Competition 2 Write-Up
CS 4786

Saarthak Chandra (sc2776)
Shweta Shrivastava (ss3646)
Vikas Nelamangala (vpn6)
Jae Young Chang (jc2379)

December 8, 2016

Contents

1 Introduction

For this competition, we were given the two following data sets:

1. The 3000 **Observations** of the bots, where each observation is a euclidean distance of the robot from a corner of the room. The exact location of the robot is not given, but only the distance.

2. The **Label**ed example of the actual location of the bot on step 100 for the first 200 runs.

The locations of the bot is given by the index of a square on a grid, as shown in Figure 1. A mapping from the Euclidean distance to the square number is shown in Table 1.

Distance	Squares
1.0	1
1.414	6
2.0	2
2.236	7,11
2.828	12
3.0	3
3.162	8,16
3.610	13,17
4.0	4
4.123	9,21
4.242	18
4.472	14,22
5.0	5,19,23
5.099	10
5.385	15
5.657	24
5.831	20
6.403	25

Table 1: A mapping from the Euclidean distance to square numbers.

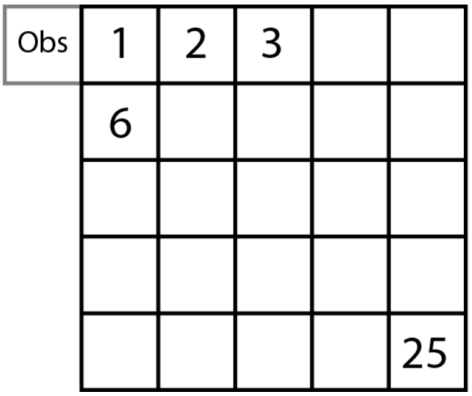


Figure 1: The grid of the robot. Observations of the sound is measured at block (0,0). Each square in the 5x5 grid is labeled by a number from 1 to 25. (Some squares have been filled in)

The goal of this competition is to find the final location of the bot after 100 observations of the distance to the robot. Furthermore, instead of there being one robot with just one behaviors, there are actually 3 robots selected by a fixed probability distribution in the beginning of the run, each coming with different behaviors. All of these parameters must be taken into consideration in order to properly determine the final locations of the robots.

