Lecture 11: Counting on Graphs
Modeling Social Data, Spring 2019
Columbia University

April 12, 2019

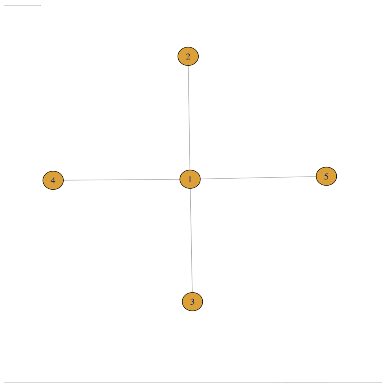
Notes from alk2225

1 An intro to igraph

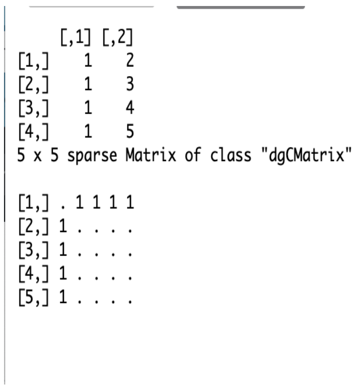
When it comes to graph analysis in R, igraph is a very popular tool.
An example of generating a star graph:

```
star <- graph.star(5, mode="undirected", center=1)
plot(star)
get.edgelist(star)
get.adjacency(star)
```

Running this code, you'll get a visual representation of the star graph, plus edge and adjacency list representations (remember those?). You'll get a graph looking like this:



and representations looking like this:



```
     [,1] [,2]
[1,]    1    2
[2,]    1    3
[3,]    1    4
[4,]    1    5
5 x 5 sparse Matrix of class "dgCMatrix"

[1,] . 1 1 1 1
[2,] 1 . . . .
[3,] 1 . . . .
[4,] 1 . . . .
[5,] 1 . . . .
```

In case it's not clear, the edge list representation of an igraph is an |E| by 2 matrix, where |E| is the number of edges and the 2 comes from needing to list both nodes in an edge. The edges are assigned in ascending order based on the node numbering.

Adjacency lists are very intuitive. On a side note, since they often have a lot of unused space, they are represented using a sparse matrix to save memory.

There are other representations such as lattice and ring you can play with as well. One of the more interesting ones worth playing with is the Watts-Strogatz models in watts.strogatz.game. Here's a Wikipedia article explaining more: Watts-Strogatz network model

2 Using igraph to compute degree distributions

Degree distributions can tell us a lot of stuff. For example, in friend networks, the greater the degree of a person, the more popular the person could be. And they do more than just tell us things on an individual level. They can teach us important information about entire populations.

Computing degree distribution just requires a little playing around with the tidyverse. We'll show an example using the Wiki degree dist dataset:

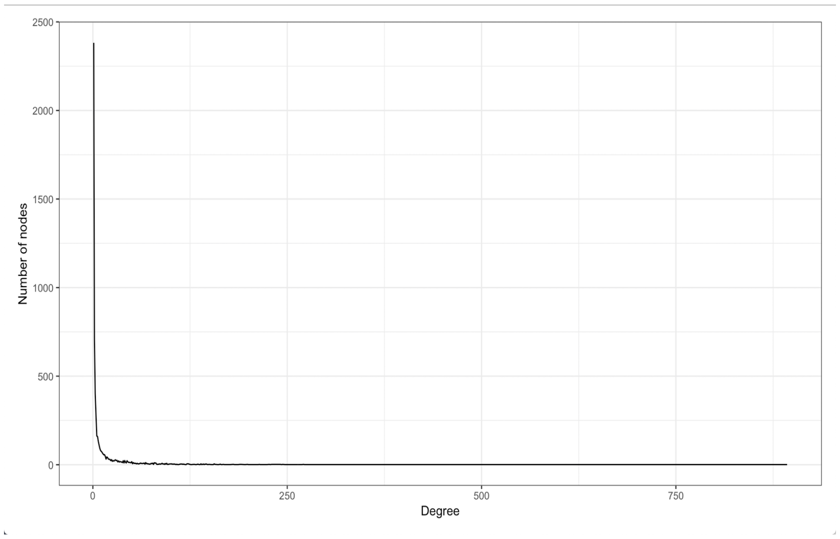
```
wiki_edges <- read.table('wiki-Vote.txt', sep="\t", header=F, col.names=c('src','dst'))
wiki_graph <- graph.data.frame(wiki_edges, directed=T) # graph representation

wiki_degree_dist <- wiki_edges %>%
  group_by(src) %>% # group by the source node
  summarize(degree=n()) %>% # get the degree of each node
  group_by(degree) %>% # group by the degree
  summarize(num_nodes=n()) # compute how many nodes had a given degree
```

