ananke-causal

Apr 29, 2020

Contents:

CHAPTER 1

How to install

Requires Python 3.7 or higher. Run the following command:

```
pip3 install ananke-causal
```

Check out the gitlab!

CHAPTER 2

Documentation

2.1 Quickstart

This is a short tutorial for users that want to get into the thick of things right away. However, we highly recommend that users read the other docs provided in order to get a feel for all of the functionality provided in Ananke. The estimators here are based on theory in our paper Semiparametric Inference For Causal Effects In Graphical Models With Hidden Variables (Bhattacharya, Nabi, & Shpitser, 2020).

Following are the necessary packages that need to be imported for this tutorial.

[1]:
```
from ananke.graphs import ADMG
from ananke.identification import OneLineID
from ananke.estimation import CausalEffect
from ananke.datasets import load_afixable_data
from ananke.estimation import AutomatedIF
import numpy as np
```

2.1.1 Creating a Causal Graph

In Ananke, the most frequently used causal graph is an **acyclic directed mixed graph (ADMG)**. Roughly, directed edges $X \to Y$ indicate X is a direct cause of Y and bidirected edges $X \leftrightarrow Y$ indicate the presence of one or more unmeasured confounders between X and Y. Let's say we are studying the efficacy of a new antiretroviral therapy vs. an old one. Call this the treatment T where $T = 1$ represents receiving the new drug and $T = 0$ represents receiving the old. Our outcome of interest is the patients CD4 counts post treatment. Thus, our target of interest (in potential outcomes notation) is the counterfactual contrast $\psi \equiv E[Y(1)] - E[Y(0)]$. This is how one may create a causal graph, with additional variables relevant to the given problem.

[2]:
```
vertices = ['Income', 'Insurance', 'ViralLoad', 'Education', 'T', 'Toxicity', 'CD4']
di_edges = [('ViralLoad', 'Income'), ('ViralLoad', 'T'), ('ViralLoad', 'Toxicity'),
            ('Education', 'Income'), ('Education', 'T'), ('Education', 'Toxicity'),
            ('Income', 'Insurance'), ('Insurance', 'T'), ('T', 'Toxicity'), ('Toxicity
↪', 'CD4'), ('T', 'CD4')]
```

(continues on next page)

(continued from previous page)

```
bi_edges = [('Income', 'T'), ('Insurance', 'ViralLoad'), ('Education', 'CD4')]
G = ADMG(vertices, di_edges, bi_edges)
G.draw(direction="LR")
```

[2]:

2.1.2 Identification of the Causal Effect

Using Ananke, we can ask whether the effect of a given treatment T on a given outcome Y is identified or not. We have implemented the one line ID algorithm provided in Nested Markov Properties for Acyclic Directed Mixed Graphs which is sound and complete in identifying $p(Y(t))$ or equivalently $p(Y|do(t))$. We show how to use this in Ananke through the following example.

[3]:
```
one_id = OneLineID(graph=G, treatments=['T'], outcomes=['CD4'])
one_id.id()
```

[3]:
```
True
```

2.1.3 Estimation of the Causal Effect

Ananke provides an easy interface in order to compute causal effects. First, we instantiate a `CausalEffect` object.

[4]:
```
ace_obj = CausalEffect(graph=G, treatment='T', outcome='CD4')  # setting up the␣
↪CausalEffect object
```

```
 Treatment is a-fixable and graph is mb-shielded.

 Available estimators are:

1. IPW (ipw)
2. Outcome regression (gformula)
3. Generalized AIPW (aipw)
4. Efficient Generalized AIPW (eff-aipw)

Suggested estimator is Efficient Generalized AIPW
```

In this case, it recommends *Efficient Generalized AIPW* which is to say, that the estimator used to compute the effect in this case looks a lot like Augmented IPW (which is doubly robust). Further, Ananke uses semiparametric theory in order to provide an estimator that achieves the lowest asymptotic variance.

Given the list of estimators, it is up to the user to specify what estimators they want to work with. For instance, if the user decides to use the efficient generalized AIPW, they only need to use the keyword given in front of it, i.e., `eff-aipw`, when computing the effect. All the nuisance models are fit using generalized linear models provided in the `statsmodels`. Users interested in using different modeling approaches can refer to the documentation on accessing the functional form of the influence functions at the end of this notebook.

Let's load up some toy data and use all of the available estimators in order to compute the causal effect.

[5]:
```
data = load_afixable_data() # load some pre-simulated data
ace_ipw = ace_obj.compute_effect(data, "ipw")
ace_gformula = ace_obj.compute_effect(data, "gformula")
ace_aipw = ace_obj.compute_effect(data, "aipw")
ace_eff = ace_obj.compute_effect(data, "eff-aipw")
```

(continues on next page)

(continued from previous page)

```
print("ace using IPW = ", ace_ipw)
print("ace using g-formula = ", ace_gformula)
print("ace using AIPW = ", ace_aipw)
print("ace using efficient AIPW = ", ace_eff)
```

```
ace using IPW =  0.4917151858666866
ace using g-formula =  0.4968077996596518
ace using AIPW =  0.5005117929752623
ace using efficient AIPW =  0.491715185866723
```

In addition, the user can run bootstraps and obtain $(1-\alpha)*100\%$ confidence level for the causal effect point estimate. The user needs to specify two arguments: number of bootstraps `n_bootstraps` and the significance level α `alpha`. The confidence interval can then be obtained via bootstrap percentile intervals, as described in All of Nonparametric Statistics. In this case, the call to the `compute_effect` returns three values: the first corresponds to the effect computed on the original data, the second and third are the pre-specified lower and upper quantiles, i.e., $(\frac{\alpha}{2}, 1-\frac{\alpha}{2})$. The default value for α is set to be 0.05. We illustrate this through the following toy example.

[6]:
```
np.random.seed(0)  # setting the seed to get consistent numbers in the documentation
ace, Ql, Qu = ace_obj.compute_effect(data, "eff-aipw", n_bootstraps = 5, alpha=0.05)
print(" ace = ", ace, "\n",
      "lower quantile = ", Ql, "\n",
      "upper quantile = ", Qu)
```

```
 ace =  0.491715185866723
 lower quantile =  0.45738153193196984
 upper quantile =  0.5898403691123407
```

While Ananke has its own built-in esimation strategy for every identifiable causal effect concerning a single treatment and outcome, users may be interested in building their own estimators based on the identifying functionals/nonparametric influence functions. This allows users to use their own preferred estimation strategies such as sample splitting, using their preferred machine learning model etc. Below, we provide an example for a new causal graph reflecting some background knowledge and we are interested in the causal effect of the treatment T on the outcome Y.