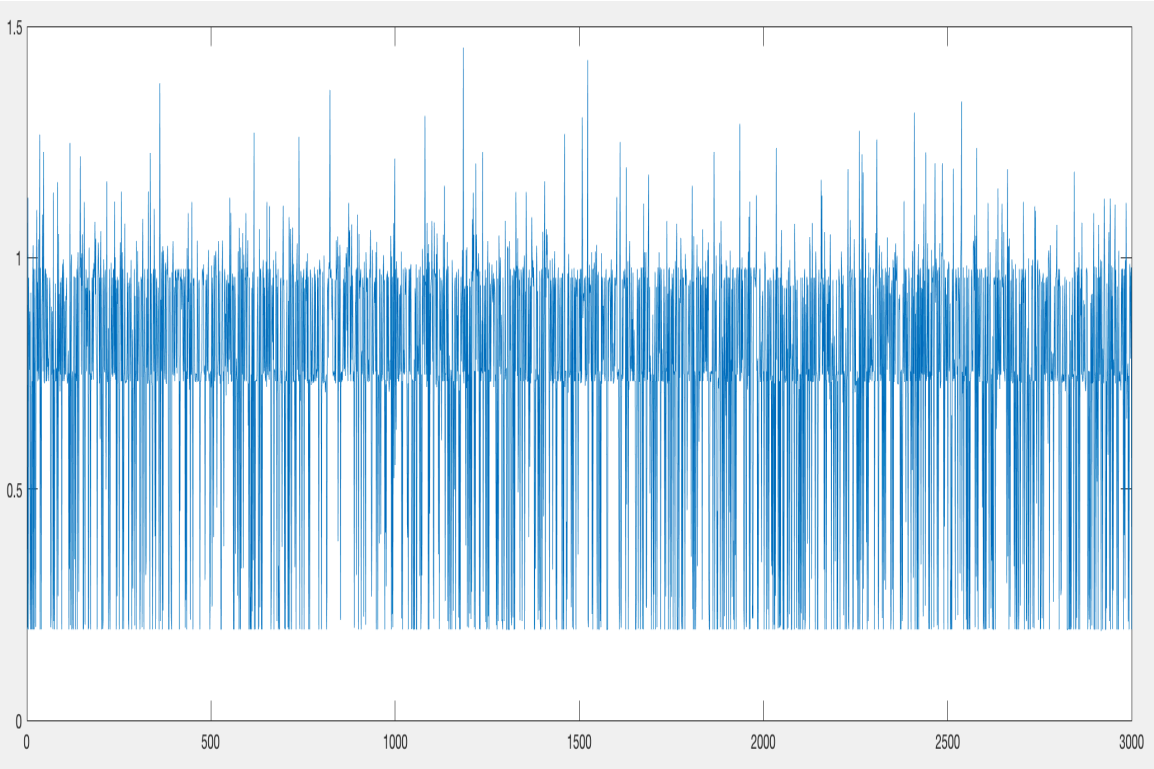
2 Understanding the Data

1. Observations.csv -

   (a) As stated in the problem statement, there were 3 bots given to us. On a first glance at the data, we noticed that there were a lot of 'repeated' euclidean distances.

   (b) Out of 18 possible Euclidean distances, the entire file had 14 values. Thus, we concluded that some of the squares had almost 0 probability of bot localization.

   (c) We did an initial plot to estimate the distribution/ratio of the labels among the 3 bots, i.e., how many bots are there in each of the 3 types. For this, we did a mere plotting of variances of the 3000 rows of the file, and plotted them as shown below.



Plot of variances of 3000 bot observations.
Aim : To see the spread, and get an estimate of the 'kind' of bots.

This shows 3 "bands", showing us there are 3 bots with variances roughly 0.3, 0.7 and 0.9. The ratio of the number of bots we obtained was roughly 1:2:2.

2. Labels.csv - Explained in section 3.3 below

3 Main Algorithm

The main algorithm is as follows:

1. Divide the 200 observations (for which we know the labels) into 3 groups corresponding to each bot.

2. Train a model for each bot using the observations obtained in previous step.

   (a) The input to the hmmtrain command was a transition matrix $[Trans]_i j = \frac{1}{25}, \forall (i,j) in (1,25)$.

   (b) The emission matrix given was a custom emission matrix - 25*14 matrix, using the Table 1, where the rows correspond to the 25 squares, and the 14 columns correspond to the euclidean distances given in the observations.csv file. (Further explanation in Section 2b)

This training provides us our initial estimates for Transmission and Emission Matrices.

3. Divide the 3000 observations into 3 groups corresponding to each bot.
4. Train a model for each bot using the groups obtained in step3. We use the Transmission and Emission matrices we obtained in Step 2 as our initial estimates for hmmtrain command.
5. Once the model is generated, use hmmdecode to generate labels for each bot.
6. Compare the labels generated with 200 sample labels and fine tune the parameters for accuracy.

This algorithm ensures that the probability distribution that specifies one of the three robots is taken care of in the beginning.

3.1 Categorizing data of 3 bots

First, we needed to divide the 3000 observations into 3 groups, one for each type of robot. To do that, we first had to generate a transition matrix for the 100 movements for each robot. In the data, there were 18 unique distances that were measured by the robots. The algorithm for generating this transition matrix was simple:

1. Create an empty 18 by 18 matrix, where an element (i,j) in the matrix represents the probability of the robot transitioning from the i'th distance to the j'th distance.
2. Traversing through the 100 observations, for every adjacent distance in each run, increment that element in the matrix.
3. Normalize the matrix.

In the end, we end up with a heat map that looks like Figure 2. The bright spots on the i'th row and the j'th column indicate a high probability of moving from the i'th distance to the j'th distance.

