Machine Learning for Data Science - Competition 2

Antoine Klopocki, Jaydev Kshirsagar, Travis Westura, Vishisht Tiwari
Kaggle Team Name: Antravishjay

1 Introduction

In this competition we are given a robot that is moving around in $\mathbb{R}^2$. The robot makes 10,000 runs, and for 1,000 timesteps we observe the angle θ that the robot's position makes with the x-axis. But are not given the robot's precise (x, y) location, and the angle itself does not determine directly the robot's position. The true position could be anywhere along the line passing through the origin and making angle θ with the x-axis.

Our task is to determine the robot's (x, y)-position on the 1,001st timestep as accurately as possible, with accuracy determined by Root Mean Square Error. Our methods achieve an accuracy score of **0.26233**.

2 Determining the Observer's Position

The competition specification describes the robot moving in the first quadrant, and the angles given in the observations file contain the angle made between the robot's location $(x_{r,t}, y_{r,t})$ and the x-axis. However, the coordinates given in the label file contain coordinates with negative values. We'll denote to these values as $(x'_{r,t}, y'_{r,t})$. Since these values are negative, the observer of the locations is positioned in the first quadrant as well, and we call it's location (a, b). First we need to determine the coordinates of this point so that we can match up the data in the observation and label files.

Consider a labeled point $(x'_{r,t}, y'_{r,t})$. The angle θ that this point forms with the x-axis is given by

$$\tan\theta = \frac{b + y'_{r,t}}{a + x'_{r,t}}.$$

We know the value of θ from the observations file. Since there are two unknowns a and b, we pick two labeled points and solve the system of equations

$$\tan\theta_1 = \frac{b + y'_{r_1,t_1}}{a + x'_{r_1,t_1}}, \quad \tan\theta_2 = \frac{b + y'_{r_2,t_2}}{a + x'_{r_2,t_2}}.$$

We use the points from run 1 time steps 205 and 216. Solving the system of equations yields the position of the location observer as $(1.5, 1.5)$. Using this position, we consider the observation angles as being measured at $(0, 0)$.

Further, we can use the 600,000 labeled points to gain intuition about the robot's movement. We compute the average of radius of these points to $(1.5, 1.5)$ to see that this average is approximately 1. Plotting the labeled points in figure 1, we see that the robot is moving roughly in a circle of radius 1 centered at $(1.5, 1.5)$, with kinks occurring at the top, bottom, left, and right of the circle.

Every run starts the robot at $(1.5, 1.5)$. But we are not given any labels for the first 200 runs, and we see that the robot stabilizes into its standard path by the time we start obtaining labels for its location.

3 Hidden Markov Models (HMM's)

We represent this problem as a Hidden Markov Model. A Hidden Markov Model is a graphical model with two types of variables: latent variables S_t and observed variables X_t. The latent variables represent states that are not visible to the observer. Each observed variable X_t depends only on the corresponding state variable S_t. And each state variable S_t depends only on the previous state variable S_{t-1}. We depict latent variables as shaded vertices.

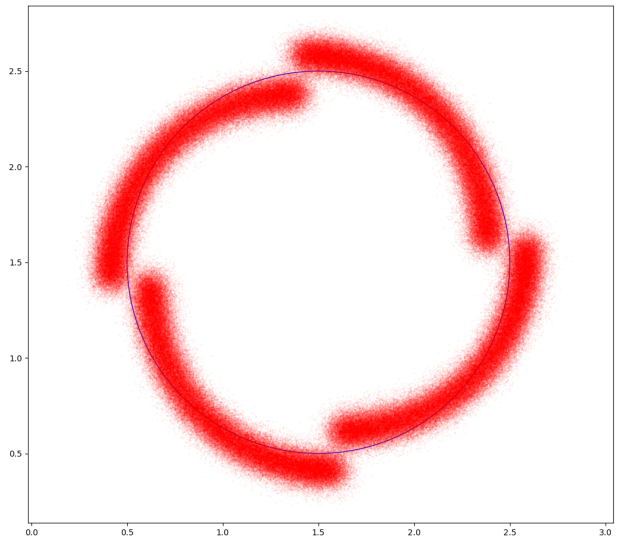


Figure 1: Locations of the robot given as labels.

