Homework Assignment 3

Due: February 27, 2018 at 12:01 am

Total points: 100

Deliverables: hw3.pdf containing typeset solutions to Problems 1.

Source code containing your implementation for Problem 2.

README file explaining how to compile and run the source code on Linux or Windows.

Guidelines: All files must be submitted by Dropbox. You can brainstorm with others, but please solve the problems and write up the answers and code by yourself. You may use textbooks (Koller & Friedman, Russell & Norvig, etc.), lecture notes, and standard programming references (e.g., online Java API documentation). Please do NOT use any other resources or references (e.g., example code, online problem solutions, etc.) without asking.

1. Inference in Graphical Models

Consider the Bayesian network shown in Figure 1, which illustrates the influence of genetics and environment on an individual's overall health. Specifically, this model encodes a set of independence relationships among the following variables: Genetics (G), Environment (E), Health (H), Disease1 (D_1), Disease2 (D_2), Disease3 (D_3), Symptom1 (S_1), Symptom2 (S_2). Use this model to answer the questions below.

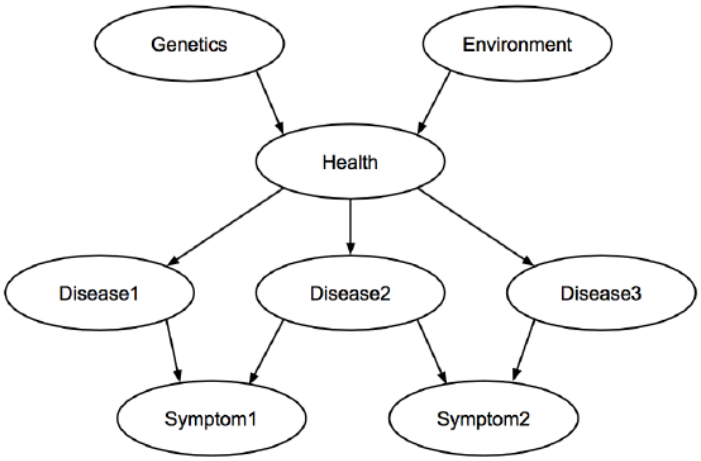


Figure 1: A Bayesian network that represents a joint distribution over the variables Genetics, Environment, Health, Disease1, Disease2, Disease3, Sympotom1, and Symptom2.

1.1 Variable Elimination in Bayesian Networks

1.1.1 (*2 points*) Write down an expression for $P(S_1)$ in terms of the un-factorized joint distribution.

1.1.2 (*3 points*) Starting with the expression you gave in the last question, write down an expression for $P(S_1)$ in terms of the factorized joint distribution and simplify it by pushing summations into the product of terms whenever possible.

1.1.3 (*2 points*) What variable elimination ordering does the resulting expression correspond to?

1.1.4 (*6 points*) Walk through an execution of the variable elimination algorithm for evaluating $P(S_1)$ using the ordering you specified in the last question. Start by writing out the initial set of factors and drawing the moralized graph. Then go through each step of the elimination algorithm and (1) specify the intermediate factors that are generated by the product and sum operations, in that order, (2) write down the new set of factors that remain after the entire step has been executed, and (3) draw the new graph induced by eliminating the variable. For clarity, please use $\phi(\cdot)$ to denote initial factors, $\psi(\cdot)$ to denote intermediate factors generated by a product operation, and $\tau(\cdot)$ to denote intermediate factors generated by a sum operation.

1.1.5 (*3 points*) Assuming each variable can take m values, what is the size of the largest intermediate factor generated during this execution of the variable elimination algorithm, i.e., how many entries does this factor have?

1.1.6 (*2 points*) What is the computational complexity (in terms of the number of possible assignments, m, and the number of variables in the network, n) of this execution of variable elimination?

1.1.7 (*2 points*) Can you do better than this? Specifically, is there another elimination ordering that yields a lower-complexity algorithm? If so, give the best such ordering and evaluate its computational complexity.

1.2 Variable Elimination with Clique Trees

A *clique tree*, also known as a *junction tree*, is a very useful data structure that serves as a "graphical flowchart" of the factor manipulation process for the variable elimination algorithm (and, as we will later see, for the message passing algorithm). We define a clique tree τ for a set of factors Φ over variables χ as an undirected graph whose nodes i are each associated with a cluster $C_i \subseteq \chi$ and whose edges ij are each associated with a sepset $S_{ij} = C_i \backslash \cap C_j$. Clique trees must satisfy two important properties: *family preservation* and *running intersection*. Before doing this question, please carefully read Section 10.1 of Koller and Friedman so that you obtain a good understanding of what a clique tree is and how it plays a role in variable elimination.

1.2.1 (*5 points*) Draw the best clique tree for the Bayesian network of Figure 1, i.e. the one induced by the variable elimination ordering with the lowest computational complexity. What is the size of

the largest intermediate factor generated in the corresponding execution of variable elimination? What is the computational complexity of this variable elimination run?

