Structure Learning for Bayesian Networks - Evolutionary Approaches

Suhas JS, Sai Praveen B

May 1, 2016

Abstract

Bayesian Networks are widely used to describe causality relationships between sets of variables. Determining Bayesian Network structure from sample data has been shown to be an NP-Hard problem. In this paper, we present alternative method that use evolutionary approaches to automatically learn Bayesian Network structure from sample data using Regularized Scoring methods. We propose a variation of Bayesian Information Criterion(BIC) to solve the structure learning problem. We then demonstrate the effectiveness of our approach by generating random networks and learning the structure by using samples from the said networks.

Introduction

Bayesian Networks(BN) are a family of graphical models representing a joint distribution for a set of random variables. BNs encode conditional dependancies between variables. These are generally referred to as *Conditional Probability Distributions* or *CPDs*. A BN is generally denoted by $\mathcal{B}_{\mathcal{P}}$. We say that the BN $\mathcal{B}_{\mathcal{P}}$ encodes the probability distribution $\mathcal{P}$ over the variables $\mathcal{X}$. The BN $\mathcal{B}_{\mathcal{P}}$is also sometimes represented as a *Directed Acyclic Graph (DAG)* $\mathcal{G}_{\mathcal{P}}$. We define $Pa_X^{\mathcal{G}}$ to be the set of all *parents* of node X in the network $\mathcal{G}$. Formally, $Pa_X^{\mathcal{G}} = \{Y : (Y \to X) \in E_G\}$.

The probability distribution encoded by $\mathcal{B}_{\mathcal{P}}$ is then written as $\mathcal{P}(\mathcal{X}) = \prod_i Pr\left(X_i | Pa_{X_i}^{\mathcal{G}}\right)$.

Given a set of variables $\mathcal{X}$, determining the best BN structure $(\mathcal{G})$ that can describe the *causality* relationship between the variables in $\mathcal{X}$ is an NP-Hard Problem. Once the structure is learnt, the next step is to determine the probability distribution over $\mathcal{X}$. This is done by estimating the CPDs from training data $\mathcal{T}$. The CPDs can be either Tabular CPDs (if $\mathcal{X}$ admits discrete values), Continuous CPDs (if $\mathcal{X}$ admits continous values) or combination of both. This phase usually involves parameter estimation and curve-fitting.

Evolutionary methods are a class of high dimensional optimization methods that optimize a function's value by simulating evolution artificially. We use Evolutionary methods to artificially *evolve* BNs by optimizing their *score* function. In this project, we aim to produce a set of *simple* BNs that explain a given dataset $\mathcal{T}$ with good accuracy[1]. These *simple* BNs also describe the most important of relationships and dependancies in $\mathcal{X}$.

Problem Setting and Related Work

We start out by defining the structure learning problem in Bayesian Networks as follows.

Given a dataset $\mathcal{T}$, and a set of variables $\mathcal{X}$, determine a DAG $\mathcal{G}$ s.t.
$\forall X \in \mathcal{X}$,$Pr\left(X|\mathcal{X}\right) = Pr\left(X|Pa_X^{\mathcal{G}}\right)$.

One of the most popular methods of inducing BNs from data, especially for the purpose of CPD estimation, is the *score-based* approach. The process assigns a *score* to each candidate BN, typically one that measures how well that BN describes the training dataset $\mathcal{T}$. Given a structure $\mathcal{G}$ and training dataset $\mathcal{T}$, we define the score as follows.

$$Score\left(\mathcal{G}, \mathcal{T}\right) = Pr\left(\mathcal{G}|\mathcal{T}\right)$$

It is no surprise that a fully connected graph $\mathcal{G}_c$encodes any complicated distribution between the variables. Hence, the score is maximized at $\mathcal{G} = \mathcal{G}_c$ i.e. $\forall \mathcal{G}, Score\left(\mathcal{G}, \mathcal{T}\right) \leq Score\left(\mathcal{G}_c, \mathcal{T}\right)$. Due to this behaviour of the score function, regularizing the score-based method on the basis of the network complexity is often observed to give better results. The regularized score-based methods use a penalty function $\mathcal{C}$ that penalizes the network for its complexity. The refined scoring function $Score'$ is now defined as follows.

$$Score'\left(\mathcal{G}, \mathcal{T}\right) = Score\left(\mathcal{G}, \mathcal{T}\right) - \mathcal{C}\left(\mathcal{G}\right)$$

The choice of the function $\mathcal{C}\left(\mathcal{G}\right)$ determines the complexity of the resulting solution network. A heavy penalty for complexity results in relatively simpler BNs and vice-versa.

