tutorial

April 27, 2020

1 Causal Inference with Interference

For this tutorial we'll be looking at a network of US states connected by geographic adjacency. The dependent variable is the number of people infected by a virus after a 30 day period. The treatment is a stay at home order issued by the governor. This provides a fairly intuitive illustration of how treatment and interference works. This tutorial uses simulated data so that treatment can be randomized and so I can make clean assumptions about how spillover works for illustrative purposes. We're going to pretend we don't know the data generating function though.

Here are the basic rules:

1. Within each state the population of infected doubles every 3 days when no stay at home order is issued and there is no interference.
2. In addition to this natural growth rate within the state, each state experiences spillover effects from neighboring states, further increasing the number of infected
3. When the governor issues a stay at hom order, the natural rate of growth slows, and the spillover effects to other states stops.

The number of infected after 30 days is a function of the number of infected at the start, whether or not a stay at home order was issued, and the proportion of adjacent states that have also issued stay at home orders. By way of examples, if all the states adjacent to a given state have issued stay at home orders, that state experiences no spillover effects. It can still spill over to other states if it has not issued a stay at home order itself.

[1]:
```
# libraries
suppressPackageStartupMessages({
    library(igraph)
    library(dplyr)
    library(usmap)
    library(ggplot2)
})
```

1.0.1 Data generation, you can mostly ignore this part

[2]:
```r
set.seed(123)

edges <- read.csv('state_edges.csv', stringsAsFactors = FALSE)
states <- state.abb[!state.abb %in% list('AK','HI')]
z <- rbinom(length(states), 1, .50)

df <- data.frame(state = states, treatment = z)

g <- graph.data.frame(edges, directed = FALSE, vertices = df)

# get the degree for each node
degree <- degree(g)

# get the number of treated and control neighbors for each node
cn <- sapply(V(g), function(x) {allneighbors = neighbors(g, x) ;␣
 ↪length(allneighbors[allneighbors$treatment == 0])})
tn <- sapply(V(g), function(x) {allneighbors = neighbors(g, x) ;␣
 ↪length(allneighbors[allneighbors$treatment == 1])})

# proportion of neighbors treated
ptn <- tn/degree

# function to generate the observed values. We'll pretend we don't know this␣
 ↪function.

# create the starting value
init <- floor(runif(48, min = 1, max = 5))

# add a random error
u <- abs(round(rnorm(48, mean = 20, sd = 10)))
# generate the data
y <- init + init * 1000 * abs(z-1) + 200 * cn + u

df <- data.frame(init_cases = init,
                 cases_1month = round(y),
                 treatment = z,
                 neighbors = degree,
                 treat_neighbors = tn,
                 control_neighbors = cn,
                 perc_treat_neighbor = ptn)
```

2 Data

Here's a look at our data set. The variables we will work with are:

cases_1month: The total number infected after a 30 day period

treatment: 1 if a stay at home policy has been implemented, 0 otherwise

perc_treat_neighbor: Number of adjacent neighboring states that have issued stay at home orders

