ExamenParcial

Yalidt Diaz - 141394
Bruno Gonzalez - 150370

7/10/2019

EXAMEN PARCIAL

1. Tablas de conteos y bootstrap

En la sección de visualización vimos un ejemplo de tabla de perfiles.

En este ejercicio construiremos intervalos de confianza para una tabla de perfiles usando bootstrap. Usaremos los datos de tomadores de te (del paquete @factominer):

Nos interesa ver qué personas compran té suelto (`unpacked`), y de qué tipo (`Tea`). Empezamos por ver las proporciones que compran té según su empaque (en bolsita o suelto):

how	n	%
tea bag	170	57
tea bag+unpackaged	94	31
unpackaged	36	12

La tabla de arriba es poco informativa, buscamos comparar grupos, por ejemplo, queremos investigar si hay diferencias en los patrones de compra (en términos de precio o marca) dependiendo del tipo de té que consumen.

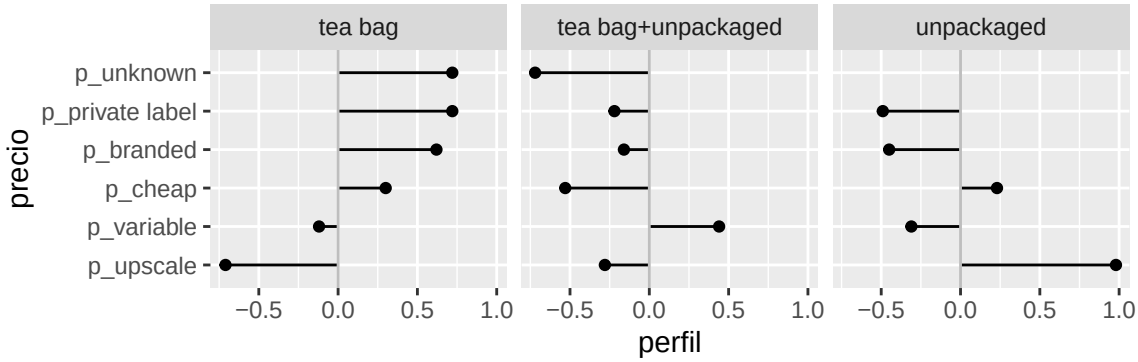
price	tea bag	tea bag+unpackaged	unpackaged
p_branded	41	21	14
p_cheap	3	1	3
p_private label	9	4	3
p_unknown	6	1	0
p_upscale	8	20	56
p_variable	32	52	25

Para facilitar la comparación podemos calcular *perfiles columna*. Comparamos cada una de las columnas con la columna marginal (la tabla de tipo de estilo de té):

price	tea bag	tea bag+unpackaged	unpackaged	promedio
p_private label	0.72	-0.22	-0.49	5
p_unknown	0.72	-0.72	-1.00	4
p_branded	0.62	-0.16	-0.45	25
p_cheap	0.30	-0.53	0.23	2
p_variable	-0.12	0.44	-0.31	36
p_upscale	-0.71	-0.28	0.98	28

Leemos esta tabla como sigue: por ejemplo, los compradores de té suelto (`unpacked`) compran té fino (`upscale`) a una tasa casi el doble (0.98) que el promedio.

También podemos graficar como:



Observación: hay dos maneras de construir la columna promedio: tomando los porcentajes sobre todos los datos, o promediando los porcentajes de las columnas como en este ejemplo.

1. Utiliza bootstrap para crear intervalos de confianza sobre los perfiles de la última tabla.

Primero definimos la funcion bootstrap, la cual seleccionará las muestras aleatoreas con reemplazo, y posterior mente calcula la estadística de interés, que en este caso son los *perfiles columna*.

```
perfiles_boot <- function(x){
  m <- sample_n(x, size =  300 , replace = TRUE)
  tabla <- m %>%
    count(how, price) %>%
    group_by(how) %>%
    mutate(prop_price = (100 * n / sum(n))) %>%
    group_by(price) %>%
    mutate(prom_prop = mean(prop_price)) %>%
    mutate(perfil = (prop_price / prom_prop - 1) %>% round(2))
  tabla
}
```

Despues corremos $B = 10000$ replicaciones Bootstrap.

```
perfiles_rep <- rerun(10000, perfiles_boot(tea)) %>% bind_rows(.id = 'muestra')
```

Posteriormente calculamos los errores estándard

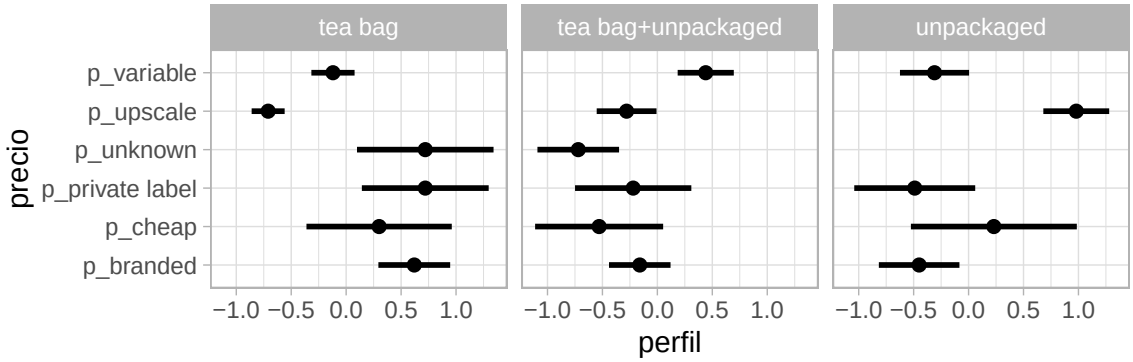
```
perfiles_se <- perfiles_rep %>%
  group_by(how, price) %>%
  summarise(se = sd(perfil))
```

