EST-24107: Simulación

Profesor: Alfredo Garbuno Iñigo — Primavera, 2022 — Aplicaciones de *bootstrap*.
Objetivo: Que veremos.
Lectura recomendada: Referencia.

1. BOOTSTRAP PARAMÉTRICO

- Supongamos que tenemos una muestra $X_1, \ldots, X_N \overset{\text{iid}}{\sim} \mathbb{P}(x; \theta^\star)$. Es decir, tenemos un modelo paramétrico que da lugar a nuestros datos.
- En este tipo de problemas de inferencia suponemos la familia paramétrica

$$\mathcal{P}_\Theta = \{\mathbb{P}(\cdot; \theta) : \theta \in \Theta\} \,, \tag{1}$$

  donde Θ denota el `espacio parametral` (los posibles valores de los parámetros de un modelo).
- En esta tarea no conocemos el valor específico de $\theta^\star$. Por lo tanto, lo tenemos que estimar. Usualmente a través de resolver un problema de optimización

$$\hat{\theta}_{\mathsf{MLE}} = \arg\max_{\theta \in \Theta} \prod_{i=1}^{N} \mathbb{P}(X_i; \theta) \,. \tag{2}$$

  cuya solución llamamos `estimador de máxima verosimilitud`.
- Adicional, nos encantaría poder establecer una cuantificación de la incertidumbre sobre este valor. En particular, reportar

$$\mathsf{ee}\left(\hat{\theta}_{\mathsf{MLE}}\right) = \left(\mathbb{V}(\hat{\theta}_{\mathsf{MLE}})\right)^{1/2} \,. \tag{3}$$

- Para algunos modelos es fácil poder estimarlo, utilizando propiedades asintóticas y/o analíticas de nuestros estimadores (lo ven en el curso de Estadística Matemática).
- Sin embargo, ¿qué pasa si nuestro estimador no tiene fórmulas cerradas para el cálculo del error estándar? ¿O si nuestro tamaño de muestra no sugiere que los supuestos del TLC se cumplen?

1.0.1. ***Definición*** *[Método* bootstrap *paramétrico]:* El error estándar estimado para $\hat{\theta}_{\mathsf{MLE}}$ por medio del *bootstrap* paramétrico se calcula como sigue:

1. Se calcula $\hat{\theta}_{\mathsf{MLE}}$ para la muestra observada.
2. Se simula una muestra iid de tamaño N de $X_1^{(b)}, \ldots, X_N^{(b)} \overset{\text{iid}}{\sim} \mathbb{P}(x; \hat{\theta}_{\mathsf{MLE}})$ (muestra *bootstrap*).
3. Se recalcula el estimador de máxima verosimilitud para la muestra *bootstrap*, lo cual denotamos por $\hat{\theta}_{\mathsf{MLE}}^{(b)} = s(X_1^{(b)}, \ldots, X_N^{(b)})$.
4. Se repiten los pasos 2–3 muchas veces ($B = 1{,}000$ – $10{,}000$).
5. Se calcula la desviación estándar de los valores $\hat{\theta}_{\mathsf{MLE}}^{(b)}$ obtenidos. Este es el error estándar estimado para el estimador $\hat{\theta}_{\mathsf{MLE}}$.

1.0.2. ***Observación:***

- Nota cómo cambiamos el mecanismo de remuestreo $\hat{\mathbb{P}}_N$ por $\mathbb{P}(x; \hat{\theta}_{\mathsf{MLE}})$.
- En espíritu es lo mismo, pero estamos dispuestos a incorporar mayores supuestos en nuestra tarea de inferencia.

1.1. Ejemplo: Datos normales

Como ejercicio, podemos encontrar los estimadores de máxima verosimilitud cuando tenemos una muestra $X_1, \ldots, X_N \overset{\text{iid}}{\sim} \mathsf{N}(\mu, \sigma^2)$ (puedes derivar e igualar a cero para encontrar el mínimo). También podemos resolver numéricamente.

