Ciencia de Redes Aplicada con R

George G. Vega Yon, Ph.D.

2025-08-23

Table of contents

1 Prefacio

Los métodos estadísticos para sistemas en red están presentes en la mayoría de las disciplinas. A pesar de las diferencias de lenguaje entre áreas, muchos métodos desarrollados para estudiar problemas específicos pueden ser útiles fuera de su contexto original; esta es la premisa de este libro. **Ciencia de Redes Aplicada con R** proporciona ejemplos utilizando el lenguaje de programación R para estudiar sistemas en red. Aunque la mayoría de los casos tratan sobre análisis de redes sociales, los métodos presentados aquí pueden aplicarse a contextos como redes biológicas, redes de transporte y muchos otros.

Todo el libro fue escrito utilizando quarto–un sistema de programación literaria que permite mezclar texto y código–lo que significa que todo el código presentado es 100% ejecutable y, por tanto, reproducible. El código fuente está disponible en GitHub en https://github.com/gvegayon/appliedsnar. Se anima a los lectores a descargar el código y ejecutarlo en sus máquinas utilizando RStudio o VScode.

Además de la programación en R, estaremos utilizando RStudio. Para el manejo de datos, utilizaremos `dplyr` y `data.table`. Los paquetes de manejo y visualización de datos de redes que utilizaremos son `igraph`, netdiffuseR, la suite statnet, y `netplot`.

1.1 Sobre el proyecto

Este proyecto comenzó hace más de seis años como parte de una serie de talleres y tutoriales que impartí en el **Centro de Análisis de Redes Aplicadas** de USC. Hoy, lo uso para recopilar y estudiar métodos estadísticos para analizar redes, con énfasis en sistemas sociales y biológicos. Además, el libro utilizará métodos de computación estadística como componente central.

1.2 Sobre el Autor

Soy Profesor Asistente de Investigación en la **División de Epidemiología de la Universidad de Utah**, donde trabajo estudiando Sistemas Complejos utilizando Computación Estadística. Nací y crecí en Chile. Tengo más de diez años de experiencia desarrollando software científico con enfoque en computación de alto rendimiento, visualización de datos y análisis de redes sociales. Mi formación es en Políticas Públicas (M.A. UAI, 2011), Economía (M.Sc. Caltech, 2015), y Bioestadística (Ph.D. USC, 2020).

Obtuve mi Ph.D. en Bioestadística bajo la supervisión del **Prof. Paul Marjoram** y la **Prof. Kayla de la Haye**, con mi disertación titulada "*Essays on Bioinformatics and Social Network Analysis: Statistical and Computational Methods for Complex Systems.*"

Si desea aprender más sobre mí, por favor visite mi sitio web en https://ggvy.cl.

1.3 Sobre la versión en Español

Esta versión en español del libro fue creada utilizando traducción automática con inteligencia artificial. Aunque se ha hecho un esfuerzo por mantener la precisión técnica y el contexto, algunos términos especializados y conceptos pueden requerir revisión adicional. La versión original en inglés permanece como la referencia autorizada.

Los lectores que encuentren errores de traducción o áreas que requieran clarificación son bienvenidos a contribuir reportando problemas en el repositorio de GitHub del proyecto.

1.4 Divulgación sobre IA

A partir de mediados de 2023, he estado utilizando IA para ayudarme a escribir este libro. Principalmente, uso una combinación de GitHub co-pilot, que ayuda con código y texto, y Grammarly, que ayuda con gramática y estilo. El papel de la IA ha sido ayudarme a escribir más rápido, pero no ha estado involucrada en la conceptualización del libro o el desarrollo de los métodos presentados aquí.

2 Introducción

> ⚠ Nota de Traducción
>
> Esta versión del capítulo fue traducida de manera automática utilizando IA. El capítulo aún no ha sido revisado por un humano.

El Análisis de Redes Sociales y la Ciencia de Redes tienen una larga tradición académica. Desde modelos de difusión social hasta redes de interacción de proteínas, estas disciplinas de sistemas complejos cubren una variedad de problemas a través de campos científicos. Sin embargo, aunque estos podrían verse como ampliamente diferentes, el objeto bajo el microscopio es el mismo: las redes.

Con una larga historia (e insuficiente colaboración interdisciplinaria, si me permiten decir) de avances científicos que ocurren de manera algo aislada, el potencial para la polinización cruzada entre disciplinas dentro de la ciencia de redes es inmenso.

Este libro intenta compilar los muchos métodos disponibles en el ámbito de las ciencias de la complejidad, proporcionar un examen matemático profundo–cuando sea posible–y proporcionar algunos ejemplos que ilustren su uso.

3 Fundamentos de R

> ⚠ Nota de Traducción
>
> Esta versión del capítulo fue traducida de manera automática utilizando IA. El capítulo aún no ha sido revisado por un humano.

R (R Core Team 2024) es un lenguaje de programación orientado a la computación estadística. R se ha convertido en el lenguaje de programación *de facto* en la comunidad de redes sociales debido al gran número de paquetes disponibles para análisis de redes. Los paquetes de R son colecciones de funciones, datos y documentación que extienden R. Un buen libro de referencia tanto para usuarios novatos como avanzados es "The Art of R programming" Matloff (2011)[1].

3.1 Obtener R

Puedes obtener R desde el sitio web de Comprehensive R Archive Network [CRAN] (enlace). CRAN es una red de servidores en todo el mundo que almacenan versiones idénticas y actualizadas de código y documentación para R. El sitio web de CRAN también tiene mucha información sobre R, incluyendo manuales, FAQs y listas de correo.

Aunque R viene con una Interfaz Gráfica de Usuario [GUI], recomiendo obtener una alternativa como RStudio o VSCode. RStudio y VSCode son excelentes compañeros para programar en R. Mientras que RStudio es más común entre los usuarios de R, VSCode es un IDE de propósito más general que puede usarse para muchos otros lenguajes de programación, incluyendo Python y C++.

[1] Aquí una versión pdf gratuita distribuida por el autor.

3.2 Cómo instalar paquetes

Hoy en día, hay dos formas de instalar paquetes de R (que yo conozca), ya sea usando `install.packages`, que es una función que viene con R, o usando el paquete de R `devtools` para instalar un paquete desde algún repositorio remoto que no sea CRAN, aquí hay algunos ejemplos:

```
# Esto instalará el paquete igraph desde CRAN
> install.packages("netdiffuseR")

# ¡Esto instalará la versión más reciente desde el repositorio GitHub del proyecto!
> devtools::install_github("USCCANA/netdiffuseR")
```

El primero, usando `install.packages`, instala la versión de CRAN de `netdiffuseR`, mientras que la línea de código instala cualquier versión que esté publicada en https://github.com/USCCANA/netdiffuseR, que usualmente se llama la versión de desarrollo.

En algunos casos, los usuarios pueden querer/necesitar instalar paquetes desde la línea de comandos ya que algunos paquetes necesitan configuración extra para ser instalados. Pero no necesitaremos ver eso ahora.

3.3 Una Introducción ~~suave~~ Rápida y Sucia a R

Algunas tareas comunes en R

0. Obtener ayuda (y leer el manual) es *LO MÁS IMPORTANTE* que deberías saber. Por ejemplo, si quieres leer el manual (archivo de ayuda) de la función `read.csv`, puedes escribir cualquiera de estos:

   ```
   ?read.csv
   ?"read.csv"
   help(read.csv)
   help("read.csv")
   ```

   Si no estás completamente seguro de cuál es el nombre de la función, siempre puedes usar la *búsqueda difusa*

```r
help.search("linear regression")
??"linear regression"
```

1. En R, puedes crear nuevos objetos usando el operador de asignación (`<-`) o el signo igual `=`, por ejemplo, los siguientes dos son equivalentes: `r a <- 1 a = 1` Históricamente, el operador de asignación es el más comúnmente usado.

2. R tiene varios tipos de objetos. Las estructuras más básicas en R son `vectors`, `matrix`, `list`, `data.frame`. Aquí hay un ejemplo de creación de varios de estos (cada línea está encerrada con paréntesis para que R imprima el elemento resultante):

```r
(a_vector     <- 1:9)
```

```
[1] 1 2 3 4 5 6 7 8 9
```

```r
(another_vect <- c(1, 2, 3, 4, 5, 6, 7, 8, 9))
```

```
[1] 1 2 3 4 5 6 7 8 9
```

```r
(a_string_vec <- c("I", "like", "netdiffuseR"))
```

```
[1] "I"           "like"        "netdiffuseR"
```

```r
(a_matrix     <- matrix(a_vector, ncol = 3))
```

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

```r
# Las matrices también pueden ser de strings
(a_string_mat <- matrix(letters[1:9], ncol=3))
```

```
     [,1] [,2] [,3]
[1,] "a"  "d"  "g"
[2,] "b"  "e"  "h"
[3,] "c"  "f"  "i"
```

```r
# El operador `cbind` hace "column bind"
(another_mat  <- cbind(1:4, 11:14))
```

```
     [,1] [,2]
[1,]    1   11
[2,]    2   12
[3,]    3   13
[4,]    4   14
```

```
# El operador `rbind` hace "row bind"
(another_mat2 <- rbind(1:4, 11:14))
```

```
     [,1] [,2] [,3] [,4]
[1,]    1    2    3    4
[2,]   11   12   13   14
```

```
(a_string_mat <- matrix(letters[1:9], ncol = 3))
```

```
     [,1] [,2] [,3]
[1,] "a"  "d"  "g"
[2,] "b"  "e"  "h"
[3,] "c"  "f"  "i"
```

```
(a_list       <- list(a_vector, a_matrix))
```

```
[[1]]
[1] 1 2 3 4 5 6 7 8 9

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

```
# ¡igual pero con nombres!
(another_list <- list(my_vec = a_vector, my_mat = a_matrix))
```

```
$my_vec
[1] 1 2 3 4 5 6 7 8 9

$my_mat
     [,1] [,2] [,3]
```

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

```
# Los data frames pueden tener múltiples tipos de elementos; es
# una colección de listas
(a_data_frame <- data.frame(x = 1:10, y = letters[1:10]))
```

```
   x y
1  1 a
2  2 b
3  3 c
4  4 d
5  5 e
6  6 f
7  7 g
8  8 h
9  9 i
10 10 j
```

3. Dependiendo del tipo de objeto, podemos acceder a sus componentes usando indexación:

```
# Primeros 3 elementos
a_vector[1:3]
```

```
[1] 1 2 3
```

```
# Tercer elemento
a_string_vec[3]
```

```
[1] "netdiffuseR"
```

```
# Una sub matriz
a_matrix[1:2, 1:2]
```

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

```
# Tercera columna
a_matrix[,3]
```

```
[1] 7 8 9
```

```
# Tercera fila
a_matrix[3,]
```

```
[1] 3 6 9
```

```
# Primeros 6 elementos de la matriz. R almacena matrices
# por columna.
a_string_mat[1:6]
```

```
[1] "a" "b" "c" "d" "e" "f"
```

```
# Estos tres son equivalentes
another_list[[1]]
```

```
[1] 1 2 3 4 5 6 7 8 9
```

```
another_list$my_vec
```

```
[1] 1 2 3 4 5 6 7 8 9
```

```
another_list[["my_vec"]]
```

```
[1] 1 2 3 4 5 6 7 8 9
```

```
# Los data frames son como listas
a_data_frame[[1]]
```

```
 [1]  1  2  3  4  5  6  7  8  9 10
```

```
a_data_frame[,1]
```

```
 [1]  1  2  3  4  5  6  7  8  9 10
```

```
a_data_frame[["x"]]
```

```
 [1]  1  2  3  4  5  6  7  8  9 10
```

```r
a_data_frame$x
```

```
 [1]  1  2  3  4  5  6  7  8  9 10
```

4. Declaraciones de flujo de control

```r
# El bucle for de toda la vida
for (i in 1:10) {
  print(paste("Estoy en el paso", i, "/", 10))
}
```

```
[1] "Estoy en el paso 1 / 10"
[1] "Estoy en el paso 2 / 10"
[1] "Estoy en el paso 3 / 10"
[1] "Estoy en el paso 4 / 10"
[1] "Estoy en el paso 5 / 10"
[1] "Estoy en el paso 6 / 10"
[1] "Estoy en el paso 7 / 10"
[1] "Estoy en el paso 8 / 10"
[1] "Estoy en el paso 9 / 10"
[1] "Estoy en el paso 10 / 10"
```

```r
# Un buen ifelse

for (i in 1:10) {

  if (i %% 2) # Operando módulo
    print(paste("Estoy en el paso", i, "/", 10, "(y soy impar)"))
  else
    print(paste("Estoy en el paso", i, "/", 10, "(y soy par)"))

}
```

```
[1] "Estoy en el paso 1 / 10 (y soy impar)"
[1] "Estoy en el paso 2 / 10 (y soy par)"
[1] "Estoy en el paso 3 / 10 (y soy impar)"
[1] "Estoy en el paso 4 / 10 (y soy par)"
[1] "Estoy en el paso 5 / 10 (y soy impar)"
```

```
[1] "Estoy en el paso 6 / 10 (y soy par)"
[1] "Estoy en el paso 7 / 10 (y soy impar)"
[1] "Estoy en el paso 8 / 10 (y soy par)"
[1] "Estoy en el paso 9 / 10 (y soy impar)"
[1] "Estoy en el paso 10 / 10 (y soy par)"
```

```r
# Un while
i <- 10
while (i > 0) {
  print(paste("Estoy en el paso", i, "/", 10))
  i <- i - 1
}
```

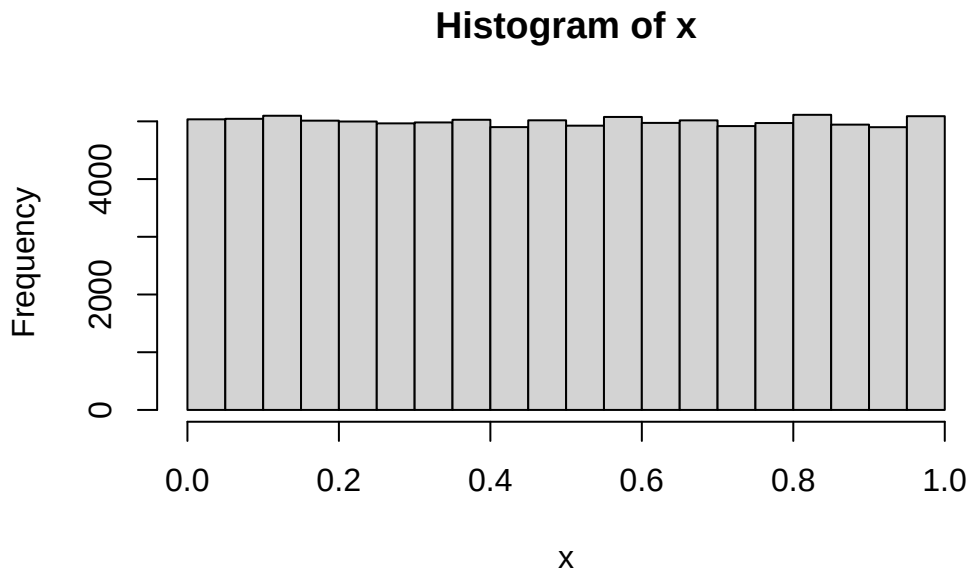
```
[1] "Estoy en el paso 10 / 10"
[1] "Estoy en el paso 9 / 10"
[1] "Estoy en el paso 8 / 10"
[1] "Estoy en el paso 7 / 10"
[1] "Estoy en el paso 6 / 10"
[1] "Estoy en el paso 5 / 10"
[1] "Estoy en el paso 4 / 10"
[1] "Estoy en el paso 3 / 10"
[1] "Estoy en el paso 2 / 10"
[1] "Estoy en el paso 1 / 10"
```

5. R tiene un conjunto convincente de funciones de generación de números pseudo-aleatorios. En general, las funciones de distribución tienen la siguiente estructura de nombres:

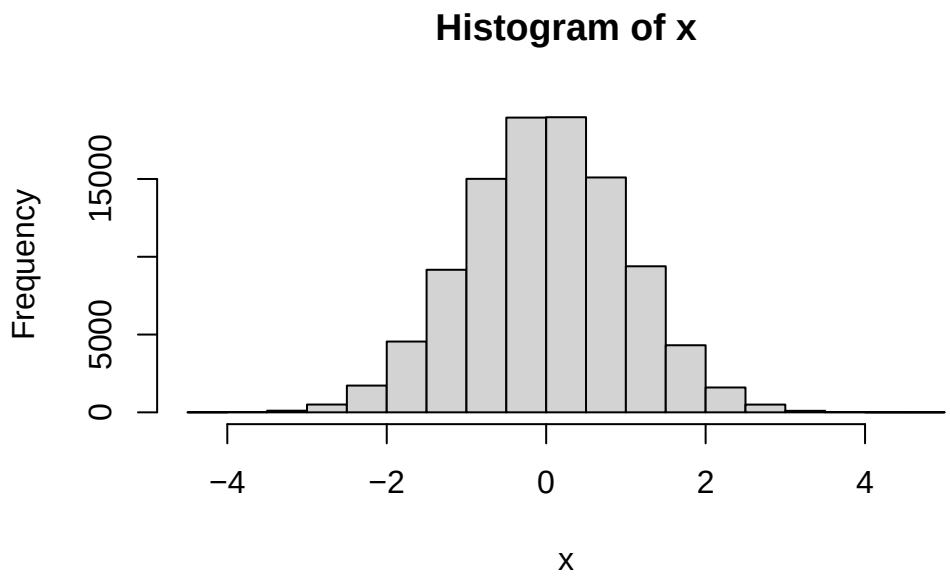
   a. Generación de Números Aleatorios: `r[nombre-de-la-distribución]`, *ej.*, `rnorm` para normal, `runif` para uniforme.
   b. Función de densidad: `d[nombre-de-la-distribución]`, ej. `dnorm` para normal, `dunif` para uniforme.
   c. Función de Distribución Acumulativa (CDF): `p[nombre-de-la-distribución]`, *ej.*, `pnorm` para normal, `punif` para uniforme.
   d. Función inversa (cuantil): `q[nombre-de-la-distribución]`, ej. `qnorm` para la normal, `qunif` para la uniforme.

   Aquí hay algunos ejemplos:

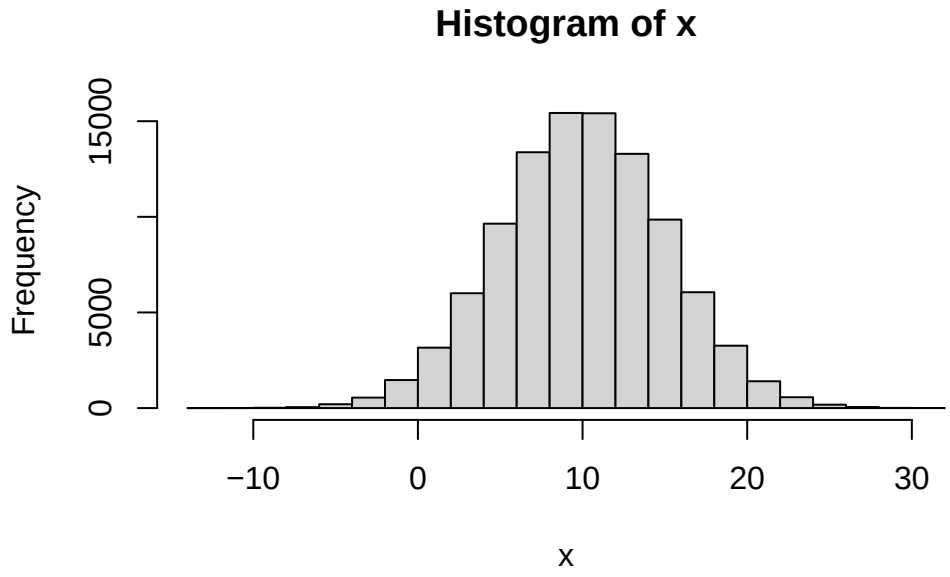
```r
# Para asegurar reproducibilidad
set.seed(1231)

# 100,000 números Unif(0,1)
x <- runif(1e5)
hist(x)
```



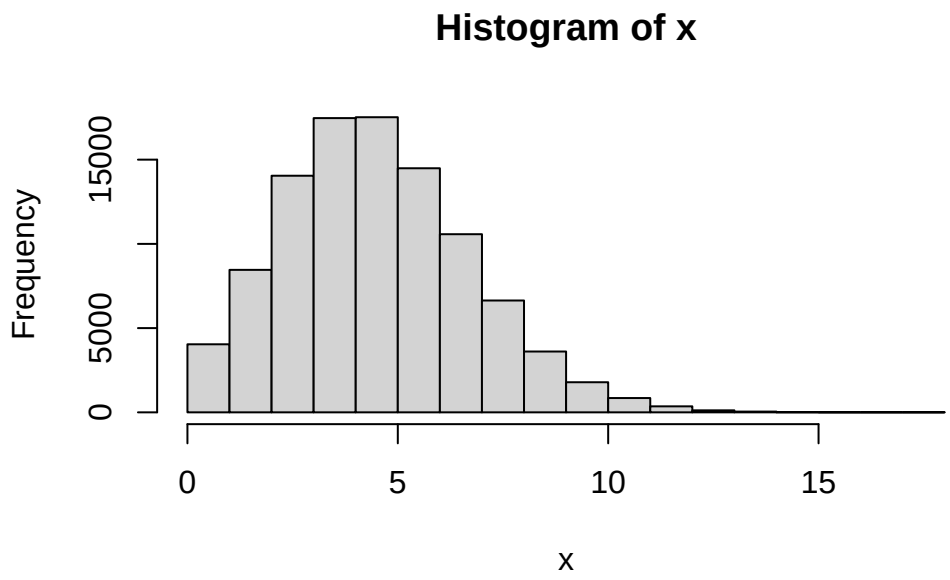

```r
# 100,000 números N(0,1)
x <- rnorm(1e5)
hist(x)
```



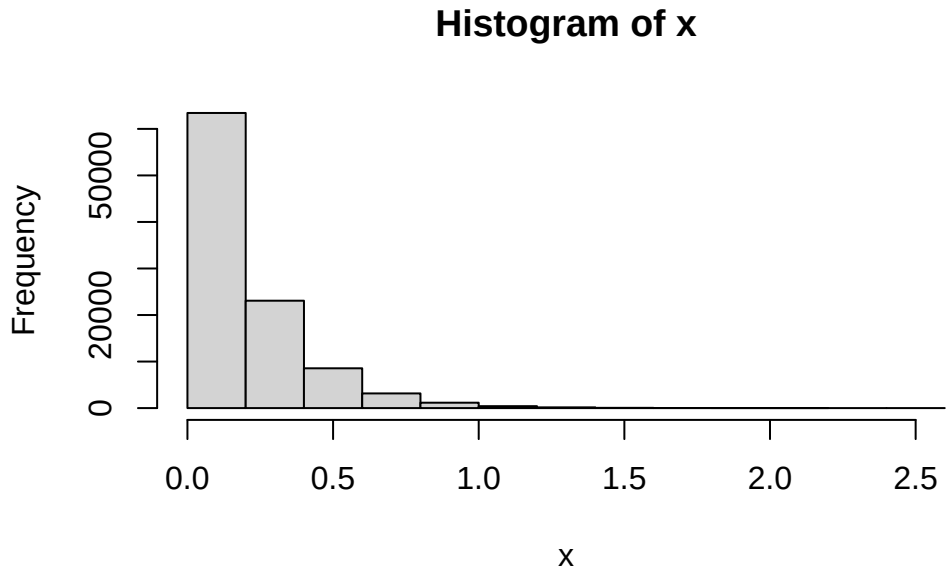

```r
# 100,000 números N(10,25)
x <- rnorm(1e5, mean = 10, sd = 5)
hist(x)
```




```r
# 100,000 números Poisson(5)
x <- rpois(1e5, lambda = 5)
hist(x)
```




```
# 100,000 números rexp(5)
x <- rexp(1e5, 5)
hist(x)
```



Más distribuciones están disponibles en `??Distributions`.

Para una buena introducción a R, echa un vistazo a "The Art of R Programming" por

Norman Matloff. Para usuarios más avanzados, echa un vistazo a "Advanced R" por Hadley Wickham.

Part I

Aplicaciones

4 Redes escolares

> ⚠ Nota de Traducción
>
> Esta versión del capítulo fue traducida de manera automática utilizando IA. El capítulo aún no ha sido revisado por un humano.

Este capítulo proporciona un ejemplo de principio a fin para procesar datos tipo encuesta en R. El capítulo presenta el conjunto de datos del Estudio de Redes Sociales [SNS]. Puedes descargar los datos para este capítulo aquí, y el libro de códigos para los datos proporcionados aquí está en el apéndice.

Los objetivos para este capítulo son:

1. Leer los datos en R.
2. Crear una red con ellos.
3. Calcular estadísticas descriptivas.
4. Visualizar la red.

4.1 Preprocesamiento de datos

4.1.1 Leyendo los datos en R

R tiene varias formas de leer datos. Tus datos pueden ser archivos de texto plano como CSV, delimitados por tabulaciones, o especificados por ancho de columna. Para leer datos de texto plano, puedes usar el paquete `readr` (Wickham, Hester, and Bryan 2024). En el caso de archivos binarios, como archivos de Stata, Octave, o SPSS, puedes usar el paquete de R `foreign` (R Core Team 2023). Si tus datos están formateados como hojas de cálculo

de Microsoft, el paquete de R `readxl` (Wickham and Bryan 2023) es la alternativa a usar. En nuestro caso, los datos para esta sesión están en formato Stata:

```
library(foreign)

# Leyendo los datos
dat <- foreign::read.dta("03-sns.dta")

# Echando un vistazo a las primeras 5 columnas y 5 filas de los datos
dat[1:5, 1:10]
```

```
  photoid school hispanic female1 female2 female3 female4 grades1 grades2
1       1    111        1      NA      NA       0       0      NA      NA
2       2    111        1       0      NA      NA       0     3.0      NA
3       7    111        0       1       1       1       1     5.0     4.5
4      13    111        1       1       1       1       1     2.5     2.5
5      14    111        1       1       1       1      NA     3.0     3.5
  grades3
1     3.5
2      NA
3     4.0
4     2.5
5     3.5
```

4.1.2 Creando un id único para cada participante

Debemos crear un `id` único usando la escuela y el `id` de foto. Dado que ambas variables son numéricas, codificar el id es una buena forma de hacer esto. Por ejemplo, los últimos tres números son el `photoid`, y los primeros números son el `id` de la escuela. Para hacer esto, necesitamos tomar en cuenta el rango de las variables:

```
(photo_id_ran <- range(dat$photoid))
```

```
[1]    1 2074
```

Como la variable se extiende hasta 2074, necesitamos establecer las últimas 4 unidades de la variable para almacenar el `photoid`. Usaremos `dplyr` (Wickham et al. 2023) para crear esta variable y la llamaremos `id`:

```
library(dplyr)
```

```
Attaching package: 'dplyr'

The following objects are masked from 'package:stats':

    filter, lag

The following objects are masked from 'package:base':

    intersect, setdiff, setequal, union
```

```
# Creando la variable
dat <- dat |>
  mutate(id = school*10000 + photoid)

# Primeras filas
dat |>
  head() |>
  select(school, photoid, id)
```

```
  school photoid      id
1    111       1 1110001
2    111       2 1110002
3    111       7 1110007
4    111      13 1110013
5    111      14 1110014
6    111      15 1110015
```

¡Vaya, qué pasó en las últimas líneas de código! ¿Qué es ese `|>`? Bueno, ese es el operador pipe[1], y es una forma atractiva de escribir llamadas a funciones anidadas. En este caso, en lugar de escribir algo como:

```
dat_filtered$id <- dat_filtered$school*10000 + dat_filtered$photoid
subset(head(dat_filtered), select = c(school, photoid, id))
```

4.2 Creando una red

- Queremos construir una red social. Para eso, usamos una matriz de adyacencia o una lista de enlaces.
- Cada individuo de los datos SNS nominó 19 amigos de la escuela. Usaremos esas nominaciones para crear la red social.
- En este caso, crearemos la red coercionando el conjunto de datos en una lista de enlaces.

4.2.1 De encuesta a lista de enlaces

Comencemos cargando un par de paquetes útiles de R. Cargaremos `tidyr` (Wickham, Vaughan, and Girlich 2024) y `stringr` (Wickham 2023). Usaremos el primero, `tidyr`, para remodelar los datos. El segundo, `stringr`, nos ayudará a procesar cadenas usando *expresiones regulares*[2].

```
library(tidyr)
library(stringr)
```

Opcionalmente, podemos usar el tipo de objeto `tibble`, una alternativa al `data.frame` actual. Este objeto proporciona *métodos más eficientes para matrices y marcos de datos.*

[1] Introducido en R versión 4.1.0, el operador pipe de R base `|>` funciona de manera similar al pipe de `magrittr` `%>%`. Las diferencias clave entre estos dos se explican en https://www.tidyverse.org/blog/2023/04/base-vs-magrittr-pipe/.

[2] Por favor, consulta el archivo de ayuda `?'regular expression'` en R. El paquete de R `rex` (Ushey, Hester, and Krzyzanowski 2021) es un compañero amigable para escribir expresiones regulares. También hay un complemento de RStudio ordenado (pero experimental) que puede ser muy útil para entender cómo funcionan las expresiones regulares, el complemento regexplain.

```r
dat <- as_tibble(dat)
```

Lo que me gusta de los `tibbles` es que cuando los imprimes en la consola, estos se ven bien:

```r
dat
```

```
# A tibble: 2,164 x 100
   photoid school hispanic female1 female2 female3 female4 grades1 grades2
     <int>  <int>    <dbl>   <int>   <int>   <int>   <int>   <dbl>   <dbl>
 1       1    111        1      NA      NA       0       0    NA      NA
 2       2    111        1       0      NA      NA       0     3      NA
 3       7    111        0       1       1       1       1     5       4.5
 4      13    111        1       1       1       1       1     2.5     2.5
 5      14    111        1       1       1       1      NA     3       3.5
 6      15    111        1       0       0       0       0     2.5     2.5
 7      20    111        1       1       1       1       1     2.5     2.5
 8      22    111        1      NA      NA       0       0    NA      NA
 9      25    111        0       1       1      NA       1     4.5     3.5
10      27    111        1       0      NA       0       0     3.5    NA
# i 2,154 more rows
# i 91 more variables: grades3 <dbl>, grades4 <dbl>, eversmk1 <int>,
#   eversmk2 <int>, eversmk3 <int>, eversmk4 <int>, everdrk1 <int>,
#   everdrk2 <int>, everdrk3 <int>, everdrk4 <int>, home1 <int>, home2 <int>,
#   home3 <int>, home4 <int>, sch_friend11 <int>, sch_friend12 <int>,
#   sch_friend13 <int>, sch_friend14 <int>, sch_friend15 <int>,
#   sch_friend16 <int>, sch_friend17 <int>, sch_friend18 <int>, ...
```

```r
# Tal vez demasiados pipes... ¡pero es genial!
net <- dat |>
  select(id, school, starts_with("sch_friend")) |>
  gather(key = "varname", value = "content", -id, -school) |>
  filter(!is.na(content)) |>
  mutate(
    friendid = school*10000 + content,
```

```
    year     = as.integer(str_extract(varname, "(?<=[a-z])[0-9]")),
    nnom     = as.integer(str_extract(varname, "(?<=[a-z][0-9])[0-9]+"))
  )
```

Veamos esto paso a paso:

1. Primero, subconjuntamos los datos: Queremos mantener `id, school, sch_friend*`. Para este último, usamos la función `starts_with` (del paquete `tidyselect`). Este último nos permite seleccionar todas las variables que comienzan con la palabra "`sch_friend`", lo que significa que `sch_friend11, sch_friend12, ...` serán seleccionadas.