Por último calculamos los intervalos

```
perfiles_int <- tabla %>%
  left_join(perfiles_se) %>%
  mutate(Int_inf = perfil+qnorm(0.025)*se, Int_sup = perfil+qnorm(0.975)*se)
kable(select(perfiles_int, how, price, perfil, Int_inf, Int_sup), digits = 2)
```

how	price	perfil	Int_inf	Int_sup
tea bag	p_branded	0.62	0.29	0.95
tea bag	p_cheap	0.30	-0.36	0.96
tea bag	p_private label	0.72	0.14	1.30
tea bag	p_unknown	0.72	0.10	1.34
tea bag	p_upscale	-0.71	-0.86	-0.56
tea bag	p_variable	-0.12	-0.32	0.08
tea bag+unpackaged	p_branded	-0.16	-0.44	0.12
tea bag+unpackaged	p_cheap	-0.53	-1.11	0.05
tea bag+unpackaged	p_private label	-0.22	-0.75	0.31
tea bag+unpackaged	p_unknown	-0.72	-1.09	-0.35
tea bag+unpackaged	p_upscale	-0.28	-0.55	-0.01
tea bag+unpackaged	p_variable	0.44	0.18	0.70
unpackaged	p_branded	-0.45	-0.82	-0.08
unpackaged	p_cheap	0.23	-0.53	0.99
unpackaged	p_private label	-0.49	-1.04	0.06
unpackaged	p_upscale	0.98	0.68	1.28
unpackaged	p_variable	-0.31	-0.62	0.00

2. Modifica la última gráfica para representar los intervalos de confianza.



3. Comenta tus observaciones.

En la categoria de “tea bag” la mayor tasa de compra corresponde a los precios “private label” y “unknown” que son .72 aproximadamente, sin embargo el intervalo deconfianza para ambdos casos es relativamente alto, lo que le quita certidumbre a la estimación. Por otro lado, en est misma categoría, los que tienen menor perfil de compra es “p_upscale”, con un intervalo muy chico lo que permite concluir con mayor certeza que este patrón se mantiene en la población total.

En el caso de “tea_bag+unpackaged” la mayor tasa de compra es a una precio “p_variable”,cuyo intervalo se enceuntra entre .18 y.70, y la menor tasa(negativa) es a precio “p_known”. Por útimo, en la categoría unpackaged la mayor tasa de compra es a precio “p_upscale” a una tasa de casi el doble .89 con un intervalo de confianza de .68 y 1.28, en el caso de la menor tasa de compra se obtuvó que la menor fue a precio p_private_label.

En general, aquellos intervalos con mayor rango son aquellos donde hay menos observaciones(‘p_cheap’, ‘p_private label’,‘p_unknown’). Estos intervalos con tanta variación y que tienen una longuitud (segmento) muy amplia no son tan confiables, se esperaría que la longuitud fuera lo menor posible para tener mayor certeza de que valor tomaría la variable estimada.

En conclusión pareciera que el mejor comportamiento de compra se desarrolla en la categoría p_unpackaged a un precio upscale, por lo que se recomendaría trabajar en dicho segmento, porque ahí la gente tiene un perfil de compra mas agudo que en los demás casos.

2. Cuantificando el error Monte Carlo

Recordemos que ante la pregunta ¿cuántas muestras bootstrap se necesitan? el error que podemos disminuir al aumentar el número de replicaciones es el error de Monte Carlo, y una manera de cuantificarlo es haciendo bootstrap del bootstrap.

Retomemos el ejemplo de la media de las calificaciones de ENLACE de español 3o de primaria en el estado de México. Nos interesa la media de las calificaciones y usaremos el estimador *plug-in*.

1. Crea un intervalo del 90% para $\hat{\theta}$ usando los percentiles de la distribución bootstrap, y $B = 100$ replicaciones.

Primero creamos la función bootstrap que genera las muestras y calcula el parámetro, en este caso, la mediana.

```
enlace_boot <- function(x,col){
  col <- enquo(col)
  n <- nrow(x)
  muestra <- sample_n(x,n, replace = TRUE)
  muestra %>%
    select(!!col) %>%
    unlist() %>%
    median()
}
```

Posteriormente se hacen las $B = 100$ simulaciones bootstrap.

```
enlace_rep <- rerun(100, enlace_boot(enlace,esp_3))%>%
  flatten_dbl()
```

Por último, calculamos el intervalos usando los percentiles de la distrubucion bootstrap.

```
en_int <- quantile(enlace_rep, c(0.05, 0.95))
en_int
```

```
##  5% 95%
## 546 549
```

Las calificaciones de 3er año de primaria en el estado de México se encuentra en un intervalo al 95% de confianza entre 546 y 549 puntos.