1.2.2 (*5 points*) Draw the worst clique tree for the Bayesian network of Figure 1, i.e. the one induced by the variable elimination ordering with the highest computational complexity. What is the size of the largest intermediate factor generated in the corresponding execution of variable elimination? What is the computational complexity of this variable elimination run?

1.3 From Variable Elimination to Message Passing

Before doing this question, please review the lecture notes and/or read Section 10.2 of Koller Friedman, paying careful attention to Section 10.2.2, so that you understand how to perform sum-product message passing using clique tree. Note that the sum-product algorithm described in the textbook is equivalent to the junction tree algorithm with Shafer-Shenoy updates introduced in class. From now on, we will simply refer to this as sum-product message passing.

Suppose you design a clique tree τ for the Bayesian network of Figure 1 and run sum-product message passing to calibrate τ. Denote the calibrated beliefs over each cluster as $\beta_i(C_i)$ and the calibrated beliefs over each sepset as $\mu_{ij}(S_{ij})$.

1.3.1 (*2 points*) What exactly is a "message" in the sum-product message passing algorithm?

1.3.2 (*2 points*) Assume there are N cliques in τ. How many messages must be sent to fully calibrate the clique tree?

1.3.3 (*3 points*) Give an expression for the full joint distribution in terms of the cluster and/or sepset beliefs.

1.3.4 (*3 points*) Give an expression for $P(S_1)$ in terms of the cluster and/or sepset beliefs.

1.4 Generalized Belief Propagation

In Generalized Belief Propagation (GBP) we pass messages between clusters of nodes, rather than individual nodes, which can lead to better approximations. For this question, refer to Koller & Friedman Section 10.3.

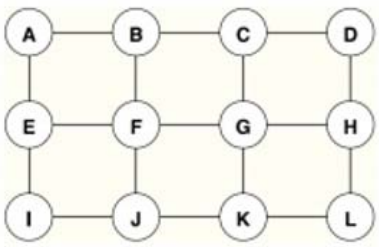


Figure 2: Markov Network for Generalized Belief Propagation

1.4.1 (*2 points*) Draw the region graph for the undirected model in Figure 2, assuming overlapping clusters of four nodes. (*Hint*: see Figure 10.12)

1.4.2 (*3 points*) Assume that this pairwise Markov Random Fields has node potentials ϕ_a for all $\alpha \in \{A, B, \dots, L\}$, and edge potentials ψ_{ab} for all $(a, b) \in E$, the edge set of the model. Write down the belief equations for $\beta[G], \beta[CG], \beta[BCFG]$. These equations should be in terms of node potentials, edge potentials, and messages from regions to their subregions. (*Hint*: use Eq. 10.36)

1.4.3 (*2 points*) Write down the message sent from region CG to region G. (*Hint*: use Eq. 10.37)

1.4.4 (*3 points*) Use the belief equation you derived in 1.4.2 as well as the marginalization consistency condition for beliefs (if $r \to r'$ then $\sum_{C_r - C_{r'}} \beta_r[C_r] = \beta_{r'}[C_{r'}]$, to derive the message sent from region CG to region G.

2 Semi-supervised Image Segmentation with Loopy BP

(*50 points*) Given an image of $l \times w$ pixels a K-ary segmentation is a clustering that assigns each pixel to one of K-classes, typically under the assumption that neighboring pixels are more likely to belong to the same class.

In this question, you will implement an application of Loopy BP to the problem of interactive (or semi-supervised) image segmentation, where the goal is to produce a binary segmentation given user-provided scribbles or strokes on an image. Consider the image shown in figure 3(a). The user (let's call him John) wants to cut himself out of this picture and paste it against a different background (he prefers mountains to beaches). Unfortunately, John is also lazy, and is only willing to use a coarse paint-like interface and give us foreground/background labels for a few pixels. In figure 3(b), the green scribble corresponds to foreground and the blue scribble corresponds to background. The most common graphical model approach for image segmentation represents the image as a grid-graph pairwise Markov random field (Figure 4) where each node corresponds to a pixel. Note that the value of a node is the cluster it belongs to. Formally, the observed image is denoted $Y = \{Y_i\}$ and $X = \{X_i\}, X_i \in \{1, \dots, K\}$ is the segmentation. The Markov random field has distribution

$$P(X, Y) = \frac{1}{Z} \prod_i \Phi(x_i, y_i) \prod_{(i,j) \in E} \Psi(x_i, x_j),$$

Where Φ is the node potential (also called the observation model or likelihood), the effect that pixel y_i has on the label of x_i; Ψ is the edge potential, how the label of x_i is influenced by the labels of its neighbors.