Using constraints is another way of learning BN structure. These constraints are typically conditional independence statements. The conditional independence tests that are used in practice are statistical tests on the data set. Using these constraints, one can ascertain the existence of an edge between two variables and, sometimes, even the direction of that edge. This class of algorithms include SGS, IC ("inductive causation") etc.

Constraint-based algorithms have certain disadvantages. The most important one, observed frequently in practice, is their poor robustness or high sensitivity to inputs i.e. the structure of the BN, even for small changes of the input (like single errors in the independence tests) leads to large structural changes in the output graph.

In this project, we couple regularized score-based methods with evolutionary approaches to capture the most important of relationships between variables in $\mathcal{X}$ in the form of a sparse directed graph $\mathcal{G}$. In the initial approaches, the structure is built using a genetic algorithm and with or without the knowledge of a topologically correct order on the variables of the network. Genetic algorithms encode potential solutions to a problem in a chromosome-like data structure, exploring and exploiting the search space using dedicated operators. Throughout the years, different strategies and operators have been developed in order to perform an efficient search over the considered space of individuals: selection, mutation and crossing operators, etc. We use a heavily penalizing cost function $\mathcal{C}\left(\mathcal{G}\right)$ as we'll see later. This results in very sparse networks that capture the strongest dependancies, omitting weak ones.

We represent the graph G with N nodes as a gene of length N^2. We take the adjacency matrix for the graph G and convert it into a one-dimensional representation (row-major). We then use this representation as our *gene* for the genetic algorithm described below.

For example, if we have the following directed graph G, we can derive the *gene* for G as follows.

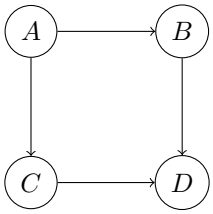

Figure 1: Sample Graph G

We can generate the adjacency matrix A^G and the gene $g_G$using the above procedure. The respective values are given in Figure 2.

$$\begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \rightarrow \begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Figure 2: Adjacency Matrix and Gene for G

Approach

We use Holland's *cannonical* genetic algorithm[2] for this project. We describe this algorithm in Algorithm 1.

Algorithm 1 Holland's Cannonical GA

Input: A Random Population $\mathcal{F}_0$ of size λ,f
Output: Population $\mathcal{F}_{T+1}$
  for t in $\{1..T\}$ **do**
    Evaluate $\mathcal{F}_t$
    Select $f\lambda$ individuals
    Apply crossover operator to generate $1 - f\lambda$ individuals
    Apply mutation operator on newly generated individuals
    $\mathcal{F}_{t+1} \leftarrow$ selected individuals $\cup$ newly generated individuals
  end for

We use a modified version of the algorithm described above. The selection operator uses $\epsilon - greedy$ strategy to select individuals.

Our approach consists of 2 major components: Bayesian Network Breeding and Bayesian Network Scoring

Breeding

First, from any given population $\mathcal{F}$ of networks(initial population is randomly selected), we shortlist $k_{accept}.\|F\|$ networks from the population to form $\mathcal{F}_{accept}$. The shortlisting is by selecting the top networks in the sequence sorted by $Score(P)$. Scoring is discussed in further sections. $k_{accept}$is a fixed

ratio. From $\mathcal{F}_{accept}$ we randomly sample pairs of networks (P_1, P_2) and create a child network C
For breeding of networks: Typically 2 suitable methods are used. Both mimic evolution in real life. In breeding, we take 2 networks $P_1$and $P_2$and create a child network C. Once the child network is created, we ensure that there are no self-edges in the final network.

