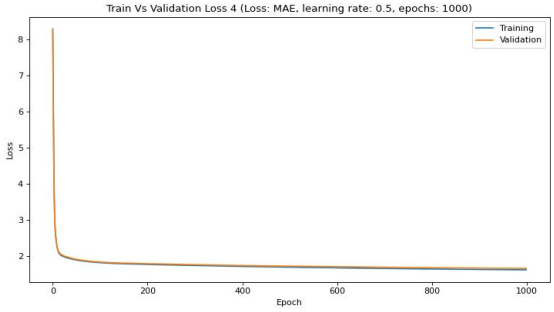
b) Abalone Dataset (1)

Mean Absolute Error







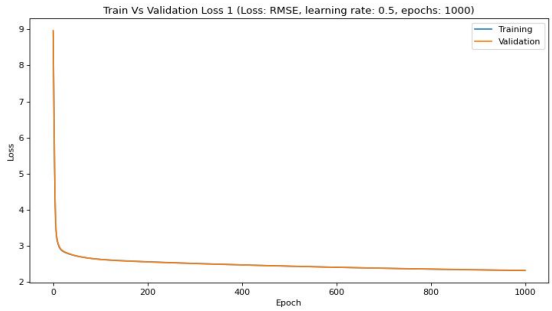


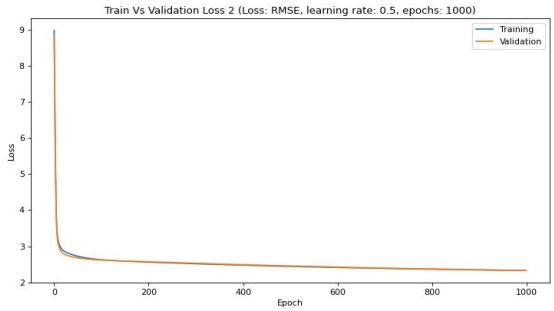
Best loss achieved in 2nd Fold

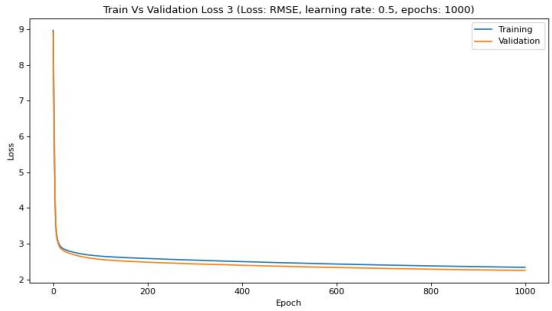
Min train cost achieved : [1.65940339]

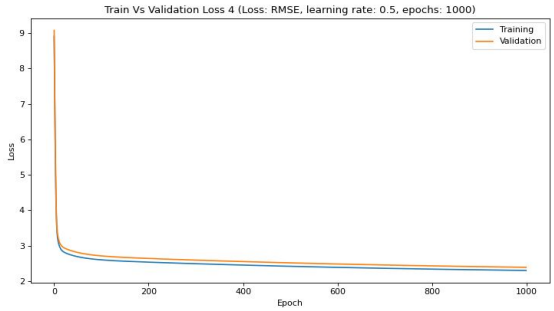
Min test cost achieved : [1.55249314]

Root Mean Squared Error









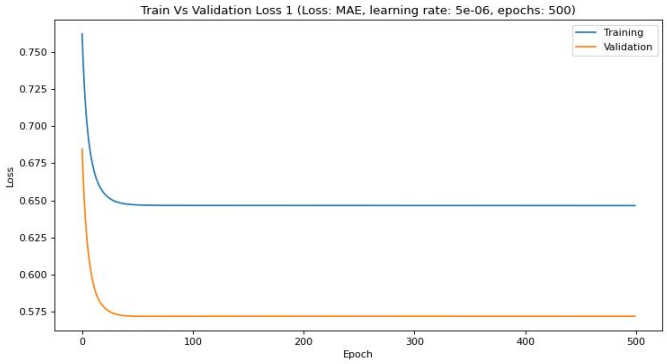
Best loss achieved in 4th Fold

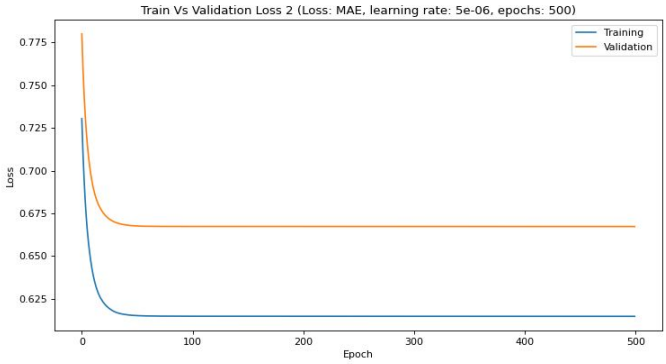
Min train cost achieved : [2.33479558]