We can visualize the degree of the node, one example as follows:

```
ggplot(wiki_degree_dist, aes(x = degree, y = num_nodes)) +
  geom_line() +
  xlab('Degree') +
  ylab('Number of nodes')
```

And the resulting plot:



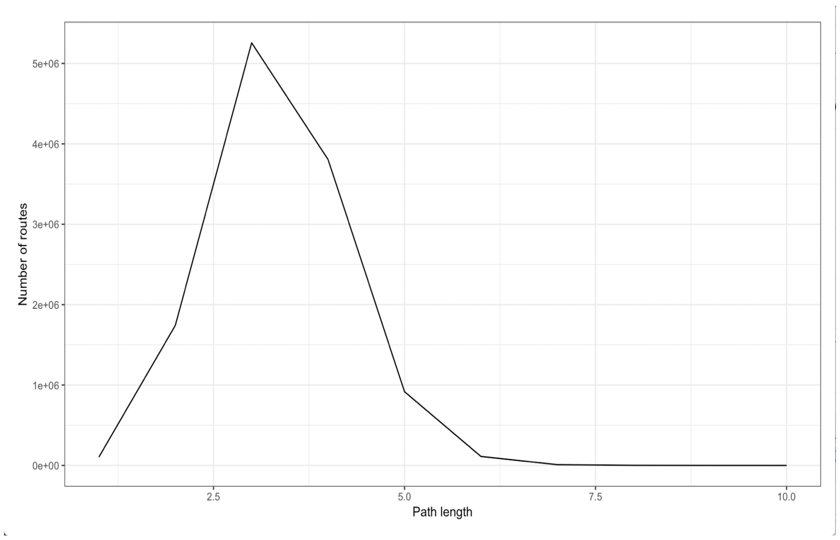
What do you think we can learn based on what we know about Wiki degrees?

igraph also allows one to compute and plot a histogram of the average path length between nodes:

```
count <- path.length.hist(wiki_graph)$res
plot_data <- data.frame(path_length = 1:length(count), count)
ggplot(plot_data, aes(x = path_length, y = count)) +
  geom_line() +
  xlab('Path length') +
  ylab('Number of routes')
```

and the corresponding plot:

Again, what do you think we can learn based on what we know about Wiki degree? And how would you go about computing average path length?



3 Single source shortest path and applications

One very common application of graphs is the single source shortest path problem. Google Maps provides just one of countless many of such applications.

In the realm of undirected graphs, the algorithm to compute shortest path uses a well-known graph algorithm called breadth-first search, a.k.a. BFS. An implementation is as follows:

```
• init all nodes to be infinitely far away, except for source at distance = 0
• init current boundary to be source node
• while boundary not empty
    – create empty list for new boundary
    – for all nodes in current boundary
        ∗ loop over undiscovered neighbors
        ∗ set neighbors distance = current distance + 1
        ∗ add neighbor to next boundary
```

See if you can implement this algorithm in R.

