CS 181 Assignment 4: Bayesian Networks and HMMs

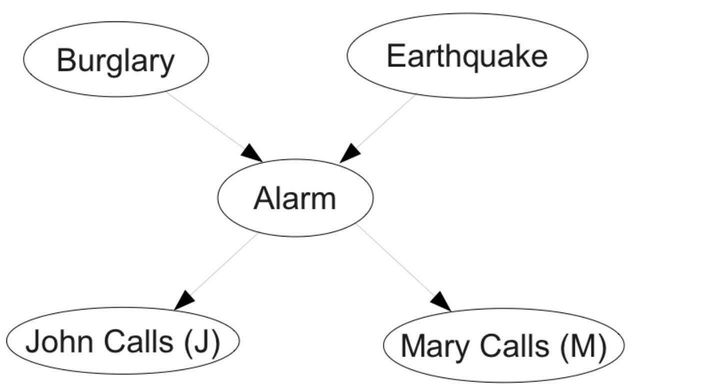
Professor David C. Parkes

Out Thursday March 24th
Due Noon Tuesday April 5th

General Instructions: This assignment consists of a theoretical component and an experimental component. You may work with one other person on this assignment (or you may work alone if you prefer). Each group should turn in one writeup.

Problem 1

[**15 Points**] Consider the Bayesian Network constructed in class that describes an alarm that may be triggered by a burglary and may be triggered by an earthquake, and phone calls from John and Mary, each of whom may call in the event of the alarm sounding.



(a) [**5 Points**] Let S be the set of all nodes in the network. Using the idea of d-separation, for each pair of nodes $U, V \in S$, list the subsets of $S \setminus \{U, V\}$ that when observed would render U, V independent. As an example, the pair of nodes B, A fail to be independent no matter what subset of $\{E, J, M\}$ are observed. Complete the following table with your findings:

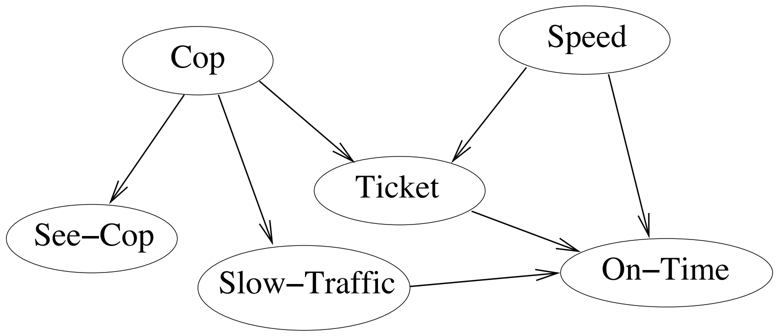
Node Pair (U,V)	$S' \in S \setminus \{U, V\}, I(U, V \mid S')$
B,A	Impossible
B,E B,J ...	

(b) **[5 Points]** Build another Bayesian Network that has the same independence properties, using variable order M, J, A, B, E (see class notes from lecture 12 for the algorithm for constructing Bayesian Networks).

(c) **[5 Points]** How many parameters are there in the Bayesian Network in (a) and the Bayesian Network in (b)? Give three reasons why the Bayesian Network in (a) is likely preferred over the Bayesian Network in (b).

Problem 2

[15 Points] The following figure shows a simple Bayesian network to help a taxi driver decide whether or not to speed. Each of the six attributes is Boolean.



The attribute Cop indicates whether or not a highway patrol cop is present on the freeway; See-Cop indicates whether or not the driver detects a cop; Slow-Traffic indicates whether or not traffic is driving slower than the speed limit; Speed indicates whether or not the driver speeds; Ticket indicates whether or not the driver gets a speeding ticket; and On-Time indicates whether or not the driver arrives on time to pick up a valuable passenger.