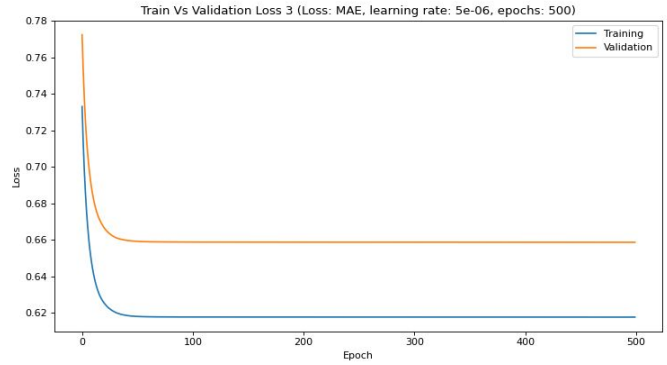
Min test cost achieved : [2.30640283]

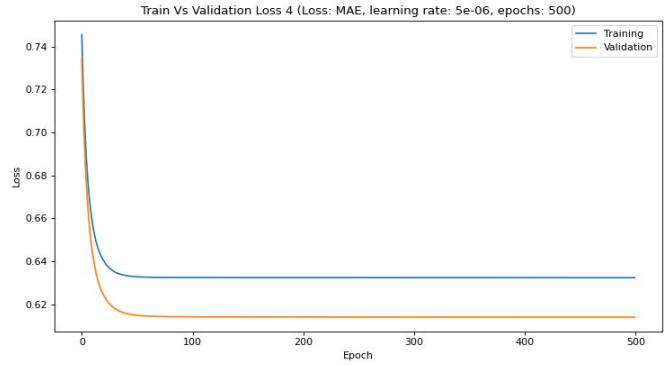
Video game Dataset

Mean Absolute Error







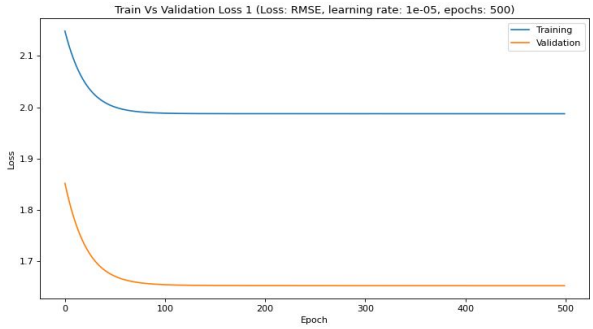


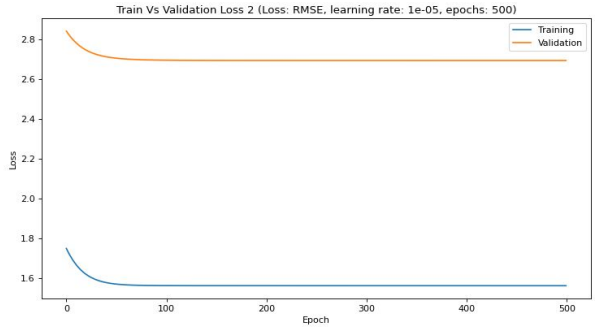
Best loss achieved in 1st Fold

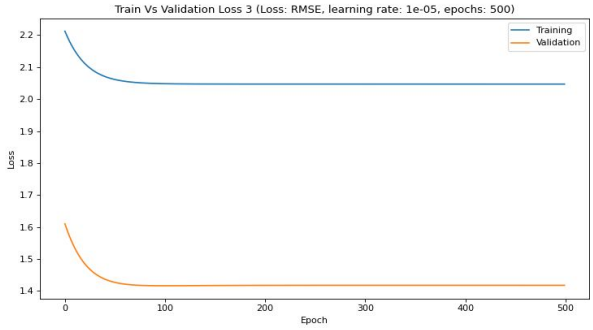
Min train cost achieved : [0.64646289]