Cross-overs

We represent the networks in binary format. Each network of N nodes is represented by a ${}^nC_22$ bit vector. This vector is treated as the gene corresponding to the given network.
A Cross-Over is when the parent's network gene is split into parts and recombined using a random mask M (also of the same number of bits.) The final coding for the BN is

$$C = M.P_1 + \bar{M}.P_2$$

The mask M is generally sampled randomly but usually chunks of the mask is clumped together to ensure similar units occur frequently. For our case the $chunk = N - 1$ because every block describes the influence of one node in the network(i.e the binary vector that specifies adjacency from some node to all others.)

Mutations

Under the same format, a bit can be flipped in the child C to produce C'. This occurs with probability $P_{mutation}$. The final child now looks like this:

$$C = K \oplus (M.P_1 + \bar{M}.P_2)$$

where K is the mutation vector(randomly sampled using Bernouilli trials).

Scoring

Scoring has 2 parts to it: Training of the network using the data and Finding goodness of fit.

Maximum Likelihood Training

Since the BN training is on fully observed data, we use Maximum Likelihood estimation(no priors) to find the probability distribution for each CPD $P(X|Pa_x)$. The representation is tabular and thus the fit, given the data, is exact(however, exponential in the size of the parent set for the node and therefore justifies the goodness of fit mechanism that we will be using.). Given the sample points, we have:

$$Pr(X = \varepsilon(X)|Pa_x = \varepsilon(Pa_x)) = \frac{N(X = \varepsilon(X), Pa_x = \varepsilon(Pa_x))}{\sum N(X = x, Pa_x = \varepsilon(Pa_x))}$$

Bayesian Information Criterion[3]

After much trial and error, we have decided to use a slightly modified version of BIC for a good measure of how well the resultant trained network fits the data. For this, we expect that a network that is a characteristically bad approximation to the original network will be unable to match the probabilities of the data and thus give a low score.
The metric we use uses 2 components: the Information Gain(KL-divergence) $D(P||Q)$ and a network complexity measure $k.ln(N)$ where k is number of free parameters(Table size) and N is number of samples.
The original BIC metric uses the log likelihood$log(L)$ of the system for the first term. This includes the entropy of the target distribution $H(P)$ which we is a constant for all approximations$Q$. We choose to ignore this measure, and also to arbitrarily vary the weights of each of the 2 components in order to obtain the best fit possible.

$$Fitness = D(P||Q) - \alpha_{BIC}.k.ln(N)$$

For our case, we have:

$$Fitness = N.\sum_{x}\sum_{Pa_x} Pr(x).Pr(Pa_x).log(\frac{Pr(X, Pa_x)}{Pr(X)Pr(Pa_x)}) - \alpha_{BIC}.\sum_{x} 2^{\|Pa_x\|}.ln(N)$$

We observe that in the standard BIC the emphasis increases on the first term as N grows large. This has some problems on really large datasets as they start converging to fully connected networks(The second term becomes very small because of large N). We therefore propose a simple modification to the heuristic:

$$Fitness = \sum_{x}\sum_{Pa_x} Pr(x).Pr(Pa_x).log(\frac{Pr(X, Pa_x)}{Pr(X)Pr(Pa_x)}) - \frac{\alpha_{BIC}}{2^{\|\mathcal{X}\|}}.\sum_{x} 2^{\|Pa_x\|}$$

Results

For testing the above algorithm, we used a random network of with 5 nodes and 8 edges. This random network was generated using `weka.classifiers.bayes.net.BayesNetGenerator[4]` class. We also used the same class to generate datasets of size 10^4 data points each. We then generated 100 random networks and subjected them to 100 iterations of *selection* and *breeding*. The plots shown use values averaged over 20 runs.

We first look at the network used to generate the dataset and a sample network returned by our problem.