2. Podemos estimar el error estándar de Monte Carlo de los extremos de los intervalos (percentiles 0.05 y 0.95) haciendo bootstrap de la distribución bootstrap:

- Selecciona muestras con reemplazo de tamaño B de la distribución bootstrap,
- Calcula los percentiles de interés (0.05 y 0.95),

Primero construimos la función bootstrap de la distribución bootstrap

```
enlace_boot_boot <- function(x){
  n <- length(x)
  muestra <- sample(x, size = n, replace = TRUE)
  tibble(SE_inf = quantile(muestra, c(0.025,0.975))[1], SE_sup = quantile(muestra, c(0.025,0.975))[2])
}
```

Con la función hacemos las repeticiones

```
enlace_boot_rep <- rerun(1000,enlace_boot_boot(enlace_rep)) %>%
  bind_rows(.id = 'muestra')
```

- Calcula la desviación estándar de los percentiles (una para cada extremo), esta será tu aproximación al error de Monte Carlo

```
EMC100 <- map_dbl(enlace_boot_rep, sd)
```

3. ¿Cuál es el error estándar de Monte Carlo con $B = 100, 1000, 10000$ replicaciones para cada extremo del intervalo de percentiles?

Para el caso de $B = 100$

```
EMC100[2:3]
```

```
##    SE_inf    SE_sup
## 0.2094879 0.1596374
```

Para el caso de $B = 1000$

```
enlace_rep <- rerun(1000, enlace_boot(enlace,esp_3))%>%flatten_dbl()
enlace_boot_rep <- rerun(1000,enlace_boot_boot(enlace_rep)) %>%
  bind_rows(.id = 'muestra')
map_dbl(enlace_boot_rep, sd)[2:3]
```

```
##    SE_inf    SE_sup
## 0.0000000 0.0890622
```

Para el caso de $B = 10000$

```
enlace_rep <- rerun(10000, enlace_boot(enlace,esp_3))%>%flatten_dbl()
enlace_boot_rep <- rerun(1000,enlace_boot_boot(enlace_rep)) %>%
  bind_rows(.id = 'muestra')
map_dbl(enlace_boot_rep, sd)[2:3]
```

```
##     SE_inf     SE_sup
## 0.00000000 0.02234949
```

Las corridas muestran la que el error Monte Carlo va disminuyendo conforme se aumentan el número de replicaciones. Con la simulación que obtuvimos vemos que conforme va aumentando el número de replicaciones(simulaciones) el error de Monte Carlo va dismunuyendo, tendiendo a cero, por eso los intervalos que obtuvimos son cada vez más pequeños y cercanos a cero.

3. Cobertura de intervalos de confianza

En este problema realizarás un ejercicio de simulación para comparar la exactitud de distintos intervalos de confianza. Simularás muestras de una distribución Poisson con parámetro $\lambda = 2.5$ y el estadístico de interéses $\theta = exp(-2\lambda)$.

Sigue el siguiente proceso:

i) Genera una muestra aleatoria de tamaño $n = 60$ con distribución $Poisson(\lambda)$, parámetro $\lambda = 2.5$ (en R usa la función `rpois()`).

ii) Genera $10,000$ muestras bootstrap y calcula intervalos de confianza del 95% para $\hat{\theta}$ usando 1) el método normal, 2) percentiles y 3) BC_a.

Primero definimos la función del parámetro

```
poiss_boot <- function(x, ind){
  exp(-2*mean(x[ind]))
}
```

Posteriormente definimos la funcion que genera los intervalos de confianza

```
poiss_intervalos <- function(n=60) {
  poiss_muestra <- rpois(n,2.5)

  poiss_rep <-  boot(poiss_muestra, poiss_boot,10000)
  poiss_int <-boot.ci(poiss_rep, type = c("norm", "perc", "bca"))

  data.frame(metodo = c('normal','percentil','BCa'),
             theta = poiss_int$t0,
             inferior = c(poiss_int$normal[2],poiss_int$percent[4],poiss_int$bca[4]),
             superior = c(poiss_int$normal[3],poiss_int$percent[5],poiss_int$bca[5]))
}
```

Finalmente, los intervalos son:

```
poiss_intervalos() %>%
  kable(digits = 4)
```

metodo	theta	inferior	superior
normal	0.005	0.0003	0.0089
percentil	0.005	0.0023	0.0107
BCa	0.005	0.0022	0.0101

iii) Revisa si el intervalo de confianza contiene el verdadero valor del parámetro ($\theta = exp(-2 \cdot 2.5)$), en caso de que no lo contenga registra si falló por la izquierda o falló por la derecha.

