CISC 452 Assignment 1

Aaron Safer-Rosenthal | 17asr | 20068164

Simple Learning Feedback

Confusion Matrix				
Output Class	Setosa	10	8	0
	Versicolor	0	2	1
	Virginica	0	0	7
		Setosa	Versicolor	Virginica
		Target Class		

	Accuracy	Relevance	Recall
Setosa	100%	100%	100%
Versicolor	46.66%	20%	20%
Virginica	86.66%	87.5%	70%

My data pre-processing consisted of a few steps.

First, I read the data into a data frame. Following this, I created two more data frames, one containing the class names and one containing the features. I then made one last data frame, per flower, that would give each class name a binary value. For example, the Setosa frame would have 1's for every Setosa class name instance and 0 for everything else.

I used four input nodes, because each flower had four features.

I did not use any hidden nodes, because this was not required for my implementation of a simple learning feedback model.

I used one output node, which would return the value 0 or 1. It would return 0, when the current node was not predicted to be a part of the set that this perceptron was being trained for and 1 if it was the node. I.e. In the Setosa perceptron, if a flower were predicted to be a Setosa, it would output 1. But, if it were predicted to be a Virginica, it would output 0.

I chose 0.01 for my learning rate. Ultimately, this was the result of trial and error.

Pocket Algorithm

Confusion Matrix				
Output Class	Setosa	10	10	0
	Versicolor	0	6	0
	Virginica	0	0	10
		Setosa	Versicolor	Virginica
		Target Class		

	Accuracy	Relevance	Recall
Setosa	100%	100%	100%
Versicolor	53.33%	37.5%	60%
Virginica	100%	100%	100%

My pocket algorithm worked similarly to the simple learning feedback. It used the same data processing functions, as well as the same number of input, hidden (none) and output nodes. Also similar to the simple learning feedback perceptron, I determined the learning rate through trial and error.

The main difference between them is that the pocket algorithm uses a “pocket” to store weight values. These weight values are taken from the longest run of correct predictions that the perceptron can currently make. When it makes a longer run, it updates the pocket with those weight values. At the end of each epoch, the weight values are updated to match the pocket.

It is worth noting that both perceptrons work better when the data is randomized. In my data processing functions I have a line of code that randomizes them, but I commented it out, because I was not sure if the confusion matrix was supposed to based on the given data in order, or randomized.