[7]:
```
vertices = ['C2', 'C1', 'T', 'M1', 'M2', 'Y']
di_edges = [('C2', 'T'), ('C2', 'M1'), ('C2', 'M2'), ('C2', 'Y'),
            ('C1', 'T'), ('C1', 'M2'),
            ('T', 'M1'), ('M1', 'M2'), ('M2', 'Y'), ('T', 'Y')]
bi_edges = [('T', 'M2'), ('M1', 'Y')]
G = ADMG(vertices, di_edges, bi_edges)
influence = AutomatedIF(G, 'T', 'Y')
print("beta primal = ", influence.beta_primal_, "\n")
print("beta dual = ", influence.beta_dual_, "\n")
print("np-IF = ", influence.nonparametric_if_, "\n")
print("efficient IF = \n", influence.eff_if_, "\n")
G.draw(direction="LR")
```

```
 Treatment is p-fixable and graph is mb-shielded.

 Available estimators are:

1. Primal IPW (p-ipw)
2. Dual IPW (d-ipw)
3. APIPW (apipw)
4. Efficient APIPW (eff-apipw)
```

(continues on next page)

(continued from previous page)

```
Suggested estimator is Efficient APIPW

beta primal =  I(T=t) x 1/[p(T|C1,C2)p(M2|M1,C1,T,C2)] x _T p(T|C1,C2)p(M2|M1,C1,T,
↪C2) x Y

beta dual =  [p(M1|T=t,C2)/p(M1|T,C2)] x E[Y|T=t,M1,M2,C2]

np-IF =  E[primal or dual|C2] - E[primal or dual] + E[primal or dual|C1,C2] -␣
↪E[primal or dual|C2] + E[dual|C1,T,C2] - E[dual|C2,C1] + E[primal|M1,C1,T,C2] -␣
↪E[primal|C2,C1,T] + E[dual|M2,C1,C2,M1,T] - E[dual|C2,C1,T,M1] + E[primal|M2,C1,C2,
↪M1,Y,T] - E[primal|C2,C1,T,M1,M2]

efficient IF =
 E[primal or dual|C2] - E[primal or dual] + E[primal or dual|C1] - E[primal or dual]␣
↪+ E[dual|C1,T,C2] - E[dual|C1,C2] + E[primal|M1,T,C2] - E[primal|T,C2] + E[dual|M1,
↪M2,C1,T,C2] - E[dual|M1,C1,T,C2] + E[primal|M1,Y,M2,T,C2] - E[primal|T,M1,M2,C2]
```

[7]:

2.2 Causal Inference with Graphical Models

Broadly speaking, in causal inference we are interested in using data from observational studies (as opposed to randomized controlled trials), in order to answer questions of the following form – What is the causal effect of setting via an **intervention** (possibly contrary to fact) some variable A to value a on some outcome Y. That is, causal inference concerns itself with the expression of counterfactual distributions obtained from observational distributions through the process of interventions. Causal effects may sometimes be determined using randomized controlled trials (RCTs), but these are often expensive or unethical – think forcing a random portion of the population to smoke to determine the effect of smoking on lung cancer. Also note that RCTs themselves may be subject to all kinds of messiness such as missing data and dropout from clinical trials so that even if data from an RCT were available, causal inference helps eliminate bias in the estimate of the causal effect arising from such messiness.

The goal of causal inference then is to determine whether such causal effects can be teased out from purely observational data or messy data. When such counterfactual quantities can be written as functions of the observed data, they are said to be **identified**. Not all causal effects can be identified, but many can – if we impose enough assumptions on the observed data distribution. Assumptions then, is the price we pay for identifiability, and so being transparent about our assumptions when making causal claims is paramount. This is where **graphical models** allow us to concisely, and intuitively state our set of assumptions. In the words of Miguel Hernán, "Graphical models allow us to draw our assumptions before our conclusions."

A graph $\mathcal{G}$ is defined by a set of vertices V and edges that could be directed (e.g., $X \rightarrow$ Y) interpreted as X being a direct cause of Y, bidirected (e.g., $X \leftrightarrow Y$) interpreted as the existence of unmeasured common causes of both X and Y ($X \leftarrow U \rightarrow Y$), or undirected (e.g., $X - Y$) interpreted as a symmetric relationship between X and Y. In Ananke, we currently support acyclic graphical models i.e., we do not allow for cyclic causality. So how exactly do graphs help us encode assumptions? Consider the **Directed Acyclic Graph (DAG)** below.

[1]:
```
# imports for this notebook

from ananke import graphs
from ananke import identification

vertices = ['A', 'M', 'Y', 'C']
edges = [('A', 'M'), ('M', 'Y'), ('C', 'A'), ('C', 'Y')]
dag = graphs.DAG(vertices, edges)
# the direction LR is just to lay the vertices of the graph out from left to right
```

(continues on next page)

(continued from previous page)

```
# instead of top to bottom which is the default
dag.draw(direction='LR')
```

[1]:

All interventional distributions in causal models of a DAG are identified by application of the g-formula. For example, $p(Y|\mathrm{do}(a))$ written as $p(Y(a))$ in potential outcomes notation which we will use more commonly here, is identified as

$$p(Y(a)) = \sum_{C,M} p(Y|M,C)p(M|A)p(C)|_{A=a}.$$

But as alluded to earlier, this comes at a price. The assumption made by a DAG (in addition to conditional independence constraints that can be read off by the graphical criterion of d-separation) is that of causal sufficiency, i.e., the absence of bidirected edges assumes the absence of unmeasured common causes. If we are unwilling to assume causal sufficiency, we can impose generalized independence constraints given by Nested Markov models of an **Acyclic Directed Mixed Graph (ADMG)** representing marginals of DAG models. The ADMG corresponding to the scenario where C is unobserved is shown below.

[2]:
```
vertices = ['A', 'M', 'Y']
di_edges = [('A', 'M'), ('M', 'Y')]
bi_edges = [('A', 'Y')]
admg = graphs.ADMG(vertices, di_edges=di_edges, bi_edges=bi_edges)
admg.draw(direction='LR')
```

[2]:

In this case $p(Y(a))$ is still identified but not all interventional distributions of an ADMG are identified. A sound and complete algorithm for identification in ADMGs is known due to Tian and Pearl, and Shpitser and Pearl. In Ananke, we use a purely graphical formulation of the aforementioned works that uses the fixing operation (ϕ) on nested Markov models, in order to answer identification queries. The following is an example of identification using Ananke for the front-door graph shown above.

[3]:
```
outcomes = ['Y']
treatments = ['A']
id_pya = identification.OneLineID(graph=admg, treatments=treatments,␣
↪outcomes=outcomes)
print('Identifed =', id_pya.id(), '; Functional =', id_pya.functional())
```

```
{('M',): ['Y', 'A'], ('Y',): ['M', 'A']}
Identifed = True ; Functional = M AY(p(V);G) AM(p(V);G)
```

Another implicit assumption made in the above examples was that our data consisted of independent and identically distributed (iid) samples. **Chain Graphs (CGs)**, consisting of directed and undirected edges such that there are no partially directed cycles, have emerged as a popular graphical model of **interference** (a violation of the independence assumption). Consider a scenario in which our population consists of dyads (say couples) capable of influencing each others outcomes. This may be depicted as shown below.