```
dat |>
  select(id, school, starts_with("sch_friend"))
```

```
# A tibble: 2,164 x 78
        id school sch_friend11 sch_friend12 sch_friend13 sch_friend14
     <dbl>  <int>        <int>        <int>        <int>        <int>
 1 1110001    111           NA           NA           NA           NA
 2 1110002    111          424          423          426          289
 3 1110007    111          629          505           NA           NA
 4 1110013    111          232          569           NA           NA
 5 1110014    111          582          134           41          592
 6 1110015    111           26          488           81          138
 7 1110020    111          528           NA          492          395
 8 1110022    111           NA           NA           NA           NA
 9 1110025    111          135          185          553           84
10 1110027    111          346          168          559            5
# i 2,154 more rows
# i 72 more variables: sch_friend15 <int>, sch_friend16 <int>,
#   sch_friend17 <int>, sch_friend18 <int>, sch_friend19 <int>,
#   sch_friend110 <int>, sch_friend111 <int>, sch_friend112 <int>,
#   sch_friend113 <int>, sch_friend114 <int>, sch_friend115 <int>,
#   sch_friend116 <int>, sch_friend117 <int>, sch_friend118 <int>,
#   sch_friend119 <int>, sch_friend21 <int>, sch_friend22 <int>, ...
```

2. Luego, lo remodelamos a formato *largo*: Transponiendo todos los `sch_friend*` a formato largo. Hacemos esto usando la función `gather` (del paquete `tidyr`); una alternativa a la función `reshape`, que encuentro más fácil de usar. Veamos cómo funciona:

```
dat |>
  select(id, school, starts_with("sch_friend")) |>
  gather(key = "varname", value = "content", -id, -school)
```

```
# A tibble: 164,464 x 4
        id school varname      content
     <dbl>  <int> <chr>          <int>
 1 1110001    111 sch_friend11      NA
 2 1110002    111 sch_friend11     424
 3 1110007    111 sch_friend11     629
 4 1110013    111 sch_friend11     232
 5 1110014    111 sch_friend11     582
 6 1110015    111 sch_friend11      26
 7 1110020    111 sch_friend11     528
 8 1110022    111 sch_friend11      NA
 9 1110025    111 sch_friend11     135
10 1110027    111 sch_friend11     346
# i 164,454 more rows
```

   En este caso, el parámetro `key` establece el nombre de la variable que contendrá el nombre de la variable que fue remodelada, mientras que `value` es el nombre de la variable que contendrá el contenido de los datos (por eso los nombré así). El bit `-id, -school` le dice a la función que "elimine" esas variables antes de remodelar. En otras palabras, "remodela todo excepto `id` y `school`."

   También, nota que pasamos de 2164 filas a 19 (nominaciones) * 2164 (sujetos) * 4 (ondas) = 164464 filas, como se esperaba.

3. Como los datos de nominación pueden estar vacíos para algunas celdas, necesitamos cuidar esos casos, los `NA`s, así que filtramos los datos:

```
dat |>
  select(id, school, starts_with("sch_friend")) |>
  gather(key = "varname", value = "content", -id, -school) |>
  filter(!is.na(content))
```

```
# A tibble: 39,561 x 4
        id school varname      content
     <dbl>  <int> <chr>          <int>
 1 1110002    111 sch_friend11     424
 2 1110007    111 sch_friend11     629
 3 1110013    111 sch_friend11     232
 4 1110014    111 sch_friend11     582
 5 1110015    111 sch_friend11      26
 6 1110020    111 sch_friend11     528
 7 1110025    111 sch_friend11     135
 8 1110027    111 sch_friend11     346
 9 1110029    111 sch_friend11     369
10 1110030    111 sch_friend11     462
# i 39,551 more rows
```

4. Y finalmente, creamos tres nuevas variables de este conjunto de datos: `friendid`,, `year`, y `nom_num` (número de nominación). Todo usando expresiones regulares:

```
dat |>
  select(id, school, starts_with("sch_friend")) |>
  gather(key = "varname", value = "content", -id, -school) |>
  filter(!is.na(content)) |>
  mutate(
    friendid = school*10000 + content,
    year     = as.integer(str_extract(varname, "(?<=[a-z])[0-9]")),
    nnom     = as.integer(str_extract(varname, "(?<=[a-z][0-9])[0-9]+"))
    )
```

```
# A tibble: 39,561 x 7
        id school varname      content friendid  year  nnom
     <dbl>  <int> <chr>          <int>    <dbl> <int> <int>
 1 1110002    111 sch_friend11     424  1110424     1     1
 2 1110007    111 sch_friend11     629  1110629     1     1
 3 1110013    111 sch_friend11     232  1110232     1     1
 4 1110014    111 sch_friend11     582  1110582     1     1
 5 1110015    111 sch_friend11      26  1110026     1     1
 6 1110020    111 sch_friend11     528  1110528     1     1
```

```
 7 1110025    111 sch_friend11     135  1110135     1     1
 8 1110027    111 sch_friend11     346  1110346     1     1
 9 1110029    111 sch_friend11     369  1110369     1     1
10 1110030    111 sch_friend11     462  1110462     1     1
# i 39,551 more rows
```

La expresión regular `(?<=[a-z])` coincide con una cadena precedida por cualquier letra de *a* a *z*. En contraste, la expresión `[0-9]` coincide con un solo número. Por lo tanto, de la cadena `"sch_friend12"`, la expresión regular solo coincidirá con el `1`, ya que es el único número seguido por una letra. La expresión `(?<=[a-z][0-9])` coincide con una cadena precedida por una letra minúscula y un número de un dígito. Finalmente, la expresión `[0-9]+` coincide con una cadena de números–así que podría ser más de uno. Por lo tanto, de la cadena `"sch_friend12"`, obtendremos `2`:

```
str_extract("sch_friend12", "(?<=[a-z])[0-9]")
```

```
[1] "1"
```

```
str_extract("sch_friend12", "(?<=[a-z][0-9])[0-9]+")
```

```
[1] "2"
```

Y finalmente, la función `as.integer` coerciona el valor de retorno de la función `str_extract` de `character` a `integer`. Ahora que tenemos esta lista de enlaces, podemos crear un objeto igraph

4.2.2 Red igraph

Para coercionar la lista de enlaces en un objeto igraph, usaremos la función `graph_from_data_frame` en igraph (Csárdi et al. 2024). Esta función recibe los siguientes argumentos: un marco de datos donde las dos primeras columnas son "source" (ego) y "target" (alter), un indicador de si la red es dirigida o no, y un marco de datos opcional con vértices, en cuya primera columna debería contener los ids de vértice.

Usar el argumento opcional `vertices` es una buena práctica: Le dice a la función qué `ids` debería esperar. Usando el conjunto de datos original, crearemos un marco de datos con vértices de nombre:

```r
vertex_attrs <- dat |>
  select(id, school, hispanic, female1, starts_with("eversmk"))
```

Ahora, usemos la función `graph_from_data_frame` para crear un objeto `igraph`:

```r
library(igraph)

ig_year1 <- net |>
  filter(year == "1") |>
  select(id, friendid, nnom) |>
  graph_from_data_frame(
    vertices = vertex_attrs
  )
```

```
Error in graph_from_data_frame(select(filter(net, year == "1"), id, friendid, : Some ver
```

¡Ups! Parece que los individuos están nominando a otros estudiantes no incluidos en la encuesta. ¿Cómo resolver eso? Bueno, ¡todo depende de lo que necesites hacer! En este caso, iremos por la estrategia de *elimínalos-silenciosamente-y-no-digas-nada*:

```r
library(igraph)

ig_year1 <- net |>
  filter(year == "1") |>

  # Línea extra, todas las nominaciones deben estar en ego también.
  filter(friendid %in% id) |>

  select(id, friendid, nnom) |>
  graph_from_data_frame(
    vertices = vertex_attrs
    )

ig_year1
```

```
IGRAPH dc7b2f5 DN-- 2164 9514 --
+ attr: name (v/c), school (v/n), hispanic (v/n), female1 (v/n),
| eversmk1 (v/n), eversmk2 (v/n), eversmk3 (v/n), eversmk4 (v/n), nnom
| (e/n)
+ edges from dc7b2f5 (vertex names):
 [1] 1110007->1110629 1110013->1110232 1110014->1110582 1110015->1110026
 [5] 1110025->1110135 1110027->1110346 1110029->1110369 1110035->1110034
 [9] 1110040->1110390 1110041->1110557 1110044->1110027 1110046->1110030
[13] 1110050->1110086 1110057->1110263 1110069->1110544 1110071->1110167
[17] 1110072->1110289 1110073->1110014 1110075->1110352 1110084->1110305
[21] 1110086->1110206 1110093->1110040 1110094->1110483 1110095->1110043
+ ... omitted several edges
```

Así que tenemos nuestra red con 2164 nodos y 9514 enlaces. Los siguientes pasos: obtener algunas estadísticas descriptivas y visualizar nuestra red.

4.3 Estadísticas descriptivas de red

Aunque podríamos hacer todas las redes a la vez, en esta parte, nos enfocaremos en calcular algunas estadísticas de red para una sola escuela. Comenzamos por la escuela 111. La primera pregunta que deberías estar haciéndote ahora es, "¿cómo puedo obtener esa información del objeto igraph?." Los atributos de vértices y enlaces se pueden acceder a través de las funciones `V` y `E`, respectivamente; además, podemos listar qué atributos de vértice/enlace están disponibles:

```
vertex_attr_names(ig_year1)
```

```
[1] "name"     "school"   "hispanic" "female1"  "eversmk1" "eversmk2" "eversmk3"
[8] "eversmk4"
```

```
edge_attr_names(ig_year1)
```

```
[1] "nnom"
```

Tal como haríamos con marcos de datos, acceder a atributos de vértice se hace a través del operador signo de dólar `$`. Junto con la función `V`; por ejemplo, acceder a los primeros diez elementos de la variable `hispanic` se puede hacer de la siguiente manera:

```
V(ig_year1)$hispanic[1:10]
```

```
 [1] 1 1 0 1 1 1 1 1 0 1
```

Ahora que sabes cómo acceder a atributos de vértice, podemos obtener la red correspondiente a la escuela 111 identificando qué vértices son parte de ella y pasar esa información a la función `induced_subgraph`:

```
# ¿Qué ids son de la escuela 111?
school111ids <- which(V(ig_year1)$school == 111)

# Creando un subgrafo
ig_year1_111 <- induced_subgraph(
  graph = ig_year1,
  vids  = school111ids
)
```

La función `which` en R devuelve un vector de índices indicando qué elementos pasan la prueba, devolviendo verdadero y falso, de lo contrario. En nuestro caso, resultará en un vector de índices de los vértices que tienen el atributo `school` igual a 111. Con el subgrafo, podemos calcular diferentes medidas de centralidad[3] para cada vértice y almacenarlas en el objeto igraph mismo:

```
# Calculando medidas de centralidad para cada vértice
V(ig_year1_111)$indegree   <- degree(ig_year1_111, mode = "in")
V(ig_year1_111)$outdegree  <- degree(ig_year1_111, mode = "out")
V(ig_year1_111)$closeness  <- closeness(ig_year1_111, mode = "total")
V(ig_year1_111)$betweeness <- betweenness(ig_year1_111, normalized = TRUE)
```

[3]Para más información sobre las diferentes medidas de centralidad, por favor echa un vistazo al artículo "Centrality" en Wikipedia.

Desde aquí, podemos *volver* a nuestros viejos hábitos y obtener el conjunto de atributos de vértice como un marco de datos para que podamos calcular algunas estadísticas de resumen sobre las medidas de centralidad que acabamos de obtener

```
# Extrayendo cada característica de vértice como un data.frame
stats <- as_data_frame(ig_year1_111, what = "vertices")

# Calculando cuantiles para cada variable
stats_degree <- with(stats, {
 cbind(
   indegree   = quantile(indegree, c(.025, .5, .975), na.rm = TRUE),
   outdegree  = quantile(outdegree, c(.025, .5, .975), na.rm = TRUE),
   closeness  = quantile(closeness, c(.025, .5, .975), na.rm = TRUE),
   betweeness = quantile(betweeness, c(.025, .5, .975), na.rm = TRUE)
 )
})

stats_degree
```

```
      indegree outdegree    closeness  betweeness
2.5%         0         0 0.0005915148 0.000000000
50%          4         4 0.0007487833 0.001879006
97.5%       16        16 0.0008838413 0.016591048
```

La función `with` es algo similar a lo que `dplyr` nos permite hacer cuando queremos trabajar con el conjunto de datos pero sin mencionar su nombre cada vez que pedimos una variable. Sin usar la función `with`, lo anterior podría haberse hecho de la siguiente manera:

```
stats_degree <-
 cbind(
   indegree   = quantile(stats$indegree, c(.025, .5, .975), na.rm = TRUE),
   outdegree  = quantile(stats$outdegree, c(.025, .5, .975), na.rm = TRUE),
   closeness  = quantile(stats$closeness, c(.025, .5, .975), na.rm = TRUE),
   betweeness = quantile(stats$betweeness, c(.025, .5, .975), na.rm = TRUE)
 )
```

A continuación, calcularemos algunas estadísticas a nivel de grafo:

```
cbind(
  size    = vcount(ig_year1_111),
  nedges  = ecount(ig_year1_111),
  density = edge_density(ig_year1_111),
  recip   = reciprocity(ig_year1_111),
  centr   = centr_betw(ig_year1_111)$centralization,
  pathLen = mean_distance(ig_year1_111)
  )
```

```
     size nedges     density     recip      centr pathLen
[1,]  533   2638 0.009303277 0.3731513 0.02179154 4.23678
```

Censo triádico

```
triadic <- triad_census(ig_year1_111)
triadic
```

```
 [1] 24059676   724389   290849     3619     3383     4401     3219     2997
 [9]      407       33      836      235      163      137      277       85
```

Para obtener una vista más agradable de esto, podemos usar una tabla que recuperé de `?triad_census`. Además, podemos normalizar el objeto `triadic` por su suma en lugar de mirar conteos en bruto. De esa manera, obtenemos proporciones en su lugar[4]

```
knitr::kable(cbind(
  Pcent = triadic/sum(triadic)*100,
  read.csv("triadic_census.csv")
  ), digits = 2)
```

[4]Durante nuestro taller, la Prof. De la Haye sugirió usar $\binom{n}{3}$ como una constante normalizadora. ¡Resulta que `sum(triadic) = choose(n, 3)`! Así que cualquier enfoque es correcto.

Pcent	code	description
95.88	003	A,B,C, the empty graph.
2.89	012	A->B, C, the graph with a single directed edge.
1.16	102	A<->B, C, the graph with a mutual connection between two vertices.
0.01	021D	A<-B->C, the out-star.
0.01	021U	A->B<-C, the in-star.
0.02	021C	A->B->C, directed line.
0.01	111D	A<->B<-C.
0.01	111U	A<->B->C.
0.00	030T	A->B<-C, A->C.
0.00	030C	A<-B<-C, A->C.
0.00	201	A<->B<->C.
0.00	120D	A<-B->C, A<->C.
0.00	120U	A->B<-C, A<->C.
0.00	120C	A->B->C, A<->C.
0.00	210	A->B<->C, A<->C.
0.00	300	A<->B<->C, A<->C, the complete graph.

4.4 Graficando la red en igraph

4.4.1 Gráfico único

Echemos un vistazo a cómo se ve nuestra red cuando usamos los parámetros predeterminados en el método plot del objeto igraph:

```
plot(ig_year1)
```

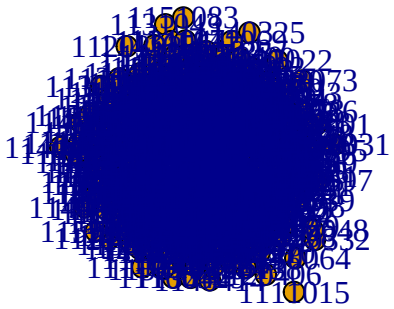


Figure 4.1: Un gráfico de red no muy agradable. Esto es lo que obtenemos con los parámetros predeterminados en igraph.

No muy agradable, ¿verdad? Un par de cosas con este gráfico:

1. Estamos viendo todas las escuelas simultáneamente, lo que no tiene sentido. Así que, en lugar de graficar `ig_year1`, nos enfocaremos en `ig_year1_111`.

2. Todos los vértices tienen el mismo tamaño y se están solapando. En lugar de usar el tamaño predeterminado, dimensionaremos los vértices por indegree usando la función `degree` y pasando el vector de grados a `vertex.size`.[5]

3. Dado el número de vértices en estas redes, las etiquetas no son útiles aquí. Así que las eliminaremos estableciendo `vertex.label = NA`. Además, reduciremos el tamaño de la punta de las flechas estableciendo `edge.arrow.size = 0.25`.

4. Y finalmente, estableceremos el color de cada vértice para que sea una función de si el individuo es hispano o no. Para esta última parte necesitamos ir un poco más de programación:

[5]Descubrir cuál es el tamaño de vértice óptimo es un poco complicado. Sin ponerse demasiado técnico, no hay otra forma de obtener un tamaño de vértice *agradable* que no sea simplemente jugar con diferentes valores de él. Una solución agradable a esto es usar `netdiffuseR::igraph_vertex_rescale` que reescala los vértices para que estos mantengan su relación de aspecto a una proporción predefinida de la pantalla.

```
col_hispanic <- V(ig_year1_111)$hispanic + 1
col_hispanic <- coalesce(col_hispanic, 3)
col_hispanic <- c("steelblue", "tomato", "white")[col_hispanic]
```

Línea por línea, hicimos lo siguiente:

1. La primera línea agregó uno a todos los valores no `NA` para que los 0s (no hispanos) se convirtieran en 1s y los 1s (hispanos) se convirtieran en 2s.

2. La segunda línea reemplazó todos los `NA`s con el número tres para que nuestro vector `col_hispanic` ahora vaya de uno a tres sin `NA`s en él.

3. En la última línea, creamos un vector de colores. Esencialmente, lo que estamos haciendo aquí es decirle a R que cree un vector de longitud `length(col_hispanic)` seleccionando elementos por índice del vector `c("steelblue", "tomato", "white")`. De esta manera, si, por ejemplo, el primer elemento del vector `col_hispanic` fuera un 3, nuestro nuevo vector de colores tendría un `"white"` en él.

Para asegurarnos de que sabemos que estamos en lo correcto, imprimamos los primeros 10 elementos de nuestro nuevo vector de colores junto con la columna original `hispanic`:

```
cbind(
  original = V(ig_year1_111)$hispanic[1:10],
  colors   = col_hispanic[1:10]
  )
```

```
      original colors
 [1,] "1"      "tomato"
 [2,] "1"      "tomato"
 [3,] "0"      "steelblue"
 [4,] "1"      "tomato"
 [5,] "1"      "tomato"
 [6,] "1"      "tomato"
 [7,] "1"      "tomato"
 [8,] "1"      "tomato"
 [9,] "0"      "steelblue"
[10,] "1"      "tomato"
```

Con nuestro agradable vector de colores, ahora podemos pasarlo a `plot.igraph` (que llamamos implícitamente simplemente llamando `plot`), a través del argumento `vertex.color`:

```
# Gráfico elegante
set.seed(1)
plot(
  ig_year1_111,
  vertex.size     = degree(ig_year1_111)/10 +1,
  vertex.label    = NA,
  edge.arrow.size = .25,
  vertex.color    = col_hispanic
  )
```

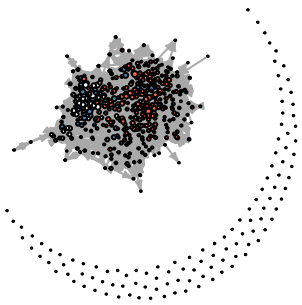


Figure 4.2: Red de amigos en tiempo 1 para la escuela 111.

¡Agradable! Así que se ve mejor. El único problema es que tenemos muchos aislados. Intentemos de nuevo dibujando el mismo gráfico sin aislados. Para hacer eso, necesitamos filtrar el grafo, para lo cual usaremos la función `induced_subgraph`

```r
# ¿Qué vértices no son aislados?
which_ids <- which(degree(ig_year1_111, mode = "total") > 0)

# Obteniendo el subgrafo
ig_year1_111_sub <- induced_subgraph(ig_year1_111, which_ids)

# Necesitamos obtener el mismo subconjunto en col_hispanic
col_hispanic <- col_hispanic[which_ids]
```

```r
# Gráfico elegante
set.seed(1)
plot(
  ig_year1_111_sub,
  vertex.size     = degree(ig_year1_111_sub)/5 +1,
  vertex.label    = NA,
  edge.arrow.size = .25,
  vertex.color    = col_hispanic
  )
```

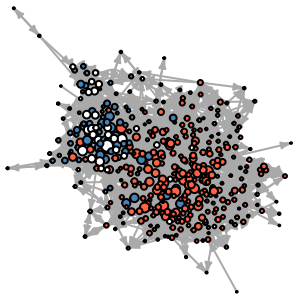


Figure 4.3: Red de amigos en tiempo 1 para la escuela 111. El grafo excluye aislados.

¡Ahora eso está mejor! Un patrón interesante que surge es que los individuos parecen agruparse por si son hispanos o no.

Podemos escribir esto como una función para evitar copiar y pegar el código n veces (suponiendo que queremos crear un gráfico similar a este n veces). Hacemos esto último en la siguiente subsección.

4.4.2 Múltiples gráficos

Cuando te estás repitiendo repetidamente, es una buena idea escribir una secuencia de comandos como una función. En este caso, dado que ejecutaremos el mismo tipo de gráfico para todas las escuelas/ondas, escribimos una función en la que las únicas cosas que cambian son: (a) el id de la escuela, y (b) el color de los nodos.

```
myplot <- function(
  net,
  schoolid,
  mindgr = 1,
  vcol   = "tomato",
  ...) {

  # Creando un subgrafo
  subnet <- induced_subgraph(
    net,
    which(degree(net, mode = "all") >= mindgr & V(net)$school == schoolid)
  )

  # Gráfico elegante
  set.seed(1)
  plot(
    subnet,
    vertex.size     = degree(subnet)/5,
    vertex.label    = NA,
    edge.arrow.size = .25,
    vertex.color    = vcol,
    ...
```

```
    )
}
```

La definición de la función:

1. El `myplot <- function([argumentos]) {[cuerpo de la función]}` le dice a R que vamos a crear una función llamada `myplot`.

2. Declaramos cuatro argumentos específicos: `net`, `schoolid`, `mindgr`, y `vcol`. Estos son un objeto igraph, el id de la escuela, el grado mínimo que los vértices deben tener para ser incluidos en la figura, y el color de los vértices. Observa que, comparado con otros lenguajes de programación, R no requiere declarar los tipos de datos.

3. El objeto de puntos suspensivos, `...`, es un objeto especial en R que nos permite pasar otros argumentos sin especificar cuáles. Si echas un vistazo al bit `plot` en el cuerpo de la función, verás que también agregamos `....` Usamos los puntos suspensivos para pasar argumentos extra (diferentes de los que definimos explícitamente) directamente a `plot`. En la práctica, esto implica que podemos, por ejemplo, establecer el argumento `edge.arrow.size` al llamar `myplot`, ¡incluso aunque no lo incluimos en la definición de la función! (Ver `?dotsMethods` en R para más detalles).

En las siguientes líneas de código, usando nuestra nueva función, graficaremos la red de cada escuela en el mismo dispositivo de graficado (ventana) con la ayuda de la función `par`, y agregaremos leyenda con el `legend`:

```
# Graficando todos juntos
oldpar <- par(no.readonly = TRUE)
par(mfrow = c(2, 3), mai = rep(0, 4), oma= c(1, 0, 0, 0))
myplot(ig_year1, 111, vcol = "tomato")
myplot(ig_year1, 112, vcol = "steelblue")
myplot(ig_year1, 113, vcol = "black")
myplot(ig_year1, 114, vcol = "gold")
myplot(ig_year1, 115, vcol = "white")
par(oldpar)

# Una leyenda elegante
legend(
```

```
  "bottomright",
  legend = c(111, 112, 113, 114, 115),
  pt.bg  = c("tomato", "steelblue", "black", "gold", "white"),
  pch    = 21,
  cex    = 1,
  bty    = "n",
  title  = "Escuela"
  )
```

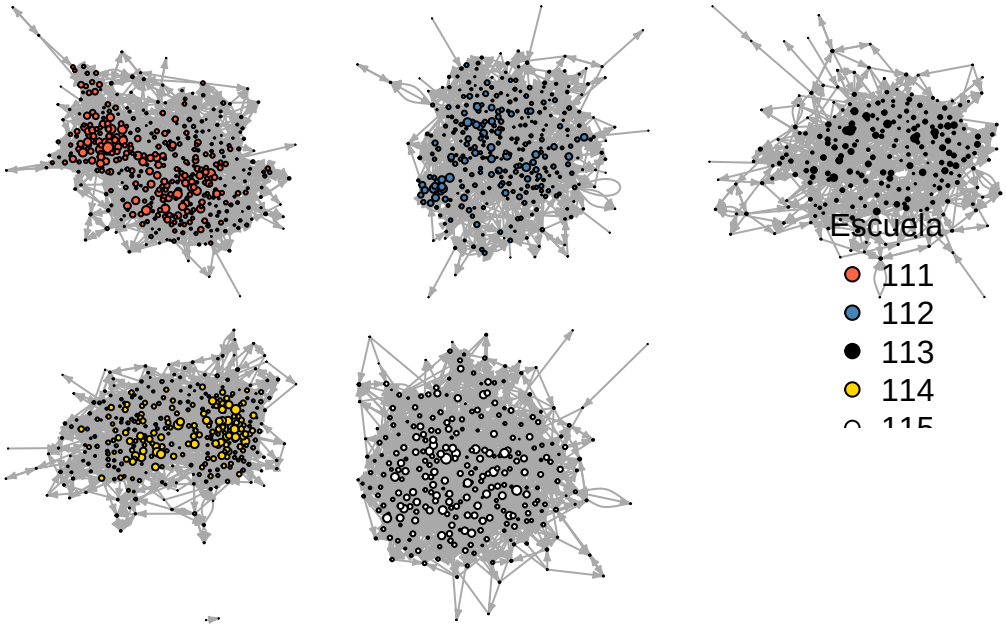


Figure 4.4: Las 5 escuelas en tiempo 1. Nuevamente, los grafos excluyen aislados.

Entonces, ¿qué pasó aquí?

- `oldpar <- par(no.readonly = TRUE)` Esta línea almacena los parámetros actuales para graficar. Dado que vamos a estar cambiándolos, ¡más vale asegurarnos de que podemos volver!.
- `par(mfrow = c(2, 3), mai = rep(0, 4), oma=rep(0, 4))` Aquí estamos estableciendo varias cosas al mismo tiempo. `mfrow` especifica cuántas *figuras* se dibujarán, y en qué orden. En particular, estamos pidiendo al dispositivo de graficado que haga espacio para 2*3 = 6 figuras organizadas en dos filas y tres columnas dibujadas por fila.

`mai` especifica el tamaño de los márgenes en pulgadas, establecer todos los márgenes iguales a cero (que es lo que estamos haciendo ahora) da más espacio al gráfico. Lo mismo es cierto para `oma`. Ver `?par` para más información.

- `myplot(ig_year1, ...)` Esto es simplemente llamar nuestra función de graficado. La parte elegante de esto es que, dado que establecimos `mfrow = c(2, 3)`, R se encarga de *distribuir* los gráficos en el dispositivo.
- `par(oldpar)` Esta línea nos permite restaurar los parámetros de graficado.

4.5 Pruebas estadísticas

4.5.1 ¿Está correlacionado el número de nominación con el indegree?

Hipótesis: Los individuos que, en promedio, están entre las primeras nominaciones de sus pares son más populares

```
# Obteniendo todos los datos en formato largo
edgelist <- as_long_data_frame(ig_year1) |>
  as_tibble()

# Calculando indegree (de nuevo) y número promedio de nominación
# Incluir "En una escala del uno al cinco qué tan cerca te sientes"
# También para amigos egocéntricos (A. Amigos)
indeg_nom_cor <- group_by(edgelist, to, to_name, to_school) |>
  summarise(
    indeg   = length(nnom),
    nom_avg = 1/mean(nnom)
  ) |>
  rename(
    school = to_school
  )
```

```
`summarise()` has grouped output by 'to', 'to_name'. You can override using the
`.groups` argument.
```

```
indeg_nom_cor
```

```
# A tibble: 1,561 x 5
# Groups:   to, to_name [1,561]
      to to_name school indeg nom_avg
   <dbl> <chr>    <int> <int>   <dbl>
 1     2 1110002    111    22   0.222
 2     3 1110007    111     7   0.175
 3     4 1110013    111     6   0.171
 4     5 1110014    111    19   0.134
 5     6 1110015    111     3   0.15
 6     7 1110020    111     6   0.154
 7     9 1110025    111     6   0.214
 8    10 1110027    111    13   0.220
 9    11 1110029    111    14   0.131
10    12 1110030    111     6   0.222
# i 1,551 more rows
```

```
# Usando correlación de Pearson
with(indeg_nom_cor, cor.test(indeg, nom_avg))
```

```
    Pearson's product-moment correlation

data:  indeg and nom_avg
t = -12.254, df = 1559, p-value < 2.2e-16
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 -0.3409964 -0.2504653
sample estimates:
       cor
-0.2963965
```

```
save.image("03.rda")
```

5 Simulación y Visualización

> ⚠ Nota de Traducción
>
> Esta versión del capítulo fue traducida de manera automática utilizando IA. El capítulo aún no ha sido revisado por un humano.

En este capítulo, construiremos y visualizaremos redes artificiales utilizando Modelos Exponenciales de Grafos Aleatorios [ERGMs.] Junto con el capítulo 3, este será un ejemplo extendido de cómo leer datos de redes y visualizarlos utilizando algunos de los paquetes de R disponibles.

Para este capítulo, utilizaremos los siguientes paquetes de R:

- `ergm`: Para simular y estimar ERGMs.
- `sna`: Para visualizar redes.
- `igraph`: También para visualizar redes.
- `intergraph`: Para convertir entre objetos `igraph` y `network`.
- `netplot`: Nuevamente, para visualización.
- `netdiffuseR`: Para una función única que usamos para ajustar el tamaño de vértices en igraph.
- `rgexf`: Para construir figuras interactivas (html).

Puedes usar el siguiente bloque de código para instalar cualquier paquete faltante:

```
# Creating the list to install
pkgs <- c(
  "ergm", "sna", "igraph", "intergraph", "netplot", "netdiffuseR", "rgexf"
  )

# Checking if we can load them and install them if not available
for (pkg in pkgs) {
```

```
  if (!require(pkg, character.only = TRUE)) {

    # If not present, will install it
    install.packages(pkg, character.only = TRUE)

    # And load it!
    library(pkg, character.only = TRUE)

  }
}
```

Una versión grabada está disponible aquí.

5.1 Modelos de Grafos Aleatorios

Aunque hay toneladas de datos de redes sociales, utilizaremos uno artificial para este capítulo. Hacemos esto ya que siempre es útil tener más ejemplos simulando redes aleatorias. Para este capítulo, clasificaremos los modelos de grafos aleatorios para muestrear y generar redes en tres categorías:

1. **Exógenos**: Grafos donde la estructura está determinada por una regla macro, ej., densidad esperada, distribución de grados, o secuencia de grados. En estos casos, los enlaces se asignan para cumplir con una macro-propiedad.

2. **Endógenos**: Esta categoría incluye todos los Grafos Aleatorios generados basándose en información endógena, ej., mundo pequeño, libre de escala, etc. Aquí, una regla de creación de enlaces da origen a una macro propiedad, por ejemplo, apego preferencial en redes libres de escala.

3. **Modelos Exponenciales de Grafos Aleatorios**: En general, dado que los ERGMs componen una familia de modelos estadísticos, siempre (o casi siempre) podemos encontrar una especificación de modelo que coincida con las categorías anteriores. Ya sea que estemos pensando en secuencia de grados, apego preferencial, o una mezcla de ambos, los ERGMs pueden ser la línea base para cualquiera de esos modelos.

Los últimos, ERGMs, son una generalización que cubre todas las clases. Debido a eso, utilizaremos ERGMs para generar nuestra red artificial.

5.2 Redes Sociales en Escuelas

Un tipo común de red que analizamos son las redes de amistad. En este caso, utilizaremos ERGMs para simular redes de amistad dentro de una escuela. En nuestro mundo simulado, estas redes estarán dominadas por los siguientes fenómenos:

- Baja densidad,
- Homofilia racial,
- Balance estructural,
- Y homofilia de edad.

Si has estado prestando atención a los capítulos anteriores, notarás que, de estas cinco propiedades, solo una constituye grafos de Markov. Dentro de un enlace, la homofilia y la densidad solo dependen del ego y el alter. En la homofilia racial, solo importa la raza del ego y del alter para la formación del enlace, pero, en el caso del Balance estructural, el ego es más probable que se haga amigo del alter si un amigo del ego es amigo del alter, es decir, "el amigo de mi amigo es mi amigo."

Los pasos de simulación son los siguientes:

1. Sortear una población de n estudiantes y distribuir aleatoriamente raza y edad entre ellos.
2. Crear un objeto `network`.
3. Simular los enlaces en la red vacía.

Aquí está el código:

```
set.seed(712)
n <- 200

# Step 1: Students
race   <- sample(c("white", "non-white"), n, replace = TRUE)
age    <- sample(c(10, 14, 17), n, replace = TRUE)

# Step 2: Create an empty network
library(ergm)
library(network)
```

```
net <- network.initialize(n)

net %v% "race"   <- race
net %v% "age"    <- age

# Step 3: Simulate a graph
net_sim <- simulate(
    net ~ edges +
    nodematch("race") +
    ttriad +
    absdiff("age"),
    coef = c(-4, .5, .25, -.5)
    )
```

¿Qué acaba de pasar? Aquí hay un desglose línea por línea:

1. `set.seed(712)` Dado que esta es una simulación aleatoria, necesitamos fijar una semilla para que sea reproducible. De otra manera, los resultados cambiarían con cada iteración.

2. `n <- 200` Estamos asignando el valor `200` al objeto `n`. Esto hará las cosas más fáciles ya que, si es necesario, cambiar el tamaño de las redes puede hacerse en la parte superior del código.

3. `race <- sample(c("white", "non-white"), n, replace = TRUE)` Estamos muestreando 200, o en realidad, `n` valores del vector `c("white", "non-white")` con reemplazo.

4. `age <- sample(c(10, 14, 17), n, replace = TRUE)` ¡Lo mismo que antes, pero con edades!

5. `library(ergm)` ¡Cargando el paquete de R ergm, que necesitamos para simular las redes!

6. `library(network)` Cargando el paquete de R `network`, que necesitamos para crear el grafo vacío.

7. `net <- network.initialize(n)` Creando un grafo vacío de tamaño `n`.

8. `net %v% "race" <- race` Usando el operador `%v%`, podemos acceder a las características de los vértices en el objeto red. Dado que race no existe en la red aún, el operador simplemente la crea. Nota que el número de vértices coincide con la longitud del vector race.

9. `net %v% "age" <- age` ¡Lo mismo que con race!

10. `net_sim <- simulate(` ¡Simulando un ERGM! Un par de observaciones aquí:

    a. El LHS (lado izquierdo) de la ecuación tiene la red, `net`

    b. El RHS (lo adivinaste) tiene los términos que gobiernan el proceso.

    c. Para baja densidad, usamos el término `edges` con un -4.0 correspondiente para el parámetro.

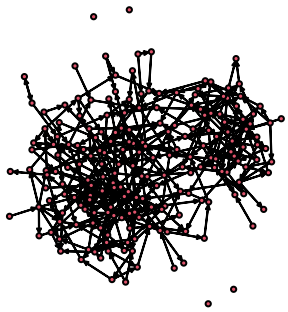
    d. Para homofilia racial, usamos el `nodematch("race")` con un valor de parámetro correspondiente de 0.5.

    e. Para balance estructural, usamos el término `ttriad` con parámetro 0.25.

    f. Para homofilia de edad, usamos el término `absdiff("age")` con parámetro -0.5. Esto es, en rigor, un término que captura heterofilia. No obstante, la heterofilia es lo opuesto a la homofilia.

Echemos un vistazo rápido al grafo resultante:

```
library(sna)
gplot(net_sim)
```



¡Ahora podemos empezar a ver si obtuvimos lo que queríamos! Antes de eso, guardemos la red como un archivo de texto plano para que podamos practicar leyendo redes de vuelta en R!

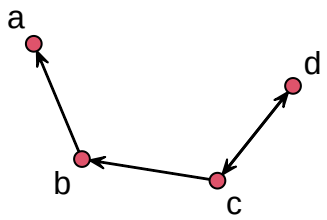
```
write.csv(
  x         = as.edgelist(net_sim),
  file      = "06-edgelist.csv",
  row.names = FALSE
  )

write.csv(
  x         = as.data.frame(net_sim, unit = "vertices"),
  file      = "06-nodes.csv",
  row.names = FALSE
  )
```

5.3 Leyendo una red

El primer paso para analizar datos de red es leerlos. Muchas veces encontrarás datos en forma de una matriz de adyacencia. Otras veces, los datos vendrán en forma de una lista de

enlaces. Otro formato común es la lista de adyacencia, que es una versión comprimida de una lista de enlaces. Veamos cómo se ven los formatos para la siguiente red:

```
example_graph <- matrix(0L, 4, 4, dimnames = list(letters[1:4], letters[1:4]))
example_graph[c(2, 7)] <- 1L
example_graph["c", "d"] <- 1L
example_graph["d", "c"] <- 1L
example_graph <- as.network(example_graph)
set.seed(1231)
gplot(example_graph, label = letters[1:4])
```



- **Matriz de adyacencia** una matriz de tamaño n por n donde la entrada ij-ésima representa el enlace entre i y j. En una red dirigida, decimos que i se conecta a j, por lo que la i-ésima fila muestra los enlaces que i envía al resto de la red. De igual manera, en un grafo dirigido, la j-ésima columna muestra los enlaces enviados a j. Para grafos no dirigidos, la matriz de adyacencia es usualmente diagonal superior o inferior. La matriz de adyacencia de un grafo no dirigido es simétrica, por lo que no necesitamos reportar la misma información dos veces. Por ejemplo:

```
as.matrix(example_graph)
```

```
  a b c d
a 0 0 0 0
b 1 0 0 0
c 0 1 0 1
d 0 0 1 0
```

- **Lista de enlaces** una matriz de tamaño $|E|$ por 2, donde $|E|$ es el número de enlaces. Cada entrada representa un enlace en el grafo.