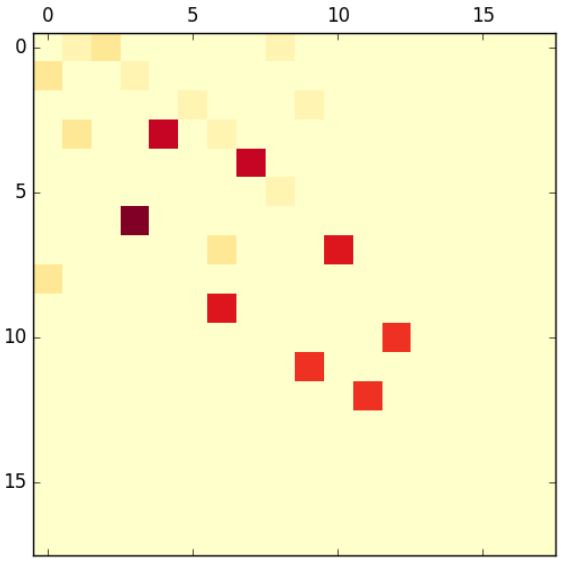


Figure 2: Heat map of the transition matrix for a robot.

By observing these transition matrices, we were visually able to categorize each robot run into three groups: One with 3 hot spots in the matrix, one with 5 hot spots, and one with 8 hot spots.

In terms of the behavior of the robot, a transition matrix with 3 bright spots indicated that the robot was perpetually cycling between three different distances on the map. A robot with more hot spots indicates a more varied movement.

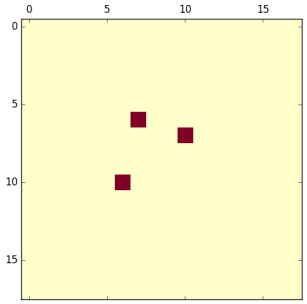


Figure 3: Group 1.

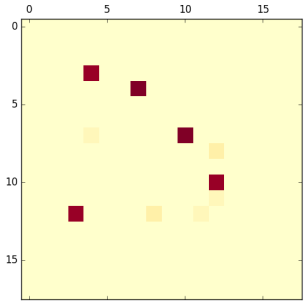


Figure 4: Group 2.

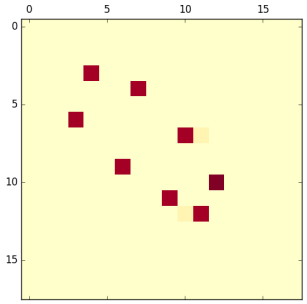


Figure 5: Group 3.

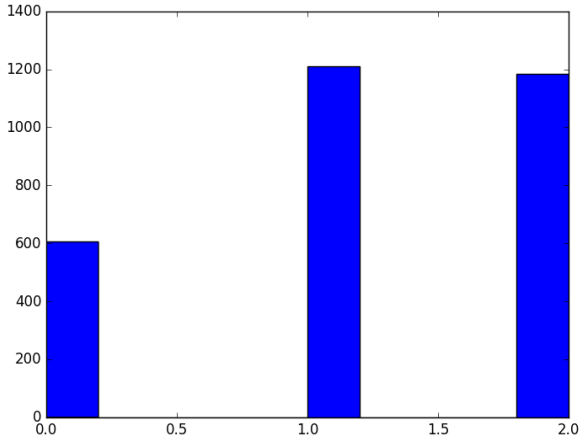


Figure 6: Probability distribution of the 3 types of robots.

Categorizing these three different groups together, we measured the probability distributions of each of the robot groups, as shown in Figure 6.

We found that Group 1 was half as likely to be picked than Groups 2 and 3.

This method was highly advantageous since we don't need information regarding the actual positions of the robots – only the distances. Therefore, we can use the 2800 unlabeled observations to generate an averaged transition matrix for each of the 3 types, making our model more robust with more data. With the group of each observations identified, we can also use the unlabeled observations to better train our model.

After the split, we obtained

- 602 bots in model 1 : Path : 3.6056 - 4.2426 - 3.1623
- 1214 bots in model 2 : Path : 3.6056 - 2.8284 - 3.6056 - 4.2426 - 5
- 1184 bots in model 3 : Path : 4.4721 - 4.1231 - 3.1623 - 2.2361 - 2.8284 3.6056 - 4.2426 - 5
  The bots also had a small probability of jumping to random white squares, with bot1 having the least probability of breaking the path, and bot3 having the highest among the three.

3.2 HMM training using Baum Welch Algorithm

- Our model takes as input all the 3000 rows of bot observations. As stated, we used the Baum-Welch algorithm for training the HMM model. First, as a preprocessing step, we separated the first 200 bot observations into 3 files, each containing the bot observations of only one type(based on the criteria given in the section above). Then for each of the 3 files, we trained an HMM model using

the hmmtrain command, to obtain the transition and emission probabilities of the 3 bot sequences.

