Hunter Harralson

Psych 186B: Homework 5

In my code, the main actions occur in Hw5.m with calls to functions for each specific action. I will outline each function and my processes for them.

1. The first thing I did was create an excel file for the 20 ships the network would learn from. In Hw5.m I read the file in and store it in a cell array matrix called *training_data*.

2. I then passed *training_data* into the function, **getFeatures**, which essentially looped through each column, or ship, in the training data and extracted the features (name, murds, transponder frequency, color, and ratio).

   How I represented the features: For each feature, give the ship a probability that the feature belongs to a certain planet (Klingon, Romulan, Antarean, or Federation). So, for each feature of each ship, there were four values. For example, if it was obvious that the name of the current ship belonged to a ship of the Klingon fleet, the four values stored would be 1, 0, 0, 0 because the probability of the ship being a Klingon based on that feature is 1. The reason 1 is stored in the first index as opposed to say the third index for this feature is because it represents Klingon, which is the first planet listed in the way I set the model up. The order is Klingon, Romulan, Antarean, Federation. So, if it was obvious that the ship color belonged to Antarean, then the 1 would be in the third index.

   I then stacked the four values for each feature on top of one another. So, for five total features, each having four values, each ship would have 20 values stores. The output of getFeatures is a 20x20 matrix called *input_data*. Each column of this matrix represents a ship, with each row representing the probability of a ship belonging to a certain planet depending on the row. Each column was then essentially an f vector, which I normalized at the very end of the function.

3. Next, I passed this matrix of f vectors, *input_data*, into the function **linearAssociator**, which returned *A*, the linear associator. In this function, I wrote a nested for loop that created a vector of values by placing 1 (planet probability) in the appropriate spot based on which planet should be associated with each ship or f vector (gi). For each fi and gi association, I added the individual linear associator, *Ai*, to the overall linear associator, *A.*

4. Following this, I applied learning through the use of the delta rule and gradient descent with the function **errorCorrection**. This function returns the updated linear associator after learning, called *newA*. The first thing I did was similar to the nested for loop in the last function that created desired outputs based on the correct planet. For this, I created a 20x20 matrix that stores the desired output vectors. I then created a vector of 20 random values from 1 to 20 so

that error correction did not occur sequentially, but randomly based on which number is next in the random number vector. In my for loop for learning, I randomly chose f vectors from *input_data*, computed actual output, *g_prime*, and compared it to the desired output, which came from the matrix I had just created. I then applied the delta learning rule to the linear associator and decreased the learning constant as 1/current learning trial number.

5. After learning occurred, I read in the noisy data and stored it as *noisy_data*. I then sent it through getFeatures, which was giving me errors at first because my feature extraction did not account for blank values and special characters. For this reason, I had to adjust the function to handle these cases. The way I represented the noisy data in the excel file was to leave blanks where there were underscores for name, and leave blanks for missing values. I input colors and frequency just as they are listed on the assignment. The way I changed the function was to recognize blank spaces as NaN and thus assign 0s to all four values for the given feature. I also added an if statement for color such that if the word or occurred to just set color to the first color listed and ignore the second. I figured that giving it one color was better than treating it as a blank and thus assigning no value for the color. In dealing with the less than and greater than cases for frequency, I just assigned values of zero to the feature vector if the input was NaN.