[4]:
```
head(df)
```

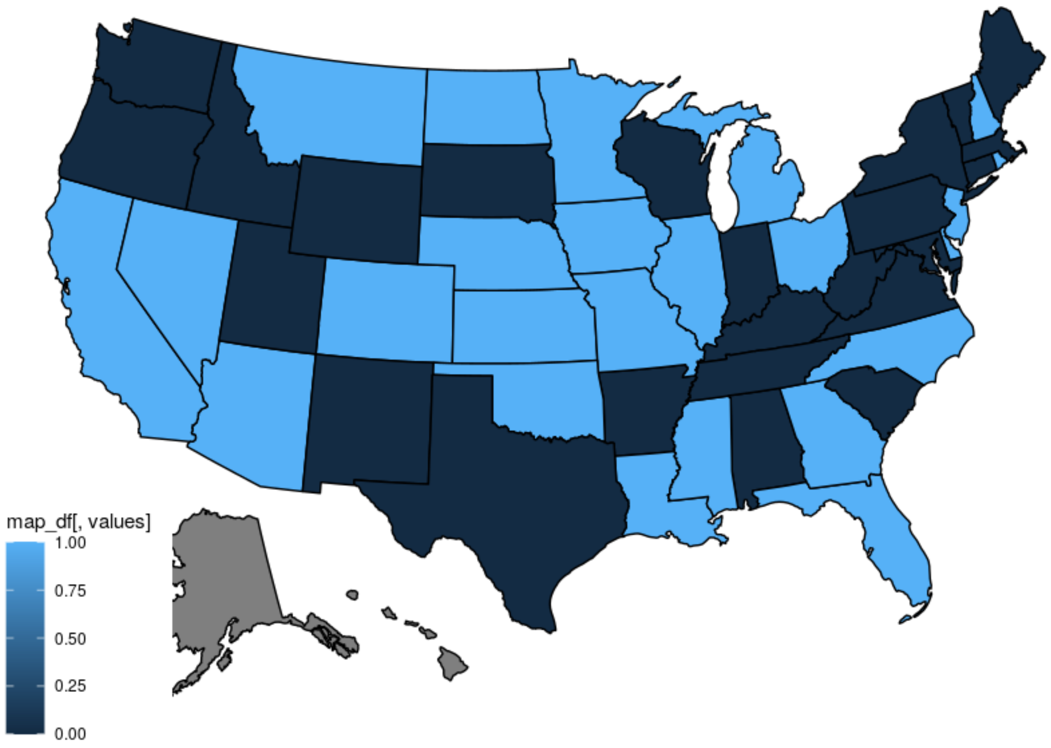
A data.frame: 6 × 7

	init_cases <dbl>	cases_1month <dbl>	treatment <int>	neighbors <dbl>	treat_neighbors <int>	control_neigh <int>
AL	2	2230	0	4	3	1
AZ	4	423	1	5	3	2
AR	1	1424	0	6	4	2
CA	2	222	1	3	2	1
CO	4	624	1	7	4	3
CT	1	1435	0	3	1	2

2.0.1 And here's a simple map of our simulated network. Light blue are treated states

[5]:
```
map <- data.frame(state = rownames(df), treatment = df$treatment)
plot_usmap(regions = 'states', data = map, values = 'treatment')
```

```
Warning message:
"Use of `map_df$x` is discouraged. Use `x` instead."
Warning message:
"Use of `map_df$y` is discouraged. Use `y` instead."
Warning message:
"Use of `map_df$group` is discouraged. Use `group` instead."
```



3 Process

The test consits of the following steps:

1. Write down the interference model and choose an appropriate test statistic.
2. Remove the hypothesized effects of treatment and interference from the observed outcomes and calculate the test statistic. This is called the uniformity trial and estimates the potential outcome if all units recieved the control.
3. Randomly assign treatments, and then remove the hypothesized treatment effects from the observed outcomes.

4. Calculate the test statistic on the randomized and adjusted outcomes. Repeat these two steps many times.
5. Calculate the p value associated with the tested parameter as the proportion ofrandomized test statistics larger than the observed test statistic

I will walk through each step.

4 Step 1: The Interference Model

4.0.1 Direct Effects

To define the model lets work with something simple and say the number of infected is a linear function of whether or not a state issued a stay at home order:

$$y_i = \beta_1 \times z_i$$

Where z is where or not a stay at home issue was ordered and β_1 is the effect of the treatment. In addition we think there was some spillover effect from the treatment. We assume people in stay at home states stop traveling and thus reduce the spread of the virus. Individuals in states without the treatment continue to travel and spread the virus at an increased rate. We'll assume the rate of spillover is a function of how many adjacent states also issued stay at home orders.

$$y_i = \beta_1 \times z_i + \beta_2 \times x$$

Where x is the number of adjacent states that were treated and β_2 is the spillover effect.

