10708 Graphical Models: Homework 2

Due October 15th, beginning of class

October 1, 2008

Instructions: There are six questions on this assignment. Each question has the name of one of the TAs beside it, to whom you should direct any inquiries regarding the question. Please submit your homework in two parts, one for each TA. Also, please put each TA's questions in the same order they appeared in the homework. The last problem involves coding, which should be done in MATLAB. Do *not* attach your code to the writeup. Instead, copy your implementation to

`/afs/andrew.cmu.edu/course/10/708/Submit/your_andrew_id/HW2`

Refer to the web page for policies regarding collaboration, due dates, and extensions.

1 I-equivalence [20 pts] [Amr]

Let $\mathcal{G}_1$ and $\mathcal{G}_2$ be two graphs over $\mathcal{X}$. In this question we will explore when $\mathcal{G}_1$ and $\mathcal{G}_2$ are I-equivalent.

1. [3 pts] Prove that two network structures $\mathcal{G}_1$ and $\mathcal{G}_2$ are I-equivalent if the following two conditions hold:

   (a) The two graphs have the same set of trails, and

   (b) A trail is active in $\mathcal{G}_1$ iff it is active in $\mathcal{G}_2$.

   (*Hint:* Use the notion of d-separation.)

2. [3 pts] Prove that if $\mathcal{G}_1$ and $\mathcal{G}_2$ have the same skeleton and the same set of v-structures then they are I-equivalent.(*Hint:* use the result from part 1)

3. [2 pts] Can part 2 be extended to an if and only if statement? If so, prove the other direction. If not, provide an example of two I-equivalent graphs $\mathcal{G}_1$ and $\mathcal{G}_2$ that have the same skeleton, but different v-structures.

Your answers to the above questions should convince you that same v-structures, although sufficient, are not necessary for I-equivalence. In the following parts, you will provide a condition that precisely relates I-equivalence and similarity of network structures. We begin with a few definitions you will need:

Definition 1 (Minimal Active Trail) *Consider an active trail $T = X_1, X_2, \ldots, X_m$. We call this active trail* minimal *if no subset of the nodes in T forms an active trail between X_1 and X_m. In other words, T is minimal if no other active trail between X_1 and X_m "shortcuts" any of the nodes in T.*

Definition 2 (Triangle) *Consider a trail $T = X_1, X_2, \ldots, X_m$. We call any three consecutive nodes in the trail a* triangle *if their undirected skeleton is fully connected (i.e., forms a 3-clique). In other words, X_{i-1}, X_i, X_{i+1} form a triangle if we have $X_{i-1} \rightleftharpoons X_i \rightleftharpoons X_{i+1}$ and $X_{i-1} \rightleftharpoons X_{i+1}$.*

4. [3 pts] Prove that the only possible triangle in a minimal active trail is one where $X_{i-1} \leftarrow X_i \rightarrow X_{i+1}$, with an edge between X_{i-1} and X_{i+1}, and where either X_{i-1} or X_{i+1} is the center of a v-structure in the trail. (*Hint:* prove by cases.)

5. [4 pts] Consider two networks $\mathcal{G}_1$ and $\mathcal{G}_2$ that have the same skeleton and same immoralities. Prove, using the notion of minimal active trail, that $\mathcal{G}_1$ and $\mathcal{G}_2$ imply precisely the same conditional independence assumptions, i.e., that if X and Y are d-separated given $\mathbf{Z}$ in $\mathcal{G}_1$, then X and Y are also d-separated given $\mathbf{Z}$ in $\mathcal{G}_2$. (*Hint:* prove by contradiction.)

6. [5 pts] Finally, prove that two networks $\mathcal{G}_1$ and $\mathcal{G}_2$ that induce the same conditional independence assumptions must have the same skeleton and the same immoralities. (*Hint:* prove by contradiction.)

2 P-MAPS, minimal I-maps and PDAGs [10 points] [Amr]

In this question we will analyze what happens when you apply the PDAG learning algorithm you implemented in HW1 on a distribution that does not have a P-map. In this problem, use $d = n - 2$.

