Machine Learning Assignment 1: Project Report

Done by

1. Arunachala Amuda Murugan (2021A7PS0205H)
2. Maitreya Manohar (2021A7PS2663H)
3. Ayush Bhauwala (2021A7PS0180H)

Part 0: Data Preprocessing

1. Dropping NaN values
2. Imputing missing values (using mean and mode of the columns depending on the data type)
3. Normalizing/standardizing the data based on the train set's mean and variance. The test set is also standadized using the train sets's mean and variance, to avoid any test set data leak.
4. Creation of N number of train test splits as required

Part 1: Perceptron

1. PM1 : Perceptron Model that classifies the dataset shuffling and without normalisation.
2. PM2 : Perceptron Model that classifies the dataset with shuffled training data.
3. PM3 : Perceptron Model that classifies the dataset with normalised training data.
4. PM3 : Perceptron Model that classifies the dataset with shuffled features.

	threshold	delta	method	mean_ accuracy	std_accuracy	N epochs
1	100	0.001	PM3-Infinite	94.24	0.94	None
2	None	NaN	PM3-Epochs	94.24	0.94	1000
3	100	0.001	PM2-Infinite	92.22	0.61	None
4	None	NaN	PM1-Epochs	91.41	2.26	1000
5	None	NaN	PM4-Epochs	91.41	2.26	1000

Observations:

1. The accuracies for PM1 and PM2 are different as the perceptron algorithm is very sensitive to order of feature tuples, as the weight vector is updated every wrong prediction. Hence the models.decision boundaries learnt/trained are also different.
2. The perceptron model trained on the normalized data has the best accuracy (PM3). This is because it now treats every feature with equal weightage. Hence it can learn the best relations. Additionally we see that the accuracy on infinite epochs till convergence and on 1000 epochs are the same. This is because the decision boundary learnt perfectly classified all the training points, so it didnt have to update anymore.
3. The accuracies for PM1 and PM4 are the same as shuffling the order of the features don’t have an effect on the decision boundary, as all we are doing is swapping dimension order. So while the decision boundary is moved in the exact same order, the weights for every dimension remain the same, and hence classify each point the same way

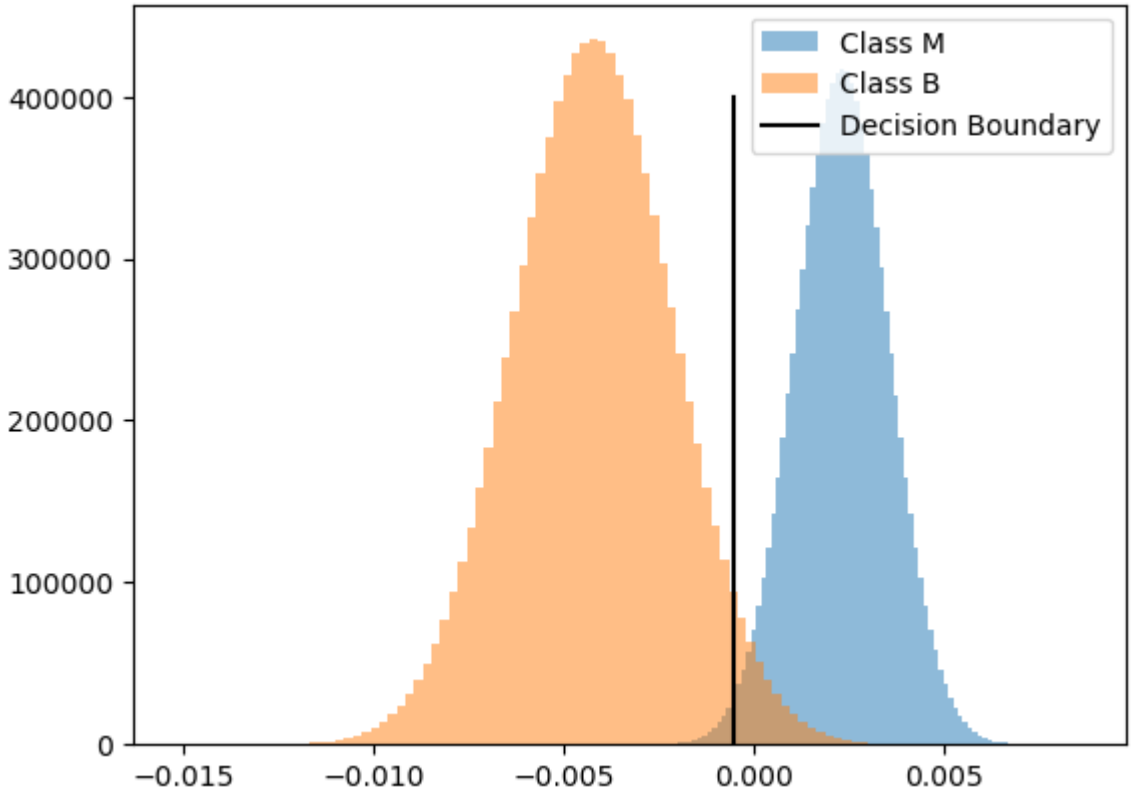
Part 2: Fisher's Linear Discriminant

FLDM1

This model was trained on unnormalized data. This model transforms the data in 32 dimensions to one dimension.
We achieved an accuracy of 94.9%.