[TransBOT1 , EmisBOT1] = hmmtrain(bot1,trans,emis,'SYMBOLS',symbolsInData,'Verbose',true), where symbolsInData is an array of the 14 symbols that the HMM emits.

This gave us the initial transition and emission matrices, for the 3 bot sequences. Next we divided the entire sequence of 300 observations and split them into 3 groups. For each of the 3 groups, we used hmmtrain, as shown above with the TransBOT1 as the starting transition matrix, and EmisBOT1 as the starting emission matrix. This helped our training algorithm to converge quickly, taking us only 3 iterations to do so.

Once we did this,using the hmmdecode command in Matlab, we found the posterior state probabilities of the 3 bot sequences, looping over each of the rows of the bot sequence. In each iteration, for a particular bot run, we found the square with the maximum posterior probability, only for the 100th bot observation, and assigned that as the label for the given bot.

Once we generated the labels for all the 3000 bot runs, we used the validation set (the 200 labels given to us), and checked for the accuracy of our model.

Reason for using Baum Welch Algorithm : Given our set of observations, we wanted to train our model and Baum-Welch algorithm is a good fit as it is used for training HMM's using the EM variant of forward-backward algorithm.

- Number of states in our model : 14. This was obtained once we found the number of unique euclidean distances in the observations.csv file, as stated in Section 2 above.
- The starting Emission matrix was a 25*14 matrix , which had a 1 corresponding to the corresponding squares which we calculated as shown in the 1. For ex : the 1^{st} column of the matrix had a one against the first cell, as bots at distance 1 could only emit symbol/label 1, and so on. $[Emission]_{ij}$ = 1, if square i can be emitted by the euclidean distance j.

3.3 Testing with Labels

Testing and adjusting our method is fairly simple, since the first 200 labels of the 3 bots were given to us – a sizable number of data. After running our main method, calibration of our parameters was done by seeing the success rate of the calculated square number with the given labels.

- 1. Supervision: The labels given to us served a very important metric to test our model after training. We were given 200 labels, which had a mix of labels for all the 3 bots. After classification of the bots into 3 separate patterns (3 bots with different patterns), we were able to judge which squares the 3 bots would land up in, and map the euclidean distance in the 100^{th} column to the square number.
   For example : Bot 1 $Pattern$: "$3.6056 - 4.2426 - 3.1623$"
   We found that row1 in the observations.csv file corresponds to bot1, and the 100^{th} euclidean distance given was **3.6056, which also happened to be the starting point** for the bot/s given to us.
   This label was mapped to square 13(and not square 17), thus telling us that the bots indeed start at the center of the 25*25 grid : a piece of information missing in the instructions.
   2. Once we trained our models, the labels helped us minimize Kaggle submissions by validating our model outputs. We would test our model output by matching the labels that our model produced for the first 200 rows of the observations.csv file against the labels given to us as a part of the competition. Only if the match rate was high, would we submit the output file to Kaggle.
   3. We noticed that if the accuracy of our model on the 200 labels given was 80%, Kaggle would score it at an accuracy of nearly 75% hence showing that it was a good '**validation set**' for our output data.
- **Unlabeled examples**: The unlabeled examples were used in conjunction with the labeled examples when we trained our HMM. However, the training was done after we split the 3000 bot

observations into 3 separate bot sequences. Thus, the sequences of each of the 3 bots had a mix of the labeled and unlabeled examples.

4 Failed Attempts

Here, we outline some failed attempts during this competition. These failures were used to better adjust our method to yield a model that would achieve better results.

4.1 Particle Filtering with Distance Information

One method that we tried was by applying particle filtering with only the transition probabilities with distances *only*. The algorithm was an adapted version of the Particle Filter Method, and it is as follows:

1. Identify the Group of the robot using the process above, and measure the overall transition matrix of the distances.
2. Generate N samples positioned at the center of the board (square 13), and assign them in X. Also, generate a N sized array, W, that stores the weighting of each sample in X.
3. For N elements,
   (a) calculate an index j from the weights W, and calculate the next square value, X_{new} by using X_j and the transition matrix.
   (b) If X_{new} agrees with the distance of the next observation, set $W[j]$ as 1. If not, set it as 0.
4. Apply the previous step until all of the time steps are exhausted. At the end, you will end up with the final position of the robot.

