

I first inputted the categories of each spaceship to be faithful to the original. However, each category was put in different functions, and each functions outputted a few dimensions. For reference, the figure below shows how the functions were coded.

```
K1 = [TurnAscii('Grotz'), TurnWarp(6.9), TurnFreq(1006.4), TurnColor('Black'), TurnRatio(3.5)];  
K2 = [TurnAscii('Tlarr'), TurnWarp(7.0), TurnFreq(994.3), TurnColor('Black'), TurnRatio(2.3)];  
K3 = [TurnAscii('Tribok'), TurnWarp(7.3), TurnFreq(978.1), TurnColor('Dark Gray'), TurnRatio(2.8)];  
K4 = [TurnAscii('Brogut'), TurnWarp(7.1), TurnFreq(1005.4), TurnColor('Dark Gray'), TurnRatio(3.0)];  
K5 = [TurnAscii('Glorek'), TurnWarp(7.1), TurnFreq(1001.8), TurnColor('Light Gray'), TurnRatio(1.0)];
```

In each function, the categories were split into specific features. These specific features were yes or no, coded in binary. For example, for a certain feature, if the value was bigger than a certain number, the output is 1, otherwise, 0.

## Functions

For the 'Name' category, the inputted characters were turned all lower-cased, then turned into ascii, and averaged. If the average was less than around 90, the output would be 1, otherwise, 0. This is because if the average is less than 90, it must mean the planet name consists of mostly numbers, which indicates it has to be from Antarean.

For the 'Warp Drive Vibration Index' category, the binary were set depending on the size. For example, if the number is super large ( > 7.1 ), it would be 1, otherwise, 0. This was done for each range size, with each range corresponding to each planet.

For the 'Hailing Transponder Freq' category, binary numbers were once again set depending on the range of numbers, each range roughly corresponding to each planet.

'Color' category was the most complicated of them all, but essentially, the color was split into "brightness" and "color name", and "colorfulness." The input was separated into two different characters, separated by space: brightness and color name. If there were no white space, the program assumes there were no brightness in the input, and only the color name. Brightness was either light or dark, corresponding to 0 and 1 respectively. If there were no mentions of brightness (no prefix light/dark), the answer was 0.5, which essentially means ambiguous. There were a total of 8 different colors, which we all binarily coded into 3 different dimensions (since 3 digits in binary can code for 8 different instances.) For example, white was 000, yellow was 001, orange was 010 and so on. Furthermore, another dimension was added, called colorfulness. This dimension would be 0 if it was black, white, or gray, and 1 otherwise.

Finally, the 'ratio of long to short axis' was converted similarly to the 'Warp Drive Vibration Index' and 'Hailing Transponder Fre', in which there were ranges of numbers, each range corresponding to a certain planet.

In total, there were a total of 23 dimensions/features per spaceship. Each feature of the spaceship were put in the rows, and each spaceship corresponded to the column, leading to a 23x20 f-matrix (input matrix).

### **Converting: Normalized + mean 0 + -0.5 to 0.5**

This f matrices were then made from -0.5 to 0.5, centered roughly at zero, and then normalized. This was accomplished by first getting the max of a category from the min of a category, which would later serve as the lower and upper bound (-0.5 and 0.5, respectively). Then a new f was calculated by first subtracting each feature in the category by the categories' min, which would mean that the min for each category is currently set at 0. This value was then divided by the result of (max - min), which would effectively give me values from 0 to 1, where the 0 corresponds to the min, and 1 is the max of the category. This new f was then subtracted by 0.5 to give me a range from -0.5 to 0.5. The center is approximately at 0, though not exactly because the distribution of each category may not have been normally distributed. Finally, all features of each spaceship were normalized by dividing the current f matrix with the length of the fmatrix, using the built in norm function.

### **Test/Noisy Data**

The same functions and converting method were used for the test data (or the noisy data), which will be referred to as t-matrix (test-matrix).

The test data was complicated because some category of a spaceship was missing. For the missing data, I inputted a zero. Shown below.

```
Test(:,18) = [TurnAscii('E4511') TurnWarp(0), TurnFreq(0),TurnColor('0'), TurnRatio(0)]';
```

If 0 was the input, the functions would output 0.5 for all the dimensions. For example, the TurnColor is supposed to output 5 dimensions, each either 0 or 1s (corresponding to brightness, 3 dims for color name, and colorfulness). The output for t-matrix for the colors category would be five 0.5, which means that the output is ambiguous, and should not be affecting the calculations for the Af/g' matrix.

## **Association Matrix**

The A matrix was created using the recently calculated f-matrix and a newly created-g matrix. An association was made through the outer-product of g and a transposed f. There were 20 iterations for each spaceship, and, the new association was added to the A-matrix for each iteration. The g-matrix was created by randomly sampling from a uniform distribution from 0 to 1, then subtracted to get values from -0.5 to 0.5. This was then normalized to get the length to equal around 1. The g-matrix was only created 4 times, at iterations 1, 6, 11, and 16. This was so that each g-matrix would correspond to a certain planet. In other words, since iterations 1-5 were all adding f-matrices of different spaceships corresponding to the same planet, the g-matrix, or the output, should be the same for all 5 of them because we are trying to find the planet of origin. Same goes for iterations 6-10, 11-15, and 16-20.

