Bayesian Networks Bootstraap Aggregation Greedy Search

Carlos Rodolfo Huerta Santiago s3743071

University of Groningen, P.O.Box 72 9700AB Groningen

1 Introduction

Bayesian networks is the marriage between graph theory probability and statistics, they belong to the so called family of 'probabilistic graphical models', given a set of variables (nodes) each one represents a random variable and the edged between the nodes represent the conditional probability dependencies of our nodes. More formally: Each variable X_i is represented as a node in a **DAG-directed acyclic graph**.

$$P(X_1, X_2, \ldots, X_N) = \prod_{i=1}^{N} P(X_i \mid pa(X_i)) \tag{1}$$

Where $pa(X_i)$ is the set of parents of the node X_i But how do we 'learn' to represent this conditional dependencies from data? In general learning a network structure is a NP-hard problem, so we need other learning approaches such as: **Greedy Search**. On this paper we will talk about the Greedy Search approach to learning a network from data as well as an improvement to this technique called **Bootstrap Aggregation Greedy Search** at the end we will talk about its advantages, disadvantages and limitations of the algorithm as well as a brief comparison with other useful learning approaches like the **Structure MCMC algorithm**

2 Learning a Graph Network Structure

To be able to understand how we will approach this problem we must first define some concepts on Bayesian Networks.
Consider $P(graph \mid data)$ as how much is a current graph supported by the observed data. This is also called **posterior probability of the graph**
Now the **marginal likelihood** $P(data|Graph)$ describes to which probability we observe this dataset given a particular graph structure. How well does the

graph explains the observed data.
Next we will make use of Bayes theorem to derive a way that will let us compute the **posterior probability** term.

$$P(graph|data) = \frac{P(data|graph) \cdot P(graph)}{P(data)} \tag{2}$$

Since $P(data)$ is just a normalization constant that ensures that the numerator of (2) sums to 1 to be considered a valid probability density function. Then we are going to focus on the numerator of (2). So it becomes:

$$\propto P(data|graph) \cdot P(graph) \tag{3}$$

By assuming that the variables $X_1 \dots X_n$ are multivariate Gaussian distributed with unknown parameters μ and Σ and that the unknown parameters are normal-Wishart distributed one can show that (2) reduces to the **BGe** score metric.

$$\begin{aligned} &= P(graph) \cdot \int P(data, \theta(graph)|graph) d\theta(graph) \\ &= score_{BGe}(graph|data) \end{aligned} \tag{4}$$

Now given a graph we can score on how likely it was generated by the data we have, this comes with a severe limitation as we will see in the next section.

3 Greedy Search

We have been using the BGe score metric to define the graph model 'fitness' to our data. But in order to due this we would need to score all the possible directed acyclical graphs (DAGS) that our data can form. This is a function dependent on the number of nodes in our data grows super exponentially [insert ref], so searching for all possible BGe scores would be a non-computationally feasible task. This is where the **Greedy Search** algorithm comes to play.
The main idea of the **Greedy Search** algorithm is to heuristically search for the best graph that fits the data, lets call it M^* by comparing the current graph score that we have to its neighbors.

$$P(M^*|data) \ > \ P(graph|data) \tag{5}$$

Algorithm 1: Vanilla Greedy Search Algorithm

```
Data: data matrix, incidence_matrix
Result: Incidence Matrix that best fit's the data
The Initial Incidence Matrix can be a matrix of just zeros;
$G_1 \leftarrow incidence\_matrix$;
Get all the neighbors $G_{1,i}, \dots, G_{i,N}$ of $G_i$;
for it $\in 1, 2, \dots$ do
    Compute the current score of the graph $G_i$;
    currScore = computeScores($G_i, data$);
    neigs = getNeighbours($G_i$);
    Compute their BGe scores;
    scores = computeScores(neigs);
    If the current score is greater than any of the other scores then break;
    if currScore $>$ scores $\forall G_{i,k}$ then
        return $G_i$;
    else
        Set the graph for the next iteration $G_{i+1}$ to $G^*$ which is the graph
          with the maximum score out of the list of neighbors of $G_i$;
        $G_{i+1} = G^*$
    end
end
```

Using an arbitrary initial Graph G_1 the **Greedy Search** algorithm scores the neighbors of a particular graph G that can be reached by single-edge operations. Then it compares the maximum score of the neighbors to the score of the current graph and repeats this procedure until we can no longer increase our score ie; all the neighbors of the graph G have lower scores than G.

3.1 Limitations of the Greedy Search Algorithm

Even though Greedy Search offers us a valid and straightforward way to fit a graph to our data it has many limitations, some of them are:

- Greedy search algorithm can get stuck in local minima, so many initializations of the algorithm are required to try to escape from it.
- It is possible that some graphs of the neighborhood of G have the same BGe score as G so a condition to break an infinite cycle of bounce is required.
- If the number of nodes is very large then computing the BDe score of all the valid neighbours of G can a be a **very expensive** computational task, reducing execution times of our algorithm drastically.

To make-up for some of this limitations various improvement over greedy search algorithms have been proposed [insert ref] On the next section of our paper we will discuss one of them **Bootstrap Aggregation Greedy Search**.

4 Bootstrap Aggregation Greedy Search

In this section we will explain what boostraping is and then proceed to analyze the results of the **Bootstrap Aggregation Greedy Search** algorithm on the **RAF-pathway** data.

4.1 Bootstrap

The bootstrap technique is widely used and recognized for its inferential power in statistics [insert ref]. Bootstrapping makes use of the data and as if it was a probability distribution and takes samples **with replacement** with it. Then by using a **model averaging** technique we will sum over all greedy search results and dividing by the number of times we took a sample from our data. This is particularly useful with small data since the search space of Bayesian networks is not sharply peaked around a single model. [ref]

4.2 Bootstrap Aggregation Algorithm

Hi ill be a typed algorithm

Appendix