CS 476/676, Spring 2015
Problem Set #1b
(Due by 11:59pm on Tuesday, March 3)

---

1 Instructions

This assignment includes programming problems, modeling problems, and analytical questions. Please read the document carefully.

1.1 What to Hand In

All of your submission files should be handed in as a single archive named `hw1b-username1-username2-username3.zip`, where the `usernames` have been replaced with the JHED IDs of your group members. (Your group is allowed to have between 2 and 4 members; see Section 1.2 for details on collaboration.) The `Deliverables` section of each problem provides details of what your zip archive should contain.

Hand in the zip archive by creating a private note to the instructors on Piazza with the title *Submission 1b from* `<list of names of team members>`, and attaching your zip file to that note. The note should be submitted to the `submission1b` folder.

1.2 Submission Policies

Please note the following:

- **Collaboration:** Please work in groups of size 2, 3 (preferable) or 4 people. The homeworks are a way for you to work through the material you're learning in this class on your own. But, by working in a group, and debugging each other's solutions, you'll have a chance to learn the material in more depth. The recommended format for tackling these problem sets is the following. Write a high level sketch of the solution for all of the problems on your own. Meet as a group to brainstorm your solutions and converge on a solution as a group. It is important that you have a good understanding of how you'd have approached the problem independently before discussing your solution with the other group members. Developing this intuition will serve you well in the final exam where you will be required to work on your own. Persuant to your group meeting, write up the solutions on your own. Thereafter, meet as a group to clean up and submit a final write up as a group. By now, each of you should have a solid understand of the concepts involved, and by meeting as a group, you've had a chance to see common ways in which one can make mistakes. Submit your final solution as a final writeup for the group. Your submission should include the names of every team member. Also, name your file as `hw1b-username1-username2-username3.zip`.

- **Late Submissions:** We allow each student to use up to 3 late days over the semester. You have late days, not late hours. This means that if your submission is late by any amount of time past the deadline, then this will use up a late day. If it is late by any amount beyond 24 hours past the deadline, then this will use a second late. **If you jointly submit an assignment as a team, then every team member will lose late**

days if the assignment is submitted late. If you collaborate with team members but independently submit your own version, then late hours will only apply to you.

2 Problem Set 1b (115 Points)

2.1 Inference in a Simple Model [50 points]

Suppose Mary wants to be able to predict, based on the properties of the world around her, how likely she is to get sick with the flu in the near future. She decides to do this by encoding the properties of the world as variables in a graphical model. In this homework and the next, you'll consider a variety of models that describe the flu in a population.

In this section, we will provide you with a simple flu contagion model which contains three binary random variables. Your job will be to write a program that can read in this model and output solutions to queries (e.g. "what is the probability of X given Y?"). The network is shown below:

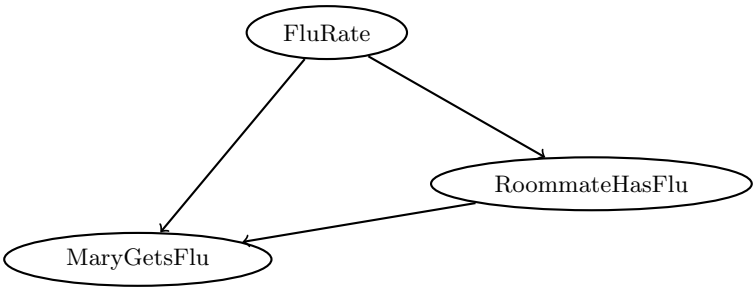


Figure 1: Simple network.

The first random variable (**MaryGetsFlu**) describes whether or not Mary will contract the flu virus within the next week. The probability of Mary catching the flu increases if her roommate has the flu, described with the binary variable **RoommateHasFlu**. If the population's rate of the flu in Mary's city (**FluRate**) is `Elevated` (as opposed to `NotElevated`) then the probability of Mary catching the flu and the probability of the roommate having the flu are both increased.

This model is encoded in the two supplied files: `network-simple.txt` and `cpd-simple.txt`.

