Analytics Engineer Challange for CLUE

Vincenzo Grasso - vincenzo.grass@gmail.com

31/01/2022

A - Message Conversion Rates

This is the typical example of Simpon's Paradox. It's a phenomenon in in which a trend appears in groups of data but **disappears or reverses when the groups are combined**. This means that the paradox **cannot be solved** unless on adds an *extra* dimension, i.e. the country of each user or any other confounding variable. Alternatively, on can work on *balancing* the numbers. The problem lies in the fact that for iOS, far more Bs than As were sent, and for Android the reverse. If you can collect more data so the numbers balance out, the paradox should go away. I would not "downsample", which is essentially throwing away data. I would first start to have a look at country information, not shown in this contingency table.

There could be **another way** to look at this problem: if the data has been properly randomized is unlikely to look at those unbalanced proportions.

- IOS a/b sample ratio: 174000 / 540000 = 0.32
- Android a/b sample ratio: 526000 / 160000 = 3.29

If this is **by design**, regression adjust/stratified average treatment effect may be helpful for estimate Both treatment effect.

Reference can be found here.

- I create a `weight` vector

```r
library(tidyverse)

y <- rep(0:1, c(4, 4))
trt <- rep(c("A", "B"), 4)
platform <- rep(c("IOS", "IOS", "Android", "Android"), 2)
all_cnt <- c(174000, 540000, 526000, 160000)
subscriptions <- c(16200, 46800, 38400, 11000)
non_subscriptions <- all_cnt - subscriptions

weight <- c(non_subscriptions, subscriptions)

df <- tibble(
  y,
  trt = factor(trt, levels = c("B", "A")),
  platform,
  weight
)
print(df)
```

```
## # A tibble: 8 x 4
##       y trt   platform weight
##   <int> <fct> <chr>     <dbl>
## 1      0 A     IOS       157800
## 2      0 B     IOS       493200
## 3      0 A     Android   487600
## 4      0 B     Android   149000
## 5      1 A     IOS        16200
## 6      1 B     IOS        46800
## 7      1 A     Android    38400
## 8      1 B     Android    11000
```

- I try to run a linear model (`lm()`), accounting for the weights.

```
df_long <- df %>%
  rowwise() %>%
  mutate(ids = list(seq(1, weight))) %>%
  unnest(ids)
# evaluate model
summary(lm(y ~ trt + platform, df_long))
```

```
##
## Call:
## lm(formula = y ~ trt + platform, data = df_long)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.09231 -0.08692 -0.07327 -0.07327  0.93212
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 0.0678837  0.0005274 128.726   <2e-16 ***
## trtA        0.0053836  0.0005386   9.995   <2e-16 ***
## platformIOS 0.0190396  0.0005387  35.342   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.2716 on 1399997 degrees of freedom
## Multiple R-squared:  0.0009621,  Adjusted R-squared:  0.0009606
## F-statistic: 674.1 on 2 and 1399997 DF,  p-value: < 2.2e-16
```

for Both: A = 0.54%. **Message A seems to be better on iOS and on Android, and on Both**, using this approach.

B - Cumulative Subscription Rates in SQL

B.1 I will use Postgresql-flavor dialect; and I will try to explain my reasoning.

- **STEP 1 - Temporary Table Creation**

Create a temporary table that joins `users` and `subscriptions`; if I user performed several purchases, I will take the `min(event_time)`, aka the first `subscription_product_id` purchased by the user, in order to attribute him/her/them to the relevant cohort.

I can easily calculate:

- The `cohort_week` column
- The `days_to_conv` as a difference between `first_seen - event_time`
- The `range` based on `days_to_conv`: 1D - if `days_to_conv` is within 1, 30 90 or 360 days.