Los tres intervalos de confianza contienen el verdadero valor del parámetro 0.0067379

a) Repite el proceso descrito 1000 veces y llena la siguiente tabla:

Primero corremos las 100 repeticiones

```
poiss_rep_int <- rerun(1000, poiss_intervalos()) %>% bind_rows(.id = 'muestra')
```

Posteriormente calculamos los fallos y combertura

```
poiss_rep_int <- poiss_rep_int %>%
  mutate(fallo_izquierda = exp(-2*2.5)<inferior,
         fallo_derecha = exp(-2*2.5)>superior,
         Longitud = superior-inferior)
```

Así tenemos la siguiente tabla:

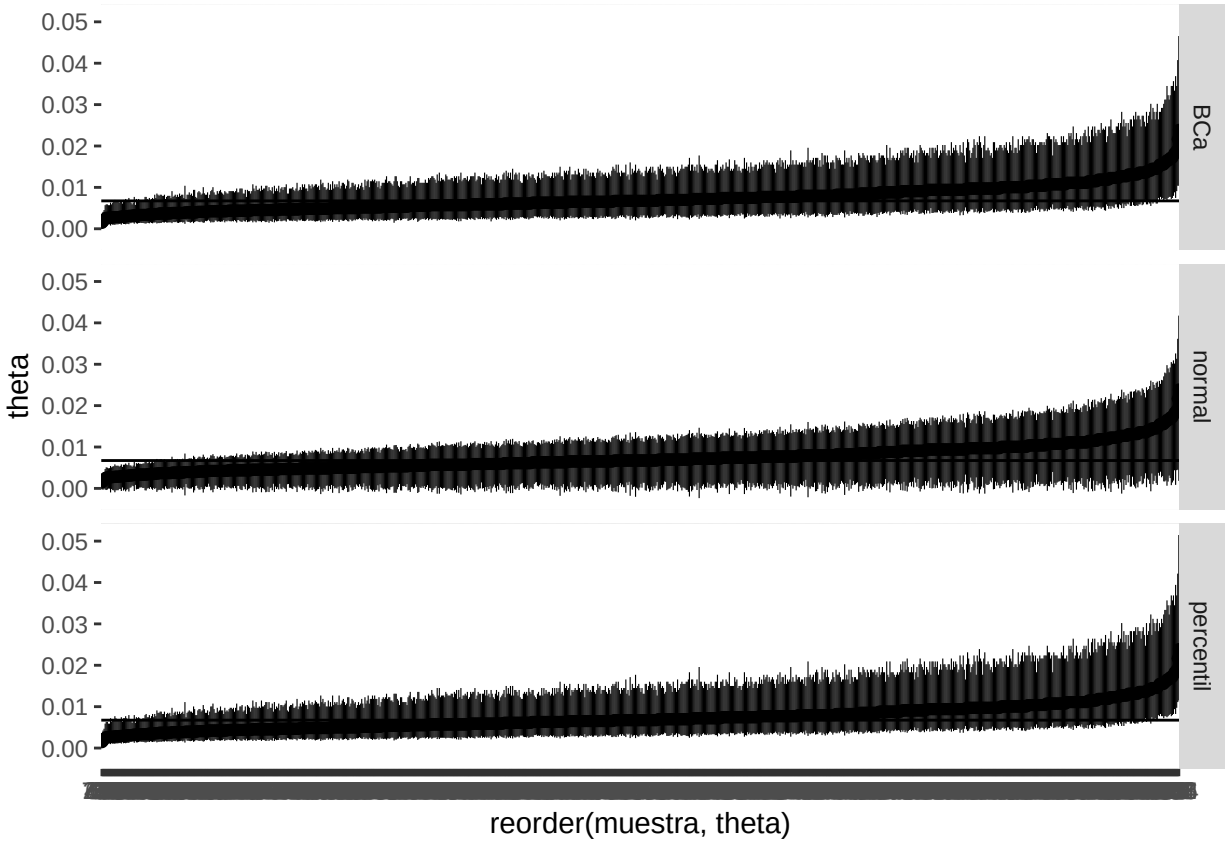
```
poiss_rep_int %>%
  group_by(metodo) %>%
  summarise(P_fallo_izquierda = sum(fallo_izquierda)/n(),
            P_fallo_derecha = sum(fallo_derecha)/n(),
            Cobertura = 1 - P_fallo_izquierda - P_fallo_derecha,
            Longitud_promedio = mean(Longitud)) %>%
  kable()
```

metodo	P_fallo_izquierda	P_fallo_derecha	Cobertura	Longitud_promedio
BCa	0.021	0.022	0.957	0.0117646
normal	0.000	0.058	0.942	0.0126198
percentil	0.027	0.014	0.959	0.0123914

La columna cobertura es una estimación de la cobertura del intervalo basada en las simulaciones, para calcularla simplemente escribe el porcentaje de los intervalos que incluyeron el verdadero valor del parámetro. La longitud promedio es la longitud promedio de los intervalos de confianza bajo cada método.

b) Realiza una gráfica de páneles, en cada panel mostrarás los resultados de uno de los métodos (normal, percentiles y BC_a), en el vertical graficarás los límites de los intervalos.

```
  ggplot(poiss_rep_int) +
    geom_pointrange(aes(x = reorder(muestra,theta),
                        ymin = inferior,
                        y=theta,
                        ymax = superior),
                    size=0.2) +
    geom_hline(yintercept = exp(-2.5*2)) +
    facet_grid(metodo~.)
```



c) Repite los incisos a) y b) seleccionando muestras de tamaño 300.

