UNIVERSITÄT REGENSBURG

Institute of Genomics & Practical Bioinformatics
Master of Science Computational Science

The max-min-hill-climbing algorithm

Report
in practical Bioinformatics

by

Michael Bauer

Matrikelnummer: 152 8558

Tutor: Dr. Giusi Moffa
Adviser: Prof. Dr. Rainer Spang
Date:

Contents

List of Figures

List of Tables

1 Abstract

In this report we present a new implementation of the max-min-hill-climbing algorithm (MMHC) for R, first stated in [TBA]. It combines both: greedy search and constraint-based learning techniques. We will discuss those two methods seperately and how they effect running time. We also want to work out the importance of this algorithm. The main goal of it is to reconstruct Bayesian networks from estimated data. Bayesian networks play a great role in science, economics, sports and many more fields where the observational data can get extremely big. It is not the first time R provides this algorithm but we come up by using RCPP (C++ interface for R) for our implementation to have a better chance to deal with big data. Since running time is getting more important we tried to improve it for this algorithm and beat the existing one.

2 Introduction

The max-min-hill-climbing algorithm is one of the state of the art algorithms in statistical computing. The primal aim of this algorithm is reconstructing BN out of estimated data. A Bayesian network is a Directed Acyclic Graph (DAG) whose nodes are random variables and eges represent conditional dependencies. If two random variables are connected they are said to be dependent. If there is no connection they are said to be conditional independent. BNs are more important than one can imagine. They play a great role in everdays life. For example [NBBCW] use them to predict the effect of missense mutations on the protein function. But not only medical science uses Bayesian networks. Another example where scientists used them was football. In [PKA] they refer to an article where european football clubs tried to predict injuries of their players depending on BNs. With a simple Google search you may also find the prediction of stock exchanges and many more. Wikipedia provides a simple but descriptive example which illustrates BNs in a nice way.

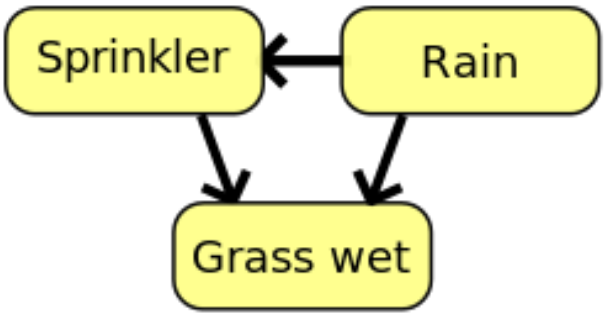


Figure 2.1: A simple example of a Bayesian network (source: http://en.wikipedia.org/wiki/Bayesian_network#mediaviewer/File:SimpleBayesNetNodes.svg).

Reconstructing those networks is not easy, more precisely it is a np-hard problem. It is not only the amount of data which leads to a bad running time, also the dependencies between nodes can slow the code down.
In the first step of this algorithm we try to reconstruct the skeleteon of the graph. Therefor we iterate over all variables - we select a fixed variable (we call it "target" **T**) in each iteration step - and find those variables which are dependent to the selected one. The more variables we find the longer a single iteration takes. The dependent variables are then said to be a parent

or a child of **T**. It may happen that there are false positive ones in our set (which we will call **CPC**). For this reason we have to check again if the parents/children in the set really belong to our selected variable **T**. The relation between **T** and its parents/children is symmetric. That means for a target **T** and $\mathbf{A} \in \mathbf{CPC}_T$ it follows:

$$\mathbf{T} \longleftrightarrow \mathbf{A} \Longleftrightarrow \mathbf{A} \longleftrightarrow \mathbf{T} \tag{2.1}$$

