Jeannie Fu (jf586)
Nikita Gupta (ng354)
Richa Deshpande (rd382)
Tori Seidenstein (tbs52)
Kaggle Team Name: RasberryPie

December 7, 2016

# CS 4786 Competition #2: Find the Bot Challenge

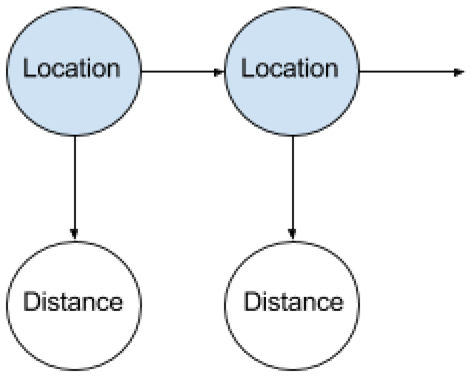
## Overview

Before creating our first model, we established a fundamental understanding of the challenge and data provided to us. Row indexing is 0-4 (as noted by the Professor in a Piazza post), column indexing is 1-5, and the observer is located at (0,0). We calculated the distance to each grid square from the observer based on the row and column number. These distances are the emission values from a robot at a given square. The distances are shown in the table below with duplicate distances color coded. Each square is given a label 1-25. We noticed that there were 18 unique distances.

| **OBSERVER** | **1**: 1.0000 | **2**: 2.0000 | **3**: 3.0000 | **4**: 4.0000 | **5**: 5.0000 |
|---|---|---|---|---|---|
| | **6**: 1.4142 | **7**: 2.2361 | **8**: 3.1623 | **9**: 4.1231 | **10**: 5.0990 |
| | **11**: 2.2361 | **12**: 2.8284 | **13**: 3.6056 | **14**: 4.4721 | **15**: 5.3852 |
| | **16**: 3.1623 | **17**: 3.6056 | **18**: 4.2426 | **19**: 5.0000 | **20**: 5.8310 |
| | **21**: 4.1231 | **22**: 4.4721 | **23**: 5.0000 | **24**: 5.6569 | **25**: 6.4031 |

We used the labeled data for the first 200 runs as preliminary validation for results we obtained. After creating each of our models, we compared our predicted 100th state for the first 200 runs with the known 100th state to evaluate the accuracy. This helped us minimize Kaggle submissions because we had a rough measure of accuracy, and could tell if the data would be worth submitting to Kaggle.

In our HMM the hidden states represented locations, and the observed states the distance the bot was from the origin. The location was not visible, however the distance was dependent on the hidden states.



We decided to code our approaches in MATLAB. The built-in Hidden Markov Model functions that we used are listed below. They are explained in more detail in our approach sections.

```
[ESTTR,ESTEMIT] = hmmtrain(seq,TRGUESS,EMITGUESS,'Symbols',SYMBOLS)
% Returns ESTTR, estimated transition matrix, and ESTEMIT, estimated emission matrix,
% using Baum-Welch. Inputs are seq, a sequence of emissions and initial matrices
% TRGUESS and EMITGUESS. SYMBOLS is a numeric array of possible emissions

STATES = hmmviterbi(seq,TRANS,EMIS,'Symbols',SYMBOLS)
% Returns STATES, the most likely path through a HMM given seq, a sequence of
% emissions, matrices TRANS and EMIS, and SYMBOLS, a numeric array of possible
% emissions
```

## Approach #1: Basic approach based on distance

Our first submission was simply a benchmark for our later approaches as a rough estimate of bot locations. We constructed our submission by mapping the distance at the 100th step to the square with that distance, using the table on the previous page. In the case where multiple squares could have produced that emission, we chose one randomly. This naive approach achieved a Kaggle score of 88.9%.

## Approach #2: Basic approach based on distance and manual observation of data

By manually looking at the labels.csv file, we noticed that among grid squares with shared distances, some of the squares occurred more than other squares of the same distance. We adjusted our naive model from approach #1 slightly. In the case where multiple squares could have produced a given emission, we selected the square that occurred most frequently in labels.csv. This approach achieved a slightly better Kaggle score of 91.5%.

