# CS 475 Machine Learning: Homework 5
## Graphical Models

Due: Tuesday November 24, 2015, 11:59pm

100 Points Total Version 1.0

**Make sure to read from start to finish before beginning the assignment.**

## 1 Programming (60 points)

In this assignment you will implement a loopy belief propagation (BP) algorithm for calculating marginal probabilities in a loopy MRF (more specifically, a factor graph). You will implement loopy belief propagation using provided functions for each factor (there is no learning in the homework, only inference). As background, we first summarize inference on a linear chain factor graph using message passing (sum product.) We will then explain how this algorithm can be adapted to a loopy graph structure. Since this new structure is not a DAG, we no longer have a guarantee that we will get the correct answer. Therefore, loopy BP is an approximate inference algorithm.

### 1.1 Message Passing on a Chain Factor Graph

In this section we briefly introduce the sum product algorithm on a chain factor graph for computing the marginal of each variable on the chain.

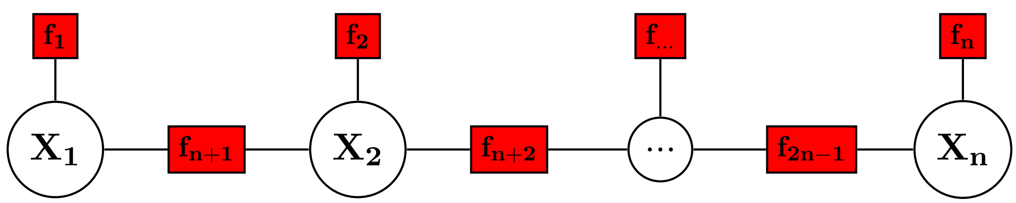



Figure 1: The chain factor graph.

Each of the variables in our chain (the $x_i$) are $k$-ary discrete variables.

We can compute the marginal probabilities for each $x_i$ in this chain using the sum product algorithm. We review the algorithm here. Our presentation is adapted from section 8.4.4 of Bishop. For more details see the book (http://research.microsoft.com/en-us/um/people/cmbishop/prml/Bishop-PRML-sample.pdf).

Our goal is to find the marginal probability of a node in the factor graph. Due to the linear structure of the factor graph, the sum product algorithm lets us write this marginal probability as:

$$p(x) = \prod_{s \in N(x)} \left[ \sum_{X_s} F_s(x, X_s) \right] \tag{1}$$

where:

$N(x)$ is the set of all factor node neighbors of $x$

$F_s(x, X_s)$ represents the product of all the factors "downstream" of $f_s$

Part of the above equation will be used many times, so for the sake of computation and intuition, we will define the sum term as a "message" $\mu$:

$$\mu_{f_s \to x}(x) := \sum_{X_s} F_s(x, X_s) \tag{2}$$

Messages can be defined using recursion: $F_s(x, X_s) = f_s(x, x_1, \ldots, x_M) \prod_{m \in N(f_s) \setminus x} \mu_{x_m \to f_s(x_m)}$, where $X_s = \{x_m | m \in N(f_s) \setminus x\}$. Our base case is:

$$\mu_{f \to x}(x) := f(x) \text{ iff the only neighbor of f is x} \tag{3}$$

The sum product algorithm defines the message from a variable node to a factor node as the product of the messages it receives from its "downstream" factors:

$$\mu_{x \to f_s} := \prod_{l \in N(x) \setminus s} \mu_{f_l \to x}(x) \tag{4}$$

As before, there is a base case for this equation:

$$\mu_{x \to f}(x) := 1 \text{ iff the only neighbor of x is f}$$

We can now find the marginal probabilities using Eq. (1). While our presentation focused on chains, the Sum Product algorithm applies to any tree structured factor graph. While seemingly simple, the details may be a bit opaque. To help clarify them, we will describe the message passing procedure on our specific chain graph in Fig. (1).

Let's say, we want to compute $p(x_2)$. According to Eq. (1,2), we have $p(x_2) = \mu_{f_{n+1} \to x_2}(x_2)\mu_{f_{n+2} \to x_2}(x_2)\mu_{f_2 \to x_2}(x_2)$. So we need to compute messages $\mu_{f_{n+1} \to x_2}, \mu_{f_{n+2} \to x_2}$ and $\mu_{f_2 \to x_2}$. From Eq. (2,3), these messages can be computed as:

$$\mu_{f_{n+1} \to x_2}(x_2) = \sum_{x_1} \mu_{x_1 \to f_{n+1}}(x_1) f_{n+1}(x_1, x_2) \tag{5}$$