To find the decision boundary we equated the gaussian distribution of the first class multiplied by the prior probability of the first class to the gaussian distribution of the second class multiplied by its prior probability. This is a quadratic equation and hence we get 2 roots. To find the root suitable to serve as the decision boundary, we found the sum of the distances between the root and the means of each class. The root which gives a smaller value for the sum is the one that serves as the decision boundary.
We verified that the decision boundary we arrived at is correct as we got the same classification of testing points using generative approach.

The following graph shows the decision boundary we arrived at.



FLDM2

The same training data that was used to train FLDM1 was used to train FLDM2. But only this the order of the features was shuffled.

We found that the accuracy of the model did not change, and neither did the decision boundary. The only thing that changes is the direction of the line we project the data points onto.

This is because FLDA does not depend on the order of the features of the data points.

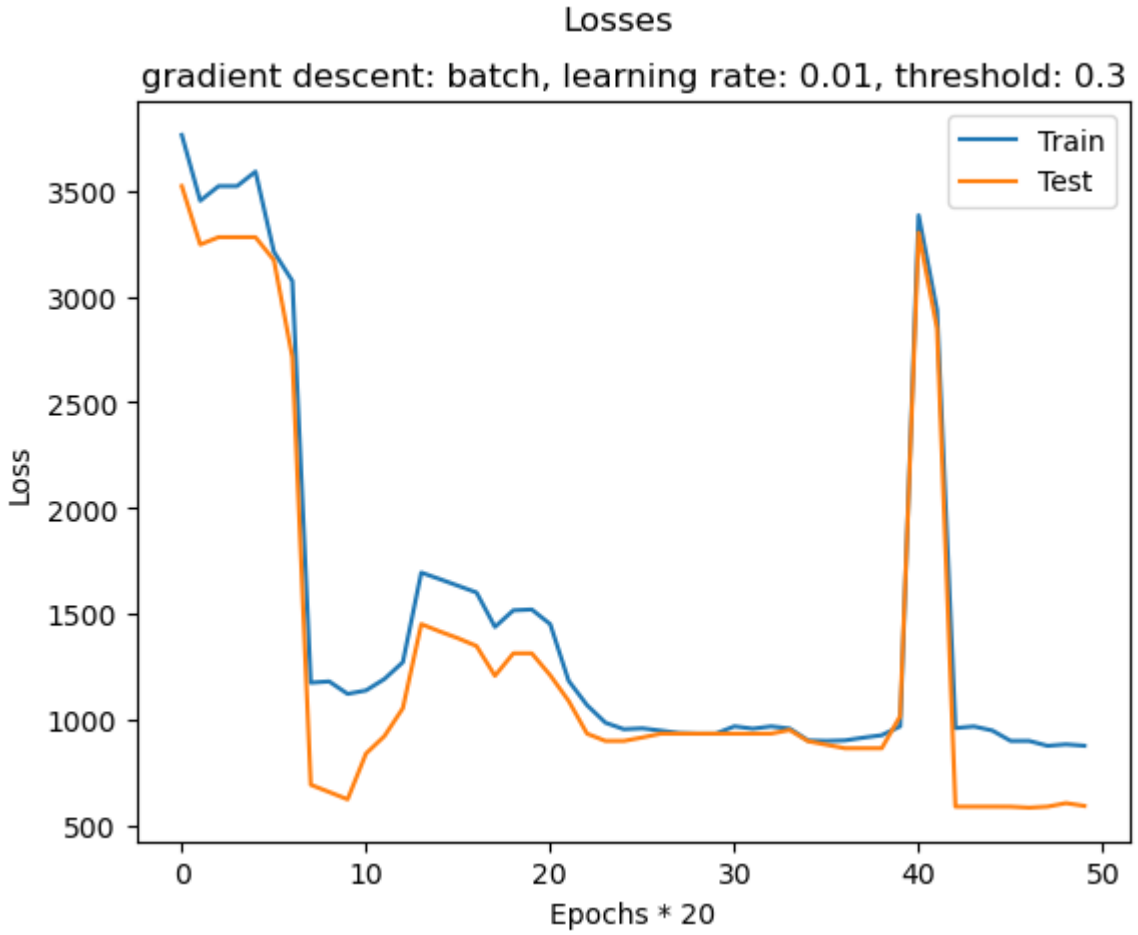
Part 3: Logistic Regression

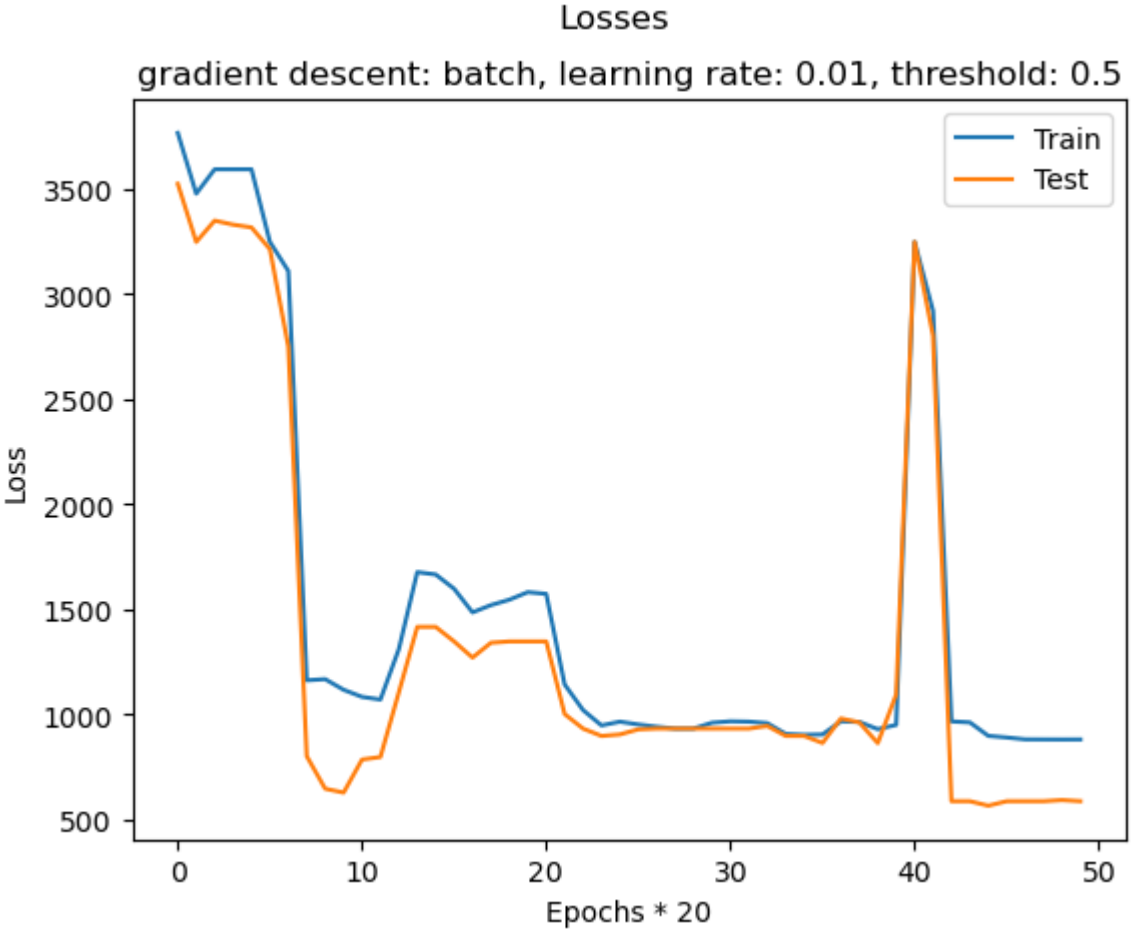
Trained logistic regression models using the following hyperparameters:

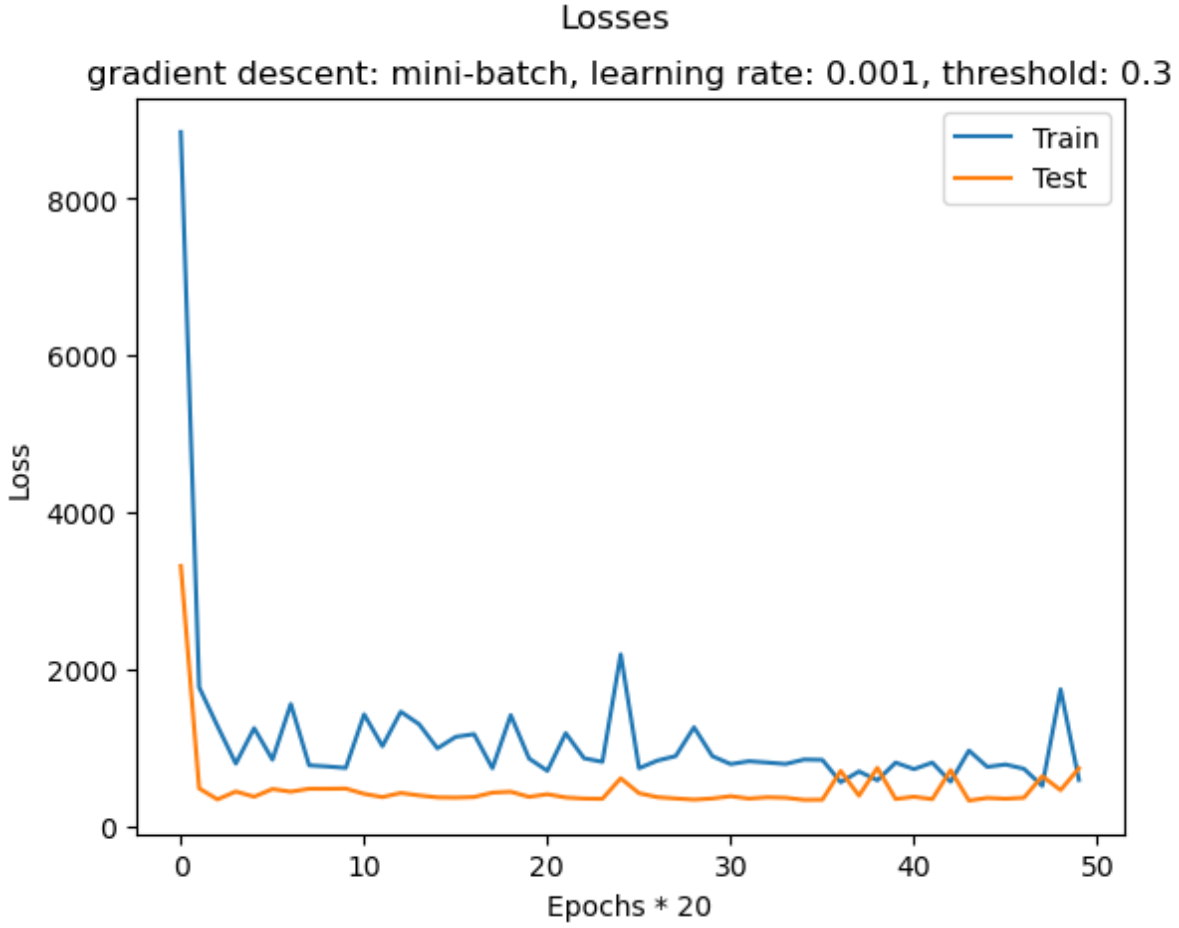
1. Thresholds: **0.3, 0.4, 0.5, 0.6, 0.7**
2. Gradient descents: **batch, mini-batch, stochastic**
3. Learning rates: **0.01, 0.001, 0.0001**

