An Interactive Web Application for Automatic Causal Inference

Nikita Janakarajan Lea Künstler Tian You Michael Zellinger

IML Causal Group, ETH Zurich

ABSTRACT

We present an interactive web app for conducting causal inference on medical data from the intensive care unit (ICU). Our app allows the user to encode a medical hypothesis in the form of a causal graph. The user can then upload data and compute the influence of dynamical variables on each other. For example, how will administering 10 mcg of adrenaline to a patient now affect the patient's heart rate in 15 minutes? Going back and forth between tweaking the causal graph and fitting the resulting causal model allows the user to both 1) refine their scientific hypothesis and 2) obtain plausible estimates of causal effects.

1 INTRODUCTION

Modern intensive care units are becoming increasingly data-driven environments. With patients continuously plugged into a dazzling array of medical equipment, clinicians collect more and more data at a faster pace.

The resulting data sets take the form of stratified multivariate time series. Each individual patient represents a stratum at which we collect a multivariate time series made up of the followings:

- physiological signals: heart rate, blood pressure, systemic vascular resistance, ...
- laboratory test results: pH, base excess, electrolytes, ...
- drug treatments: adrenaline, iloprost, dobutamine, ...
- static patient information: age, sex, weight, ...

Doctors would like to leverage this data to improve medical knowledge and support medical decision-making. On the one hand, there is fundamental uncertainty about the effect of drugs on patients. Even if a drug proves reliable in one context, it is not clear whether it would help or harm patients in another. On the other hand, physicians must often choose among several seemingly equivalent drugs: such decisions often rely more on an individual doctor's experience and intuition rather than rigorous scientific evidence.

Since intensive care physicians do not conduct randomized experiments on patients, data sets from the ICU constitute observational data. Analyzing such data requires tools from causal inference because standard machine learning approaches cannot predict the results of actively intervening on patients.

As a first step towards helping doctors and medical data scientists leverage the reams of data collected in today's intensive care units, we present an interactive web app for conducting causal inference on such data. This report is laid out as follows. In Section 2, we describe how the user interacts with our app. In Section 3, we describe our computational and mathematical methods. In Section 4, we give an overview over our software implementation. Finally, in Section 5 we summarize our report and mention fruitful avenues for future work.

patient_id	time	age	bmi	HR	BP	adre
1	0	57	21	100	120	0
1	1	57	21	95	125	0
1	2	57	21	98	130	0
...	...	...	...	...	...	...
2	0	43	23	60	80	5
2	1	43	23	65	60	5
2	2	43	23	70	50	5
...	...	...	...	...	...	...

Table 1: Format of uploaded data

2 WORKFLOW

We give a short overview of our app's three main components and walk the reader through the workflow of our interactive causal inference application.

2.1 Data Upload

We assume the availability of a pre-processed stratified time series data for patients in the intensive care unit of a hospital, *i.e* the data has undergone missing value treatment, noise treatment, and normalization. In the current version of our application, only files in "csv" format can be uploaded from the user's local machine. An example of the data schema is shown in Table 1.

2.2 Causal Graph Definition

The next step in the workflow requires the user to teach the system. The user must provide the system with their knowledge of causal relations between the attributes of interest. This information is extracted from the user in an interactive manner through a causal graph editor.

The representation of causal graphs is a two-step process. First, the user creates groups for variables with similar interpretation and specifies whether each group contains "dynamic" or "static" variables. For example, in Figure 1 the user may create a group "Physiology" containing the variables heart rate, cardiac output, and stroke volume. These physiological variables are "dynamic" since they vary with time. The attribute names from the uploaded data are made available to the user for exclusive selection - each attribute can only be assigned to a single group. In the second step, the user then defines details for each of these grouped nodes - members of the grouped node are connected based on the user's domain knowledge of causality as shown in Figure 2.

