BITS PILANI HYDERABAD CAMPUS

Submitted in the partial fulfilment of Assignment 1 as part of the course BITS F464 – Machine Learning

S.No	Name	ID	Contributed: Yes / No
1	SHANTAM SRIVASTAVA	2020A4PS2019H	YES
2	NEEL MULAY	2020A4PS2269H	YES
3	DHRUV RAGHAVAN	2020AAPS2202H	YES

Part A – Perceptron Learning Algorithm

PM1 vs PM2

The Performance Metrics of models PM1 and PM2 are,

PM1 accuracy mean = 0.9026881720430108
PM1 accuracy variance = 0.00030899525956757946
PM1 recall = 0.8526182667517519
PM1 precision = 0.8861760014042703

PM2 accuracy mean = 0.9010752688172043
PM2 accuracy variance = 0.0006601919297028546
PM2 recall = 0.7970897866302525
PM2 precision = 0.9293819866659968

From this we observe that the model PM2 having shuffled training examples has similar accuracy as compared to PM1. A significant difference in both models is the recall parameter, which is higher for PM1, which suggests that it has fewer negatives than PM2, meaning, PM1 is better than PM2 at identifying positive (Malignant Tumor) instances. On the other than, since precision is higher for PM2 we can conclude that it has lower false positives as compared to PM2.

PM1 vs PM3

The Performance Metrics of models PM1 and PM3 are,

PM1 accuracy mean = 0.9026881720430108
PM1 accuracy variance = 0.00030899525956757946
PM1 recall = 0.8526182667517519
PM1 precision = 0.8861760014042703

PM3 accuracy mean = 0.9478494623655914
PM3 accuracy variance = 8.122326280494907e-05
PM3 recall = 0.9301338930657932
PM3 precision = 0.930222888039163

From the above data we can observe and conclude that the model PM3 having normalized training and testing datapoints, is a better model than PM1 in all performance parameters. This is because while normalizing, we make the mean 0 and variance 1 of each feature, this makes sure that bigger numbers don't solely dictate the model performance. Resulting in a model which has a much higher performance.

PM1 vs PM4

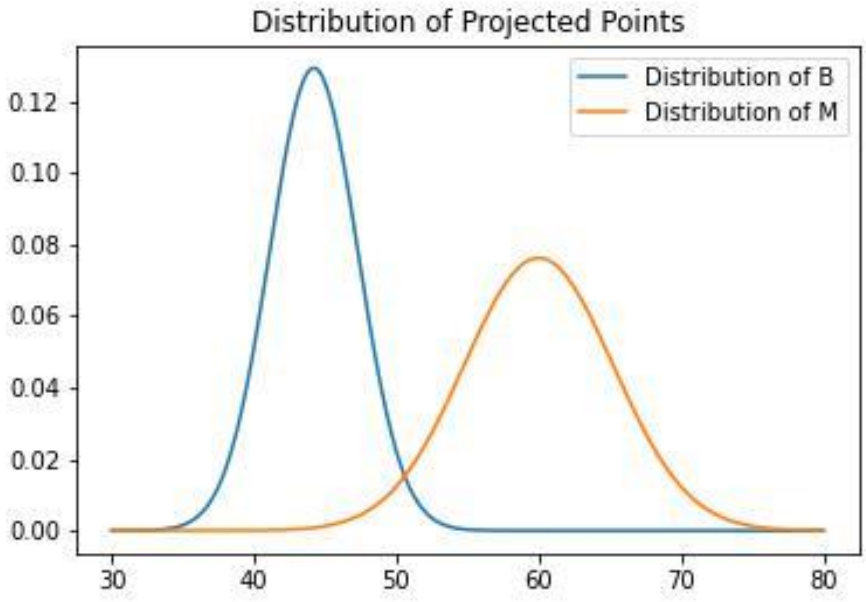
The Performance Metrics of models PM1 and PM4 are,

PM1 accuracy mean = 0.9026881720430108
PM1 accuracy variance = 0.00030899525956757946
PM1 recall = 0.8526182667517519
PM1 precision = 0.8861760014042703

PM4 accuracy mean = 0.9026881720430108
PM4 accuracy variance = 0.00030899525956757946
PM4 recall = 0.8526182667517519
PM4 precision = 0.8861760014042703

From this we observe that model PM4, having shuffled features but the same training examples as PM1 has the same model performance parameters as PM1. This is expected since we are not changing the training examples, just shuffling the order of the features.

