Matthew Leifer
matthewleifer@college.harvard.edu
CS181-S17

Assignment #5
Due: 5:00pm April 14, 2016

Collaborators: Limor Gultchin

Homework 5: Graphical Models and MDPs

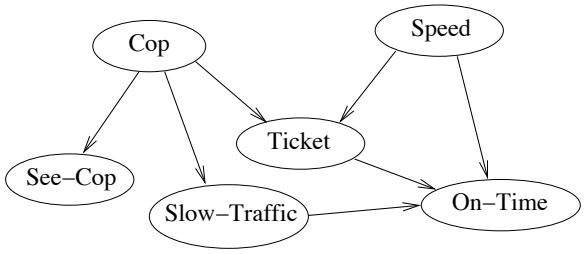
Introduction

There is a mathematical component and a programming component to this homework. Please submit your PDF and Python files to Canvas, and push all of your work to your GitHub repository. If a question requires you to make any plots, please include those in the writeup.

Bayesian Networks [7 pts]

Problem 1

In this problem we explore the conditional independence properties of a Bayesian Network. Consider the following Bayesian network representing an Uber driver taking a passenger to the airport. Each random variable is binary (true/false).



The random variables are:

- Cop: is there a State trooper present on the highway?
- See-Cop: has the driver seen the trooper?
- Slow-Traffic: is the traffic moving slowly?
- Ticket: does the driver get a speeding ticket?
- Speed: does the passenger demand that the driver speeds?
- On-Time: does the passenger get to the airport on time?

For each of these questions, use the method of d-separation, and show your working.

1. Is $I(\mathsf{Cop}, \mathsf{Speed})$? If NO, give intuition for why.
2. Is $I(\mathsf{Cop}, \mathsf{Speed} \mid \mathsf{Ticket})$? If NO, give intuition for why.
3. Is $I(\mathsf{See\text{-}Cop}, \mathsf{On\text{-}Time})$? If NO, give intuition for why.
4. Is $I(\mathsf{Cop}, \mathsf{On\text{-}Time} \mid \mathsf{Slow\text{-}Traffic})$? If NO, give intuition for why.
5. Modify the network to model the setting where a trooper must still be present for a driver to get a ticket, but tickets are delivered electronically and the driver does not need to stop if caught.
6. For this modified network, what are two random variables, X, such that if either was known we would have $I(\mathsf{See\text{-}cop}, \mathsf{On\text{-}Time} \mid X)$? Give intuition for your answer.

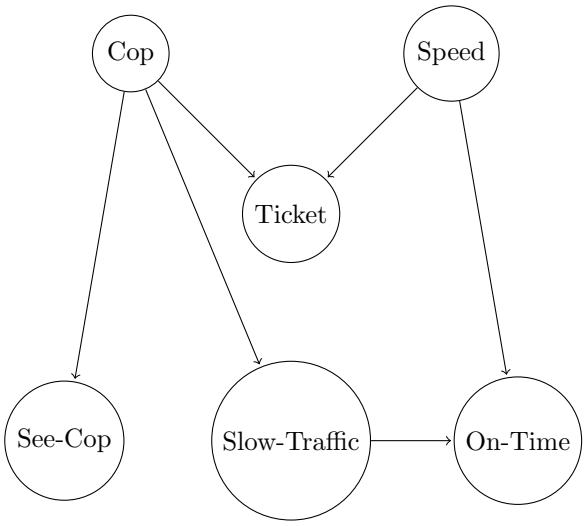
Solution

1. Yes because the path through tickets is blocked because neither ticket nor its descendents are in the evidence and a potential path from speed through on-time is also blocked becuase on-time has converging arrows.
2. No this is not true becuase the ticket node has converging arrows and is in the evidence. Intuitively, if we know that the driver got a ticket, then he was speeding and there was a cop and so given the ticket status, cop and speed are not conditionally independent.
3. No the path through see-cop, cop, slow-traffic or ticket, to on-time is unblocked. If the driver sees

a cop, then there's a cop which could mean that there is either slow traffic (or if he's speeding the possibility of a ticket) either of which would affect whether he's on time.

4. No they're not conditionally independent becuase the path from cop to ticket to on-time is unblocked. Regardless of what the status of slow-traffic is, a cop (or lack thereof) could affect whether the driver gets a ticket and thus will affect if he's on time. If we knew slow-traffic and ticket, then they would be independent.