The algorithm finds many more applications outside of maps and navigation. For example, let's say that you wanted to find out how many distinct groups of friends there were on a social network. This is an application of connected components, which may efficiently rely on BFS to generate the correct result (one savvy classmate suggested another efficient solution making use of the union find data structure, but that's beyond the scope of this course).

4 Mutual friends and counting triangles

The final application of graphs we'll look at is counting mutual friends. In a social network, for example, we are often interested in how many friends of friends are also friends.

We can use the following pseudo-code to implement the algorithm:

```
• For all nodes in the graph
    – Compute its neighbors
    – For each unique pair in the neighbors list (pairwise generation)
```

    * Increment the number of mutual friends between the nodes in the pair by 1 since they have the the original node in common (would require a matrix to compute)

Using a variant of this algorithm, one can also compute the number of 3-friend groups aka triangles (can you write the pseudocode for this?).

On a side note, around this time, the question came up regarding whether an adjacency list was really constant, or O(1), access time considering some nodes may be adjacent to multiple nodes and iterating through the list would take a long time. In the naive implementation where only the nodes are hashed to a list of adjacent nodes, then there would be a slight load factor added onto the running time, though assuming very few nodes would have a large degree, the load factor would be negligible. If you absolutely needed a smarter adjacency list representation, there are ways to do it. Those, however, are beyond the scope of this lecture.

5 The fallacy of causation

Time and time again, we always say: correlation (or prediction) does not imply causation, yet this happens far too often the scientific world because it is so easy to make this mistake.

Prediction: make a forecast but assuming the universe stays as it is Causation: anticipating what will happen when the universe changes

Before moving on, it is important that we ask the right questions. It is far too tempting to ask reverse causal inference questions such as:

- What makes an email spam?
- What caused my kid to get sick?
- Why did the stock market drop?

The problem is that these questions are very difficult to answer. It is better to ask forward causal inference questions such as:

- How does education impact future earnings?
- What is the effect of advertising on sales?
- How does hospitalization affect health?

These are still hard questions, but they are far less contentious. With enough data, conclusions can be more easily drawn. This does not mean, however, that we can let our guard down.

6 A simple example

Consider the last question: How does hospitalization affect health?

Suppose someone claimed that if a patient visited the hospital today, they would be healthy tomorrow. What's wrong with this claim?

When someone makes a blanket causation statement like this, they ignore factors called confounds which might literally be be changing together with the implied cause. In this case, said person ignores the fact that a person's health today likely influences their health tomorrow. Therefore, since we're observing both the patient's hospital visit status AND their health today, we cannot make any causation claim.

To put it mathematically, the causal effect we're trying to measure is:

$$\Delta_{obs} = s_{hospital} - s_{home}$$

where

$$s_{hospital}$$

is the number of patients that were sick and went to the hospital and

$$s_{home}$$

is the number of patients that were sick and stayed home. In an ideal world, this would be the only thing we would be measuring. Unfortunately, we must take into account the confounding factor. In this case, assuming that all healthy people stay home and sick people go to the hospital in our dataset, we actually have that:

$$\Delta_{obs} = (s_{hospital} - s_{home}) - (s_{home} - h_{home})$$

where we introduce the selection bias term (the latter) and a new variable,

$$h_{home}$$

, which indicates the number of healthy people that stayed home. This term is in practice quite difficult to calculate and will often be negative, meaning that without the selection bias term, we could be massively underestimating.

In summary, the observed difference equation:

$$\Delta_{obs} = obs - bias$$

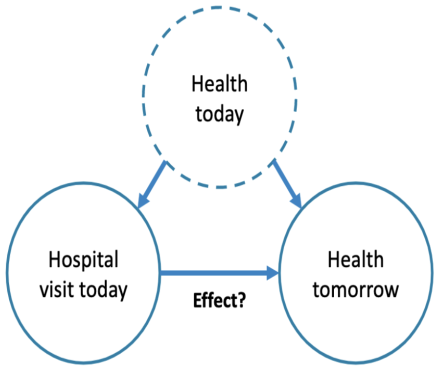
The nightmarish situation in terms of selection bias is Simpson's paradox, in which selection bias can be so large that the observational and causal estimates give opposite effect. Consider, for instance, this:

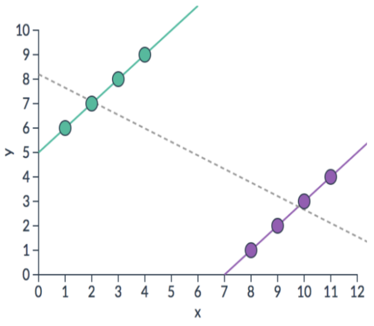
The two colored trend lines are what we should be fitting since they are isolated. However, if we didn't take into account selection bias, we would attempt to fit a regression line onto both sets of data, evidence by the dotted line. This naive regression, in effect, would tell us that going to hospitals makes you less healthy (see what I've been telling you y'all, the government is secretly making y'all sick, I don't need no hospitals!).

7 Dealing with confounds

The way we do this is to change one and only one thing, then compare outcomes. We can only estimate what would have happened if we did something else.

We can use random assignment to create two groups that differ only in what they receive.


Health today
Hospital visit today
Effect?
Health tomorrow


y
x
0
1
2
3
4
5
6
7
8
9
10
11
12

Notes from du2147

1 Introduction

In this lecture, we covered the following key topics:

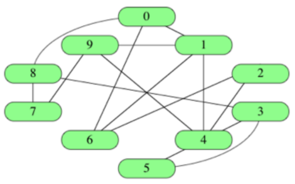
- Network Representations (Data Structures)
- Network Topologies and Watts-Strogatz Networks
- Describing Networks
- Causality and (Natural) Experiments

2 Network Representations (Data Structures)

We discussed three different data structures that can be used to represent networks: Edge list, adjacency matrix and adjacency list.

2.1 Edge List

One way to represent a network is a list of $|E|$ edges, called an edge list. Each edge is represented as a tuple of the two vertices the edge is incident on. Even though edge lists are simple to store, performing computations on them can be difficult. The space complexity of an edge list is $\Theta(E)$. If we wanted to check whether an edge exists between two vertices, we would need to search the entire list, which would take $O(E)$ time.



[[0,1], [0,6], [0,8], [1,4], [1,6],
[1,9], [2,4], [2,6], [3,4], [3,5],
[3,8], [4,5], [4,9], [7,8], [7,9]]

Simple for storage, but difficult
to compute with

Figure 1: A network with 10 vertices (left) and the edge list representation of its edges (right).

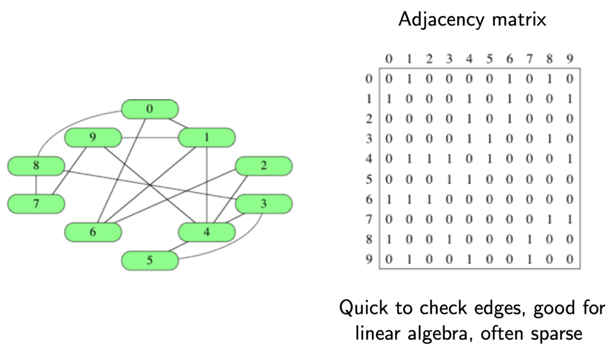
2.2 Adjacency Matrix

The second approach involves building a $|V|x|V|$ matrix of 1s and 0s for a network with $|V|$ vertices where row i, column j is 1 if there exists an edge between vertices i and j. The space complexity of an adjacency matrix is $\Theta(V^2)$. Thus, if our network was relatively sparse, the matrix would mostly consist of 0s and take up as much space as a dense network to represent. Yet, checking whether an edge exists between vertices i and j can be performed in constant time $O(1)$.

