1. Code Implementation

I implemented the Perceptron algorithm using the provided interface. My shuffle function merges the data into a single Numpy array, shuffles the rows in the array and then splits them again. I used a similar technique for randomly shuffling and splitting the training and test data. As was instructed I only used Numpy for the computational part of the lab (except for the part where I was instructed to use Scikit-learn).

After running my code on the debug data set, I got the expected weight vector of REDACTED > with an accuracy of 88%. The test data gave me a weight vector of REDACTED with and accuracy of REDACTED

2. Custom datasets

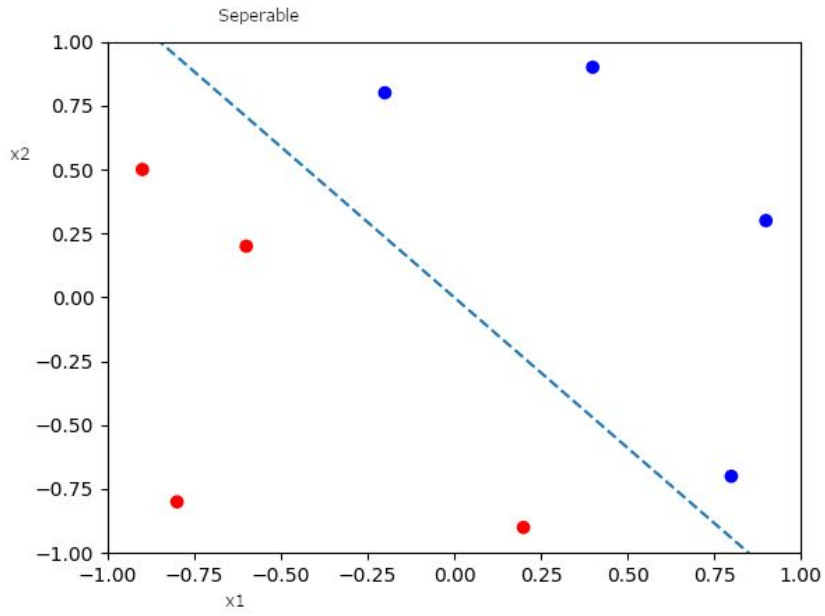
Linearly Separable

X1	X2	Out
.9	.3	1
REDACTED	REDACTED	1
		1
		1
		0
		0
-.6	.2	0
-.8	-.8	0

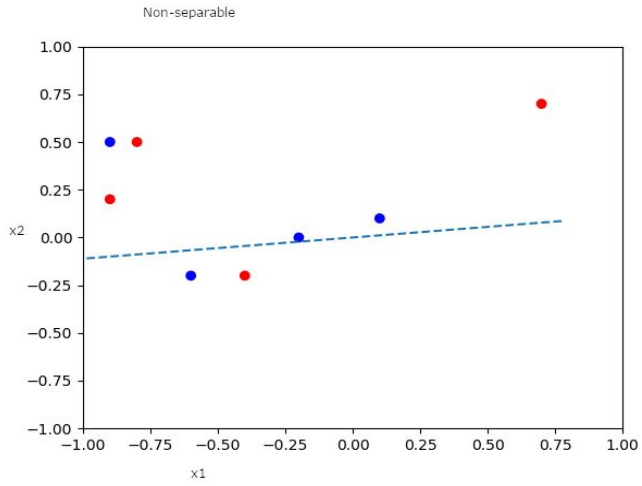
Non-Linearly Separable

X1	**X2**	**Out**
-.9	.5	1
-.2	0	1
REDACTED	REDACTED	1
		1
		0
		0
		0
.9	.9	0

3. Train on My Datasets



I originally set a minimum of 10 epochs, but I found that the linearly separable data could fit itself in 3-5. The perceptron had a rough time with the non-linearly separable set. It could, on occasion get about 68% correct, but most of the time it would end up getting 50%. After dropping the learning rate(to around .001) and upping the minimum epochs a little (like 20) I could get it to hit a score of > 50% slightly more regularly, but it didn't seem to make too big of a difference. The graph of the linearly separable data was trained with a learning rate of .1. The perceptron had a really easy time coming up with a boundary through these data.



The non-separable data gave the perceptron some trouble. It really struggled to do better than 50% on these data. Sometimes I would get lucky when shuffling the data

and it would get 62% correct, but these data don't really follow any sort of linear distribution, so using a linear model isn't going to give us any sort of predictive power on these data.

4. Voting Task

Number of Epochs	Test Accuracy	Training Accuracy
27	91%	98%
30	93%	98%
28	93%	98%
20	95%	96%
96	96%	97%
AVG=40.2	AVG=93.6%	AVG=97.4%

One thing that is immediately apparent is that the training accuracy does not change much even though the test accuracy can vary. In this test the test accuracy is always lower than the training accuracy. The number of epochs seems to loosely correlate with test accuracy.
The below table contains the weight vectors from my tests that I ran on the voting test set. Due to the limited horizontal space I included the labels below the table.