[4]:
```
vertices = ['C1', 'A1', 'M1', 'Y1', 'C2', 'A2', 'M2', 'Y2']
di_edges = [('A1', 'M1'), ('M1', 'Y1'), ('C1', 'A1'), ('C1', 'Y1'),
            ('A2', 'M2'), ('M2', 'Y2'), ('C2', 'A2'), ('C2', 'Y2'),
            ('M1', 'Y2'), ('M2', 'Y1')]
ud_edges = [('M1', 'M2')]
cg = graphs.CG(vertices, di_edges=di_edges, ud_edges=ud_edges)
cg.draw(direction='LR')
```

[4]:

All interventional distributions of a CG are identified by a CG version of the g-formula. Note however, that chain graphs assume causal sufficiency (lack of bidirected edges). If we'd like to further relax this assumption, we use

Segregated Graphs (SGs). Once again, if we do not observe Cs in the CG shown above, we obtain the SG below. A sound and complete algorithm for identification in SGs was provided by Sherman and Shpitser.

[5]:
```
vertices = ['A1', 'M1', 'Y1', 'A2', 'M2', 'Y2']
di_edges = [('A1', 'M1'), ('M1', 'Y1'),
            ('A2', 'M2'), ('M2', 'Y2'),
            ('M1', 'Y2'), ('M2', 'Y1')]
ud_edges = [('M1', 'M2')]
bi_edges = [('A1', 'Y1'), ('A2', 'Y2')]
sg = graphs.SG(vertices, di_edges=di_edges, ud_edges=ud_edges, bi_edges=bi_edges)
sg.draw(direction='LR')
```

[5]:

We end this section by providing a hierarchy of graphical models (shown below). Two of the models in this hierarchy – bidirected graphs (BGs), and undirected graphs (UGs) are often not considered by themselves in causal analysis but are building blocks of more complicated graphical models that are, as reflected in the graph hierarchy. At the top of the hierarchy, are SGs comprised of directed, bidirected, and undirected edges. An SG with no undirected edges is an ADMG, and an SG with no bidirected edges is a CG. A BG is an ADMG with no directed edges, and a UG is a CG with no directed edges. Finally, a DAG is an ADMG with no bidirected edges *or* alternatively a CG with no undirected edges. As we go further up in the hierarchy we relax more assumptions, but identification theory becomes trickier, and so does estimation. For example, a generalized likelihood for BGs, ADMGs, and SGs is not known.

[6]:
```
# a graph of the hierarchy of graphs
vertices = ['SG', 'ADMG', 'CG', 'DAG', 'BG', 'UG']
edges = [('SG', 'ADMG'), ('SG', 'CG'), ('ADMG', 'DAG'), ('CG', 'DAG'), ('ADMG', 'BG'),
↪ ('CG', 'UG')]
graphs.UG(vertices, edges).draw()
```

[6]:

2.3 Semiparametric Inference For Causal Effects

In this section, we discuss how to estimate the effect of treatment T on outcome Y commonly expressed through the use of counterfactuals of the form $Y(t)$ or $Y|\text{do}(t)$ – which reads as the potential outcome Y had treatment been assigned to t. We follow the estimation procedure outlined in our work Semiparametric Inference For Causal Effects In Graphical Models With Hidden Variables (Bhattacharya, Nabi, & Shpitser, 2020). If the outcome is continuous, the effect is typically captured by the average causal effect (ACE) defined as,

$\psi = E[Y(1)] - E[Y(0)]$, (when Y is continuous.)

When Y is binary, the effect is typically reported as the log of odds ratios (LOR) as follows.

$\psi = \log(\frac{p(Y(1)=1)/p(Y(1)=0)}{p(Y(0)=1)/p(Y(0)=0)})$, (when Y is binary.)

In Ananke, we consider identification and estimation of these quantities using the language of causal graphs. We illustrate these through a series of examples with increasing levels of complications. The good news is, all the user needs to specify is the **data (as a Pandas data frame)** and a story for what they believe the world looks like in the form of an **acyclic directed mixed graph (ADMG)** where each bidirected edge encodes unmeasured common confounders. Ananke will then provide suggestions for the best estimator given the graphical story provided by the user. If it is not possible to compute the ACE/LOR given this story (i.e., the quantity is not *identified*) then Ananke will also inform the user if this is the case.

Following are the necessary packages that need to be imported.

[1]:
```
from ananke.graphs import ADMG
from ananke.estimation import CausalEffect
from ananke.datasets import load_afixable_data, load_conditionally_ignorable_data,␣
↪load_frontdoor_data
```

(continues on next page)

(continued from previous page)

```
from ananke.estimation import AutomatedIF
import numpy as np
```

Let's say we are studying the efficacy of a new antiretroviral therapy vs. an old one. Call this the treatment T where $T = 1$ represents receiving the new drug and $T = 0$ represents receiving the old. Our outcome of interest is the patients CD4 counts post treatment.

2.3.1 Estimation via Adjustment

As a first pass, an analyst might check the causal effect under the causal graphical model where all common confounders of the treatment and outcome are measured. A highly simplistic story would be as follows. The initial viral load of the patient affects both which treatment the patient is assigned to as well as their final outcome. This is represented graphically as follows. Just be careful not to have spaces in your variable names, the machine learning packages underlying Ananke are not compatible with that.

[2]:
```
vertices = ['ViralLoad', 'T', 'CD4']
di_edges = [('ViralLoad', 'T'), ('ViralLoad', 'CD4'), ('T', 'CD4')]
G = ADMG(vertices, di_edges)
G.draw(direction="LR")
```

[2]:

We now load some toy data on these three variables, and ask Ananke to estimate the effect. We see that as soon as we instantiate the `CausalEffect` object, Ananke offers a list of estimators that are available under the specified causal graph. Further, it recommends one that achieves the semiparametric efficiency bound, i.e., the estimator with the lowest variance.

[3]:
```
ace_obj = CausalEffect(graph=G, treatment='T', outcome='CD4')  # setting up the␣
↪CausalEffect object
```

```
 Treatment is a-fixable and graph is mb-shielded.

 Available estimators are:

1. IPW (ipw)
2. Outcome regression (gformula)
3. Generalized AIPW (aipw)
4. Efficient Generalized AIPW (eff-aipw)

Suggested estimator is Efficient Generalized AIPW
```

In this case, it recommends *Efficient Generalized AIPW* which is to say, that the estimator used to compute the effect in this case looks a lot like Augmented IPW. Further, Ananke uses semiparametric theory in order to provide an estimator that achieves the lowest asymptotic variance. Turns out, the efficient estimator in this case is trivial, but the next case demonstrates a harder scenario.

