CS 475 Machine Learning: Homework 5
Graphical Models
Due: Friday Nov 18, 2016, 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 the sum product algorithm for calculating marginal probabilities, and the max sum algorithm for finding the configuration that gives the highest global probability in a linear chain MRF (more specifically, a factor graph). You will implement sum product and max sum on top of potential functions that we give you (there is no learning in the homework, only exact inference).

1.1 The Factor Graph

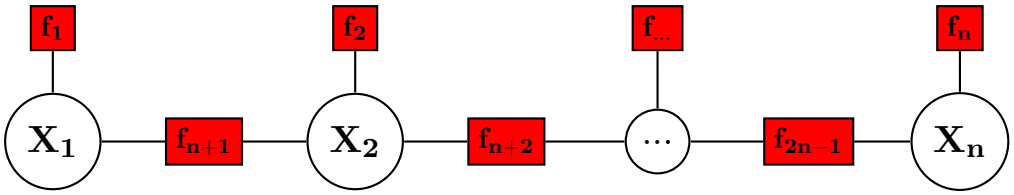


Figure 1: The factor graph that will be used in this assignment

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

1.2 Sum Product Algorithm

There are several presentations of the sum product algorithm. We will follow the presentation in class, which is based onsection 8.4.4 of Bishop. For more details and diagrams, you can see this chapter which is available freely online (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 our factor graph. Due to the linear structure of our 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" μ:

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

If you look at the diagram in the book, you will see the recursive nature of $F_s(x, X_s) = f_s(x, x_1, \ldots, x_M) \prod_{m \in N(f_s) \setminus x} \mu_{x_m \to f_s(x_m)}$, where $X_s = \{x_m | m \in N(f_s) \setminus x\}$. Our base case is:

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

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:

$$\mu_{x \to f_s} := \prod_{l \in N(x) \setminus s} \mu_{f_l \to x}(x) \tag{3}$$

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

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

And we're done: we can find marginal probabilities using equation 1. While seemingly simple, the details may be a bit opaque. To help clarify them, you will implement Sum Product on the linear chain factor graph above.

1.3 Max Sum Algorithm

Instead of finding a marginal probability of a set of variables taking some values, in this task our goal is to find a setting of the variables that has the maximal probability. We can find this configuration, and the associated probability using the "Max-Sum Algorithm". Conceptually, the main difference between sum-product and max-sum is that, while sum-product finds the distribution over a set of variables, of which the max is *locally* most likely, max-sum finds the configuration that is globally most likely, i.e., maximizing the joint distribution:

$$x^{max} = \arg\max_x p(x) \tag{4}$$

The value of the corresponding probability is given by:

$$p(x^{max}) = \max_x p(x) \tag{5}$$

The message passing scheme in max product is similar to that in sum product. We only need to replace the "marginalization" with "maximization" in Eq. (1) and (2) as:

$$p(x) = \prod_{s \in N(x)} \left[\max_{X_s} F_s(x, X_s) \right] \tag{6}$$

$$\mu_{f_s \to x}(x) := \max_{X_s} F_s(x, X_s) \tag{7}$$

The joint probability $p(x)$ is usually quite small, which might lead to inaccuracy of the computation due to the underflow problem. Therefore in practice we take the logarithm of probability values and push "log" into all terms so that all the products of probabilities are now replaced with sums of logarithm of probabilities. That is why the algorithm is called "Max-Sum". You might see "Max-Product" which refers to the same algorithm. The distinction refers to whether or not you use logs.

In order to recover the configuration of all variables (i.e., the value for each variable) after the largest joint probability is found, we also need to record the assignment $\arg\max_{X_s} F_s(x, X_s)$ for X_s along with the message $\mu_{f_s \to x}(x)$, which means that X_s is configured as $\arg\max_{X_s} F_s(x, X_s)$ if x is configured as some value. Consequently, at the moment the largest joint probability is computed, we can backtrack to obtain values for each variable.

Putting everything together, the goal is to compute:

$$\max_x p(x) = \max_x \sum_{s \in N(x)} \left[\mu_{f_s \to x}(x) \right] \tag{8}$$

$$\mu_{f_s \to x}(x) := \begin{cases} \log f_s(x) & \text{if the only neighbor of } f_s \text{ is } x \\ \max\limits_{x_1, \ldots, x_M} \left[\log f_s(x, x_1, \ldots, x_M) + \sum\limits_{m \in N(f_s) \setminus x} \mu_{x_m \to f_s(x_m)} \right] & \text{otherwise} \end{cases} \tag{9}$$

$$\mu_{x \to f_s}(x) := \begin{cases} 0 & \text{if the only neighbor of } x \text{ is } f_s \\ \sum\limits_{l \in N(x) \setminus s} \mu_{f_l \to x}(x) & \text{otherwise} \end{cases} \tag{10}$$

where $X_s = \{x_m | m \in N(f_s) \setminus x\}$ is recorded for x along with the message $\mu_{f_s \to x}(x)$.

You can find details on this algorithm in section 8.4.5 of Bishop.

