MACHINE LEARNING - BITS F464

SECOND SEMESTER 2022-23

ASSIGNMENT - 1

DATE: 26/03/23

SUBMITTED BY

NAME - ID

Darshan Chandak - 2020A7PS2085H
Mufaddal Jiruwala - 2020A7PS1720H
Sai Hemanth Ananthoju - 2020A7PS0116H

Processing Unnormalized Data:

We first pre-process the data after extracting it from the dataset. We make a separate DataFrame for the target attribute, and do a test-train split of 67% training data to 33% testing data.

There are certain missing values in the given dataset. We impute the missing values with the corresponding mean values of the column.

For Part A,

We assign the target attribute, i.e., predicting if a tumour is Malignant or Benign some value.

Malignant is assigned -1, and Benign is assigned 1.

For Part B and Part C,

We assign the target attribute, i.e., predicting if a tumour is Malignant or Benign some value.

Malignant is assigned 0, and Benign is assigned 1.

Processing Normalized Data:

Normalization in machine learning is the process of scaling numerical values so that they fall within a specified range, such as 0 to 1. Scaling data is important because features with very large or small values can skew the results of predictive modelling. In our case, we are **standardizing** our data. This is also known as Z-score Normalization. This is done so that they have a mean of zero and a standard deviation of one.

This technique is useful for Machine Learning Algorithms that use Gradient Descent as an optimisation technique. This makes the features unitless and reduces the effect of large outliers.

In our case, we are normalising the data using the following function:

$$z = \frac{x_i - \mu}{\sigma}$$

Where μ is the mean, and σ is the standard deviation.

Part A - Perceptron Algorithm

The Perceptron algorithm is a linear machine learning algorithm for binary classification tasks.The Perceptron algorithm works by assigning random weights to the input features and adding a bias term. The weighted sum of the inputs and the bias is then passed through an activation function, usually a step function, that outputs either 0 or 1. The output is compared with the true label and the weights are updated accordingly. The algorithm repeats this process for each row of data until it converges to a solution or reaches a maximum number of iterations.

$$w_i^{(k+1)} = w_i^k + (ita).(y_i - y_o)(x_i)$$

w is the weight vector, (ita) is the learning rate, (yi) is the true label, (yo) is the predicted label, x is the input vector and k is the iteration index.

The formula shows that the algorithm adjusts the weights and biases by adding a fraction of the error multiplied by the input vector. The error is the difference between the true label and the predicted label. The learning rate controls how much the algorithm changes the weights and biases at each step.

Learning Task 1: Build a classifier (Perceptron Model - PM1) using the perceptron algorithm. Figure out whether the data set is linearly separable by building the model. By changing the order of the training examples, build another classifier (PM2) and outline the differences between the models – PM1 and PM2.

Perceptron on Unnormalized Data (PM1):

We obtain our hyperplane through 10 test-train splits.
After obtaining the hyperplane and classifying our training data points, we calculate our performance metrics.

We use the formula as defined above on unnormalized data, and we obtain the results as follows:

	1	2	3	4	5	6	7	8	9	10
accuracy	0.952128	0.898936	0.904255	0.835106	0.877660	0.893617	0.856383	0.904255	0.872340	0.941489
precision	0.933884	0.981651	0.937500	0.937500	0.916667	0.896552	0.989796	0.930435	0.950000	0.930233
recall	0.991228	0.862903	0.905172	0.782609	0.894309	0.928571	0.788618	0.914530	0.833333	0.983607

From the above results, we can see that we obtain the best accuracy of 95.21% in split 1.

Perceptron on Unnormalized Data with Permuted Order (PM2):

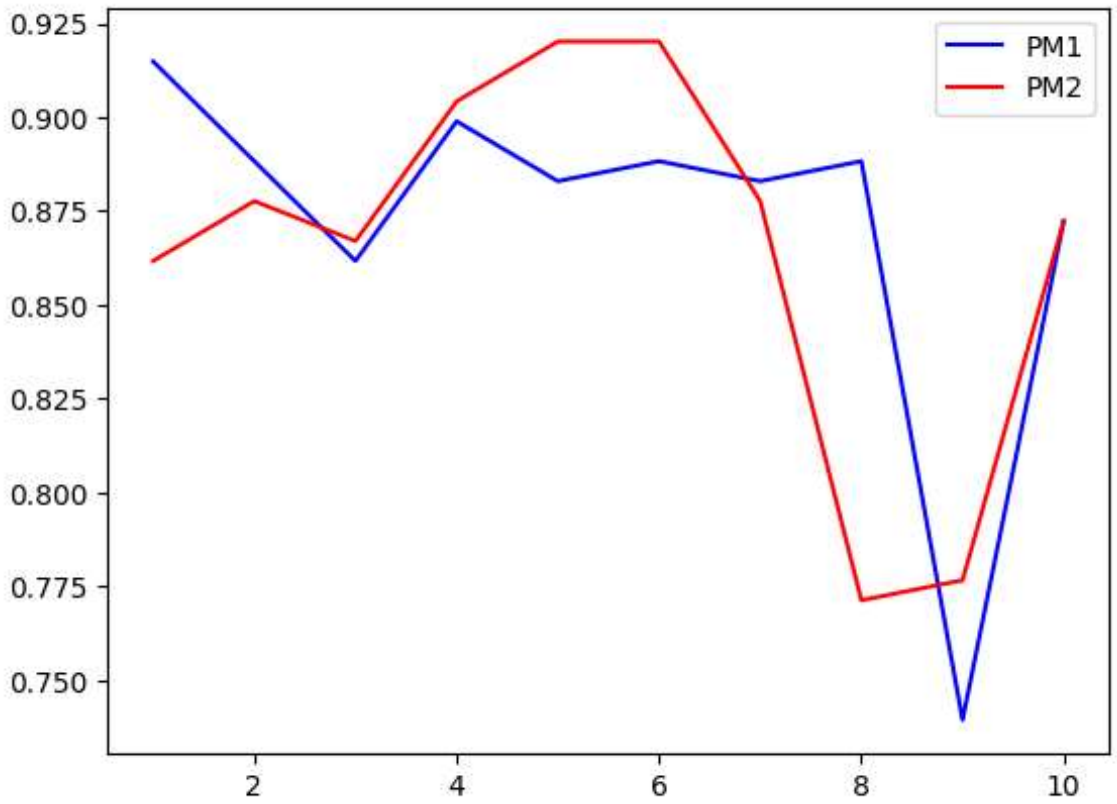
Here, we are changing the order of the training examples, and using the same method as described above to obtain a hyperplane.
We again perform 10 test-train splits and obtain different hyperplanes based on the training examples selected. We then calculate the performance metrics for this.

In this case, we obtain the results for each of the splits as follows:

	1	2	3	4	5	6	7	8	9	10
accuracy_perceptron	0.909574	0.882979	0.819149	0.840426	0.930851	0.920213	0.904255	0.787234	0.776596	0.867021
precision_perceptron	1.000000	0.990385	0.836066	0.797203	0.958333	0.894309	0.948718	0.963855	0.950000	0.953271
recall_perceptron	0.850877	0.830645	0.879310	0.991304	0.934959	0.982143	0.902439	0.683761	0.666667	0.836066

From the table, we can see that the maximum accuracy of 93.08% is obtained in split 5.

We then compare the results obtained by PM1 and PM2:



X-axis represents the Split number and the Y-axis represents accuracy. We see that PM2 has managed to obtain higher accuracy in split 5.

Learning Task 2: Build a classifier (Perceptron Model - PM3) using the perceptron algorithm on the normalized data and figure out the difference between the two classifiers (PM1 and PM3).

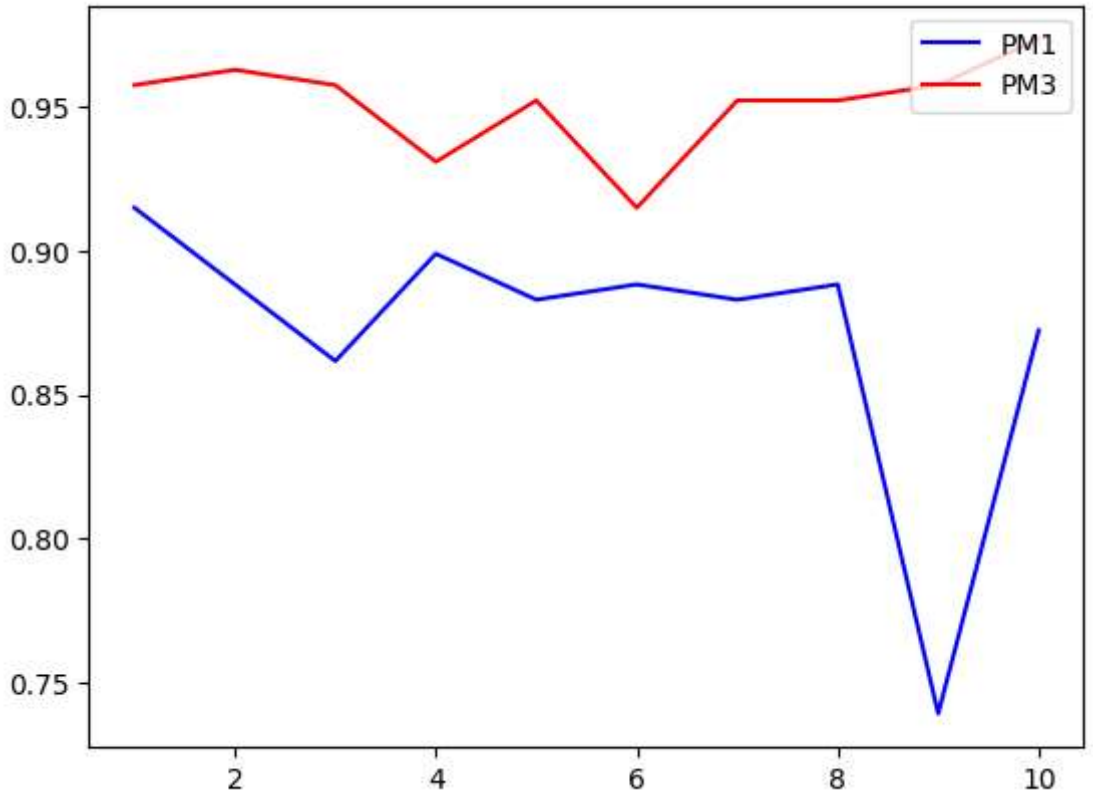
Perceptron on Normalized Data (PM3):

In this case, we use the perceptron algorithm on Normalized data. The Normalization procedure is described above.

After applying the perceptron algorithm as above, we see that the accuracy per split increases significantly as compared to unnormalized data:

	1	2	3	4	5	6	7	8	9	10
accuracy	0.984043	0.984043	0.978723	0.989362	0.973404	0.989362	0.973404	0.978723	0.984043	0.989362
precision	0.983871	1.000000	1.000000	0.991071	0.971963	1.000000	0.975410	0.991379	0.982609	1.000000
recall	0.991870	0.973684	0.969231	0.991071	0.981132	0.982456	0.983471	0.974576	0.991228	0.982759

We see that the maximum accuracy of 98.94% is obtained in split 4.