Given the list of estimators, it is up to the user to specify what estimators they want to work with. For instance, if the user decides to use the efficient generalized AIPW, they only need to use the keyword given in front of it, i.e., `eff-aipw`, when computing the effect. All the nuisance models are fit using generalized linear models provided in the `statsmodels`. Users interested in using different modeling approaches can refer to the documentation on accessing the functional form of the influence functions at the end of this notebook.

[4]:
```
data = load_conditionally_ignorable_data() # loading the data
ace = ace_obj.compute_effect(data, "eff-aipw") # computing the effect
print("ace = ", ace, "\n")
```

```
ace =  0.9848858526281847
```

A realistic model will rarely be so simple. So what if we want to build a more complicated model such as the one shown below. Where, bidirected edges ($\leftrightarrow$) are used to encode unmeasured common confounders. For example, we hypothesize that we do not have information on some unmeasured common confounders between an individual's income and treatment assignment.

[5]:
```
vertices = ['Income', 'Insurance', 'ViralLoad', 'Education', 'T', 'Toxicity', 'CD4']
di_edges = [('ViralLoad', 'Income'), ('ViralLoad', 'T'), ('ViralLoad', 'Toxicity'),
            ('Education', 'Income'), ('Education', 'T'), ('Education', 'Toxicity'),
            ('Income', 'Insurance'), ('Insurance', 'T'), ('T', 'Toxicity'), ('Toxicity
↪', 'CD4'), ('T', 'CD4')]
bi_edges = [('Income', 'T'), ('Insurance', 'ViralLoad'), ('Education', 'CD4')]
G = ADMG(vertices, di_edges, bi_edges)
G.draw(direction="LR")
```

[5]:

[6]:
```
ace_obj = CausalEffect(graph=G, treatment='T', outcome='CD4')  # setting up the ACE␣
↪object
```

```
 Treatment is a-fixable and graph is mb-shielded.

 Available estimators are:

1. IPW (ipw)
2. Outcome regression (gformula)
3. Generalized AIPW (aipw)
4. Efficient Generalized AIPW (eff-aipw)

Suggested estimator is Efficient Generalized AIPW
```

In the following, we print the outputs of all four estimators.

[7]:
```
data = load_afixable_data() # load some pre-simulated data
ace_ipw = ace_obj.compute_effect(data, "ipw")
ace_gformula = ace_obj.compute_effect(data, "gformula")
ace_aipw = ace_obj.compute_effect(data, "aipw")
ace_eff = ace_obj.compute_effect(data, "eff-aipw")
print("ace using IPW = ", ace_ipw)
print("ace using g-formula = ", ace_gformula)
print("ace using AIPW = ", ace_aipw)
print("ace using efficient AIPW = ", ace_eff)
```

```
ace using IPW =  0.4917151858666866
ace using g-formula =  0.49680779965965227
ace using AIPW =  0.5005117929752627
ace using efficient AIPW =  0.4917151858667226
```

In addition, the user can run bootstraps and obtain $(1-\alpha)*100\%$ confidence level for the causal effect point estimate. The user needs to specify two arguments: number of bootstraps `n_bootstraps` and the significance level α `alpha`. The confidence interval can then be obtained via bootstrap percentile intervals, as described in All of Nonparametric Statistics. In this case, the call to the `compute_effect` returns three values: the first corresponds to the effect

computed on the original data, the second and third are the pre-specified lower and upper quantiles, i.e., $(\frac{\alpha}{2}, 1 - \frac{\alpha}{2})$. The default value for α is set to be 0.05. We illustrate this through the following toy example.

[8]:
```
np.random.seed(0)  # setting the seed to get consistent numbers in the documentation
ace, Ql, Qu = ace_obj.compute_effect(data, "eff-aipw", n_bootstraps=5, alpha=0.05)
print(" ace = ", ace, "\n",
      "lower quantile = ", Ql, "\n",
      "upper quantile = ", Qu)
```

```
 ace =  0.4917151858667208
 lower quantile =  0.45738153193197817
 upper quantile =  0.5898403691123565
```

2.3.2 Estimation beyond Adjustment

The previous examples showed that there exists a large class of causal graphs for which we can obtain the causal effect via covariate adjustment. But what happens when covariate adjustment fails? It turns out, that there is an even wider class of models for which we can obtain the causal effect, *even* when we cannot block all backdoor paths from the treatment to the outcome. Revisiting our running example on antiretroviral therapies: Let's first see an example where the causal effect is not (non-parametrically) identified from the observed data.

[9]:
```
vertices = ['ViralLoad', 'T', 'CD4']
di_edges = [('ViralLoad', 'T'), ('ViralLoad', 'CD4'), ('T', 'CD4')]
bi_edges = [('T', 'CD4')]
G = ADMG(vertices, di_edges, bi_edges)
G.draw(direction="LR")
```

[9]:

You will notice that this differs ever so slightly yet crucially from our first example in that we as analysts now believe that there are unmeasured confounders between the treatment (T) and the outcome (CD4). If we ask Ananke to estimate the causal effect under this model of the world, it will declare (correctly) that the effect is not identified non-parametrically under the given model.

[10]:
```
ace_obj = CausalEffect(graph=G, treatment='T', outcome='CD4')
```

```
Effect is not identified!
```

So what can we do then? Well, if we believe another model wherein the effect of the treatment is mediated completely through some biologically known mechanism whose by-product we can measure. Say this occurs through a reduction in viral load, and that it addition to a baseline measurement we have a second follow-up measurement.

[11]:
```
vertices = ['ViralLoad1', 'T', 'ViralLoad2', 'CD4']
di_edges = [('ViralLoad1', 'T'), ('ViralLoad1', 'CD4'), ('ViralLoad1', 'ViralLoad2'),
            ('T', 'ViralLoad2'), ('ViralLoad2', 'CD4')]
bi_edges = [('T', 'CD4')]
G = ADMG(vertices, di_edges, bi_edges)
G.draw(direction="LR")
```

[11]:

It so happens that this is quite similar to the front-door graph that may be familiar to some users, and the effect is now identified! Here is how we can estimate it using Ananke.

[12]:
```
ace_obj = CausalEffect(graph=G, treatment='T', outcome='CD4')  # setting up the ACE␣
↪object
data = load_frontdoor_data()
ace = ace_obj.compute_effect(data, "eff-apipw")
print("ace = ", ace)
```

```
 Treatment is p-fixable and graph is mb-shielded.

 Available estimators are:

1. Primal IPW (p-ipw)
2. Dual IPW (d-ipw)
3. APIPW (apipw)
4. Efficient APIPW (eff-apipw)

Suggested estimator is Efficient APIPW

ace =  -0.7813625198154692
```

2.3.3 Useful Graphical Criteria

In the previous section, we have seen that causal effects are not always identified, and if they are, they could be identified and estimated in different ways. Here we provide some simple graphical criteria that a user should keep in mind while creating their graphical model that may help guide them to obtain the best estimator possible. These graphical criteria and their corresponding estimators can be found in our work Semiparametric Inference For Causal Effects In Graphical Models With Hidden Variables.

