CMPE 452 Assignment 1: Perceptron

Part A

Iris Setosa	33.33%	0%	0%	100%
Iris Versicolor	0%	30%	3.33%	91%
Iris Verginica	0%	0%	33.33%	100%
	100%	100%	91%	96.7%
	Iris Setosa	Iris Versicolor	Iris Verginica	

Preprocessing the data required extraction of the training and test data from their respective text files and converting each row in the text file to an array and maintaining each one of those arrays in an array (2D array). Next, the last index of each array held a string value describing one of the three Iris flower types. I changed that data so that each Iris flower type was represented by a specific binary encoding. To produce that binary encoding, I made the last element of each data point (each internal array) an array of integers where index 0 corresponded to the 1st bit, index 1 corresponded to the 2nd bit, and index 2 corresponded to the 3rd bit. The decision to make Iris Setosa 000, Iris Versicolor 110, and Iris Verginica 111 was arbitrary.

The minimum number of inputs required to build this perceptron model is 3, since 5 inputs can be represented by 3 bits. I decided however, to have 5 input nodes, each node corresponding to one of the 5 inputs. In this case, bias is absorbed so bias is counted as an input. There was no need to build a multilayer neural network since an accurate multi-class perceptron classifier could be developed. With that, no hidden nodes were used. I chose to implement a perceptron with 3 output nodes. 2 outputs are the minimum for a 3-class perceptron since 2 bits can represent 3 outputs. I chose to go with 3 outputs for the sake of simplicity. For the initial weight values, I used a random value generator that gave each element of the weight matrix a random float between -1 and 1. I chose the range -1 and 1 because the actual feature values of the data points were small and were similar to one another. I also limited the number of times the training dataset is looped through. I chose to use 100 epochs, but that is arbitrary. I knew 1 epoch would not be sufficient and a number like 10,000 would be too much. Finally, I chose a learning rate of 0.9 because it falls in the acceptable range of 0.1 to 0.9. I chose not to implement a function that reduces or increases the learning rate depending on the results because I valued simplicity.

Part B

Iris Setosa	33.33%	0%	0%	100%
Iris Versicolor	0%	23.33%	10%	77%
Iris Verginica	0%	0%	33.33%	100%
	100%	100%	70%	90%
	Iris Setosa	Iris Versicolor	Iris Verginica	

The pocket algorithm was slightly less accurate overall than a regular perceptron algorithm, but only by 6.7%. Both algorithms are highly accurate. In the case of the pocket algorithm, it misclassified Iris Verginica as Iris Versicolor a few more times than did the Perceptron algorithm. As a result, the precision of classifying Iris Versicolor was lower, and the recall of classifying Iris Verginica was also lower.

In terms data preprocessing, the approach and implementation were the same for pocket and perceptron. The different design decisions, including number of outputs, number of inputs, the decision to not include a hidden layer, the number of epochs, the initial weight values, and the learning rate were the same. The pocket algorithm builds on the perceptron model, using the same simple feedback learning mechanism, except using memory to remember the longest run weight matrix and selecting that weight matrix for the test data. This means the design considerations for both algorithms could remain identical.