Primero hacemos las repeticiones con el tamano de muestra 300.

```
poiss_rep_int_300 <- rerun(1000, poiss_intervalos(300))%>%
  bind_rows(.id = 'muestra') %>%
  mutate(fallo_izquierda = exp(-2*2.5)<inferior,
         fallo_derecha = exp(-2*2.5)>superior,
         Longitud = superior-inferior)
```

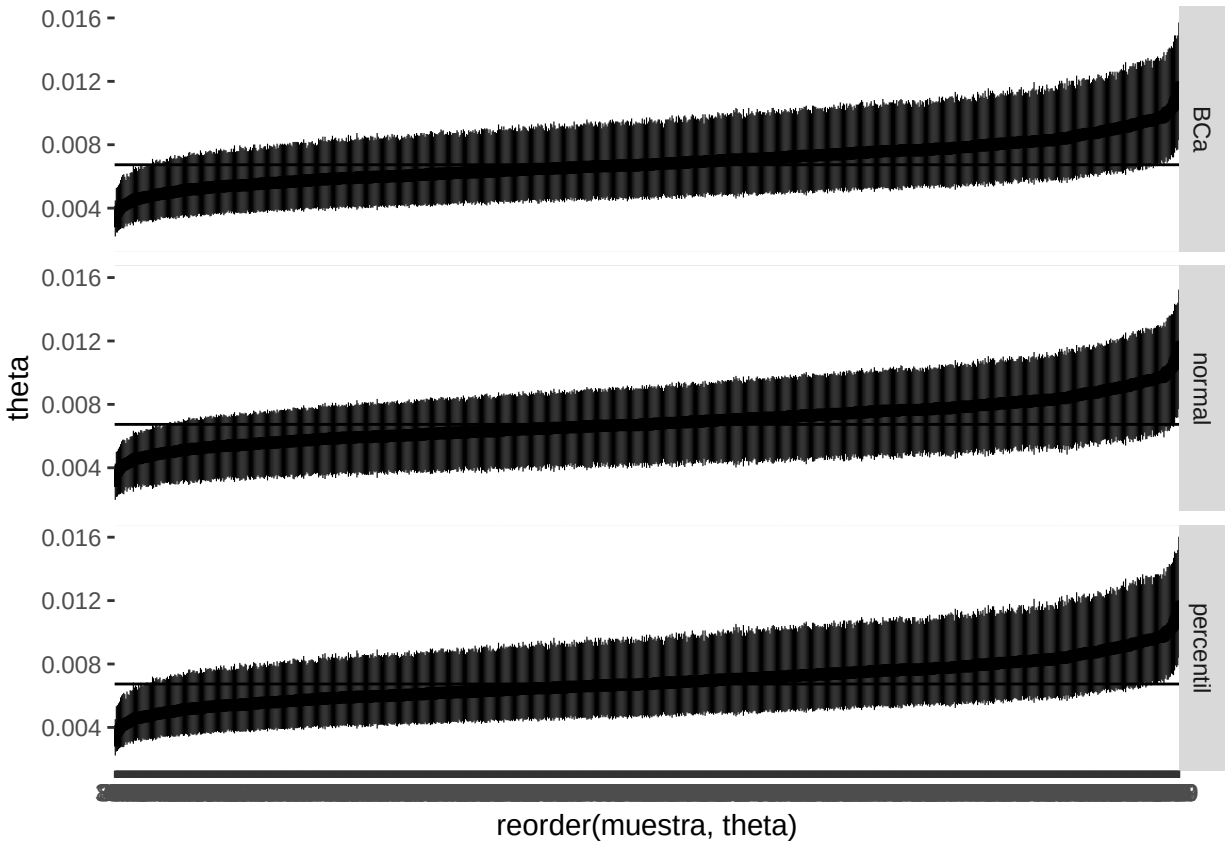
Así obtenemos la siguiente tabla:

```
poiss_rep_int_300 %>%
  group_by(metodo) %>%
  summarise(P_fallo_izquierda = sum(fallo_izquierda)/n(),
            P_fallo_derecha = sum(fallo_derecha)/n(),
            Cobertura = 1 - P_fallo_izquierda - P_fallo_derecha,
            Longitud_promedio = mean(Longitud))%>%
  kable()
```

metodo	P_fallo_izquierda	P_fallo_derecha	Cobertura	Longitud_promedio
BCa	0.020	0.030	0.950	0.0049177
normal	0.005	0.050	0.945	0.0049909
percentil	0.026	0.027	0.947	0.0049767

Y la siguiente gráfica:

```
ggplot(poiss_rep_int_300) +
  geom_pointrange(aes(x = reorder(muestra,theta),
                      ymin = inferior,
                      y=theta,
                      ymax = superior),
                  size=0.2) +
  geom_hline(yintercept = exp(-2.5*2)) +
  facet_grid(metodo~.)
```



Con las gráficas se ve claramente el fenómeno de la expansión y la aceleración porque la cobertura al principio esta por debajo del parámetro y conforme se van haciendo las replicaciones poco a poco se va teniendo mayor cobertura del parámetro.El método de intervalos que cubre mas es el método normal aunque al inicio tiene un porcentaje de fallo similar al de los demás métodos.