$$\mu_{f_{n+2} \to x_2}(x_2) = \sum_{x_3} \mu_{x_3 \to f_{n+2}}(x_3) f_{n+2}(x_2, x_3) \tag{6}$$

$$\mu_{f_2 \to x_2}(x_2) = f_2(x_2) \tag{7}$$

where Eq. (7) is already a base case while Eq. (5,6) are not. We continue to expand $\mu_{x_1 \to f_{n+1}}(x_1)$ in Eq. (5) and $\mu_{x_3 \to f_{n+2}}(x_3)$ in Eq. (6), according top Eq. (4) and (3):

$$\mu_{x_1 \to f_{n+1}}(x_1) = \mu_{f_1 \to x_1}(x_1) = f_1(x_1) \tag{8}$$

$$\mu_{x_3 \to f_{n+2}}(x_3) = \mu_{f_3 \to x_3}(x_3)\mu_{f_{n+3} \to x_3}(x_3) = f_3(x_3)\mu_{f_{n+3} \to x_3}(x_3) \tag{9}$$

where Eq. (8) is already a base case while Eq. (9) is not. So we continue to expand $\mu_{f_{n+3} \to x_3}(x_3)$ in Eq. (9) and so on, until we reach the other end of the chain.

To summarize, to compute the marginal of a variable in the chain, we need to compute the messages starting from both ends of the chain one by one, until that variable has collected all its incoming messages, where each message is computed only once and then is finalized.

### 1.2 Message Passing on a Loopy Factor Graph

In this assignment you are asked to compute the marginal probability of a variable in a loopy factor graph. The graph is shown in Fig. 2. This graph is exactly the same as our linear chain, except that node $x_N$ is connected to $x_1$.

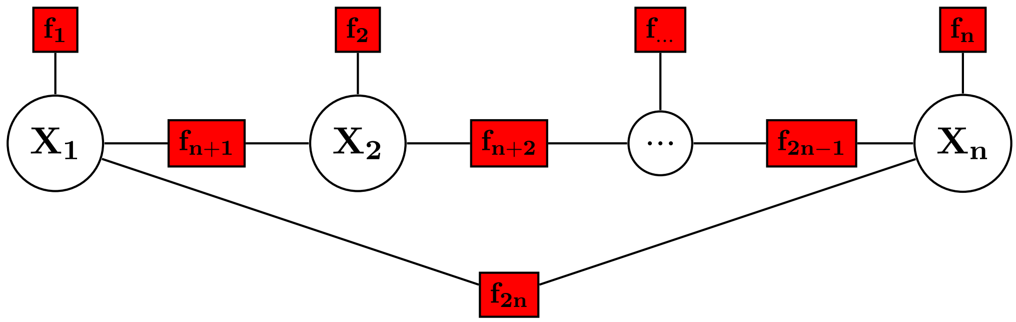



Figure 2: The loopy factor graph.

As before, each of the variables in our loop (the $x_i$) are $k$-ary discrete variables.

Unfortunately, the procedures described in Sum Product Algorithm work for tree structured graphs only and cannot be directly applied in a loopy factor graph. If we directly applied the algorithm, we would introduce infinite recursion since there would be no base case.

Therefore, we modify the algorithm to adapt it to loopy graphs. Our change will be to start passing incomplete messages, messages that are missing information. This will obviate the problem of not having a base case to start from. We will do this by specifying an order for computing messages and then updating messages in turn. However, since messages are missing information, we cannot stop once all messages have reached their final destination. Instead, we will continue passing messages throughout the graph for $T$ iterations, or more generally, until the values of the messages (and the marginals) have converged. Note that this new algorithm has no guarantee of correctness or convergence, but provides approximate marginals. In many cases, the results provide a good approximation to actual marginals.

The modified algorithm falls in a broader class of algorithms called *Loopy Belief Propagation*, where "Belief" here refers to how a node thinks part of the graph should be, i.e., the part that is in the opposite direction of where the message is passed. Beliefs are stored in the messages and are passed in the graph in order to update other messages. Each message is updated $T$ times.

The modified algorithm is as follows:

1. Initialize $\mu_{x_1 \to f_{n+1}}(x_1) = 1$ and $\mu_{x_1 \to f_{2n}}(x_1) = 1$ for all values $x_1$ can take.
2. For $t$ from 1 to $T$
   (a) For $i$ from 1 to $n$
      i. Compute $\mu_{f_{n+i} \to x_{1+i\%n}}(x_{1+i\%n})$ from Eq. (2).
      ii. Compute $\mu_{x_{1+i\%n} \to f_{n+1+i\%n}}(x_{1+i\%n})$ from Eq. (4).
   (b) For $i$ from $n$ to 1
      i. Compute $\mu_{f_{n+i} \to x_i}(x_i)$ from Eq. (2).
      ii. Compute $\mu_{x_i \to f_{n+(i-2)\%n+1}}(x_i)$ from Eq. (4).