```
as.edgelist(example_graph)
```

```
     [,1] [,2]
[1,]    2    1
[2,]    3    2
[3,]    3    4
[4,]    4    3
attr(,"n")
[1] 4
attr(,"vnames")
[1] "a" "b" "c" "d"
attr(,"directed")
[1] TRUE
attr(,"bipartite")
[1] FALSE
attr(,"loops")
[1] FALSE
attr(,"class")
[1] "matrix_edgelist" "edgelist"        "matrix"          "array"
```

El comando convierte el objeto `network` en una matriz con un conjunto de atributos (que también se imprimen.)

- **Lista de adyacencia** Este formato de datos usa menos espacio que las listas de enlaces ya que los enlaces se agrupan por ego (fuente.)

```
igraph::as_adj_list(intergraph::asIgraph(example_graph))
```

```
[[1]]
+ 1/4 vertex, from edf38bb:
[1] 2

[[2]]
+ 2/4 vertices, from edf38bb:
[1] 1 3

[[3]]
+ 3/4 vertices, from edf38bb:
[1] 2 4 4

[[4]]
+ 2/4 vertices, from edf38bb:
[1] 3 3
```

La función `igraph::as_adj_list` convierte el objeto igraph en una lista de tipo lista de adyacencia. En texto plano se vería algo así:

```
2
1 3
2 4 4
3 3
```

Aquí trataremos con una lista de enlaces que incluye información de nodos. En mi opinión, esta es una de las mejores maneras de compartir datos de red. Leamos los datos en R usando la función `read.csv`:

```
edges <- read.csv("06-edgelist.csv")
nodes <- read.csv("06-nodes.csv")
```

Ahora tenemos dos objetos de clase `data.frame`, edges y nodes. Inspeccionémoslos usando la función `head`:

```
head(edges)
```

```
  V1  V2
1  1  99
2  2 111
3  3 102
4  3 117
5  4 164
6  5  12
```

```
head(nodes)
```

```
  vertex.names      race age
1            1 non-white  10
2            2     white  10
3            3     white  17
4            4 non-white  14
5            5 non-white  17
6            6 non-white  14
```

Siempre es importante mirar los datos antes de crear la red. La mayoría de errores comunes ocurren antes de leer los datos y podrían pasar desapercibidos en muchos casos. Algunos ejemplos:

- Los encabezados en el archivo podrían ser tratados como datos, o los archivos pueden no tener encabezados.
- Las columnas Ego/alter pueden aparecer en el orden incorrecto. Tanto los paquetes `igraph` como `network` toman la primera y segunda columnas de las listas de enlaces como ego y alter.
- Los aislados, que no aparecerían en la lista de enlaces, pueden estar faltando del conjunto de información de nodos. Este es uno de los errores más comunes.
- Los nodos que aparecen en la lista de enlaces pueden estar faltando de la lista de nodos.

Tanto `igraph` como network tienen funciones para leer listas de enlaces con una lista de nodos correspondiente; las funciones `graph_from_data_frame` y `as.network`, respectivamente. Aunque, para ambos casos, puedes evitar usar una lista de nodos, es altamente recomendado ya que entonces (a) te asegurarás de que los aislados estén incluidos y (b) te darás cuenta de posibles problemas en los datos. Un error frecuente en `graph_from_data_frame` es nodos presentes en la lista de enlaces pero no en el conjunto de nodos.

```
net_ig <- igraph::graph_from_data_frame(
  d        = edges,
  directed = TRUE,
  vertices = nodes
)
```

Usando `as.network` del paquete `network`:

```
net_net <- network::as.network(
  x        = edges,
  directed = TRUE,
  vertices = nodes
)
```

Como puedes ver, ambas sintaxis son muy similares. El punto principal aquí es que mientras más explícitos seamos, mejor. Sin embargo, R puede ser brillante; ser *tímido*, es decir, no lanzar advertencias o errores, no es poco común. En la siguiente sección, finalmente comenzaremos a visualizar los datos.

5.4 Visualizando la red

Nos enfocaremos en tres atributos diferentes que podemos usar para esta visualización: Tamaño de nodo, forma de nodo, y color de nodo. Aunque no hay reglas particulares, algunas ideas que puedes seguir son:

- **Tamaño de nodo** Úsalo para describir una medición continua. Esta característica se usa a menudo para destacar nodos importantes, ej., usando una de las muchas mediciones de grado disponibles.

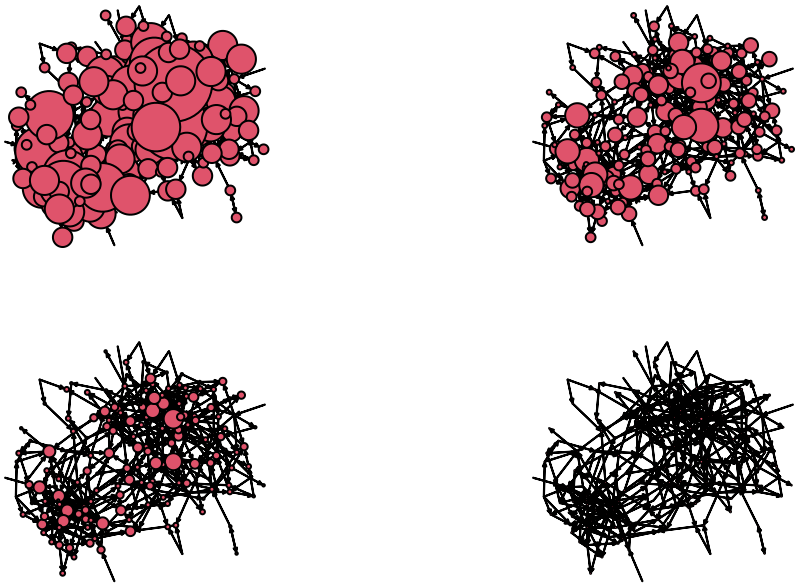
- **Forma de nodo** Las formas pueden usarse para representar valores categóricos. Una buena figura no presentará demasiadas de ellas; menos de cuatro tendría sentido.
- **Color de nodo** Como las formas, los colores pueden usarse para representar valores categóricos, por lo que aplica la misma idea. Además, no es loco usar tanto forma como color para representar la misma característica.

Nota que no hemos hablado de algoritmos de diseño. Los paquetes de R para construir grafos usualmente tienen reglas internas para decidir qué algoritmo usar. Discutiremos eso más adelante. Empecemos por tamaño.

5.4.1 Tamaño de vértice

Encontrar la escala correcta puede ser algo difícil. Dibujaremos el grafo cuatro veces para ver qué tamaño sería el mejor:

```
# Sized by indegree
net_sim %v% "indeg" <- sna::degree(net_sim, cmode = "indegree")

# Changing device config
op <- par(mfrow = c(2, 2), mai = c(.1, .1, .1, .1))

# Plotting
glayout <- gplot(net_sim, vertex.cex = (net_sim %v% "indeg") * 2)
gplot(net_sim, vertex.cex = net_sim %v% "indeg", coord = glayout)
gplot(net_sim, vertex.cex = (net_sim %v% "indeg")/2, coord = glayout)
gplot(net_sim, vertex.cex = (net_sim %v% "indeg")/10, coord = glayout)
```



```
# Restoring device config
par(op)
```

Línea por línea hicimos lo siguiente:

1. `net_sim %v% "indeg" <- degree(net_sim, cmode = "indegree")` Creamos un nuevo atributo de vértice llamado indegree y lo asignamos al objeto red. El indegree se calcula usando la función `degree` del paquete `sna`. Dado que `igraph` también tiene una función `degree`, nos estamos asegurando de que R use la de `sna` y no la de `igraph`. La notación `package::function` es útil para estos casos.

2. `op <- par(mfrow = c(2, 2), mai = c(.1, .1, .1, .1))` Esto cambia la información del dispositivo gráfico a (a) `mfrow = c(2,2)` tener una cuadrícula 2x2 por fila, significando que las nuevas figuras se agregarán de izquierda a derecha y luego de arriba a abajo, y (b) establecer los márgenes en la figura para que sean 0.1 pulgadas en los cuatro tamaños.

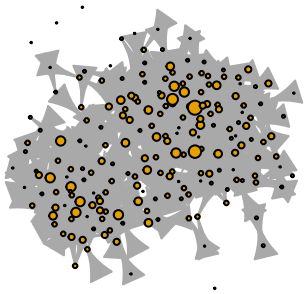
3. `glayout <- gplot(net_sim, vertex.cex = (net_sim %v% "indeg") * 2)` generando la gráfica **y** registrando el diseño. La función `gplot` devuelve una matriz de tamaño `# vertices` por 2 con las posiciones de los vértices. También estamos pasando el argumento `vertex.cex`, que usamos para especificar el tamaño de

cada vértice. En nuestro caso, decidimos dimensionar los vértices proporcional a su indegree *por dos*.

4. `gplot(net_sim, vertex.cex = net_sim %v% "indeg", coord = glayout)`, nuevamente, estamos dibujando el grafo usando las coordenadas del dibujo anterior, pero ahora los vértices son la mitad del tamaño de la figura original.

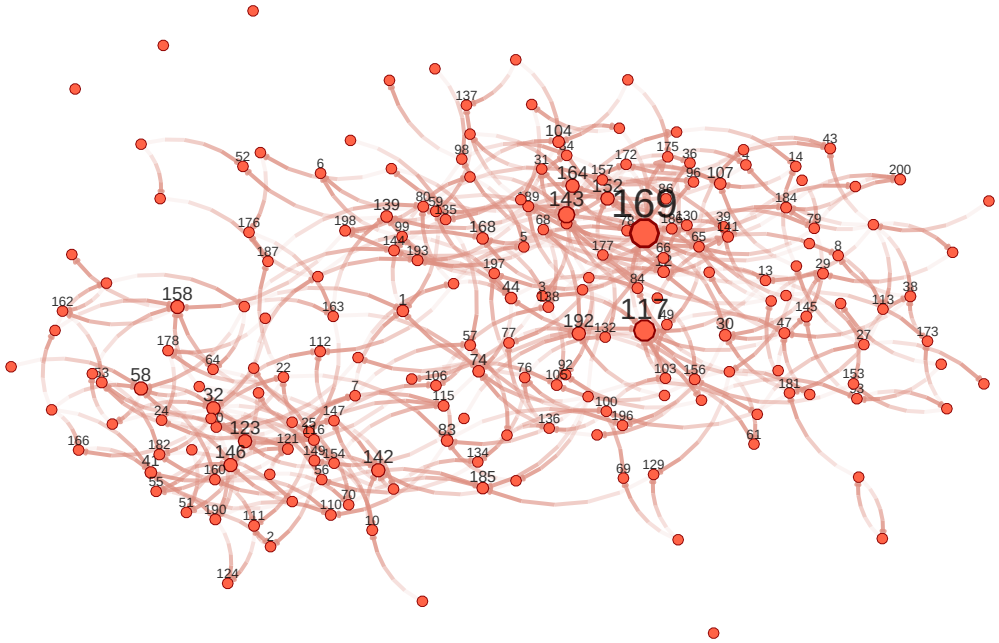
Las otras dos llamadas son similares a la cuatro. Si usáramos igraph, establecer el tamaño puede ser más accesible gracias al paquete de R netdiffuseR. Comencemos convirtiendo nuestra red a un objeto igraph con el paquete de R `intergraph`.

```
library(intergraph)
library(igraph)

# Converting the network object to an igraph object
net_sim_i <- asIgraph(net_sim)

# Plotting with igraph
plot(
  net_sim_i,
  vertex.size = netdiffuseR::rescale_vertex_igraph(
    vertex.size = V(net_sim_i)$indeg,
    minmax.relative.size = c(.01, .1)
  ),
  layout       = glayout,
  vertex.label = NA
)
```



También podríamos haber probado netplot, que debería hacer las cosas más fáciles y hacer un mejor uso del espacio:

```
library(netplot)
nplot(
  net_sim, layout = glayout,
  vertex.color = "tomato",
  vertex.frame.color = "darkred"
  )
```



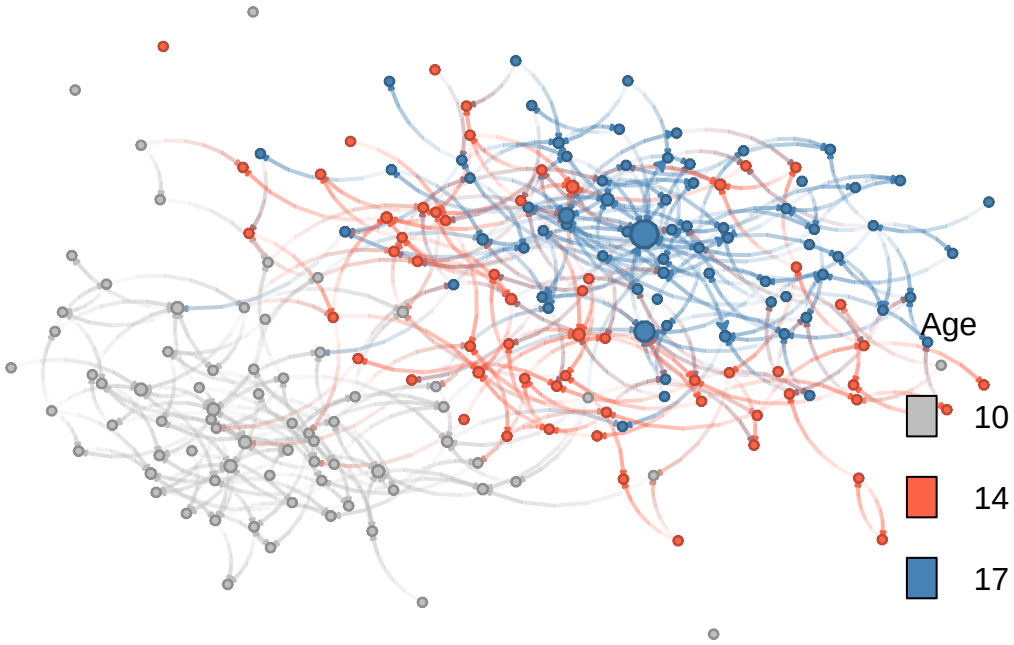
Con una buena idea para el tamaño, ahora podemos empezar a mirar el color de vértice.

5.4.2 Color de vértice

Para el color, usaremos la edad del vértice. Aunque la edad es, por definición, continua, solo tenemos tres valores para la edad. Debido a esto, podemos tratar la edad como categórica. En lugar de usar `nplot` iremos adelante con `nplot_base`. En esta versión del libro, el paquete `netplot` no tiene una manera fácil de agregar leyendas con la función central, `nplot`; por lo tanto, usamos `nplot_base` que es compatible con la función de R `legend`, como veremos ahora:

```
# Specifying colors for each vertex
vcolors_palette <- c("10" = "gray", "14" = "tomato", "17" = "steelblue")
vcolors <- vcolors_palette[as.character(net_sim %v% "age")]
net_sim %v% "color" <- vcolors

# Plotting
nplot_base(
  net_ig,
  layout = glayout,
  vertex.color = net_sim %v% "color",
```

```
  )

# Color legend
legend(
  "bottomright",
  legend = names(vcolors_palette),
  fill   = vcolors_palette,
  bty    = "n",
  title  = "Age"
  )
```



Línea por línea, esto es lo que acabamos de hacer:

1. `vcolors <- c("10" = "gray", "14" = "tomato", "17" = "steelblue")` creamos un vector de caracteres con tres elementos, `"gray"`, `"tomato"`, y `"blue"`. Además, el vector tiene nombres asignados, `"10"`, `"14"`, y `"17"`– las edades que tenemos en la red– para que podamos acceder a sus elementos indexando por nombre, ej., si escribimos `vcolors["10"]` R devuelve el valor `"gray"`.

2. `vcolors <- vcolors[as.character(net_sim %v% "age")]` hay varias cosas pasando en esta línea. Primero, extraemos el atributo "age" de la red usando el operador `%v%`. Luego transformamos el vector resultante de tipo entero a tipo

carácter con la función `as.character`. Finalmente, usando el **vector de caracteres** resultante con valores `"10", "14", "17", ...`, recuperamos valores de `vcolors` indexando por nombre. El vector resultante es de longitud igual al conteo de vértices en la red.

3. `net_sim %v% "color" <- vcolors` crea un nuevo atributo de vértice, `color`. El valor asignado es el resultado de hacer subset de `vcolors` por las edades de cada vértice.

4. `nplot_base(...` finalmente dibuja la red. Pasamos las coordenadas de vértice previamente computadas y los colores de vértice con el nuevo atributo `color`.

5. `legend(...)` Veamos un parámetro a la vez:

   a. `"bottomright"` dice la posición general de la leyenda

   b. `legend = names(vcolors)` pasa la leyenda actual (texto); en nuestro caso las edades de individuos.

   c. `fill = vcolors` pasa los colores asociados con el texto.

   d. `bty = "n"` suprime envolver la leyenda dentro de una caja.

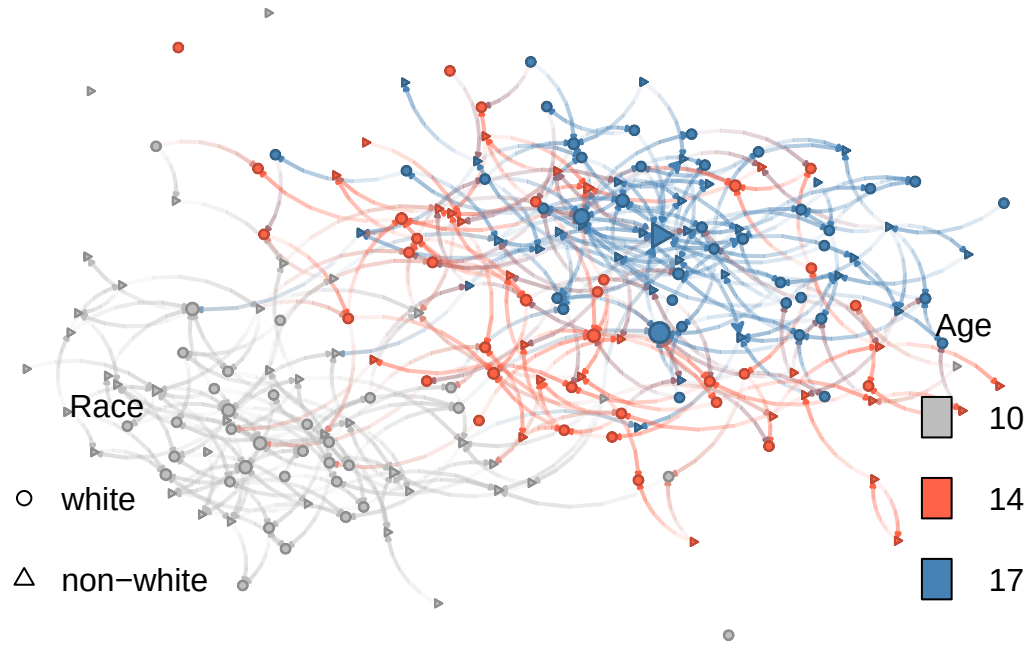
   e. `title = "Age"` establece el título como "Age".

5.4.3 Forma de vértice

Para la forma, usaremos la raza del vértice. Aunque la raza es, por definición, categórica, solo tenemos dos valores para la raza. Debido a esto, podemos tratar la raza como categórica.

```
# Specifying the shapes for each vertex
vshape_list <- c("white" = 15, "non-white" = 3)
vshape      <- vshape_list[as.character(net_sim %v% "race")]
net_sim %v% "shape" <- vshape

# Plotting
nplot_base(
  net_ig,
  layout = glayout,
  vertex.color = net_sim %v% "color",
```

```
  vertex.nsides = net_sim %v% "shape"
  )

# Color legend
legend(
  "bottomright",
  legend = names(vcolors_palette),
  fill   = vcolors_palette,
  bty    = "n",
  title  = "Age"
  )

# Shape legend
legend(
  "bottomleft",
  legend = names(vshape_list),
  pch    = c(1, 2),
  bty    = "n",
  title  = "Race"
  )
```



Ahora comparemos la figura con nuestro ERGM original:

1. **Baja densidad (`edges`)** Sin baja densidad, la figura sería una maraña de cabello.
2. **Homofilia racial (`nodematch("race")`)** Aunque no sorprendentemente evidente, los nodos tienden a formar pequeños grupos por forma, que, en nuestro modelo, representa raza.
3. **Balance estructural (`ttriad`)** Una fuerza, en este caso, opuesta a la baja densidad, mayor prevalencia de triadas transitivas hace que los individuos se agrupen.
4. **Homofilia de edad (`absdiff("age")`)** Esta es la característica más prominente del grafo. En él, los nodos se agrupan por edad.

De las cuatro características, **homofilia de edad** es la que se destaca. ¿Por qué es este el caso? Si miramos nuevamente los parámetros usados en el ERGM y cómo estos interactúan con los atributos de los vértices, encontraremos la respuesta:

- Las log-odds de un nuevo enlace racialmente homofílico son $1 \times \theta_{\text{race-homophily}} = 0.5$.
- Pero, las log-odds de un enlace heterofílico de edad entre, digamos, jóvenes de 14 y 17 años es $|17 - 14|\theta_{\text{age-homophily}} = 3 \times -0.5 = -1.5$.
- Por lo tanto, el efecto de heterofilia (que es justo lo opuesto a homofilia) es significativamente mayor, en realidad tres veces en este caso, que el efecto de homofilia racial.

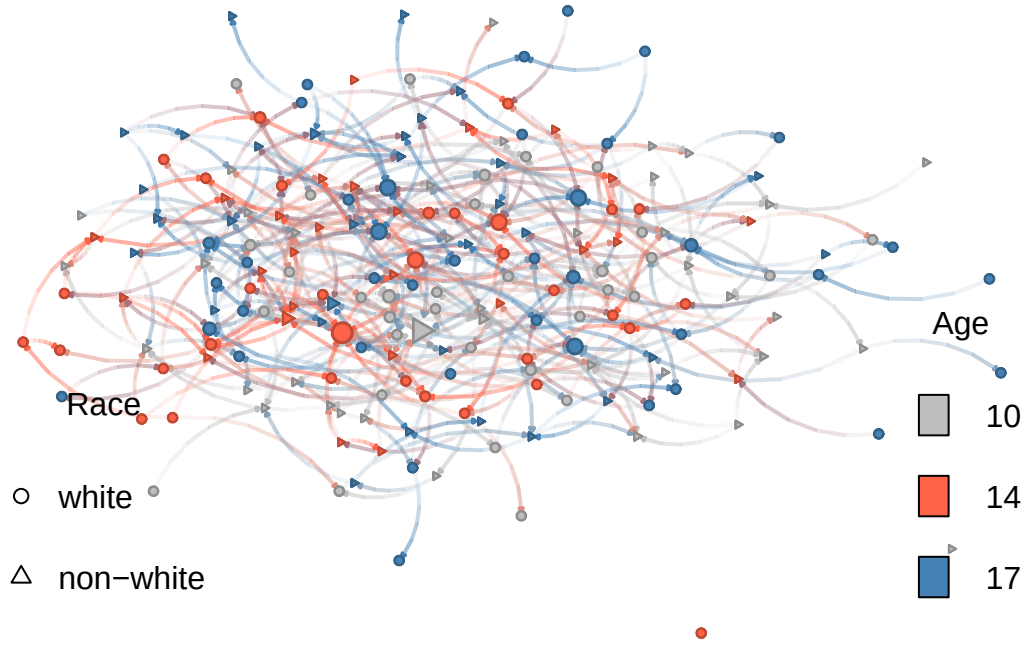
Esta observación se vuelve clara si ejecutamos otra simulación con la misma semilla, pero ajustando para el tamaño máximo que el efecto de homofilia de edad puede tomar. Una manera rápida y sucia de lograr esto es re-ejecutar la simulación con el término `nodematch` en lugar del término `absdiff`. De esta manera, (a) explícitamente operacionalizamos el término como homofilia (antes era heterofilia,) y (b) tenemos ambos efectos de homofilia con la misma influencia en el modelo:

```
net_sim2 <- simulate(
    net ~ edges +
    nodematch("race") +
    ttriad +
    nodematch("age"),
```

```
    coef = c(-5, .5, .25, .5) # This line changed
    )
```

Re-haciendo la gráfica. Del dibujo de grafo anterior, solo la estructura del grafo cambió. Los atributos de vértice son los mismos por lo que podemos ir adelante y re-usarlos. Como mencioné anteriormente, la función `nplot_base` actualmente soporta objetos `igraph`, por lo que usaremos `intergraph::asIgraph` para que funcione:

```
# Plotting
nplot_base(
  asIgraph(net_sim2),
  # We comment this out to allow for a new layout
  # layout = glayout,
  vertex.color = net_sim %v% "color",
  vertex.nsides = net_sim %v% "shape"
  )

# Color legend
legend(
  "bottomright",
  legend = names(vcolors_palette),
  fill   = vcolors_palette,
  bty    = "n",
  title  = "Age"
  )

# Shape legend
legend(
  "bottomleft",
  legend = names(vshape_list),
  pch    = c(1, 2),
  bty    = "n",
  title  = "Race"
  )
```



Como se esperaba, ya no hay un efecto dominante en homofilia. Una cosa importante que podemos aprender de este ejemplo final es que los fenómenos no siempre se mostrarán en la visualización de grafos. Un análisis cuidadoso en redes complejas es imprescindible.

6 Redes egocéntricas

> ⚠ Nota de Traducción
>
> Esta versión del capítulo fue traducida de manera automática utilizando IA. El capítulo aún no ha sido revisado por un humano.

En el análisis de redes sociales egocéntricas (ESNA, para nuestro libro,) en lugar de tratar con una sola red, tenemos tantas redes como participantes en el estudio. Los egos–los sujetos principales del estudio–se analizan desde la perspectiva de su red social local. Para una vista más extendida de ESNA, revisa "*Análisis de redes egocéntricas con R*" de Raffaele Vacca.

En este capítulo, muestro cómo trabajar con un tipo particular de datos ESNA: información generada por la herramienta Network Canvas. Puedes descargar un archivo ZIP "artificial" que contiene las salidas de un proyecto de Network Canvas aquí[1]. Asumimos que el archivo ZIP fue extraído a la carpeta `data-raw/egonets`. Puedes proceder y extraer el ZIP por punto y clic o usar el siguiente código R para automatizar el proceso:

```
[1] FALSE
```

```
unzip(
  zipfile = "data-raw/networkCanvasExport-fake.zip",
  exdir   = "data-raw/egonets"
  )
```

Esto extraerá todos los archivos en `networkCanvasExport-fake.zip` a la subcarpeta `egonets`. Echemos un vistazo a los primeros archivos:

[1] Agradezco a Jacqueline M. Kent-Marvick, quien me proporcionó lo que usé como línea base para generar la exportación artificial de Network Canvas.