4. Cobertura en la práctica

En el caso del conteo rápido es posible evaluar la cobertura del intervalo de confianza bootstrap usando los resultados de elecciones pasadas, para ello usaremos los resultados de las elecciones 2006 (datos `election_2006` del paquete `estcomp`) repetirás los siguientes dos pasos 100 veces (asegurate de que tu ejercicio de simulación sea replicable):

1. Selecciona una muestra estratificada de `election_2006` usando los tamaños de muestra que indica la tabla `strata_sample_2006` (donde `n` era el tamaño de muestra por estrato y `N` es el número de casillas en el mismo).

Para elegir la muestra se define la siguiente funcion:

```
muestra_2006 <- function(df=election_2006){
  df %>%
    select(stratum,pri_pvem,pan,panal,prd_pt_conv,psd,otros) %>%
    left_join(strata_sample_2006) %>%
    split(.$stratum) %>%
    map_df(~sample_n(., size=first(.$n)))
}
```

2. Utiliza estimador de razón y bootstrap para construir intervalos de confianza para todos los candidatos.

Primer construimos una función para obtener el estimador de razón:

```
estimador_razon <- function(df){
  df %>%
    pivot_longer(pri_pvem:otros, names_to = 'partido', values_to = 'votos') %>%
    mutate(v_total = N*votos/n, total = sum(v_total)) %>%
    group_by(partido) %>%
    summarise(p=sum(v_total)/mean(total)) %>%
    pivot_wider(names_from = partido, values_from = p)
}
```

Ayudándonos de esta función, se construye una fución para obtener el estimador bootstrap:

```
elecciones_boot <- function(df=muestra){
  muestra_boot <- df %>%
    group_by(stratum) %>%
    sample_frac(size = 1, replace = TRUE) %>%
    ungroup()
    estimador_razon(muestra_boot)
}
```

Asi, podemos obtener una distribucion bootstrap

```
muestra1 <- muestra_2006()
elecciones_muestras_boot <- rerun(1000,elecciones_boot(muestra1)) %>% bind_rows()
```

Y con esto el intervalo de confianza:

```
elecciones_pi <- estimador_razon(muestra1)
elecciones_es <- map_dbl(elecciones_muestras_boot,sd)
rbind(razon = elecciones_pi) %>%
  rbind(inferior = elecciones_pi + elecciones_es*qnorm(0.025)) %>%
  rbind(superior =elecciones_pi + elecciones_es*qnorm(0.975)) %>%
  t()
```

```
##                  razon    inferior   superior
## otros       0.028630387 0.028219566 0.02904121
## pan         0.358575842 0.355941574 0.36121011
## panal       0.009958717 0.009630307 0.01028713
## prd_pt_conv 0.352538506 0.350208144 0.35486887
## pri_pvem    0.223109441 0.221150183 0.22506870
## psd         0.027187107 0.026838376 0.02753584
```

Evalúa la cobertura del intervalo para cada candidato a lo largo de las 100 muestras, presenta los resultados en una tabla que incluya la longitud media de los intervalos y la cobertura observada.

Para hacer las 100 repeticiones, definimos la funcion siguiente:

```
elecciones_boot_rep <- function(df=election_2006){
  muestra <- muestra_2006(df)
  razon_pi <- estimador_razon(muestra)
  muestras_boot <- rerun(1000,elecciones_boot(muestra)) %>% bind_rows()
  error <- map_dbl(muestras_boot,sd)

  rbind(razon=razon_pi) %>%
    rbind(inferior = razon_pi+error*qnorm(0.025)) %>%
    rbind(superior = razon_pi+error*qnorm(0.975)) %>%
    t() %>%
    as.data.frame() %>%
    rownames_to_column('partido')

}
```

Con esta funcion corremos las 100 repeticiones:

```
elecciones_repeticiones <- rerun(100, elecciones_boot_rep()) %>%
  bind_rows(.id = 'repeticion')
```

Los resultados se presentan en la siguiente tabla:

```
elecciones_repeticiones %>%
mutate(fallo_izquierda = razon<inferior,
       fallo_derecha = superior<razon,
       longitud = superior-inferior) %>%
group_by(partido) %>%
summarise(fallo_izquierda = sum(fallo_izquierda)/n(),
          fallo_derecha = sum(fallo_derecha)/n(),
          cobertura = 1 - fallo_izquierda - fallo_derecha,
          longitud = mean(longitud)) %>%
  kable(round = 4)
```

partido	fallo_izquierda	fallo_derecha	cobertura	longitud
otros	0	0	1	0.0010367
pan	0	0	1	0.0052655
panal	0	0	1	0.0005689
prd_pt_conv	0	0	1	0.0046776
pri_pvem	0	0	1	0.0039224
psd	0	0	1	0.0006787