After messages are passed as shown above, we will compute the marginal $p(x_i)$ for any $i$ according to Eq. (1,2).

### 1.3 Implementation

For the factor graph in this assignment, there are unary ($f_1$ to $f_n$) and binary ($f_{n+1}$ to $f_{2n}$) factors, and each factor $f_i$ is associated with a potential function $\psi_i$. Each unary factor will have a potential function $\psi_i(a)$ which returns a real non-negative value corresponding to the potential when variable $x_i$ takes value $a$. Each factor node between two variable nodes will have a potential function $\psi_i(a, b)$ which returns a value corresponding to the potential when variable $x_{i-n}$ takes value $a$ and node $x_{i-n+1}$ takes value $b$ (except for the factor $f_{2n}$ where $x_n$ takes value $a$ and $x_1$ takes value $b$). Recall that every $x_i$ can take values from 1 to $k$.

We will provide you code that gives you values of $\psi_i(a)$ and $\psi_i(a, b)$. They will be in the class `cs475.loopMRF.LoopMRFPotentials` and have the signatures:

```
public double potential(int i, int a)
public double potential(int i, int a, int b)
```

There are $n$ nodes in this loop, so the value of `i` must be between 1 and $n$ (inclusive) in the first method and between $n+1$ and $2n$ (inclusive) in the second method. Since every $x_i$ can take values from 1 to $k$ (inclusive), you must only call this function with values for `a` and `b` between 1 and $k$ (inclusive). You will be able to get values for $n$ and $k$ by calling the following functions in `cs475.loopMRF.LoopMRFPotentials`:

```
public int loopLength()  // returns n
public int numXValues()   // returns k
```

These values will be read into `cs475.loopMRF.LoopMRFPotentials` from a text file that must be provided in the constructor:

```
public LoopMRFPotentials(String data_file, int iterations)
```

We are providing you with a sample of this data file, `sample_mrf_potentials.txt`. The format is `"n k"` on the first line and either `"i a potential"` or `"i a b potential"` on subsequent lines. Feel free to try out new loops to get different probability distributions, just make sure it contains all the needed potential values.

Your code will work by calculating these messages given the value of the potential functions between the variable nodes in the loop. For details on how to do this, you can refer to your notes from class, or see Bishop's examples in the book.

#### 1.3.1 Command Line Arguments

We have already added 2 command line options in `cs475.loopMRF.LoopMRFPotentials` for you.

```
registerOption("data", "String", true, "The data to use.");
registerOption("iterations", "int", true, "The number of iterations. default: 50");
```

### 1.4 What You Need to Implement

We have provided you with class `LoopyBP`, with one method left blank that you will need to implement:

```
public class LoopyBP {
    public double[] marginalProbability(int x_i) {
        // TODO
    }
}
```

The method should return a double array where the $j$th element is the probability that $x_i = j$. The length of this array should be $k + 1$ and you should leave the 0 index as 0. These are probabilities so don't forget to normalize to sum to 1.

### 1.5 How We Will Run Your Code

We will run your code by providing you with a single command line argument which is the data file:

```
java cs475.loopMRF.LoopMRFTester -data mrf_potentials.txt -iterations T
```

Note that we will use new data files with different values of $n$ and $k$, so make sure your code works for any reasonable input.

Your output should just be the results of the print statements in the code given. **Do not print anything else in the version you hand in.**

## 2 Analytical (40 points)

**1. (10 points)** Consider the Bayesian Network given in Figure 3(a). Are the sets $\mathbf{A}$ and $\mathbf{B}$ d-separated given set $\mathbf{C}$ for each of the following definitions of $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C}$? Justify each answer.

a. $\mathbf{A} = \{x_1\}$, $\mathbf{B} = \{x_9\}$, $\mathbf{C} = \{x_5, x_{14}\}$