```
head(list.files(path = "data-raw/egonets"))
## [1] "I_-59190_BRB9111_attributeList_Person.csv"
## [2] "I_-59190_BRB9111_edgeList_Knows.csv"
## [3] "I_-59190_BRB9111_ego.csv"
## [4] "I_-59190_BRB9111.graphml"
## [5] "I-100BB_00B95-90_attributeList_Person.csv"
## [6] "I-100BB_00B95-90_edgeList_Knows.csv"
```

Como puedes ver, para cada ego en el conjunto de datos, hay cuatro archivos:

- `...attributeList_Person.csv`: Atributos de los alters.
- `...edgeList_Knows.csv`: Lista de enlaces indicando los vínculos entre los alters.
- `...ego.csv`: Información sobre los egos.
- `...graphml`: Y un archivo `graphml` que contiene las redes egocéntricas.

Las siguientes secciones ilustrarán, archivo por archivo, cómo leer la información en R, aplicar cualquier procesamiento requerido, y almacenar la información para uso posterior. Comenzamos con los archivos `graphml`.

6.1 Archivos de red (graphml)

Los archivos `graphml` pueden leerse directamente con la función `read_graph` de `igraph`. La clave es aprovechar las listas de R para evitar escribir una y otra vez el mismo bloque de código, y, en su lugar, manejar los datos a través de listas.

Al igual que cualquier función de lectura de datos, la función `read_graph` requiere una ruta de archivo al archivo de red. **La función que usaremos para listar los archivos requeridos es `list.files()`**:

```
# Comenzamos cargando igraph
library(igraph)

# Listando todos los archivos graphml
graph_files <- list.files(
```

```
  path       = "data-raw/egonets", # ¿Dónde están estos archivos?
  pattern    = "*.graphml",        # Especificar un patrón para solo listar graphml
  full.names = TRUE                # Y nos aseguramos de usar el nombre completo
                                   # (ruta.) De lo contrario, solo obtendríamos nombres.
  )

# Echando un vistazo a los primeros tres archivos que obtuvimos
graph_files[1:3]
## [1] "data-raw/egonets/I_-59190_BRB9111.graphml"
## [2] "data-raw/egonets/I-100BB_00B95-90.graphml"
## [3] "data-raw/egonets/I-1BB79950-0-7.graphml"

# Aplicando read_graph de igraph
graphs <- lapply(
  X      = graph_files,       # Lista de archivos a leer
  FUN    = read_graph,        # La función a aplicar
  format = "graphml"          # Argumento pasado a read_graph
  )
```

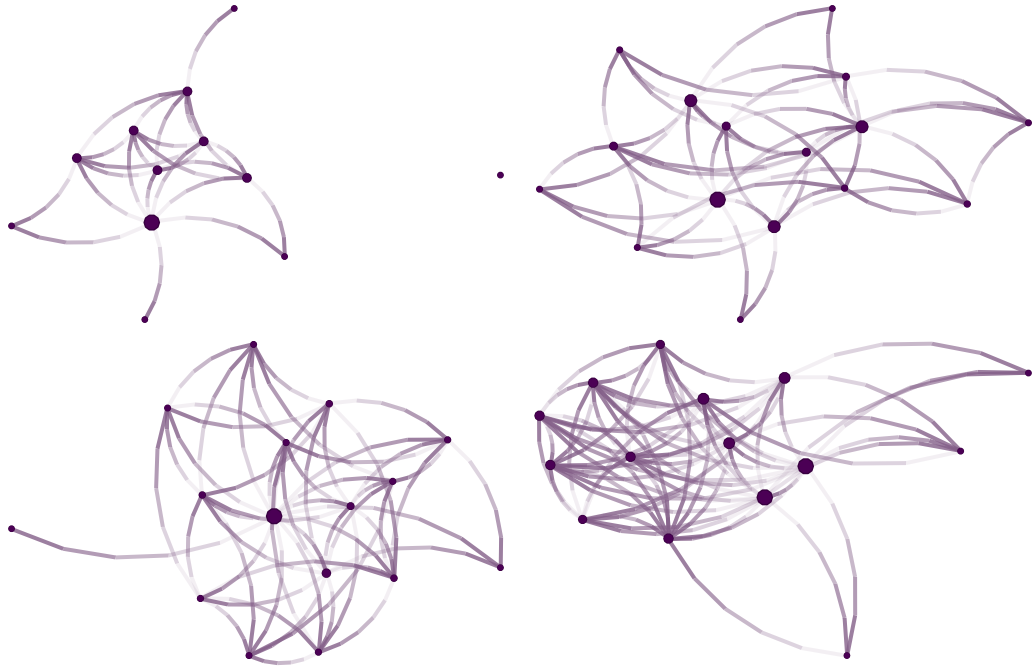
Si la operación tuvo éxito, el bloque de código anterior debería generar una lista de objetos `igraph` llamada `graphs`. Echemos un vistazo a los primeros dos:

```
graphs[[1]]
## IGRAPH b55dc21 U--- 12 25 --
## + attr: age (v/n), healthy_diet (v/n), gender_1 (v/l), eat_with_2
## | (v/l), id (v/c)
## + edges from b55dc21:
##  [1] 1-- 3 1-- 2 1-- 6 1-- 5 1-- 4 1-- 8 1--11 1--10 2-- 3 3-- 7 3-- 4 3-- 5
## [13] 3-- 6 2-- 7 2-- 4 2-- 5 2-- 6 5-- 6 6--10 7-- 9 4-- 5 5-- 7 4--11 6-- 7
## [25] 4-- 7
graphs[[2]]
## IGRAPH 09628f3 U--- 16 47 --
## + attr: age (v/n), healthy_diet (v/n), gender_1 (v/l), eat_with_2
## | (v/l), id (v/c)
## + edges from 09628f3:
##  [1]  7--13  1-- 5  1-- 6  1-- 4  1-- 2  7--15  1-- 3 11--13  1--10  1--16
```

```
## [11]  4-- 6  2-- 6  6-- 7  1--11 11--15  6-- 9  6-- 8  3-- 9  5--15  4-- 5
## [21]  2-- 5  5-- 8  5-- 7  5--10  3-- 5  6--14 12--13  6--13  3--13  2-- 3
## [31]  3-- 4  3--16  3--11 10--14  7--14  2-- 4  2--10  2--15 10--12  4-- 7
## [41]  6--10  5--11  9--10  1-- 9  1--12  3--12  4--14
```

Como siempre, una de las primeras cosas que hacemos con redes es visualizarlas. Usaremos el paquete de R `netplot` (de un servidor) para dibujar las figuras:

```
library(netplot)
library(gridExtra)

# El diseño del grafo es aleatorio
set.seed(1231)

# grid.arrange permite poner múltiples gráficos netplot en la misma página
grid.arrange(
  nplot(graphs[[1]]),
  nplot(graphs[[2]]),
  nplot(graphs[[3]]),
  nplot(graphs[[4]]),
  ncol = 2, nrow = 2
)
```



¡Excelente! Dado que los nodos en nuestra red tienen características, podemos agregar un poco de color. Usaremos la variable `eat_with_2`, codificada como `TRUE` o `FALSE`. Los colores de los vértices pueden especificarse usando el argumento `vertex.color` de la función `nplot`. En nuestro caso, especificaremos colores pasando un vector con longitud igual al número de nodos en el grafo. Además, dado que haremos esto múltiples veces, vale la pena escribir una función:

```
# Una función para colorear por la variable come con
color_it <- function(net) {

  # Codificando eat_with_2 para ser 1 (FALSE) o 2 (TRUE)
  eatswith <- V(net)$eat_with_2

  # Subconjuntando el color
  ifelse(eatswith, "purple", "darkgreen")

}
```

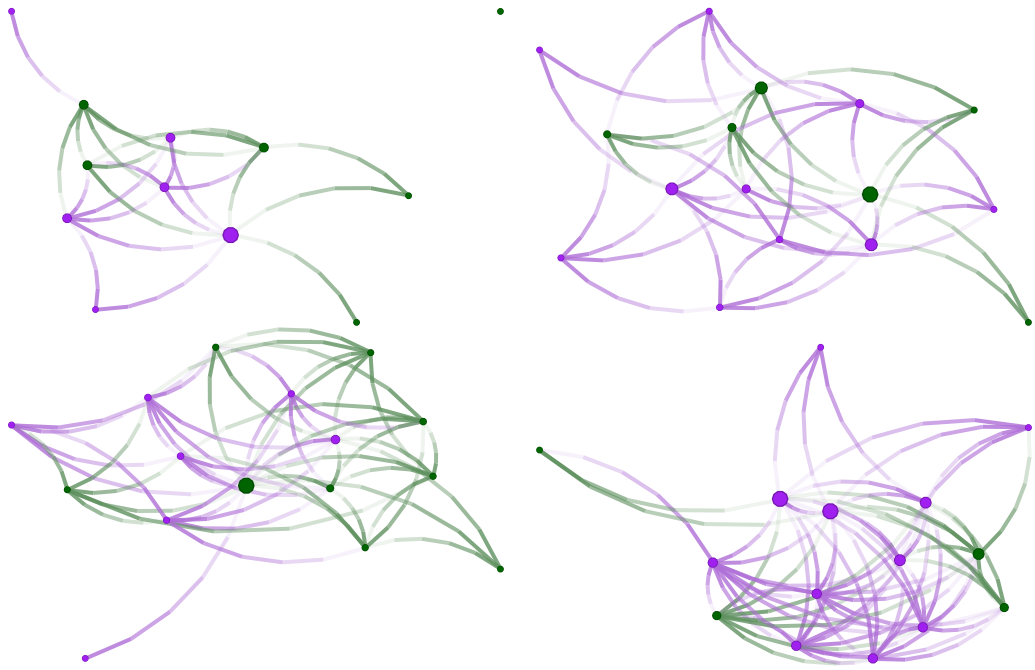
Esta función toma dos argumentos: una red y un vector de dos colores. Los atributos de vértice en `igraph` pueden accederse a través de la función `V(...)$...`. Para este ejemplo, para acceder al atributo `eat_with_2` en la red `net`, escribimos `V(net)$eat_with_2`. Finalmente,

individuos con `eat_with_2` igual a true serán coloreados `purple`; de lo contrario, si es igual a `FALSE`, serán coloreados `darkgreen`. Antes de graficar las redes, veamos qué obtenemos cuando accedemos al atributo `eat_with_2` en el primer grafo:

```
V(graphs[[1]])$eat_with_2
##  [1]  TRUE  TRUE FALSE FALSE  TRUE  TRUE FALSE FALSE  TRUE  TRUE FALSE FALSE
```

Un vector lógico. Ahora redibujemos las figuras:

```
grid.arrange(
  nplot(graphs[[1]], vertex.color = color_it(graphs[[1]])),
  nplot(graphs[[2]], vertex.color = color_it(graphs[[2]])),
  nplot(graphs[[3]], vertex.color = color_it(graphs[[3]])),
  nplot(graphs[[4]], vertex.color = color_it(graphs[[4]])),
  ncol = 2, nrow = 2
)
```



Dado que la mayoría del tiempo, estaremos tratando con muchas redes egocéntricas; puedes querer dibujar cada red independientemente; el siguiente bloque de código hace eso. Primero, si es necesario, creará una carpeta para almacenar las redes. Luego, usando la función `lapply`, usará `netplot::nplot()` para dibujar las redes, agregar una leyenda, y guardar el

grafo como `.../graphml_[número].png`, donde `[número]` irá de `01` al número total de redes en `graphs`.

```
if (!dir.exists("egonets/figs/egonets"))
  dir.create("egonets/figs/egonets", recursive = TRUE)

lapply(seq_along(graphs), function(i) {

  # Creando el dispositivo
  png(sprintf("egonets/figs/egonets/graphml_%02i.png", i))

  # Dibujando el gráfico
  p <- nplot(
    graphs[[i]],
    vertex.color = color_it(graphs[[i]])
    )

  # Agregando una leyenda
  p <- nplot_legend(
    p,
    labels = c("come con: FALSE", "come con: TRUE"),
    pch    = 21,
    packgrob.args = list(side = "bottom"),
    gp            = gpar(
      fill = c("darkgreen", "purple")
    ),
    ncol = 2
  )

  print(p)

  # Cerrando el dispositivo
  dev.off()
})
```

6.2 Archivos de persona

Como antes, listamos los archivos que terminan en `Person.csv` (con la ruta completa,) y los leemos en R. Aunque R tiene la función `read.csv`, aquí uso la función `fread` del paquete de R `data.table`. Junto con `dplyr`, `data.table` es una de las herramientas de manipulación de datos más populares en R. Además de la sintaxis, la mayor diferencia entre las dos es el rendimiento; `data.table` es significativamente más rápido que cualquier otro paquete de manejo de datos en R, y es una gran alternativa para manejar grandes conjuntos de datos. El siguiente bloque de código carga el paquete, lista los archivos, y los lee en R.

```
# Cargando data.table
library(data.table)

# Listando los archivos
person_files <- list.files(
  path       = "data-raw/egonets",
  pattern    = "*Person.csv",
  full.names = TRUE
  )

# Cargando todos en una sola lista
persons <- lapply(person_files, fread)

# Mirando el primer elemento
persons[[1]]
##     nodeID   age
##      <int> <int>
##  1:      1    45
##  2:      2    32
##  3:      3    31
##  4:      4    45
##  5:      5    43
##  6:      6    47
##  7:      7    45
##  8:      8    62
##  9:      9    28
```

```
## 10:      10    41
## 11:      11    41
## 12:      12    46
## 13:      13    46
## 14:      14    46
## 15:      15    62
## 16:      16    41
```

Una tarea común es agregar un identificador a cada conjunto de datos en `persons` para que sepamos a qué ego pertenecen. De nuevo, la función `lapply` es nuestra amiga:

```
persons <- lapply(seq_along(persons), function(i) {
  persons[[i]][, dataset_num := i]
})
```

En `data.table`, las variables se crean usando el símbolo `:=`. El fragmento de código anterior es equivalente a esto:

```
for (i in 1:length(persons)) {
  persons[[i]]$dataset_num <- i
}
```

Si es necesario, podemos transformar la lista `persons` en un objeto `data.table` (es decir, un solo `data.frame`) usando la función `rbindlist`[2]. El siguiente bloque de código usa esa función para combinar los `data.table`s en un solo conjunto de datos.

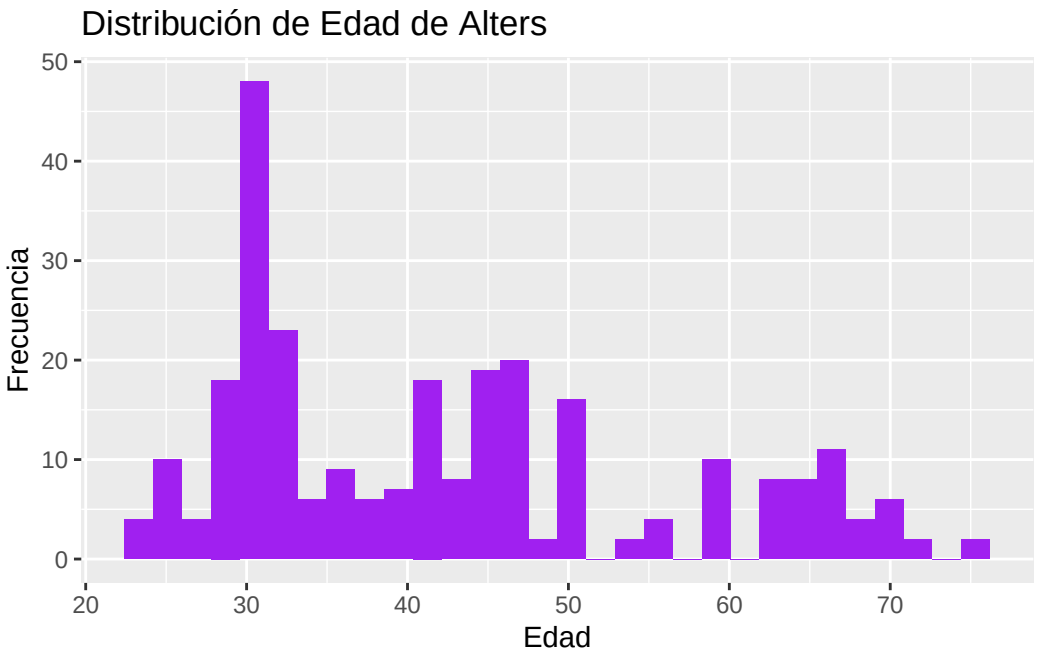
```
# Combinando los conjuntos de datos
persons <- rbindlist(persons)
persons
##       nodeID   age dataset_num
##        <int> <int>       <int>
##    1:      1    45           1
##    2:      2    32           1
```

[2]Aunque no es lo mismo, `rbindlist` (casi siempre) produce el mismo resultado que llamar la función `do.call`. En particular, en lugar de ejecutar la llamada `rbindlist(persons)`, podríamos haber usado `do.call(rbind, persons)`.

```
##   3:      3    31           1
##   4:      4    45           1
##   5:      5    43           1
##  ---
## 271:      7    43          19
## 272:      8    48          19
## 273:      9    70          19
## 274:     10    46          19
## 275:     11    50          19
```

Ahora que tenemos un solo conjunto de datos, podemos hacer algo de exploración de datos. Por ejemplo, podemos usar el paquete `ggplot2` para dibujar un histograma de las edades de los alters.

```
# Cargando el paquete ggplot2
library(ggplot2)

# Histograma de edad
ggplot(persons, aes(x = age)) +            # Iniciando el gráfico
  geom_histogram(fill = "purple") +      # Agregando un histograma
  labs(x = "Edad", y = "Frecuencia") +       # Cambiando las etiquetas del eje x/y
  labs(title = "Distribución de Edad de Alters") # Agregando un título
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



6.3 Archivos de ego

Los archivos de ego contienen información sobre los egos (¡obvio!.) De nuevo, los leeremos todos a la vez usando `list.files` + `lapply`:

```
# Listando archivos que terminan con *ego.csv
ego_files <- list.files(
  path       = "data-raw/egonets",
  pattern    = "*ego.csv",
  full.names = TRUE
  )

# Leyendo los archivos con fread
egos <- lapply(ego_files, fread)

# Combinándolos
egos <- rbindlist(egos)
head(egos)
##                    networkCanvasEgoUUID networkCanvasCaseID
##                                  <char>              <char>
```

```
## 1: I-11ca3a78c-62f131f37169-c139217a1f6    I_-59190_BRB9111
## 2: I-fef-ab-4-5a--7-35c4f23-96eb32-34ea    I-100BB_00B95-90
## 3: I2f1bd0b6d-f71f4664cf-d-26-97408f22d      I-1BB79950-0-7
## 4: Id36bb-3b2bcbd2a6239b1103134c6b3d1d6    I000091I_RB010B5
## 5: I436d32fc67fb5c6-23-244f353849b120cd    I019051R0_RRR0-0
## 6: Ibf1f-2-34162bb5f2c36b8241--316a-fff    I01B11-I1101_44R
##                  networkCanvasSessionID
##                                  <char>
## 1: I612b7a1af---0880b-70698204-b-8dbf09
## 2: If5e0-f-26cbec070760f-e6b6d26ebfb06f
## 3: I825c293a1304-e5-cbea8a80aae05b305fa
## 4: I1b8a7d0f6b4-8298c9-848-9186d68a7f3c
## 5: Ie620be37b75983c49ac63-38-425227c959
## 6: Ie3-134323ed40-0e-d954b3d-febbcb9363
##                      networkCanvasProtocolName        sessionStart
##                                          <char>               <POSc>
## 1: Postpartum social networks with sociogram_V5 2023-02-22 23:41:59
## 2: Postpartum social networks with sociogram_V5 2023-02-10 21:46:02
## 3: Postpartum social networks with sociogram_V5 2023-03-01 16:52:09
## 4: Postpartum social networks with sociogram_V5 2023-01-26 20:38:07
## 5: Postpartum social networks with sociogram_V5 2023-02-06 14:55:57
## 6: Postpartum social networks with sociogram_V5 2023-03-16 18:20:02
##          sessionFinish     sessionExported
##                 <POSc>              <POSc>
## 1: 2023-02-23 01:47:00 2023-02-23 01:47:08
## 2: 2023-02-11 01:29:32 2023-02-11 01:34:12
## 3: 2023-03-02 16:51:20 2023-03-02 17:04:42
## 4: 2023-01-26 22:03:20 2023-01-26 22:03:34
## 5: 2023-02-06 15:49:38 2023-02-06 15:56:42
## 6: 2023-03-17 21:11:09 2023-03-17 21:16:15
```

Algo genial sobre `data.table` es que, dentro de corchetes cuadrados, podemos manipular los datos refiriéndonos a las variables directamente. Por ejemplo, si quisiéramos calcular la diferencia entre `sessionFinish` y `sessionStart`, usando R base haríamos lo siguiente:

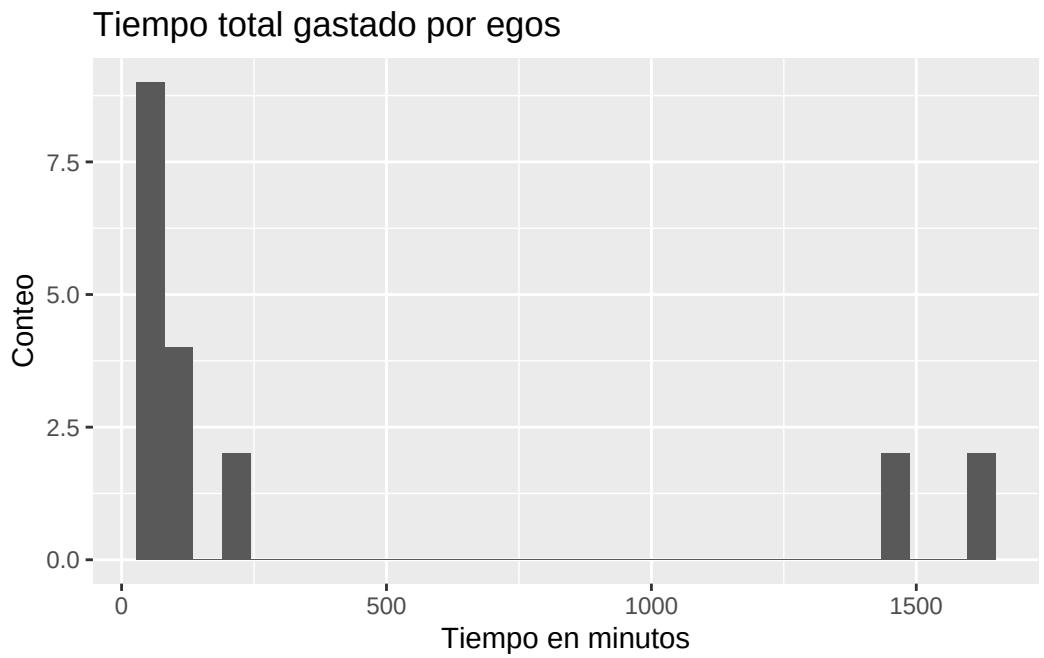
```
egos$total_time <- egos$sessionFinish - egos$sessionStart
```

Mientras que con `data.table`, la creación de variables es mucho más directa (nota que en lugar de usar `<-` o `=` para asignar una variable, usamos el operador `:=`):

```
# ¿Cuánto tiempo?
egos[, total_time := sessionFinish - sessionStart]
```

También podemos visualizar esto usando `ggplot2`:

```
ggplot(egos, aes(x = total_time)) +
  geom_histogram() +
  labs(x = "Tiempo en minutos", y = "Conteo") +
  labs(title = "Tiempo total gastado por egos")
## Don't know how to automatically pick scale for object of type <difftime>.
## Defaulting to continuous.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



6.4 Archivos de lista de enlaces

Como mencioné antes, dado que estamos leyendo los archivos `graphml`, usar la lista de enlaces puede no ser necesario. Sin embargo, el proceso para importar el archivo de lista de enlaces a R es el mismo que hemos estado aplicando: listar los archivos y leerlos todos a la vez usando `lapply`:

```
# Listando todos los archivos que terminan en Knows.csv
edgelist_files <- list.files(
  path = "data-raw/egonets",
  pattern = "*Knows.csv",
  full.names = TRUE
  )

# Leyendo todos los archivos a la vez
edgelists <- lapply(edgelist_files, fread)
```

Para evitar confusión, también podemos agregar ids correspondientes al número de archivo. Una vez que hagamos eso, podemos combinar todos los archivos en un solo objeto `data.table` usando `rbindlist`:

```
edgelists <- lapply(seq_along(edgelists), function(i) {
  edgelists[[i]][, dataset_num := i]
})

edgelists <- rbindlist(edgelists)

head(edgelists)
##    edgeID  from    to                networkCanvasEgoUUID
##     <int> <int> <int>                              <char>
## 1:      1     1     5 I839f-8fa8f8aeb8-eaf---ba8-cf3908f3a
## 2:      2     1    10 If81a9c0f-9f4f28ccf-c4c923a-8-0f5fce
## 3:      3     1     9 I899ffe-27-3-a3-ca2fb7f7-ca8e7715ce9
## 4:      4     1    10 I814efaba88cbb02caa8c89790-83beeaf9-
## 5:      5     7     6 Ifd-0eec2e08974eaf2b79f-9efb7e3-8998
## 6:      6     2     6 I-28fe89cc-fc5db3825b92-ae87c-c18e3d
```

```
##                         networkCanvasUUID              networkCanvasSourceUUID
##                                    <char>                                <char>
## 1: I720400eb19bccce-77cee773289b02-fe7e I4d5--16a08f8ba463c6458f8979e-65fa9d
## 2: I-b469c0-60f8bbb543-32-628-216f9-038 I-6cf8-f3da-4-96-87efaf5daaa48ba5e5c
## 3: Ifa4933-9baaf5fc-f-e4f5c5e5-ff34-f-f I5-f69a6eaa-5956e8897ca999-ffb6ed-e1
## 4: I4cb-904496b1-6194bcb51b58444b40-ef8 I3e5-6c8d5e0f086--e-5ab45-4-5aaa5-0e
## 5: I0ab7--b7a0ee71e54c1e93cdb-4ca5ab1-b I5-b-9-7eca5ab5-91915ba9b6565a6e42cc
## 6: Ic80142fc4c431009e84b3-ab3f-9b0eab03 Ie0a24eea4e01a4340343a0-66723-a-9970
##                  networkCanvasTargetUUID dataset_num
##                                    <char>       <int>
## 1: Id1c8befd46bdd195c-ce91a8-bc0---4f0e           1
## 2: I757b4a-3ea4d95--b9ebb9db3d55dcbaf-c           1
## 3: I92a62925ff9-e2f27-6ef97d-29fb729624           1
## 4: I7f--da48-46a64-b972c-ef6bbec--64cb4           1
## 5: I-eaa7e95659-9cf01a4f5fd69af54e6-d60           1
## 6: I69060e8a-454609-faa04cd3eeb-5-9550-           1
```

6.5 Juntando todo

En esta última parte del capítulo, usaremos los paquetes `igraph` y `ergm` para generar características (covariables, controles, variables independientes, o como las llames) a nivel de red egocéntrica. Una vez más, la función `lapply` es nuestra amiga

6.5.1 Generando estadísticas usando igraph

El paquete de R `igraph` tiene múltiples rutinas de alto rendimiento para calcular estadísticas a nivel de grafo. Por ahora, nos enfocaremos en las siguientes estadísticas: conteo de vértices, conteo de enlaces, número de aislados, transitividad, y modularidad basada en centralidad de intermediación:

```
net_stats <- lapply(graphs, function(g) {

  # Calculando modularidad
  groups <- cluster_edge_betweenness(g)
```

```
  # Calculando las estadísticas
  data.table(
    size      = vcount(g),
    edges     = ecount(g),
    nisolates = sum(degree(g) == 0),
    transit   = transitivity(g, type = "global"),
    modular   = modularity(groups)
  )
})
```

Observa que contamos aislados usando la función `degree()`. Podemos combinar las estadísticas en un solo `data.table` usando la función `rbindlist`:

```
net_stats <- rbindlist(net_stats)

head(net_stats)
##     size edges nisolates   transit     modular
##    <num> <num>     <int>     <num>       <num>
## 1:    12    25         1 0.6750000 0.012000000
## 2:    16    47         0 0.4332130 0.003395201
## 3:    16    58         0 0.5612009 0.002675386
## 4:    15    75         0 0.8515112 0.000000000
## 5:    15    52         0 0.5780488 0.000000000
## 6:    17    68         0 0.6291161 0.025735294
```

6.5.2 Generando estadísticas basadas en ergm

El paquete de R `ergm` tiene un conjunto mucho más grande de estadísticas a nivel de grafo que podemos agregar a nuestros modelos.[3] La clave para generar estadísticas basadas en el paquete `ergm` es la función `summary_formula`. Antes de empezar a usar esa función, primero necesitamos convertir las redes `igraph` a objetos `network`, que son la clase de objeto nativa para el paquete `ergm`. Usamos el paquete de R `intergraph` para eso, y en particular, la función `asNetwork`:

[3] ¡Hay una razón obvia, los ERGMs son todo sobre estadísticas a nivel de grafo!

```r
# Cargando los paquetes requeridos
library(intergraph)
library(ergm)
## Loading required package: network
##
## 'network' 1.19.0 (2024-12-08), part of the Statnet Project
## * 'news(package="network")' for changes since last version
## * 'citation("network")' for citation information
## * 'https://statnet.org' for help, support, and other information
##
## Attaching package: 'network'
## The following objects are masked from 'package:igraph':
##
##     %c%, %s%, add.edges, add.vertices, delete.edges, delete.vertices,
##     get.edge.attribute, get.edges, get.vertex.attribute, is.bipartite,
##     is.directed, list.edge.attributes, list.vertex.attributes,
##     set.edge.attribute, set.vertex.attribute
##
## 'ergm' 4.9.0 (2025-06-09), part of the Statnet Project
## * 'news(package="ergm")' for changes since last version
## * 'citation("ergm")' for citation information
## * 'https://statnet.org' for help, support, and other information
## 'ergm' 4 is a major update that introduces some backwards-incompatible
## changes. Please type 'news(package="ergm")' for a list of major
## changes.

# Convirtiendo todos los objetos "igraph" en graphs a objetos "network"
graphs_network <- lapply(graphs, asNetwork)
```

Con los objetos de red listos, podemos proceder a calcular estadísticas a nivel de grafo usando la función `summary_formula`. Aquí solo veremos: el número de triángulos, homofilia de género, y homofilia de dieta saludable:

```r
net_stats_ergm <- lapply(graphs_network, function(n) {

  # Calculando las estadísticas
```

```
  s <- summary_formula(
    n ~ triangles +
      nodematch("gender_1") +
      nodematch("healthy_diet")
    )

  # Guardándolas como un objeto data.table
  data.table(
    triangles       = s[1],
    gender_homoph   = s[2],
    healthyd_homoph = s[3]
  )
})
```

Una vez más, usamos `rbindlist` para combinar todas las estadísticas de red en un solo objeto `data.table`:

```
net_stats_ergm <- rbindlist(net_stats_ergm)
head(net_stats_ergm)
##    triangles gender_homoph healthyd_homoph
##        <num>          <num>            <num>
## 1:        27             11                3
## 2:        40             30               20
## 3:        81             40               29
## 4:       216             33               38
## 5:        79             44               19
## 6:       121             38               16
```

6.6 Guardando los datos

Terminamos el capítulo guardando todo nuestro trabajo en cuatro conjuntos de datos:

- Estadísticas de red (como un archivo csv)
- Objetos igraph (como un archivo rda, que podemos leer de vuelta usando `read.rds`)

- Objetos de red (ídem)
- Archivos de persona (información de alters, como un archivo csv.)

Los archivos CSV pueden guardarse usando `write.csv` o, como hacemos aquí, `fwrite` del paquete `data.table`:

```
# Verificando que el directorio existe
if (!dir.exists("data"))
  dir.create("data")

# Atributos de red
master <- cbind(egos, net_stats, net_stats_ergm)
fwrite(master, file = "data/network_stats.csv")

# Redes
saveRDS(graphs, file = "data/networks_igraph.rds")
saveRDS(graphs_network, file = "data/networks_network.rds")

# Atributos
fwrite(persons, file = "data/persons.csv")
```

7 Difusión en Redes

> ⚠ Nota de Traducción
>
> Esta versión del capítulo fue traducida de manera automática utilizando IA. El capítulo aún no ha sido revisado por un humano.

Este capítulo está basado en los tutoriales de netdiffuseR de 2018 y 2019 en la conferencia Sunbelt. El código fuente de los tutoriales, enseñados por Thomas W. Valente y George G. Vega Yon (autor de este libro), está disponible aquí.

7.1 Difusión de innovaciones en redes

7.1.1 Redes de difusión

- Explica cómo las nuevas ideas y prácticas (innovaciones) se extienden dentro y entre comunidades.
- Aunque muchos factores han demostrado influir en la difusión (Espaciales, Económicos, Culturales, Biológicos, etc.), las Redes Sociales son prominentes.
- Hay muchos componentes en el modelo de red de difusión, incluyendo exposiciones de red, umbrales, infectividad, susceptibilidad, tasas de riesgo, tasas de difusión (modelo bass), agrupamiento (I de Moran), y así sucesivamente.

7.1.2 Umbrales

- Uno de los conceptos canónicos es el umbral de red. Los umbrales de red (Valente, 1995; 1996), τ, se definen como la proporción requerida o número de vecinos que te

llevan a adoptar un comportamiento particular (innovación), $a = 1$. En términos (muy) generales

$$a_i = \begin{cases} 1 & \text{if } \tau_i \leq E_i \\ 0 & \text{Otherwise} \end{cases} \qquad E_i \equiv \frac{\sum_{j \neq i} \mathbf{X}_{ij} a_j}{\sum_{j \neq i} \mathbf{X}_{ij}}$$

Donde E_i es la exposición de i a la innovación y $\mathbf{X}$ es la matriz de adyacencia (la red).

- Esto puede ser generalizado y extendido para incluir covariables y otros esquemas de ponderación de red (de eso se trata **netdiffuseR**).

7.2 El paquete de R netdiffuseR

7.2.1 Visión general

netdiffuseR es un paquete de R que:

- Está diseñado para Visualizar, Analizar, y simular datos de difusión de red (en general).
- Depende de algunos paquetes bastante populares:
  - *RcppArmadillo*: Así que es rápido,
  - *Matrix*: Así que es grande,
  - *statnet* e *igraph*: Así que no es desde cero
- Puede manejar grafos grandes, ej., una matriz de adyacencia con más de 4 mil millones de elementos (PR para RcppArmadillo)
- Ya está en CRAN con ~6,000 descargas desde su primera versión, Feb 2016,
- Muchas características para hacer fácil leer datos de red (dinámicos), haciéndolo un compañero de otros paquetes de redes.

7.2.2 Conjuntos de datos

- **netdiffuseR** tiene los tres conjuntos de datos clásicos de Redes de Difusión:
  - `medInnovationsDiffNet` Doctores y la innovación de Tetraciclina (1955).
  - `brfarmersDiffNet` Agricultores brasileños y la innovación de Semilla de Maíz Híbrido (1966).
  - `kfamilyDiffNet` Mujeres coreanas y métodos de Planificación Familiar (1973).

