CMSC 422: Assignment 1 - Perceptrons – Spring 2019

The purpose of this assignment is to help you gain some familiarity with perceptron learning. Problems 1 and 2 are "paper and pencil" problems (handwritten answers are fine); problem 3 involves doing some simple computational experiments based on the code from Marsland's textbook.

1. Elementary Perceptron Calculation by Hand (paper & pencil problem; a calculator may be used)

Consider an elementary perceptron with four input nodes, a bias node, no hidden nodes, and a single linear threshold output/response node *r* with a_r in {0, 1}.

a) If there are four arbitrary input patterns in random positions, each with 4 random real-valued components (i.e., having dimensionality $d = 4$), what fraction of all possible binary classifications or dichotomies of these patterns would one expect an elementary perceptron to be able to learn?

b) Suppose that four specific input patterns for training are given as

$$\vec{a}^1 = \begin{bmatrix} 1 \\ 1 \\ -1 \\ -1 \end{bmatrix} \qquad \vec{a}^2 = \begin{bmatrix} -1 \\ -1 \\ 1 \\ 1 \end{bmatrix} \qquad \vec{a}^3 = \begin{bmatrix} 1 \\ -1 \\ 1 \\ -1 \end{bmatrix} \qquad \vec{a}^4 = \begin{bmatrix} -1 \\ 1 \\ -1 \\ 1 \end{bmatrix}$$

with a target/correct output of 1 when $\vec{a}^1$ or $\vec{a}^2$ are input, and 0 when $\vec{a}^3$ or $\vec{a}^4$ are input. Assume that the initial four dimensional $\vec{w}_r = \vec{0}$ and that there is an additional bias w_{r0} = -1 (the bias node a_0's activation level is 1), and that a learning rate $\eta = 1$ is used. Show the output node's new weight vector $\vec{w}_r$ and bias value w_{r0} after each step of one pass through these four training examples presented in the order given (left to right) using the canonical perceptron learning algorithm as described in class.

c) Could this elementary perceptron eventually learn to perform this classification by repeatedly cycling through the patterns in a random order? If you say no, then prove that your answer is correct by analyzing the algebraic constraints on the weights and bias value implied by the training data. If you say yes, then give a specific set of weights $\vec{w}_r$ and bias value w_{r0} that enable the perceptron to correctly perform the classification.

2. Measuring Performance

A perceptron with output values of {0, 1} is trained on 1,000 examples of a binary classification problem. Following training, the perceptron classifies 940 of these training examples correctly and 60 incorrectly. When subsequently applied to 200 different, randomly selected examples drawn independently from the same probability distribution, the results are:

		Target Class	
		1	0
Classifier	1:	73	15
Results	0:	19	93

In the context of this information, give the following for this trained perceptron:

a) name of the table displayed here
b) apparent error rate
c) best estimated true error rate
d) 95% confidence interval for the estimated true error rate
e) sensitivity for class 1
f) specificity for class 1

3. A Perceptron that Learns to Classify Radar Signals

You will be provided with Python code in a file *Percept.py* based on Chapter 3 of Marsland's ML textbook (from http://stephenmonika.net, where you can find the original code). Marsland's code has been modified a bit: some comments have been added and a bug has been fixed. You will also be provided with file *radarData.txt* that consists of hundreds of examples of radar pulse returns, with one example on each line. The first 34 real-valued numbers on a line are measurements of the reflected radar signal. The last entry on each line specifies the classification of that signal as either 1 (a good, high-quality return signal) or 0 (a poor, corrupted return signal), values that have been assigned by human radar experts, making this a binary classification task.

a) Using the code provided, you are to write a script file *runp.py* that trains a perceptron to classify radar signals like this as to whether they are good (1) or not (0) based on the 34 input features. Use 80% of the data for training, 20% for testing. To keep life simple, *take every fifth example in the original data set to be testing data*, using the remaining data as training data. Do at least five or 6 runs to get the best post-training results you can. In doing this you may choose to change the initial weights, the learning rate, weight initialization method, and number of epochs until termination to optimize your results. The perceptron should print its results to a file (*Results.txt*). This output file should start with the number of training and testing examples used, the learning rate, number of epochs until termination, and any other run-specific information. It should then report the initial pre-training weights, confusion matrix, and fraction of all possible inputs that are classified correctly before training (separately for both the training and testing data), and subsequently the same information post-training. In generating these results and the answers below, *only report the results that you obtained for the best run that you did*.

b) Answer the following questions based solely on your *best* post-training results in (a):

(i) What are the apparent and estimated true error rates for the trained perceptron?

(ii) Based on the confusion matrix, what was the most common error made by the perceptron on the test data?

(iii) Looking at the data, would you expect that standardizing the input features, as described in Sect. 3.4.5 of Marsland's ML text, and then retraining the perceptron on this standardized data, would substantially improve the performance of the perceptron? Explain why or why not.

Important: Save your results from Problem 3 – you will need these for later comparisons.

What should I turn in?

The hardcopy and electronic submissions are due at different times. The hard copy portion is due at the start of class Thurs. 2/14/19, the electronic submission is due earlier at 11:30 pm Weds. 2/13/19.

Hardcopy: Your answers to the questions in problems 1 - 3.

Electronic submission: For problem 3, turn in a single zip file that includes the result files *Results.txt* plus your script file *runp.py*, just for your single best run. Be sure to include any files you wrote that are needed to make your code run, including the modified *Percept.py* code if you made changes to it. Use the Computer Science Department project submission server at https://submit.cs.umd.edu to submit this zip file.