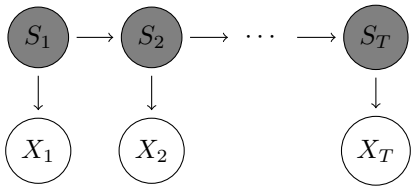


Figure 2: Hidden Markov Model

In our problem the observed values are the angles that the robot's position makes with the x-axis. The robot is moving around in the plane $\mathbb{R}^2$. This space is continuous, so our observations are angles in the continuous range $\left(0, \frac{\pi}{2}\right)$. We discretize this space in order to use a Hidden Markov Model. We divide the first quadrant into K sectors, $\left[0, \frac{1}{K}\frac{\pi}{2}\right), \left[\frac{1}{K}\frac{\pi}{2}, \frac{2}{K}\frac{\pi}{2}\right), \ldots, \left[\frac{K-1}{K}\frac{\pi}{2}, \frac{\pi}{2}\right)$, where K is a parameter that we choose. The observation, rather than being a value in $\left(0, \frac{\pi}{2}\right)$, is instead given by an integer $1, 2, \ldots, K$ representing the segment of the interval in which the angle lies. As a further refinement of this technique, we find the minimum and maximum angles that occur in the observations file, $\theta_{\min}$ and $\theta_{\max}$, and divide the shorter interval $[\theta_{\min}, \theta_{\max}]$.

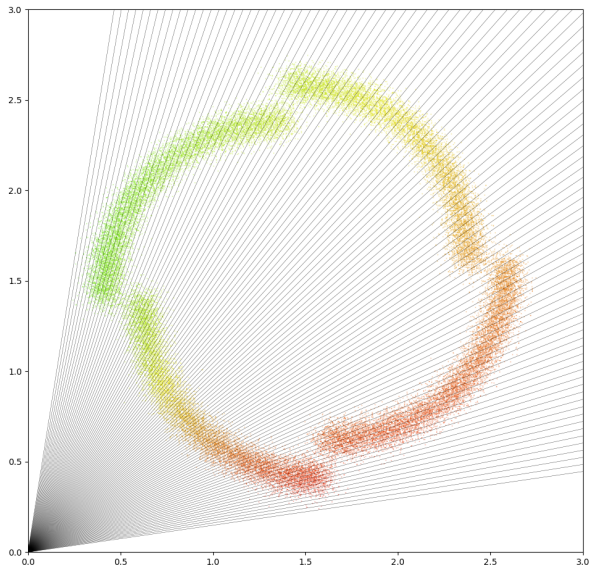


Figure 3: Labeled points colored based on the segment in which they lie with $K = 100$.

Given an observation and the corresponding state, we need to map the state to the robot's position. We outline a procedure for doing this in .

Given the observations, we need to estimate the transition matrix A and emission matrix B of the Hidden Markov Model. The transition matrix is defined by

$$a_{i,j} = \mathbf{P}(S_t = j \mid S_{t-1} = i),$$

that is, each entry $a_{i,j}$ gives the probability of being in state i given that the previous state is state j. With N states, A is an $N \times N$-matrix. The emission matrix is defined by

$$b_{i,k} = \mathbf{P}(X_t = k \mid S_t = i),$$

that is, each entry $b_{i,k}$ gives the probability of the observation k being emitted given state i. With N states and K possible observations, B is an $N \times K$-matrix. In section 4 we describe our process for estimating these matrices using the Baum Welch algorithm.

4 Baum Welch

The Baum Welch algorithm is an Expectation Maximization (EM) algorithm for learning the parameters of Hidden Markov Models. We use a forward-backwards algorithm to perform inference for the expectation step and then update the HMM parameters in the maximization step. We thereby find the maximum likelihood estimate of the parameters of the model. The algorithm takes as parameters a triple (A, B, π), where A and B are the transition and emission matrices and π is the initial state distribution, that is, the probability of the first state being state i is given by $\pi_i = \mathbf{P}(S_1 = i)$.

4.1 Explanation of Algorithm

Let N be the number of states, K be the number of observations, and T be the number of time steps.