2.3 Adjacency List

The last data structure we covered, adjacency list is a combination of edge list and adjacency matrix. An adjacency list of a network with $|V|$ vertices has length $|V|$ where each element i is a list of vertices adjacent to vertex i. The space complexity of an adjacency matrix is $\Theta(E)$ where $|E|$ is the number of edges in the network. Checking whether a vertex i has an edge to vertex j takes $O(d)$ time where d is the degree of the vertex i.

Note: See Networks.Rmd for the representation of some toy and real networks as various data structures using igraph.



	0	1	2	3	4	5	6	7	8	9
0	0	1	0	0	0	0	1	0	1	0
1	1	0	0	0	1	0	1	0	0	1
2	0	0	0	0	1	0	1	0	0	0
3	0	0	0	0	1	1	0	0	1	0
4	0	1	1	1	0	1	0	0	0	1
5	0	0	0	1	1	0	0	0	0	0
6	1	1	1	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	1	1
8	1	0	0	1	0	0	0	1	0	0
9	0	1	0	0	1	0	0	1	0	0

Figure 2: A network with 10 vertices (left) and the adjacency matrix representation of its edges (right).

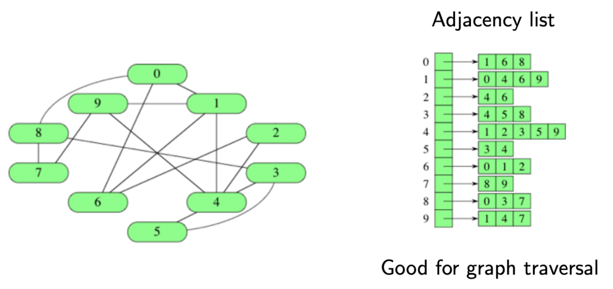


Figure 3: A network with 10 vertices (left) and the adjacency list representation of its edges (right).

3 Network Topologies and Watts-Strogatz Networks

While going over the capabilities of igraph, an R library (see Networks.Rmd), we learned about several network topologies (Figures 4, 5 and 6) and generated some small world networks using a random graph generation model called the Watts-Strogatz model (Figures 7, 8, and 9).

4 Describing Networks

Higher the dimension and complexity of a network, harder it gets to visualize. We discussed four different statistics for describing and analyzing a network, and algorithms one can use to calculate these statistics:

- Degree: How many connections does a node have?
  - Degree distributions

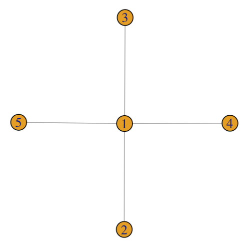


Figure 4: Star Network

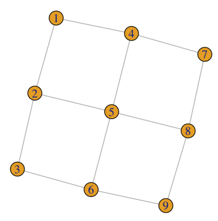


Figure 5: Lattice Network

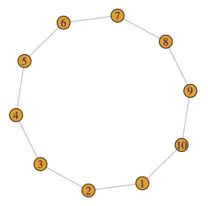


Figure 6: Ring Network

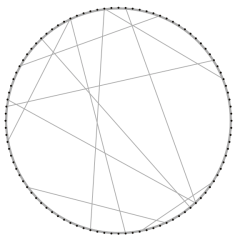


Figure 7: Mostly a ring

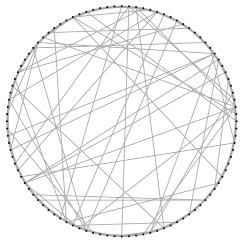


Figure 8: Some rewiring

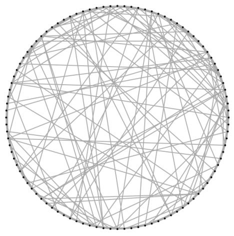
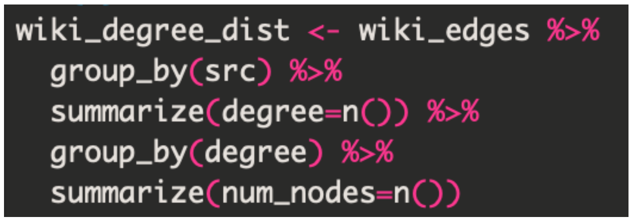