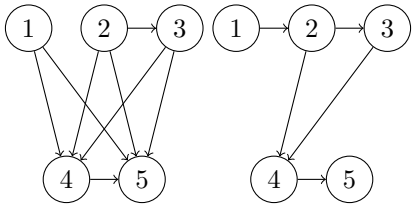

Figure 3: Generator Network $\mathcal{G}$ and Sample Solution Network $\mathcal{S}$

Notice that all edges in $\mathcal{S}$, except for $1 \to 2$, exist in $\mathcal{G}$. This network has a precision of 0.8 and recall of 0.5.

We now look at the average fitness of the individuals as a function of generation.

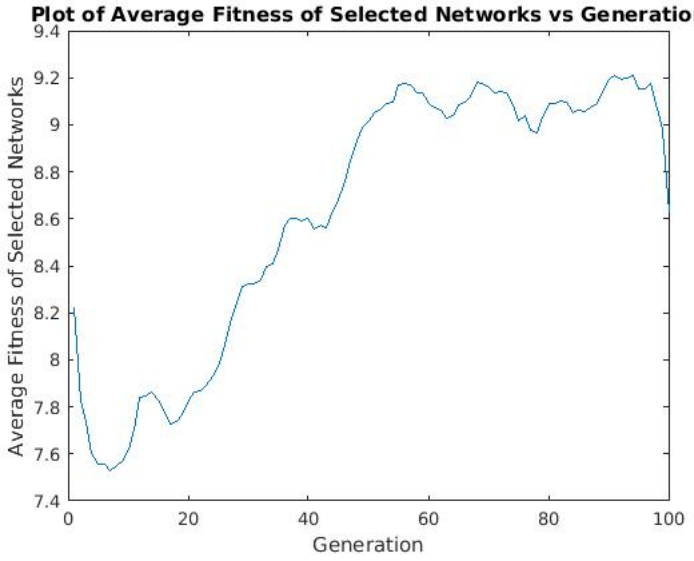

Figure 4: Average Fitness vs Generation

We observe that the average fitness of the individuals increases with generation. However, we also see the average slump at some intervals. This can be attributed to the ϵ-selection for selecting parents for next generation. Overall, we observe a trend of increasing average fitness of the networks.

We now look at the maximum fitness of the selected individuals as a function of generation.

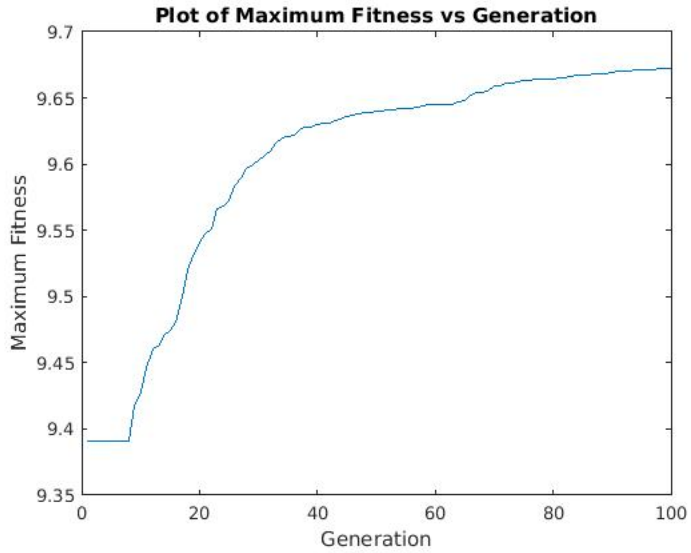

Figure 5: Maximum Fitness vs Generation

We again observe that the maximum fitness is monotonic in nature. This is guaranteed by evolutionary approaches. The fitness of the reference network (the one used to generate the dataset) was 9.90. We see that networks generated using our approach approach the fitness of the reference network. We had to restrict the number of generations to 100 for computational reasons. It's possible that maximum-fitness network has better fitness than the reference network and this was confirmed in some of the 20 runs.

Now, we study the effect of α_{BIC} on the precision and recall for the edges.