Once groups are created, the user can add edges between groups and between members within the same group. For edges connecting two dynamic nodes, users need to specify the time steps, because many time series data depends on not only one time point but many points in the past. A concrete example would be injection of adrenaline would affect a patient's heart rate for the next few hours. For edges with one or more static nodes, no time step can be specified.

When the user finishes creating or updating the graph, they press the "confirm graph" button that proceeds to send the causal graph to the back-end for further processing.

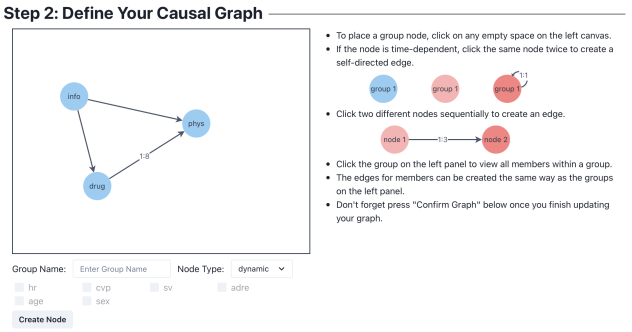

Figure 1: The editor for the grouped graph, where the user can group selected variables and specify relationships between these groups through edges. If the grouped nodes contain dynamic variables, the user can also specify the number of time steps that should be taken into account while calculating effects.

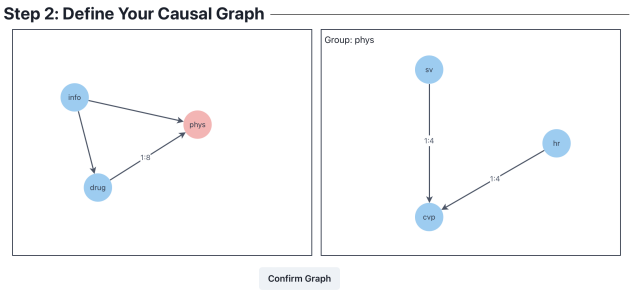

Figure 2: Once the variables are grouped, the user can specify detailed relationships between them. To the left is the grouped graph editor. Selecting a group highlights it in red and the editor on the right pops up. In a similar manner to the grouped graph editor, the user can define the causal graph within this group on the right side editor by clicking on the groups to connect and specifying the time steps to take into account.

2.3 Causal Effect Estimation

To study causal effects of an intervention on an attribute of choice, we provide a causal estimation pane (see Figure 3) where the user can select the cause and effect variables from a pre-populated drop-down button, and the time steps to effect. This information combined with the graph structure is converted into a table-like format in the back-end of our application, where the effect variable is our dependent variable, and our intervention variables and all variables that affect it are structured as independent variables stratified by the specified time interval. The user must also specify the minimum and maximum value of the intervention for which they want to test the effects. Upon doing so, a random forest regressor then processes this data and returns the causal effect over a range of time steps and intervention values, which were generated uniformly from the specified min and max values. The user can view the effect trend of the intervention at different dosage values by communicating through a slider - a time series chart is displayed for the intervention value returned by the slider, as shown in our example 4. The user can visualize different trends for different values in this manner and draw conclusions as they see fit. Should the graph look suspicious, or incorrect, the user can redraw the causal graphs and repeat the process, making this a mixed-initiative system.

3 Methods

In this section, we describe our methods for representing causal graphs and computing causal effects.

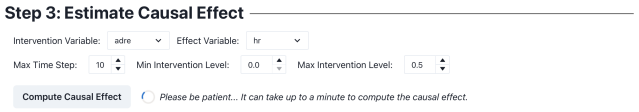

Figure 3: The estimation pane allows the user to specify the cause (adre) and effect (hr) variables, and the time steps (10) to effect. Additionally, the users can also specify the minimum (0.0 µg/kg) and maximum (0.5 µg/kg) intervention values for which the effects should be computed.

