Russell Kunes
russellkunes@college.harvard.edu
CS181-S17

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

Collaborators: John Doe, Fred Doe

Homework 5: Graphical Models and MDPs

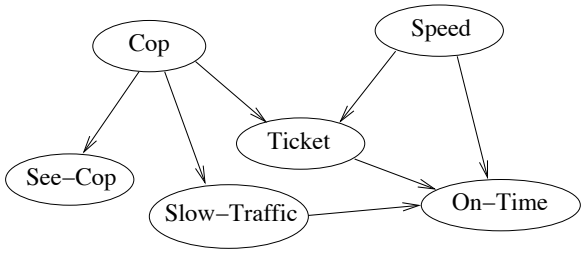
Introduction

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, there are no paths between Cop and Speed, since all of them have arrows that are converging.
2. NO, there is a path from Cop to Speed through ticket, since the arrows are converging at Ticket, which is known. Intuitively, knowing one "explains away" the other.
3. NO, there is at least one path from See-Cop to On-Time. For example, See-Cop →Cop→Slow-Traffic→On-Time. Intuitively, seeing a cop makes it more likely that the person either gets a ticket or traffic is slowed down which makes him/her less likely to be on time.
4. NO, there is a path through Ticket. Intuitively, even if you know traffic is slow, getting a ticket could make it even slower.
5. We can modify this by cutting off the edge between Ticket and On-Time. If tickets are delivered electronically, then On-Time shouldn't depend directly on Ticket.

6. X could either be Cop or Slow- Traffic. Clearly, Cop cuts off all paths between See-Cop and On-Time. Slow- Traffic also cuts off all paths since the other path that goes through Cop, Ticket, and Speed has converging arrows. Intuitively, the way that a Cop affects whether the driver is on time is by slowing down traffic; and if we know that there is a cop this contains all the information we get from the driver seeing the cop. Also, since giving tickets no longer slows down the driver, the only way the cop can slow down the driver is by slowing traffic. If we know whether traffic is slowed, we know all the information.

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

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

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. Starting from the right hand side, we have:

$$\sum_{\mathbf{s}_n, \mathbf{s}_{n+1}} \alpha_n(\mathbf{s}_n) p(\mathbf{s}_{n+1} \,|\, \mathbf{s}_n) p(\mathbf{x}_{n+1} \,|\, \mathbf{s}_{n+1})$$

$$= \sum_{\mathbf{s}_n, \mathbf{s}_{n+1}} p(\mathbf{x}_1, \ldots, \mathbf{x}_n, \mathbf{s}_n) p(\mathbf{s}_{n+1} \,|\, \mathbf{s}_n) p(\mathbf{x}_{n+1} \,|\, \mathbf{s}_{n+1})$$

We use the fact that $\mathbf{s}_{n+1}$ is conditionally independent of $\mathbf{x}_1, \ldots, \mathbf{x}_n, \mathbf{s}_n$ by the assumption of the hidden markov model:

$$= \sum_{\mathbf{s}_n, \mathbf{s}_{n+1}} p(\mathbf{x}_1, \ldots, \mathbf{x}_n, \mathbf{s}_n, \mathbf{s}_{n+1}) p(\mathbf{x}_{n+1} \,|\, \mathbf{s}_{n+1})$$

Then use the fact that x_{n+1} is conditionally independent of everything else given s_{n+1}:

$$= \sum_{\mathbf{s}_n, \mathbf{s}_{n+1}} p(\mathbf{x}_1, \ldots, \mathbf{x}_n, \mathbf{s}_n, \mathbf{s}_{n+1}, \mathbf{x}_{n+1})$$

Summing out over $\mathbf{s}_n$ and $\mathbf{s}_{n+1}$ gives:

$$= p(\mathbf{x}_1, \ldots, \mathbf{x}_n, \mathbf{x}_{n+1}) \propto p(\mathbf{x}_{n+1} \,|\, \mathbf{x}_1, \ldots, \mathbf{x}_n)$$

2. Starting from the right hand side again, we have:

$$\alpha_t(\mathbf{s}_t)p(\mathbf{s}_{t+1} \mid \mathbf{s}_t)p(\mathbf{x}_{t+1} \mid \mathbf{s}_{t+1})\beta_{t+1}(\mathbf{s}_{t+1})$$

$$= p(\mathbf{x}_1, \ldots, \mathbf{x}_t, \mathbf{s}_t)p(\mathbf{s}_{t+1} \mid \mathbf{s}_t)p(\mathbf{x}_{t+1} \mid \mathbf{s}_{t+1})p(\mathbf{x}_{t+2}, \ldots, \mathbf{x}_n \mid \mathbf{s}_{t+1})$$

Using the same conditional probability rules as above this simplifies to:

$$= p(\mathbf{x}_1, \ldots, \mathbf{x}_t, \mathbf{s}_t, \mathbf{s}_{t+1})p(\mathbf{x}_{t+1} \mid \mathbf{s}_{t+1})p(\mathbf{x}_{t+2}, \ldots, \mathbf{x}_n \mid \mathbf{s}_{t+1})$$

Using the same reasoning for $\mathbf{x}_{t+1}$:

$$= p(\mathbf{x}_1, \ldots, \mathbf{x}_t, \mathbf{s}_t, \mathbf{s}_{t+1}, \mathbf{x}_{t+1})p(\mathbf{x}_{t+2}, \ldots, \mathbf{x}_n \mid \mathbf{s}_{t+1})$$