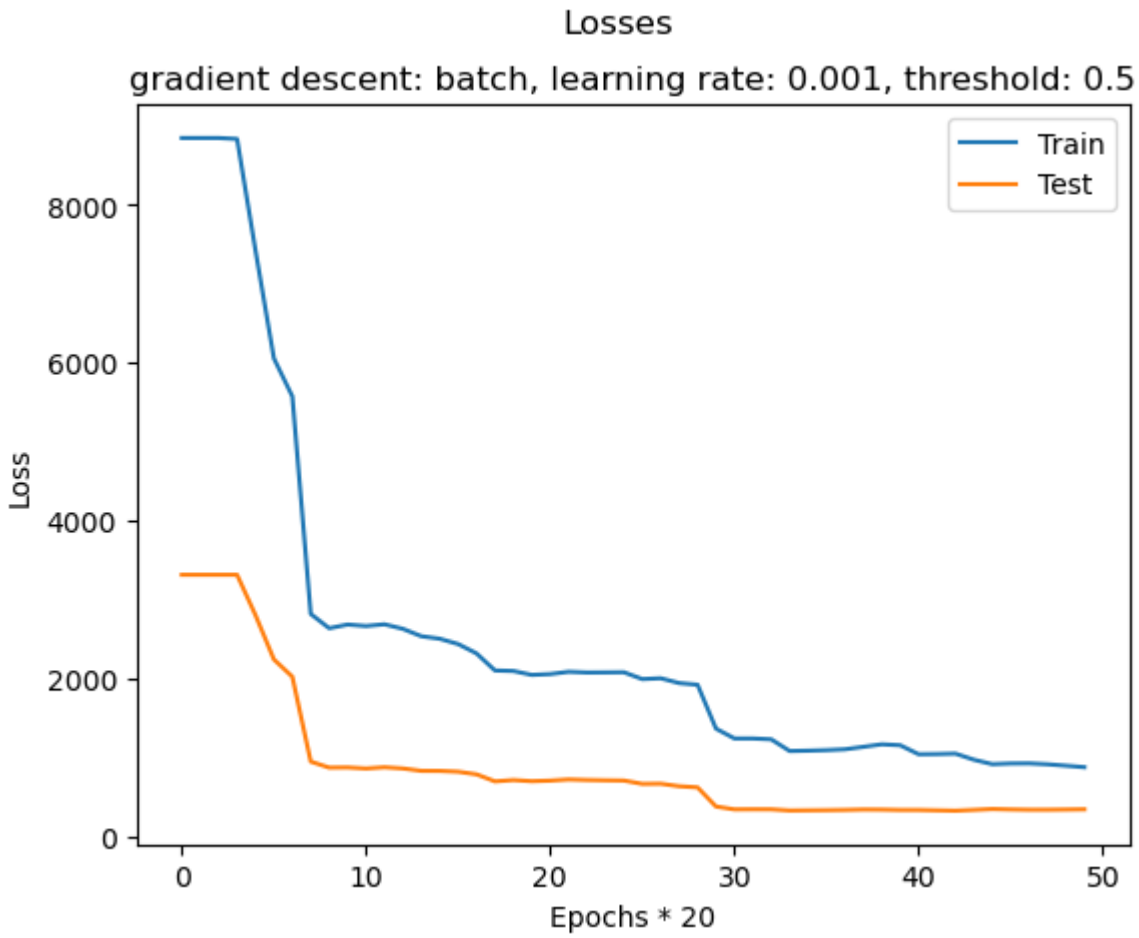
The models were first trained on unnormalized data, and then on normalised data. The observations for each of them have been documented below.