5. If tickets are delivered electronically, they will not affect the on-time status and so we should remove the arrow from Ticket to On-Time and leave everything else unchanged.



6. If we knew either Cop or Slow-Traffic, then See-cop and on-time would be conditionally independent because the path between them would be blocked. Intuitively if we knew there was a cop, then whether or not we knew we were on time would not affect if we saw the cop or not because seeing the cop doesn't affect our speed. Alternatively if we knew that there were slow traffic and then we observe that there is a cop, that observation can't change whether or not we're on time because there's already slow traffic and and our speed won't be affected.

Hidden Markov Models [5 pts]

Problem 2

In this problem, you will derive the expressions for using the α- and β- values computed in the forward-backward algorithm on HMMs for the inference tasks of predicting the next output $\mathbf{x}_{n+1}$ in a sequence, and calculating the probability of a sequence of states $\mathbf{s}_t, \mathbf{s}_{t+1}$.

Recall that for output sequence $\mathbf{x} = (\mathbf{x}_1, \ldots, \mathbf{x}_n)$, the α-values are defined, for all t, and all $\mathbf{s}_t \in \{S_k\}_{k=1}^c$, as:

$$\alpha_t(\mathbf{s}_t) = p(\mathbf{x}_1, \ldots, \mathbf{x}_t, \mathbf{s}_t).$$

Similarly, the β-values are defined, for all t, and all $\mathbf{s}_t \in \{S_k\}_{k=1}^c$ as:

$$\beta_t(\mathbf{s}_t) = \begin{cases} p(\mathbf{x}_{t+1}, \ldots, \mathbf{x}_n \,|\, \mathbf{s}_t) & \text{, if } 1 \leq t < n \\ 1 & \text{otherwise.} \end{cases}$$

You will also find it useful to recall that, for all t, and all $\mathbf{s}_t \in \{S_k\}_{k=1}^c$,

$$\alpha_t(\mathbf{s}_t)\beta_t(\mathbf{s}_t) = p(\mathbf{x}_1, \ldots, \mathbf{x}_n, \mathbf{s}_t).$$

1. Show that
$$p(\mathbf{x}_{n+1} \,|\, \mathbf{x}_1, \ldots, \mathbf{x}_n) \propto \sum_{\mathbf{s}_n, \mathbf{s}_{n+1}} \alpha_n(\mathbf{s}_n) p(\mathbf{s}_{n+1} \,|\, \mathbf{s}_n) p(\mathbf{x}_{n+1} \,|\, \mathbf{s}_{n+1})$$

2. Show that
$$p(\mathbf{s}_t, \mathbf{s}_{t+1} \,|\, \mathbf{x}_1, \ldots, \mathbf{x}_n) \propto \alpha_t(\mathbf{s}_t) p(\mathbf{s}_{t+1} \,|\, \mathbf{s}_t) p(\mathbf{x}_{t+1} \,|\, \mathbf{s}_{t+1}) \beta_{t+1}(\mathbf{s}_{t+1})$$

Solution

1.
$$\begin{aligned} & p(\mathbf{x}_{n+1} \,|\, \mathbf{x}_1, \ldots, \mathbf{x}_n) \\ &= \sum_{\mathbf{s}_{n+1}} p(\mathbf{x}_{n+1}, \mathbf{s}_{n+1} \,|\, \mathbf{x}_1, ..., \mathbf{x}_n) \\ &= \sum_{\mathbf{s}_{n+1}} p(\mathbf{x}_{n+1} \,|\, \mathbf{s}_{n+1}) p(\mathbf{s}_{n+1} \,|\, \mathbf{x}_1, ..., \mathbf{x}_n) \\ &= \sum_{\mathbf{s}_{n+1}} p(\mathbf{x}_{n+1} \,|\, \mathbf{s}_{n+1}) \sum_{\mathbf{s}_n} p(\mathbf{s}_{n+1}, \mathbf{s}_n \,|\, \mathbf{x}_1, ..., \mathbf{x}_n) \\ &= \sum_{\mathbf{s}_{n+1}, \mathbf{s}_n} p(\mathbf{x}_{n+1} \,|\, \mathbf{s}_{n+1}) p(\mathbf{s}_{n+1} \,|\, \mathbf{s}_n) p(\mathbf{s}_n \,|\, \mathbf{x}_1, ..., \mathbf{x}_n) \\ &\propto \sum_{\mathbf{s}_{n+1}, \mathbf{s}_n} p(\mathbf{x}_{n+1} \,|\, \mathbf{s}_{n+1}) p(\mathbf{s}_{n+1} \,|\, \mathbf{s}_n) \alpha_n(\mathbf{s}_n) \end{aligned}$$