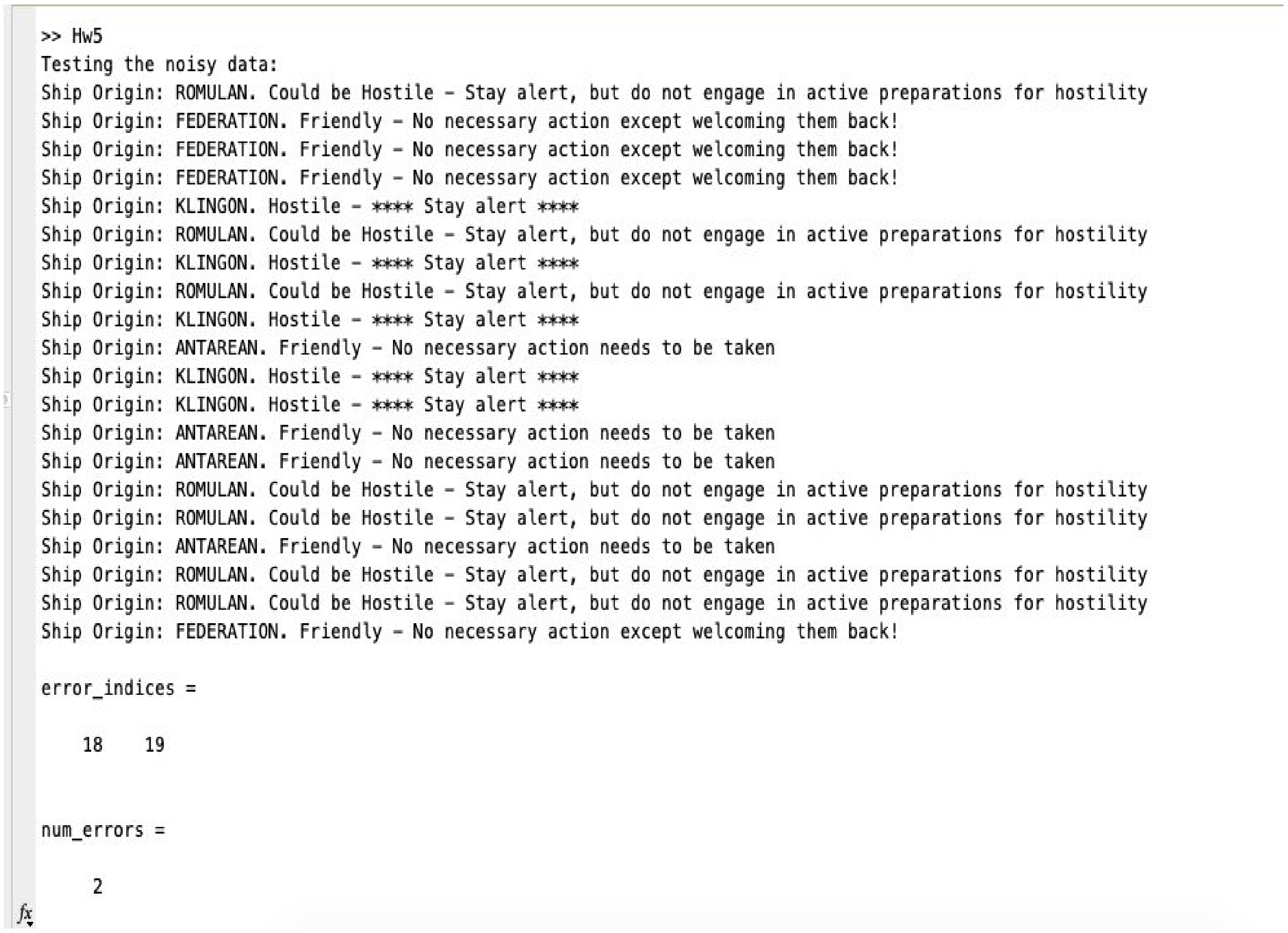
6. In testing the correctness of the model in classification, I created the function **testPlanets**, which found the max value of each gprime for each ship. It then found the index of that max value which corresponded to a planet of origin. For example, if the max value occurred at index 2, then the model had classified that ship as belonging to the Romulans. I then stored this index, or origin planet, for each ship and returned these in a vector called *planet_vec*. This function also displayed the planet classification for each ship and the required action.

7. The last step was passing this *planet_vec* into the function **errorCheck**, which compared each planets classification as determined by the model with what it should be, as determined by me in eyeballing the noisy data, which is pretty obvious. This function then returns a vector storing the indices of the ships that were incorrectly identified. For example, if the model incorrectly classified ship 6 and 12, it would return a vector storing [6 12]. In the Hw5.m file, the last thing I did was tally up these errors by taking the size of this vector in a variable called *num_errors*.

So, how did the neural network perform?

The number of errors tinkers between 1 and 2, with the most difficulty coming from the 18th and 19th ships. This makes sense because the 18th ship has no data besides its name, which is clearly Antarean to me, but not to my model. Perhaps I could find a way to place more of an emphasis on the weight of a feature when it is obvious that one feature belongs to a certain planet.
For the 19th ship, I believe the error occurs in the fact that I chose not to deal with the greater than symbol and instead ignored that feature for the ships that had it. This along with the fact that the color is ambiguous and the only other feature, Ratio, is in the middle, likely leads to a mis-classification.

My output:



```
>> Hw5
Testing the noisy data:
Ship Origin: ROMULAN. Could be Hostile - Stay alert, but do not engage in active preparations for hostility
Ship Origin: FEDERATION. Friendly - No necessary action except welcoming them back!
Ship Origin: FEDERATION. Friendly - No necessary action except welcoming them back!
Ship Origin: FEDERATION. Friendly - No necessary action except welcoming them back!
Ship Origin: KLINGON. Hostile - **** Stay alert ****
Ship Origin: ROMULAN. Could be Hostile - Stay alert, but do not engage in active preparations for hostility
Ship Origin: KLINGON. Hostile - **** Stay alert ****
Ship Origin: ROMULAN. Could be Hostile - Stay alert, but do not engage in active preparations for hostility
Ship Origin: KLINGON. Hostile - **** Stay alert ****
Ship Origin: ANTAREAN. Friendly - No necessary action needs to be taken
Ship Origin: KLINGON. Hostile - **** Stay alert ****
Ship Origin: KLINGON. Hostile - **** Stay alert ****
Ship Origin: ANTAREAN. Friendly - No necessary action needs to be taken
Ship Origin: ANTAREAN. Friendly - No necessary action needs to be taken
Ship Origin: ROMULAN. Could be Hostile - Stay alert, but do not engage in active preparations for hostility
Ship Origin: ROMULAN. Could be Hostile - Stay alert, but do not engage in active preparations for hostility
Ship Origin: ANTAREAN. Friendly - No necessary action needs to be taken
Ship Origin: ROMULAN. Could be Hostile - Stay alert, but do not engage in active preparations for hostility
Ship Origin: ROMULAN. Could be Hostile - Stay alert, but do not engage in active preparations for hostility
Ship Origin: FEDERATION. Friendly - No necessary action except welcoming them back!

error_indices =

    18    19


num_errors =

     2
```