Adjustment Fixability

If the treatment of interest (say T) has no bidirected path to any of its descendants (i.e., there is no $T \leftrightarrow \cdots \leftrightarrow X$ for any X that is a descendant of T), then we say the treatment is adjustment or a-fixable. In this case, Ananke will find estimators that are based on IPW and covariate adjustment as well as a semiparametric estimator that has the form of **Augmented IPW**. Ananke will always recommend **Generalized AIPW** as the preferred estimator due to its double robustness property. Further, if the graph satisfies a property known as the mb-shielded (Markov blanket shielded) property, Ananke will be able to provide the most **efficient influence function** based estimator – one that achieves the semiparametric efficiency bound, and recommend using this instead when possible. Below is an example of a causal graph where the treatment is a-fixable and the graph is mb-shielded.

[13]:
```
vertices = ['Income', 'Insurance', 'ViralLoad', 'Education', 'T', 'Toxicity', 'CD4']
di_edges = [('ViralLoad', 'Income'), ('ViralLoad', 'T'), ('ViralLoad', 'Toxicity'),
            ('Education', 'Income'), ('Education', 'T'), ('Education', 'Toxicity'),
            ('Income', 'Insurance'), ('Insurance', 'T'), ('T', 'Toxicity'), ('Toxicity
↪', 'CD4'), ('T', 'CD4')]
bi_edges = [('Income', 'T'), ('Insurance', 'ViralLoad'), ('Education', 'CD4')]
G = ADMG(vertices, di_edges, bi_edges)
ace_obj = CausalEffect(graph=G, treatment='T', outcome='CD4')  # setting up the ACE␣
↪object
G.draw(direction="LR")
```

```
 Treatment is a-fixable and graph is mb-shielded.

 Available estimators are:

1. IPW (ipw)
2. Outcome regression (gformula)
3. Generalized AIPW (aipw)
4. Efficient Generalized AIPW (eff-aipw)
```

(continues on next page)

(continued from previous page)

```
Suggested estimator is Efficient Generalized AIPW
```

[13]:

Primal Fixability

If the treatment of interest (say T) has no bidirected path to any of its **children** (i.e., there is no $T \leftrightarrow \cdots \leftrightarrow X$ for any X that is a child of T), then we say the treatment is primal or p-fixable. Notice that due to the graphical criterion being related to children as opposed to all descendants, p-fixability is strictly more general than a-fixability. Further, p-fixability covers many important and popular classes of causal graphs such as the front-door graph.

When T is p-fixable, Ananke finds estimators based on Primal IPW, Dual IPW, and **Augmented Primal IPW**. Ananke will always recommend Augmented Primal IPW as the preferred estimator. As before, when the graph is mb-shielded, Ananke is also able to provide the efficient influence function based estimator and will instead recommend **Efficient Augmented Primal IPW**. Below is an example of an mb-shielded graph where T is p-fixable.

[14]:
```
vertices = ['ViralLoad', 'Income', 'T', 'Toxicity', 'Adherence', 'CD4']
di_edges = [('ViralLoad', 'T'), ('ViralLoad', 'Toxicity'), ('ViralLoad', 'Adherence'),
↪ ('ViralLoad', 'CD4'),
            ('Income', 'T'), ('Income', 'Adherence'),
            ('T', 'Toxicity'), ('Toxicity', 'Adherence'), ('Adherence', 'CD4'), ('T',
↪'CD4')]
bi_edges = [('T', 'Adherence'), ('Toxicity', 'CD4')]
G = ADMG(vertices, di_edges, bi_edges)
ace_obj = CausalEffect(graph=G, treatment='T', outcome='CD4')  # setting up the ACE␣
↪object
G.draw(direction="LR")
```

```
 Treatment is p-fixable and graph is mb-shielded.

 Available estimators are:

1. Primal IPW (p-ipw)
2. Dual IPW (d-ipw)
3. APIPW (apipw)
4. Efficient APIPW (eff-apipw)

Suggested estimator is Efficient APIPW
```

[14]:

Nested Fixability

Finally, this if the treatment is neither a-fixable nor p-fixable, Ananke checks if it is what we call nested fixable. That is, the treatment is nested fixable if running Algorithm 2 in our paper succeeds in finding an estimator. Ananke will then allow the user to use either Nested IPW or **Augmented Nested IPW** but its recommendation will be the latter. Below is an example.

[15]:
```
vertices = ['Income', 'Exercise', 'T', 'Toxicity', 'Adherence', 'CD4', 'ViralLoad']
di_edges = [('ViralLoad', 'T'), ('ViralLoad', 'CD4'), ('Income', 'T'), ('Exercise',
↪'CD4'),
            ('T', 'Toxicity'), ('T', 'CD4'), ('Toxicity', 'Adherence'), ('Adherence',
↪'CD4')]
bi_edges = [('Income', 'Toxicity'), ('Exercise', 'Income'), ('Exercise', 'T'),
```

(continues on next page)

(continued from previous page)

```
            ('ViralLoad', 'Adherence'), ('ViralLoad', 'CD4')]
G = ADMG(vertices, di_edges, bi_edges)
ace_obj = CausalEffect(graph=G, treatment='T', outcome='CD4')  # setting up the ACE␣
↪object
G.draw(direction="LR")
```

```
 Effect is identified.

 Available estimators:

1. Nested IPW (n-ipw)
2. Augmented NIPW (anipw)

Suggested estimator is Augmented NIPW
```

[15]:

Note that naturally the class of models that satisfy a-fixability, p-fixability and nested fixability are subsets of each other. That is,

$$\text{adjustment fixability} \subset \text{primal fixability} \subset \text{nested fixability}.$$

Thus, if the user wanted to use Nested IPW for a-fixable graphs, Ananke would not stop you. However, the reverse will result in an error from Ananke. Further, if the effect is not identified, Ananke will not allow the user to use any of the estimators.

2.3.4 Accessing the Functional Form of the Influence Function

For advanced users, we also provide tools to access the functional form of the influence funciton directly. If the treatment is **adjustment fixable**, the code spits out the functional form of the non-parametric influence function given in Theorem 2 in our paper. If the treatment is **primal fixable**, the code provides the functional form of the non-parametric influence function given in Theorem 11 in our paper. Further, if the graph is **mb-shielded**, we provide the respective efficient influence function. In the following examples we are interested in the effect of treatment T on outcome Y.