You are given the following information:

$$\begin{aligned}
P(\mathsf{Cop} = T) &= 0.1 \\
P(\mathsf{See\text{-}Cop} = T \mid \mathsf{Cop} = T) &= 0.6 \\
P(\mathsf{See\text{-}Cop} = T \mid \mathsf{Cop} = F) &= 0 \\
P(\mathsf{Slow\text{-}Traffic} = T \mid \mathsf{Cop} = T) &= 0.8 \\
P(\mathsf{Slow\text{-}Traffic} = T \mid \mathsf{Cop} = F) &= 0.3 \\
P(\ \mathsf{Ticket} = T \mid \mathsf{Cop}, \mathsf{Speed}, \mathsf{Slow\text{-}Traffic}) &= \begin{cases} 0.5 & \text{if } \mathsf{Cop} \text{ and } \mathsf{Speed} \text{ are } T \\ 0 & \text{otherwise} \end{cases} \\
P(\mathsf{On\text{-}Time} = T \mid \mathsf{Ticket}, \mathsf{Speed}) &= \begin{cases} 0 & \text{if } \mathsf{Ticket} \text{ is } T \\ 0.9 & \text{if } \mathsf{Ticket} \text{ is } F \text{ and } \mathsf{Speed} \text{ is } T \\ 0.3 & \text{if } \mathsf{Ticket} \text{ is } F,\ \mathsf{Speed} \text{ is } F, \text{ and } \mathsf{Slow\text{-}Traffic} \text{ is } F \\ 0.1 & \text{if } \mathsf{Ticket} \text{ is } F,\ \mathsf{Speed} \text{ is } F, \text{ and } \mathsf{Slow\text{-}Traffic} \text{ is } T \end{cases}
\end{aligned}$$

P(Speed) is not needed for the following calculations. It is the decision of the taxi driver whether or not to speed (and so either P(Speed=T) = 1 or P(Speed=T) = 0).

(a) **[10 Points]** Suppose the driver does not detect a cop and the traffic is slow. Use variable elimination to determine

  (i) **[5 Points]** the probability of being late when the driver decides to speed and when the driver decides not to speed.

  (ii) **[5 Points]** the probability of getting a ticket when the driver decides to speed and when the driver decides not to speed.

(b) **[5 Points]** Suppose that the cost of a ticket is \$150, and the cost of arriving late is \$10. What is the expected utility of speeding and not speeding? What would be the optimal decision?

Problem 3

[25 Points] In this exercise, you will attempt to use Hidden Markov Models for reconstructing a robot's sequence of locations visited in a grid world based on the color sequences it observes, after learning from fully labeled sequences.

Acknowledgement: we'd like to thank Rob Schapire for allowing us to use his robot datasets for this homework.

The support code and datasets for this problem and the next can be found here
`http://www.seas.harvard.edu/courses/cs181/docs/asst4.tar.gz`

As with the previous homework, we have included a web interface for unit tests and plots. The web interface can be run using `hw4` or `hw4.bat` (through command line). You may need to restart the web interface when making changes to files other than `hmm.py`. You may also experience the web interface timing out for more time-consuming tasks, in which you can resort to the command-line interface. In any case, the command-line interface will be useful for getting immediate feedback from print statements and for printing out model parameters and inferred sequences of states. You can see the various configuration options of of a python module by typing for example `python viterbi.py --help`. In particular, using the `-v` option allows you to print out model parameters among other things.

This problem set uses NumPy for storing and manipulating the model parameters. NumPy can be downloaded at `http://numpy.scipy.org/`
You can see examples of how to declare and access data structures (i.e. arrays) when using NumPy by looking at the `BaumWelchTest` class in `test_hmm.py`. NumPy is imported in `hmm.py` through `from numpy import *` and will be available for all python modules with the line `from hmm import *`.

The files relevant to this problem include:

- `robot_no_momentum.data:` The full dataset for the case where the robot does not have momentum. Data points are separated by ".". The full dataset is separated into training data and testing data by "..". As will be further explained later, the first column gives the x:y position of the robot, while the second column gives the color observed by the robot.
- `robot_small.data:` An extremely small subset of the no-momentum robot dataset, with only 5 data points (3 for training and 2 for testing).
- `robot_with_momentum.data:` The full dataset for the case where the robot does have momentum.
- `dataset.py:` Support code in Python for loading data and separating data into training and testing.
- `hmm.py:` Support code in Python for implementing supervised, unsupervised learning and inference in hidden Markov models.
- `viterbi.py:` Support code in Python for training an HMM from data and calling the Viterbi algorithm for finding the most likely sequence of hidden states to have generated an observation, given a learned HMM. At the command line, you can type `python viterbi.py [options] datafile.data (pass -h for more info)`, where N is the number of hidden states for the HMM.