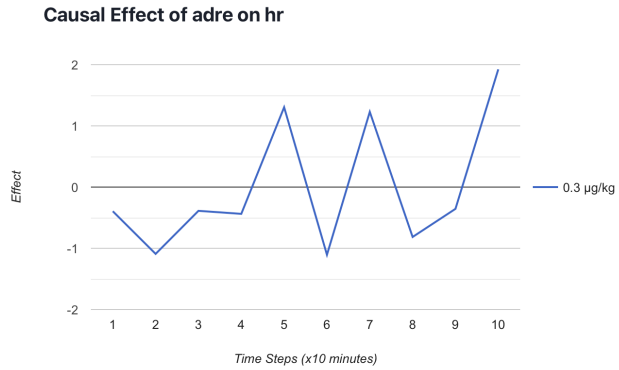

Figure 4: The time series chart our application returns showing the effect of the selected variable, adre (adrenaline) on hr (heart rate) at 0.3 µg/kg of adre administered over the course of 1.5 hours.

3.1 Graph Representation

When medical data is gathered from a patient, multiple variables are retrieved. These variables are often interconnected, and a change in one variable can affect another. To represent these causal relations, directed acyclic graphs are frequently used. In this model, the nodes represent the variables, and the edges denote the relations.

Since many variables are present in medical data, we decided to split this graph into two layers to be able to represent them in a more visually appealing way. The variables are split into multiple groups, such as patient info and physiology, and the causal relations are defined between the groups rather than between each variable, as shown on the left panel of Figure 2. To also allow defining the causal relation on a variable level, these grouped nodes consist of a graph. In the the right panel of Figure 2, we show how causal relationships are defined between nodes within the physiology group.

As the medical data of one patient evolves, we declare every variable to be static or dynamic. Static variables typically do not change over the period the data is from, and dynamic variables change fast-paced. To also represent the evolution over time, edges between two dynamic nodes have a time attribute, see figure 2. This attribute reflects the time steps that an effect of a variable would last.

3.2 Statistical Methodology

We quantify the causal effect of a dynamic variable X on another dynamic variable Y by the conditional expectation

$$\mathbb{E}[Y_{t+\Delta t}|\mathrm{do}(X_t = a)], \tag{1}$$

where $t+\Delta t$ and t are the times at which we measure variables Y and X, respectively. The expression $\mathrm{do}(X_t = a)$ comes from Pearl's do-calculus and signifies intervening on the variable X_t by setting its value to a.

To compute the expectation in Equation (1), we make use of Pearl's backdoor adjustment formula [1]. Specifically, if the variables $\{Z^1, \ldots, Z^p\}$ are the direct parents of X_t in the causal graph,

then the expansion

$$\mathbb{E}[Y_{t+\Delta t}|\mathrm{do}(X_t = a)] = \int \mathbb{E}[Y_{t+k}|X_t = a; Z^1 = z^1, ..., Z^p = z^p]\mathrm{d}P(\mathrm{d}z^1 ... \mathrm{d}z^p) \tag{2}$$

holds.

To evaluate the right-hand side of Equation (2), we model the integrand – a standard conditional expectation – by fitting a random forest to the data. Approximating the marginal distribution $P(z^1 ... z^p)$ by the corresponding empirical distribution function, we compute the integral as the average

$$\hat{\mathbb{E}}[Y_{t+\Delta t}|\mathrm{do}(X_t = a)] = \frac{1}{N}\sum_{i=1}^{N} \hat{f}(Z^{1,(i)}, ..., Z^{p,(i)}), \tag{3}$$

where $\hat{f}$ is the random forest and the values $Z^{1,(i)}, ..., Z^{p,(i)}$ are the observed values of variables $Z^1, ..., Z^p$ in the i-th observation.

4 Implementation

This section gives an introduction into how our app is build. We created a graphical user interface in the frontend which communicates with the backend, where most of the computation happens. To connect the front- and backend, we used FastAPI, which allowed us building an API in Python.

4.1 Frontend Design

In this subsection, we highlight some design decisions on the user interface and describe the programming style of the frontend.