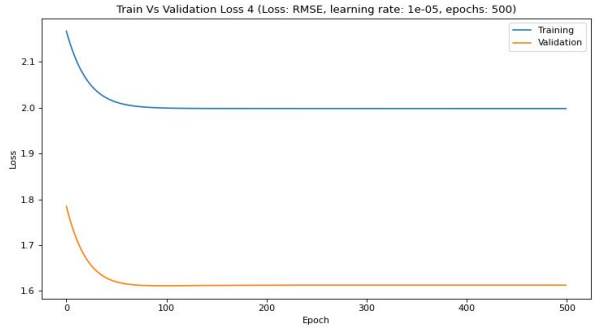
Min test cost achieved : [0.57171588]

Root Mean Squared Error









Best loss achieved in 3rd Fold

Min train cost achieved : [2.04690542]

Min test cost achieved : [1.41622445]

(c) Looking at the loss values directly we can say that MAE loss is lower than RMSE loss but that won't make sense since both the losses are of different metrics and cannot be compared. Therefore we use a common loss function to compare both the models.

First using MAE loss function we get :

MAE loss model: 1.7143

RMSE model: 1.9562

Then using RMSE loss function we get:

MAE loss model: 2.5293

RMSE loss model: 2.5021

Looking at the losses, the RMSE model has eros pretty close, but inthe MAE loss function the models have a decent amount of difference in loss with MAE having a much lower loss, Hence we can conclude that probably Mean Absolute Error acts as a better loss function.

(d) MAE x $\sqrt{(n)} \geq$ RMSE.

The difference between Root Mean Square Error and Mean Absolute Errors gives the variance of the errors. Hence if the values are similar, Root mean square Error and Mean absolute Error will have similar magnitude. In such a case RMSE should be preferred since it is a convex function and unlike MAE it is differentiable at zero.

e) Optimal parameters: [[3.59143863]
 [-0.38042345]
 [0.19594323]
 [12.12073487]
 [8.79813377]
 [10.6020379]
 [-22.30776771]
 [-10.85386195]
 [6.86947032]]

Fold number 2 (Best fold in part b in MAE)
Min train cost achieved : [1.65940339]
Min test cost achieved : [1.55249314]
Train loss vs optimal parameters: [0.80463522]
Test loss vs optimal parameters: [0.80456975]

Problem 2) Logistic regression

Analyzing the dataset:

- Using pd.isnull().sum()
  None of the columns have null values
- Using pd.info()
  All the rows except the last row which is the binary label (‘int64’), are ‘float64’
- Using pd.describe()
  The row with y label has a mean of 0.44 and standard deviation of 0.49 which shows that the data is well balanced (because only possible values are 0 and 1)