The file `network-simple.txt` encodes the variables and network structure. The first line of this file simply reads `3`. This will tell the program that there are 3 random variables in the network. The next 3 lines each contain the names of the variables in the network followed by the values (a comma-separated list) the variable can take (one variable per line). For example, the following line:

```
RoommateHasFlu Yes,No
```

indicates that the network will contain a variable called *RoommateHasFlu* which can take the values `Yes` and `No`. All variables in this simple network happen to be binary, but your code should also be able to handle multivariate discrete variables.

After the variable declarations, all subsequent lines encode edges in the network. Each line contains one variable name, followed by a right arrow, followed by a second variable name, such that there is a directed edge from the first to the second. For example, the following line:

```
FluRate -> RoommateHasFlu
```

indicates a directed edge from the *FluRate* node to the *RoommateHasFlu* node. For clarity, the input files should only use right arrows (`->`) and not left arrows.

The file `cpd-simple.txt` encodes the conditional probability distributions in this model. Each line defines a conditional probability, where the set of conditional probabilities that need to be defined are determined by the network structure. The first part of the line is a comma-separated list of variable/value pairs (of the form `VariableName=Value`) for the left-hand side of the conditional probability definition. The second part (following a space) is a comma-separated list of variable/value pairs for the right-hand side of the conditional probability. The third part (following another space) is a floating point number indicating the probability. For example, the following line:

```
MaryGetsFlu=Yes FluRate=Elevated,RoommateHasFlu=Yes 0.8
```

should be interpreted to mean:

$$P(MaryGetsFlu = \texttt{Yes} | FluRate = \texttt{Elevated}, RoommateHasFlu = \texttt{Yes}) = 0.8$$

Once you have the network structure and the conditional probability distributions, you should be able to compute the probability of any set of network variables conditioned on any other set. You will need to marginalize over the values of variables not specified in the right-hand side, for variables that the variables in the left-hand side depend on. For example, computing the marginal probability $P(MaryGetsFlu = \texttt{Yes})$ will involve summing over all possible values for the other two variables.

You should compute the probability distributions in the same way that you would on paper, i.e. by multiplying together the appropriate factors and summing over marginalized variables. In other words, you shouldn't be implementing any of the more complex inference procedures described in later chapters of the book.

A note on **prior probabilities**: When marginalizing over unobserved variables which have no parents, such as the *FluRate* variable in the query $P(MaryGetsFlu = \texttt{Yes})$, you will need to use the prior distribution for this variable. Rather than defining prior probabilities for variables in the `cpd` file, your implementation should simply use the uniform distribution as the prior.

2.1.1 Deliverables [50 points]

You will hand in a program called `bayes-query` that takes network files and queries as input and outputs the corresponding probability. The program will be executed with commands of the following format:

```
./bayes-query network_filename cpd_filename lefthand_side [righthand_side]
```

The left-hand and right-hand sides will follow the same format as the `cpd` file. For example, the following command:

```
./bayes-query network-simple.txt cpd-simple.txt MaryGetsFlu=Yes,FluRate=Elevated RoommateHasFlu=Yes
```

should output $P(MaryGetsFlu = \texttt{Yes}, FluRate = \texttt{Elevated} | RoommateHasFlu = \texttt{Yes})$. **The last line of your program output must be the solution to the query (a probability).**

When grading, we will take the last line (if there are multiple lines) of your program output and treat it as the solution.

The right-hand side is optional in queries. For example, the following command:

```
./bayes-query network-simple.txt cpd-simple.txt MaryGetsFlu=No
```

should output $P(MaryGetsFlu = \texttt{No})$.

Important We must be able to run your program on the CS department's `ugrad` Linux servers.[1] If you are not in the CS department and you do not have an account to access these machines, please contact the course staff for an account. Matlab is available on these systems, for those who wish to use it.

We will grade your program by running queries on new model files that you have not seen, so be sure it can work on arbitrary input according to the guidelines above.