The resulting table should look something like this:

ID	first_seen	event_time	days_to_conv	cohort_week	range
1448	2018-08-20	2018-08-21	1	2018-08-20	1D
1207	2018-08-20	2018-08-21	1	2018-08-20	1D
1528	2018-08-25	NULL	NULL	2018-08-20	1D
1985	2018-08-22	2018-08-23	1	2018-08-20	1D
1971	2018-08-23	2018-08-24	1	2018-08-20	1D
1661	2018-08-25	NULL	NULL	2018-08-20	1D
1090	2018-08-23	NULL	NULL	2018-08-20	1D
1525	2018-08-25	2018-08-27	2	2018-08-20	30D
1973	2018-08-25	2018-09-09	15	2018-08-20	30D
1378	2018-08-20	NULL	NULL	2018-08-20	30D
1194	2018-08-22	2018-09-07	16	2018-08-20	30D
1651	2018-08-25	NULL	NULL	2018-08-20	30D
1310	2018-08-23	2018-11-17	86	2018-08-20	90D
1375	2018-08-20	2018-11-04	76	2018-08-20	90D
1631	2018-08-25	2018-12-05	102	2018-08-20	360D
1029	2018-08-22	NULL	NULL	2018-08-20	360D
1030	2018-08-25	NULL	NULL	2018-08-20	360D
1253	2018-08-20	NULL	NULL	2018-08-20	360D

- **STEP 2 - Cumulative Calculations**

The range column type is problematic because it is difficult to sort. You need this in the definition of a window function, and you have to use a rather clunky expression for it… I use the `LEFT()` function to take away the "D" and cast (`::int`) in order to sort. It would have been easier to create it as an integer directly.

- The main idea is to use a convenient **window function** via `partition by cohort_week order by left(range, -1)::int`

```sql
select
    cohort_week,
    total as tot_size,
    range,
    converted,
    round(converted::numeric/ total* 100, 2) as ratio,
    round(sum(converted::numeric/ total* 100) over w, 2) as cum_ratio
from (
    select
        cohort_week,
        count(*),
        (
            select count(*)
            from my_table s
            where s.cohort_week = t.cohort_week
        ) as total,
        range,
        count(event_time) as converted
    from my_table t
    group by cohort_week, range, total
    ) s
window w as (partition by cohort_week order by left(range, -1)::int)
order by cohort_week, left(range, -1)::int
```

I created a table like follows and tested via DBFiddle in Postgres14 (link).

```
drop table if exists my_table;
create table my_table(id serial primary key,
                      first_seen date,
                      event_time date,
                      days_to_conv int,
                      cohort_week date,
                      range text);
insert into my_table values
(1448, '2018-08-20', '2018-08-21', 1, '2018-08-20', '1D'),
(1207, '2018-08-20', '2018-08-21', 1, '2018-08-20', '1D'),
(1528, '2018-08-25', null, null, '2018-08-20', '1D'),
(1985, '2018-08-22', '2018-08-23', 1, '2018-08-20', '1D'),
(1971, '2018-08-23', '2018-08-24', 1, '2018-08-20', '1D'),
(1661, '2018-08-25', null, null, '2018-08-20', '1D'),
(1090, '2018-08-23', null, null, '2018-08-20', '1D'),
(1525, '2018-08-25', '2018-08-27', 2, '2018-08-20', '30D'),
(1973, '2018-08-25', '2018-09-09', 15, '2018-08-20', '30D'),
(1378, '2018-08-20', null, null, '2018-08-20', '30D'),
(1194, '2018-08-22', '2018-09-07', 16, '2018-08-20', '30D'),
(1651, '2018-08-25', null, null, '2018-08-20', '30D'),
(1310, '2018-08-23', '2018-11-17', 86, '2018-08-20', '90D'),
(1375, '2018-08-20', '2018-11-04', 76, '2018-08-20', '90D'),
(1631, '2018-08-25', '2018-12-05', 102, '2018-08-20', '360D'),
(1029, '2018-08-22', null, null, '2018-08-20', '360D'),
(1030, '2018-08-25', null, null, '2018-08-20', '360D'),
(1253, '2018-08-20', null, null, '2018-08-20', '360D');
```

Result:

cohort_week	tot_size	range	converted	ratio	cum_ratio
2018-08-20	18	1D	4	22.22	22.22
2018-08-20	18	30D	3	16.67	38.89
2018-08-20	18	90D	2	11.11	50.00
2018-08-20	18	360D	1	5.56	55.56

B.2 - Calculate all offsets from D1, D2, . . . , D360. Do not repeat a block of similar SQL for 360 times.