(a) [**5 Points**] Learning a HMM model from labeled data.

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

Recall that a Hidden Markov Model is specified by three sets of probabilities: the initial probabilities of starting in each state, the probabilities of transitioning between each pair of hidden states, and the probabilities of each output in each state. 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-output pairs, i.e., sequences of the form

$$x_1, o_1, x_2, o_2, \cdots, x_n, o_n$$

For now, we assume that the state variables are available while training. Given these sequences, you need to estimate the probabilities that define the HMM. See the class notes of lecture 15 for details.

We will make one modification to the simple maximum likelihood approach presented in the notes. It is often preferable to smooth the estimates using Laplace smoothing, which pretends that you've seen everything one more time than you actually did. For example, if we were keeping count of the number of heads and the number of tails for learning the bias of the coin, this rule is equivalent to starting each of our counts at one, rather than zero. Your code should use Laplace-smoothed estimates of all the HMM parameters.

Testing: Your code should now pass the `test_simple_hmm_learning` test.

(b) **[5 Points]** Computing the most likely sequence of hidden states.

Given a learned 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 each of the provided datasets consists of test sequences of state-output pairs. The Viterbi code accepts as input just the output part of each of these sequences, and from this, will compute the most likely sequence of states to produce such an output 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.

We have provided an implementation of Viterbi in the `most_likely_states` function. It works on short sequences, but has a numerical problem: running it on long sequences of observations will result in probabilities going to zero.

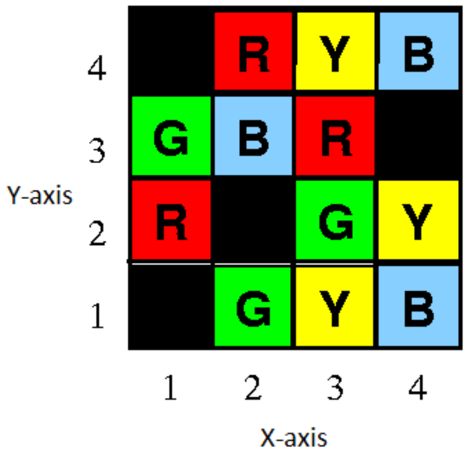
Your task: change the code to use logs of the probabilities instead of the probabilities directly. Confirm that your code still passes the `test_viterbi_simple_sequence` test, and now also passes the `test_viterbi_long_sequence` test.

(c) **[15 Points]** Experimenting with toy-robot data.

Now that you have written code for learning from fully observed training data, and a working Viterbi alogrithm, we want to infer the sequence of locations visited by a robot based on its sensor readings. We will try to use HMMs to solve this problem. First, let's find out more about the robot and the dataset.

The robot is wandering through the following small world:

The robot can only occupy the colored squares. At each time step, the robot attempts to move up, down, left or right, where the choice of direction is made uniformly 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. However, 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 the following. The first column gives the robot's location, and the gives the color perceived by the robot.

```
3:3 r
3:3 r
3:4 y
2:4 b
3:4 y
3:3 r
2:3 b
1:3 g
2:3 b
2:4 r
3:4 y
4:4 y
```

In this sequence, 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 running your learning program on this problem, you will build an HMM model of this world. Then, given only sensor information (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 positions.

We also are 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 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 uniformly at random. Data for this problem is in `robot_with_momentum.data`.

(i) [**3 Points**] Define the HMM for modeling the no-momentum case in terms of states, observations, and the model parameters necessary.

(ii) [**12 Points**] Run `viterbi.py` on the two datasets, `robot_with_momentum.data` and `robot_no_momentum.data`, to learn an HMM for each condition, and also to infer the sequences of locations visited by the robot. Examine the results, and explore when, how, and why a hidden Markov model is successful or not in inferring the robot's true sequences of positions. Your code should pass the `test_small_robot_dataset` test.

Give a *brief* write-up (say, in 1-3 paragraphs for each problem) of what you found. Include any observations you may have made about how well HMMs work on these problems and why. Your observations can be quantitative (for instance, the test performance was x, which is computed by one of the support tasks) or anecdotal (for instance, the method worked really well at recovering these sequences of positions, but not these other ones). Try to think of plausible explanations for your observations, and in the case of failures, try to think of modifications of our approach that might lead to greater success.

