Final Report
of
Project in Probabilistic Models

Haibo Jin
ID: 014343698

University of Helsinki
Department of Computer Science
Master student

Contents

1 Introduction

The goal of the project is to infer the structure of a Bayesian network as well as its distributions. There are 26 variables in the network all with 3 discrete values. Two separate data is given, namely training data and testing data. Training data is for structure learning and parameter estimation while testing data is for probability prediction. Figure 1 shows an example of the training and testing data. Two files of outputs are required for the task: 1.A ranked list of all possible arcs of the model; 2.A normalised probability distribution over 1500 vectors in the testing data.

```
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
3 1 1 1 3 2 3 1 3 1 3 1 3 3 3 3 2 1 2 1 1 1 1 3 2 1
3 1 2 3 1 3 3 1 1 2 2 3 3 3 3 1 1 1 2 1 3 3 2 2 1 1
3 3 1 1 3 2 3 1 3 1 3 2 3 2 2 3 2 2 2 2 3 3 2 3 2 1
3 1 1 2 3 2 2 1 2 1 2 1 3 1 3 2 3 1 2 1 3 3 3 1 1 1
2 1 1 3 1 3 3 1 3 2 2 2 3 3 3 3 3 1 2 2 3 3 1 2 1 1
3 1 2 3 1 2 3 1 1 3 2 1 3 3 1 2 2 1 2 1 1 3 1 2 1 1
...
```

Figure 1: An example of the training and testing data.

The programming language is Octave for all the implementations.

2 Methods

In order to get the two output files, the task can be divided into two subtasks, one is structure learning and the other is parameter estimation. Further more, structure learning consists of two parts: a scoring function and a search strategy.

2.1 Parameter Estimation

For the parameter estimation task, I use the Bayesian parameter learning method in all of the implementations. The method evolves Dirichlet distribution to express our biased preferences over observed data. A parameter of a variable i (i.e. conditional probability of variable i), can be calculated as follows:

$$\theta_{ijk} = \frac{\alpha_{ijk} + n_{ijk}}{\sum_k (\alpha_{ijk} + n_{ijk})} \tag{1}$$

where j represents values of the parents of variable i and k represents value of i itself. n_{ijk} is the count of the observed data while α_{ijk} is the bias value. To conveniently set the bias values for all the combinations, we can just set a value of α and use it to calculate α_{ijk} through the following equation:

$$\alpha_{ijk} = \frac{\alpha}{r_i \cdot q_i} \tag{2}$$

In all the implementations, the α is set to be 100. More details of the α value will be discussed later.

2.2 Scoring Function

I use the BDeu scoring function to evaluate the goodness of a structure. The following equation computes the BDeu score of a structure based on the data:

$$BDeu(N:D,\alpha) = \sum_{i}^{n}\{\sum_{j}^{q_i} log\tau(\frac{\alpha}{q_i}) - log\tau(\frac{\alpha}{q_i}+n_{ij}) + \sum_{k}^{r_i} log\tau(\frac{\alpha}{r_i \cdot q_i}+n_{ijk}) - log\tau(\frac{\alpha}{r_i \cdot q_i})\} \tag{3}$$

in which α is set to 1 in the implementations.

2.3 Search Strategy

All of my search strategies are based on greedy hill climbing algorithm. It is a heuristic searching algorithm, picking a best operator in each iteration to improve the BDeu score until there exists no improvement. There are three operators in total: arc addition, arc deletion and arc reversal. After each iteration, the whole network is supposed to be a directed acyclic graph (DAG), otherwise, the operator will not be executed.

2.3.1 Strategy 1

Considering that counting the combinations of values of a variable and its parents in the data takes too much time, I restrict the number of parents under 10. This restriction is also reasonable since most variables do not have that many parents in a real network. Additionally, too many arcs also slow down the efficiency of the algorithm. Thus, the initial network is set to empty in this strategy, in other words, there is no arc initially.

2.3.2 Strategy 2

However, the result of strategy 1 is deterministic, which is not sustainable. To start from a random graph with a not bad efficiency, I make the initial random arcs sparse. The rest part remains the same as strategy 1.

2.3.3 Strategy 3

Strategy 2 is yet too naive and unreliable, since the result comes from one random running. It is natural to run more times and pick the best one, just as the original greedy hill climbing algorithm. But in order to explore greedy hill climbing in some other ways, I run it for 15 times and accumulate the counts

of each arcs instead of picking the best one. The reason for doing this is that I think a local optima also contributes some knowledge about the real structure, so we can make use of such knowledge rather than just dropping it. Then we can get a ranked list of all the arcs, in which arcs with more counts ranked higher. According to the previous results, we know the number of arcs is around 45, so the first 45 arcs is chosen to construct the network (also need to make sure it is a DAG, if an arc makes it not a DAG, then it will not be added).

2.3.4 Strategy 4

In probability prediction, each variable is actually computed independently, thus, whether a network is a DAG or not does not affect the prediction. Even though the real network is a DAG, we are just approximating the real structure, we just want to get a more precise structure, no matter if it is a DAG. Thus, based on the ranked arcs of strategy 3, I choose the first 51 arcs to construct the network, which gets the highest BDeu score over the ranked arcs.

