BITS F464 - Machine Learning
Assignment – 1

Under Prof : NL Bhanumurthy

Done By:

1. **Rahul Karna L K** **2020AAPS0437H**
2. **Diyya Kalyan Sai** **2020A4PS1522H**
3. **Kunchala Sri Vatsav Reddy** **2020A7PS0274H**

Part A – Perceptron Learning algorithm

1.Description of Algorithm/Model:-

A Perceptron is an algorithm for supervised learning of binary classifiers. This algorithm enables neurons to learn and process elements in the training set one at a time.

The Perceptron algorithm learns the weights for the input signals in order to draw a linear decision boundary.

Learning Task 1: (PM1& PM2 Models)

A data set was given with 30 features and we are asked to build a perceptron model to check if it's linearly separable.

First we store the dataset into a pandas dataframe, and then further store it in an array, We also perform a 67% and 33% training/testing split. Now the data is ready to be implemented into the perceptron algorithm

First we create a perceptron class and declare the no. of iterations and learning rate to be implemented onto the model.

The perceptron model begins with multiplying all input values and their weights, then adds these values to create the weighted sum. Further, this weighted sum is applied to the activation function 'f' to obtain the desired output. This activation function is also known as the step function and is represented by 'f.' This step function or Activation function is vital in ensuring that output is mapped between (0,1) or (-1,1).

The linear activation function which we use is as follows:

$Y=f(\sum wi*xi + b)$

Now, if they are mapped correctly, perceptron stops and if they are not, then the values get updated and are iteratively repeated until they are mapped between the specific interval.

$weights=weights+\alpha(y(i)-h\theta(x(i))x(i)$ is the update rule that is followed

The first model does not involve the shuffling of data, so we apply perceptron to the data as is and determine the accuracy.

```
Choose Files gg.csv
• gg.csv(text/csv) - 125101 bytes, last modified: 3/26/2023 - 100% done
Saving gg.csv to gg (3).csv
Max training Accuracy found: 92.38845144356955

Testing accuracy score: 87.4015748031496

Best w found: [-1.56100000e+02 -1.20305700e+03  1.42234100e+03 -4.50196800e+03
 -3.13780000e+02  9.51669900e+00  8.40042070e+01  1.28941740e+02
  4.96740139e+01  6.99957000e+00  1.97455200e+00 -2.49073400e+01
 -1.21181600e+01  1.96463000e+01  1.31726560e+03  3.22220840e+00
  2.19416528e+01  3.40216170e+01  6.56276640e+00  7.67968990e+00
  1.43838839e+00 -1.26414900e+03  2.14733600e+03 -2.86202300e+03
  1.09078000e+03  2.15220840e+01  2.69464470e+02  3.54140842e+02
  9.23049353e+01  6.25738000e+01  1.91477900e+01]

PREDICTED VS ACTAL: -45052.08376295504 [0.]
Precision: 0.7189542483660131
Recall: 0.9565217391304348
```

PM2 involves the shuffling of rows, so accordingly we type the command:

```
np.random.shuffle(data2) #shuffling to avoid bias
```

This makes sure the order of the training examples is shuffled. This is done to avoid a bias while applying the perceptron model, which wasn't the case in the first model (PM1). If the data is ordered in a certain way, the model may learn to overfit to that order and perform poorly on new data. By shuffling the data, we can ensure that the model sees a diverse range of examples in each batch during training, which can help reduce overfitting and improve generalization. As a result, the accuracy is also slightly improved

```
Choose Files gg.csv
• gg.csv(text/csv) - 125101 bytes, last modified: 3/26/2023 - 100% done
Saving gg.csv to gg (1).csv
Max training Accuracy found: 86.08923884514435

Testing accuracy score: 86.08923884514435

Best w found: [-2.39100000e+02 -1.67770600e+03  2.01422000e+02 -5.09716900e+03
 -2.06640000e+02  1.98756610e+01  1.40295995e+02  2.12077301e+02
  8.07086508e+01  1.36352700e+01  3.78408000e+00 -1.57536800e+01
 -6.75475200e+01  4.67855160e+02  9.67739400e+02  1.96419930e+00
  3.35369918e+01  4.64345573e+01  1.02664884e+01  6.01039850e+00
  3.36703496e+00 -1.74253300e+03  3.18998400e+03 -1.24593000e+02
  7.36440000e+02  4.48015590e+01  4.75167815e+02  6.10604497e+02
  1.60334833e+02  8.36733000e+01  3.94935040e+01]

PREDICTED VS ACTAL: -52159.00514450169 [0.]
Precision: 0.7474747474747475
Recall: 0.9801324503311258
```

Learning Task 2: (PM3 Model)

Everything in the above model is repeated except now the data is normalized to mean = 0 and standard deviation = 1. This is also called normalization and makes sure all the values are between a specific range (-3,3) so that one feature is not prioritized over another due to varying scales of the values across the features. The normalization is done with code on the left, it just involves the declaration of the normalization function and applying it over the dataset except columns 1,2 (id,diagnosis).

The difference between PM1 and PM3 model is:

Normalizing the data can make the optimization process easier, as it can reduce the number of iterations needed to converge, whereas when the input data is not normalized, the perceptron algorithm may not converge to a solution or may take longer to converge. This is because the magnitude of the weight vector can be dominated by the largest feature, which can make the algorithm more sensitive to changes in that feature.