## Approach #3: HMM with 25x25 Transition Matrix

Next we tried an approach using Hidden Markov Models. In this first approach with HMM, we created a 25x25 transition matrix and a 25x18 emission matrix. For now, we decided to neglect the fact that there are 3 different bots, and to try to just get the built-in MATLAB functions working. Each of these matrices and the rationale behind the dimensions are explained in more detail below.

We created an initial 25x25 transition matrix, where TRANS(i,j) represents the probability of jumping from state i to state j. We decided to use 25 states, each of which represent the grid locations numbered 1-25. The states would map directly to a grid location, such that state #1 maps to location #1, state #2 maps to location #2, etc. We assumed that the bots were much more likely to move one step up, down, left, or right, and that there was a small probability they would jump to an entirely new location on the grid. We initialized our transition matrix to address three scenarios:

Figure 1 illustrates the scenario where the bot is in the corner and there is an approximately 50% chance that the bot will move along either edge. Figure 2 illustrates the scenario where the bot is along one edge and has approximately a 33.3% chance of moving up, down, or towards the center of the grid. Figure 3 illustrates that when the bot is in an interior square, it has an approximately 25% percent chance of moving up, down, left or right.

| | | | | |
|---|---|---|---|---|
| Bot | .5 | | | |
| .5 | | | | |
| | | | | |
| | | | | |
| | | | | |

Figure 1

| | | | | |
|---|---|---|---|---|
| .33 | | | | |
| Bot | .33 | | | |
| .33 | | | | |
| | | | | |
| | | | | |

Figure 2

| | | | | |
|---|---|---|---|---|
| | .25 | | | |
| .25 | Bot | .25 | | |
| | .25 | | | |
| | | | | |
| | | | | |

Figure 3

Thus we created a transition matrix where the the 25 rows represent the bot's position at some arbitrary time t and the 25 columns represent the bot's position at time t+1. We initialized the matrix based on the scenarios described in the figures above. We zeroed out all other entries of our initial transition matrix to reflect an initial guess that a bot had a zero probability chance of jumping to a non-neighboring square.

We also created a 25x18 emission matrix. The rows represented the 25 squares in the grid and the columns the 18 unique distances. We created an emission matrix where EMIS(i,j) represents the probability of state i emitting symbol j, as required by the built-in MATLAB function we used. We created a mapping of the distance emissions to the symbol index, by representing each unique distance in order from 1-18. The emission matrix is actually deterministic, because we know exactly which distance a given square will emit. Because each square was correlated with a single distance, the emission matrix was initialized so for a state i that emits symbol j, EMIS(i,j) = 1 and all other columns in row i are set to 0.

After creating our initial transition and emission matrices, we ran MATLAB's built-in function hmmtrain which runs the Baum Welch algorithm. We passed in initialized matrices but the values were not completely accurate and just estimates. This was not a problem since Baum Welch adjusts the transition and emission matrices iteratively until they converge to reflect the hidden and observed variables. A function call example is given again below (refer to “Overview” section of writeup for function spec):

```
[ESTTR,ESTEMIT] = hmmtrain(seq,TRGUESS,EMITGUESS,'Symbols',SYMBOLS)
```

We tried running hmmtrain first on just a single row of observations.csv, which represents a single sequence of emissions, and then on a slightly larger set of 10 rows. We input SYMBOLS to represent the 18 unique distances in order of squares. Since we actually do not need an estimated emission matrix, we chose to call this function with a random 25x18 matrix for EMITGUESS, and did not use the output emission matrix ESTEMIT. We recognize that Baum Welch does the matrices independently, so a random emission matrix will not affect the output transition matrix.