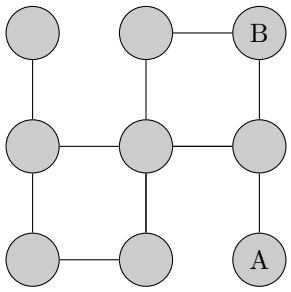
2.
$$\begin{aligned} & p(\mathbf{s}_t, \mathbf{s}_{t+1} \,|\, \mathbf{x}_1, ..., \mathbf{x}_n) \propto \\ & p(\mathbf{x}_1, ..., \mathbf{x}_n, \mathbf{s}_t, \mathbf{s}_{t+1}) \\ &= p(\mathbf{x}_1, ..., \mathbf{x}_{t+1}, \mathbf{s}_t, \mathbf{s}_{t+1}) p(\mathbf{x}_{t+2}, ..., \mathbf{x}_n \,|\, \mathbf{x}_1, ..., \mathbf{x}_{t+1}, \mathbf{s}_t, \mathbf{s}_{t+1}) \end{aligned}$$

$$
\begin{aligned}
&= p(\mathbf{x}_1, ..., \mathbf{x}_{t+1}, \mathbf{s}_t, \mathbf{s}_{t+1}) p(\mathbf{x}_{t+2}, ..., \mathbf{x}_n \mid \mathbf{s}_{t+1}) \\
&= p(\mathbf{x}_1, ..., \mathbf{x}_t, \mathbf{s}_t, \mathbf{s}_t) p(\mathbf{x}_{t+1} \mid \mathbf{x}_1, ..., \mathbf{x}_t, \mathbf{s}_t, \mathbf{s}_{t+1}) \beta_{t+1}(s_{t+1}) \\
&= p(\mathbf{x}_1, ..., \mathbf{x}_t, \mathbf{s}_t) p(\mathbf{s}_{t+1} \mid \mathbf{x}_1, ..., \mathbf{x}_t, \mathbf{s}_t) p(\mathbf{x}_{t+1} \mid \mathbf{s}_{t+1}) \beta_{t+1}(s_{t+1}) \\
&= \alpha_t(\mathbf{s}_t) p(\mathbf{s}_{t+1} \mid \mathbf{s}_t) p(\mathbf{x}_{t+1} \mid \mathbf{s}_{t+1}) \beta_{t+1}(s_{t+1})
\end{aligned}
$$

Markov Decision Processes [7 pts]

Problem 3

In this problem we will explore the calculation of the *MDP value function* V in a 2D exploration setting, without time discounting and for a finite time horizon. Consider a robot navigating the following grid:



The robot moves in exactly one direction each turn (and must move). The robot's goal is to maximize its score in the game after T steps. The score is defined in the following way:

- If the robot attempts to move off the grid, then the robot loses a point (-1) and stays where it is.
- If the robot moves onto node A, it receives 10 points, and if it moves onto node B, it receives 5 points. Otherwise, it receives 0 points.

1. Model this as a Markov decision process: define the states S, actions A, reward function $r : S \times A \mapsto \mathbb{R}$, and transition model $p(s' \mid s, a)$ for $s', s \in S$ and $a \in A$.
2. Consider a *random policy* π, where in each state the robot moves uniformly at randomly in any of its available directions (including off the board). For every position on the grid calculate the value function, $V_t^\pi : S \mapsto \mathbb{R}$, under this policy, for $t = 2, 1$ steps left to go. You can find LaTeX code for the tables in the solution template. Note that you should have 2 tables, one for each time horizon.
3. Now assume that the robot plays an *optimal policy* π_t^* (for t time steps to go). Find the optimal policy in the case of a finite time horizon of $t = 1, 2$ and give the corresponding MDP value functions $V_t^* : S \mapsto \mathbb{R}$, under this optimal policy. You can indicate the optimal policy for each time horizon on the corresponding V_t^* table via arrows or words in the direction that the robot should move from that state.
4. Now consider the situation where the robot does not have complete control over its movement. In particular, when it chooses a direction, there is a 80% chance that it will go in that direction, and a 10% chance it will go in the two adjacent (90° left or 90° right) directions. Explain how this changes the elements S, A, r, and $p(s' \mid s, a)$ of the MDP model. Assume the robot uses the same policy π_t^* from the previous question (now possibly non-optimal), and write this as π_t, and tie-break in favor of N, then E, then S then W. Give the corresponding MDP value functions $V_t^\pi : S \mapsto \mathbb{R}$, for this policy in this partial control world, for $t = 2, 1$ steps left to go. Is the policy still optimal?