Through this, we have an association matrix that stores 20 spaceships details, f, with 4 different output gs, each gs that corresponds to a certain planet.

## **Comparing Test-Data to Stored Archival-Data**

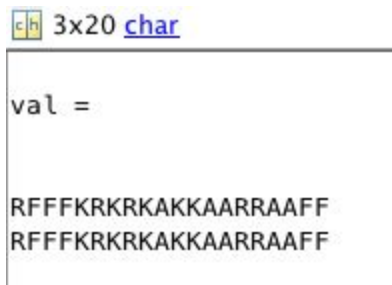
Now, we want to compare how well the Association-matrix created from the Archival Data compares to the noisy test data. I first calculated the g' matrix for each spaceship, which was done by A-matrix \* Testing data. This created g', the output, which are the program's guess for the type of spaceship from the archival data, g.

For each g', the output was compared with every stored g (the archival data of spaceships) by the cosine angle, which is the dot-product of the g' with one of the stored g, divided by the norm of the g'. This meant that for each g', the cosine angle had to be calculated 20 times (one g' with each of 20 gs) and this had to be repeated for each g'. So an inner loop calculated the cosine for one g' with every stored g, and the outer loop would repeat this for every g'.

Essentially, by comparing one output g' with all the stored g, we can see how likely that this new spaceship is from the same planet of all the previously seen spaceships. Ideally, if the newly identified spaceship is close to a spaceship from a certain planet, the new spaceship should have a cosine angle that is close to those known-spaceships that came from the same planets.

In one iteration of the outer loop, 20 cosine angles were calculated for one  $g'$  with every  $g$ . The first 5 cosine angles were then averaged, because they all correspond to the same planet of Klingon. If they are from the same planet, the output of  $g'$  should be somewhat similar to all 5 of the stored  $g$ s. The next 5 cosine, which corresponded with the planet Romulan, were averaged as well, and so on. After these calculations, I ended up with 4 average cosines, each average cosine corresponding to how likely this  $g'$  was from one of the 4 planets. The higher the number of the average cosine was for a particular planet, the more likely the spaceship was from this planet. The biggest average cosine was then determined to be the planet, and the corresponding letter of the planet was outputted as follows:

## Results



```

ch 3x20 char
val =
RFFFKRKRKAKKAARRAAFF
RFFFKRKRKAKKAARRAAFF

```

Above figure shows the typical result I got. The top layer is my program's guess from the planet. The bottom layer is my own guess. K correlates with Klingon, R correlates with Romulan, A correlates with Antarean, F correlates with Federation. As the results reveal, the program was on par with my own guess.

It is important to note that since a random  $g$  is created every time the program run, the output differs a little each run. To get the average correction rate, I ran this program 1000 times (each time the  $g$  changes) and calculated, in average, how correct the program is to my guess. The average correct rate came out to be 99.9%. This means that if I run it 1000 times, essentially I get 20/20 correct every time, except for a few rare instances where the program get 19/20 right. What this means is the program, without even needing supervised learning, is able to guess the planet from the noisy input. Since the programs answer are related to my guess, the program successfully was able to successfully mimic the human cognition in coming up with the right answer (whether my own guess was right or not, the program's answer was as good as humans).

The bottom table indicate the full output of the planet of origin in order, and the actions required for a typical output of my program.

<b>Data</b>	<b>Planet of Origin</b>	<b>Required Action</b>
1	Romulan	Alert
2	Federation	Friendly
3	Federation	Friendly
4	Federation	Friendly
5	Klingon	Hostile
6	Romulan	Alert
7	Klingon	Hostile
8	Romulan	Alert
9	Klingon	Hostile
10	Antarean	Friendly
11	Klingon	Hostile
12	Klingon	Hostile
13	Antarean	Friendly
14	Antarean	Friendly
15	Romulan	Alert
16	Romulan	Alert
17	Antarean	Friendly
18	Antarean	Friendly
19	Federation	Friendly
20	Federation	Friendly

## Notes

There were a few instances where the input was really noisy, for example the Hailing Transponder Frequency read “ > 1000” or “ <1000”. When this happened, I simply got the standard deviation of the archival data for the frequency, and added one sd to 1000 for “>1000” and subtracted one std for “< 1000.” I used std because this was the the guess that should be around the ballpark for how a person would guess the frequency would be. However, even if I set it at 1001 or 999 for example, the program still worked fine mostly due to the other categorical data that swayed the noisy data to the correct planet. For the color that was “Black or Dark Blue,” it did not matter if I put black, dark blue, or blank. The color did not affect my output.

Interestingly, I never gave out any answer with 100% certainty from inputs that were “give-aways”. For example, if the ‘Ratio of long to short axis” was less than 1.5, the planet should always be from Antarean, but I did not program so that the answer would always be Antarean. The binary for whether it was less than 1.5 was given a relatively same weight as everything else. Perhaps, this type of programming helped me get the output for test-data 7, which was an example of an atypical Klingon. It was one of the harder ones to guess myself, but the program guessed the same answer as well. Furthermore, this may also explain why the noisy data of color “Black or Dark Blue” did not affect my output; because the other data were given as much consideration.