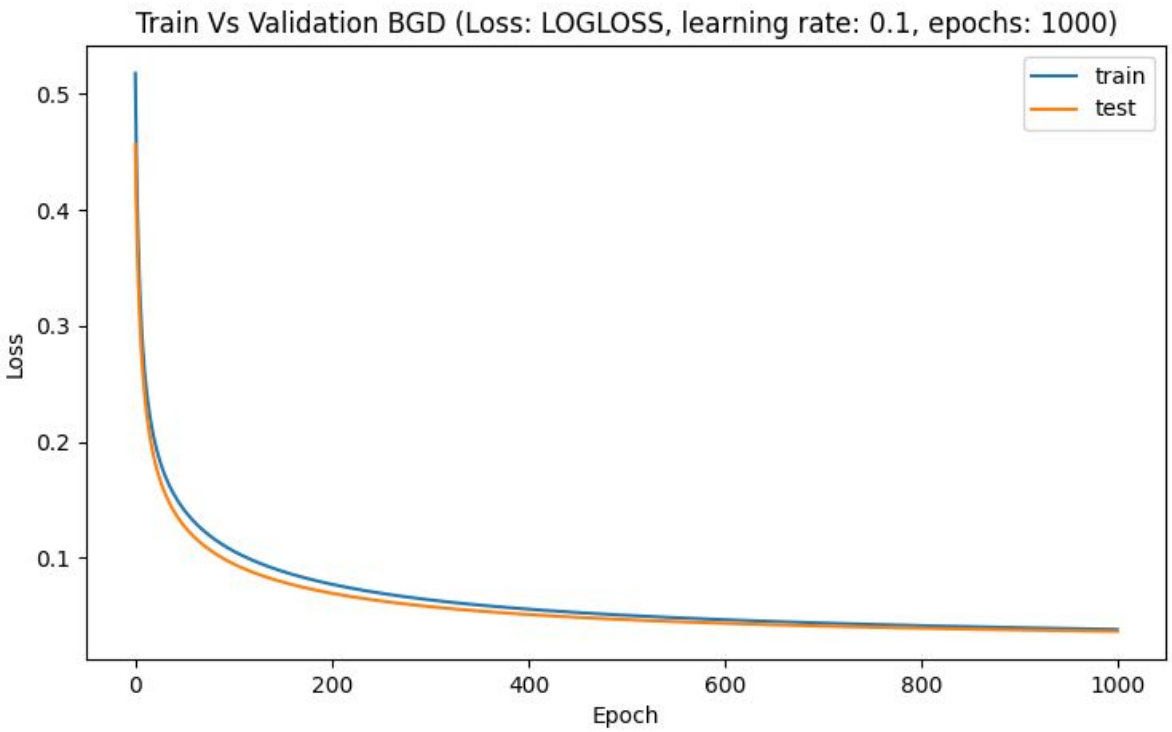
a) Batch Gradient Descent (learning rate = 0.1, epochs = 1000)

   BGD Train Accuracy : 0.9895941727367326
   BGD Test Accuracy : 0.9927007299270073

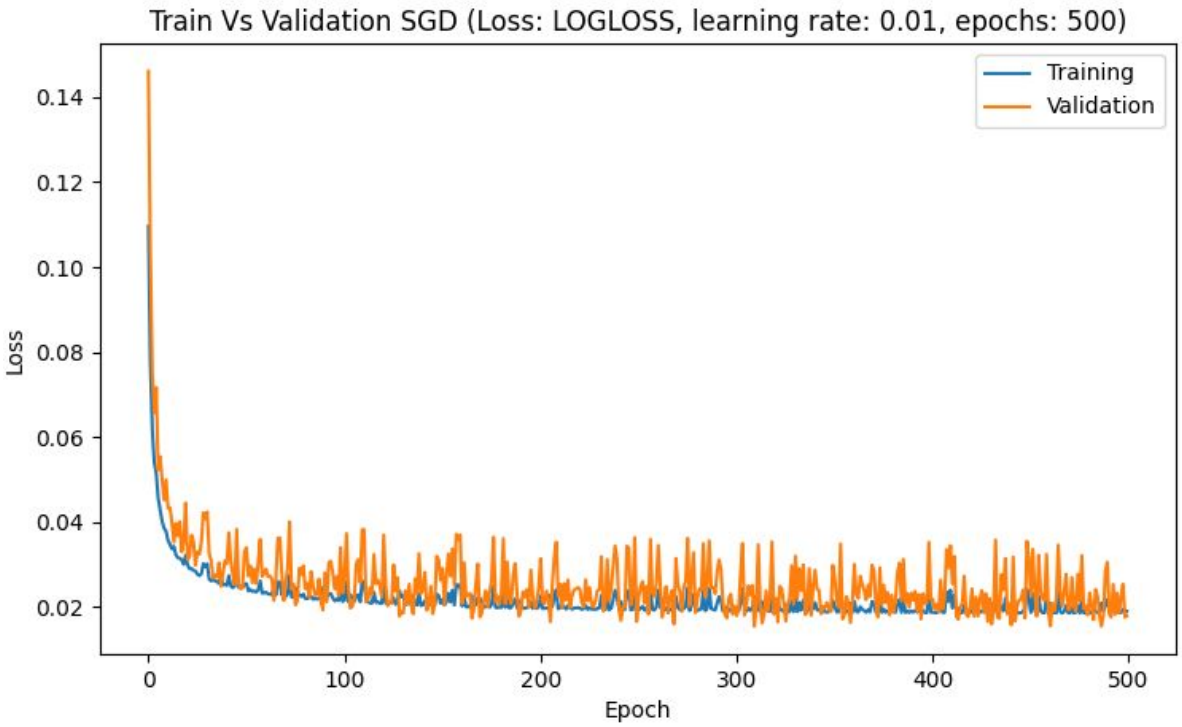
   Stochastic Gradient Descent (learning rate = 0.01, epochs = 500)