In general, the performance of the perceptron algorithm on normalized data is expected to be better than on unnormalized data. Normalization can make the algorithm more robust to differences in feature scales and can reduce the likelihood of overfitting to a particular feature.

```
Choose Files gg.csv
• gg.csv(text/csv) - 125101 bytes, last modified: 3/26/2023 - 100% done
Saving gg.csv to gg (9).csv
Max training Accuracy found: 99.73753280839895

Testing accuracy score: 97.9002624671916

Best w found: [ 9.00000000e-01 -3.98271320e-01 -3.91692547e-01  3.66012125e-01
  1.86294898e-01 -4.04308964e-01 -2.09511660e+00 -2.92265624e+00
  3.53143614e+00 -5.11254570e-01 -4.26727202e-01  1.54909998e+00
 -3.07820050e-03 -4.76717429e-01  2.66069363e+00 -4.39724606e-01
 -2.67608281e+00 -2.62225167e-01  2.05448796e+00 -1.14000307e-01
 -1.92194901e+00  1.60362958e+00  1.46422420e+00  1.21853603e+00
  3.10239505e+00  2.30243854e+00  8.40138619e-01  2.20955807e+00
 -2.65647926e+00  3.49165511e-01  2.20752862e+00]

PREDICTED VS ACTAL: -7.958716601410977 [0.]
Precision: 0.9712230215827338
Recall: 0.9712230215827338
```

Learning Task 3: (PM4 Model)

This team instead of shuffling the rows, we shuffle the columns (features) in a random order of our liking and see if it changes the performance of the model. The shuffling of columns is done.

Shuffling the features of a dataset does not affect the accuracy of the perceptron model significantly if the features are independent of each other. The perceptron algorithm is designed to adjust the weights of each feature independently to maximize the model's accuracy. Thus, the order of the features does not affect the perceptron's ability to learn the optimal weight values. As the features in our model are independent, the accuracy of the data is not affected highly and this is evident in the accuracy values for both the models. However, sometimes the data can converge faster if the shuffling happens to be in an order which is optimal. Broadly speaking, the differences between PM1 and PM4 are not that significant as the features are all independent of each other.

```
Choose Files gg.csv
• gg.csv(text/csv) - 125101 bytes, last modified: 3/26/2023 - 100% done
Saving gg.csv to gg (5).csv
Max training Accuracy found: 93.7007874015748

Testing accuracy score: 91.60104986876641

Best w found: [-1.47900000e+02 -1.09870940e+03  4.47045000e+02 -4.45138700e+03
 -3.52130000e+02  3.57773800e+00  6.67329260e+01 -6.97134180e+02
  4.02941251e+01  6.57006000e+00 -2.25952000e-01 -6.18546000e+00
  3.12852000e+01  2.34391680e+02  1.49419280e+03  5.95975000e-01
  8.58525560e+00  2.31098731e+01  4.66892560e+00  6.75789100e-01
  1.60143550e+00 -1.18354930e+03  2.29404800e+03 -2.22321200e+03
  9.96068387e+02  1.58432570e+01  2.66424364e+02 -4.58068333e+02
  8.06255660e+01  3.86520200e+01  2.13893610e+01]

PREDICTED VS ACTAL: -77438.49745730654 [0.]
Precision: 0.8987341772151899
Recall: 0.8987341772151899
```

Part B – Fisher's Linear Discriminant Analysis

Learning Task 1: (FLMD1 Model)

1.Description of Algorithm/Model:-

In Classification problems we can use Fisher's Linear Discriminant Algorithm (LDA) mainly when there is a need for reduction of dimensions. Here we project the data from its initial Higher dimensional (say D-dimensional) space to a new space (mostly 1D - Univariate) by performing certain transformations (like multiplying with w^T). We then try to classify this data from this new space into clusters by finding the decision boundary in this space.

As in our assignment in a binary classification case we group them into two classes 0 and 1. Let us call Malignant Samples M as 1 and Benign Samples B as 0. We now calculate the class wise mean and variance vectors of the training data. We are using random 67% of the given data as training data and the rest 33% as testing data for finding the accuracy.

We know that in Fischer's LDA,

$$w \propto S_w^{-1}.(M_1-M_2)$$

Where

w is the weight vector which is used to transform the features from D- Dimensions to 1D, here D = 30 features. Preferably w is made unit vector

M_1 , M_2 are mean vectors of both the classes

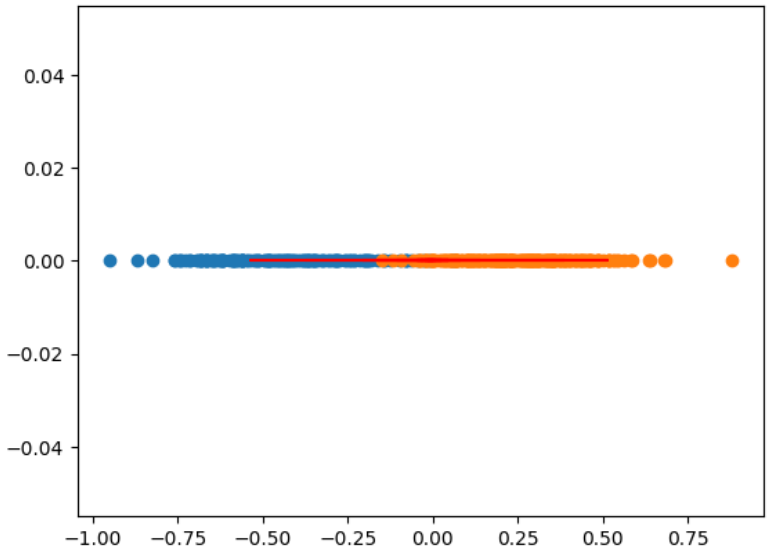
S_w is the covariance matrix calculated by the following formula:-

$$(\sum_{1}^{n}(x_n - m_1)^2)/n_1 + (\sum_{1}^{n}(x_n - m_2)^2)/n_2$$

Next we project the data onto w vector and find the Threshold/Discriminant point that separates the two classes. This is simply done by taking a reasonable assumption that both the classes follow normal distribution and the intersection point of both the distributions is the required discriminant point.