```
brfarmersDiffNet
```

```
Dynamic network of class -diffnet-
 Name               : Brazilian Farmers
 Behavior           : Adoption of Hybrid Corn Seeds
 # of nodes         : 692 (1001, 1002, 1004, 1005, 1007, 1009, 1010, 1020, ...)
 # of time periods  : 21 (1946 - 1966)
 Type               : directed
 Num of behaviors   : 1
 Final prevalence   : 1.00
 Static attributes  : village, idold, age, liveout, visits, contact, coo... (146)
 Dynamic attributes : -
```

```
medInnovationsDiffNet
```

```
Dynamic network of class -diffnet-
 Name               : Medical Innovation
 Behavior           : Adoption of Tetracycline
 # of nodes         : 125 (1001, 1002, 1003, 1004, 1005, 1006, 1007, 1008, ...)
 # of time periods  : 18 (1 - 18)
 Type               : directed
 Num of behaviors   : 1
 Final prevalence   : 1.00
 Static attributes  : city, detail, meet, coll, attend, proage, length, ... (58)
 Dynamic attributes : -
```

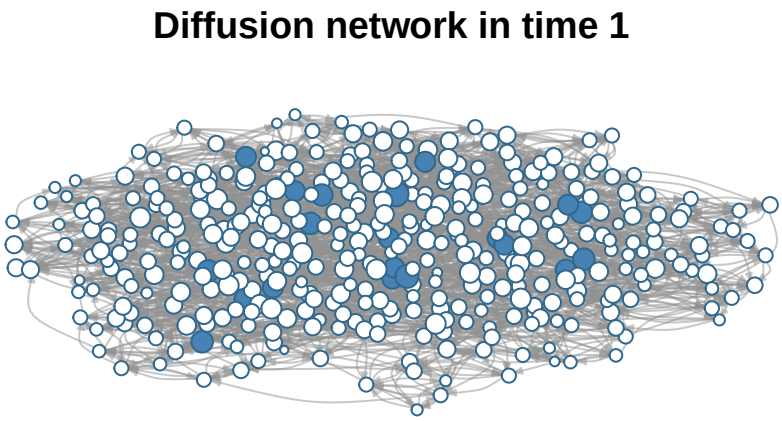
```
kfamilyDiffNet
```

```
Dynamic network of class -diffnet-
 Name               : Korean Family Planning
```

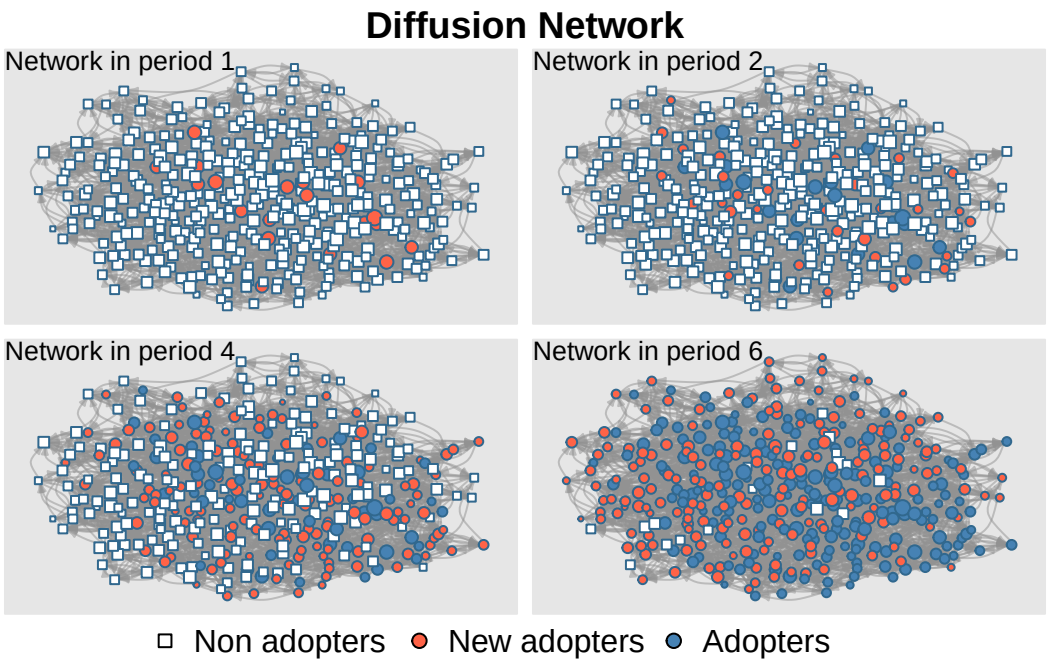
```
Behavior           : Family Planning Methods
# of nodes         : 1047 (10002, 10003, 10005, 10007, 10010, 10011, 10012, 10014,
# of time periods  : 11 (1 - 11)
Type               : directed
Num of behaviors   : 1
Final prevalence   : 1.00
Static attributes  : village, recno1, studno1, area1, id1, nmage1, nmag... (430)
Dynamic attributes : -
```

7.2.3 Métodos de visualización

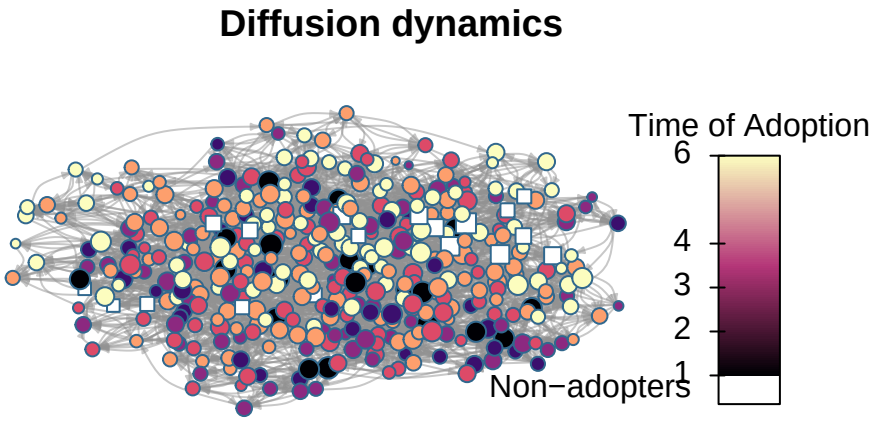
```
set.seed(12315)
x <- rdiffnet(
  400, t = 6, rgraph.args = list(k=6, p=.3),
  seed.graph = "small-world",
  seed.nodes = "central", rewire = FALSE, threshold.dist = 1/4
  )
plot(x)
```



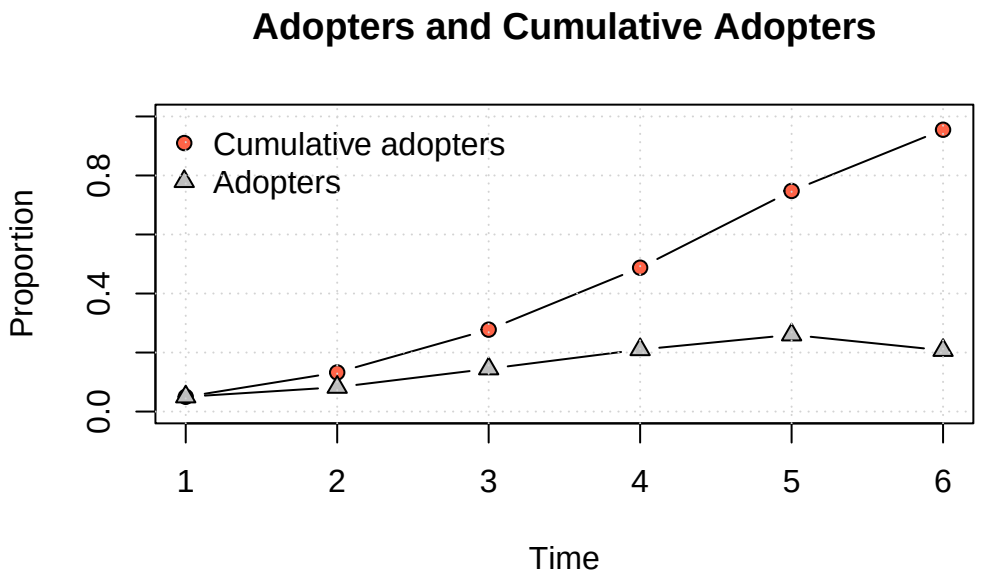
```
plot_diffnet(x)
```



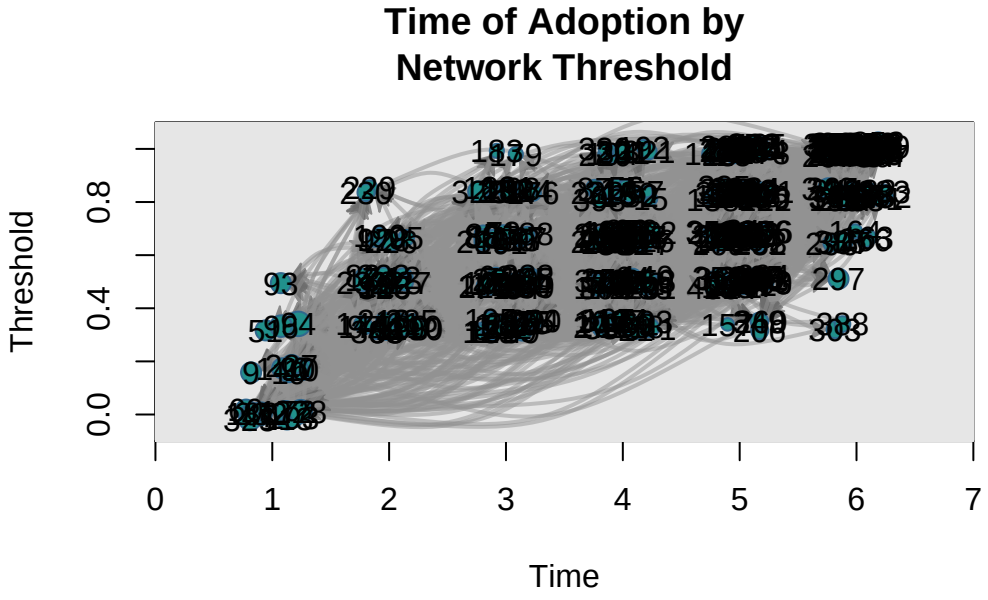

```
plot_diffnet2(x)
```



```
plot_adopters(x)
```

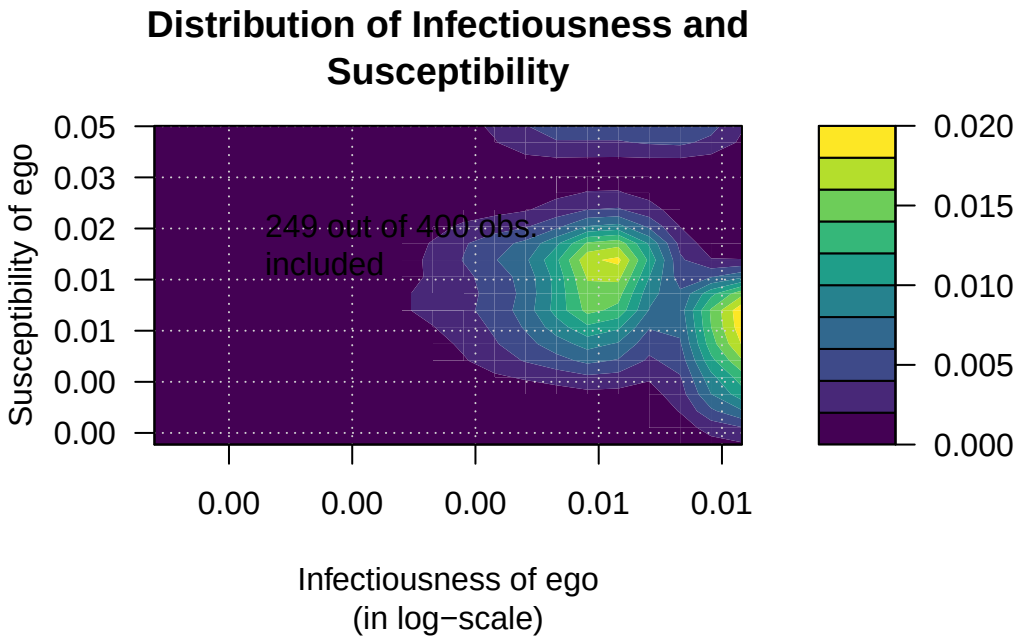



```
plot_threshold(x)
```

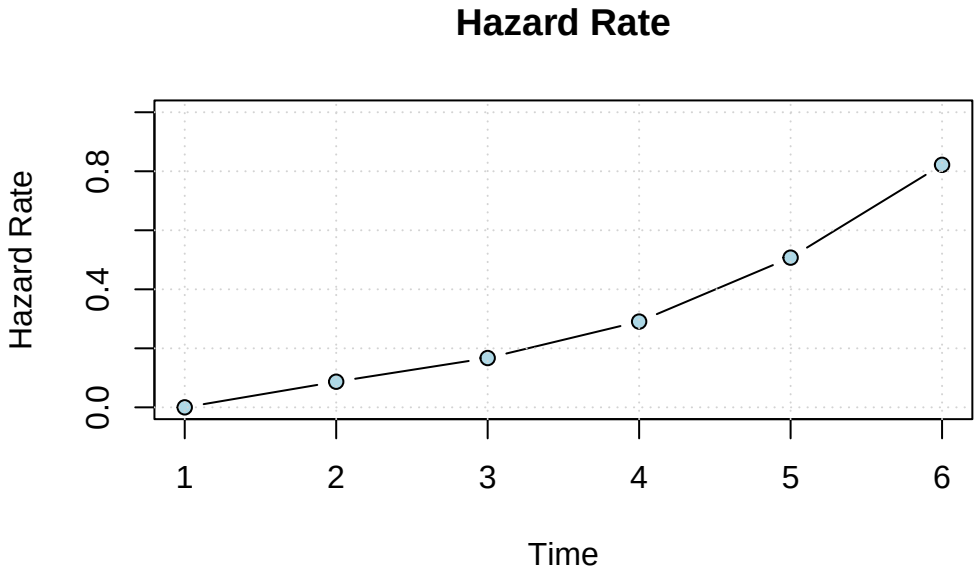


```
plot_infectsuscep(x, K=2)
```

```
Warning in plot_infectsuscep.list(graph$graph, graph$toa, t0, normalize, : When
applying logscale some observations are missing.
```




```
plot_hazard(x)
```



7.2.4 Problemas

1. Usando el objeto diffnet en `intro.rda`, usa la función `plot_threshold` especificando formas y colores según las variables ItrustMyFriends y Age. ¿Ves algún patrón?

7.3 Simulación de procesos de difusión

Antes de comenzar, una revisión de los conceptos que usaremos aquí

1. Exposición: Proporción/número de vecinos que han adoptado una innovación en cada punto en el tiempo.
2. Umbral: La proporción/número de tus vecinos que habían adoptado en o un período de tiempo antes de que ego (el individuo focal) adoptara.
3. Infectividad: Cuánto la adopción de i afecta a sus alters.
4. Susceptibilidad: Cuánto la adopción de los alters de i la afecta.
5. Equivalencia estructural: Qué tan similar es i a j en términos de posición en la red.

7.3.1 Simulando redes de difusión

Simularemos una red de difusión con los siguientes parámetros:

1. Tendrá 1,000 vértices,
2. Abarcará 20 períodos de tiempo,
3. Los adoptadores iniciales (semillas) serán seleccionados al azar,
4. Las semillas serán un 10% de la red,
5. El grafo (red) será mundo pequeño,
6. Usará el algoritmo WS con $p = .2$ (probabilidad de reconexión).
7. Los niveles de umbral serán distribuidos uniformemente entre [0.3, 0.7]

Para generar esta red de difusión, podemos usar la función `rdiffnet` incluida en el paquete:

```
# Setting the seed for the RNG
set.seed(1213)

# Generating a random diffusion network
```

```
net <- rdiffnet(
  n              = 1e3,                          # 1.
  t              = 20,                           # 2.
  seed.nodes     = "random",                     # 3.
  seed.p.adopt   = .1,                           # 4.
  seed.graph     = "small-world",                # 5.
  rgraph.args    = list(p=.2),                   # 6.
  threshold.dist = function(x) runif(1, .3, .7) # 7.
  )
```

- La función `rdiffnet` genera redes de difusión aleatorias. Características principales:

  1. Simulando grafo aleatorio o usando el tuyo propio,
  2. Estableciendo niveles de umbral por nodo,
  3. Reconexión de red a lo largo de la simulación, y
  4. Estableciendo los nodos semilla.

- El algoritmo de simulación es el siguiente:

  1. Si se requiere, se crea un grafo de referencia,
  2. Se establece el conjunto de adoptadores iniciales y distribución de umbral,
  3. Se crea el conjunto de t redes (si se requiere), y
  4. La simulación comienza en t=2, asignando adoptadores basándose en exposiciones y umbrales:

  a. Para cada $i \in N$, si su exposición en $t-1$ es mayor que su umbral, entonces adopta, de otra manera, continúa sin cambio.

  b. siguiente i

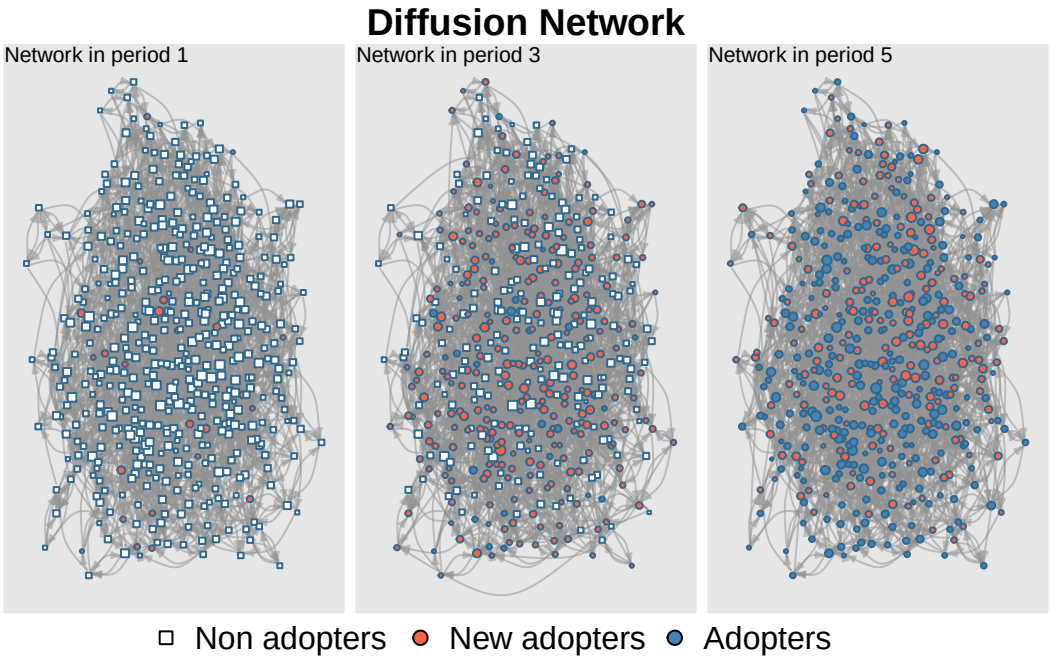
7.3.2 Esparcimiento de rumores

```r
library(netdiffuseR)

set.seed(09)
diffnet_rumor <- rdiffnet(
  n = 5e2,
  t = 5,
  seed.graph = "small-world",
  rgraph.args = list(k = 4, p = .3),
  seed.nodes = "random",
  seed.p.adopt = .05,
  rewire = TRUE,
  threshold.dist = function(i) 1L,
  exposure.args = list(normalized = FALSE)
  )
```

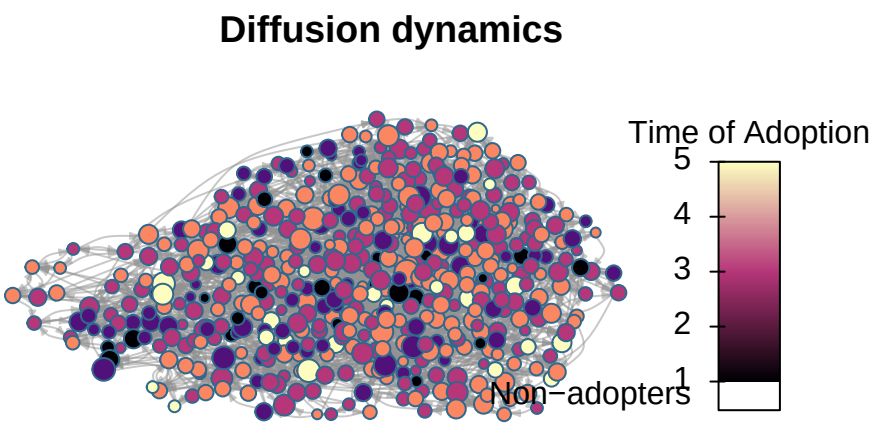
```r
summary(diffnet_rumor)
```

```
Diffusion network summary statistics
 Name     :  A diffusion network
 Behavior : Random contagion
------------------------------------------------------------------------------
 Period   Adopters   Cum Adopt. (%)   Hazard Rate   Density   Moran's I (sd)
-------- ---------- ---------------- ------------- --------- ----------------
      1         25        25 (0.05)             -      0.01 -0.00 (0.00)
      2         78       103 (0.21)          0.16      0.01  0.01 (0.00) ***
      3        187       290 (0.58)          0.47      0.01  0.01 (0.00) ***
      4        183       473 (0.95)          0.87      0.01  0.01 (0.00) ***
      5         27       500 (1.00)          1.00      0.01                -
------------------------------------------------------------------------------
 Left censoring  : 0.05 (25)
 Right centoring : 0.00 (0)
 # of nodes      : 500

 Moran's I was computed on contemporaneous autocorrelation using 1/geodesic
 values. Significane levels  *** <= .01, ** <= .05, * <= .1.
```

```
plot_diffnet(diffnet_rumor, slices = c(1, 3, 5))
```



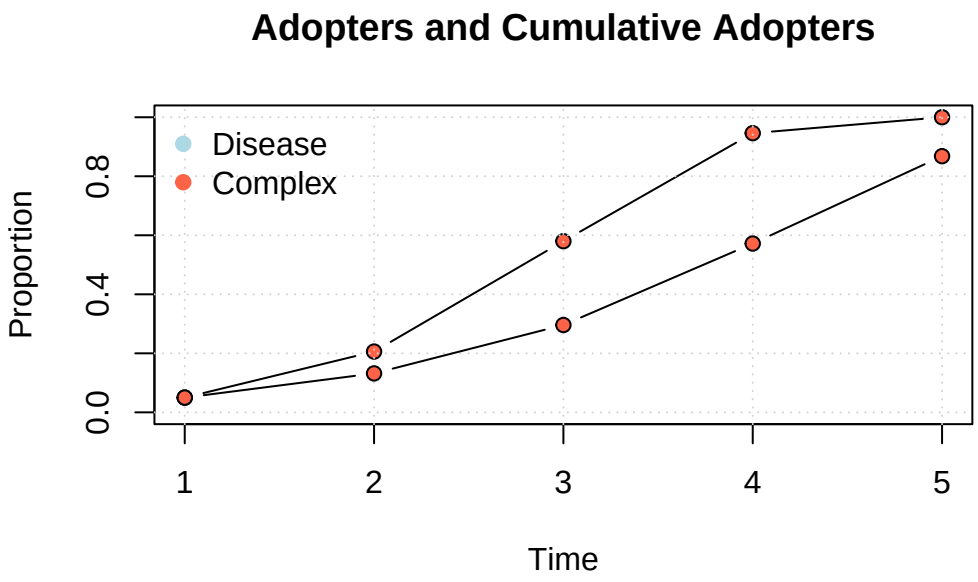

```
# We want to use igraph to compute layout
igdf <- diffnet_to_igraph(diffnet_rumor, slices=c(1,2))[[1]]
pos <- igraph::layout_with_drl(igdf)

plot_diffnet2(diffnet_rumor, vertex.size = dgr(diffnet_rumor)[,1], layout=pos)
```



7.3.3 Difusión

```
set.seed(09)
diffnet_complex <- rdiffnet(
  seed.graph = diffnet_rumor$graph,
  seed.nodes = which(diffnet_rumor$toa == 1),
  rewire = FALSE,
  threshold.dist = function(i) rbeta(1, 3, 10),
  name = "Diffusion",
  behavior = "Some social behavior"
)
```

```
plot_adopters(diffnet_rumor, what = "cumadopt", include.legend = FALSE)
plot_adopters(diffnet_complex, bg="tomato", add=TRUE, what = "cumadopt")
legend("topleft", legend = c("Disease", "Complex"), col = c("lightblue", "tomato"),
       bty = "n", pch=19)
```



7.3.4 Emparejamiento de mentores

```
# Finding mentors
mentors <- mentor_matching(diffnet_rumor, 25, lead.ties.method = "random")

# Simulating diffusion with these mentors
set.seed(09)
diffnet_mentored <- rdiffnet(
  seed.graph = diffnet_complex,
  seed.nodes = which(mentors$`1`$isleader),
  rewire = FALSE,
  threshold.dist = diffnet_complex[["real_threshold"]],
  name = "Diffusion using Mentors"
)

summary(diffnet_mentored)
```

```
Diffusion network summary statistics
 Name      :  Diffusion using Mentors
```

```
 Behavior : Random contagion
-------------------------------------------------------------------------------
 Period   Adopters   Cum Adopt. (%)   Hazard Rate   Density   Moran's I (sd)
-------- ---------- ---------------- ------------ --------- ----------------
      1         25        25 (0.05)            -      0.01 -0.00 (0.00)
      2         93       118 (0.24)         0.20      0.01  0.01 (0.00) ***
      3        155       273 (0.55)         0.41      0.01  0.01 (0.00) ***
      4        162       435 (0.87)         0.71      0.01  0.01 (0.00) ***
      5         62       497 (0.99)         0.95      0.01 -0.00 (0.00)
-------------------------------------------------------------------------------
 Left censoring  : 0.05 (25)
 Right centoring : 0.01 (3)
 # of nodes      : 500

 Moran's I was computed on contemporaneous autocorrelation using 1/geodesic
 values. Significane levels  *** <= .01, ** <= .05, * <= .1.
```

```
cumulative_adopt_count(diffnet_complex)
```

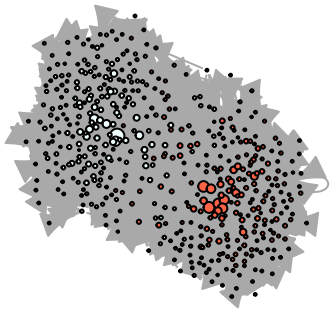
```
         1      2          3           4           5
num  25.00 66.000 148.000000 286.0000000 434.0000000
prop  0.05  0.132   0.296000   0.5720000   0.8680000
rate  0.00  1.640   1.242424   0.9324324   0.5174825
```

```
cumulative_adopt_count(diffnet_mentored)
```

```
         1       2          3           4           5
num  25.00 118.000 273.000000 435.0000000 497.0000000
prop  0.05   0.236   0.546000   0.8700000   0.9940000
rate  0.00   3.720   1.313559   0.5934066   0.1425287
```

7.3.5 Ejemplo cambiando umbral

```r
# Simulating a scale-free homophilic network
set.seed(1231)
X <- rep(c(1,1,1,1,1,0,0,0,0,0), 50)
net <- rgraph_ba(t = 499, m=4, eta = X)

# Taking a look in igraph
ig  <- igraph::graph_from_adjacency_matrix(net)
plot(ig, vertex.color = c("azure", "tomato")[X+1], vertex.label = NA,
     vertex.size = sqrt(dgr(net)))
```



```r
# Now, simulating a bunch of diffusion processes
nsim <- 500L
ans_1and2 <- vector("list", nsim)
set.seed(223)
for (i in 1:nsim) {
  # We just want the cum adopt count
  ans_1and2[[i]] <-
    cumulative_adopt_count(
      rdiffnet(
```

```
        seed.graph = net,
        t = 10,
        threshold.dist = sample(1:2, 500L, TRUE),
        seed.nodes = "random",
        seed.p.adopt = .10,
        exposure.args = list(outgoing = FALSE, normalized = FALSE),
        rewire = FALSE
        )
      )

  # Are we there yet?
  if (!(i %% 50))
    message("Simulation ", i," of ", nsim, " done.")
}
## Simulation 50 of 500 done.
## Simulation 100 of 500 done.
## Simulation 150 of 500 done.
## Simulation 200 of 500 done.
## Simulation 250 of 500 done.
## Simulation 300 of 500 done.
## Simulation 350 of 500 done.
## Simulation 400 of 500 done.
## Simulation 450 of 500 done.
## Simulation 500 of 500 done.

# Extracting prop
ans_1and2 <- do.call(rbind, lapply(ans_1and2, "[", i="prop", j=))

ans_2and3 <- vector("list", nsim)
set.seed(223)
for (i in 1:nsim) {
  # We just want the cum adopt count
  ans_2and3[[i]] <-
    cumulative_adopt_count(
      rdiffnet(
        seed.graph = net,
```

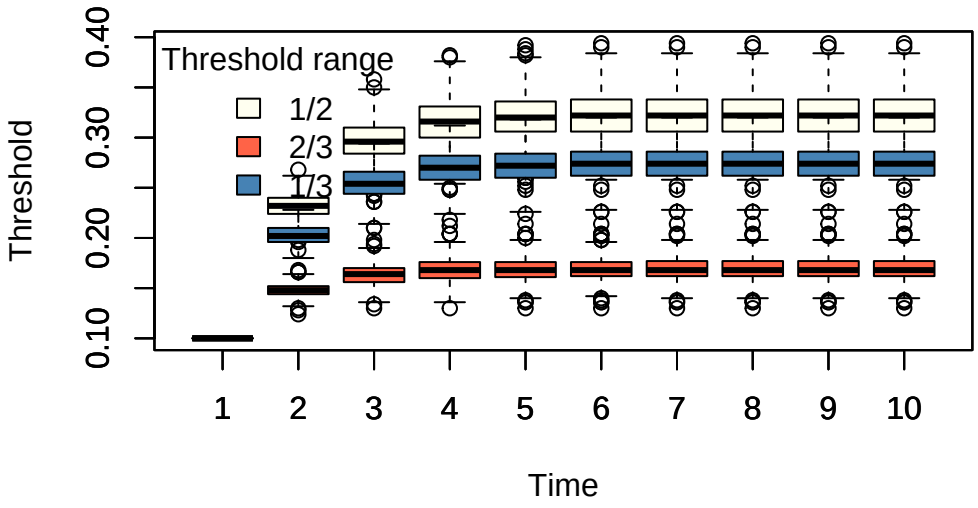
```
        t = 10,
        threshold.dist = sample(2:3, 500L, TRUE),
        seed.nodes = "random",
        seed.p.adopt = .10,
        exposure.args = list(outgoing = FALSE, normalized = FALSE),
        rewire = FALSE
        )
      )

  # Are we there yet?
  if (!(i %% 50))
    message("Simulation ", i," of ", nsim, " done.")
}
## Simulation 50 of 500 done.
## Simulation 100 of 500 done.
## Simulation 150 of 500 done.
## Simulation 200 of 500 done.
## Simulation 250 of 500 done.
## Simulation 300 of 500 done.
## Simulation 350 of 500 done.
## Simulation 400 of 500 done.
## Simulation 450 of 500 done.
## Simulation 500 of 500 done.

ans_2and3 <- do.call(rbind, lapply(ans_2and3, "[", i="prop", j=))
```

Podemos simplificar usando la función `rdiffnet_multiple`. Las siguientes líneas de código logran lo mismo que el código anterior evitando el bucle for (desde la perspectiva del usuario). Además de los parámetros usuales pasados a `rdiffnet`, la función `rdiffnet_multiple` requiere `R` (número de repeticiones/simulaciones), y `statistic` (una función que devuelve la estadística de interés). Opcionalmente, el usuario puede elegir especificar el número de clusters para ejecutarlo en paralelo (múltiples CPUs):

```
ans_1and3 <- rdiffnet_multiple(
  # Num of sim
  R              = nsim,
```

```
  # Statistic
  statistic      = function(d) cumulative_adopt_count(d)["prop",],
  seed.graph     = net,
  t              = 10,
  threshold.dist = sample(1:3, 500, TRUE),
  seed.nodes     = "random",
  seed.p.adopt   = .1,
  rewire         = FALSE,
  exposure.args  = list(outgoing=FALSE, normalized=FALSE),
  # Running on 4 cores
  ncpus          = 4L
  )
```

```
boxplot(ans_1and2, col="ivory", xlab = "Time", ylab = "Threshold")
boxplot(ans_2and3, col="tomato", add=TRUE)
boxplot(t(ans_1and3), col = "steelblue", add=TRUE)
legend(
  "topleft",
  fill = c("ivory", "tomato", "steelblue"),
  legend = c("1/2", "2/3", "1/3"),
  title = "Threshold range",
  bty ="n"
)
```



7.3.6 Problemas

1. Dados los siguientes tipos de redes: Mundo pequeño, Libre de escala, Bernoulli, ¿qué conjunto de n iniciadores maximiza la difusión?

7.4 Inferencia estadística

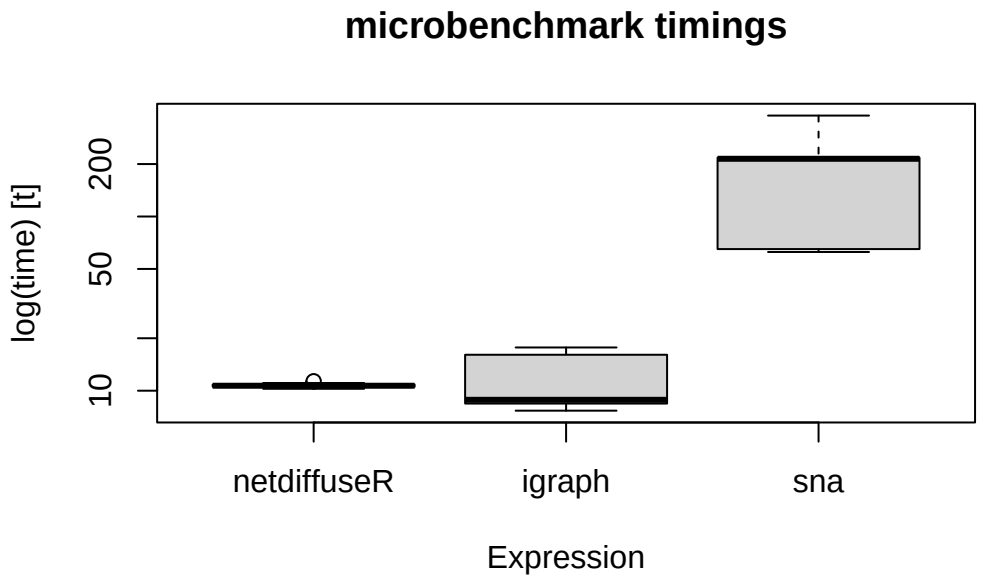
7.4.1 I de Moran

- La I de Moran prueba la autocorrelación espacial.
- **netdiffuseR** implementa la prueba en `moran`, que es adecuada para matrices dispersas.
- Podemos usar la I de Moran como una primera mirada para ver si algo está pasando: ya sea influencia u homofilia.

7.4.2 Usando geodésicas

- Un enfoque es usar la matriz geodésica (longitud de camino más corto) para considerar influencia indirecta.
- **netdiffuseR** tiene una función para hacerlo, la función `approx_geodesic`, que, usando potencias de grafo, calcula el camino más corto hasta `n` pasos. Esto podría ser más rápido (si solo te importa hasta `n` pasos) que `igraph` o `sns`:

```
# Extracting the large adjacency matrix (stacked)
dgc <- diag_expand(medInnovationsDiffNet$graph)
ig  <- igraph::graph_from_adjacency_matrix(dgc)
mat <- network::as.network(as.matrix(dgc))

# Measuring times
times <- microbenchmark::microbenchmark(
  netdiffuseR = netdiffuseR::approx_geodesic(dgc),
  igraph = igraph::distances(ig),
  sna = sna::geodist(mat),
  times = 50, unit="ms"
)
```



- El método `summary.diffnet` ya ejecuta Moran para ti. Lo que pasa bajo el capó es:

```r
# For each time point we compute the geodesic distances matrix
W <- approx_geodesic(medInnovationsDiffNet$graph[[1]])

# We get the element-wise inverse
W@x <- 1/W@x

# And then compute moran
moran(medInnovationsDiffNet$cumadopt[,1], W)
```

```
$observed
[1] 0.06624028

$expected
[1] -0.008064516

$sd
[1] 0.03265066

$p.value
[1] 0.02286087

attr(,"class")
[1] "diffnet_moran"
```

7.4.3 Dependencia estructural y pruebas de permutación

- Un método estadístico novedoso (trabajo en progreso) para probar efectos de influencia de red.
- Incluido en el paquete, prueba si una estadística de red particular depende de la estructura de red
- Adecuado para ser aplicado a umbrales de red (¡no puedes usar umbrales en modelos tipo regresión!)

7.4.4 Idea

- Sea $\mathcal{G} = (V, E)$ un grafo, γ un atributo de vértice, y $\beta = f(\gamma, \mathcal{G})$, entonces

$$\gamma \perp \mathcal{G} \implies \mathbb{E}\left[\beta(\gamma, \mathcal{G})|\mathcal{G}\right] = \mathbb{E}\left[\beta(\gamma, \mathcal{G})\right]$$

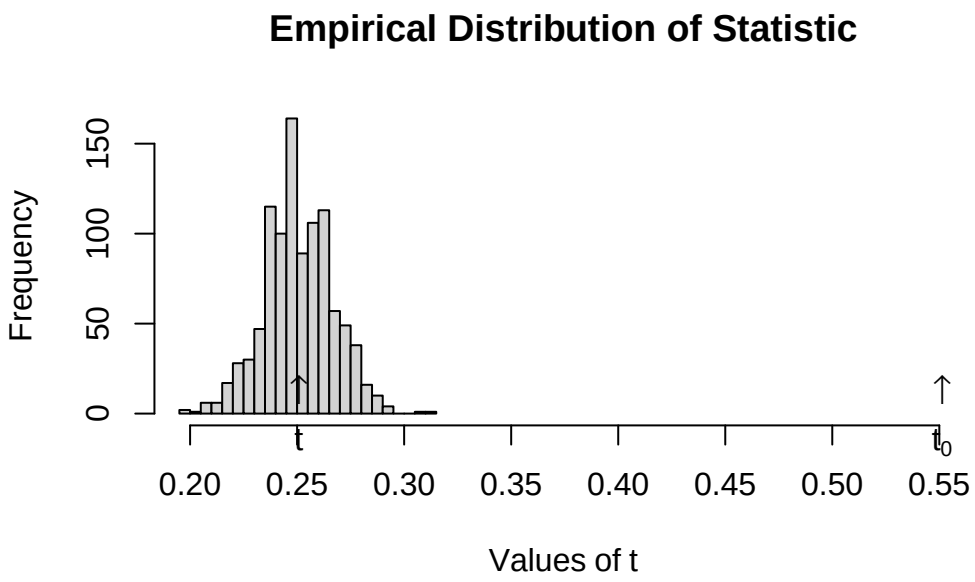
- Por ejemplo, si el tiempo de adopción es independiente de la estructura de la red, entonces el nivel de umbral promedio será independiente de la estructura de red también.
- Otra manera de ver esto es que la prueba nos permitirá ver qué tan probable es tener esta combinación de estructura de red y umbral de red (si es poco común, entonces decimos que el modelo de difusión es altamente probable)

7.4.4.1 Ejemplo TOA no aleatorio

- Para usar esta prueba, **netdiffuseR** tiene la función `struct_test`.
- Simula redes con la misma densidad, y calcula una estadística particular cada vez, generando una EDF (Función de Distribución Empírica) bajo la hipótesis Nula (valores p).