(a) Input image (b) Scribbles

Figure 3: Interactive image segmentation

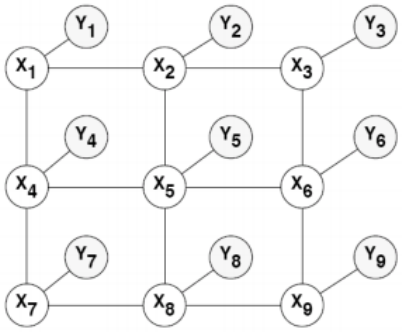
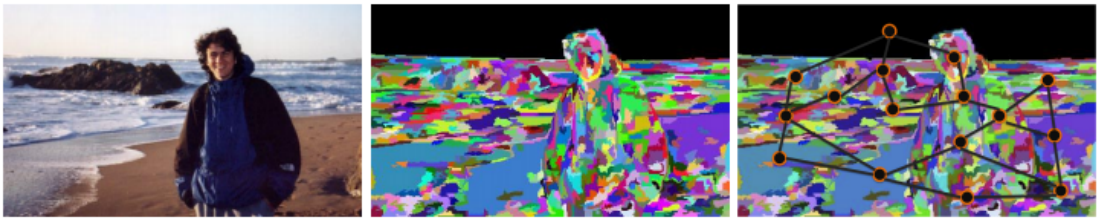


Figure 4: An example of a Markov Random Field for image segmentation

One common problem with this model is that even a (relatively) small image of size 300×300 pixels would contain 9×10^4 nodes in the MRF, and the size of the adjacency matrix for this graph would be 81×10^8 (although it would be sparse). A commonly used trick to get around this problem is to "over-segment" the image into small segments (called superpixels), and then construct an MRF where the nodes correspond to superpixels instead of pixels. All the pixels in this superpixel image which have the same color belong to the same superpixel, and will have the same foreground/background label because our MRF will assign labels on superpixels. Figure 5(b) shows a visualization of these superpixels where all the superpixels have been filled with random colours. Figure 5(c) shows what the MRF structure over superpixels would look like, where the nodes correspond to superpixels, and neighbouring (adjoining) superpixels have been connected by an edge. Note: The figure is a simplified visualization, and your resultant graph will have a lot more nodes and edges.



(a) Input image (b) Superpixels (c) Neighbourhood graph

Figure 5: MRF on superpixels

2.1 Segmentation

2.1.1 We have provided you with a scribble mask image (where a pixel has value 1 when it lies on the foreground scribble, 2 when it lies on the background scribble, and 0 otherwise). You will use the foreground and background pixels to learn Gaussian Mixture Models for each class. Set the number of components in each GMM to be 5. Your features for pixels will be Luv color vectors. For GMMs you may use your implementation from HW1 or external implementations. A Matlab implementation is available here: http://www.it.lut.fi/project/gmmbayes/
Consider looking at the function *gmmb_create*. Report the mean vectors (in the form of two 3 × 5 tables) for the foreground and background GMMs in your writeup.

2.1.2 We have provided you with a superpixel map, where each pixel holds the index of the superpixel it belongs to. Write a function called *convert_sp_label_ to_adj_mat.m* that takes in this matrix of superpixel labels and returns the adjacency matrix for the superpixel neighborhood graph. If the image has been broken into S superpixels, this adjacency matrix will be of size S × S and will contain 1 whenever two superpixels are adjacent in the image (and 0 otherwise).

Hint: Consider scanning rows and columns of the superpixel map to find transitions.
Display the adjacency matrix using the function *imagesc* and include a snapshot in your writeup.

2.1.3 Your features at superpixels (Y_i) will be the average colour (Luv vectors) of contained pixels. The node potential will be defined as the likelihood of these feature vectors under foreground and background GMMs:

$$\Phi(X_i = \text{fg}, y_i) = P(y_i|\text{GMM}_{\text{fg}})$$
$$\Phi(X_i = \text{bg}, y_i) = P(y_i|\text{GMM}_{\text{bg}})$$

Consider looking at the function $gmmb\_pdf$.
The edge potential (also called the Potts Model) will be defined as

$$\Psi(x_i, x_j) = \exp\{-\beta \times I(x_i \neq x_j)\},$$

where I is an indicator function.

2.1.4 Write a function *lbp.m* that takes as input the graph structure (adjacency matrix), the node potentials, and the edge potentials, and returns the MAP estimates of the states of the nodes via (sum-product) Loopy Belief Propagation. Initialize $m_{ij}(x_i)$ for all $i \neq j$. Stop running loopy belief propagation once the maximum absolute difference between an old message and a new message is less than 10^{-5}. Remember to compute the messages in log-space, for numerical stability. Also remember to normalize messages as listed on the lecture slides. You do not need to dampen messages to ensure convergence on this image.

2.1.5 Vary β in the range [0, 10] (take steps of size 2) and propagate superpixel labels back to pixel-level foreground/background segmentations (where all pixels in the image have been assigned to either foreground (1) or background (0)). Include plots of these segmentations in your writeup. Comment on the behavior as β increases. Why is $\beta = 0$ special? What would happen as $\beta \to \infty$?