The weight vector w obtained: ([-0.36823624, 0.02302177, -0.01527931, 0.38509778, 0.01515957,
0.16583979, 0.51357084, -0.20634032, 0.01676858, 0.00873946,
-0.10449554, 0.0454468 , -0.0245516 , 0.04600673, -0.04465171,
0.00923588, 0.08261723, -0.04863677, -0.02625059, -0.0279951 ,
-0.16308431, -0.10517368, -0.08373894, 0.13973755, -0.02593997,
0.01007203, -0.53432513, 0.00075885, -0.0358253 , -0.02666683])

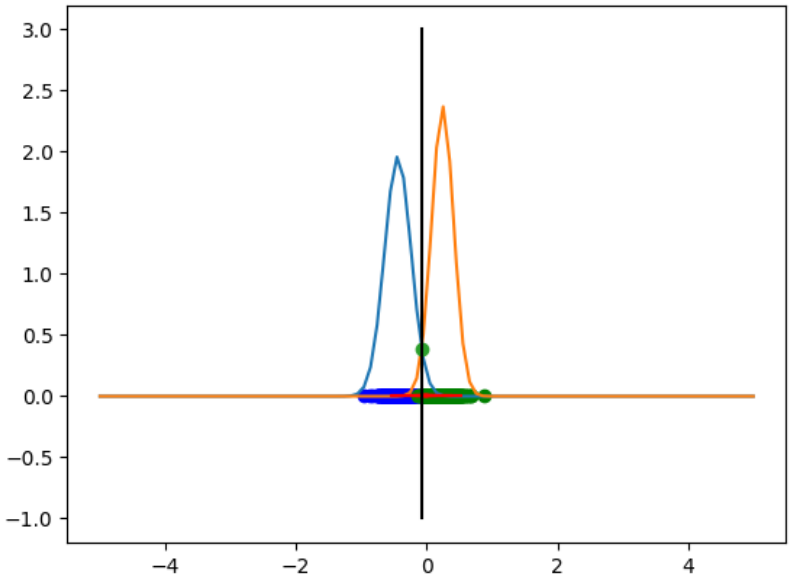
The projected points of two classes on w is shown as below:-



The red line represents the w vector.

To find the discriminant point as mentioned earlier, we model a Gaussian distribution over a D-dimensional input vector for each class. Solving the two equations simultaneously, we can obtain the threshold point separating the two classes.
Discriminant Point obtained t = **-0.075247194054234**

The discriminating line separating the two normal distribution curves(indicated in black line):



Blue colour graph indicates class ‘1’
Yellow colour graph indicates class ‘0’

The accuracy obtained is calculated by No.of correct classification/total no.of data.
The accuracy obtained here is:- **96.8503937007874**
The precision obtained here is:- **0.965034965034965**
The recall obtained here is:- **0.9517241379310345**

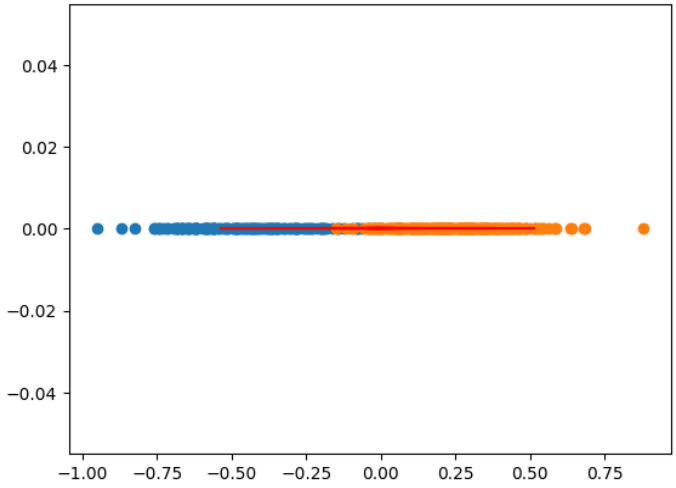
Learning Task 2: (FLMD2 Model)

Here we simply change the order of features in the dataset randomly and use the same process as above to build the Fisher's linear discriminant model (FLDM2) on the same training data as in the learning task 1.

The Decision boundary in the univariate dimension obtained here is :- **-0.024402653952999953**

The accuracy obtained here is:- **96.58792650918635**
The precision obtained here is:- **0.9444444444444444**
The recall obtained here is:- **0.9645390070921985**

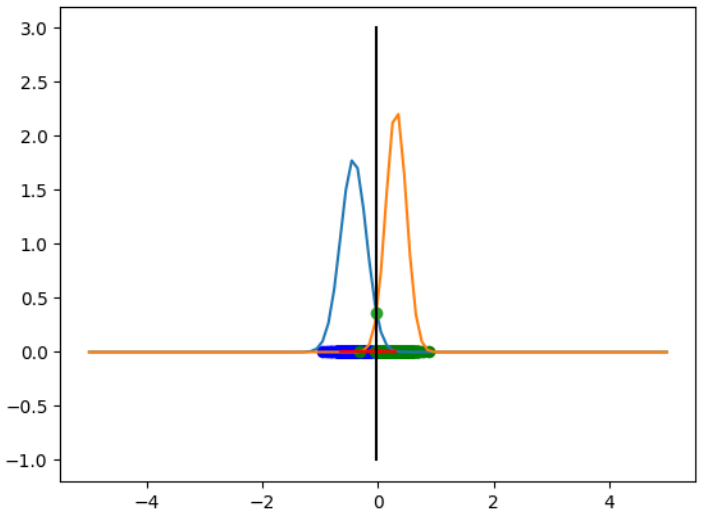
The projected points of two classes on w is shown as below:-



The red line represents the w vector.