Figure 9: Lots of rewiring

- Path Length: What's the shortest path between two nodes?
  - Breadth first search
- Clustering: How many friends of friends are also friends?
  - Triangle counting
- Components: How many disconnected parts does the graph have?
  - Connected components

4.1 Degree Distributions

The degree of a node in a network is the number of connections it has to other nodes. The degree distribution of a network is the probability distribution of these degrees across the entire network. Working with some real life networks, we plotted some degree distributions and learned ways to make these plots more readable (See Networks.Rmd). Below are the code for and the visualization of the out-degree distribution (edges coming out of nodes) of the Wikipedia voting network and its log visualization:



```
wiki_degree_dist <- wiki_edges %>%
  group_by(src) %>%
  summarize(degree=n()) %>%
  group_by(degree) %>%
  summarize(num_nodes=n())
```

Figure 10: R code for out-degree distribution

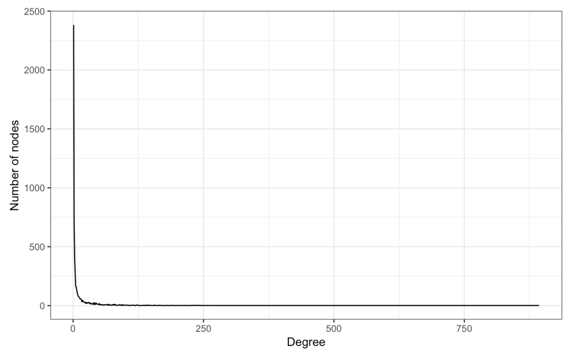


Figure 11: Visualization of Out-Degree Distribution

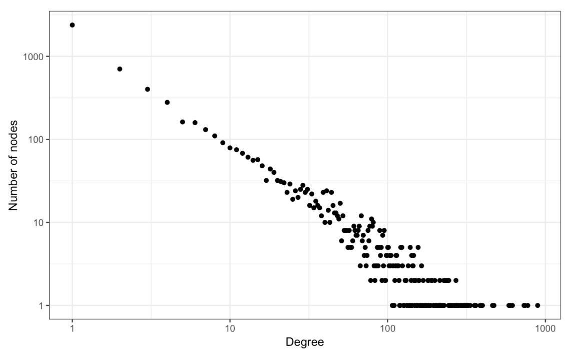


Figure 12: Visualization of Out-Degree Log Distribution

4.2 Path Length

Path length is the shortest distance between two nodes of a network. The path length distribution of a network is the distribution of the shortest paths between all node pairs in the network (See Network.Rmd for igraph way of calculating path lengths of a network and its visualization). The mean and median path lengths of a network are useful ways to analyze and understand a network. To calculate single-source shortest paths to all other nodes of a network, we can use an algorithm called Breadth First Search:

Breadth First Search

```
Initialize all node distances to infinitely far
Set source node distance to zero
Initialize current boundary as source node

While current boundary is not empty: {
Create empty list for new boundary
For each node in current boundary: {
loop over undiscovered neighbors and set neighbor distance as current distance+1 and add neighbor to the next boundary
}
Set current boundary = next boundary
}
```

Note: See counting_on_networks.Rmd for BFS R code.

4.3 Connected Components

In a network, a connected component is a subgraph in which any two nodes are connected to each other by paths, and which is connected to no additional nodes in the overall graph. The algorithm to determine the connected components in a network uses BFS and is therefore fairly straightforward.

Connected Components Algorithm

(1) Choose any node and perform BFS, labeling all the other nodes accessible from the chosen source node as component x.
(2) Update label, choose any unlabeled node in the network and repeat (1) until all nodes are labeled.