for a target **A** and $\mathbf{T} \in \mathbf{CPC}_A$. Since this relation holds we have to check in a second step if this symmetrie is fullfild. For that we check for every $\mathbf{X} \in \mathbf{CPC}_T$ if $\mathbf{T} \in \mathbf{CPC}_X$. In the end we did all our calculations twice to estimate the skeleton of the graph.
Though this is a great approach, we are interested in the whole Directed Acyclic Graph. The second part of the algorithm will then take this skeleton and add directed edges to it such that it does not become cyclic and is fully directed. We will see that this is based on one formula which is not complicated to understand but extremely tricky for implementation. Once we observe the Bayesian network from our data we then can look at the running time of the algorithm and where the time gets lost but also where we optimized to save time. But more important for us was to beat the existing algorithm for R (part of the "bnlearn" package). The goal for us was to be faster. So after a brief discussion of our implementation we will see if it was possible to optimize the code with RCPP to beat the "bnlearn" algorithm.

3 Background

Before we are able to analyze our implementation and talk about it in detail we need some mathematical background. In this section we fully follow [TBA][p. 5-7]. For the proofs of the Lemmas and Theorems we also reference to this paper.

3.1 Notation

We introduce our notation which is completely consistent to [TBA].
We denote

1. variables with an upper-case letter (e.g., A, V_i),
2. a state or a value of that variable by the same lower-case letter (e.g., a, v_i),
3. a set of variables by upper-case bold face (e.g., $\mathbf{Z}, \mathbf{Pa}_i$),
4. an assignment of state or value to each variable in the given set with the corresponding lower-case bold-face letter (e.g., $\mathbf{y}, \mathbf{pa}_i$),
5. special sets of variables (e.g. the set of all variables $\mathcal{V}$) with calligraphic fonts.

3.2 Definition (conditional independence)

Two variables X and Y are conditionally independent given $\mathbf{Z}$ with respect to a probability distribution P, denoted as $Ind_P(X; Y|\mathbf{Z})$, if for all $x, y, \mathbf{z}$ where $P(\mathbf{Z} = \mathbf{z}) > 0$,

$$P(X = x, Y = y|\mathbf{Z} = \mathbf{z}) = P(X = x|\mathbf{Z} = \mathbf{z})P(Y = y|\mathbf{Z} = \mathbf{z}) \tag{3.1}$$

or

$$P(X, Y|\mathbf{Z}) = P(X|\mathbf{Z})P(Y|\mathbf{Z}) \tag{3.2}$$

for short. If X, Y are dependent given $\mathbf{Z}$ we denote $Dep_P(X; Y|\mathbf{Z}$.

3.3 Definition (Bayesian network)

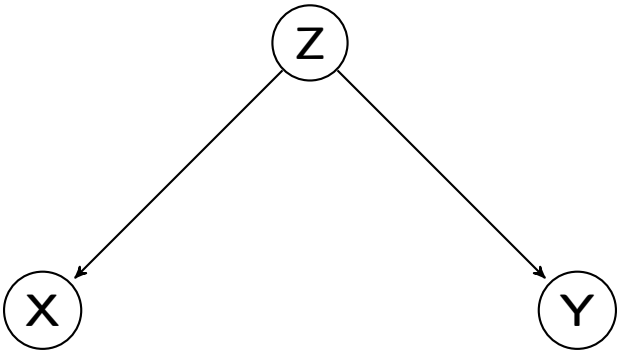
Let P be a discrete joint probability distribution of the random variables in some set $\mathcal{V}$ and $\mathcal{G} =< \mathcal{V}, E >$ be a Directed Acyclic Graph (DAG). We call $< \mathcal{G}, P >$ a (discrete) *Bayesian network* if $< \mathcal{G}, P >$ satisfies the Markov condition.

3.4 Definition (Markov condition)

Any node in a Bayesian network is conditionally independent of its non-descendants, given its parents.