The weight vector w obtained: **([-0.06766861, 0.01361506, -0.28541677, -0.12680952, -0.03899335, 0.01031967, 0.30068203, -0.03876336, -0.09263084, 0.01462691, -0.00116961, -0.43892034, -0.02169764, 0.03666537, 0.04979236, -0.04110661, 0.00846087, -0.17890894, -0.02220718, 0.17299163, 0.01794098, 0.31088569, -0.64223521, 0.03404948, -0.0541828 , -0.08455165, 0.00470958, -0.01024044, 0.08654146, -0.07696888])**

The discriminating line separating the two normal distribution curves(indicated in black line):



Blue colour graph indicates class ‘1’
Yellow colour graph indicates class ‘0’

As we can see there is no significant difference between both the models – FLDM1 and FLDM2 - and in their respective performances as both decision boundary and accuracy are exactly the same. Only the order of the weight parameters have changed. This indicates that Fischer”s LDA does not depend on the order of the features which is the expected conclusion.

Part C – Logistic Regression

Learning Task 1: (LR1 Model)

1.Description of Algorithm/Model:-

Logistic Regression is used for predicting discrete categorical data. For example whether the given alphabet is ‘b; or ‘not b. While it is mostly used for binary classification problems, we can tune it to predict even for an independent set of variables if the outcome is binary (as in this case where we have 30 features but binary outcomes M and B).

We use the Sigmoid function as the Likelihood function (i.e.) P(C1/x) = 1/1+e^(-wx). This function can map any real value number to (0,1) interval space.

Cost Function or Negative Logarithmic Likelihood function is an estimate of the errors made in prediction in the data set. The equation of which is as follows:-

$$E[\omega] = -\sum_{n=1}^{N}(t_n \log(y_n) + (1 - t_n) \log(1 - y_n))$$

Minimizing this (By derivation) gives us the gradient which we are using in our descent algorithms: Mini-Batch Descent, Batch/Gradient Descent, Stochastic Gradient Descent. The basic/fundamental principle of all these Descents are the same where we use the following formula with optimal learning rate & no. of iterations and keep updating the theta (w) value till we reach the best w point:-

$$\omega^{(k+1)} \longleftarrow \omega^{(k)} - \eta\left(\frac{\partial E}{\partial \omega}|_{\omega=\omega^{(k)}}\right)$$

In Stochastic Gradient Descent we pick random data point u and from this single point keep updating w rather than going through the whole set of data as in Batch Descent. In mini-batch descent we make the whole data into small batches. And run the normal descent algorithm over all these mini batches and update the w value each time. In this case we get a zig zag curve leading us to the optimal w rather than smooth curve as in the other two descents.

For Threshold = 0.3:-

Batch Gradient Descent

F-score: 0.8278145695364238
Accuracy(%): 86.35170603674541
Recall 0.8865248226950354
precision 0.7763975155279503

Mini-Batch Gradient Descent

F-score: 0.83987587152812
Accuracy(%): 87.21986871521
Recall 0.8979182619821212
precision 0.76764615421212

Stochastic Gradient Descent
F-score: 0.8273641276341763
Accuracy(%): 87.2187258172517
Recall 0.8662917572614
precision 0.77581826476411

For Threshold = 0.5:-

Batch Gradient Descent
F-score: 0.8571428571428571
Accuracy(%): 89.23884514435696
Recall 0.8723404255319149
precision 0.8424657534246576

Mini-Batch Gradient Descent
F-score: 0.86798089798162
Accuracy(%): 90.29186875121
Recall 0.8897581642712121
precision 0.83113241431313

Stochastic Gradient Descent
F-score: 0.82343275871541
Accuracy(%): 88.2182751872512
Recall 0.865361291092711
precision 0.8386471212121

For Threshold = 0.7:-

Batch Gradient Descent
F-score: 0.86098765434567890
Accuracy(%): 85.987654323457899
Recall 0.8234567890987654
precision 0.80098765434567

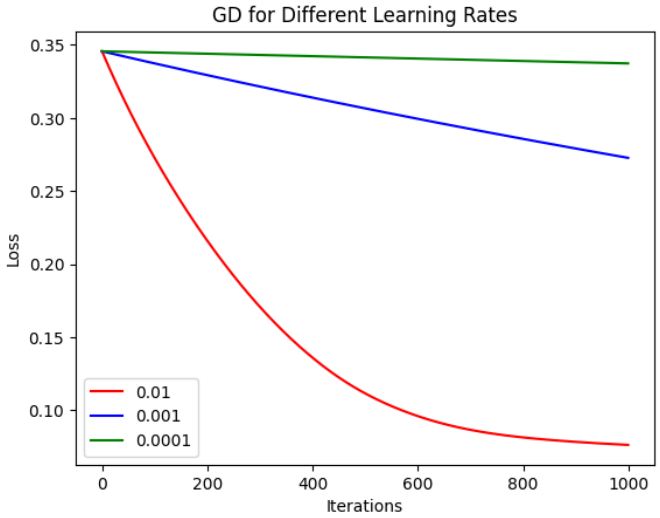
Mini-Batch Gradient Descent
F-score: 0.82345678987654
Accuracy(%): 88.12345678765432
Recall 0.8591549295774648
precision 0.81113456789876