As explained above, the accuracy of the perceptron algorithm on normalised data (PM3) is much higher than that of unnormalised data (PM1).

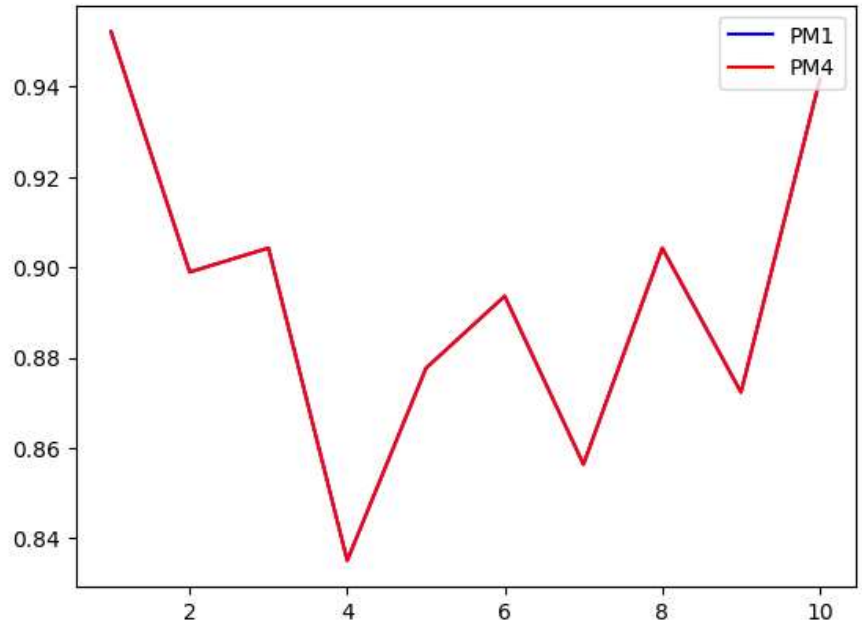
Perceptron on Unnormalized Data with Shuffled features (PM4):

In this case, we are shuffling the features of the dataset, and building a classifier.

After building the classifier, we observe the results as follows, across 10 splits:

	1	2	3	4	5	6	7	8	9	10
accuracy	0.952128	0.898936	0.904255	0.835106	0.877660	0.893617	0.856383	0.904255	0.872340	0.941489
precision	0.933884	0.981651	0.937500	0.937500	0.916667	0.896552	0.989796	0.930435	0.950000	0.930233
recall	0.991228	0.862903	0.905172	0.782609	0.894309	0.928571	0.788618	0.914530	0.833333	0.983607

There is no difference between the accuracies of PM1 and PM4. Since just shuffling the feature vector does not impact the performance of the model.



As explained above, since there is no difference between the accuracies of PM1 and PM4, the graph overlaps.

Part B - Fisher's Linear Discriminant Analysis

Fisher's Linear Discriminant Analysis (LDA) is a method used for Dimensionality Reduction and classification of data into two or more classes.

The main goal of LDA is to find a linear combination of features that can best separate different classes in the data. The method maximizes the ratio of the between-class variance to the within-class variance, which means that it tries to find a projection of the data that maximizes the distance between the means of different classes while minimizing the variation within each class.

Building Fisher's linear discriminant model (FLDM1):

We have two classes in our dataset. One corresponds to Malignant Tumour and the other corresponds to Benign Tumour.

We calculate the variance of both classes and add them to get Sw or the in-class variance.

We calculate the in-class variance using the following formula:

$$S_w = \left(\frac{1}{N_1} \sum_{n=1}^{N_1} \left(x_n - m_1\right)\left(x_n - m_2\right)^T \right) + \left(\frac{1}{N_2} \sum_{n=1}^{N_2} \left(x_n - m_2\right)\left(x_n - m_2\right)^T \right)$$

Where m1 is the mean of the first class and m2 is the mean of the second class.

We now seek to maximize this in-class variance.

We project the given data space into a single dimension using a transformation function. Now after projecting the data points to one dimension we calculate the projected mean and projected standard deviation of both classes.

Mean before projecting:

$$m_1^{'} = \left(\frac{1}{N_1} \sum_{n=1}^{N_1} x_n \right)$$

Mean after projecting:

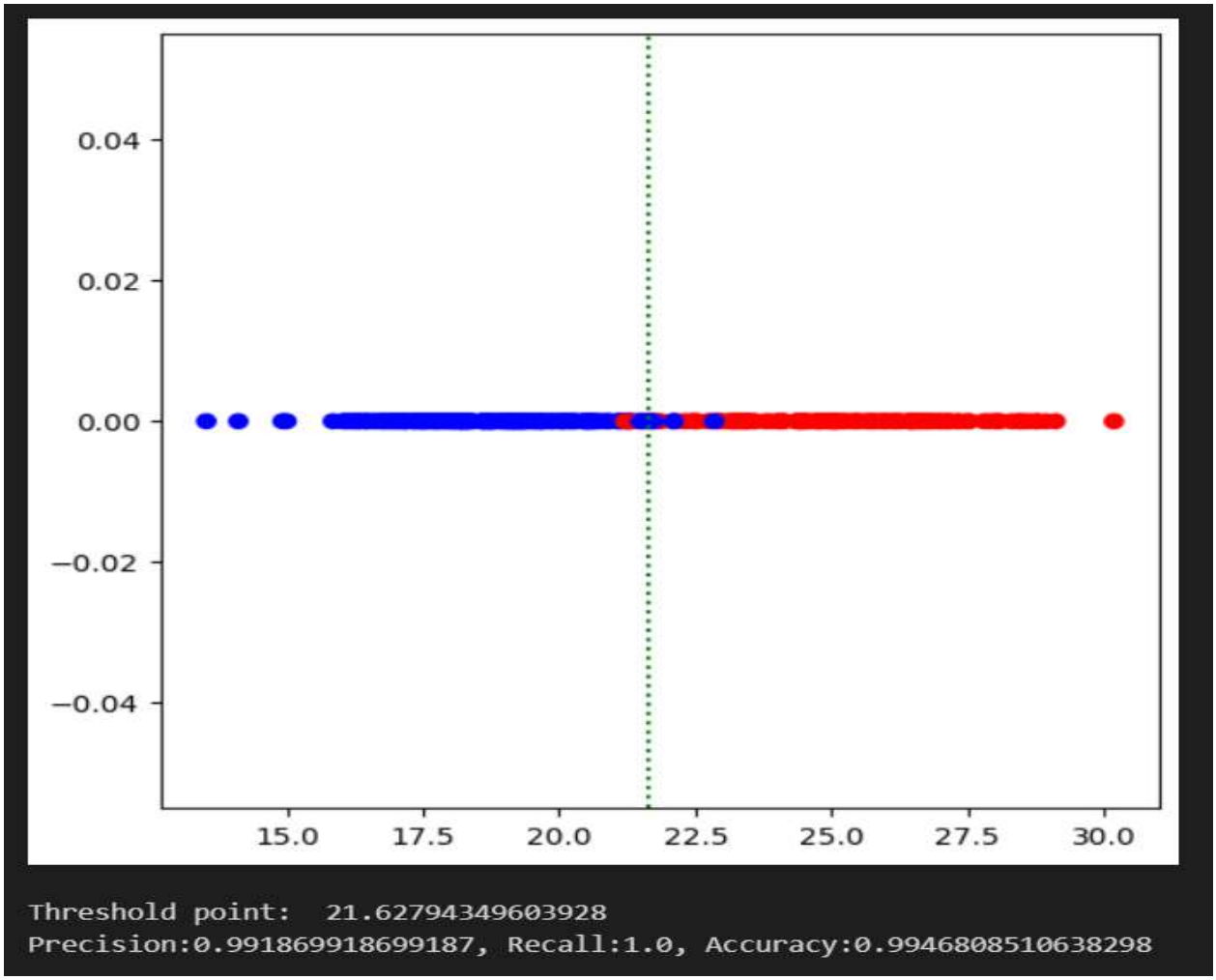
$$m_1 = \left(\frac{1}{N_1} \sum_{n=1}^{N_1} w^T . x_n \right)$$

Learning Task 1: Build Fisher's linear discriminant model (FLDM1) on the training data and thus reduce 32 dimensional problem to univariate dimensional problem. Find out the decision boundary in the univariate dimension using generative approach. You may assume gaussian distribution for both positive and negative classes in the univariate dimension.

We run the Fisher's linear discriminant model (FLDM1) for the datapoints.

The **blue** data points are for the **Benign tumour** and the **red** data points are for **Malignant tumour.**

The **green** line defines the **decision boundary** to separate the data points of two classes or to separate the two kinds of tumours.

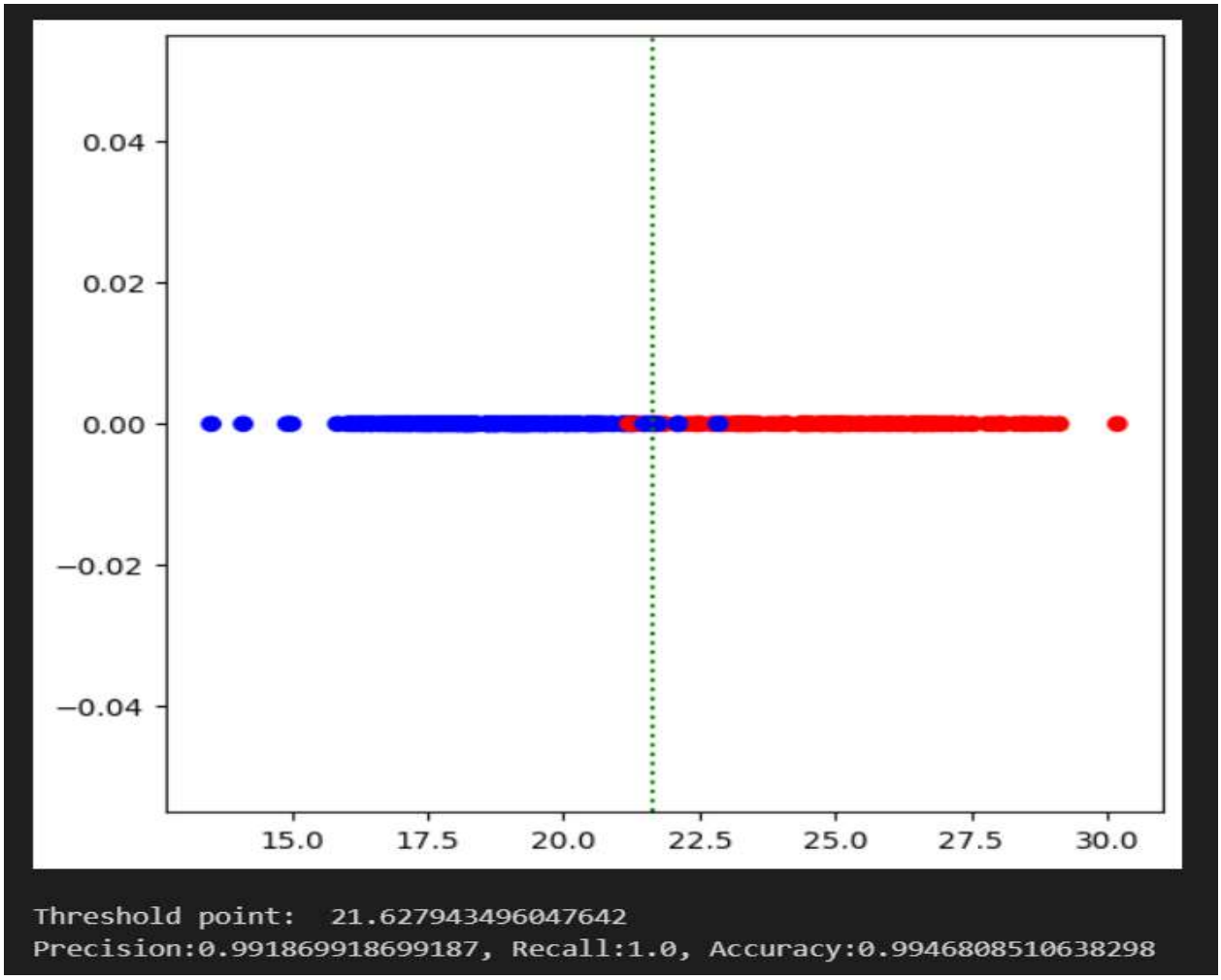


We obtain maximum accuracy of 99.46% in Split 4.
These are the projected data points on x-axis.