3.5 Explanation

With those definitions we have our first concept we will discuss briefly. We will explain the definitions by using the following picture:



The Markov condition states that X and Y given $\mathbf{Z}$ must be conditionally independent of each other. This comes from the fact that X is a non-descendant of Y and vice versa and $\mathbf{Z}$ is a parent of both nodes. By fullfilling this condition we get from section 3.3 that this graph is a Bayesian network and with section 3.2 we have: $P(X, Y|\mathbf{Z}) = P(X|\mathbf{Z})P(Y|\mathbf{Z})$.

3.6 Defintion (collider)

A node W of a path p is a *collider* if p contains two incomming edges into W.

3.7 Definition (blocked path)

A path p from node X to node Y is *blocked* by a set of nodes $\mathbf{Z}$, if there is a node W on p for which one of the following two conditions hold:

1. W is not a collider and $W \in \mathbf{Z}$, or
2. W is a collider and neither W or its descendants are in $\mathbf{Z}$ [P88]

3.8 Definition (d-seperation)

Two nodes X and Y are *d-seperated* by $\mathbf{Z}$ in graph $\mathcal{G}$ (denoted as $Dsep_{\mathcal{G}}(X; Y|\mathbf{Z})$) if and only if every path from X to Y is blocked by $\mathbf{Z}$. Two nodes are *d-connected* if they are not *d-seperated*.

3.9 Definition (faithful)

If all and only the conditional independencies true in the distribution P are entailed by the Markov condition applied to $\mathcal{G}$, we will say that P and $\mathcal{G}$ are *faithful to each other* ([SGSN]). Furthermore, a distribution P is *faithful* if there exists a graph $\mathcal{G}$, to which it is faithful.

3.10 Definition (faithfulness condition)

A Bayesian network $< \mathcal{G}, P >$ satisfies the *faithfulness condition* if P embodies only independencies that can be represented in the DAG $\mathcal{G}$ ([SGSN]). We will call such a Bayesian network a *faithful network*.

3.11 Theorem

In a faithful Bayesian network $< \mathcal{G}, P >$ the following equivalence holds ([P88])

$$Dsep_{\mathcal{G}}(X; Y|\mathbf{Z}) \Longleftrightarrow Ind_P(X; Y|\mathbf{Z}) \tag{3.3}$$

3.12 Remark and Explanation

Remark: For the rest of this report we assume faithfulness of the networks to learn. For this reason we don't want to explain the corresponding definitions in detail. Just note, that the definitions are neccessary for mathematical correctness.

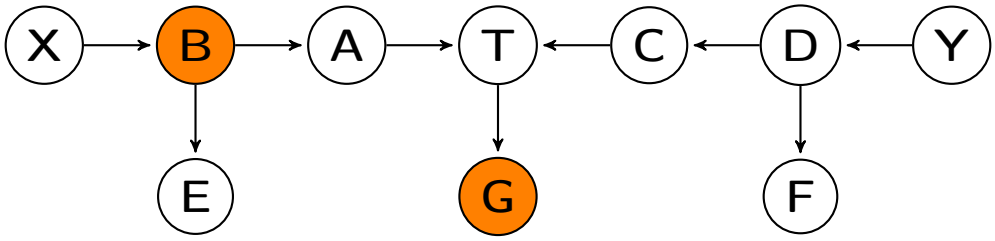
Explanation: The definition of a collider already tells everything about it. To illustrate a collider, we have:

Here Z is a *collider* because it has two incoming edges. In this case if we just look for $P(X;Y|\{\})$, the path between X and Y would be blocked and for this X and Y are *d-seperated*. If we look for $P(X;Y|\mathbf{Z})$, then this path is not blocked and we state that X and Y are *d-connected*.
Because of section 3.11 and the faithfulness assumptions we have say for the rest of our report that the terms d-seperation and conditional independence are equivalent. With this we already know that X and Y are conditional dependent given $\mathbf{Z}$ in the example above. This brings us a big step closer to learn the structure of a Bayesian network from observational data. Before we start looking at the algorithms, we want to give you two other examples for d-seperation of variables.