5 Step 2: The Uniformity Trial

The uniformity trial is the outcome that would have occured if all observations would have been assigned to the control group. Since we can't actually observe this, we instead remove the hypothesized treatment effects from the observed values. The first step is to hypothesize values for the parameters we are measuring. Ideally you want to search a grid of values, but I'll justpick one value for each parameter. As a starting point, let's run a simple linear model to estimate the coefficients.

```
[86]: summary(lm(cases_1month ~ treatment + treat_neighbors, data = df))
```

```
Call:
lm(formula = cases_1month ~ treatment + treat_neighbors, data = df)

Residuals:
     Min       1Q   Median       3Q      Max
-1571.42  -477.77   -48.72   160.71  1850.75

Coefficients:
```

```
                 Estimate Std. Error t value Pr(>|t|)
(Intercept)      3010.806    286.883  10.495 1.12e-13 ***
treatment       -2513.113    255.321  -9.843 8.52e-13 ***
treat_neighbors    -8.388     97.684  -0.086    0.932
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 878.9 on 45 degrees of freedom
Multiple R-squared:  0.6851,Adjusted R-squared:  0.6711
F-statistic: 48.96 on 2 and 45 DF,  p-value: 5.1e-12
```

For illustrative purposes lets assume the value of the treatment parameter is known and we will only vary the interference parameter. In an actual application you'll want to grid search across both of these parameters. Let's define the parameters now based on the results of the regression:

[85]:
```
B1 <- -2513
B2 <- -8
```

Now we need to remove the estimated treatment effect from the network:

[79]:
```
df$y0 <- round(df$cases_1month - B1*df$treatment - df$treat_neighbors*B2)
```

Now we calculate the test statistic for the uniformity trial. In this case I'll use the residual sum of squares:

[80]:
```
y0_rss <- lm(y0 ~ treatment + treat_neighbors, data = df) %>%
            residuals %>%
            `^`(2) %>%
            sum

y0_rss
```

34763569.4751242

6 Step 3-4: Randomly Assign Treatments, Calculate the Test Statistic, Repeat.

First we randomly re-assign the treatment labels by permuting the labels. We want a unique permutation for each time we repeat the trial. Ideally you'll want to repeat the trail 1,000+ times for each unique combination of parameter values. I'll just do 100 here.

[81]:
```
set.seed(123)
perms <- data.frame(sample(z, 48)) # start with a single permuted vector of the␣
 ↪original treatments
nperms <- 1 # initiate a counter for the number of unique permutations
```

```
while(nperms < 100){
    perms <- cbind(perms, sample(z, 48)) # permute the treatment
    perms[!(duplicated(perms)), ] # if the permutation is not unique, drop it
    nperms <- length(perms) # increase the counter
}
```