Building Fisher's linear discriminant model (FLDM2):

Learning Task 2: Change the order of features in the dataset randomly. Equivalently speaking, for an example of feature tuple (f1, f2, f3, f4, . . , f32), consider a random permutation (f3, f1, f4, f2, f6, ….., f32) and build the Fisher's linear discriminant model (FLDM2) on the same training data as in the learning task 1. Find out the decision boundary in the univariate dimension using generative approach and you may assume gaussian distribution for both positive and negative classes in the univariate dimension.

Here we just permute the order of the feature tuple and train our model to build another Fisher's linear discriminant model (FLDM2).



We obtain maximum accuracy of 99.46% in Split 4.
These are the projected data points.

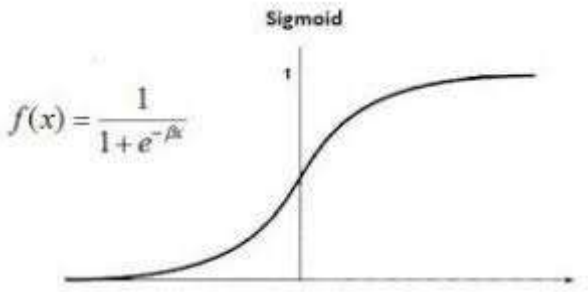
There is no change in the results of FLDM1 and FLDM2, since reordering the features amongst themselves has no effect on the prediction of the model.

Part C - Logistic Regression

Logistic Regression is a statistical method that we use to fit a regression model when the response variable is binary. This means that the outcome we are trying to predict can only take two values, such as 0 and 1.

Logistic Regression works by using the logistic function. This function takes in any value and outputs a number between 0 and 1. We can use this function to calculate the probability of an event happening.

The logistic function has a sigmoid curve, which means that as the value of the input variable increases, the probability of achieving the target variable also increases. However, the rate of increase slows down as the value of the input variable gets larger.



Logistic Regression can also be used with multiple input variables or an input vector.

Logistic Regression on Unnormalized Data (LR1):

Learning Task 1: Build a classification model (LR1) using Logistic Regression. Explain what happens to test accuracy when you vary the decision probability threshold from 0.5 to 0.3, 0.4, 0.6 and 0.7.

We use the following formula to calculate the categorical target attribute:

$$y = \frac{1}{1 + e^{-w^T x}}$$

Where w is the vector of parameters, and x is the input vector.

To calculate the Cost function or Error function, we have used the following formula:

$$E(w) = -\left(\sum_{n=1}^{N} \left(t_n \log(y_n) + (1 - t_n) \log(1 - y_n) \right) \right)$$

Where tn is the actual value of the target attribute and yn is the predicted value of the target attribute.

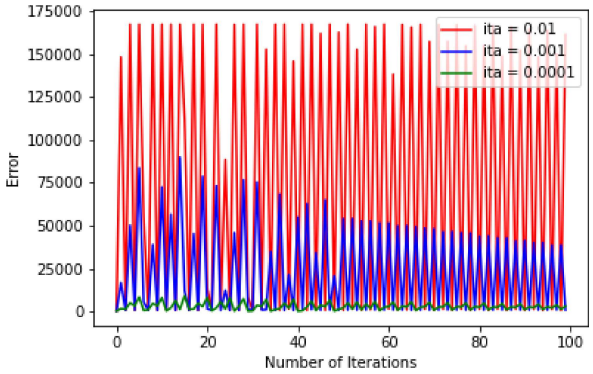
Batch Gradient Descent:

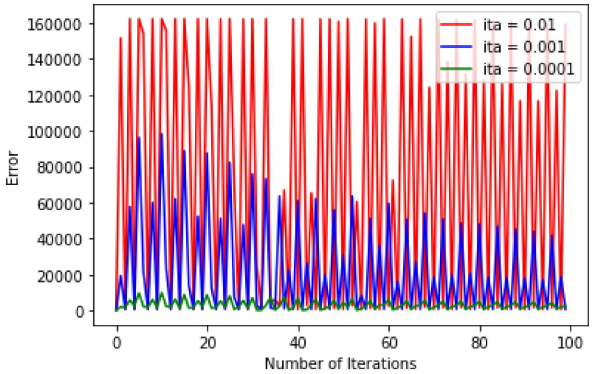
We run 100 iterations with 3 different learning rates. We use the following equation for calculating the parameter vector w:

$$w_{required}^{(k+1)} = w_{required}^{(k)} - \left(\frac{1}{N} \right).(ita).\left(\frac{d(E)}{dw} \right)$$

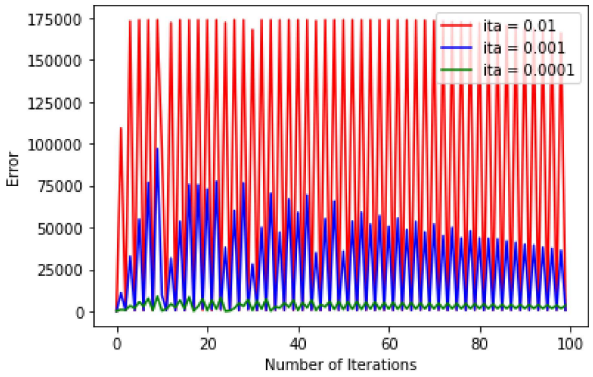
Graphs of Number of Iterations vs Error:

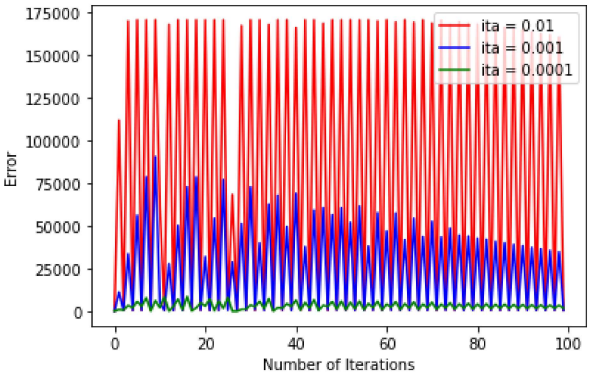
Split - 1 & 2:



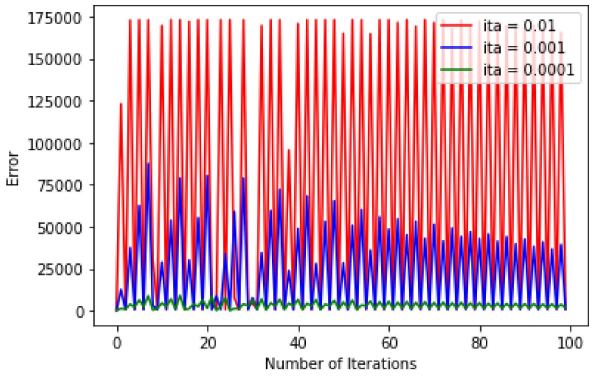


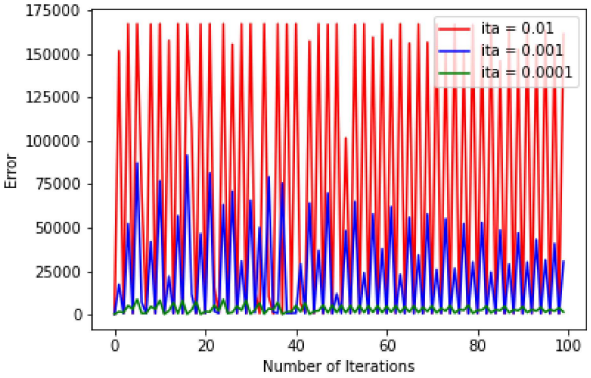
Split - 3 & 4:



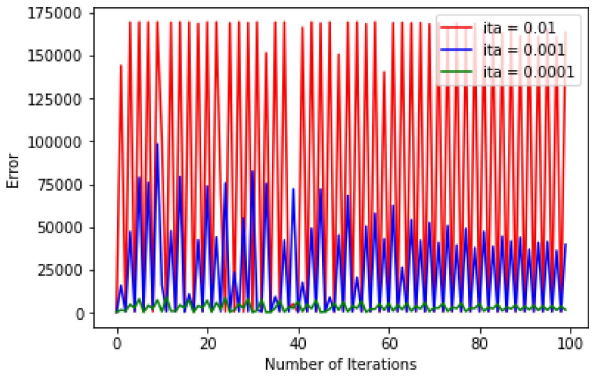


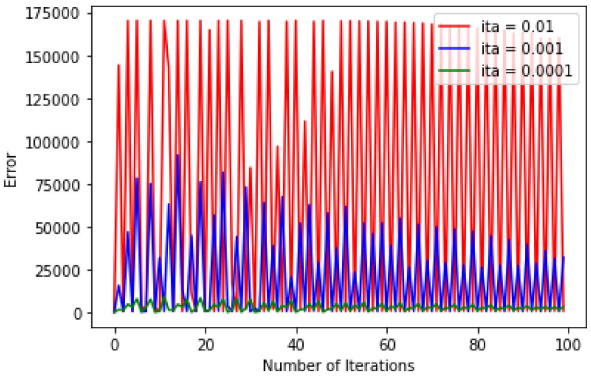
Split - 5 & 6:



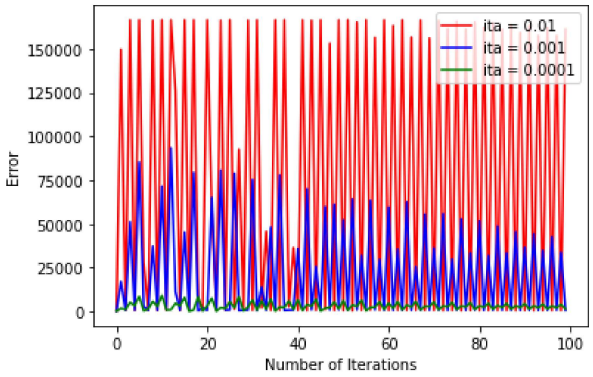


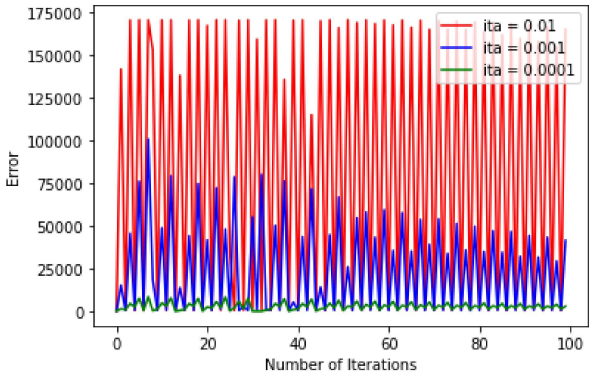
Split - 7 & 8:





Split - 9 & 10:





For all of the graphs above, we can see a drop in the magnitude of error as the number of iterations increases, and the learning rate decreases.