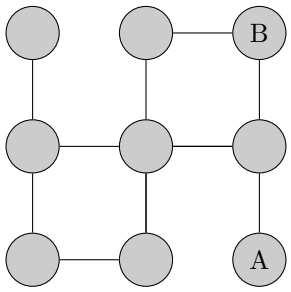
We also have that $\mathbf{x}_{t+2}, \ldots, \mathbf{x}_n$ is conditionally independent given $\mathbf{s}_{t+1}$ from $\mathbf{x}_1 \ldots, \mathbf{x}_t, \mathbf{s}_t$, so applying the same rule:

$$= p(\mathbf{x}_1, \ldots, \mathbf{x}_n, \mathbf{s}_t, \mathbf{s}_{t+1}) \propto p(\mathbf{s}_t, \mathbf{s}_{t+1} \mid \mathbf{x}_1, \ldots, \mathbf{x}_n)$$

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. The state S are the 9 positions on the grid. The set of actions is {Up,Down,Left,Right}. The reward function takes in an action from the set above and a state; if the the proposed action moves the robot off the grid, then $r(s, a) = -1$. If the proposed action moves the robot onto B, then $r(s, a) = 5$. If the proposed action moves the robot onto A, then $r(s, a) = 10$. The transition model is deterministic: if the proposed

action takes the robot off the grid, it stays with probability 1. Otherwise it moves in the proposed direction with probability 1.

2. I will label the states starting from the top left with State 1 and moving across from left to right.

s	$V_1^\pi(s)$
State 1	-3/4
State 2	3/4
State 3	-1/2
State 4	-1/4
State 5	0
State 6	7/2
State 7	-1/2
State 8	-1/2
State 9	-3/4

s	$V_2^\pi(s)$
State 1	-11/8
State 2	1
State 3	5/16
State 4	-5/8
State 5	7/8
State 6	65/16
State 7	-15/16
State 8	-7/8
State 9	-7/16

3.

s	$V_1^*(s)$	Direction
State 1	0	Down
State 2	5	Right
State 3	0	{Left,Down}
State 4	0	{Up,Right,Down}
State 5	0	Any
State 6	10	Down
State 7	0	{Right, Up}
State 8	0	{Left,Up}
State 9	0	Up

s	$V_2^*(s)$	Direction
State 1	0	Down
State 2	5	Right
State 3	10	Down
State 4	0	{Up,Right,Down}
State 5	10	Right
State 6	10	Down
State 7	0	{Right, Up}
State 8	0	{Left,Up}
State 9	10	Up

4. The states stay the same. The actions now represent the intended direction, and $p(s'|s,a)$ is no longer deterministic; s' is the intended destination 0.8 of the time, and the robot attempts to move to adjacent squares 0.2 of the time (0.1 with 90 degrees, 0.1 with -90 degrees). The reward function is now an expected

value calculation over the 3 possible directions the robot will move rather than a deterministic value; i.e. the sum over the value of the move times the probability it takes that move given the action.

s	$V_1^\pi(s)$	Direction
State 1	-0.2	Down
State 2	3.9	Right
State 3	-0.1	{Tie: Down}
State 4	-0.1	{Tie: Up}
State 5	0	Tie: Up
State 6	7.9	Down
State 7	-0.1	Tie: Up
State 8	-0.1	Tie:Up
State 9	-0.2	Up

s	$V_2^\pi(s)$	Direction
State 1	-0.32	Down
State 2	4.21	Right
State 3	6.6	Down
State 4	-0.27	{Tie: Up}
State 5	6.7	Right
State 6	8.53	Down
State 7	-0.2	{Tie: Up}
State 8	-0.12	{Tie: Up}
State 9	6.08	Up

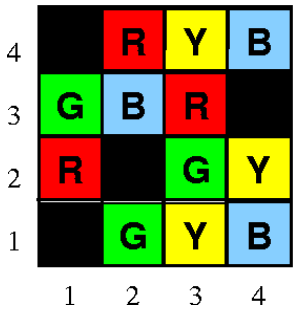
The policy is no longer optimal. At state 4, the robot moves up. The optimal move is clearly to move right since it has no chance of falling off of the grid and losing points.

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

1. First, I implemented the complete data MLE for part a with the Laplace smoothing added in. Then I implemented the forward Viterbi algorithm and took the log; and made sure it passed all the tests. Running the viterbi.py file on both the with momentum and without momentum datasets, we see that the with momentum data has an error of .133, and the without momentum data has an error of .188. Examining the mistakes manually, I noticed that for the with momentum case a lot of mistakes happen at 3:3 and 3:2. For the without momentum data, the mistakes are more evenly distributed. It is somewhat surprising the the without momentum data has a higher error, since this process is truly Markovian; the probability distribution over next states only depends on the current state. On the other hand, the with momentum data breaks our Markovian assumption since the robots moves depend on more history than just the last state on the grid. We are also assuming that the observations are conditionally independent of everything else, given the corresponding state. This holds in both robot examples, since the probability distribution over color observed does only depend on its current state on the grid. One hypothesis about why the with momentum data was easier for the Viterbi algorithm is that it is generally more predictable; i.e. there is lower variance in the distributions in transition matrix of the hidden states. Since the map that the robot moves on is quite restricted, the Markov assumption is not terrible; the last state does still contain information about the robot's momentum and still contains all of the information about possible next states (just the probabilities over these states depends on momentum) so Markov may be a good approximation. The overall decrease in variability in the robot's movement probably outweighed the distribution not really being Markov. It makes sense that 3:2 and 3:3 are where the most mistakes were made, since this is the least contrained area on the map.