Creating the `bayes-query` program You can write your program in whatever programming language you like, as long as it can be executed on the `ugrad` servers, as mentioned above. However, we will execute your program using the commands described above, which means that you must develop an executable named `bayes-query`. This executable can simply be a shell script which acts as a wrapper to your program. For example, if you implemented your program as a Python program called `bayes-query.py`, then you should create a file called `bayes-query` which contains the following:

```
python bayes-query.py $*
```

You then need to run the following command so that you can actually execute the `bayes-query` script you created:

```
chmod +x bayes-query
```

Thus, when you run `bayes-query`, it will actually run the command `python bayes-query.py` followed by whatever command line arguments were passed in.

Additional Programming Requirements There are two additional requirements that we ask of you:

1. You must print out all the probabilities with 13 significant figures in scientific notation. Below are examples for Matlab, Python, and Java of how to do such a printout for a variable named x.

   Matlab:

   ```
   x = 1.0 / 7.0
   fprintf('%.13e\n', x)
   ```

[1] For more information see:
http://www.cs.jhu.edu/support/wiki/index.php/Linux_Clients_on_the_CS_Undergrad_Net
http://www.cs.jhu.edu/support/wiki/index.php/Category:Computers_Available_on_the_CS_Network

Python:

```
x = 1.0 / 7.0
print '%.13e' % (x)
```

Java:

```
double x = 1.0 / 7.0;
System.out.printf("%.13e", x);
```

2. If you are using Matlab, you must use the `bayes-query` script below. Of course, this requires that you put your Matlab function in the file `bayes_query.m` and that it accepts the three/four command line arguments in order. Note that this may leave some empty lines or the line `\n` at the end of your output. We will just ignore those lines and take the last set of numbers that you printed. Please do not augment or change the scripts below in any way.

```
#!/bin/bash
#
if [ $# -eq 3 ]
then
matlab -nodisplay -nosplash -r "try, bayes_query '$1' '$2' '$3', catch err, disp(err), fprintf('ERROR\n'), end, quit"
elif [ $# -eq 4 ]
then
matlab -nodisplay -nosplash -r "try, bayes_query '$1' '$2' '$3' '$4', catch err, disp(err), fprintf('ERROR\n'), end, quit"
else
echo ERROR
fi
```

2.1.2 Testing and Debugging

The simple model is small enough that you should be able to work out the solution to queries on paper. We recommend coming up with the solution first, then verifying that your inference program gives the same solution. You are also free to write your own input files to check your solutions to other networks.

A common unit test when writing inference programs is to ensure that your probabilities sum to 1: specifically, the distribution over all combinations of values to the left-hand variables in a query should sum to 1. A related sanity check is to ensure that your probabilities always fall within the range [0,1].

2.2 Designing an Extended Model [30 points]

In this section, you will design your own model rather than simply performing inference on a model that we give you. You will modify and extend the model above to include additional variables (described below). It is your job to add edges between the given variables and to construct a corresponding conditional probability table. You have some flexibility in how you define the network and conditional probabilities, but you will need to design this in such a way to conform to the properties that we describe below.

2.2.1 Model Properties

Your extended model will include the three variables from 2.1, but now the `FluRate` can take three discrete values: `Mild`, `Moderate`, `Severe`.

In addition to the current flu rate, you will consider an additional variable describing the city's flu rate two weeks ago (**PreviousFluRate**), which can take the same three values as *FluRate*. The government takes about two weeks to accurately estimate the population's flu rate, so in practice the flu rate from two weeks ago can be used to predict the current flu rate, which is unknown. Additionally, we will consider whether or not we are in one of the "flu season" months (October-March in the U.S.) with the binary variable **IsFluSeason**. The flu follows a somewhat predictable pattern each year, and the flu rate can be roughly predicted based on the time of year.

Similar to her roommate, Mary is also at risk of catching the flu from her coworker, with whom she shares as office. We'll introduce the binary variable **CoworkerHasFlu** which is similar to *RoommateHasFlu*. Finally, whether or not Mary was given the flu vaccine this past year (**MaryIsVaccinated**) will affect her risk of catching the flu.