For the forward procedure we calculate $\alpha_i(t) = \mathbf{P}(X_1 = x_1, X_2 = x_2, \ldots, X_t = x_t, S_t = i \mid A, B, \pi)$, which is the probability of obtaining observations $y_1, y_2, \ldots, y_t$ and being in state i at time t. We recursively compute

$$\alpha_i(t) := \pi_i b_{i,x_1},$$
$$\alpha_i(t+1) := b_{i,x_{t+1}} \sum_{j=1}^{N} \alpha_j(t) a_{j,i}.$$

For the backward procedure we calculate $\beta_i(t) = \mathbf{P}(X_{t+1} = x_{t+1}, \ldots, X_T = x_T \mid S_t = i, A, B, \pi)$, the probability of the observations $x_{t+1}, \ldots, x_T$ occurring given the tth state is state i. Again we compute recursively to set

$$\beta_i(T) := 1,$$
$$\beta_t(t) := \sum_{j=1}^{N} \beta_j(t+1) a_{i,j} b_{j,x_{t+1}}.$$

Before performing the updates, we first calculate two temporary variables. We define $\gamma_i(t)$ to be the probability of being in state i at time t given a set of observations $X = (X_1 = x_2, \ldots, X_T = x_T)$. Applying Bayes's theorem we have

$$\gamma_i(t) := \mathbf{P}(S_t = i \mid X, A, B, \pi) = \frac{\mathbf{P}(S_t = i, X \mid A, B, \pi)}{\mathbf{P}(X \mid A, B, \pi)} = \frac{\alpha_i(t)\beta_i(t)}{\sum_{j=1}^{N} \alpha_j(t)\beta_j(t)}.$$

Next we define $\xi_{i,j}(t)$ to be the probability of being in state i at time t and state j at time $t+1$ given a sequence of observations X with parameters A, B, and π.

$$\xi_{i,j}(t) := \mathbf{P}(S_t = i, S_{t+1} = j \mid X, A, B, \pi) = \frac{\mathbf{P}(S_t = i, S_{t+1} = j, X \mid A, B, \pi)}{\mathbf{P}(X \mid A, B, \pi)},$$
$$= \frac{\alpha_t(t) a_{i,j} \beta_j(t+1) b_{j,t+1}}{\sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i(t) a_{i,j} \beta_j(t+1) b_{j,t+1}}.$$

We use these temporary variables to update the parameters of our Hidden Markov Model. The vector π is updated so each entry π_i is the probability of being in state i at the first time $t = 1$.

$$\pi_i := \gamma_i(1)$$

The values $a_{i,j}$ are updated to be the expected number of transitions from state i to j divided the total number of transitions from i (including from i back to itself).

$$a_{i,j} := \frac{\sum_{t=1}^{T-1} \xi_{i,j}(t)}{\sum_{t=1}^{T-1} \gamma_i(t)}$$

And finally the values $b_{i,k}$ is set to the expected number of times the observation k is emitted from state i over the total number of times state i occurs.

$$b_{i,k} := \frac{\sum_{t=1}^{T} \mathbf{1}_{X_t=k} \gamma_i(t)}{\sum_{t=1}^{T} \gamma_i(t)}$$