4.1.1 User Interface

When new users open our app, they will see the three motivating questions and the goal of the app: "Compute and visualize the causal effect in your data in 3 simple steps." Then, they can easily follow the three steps below and compute the causal graph based on the data they upload.

We tried different view arrangements like tabs and dashboards. Ultimately, we decide to use scrolling because it best fits our "step-by-step" interaction style. We considered revealing the next step only after the user completes the previous step, but we find that listing all three steps at once gives a better idea of what needs to be done and makes the app more accessible.

Although the interactions with the graph editor are relatively intuitive, we provide a guide for our users on the right panel to help them start quickly. The users can choose what variables to include because they are often not interested in all variables for the causal inference. When the nodes are selected, their background changes from blue to red, which are two contrasting and colorblind-friendly colors. When two nodes are selected for edge creation, they will be displayed in two different shades of red, and the dialog below uses the corresponding colors to help users distinguish between "from" and "to" nodes.

4.1.2 Programming

The frontend is implemented using React.js and TypeScript. We use classes for all objects, from sections in the app to edges and nodes in the graph. The object-oriented programming style allows us to write the code in a clean, modular and organized way. Each graph has an abstract representation as instances of GROUP and GROUPEDGRAPH. Graphically, the graphs are represented in an instance of GRAPHCANVAS by GRAPHCANVASEDGE, GRAPHCANVASNODE and GRAPHCANVASSELFEDGE.

4.2 Backend Design

The backend is implemented in Python. Similar to the frontend, we use classes to represent the graphs in the backend. Specifically, we have different classes for representing the different types of nodes and edges. We split the nodes into two different classes: CAUSALNODE and GROUPEDCAUSALNODE. The CAUSALNODE objects represent the nodes of the variables from the medical data. These CAUSALNODES are connected using Edge objects. Together, the CAUSALNODE and Edge objects form an object of the class CAUSALGRAPH.

A GROUPEDCAUSALNODE object, on the other hand, consists of a causal graph and the dynamic attribute. A GROUPEDCAUSALGRAPH object consists of multiple grouped causal nodes connected with Edge objects.

This object-based structure of the graph in the backend allows us to access the information needed to compute the causal effect easily.

5 Conclusion

We have developed a web application that allows physicians to estimate a drug's effect on a patient quickly. It considers the doctor's pre-knowledge by enabling them to input a causal graph, reflecting the causal relations between the different variables in the dataset.

To achieve a layperson-friendly application, we build a graphical user interface for drawing causal graphs. It allows the user to group the variables and create graphs within a grouped node and between them. Using this knowledge, the application then estimates the effects of an intervention variable on another variable for different drug dosages. To visualize this effect, the applications shows the effect in a time plot.

5.1 Future Improvements

The current application still has some improvements that can be implemented. At the moment, the user can only add nodes and edges to the causality graphs. They currently do not have the possibility of deleting nodes or edges, which would allow for a more interactive working cycle.

To facilitate the creation of the causal graphs further, the application could suggest a pre-grouping of the variables and give the user a predefined causal graph that they only have to adjust accordingly. This reduces the work of creating the graphs.

To further improve the user experience, the application could limit the intervention variables in the dropdown menu to variables that have one or more edges originated from it.

Further, the app could be modified to show different intervention variables in the estimation pane rather than a single one.

The frontend used classes in most cases, which prevented using hooks in many built-in packages. If we had used functions instead of classes, then we could utilize the hooks better and provide better user experience.

Acknowledgments

Michael J. Zellinger thanks Peter Bühlmann for his support.

References

[1] J. Pearl. Causal inference in statistics: An overview. *Statistics Surveys*, 3:96 – 146, 2009. doi: 10.1214/09-SS057

MEMBERS' CONTRIBUTION STATEMENTS: IML CAUSAL GROUP

In this document, each group member concisely states some of their contributions to our project.

