CS5340 Assignment 2 Part 1: Junction Tree Algorithm

1. Part 1 Overview

In the first assignment, you wrote code to perform inference on simple tree-like graphs using belief propagation. However, the sum-product algorithm does not work on loopy networks. For loopy networks to be amenable to the sum-product algorithm, we must first convert these networks into a tree-like structure.

Hence, in part 1 of the second assignment, you will write code to construct a junction tree from a Markov Random Field and perform inference on the junction tree using the sum-product algorithm.

References: Lecture 4 and 5.

Honour Code. This coding assignment (part 1 and 2) constitutes **15%** of your final grade in CS5340. Note that plagiarism will not be condoned! You may discuss with your classmates and check the internet for references, but you MUST NOT submit code/report that is copied directly from other sources!

2. Submission Instructions

Items to be submitted (zip file contain folders part1 and part2):

- **Source code**
  - jt_construction.py – code for constructing the junction tree.
  - main.py – code for junction tree sum-product.
  - factor_utils.py – code for factor helper functions.
- **Report (report1.pdf).** This should describe your implementation and be no more than one page.

Please indicate clearly your name and student number (the one that looks like A1234567X) in the report as well as the top of your source code. Zip the two files together and name it in the following format: **A1234567X_lab2.zip** (replace with your student number). The zip file structure should contain:

- **part1** – folder containing code for assignment 2 part 1
- **part2** – folder containing code for assignment 2 part 2
- **report1.pdf** – pdf document for assignment 2 part 1
- **report2.pdf** – pdf document for assignment 2 part 2.

Submit your assignment by **30 September 2020, 2359HRS** to LumiNUS. 25% of the total score will be deducted for each day of late submission.

3. Getting Started

This assignment as well as the subsequent ones require Python 3.5, or later. You need certain python packages, which can be installed using the following command:

```
pip install -r requirements.txt
```

If you have any issues with the installation, please post them in the forum, so that other students or the instructors can help accordingly.

4. Test Cases

To help with your implementation, we have provided a few sample datasets and their expected outputs to help you debug your code. They can be found in the **data/inputs** folder. The ground-truth files can be found in **data/ground-truth** and our answers (for cross-comparison) can be found in the **data/answers** folder.

Note that the ground-truth might be slightly different from the answers we have provided due to numerical imprecision. During grading, your code will be evaluated on hidden test cases on top of the validation test cases we have provided.

5. Basic Factor Operations

Similar to lab 1, you will have to implement the following basic factor operations in factor_utils.py:

- **`factor_product()`:** This function should compute the product of two factors.
- **`factor_marginalize()`**: This function should sum over the indicated variable(s) and return the resulting factor.
- **`factor_evidence()`**: This function takes in a Factor and some observed values. The factor should be reduced (take slices) to contain only unobserved variables. Note that this is slightly different from lab1.

All the factor operations make use of a custom `Factor` class from lab 1. Note that **factor_evidence()** is slightly different from **observe_evidence()** in lab1.

6. Junction Tree Algorithm

In lab 1, you noticed that even small graphs can have large joint distribution tables and running variable elimination for each variable inefficient compared to running the sum-product algorithm. Hence, we want to extend the sum-product algorithm for loopy markov random fields where we must first construct a junction tree in **jt_construction.py** in the following steps:

a. Build the junction tree graph structure by forming cliques and creating edges between cliques: **`_get_jt_clique_and_edges()[3 points]`**
b. Assign factors in the original graph to the cliques: **`_get_clique_factors()[1 point]`**

Note that we will also provide evidence variables, and the graph structure must be updated with the evidence variables.

Once the junction tree has been constructed, in **main.py**, we will perform:

a. Inference of clique potentials using the sum product algorithm on the constructed junction tree: **`_get_clique_potentials() [2 points]`**
b. Compute the node marginal probabilities e.g. $p(X_2|X_E); p(X_3|X_E)$ from the clique potentials using brute-force marginalization: **`_get_node_marginal_probabilities() [2 points]`**

To retrieve marginal probabilities for all query nodes in the graph. Note that step (b) has 1 point for more efficient solutions. *Hint: cliques have varying sizes.*

Hint: It might be useful to create additional functions for this part. Place these functions between the designated comment blocks for each file.

