Learning Causal Networks in Python

James Callan

1 Literature Review

1.1 Causal Networks

Causal Networks are graphical models which represent a set of variables, their conditional dependencies, and their causal relationships [1] as a Directed Acyclic Graph (DAG) or a Maximal Ancestral Graph (MAG). The nodes of the Graph represent the variables and the edges represent causality, the direction of the edge represents the direction of causality with parent nodes causing child nodes [1]. Causal networks also represent conditional independence with d-separation, variables X and Y are conditionally independent on set Z if nodes X and Y are d-separated by nodes in set Z [1].
In a DAG all edges have a single direction, whereas in a MAG edges may be directed or bidirectional [2].

The learning of causal networks allows relationships between variables to be uncovered and presented in a simple and human readable fashion.

1.2 D Separation

Two nodes are considered unconditionally d-connected if there exists a path between them which is unblocked. They are d-separated if the are not d-connected. A path is any sequence of adjacent edges regardless of their directionality and unblocked refers to a path that does not traverse a "collider" [3]

A collider is node in a path which is both entered and left on edges which are directed toward the node. If there exists no path between nodes X and Y which does not traverse a collider, nodes X and Y are unconditionally d-separated or d-separated conditioned on the empty set [3]. Two unconditionally d-separated nodes in a causal network are considered to be independent [4].

Two nodes X and Y are d-separated conditioned on set Z if there exists no path in which any colliders are from set Z or descendants of members of Z [3]. If nodes X and Y are d-separated by conditioning set Z then $X \perp\!\!\!\perp Y|Z$. [4]

1.3 Faithfulness

By testing for d-separation of variables X and Y given conditioning set Z on a graph we can see if $X \perp\!\!\!\perp Y|Z$ [4]. However, the reverse is only true if the distribution is faithful to the graph. That is, if a distribution is faithful to a graph then only the independence relationships represented by d-separation on the graph are present on the

graph exist [5] Assuming faithfulness allows the topology of a graph to be learned by testing pairs of variables for independence given various conditioning sets.

1.4 Discriminating paths

For the FCI and RFCI algorithms the notion of discriminating paths is needed. A path π = (A, .., X,Y,Z) is discriminating if:

1. π must include at least 3 edges
2. Y is a non-endpoint on π and is adjacent to Z on π.
3. A is not adjacent to Z and every other node is a collider on π and a parent of Z. [6]

1.5 Algorithms For Learning Causal Networks

There have been a number of algorithms developed to learn graphical causal representations from data. Over the course of this project, I plan to implement: The PC algorithm [7], the Fast Causal Inference (FCI) algorithm [6], and the Really Fast Causal Inference (RFCI) Algorithm [6].

Rather than learning an individual DAG the PC algorithm learns a set of DAGs which could represent the causal relationships found in the data. In cases where the direction of an edge is unclear no direction is specified. The output of the algorithm is a partially directed acyclic graph which represents the set DAGs with all possible orientations of undirected edges [7].

The FCI and RFCI algorithms learn partial ancestral graphs (PAGs) which represent the set of MAGs that fit the data [6]. Unlike in the PC algorithm if the direction of an edge can not be determined it is assumed that there is some set of latent variables influencing the nodes being tested. Each end of an edge can be one of three things in a PAG, an arrow in every MAG ($>$), a tail in every MAG ($-$) or an arrow in at least one MAG and a tail in at least on MAG (o), ends of an edge denoted by $*$ can be any of the specified types. This allows for a richer display of information than a partial DAG [6].

1.6 Computing a skeleton

All three algorithms begin by calculating the skeleton of the graph. The skeleton consists of the edges that are present in the partial DAG or PAG however all edges are undirected. When calculating the skeleton the separation set of each pair of variables is recorded [6, 7].

The skeleton is calculated using a series of conditional independence tests. These conditional independence tests are dependent on the type of input data, therefore conditional independence tests can be defined independently of the algorithm.

To begin the skeleton computation the fully connected undirected graph of all variables is constructed and then conditional independence tests are performed to determine which edges to remove.

For the first round of tests, the conditioning set is the empty set Each pair of adjacent variables is tested for independence using some predefined test. For variables X and Y if $X \perp\!\!\!\perp Y$, then the edge X-Y is removed and the empty set is recorded as the separation set (X,Y).

Once all edges have been tested the size of the conditioning set, k, is increased by 1. Now each remaining pair of adjacent variables X and Y are tested for independence given the conditioning set Z. Z is each subset of adj(X)\{Y} of size 1. If any test shows $X \perp\!\!\!\perp Y|Z$ remove the edge X-Y and record Z as the separation set of the pair (X,Y)

This process is repeated with k being incremented after all edges X-Y have been tested conditioned upon all subsets Z of adj(X)\{Y} of size k. It halts when there are no sets Z of size k for any edge. The edges which have not been removed form the skeleton of the graph which is needed for the PC, FCI and RFCI algorithms along with the recorded separation sets of removed edges [6, 7].

1.7 The PC Algorithm

After generating the skeleton and separation sets of pairs of nodes, the pc algorithm attempts to orient the undirected edges of the graph.

It begins by searching for colliders. For each triple of vertices A, B, and C where A and B are adjacent, B and C are adjacent, but A and C are not adjacent orient the edges A-B-C to A→B←C if B is not in the separation set of the pair (A,C).

Once colliders are specified other edges can be oriented. This is done by repeating two steps until no more edges can be given a direction:

i If there is is a directed edge from A to B, there is an undirected edge from B to C, and no edge from A to C, orient edge B-C as B→C.

ii If there is a directed path from node A to node B and an edge A-B orient the edge as A→B [7].