After running hmmtrain to obtain an estimated transition matrix, we called hmmviterbi to construct a guess at the sequence of states for each row of observations.csv. An example function call is given below (refer to “Overview” section of writeup for function spec):

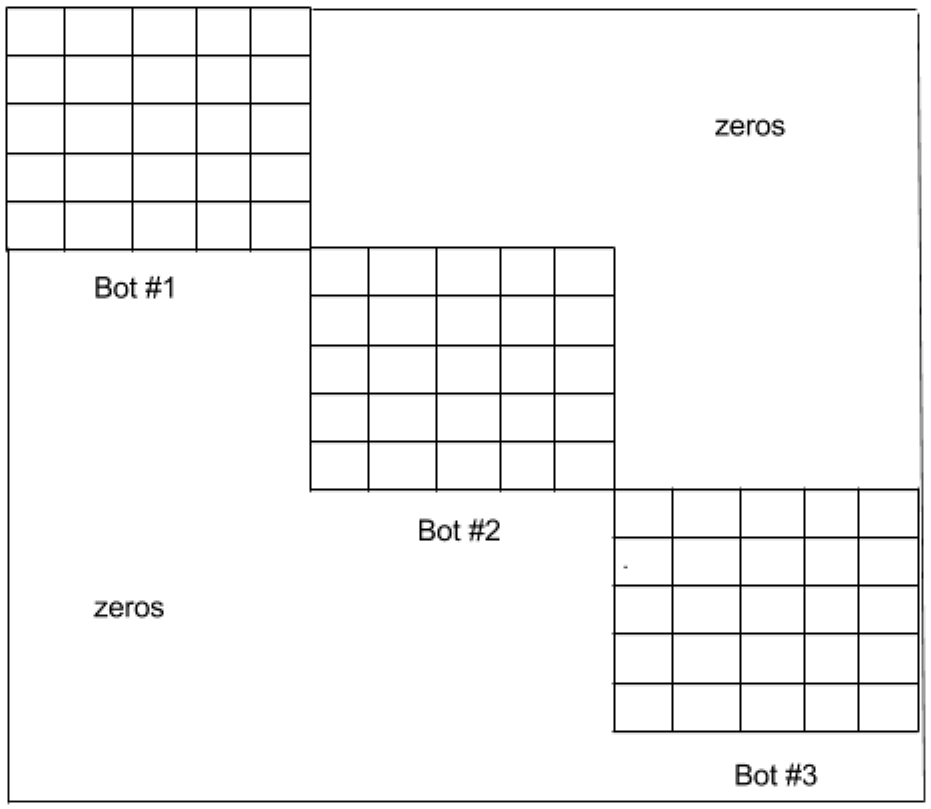
```
STATES = hmmviterbi(seq,TRANS,EMIS,'Symbols',SYMBOLS)
```

We did not submit this result to Kaggle. We could tell by comparing labels.csv and our output that our accuracy with this approach was very low, which was expected since this was a naive model that did not take the 3 bots into account.

## Approach #4: HMM with 75x75 Transition Matrix

In order to account for the fact that there were three bots, we wanted the transition matrix to include a combination of the transition matrices of all three bots. We tried to create a model where each bot had its own set of 25 states. State 1 would correspond to bot #1’s state 1, state 26 would correspond to bot #2’s state 1, state 51 would correspond to bot #3’s state 1, etc.

We used the same procedure described in Approach #3 to construct each individual bot’s transition matrix and then put the three 25x25 matrices together in a 75x75 matrix as shown below. The empty gaps were all filled in by zeroes to represent that a bot could not jump to another bot’s states (i.e. that in a single run, we constrain bots to states 1-25, 26-50, or 51-75).



We created a 75x18 emission matrix to go along with this transition matrix. We created this matrix in a similar way by combining the three individual emission matrices of the bots as described in Approach #3. To combine them we simply stacked the three 25x18 matrices vertically.

We used the same functions hmmtrain and hmmviterbi to try to reconstruct the sequence of states. However, we saw that hmmtrain seemed to only set the upper 25x25 subgrid of our transition matrix, and did not accurately set the other two subgrids.