```
# Simulating network
set.seed(1123)
net <- rdiffnet(n=500, t=10, seed.graph = "small-world")

# Running the test
test <- struct_test(
  graph     = net,
  statistic = function(x) mean(threshold(x), na.rm = TRUE),
  R         = 1e3,
  ncpus=4, parallel="multicore"
  )

# See the output
test
```

```
Structure dependence test
# Simulations     : 1,000
# nodes           : 500
# of time periods : 10
--------------------------------------------------------------------------------
 H0: E[beta(Y,G)|G] - E[beta(Y,G)] = 0 (no structure dependency)
    observed    expected       p.val
      0.5513      0.2508      0.0000
```

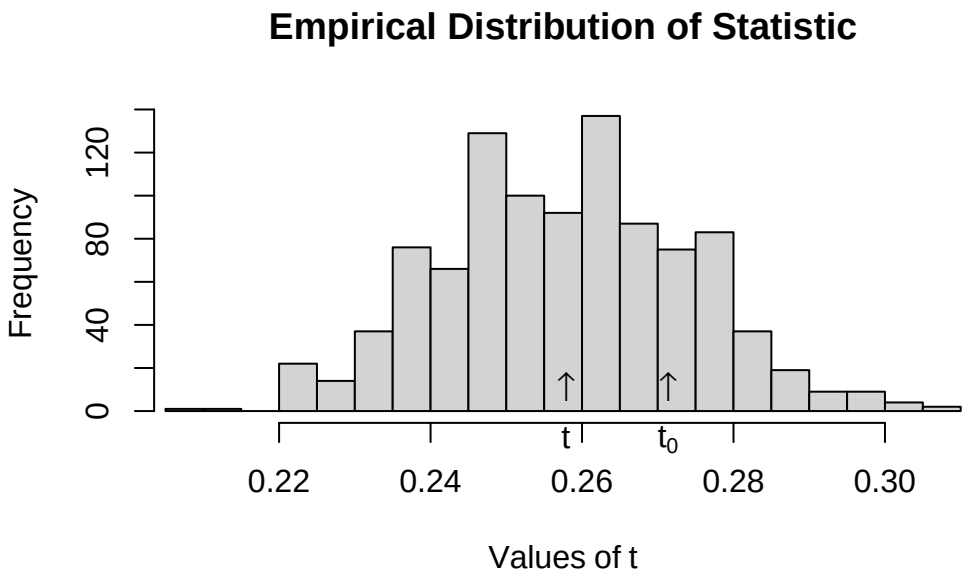


- Ahora mezclamos tiempos de adopción, para que sea aleatorio

```r
# Resetting TOAs (now will be completely random)
diffnet.toa(net) <- sample(diffnet.toa(net), nnodes(net), TRUE)

# Running the test
test <- struct_test(
  graph     = net,
  statistic = function(x) mean(threshold(x), na.rm = TRUE),
  R         = 1e3,
  ncpus=4, parallel="multicore"
  )
```

```
# See the output
test
```

```
Structure dependence test
# Simulations     : 1,000
# nodes           : 500
# of time periods : 10
--------------------------------------------------------------------------------
 H0: E[beta(Y,G)|G] - E[beta(Y,G)] = 0 (no structure dependency)
    observed    expected       p.val
      0.2714      0.2579      0.3980
```



7.4.5 Análisis de regresión

- En análisis de regresión, queremos ver si la exposición, una vez que controlamos por otras covariables, tuvo algún efecto en adoptar un comportamiento.
- El gran problema es cuando tenemos una variable latente que co-determina tanto red como comportamiento.

- El análisis de regresión será genéricamente sesgado A menos que podamos controlar por esa variable.
- Por otro lado, si puedes afirmar que tal variable no existe o que realmente puedes controlar por ella, entonces tenemos dos opciones: modelos de exposición retrasada o modelos de exposición contemporánea. Nos enfocaremos en los primeros.

7.4.5.1 Modelos de exposición retrasada

- En este tipo de modelo, usualmente tenemos lo siguiente

$$y_t = f(W_{t-1}, y_{t-1}, X_i) + \varepsilon$$

Además, en el caso de adopción, tenemos

$$y_{it} = \begin{cases} 1 & \text{if} \quad \rho \sum_{j \neq i} \frac{W_{ijt-1} y_{it-1}}{\sum_{j \neq i} W_{ijt-1}} + X_{it}\beta > 0 \\ 0 & \text{otherwise} \end{cases}$$

- En netdiffuseR, es tan fácil como hacer lo siguiente:

```
# fakedata
set.seed(121)

W   <- rgraph_ws(1e3, 8, .2)
X   <- cbind(var1 = rnorm(1e3))
toa <- sample(c(NA,1:5), 1e3, TRUE)

dn  <- new_diffnet(W, toa=toa, vertex.static.attrs = X)
```

```
Warning in new_diffnet(W, toa = toa, vertex.static.attrs = X): -graph- is
static and will be recycled (see ?new_diffnet).
```

```
# Computing exposure and adoption for regression
dn[["cohesive_expo"]] <- cbind(NA, exposure(dn)[,-nslices(dn)])
dn[["adopt"]]         <- dn$cumadopt

# Generating the data and running the model
```

```r
dat <- as.data.frame(dn)
ans <- glm(adopt ~ cohesive_expo + var1 + factor(per),
           data = dat,
           family = binomial(link="probit"),
           subset = is.na(toa) | (per <= toa))
summary(ans)
```

```
Call:
glm(formula = adopt ~ cohesive_expo + var1 + factor(per), family = binomial(link =
    data = dat, subset = is.na(toa) | (per <= toa))

Coefficients:
              Estimate Std. Error z value Pr(>|z|)
(Intercept)   -0.92777    0.05840 -15.888  < 2e-16 ***
cohesive_expo  0.23839    0.17514   1.361 0.173452
var1          -0.04623    0.02704  -1.710 0.087334 .
factor(per)3   0.29313    0.07715   3.799 0.000145 ***
factor(per)4   0.33902    0.09897   3.425 0.000614 ***
factor(per)5   0.59851    0.12193   4.909 9.18e-07 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 2745.1  on 2317  degrees of freedom
Residual deviance: 2663.5  on 2312  degrees of freedom
  (1000 observations deleted due to missingness)
AIC: 2675.5

Number of Fisher Scoring iterations: 4
```

Alternativamente, podríamos haber usado la nueva función `diffreg`

```r
ans <- diffreg(dn ~ exposure + var1 + factor(per), type = "probit")
summary(ans)
```

```
Call:
glm(formula = Adopt ~ exposure + var1 + factor(per), family = binomial(link = "prob
    data = dat, subset = ifelse(is.na(toa), TRUE, toa >= per))

Coefficients:
             Estimate Std. Error z value Pr(>|z|)
(Intercept)  -0.92777    0.05840 -15.888  < 2e-16 ***
exposure      0.23839    0.17514   1.361 0.173452
var1         -0.04623    0.02704  -1.710 0.087334 .
factor(per)3  0.29313    0.07715   3.799 0.000145 ***
factor(per)4  0.33902    0.09897   3.425 0.000614 ***
factor(per)5  0.59851    0.12193   4.909 9.18e-07 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 2745.1  on 2317  degrees of freedom
Residual deviance: 2663.5  on 2312  degrees of freedom
  (1000 observations deleted due to missingness)
AIC: 2675.5

Number of Fisher Scoring iterations: 4
```

7.4.5.2 Modelos de exposición contemporánea

- Similar a los modelos de exposición retrasada, usualmente tenemos lo siguiente

$$y_t = f(W_t, y_t, X_t) + \varepsilon$$

Además, en el caso de adopción, tenemos

$$y_{it} = \begin{cases} 1 & \text{if} \quad \rho \sum_{j \neq i} \frac{W_{ijt} y_{it}}{\sum_{j \neq i} W_{ijt}} + X_{it}\beta > 0 \\ 0 & \text{otherwise} \end{cases}$$

- Desafortunadamente, dado que y_t está en ambos lados de la ecuación, estos modelos no pueden ser ajustados usando una regresión probit o logit estándar.
- Dos alternativas para resolver esto:
    a. Usando Probit de Variables Instrumentales (ivprobit tanto en R como en Stata)
    b. Usar un Probit Autorregresivo Espacial (SAR) (SpatialProbit y ProbitSpatial en R).
- No cubriremos estos aquí.

7.4.6 Problemas

Usando el conjunto de datos stats.rda:

1. Calcula la I de Moran como lo hace la función `summary.diffnet`. Para hacerlo, necesitarás usar la función `toa_mat` (que calcula la matriz de adopción acumulativa), y `approx_geodesic` (que calcula la matriz geodésica). (ver `?summary.diffnet` para más detalles).
2. Lee los datos como objeto diffnet, y ajusta el siguiente modelo logit $adopt = Exposure * \gamma + Measure * \beta + \varepsilon$. ¿Qué pasa si excluyes los efectos fijos de tiempo?

Part II

Inferencia estadística

8 Comportamiento y coevolución

> **Nota de Traducción**
>
> Esta versión del capítulo fue traducida de manera automática utilizando IA. El capítulo aún no ha sido revisado por un humano.

> **Note**
>
> Todo el contenido de esta sección fue presentado durante la Escuela de Verano Sistemas Complejos 2024 en la Universidad del Desarrollo. La versión original se puede encontrar aquí.

8.1 Introducción

Esta sección se enfoca en la inferencia que involucra redes y un resultado secundario. Aunque hay muchas formas de estudiar la coevolución o dependencia entre red y comportamiento, esta sección se enfoca en dos clases de análisis: cuando la red es fija y cuando tanto la red como el comportamiento se influyen mutuamente.

Si tratamos la red como dada o endógena establece la complejidad de realizar inferencia estadística. El análisis de datos se vuelve mucho más directo si nuestra investigación se enfoca en resultados a nivel individual embebidos en una red y no en la red misma. Aquí, trataremos con tres casos particulares: (1) cuando los efectos de red son rezagados, (2) redes egocéntricas, y (3) cuando los efectos de red son contemporáneos.

8.2 Exposición rezagada

Si asumimos que la influencia de la red en forma de exposición está rezagada, tenemos uno de los casos más directos para la inferencia de redes (Haye et al. 2019; Valente and Vega Yon 2020; Valente, Wipfli, and Vega Yon 2019). Aquí, en lugar de tratar con modelos estadísticos complicados, el problema se reduce a estimar un modelo de regresión lineal simple. Generalmente, los efectos de exposición rezagada se ven así:

$$y_{it} = \rho \left(\sum_{j \neq i} X_{ij} \right)^{-1} \left(\sum_{j \neq i} y_{jt-1} X_{ij} \right) + \theta^{\mathbf{t}} w_i + \varepsilon, \quad \varepsilon \sim \mathrm{N}(0, \sigma^2)$$

donde y_{it} es el resultado del individuo i en el tiempo t, X_{ij} es la entrada ij-ésima de la matriz de adyacencia, θ es un vector de coeficientes, w_i es un vector de características/covariables del individuo i, y ε_i es un error distribuido normalmente. Aquí, el componente clave es ρ: el coeficiente asociado con el efecto de exposición de red.

La estadística de exposición, $\left(\sum_{j \neq i} X_{ij}\right)^{-1} \left(\sum_{j \neq i} y_{jt-1} X_{ij}\right)$, es el promedio ponderado de los resultados de los vecinos de i en el tiempo $t-1$.

8.3 Ejemplo de código: Exposición rezagada

El siguiente ejemplo de código muestra cómo estimar un efecto de exposición rezagada usando la función `glm` en R. El modelo que simularemos y estimaremos presenta un grafo de Bernoulli con 1,000 nodos y una densidad de 0.01.

$$y_{it} = \theta_1 + \rho \mathrm{Exposure}_{i,t-1} + \theta_2 w_i + \varepsilon$$

donde $\mathrm{Exposure}_{i,t-1}$ es la estadística de exposición definida arriba, y w_i es un vector de covariables.

```
## Simulando datos
n <- 1000
time <- 2
theta <- c(-1, 3)
```

```r
## Muestreando una red de Bernoulli
set.seed(3132)
p <- 0.01
X <- matrix(rbinom(n^2, 1, p), nrow = n)
diag(X) <- 0

## Covariable
W <- matrix(rnorm(n), nrow = n)

## Simulando el resultado
rho <- 0.5
Y0 <- cbind(rnorm(n))

## La exposición rezagada
expo <- (X %*% Y0)/rowSums(X)
Y1 <- theta[1] + rho * expo + W * theta[2] + rnorm(n)
```

Ahora ajustamos el modelo usando GLM, en este caso, regresión lineal

```r
fit <- glm(Y1 ~ expo + W, family = "gaussian")
summary(fit)
```

```
Call:
glm(formula = Y1 ~ expo + W, family = "gaussian")

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) -1.07187    0.03284 -32.638  < 2e-16 ***
expo         0.61170    0.10199   5.998  2.8e-09 ***
W            3.00316    0.03233  92.891  < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for gaussian family taken to be 1.071489)
```

```
    Null deviance: 10319.3  on 999  degrees of freedom
Residual deviance:  1068.3  on 997  degrees of freedom
AIC: 2911.9

Number of Fisher Scoring iterations: 2
```

8.4 Redes egocéntricas

Generalmente, cuando usamos redes egocéntricas y resultados de los egos, estamos pensando en un modelo donde una observación es el par (y_i, X_i), esto es, el resultado del individuo i y la red egocéntrica del individuo i. Cuando tal es el caso, dado que (a) las redes son independientes entre egos y (b) las redes son fijas, como el caso anterior, un modelo de regresión lineal simple es suficiente para realizar los análisis. Un modelo típico se ve así:

$$y = \theta_x^{\mathbf{t}} s(X) + \theta^{\mathbf{t}} w + \varepsilon, \quad \varepsilon \sim \mathrm{N}(0, \sigma^2)$$

Donde y es un vector de resultados, X es una matriz de redes egocéntricas, w es un vector de covariables, θ es un vector de coeficientes, y ε es un vector de errores. El componente clave aquí es $s(X)$, que es un vector de estadísticas suficientes de las redes egocéntricas. Por ejemplo, si estamos interesados en el número de vínculos, $s(X)$ es un vector del número de vínculos de cada ego.

8.5 Ejemplo de código: Redes egocéntricas

Para este ejemplo, simularemos un flujo de 1,000 grafos de Bernoulli analizando la probabilidad de deserción escolar. Cada red tendrá entre 4 y 10 nodos y tendrá una densidad de 0.4. El proceso de generación de datos es el siguiente:

$$\Pr()_\theta \left(Y_i = 1\right) = \mathrm{logit}^{-1}\left(\theta_x s(X_i)\right)$$

Donde $s(X) \equiv (\text{densidad}, \text{n vínculos mutuos})$, y $\theta_x = (0.5, -1)$. Este modelo solo presenta estadísticas suficientes. Comenzamos simulando las redes

```r
set.seed(331)
n <- 1000
sizes <- sample(4:10, n, replace = TRUE)

## Simulando las redes
X <- lapply(sizes, function(x) matrix(rbinom(x^2, 1, 0.4), nrow = x))
X <- lapply(X, \(x) {diag(x) <- 0; x})

## Inspeccionando las primeras 5
head(X, 5)
```

```
[[1]]
     [,1] [,2] [,3] [,4] [,5]
[1,]    0    1    1    1    0
[2,]    0    0    0    0    0
[3,]    0    1    0    0    0
[4,]    0    0    0    0    0
[5,]    1    0    0    1    0

[[2]]
     [,1] [,2] [,3] [,4]
[1,]    0    0    0    0
[2,]    0    0    0    0
[3,]    0    0    0    0
[4,]    1    0    1    0

[[3]]
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    0    1    0    1    0    0
[2,]    0    0    0    0    0    0
[3,]    0    1    0    0    0    1
[4,]    0    0    0    0    1    0
[5,]    0    0    0    0    0    0
[6,]    0    0    0    0    0    0

[[4]]
```

```
     [,1] [,2] [,3] [,4] [,5]
[1,]    0    1    0    1    0
[2,]    0    0    0    0    1
[3,]    0    1    0    0    0
[4,]    0    1    1    0    1
[5,]    1    0    1    0    0

[[5]]
     [,1] [,2] [,3] [,4] [,5]
[1,]    0    0    0    0    0
[2,]    1    0    0    0    0
[3,]    1    0    0    0    0
[4,]    0    0    0    0    0
[5,]    1    0    0    0    0
```

Usando el paquete de R `ergm` (Handcock et al. 2023; David R. Hunter et al. 2008), podemos extraer las estadísticas suficientes asociadas de las redes egocéntricas:

```
library(ergm)
stats <- lapply(X, \(x) summary_formula(x ~ density + mutual))

## Convirtiendo la lista en una matriz
stats <- do.call(rbind, stats)

## Inspeccionando las primeras 5
head(stats, 5)
```

```
       density mutual
[1,] 0.3000000      0
[2,] 0.1666667      0
[3,] 0.1666667      0
[4,] 0.4500000      0
[5,] 0.1500000      0
```

Ahora simulamos los resultados

```r
y <- rbinom(n, 1, plogis(stats %*% c(0.5, -1)))
glm(y ~ stats, family = binomial(link = "logit")) |>
  summary()
```

```
Call:
glm(formula = y ~ stats, family = binomial(link = "logit"))

Coefficients:
             Estimate Std. Error z value Pr(>|z|)
(Intercept)   0.07319    0.41590   0.176    0.860
statsdensity  0.42568    1.26942   0.335    0.737
statsmutual  -1.14804    0.12166  -9.436   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 768.96  on 999  degrees of freedom
Residual deviance: 518.78  on 997  degrees of freedom
AIC: 524.78

Number of Fisher Scoring iterations: 7
```

8.6 Los efectos de red son endógenos

Aquí tenemos dos enfoques diferentes: Autocorrelación Espacial [SAR], y el modelo de atributo de actor autologístico [ALAAM] (Robins, Pattison, and Elliott 2001). El primero es una generalización del modelo de regresión lineal que considera la dependencia espacial. El segundo es un pariente cercano de los ERGMs que trata las covariables como endógenas y la red como exógena. En general, los ALAAMs son más flexibles que los SARs, pero los SARs son más fáciles de estimar.

SAR Formalmente, los modelos SAR (ver LeSage 2008) pueden usarse para estimar efectos de exposición de red. La forma general es:

$$y = \rho W y + \theta^{\mathbf{t}} X + \epsilon, \quad \epsilon \sim \text{MVN}(0, \Sigma)$$

donde $y \equiv \{y_i\}$ es un vector de resultados, ρ es un coeficiente de autocorrelación, $W \in \{w_{ij}\}$ es una matriz cuadrada estocástica por filas de tamaño n, θ es un vector de parámetros del modelo, X es la matriz correspondiente con variables exógenas, y ϵ es un vector de errores que se distribuye normal multivariado con media 0 y covarianza Σ.[^notation] El modelo SAR es una generalización del modelo de regresión lineal que considera la dependencia espacial. El modelo SAR puede estimarse usando el paquete `spatialreg` en R (Roger Bivand 2022).

> **Tip**
>
> ¿Cuál es la red apropiada para usar en el modelo SAR? Según LeSage and Pace (2014), no es muy importante. Dado que $(I_n - \rho \mathbf{W})^{-1} = \rho \mathbf{W} + \rho^2 \mathbf{W}^2 + ...$

Aunque el modelo SAR fue desarrollado para datos espaciales, es fácil aplicarlo a datos de red. Además, cada entrada del vector Wy tiene la misma definición que la exposición de red, es decir

$$Wy \equiv \left\{ \sum_j y_j w_{ij} \right\}_i$$

Dado que W es estocástica por filas, Wy es un promedio ponderado del resultado de los vecinos de i, *es decir*, un vector de exposiciones de red.

ALAAM La forma más simple en que podemos pensar sobre esta clase de modelos es como si una covariable dada intercambiara lugares con la red en un ERGM, entonces la red ahora es fija y la covariable es la variable aleatoria. Aunque los ALAAMs también pueden estimar efectos de exposición de red, podemos usarlos para construir modelos más complejos más allá de la exposición. La forma general es:

$$\mathbb{P}\left(Y = y \mid W, X\right) = \exp\left\{\left(\theta^{\mathbf{t}} s(y, W, X)\right)\right\} \times \eta(\theta)^{-1}$$

$$\eta(\theta) = \sum_y \exp\left\{\left(\theta^{\mathbf{t}} s(y, W, X)\right)\right\}$$

Donde $Y \equiv \{y_i \in (0,1)\}$ es un vector de resultados individuales binarios, W denota la red social, X es una matriz de variables exógenas, θ es un vector de parámetros del modelo, $s(y, W, X)$ es un vector de estadísticas suficientes, y $\eta(\theta)$ es una constante normalizadora.

8.7 Ejemplo de código: SAR

La simulación de modelos SAR puede hacerse usando la siguiente observación: Aunque el resultado aparece en ambos lados de la ecuación, podemos aislarlo en un lado y resolverlo; formalmente:

$$y = \rho X y + \theta^{\mathbf{t}} W + \varepsilon \implies y = (I - \rho X)^{-1} \theta^{\mathbf{t}} W + (I - \rho X)^{-1} \varepsilon$$

El siguiente fragmento de código simula un modelo SAR con un grafo de Bernoulli con 1,000 nodos y una densidad de 0.01. El proceso de generación de datos es el siguiente:

```
set.seed(4114)
n <- 1000

## Simulando la red
p <- 0.01
X <- matrix(rbinom(n^2, 1, p), nrow = n)

## Covariable
W <- matrix(rnorm(n), nrow = n)

## Simulando el resultado
rho <- 0.5
library(MASS) # Para la función mvrnorm

## Identidad menos rho * X
X_rowstoch <- X / rowSums(X)
I <- diag(n) - rho * X_rowstoch

## El resultado
Y <- solve(I) %*% (2 * W) + solve(I) %*% mvrnorm(1, rep(0, n), diag(n))
```

Usando el paquete de R `spatialreg`, podemos ajustar el modelo usando la función `lagsarlm`:

```
library(spdep) # para la función mat2listw
library(spatialreg)
fit <- lagsarlm(
  Y ~ W,
  data  = as.data.frame(X),
  listw = mat2listw(X_rowstoch)
  )
```

```
## Usando texreg para obtener una impresión bonita
texreg::screenreg(fit, single.row = TRUE)
```

```
========================================
                     Model 1
----------------------------------------
(Intercept)             -0.01 (0.03)
W                        1.97 (0.03) ***
rho                      0.54 (0.04) ***
----------------------------------------
Num. obs.             1000
Parameters               4
Log Likelihood       -1373.02
AIC (Linear model)    2920.37
AIC (Spatial model)   2754.05
LR test: statistic     168.32
LR test: p-value         0.00
========================================
*** p < 0.001; ** p < 0.01; * p < 0.05
```

La interpretación de este modelo es casi la misma que una regresión lineal, con la diferencia de que tenemos el efecto de autocorrelación (`rho`). Como se esperaba, el modelo obtuvo una estimación lo suficientemente cercana al parámetro poblacional: $\rho = 0.5$.

8.8 Ejemplo de código: ALAAM

Hasta la fecha, no hay un paquete de R que implemente el marco ALAAM. Sin embargo, puedes ajustar ALAAMs usando el software PNet desarrollado por el grupo Melnet de la Universidad de Melbourne (haz clic aquí).

Debido a las similitudes, los ALAAMs pueden implementarse usando ERGMs. Debido a la novedad de esto, el ejemplo de código se dejará como un posible proyecto de clase. Publicaremos un ejemplo completo después del taller.

8.9 Coevolución

Finalmente, discutimos la coevolución cuando tanto la red como el comportamiento están embebidos en un bucle de retroalimentación. La coevolución debería ser la suposición predeterminada cuando se trata de redes sociales. Sin embargo, los modelos capaces de capturar la coevolución son difíciles de estimar. Aquí, discutiremos dos de tales modelos: Modelos Estocásticos Orientados al Actor (o Modelos Siena) (introducidos por primera vez en T. a. B. Snijders (1996); ver también Tom A. B. Snijders (2017)) y Modelos de Red Exponencial Aleatorios de familia exponencial [ERNMs,] una generalización de ERGMs (Z. Wang, Fellows, and Handcock 2023; Fellows 2012).

Siena Los Modelos Estocásticos Orientados al Actor [SOAMs] o Modelos Siena son modelos dinámicos de red y comportamiento que describen la transición de un sistema de red dentro de dos o más puntos de tiempo.

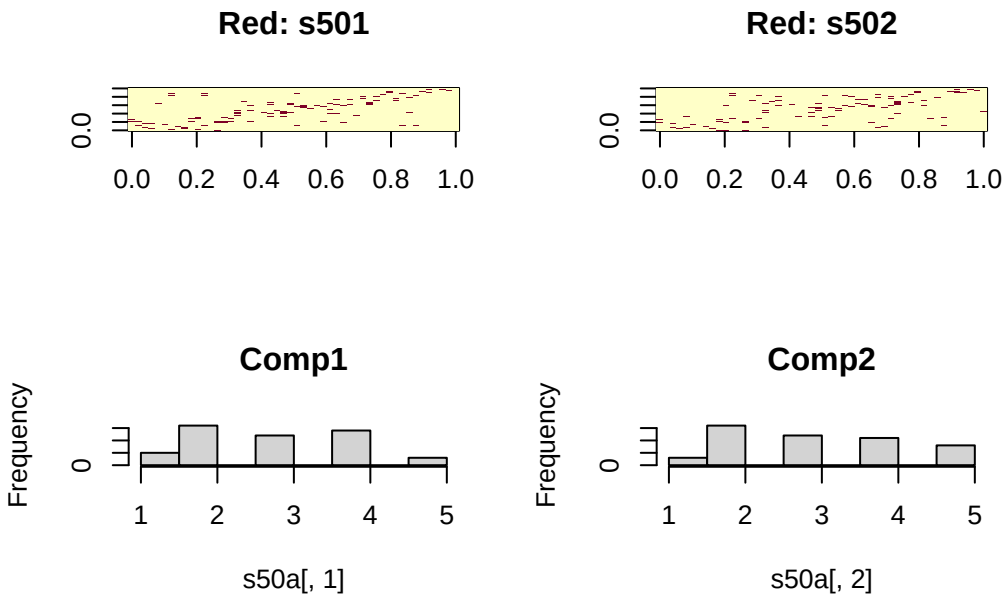
ERNMs Este modelo está estrechamente relacionado con los ERGMs, con la diferencia de que incorporan una salida a nivel de vértice. Conceptualmente, se está moviendo de tener una red aleatoria, a un modelo donde una característica de vértice dada y la red son aleatorias:

$$P_{y,\theta}(Y = y|X = x) \rightarrow P_{y,\theta}(Y = y, X = x)$$

8.10 Ejemplo de código: Siena

Este ejemplo fue adaptado del paquete de R `RSiena` (ver página `?sienaGOF-auxiliary`). Comenzamos cargando el paquete y echando un vistazo a los datos que usaremos:

```
library(RSiena)

## Visualizando la matriz de adyacencia y comportamiento
op <- par(mfrow=c(2, 2))
image(s501, main = "Red: s501")
image(s502, main = "Red: s502")
hist(s50a[,1], main = "Comp1")
hist(s50a[,2], main = "Comp2")
```




```
par(op)
```

El siguiente paso es el proceso de preparación de datos. `RSiena` no recibe datos crudos tal como están. Necesitamos declarar explícitamente las redes y la variable de resultado. Los modelos Siena también pueden modelar cambios de red

```r
## Inicializando la variable dependiente (red)
mynet1 <- sienaDependent(array(c(s501, s502), dim=c(50, 50, 2)))
mynet1
```

```
Type         oneMode
Observations 2
Nodeset      Actors (50 elements)
```

```r
mybeh  <- sienaDependent(s50a[,1:2], type="behavior")
mybeh
```

```
Type         behavior
Observations 2
Nodeset      Actors (50 elements)
```

```r
## Covariables a nivel de nodo (artificiales)
mycov  <- c(rep(1:3,16),1,2)

## Covariables a nivel de enlace (también artificiales)
mydycov <- matrix(rep(1:5, 500), 50, 50)
```

```r
## Creando el objeto de datos
mydata <- sienaDataCreate(mynet1, mybeh)

## Agregando los efectos (¡primero obtenerlos!)
myeff <- getEffects(mydata)

## Nota que Siena agrega algunos efectos predeterminados
myeff
##   name   effectName                    include fix   test  initialValue parm
## 1 mynet1 basic rate parameter mynet1 TRUE    FALSE FALSE    4.69604   0
## 2 mynet1 outdegree (density)         TRUE    FALSE FALSE   -1.48852   0
## 3 mynet1 reciprocity                 TRUE    FALSE FALSE    0.00000   0
## 4 mybeh  rate mybeh period 1         TRUE    FALSE FALSE    0.70571   0
## 5 mybeh  mybeh linear shape          TRUE    FALSE FALSE    0.32247   0
```

```
## 6 mybeh  mybeh quadratic shape       TRUE    FALSE FALSE    0.00000   0

## Agregando algunos efectos extra (automáticamente los imprime)
myeff <- includeEffects(myeff, transTies, cycle3)
##   effectNumber effectName      shortName include fix   test  initialValue parm
## 1 41           3-cycles        cycle3    TRUE    FALSE FALSE          0   0
## 2 44           transitive ties transTies TRUE    FALSE FALSE          0   0
```

Para agregar más efectos, primero, llama a la función `effectsDocumentation(myeff)`. Te mostrará explícitamente cómo agregar un efecto particular. Por ejemplo, si quisiéramos agregar exposición de red (`avExposure`,) bajo la documentación de `effectsDocumentation(myeff)` necesitamos pasar los siguientes argumentos:

```
## Y ahora, efecto de exposición
myeff <- includeEffects(
  myeff,
  avExposure,
  # Estos últimos tres son especificados por effectsDocum...
  name         = "mybeh",
  interaction1 = "mynet1",
  type         = "rate"
  )
```

```
  effectNumber effectName                              shortName  include fix
1 462          average exposure effect on rate mybeh avExposure TRUE    FALSE
  test  initialValue parm
1 FALSE          0   0
```

El siguiente paso involucra crear el modelo con (`sienaAlgorithmCreate`,) donde especificamos todos los parámetros para ajustar el modelo (ej., pasos MCMC.) Aquí, modificamos los valores de `n3` y `nsub` a la mitad de los valores predeterminados para reducir el tiempo que tomaría ajustar el modelo; sin embargo esto degrada la calidad del ajuste.