Now we calculate the performance metrics such as accuracy, precision and recall for our model:

$$\text{Accuracy} = \frac{(TP + TN)}{(TP + FP + TN + FN)}$$

$$\text{Precision} = \frac{True\ Positive}{True\ Positive + False\ Positive}$$

$$\text{Recall} = \frac{True\ Positive}{True\ Positive + False\ Negative}$$

Results:

We see that we get the maximum accuracy in Split 2, as we can see from the table below:

```
Split:
2

---ACCURACY---
Threshold        0.3        0.4        0.5        0.6        0.7
ita
0.0001     31.914894  31.914894  31.914894  31.914894  31.914894
0.0010     31.914894  31.914894  31.914894  31.914894  31.914894
0.0100     80.319149  80.319149  80.319149  80.319149  80.319149

---RECALL---
Threshold    0.3    0.4    0.5    0.6    0.7
ita
0.0001       0.0    0.0    0.0    0.0    0.0
0.0010       0.0    0.0    0.0    0.0    0.0
0.0100     100.0  100.0  100.0  100.0  100.0

---PRECISION---
Threshold        0.3        0.4        0.5        0.6        0.7
ita
0.0001      0.000000   0.000000   0.000000   0.000000   0.000000
0.0010      0.000000   0.000000   0.000000   0.000000   0.000000
0.0100     77.575758  77.575758  77.575758  77.575758  77.575758

Split:
```

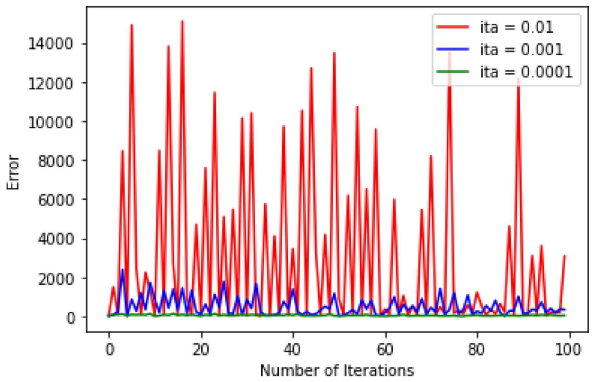
From the above results, we can see that Logistic Regression using Vector Gradient Descent with Unnormalized data gives us a maximum accuracy of 80.39% with a learning rate of 0.01.

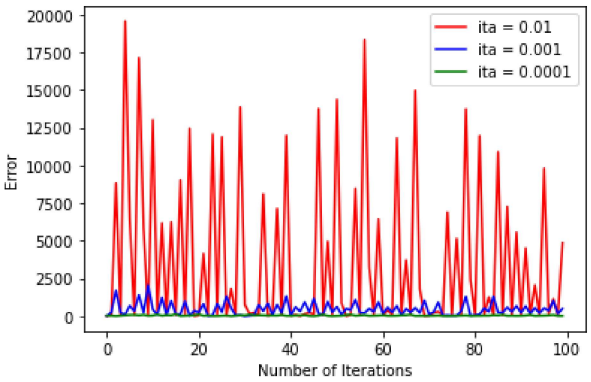
Mini - Batch Gradient Descent:

For Batch Gradient Descent, we use the same formula as we did for Vector Gradient Descent, with the only difference being that instead of using the complete dataset, we take batches of the data points.

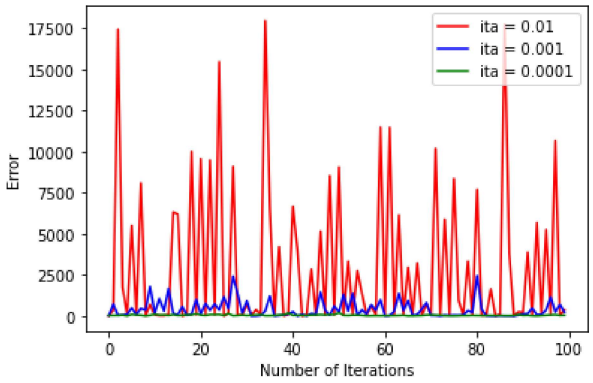
Graphs of Number of Iterations vs Error:

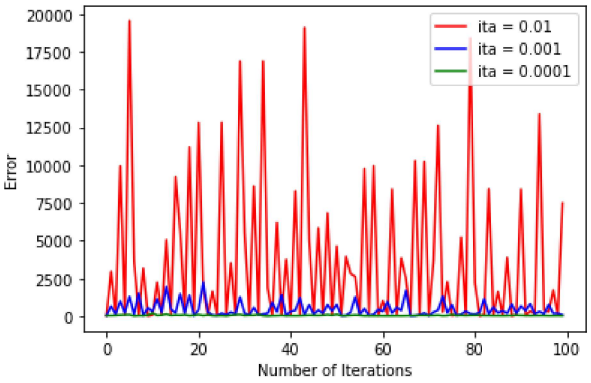
Split - 1 & 2:



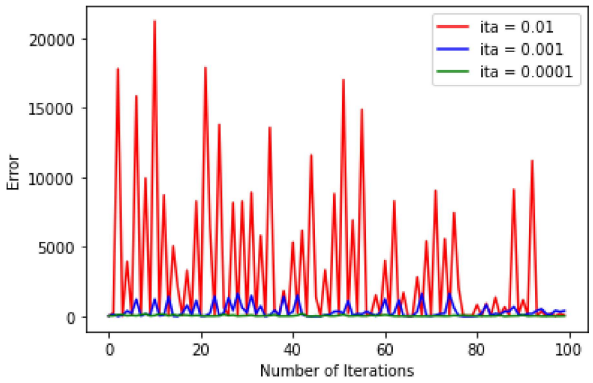


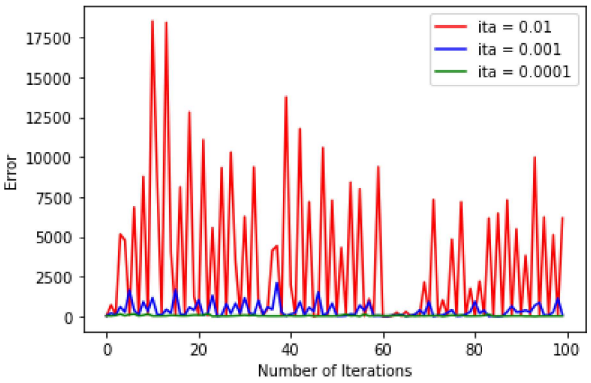
Split - 3 & 4:



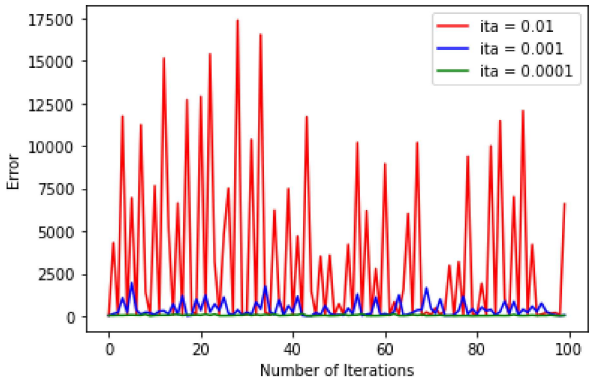


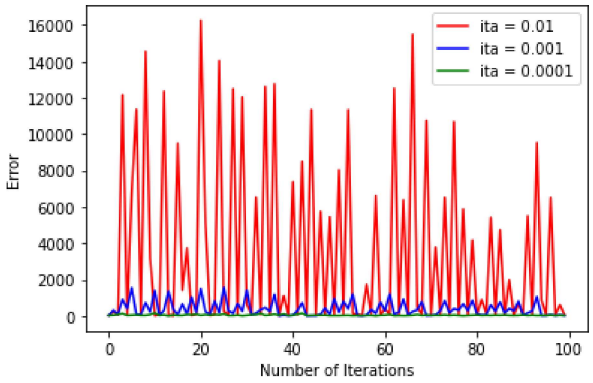
Split - 5 & 6:



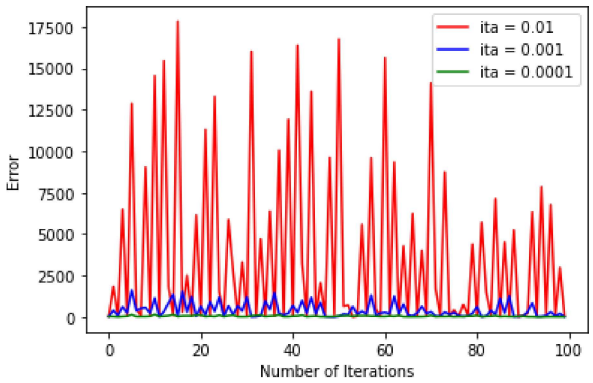


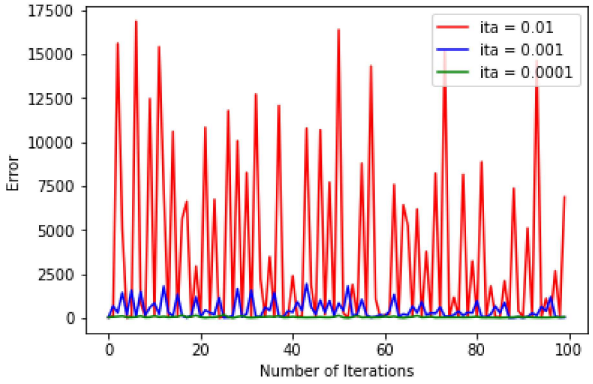
Split - 7 & 8:





Split - 9 & 10:





Results:

Now we calculate the performance metrics such as accuracy, precision and recall for our model:

```
Split:
8

---ACCURACY---
Threshold        0.3        0.4       0.5        0.6        0.7
ita
0.0001     63.297872  63.297872  8.510638  36.702128  36.702128
0.0010     63.297872  63.297872  9.042553  27.659574  36.170213
0.0100     28.191489  19.680851  9.574468   8.510638   9.574468

---RECALL---
Threshold         0.3         0.4        0.5       0.6       0.7
ita
0.0001     100.000000  100.000000  11.764706  0.000000  0.000000
0.0010     100.000000  100.000000  11.764706  0.000000  0.000000
0.0100      44.537815   31.092437  12.605042  7.563025  4.201681

---PRECISION---
Threshold        0.3        0.4        0.5        0.6       0.7
ita
0.0001     63.297872  63.297872  17.283951   0.000000  0.000000
0.0010     63.297872  63.297872  17.500000   0.000000  0.000000
0.0100     43.442623  34.905660  18.518519  12.676056  8.196721
```

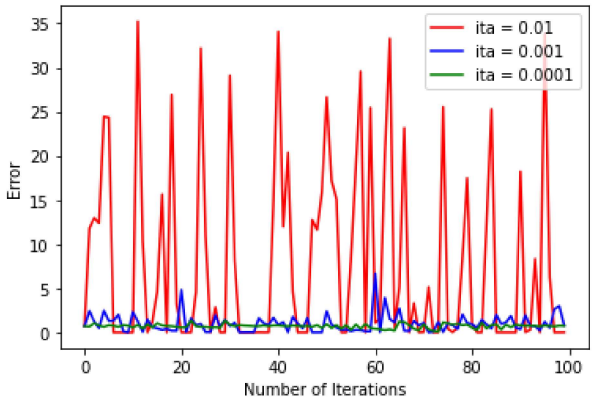
From the above metrics, we can see that we have achieved a maximum accuracy of 63.29% in split 8.

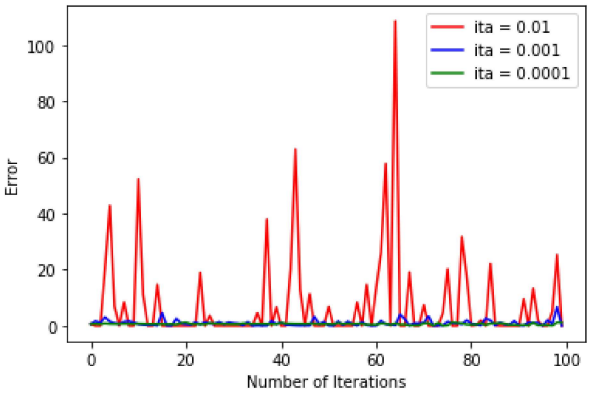
Stochastic Gradient Descent:

We use the same formula for Stochastic Gradient Descent, with the only difference being that we select random data points from our dataset to calculate our parameter values w.

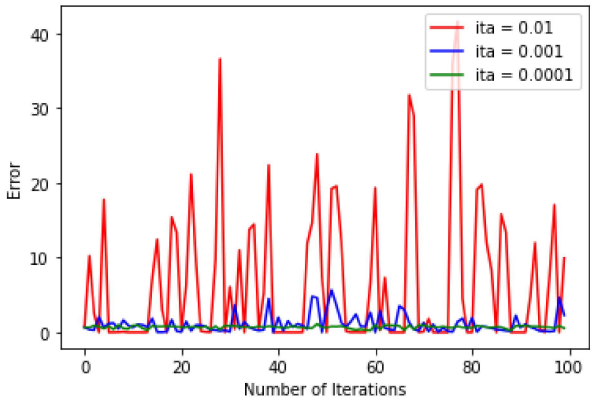
Graphs of Number of Iterations vs Error:

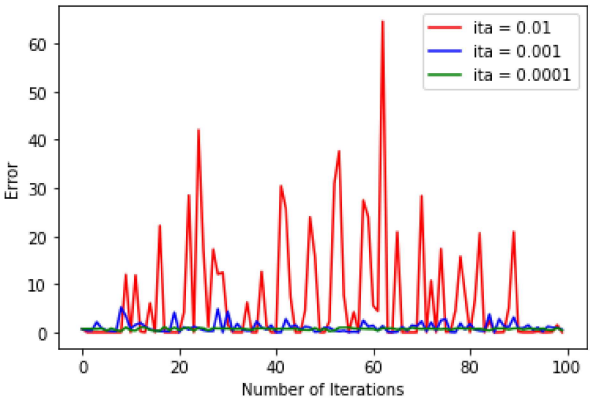
Split - 1 & 2:



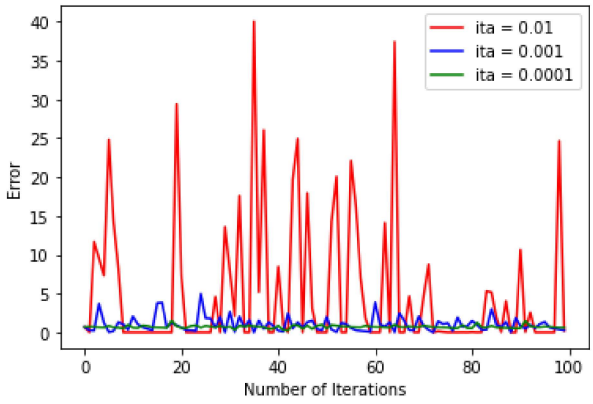


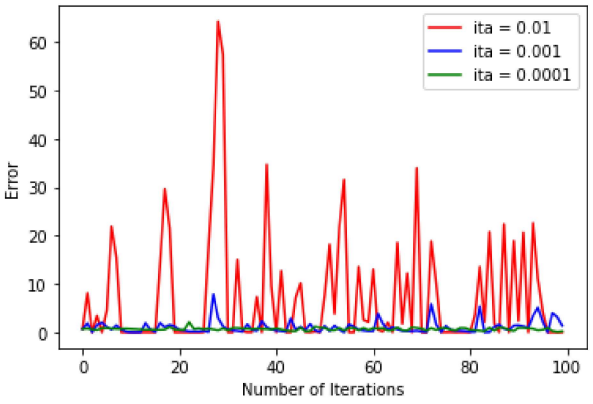
Split - 3 & 4:



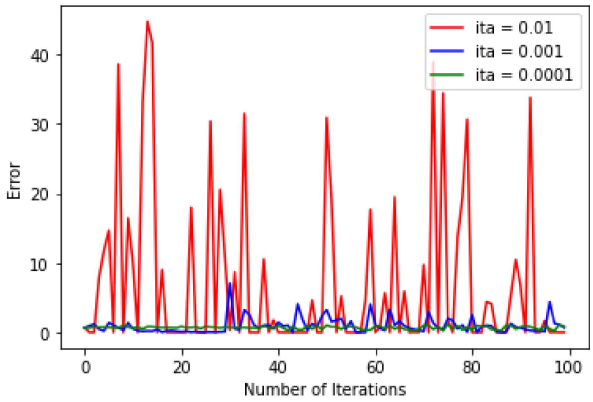


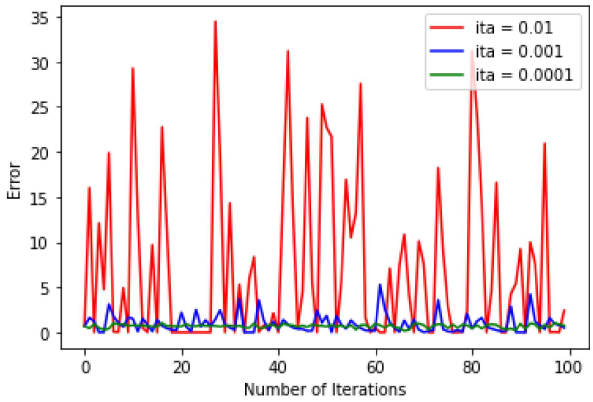
Split - 5 & 6:



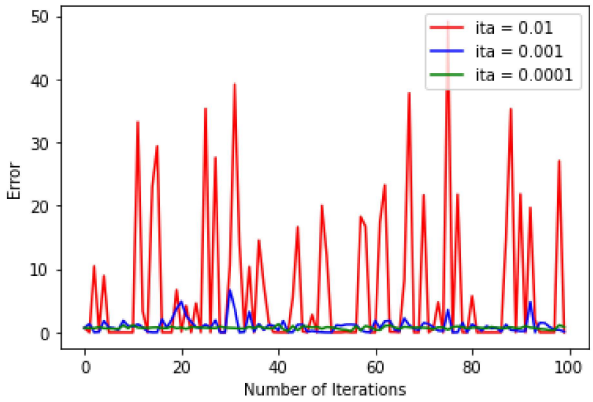


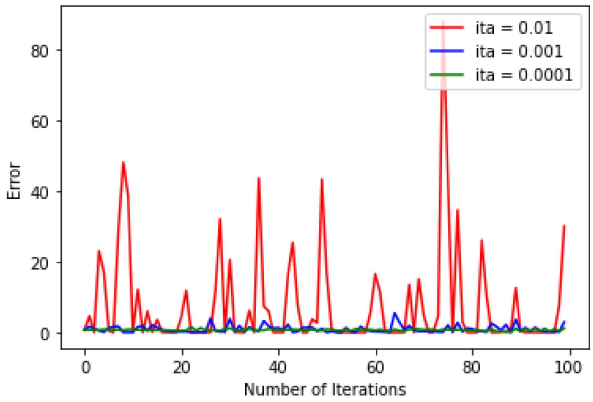
Split - 7 & 8:





Split - 9 & 10:





Results:

Now we calculate the performance metrics such as accuracy, precision and recall for our model:

```
Split:
2

---ACCURACY---
Threshold        0.3        0.4        0.5        0.6        0.7
ita
0.0001     62.234043  62.234043  86.170213  37.765957  37.765957
0.0010     62.234043  62.234043  87.765957  37.765957  37.765957
0.0100     62.234043  62.234043  49.468085  37.765957  37.765957

---RECALL---
Threshold    0.3    0.4        0.5  0.6  0.7
ita
0.0001     100.0  100.0  96.581197  0.0  0.0
0.0010     100.0  100.0  95.726496  0.0  0.0
0.0100     100.0  100.0  50.427350  0.0  0.0

---PRECISION---
Threshold        0.3        0.4        0.5  0.6  0.7
ita
0.0001     62.234043  62.234043  83.703704  0.0  0.0
0.0010     62.234043  62.234043  86.153846  0.0  0.0
0.0100     62.234043  62.234043  61.458333  0.0  0.0

Split:
```

From the above metrics, we can observe that we get maximum accuracy of 86.17% in split 2.