3.1: On Unnormalized Data




Losses
gradient descent: batch, learning rate: 0.01, threshold: 0.5
Train
Test
Loss
3500
3000
2500
2000
1500
1000
500
0
10
20
30
40
50
Epochs * 20


Losses
gradient descent: mini-batch, learning rate: 0.001, threshold: 0.3
Train
Test
Loss
8000
6000
4000
2000
0
0
10
20
30
40
50
Epochs * 20



Best combinations noted:

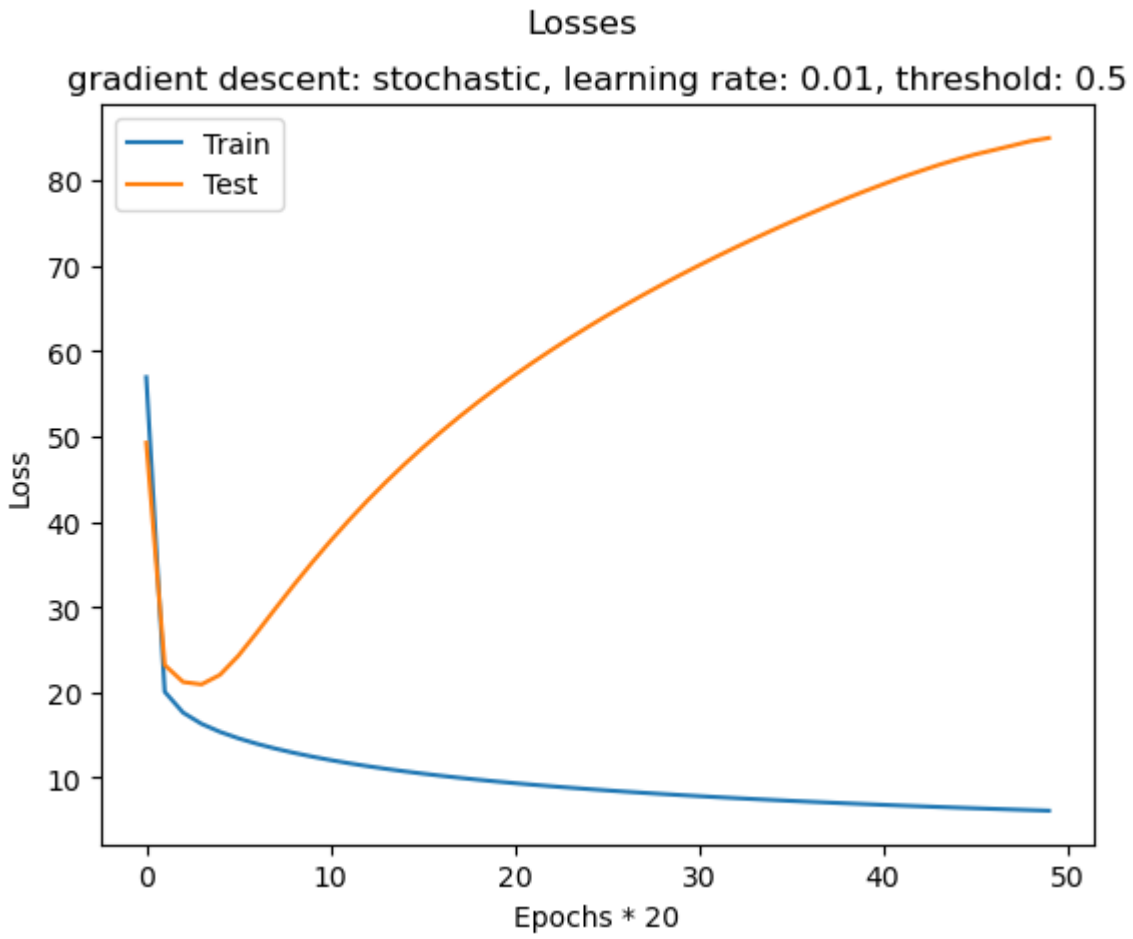
	threshold	learning_ rate	descent_type	accuracy
1	0.4	0.0010	stochastic	94.44
2	0.7	0.0001	stochastic	93.94
3	0.6	0.0010	stochastic	93.94
4	0.3	0.0100	stochastic	93.94
5	0.6	0.0100	stochastic	93.94