Solution

1. Okay so I numbered the states 1 through 9 exactly as how they appear on a telephone pad.
   So:

1 2 3
4 5 6
7 8 9
is what the numbers in all the states correspond to.
The actions we can take are {Up, Right, Down, Left} or {U, R, D, L}
The transition model in this case is pretty simple because we move deterministically (we have a perfect actuator) given a particular action. $p(s'|s,a)$ is 1 if there is a connection between s and s' in the graph above and 0 if there is no such connection. If the move would force the robot off the graph then it bounces back to its orginal state. So, for example, $p(1 \mid 1, \text{Up}) = 1$.

$r(s,a)$	U	R	D	L
1	-1	-1	0	-1
2	-1	5	0	-1
3	-1	-1	0	0
4	0	0	0	-1
5	0	0	0	0
6	5	-1	10	0
7	0	0	-1	-1
8	0	-1	-1	0
9	0	-1	-1	-1

2. Random policy value table:

s	$V_1^\pi(s)$	$V_2^\pi(s)$
1	$-\frac{3}{4}$	$-\frac{11}{8}$
2	$\frac{3}{4}$	1
3	$-\frac{1}{2}$	$\frac{5}{16}$
4	$-\frac{1}{4}$	$-\frac{5}{8}$
5	0	$\frac{7}{8}$
6	$\frac{7}{2}$	$4\frac{1}{16}$
7	$-\frac{1}{2}$	$-\frac{15}{16}$
8	$-\frac{1}{2}$	$-\frac{7}{8}$
9	$-\frac{3}{4}$	$-\frac{7}{16}$

To calculate say $V_1^\pi(1)$, I looked at all the states and rewards the robot could receive it move only 1 step from state 1. So, $V_1^\pi(1) = 0.25 \cdot -1 + 0.25 \cdot -1 + 0.25 \cdot -1 + 0.25 \cdot 0 = -0.75$. And for the two step case I did something similar. For state 6 (middle right), $V_2^\pi(6) = 0.25(0 + \cdot V_1^\pi(5)) + 0.25(-1 + V_1^\pi(6)) + 0.25(5 + V_1^\pi(3)) + 0.25(10 + V_1^\pi(9)) = 0 + 0.625 + 1.125 + 2.3125 = 4.0625$

3. Optimal policy value table:

s	$V_1^\pi(s)$	$V_2^\pi(s)$
1	0	0
2	5	5
3	0	10
4	0	0
5	0	10
6	10	10
7	0	0
8	0	0
9	0	10

Optimal policy move table (if we're in state s with 1 or 2 moves left where do we go?):

s	$V_1^\pi(s)$	$V_2^\pi(s)$
1	D	D
2	R	R
3	D;L	D
4	U;R;D	U;R;D
5	U;R;D;L	R
6	D	D
7	U;R	U;R
8	U;L	U;L
9	U	U

Here I've include all of the optimal moves (Up, down, left, or right) that could be taken at a certain state and in practice any of them could be taken to achieve the optimal payoff for the given time horizon. So if I have 1 move left and I'm in state 5 (the very middle state), I can move in any direction and get a payoff equal to 0, but if I have 2 moves left then my optimal move in state 5 is to move right.

4. So the states and the actions will remain unchanged but the transition model and the reward function will be affected because now we no longer have a perfect actuator.

s	U	R	D	L
1	-1	-0.9	-0.2	-0.9
2	-0.1	0	-0.1	-0.8
3	-0.9	-0.9	-0.1	-0.1
4	-0.1	0	-0.1	-0.8
5	0	0	0	0
6	3.9	0.7	7.9	1.5
7	-0.1	-0.1	-0.9	-0.9
8	-0.1	-0.9	-0.9	-0.1
9	-0.2	-0.9	-1	-0.9

The following table is the MDP value function as well as the corresponding moves that should be taken. We can see that using the same policy from part 3.3 is no longer optimal. To clear instances of this are the moves we make because of the tie breaking scheme for states 4 on the one step time horizon. The tie breaking scheme says that in state 4, because there's the choice of moving up, down, or right we should go right. However, in this situation, because we no longer have a perfect actuator, the value of moving up in state 4 is -0.1, but if the moved right (east) the value would be 0 because we're guaranteed to stay on the grid.