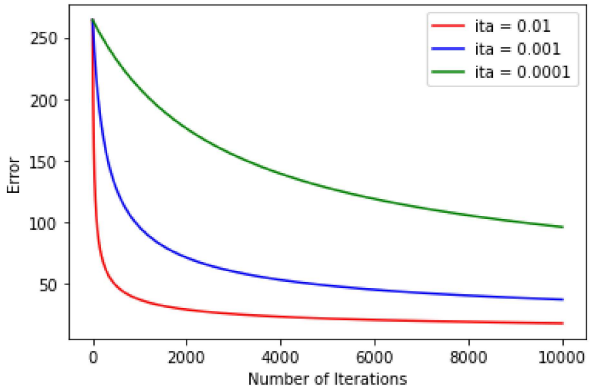
Logistic Regression on Normalized Data (LR2):

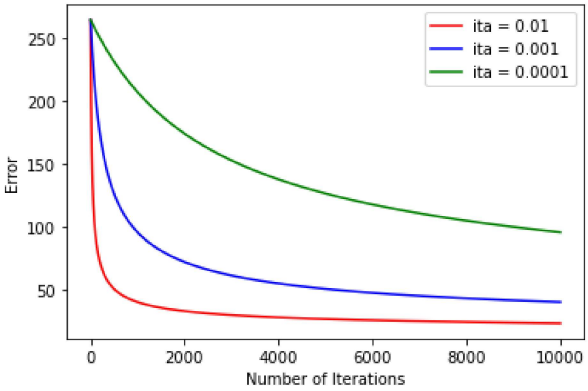
Batch Gradient Descent:

We run 10000 iterations to calculate the parameter vector w. The formula used for error calculation is the same as above (for unnormalized data).

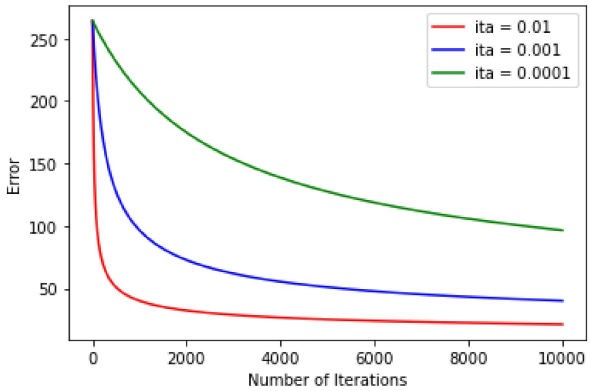
Graphs of Number of Iterations vs Error:

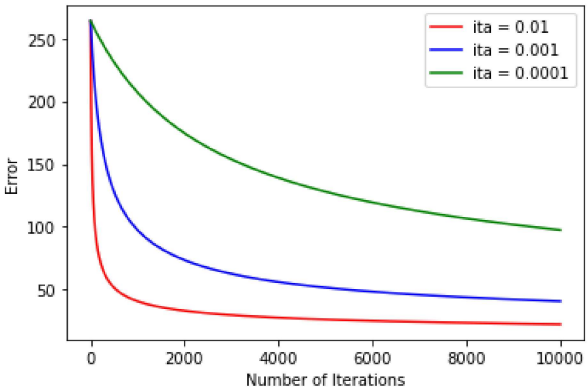
Split - 1 & 2:



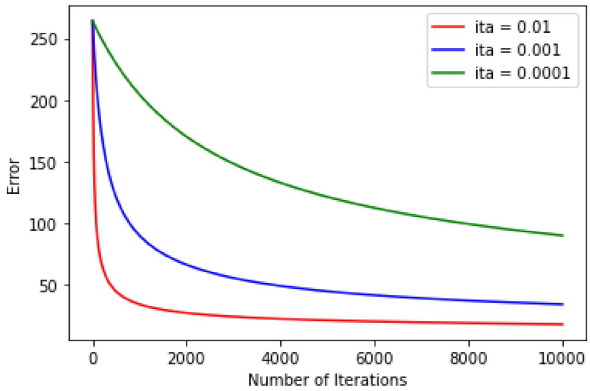


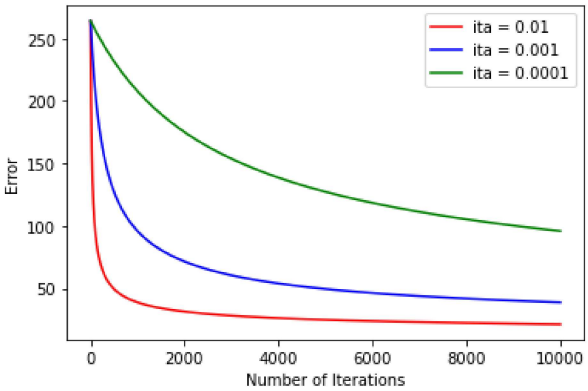
Split - 3 & 4:



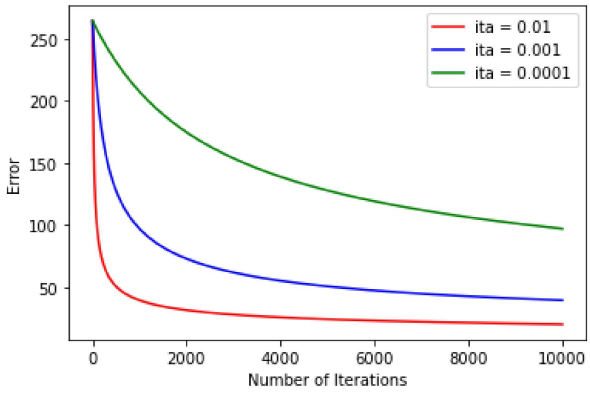


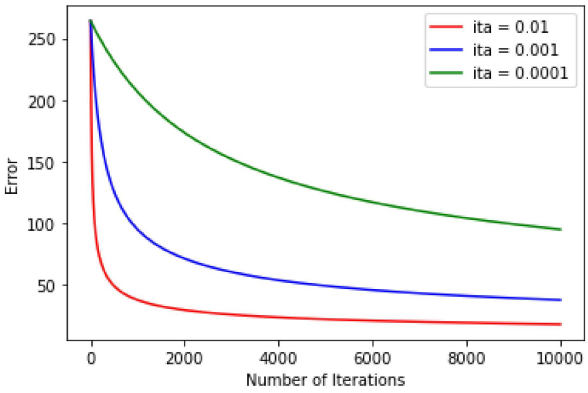
Split - 5 & 6:



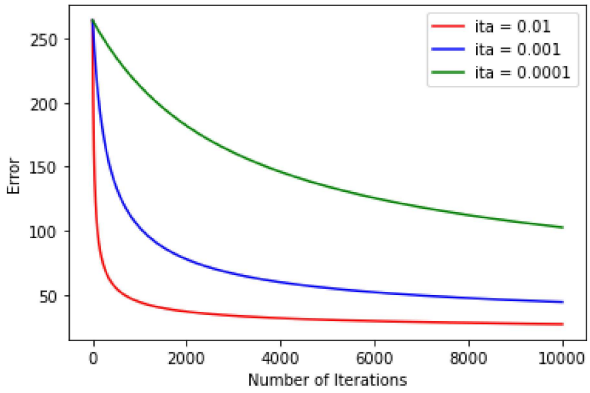


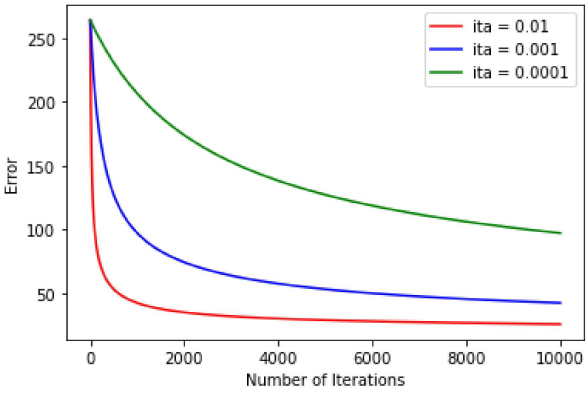
Split - 7 & 8:





Split - 9 & 10:





Results:

```
Split:
6

---ACCURACY---
Threshold        0.3        0.4        0.5        0.6        0.7
ita
0.0001     92.021277  93.617021  94.148936  91.489362  86.170213
0.0010     96.276596  97.872340  96.808511  96.276596  94.680851
0.0100     98.404255  98.936170  99.468085  99.468085  98.404255

---RECALL---
Threshold    0.3         0.4         0.5         0.6         0.7
ita
0.0001     100.0   99.074074   95.37037   86.111111  75.925926
0.0010     100.0  100.000000   95.37037   94.444444  90.740741
0.0100     100.0  100.000000  100.00000  100.000000  97.222222

---PRECISION---
Threshold        0.3        0.4        0.5        0.6    0.7
ita
0.0001     87.804878  90.677966  94.495413  98.936170  100.0
0.0010     93.913043  96.428571  99.038462  99.029126  100.0
0.0100     97.297297  98.181818  99.082569  99.082569  100.0
```

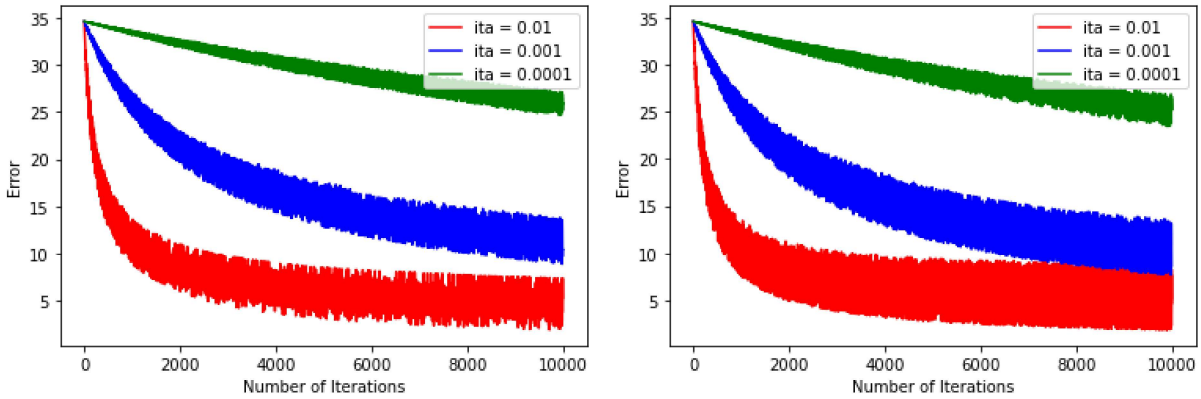
We observe a maximum accuracy of 99.46% in split 6.