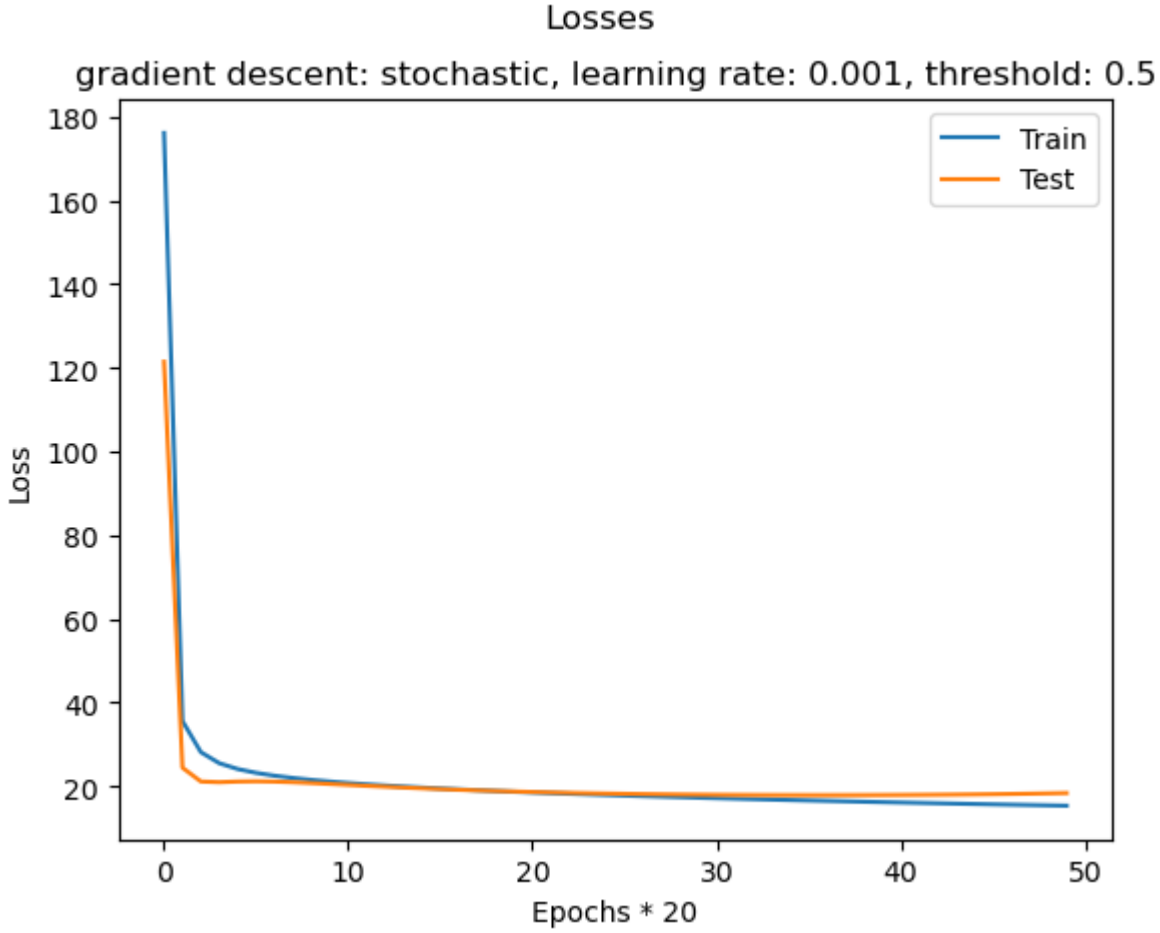
Observations:

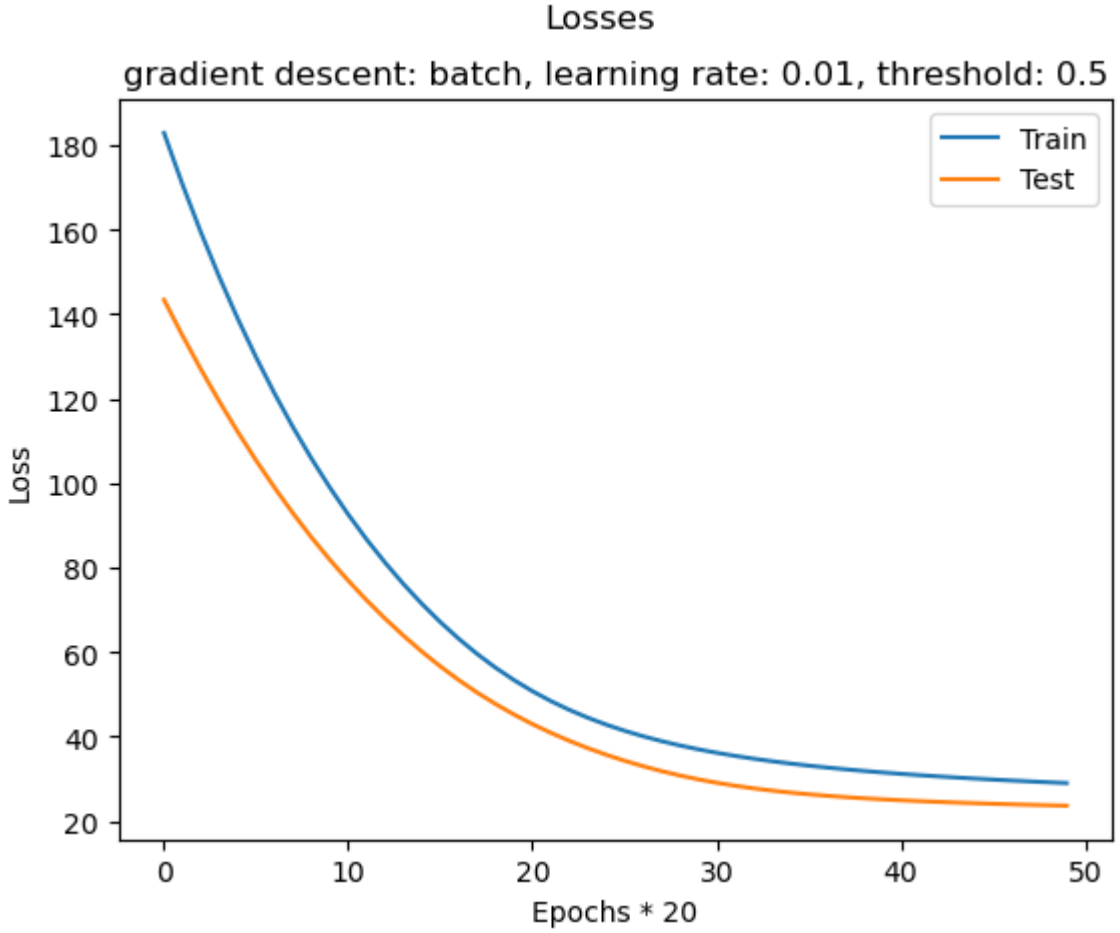
1. Stochastic descent has a noisier descent but reaches the area near the minima of the loss much faster as the updation of weights is much more common. As the weight is updated N times every epoch (i.e., for every row)
2. Batch gradient descent has a much smoother descent, but takes a longer time to reach the minima area