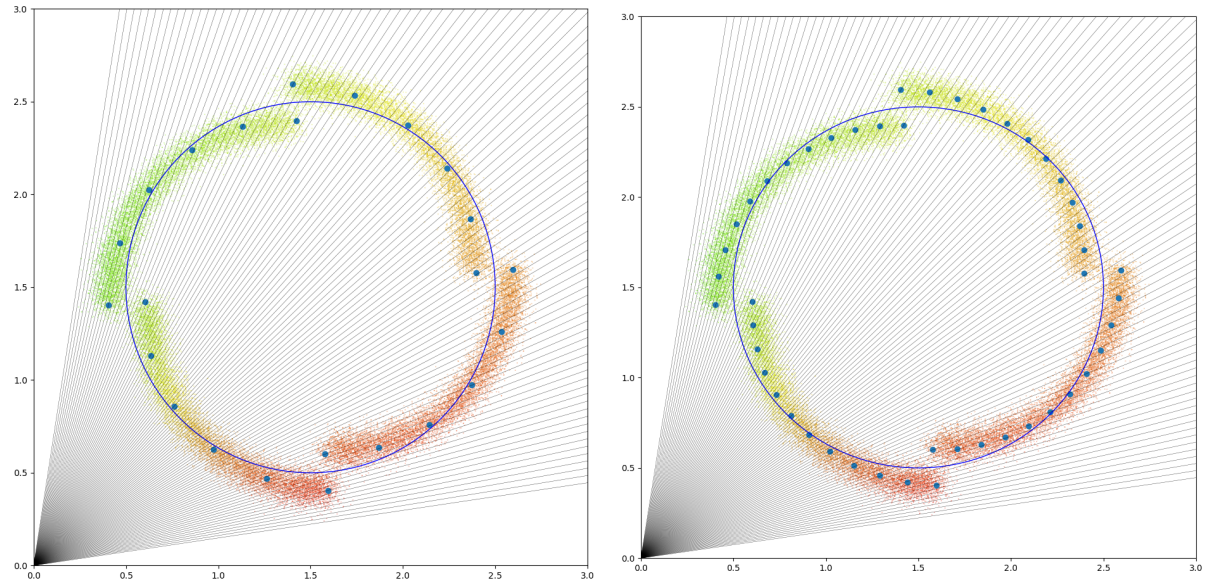
4.2 Numerical and Running Time Issues

4.3 Choice of Parameters

One way to initialize the parameters A, B, and π of the Baum Welch algorithm is to do so randomly. However, as noted in section 4.2, doing so means the algorithm takes a long time to converge. By choosing initialization parameters close to what we expect the algorithm's output to be, we can decrease the number of steps that the algorithm requires to converge.

To initialize the transition matrix, we take advantage of the fact that the robot is moving rather slowly around the circle. That is, in only one step, the robot is likely either to stay in the same state or move to a state with a location close to its previous state's location. It will not make large transitions, e.g. from the bottom to the top of the circle, in a single step. Thus it makes sense to initialize many values of the transition matrix to be 0, as there are only a small number of states to which the robot may transition from any given state.

Further, the Baum Welch converges to a local maximum. And by changing the initial parameters we can change the local maximum to which the algorithm converges. Thus we choose to initialize our transition matrix so that the states correspond to sections of the robot's path. In figure 4 we show how we pick locations to guide our construction of the transition matrix. We pick points along the robot's path and



(a) N = 24.

(b) N = 48.

Figure 4: Initial state positions for different numbers of states.

choose transition probabilities either that the robot stays in the same state or moves to the next state. For

example, for 4 states, our transition matrix is

$$\begin{bmatrix} 0.75 & 0.25 & 0 & 0 \\ 0 & 0.75 & 0.25 & 0 \\ 0 & 0 & 0.75 & 0.25 \\ 0.25 & 0 & 0 & 0.75 \end{bmatrix}$$

Further, we decide not to use any of the first 200 time steps in our model. The robot starts at $(1.5, 1.5)$, but by the time we start receiving labels at time 201, the robot has already reached its general orbit around its path. We don't have any labels for the first 200 time steps, so using these values ends up making it more difficult for our algorithms to estimate the behavior of the robot at the tail of its runs, which is the behavior in which we are most interested to predict the 1,001st location. When we try using these initial observations, our Baum Welch algorithm outputs a state to keep track of the values in the middle. Figure 5 shows how these initial observations result in points being assigned to a state that does not correspond to a location in the tail of the robot's path.

5 Viterbi

6 Mapping States to Locations

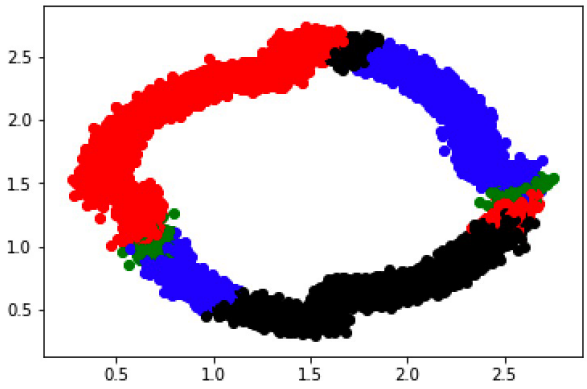


Figure 5: The green points are assigned to a state representing the middle of the circle through which the robot finishes moving by time 200.