Supongamos que tenemos la siguiente muestra:

```
set.seed(41852)
muestra ← rnorm(150, mean = 1, sd = 2)
```

Para la cual podemos calcular los estimadores de máxima verosimilitud de un modelo normal

```
mle.obs ← broom::tidy(MASS::fitdistr(muestra, "normal")) ▷
  tibble::column_to_rownames("term")
mle.obs
```

```
     estimate std.error
mean    1.136    0.1502
sd      1.839    0.1062
```

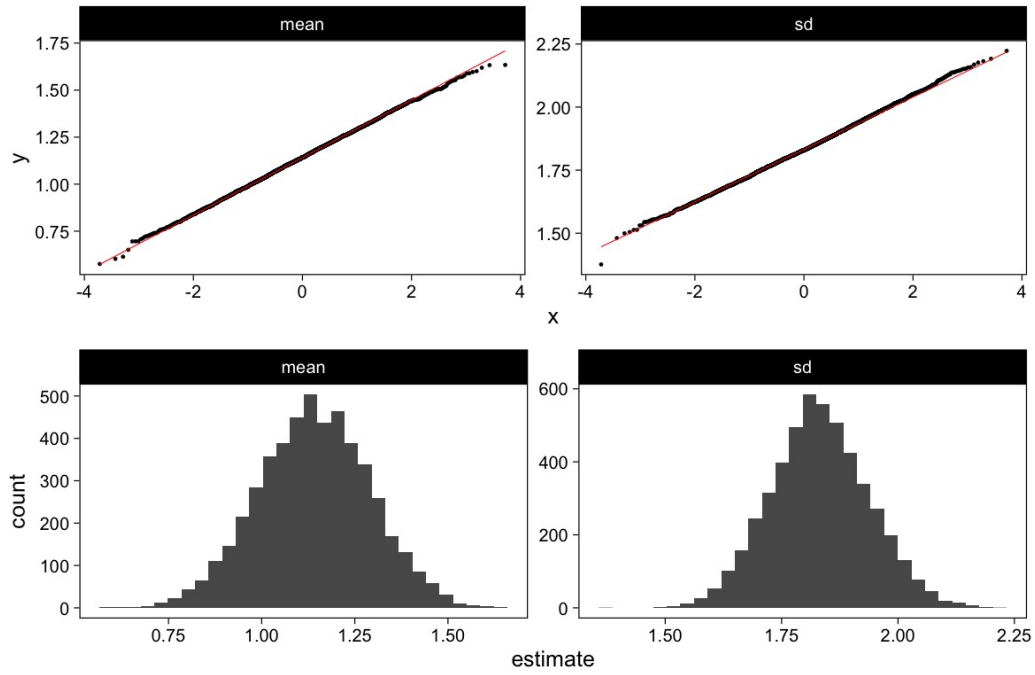
Con esta estimación podemos definir el proceso de remuestreo.

```
## paso 1: define el estimador
estimador_mle ← function(datos, modelo = "normal"){
  datos ▷
    MASS::fitdistr(modelo) ▷
    broom::tidy() ▷
    select(-std.error)
}
```

```
## paso 2: define el proceso de remuestreo
paramboot_sample ← function(data){
  rnorm(length(data),
        mean = mle.obs["mean", "estimate"],
        sd = mle.obs["sd", "estimate"])
}
```

```
## paso 3: define el paso bootstrap
paso_bootstrap ← function(id){
  muestra ▷
    paramboot_sample() ▷
    estimador_mle()
}
```

```
## paso 4: aplica bootstrap parametrico
boot_mle ← map_df(1:5000, paso_bootstrap)
```