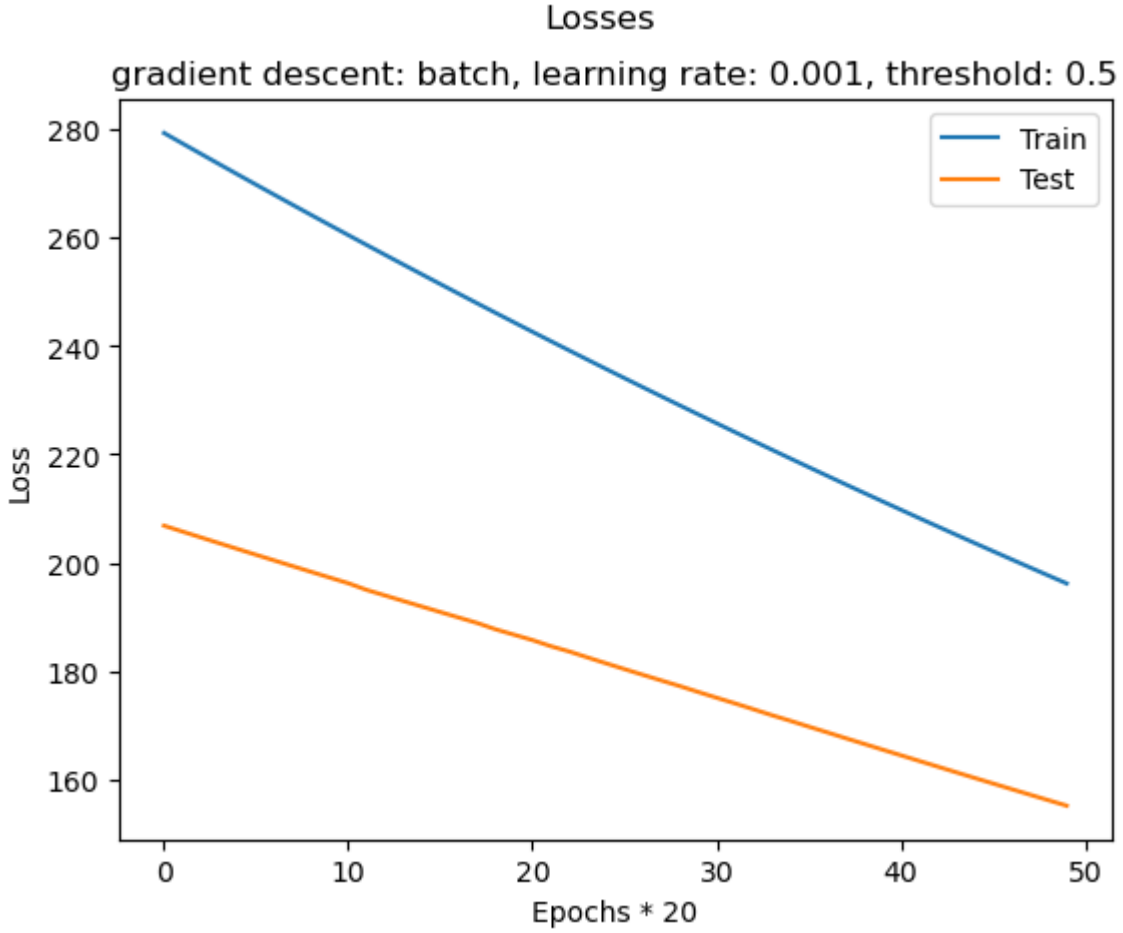
3. Mini batch is a sweet spot between the two.
4. Changing the learning rate affects the speed we reach the minima, as well as affects the level of noise in the loss curve. The noise increases with a higher learning rate, as on reaching the minima, we will have more number of epochs where we oscillate about the minima.

