Homework 03

Due: February 6, 9PM
Point total: 60

Instructions:

- Submit your PDF and/or .py file to Blackboard by the due date and time. Please do not zip your files together, as this interferes with Blackboard's preview functionality. Always show all your work, and for full credit, you must use the method that the problem instructs you to use (unless none is mentioned). Handwritten or typeset solutions are both acceptable, but unreadable submissions will be penalized. You may discuss problems with other students, but you may not write up solutions together, copy solutions from a common whiteboard, or otherwise share your written work or code. Do not use code or language that is copied from the Internet or other students; attribute the ideas *and* rephrase in your own words.

Problem 1 (12 points, 4 each)

For each matrix, find the rank of each matrix using Gauss-Jordan elimination; then also report the number of dimensions in the kernel and the number of dimensions in the range space.

i. $\begin{bmatrix} 1 & 2 & 1 \\ 3 & 1 & -1 \\ 8 & 6 & 0 \end{bmatrix}$

ii. $\begin{bmatrix} 2 & 2 & 3 & 4 \\ 5 & 1 & -1 & 0 \\ 0 & -2 & 1 & 2 \\ 3 & 4 & 1 & 1 \end{bmatrix}$

iii. $\begin{bmatrix} 1 & 2 & 3 & 4 \\ 3 & 6 & 9 & 12 \\ 9 & 18 & 27 & 36 \end{bmatrix}$

Problem 2 (10 points, 2 each)

For each matrix, find the two-sided inverse, or explain how you know there isn't one. (You may wish to check your work with a matrix multiplication.)

i. $\begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix}$

ii. $\begin{bmatrix} 1 & 3 \\ 2 & 9 \end{bmatrix}$

iii. $\begin{bmatrix} 1 & 2 & 3 \\ 3 & 6 & 8 \\ 4 & 8 & 11 \end{bmatrix}$

iv. $\begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$

v. $\begin{bmatrix} 1 & 2 & 3 \\ 3 & 2 & 1 \\ 3 & 3 & 3 \end{bmatrix}$

Problem 3 (9 points, 3 each)

In each case, project $\vec{v}$ onto the line that is the span of $\vec{s}$ to get $\text{proj}_s v$. Then, subtract to find the component of $\vec{v}$ orthogonal to that line.

i. $\vec{v} = \begin{bmatrix} 1 \\ 4 \end{bmatrix}, \vec{s} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$

ii. $\vec{v} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}, \vec{s} = \begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix}$

iii. $\vec{v} = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix}, \vec{s} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 0 \end{bmatrix}$

Problem 4 (11 points [4, 4, 3])

i. Which weights (if any) can change in a perceptron in response to a training input with all zeroes for features ($\vec{0}$)? Why can't the others change?

ii. Suppose we would like to have a perceptron that responds TRUE (1) to the binary input pattern $\begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \end{bmatrix}$, and FALSE to all other binary inputs. (We don't care about its behavior for inputs that are not vectors of 1's and 0's.) Give the parameters $w_1, \ldots, w_4, b$ for a 4-input perceptron that will respond TRUE to this input and false to all other binary inputs. (Assume the underlying function is of the form $w_1x_1 + \ldots + w_4x_4 + b \geq 0$.) Just the five working parameters are necessary; you don't need to show that they can be learned.

iii. Suppose I have a machine learning problem where the most relevant feature is an angle in the range $[-\pi/2, \pi/2]$. The perceptron ultimately should return TRUE only for examples in the range $[-\pi/4, \pi/4]$, and FALSE for all other examples. Unfortunately, a single linear classifier can't have both an upper and lower threshold, so it won't be able to learn that range. Suggest a trigonometric function that I can apply to this feature before presenting it to the perceptron, so that the modified input is now linearly separable.

Problem 5 (18 points)

For this part, download `perceptron.py`. You'll fill in the learning function that adapts the perceptron to the data.

To help visualize the data, we'll work with some three-dimensional data: color values, which are triples of (red, green, blue). The goal will be to train a perceptron classifier to respond "yes" (1) to gold or yellow colors, and "no" (0) to other colors. Since perceptrons can only represent linear decision surfaces - in this case, a plane - it will not be able to classify points with 100% accuracy; we'll shoot for simply having accuracy in excess of 70%.

The representation of the perceptron is just 4 numbers in a list, [a, b, c, d]. They represent the decision function $ax + by + cz + d \geq 0$, where (x, y, z) is the color triple of the input.

First, fill in the function `perceptron_predict()`. This function takes a perceptron [a, b, c, d] and a color value (x, y, z), and returns 1 if $ax + by + cz + d \geq 0$, and 0 otherwise. (This won't work very well as a predictor before training.)

Then fill in the function `perceptron_learn()`. This function should take a 4-element list representing a perceptron, a training example (color triple), a classification (1 for gold, 0 for otherwise), and a learning rate which you should just set to 0.01. If the perceptron's prediction matches the true classification, nothing should happen, and the function should just return the perceptron unchanged. But if the perceptron was wrong, apply the perceptron learning rule covered in class, and return a new perceptron.

It's often a good idea in machine learning to go through the training data in a random order, as well as to scale all the values to lie in [0,1]; but this code has already been written for you as a part of the `create_data()` function. In addition, I've written some matplotlib code to show you graphically what your perceptron's decision surface looks like. Once you're done, the following sequence of commands should serve to show you the perceptron's decision surface both before and after training.

```
import perceptron
ptron = perceptron.new_perceptron()
examples, classes = perceptron.create_data()
perceptron.plot_everything(ptron, examples)
ptron = perceptron.perceptron_train(ptron, examples, classes, 0.01, 100)
perceptron.plot_everything(ptron, examples)
```

The colors of the points are the actual colors of those triples. You should see gold points mostly on one side of your plane, and the other colors mostly on the other.

Submit your .py code along with your assignment PDF.