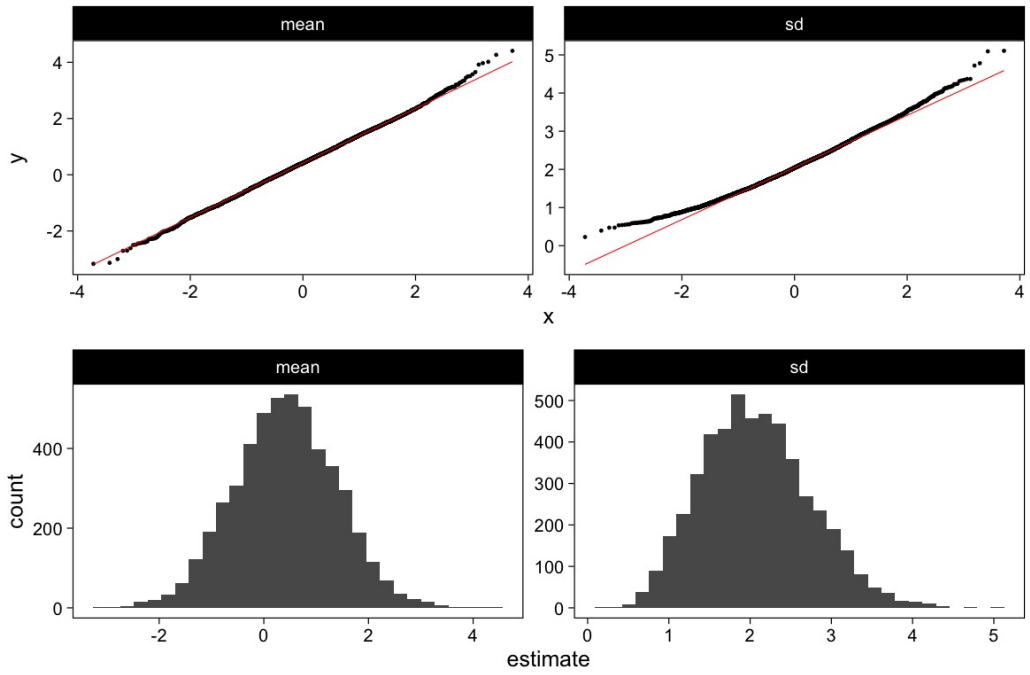
Las distribuciones son aproximadamente normales. Nótese que esto no siempre sucede, especialmente con parámetros de dispersión como σ. (Examina las curvas de nivel del ejemplo de arriba).

Ahora, supongamos que tenemos una muestra más chica. Repasa los pasos para asegurarte que entiendes el procedimiento:

```
set.seed(4182)
muestra ← rnorm(6, mean = 1, sd = 2)
mle.obs ← broom::tidy(MASS::fitdistr(muestra, "normal")) ▷
  tibble::column_to_rownames("term")
mle.obs
```

```
     estimate std.error
mean   0.3979    0.9794
sd     2.3990    0.6925
```

```
## paso 4: aplica bootstrap parametrico
boot_mle ← map_df(1:5000, paso_bootstrap)
```



Donde vemos que la distribución de σ tienen sesgo a la derecha, pues en algunos casos obtenemos estimaciones muy cercanas a cero. Podemos usar intervalos de percentiles.