Opicional (punto extra): Las muestras con las que se estima en el conteo rápido nunca llegan completas, y los faltantes suelen presentar patrones, por ejemplo, las casillas en las zonas rurales tienen mayor probabilidad de no llegar. Repite el ejercicio de simulación de arriba añadiendo un paso de casillas faltantes, lo que debes hacer es que una vez simulada una muestra *completa* cada casilla se censura de acuerdo a cierta probabilidad (tu la eliges como desees), y esta probabilidad puede depender, por ejemplo, de si la casilla es rural o

urbana o quizá puede variar por estado. Elige uno (o más) procedimiento(s) de censura de casillas y evalúa la cobertura de los intervalos en este(os) escenario(s). Puedes explorar las variables disponibles viendo la documentación de los datos (`?election_2006`).

5. Simulación de variables aleatorias

Recuerda que una variable aleatoria X tiene una distribución geométrica con parámetro p si

$$p_X(i) = P(X = i) = pq^{i-1}$$

para $i = 1, 2, ...$ y donde $q = 1 - p$.

Notemos que

$$\begin{aligned}\sum_{i=1}^{j-1} P(X = i) &= 1 - P(X \geq j - 1) \\ &= 1 - q^{j-1}\end{aligned}$$

para $j \geq 1$. por lo que podemos generar un valor de X generando un número aleatorio U y seleccionando j tal que

$$1 - q^{j-1} \leq U \leq 1 - q^j$$

Esto es, podemos definir X como:

$$X = min\{j : (1 - p)^j < 1 - U\}$$

usando que el logaritmo es una función monótona (i.e. $a < b$ implica $log(a) < log(b)$) obtenemos que podemos expresar X como

$$\begin{aligned}X &= min\big\{j : j \cdot log(q) < log(1 - U)\big\} \\ &= min\big\{j : j > log(U)/log(q)\big\}\end{aligned}$$

entonces

$$X = int\left(\frac{log(U)}{log(q)}\right) + 1$$

es geométrica con parámetro p.

Ahora, sea X el número de lanzamientos de una moneda que se requieren para alcanzar r éxitos (soles) cuando cada lanzamiento es independiente, X tiene una distribución binomial negativa.

Una variable aleatoria X tiene distribución binomial negativa con parámetros (r, p) donde r es un entero positivo y $0 < p < r$ si

$$P(X = j) = \frac{(j-1)!}{(j-r)!(r-1)!} p^r (1 - p)^{j-r}.$$

a) Recuerda la distribución geométrica ¿cuál es a relación entre la variable aleatoria binomial negativa y la geométrica?

La distribución geometrica es un caso particular de la binomian negativa cuando $r = 1$, es decir, cuando solo se quiere obtener el número de lanzamientos antes del primer exito.

b) Utiliza el procedimiento descrito para generar observaciones de una variable aleatoria con distribución geométrica y la relación entre la geométrica y la binomial negativa para generar simulaciones de una variable aleatoria con distribución binomial negativa (parámetro p = 0.7, r = 20). Utiliza la semilla 341285 y reporta las primeras 10 simulaciones obtenidas.

Para el caso de la geometrica usamos directamente la fórmula.

```r
set.seed(341285)
sim_geometrica <- function(p, n=1){
  U <- runif(n)
  q <- (1-p)
  as.integer(log(U)/log(q))+1
}
sim_geometrica(0.7,10)
```

```
##  [1] 1 1 1 1 2 2 1 1 2 1
```

Dado que la binomial negativa se puede ver como una serie de geométricas, hasta juntar el número de éxitos k deseados, puede usarse la función definida arriba para simular la binomial negativa. De esta manera tenemos::

```r
sim_binn <- function(p,r){
  sum(sim_geometrica(p,n=r))
}
rerun(10,sim_binn(0.7,20)) %>% flatten_dbl()
```

```
##  [1] 25 28 28 31 29 35 34 30 29 30
```

c) Verifica la relación

$$p_{j+1} = \frac{j(1-p)}{j+1-r} p_j$$

y úsala para generar un nuevo algoritmo de simulación, vuelve a definir la semilla y reporta las primeras 10 simulaciones.

$$\frac{p_{j+1}}{p_j} = \frac{\frac{j!}{(j+1-r)!(r-1)!}p^r(1-p)^{j+1-r}}{\frac{(j-1)!}{(j-r)!(r-1)!}p^r(1-p)^{j-r}} = \frac{j(1-p)}{j+1-r}.$$

Usando esta relación recursiba es posible construir un algoritmo similar al *poisson* visto en clase, el cual se esperaría fera más eficiente al definido usando la relación entre geométrica y binomial negativa.

```r
set.seed(341285)
sim_binn_rec <- function(p,r){
  U <- runif(1)
  i <- r
  f <- p^r
  P <- f
  while(U>=P){
    f <- i*(1-p)*f/(i+1-r)
    P <- P+f
    i <- i+1
  }
  i
}
rerun(10, sim_binn_rec(0.7,20)) %>% flatten_dbl()
```

```
##  [1] 30 29 31 30 26 25 33 27 25 40
```

d) Realiza 10,000 simulaciones usando cada uno de los algoritmos y compara el tiempo de ejecución (puedes usar la función `system.time()`.

```
system.time(rerun(10000, sim_binn(0.7,20)))
```

```
##    user  system elapsed
##    0.07    0.00    0.08
```