After a quick research in the Postgresql documentation, my idea would be to use something like:

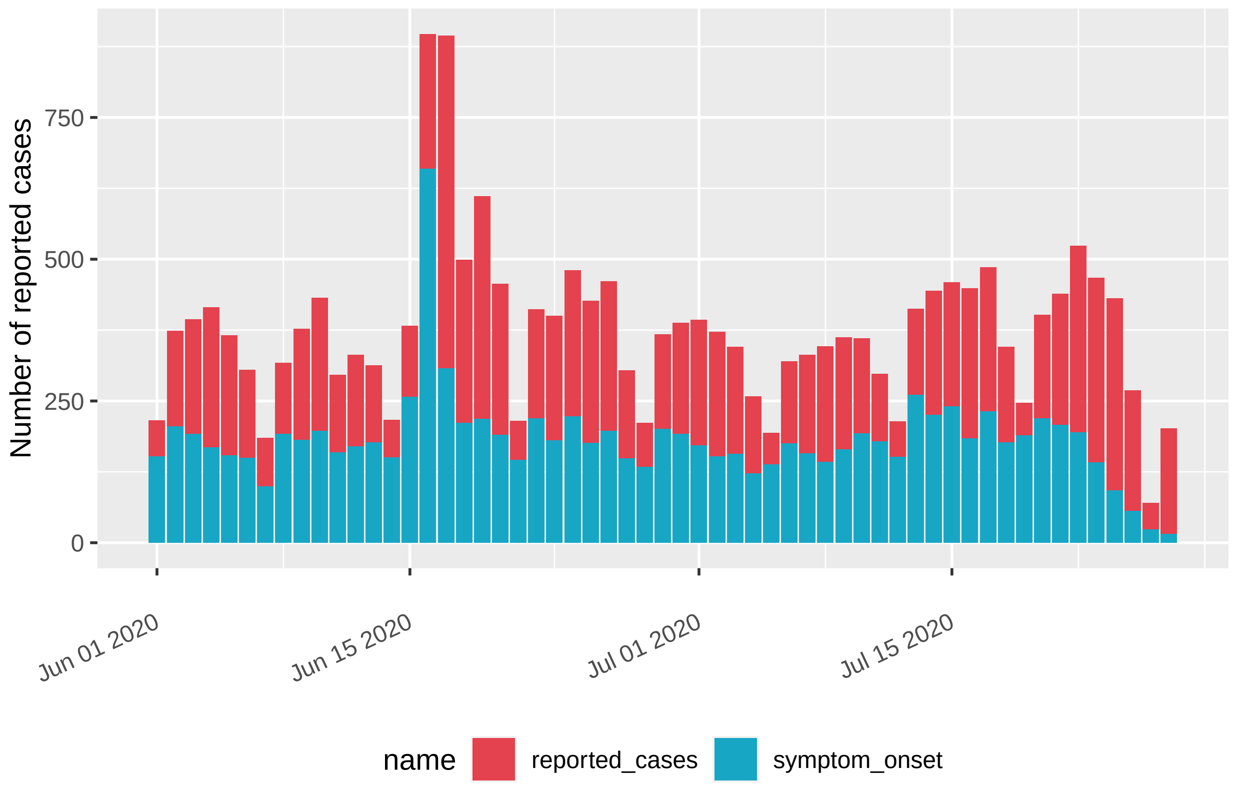
- Join with -> `WITH ranges as (select * from generate_series(1,360))`
- where `generate_series(1,360)` creating a list of offsets within the specified parameters `(1,360)` (included).

C - Current COVID-19 Cases in Germany

C.1 Create a chart that shows the number of daily cases by Symtom_Start_Date for June and July 2020.