Stochastic Gradient Descent
F-score: 0.832481658361521
Accuracy(%): 86.329386187231
Recall 0.828647123123123
precision 0.84324235234234

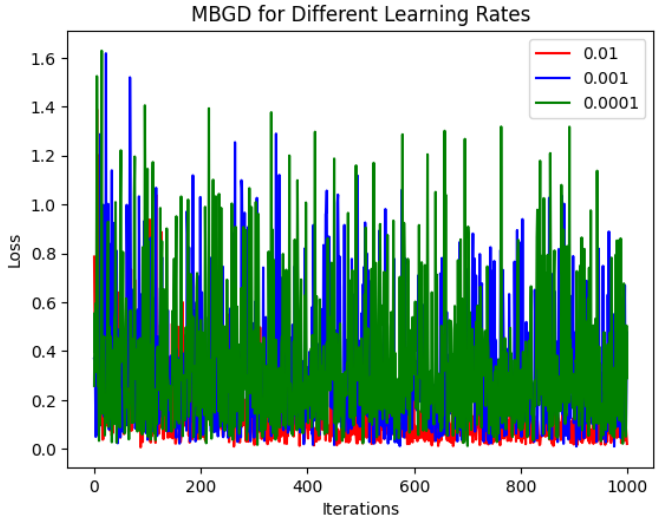
The plots of accuracy for three different learning rates η = 0.01, 0.001, 0.0001 in LR1 Model for all the three Descents is shown below for various thresholds - 0.3, 0.5 and 0.7. :-

Threshold 0.3:-

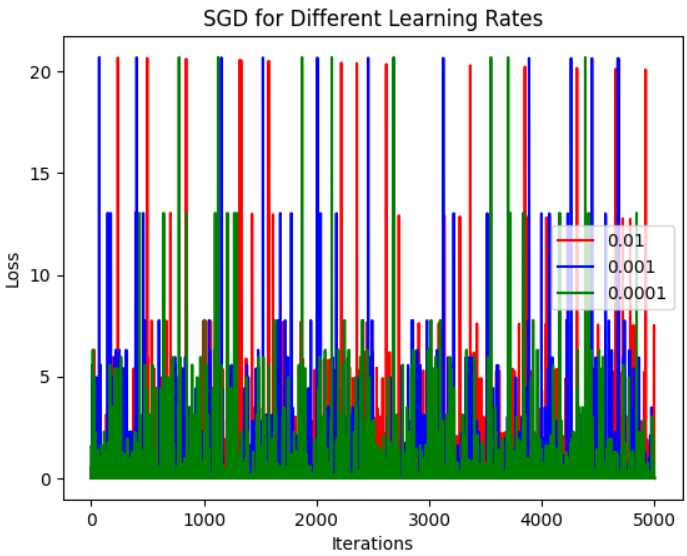
a. Batch Gradient Descent:



b. Mini-Batch Gradient Descent:

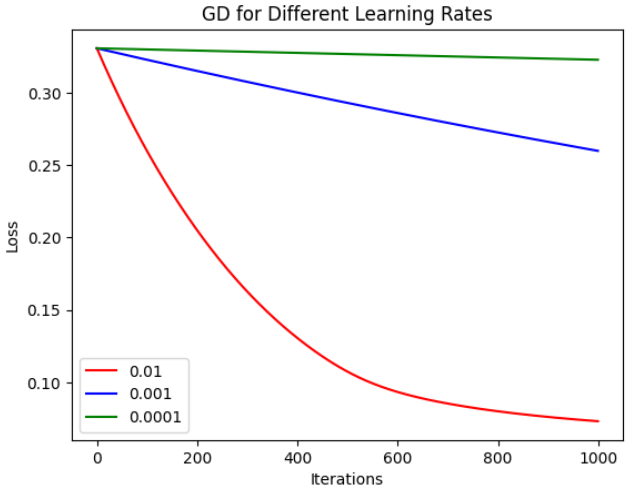


c. Stochastic Gradient Descent:

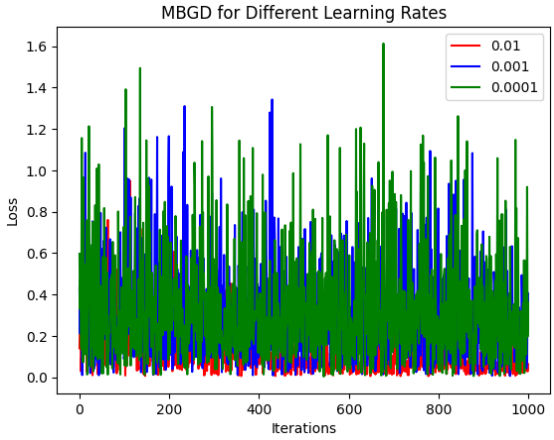


Threshold 0.5:-

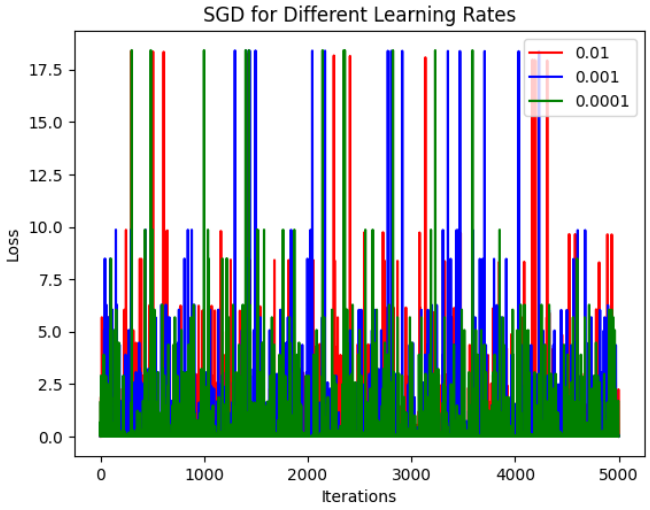
a. Batch Gradient Descent:



b. Mini-Batch Gradient Descent:

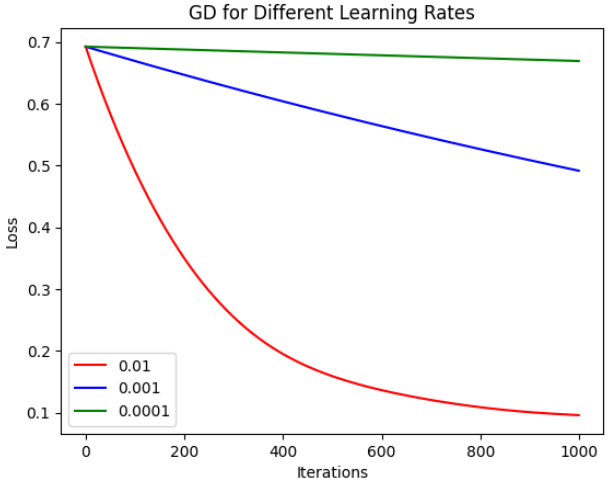


c. Stochastic Gradient Descent:

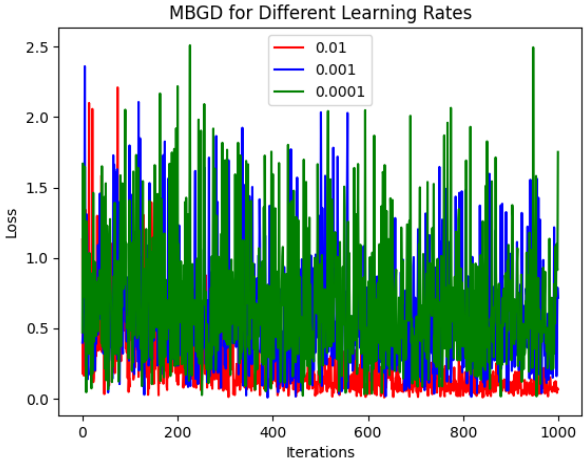


Threshold 0.7:-

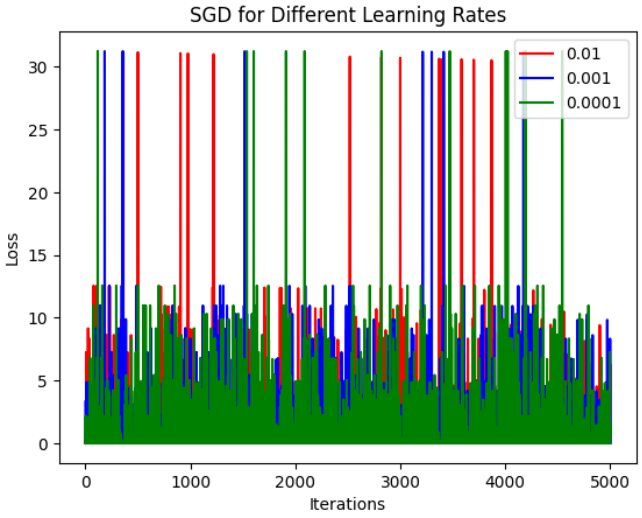
a. Batch Gradient Descent:



b. Mini-Batch Gradient Descent:



c. Stochastic Gradient Descent:



Learning Task 2: (LR2 Model):

Essentially we are following the same process as above. But here we are performing the logistic regression by replacing the missing values with their mode (Feature Engineering Task 1) and also after normalizing the data (Feature Engineering Task 2). (Z- Normalization).

For Threshold = 0.3:-

Batch Gradient Descent

F-score: 0.824875816253123
Accuracy(%): 87.328757361423
Recall 0.87187687578615123
precision 0.76546765312121

Mini-Batch Gradient Descent

F-score: 0.83895874712311
Accuracy(%): 84.19758751231321
Recall 0.8290687152812
precision 0.798687587121231

Stochastic Gradient Descent

F-score: 0.8278145695364238
Accuracy(%): 86.35170603674541
Recall 0.8865248226950354
precision 0.7763975155279503

For Threshold = 0.5:-

Batch Gradient Descent

F-score: 0.889868764121212
Accuracy(%): 99.1876876476412
Recall 0.82324239864819
precision 0.8912969862313

Mini-Batch Gradient Descent
F-score: 0.9570552147239264
Accuracy(%): 98.60194174757281
Recall 0.9176470588235294
precision 0.86764615246152

Stochastic Gradient Descent
F-score: 0.8571428571428571
Accuracy(%): 98.23884514435696
Recall 0.8723404255319149
precision 0.8424657534246576

For Threshold = 0.7:-

Batch Gradient Descent
F-score: 0.92367477619861
Accuracy(%): 92.23986875174
Recall 0.81974102545321
precision 0.897975176471