CS5340 Assignment 2 Part 2: Parameter Learning

7. Part 2 Overview

In part 2 of the second assignment, you will write code on parameter learning under complete observations. In this task, our local conditional probabilities are parameterised by $\boldsymbol{\theta}$ where the where we have $p(x_u|x_{\pi_u}, \theta_u)$. Under complete observations, a set of N observations of our random variables $\boldsymbol{X} = \{x_1^1, \dots, x_V^1, x_1^2, \dots, x_1^N, \dots, x_V^N\}$ are given, and from these observations, we derive maximum likelihood estimates (MLE) for $\boldsymbol{\theta} = \{\theta_1, \dots, \theta_V\}$.

References: Lecture 6

Honour Code. This coding assignment (part 1 and 2) constitutes **15%** of your final grade in CS5340. Note that plagiarism will not be condoned! You may discuss with your classmates and check the internet for references, but you MUST NOT submit code/report that is copied directly from other sources!

8. Submission Instructions

Items to be submitted:

- **Source code**
  - main.py – code should only be written here.
- **Report (report2.pdf).** This should describe your implementation (derivations) and be no more than one page.

Please indicate clearly your name and student number (the one that looks like A1234567X) in the report as well as the top of your source code. Zip the two files together and name it in the following format: **A1234567X_lab1.zip** (replace with your student number).

Submit your assignment by **30 September 2020, 2359HRS** to LumiNUS. 25% of the total score will be deducted for each day of late submission.

9. Getting Started

This assignment as well as the subsequent ones require Python 3.5, or later. You need certain python packages, which can be installed using the following command:

```
pip install -r requirements.txt
```

If you have any issues with the installation, please post them in the forum, so that other students or the instructors can help accordingly.

10. Test Cases

To help with your implementation, we have provided a few sample datasets and their expected outputs to help you debug your code. They can be found in the **data/inputs** folder. The ground-

truth parameters can be found in **data/gt-parameters** and our answers (for cross-comparison) can be found in the **data/answers** folder.

Note that the ground-truth might be slightly different from the answers we have provided since we have a finite number of observations. During grading, your code will be evaluated on hidden test cases on top of the validation test cases we have provided.

Test cases 1 and 2 uses the simple single node graph example in Lecture 6, and test cases 3 and 4 use the slightly more complex three node graph example in Lecture 6, with different number of observations.

Note that our hidden test cases will consist of slightly more complex graphs. Your code should scale. We will not introduce evidence (for the parameters $\boldsymbol{\theta}$) in any test case.

11. Parameter Learning Under Complete Observations

In part 2, we assume that observations are generated using a linear-Gaussian model i.e.

$$p(x_u|x_{\pi_u},\theta_u) = \mathcal{N}_u[w_{u0},..,w_{uC},\sigma_u^2] = \frac{1}{\sqrt{2\pi\sigma_2}}\exp\left\{-0.5\frac{\left(x_u-\left(\sum_{c\in x_{\pi_u}} w_{uc}x_{uc}+w_{u0}\right)\right)^2}{\sigma_u^2}\right\}$$

To derive the MLE estimates of $\boldsymbol{\theta}$**,** we can equivalently find the maximum log-likelihood estimate:

$$\begin{aligned}&\underset{\theta_u}{\text{argmax}}\sum_{n=1}^{N}\log p(x_u|x_{\pi_u},\theta_{x_u|x_{\pi_u}})\\ &\qquad=\underset{\theta_u}{\text{argmax}}\sum_{n=1}^{N}\left\{-\frac{1}{2}\log 2\pi\sigma_u^2-\frac{1}{2\sigma_u^2}\left(x_{u,n}-\left(\sum_{c\in x_{\pi_u}} w_{uc}x_{uc,n}+w_{u0}\right)\right)^2\right\}\end{aligned}$$

And taking the derivatives, we derive linear equations for each parameter $\{w_{u0},\dots,w_{uC},\sigma_u\}$ and solve for their MLE estimates using the following steps:

a. Solve for the linear equations for $\{w_{u0},\dots,w_{uC}\}$: **`_learn_node_parameter_w()[3 points].`**
b. Solve for the linear equation for the variance σ^2 using $\{w_{u0},\dots,w_{uC}\}$: **`_learn_node_parameter_var()[1 point]`**

You will call the above steps for each node in **`_get_learned_parameters() [2 points]`** to learn all parameters. Your derivations in the report are also important **`[1 point].`**

Hint: It might be useful to create additional functions for this part. Place these functions between the designated comment blocks for each file.