Consider a probability distribution $\mathcal{P}$ over 4 variables, (X_1, Y_1, X_2, Y_2), that entails the following and only the following independence assertions: $X_1 \perp X_2 | Y_1, Y_2$ and $Y_1 \perp Y_2 | X_1, X_2$. From class we know that this distribution does not have a P-map.

1. [1 pts] Draw the skeleton and final P-DAG resulting from applying the PDAG learning algorithm, from HW1, using the above independence assertions.

2. [2 pts] Given a P-DAG, we can obtain a DAG (i.e. a Bayesian network) consistent with it by repeating the following two steps until all edges are directed: 1) Randomly directing an undirected edge and 2) Propagating the constraints enforced by the new

directed edge to avoid creating extra immoralities or cycles. How many DAGs can you obtain from the skeleton learnt in part 1, if any? Either enumerate all of them or explain why you can not obtain any DAG.

3. [2 pts] While $\mathcal{P}$ does not have a P-map, it still has minimal I-map(s). Draw the minimal I-map for $\mathcal{P}$ using each of the following orders of adding variables to the graph:
   - X_1, Y_1, Y_2, X_2
   - X_1, Y_1, X_2, Y_2

   Are these minimal I-maps I-equivalent, and why?
4. [1 pts] What is the relationship between the skeleton obtained in 2.1 and the underlying skeletons of the DAGs obtained in 2.3?
5. [4 pts] Prove the following statement or provide a counter example: an edge $W - V$ will appear in the skeleton produced by *build-skeleton* if and only if $W \leftrightharpoons V$ appears in all minimal I-maps of $\mathcal{P}$. (*Hint*: each minimal I-map is associated with an ordering of adding variables to the graph)

3 Decomposable Scores [10 pts] [Dhruv]

Decomposable scoring functions are those where the score of a network given data $\mathcal{D}$ can be represented as the sum of scores of each node given its parents and the data:

$$\text{score}(\mathcal{G} : \mathcal{D}) = \sum_i \text{FamScore}(X_i | \mathbf{Pa}_i^{\mathcal{G}} : \mathcal{D})$$

In greedy structure search we explore the space of structures by applying a local operator to an existing Bayes net. Examples of local operators include adding an edge, deleting an edge, and reversing an edge. In this question you will show that if the scoring function is decomposable, then computing the change in score caused by a local operator can be computed efficiently.

1. [5 pts] Prove proposition 17.4.6 (Koller & Friedman)
2. [5 pts] Prove proposition 17.4.7 (Koller & Friedman)

4 Learning Edge Directions [15 pts] [Amr]

In this question, we consider a simpler form of structure learning for BNs: Assume we have a skeleton and want to build a BN from it. For each edge, we want to either assign a direction to this edge or delete it from the graph. For this problem, you can assume you are using some decomposable score, $\text{FamScore}(X_i | \mathbf{Pa}_{X_i})$.

1. [2 pts] Consider the skeleton $X_1 - -X_2 - -X_3$, what are the possible BNs that we are considering in this problem? What is the score of each of the graphs?

2. [3 pts] Now, consider the skeleton $X_1 - -X_2 - -X_3 - -X_4$. Does the decision about the edge $X_1 - -X_2$ affect the family score of X_3? Justify your answer.

3. [10 pts] Using the intuitions above, design a linear time dynamic programming algorithm for finding the optimal BN from a chain skeleton $X_1 - -X_2 - -X_3 - - \cdots - -X_n$.

5 Greedy Structure Search [10 pts] [Amr]

Suppose we have a general network structure search algorithm, A, that takes a set of basic operators on network structures as a parameter. This set of operators defines the search space for A, as it defines the candidate network structures that are the "immediate successors" of any current candidate network structure, *i.e.*, the successor states of any state reached in the search. Thus, for example, if the set of operators is [add an edge not currently in the network], then the successor states of any candidate network $\mathcal{G}$ is the set of structures obtained by adding a single edge anywhere in $\mathcal{G}$ (so long as acyclicity is maintained).
Given a set of operators, A does a simple greedy search over the set of network structures, starting from the empty network (no edges), using the BIC scoring function. Now, consider two sets of operators we can use in A. Let $A_{[add]}$ be A using the set of operations [add an edge not currently in the network], and let $A_{[add,delete]}$ be A using the set of operations [add an edge not currently in the network, delete an edge currently in the network].