Again, we did not submit to Kaggle. Our transition matrix was not representing the three bots correctly, and we could tell from comparing our output to labels.csv that our accuracy was very low.

## Approach #5: Separate Models for each Bot and Clustering

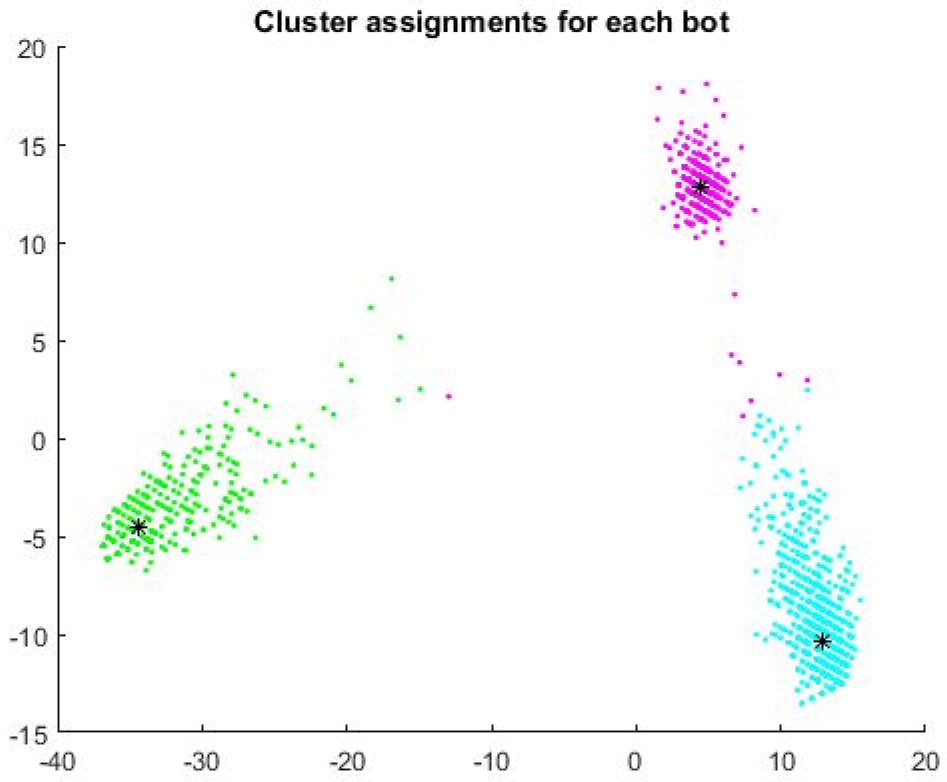
Ultimately, we decided that the best way to handle the fact that the three robots have distinct probabilistic movement patterns was to train three separate models. To do this, we first needed to identify the robot for each of the runs in the observation matrix. We did this by reworking the observations matrix so that we could run K-Means to cluster.

By manually examining the observation matrix, we realized that each bot has a set of grid squares it tends to visit most, and that the squares are different for each robot. We created a new matrix with rows representing runs and columns representing the unique distances 1 to 18 on the grid.

Then we populated the matrix such that M(i,j) was the number of times that the robot of run i visited square j during its run. Thus, our matrix represented the frequency with which each grid square was visited during the run.

We ran the K-Means algorithm to cluster the runs into 3 clusters, each representing the runs of one robot. We separated the observations data into three different data sets. There were approximately 1200 runs for Robot 1, 1200 runs for Robot 2, and 600 runs for Robot 3. We noticed after running the HMM algorithms, our state guesses for the bots tended to vary in accuracy. One bot consistently had accurate guesses, another varied slightly, and the other varied a lot. To ensure that the assignment of bots to each cluster remained the same every time we ran our code, we re-ran K-Means with the centroids produced by the algorithm set as the initial centroids. This allowed us to consistently identify the bot for which the guesses were the least accurate so that we could train it on a larger data set than the other two.