1.2. Comparación bootstrap paramétrico y no paramétrico

```
propinas ← read_csv("data/propinas.csv",
                    progress = FALSE,
                    show_col_types = FALSE) ▷
  mutate(id = 1:244)
```

```
## paso 1: define el estimador
estimador ← function(split, ...){
  muestra ← analysis(split) ▷ group_by(momento)
  muestra ▷
    summarise(estimate = mean(cuenta_total), .groups = 'drop') ▷
    mutate(term = momento)
}
```

```
## paso 2 y 3: remuestrea y calcula estimador
boot_samples ← bootstraps(propinas, strata = momento, 500) ▷
  mutate(res_boot = map(splits, estimador))
## paso 4: construye intervalos de confianza
intervalos_noparam ← boot_samples ▷
  int_pctl(res_boot, alpha = 0.05) ▷
  mutate(across(where(is.numeric), round, 2))
intervalos_noparam
```

```
# A tibble: 2 × 6
  term    .lower .estimate .upper .alpha .method
  <chr>    <dbl>     <dbl>  <dbl>  <dbl> <chr>
```

```
1 Cena     19.7      20.8   22.0    0.1 percentile
2 Comida   15.7      17.2   18.7    0.1 percentile
```

```
## paso 1: define estimador
estimador_mle_grupos ← function(muestra, modelo = "normal") {
  muestra ▷
    select(momento, cuenta_total) ▷
    group_by(momento) ▷
    nest(data = cuenta_total) ▷
    summarise(mle = map(data, function(x) {
      nobs ← nrow(x)
      unlist(x) ▷
        estimador_mle(modelo = modelo) ▷
        mutate(n = nobs)
    }))
}
```

```
mle.obs ← estimador_mle_grupos(propinas, "normal")
mle.obs ▷ unnest(mle)
```

```
# A tibble: 4 × 4
  momento term  estimate     n
  <chr>   <chr>    <dbl> <int>
1 Cena    mean     20.8    176
2 Cena    sd        9.12   176
3 Comida  mean     17.2     68
4 Comida  sd        7.66    68
```

```
## paso 2: define proceso de remuestreo
param_boot_grupos ← function(estimadores){
  estimadores ▷
    group_by(momento) ▷
    mutate(simulaciones = map(mle, function(m){
      tibble(cuenta_total = rnorm(m$n[1], m$estimate[1], sd = m$estimate[2]))
    })) ▷
    unnest(simulaciones) ▷
    select(-mle) ▷
    ungroup()
}
```

```
## paso 3: paso bootstrap
paso_bootstrap_grupos ← function(id){
  param_boot_grupos(mle.obs) ▷
    estimador_mle_grupos()
}
```

```
## paso 4: aplica bootstrap y presenta intervalos
intervalos_param ← tibble(id = 1:500)▷
  mutate(estimadores = map(id, paso_bootstrap_grupos)) ▷
  unnest(estimadores) ▷
  unnest(mle) ▷
```

```
  group_by(momento, term) ▷
  summarise(.lower = quantile(estimate, 0.025),
            .estimate = mean(estimate),
            .upper = quantile(estimate, 0.975),
            .alpha = .05,
            .method = "percentile (normal)", .groups = "drop") ▷
  filter(term == "mean") ▷ select(-term)
intervalos_param
```

```
# A tibble: 2 × 6
  momento .lower .estimate .upper .alpha .method
  <chr>    <dbl>     <dbl>  <dbl>  <dbl> <chr>
1 Cena      19.6      20.8   22.1    0.1 percentile (normal)
2 Comida    15.3      17.1   18.8    0.1 percentile (normal)
```

```
# A tibble: 2 × 6
  term   .lower .estimate .upper .alpha .method
  <chr>   <dbl>     <dbl>  <dbl>  <dbl> <chr>
1 Cena     19.7      20.8   22.0    0.1 percentile
2 Comida   15.7      17.2   18.7    0.1 percentile
```

```
# A tibble: 1 × 6
  term  .lower .estimate .upper .alpha .method
  <chr>  <dbl>     <dbl>  <dbl>  <dbl> <chr>
1 Cena    17.8      20.8   23.9    0.1 percentile (exponential)
```

1.3. Ventajas y desventajas de bootstrap paramétrico

- Ventaja: el *bootstrap* paramétrico puede dar estimadores más precisos e intervalos más angostos y bien calibrados que el no paramétrico, **siempre y cuando el modelo teórico sea razonable.**
- Desventaja: Es necesario decidir el modelo teórico, que tendrá cierto grado de desajuste vs. el proceso generador real de los datos. Si el ajuste es muy malo, los resultados tienen poca utilidad. Para el no paramétrico no es necesario hacer supuestos teóricos.
- Ventaja: el *bootstrap* paramétrico puede ser más escalable que el no paramétrico, pues no es necesario cargar y remuestrear los datos originales, y tenemos mejoras adicionales cuando tenemos expresiones explícitas para los estimadores de máxima verosimilitud (como en el caso normal, donde es innecesario hacer optimización numérica).
- Desventaja: el *bootstrap* paramétrico es conceptualmente más complicado que el no paramétrico, y como vimos arriba, sus supuestos pueden ser más frágiles que los del no paramétrico.