[16]:
```
vertices = ['Z1', 'Z2', 'C2', 'C1', 'T', 'M', 'Y']
di_edges = [('C2', 'Z1'), ('C2', 'T'), ('C2', 'M'),
            ('C1', 'Z1'), ('C1', 'T'), ('C1', 'M'),
            ('Z1', 'Z2'), ('Z2', 'T'), ('T', 'M'), ('M', 'Y'), ('T', 'Y')]
bi_edges = [('Z1', 'T'), ('Z2', 'C2'), ('C1', 'Y')]
G = ADMG(vertices, di_edges, bi_edges)
influence = AutomatedIF(G, 'T', 'Y')
print("np-IF = ", influence.nonparametric_if_, "\n")
print("beta primal = ", influence.beta_primal_, "\n")
print("efficient IF = \n", influence.eff_if_, "\n")
G.draw(direction="LR")
```

```
 Treatment is a-fixable and graph is mb-shielded.

 Available estimators are:

1. IPW (ipw)
2. Outcome regression (gformula)
3. Generalized AIPW (aipw)
```

(continues on next page)

(continued from previous page)

```
4. Efficient Generalized AIPW (eff-aipw)

Suggested estimator is Efficient Generalized AIPW

np-IF =  I(T=t) x 1/p(T|C2,Z1,C1,Z2) x (Y - E[Y|T=t,C2,Z1,C1,Z2]) + E[Y|T=t,C2,Z1,C1,
↪Z2] -

beta primal =  I(T=t) x 1/p(T|C2,Z1,C1,Z2) x Y

efficient IF =
 E[primal|C1] - E[primal] + E[primal|T,M,Y,C1] - E[primal|T,M,C1] + E[primal|C2] -␣
↪E[primal] + E[primal|C2,Z1,C1] - E[primal|C2,C1] + E[primal|T,M,C2,C1] - E[primal|T,
↪C2,C1] + E[primal|Z1,C2,Z2] - E[primal|C2,Z1]
```

[16]:

[17]:
```
vertices = ['C2', 'C1', 'T', 'M1', 'M2', 'Y']
di_edges = [('C2', 'T'), ('C2', 'M1'), ('C2', 'M2'), ('C2', 'Y'),
            ('C1', 'T'), ('C1', 'M2'),
            ('T', 'M1'), ('M1', 'M2'), ('M2', 'Y'), ('T', 'Y')]
bi_edges = [('T', 'M2'), ('M1', 'Y')]
G = ADMG(vertices, di_edges, bi_edges)
influence = AutomatedIF(G, 'T', 'Y')
print("beta primal = ", influence.beta_primal_, "\n")
print("beta dual = ", influence.beta_dual_, "\n")
print("np-IF = ", influence.nonparametric_if_, "\n")
print("efficient IF = \n", influence.eff_if_, "\n")
G.draw(direction="LR")
```

```
 Treatment is p-fixable and graph is mb-shielded.

 Available estimators are:

1. Primal IPW (p-ipw)
2. Dual IPW (d-ipw)
3. APIPW (apipw)
4. Efficient APIPW (eff-apipw)

Suggested estimator is Efficient APIPW

beta primal =  I(T=t) x 1/[p(T|C2,C1)p(M2|T,C2,C1,M1)] x _T p(T|C2,C1)p(M2|T,C2,C1,
↪M1) x Y

beta dual =  [p(M1|T=t,C2)/p(M1|T,C2)] x E[Y|T=t,M2,C2,M1]

np-IF =  E[primal or dual|C2,C1] - E[primal or dual|C1] + E[primal or dual|C1] -␣
↪E[primal or dual] + E[dual|T,C2,C1] - E[dual|C1,C2] + E[primal|T,C2,C1,M1] -␣
↪E[primal|C1,C2,T] + E[dual|T,C1,M1,M2,C2] - E[dual|C1,C2,T,M1] + E[primal|T,C1,M1,
↪M2,Y,C2] - E[primal|C1,C2,T,M1,M2]

efficient IF =
 E[primal or dual|C2] - E[primal or dual] + E[primal or dual|C1] - E[primal or dual]␣
↪+ E[dual|T,C2,C1] - E[dual|C2,C1] + E[primal|T,C2,M1] - E[primal|T,C2] + E[dual|T,
↪C2,C1,M1,M2] - E[dual|T,C2,C1,M1] + E[primal|T,C2,M1,M2,Y] - E[primal|T,M2,C2,M1]
```

[17]:

2.4 Identification With Surrogates

Causal identification is usually framed as a problem where one wishes to recover some causal query $p(Y|do(a))$ from some observed data distribution $p(V)$ (Shpitser and Pearl, 2006; Huang and Valtorta, 2006). Recall $p(Y(a))$ and $p(Y|do(a))$ are equivalent but the do-notation used here is notationally cleaner.

However, recently there has been interest in the scenario where identification is performed with respect to other distributions. An analyst might have access to a set of experiments (where certain variables have been intervened to fixed values), and while the causal query might not be identified in any one experiment, jointly they might suffice. Lee et al., 2019 provide a sound and complete algorithm when the experiments are of the form $p(V \setminus X|do(x))$ for some intervened variables $X = x$.

Lee and Shpitser, 2020 (a different Lee!) extend the applicability of this algorithm by showing that it remains sound and complete for experiments which are ancestral subgraphs, with respect to the original graph under the intervention of the experiment, while recasting the results of Lee et al., 2019 using the one-line ID formulation provided in Richardson et al. 2017.

Following are the necessary packages that need to be imported.

[1]:
```
from ananke import graphs
from ananke import identification
```

Let's say that we are interested in a system represented by the following graph. We can think of $X1$ as a treatment for cardiac disease, $X2$ as a treatment for obesity (say a particular diet), W as blood pressure (which perhaps for the purposes of this toy example is influenced only by the cardiac treatment), and Y as the final health outcome. As a first pass, we may posit the following causal graph, indicating that we only believe unmeasured confounding to exist between a person's treatment assignment and diet. We are interested in the causal query represented by $p(Y|do(x_1, x_2))$.

[2]:
```
vertices = ["X1", "X2", "W", "Y"]
di_edges = [("X1", "W"), ("W", "Y"), ("X2", "Y")]
bi_edges = [("X1", "X2")]
G = graphs.ADMG(vertices, di_edges, bi_edges)
G.draw(direction="TD")
```

[2]:

If we query the `OneLineID` algorithm in Ananke, we will see that this query is indeed identified.

[3]:
```
one_id = identification.OneLineID(graph=G, treatments=['X1', 'X2'], outcomes=['Y'])
one_id.id()
```

[3]:
```
True
```

However, since we are in a clinical healthcare setting, we should expect the presence of more confounding variables. So we update our causal graph to include other hidden confounders between the treatments and outcomes.

[4]:
```
vertices = ["X1", "X2", "W", "Y"]
di_edges = [("X1", "W"), ("W", "Y"), ("X2", "Y")]
bi_edges = [("X1", "X2"), ("X1", "W"), ("X2", "Y")]
G = graphs.ADMG(vertices, di_edges, bi_edges)
G.draw(direction="TD")
```

[4]:

We can now verify that under this model, the query is not identified from the observed data distribution $p(V) := p(Y, W, X_2, X_1)$ using `OnelineID`:

[5]:
```
one_id = identification.OneLineID(graph=G, treatments=['X1', 'X2'], outcomes=['Y'])
one_id.id()
```

[5]:
```
False
```

It seems that we are stuck! However, in the next section we discuss how access to smaller subsets of experimental data on the variables involved may help us identify our causal query.