Part B - Fischer's Linear Discriminant Analysis



Example Distribution of Projected Points

FLDM1 vs FLDM2

In FLDM2, we shuffle the features from the same training set used for FLDM1. Below is the comparison between the various performance metrics of both the models,

FLDM1 accuracy mean = 0.9623655913978496
FLDM1 accuracy variance = 0.00010983928777893378
FLDM1 recall = 0.9353039834778494
FLDM1 precision = 0.9676008987992948

FLDM2 accuracy mean = 0.9623655913978496
FLDM2 accuracy variance = 0.00010983928777893378
FLDM2 recall = 0.9353039834778494
FLDM2 precision = 0.9676008987992948

Here, we can observe that both models exhibit the exact same performance metrics. So, we further check the omega vector in both cases on which the data is projected on.

	Omega
0	0.080922
1	241.817
2	5.715697
3	40.11733
4	1.068362
5	0.097564
6	0.01305
7	-0.16055
8	0.030496
9	6.476632
10	18.39631
11	-0.01851
12	9.143815
13	37.4286
14	40.4277
15	-38.1872
16	0.140073
17	1.932626
18	-0.01668
19	78.01246
20	-1.65333
21	3.422285
22	-0.07798
23	5.551381
24	0.530024
25	10.95403
26	3.194941
27	-97.4607
28	-81.2737
29	17.52985

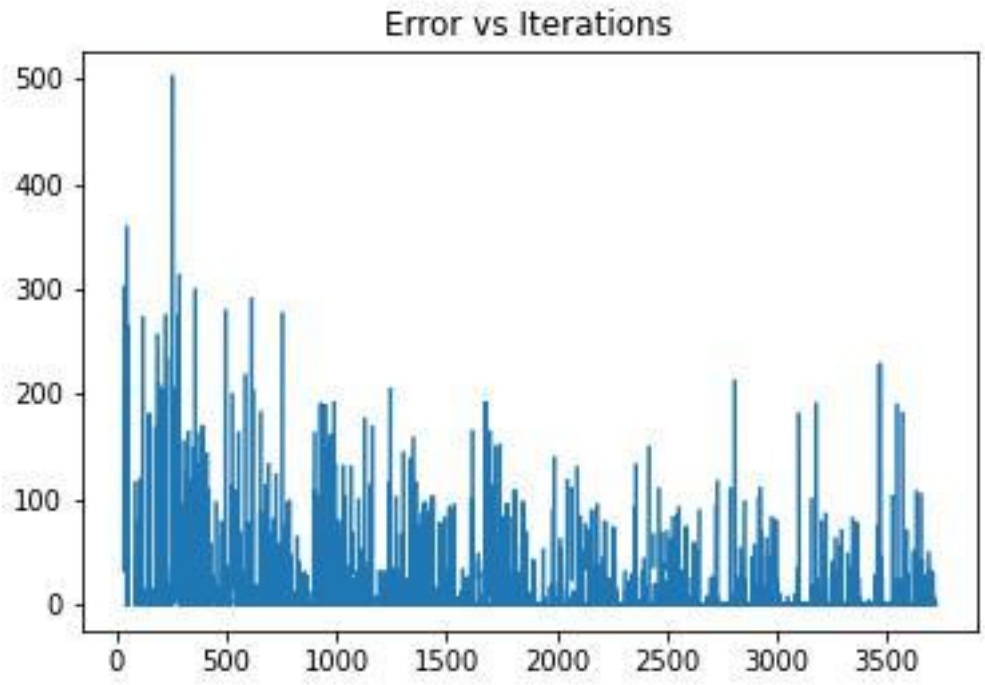
	Omega
0	-1.65333
1	0.097564
2	0.030496
3	0.01305
4	1.932626
5	-81.2737
6	37.4286
7	17.52985
8	10.95403
9	0.140073
10	5.551381
11	1.068362
12	-0.16055
13	-0.01851
14	-38.1872
15	18.39631
16	-97.4607
17	241.817
18	40.11733
19	0.530024
20	3.422285
21	0.080922
22	-0.07798
23	-0.01668
24	40.4277
25	3.194941
26	9.143815
27	5.715697
28	6.476632
29	78.01246

From these two omega vectors that we get from FLDM1 and FLDM2, we notice that the weights are identical but shuffled in the same manner as the features. This explains why both the models have the same performance metrics.

Part C – Logistic Regression

Learning Task 1 (LR1)

Stochastic Gradient Descent –



Learning Rate used for this graph is 0.0001, with a probability threshold of 0.5 (graphs for other learning rates and probability threshold are available in the Python file)

	Accuracy of SGD for LR1	Mean	Variance
0	0.01	0.791935	0.025853
1	0.001	0.805376	0.017741
2	0.0001	0.875806	0.003015

Here we notice an increase in mean accuracy with a decrease in learning rate. Also, the variance reduces for a lower learning rate. These observations occur because for a higher learning rate, the error values oscillate around the minima whereas as we reduce the learning rate the error reaches closer to the minima. The probability threshold used for these observations is 0.5.