If we are looking for $Ind_P(X;Y|\mathbf{Z})$ with $\mathbf{Z} = \{B, D\}$ we learn that X and Y are conditionally independent given $\mathbf{Z}$. In other words they are d-seperated in the path because of the following reasons:

- T is a collider and blocks the path between X and Y.
- The nodes B and B are no colliders but they are elements of $\mathbf{Z}$.

The situation becomes a bit more difficult if we take a look at the next example:



We learn that the path between X and Y remains blocked by looking for $Ind_P(X; Y|\mathbf{Z})$ with $\mathbf{Z} = \{B, G\}$, i.e. X and Y are conditionally independent given $\mathbf{Z}$. If we would look at the path between A and Y we would learn that A and Y are d-connected. This comes from:

- T is a collider but its descendant $G \in \mathbf{Z}$, i.e. T would not block the path.
- The node B is no collider but it is an element of $\mathbf{Z}$. For that it blocks the path.

As we could see, detecting conditional independence of two nodes is quite difficult in small graphs. Since we normally observe large data sets with a couple of nodes, a concept for this is needed. As we will see, statistical methods, such as hypothesis testing is a useful friend for this task.

4 Functions of bnlearn

As we stated, our purpose was to implement the max-min-hill-climbing algorithm in a way that it will be faster and more efficient than the existing is. That's why we want to give you a brief introduction to the "mmpc" and "mmhc" functions of the "bnlearn" package. We also want to present some numbers depending on running time which show you the effectiveness of the two algorithms. Afterwards we explain our algorithm and we present our results. You will see that there will be some differences on the running time of both implementations.

4.1 mmpc(data.frame)

This function represents the first part of the algorithm, which returns the skeleton of the graph. You can choose between several testing methods. The input is a R data frame and the return value of it is of the class "bn". With the plot function you are able to plot the whole skeleton of the graph.

4.2 mmhc(data.frame)

This function is similiar to the mmpc(). The differences are, that it returns a DAG and you can choose a score function which is used to direct the edges.

5 The max-min-hill-climbing algorithm

That should be enough of bnlearn. We now want to go into the implementation of the algorithm. First of all we want you present the pseudo code and then talk about it. Afterwards we present the statistical methods behind the algorithm and how the effect running time. In the end we want to present an example which should be reconstructed by our algorithm.

5.1 Pseudo code MMHC

function MMHC $(\mathcal{D})$ #input: data $\mathcal{D}$
    for every variable $X \in \mathcal{V}$ do
        $\mathbf{PC}_X = MMPC(X, \mathcal{D})$
    end for

    $\mathbf{A} = BDeu(\mathbf{PC})$ # where **PC** contains all $\mathbf{PC}_X \forall X$
    $return(\mathbf{A})$ #returns an adjacency matrix A; a DAG

As mentioned before, the MMHC function has to parts. First we find the skeleton and then we add directed edges to the skeleton graph. Of course, this is not as easy as it seems to, so let's take a closer look to the several parts of MMHC.

5.1.1 max-min parents and children (MMPC))

The max-min parents and children is the algorithm which reconstructs the graph skeleton out of data. To bring you the concept of this closer, let us first present the pseudo code of it:

Bibliography

[TBA] Ioannis Tsamardinos, Laura E. Brown, Constantin F. Aliferis,
The max-min hill-climbing Bayesian network structure learning algorithm,
Springer Science + Business Media, Inc. 2006

[NBBCW] Chris J. Needham1, James R. Bradford, Andrew J. Bulpitt, Matthew A. Care and David R. Westhead,
Predicting the effect of missense mutations on protein function: analysis with Bayesian networks,
http://www.biomedcentral.com/1471-2105/7/405

[PKA] http://www-ekp.physik.uni-karlsruhe.de/županc/WS1011/docs/Datenanalyse2010_3.pdf

[P88] Pearl, 1988

[SGSN] Spirtes, Glymour & Scheines, 1993, 2000;
Neapolitan, 2003