2.4.1 GID

What if we had access to experiments? We construct two experiments - one where X_1 is fixed to value x_1, and another where X_2 is fixed to value x_2. Perhaps, it is not possible to run an experiment where both X_1 and X_2 are intervened upon (financial reasons, ethical reasons, etc.) but these smaller experiments are indeed possible.

[6]:
```
vertices = ["X1", "X2", "W", "Y"]
di_edges = [("X1", "W"), ("W", "Y"), ("X2", "Y")]
bi_edges = [("X1", "X2"), ("X1", "W"), ("X2", "Y")]
G1 = graphs.ADMG(vertices, di_edges, bi_edges)
G1.fix(["X1"])
G1.draw(direction="TD")
```

[6]:

[7]:
```
vertices = ["X1", "X2", "W", "Y"]
di_edges = [("X1", "W"), ("W", "Y"), ("X2", "Y")]
bi_edges = [("X1", "X2"), ("X1", "W"), ("X2", "Y")]
G2 = graphs.ADMG(vertices, di_edges, bi_edges)
G2.fix(["X2"])
G2.draw(direction="TD")
```

[7]:

It happens that the causal query is indeed identified:

[8]:
```
g_id = identification.OnelineAID(graph=G, treatments=["X1", "X2"], outcomes=["Y"])
```

[9]:
```
g_id.id(experiments=[G1, G2])
```

[9]:
```
True
```

with corresponding identifying functional

[10]:
```
g_id.functional(experiments=[G1, G2])
```

[10]:
```
'W X2,Y p(W,X2,Y | do(X1))X1,W p(W,X1,Y | do(X2))'
```

2.4.2 AID

The astute reader will notice that perhaps we didn't need all of the experimental distributions. Rather, a margin would have sufficed - on the first experiment, we could have marginalized Y and X_2, and still achieved identification. The reason is that the intrinsic set (and its parents) would have been identified anyways. Based on the results provided in Lee and Shpitser, 2020, we consider identification from the following experiment. The first graph is an ancestral subgraph with respect to $G(V(x_1))$. The second graph remains unchanged, as does the graph defining the system we are interested in.

[11]:
```
vertices = ["X1","W"]
di_edges = [("X1", "W")]
bi_edges = [("X1", "W")]
G1 = graphs.ADMG(vertices, di_edges, bi_edges)
G1.fix(["X1"])
G1.draw(direction="TD")
```

[11]:

[12]:
```
a_id = identification.OnelineAID(graph=G, treatments=["X1", "X2"], outcomes=["Y"])
```

[13]:
```
a_id.id(experiments=[G1, G2])
```

[13]:
```
True
```

[14]:
```
a_id.functional(experiments=[G1, G2])
```

[14]:
```
'W  p(W | do(X1))X1,W p(W,X1,Y | do(X2))'
```

The causal query remains identified, but the identification formula has changed. Notably, no fixing operations are needed under the first experiment $p(W|do(X1))$ since it is exactly the required kernel.

2.5 Linear Structural Equation Models

The use of structural equation models and path analysis in causal models associated with a graph can be traced back to the works of Sewall Wright and Trygve Haavelmo. In this notebook we demonstrate how graphs can be used to postulate causal mechanisms in the presence of confounding, and provide examples of model selection, and computation of causal effects, when the causal mechanism arises as a result of a linear system of structural equations.

Given an ADMG $\mathcal{G}$, the linear structural equation model (SEM) with correlated errors (corresponding to unmeasured confounding) associated with $\mathcal{G}$ is defined as the set of multivariate normal distributions with covariance matrices of the form

$\Sigma = (IB)^{-T}\Omega(IB)^{-1}$,

where $\omega_{ij} = \omega_{ji} = 0$ unless $i \leftrightarrow j$ exists in $\mathcal{G}$, and $b_{ij} = 0$ unless $j \rightarrow i$ exists in $\mathcal{G}$. The matrix Ω and therefore Σ is assumed to be positive semi-definite. Since $\mathcal{G}$ is acyclic, B is assumed to be lower triangular.

Following are the packages that need to be imported for the examples below.

[1]:
```
from ananke.graphs import ADMG
from ananke.models import LinearGaussianSEM
import pandas as pd
import numpy as np
import warnings
warnings.filterwarnings('ignore')
```

2.5.1 Arid Graphs

Before going further, it is important to define the class of graphs that we will be working with because statistical models associated with arbitrary Acyclic Directed Mixed Graphs (ADMGs) are not always identified. However, it has been shown by Drton, Foygel, and Sullivant that SEMs associated with a certain class of ADMGs that lack convergent aborescences or C-trees, hence the coinage arid graphs, are everywhere identified. Here, we will focus on linear SEMs associated with **Maximal Arid Graphs (MArGs)** introduced by Shpitser, Evans, and Richardson and use the maximal arid projection defined therein, in order to convert a non-arid ADMG to the corresponding MArG that implies the same conditional and nested Markov constraints.

Consider the non-arid graph $\mathcal{G}$ below.

[2]:
```
# create a non-arid graph
# the bow arc A->C, A<->C is a C-tree
vertices = ["A", "B", "C", "D"]
di_edges = [("A", "B"), ("A", "C"), ("B", "C"), ("C", "D")]
bi_edges = [("A", "C"), ("B", "D")]
G = ADMG(vertices, di_edges=di_edges, bi_edges=bi_edges)
G.draw(direction="LR")
```

[2]:

We can obtain the corresponding maximal arid projection $\mathcal{G}^\dagger$ like so.

[3]:
```
G_arid = G.maximal_arid_projection()
G_arid.draw(direction="LR")
```

[3]:

2.5.2 Fitting and Causal Effect Estimation

We now demonstrate how to fit data originating from a system of linear structural equation models associated with an ADMG, as defined in the first section of the notebook. We first generate data according to a linear SEM associated with the MArG $\mathcal{G}^\dagger$ above.

[4]:
```
# define number of samples and number of vertices
N = 5000
dim = 4

# define the Omega matrix
omega = np.array([[1, 0, 0, 0],
                  [0, 1, 0, 0.8],
                  [0, 0, 1, 0],
                  [0, 0.8, 0, 1]])

# define the B matrix
beta = np.array([[0, 0, 0, 0],
                 [3, 0, 0, 0],
                 [1.2, -1, 0, 0],
                 [0, 0, 2.5, 0]])


# generate data according to the graph
true_sigma = np.linalg.inv(np.eye(dim) - beta) @ omega @ np.linalg.inv((np.eye(dim) -␣
↪beta).T)
X = np.random.multivariate_normal([0] * dim, true_sigma, size=N)
data = pd.DataFrame({"A": X[:, 0], "B": X[:, 1], "C": X[:, 2], "D": X[:, 3]})
```

We then check that we are able to recover the true parameters of the data generating distribution. The **LinearGaussianSEM** class has a special draw functionality to draw the associated MArG and label the parameters corresponding to each edge.