Batch Gradient Descent –

Here we didn't get an error function because of 'nan' and 'infy' values, which happened because the data was not normalized in LR1.

	Accuracy of BGD for LR1	Mean	Variance
0	0.01	0.9	0.001747
1	0.001	0.897849	0.000827
2	0.0001	0.833871	0.025462

Here we notice a reducing mean accuracy with reducing learning rate. This is because for a lower number of iterations, the lower learning rate model could not reach the minima of the error function. We cannot increase the number of iterations here because in batch gradient descent we must calculate error value for each datapoint for a single iteration, for a high number of iterations it would require a lot of computation power and time.

Mini-Batch Gradient Descent –

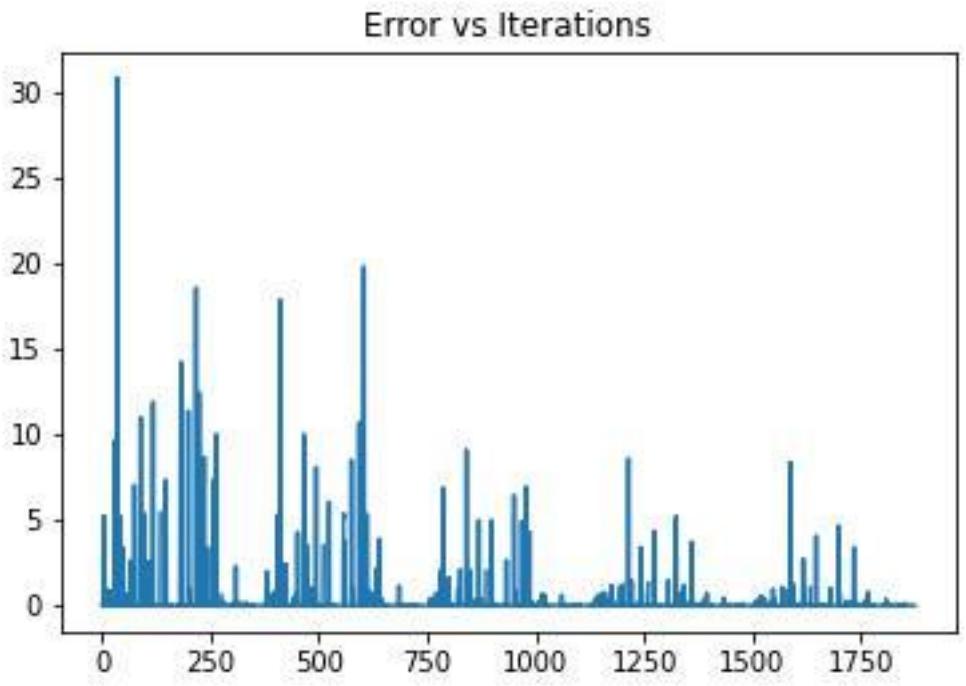
Batch Size used in our model is 64. Here we didn't get an error function because of 'nan' and 'infy' values, which happened because the data was not normalized in LR1.

	Accuracy of MBGD for LR1	Mean	Variance
0	0.01	0.841398	0.003117
1	0.001	0.869892	0.001907
2	0.0001	0.881183	0.001806

Here also, we notice an increase in mean accuracy with a decrease in learning rate. The variance also reduced for a lower learning rate. For a higher learning rate, the error doesn't converge to a minima for a higher number of iterations, instead it oscillates far from the minima hence the reduction in accuracy.

Learning Task 2 (LR2)