```
## Fases 2 y 3 más cortas, solo para ejemplo:
myalgorithm <- sienaAlgorithmCreate(
  nsub = 2, n3 = 500, seed = 122, projname = NULL
  )
## If you use this algorithm object, siena07 will create/use an output file /tmp/Rtmpc
## This is a temporary file for this R session.

## Ajustando e imprimiendo el modelo
ans <- siena07(
  myalgorithm,
  data = mydata, effects = myeff,
  returnDeps = TRUE, batch = TRUE
  )
##
## Start phase 0
## theta:  4.696 -1.489  0.000  0.000  0.000  0.706  0.000  0.322  0.000
##
## Start phase 1
## Phase 1 Iteration 1 Progress: 0%
## Phase 1 Iteration 2 Progress: 0%
## Phase 1 Iteration 3 Progress: 0%
## Phase 1 Iteration 4 Progress: 0%
## Phase 1 Iteration 5 Progress: 0%
## Phase 1 Iteration 10 Progress: 1%
## Phase 1 Iteration 15 Progress: 1%
## Phase 1 Iteration 20 Progress: 1%
## Phase 1 Iteration 25 Progress: 2%
## Phase 1 Iteration 30 Progress: 2%
## Phase 1 Iteration 35 Progress: 2%
## Phase 1 Iteration 40 Progress: 3%
## Phase 1 Iteration 45 Progress: 3%
## Phase 1 Iteration 50 Progress: 3%
## theta:  5.380 -1.734  0.481  0.147  0.166  1.168 -0.340  0.250  0.140
##
## Start phase 2.1
## Phase 2 Subphase 1 Iteration 1 Progress: 33%
```

```
## Phase 2 Subphase 1 Iteration 2 Progress: 33%
## theta  6.044 -1.877  0.840  0.300  0.473  1.096 -0.332  0.112  0.191
## ac  0.375 -0.183  3.835  4.574 23.412  0.922  0.908  0.626  3.302
## Phase 2 Subphase 1 Iteration 3 Progress: 33%
## Phase 2 Subphase 1 Iteration 4 Progress: 33%
## theta  6.1075 -2.0372  1.2512  0.0341  0.5207  1.2593 -0.1534  0.3018  0.1928
## ac  0.9859  0.2280 -0.0703 -2.2973 -2.7455  1.1518  1.0749  0.8732  0.5399
## Phase 2 Subphase 1 Iteration 5 Progress: 33%
## Phase 2 Subphase 1 Iteration 6 Progress: 33%
## theta  6.8650 -2.3185  1.7577  0.2694  0.7636  1.1997 -0.0433  0.3000  0.1762
## ac  0.4789  0.4883  0.0199 -1.9449 -2.2609  1.1444  1.0397  0.9282  0.6470
## Phase 2 Subphase 1 Iteration 7 Progress: 33%
## Phase 2 Subphase 1 Iteration 8 Progress: 33%
## theta  6.801140 -2.485273  1.966471  0.301197  0.817324  1.228010  0.000627  0.3777
## ac  0.4538  0.4933 -0.0158 -1.9326 -2.2588  1.1102  1.0395  0.9239  0.6905
## Phase 2 Subphase 1 Iteration 9 Progress: 33%
## Phase 2 Subphase 1 Iteration 10 Progress: 33%
## theta  6.1258 -2.5124  1.8704  0.1206  0.6054  1.4538 -0.0145  0.5091 -0.0671
## ac  0.472  0.103 -0.330 -1.625 -1.991  1.152  1.090  0.767  0.514
## Phase 2 Subphase 1 Iteration 1 Progress: 33%
## Phase 2 Subphase 1 Iteration 2 Progress: 33%
## Phase 2 Subphase 1 Iteration 3 Progress: 33%
## Phase 2 Subphase 1 Iteration 4 Progress: 33%
## Phase 2 Subphase 1 Iteration 5 Progress: 33%
## Phase 2 Subphase 1 Iteration 6 Progress: 34%
## Phase 2 Subphase 1 Iteration 7 Progress: 34%
## Phase 2 Subphase 1 Iteration 8 Progress: 34%
## Phase 2 Subphase 1 Iteration 9 Progress: 34%
## Phase 2 Subphase 1 Iteration 10 Progress: 34%
## theta  6.4976 -2.5900  1.9265  0.2750  0.8543  1.0420  0.0446  0.3234 -0.0686
## ac -0.212 -0.697 -0.818 -0.831 -0.765 -0.442 -0.268 -0.240 -0.267
## theta:  6.4976 -2.5900  1.9265  0.2750  0.8543  1.0420  0.0446  0.3234 -0.0686
##
## Start phase 2.2
## Phase 2 Subphase 2 Iteration 1 Progress: 48%
## Phase 2 Subphase 2 Iteration 2 Progress: 48%
```

```
## Phase 2 Subphase 2 Iteration 3 Progress: 48%
## Phase 2 Subphase 2 Iteration 4 Progress: 48%
## Phase 2 Subphase 2 Iteration 5 Progress: 48%
## Phase 2 Subphase 2 Iteration 6 Progress: 48%
## Phase 2 Subphase 2 Iteration 7 Progress: 49%
## Phase 2 Subphase 2 Iteration 8 Progress: 49%
## Phase 2 Subphase 2 Iteration 9 Progress: 49%
## Phase 2 Subphase 2 Iteration 10 Progress: 49%
## theta  6.7988 -2.5810  1.9335  0.3941  0.7843  1.1111  0.0358  0.3252 -0.0415
## ac -0.0757 -0.3672 -0.4025 -0.3676 -0.4157 -0.0166  0.0222 -0.0874  0.0651
## theta:  6.7988 -2.5810  1.9335  0.3941  0.7843  1.1111  0.0358  0.3252 -0.0415
##
## Start phase 3
## Phase 3 Iteration 500 Progress 100%

ans
## Estimates, standard errors and convergence t-ratios
##
##                                                   Estimate   Standard   Convergence
##                                                              Error      t-ratio
## Network Dynamics
##   1. rate basic rate parameter mynet1              6.7988  ( 1.2496   )   -0.0460
##   2. eval outdegree (density)                     -2.5810  ( 0.1505   )    0.0113
##   3. eval reciprocity                              1.9335  ( 0.2627   )    0.0205
##   4. eval 3-cycles                                 0.3941  ( 0.2742   )   -0.0658
##   5. eval transitive ties                          0.7843  ( 0.2379   )   -0.0014
##
## Behavior Dynamics
##   6. rate rate mybeh period 1                      1.1111  ( 1.3096   )   -0.0199
##   7. rate average exposure effect on rate mybeh    0.0358  ( 0.4373   )   -0.0190
##   8. eval mybeh linear shape                       0.3252  ( 0.2329   )   -0.0048
##   9. eval mybeh quadratic shape                   -0.0415  ( 0.1139   )    0.0992
##
## Overall maximum convergence ratio:    0.2163
##
##
```

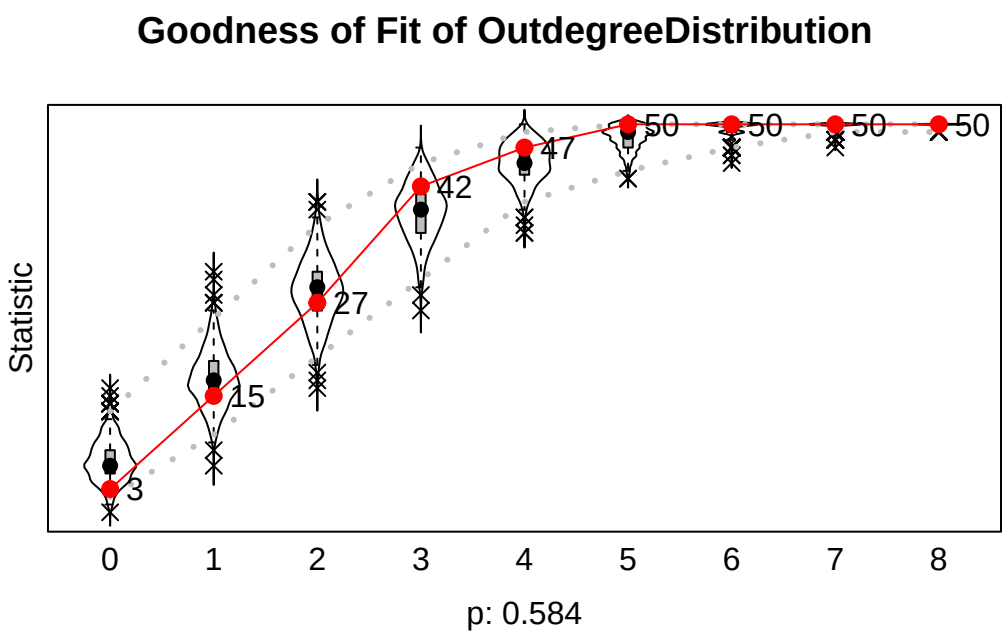
```
## Total of 940 iteration steps.
```

Como regla general, valores t absolutos por debajo de 0.1 muestran buena convergencia, por debajo de 0.2 decimos "razonablemente bien," y por encima no hay convergencia. Resaltemos dos de los efectos que tenemos en nuestro modelo

1. Los vínculos transitivos (número cinco) son positivos 0.78 con un valor t menor que 0.01. Por lo tanto, decimos que la red tiene una tendencia hacia la transitividad (equilibrio) que es significativa.
2. El efecto de exposición (número siete) también es positivo, pero pequeño, 0.03, pero aún significativo (valor t de -0.01)

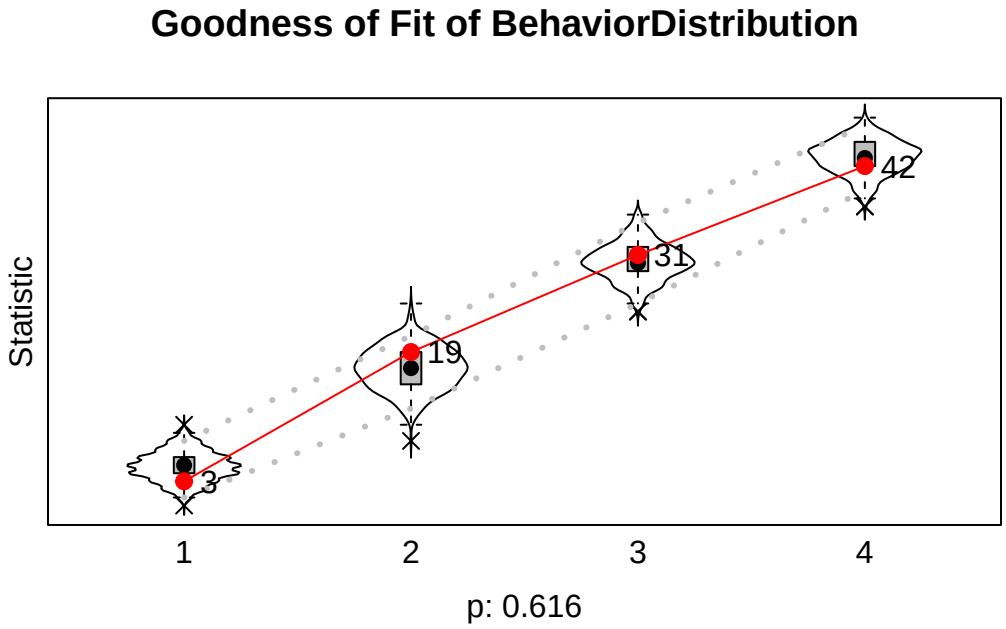
Como con ERGMs, también hacemos bondad de ajuste:

```
sienaGOF(ans, OutdegreeDistribution, varName="mynet1") |>
  plot()
```




```
sienaGOF(ans, BehaviorDistribution, varName = "mybeh") |>
  plot()
```

```
Note: some statistics are not plotted because their variance is 0.
This holds for the statistic: 5.
```



8.11 Ejemplo de código: ERNM

Hasta la fecha, no hay un lanzamiento CRAN para el modelo ERNM. La única implementación de la que estoy al tanto es de uno de los autores principales, que está disponible en GitHub: https://github.com/fellstat/ernm. Desafortunadamente, la versión actual del paquete parece estar rota.

Al igual que el caso ALAAM, como los ERNMs están estrechamente relacionados con los ERGMS, ¡construir un ejemplo usando el paquete ERGM podría ser una gran oportunidad para un proyecto de clase!

9 Modelos Exponenciales de Grafos Aleatorios

Recomiendo encarecidamente leer la viñeta del paquete de R `ergm`.

```
vignette("ergm", package="ergm")
```

> El propósito de los ERGMs, en pocas palabras, es describir de manera parsimoniosa las fuerzas de selección local que dan forma a la estructura global de una red. Para este fin, un conjunto de datos de red, como los que se muestran en la Figura 1, puede ser considerado como la respuesta en un modelo de regresión, donde los predictores son cosas como "propensión de individuos del mismo sexo a formar asociaciones" o "propensión de individuos a formar triángulos de asociaciones". En la Figura 1(b), por ejemplo, es evidente que los nodos individuales parecen agruparse en grupos de las mismas etiquetas numéricas (que resultan ser las calificaciones de los estudiantes, del 7 al 12); por lo tanto, un ERGM puede ayudarnos a cuantificar la fuerza de este efecto intra-grupo.
>
> — (David R. Hunter et al. 2008)

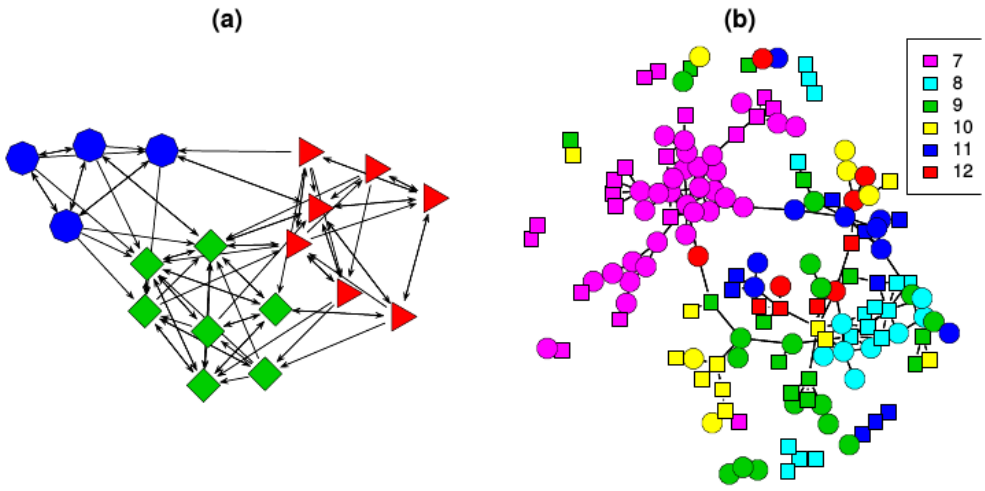


Figure 1: The **(a)** `samplike` and **(b)** `faux.mesa.high` networks described in Section 2. The values of nodal covariates may be indicated using various colors, shapes, and labels of nodes.

Figure 9.1: Fuente: Hunter et al. (2008)

En pocas palabras, usamos ERGMs (que en inglés son "Exponential Random Graph Models") como una interpretación paramétrica de la distribución de $\mathbf{Y}$, que toma la forma canónica:

$$\mathbb{P}_{\mathcal{Y}}\left(\mathbf{Y}=\mathbf{y};\ \right)=\frac{\exp\left\{\theta^{\mathrm{T}}\mathbf{g}(\mathbf{y})\right\}}{\kappa\left(\theta,\mathcal{Y}\right)},\quad \mathbf{y}\in\mathcal{Y} \tag{9.1}$$

Donde $\theta\in\Omega\subset\mathbb{R}^q$ es el vector de coeficientes del modelo y $\mathbf{g}(\mathbf{y})$ es un q-vector de estadísticas basadas en la matriz de adyacencia $\mathbf{y}$.

El modelo Equation 9.1 puede expandirse reemplazando $\mathbf{g}(\mathbf{y})$ con $\mathbf{g}(\mathbf{y},\mathbf{X})$ para permitir información adicional de covariables $\mathbf{X}$ sobre la red. El denominador $\kappa\left(\theta,\mathcal{Y}\right)=\sum_{\mathbf{y}\in\mathcal{Y}}\exp\left\{\theta^{\mathrm{T}}\mathbf{g}(\mathbf{y})\right\}$ es el factor normalizador que asegura que la ecuación Equation 9.1 sea una distribución de probabilidad legítima. Incluso después de fijar $\mathcal{Y}$ para que sean todas las redes que tienen tamaño n, el tamaño de $\mathcal{Y}$ hace que este tipo de modelo estadístico sea difícil de estimar ya que hay $N=2^{n(n-1)}$ redes posibles! (David R. Hunter et al. 2008)

Desarrollos posteriores incluyen nuevas estructuras de dependencia para considerar efectos de vecindario más generales. Estos modelos relajan las suposiciones de dependencia markoviana de un paso, permitiendo la investigación de configuraciones de mayor alcance, como rutas más largas en la red o ciclos más grandes (Pattison y Robins 2002). Se han desarrollado

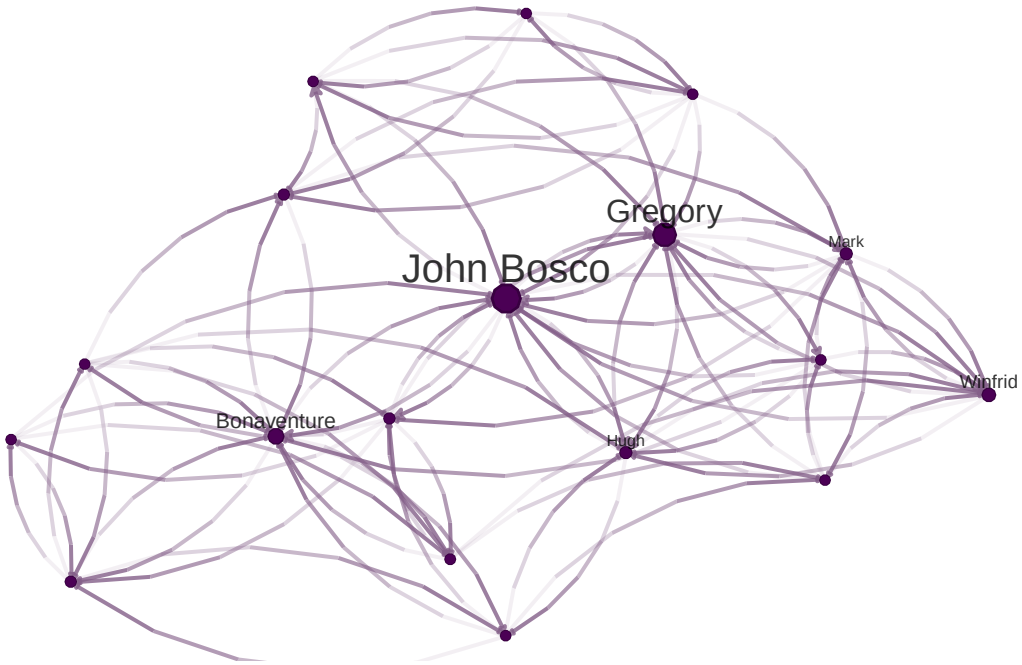
modelos para estructuras de red bipartitas (Faust y Skvoretz 1999) y tripartitas (Mische y Robins 2000). (David R. Hunter et al. 2008, 9)

9.1 Un ejemplo ingenuo

En el caso más simple, los ERGMs equivalen a una regresión logística. Por simple, me refiero a casos sin términos markovianos–motivos que involucran más de un enlace–por ejemplo, el grafo de Bernoulli. En el grafo de Bernoulli, los vínculos son independientes, por lo que la presencia/ausencia de un vínculo entre los nodos i y j no afectará la presencia/ausencia de un vínculo entre los nodos k y l.

Ajustemos un ERGM usando el conjunto de datos `sampson` en el paquete `ergm`.

```
library(ergm)
library(netplot)
data("sampson")
nplot(samplike)
```



Usar `ergm` para ajustar un grafo de Bernoulli requiere usar el término `edges`, que cuenta cuántos vínculos hay en el grafo:

```r
ergm_fit <- ergm(samplike ~ edges)
## Starting maximum pseudolikelihood estimation (MPLE):
## Obtaining the responsible dyads.
## Evaluating the predictor and response matrix.
## Maximizing the pseudolikelihood.
## Finished MPLE.
## Evaluating log-likelihood at the estimate.
```

Dado que esto es equivalente a una regresión logística, podemos usar la función `glm` para ajustar el mismo modelo. Primero, necesitamos preparar los datos para que podamos pasarlos a `glm`:

```r
y <- sort(as.vector(as.matrix(samplike)))
y <- y[-c(1:18)] # Eliminamos la diagonal del modelo, que es todo 0.
y
##   [1] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [38] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [75] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [112] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [149] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [186] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1
## [223] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [260] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [297] 1 1 1 1 1 1 1 1 1 1
```

Ahora podemos ajustar el modelo GLM:

```r
glm_fit <- glm(y~1, family=binomial("logit"))
```

Los coeficientes de ambos ERGM y GLM deberían coincidir:

```r
glm_fit
##
## Call:  glm(formula = y ~ 1, family = binomial("logit"))
##
```

```
## Coefficients:
## (Intercept)
##     -0.9072
##
## Degrees of Freedom: 305 Total (i.e. Null);  305 Residual
## Null Deviance:       367.2
## Residual Deviance: 367.2     AIC: 369.2
ergm_fit
##
## Call:
## ergm(formula = samplike ~ edges)
##
## Maximum Likelihood Coefficients:
##   edges
## -0.9072
```

Además, en el caso del grafo de Bernoulli, podemos obtener la estimación usando la función Logit:

```
pr <- mean(y)
# Función Logit:
# Alternativamente podríamos haber usado log(pr) - log(1-pr)
qlogis(pr)
## [1] -0.9071582
```

De nuevo, el mismo resultado. El grafo de Bernoulli no es el único modelo ERGM que puede ajustarse usando una regresión logística. Además, si todos los términos del modelo son términos no-Markov, `ergm` automáticamente usa por defecto una regresión logística.

9.2 Estimación de ERGMs

El objetivo final es realizar inferencia estadística sobre el modelo propuesto. En un entorno *estándar*, podríamos usar Estimación de Máxima Verosimilitud (MLE, por sus siglas en inglés), que consiste en encontrar los parámetros del modelo θ que, dados los datos observados,

maximicen la verosimilitud del modelo. Para esto último, generalmente usamos el método de Newton. El método de Newton requiere calcular la log-verosimilitud del modelo, lo que puede ser desafiante en ERGMs.

Para ERGMs, dado que parte de la verosimilitud involucra una constante normalizadora que es una función de todas las redes posibles, esto no es tan directo como en el entorno regular. Debido a esto, la mayoría de los métodos de estimación se basan en simulaciones.

En `statnet`, el método de estimación predeterminado se basa en un método propuesto por (Geyer and Thompson 1992), MLE de Cadena de Markov, que usa Monte Carlo de Cadena de Markov para simular redes y una versión modificada del algoritmo Newton-Raphson para estimar los parámetros.

La idea del MC-MLE para esta familia de modelos estadísticos es aproximar la expectativa de las razones de constantes normalizadoras usando la ley de los grandes números. En particular, lo siguiente:

$$\begin{aligned}
\frac{\kappa\left(\theta, \mathcal{Y}\right)}{\kappa\left(\theta_0, \mathcal{Y}\right)} &= \frac{\sum_{\mathbf{y}\in\mathcal{Y}} \exp\left\{\theta^{\mathrm{T}}\mathbf{g}(\mathbf{y})\right\}}{\sum_{\mathbf{y}\in\mathcal{Y}} \exp\left\{\theta_0^{\mathrm{T}}\mathbf{g}(\mathbf{y})\right\}} \\
&= \sum_{\mathbf{y}\in\mathcal{Y}} \left(\frac{1}{\sum_{\mathbf{y}\in\mathcal{Y}} \exp\{\theta_0^{\mathrm{T}}\mathbf{g}(\mathbf{y})\}} \times \exp\left\{\theta^{\mathrm{T}}\mathbf{g}(\mathbf{y})\right\} \right) \\
&= \sum_{\mathbf{y}\in\mathcal{Y}} \left(\frac{\exp\left\{\theta_0^{\mathrm{T}}\mathbf{g}(\mathbf{y})\right\}}{\sum_{\mathbf{y}\in\mathcal{Y}} \exp\{\theta_0^{\mathrm{T}}\mathbf{g}(\mathbf{y})\}} \times \exp\left\{(\theta-\theta_0)^{\mathrm{T}}\mathbf{g}(\mathbf{y})\right\} \right) \\
&= \sum_{\mathbf{y}\in\mathcal{Y}} \left(\mathbb{P}_{\times}\left(Y = y|\mathcal{Y}, \theta_0\right) \exp\left\{(\theta-\theta_0)^{\mathrm{T}}\mathbf{g}(\mathbf{y})\right\}\right) \\
&= \mathrm{E}_{\theta_0}\left(\exp\left\{(\theta-\theta_0)^{\mathrm{T}}\mathbf{g}(\mathbf{y})\right\}\right)
\end{aligned}$$

En particular, el algoritmo MC-MLE usa este hecho para maximizar la razón de log-verosimilitud. La función objetivo misma puede aproximarse simulando m redes de la distribución con parámetro θ_0:

$$l(\theta) - l(\theta_0) \approx (\theta-\theta_0)^{\mathrm{T}}\mathbf{g}(\mathbf{y}_{obs}) - \log\left[\frac{1}{m}\sum_{i=1}^{m} \exp\left\{(\theta-\theta_0)^{\mathrm{T}}\right\}\mathbf{g}(\mathbf{Y}_i)\right]$$

Para más detalles, ver (David R. Hunter et al. 2008). Un bosquejo del algoritmo sigue:

1. Inicializar el algoritmo con una conjetura inicial de θ, llamarlo $\theta^{(t)}$ (debe ser una conjetura bastante buena)

2. Mientras (no haya convergencia) hacer:

a. Usando $\theta^{(t)}$, simular M redes por medio de pequeños cambios en la $\mathbf{Y}_{obs}$ (la red observada). Esta parte se hace usando un método de muestreo de importancia que pondera cada red propuesta por su verosimilitud condicional en $\theta^{(t)}$

b. Con las redes simuladas, podemos hacer el paso de Newton para actualizar el parámetro $\theta^{(t)}$ (esta es la parte de iteración en el paquete `ergm`): $\theta^{(t)} \rightarrow \theta^{(t+1)}$.

c. Si se ha alcanzado la convergencia (lo que usualmente significa que $\theta^{(t)}$ y $\theta^{(t+1)}$ no son muy diferentes), entonces parar; de lo contrario, ir al paso a.

Lusher, Koskinen, and Robins (2013);Admiraal and Handcock (2006);T. A. Snijders (2002);P. Wang et al. (2009) proporcionan detalles sobre el algoritmo usado por PNet (el mismo que el usado en `RSiena`), y Lusher, Koskinen, and Robins (2013) proporciona una breve discusión sobre las diferencias entre `ergm` y `PNet`.

9.3 El paquete `ergm`

De la sección anterior:[1]

```
library(igraph)

library(dplyr)

load("03.rda")
```

En esta sección, usaremos el paquete `ergm` (del conjunto de paquetes `statnet` Handcock et al. (2023),) y el paquete `intergraph` (Bojanowski 2023). Este último proporciona funciones para ir y venir entre objetos `igraph` y `network` de los paquetes `igraph` y `network` respectivamente[2]

[1] Puedes descargar el archivo 03.rda desde este enlace.
[2] Sí, las clases tienen el mismo nombre que los paquetes.

```
library(ergm)
library(intergraph)
```

Como una nota lateral bastante importante, el orden en que se cargan los paquetes de R importa. ¿Por qué es importante mencionarlo ahora? Bueno, resulta que al menos un par de funciones en el paquete `network` tienen el mismo nombre que algunas funciones en el paquete `igraph`. Cuando se carga el paquete `ergm`, dado que depende de `network`, cargará el paquete `network` primero, lo que *enmascarará* algunas funciones en `igraph`. Esto se vuelve evidente una vez que cargas `ergm` después de cargar `igraph`:

```
Los siguientes objetos están enmascarados desde 'package:igraph':

  add.edges, add.vertices, %c%, delete.edges, delete.vertices, get.edge.attribute, get.e
  get.vertex.attribute, is.bipartite, is.directed, list.edge.attributes, list.vertex.att
  set.edge.attribute, set.vertex.attribute
```

¿Cuáles son las implicaciones de esto? Si llamas la función `list.edge.attributes` para un objeto de clase `igraph` R devolverá un error ya que la primera función que coincide con ese nombre viene del paquete `network`! Para evitar esto puedes usar la notación de doble dos puntos:

```
igraph::list.edge.attributes(my_igraph_object)
network::list.edge.attributes(my_network_object)
```

Usando la función `asNetwork`, podemos coercionar el objeto `igraph` en un objeto network para que podamos usarlo con la función `ergm`:

```
# Creando la nueva red
network_111 <- intergraph::asNetwork(ig_year1_111)

# Ejecutando un ergm simple (solo ajustando cuenta de enlaces)
ergm(network_111 ~ edges)
## Warning in ergm.getnetwork(formula): This network contains loops
## Starting maximum pseudolikelihood estimation (MPLE):
## Obtaining the responsible dyads.
```

```
## Evaluating the predictor and response matrix.
## Maximizing the pseudolikelihood.
## Finished MPLE.
## Evaluating log-likelihood at the estimate.
##
## Call:
## ergm(formula = network_111 ~ edges)
##
## Maximum Likelihood Coefficients:
##  edges
## -4.734
```

Entonces, ¿qué pasó aquí? Obtuvimos una advertencia. Resulta que nuestra red tiene bucles (¡no pensé en eso antes!), que son arcos que conectan un nodo consigo mismo. Echemos un vistazo a eso con la función `which_loop`

```
E(ig_year1_111)[which_loop(ig_year1_111)]
## + 1/2638 edge from ff93ece (vertex names):
## [1] 1110111->1110111
```

Podemos deshacernos de estos usando el `igraph::-.igraph`. Eliminemos los aislados usando el mismo operador

```
# Creando la nueva red
network_111 <- ig_year1_111

# Eliminando bucles
network_111 <- network_111 - E(network_111)[which(which_loop(network_111))]

# Eliminando aislados
network_111 <- network_111 - which(degree(network_111, mode = "all") == 0)

# Convirtiendo la red
network_111 <- intergraph::asNetwork(network_111)
```

`asNetwork(simplify(ig_year1_111))` `ig_year1_111 |> simplify() |> asNetwork()`

Un problema que tenemos con estos datos es el hecho de que algunos vértices tienen valores faltantes en las variables `hispanic`, `female1`, y `eversmk1`. Por ahora, procederemos imputando valores basados en los promedios:

```
for (v in c("hispanic", "female1", "eversmk1")) {
  tmpv <- network_111 %v% v
  tmpv[is.na(tmpv)] <- mean(tmpv, na.rm = TRUE) > .5
  network_111 %v% v <- tmpv
}
```

Echemos un vistazo a la red

```
nplot(
  network_111,
  vertex.color = ~ hispanic
  )
```

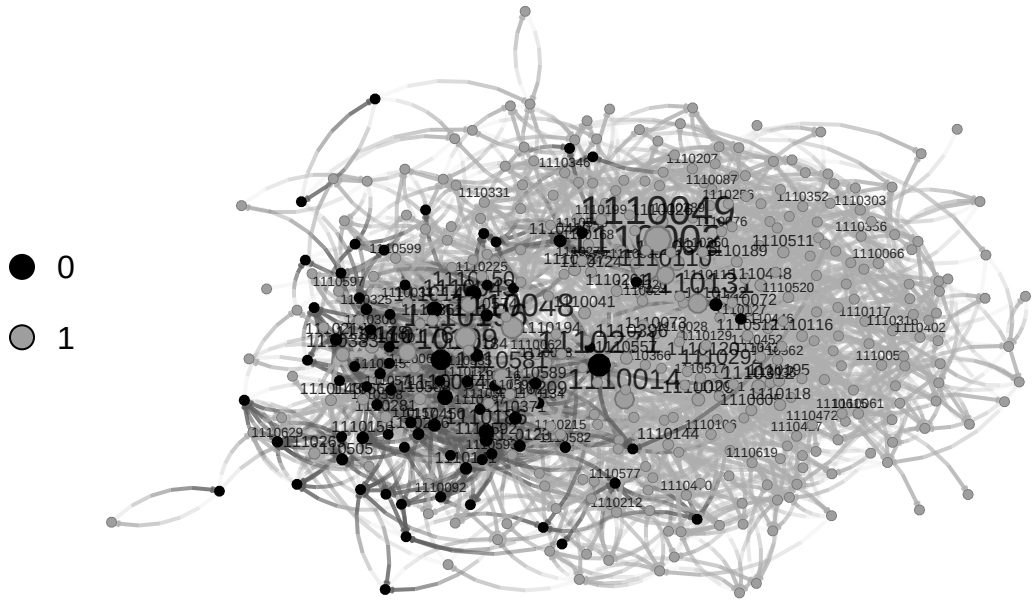


Figure 9.2

Colorando por `hispanic`, podemos ver que los nodos parecen estar agrupados por este atributo. La inspección visual en la ciencia de redes puede ser muy útil, pero, para hacer declara-

ciones más formales, necesitamos usar modelos estadísticos; aquí es donde los ERGMs brillan.

9.4 Estimando ERGMs

Aunque no existe una solución única para estimar estos modelos, el siguiente flujo de trabajo me ha funcionado en el pasado:

1. Estimar el modelo más simple, agregando una variable a la vez.
2. Después de cada estimación, ejecutar la función `mcmc.diagnostics` para ver qué tan bien (o mal) se comportaron las cadenas.
3. Ejecutar la función `gof` y verificar qué tan bien el modelo coincide con las estadísticas estructurales de la red.

Qué usar:

1. `control.ergms`: Número máximo de iteraciones, semilla para Pseudo-RNG, cuántos núcleos
2. `ergm.constraints`: De dónde muestrear la red. Da estabilidad y (en algunos casos) convergencia más rápida ya que al restringir el modelo estás reduciendo el tamaño de la muestra.

Aquí hay un ejemplo de un par de modelos que podríamos comparar

> **Note**
>
> Nota que este documento puede no incluir los mensajes usuales que el comando `ergm` genera durante el procedimiento de estimación. Esto es solo para hacerlo más amigable para imprimir.