Given a reference gene G_r and another gene G_i, we define precision and recall for the pair as follows.

$$TP = \sum_k I\{G_{ik} = 1, G_{rk} = 1\}$$

$$FP = \sum_k I\{G_{ik} = 1, G_{rk} = 0\}$$

$$FN = \sum_k I\{G_{ik} = 0, G_{rk} = 0\}$$

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

where, $I\{p\}$ is the indicator function and $I\{p\} = \begin{cases} 1 & p\, is\, True \\ 0 & p\, is\, False \end{cases}$

We now observe the effect of α_{BIC} on precision and recall. The plots show dependance of average precision and recall of the 101^{st} generation networks using the reference network for calculations.

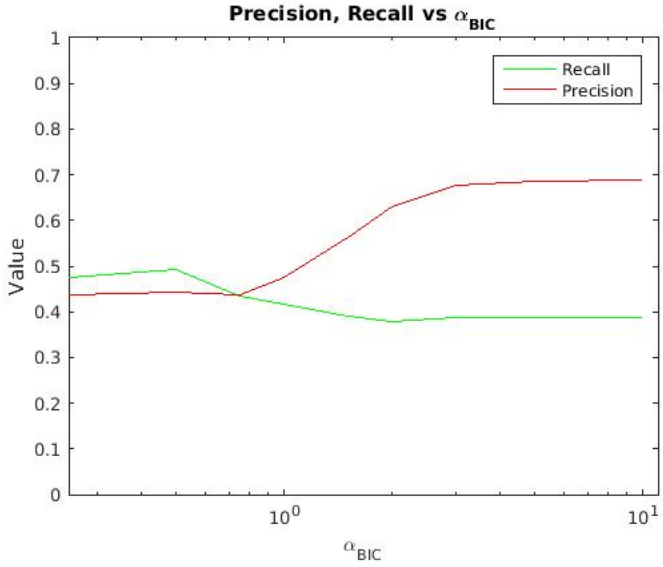

Figure 6: Precision, Recall vs α_{BIC}

This plot reveals the expected nature of the effect of α_{BIC} on precision and recall. Increasing α_{BIC} reduces the Recall, but increases the precision of the resulting networks. This is expected as a higher value for α_{BIC} puts more emphasis on complexity of the network and hence, due to *selection*, we obtain very simple networks. We also claim that the resulting networks capture all the important depedancies between the variables in $\mathcal{X}$. This can be explained using the proposed fitness function. If the networks capture the dependancies, they give have higher mutual information. Thus, by *selection*, our networks explain the training dataset very well.

Conclusion

We have proposed and implemented a *flexible* scoring function for evaluating candidate structures in solving the Bayesian Structure Learning problem. Our scoring function when coupled with evolutionary approaches seems to perform quite well in capturing all the important dependancies between variables in a variable set$\mathcal{X}$ and returns a very simple Bayesian Network. The resulting solutions also seem to have very good precision on average.

Going forward, we propose a few additions to our methods to simulate evolution realistically.

- Allow for variable population size. This doesn't necessarily lead to an exponential increase in population of organisms if some control measures are employed.
- Assign a TTL (Time To Live) for each organism. Organisms are *killed off* after they complete their TTL.

- Sample the above TTL from a distribution based on the fitness of the organism. Each organism survives at least one generation.
- Simulate an *epidemic* of sorts that *kills* organisms probabilistically with the probabilities sampled from the fitness of the organisms. This epidemic is invoked probabilistically based on the population size and population fitness.
- Explore various parameter settings by performing simulations.

Due to computational limitations, we were forced to use networks of size 5. However, precision and recall on these networks aren't very representative of the method as these networks had only 8 edges. Further work could be done exploring different parameter settings with higher network sizes (10,20 etc).

References

[1] Brouard, Thierry, Alain Delaplace, and Hubert Cardot. Evolutionary methods for learning bayesian network structures. INTECH Open Access Publisher, 2008.

[2] Holland, John H. Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence. U Michigan Press, 1975.

[3] Schwarz, Gideon. "Estimating the dimension of a model." The annals of statistics 6.2 (1978): 461-464.

[4] Holmes, Geoffrey, Andrew Donkin, and Ian H. Witten. "Weka: A machine learning workbench." Intelligent Information Systems, 1994. Proceedings of the 1994 Second Australian and New Zealand Conference on. IEEE, 1994.