With Kaggle, this submission scored **0.70571**.

The weakness of this method is that it is a model that only takes into consideration the distance of the robots, and the actual positions of the robots were not properly used to run the particle filter. A correct model of this system would allow the actual positions of the robot as a hidden variable in the Hidden Markov Model, and this naive model didn't really use a probabilistic model that can be trained by using the 100 observations.

4.2 HMM training using Baum-Welch Algorithm

1. Random Transition Matrix, Random Emission Matrix Having failed in the previous attempt above,we moved to training an HMM, using the Baum-Welch Algorithm.
   (a) We split the bot observations into 3 separate groups, similar to the method above.
   (b) The symbols emitted by our HMM are [1,1.4142,2,2.2361,2.8284,3,3.1623,3.6056,4,4.1231,4.2426, 4.4721,5,5.6569]. These are the 14 unique euclidean values in the observations.csv file.
   (c) Use a random emission matrix of size 25*14.
   (d) Use a random transition matrix , $Transition_{ij}$ = rand(25,25).
   (e) Use hmmtrain in Matlab, to train the HMM. The algorithm used is the default forward-backward Baum-Welch algorithm.
   (f) Use hmmdecode, in Matlab iteratively for each row, and find out the square emitted by the bot ,with maximum probability, in the 100^{th} column.
   (g) After having done the above, for all the 3 bot observations separately, we combine the obtained results into on final file with 3000 labels for all the bots runs.

   One observation we made here is that our training algorithm - Baum Welch did not converge, despite the maximum iterations being as high as 200. The relative difference in log-likelihood did reach an order 10^{-8}, which is acceptable, however the transition matrix differences did not reach the desired 1.0 * 10^{-6} level, and neither did the emission matrix reach the desired threshold. We also noticed a continual oscillation in the values of the transition matrix as well as the emission matrix.

We **did not submit this to Kaggle**, as we checked our accuracy with the lables.csv given. This was an example, where we used the labels as our validation set and rejected this model, which had flawed parameters.

The weakness in this method was that we chose a random transition and emission matrix, which did not converge. Hence, we believed that if allowed to converge, the HMM could predict the bot locations more precisely.

We also noticed , that within the labels file, the labels of the first bot as per the sequence 3.1 matched completely, however there were errors in the labels of the other 2 bot sequences.

2. Random Transition Matrix, Custom Emission Matrix

   (a) This method had the exact same procedure as the method stated above, with one modification.

   (b) Here, we used a custom emission matrix, of size 25 * 14, since we found only 14 unique euclidean distances stated above. The entry $Emission_{ij} = 1$ iff as per Table 1 , the $Distance_i$ emits j. Ex: Distance 1 is emitted by square 1 only, hence $Emission_{11}$=1, and $Emission_{1j}$=0, where j≠1.

   (c) We tested the accuracy of our model by comparing the first 200 lables in our output file with the labels given to us, and saw an accuracy of 80%.

   With Kaggle, this submission scored an accuracy of 80%. The weakness in this method was that we chose a random transition matrix, which did not converge.The transition matrix did converge on this run, hence we realized that our transition matrix parameters were to be chosen carefully, as random values would not converge. Hence, we believed that if allowed to converge, the HMM could predict the bot locations more precisely.

3. Custom Transition , Custom Emission Matrix

   (a) This method is the one which led to our highest **accuracy of 98.714%**. This has been described in section 3 above.

5 Main Takeaways

If there were to be 2 main takeaways from this competition , they would be :

1. Categorizing/Splitting the data really helped us. When we want to train an HMM, and the data given is a mixture of HMM's, splitting the data into 3 separate sets of observations is a key step. The method of splitting will depend on the kind of data given, in our case, euclidean distances.

2. Convergence of EM algorithms , like the ones we used here - Baum Welch, does depend on the starting parameters (Transition/Emission matrices in our case). Cleverly choosing them leads to faster convergence and (possibly) more accurate predictions.

****