1. [5 pts] Show a distribution where, regardless of the amount of data in our training set (*i.e.*, even with infinitely many samples), the answer produced by $A_{[add]}$ is worse (*i.e.*, has a lower BIC score) than the answer produced by $A_{[add,delete]}$. (It's easiest to represent your true distribution in the form of a Bayesian network, *i.e.*, a network from which sample data is generated.)

2. [5 pts] Show a distribution where, regardless of the amount of data in our training set, $A_{[add,delete]}$ will converge to a local maximum. In other words, the answer returned by the algorithm has a lower score than the optimal (highest-scoring) network. What can we conclude about the ability of our algorithm to find the optimal structure?

Note: In 5.1 and 5.2, we are looking for *a full specification* of each distribution in terms of the graph structure and the local CPTs. Please also use your *fully* specified distributions to justify the behavior in 5.1 and 5.2.

6 Tree-Augmented Naïve Bayes [35 pts] [Dhruv]

In many classification tasks naïve Bayes is either competitive with, or is, the best method, even though, naïve Bayes ignores dependencies between features. This seems to be an argument against structure learning, until one realizes that most structure learning methods are trying to model the joint distribution, which does not necessarily corresponds to a good estimate of the class-conditional distribution.

Tree-Augmented Naïve Bayes (TAN) is a model augments naïve Bayes by adding correlations between features such that each feature has the class node, and one other feature, as parents. Figure 2 is an example of a TAN model. The advantage of TAN over NB is that NB assumes full conditional independence for the features, which can hurt in some cases by double counting information. A TAN model can reduce some double counting problems by modeling correlation between features.

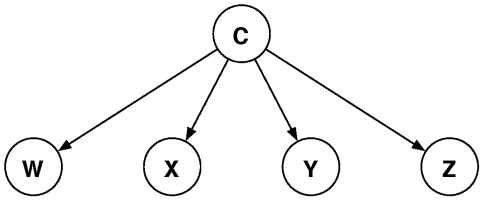

Figure 1: An example of naïve Bayes.

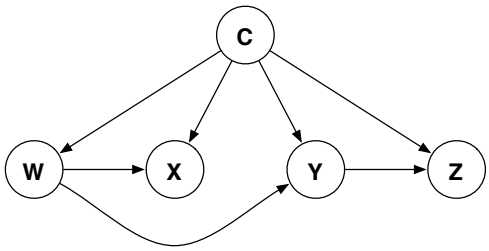

Figure 2: An example of tree-augmented naïve Bayes. Note that the induced graph on the evidence variables (w, x, y, z) forms a tree.

The algorithm for learning TAN models is a variant of the Chow-Liu algorithm for learning tree-structured Bayes nets. Let C represent the class variable, and $\{X_i\}_{i=1}^n$ be the features (non-class variables).

1. Compute the conditional mutual information given C between each pair of distinct variables,

$$I(X_i; X_j|C) = \sum_{x_i,x_j,c} \tilde{P}(x_i, x_j, c) \log \frac{\tilde{P}(x_i, x_j|c)}{\tilde{P}(x_i|c)\tilde{P}(x_j|c)}$$

   where $\tilde{P}(\cdot)$ is an empirical distribution (computed using the training data). Intuitively,

this quantity represents the gain in information of adding X_i as a parent of X_j given that C is already a parent of X_j.