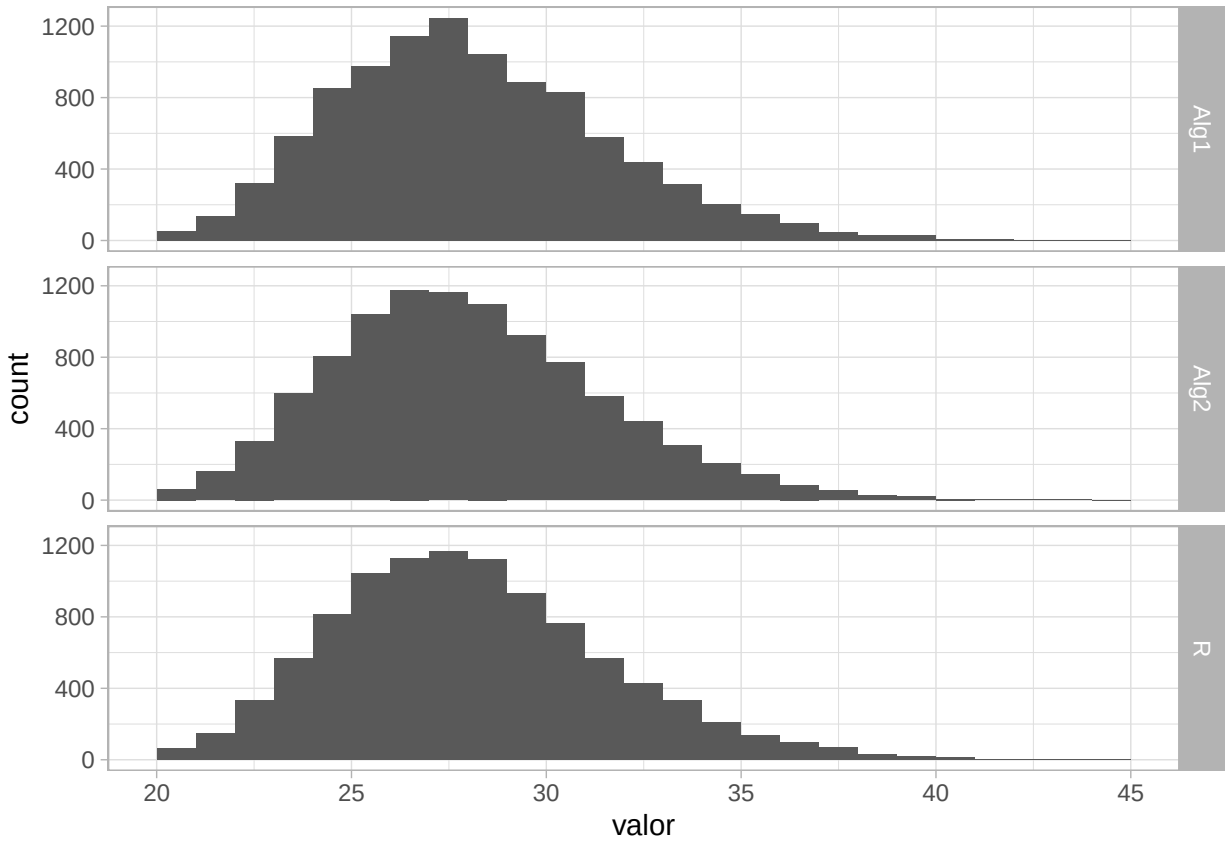
```
system.time(rerun(10000, sim_binn_rec(0.7,20)))
```

```
##    user  system elapsed
##    0.06    0.00    0.07
```

Como se esperaba, en la función recursiva de la binomial el tiempo de ejecución es menor por el while que tiene la función, ya que solo calcula en función al valor de U, sin hacer cálculos innecesarios.

e) Genera un histogrma para cada algoritmo (usa 1000 simulaciones) y comparalo con la distribución construida usando la función de R *dnbinom*.

```
binn_alg1 <- rerun(10000, sim_binn(0.7,20)) %>% flatten_dbl()
binn_alg2 <- rerun(10000, sim_binn_rec(0.7,20))%>% flatten_dbl()
binn_R <- (rnbinom(10000, size = 20, p = 0.7)+20)
binn_hist <- tibble(Alg1 = binn_alg1,
                        Alg2 = binn_alg2,
                        R = binn_R) %>%
  pivot_longer(Alg1:R,names_to = 'modelo', values_to = 'valor')
ggplot(binn_hist) +
  geom_histogram(aes(valor), breaks = 20:45) +
  facet_grid(modelo~.) +
  theme_light()
```



Los tres algormitmos arrogan resultados muy similares, por lo que posiblemente la diferencia sea la eficiencia de cómputo.