2.3.5 Strategy 5

Strategy 5 is still based on the ranked arcs of strategy 3. Since we have got some arcs that are more likely to be the real ones, we can narrow the search space by focusing on these higher ranked arcs. That is to say, I pick the first 100 arcs in the ranked list, and set them to be the valid arcs. Then, we can start a greedy hill climbing search just like strategy 1, but only consider the three operators on these valid arcs.

3 Results

The results of strategy 1 to strategy 5 is listed in Table 1. A larger ROC curve means a better structure and a smaller KL divergence means a better probability prediction.

	ROC curve	KL divergence
Strategy 1	0.890	1.653
Strategy 2	0.929	1.653
Strategy 3	0.938	1.667
Strategy 4	0.938	1.650
Strategy 5	0.856	1.653

Table 1: Results of strategy 1 to strategy 5. ROC curve shows the correctness of structure and KL divergence shows the correctness of probability prediction.

4 Discussion

As we can see from the results, a simple implementation of greedy hill climbing (strategy 1) can get a satisfying result, which shows the effectiveness of greedy hill climbing. In terms of the ROC curves, there is a progress compared to previous strategies, which implies the effectiveness of the improvements on structure learning approaches. However, the probability predictions do not improve much and one of them is even worse. After analysing the results, I think there could be two reasons: Firstly, a good ROC curve does not necessarily gives a good structure. For example, strategy 4 and 5 get good ROC curves, but we need one more step, selecting most likely arcs to construct a network. In other words, if we cannot estimate the optimal structure over the ranked arcs, then we lose some information that we get for learning structures. Secondly, we should notice that the bias value α in parameter estimation has always been 100, which may seem too large compared to the total data size. It is reasonable to think that such a large bias value gives too much smoothing so that a slight change of the structure cannot make much difference to parameter estimations. Based on this assumption, we can see that strategy 4 gives the best result. By setting a proper value of α, the result of strategy 4 may get better to some extent, though it only gets a slightly better result compared to other strategies currently.

Additionally, my explorations mostly focus on the improvement of search strategies. Since the estimations are basically based on the data, some techniques, such as bootstrapping, can be used to generate more data for a more precise estimation.

Due to the limitation of time, I cannot make more attempts on the algorithm, especially the search strategy. For future work, I think I will focus on the reusing of the ranked arcs for constructing a network. According to the results, as more and more local optima being accumulated (strategy 3 only accumulates 15 local optimas), we can get a quite good ranked list of arcs. Then we should find a way to optimally transform this ranked list to a good network. To sum up, the principle of my implementations is to optimally reuse the result of previous searchings. In terms of this, I think strategy 5 can get a good result, with running times getting larger, though its current performance is not that good. Strategy 5 does reuse the previous information and consequently reduce the search space in a reasonable way.

5 Diary entries

5.1 Diary 1

For the first version of structure learning, I am considering to implement a common version with greedy hill climbing being search strategy and BDeu score as evaluate score.

When in the progress of the programming, the biggest problem to me is the efficiency problem. Firstly, since I use Octave as programming language,

it may not efficient at for-loops. So I try to vectorize as many for-loops as I can. Then, I found the function of calculating BDeu score takes too much time, especially the part for getting the counts of a specific variable with its parents. Even though I have tried some methods to get some improvements, for example, using built-in function like cartprod() to calculate Cartesian products instead of programming a function myself, the efficiency is still not satisfying. So, I have to simplify the search strategy, such as skipping the calculation of the variable with over 10 parents, and set the initial graph as empty instead of a random graph.

With such modifications, now the algorithm can be used to the training data with 2500 samples. Taking about 10 minutes, the result finally came out. However, the connection of the graph is less than 50, which is too sparse. I think this is because the initial graph is empty and it gets stuck in a local optima. Thus, initial randomness is still necessary for greedy hill climbing and more work needs to be done in the future. I think I may try some other strategy to compare with this greedy hill climbing algorithm.

5.2 Diary 2

Since last time I didn't finish estimating the probabilities of the testing vectors, so I finish this part this week. I use Bayesian parameter estimation to predict the probabilities. I set the equivalent sample size as 100, about which I am not sure whether it is suitable or not. Everything seems fine and the efficiency is quite OK. I just need to see the result of the probability part to know how it is going.

About the structure learning part, I remain the same prediction as last week because of the limitation of my time. Sorry about that. Since the result of the prediction was not bad, I will keep trying greedy hill climbing algorithm next time, maybe try to extend the search space more globally.

5.3 Diary 3

Based on my previous program, I extend the search space of the algorithm by restarting the searching from a random graph for 15 times. I do not just pick one structure that gets the best score, instead, I count the times for every connection after 15 times iteration. Then I rank the connections by their times in descend order. I do this because I think we cannot promise finding the global optima using greedy hill climbing since the search space is too large. But we can get a local optima every time. Under such an assumption that a part of the local optima is the same as a part of the global optima, so we can accumulate the counts of connections of every local optima to approximate the global optima. To get a structure for calculating probabilities, I choose the first 45 arcs based on the rank of arcs.

5.4 Diary 4

The algorithm of structure learning remains the same as third extra submission.

For parameter estimation, there is only a slight change. The equivalent sample size α of Dirichlet distribution is set to be 1 in this submission. Previously, α was set to be 100. Since the performance of my third extra submission is improved, yet the progress is too small. I think maybe the α is too large so that it gives too much smoothing to the parameters. In other words, even if there is an improvement in structure, the improvement may be smoothed under a large α when estimating parameters.