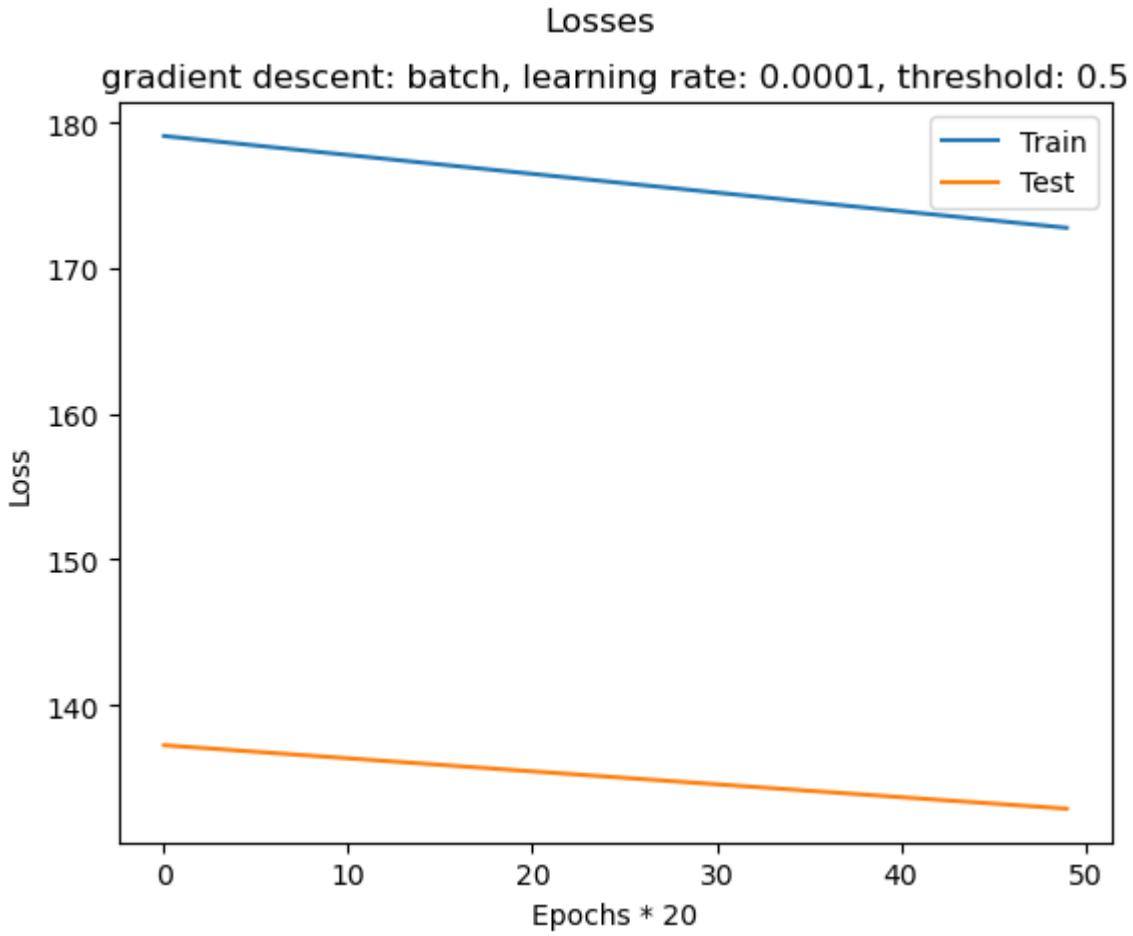
3.2: On Normalised Data




Losses
gradient descent: stochastic, learning rate: 0.001, threshold: 0.5
Train
Test
Loss
Epochs * 20
180
160
140
120
100
80
60
40
20
0
10
20
30
40
50


Losses
gradient descent: batch, learning rate: 0.01, threshold: 0.5
Train
Test
Loss
Epochs * 20
180
160
140
120
100
80
60
40
20
0
10
20
30
40
50


Losses
gradient descent: batch, learning rate: 0.001, threshold: 0.5
Train
Test
Loss
280
260
240
220
200
180
160
0
10
20
30
40
50
Epochs * 20



Best combinations noted:

	threshold	learning_rate	descent_type	accuracy
1	0.3	0.0100	mini-batch	97.98
2	0.4	0.0100	mini-batch	97.98
3	0.5	0.0001	stochastic	96.97
4	0.5	0.0100	mini-batch	96.97
5	0.4	0.0001	stochastic	96.97

Observations:

5. The curves in the normalised data are much smoother than the loss curves of the models trained on the unnormalized data. The losses are also much smaller. This shows that the models have learnt much better
6. This is also seen in the improvement of accuracies in the top models trained on this data

7. Other than the observations noted in the unnormalized section, we see that in a few cases where the graphs can help us conclude some more points.
8. The stochastic descent achieves its minima extremely fast, and the loss increases after a point showing that we have slightly overfit our model. This is more noticeable when comparing the train and test curves together.
9. Also, for lower learning rates, batch gradient descent has an extremely slow rate of decreasing its loss.

The other graphs and accuracies for the different models can be seen in the relevant notebooks.