   SGD Train Accuracy : 0.991675338189386
   SGD Test Accuracy : 0.9927007299270073

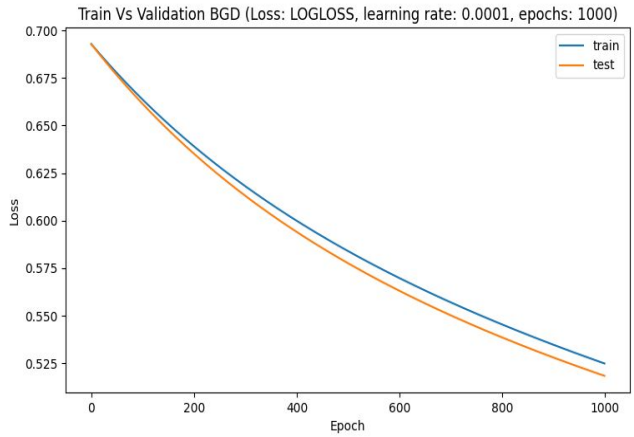
b) Batch Gradient Descent

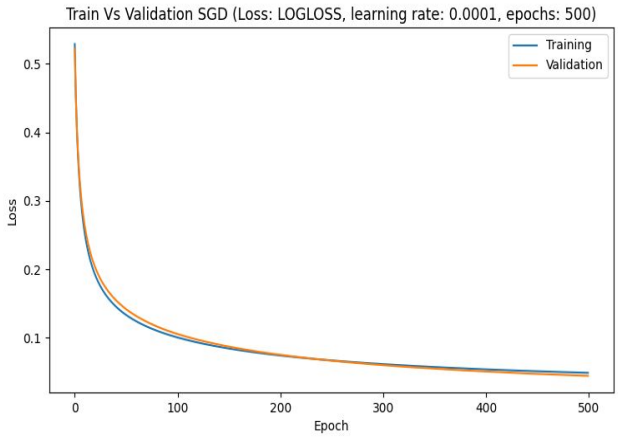


Stochastic Gradient Descent

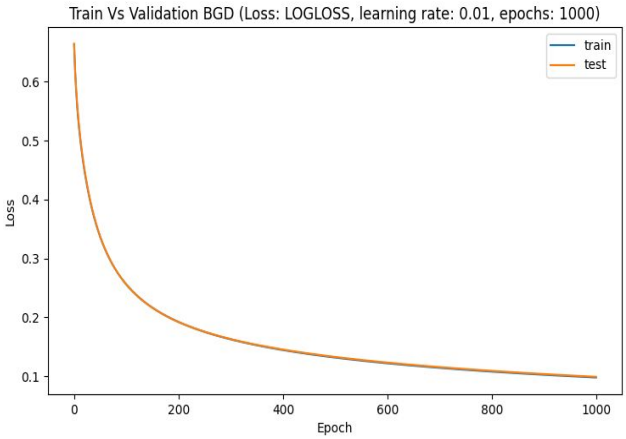


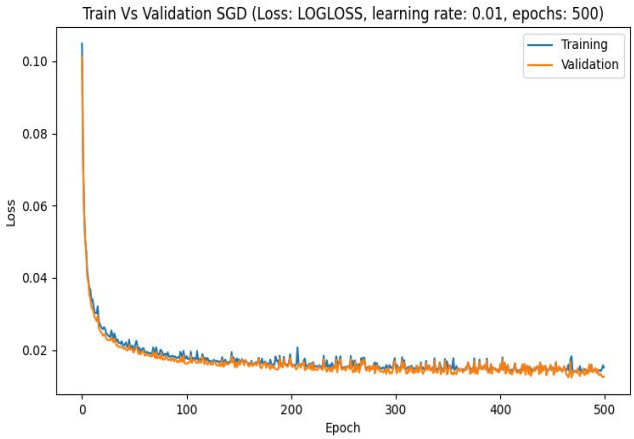
c) Learning rate : 0.0001



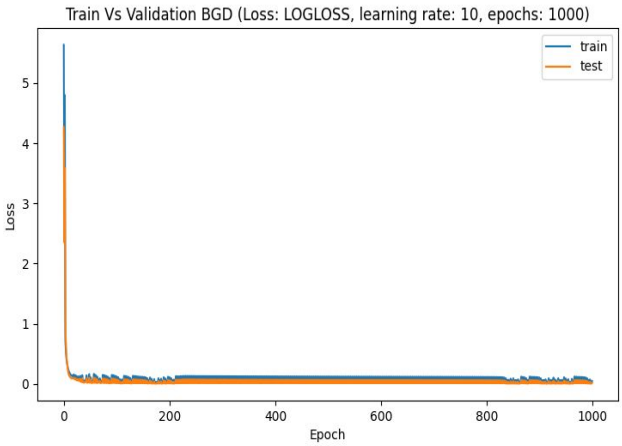


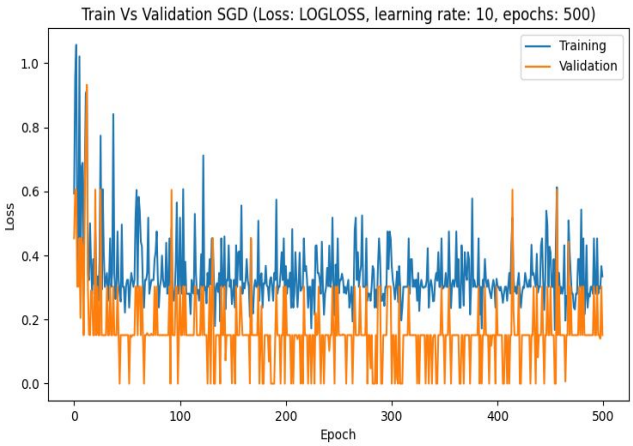
Learning rate : 0.01





Learning rate : 10





Comparison

---

a) The loss plot of batch gradient descent is smooth whereas that of stochastic gradient descent has spikes which is due to the reason we consider one sample at a time while performing gradient descent
b) BGD takes longer to converge approx 1000 epochs whereas SGD converges faster
c) Using LogisticRegression(max_iter=1000)
   SKL Train Accuracy : 0.9864724245577523
   SKL Test Accuracy : 0.9963503649635036
d) Using SGDClassifier(loss="log", max_iter=500, alpha=0.01)
   SKL Train Accuracy : 0.9802289281997919
   SKL Test Accuracy : 0.9854014598540146
   Whereas my SGD implementation accuracy was:
   SGD Train Accuracy : 0.991675338189386
   SGD Test Accuracy : 0.9927007299270073
   We can say that both are pretty similar in accuracy

Problem 3)
Since we use MSE in calculating the gradient descent

Cost = $\sum\{(y\_pred - y)^2\} / n\_samples$

y_pred can be calculated using the sigmoid function.

Gradient of sigmoid function: (y_pred - y) * (y_pred) * (1 - y_pred)