0			1.5	.					.								
			2.1														
			1.8														
			1.8														
			2.4					-1.2									

1. handicapped-infants: 2 (y,n)
2. water-project-cost-sharing: 2 (y,n)
3. adoption-of-the-budget-resolution: 2 (y,n)
4. physician-fee-freeze: 2 (y,n)
5. el-salvador-aid: 2 (y,n)
6. religious-groups-in-schools: 2 (y,n)
7. anti-satellite-test-ban: 2 (y,n)
8. aid-to-nicaraguan-contras: 2 (y,n)
9. mx-missile: 2 (y,n)
10. immigration: 2 (y,n)

TA NOTE: VERY BAD FORMATTING - author might have abbreviated issues for column headings(and included a legend below) or used vertical column headings

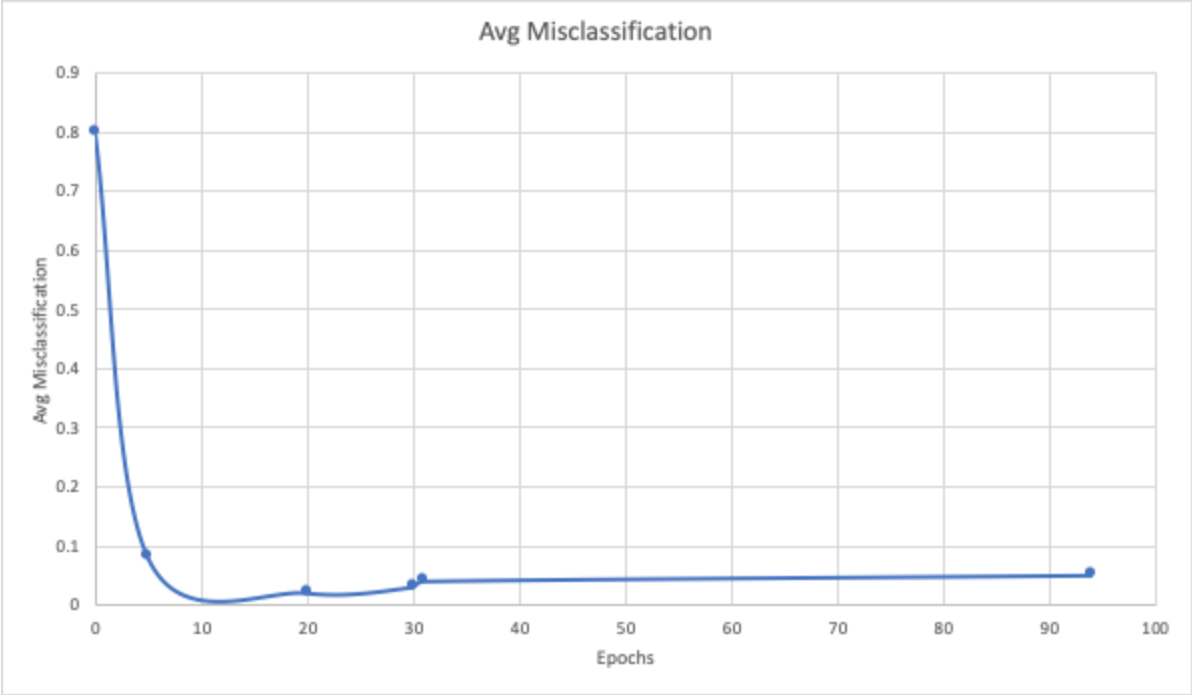
Definitely should not have a single number broken up on 2 lines

Also a column on the left that says Trial 1, Trial 2, etc.

All figures should be clean & labeled, i.e. something that could be dropped into a journal article

11. synfuels-corporation-cutback: 2 (y,n)
12. education-spending: 2 (y,n)
13. superfund-right-to-sue: 2 (y,n)
14. crime: 2 (y,n)
15. duty-free-exports: 2 (y,n)
16. export-administration-act-south-africa: 2 (y,n)
17. Bias

The inputs that seem to have the greatest effect are [redacted]. There were a couple that didn't seem to have much of an effect at all such as [redacted]. What is interesting is that sometimes a feature will have a large impact on one instance and not have much of an effect in another. This shows that on data sets such as this there are many different ways to split the data while still having good predictive power.



From this chart we can see that after 10 epochs the model does not become any more accurate on the test data. The model can only fit the data so well, so there is a point of diminishing returns on the number of epochs you use when fitting the data. In my model I required that it had a run of improvement (or no change in score) before it would stop. I believe this is why my perceptron tends to run a little longer than needed. As we can see in the graph one of my runs through the data made it 94 epochs, with very little payoff for running that long. If I had more time to work on this lab I would consider implementing a function that detects when the weights are ping-ponging between two values so that I can stop training sooner.

5. Scikit-Learn

After playing around with scikit-learn I was able to achieve similar results to my implementation of the perceptron algorithm. After running the voting task using the same parameters specified in my implementation. It appears that it likes to stop sooner than my implementation. I could get it to converge at just 6 epochs. With a lr of .1 the accuracy averaged 95% on the test data. If I increase the number of iterations to 30, I averaged 97% on the test data. I also ran the perceptron algorithm on my nonlinear separable data. It did slightly better than my algorithm did, averaging around 60% vs 50%. This is a little surprising to me, I was expecting my code to have about the same accuracy on those data. (TA NOTE: Would be appropriate to elaborate, conjecture why it was not the same, if you suspect the difference is material, etc.)

6. Iris

Only did the first model where each class got its own perceptron. It wasn't nearly as accurate as I was hoping for. I was averaging and accuracy of about 50% correctly classified on my test data. My guess is that this is because of the fact that we are relying on the net value on ties. This situation seems to appear quite frequently, and I don't see the net as being a good tie breaker.

TA NOTE: For full credit, author should include some kind of figure (table or graph) that demonstrates the author actually performed the experiment described and supports the discussion.

Conversely, one should never include a figure without some discussion of what the figure shows and why it is relevant/important (e.g. if the spec requires a figure, it implicitly requires some discussion of the figure).