4.4 Clustering

Clustering is the task of grouping a set of objects in such a way that objects in the same group are more similar to each other than to those in other groups. One way to think about it is in terms of mutual friends (connected

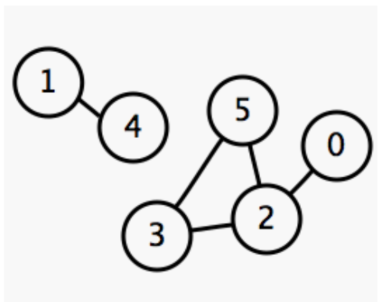


Figure 13: A network with two connected components.

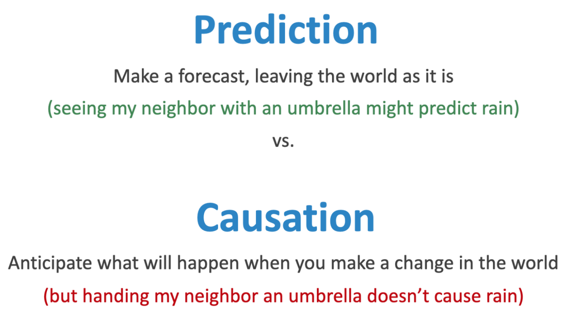
nodes). How many friends of friends are also friends? For this end, we covered an algorithm referred to as triangle counting (See counting_on_networks.Rmd).

Counting Triangles

```
Initialize counter for the number of triangles at each node
Loop over each node: {
Get the list of friends for node
Add a count of 1 for each pair of node's friends that are connected
}
```

5 Causality and (Natural) Experiments

We started the new topic with a discussion on the difference between 'Prediction' and 'Causation'. We concluded that while 'Prediction' focuses on observing an environment and forecasting what is going to happen based on observation, 'Causation' focuses on changing the environment and anticipating the effects.



Prediction

Make a forecast, leaving the world as it is

(seeing my neighbor with an umbrella might predict rain)

vs.

Causation

Anticipate what will happen when you make a change in the world

(but handing my neighbor an umbrella doesn't cause rain)

Figure 14: Prediction versus Causation

Then we looked at the difference between 'Reverse Causal Inference' (causes of effects) and 'Forward Causal Inference' (effects of causes). While 'Reverse Causal Inference' is questioning 'What caused Y?', 'Forward Causal Inference' is asking 'What are the effects of X?'. In general, 'Forward Causal Inference' is easier than 'Reverse Causal Inference'.

Examples of 'Reverse Causal Inference':

- What makes an email spam?
- What caused my kid to get sick?
- Why did the stock market drop?

Examples of 'Forward Causal Inference':

- How does education impact future earnings?
- What is the effect of advertising on sales?
- How does hospitalization effect health?

We then took a closer look at the last example, 'How does hospitalization effect health?':

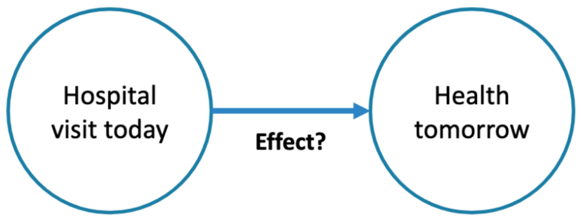


Figure 15: Arrow means 'X causes Y'

As we discussed in class, this representation of the cause-effect relationship between going to the hospital today and one's health tomorrow can lead to a problematic conclusion: Going to the hospital makes you sick. In such a case, regardless of how many observations one might have, the conclusions one draws will be biased. Thus, this relationship cannot be explained independently of some other common factor.

Confounds

The effect and cause might be confounded by a common cause and be changing together as a result. In fact, there could be many confounds. Going back to our hospital/health example, it is easy to recognize that visiting the hospital today and one's health tomorrow both depend on another cause, one's health today. If one only observes the cause and effect changing together, one cannot estimate the effect of hospitalization alone on health tomorrow.