As the differentiation of the cost matches our gradient, we now check the boundary conditions that is when y_pred approaches 1 or 1-y_pred approaches 0.

Differentiation of cost:
Gradient = (y_pred - y) * (y_pred) * (1 - pred)

Therefore:
As y_pred approaches 1 -> (1-y_pred) approaches 0 -> Gradient approaches 0
As y_pred approaches 0 -> gradient becomes 0

Hence Mean Squared Error does not penalize for the missclassified 1, which might lead to a big loss. Instead of MSE if we use LOGLOSS the gradient is calculated using:
Gradient = (y_pred - y)
This penalizes any missclassified 1s, which would make out model learn more effectively

So, as our y_pred approaches 1, our (1-y') term approaches 0, and hence makes the gradient descent approach 0. If our y' approaches 0, y' term becomes 0, making gradient 0 again.
This implies that MSE does not penalize our misclassification for 1, which might contribute to a significant loss.
If we used cross-entropy(also knows as log loss), our gradient is given by the following formula:
Gradient = (y'-y)
Our misclassification of 1 will be penalized, thus making our model learning effective.

Problem 4)

(a) β0 = -0.12951238
β1 = 0.01195419
β2 = -0.19733212

(b)
$1 / (1 + e \text{ ^ } (0.13 - 0.016x^1 + 0.2x^2))$

(c) exp(β1) = 1.017049029315361
exp(β2) = 0.8239341453511093
Thus the coefficients are almost equal and so they contribute equally to the regression

(d) Estimated probability = 0.65102163