Although this write-up is quite open ended, **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 these assumptions match the actual domain?
- To what extent was, or was not, performance hurt by making such realistic (or unrealistic) assumptions?

Problem 4

[**45 Points**] In this exercise, you will implement the Expectation-Maximization (EM) algorithm for HMMs, which goes by the name Baum-Welch (BW), and use it for a classification problem where we want to decide which city a sequence of weather info came from. We need to use EM because the hidden states are not available in this data. First, let's find out more about this weather problem and the data given.

We are given training sequences from multiple cities, and will train an HMM for each city using

EM. We can then use these HMMs to classify which city a new sequence of observations might be from.

The files `weather_*.data` contain observation sequences that look like this (except longer):

```
boston rain clouds rain clouds rain sun
.
boston rain sun rain sun clouds clouds
.
seattle rain sun clouds rain clouds rain fog
.
seattle clouds sun rain clouds rain rain rain
.
phoenix clouds clouds clouds clouds rain fog
.
LA sun sun sun sun clouds sun clouds clouds
```

(Note that this is the same file format as before, where data points are separated by "." and the full dataset is separated into training data and testing data by "..". However, the semantics are different here. Each line is a sequence generated from some unknown sequence of hidden states, but from the appropriate model: boston, seattle, etc.)

Other files relevant to this problem and not used in the previous problem include:

- `classfy.py`: Support code in Python for performing learning and classification with HMMs. At the command line, you can type `python classify.py [options] N datafile.data (pass -h for more info)`, where N is the number of hidden states for the HMM.

(a) [**3 Points**] Describe the HMM for modeling this weather problem in terms of hidden states, observations, and the model parameters necessary.

(b) [**22 Points**] Implementing EM.
Since the hidden states are not available in this data set, we need to use EM. The functions implementing the E-step and M-step updates are called by the `baumwelch` function in `hmm.py`.

   (i) [**7 Points**] Fill in the `compute_expectation_step` function in `hmm.py` for computing the expected sufficient statistics given a model and the data. Functions for computing alphas and betas, and also gamma and xi values have been provided for you in `hmm.py`. (see lecture notes 15)

   (ii) [**5 Points**] Fill in the `compute_maximization_step` function in `hmm.py` to compute the new model by maximizing the likelihood of the expected sufficient statistics from the E-step.
   Your code should now pass the `test_bw_simple_weather_model` test.

(iii) **[5 Points]** To better understand EM, use the two unit tests `test_bw_beta_all_equal` and `test_bw_gamma_first_col_equal` to answer the following questions. The unit tests run a simple model for one iteration of EM and outputs the beta and gamma values. Include these numbers in your writeup.

(1) Why are the betas all equal for each time step?

(2) Why are the gamma values for the first state the same for all time periods?

(iv) **[5 Points]** You may have noticed in the previous problem that the alpha and beta values get small for long sequences. If you look at the `get_alpha` function, you can see that there is normalization code. Why is the sum of the logs of the normalization factors the log likelihood of the returned observation sequence?

(c) **[20 Points]** Using HMMs for classification.
Now that we have the EM algorithm in place, we can use EM to train an HMM for each city. We can then classify a new sequence of observations by computing the probability of that sequence under each of the models, and returning the city corresponding to the highest probability.

You will explore several different classification tasks:

- Classifying sequences from Boston and LA.
(`weather_bos_la.data`)
- Classifying sequences from Boston and Seattle.
(`weather_bos_sea.data`)
- Classifying sequences from Boston, Phoenix, Seattle and LA.
(`weather_all.data`)

(i) **[5 Points]** Run `classify.py` on each data set to learn an HMM for each city from the training data, and then use them to classify the test data. Vary the number of hidden states from 1 to 6. Plot the final classification accuracy vs the number of hidden states. There is a web-interface task created for this.

(ii) **[5 Points]** Plot $log(P(D|\Theta))$ for training the model for Boston with varying numbers of hidden states. Do you see any patterns? There is a web-interface task created for this.

(iii) **[5 Points]** By looking at the data and the learned models, try to explain your results. When do more hidden states help? When are they not needed? What is it about the observations that makes more states unnecessary?

(iv) **[5 Points]** The data, as you've probably guessed, was generated synthetically. How many hidden states do you think were in our generating model? Why?