Stochastic Gradient Descent –



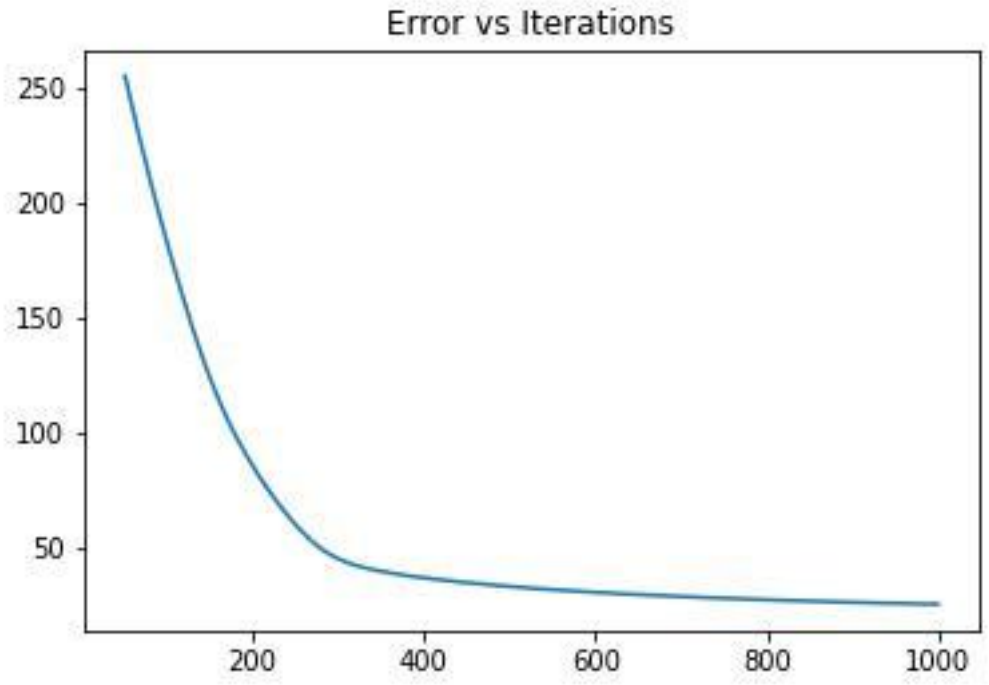
Learning Rate used for this graph is 0.01, with a probability threshold of 0.5 (graphs for other learning rates and probability threshold are available in the Python file)

	Accuracy of SGD for LR2	Mean	Variance
0	0.01	0.966667	9.13E-05
1	0.001	0.921505	0.000151
2	0.0001	0.895699	0.000192

Here, for the normalized training and test data, we can observe that the error function has a decreasing trend, and the last three sections are repeating. This suggests that the minimum error was found in lesser number of iterations than what is depicted in the graph.

Here, for a reduced learning rate we have also noticed a reduced accuracy. This is because the number of iterations for a lower learning rate must be high to reach the error minima.

Batch Gradient Descent –

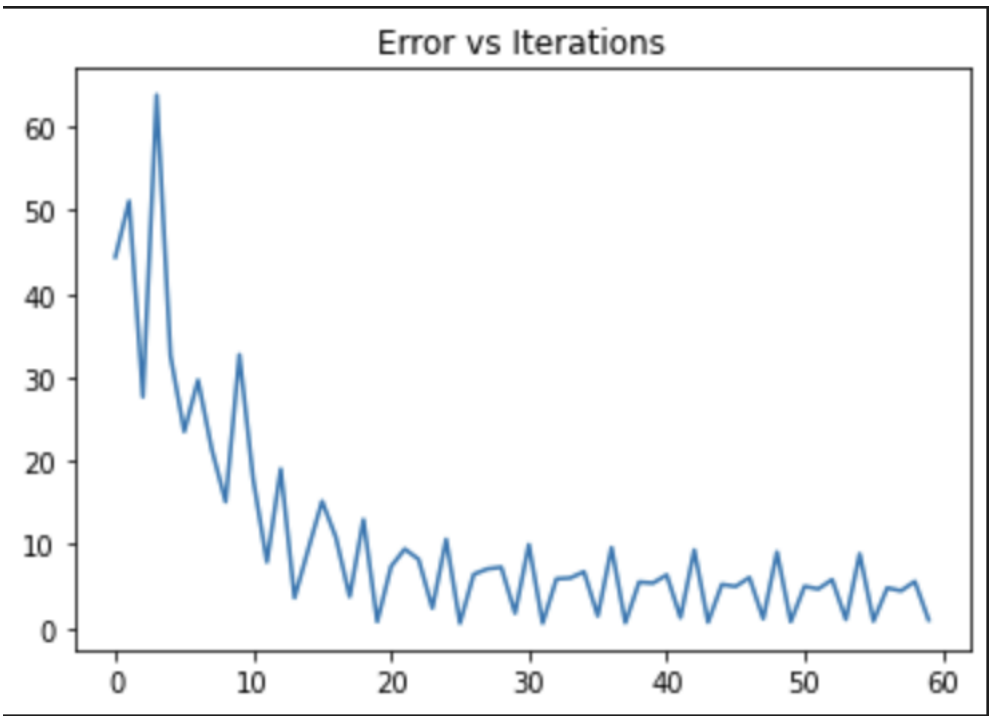


Learning Rate used for this graph is 0.0001, with a probability threshold of 0.5 (graphs for other learning rates and probability threshold are available in the Python file)

	Accuracy of BGD for LR2	Mean	Variance
0	0.01	0.963298	0.000121
1	0.001	0.956915	0.000614
2	0.0001	0.96383	0.000384

Here we notice similar accuracy for different learning rates, this suggests that the specific choice of learning rate is not a limiting factor for good performance of the model. From the graph we can see that the error function decreases significantly at the start and gradually approaches a minima. This is characteristic of the Batch Gradient Descent. Although computations times are significantly higher than stochastic and mini-batch since we are computing the errors of all data points in each iteration.