2. Build a complete undirected graph on the features $X_1, \ldots, X_n$ where the weight of the edge between X_i and X_j is $I(X_i; X_j|C)$. Call this graph $\mathcal{G}_F$.
3. Find a maximum weighted spanning tree[1] on $\mathcal{G}_F$. Call it $\mathcal{T}_F$.
4. Pick an arbitrary node in $\mathcal{T}_F$ as the root, and set the direction of all the edges in $\mathcal{T}_F$ to be outward from the root. Call the directed tree $\mathcal{T}'_F$. (*Hint:* Use DFS).
5. The structure of the TAN model consists of a naïve Bayes model on $C, X_1, \ldots, X_n$ augmented by the edges in $\mathcal{T}'_F$.

The task is to learn a boolean function, classifying input features as class 0 or 1. The data can be found in `corral.mat`, where each row is an instance containing six boolean features, and the last column is the class. The data is generated in `gen_corral_data.m`, if you are interested in knowing the true mapping.

6.1 Structure Learning [20 pts]

Implement the above algorithm for learning the structure of a TAN model, and submit your code as `tanstruct.m`. Using the corral data, draw the structure (directed acyclic graph) produced using this algorithm in your writeup.

6.2 Representation

In this part you will implement the representation of a general Bayesian network. Using this representation, you will learn parameters of this model, and perform a simple inference step[2] to perform classification in the next question.

Hint: The steps you should take in implementing this are as follows:

1. A data structure to represent a *factor*, a mapping from an assignment of variables to a real value. Conditional probability tables can be viewed as factors. For example, in figure 1, the conditional probability table for W would map the assignment $(\mathrm{W} = 1,\ \mathrm{C} = 1)$ to some value c. The easiest way to encode a factor is as a multidimensional array where each dimension corresponds to a variable. See `table_factor.m`.
2. A data structure to represent a Bayesian network. The easiest way to do this is just to store a list of all the conditional probability tables as factors.

[1] Kruskal's or Prim's algorithm can be used to find a maximum weighted spanning tree. It is okay to use external implementations, but only for finding the maximum spanning tree.

[2] Do not implement variable elimination.

3. A data structure that represent an assignment to variables. The easiest way to do this is as a pair of vectors: $vars$ and $vals$. Note that $vals(i)$ is the value assigned to variable $vars(i)$. See `assignment.m`.

4. A function (`assign_prob.m`) that takes a Bayesian network and an assignment to all the variables, and returns the probability of that assignment.

Note: You will not receive full marks for an implementation that stores the full joint distribution explicitly.

Note 2: There is nothing to report in this part. You will use (and upload) this code as part of the next question.

6.3 Learning and Classification [15 pts]

In this question you will compare the classification accuracy of naïve Bayes and TAN. Perform a variant of leave-one-out cross-validation (LOOCV), that is, hold out one instance, learn the structure and parameters from a random subset (of size m) of the rest, and then classify this datapoint.

1. Learn the structure of a TAN model and estimate the parameters using the following smoothing estimator. For the parameter corresponding to $P(x_i|\mathbf{Pa}_i)$ estimate it using

$$\theta_{x_i|\mathbf{Pa}_i} = \alpha\tilde{P}(x_i|\mathbf{Pa}_i) + (1-\alpha)\tilde{P}(x_i)$$
$$\alpha = \frac{m\tilde{P}(\mathbf{Pa}_i)}{m\tilde{P}(\mathbf{Pa}_i) + s}$$

   where s is a smoothing parmeter. For this question, use $s = 5$. This is known as back-off smoothing.

2. Learn a naïve Bayes model and estimate the parameters using back-off smoothing.

3. Compare the classification accuracy of naïve Bayes and TAN on the test set.

   Note: In order to classify, you will need to compute $P(C|X_1, X_2, ..., X_n)$ where $\{X_i\}$ are the features. We will develop machinery for general inference queries in the next homework. For this question, you should use Bayes rule, and `assign_prob.m` to get the answer to this specific query. Do NOT implement any general inference algorithm like variable elimination.

Plot the classification accuracy (for the two models) as a function of m, the size of the random subset of training data (vary m from 10 – 63). Submit the code used to run these experiments as `tancompare.m`. Comment on the trend observed in the plot. Specifically, do you expect TAN to always (meaning, on all datasets) outperform NB?