Paul Johnston

CS 478

Perceptron

1. This perceptron uses numpy matrices to store weight vectors and if there are more than 2 output classes possible, multiple vectors are stored, each acting as a perceptron, and all are trained for every input. The stopping criteria allows for two variables. The first being the threshold to denote "significant progress", which is measured by the accuracy rate of an epoch, while the other is a cap on number of epochs without significant improvement. For implementation details, see the code submission.

2. The two ARFFs created are "linearCustom.arff" and "nonlinearCustom.arff". They can be found with the code submission as well. Below are the text contents of the files

   a.

@relation linearCustom @attribute a1 real @attribute a2 real @attribute class {0,1} @data -0.3, 0.2, 1 -0.2, 0.1, 1 0.1, 0.25, 1 -0.1, 0.25, 1 0.2, 0.13, 0 0.3, 0.05, 0 0.15, -0.05, 0 -0.25, -0.3, 0	@relation nonlinearCustom @attribute a1 real @attribute a2 real @attribute class {0,1} @data -0.3, 0.2, 1 -0.2, 0.1, 1 0.1, 0.25, 1 -0.25, -0.3, 1 -0.1, 0.25, 0 0.2, 0.13, 0 0.3, 0.05, 0 0.15, -0.05, 0

3. Below is a table of results from varying learning rates, progress thresholds and epoch caps, an analysis of the data and discussion of stopping criteria follows.

Linearly Separable Custom ARFF				
Learning Rate	Progress Threshold	Epoch Cap	Total Epochs	Accuracy
0.1	0.01	5	8	100%
0.5	0.01	5	8	100%
1	0.01	5	9	100%
3	0.01	5	11	100 %
3	0.5	8	9	100%
3	1.5	8	8	100%

   a.

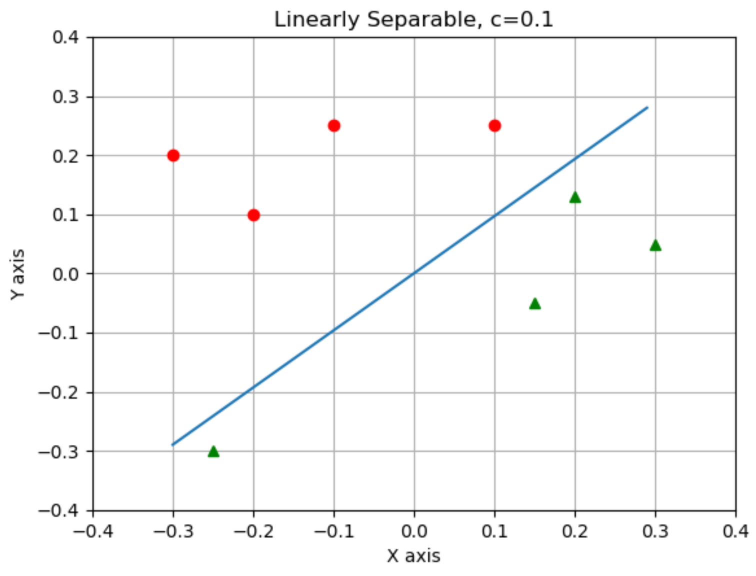
Non Linearly Separable Custom ARFF				
Learning Rate	Progress Threshold	Epoch Cap	Total Epochs	Accuracy
0.1	0.01	5	49	75%
0.5	0.01	5	273	75%
1	0.01	5	167	75%
3	0.01	5	222	75%
3	0.5	8	9	87.5%
3	1.5	8	8	75%

   b.

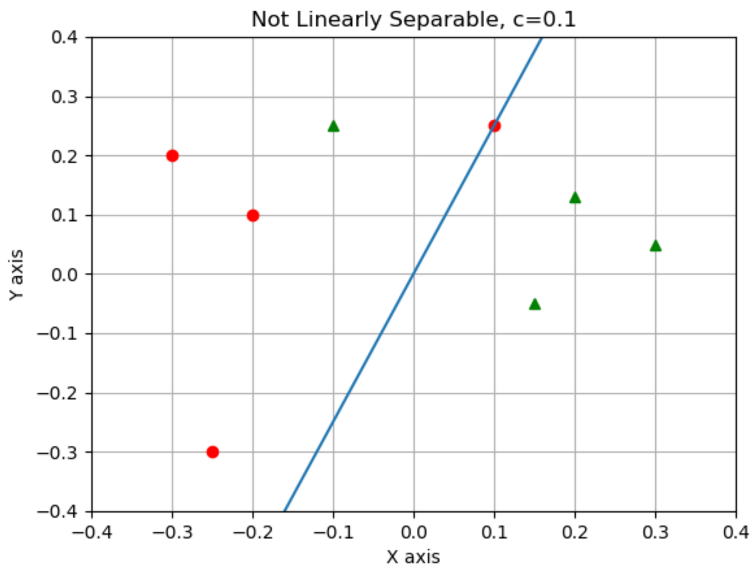
c. It seems that changing the learning rate had a greater effect on the non linearly separable dataset. I suspect that with a higher learning rate, the perceptron was more susceptible to noise and the progress threshold was too low to counter that. It becomes apparent when looking at the last row of the non linearly separable data where the learning rate is high as well as the progress threshold.

d. My specific stopping criteria is having a "progress threshold" which determines a range for "significant change" to have occurred. For instance, if from one epoch to another the accuracy changes by 0.05% and the progress threshold is 0.01, this epoch of testing would be deemed "significant" and the number of epochs that have passed without significant change would be zeroed out. However, if the values were reversed and the progress threshold was 0.05 and the accuracy changed by 0.01, the epoch would be deemed insignificant, and a counter of epochs that have passed with no significant change would be incremented. The second criteria of stopping is how many epochs can go by with no significant change before stopping. For example, if the epoch cap were 5, and 4 epochs had gone by with no significant change in accuracy, and a fifth epoch goes by with no significant change in accuracy, the perceptron would stop, and the weights at that epoch would be the final weights with which to test.