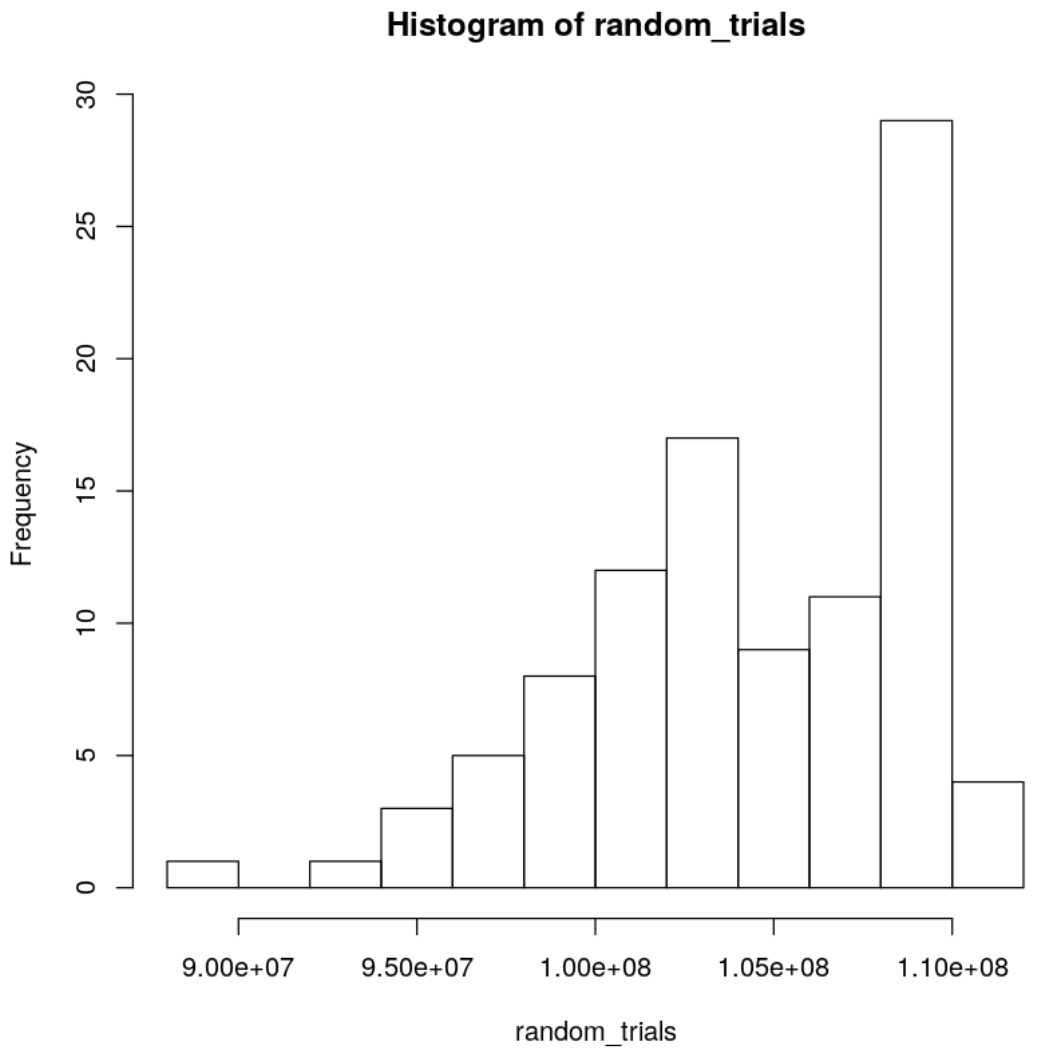
Now we repeat the uniformity trial for each randomized treatment vector, calculate the test statistics, and save the result.

[82]:
```
random_trials <- c()
for(i in 1:length(perms)){

    # now generate a randomized treatment vector and assign it to the random␣
 ↪samples
    fdf <- df
    fdf$fake_treat <- perms[,i]
    fdf <- tibble::rownames_to_column(fdf, "state")

    # create the new graph object to find the number of adjacent treated nodes
    fakeg <- graph.data.frame(edges, directed = FALSE, vertices = fdf)

    # for each node, find the proportion of neighbors that were not treated
    fdf$ftn <- sapply(V(fakeg), function(x) {allneighbors = neighbors(fakeg, x)␣
 ↪; length(allneighbors[allneighbors$fake_treat == 1])})
    fdf$fcn <- sapply(V(fakeg), function(x) {allneighbors = neighbors(fakeg, x)␣
 ↪; length(allneighbors[allneighbors$fake_treat == 0])})

    # remove the treatment
    fdf$fy0 <- round(fdf$cases_1month - fdf$fake_treat*B1 - fdf$ftn*B2)

    # And now we calculate our test statistic for the randomized uniformity␣
 ↪trial and return it
    fy0_rss <- lm(fy0 ~ fake_treat + ftn, data = fdf) %>%
                        residuals %>%
                        `^`(2) %>%
                        sum

    random_trials <- append(random_trials, fy0_rss)
}
```

7 Step 5: Calculate your p value

Let's take a look at the distribution of our randomized trials:

[83]:
```
hist(random_trials)
```



Now we calculate the p value, the proportion of randomized test statistics that are greater than your observed test statistic. A higher p value is stronger evidence for a given statistic.

[84]:
```
num <- sum(y0_rss < random_trials)
p <- num/100
p
```

1

To calculate the confidence interval, we repeat the above steps for a wide range of parameter values. Combinations with a p value > .05 fall within your 95% confidence interval. Your parameter estimate is the value pair with the highest p-value. In this instance, a p-value of 1 on my first parameter combination indicates I would need to run many more iterations to get an accurate estimate.