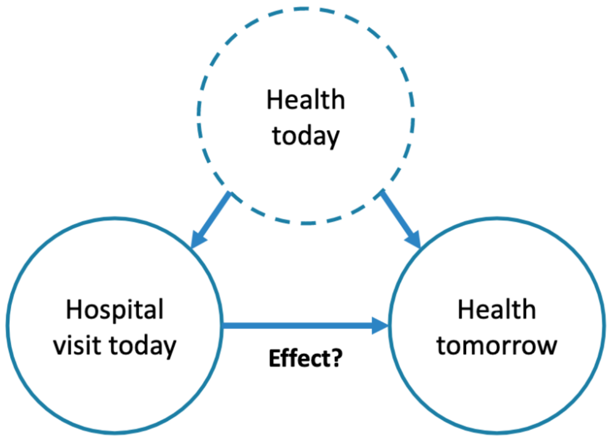


Figure 16: Dashed circle means 'unobserved'.

Observational Estimates

Let us say all sick people in our dataset went to the hospital today and healthy people stayed at home. The observed difference in health tomorrow is:

$$\begin{aligned}\Delta_{obs} = {} & [(\text{Sick and went to hospital}) - (\text{Sick if stayed home})] + \\ & [(\text{Sick if stayed home}) - (\text{Healthy and stayed home})]\end{aligned}$$

Figure 17: Here we are trying to compare something that did happen with something that could have happened. Unfortunately, we cannot clone people and make their clones stay at home.

Selection Bias

Selection bias is the baseline difference between those who opted in to the treatment and those who did not.

$$\begin{aligned}\Delta_{obs} = {} & \overbrace{[(\text{Sick and went to hospital}) - (\text{Sick if stayed home})]}^{\text{Causal effect}} + \\ & \underbrace{[(\text{Sick if stayed home}) - (\text{Healthy and stayed home})]}_{\text{Selection bias}}\end{aligned}$$

Figure 18: Selection bias is likely negative here, making the observed difference an underestimate of the causal effect.

Simpson's Paradox

Selection bias can be so large that observational and causal estimates give opposite effects (e.g. going to the hospital makes you sick). This is called the Simpson's Paradox.

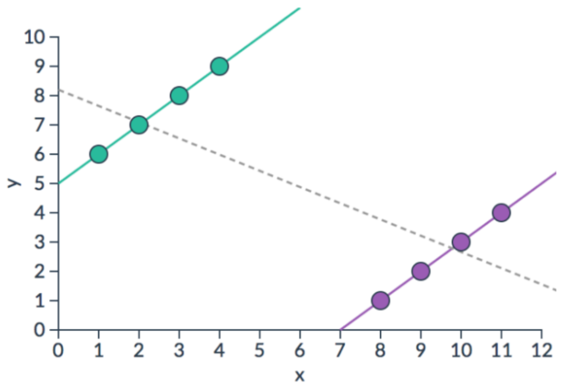


Figure 19: Simpson's Paradox

Controlled Experiments

Counterfactuals

To isolate the causal effect, we have to change one and only one thing (hospital visits) and compare the outcomes. Unfortunately, we never get to observe what would have happened if we did something else so we have to estimate it.

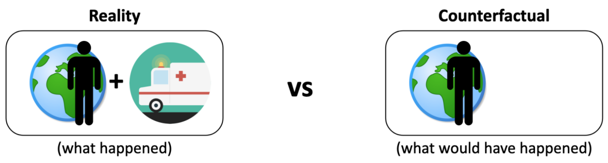


Figure 20: Reality versus Counterfactual

Random Assignment

We cannot clone people but by randomly assigning their actions with a coin flip, we can get results almost as good as cloning. In the hospital example, we can use randomization to create two groups that differ only in which treatment they receive. This allows us to restore symmetry.

[width=0.5]figures/random.png

Figure 21: Randomly assign both sick and healthy people to which treatment they will receive.

[width=0.5]figures/worlds.png

Figure 22: The resulting worlds closely resemble the reality versus counterfactual separation we desire.