4. Below are the graphs for the custom linear dataset and non linear dataset respectively:



a.



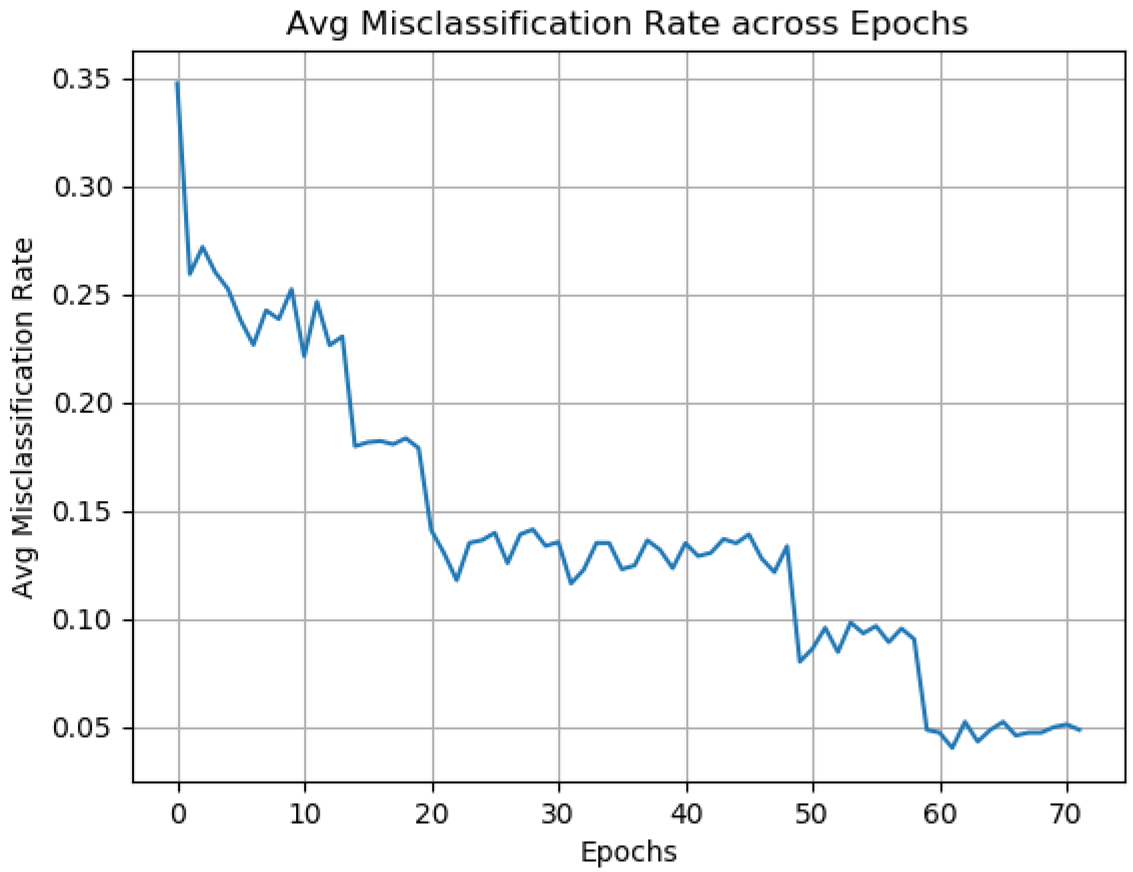
b.

5. Voting set

Test # (rand split 70/30)	Training accuracy	Testing accuracy	# epochs required
1	0.944	0.935	8
2	0.975	0.942	36
3	0.937	0.899	21
4	0.968	0.949	20
5	0.975	0.942	18
AVERAGE	0.959	0.933	20.6

a.

b. From looking at the weights, it seems that the 10th question of the vote is one of the biggest, if not the biggest, factor in determining the political party of the voter. The weight on this value is consistently higher than the others. A few other significant inputs are the 5th and 8th. The most insignificant value I noticed was the 12th which consistently was a number to the negative 17th power, as well as the 14th value.



c.

d. To plot this graph, I found the RMSE for each epoch and stored that in an array. After finding the arrays of RMSEs for the 5 trials, I padded the smaller ones with zeros to be equal to the size of the largest array. Next I added them up using numpy and divided by 5, therefore only taking those misclassification rates that made it to the longer epochs and so forth.

6. The experiment I chose was using the Iris dataset and multiple perceptrons to identify the irises. I trained these perceptrons simultaneously, keeping separate weight arrays for each "perceptron". I take an input, see if there are more than 2 output classes, if there are, I make the appropriate number of weight arrays, and then see what the given label was, if it is the same as the index of the weight array, I mark the target as 1, if they differ, the

target becomes 0. For example, if I was given input for an Iris-Setosa, and my matrix array was at index “0”, then the target would become 1, because I would want to correctly identify this flower for this matrix. For the other matrix arrays however, the target would be 0, because those matrices would be trying to identify Iris-versicolor and Iris-virginica, respectively. Below are some results

Test # (rand split 70/30)	Training accuracy	Testing accuracy	# epochs required
1	0.742	0.711	253
2	0.838	0.933	163
3	0.695	0.622	83
4	0.638	0.733	113
5	0.6	0.555	99
Average	0.702	0.710	142.2

a.
b. I found that my accuracy increased slightly after changing the progress threshold to 0.01. To see the full implementation with multiple matrices, see the code submission