Lea

- **Backend** I mainly worked on the backend during this project. I helped Michael implement the different classes by writing the functions. I did a lot of debugging, hunting down the various errors when trying to compute the causal effect.
- **Frontend** In the beginning, I contributed to figuring out how to upload data and work with it in the backend. During the course of the project, I looked into how to best implement a multipage layout, which was later discarded since we switched back to a single page layout.
- **Report/Poster** I contributed to writing the report. For the poster, I designed an initial layout. I was also present at the poster session with Nikita.

Nikita

1. **Frontend**
   - **Intervention range and Dosage Slider:** I implemented a numeric step incrementor button using the Chakra-UI to allow users to specify a minimum and maximum value of the intervention they would like to test. In addition, I provide a combined step incrementor with slider feature for the users to select the intervention value (dosage) at which to display the time series chart of the causal effects. I contributed to the back and forth communication of the selected parameter values and results of causal effect between the front-end and back-end.
   - **Time-Series Visualizer:** Implemented a dynamic time series chart that takes the computed causal effects at the selected intervention value (including the logic for retrieving this data) and displays it with its confidence interval as a shaded area around the main graph. If no confidence interval is computed, the visualizer simply displays the main graph.
   - **Graph Communication:** I implemented both the communication script to send the graph to the back-end and the confirm button that triggers it, which was later adapted to chakra-UI by Tian. The graph is first processed to remove unimportant/useless variables such as position of the node on the editor before being sent to the back-end.
   - **Deprecated:** I fixed node selection and implemented the edge selection in the older version of the graph editor. I also implemented the Heatmap displaying the correlation between variables in the uploaded data file and a drop-down functionality to choose which variables to display in the heatmap. I organized the displayed the heatmap and time series in a panel-view in the earlier version of the app. All back and forth communication with respect to the heatmap and variable selection between the front-end and backend including buttons to confirm selection was also implemented by me. I also experimented with axios for communicating between the frontend and backend. Additionally, modified the edge dictionary to allow multiple incoming and outgoing edges.
2. **Backend**
   - **Deprecated:** I implemented the Correlation function that computes the correlation between different variables in the uploaded data and returns them in a SON appropriate format that can be parsed in the frontend to visualize the heatmap. Contributed to the old version of the causal graph class. Implemented function to receive nodes and edges of the graph from frontend. I also tried to fix the causal computation function in the backend which was eventually rewritten to a functional version by Michael.
3. **Documentation/Poster/Miscellaneous:** I fixed bugs in components where I was not the main contributor. I contributed to the README of our repository where progress and issues were highlighted for the milestones as well as steps to ensure reproducibility. I contributed to and presented the Poster along with Lea. I also contributed to the writing of this report.

Michael

- Proposed project
- Helped organize our group
- Suggested our current overall one-page layout
- Wrote the backbone of our graph editor in TypeScript from scratch
- Wrote graph classes in TypeScript from scratch
- Wrote our object-oriented causal graph framework in Python from scratch
- Wrote our module for parsing causal graphs from JSON
- Wrote backend functions for computing causal effects
- Wrote a few functions for communicating between frontend and backend
- Wrote the text on our poster
- Contributed to our group's report

Tian

- **Graph editor:** Added and improved many functionalities in the graph editor based on Michael's framework. Graphed edges and added their labels using the package Xarrow and SVG. Automated node drawing within each group. Designed and implemented self-directed edges. Implemented node selection with different colors.
- **User interface:** (Re)designed the user interface and integrated the package Chakra UI in the app. Updated the looks of all input box, check boxes, buttons and most texts. Reorganized all components in the app to make it look cleaner. Picked and implemented the color theme. Adjusted details like node size, arrow shapes, and text alignment.
- **Documentation:** Wrote and implemented the introduction /teaser part and graph editor user guide in the app. Contributed writing this report, creating graphs in the poster, and the README file on Gitlab. I was present via Zoom during the poster session.