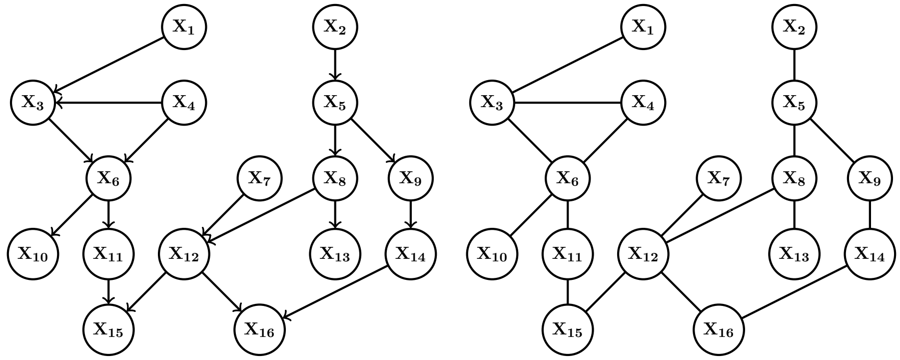
b. $\mathbf{A} = \{x_{11}\}$, $\mathbf{B} = \{x_{13}\}$, $\mathbf{C} = \{x_1, x_{15}\}$

c. $\mathbf{A} = \{x_4\}$, $\mathbf{B} = \{x_5\}$, $\mathbf{C} = \{x_{10}, x_{16}\}$

d. $\mathbf{A} = \{x_3, x_4\}$, $\mathbf{B} = \{x_{13}, x_9\}$, $\mathbf{C} = \{x_{10}, x_{15}, x_{16}\}$

Now consider a Markov Random Field in Figure 3(b), which has the same structure as the previous Bayesian network. Re-answer each of the above questions with justifications for your answers.

**2. (10 points)** Let $X = (X_1, ..., X_{16})^T$ be a random vector with distribution given by the graphical model in Figure 3(a). Consider variables $X_i \in \{X_6, X_8, X_{12}\}$. For each $X_i$, what is the minimal subset of the variables, $A \subset \mathcal{X} - \{X_i\}$, such that $X_i$ is independent of the rest of the variables $(\mathcal{X} - (A \cup \{X_i\}))$ given A? Justify your answer. Answer the same questions for Figure 3(b).




(a) Directed Graphical Model (b) Undirected Graphical Model

Figure 3: Two graphs are the same. However since (a) is directed and (b) is undirected the two graphs have different a conditional independence interpretation.

**4. (10 points)** The notation $(A \perp B|C)$ means A and B are conditionally independent given C. Prove or disprove (by providing a counter-example) each of the following properties of independence:

(a) $(X \perp Y, W|Z)$ implies $(X \perp Y|Z)$.

(b) $(X \perp Y|Z)$ and $(X \perp W|Y, Z)$ imply $(X \perp Y, W|Z)$.

(c) $(X \perp Y|Z)$ and $(X \perp W|Z)$ imply $(X \perp Y, W|Z)$.

**5. (10 points)** A Markov Random Field usually cannot help us with the factorization of the distribution function. However, some MRFs can be converted to Bayesian Networks. For example, consider the graph structure in Figure 4. From this graph, we know that $X_2$

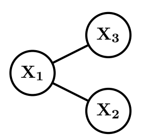



Figure 4: The Original Undirected Graph

and $X_3$ are conditionally independent given $X_1$. We can draw the corresponding directed graph as Figure 5. This suggests the following factorization of the joint probability:

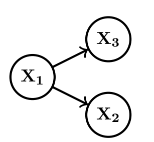



Figure 5: The Converted Directed Graph

$$P(X_1, X_2, X_3) = P(X_3|X_1)P(X_2|X_1)P(X_1)$$

Now consider the following graphical model in Figure 6.
As before, we can read the conditional independence relations from the graph.

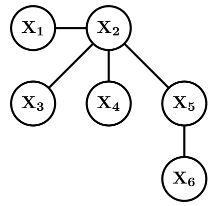



Figure 6: An Undirected Graph

(a) Following the example above, show a factorization of the joint distribution.

(b) Is this factorization unique, meaning, could you have written other factorizations that correspond this model?

(c) If the factorization is unique, show why it is unique. If it is not unique, provide an alternate factorization.

## 3 What to Submit

In each assignment you will submit two things.

1. **Code:** Your code as a zip file named `library.zip`. **You must submit source code (.java files)**. We will run your code using the exact command lines described above, so make sure it works ahead of time. Remember to submit all of the source code, including what we have provided to you.

2. **Writeup:** Your writeup as a **PDF file** (compiled from latex) containing answers to the analytical questions asked in the assignment. Make sure to include your name in the writeup PDF and use the provided latex template for your answers.

Make sure you name each of the files exactly as specified (library.zip and writeup.pdf).

To submit your assignment, visit the "Homework" section of the website (http://www.cs475.org/.)

## 4 Questions?

Remember to submit questions about the assignment to the appropriate group on the class discussion board: http://bb.cs475.org.