Mini-Batch Gradient Descent:

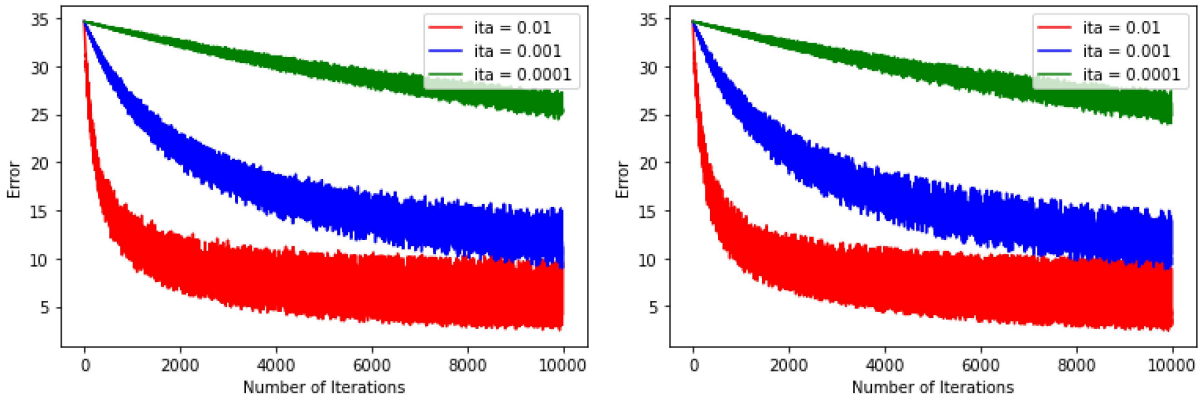
We run 10000 iterations to calculate the parameter vector w. We take data in batches from our dataset.

Graphs of Number of Iterations vs Error:

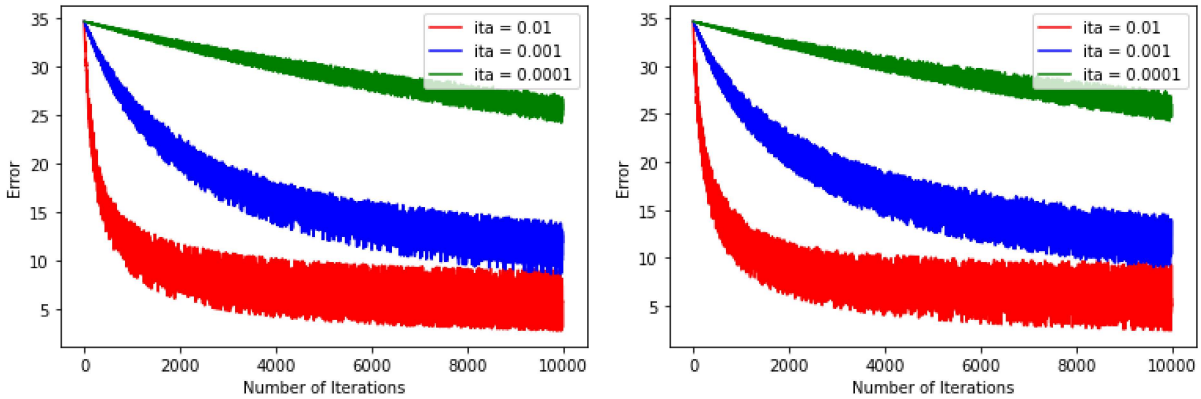
Split - 1 & 2:



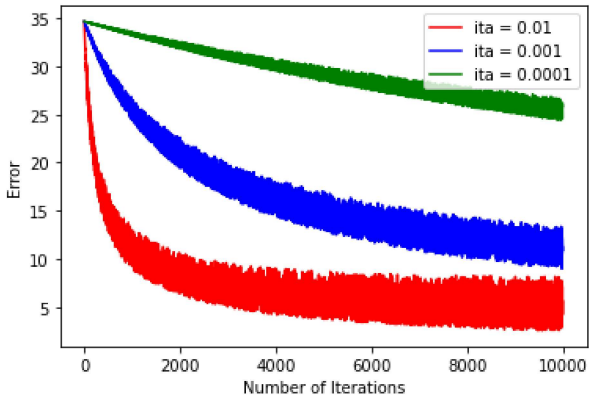
Split - 3 & 4:

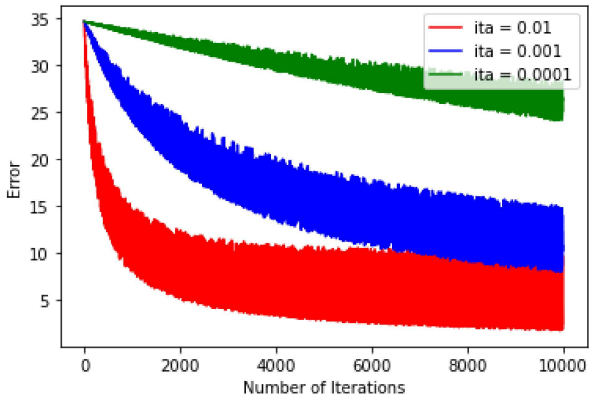


Split - 5 & 6:

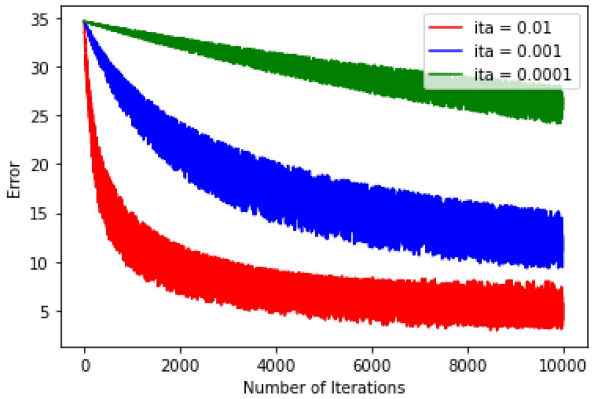


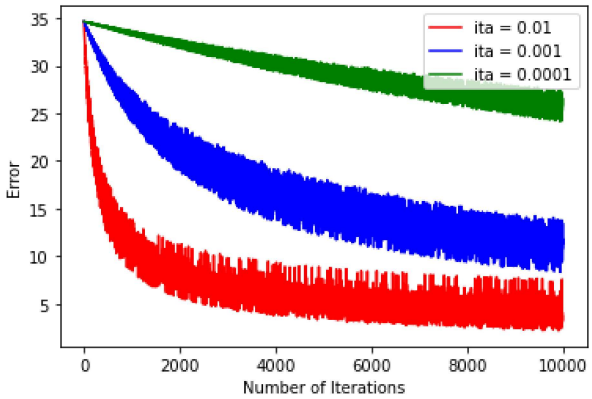
Split - 7 & 8:





Split - 9 & 10:





Results:

```
Split:
2

---ACCURACY---
Threshold        0.3        0.4        0.5        0.6        0.7
ita
0.0001     69.680851  86.702128  92.553191  59.574468  37.765957
0.0010     93.617021  94.148936  93.617021  89.893617  85.106383
0.0100     97.872340  97.872340  95.744681  94.680851  93.617021

---RECALL---
Threshold         0.3        0.4        0.5        0.6        0.7
ita
0.0001     100.000000  99.145299  92.307692  35.042735   0.000000
0.0010      98.290598  94.871795  92.307692  85.470085  77.777778
0.0100      99.145299  99.145299  94.871795  93.162393  91.452991

---PRECISION---
Threshold        0.3        0.4        0.5        0.6        0.7
ita
0.0001     67.241379  82.857143  95.575221  100.000000   0.000000
0.0010     92.000000  95.689655  97.297297   98.039216  97.849462
0.0100     97.478992  97.478992  98.230088   98.198198  98.165138
```

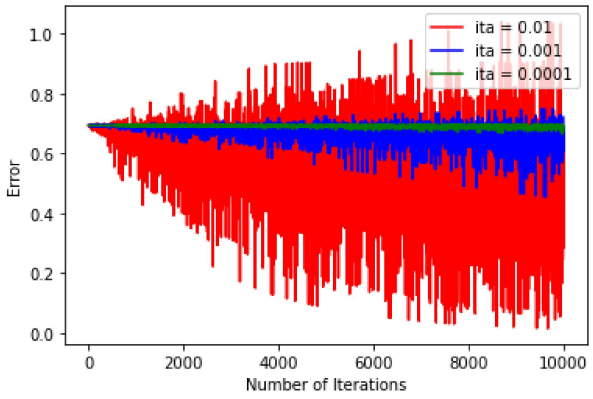
From the above metrics, we can see that the maximum accuracy we have achieved is 97.87% in Split 2.

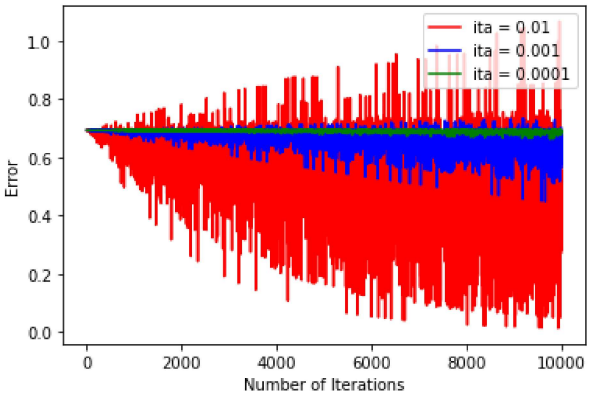
Stochastic Gradient Descent:

We run 10000 iterations to calculate the parameter vector w. We take random data points from our dataset.

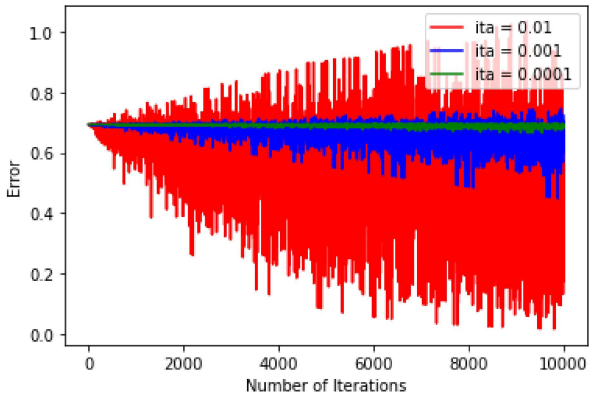
Graphs of Number of Iterations vs Error:

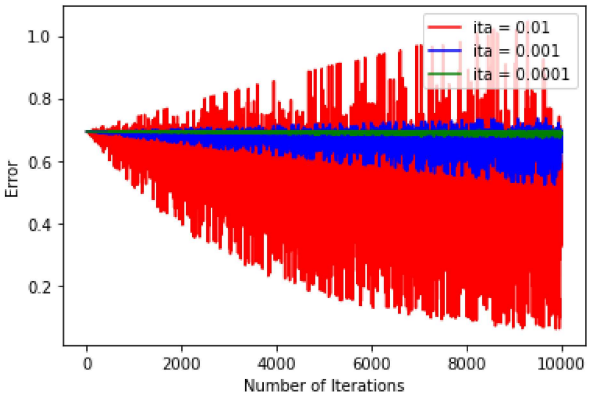
Split - 1 & 2:



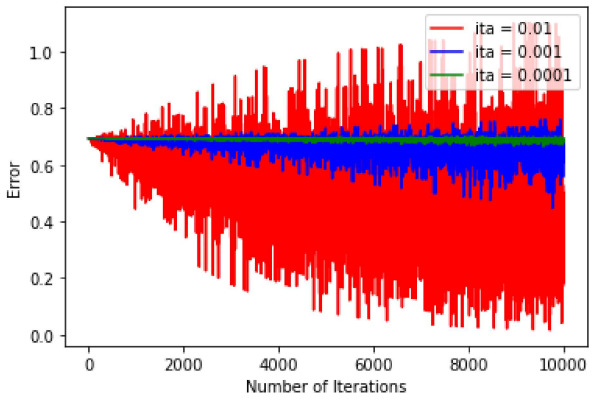


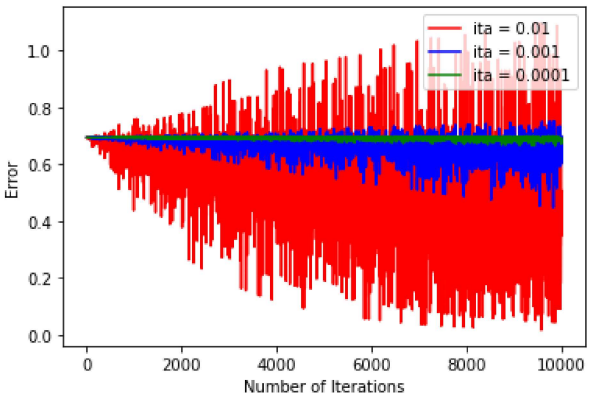
Split - 3 & 4:



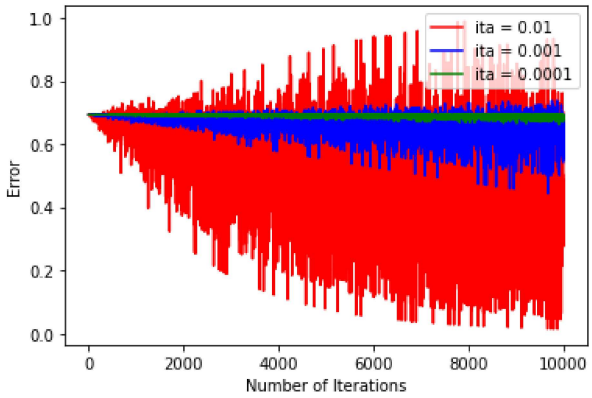


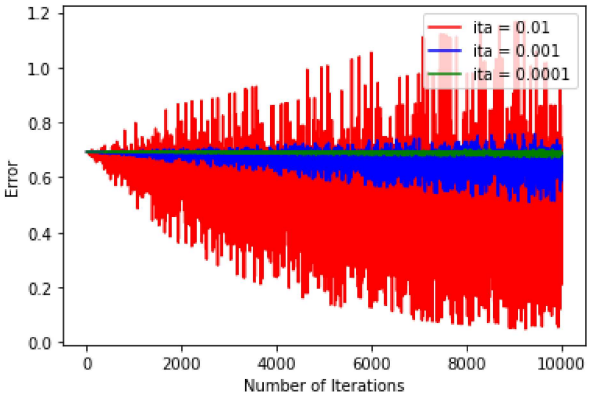
Split - 5 & 6:



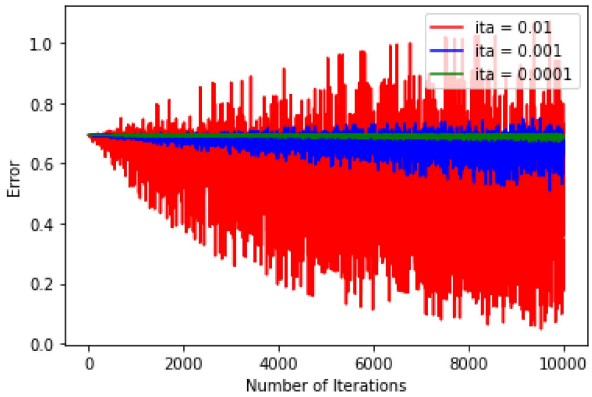


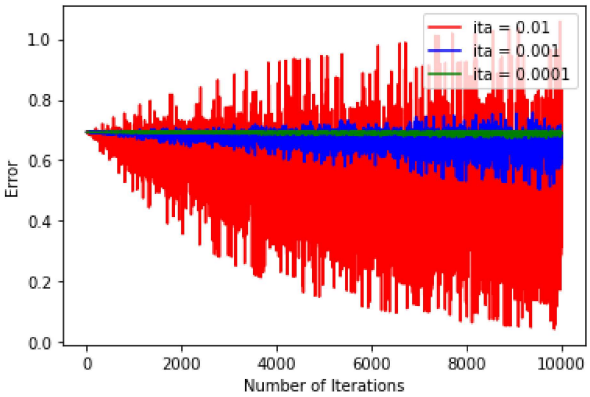
Split - 7 & 8:





Split - 9 & 10:





Results:

```
Split:
5

---ACCURACY---
Threshold        0.3        0.4        0.5        0.6        0.7
ita
0.0001     62.765957  62.765957  94.148936  37.234043  37.234043
0.0010     62.765957  63.297872  94.148936  37.234043  37.234043
0.0100     80.851064  91.489362  95.212766  81.914894  51.595745

---RECALL---
Threshold    0.3    0.4        0.5        0.6        0.7
ita
0.0001     100.0  100.0  94.915254   0.000000   0.000000
0.0010     100.0  100.0  94.915254   0.000000   0.000000
0.0100     100.0  100.0  96.610169  72.033898  22.881356

---PRECISION---
Threshold        0.3        0.4        0.5        0.6    0.7
ita
0.0001     62.765957  62.765957  95.726496   0.000000    0.0
0.0010     62.765957  63.101604  95.726496   0.000000    0.0
0.0100     76.623377  88.059701  95.798319  98.837209  100.0
```

From the above results, we can see that we obtain maximum accuracy of 95.21% in split 5.

Part D - Comparative Analysis

Learning Task 1: Perform a comparative study of models PM1, PM3, PM4, FLDM1, FLDM2, LR1 and LR2. The average performance metrics of 10 random training and testing splits should be considered for this comparative study. Find out the best performing model and if possible explain the reasons for that model to outcast other models.

In terms of **accuracy**, **Logistic Regression** model on **normalized** data i.e**. LR2** performed the best**.**

But this prediction is bound to change as the test-train data splits changes**.**

Possible Reasons for this maybe:

1) Logistic regression can handle non-linear decision boundaries.

Fisher's Linear Discriminant Analysis and the Perceptron Algorithm assumes linear decision boundaries between classes, whereas logistic regression can handle non-linear decision boundaries as well.

2) Logistic regression produces probabilistic outputs.

Unlike the perceptron algorithm, which produces binary outputs, logistic regression produces probabilistic outputs**.** This is useful because it allows you to understand the confidence level of the model's predictions**.**

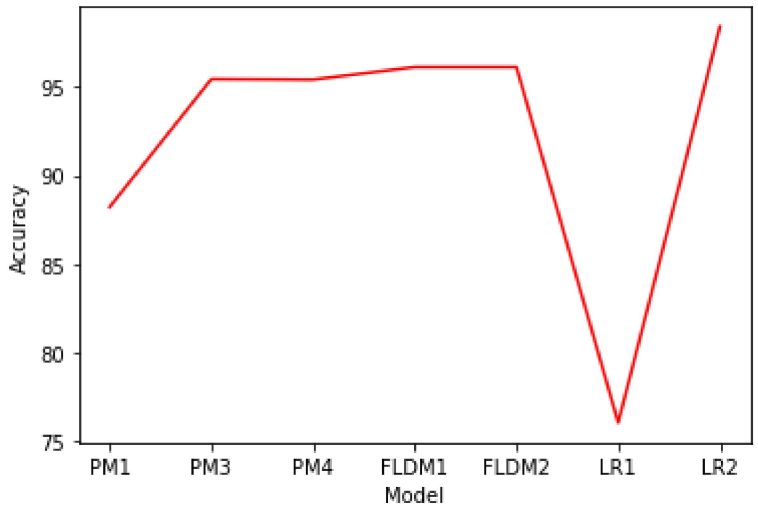
3) Logistic regression can handle imbalanced datasets.

Logistic regression can handle datasets where the number of examples in each class is imbalanced, whereas the Perceptron Algorithm and Fisher's Linear Discriminant Analysis may struggle with imbalanced datasets**.**

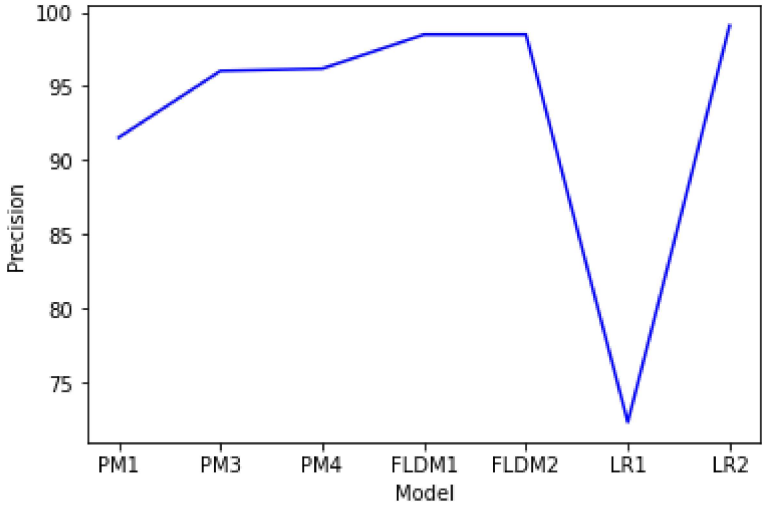
4) Logistic regression can be regularized.

Logistic regression can be regularized to prevent over fitting, whereas Fisher's Linear Discriminant Analysis and the Perceptron Algorithm do not have a natural way to handle overfitting**.**

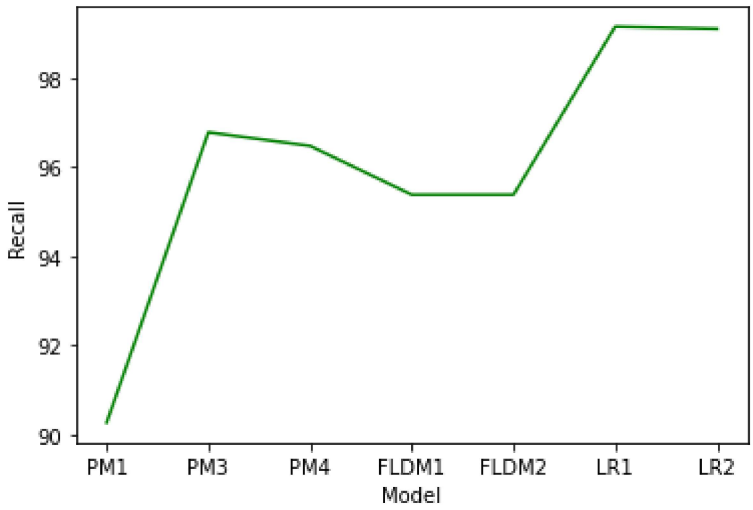
Comparative Study based on the Calculated Performance Metrics:



Accuracy



Precision



Recall