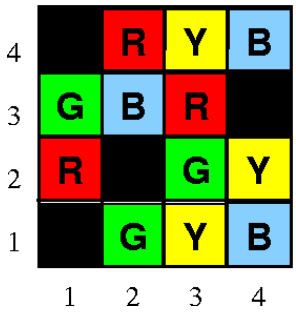
s	$V_1^\pi(s)$, a	$V_2^\pi(s)$, a
1	$-0.2, \downarrow$	$-0.32, \downarrow$
2	$3.9, \rightarrow$	$4.21, \rightarrow$
3	$-0.1, \downarrow$	$6.6, \downarrow$
4	$-0.1, \uparrow$	$-0.27, \uparrow$
5	$0, \uparrow$	$6.7, \rightarrow$
6	$7.9, \downarrow$	$8.53, \downarrow$
7	$-0.1, \uparrow$	$-0.2, \uparrow$
8	$-0.1, \uparrow$	$-0.12, \uparrow$
9	$-0.2, \uparrow$	$6.08, \uparrow$

The Viterbi Algorithm [15 pts]

Problem 4

In this problem, you will use Hidden Markov Models to reconstruct state sequences in data after learning from complete-data labeled sequences.

We consider a simple robot that moves across colored squares. At each time step, the robot attempts to move up, down, left or right, where the choice of direction is made at random. If the robot attempts to move onto a black square, or to leave the confines of its world, its action has no effect and it does not move at all. The robot can only sense the color of the square it occupies. Moreover, its sensors are only 90% accurate, meaning that 10% of the time, it perceives a random color rather than the true color of the currently occupied square. The robot begins each walk in a randomly chosen colored square.



In this problem, state refers to the location of the robot in the world in $x : y$ coordinates, and output refers to a perceived color (`r`, `g`, `b` or `y`). Thus, a typical random walk looks like this:

```
3:3 r
3:3 r
3:4 y
2:4 b
3:4 y
3:3 r
2:3 b
```

Here, the robot begins in square 3:3 perceiving red, attempts to make an illegal move (to the right), so stays in 3:3, still perceiving red. On the next step, the robot moves up to 3:4 perceiving yellow, then left to 2:4 perceiving blue (erroneously), and so on. By learning on this data, you will build an HMM model of the world. Then, given only sensor observations (i.e., a sequence of colors), the Viterbi code will re-construct an estimate of the actual path taken by the robot through its world.

The data for this problem is in `robot_no_momentum.data`, a file containing 200 training sequences (random walks) and 200 test sequences, each sequence consisting of 200 steps. We are also providing data on a variant of this problem in which the robot's actions have "momentum" meaning that, at each time step, with 85% probability, the robot continues to try to move in the direction of the last move. So, if the robot moved (successfully) to the left on the last move, then with 85% probability, it will again attempt to move left. If the robot's last action was unsuccessful, then the robot reverts to choosing an action at random. Data for this problem is in `robot_with_momentum.data`.

[Acknowledgment: thanks Rob Schapire for allowing us to use his robot dataset for this homework.]

1. **Learning a HMM model from labeled data.**

   Recall that a Hidden Markov Model is specified by three sets of probabilities: the initial probabilities of starting in each state ($\boldsymbol{\theta}$), the probabilities of transitioning between each pair of hidden states ($\mathbf{T}$), and the probabilities of each output in each state ($\boldsymbol{\pi}$). Your job will be to compute estimates of these probabilities from data. We are providing you with training data consisting of one or more sequences of state-observation pairs, i.e., sequences of the form

   $$\mathbf{s}_1, \mathbf{x}_1, \mathbf{s}_2, \mathbf{x}_2, \cdots, \mathbf{s}_n, \mathbf{x}_n$$

   For this problem, we will assume that the states are observed while training (complete data assumption). Given these sequences, you need to estimate the probabilities that define the HMM.

   Note: We will make one modification to complete-data maximum-likelihood estimation described in Lecture 17 for all parameters. Instead of estimating using MLE we will use a method known as *Laplace smoothing*, where an additional pseudo-count is added to each class. For instance, instead of using $\hat{\theta}_k = \frac{N_{1,k}}{N}$ for the estimate of the starting state, we will use $\hat{\theta}_k = \frac{N_{1,k}+1}{N+c}$, where there are c states in total. This ensures that no sequences have zero probability. (This method can also be shown to correspond to a Bayesian approach, with the pseudocount arising from a Dirichlet prior on the Categorical distribution.)