- I start by simply importing and cleaning the file.
- Then I plot the cases by `symptom_start_date` and **if not available**, by `reporting_date`.

```r
library(tidyverse)
library(janitor)
library(ggplot2)
library(lubridate)
library(surveillance)

df <- read_delim("data/RKI_COVID19_Simplified_2020-07-29.csv", delim = ",")

df %>%
  janitor::clean_names() %>%
  ## I create the delay variable:
  mutate(delay = reporting_date - symptom_start_date) %>%
  mutate(delay = as.numeric(delay)) %>%
  # filter positive cases, some cases are negative;
  # I also remove delays that are extremely large
  filter(cases > 0) %>%
  filter(delay < 14) -> df


df %>%
  group_by(date = reporting_date) %>%
  summarise(reported_cases = sum(cases)) -> df_rep

df %>%
  group_by(date = symptom_start_date) %>%
  summarise(symptom_onset = sum(cases)) -> df_sym

df_sym %>%
  left_join(df_rep) %>%
  replace(is.na(.), 0) -> a

## ggplottin'...

a %>%
  filter(date >= "2020-06-01") %>%
  pivot_longer(
    -date
  ) %>%
  mutate(date = as.Date(date)) %>%
  ggplot(aes(x=date, y=value, fill=name))+
  geom_col()+
  scale_fill_manual(values = c("#e4424e", "#18A6C5"))+
  scale_x_date(date_labels = "%b %d %Y") +
  theme(axis.text.x = element_text(angle = 25, vjust = 0.5, hjust=1)) +
  theme(legend.position="bottom") +
  labs(x ="", y = "Number of reported cases")+
  labs(caption="Onset of symptoms, alternatively report date") +
  theme(plot.caption = element_text(hjust=0.5, size=rel(1.2)))
```



Onset of symptoms, alternatively report date

- This graph does not fully reflect the temporal progression of the pandemic, since the time intervals between actual onset of illness and diagnosis, reporting and data transmission can vary. Moreover, we know that COVID19 has a very high % of asymptomatic cases.
- The variable `delay` is a discrete random variable with some probability distribution (similar to the *incubation period* for any infectious disease, or the time an user takes to update his information into the **Clue** app… - :D)
- One can see that the closer we get to the last date in time, we have an impression of downward bias.
- … but how to solve this?

C.2 Find a way to estimate how many cases are still unreported for the Symtom_Start_Dates leading up to the 28th of July aka:

Nowcasting

- Nowcasting is somehow the opposite of Forecasting; basically we want to recreate the most credible situation in the **ideal** scenario of **no reporting delays**.
- Very common topic in Epidemiology, but also for Insurances claims in actuarian sciences.
- I can imagine a similar situation where **Clue** Data Team wants to use Nowcasting for inferring the reporting delays of its user base.

Epidemiologists - especially after the AIDS pandemic in the '80 for MSM (men who have sex in men) - developed a variety of techniques.

- Main statistical intuition is that we have **right-censored** data.
- We need to find a way to model the **delay** and somehow impute the missing % of unreported cases to the most recent observations.

There is an easy way in R via the package called `surveillance` (link here: CRAN) that computes the Nowcasting. The user must define:

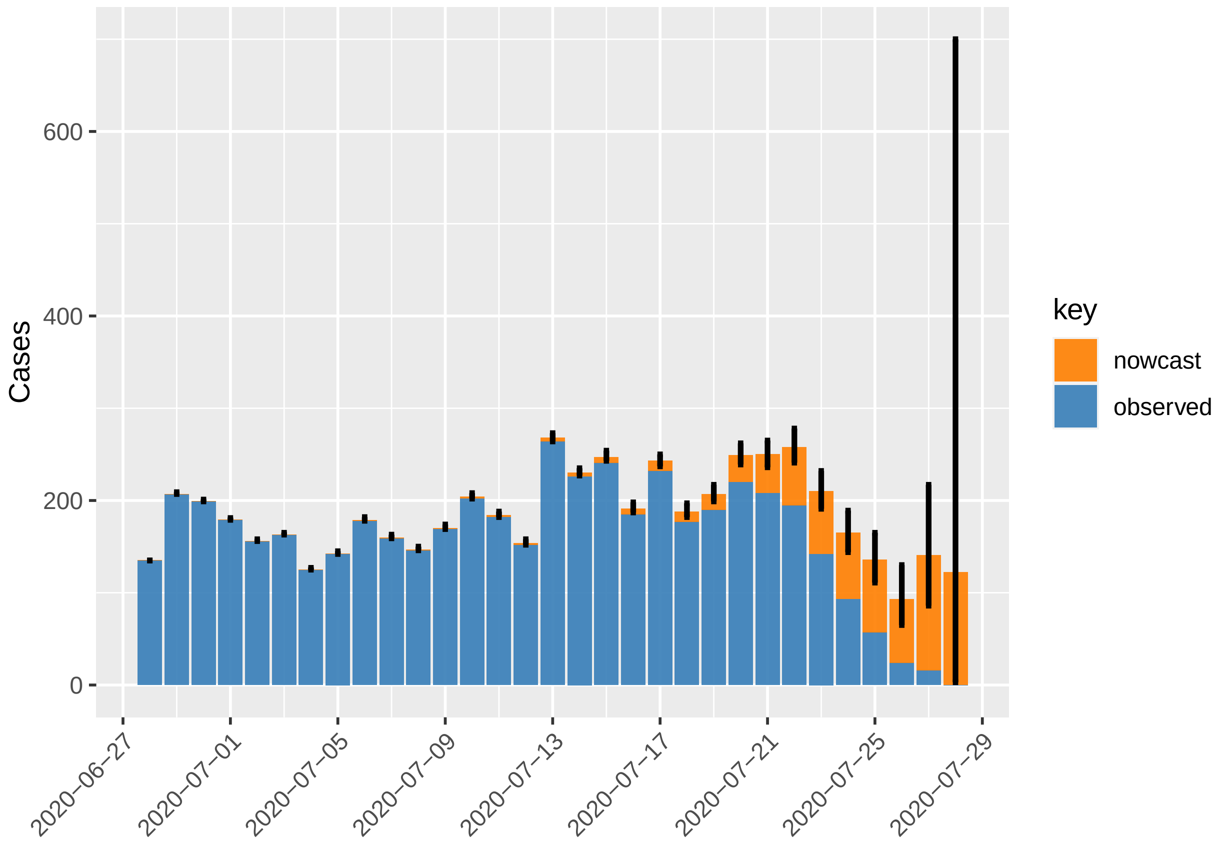
- the sliding window of the delay
- the probability distribution of the delay
- *big elephant in the room* = we assume time homogeneity of the delay, which is very unlikely especially in the COVID19 pandemic where the authories testing capabilities and availability veried dramatically over time. For this I am using a **Bayesian-truncated distribution** assuming **stability** over time via estimation of Poisson / Gamma, only in the `last m=30 days`
- The easiest way - actually - could be to model the `delay` value just by using its mean or median. This would actually very *naive* because a correct approach would be to look at `delay` as a random variable.

```
library(tidyverse)
library(janitor)
library(lubridate)
library(surveillance)

df <- read_delim("data/RKI_COVID19_Simplified_2020-07-29.csv", delim = ",") %>%
  janitor::clean_names() %>%
  mutate(delay = reporting_date - symptom_start_date) %>%
  # filter positive cases, some cases are negative
  filter(cases > 0) %>%
  filter(symptom_start_date >= "2020-06-01") %>%
  select(symptom_start_date, reporting_date, cases) %>%
  # the data is presented as "count", I use "uncount"
  ## -> in order to transform this time series to the INDIVIDUAL REPORT LEVEL
  uncount(cases)
df <- as.data.frame(df)

# Only do nowcasts for the last max_delay days!
now <- max(df$reporting_date)
max_delay <- 30
safePredictLag <- 0
so_range <- c(min(df$symptom_start_date, na.rm = TRUE), now)

#Fix nowcast time points so they don't depend on the imputed data.
nowcastDates <- seq(from = now - safePredictLag - max_delay,
                    to = now - safePredictLag, by = 1)

# create a "survelliance time series" object, quite self-explanatory
sts <- linelist2sts(
  as.data.frame(df),
  dateCol = "symptom_start_date",
  aggregate.by = "1 day",
  dRange = so_range)

nc.control <- list(
  N.tInf.max = 4e3,
  # Delay prior is modeled as stable over time via a Poisson / Gamma
  N.tInf.prior = structure("poisgamma",
                          mean.lambda = mean(observed(sts)),
                          var.lambda = 5 * var(observed(sts))
  ),
  predPMF = TRUE,
  dRange = so_range)

nc <- nowcast(now = now, when = nowcastDates, data = as.data.frame(df),
              dEventCol = "symptom_start_date",
              dReportCol = "reporting_date",
              aggregate.by = "1 day",
              D = 30,
              method = "bayes.trunc",
              ## only use last 30 days for the delay estimation
              m = 30,
              control = nc.control
)
```

```
## Building reporting triangle...
## No. cases:  10610
## No. cases within moving window:  4964
## bayes prep...
## (E,V) of prior for lambda = ( 182.931034482759,36716.6878684767 )
## bayes.trunc...
```

```
##Convert to tibble (in wide format)
nc_tidy <- nc %>%
  as_tibble() %>%