1.8 The FCI algorithm

the FCI algorithm starts in the same way as the PC algorithm using exactly the process of conditional independence tests to estimate a skeleton of the final graph. A list of unshielded triples M must also be found. Unshielded triples are 3 nodes, A, B and C where A and B are adjacent, B and C are adjacent, but A and C are not adjacent. In this skeleton however undirected edges are represented as "o-o". Due to the presence of latent variables this may not be the final skeleton of the graph. It also orients the colliders in a similar way to the pc algorithm, going through all unshielded triples (A,B,C) and orienting the edges $A*-oBo-*C$ as $A*->B<-*C$ if B is not in the seperating set of the pair (A,C).

Next, the final skeleton of the graph is obtained. To do this we must be able to find the possible d-separation set in the graph of a pair of variables X and Y. This set consists of all nodes Z which are colliders on some path π between X and Y and on every subpath of π (A,B,C) B is a collider or (A,B,C) is a triangle in the graph.

Once the final skeleton has been estimated, repeat the step to orient collider edges on the new skeleton. Finally the edges can be oriented using the rules ($R1 - R10$) found in [8]. [6]

1.9 RFCI Algorithm

Like the other two algorithms the RFCI begins by estimating the skeleton of the final graph.

Next the colliders in the graph are found, this is done differently than in the two previous algorithms. We created and empty list L and for each triple (A,B,C) in the list of unshielded triples we check if both:

i $A \perp\!\!\!\perp B \mid ((sepset(A,B) \setminus \{C\}) \cup S)$

ii $B \perp\!\!\!\perp C \mid ((sepset(A,B) \setminus \{C\}) \cup S)$

where sepset(X,Y) is the separating set of Nodes X and Y and S is the set of selection variables. If both of these tests hold, add to list L. If only the first test holds, find the minimal separating set of A and B and record it as the separating set of A and B. Next, remove the edge A-B, add any newly create unshielded triples to M, and remove any destroyed triples from L and M. If only the second test passes do the same as above but for nodes B and C.

Finally the edges must be oriented. To begin edge orientation apply rules $R1 - R3$ found in [8]. Next triangles (X,Y,Z) must be found with edges $Yo-*Z$, $X <-* Y$, and $X --> Z$. Once a triangle is found the shortest discriminating path from X to Z must be found.

If a path π exists take each pair of adjacent variables A and B on π and creator a counter $l = 0$. Next find a subset $\mathbf{Y} \subseteq (sepset(X, Y) \setminus \{A, B\})$ where $|Y| = l$. Then if $A \perp\!\!\!\perp B|Y \cup S$ let sepset(A,B) = Y, delete the edge $A * - * B$ from the graph and update newly formed unshielded triples orientation using the collider orientation described earlier. If the conditional independence test fails increment l and try again, testing with all subsets Y of size l, repeat this process until no set Y exists where —Y— = l or the edge $A * - * B$ is removed.

If no edges are removed from π by the above process then either: orient $Yo-*Z$ as $Y --> Z$ if Y is in the sperating set of X and Z or orient $X <-* Yo-*Z$ as $X <-> Y <-> Z$ if not.

Finally orient as many edges using rules $R5 - R10$ from [8].

This whole process must be repeated until it no longer alters any orientations. [6]

1.10 Existing Implementation in R

The three algorithms have already been implement in the pcalg package for R. R is a programming language and software environment aimed at creating statistical software. R has issues with speed and efficiency however code written in other faster languages such as c can be in used in place of pure R code.

1.11 Python

Python is one of the most widely used general purpose programming language whose usage is growing rapidly, particularly in machine learning applications. Unlike R python is widely used in industry and has many libraries designed for non statistical computation. Implementations of causal learning algorithms in python would enable programmers to easily interface with a variety of software and applications. As python is more widely used than R it will also open up the ability to learn causal networks to a wider userbase. There are many widely used python libraries which run underlying code written in other languages for the sake of efficiency and speed (e.g. numpy).

While both R and python are growing rapidly, python is growing slightly faster and is growing from a much greater user base so it is likely that it's usage will continue to overshadow R's.

2 Design

2.1 Methodology

2.1.1 Waterfall

2.1.2 Spiral

2.1.3 Agile

2.1.4 Chosen Approach

2.2 Tools

2.2.1 Version Control

2.2.2 Python Libraries

2.3 Requirements

2.3.1 Epics

2.3.2 User Stories

2.4 Abstract Design

2.5 Sprints

2.5.1 Tasks

2.5.2 Concrete Architecture

2.5.3 Unit Test

3 Implementation

3.1 Final Architecture

4 Validation

5 Conclusion

References

[1] Thomas Verma and Judea Pearl. Causal networks: Semantics and expressiveness. In *Machine Intelligence and Pattern Recognition*, volume 9, pages 69–76. Elsevier, 1990.

[2] Jiji Zhang. Causal reasoning with ancestral graphs. *Journal of Machine Learning Research*, 9(Jul):1437–1474, 2008.

[3] Judea Pearl. Causality: models, reasoning, and inference. *Econometric Theory*, 19(675-685):46, 2003.

[4] Judea Pearl. Causal inference in statistics: An overview. *Statist. Surv.*, 3:96–146, 2009.

[5] Richard Scheines. An introduction to causal inference.

[6] Diego Colombo, Marloes H Maathuis, Markus Kalisch, and Thomas S Richardson. Learning high-dimensional directed acyclic graphs with latent and selection variables. *The Annals of Statistics*, pages 294–321, 2012.

[7] Peter Spirtes and Clark Glymour. An algorithm for fast recovery of sparse causal graphs. *Social science computer review*, 9(1):62–72, 1991.

[8] Jiji Zhang. On the completeness of orientation rules for causal discovery in the presence of latent confounders and selection bias. *Artificial Intelligence*, 172(16):1873 – 1896, 2008.