Below is a plot of the cluster assignments for each bot. We performed PCA to scale our matrix down to 2 dimensions. We appended the centroid locations to the end of our matrix as well, to also scale the centroids to 2 dimensions with PCA. This should not affect the representation too much because we have a large amount of points. The numbers on the axes do not actually represent anything quantitative but are created just for plotting. Centroids are marked with a black asterisk.



For each robot, we used the built-in hmmtrain function to generate transition matrices.

```
[ESTTR,ESTEMIT] = hmmtrain(seq, TRGUESS, EMITGUESS, ‘Symbols’, SYMBOLS)
```

seq was the first 10 runs of that particular robot from the observation matrix. The rest of the inputs were the same for each robot. TRGUESS was the same 25x25 initial transition matrix we created earlier in Approach #3. EMITGUESS was a 25x18 random matrix. As described earlier, we only needed hmmtrain to create a transition matrix since our emission matrix is deterministic. Again, SYMBOLS was the 1 x 18 array of unique distances.

Next, we ran the hmmviterbi function to determine the states at each step.

```
STATES = hmmviterbi(seq, TRANS, EMIS, ‘Symbols’, SYMBOLS)
```

seq and SYMBOLS were the same as in the hmmtrain function, EMIS was our own emission matrix, and the TRANS matrix were the ones obtained from hmmtrain. We ran into the following error at this point: “A zero transition probability was encountered from state 1.“ We added a tiny bit of probability (0.0001) to our transition matrix to remedy this.

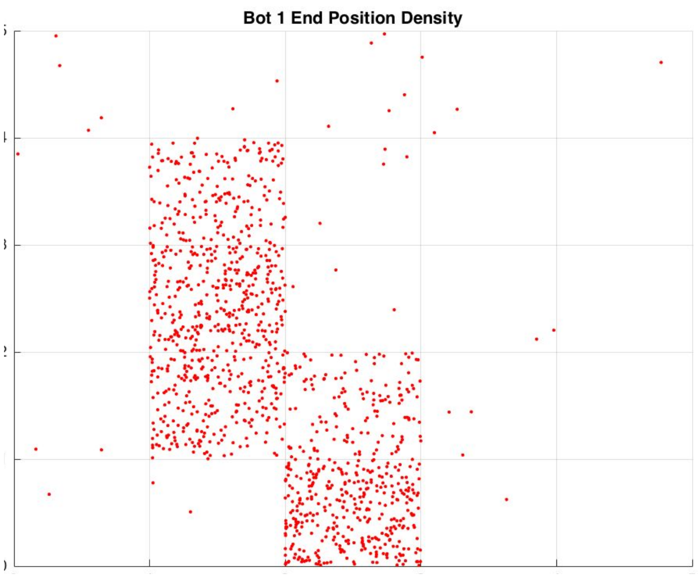
Finally, since we split the data into three data sets for each bot, we constructed the output matrix by merging the information from the three bots back together in the order of the 3000 runs, and returning the state (grid square) the robot was at during step 100 of each run.

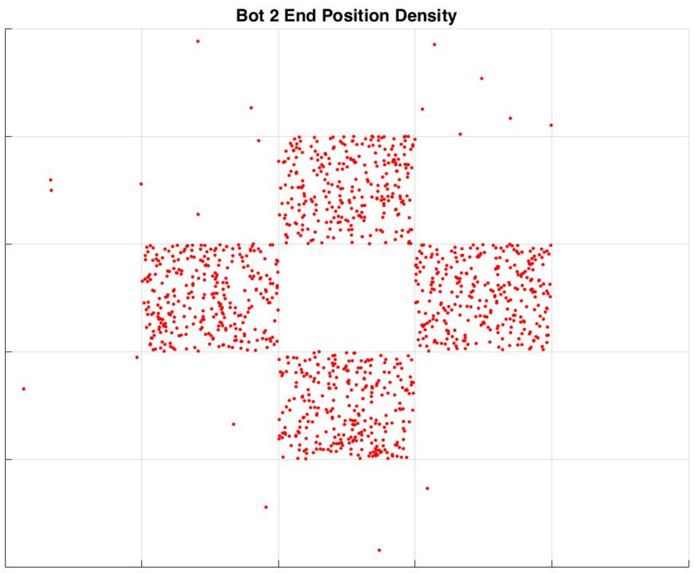
We ran this approach multiple times and got different predictions each time due to some randomness. After each run, we checked with labels.csv to see how our algorithm did by printing the 100th step for several runs for each bot. We submitted the ones we felt did the best based on labels.csv.