  # Add prediction interval
  mutate(pi_lower = nc@pi[,,1],  pi_upper=  nc@pi[,,2]) %>%
  # Return only time points which were nowcasted.
  filter(epoch %in% nowcastDates) %>%
  # Restrict to relevant columns
  select(date = epoch,
         observed,
         predicted = upperbound,
         predicted_lower= pi_lower,
         predicted_upper = pi_upper ) %>%
  # Reduce nowcast objects to only showing results during last 30
  # A consequence of using D=30 is that older delays than 30 days are not adjusted at all.
  filter(date > (max(date) - weeks(6))) %>%
  mutate(pred_obs = predicted - observed)

nc_df <- nc_tidy
nc_tidy <- nc_tidy %>%
  select(date, observed, pred_obs) %>%
  gather(key, value, - date) %>%
  ungroup()

# take the last case observed
last_linelist_case <- df %>%
  summarise(n = n(),
```

```r
            last_case = max(reporting_date)) %>%
  pull(last_case)

# Plot nowcasts + CI
nc_tidy %>%
  mutate(key = case_when(key == "pred_obs" ~ "nowcast",
                         TRUE ~ key)) %>%
  filter(date <= ymd(last_linelist_case)) %>%
  ggplot(aes(date, value)) +
  geom_col(aes(fill = key), alpha = 0.9)  +
  geom_errorbar(
    data = (nc_df %>%
              mutate(value = 1, key = NA) %>%
              filter(date > (max(date) - weeks(10)))),
    aes(ymin = predicted_lower, ymax = predicted_upper),
    width = 0.2, size = 1) +
  scale_fill_manual(values = c("#ff7f00", "#377eb8")) +
  scale_y_continuous(labels = scales::number_format(accuracy = 1)) +
  scale_x_date(date_breaks = "4 days") +
  labs(x = "", y = "Cases") +
  theme(axis.title.x = element_blank(), axis.text.x = element_text(angle = 45, hjust = 1))
```



C.3 Multiple approaches to correcting for the reporting delay

One can notice that closer we get to the end date, the higher the prediction interval, meaning that the nowcast gets more and more **unstable**.

1. My model here doesn't take into account all the covariates; I basically treat everyone equally.

- Sex
- Region (different regions can have different trasmission rules to authorities, some can be slower than others, or more affected than others.)
- Age group (different age groups can yield to different `delay` distribution or more asymptomatics (especially younger cohorts)

I would enrich the model adding the covariates.

2. I would explore different models available. In the last 2 years a lot of models assuming time/space have been developed. The state-of-the-art in Germany is the Nowcasting model used by Felix Günther (https://www.medrxiv.org/content/10.1101/2020.06.26.20140210v2), which is the actual nowcasting method applied in the official RKI reports ("Based on an persons age, the reporting week and weekday, we sampled the expected reporting delay from the original Weibull GAMLSS imputation model and derived the artificial disease onset date for each case").