1.4 Implementation

For the factor graph in this assignment, there are unary (f_1 to f_n) and binary (f_{n+1} to f_{2n-1}) factors, and each factor f_i is associated with a potential function ψ_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. 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 `chain_mrf.ChainMRFPotentials` and have the signatures:

```
potential(self, i, a)
potential(self, i, a, b)
```

There are n nodes in this chain, so the value of `i` must be between 1 and n (inclusive) in the first method and between $n + 1$ and $2n - 1$ (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 `chain_mrf.ChainMRFPotentials`:

```
chain_length(self)  // returns n
num_x_values(self)   // returns k
```

These values will be read into `chain_mrf.ChainMRFPotentials` from a text file that must be provided in the constructor:

```
__init__(self, data_file)
```

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 chains to get different probability distributions, just make sure it contains all the needed potential values.

Since our graph is a chain, which means that the number of elements in $X_s = \{x_m | m \in N(f_s) \setminus x\}$ is at most 1, we can simply create a 1D array of length k for each variable x for backtracking. **In case of a tie, always choose the lowest node index**.

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

1.5 What You Need to Implement

We have provided you with two classes, `chain_mrf.SumProduct` and `chain_mrf.MaxSum`, with one method for each left blank that you will need to implement:

```
class SumProduct:
    def marginal_probability(self, x_i):
        // TODO

class MaxSum:
    def max_probability(self, x_i):
        // TODO
```

The first should return a **python list of type float** where the j-th element is the probability that $x_i = j$. The length of this list should be $\mathbf{k+1}$ and you should leave the 0 index as 0. These are probabilities so don't forget to normalize to sum to 1.

The second should return a **float scalar** which is the logarithm of the joint probability corresponding to the most probable setting of all variables. The assignment of each variable should be stored in the class member variable `_assignments`, which is a **python list of type int** of length $\mathbf{n+1}$. Note that different values for `x_i` of the method `max_probability` should give the same result. You should use `math.log` for logarithm computations.

1.6 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:

```
python chain_mrf_tester.py mrf_potentials.txt
```

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. (12 points) Consider the Bayesian Network given in Figure 2(a). Are the sets **A** and **B** d-separated given set **C** for each of the following definitions of **A**, **B** and **C**? Justify each answer.

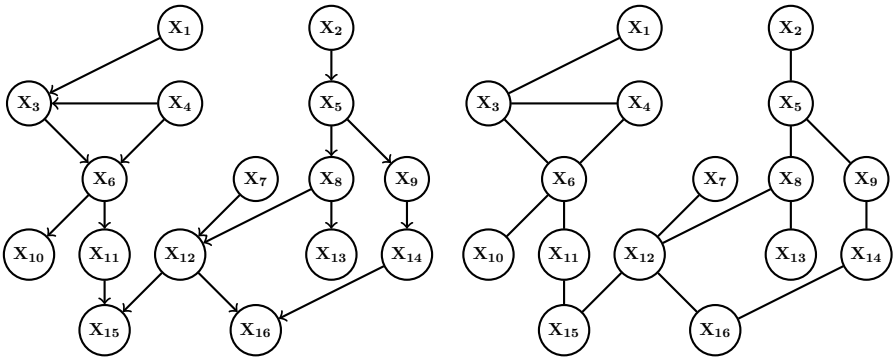
a. $\mathbf{A} = \{x_1\}$, $\mathbf{B} = \{x_9\}$, $\mathbf{C} = \{x_5, x_{14}\}$

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 2(b), which has the same structure as the previous Bayesian network. Re-answer each of the above questions with justifications for your answers.



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

Figure 2: Two graphs are the same. However since (a) is directed and (b) is undirected the two graphs have different a conditional independence interpretation.

2. (15 points) Let $X = (X_1, ..., X_d^T)$ be a random vector, which follows a d-variate Gaussian distribution $N(0, \Sigma)$. Define the precision matrix as $\Omega = \Sigma^{-1}$. Please prove that $\Omega_{ij} = 0$ implies the conditional independence between variables X_i and X_j given the remaining variables, i.e., a sparsity pattern Ω is equivalent to an adjacency matrix of a Gaussian Markov Random Field.

3. (13 points) The probability density function of most Markov Random Fields cannot be factorized as the product of a few conditional probabilities. This question explores some MRFs which can be factorized in this way.

Consider the graph structure in Figure 3. From this graph, we know that X_2 and X_3

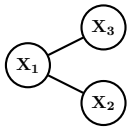


Figure 3: The Original Undirected Graph

are conditionally independent given X_1. We can draw the corresponding directed graph as Figure 4. This suggests the following factorization of the joint probability:

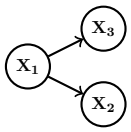


Figure 4: The Converted Directed Graph

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

Now consider the following graphical model in Figure 5.

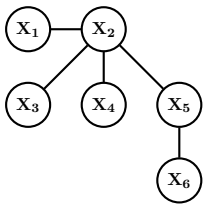


Figure 5: An Undirected Graph

As before, we can read the conditional independence relations from the graph.

(a) Following the example above, write a factorization of the joint distribution: $P(X_1, X_2, X_3, X_4, X_5, X_6)$.

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

(c) If the factorization is unique, explain 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 `code.zip`. **You must submit source code (.py 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. We will include the libraries specific in `requirements.txt` but nothing else.

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 (code.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 Piazza: https://piazza.com/class/it1vketjjo71l1.