   Task: First, familiarize yourself with the provided code. Your first programming task is to fill in the `learn_from_labeled_data` function in `hmm.py`

   Testing: Check the code passes `test_learn_from_labeled_data` in `test_hmm.py`.

2. **Computing the most likely sequence of hidden states (short sequences).**

   Now that we have learned a model, we can use the Viterbi algorithm to compute the most likely sequence of hidden states given a sequence of observations.

   The second part of the dataset consists of test sequences of state-observation pairs. The Viterbi code accepts as input just the observation part of each of these sequences, and from this, will compute the most likely sequence of states to produce such an observation sequence. The state part of these sequences is provided so that you can compare the estimated state sequences generated by the algorithm to the actual state sequences. Note that these two sequences will not necessarily be identical, even for a correct implementation of the Viterbi algorithm.

   Task: Implement the Viterbi algorithm to fill in the `most_likely_states` function in `hmm.py`

   Testing: Confirm that your code passes `test_viterbi_simple_sequence` in `test_hmm.py`.

3. **Computing the most likely sequence of hidden states (long sequences).**

   In practice, the method described in the lecture notes will not actually work for longer sequences. This issue arises due to numerical underflow to to the repeated multiplication of probabilities. We can avoid this problem, by instead computing the most likely sequence in log-space, i.e.

   $$\arg\max_{\mathbf{s}_1 \ldots \mathbf{s}_n} \log p(\mathbf{s}_1, \ldots \mathbf{s}_n, \mathbf{x}_1, \ldots, \mathbf{x}_n).$$

   Mathematically, this is the same, but will avoid the numerical issues.

   Task: Change the code in `most_likely_states` to use logs of the probabilities instead of the probabilities directly.

   Testing: Check the code now also passes `test_viterbi_long_sequence` in `test_hmm.py`.

4. **Experimenting with data.**

   Finally run `viterbi.py` on the two robot data files, `robot_no_momentum.data` and `robot_with_momentum.data` and examine the results, exploring how, why and when it works. To get the the breakdown of the error, set `debug=True` in the code. Sample command-line prompt:

   ```
   $ python viterbi.py robot_small.data -v
   ```

   You should write up *briefly* what you found. Your write-up should include any observations you have made about how well HMMs work on this problem and why. Your observations should be quantitative (for instance, the number of errors was x) as well as anecdotal (situations where the approach failed).

   Although this write-up is quite open ended, **you should be sure to discuss the following**:

   - What probabilistic assumptions are we making about the nature of the data in using a hidden Markov model?
   - How well do those assumptions match the actual process generating the data?
   - And to what extent was or was not performance hurt by making such realistic or unrealistic assumptions?

Solution

By and large the Viterbi algorithm and HMM did pretty well in predicting the data. On the small data set, it incorrectly predicting 2 out of the 9 values (22% error), on the no momentum data it incorrectly predicted 7524 out of 40000 points (18.81% error), and on the robot with momentum it incorrectly predicted 5334 out of 40000 possible attempts (13.34% error). Two of the most important assumptions that we made were that the data was generated in a way that abided by the Markov property (that the next state we went to was dependent only on the state we just came from) and we assume that the data was generated in a sequential manner. The small data seems to have not does as well simply it was a small data set and there wasn't that much to train on, but it also wasn't that far off from the larger no momentum data set either. For the situation when there's no momentum the Markov property holds because we move randomly at every state, but when there is momemtum it doesn't hold because the next state we go to depends on where we currently are and also where we just came from because with 85% probability the robot will continue in the same direction which violates the assumption that we only have to care about the current state to predict the next state. Surprisngly, even though the robot with momentum data violates the Markov assumption, the HMM still performed better on that data than the robot without momentum data. One possible explanation for this is that the fact that the robot had momentum gave the data more 'structure' and so it was easier for the algorithm to learn especially since the robot is also not a perfect sensor. The momentum gave the HMM in some sense more information about how the states were laid out and how they connected with one another. In the no momentum case the robot moves randomly and also doesn't detect color perfectly and that could be harder to train. Perhaps with more data the no momentum case would have eventually out performed the momentum data. Even though we didn't have a perfect actuator (it was faulty 10% of the time), overall I think the HMM did pretty well.