You must construct a network and conditional probability table from these variables such that the following intuitions are upheld:

1. If Mary's roommate has the flu, Mary is more likely to catch the flu.
2. Similarly, if Mary's coworker has the flu, Mary is more likely to catch the flu.
3. Since Mary comes in closer contact with her roommate than her coworker, the probability of catching the flu increases more if her roommate has the flu than if her coworker has the flu. If both have the flu, then Mary is even likelier to catch it.
4. Marys roommate and coworker are more likely to have the flu if the citys flu rate is moderate than if it is mild. Similarly, they are more likely to have the flu if the city's flu rate is severe than if it is moderate.
5. If we know the flu status of Mary's roommate and coworker, then knowing the city's flu rate won't influence our belief about Mary catching the flu.
6. Both the current flu rate and the flu rate two weeks ago are most likely to be mild if it is **not** currently the flu season. They are less likely to be moderate and very unlikely to be severe. If it **is** currently the flu season, then the flu rates are most likely to be moderate, followed by severe. The probability of being severe during the flu season is much higher than the same probability during the off season.
7. The current flu rate in Mary's city is most likely to be the same level as the flu rate two weeks ago, since the flu rate typically does not change too rapidly. The next most likely level of the current rate depends on whether it is currently the flu season.

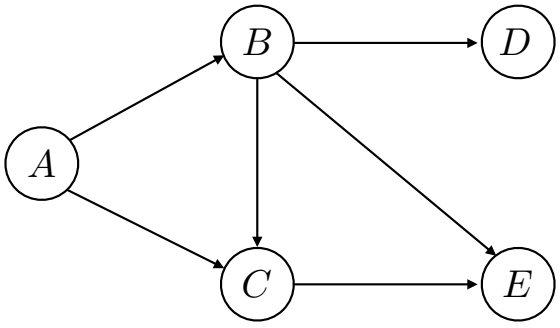
8. Mary is less likely to catch the flu if she has been vaccinated for it this year.

2.2.2 Deliverables [30 points]

We have given you the files `network-extended.txt` and `cpd-extended.txt`. The `network` file contains all of the variable names you should use (do not add additional variables), but you will need to add the directed edges. The `cpd` file is blank; you will need to add the conditional probabilities needed to describe the network you create in the `network` file. You will turn in your version of these two files.

We will grade this by querying your network in various ways and verifying that the 8 properties described in 2.2.1 are true.

2.3 Parameter Estimation [35 points]



2^5=32最多, 在CI的情况，2+2^2+2^3+2^3+2^2=26

Figure 2: Graphical model for the parameter estimation problem.

In the graphical model in Figure 2, each node is a binary random variable, and each CPD is tabular. Table 1 shows a set of samples drawn from this network, where `?` indicates that the observation is missing a value for that variable.

a. **[5 points]** What are the parameters of this network?

b. **[15 points]** Using pseudocode, design an algorithm for learning the network parameters. You may abstract away any procedures that you would like, but it must be **very** clear to us that you know and fully understand what that procedure does. Along with your pseudocode, provide a list of descriptions for each abstracted procedure you use (this is where you should demonstrate your understanding). You cannot, for example, use one line of code that calls a function `learn()`. If you feel like you are using a function that is too high-level, email us to confirm.

2.3.1 Deliverables [35 points]

You will turn in your answers as a `pdf` file called `parameter-estimation.pdf`.

	A	B	C	D	E
1	?	1	0	?	1
2	1	0	0	1	1
3	0	1	?	0	1
4	0	0	1	1	0
5	1	?	0	1	?
6	1	1	0	0	1
7	?	?	0	?	?
8	1	0	1	0	0
9	0	1	1	0	1
10	1	0	0	0	1
11	?	0	?	1	0
12	0	1	0	0	1
13	1	0	0	?	1
14	1	?	1	0	1
15	0	1	?	?	0
16	0	1	0	0	1
17	?	1	1	0	?
18	?	0	?	?	0
19	1	0	1	?	1
20	?	?	1	0	0

Table 1: Observations for the network in Figure 2.