[5]:
```
model = LinearGaussianSEM(G_arid)
model.fit(data)
model.draw(direction="LR")
```

[5]:

We now compute some causal effects through simple path analysis. Consider the total causal effect of D on A and A on D. The former is zero as expected, and the latter is obtained as the sum of contributions through all directed paths from A to D and should be close to the true value of -4.5.

[6]:
```
print("E[A(d)] =", str(model.total_effect(["D"], ["A"])))
print("E[D(a)] =", str(model.total_effect(["A"], ["D"])))
```

```
E[A(d)] = 0
E[D(a)] = -4.511181195610503
```

To demonstrate what could go wrong when we use a non-arid graph. Consider fitting the same data utilizing a model associated with the non-arid graph $\mathcal{G}$ from before. We now see that our coefficient and causal estimates are now completely awry.

[7]:
```
model = LinearGaussianSEM(G)
model.fit(data)
print("E[D(a)] =", str(model.total_effect(["A"], ["D"])))
model.draw(direction="LR")
```

```
E[D(a)] = 39.50996578597202
```

[7]:

2.5.3 Model Selection

How do we pick the correct graphical model? Linear Gaussian SEMs associated with MArGs are **curved exponential families** and thus we can use the BIC score to test for goodness of fit. The graph $\mathcal{G}^\dagger$ implies a Verma constraint (a dormant conditional independence) where $A \amalg D$ in the kernel and associated CADMG obtained by fixing C. More formally,

$$\phi_C\big(p(V); \mathcal{G}^\dagger\big) \equiv \frac{p(A,B,C,D)}{p(C|\text{mb}(C))} = p(A,B)p(D|C),$$

where $\text{mb}(C)$ is the Markov blanket of C. Consider a graph $\mathcal{G}_s$ which is the same as $\mathcal{G}^\dagger$ but with an additional directed edge $A \to D$. There are no conditional independences or Vermas implied by $\mathcal{G}_s$. That is, the model associated with $\mathcal{G}_s$ is a super model of the true model and contains one extra parameter associated with the extra directed edge. When choosing between these two hypotheses, the BIC should prefer $\mathcal{G}^\dagger$ as it is more parsimonious. Let's look at the comparison.

[8]:
```
di_edges = [("A", "B"), ("A", "C"), ("B", "C"), ("C", "D"), ("A", "D")]
bi_edges = [("B", "D")]
G_saturated = ADMG(vertices, di_edges=di_edges, bi_edges=bi_edges)

# calculate BIC for the true model
model_truth = LinearGaussianSEM(G_arid)
model_truth.fit(data)
bic_truth = 2*model_truth.neg_loglikelihood(data) + model_truth.n_params*np.log(N)

# calculate BIC for the saturated model
model_saturated = LinearGaussianSEM(G_saturated)
model_saturated.fit(data)
bic_saturated = 2*model_saturated.neg_loglikelihood(data) + model_saturated.n_
↪params*np.log(N)

print("BIC of the true model", bic_truth)
print("BIC of the saturated model", bic_saturated)
print("The BIC of the true model should be lower (better) than the saturated model.",␣
↪bic_truth < bic_saturated)
```

```
BIC of the true model 14740.955046150859
BIC of the saturated model 14745.262068032296
The BIC of the true model should be lower (better) than the saturated model. True
```

What if we pick a (slightly) incorrect model where we posit that the observed correlation between B and D is due to a directed edge $B \to D$ instead of confounding? The BIC should prefer any super model of the truth to an incorrect model.

[9]:
```
di_edges = [("A", "B"), ("B", "C"), ("C", "D"), ("B", "D"), ("A", "C")]
bi_edges = []
G_incorrect = ADMG(vertices, di_edges=di_edges, bi_edges=bi_edges)

# calculate BIC for incorrect model
model_incorrect = LinearGaussianSEM(G_incorrect)
model_incorrect.fit(data)
bic_incorrect = 2*model_incorrect.neg_loglikelihood(data) + model_incorrect.n_
↪params*np.log(N)

print("BIC of the incorrect model", bic_incorrect)
print("The BIC of the true model should be lower (better) than the incorrect model.",␣
↪bic_truth < bic_incorrect)
model_incorrect.draw(direction="LR")
```

```
BIC of the incorrect model 19257.73272661977
The BIC of the true model should be lower (better) than the incorrect model. True
```

[9]:

Finally, we consider the most interesting scenario – a model that encodes the exact same nested Markov constraints. It turns out that two nested Markov equivalent models will be statistically indistinguishable and their BIC scores will be exactly the same.

[10]:
```
di_edges = [("A", "C"), ("B", "C"), ("C", "D")]
bi_edges = [("A", "B"), ("B", "D")]
G_equivalent = ADMG(vertices, di_edges=di_edges, bi_edges=bi_edges)

# calculate BIC for the nested Markov equivalent model
model_equivalent = LinearGaussianSEM(G_equivalent)
model_equivalent.fit(data)
bic_equivalent = 2*model_equivalent.neg_loglikelihood(data) + model_equivalent.n_
↪params*np.log(N)

print("BIC of the equivalent model", bic_equivalent)
print("The difference in BIC =",
      round(abs(bic_equivalent - bic_truth), 2),
      "between equivalent models should be indistinguishable.")
```

```
BIC of the equivalent model 14740.955046150926
The difference in BIC = 0.0 between equivalent models should be indistinguishable.
```

What we've shown here is that the BIC score can be used to select between different hyoptheses. In DAG models, a greedy search procedure was posited by Chickering that surprisingly yields a globally optimal DAG model while greedily searching through the space of equivalence classes of DAGs. Such a greedy search procedure for nested Markov models is still an open problem – indeed the question of nested Markov equivalence in itself is still an open problem.

2.6 Citation

If you enjoyed this package, we would appreciate the following citations:

2.7 Contributors

- Rohit Bhattacharya
- Jaron Lee
- Razieh Nabi

2.8 Ananke Graphs

ananke.graphs

2.9 Ananke Identification

ananke.identification

2.10 Ananke Estimation

ananke.estimation

2.11 Ananke Models

ananke.models

2.12 Indices and Tables

- genindex
- modindex
- search

Bibliography

[BNS20] Rohit Bhattacharya, Razieh Nabi, and Ilya Shpitser. Semiparametric inference for causal effects in graphical models with hidden variables. *arXiv preprint arXiv:2003.12659*, 2020.

[LS20] Jaron J. R. Lee and Ilya Shpitser. Identification Methods With Arbitrary Interventional Distributions as Inputs. *arXiv preprint arXiv:2004.01157 [cs, stat]*, 2020.

[NBS20] Razieh Nabi, Rohit Bhattacharya, and Ilya Shpitser. Full law identification in graphical models of missing data: completeness results. *arXiv preprint arXiv:2004.04872*, 2020.