2. ESTRUCTURAS DE DATOS

2.1. Datos agrupados

```
departures_raw ← read_csv("https://raw.githubusercontent.com/rfordatascience/
    tidytuesday/master/data/2021/2021-04-27/departures.csv", show_col_types =
    FALSE, progress = FALSE)
```

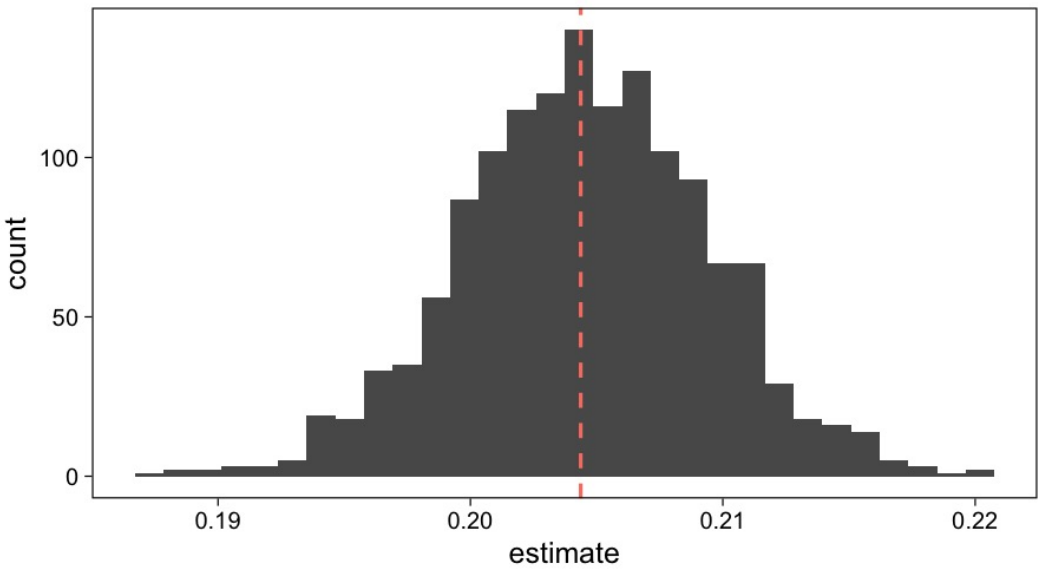
```
departures ← departures_raw %>%
  filter(departure_code < 9) %>%
  mutate(involuntary = if_else(departure_code %in% 3:4, "involuntary", "other"
      )) %>%
  filter(fyear > 1995, fyear < 2019)

departures
```

```
df ← departures %>%
  count(fyear, involuntary) %>%
  pivot_wider(names_from = involuntary, values_from = n)
df
```

```
## paso 1: define estimador
estimador_razon ← function(split, ...){
  analysis(split) ▷
    count(fyear, involuntary) ▷
    pivot_wider(names_from = involuntary, values_from = n) ▷
    mutate(prop = involuntary/(involuntary + other)) ▷
    summarise(estimate = mean(prop)) ▷
    pull(estimate)
}
```

```
dif_boot ← bootstraps(departures, 1400, apparent = TRUE) ▷
  mutate(estimate = map_dbl(splits, estimador_razon))
```