Mini-Batch Gradient Descent –



Learning Rate used for this graph is 0.01, with a probability threshold of 0.5 (graphs for other learning rates and probability threshold are available in the Python file)

	Accuracy of MBGD for LR2	Mean	Variance
0	0.01	0.964894	8.6E-05
1	0.001	0.968617	0.000246
2	0.0001	0.914894	0.00017

Here, we can see in the graph that this looks like the combination of stochastic and batch gradient descent. We also notice high accuracy of 0.01 and 0.001 learning rates, whereas if the learning rate decreases to 0.0001, the accuracy also reduces. This is because lower learning rate requires higher number of iterations to reach the minima. This can be computationally heavy and requires a higher amount of time.

Changing the Probability Threshold –

Changing the probability threshold in logistic regression causes to change the decision boundary.

Probability Threshold	Accuracy
0.3	0.968085106
0.7	0.978723404
0.5	0.968085106

We notice that the accuracy for different probability thresholds is similar. This suggests that the model is overfitting since there is no bound on the norm of omega vector. This observation is consistent with the fact that one of the cons of logistic regression is that there is overfitting if the data has a lot of dimensions.

Part D – Comparative Study

PM1 -

PM1 accuracy mean = 0.9026881720430108
PM1 accuracy variance = 0.00030899525956757946
PM1 recall = 0.8526182667517519
PM1 precision = 0.8861760014042703

PM3 -

PM3 accuracy mean = 0.9478494623655914
PM3 accuracy variance = 8.122326280494907e-05
PM3 recall = 0.9301338930657932
PM3 precision = 0.930222888039163

PM4 -

PM4 accuracy mean = 0.9026881720430108
PM4 accuracy variance = 0.00030899525956757946
PM4 recall = 0.8526182667517519
PM4 precision = 0.8861760014042703

FLDM1 –

FLDM1 accuracy mean = 0.967741935483871
FLDM1 accuracy variance = 0.00010405827263267411
FLDM1 recall = 0.9446242271816724
FLDM1 precision = 0.9700988512701694

FLDM2 –

FLDM2 accuracy mean = 0.967741935483871
FLDM2 accuracy variance = 0.00010405827263267411
FLDM2 recall = 0.9446242271816724
FLDM2 precision = 0.9700988512701694

LR1 -

Accuracy	1	2	3	4	5	6	7	8	9	10	Mean	Variance
0.01	0.9032	0.8710	0.8978	0.9355	0.8978	0.7903	0.9247	0.9462	0.9194	0.9140	0.9000	0.0017
0.001	0.9032	0.9086	0.9086	0.8602	0.8925	0.8978	0.8387	0.9462	0.9247	0.8978	0.8978	0.0008
0.0001	0.9086	0.8065	0.9194	0.8925	0.8871	0.8441	0.3656	0.9140	0.9032	0.8978	0.8339	0.0255

LR2 –

Accuracy	1	2	3	4	5	6	7	8	9	10	Mean	Variance
0.01	0.9681	0.9734	0.9574	0.9628	0.9628	0.9840	0.9628	0.9681	0.9628	0.9468	0.9649	0.0001
0.001	0.9628	0.9734	1.0000	0.9521	0.9681	0.9574	0.9521	0.9521	0.9840	0.9840	0.9686	0.0002
0.0001	0.9362	0.9096	0.9149	0.9149	0.8989	0.8989	0.9362	0.9096	0.9255	0.9043	0.9149	0.0002

From the Data shown above, after comparing all the models we can conclude that, **LR2,** which is **Logistic Regression with normalized training and testing data** is the best performing model with the highest mean accuracy and even gave 100% accuracy in one of the iterations. The reason this model is better than the other models is that:

- Normalization – That makes sure that certain big numbers in the training examples do not sway the data in one direction.
- Sigmoid Function – Sigmoid function being nonlinear can handle more complexity and can capture more nuances as compared to Perceptron and Fischer.
- Probability Threshold – Having a variable probability threshold for choosing the decision boundary gives more flexibility.
- Logistic Regression can handle outliers in a more optimal way because it has a probabilistic framework.