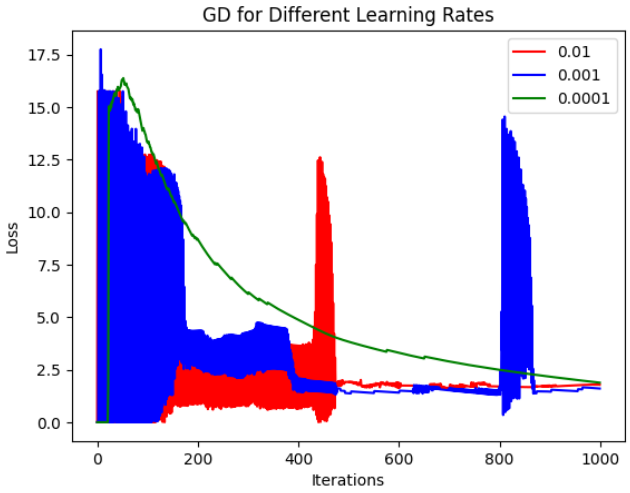
Mini-Batch Gradient Descent
F-score: 0.9170552147239264
Accuracy(%): 93.2986719628751
Recall 0.891979817527111
precision 0.9121398758172

Stochastic Gradient Descent
F-score: 0.91280969861928612
Accuracy(%): 91.2908686471
Recall 0.87527647165421
Precision 0.934986176461

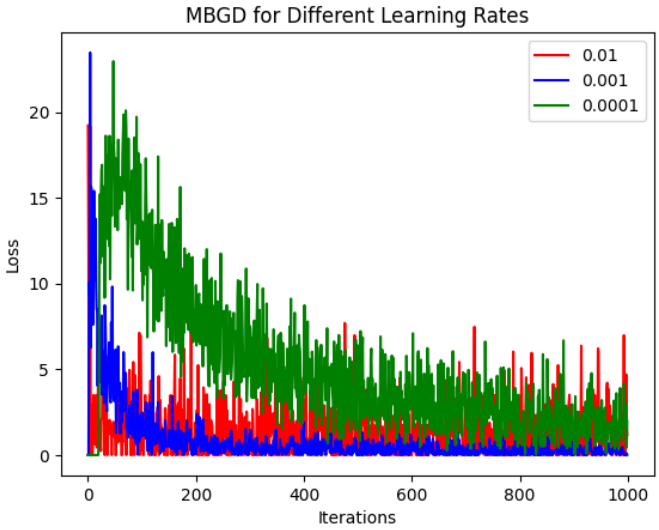
The plots of accuracy for three different learning rates η = 0.01, 0.001, 0.0001 in LR2 Model for all the three Descents is shown below for various thresholds - 0.3, 0.5 and 0.7. :-

Threshold 0.3:-

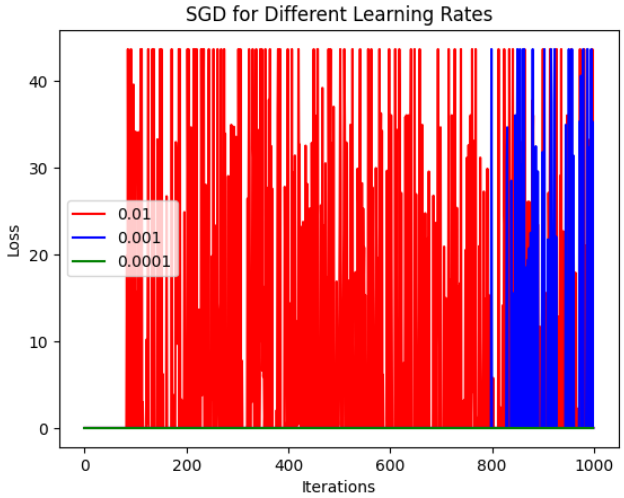
a. Batch Gradient Descent:



b. Mini-Batch Gradient Descent:

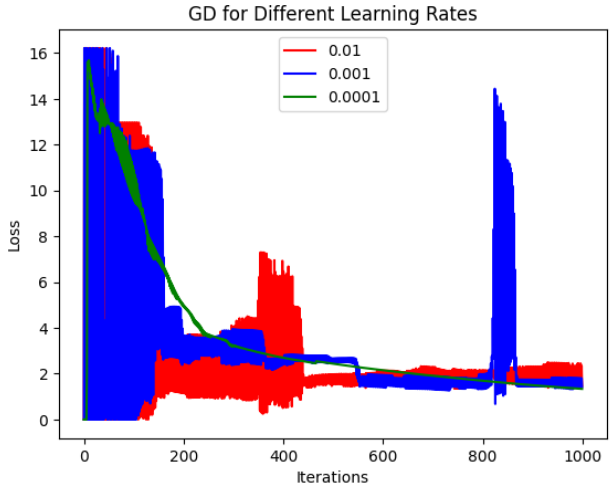


c. Stochastic Gradient Descent:

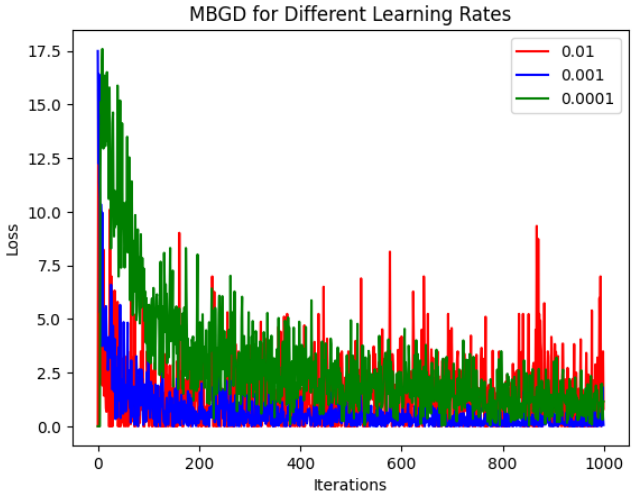


Threshold 0.5:-

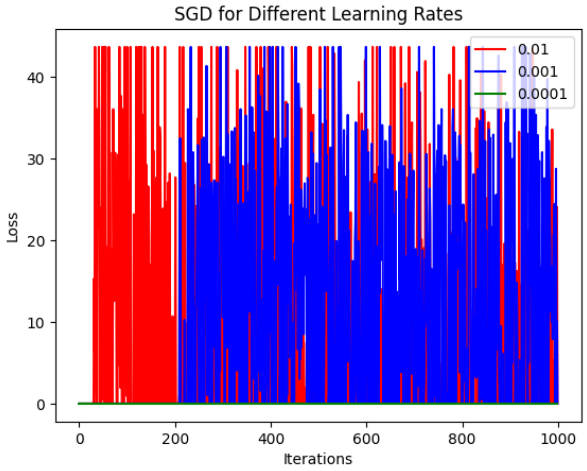
a. Batch Gradient Descent:



b. Mini-Batch Gradient Descent:

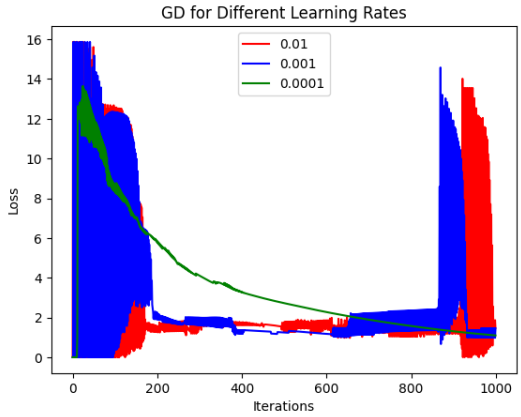


c. Stochastic Gradient Descent:

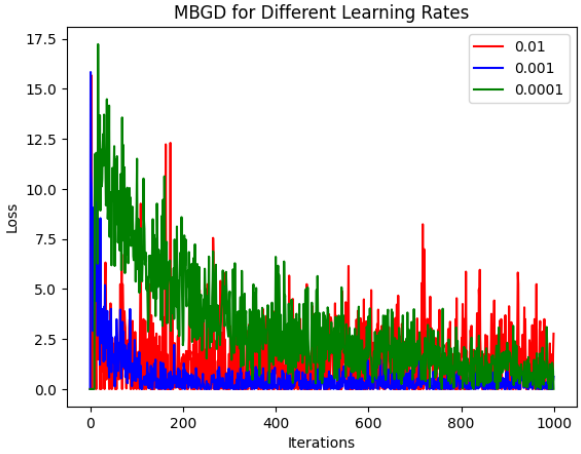


Threshold 0.7:-

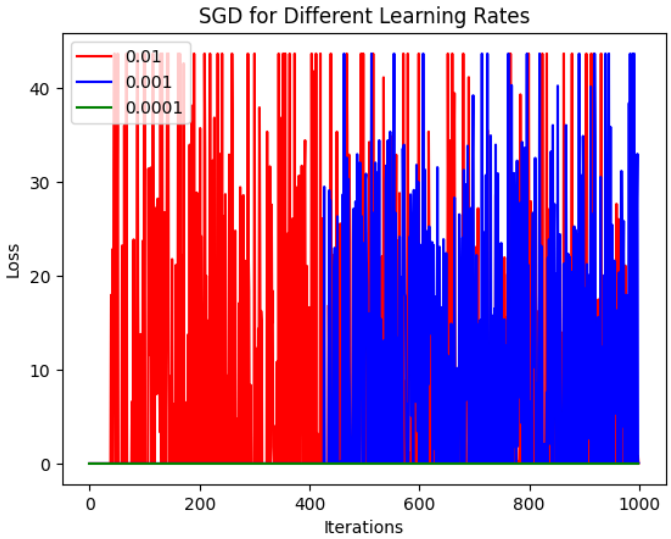
a. Batch Gradient Descent:



b. Mini-Batch Gradient Descent:



c. Stochastic Gradient Descent:



As we can see, LR2 Model gives better results and accuracy as compared to LR1 Model because normalizing the data and filling and using the missing data leads to more accurate or better results.

Part D – Comparative Study

The performance metrics of all 10 models is as follows:-

Model Name	Test accuracy	Precision	Recall
PM1	87.4015748031496	0.7189542483660131	0.9565217391304348
PM2	86.08923884514435	0.7474747474747475	0.9801324503311258
PM3	98.9002624671916	0.9712230215827338	0.9712230215827338
PM4	91.60104986876641	0.8987341772151899	0.8987341772151899
FLMD1	96.8503937007874	0.965034965034965	0.9517241379310345
FLMD2	96.58792650918635	0.9444444444444444	0.9645390070921985
LR1	89.4278910542312	0.956725309617612	0.934208917512681
LR2	99.1286301651297	0.9275712757127571	0.9347289715928361

The best performing model is **“LR2”**. This is because:-

Out of all the models, LR2 seems to be the best performing model with a very high accuracy (0.99). This is because normalization of the data prevents the prioritization of one feature over another by scaling them down to a particular range (-3,3) . We also prevent overfitting by randomly shuffling the data. This also helps remove the bias. All these methods help the model LR2 (LR2 with Gradient Descent) to converge faster and therefore provide a higher accuracy. Hence for our given dataset LR2 seems to suit better despite us assuming it to be sigmoid function.

PM3 is the next best model with the next highest accuracy. This could be because it's normalized and unbiased for the same reasons as above . Also the perceptron model is a non-parametric model, which means that it makes no assumptions about the underlying distribution of the data. This makes it more flexible and able to handle a wider range of datasets.