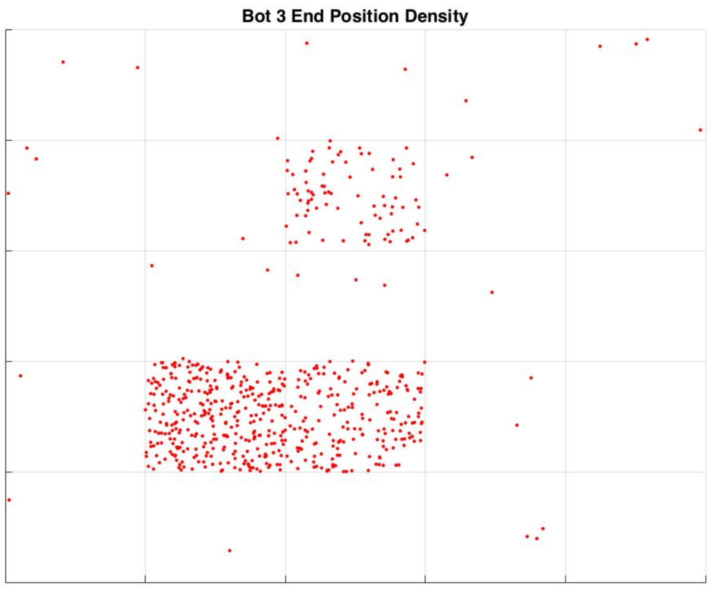
## Approach #6: Tweaking Parameters

This approach builds off of Approach #5. Once we implemented the clustering approach and the independent models for each bot, we made some optimizations. We utilized the training data to identify which bots were having the most difficulty distinguishing between equidistant grid squares. Comparing the states that our model predicted for the first 200 states with the labeled data, we saw that Bot 1 in particular difficulty distinguishing between squares 8 and 16 and squares 5 and 19. Bot 2 had some difficulty, but not as much, and Bot 3 reproduced all of the corresponding elements in labels.csv correctly. We adjusted the model for Bot 1 to train on 30 runs instead of 10 and this gave us a final Kaggle score of 98.4%.

On the next page are plots showing our guesses of where each of the bots ended up on their 100th step. We added some randomness plotting it within a square so that the density of points in each square could be more easily seen.







Our transition matrices for each of the bots are included in the zip file as bot1.csv, bot2.csv, and bot3.csv.

## Conclusion

Our first approach based on raw distance alone required guessing randomly between equidistant squares. This disregarded the fact that certain bots tend to visit certain squares more frequently and that the bots location in step 100 is highly correlated with its location in step 99. Our next few approaches built up to our final approach by showing us what did, and did not work.

Our ultimate model fit the problem better because it accounts for the distinct probability distributions of the three bots, incorporates the aspect of hidden states, and learns the bots' behavior from the observation matrix. The parameters were chosen in a principled fashion because we provided an initialized matrix but the HMM algorithm learned the transition and emission parameters based on the observations.

In terms of what we learned about the robots' paths themselves, it turns out that there are four squares that none of the robots ever visit: squares 10, 15, 20, and 25. If we had known this, we could have adjusted the emission matrices to have just 14 rows instead of 18. However, this would only be a size improvement and would not actually improve performance because in the current model the columns corresponding to those four squares are ignored by the algorithm.