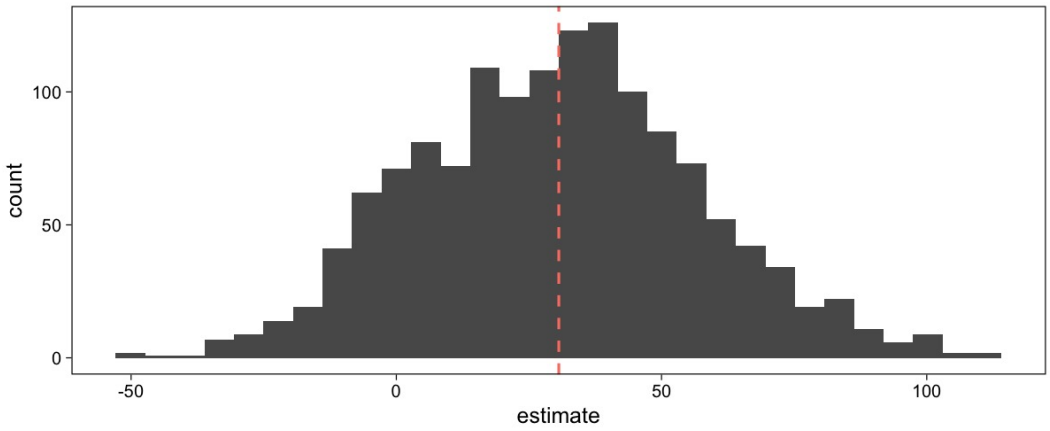


2.2. Muestras independientes

```
data_ej ← tibble(respuesta = c(94, 197, 16, 38,  99, 141, 23, 52, 104, 146,
    10, 51, 30, 40, 27, 46),
      tipo = c(rep("tratamiento", 7), rep("control", 9)))
```

```
  diferencia_medias ← function(split, ...){
  analysis(split) ▷
    group_by(tipo) ▷
    summarise(promedio = mean(respuesta)) ▷
    pivot_wider(names_from = tipo, values_from = promedio) ▷
    mutate(estimate = tratamiento - control) ▷
    pull(estimate)
}
```

```
dif_boot ← bootstraps(data_ej, 1400, strata = tipo, apparent = TRUE) ▷
  mutate(estimate = map_dbl(splits, diferencia_medias))
```

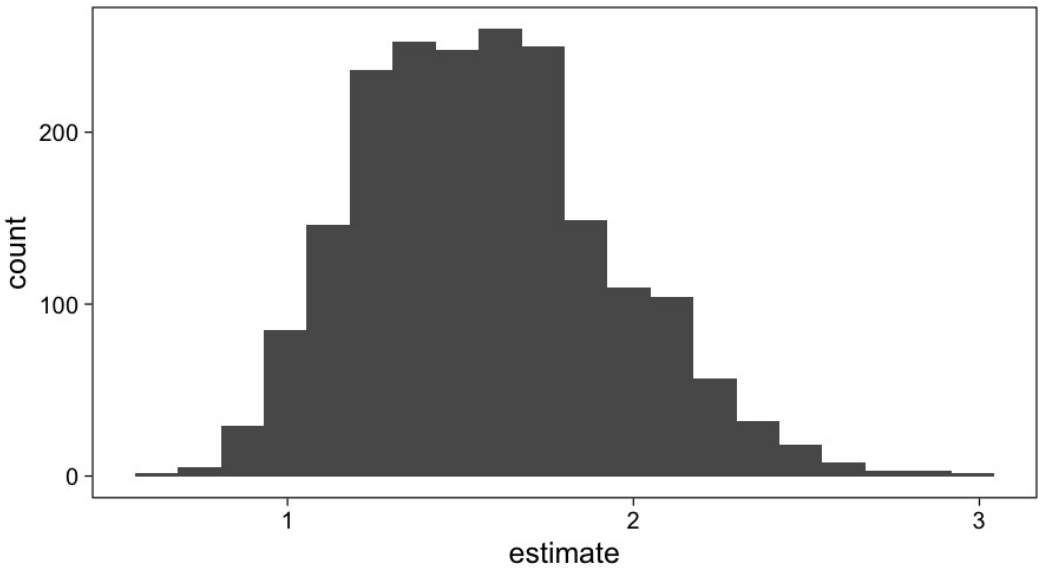


2.3. Datos pareados

```
dormir ← sleep ▷
  pivot_wider(names_from = group,
              names_prefix = "medicina_",
              values_from = extra)
dormir
```

```
estimador_dif_pareados ← function(split, ...){
  muestra ← analysis(split)
  muestra %>%
    mutate(dif_2_menos_1 = medicina_2 - medicina_1) %>%
    summarise(estimate = mean(dif_2_menos_1), .groups = "drop") %>%
    mutate(term = "diferencia 2 vs 1")
}
```

```
pareados_boot ← bootstraps(dormir,  2000, apparent = TRUE) %>%
  mutate(res_boot = map(splits, estimador_dif_pareados))
```




```
pareados_boot %>% int_pctl(res_boot, 0.05)
```

```
# A tibble: 1 × 6
  term               .lower .estimate .upper .alpha .method
  <chr>               <dbl>     <dbl>  <dbl>  <dbl> <chr>
1 diferencia 2 vs 1    0.97      1.58   2.39   0.05 percentile
```

2.4. Series temporales

Moving blocks bootstraps

1. origin windows con cumulative apagado
2. Boostrap sobre los slices

3. MODELOS DE REGRESIÓN

3.1. Observaciones

3.2. Residuales