```
ans0 <- ergm(
  network_111 ~
    edges +
    nodematch("hispanic") +
    nodematch("female1") +
```

```
    nodematch("eversmk1") +
    mutual,
  control = control.ergm(
    seed        = 1,
    MCMLE.maxit = 10,
    parallel    = 4,
    CD.maxit    = 10
    )
  )
## Warning: 'glpk' selected as the solver, but package 'Rglpk' is not available;
## falling back to 'lpSolveAPI'. This should be fine unless the sample size and/or
## the number of parameters is very big.
```

> **Versión anterior del libro**
>
> En una versión anterior del libro usamos la restricción `bd` para limitar el número máximo de vínculos salientes a 19. Aunque esta es una restricción razonable (ya que la muestra pudo haber sido restringida por construcción), la versión actual del paquete ERGM devuelve una advertencia y muestra las estadísticas GOF con signos cambiados (AIC y BIC negativos, y Logverosimilitud positiva).

Entonces, ¿qué estamos haciendo aquí:

1. El modelo está controlando por:

    a. `edges` Número de enlaces en la red (en oposición a su densidad)

    b. `nodematch("algún-nombre-de-variable-aquí")` Incluye un término que controla por homofilia/heterofilia

    c. `mutual` Número de conexiones mutuas entre $(i, j), (j, i)$. Esto puede estar relacionado con, por ejemplo, cierre triádico.

Para más sobre parámetros de control, ver Morris, Handcock, and Hunter (2008). El siguiente modelo consiste en una version más simple del modelo anterior. Excluye el término `mutual` lleva a un modelo sin términos markovianos, es decir, los estadísticos de la red son funciones de enlaces individuales y no de pares de enlaces o más. El término `mutual` incluye dos enlaces ($i \to j$ y $j \to i$). Modelos sin términos markovianos pueden ser estimados usando regresión logística, evitando la necesidad de simulaciones MCMC:

```
ans1 <- ergm(
  network_111 ~
    edges +
    nodematch("hispanic") +
    nodematch("female1") +
    nodematch("eversmk1")
    ,
  control = control.ergm(
    seed        = 1,
    MCMLE.maxit = 10,
    parallel    = 4,
    CD.maxit    = 10
    )
  )
```

Finalmente, podemos intentar estimar un modelo con términos más complejos, por ejemplo, el *geometrically weighted edge-wise shared partner* (gwesp). Éste término captura el cierre triádico, pero es más fácil de estimar que `triangle`. Más aún, para facilitar el proceso de estimación, podemos incorporar el termino *geometrically weighted dyad-wise shared partner* (gwdsp). Ambos estadísticos incluyen un parámetro de descuento (decay); por el momento, fijaremos ambos parametros en 0.5 (valores entre 0.25 y 1.5 son comunes). Más detalles sobre estos términos en David R. Hunter (2007).

```
ans2 <- ergm(
  network_111 ~
    edges +
    nodematch("hispanic") +
    nodematch("female1") +
    nodematch("eversmk1") +
    mutual +
    gwesp(decay = 0.5, fixed = TRUE) +
    gwdsp(decay = 0.5, fixed = TRUE)
    ,
  control = control.ergm(
    seed        = 1,
    MCMLE.maxit = 10,
```

```
    parallel    = 4,
    CD.maxit    = 10
    )
  )
```

Ahora, un truco agradable para ver todas las regresiones en la misma tabla, podemos usar el paquete `texreg` (Leifeld 2013) que soporta salidas de `ergm`!

```
library(texreg)
## Version:  1.39.4
## Date:     2024-07-23
## Author:   Philip Leifeld (University of Manchester)
##
## Consider submitting praise using the praise or praise_interactive functions.
## Please cite the JSS article in your publications -- see citation("texreg").
screenreg(list(ans0, ans1, ans2))
##
## ========================================================================
##                         Model 1        Model 2        Model 3
## ------------------------------------------------------------------------
## edges                    -5.62 ***      -5.49 ***      -4.73 ***
##                          (0.05)         (0.06)         (0.07)
## nodematch.hispanic        0.22 ***       0.30 ***       0.16 ***
##                          (0.04)         (0.05)         (0.03)
## nodematch.female1         0.87 ***       1.16 ***       0.47 ***
##                          (0.04)         (0.05)         (0.03)
## nodematch.eversmk1        0.33 ***       0.45 ***       0.23 ***
##                          (0.04)         (0.04)         (0.03)
## mutual                    4.10 ***                      2.72 ***
##                          (0.07)                        (0.08)
## gwesp.OTP.fixed.0.5                                     1.34 ***
##                                                        (0.03)
## gwdsp.OTP.fixed.0.5                                    -0.11 ***
##                                                        (0.01)
## ------------------------------------------------------------------------
## AIC                   22649.55       25135.63       20155.83
```

	Model 1	Model 2	Model 3
edges	−5.62***	−5.49***	−4.73***
	(0.05)	(0.06)	(0.07)
nodematch.hispanic	0.22***	0.30***	0.16***
	(0.04)	(0.05)	(0.03)
nodematch.female1	0.87***	1.16***	0.47***
	(0.04)	(0.05)	(0.03)
nodematch.eversmk1	0.33***	0.45***	0.23***
	(0.04)	(0.04)	(0.03)
mutual	4.10***		2.72***
	(0.07)		(0.08)
gwesp.OTP.fixed.0.5			1.34***
			(0.03)
gwdsp.OTP.fixed.0.5			−0.11***
			(0.01)
AIC	22649.55	25135.63	20155.83
BIC	22699.89	25175.91	20226.31
Log Likelihood	−11319.77	−12563.82	−10070.91

***$p < 0.001$; **$p < 0.01$; *$p < 0.05$

Table 9.1: Statistical models

```
## BIC                   22699.89      25175.91      20226.31
## Log Likelihood       -11319.77     -12563.82     -10070.91
## ================================================================
## *** p < 0.001; ** p < 0.01; * p < 0.05
```

O, si estás usando rmarkdown, puedes exportar los resultados usando LaTeX o html, intentemos este último para ver cómo se ve aquí:

```
library(texreg)
texreg(list(ans0, ans1, ans2))
```

En base a los resultados, podemos hacer las siguientes observaciones:

1. De los tres modelos, el último es el mejor en términos de menor AIC y BIC, y mayor Logverosimilitud.

2. Todos los términos nodematch son significativos y positivos, lo que significa que tenemos homofilia por `hispanic`, `female1`, y `eversmk1`.

3. El término `mutual` es significativo, positivo y grande (~4). Esto indica una fuerte tendencia de los individuos a establecer amistad mutua.

4. El término `gwesp` es positivo y significativo. La interpretación de este término no siempre es simple. Generalmente, decimos que un término gwesp positivo significa que hay una tendencia hacia el cierre triádico.

9.5 Bondad de Ajuste del Modelo

En términos brutos, una vez que cada cadena ha alcanzado la distribución estacionaria, podemos decir que no hay problemas con la autocorrelación y que cada punto de muestra es iid. Esto último implica que, dado que estamos ejecutando el modelo con más de una cadena, podemos usar todas las muestras (cadenas) como un solo conjunto de datos.

> Cambios recientes en el algoritmo de estimación de ergm significan que estos gráficos ya no pueden usarse para asegurar que las estadísticas medias del modelo coincidan con las estadísticas de red observadas. Para esa funcionalidad, por favor usa el comando GOF: `gof(object, GOF=~model)`.
>
> —?ergm::mcmc.diagnostics

Dado que `ans2` es el mejor modelo, veamos las estadísticas GOF. Primero, veamos cómo se comportó el MCMC. Podemos usar la función `mcmc.diagnostics` incluida en el paquete. La función es un envoltorio de un par de funciones del paquete `coda` (Plummer et al. 2006), que son llamadas sobre el objeto `$sample` que contiene las estadísticas *centradas* de las redes muestreadas. Al principio, puede ser confuso mirar el objeto `$sample`; no coincide ni con las estadísticas observadas ni con los coeficientes.

Cuando se llama `mcmc.diagnostics(ans2, centered = FALSE)`, verás muchas salidas, incluyendo un par de gráficos mostrando la traza y distribución posterior de las estadísticas *no centradas* (`centered = FALSE`). Los siguientes fragmentos de código reproducirán la salida de la función `mcmc.diagnostics` paso a paso usando el paquete coda. Primero, necesitamos *descentrar* el objeto de muestra:

```
# Obteniendo la muestra centrada
sample_centered <- ans2$sample
```

```r
# Obteniendo las estadísticas observadas y convirtiéndolas en una matriz para que podamo
# a las muestras
observed <- summary(ans2$formula)
observed <- matrix(
  observed,
  nrow  = nrow(sample_centered[[1]]),
  ncol  = length(observed),
  byrow = TRUE
  )

# Ahora descentramos la muestra
sample_uncentered <- lapply(sample_centered, function(x) {
  x + observed
})

# Tenemos que hacer de esto un objeto mcmc.list
sample_uncentered <- coda::mcmc.list(sample_uncentered)
```

Bajo el capó:

1. *Medias empíricas y sd, y cuantiles*:

```r
summary(sample_uncentered)
##
## Iterations = 327680:5963776
## Thinning interval = 32768
## Number of chains = 4
## Sample size per chain = 173
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##                         Mean     SD Naive SE Time-series SE
## edges                 2471.9  63.94    2.430          6.221
## nodematch.hispanic    1829.8  59.02    2.244          7.099
## nodematch.female1     1874.4  71.09    2.703          8.346
```

```
## nodematch.eversmk1   1755.6  58.02    2.206          6.746
## mutual                481.9  27.72    1.054          3.379
## gwesp.OTP.fixed.0.5  1762.6 113.98    4.333         13.574
## gwdsp.OTP.fixed.0.5 13175.1 644.91   24.516         66.387
##
## 2. Quantiles for each variable:
##
##                        2.5%   25%   50%   75% 97.5%
## edges                2347.3  2431  2469  2510  2605
## nodematch.hispanic   1718.3  1791  1827  1867  1955
## nodematch.female1    1736.0  1827  1871  1928  2016
## nodematch.eversmk1   1654.3  1714  1752  1794  1879
## mutual                429.3   462   481   500   542
## gwesp.OTP.fixed.0.5  1540.9  1684  1760  1835  2013
## gwdsp.OTP.fixed.0.5 11958.3 12737 13137 13596 14488
```

2. *Correlación cruzada*:

```
coda::crosscorr(sample_uncentered)
##                         edges nodematch.hispanic nodematch.female1
## edges                1.0000000          0.8566830         0.8770210
## nodematch.hispanic   0.8566830          1.0000000         0.7573169
## nodematch.female1    0.8770210          0.7573169         1.0000000
## nodematch.eversmk1   0.8229030          0.6949758         0.7455362
## mutual               0.7921269          0.7019044         0.7765035
## gwesp.OTP.fixed.0.5  0.8782943          0.7780147         0.8679069
## gwdsp.OTP.fixed.0.5  0.9678268          0.8370254         0.8591945
##                     nodematch.eversmk1    mutual gwesp.OTP.fixed.0.5
## edges                        0.8229030 0.7921269           0.8782943
## nodematch.hispanic           0.6949758 0.7019044           0.7780147
## nodematch.female1            0.7455362 0.7765035           0.8679069
## nodematch.eversmk1           1.0000000 0.7067807           0.7820246
## mutual                       0.7067807 1.0000000           0.8954091
## gwesp.OTP.fixed.0.5          0.7820246 0.8954091           1.0000000
## gwdsp.OTP.fixed.0.5          0.8098323 0.7688635           0.8575789
##                     gwdsp.OTP.fixed.0.5
```

```
## edges                          0.9678268
## nodematch.hispanic             0.8370254
## nodematch.female1              0.8591945
## nodematch.eversmk1             0.8098323
## mutual                         0.7688635
## gwesp.OTP.fixed.0.5            0.8575789
## gwdsp.OTP.fixed.0.5            1.0000000
```

3. *Autocorrelación*: Por ahora, solo veremos la autocorrelación para la cadena uno. La autocorrelación debería ser pequeña (en un entorno MCMC general). Si la autocorrelación es alta, entonces significa que tu muestra no es idd (no hay propiedad de Markov). Una forma de resolver esto es *adelgazar* la muestra.

```
coda::autocorr(sample_uncentered)[[1]]
## , , edges
##
##                   edges nodematch.hispanic nodematch.female1 nodematch.eversmk1
## Lag 0       1.000000000         0.85174677        0.90035143         0.84731435
## Lag 32768   0.604509897         0.52958876        0.66467595         0.54146395
## Lag 163840  0.369212891         0.35412936        0.40011462         0.39283115
## Lag 327680  0.184463035         0.14706318        0.24305479         0.29114538
## Lag 1638400 0.009007472        -0.02631294        0.03456016        -0.04578478
##                 mutual gwesp.OTP.fixed.0.5 gwdsp.OTP.fixed.0.5
## Lag 0       0.83296982           0.9136663          0.97089215
## Lag 32768   0.68928494           0.6870541          0.57938173
## Lag 163840  0.45203629           0.4511364          0.36536136
## Lag 327680  0.12747154           0.1829741          0.15036241
## Lag 1638400 0.05607638           0.0292786         -0.01022981
##
## , , nodematch.hispanic
##
##                   edges nodematch.hispanic nodematch.female1 nodematch.eversmk1
## Lag 0       0.851746774         1.00000000       0.796948280         0.77244980
## Lag 32768   0.540210932         0.64168452       0.620791472         0.51899205
## Lag 163840  0.348421398         0.40142712       0.409478507         0.40705581
## Lag 327680  0.156894106         0.10147048       0.226595532         0.24435622
```

```
## Lag 1638400 0.007637041        -0.04655592      -0.007443898        -0.07226516
##                 mutual gwesp.OTP.fixed.0.5 gwdsp.OTP.fixed.0.5
## Lag 0       0.69025542         0.795564260          0.82655175
## Lag 32768   0.58418987         0.623959752          0.52242442
## Lag 163840  0.42435250         0.432709863          0.35047668
## Lag 327680  0.11948613         0.174673754          0.13223947
## Lag 1638400 0.05123385         0.007771714         -0.02509792
##
## , , nodematch.female1
##
##                  edges nodematch.hispanic nodematch.female1 nodematch.eversmk1
## Lag 0       0.90035143        0.796948280        1.00000000         0.79308088
## Lag 32768   0.60258009        0.536790742        0.73348572         0.56527326
## Lag 163840  0.32695326        0.324827911        0.37903351         0.38465484
## Lag 327680  0.11123941        0.111356563        0.21164986         0.27852000
## Lag 1638400 0.07516871       -0.002982911        0.09513563        -0.06106261
##                 mutual gwesp.OTP.fixed.0.5 gwdsp.OTP.fixed.0.5
## Lag 0       0.79999770          0.88951827          0.87107087
## Lag 32768   0.66026068          0.69041074          0.56750224
## Lag 163840  0.38096094          0.39655308          0.32909183
## Lag 327680  0.07195343          0.12801382          0.08708550
## Lag 1638400 0.11308394          0.09505784          0.04044172
##
## , , nodematch.eversmk1
##
##                  edges nodematch.hispanic nodematch.female1 nodematch.eversmk1
## Lag 0       0.84731435          0.7724498        0.79308088          1.0000000
## Lag 32768   0.54365021          0.5122084        0.63731624          0.6624233
## Lag 163840  0.31661036          0.3735145        0.38613833          0.4631210
## Lag 327680  0.12487907          0.1838576        0.21306300          0.2895293
## Lag 1638400 -0.04843904        -0.0871528       -0.05860075         -0.1732522
##                 mutual gwesp.OTP.fixed.0.5 gwdsp.OTP.fixed.0.5
## Lag 0       0.73949417          0.82596806          0.82962588
## Lag 32768   0.59917271          0.64206417          0.53636921
## Lag 163840  0.36156912          0.38961111          0.32502461
## Lag 327680  0.06513275          0.10756201          0.12157020
```

```
## Lag 1638400 -0.02126220         -0.05784788         -0.05300152
##
## , , mutual
##
##                   edges nodematch.hispanic nodematch.female1 nodematch.eversmk1
## Lag 0       0.832969818          0.6902554        0.79999770         0.73949417
## Lag 32768   0.655957685          0.5420351        0.68082349         0.60650896
## Lag 163840  0.453711697          0.3807566        0.46832909         0.47654529
## Lag 327680  0.189592365          0.1950812        0.25444882         0.34592646
## Lag 1638400 0.004326463         -0.0397559        0.03453263        -0.04956656
##                 mutual gwesp.OTP.fixed.0.5 gwdsp.OTP.fixed.0.5
## Lag 0       1.00000000          0.91166903         0.790126039
## Lag 32768   0.77089766          0.75016262         0.613992396
## Lag 163840  0.51073753          0.53492576         0.445401161
## Lag 327680  0.19116258          0.22568907         0.144338541
## Lag 1638400 0.01557354          0.02072621        -0.002615637
##
## , , gwesp.OTP.fixed.0.5
##
##                   edges nodematch.hispanic nodematch.female1 nodematch.eversmk1
## Lag 0       0.91366631          0.79556426        0.88951827         0.82596806
## Lag 32768   0.65221015          0.57748172        0.71103822         0.63534710
## Lag 163840  0.40355191          0.36375124        0.42906065         0.45848417
## Lag 327680  0.16182896          0.15751100        0.23871746         0.30543634
## Lag 1638400 0.02774368         -0.01550744        0.04360474        -0.05912761
##                 mutual gwesp.OTP.fixed.0.5 gwdsp.OTP.fixed.0.5
## Lag 0       0.91166903          1.00000000          0.88286005
## Lag 32768   0.75714027          0.74743488          0.62020864
## Lag 163840  0.47254922          0.48618158          0.40054594
## Lag 327680  0.14530942          0.18887509          0.12356707
## Lag 1638400 0.05355936          0.04005611          0.01544559
##
## , , gwdsp.OTP.fixed.0.5
##
##                   edges nodematch.hispanic nodematch.female1 nodematch.eversmk1
## Lag 0       0.97089215          0.82655175        0.87107087         0.82962588
```

```
## Lag 32768   0.56904543         0.49933571         0.63747170         0.51566990
## Lag 163840  0.31817376         0.32249022         0.37414872         0.32999678
## Lag 327680  0.15987338         0.11941690         0.21123819         0.24152798
## Lag 1638400 0.01040193        -0.02676337         0.01558998        -0.05445063
##                 mutual gwesp.OTP.fixed.0.5 gwdsp.OTP.fixed.0.5
## Lag 0       0.79012604           0.8828601          1.00000000
## Lag 32768   0.64720629           0.6437981          0.56197239
## Lag 163840  0.40245842           0.4041699          0.32130564
## Lag 327680  0.09213271           0.1437246          0.12697167
## Lag 1638400 0.06161009           0.0289774         -0.01011366
```

4. *Diagnóstico de Geweke*: Del archivo de ayuda de la función:

   "Si las muestras se extraen de la distribución estacionaria de la cadena, las dos medias son iguales y la estadística de Geweke tiene una distribución normal estándar asintóticamente. […] El Z-score se calcula bajo la suposición de que las dos partes de la cadena son asintóticamente independientes, lo que requiere que la suma de frac1 y frac2 sea estrictamente menor que 1."

   —?coda::geweke.diag

Echemos un vistazo a una sola cadena:

```
coda::geweke.diag(sample_uncentered)[[1]]
##
## Fraction in 1st window = 0.1
## Fraction in 2nd window = 0.5
##
##               edges  nodematch.hispanic   nodematch.female1  nodematch.eversmk1
##             -1.0828             -0.9888             -1.7401             -0.8217
##              mutual gwesp.OTP.fixed.0.5 gwdsp.OTP.fixed.0.5
##             -1.4265             -1.1219             -0.8655
```

5. *(no incluido) Diagnóstico de Gelman*: Del archivo de ayuda de la función:

   Gelman y Rubin (1992) proponen un enfoque general para monitorear la convergencia de salida MCMC en el que se ejecutan m $>$ 1 cadenas paralelas con valores

> iniciales que están sobre-dispersos relativo a la distribución posterior. La convergencia se diagnostica cuando las cadenas han 'olvidado' sus valores iniciales, y la salida de todas las cadenas es indistinguible. El diagnóstico gelman.diag se aplica a una sola variable de la cadena. Se basa en una comparación de varianzas dentro de cadena y entre cadenas, y es similar a un análisis de varianza clásico.
> —?coda::gelman.diag

Como diferencia del estadístico de diagnóstico anterior, este usa todas las cadenas simultáneamente:

```
coda::gelman.diag(sample_uncentered)
## Potential scale reduction factors:
##
##                       Point est. Upper C.I.
## edges                       1.05       1.14
## nodematch.hispanic          1.02       1.04
## nodematch.female1           1.07       1.20
## nodematch.eversmk1          1.07       1.22
## mutual                      1.07       1.20
## gwesp.OTP.fixed.0.5         1.09       1.26
## gwdsp.OTP.fixed.0.5         1.06       1.18
##
## Multivariate psrf
##
## 1.15
```

Como regla general, valores en el $[.9, 1.1]$ son buenos.

Una característica agradable de la función `mcmc.diagnostics` son los gráficos bonitos de traza y distribución posterior que genera. Si tienes el paquete de R `latticeExtra` (Sarkar and Andrews 2022), la función anulará los gráficos predeterminados usados por `coda::plot.mcmc` y usará lattice en su lugar, creando gráficos de mejor apariencia. El siguiente fragmento de código llama la función `mcmc.diagnostic`, pero suprimimos el resto de la salida (ver figura **?@fig-coda-plots**).

```
# [2022-03-13] Esta línea está fallando por lo que podría ser un error de ergm
# mcmc.diagnostics(ans0, center = FALSE) # Suprimiendo toda la salida
```

Si llamamos la función `mcmc.diagnostics`, este mensaje aparece al final:

> Los diagnósticos MCMC mostrados aquí son de la última ronda de simulación, previo al cálculo de las estimaciones finales de parámetros. Porque las estimaciones finales son refinamientos de aquellas usadas para esta ejecución de simulación, estos diagnósticos pueden subestimar el rendimiento del modelo. Para evaluar directamente el rendimiento del modelo final en estadísticas del modelo, por favor usa el comando GOF: gof(ergmFitObject, GOF=~model).
>
> —`mcmc.diagnostics(ans0)`

¡No está tan mal (aunque el término `mutual` podría hacerlo mejor)![3] Primero, observa que en la figura, vemos cuatro líneas diferentes; ¿por qué es eso? Dado que estábamos ejecutando en paralelo usando cuatro núcleos, el algoritmo ejecutó cuatro cadenas del algoritmo MCMC. Una prueba visual es ver si todas las cadenas se movieron más o menos al mismo lugar; en tal caso, podemos empezar a pensar sobre convergencia del modelo desde la perspectiva MCMC.

Una vez que estamos seguros de haber alcanzado convergencia en el algoritmo MCMC, podemos empezar a pensar sobre qué tan bien nuestro modelo predice las propiedades de la red observada. Además de las estadísticas que definen nuestro ERGM, el comportamiento predeterminado de la función `gof` muestra GOF para:

a. Distribución de grado de entrada,
b. Distribución de grado de salida,
c. Socios compartidos por enlace, y
d. Geodésicas

Echemos un vistazo

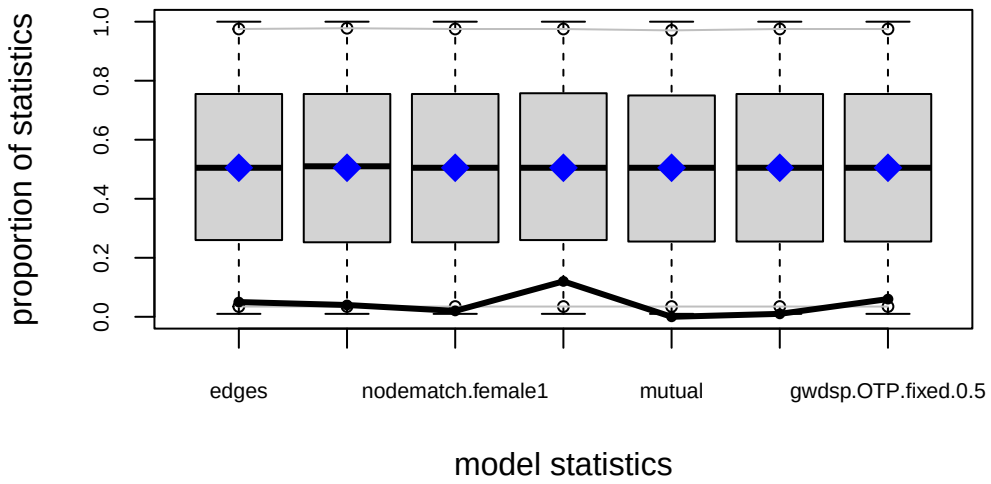
```
# Calculando e imprimiendo estadísticas GOF
ans_gof <- gof(ans2)
ans_gof
##
## Goodness-of-fit for in-degree
##
##           obs min  mean max MC p-value
```

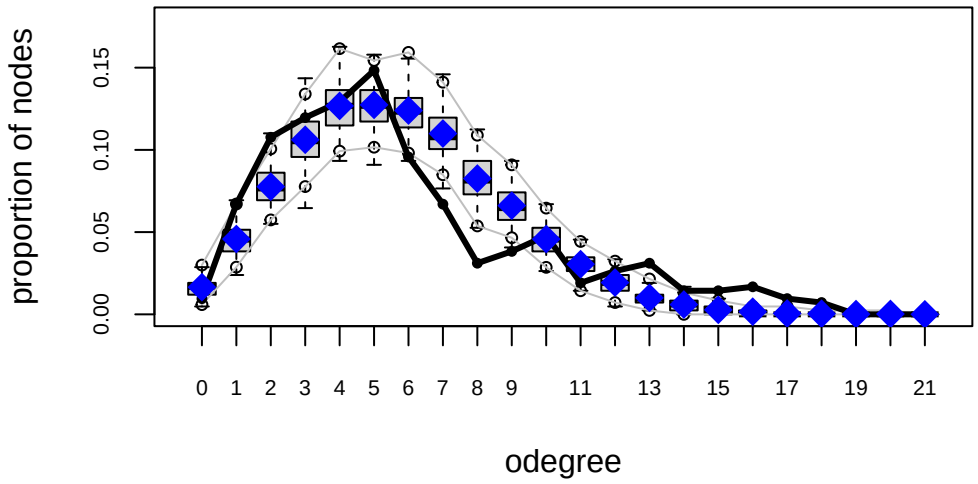
[3] El sitio web wiki de statnet tiene un ejemplo muy agradable de gráficos de diagnóstico MCMC (muy) malos y buenos aquí.

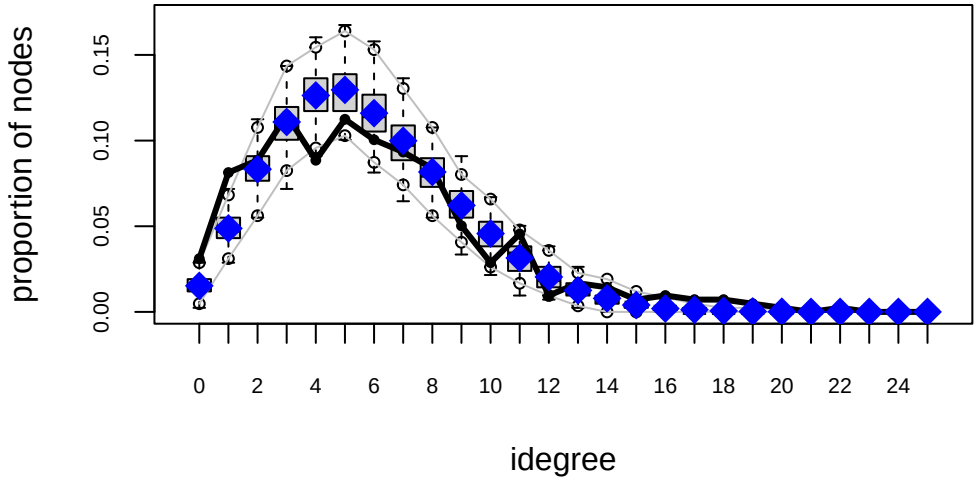
```
## idegree0   13   1  6.36  12       0.00
## idegree1   34  12 20.37  30       0.00
## idegree2   37  21 34.82  51       0.80
## idegree3   48  30 46.34  66       0.82
## idegree4   37  38 52.81  67       0.00
## idegree5   47  42 54.17  72       0.36
## idegree6   42  34 48.49  66       0.44
## idegree7   39  21 41.75  59       0.70
## idegree8   35  15 34.14  51       0.86
## idegree9   21  14 25.98  38       0.44
## idegree10  12   9 19.11  35       0.12
## idegree11  19   4 13.15  21       0.14
## idegree12   4   3  8.53  16       0.12
## idegree13   7   1  5.25  11       0.58
## idegree14   6   0  3.31   8       0.18
## idegree15   3   0  1.67   6       0.48
## idegree16   4   0  0.82   6       0.06
## idegree17   3   0  0.53   4       0.04
## idegree18   3   0  0.26   2       0.00
## idegree19   2   0  0.10   2       0.02
## idegree20   1   0  0.04   2       0.06
## idegree22   1   0  0.00   0       0.00
##
## Goodness-of-fit for out-degree
##
##           obs min  mean max MC p-value
## odegree0    4   2  6.90  14       0.26
## odegree1   28  10 19.15  30       0.06
## odegree2   45  23 32.45  46       0.02
## odegree3   50  27 44.37  60       0.40
## odegree4   54  34 52.97  71       0.96
## odegree5   62  38 53.24  72       0.18
## odegree6   40  39 51.80  75       0.02
## odegree7   28  32 45.95  61       0.00
## odegree8   13  22 34.51  52       0.00
## odegree9   16  17 27.64  39       0.00
```

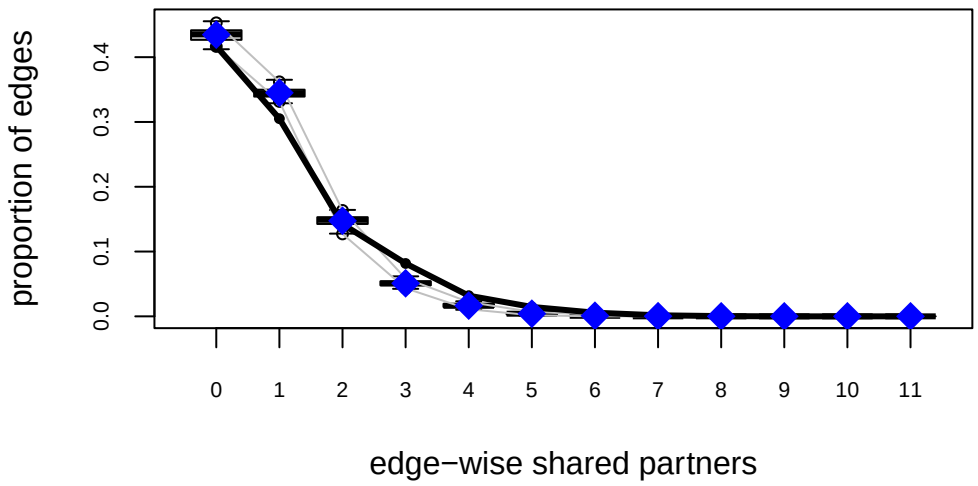
```
## odegree10  20  11 19.13  28       0.92
## odegree11   8   2 12.76  20       0.18
## odegree12  11   2  8.08  14       0.42
## odegree13  13   0  4.18  10       0.00
## odegree14   6   0  2.49   7       0.06
## odegree15   6   0  1.15   4       0.00
## odegree16   7   0  0.68   4       0.00
## odegree17   4   0  0.22   3       0.00
## odegree18   3   0  0.15   1       0.00
## odegree19   0   0  0.06   1       1.00
## odegree20   0   0  0.10   2       1.00
## odegree22   0   0  0.01   1       1.00
## odegree23   0   0  0.01   1       1.00
##
## Goodness-of-fit for edgewise shared partner
##
##           obs min    mean  max MC p-value
## esp.OTP0 1032 991 1046.85 1114       0.60
## esp.OTP1  755 777  830.99  905       0.00
## esp.OTP2  352 288  355.30  414       0.78
## esp.OTP3  202 101  122.85  148       0.00
## esp.OTP4   79  25   39.83   54       0.00
## esp.OTP5   36   3    9.82   26       0.00
## esp.OTP6   14   0    2.26    9       0.00
## esp.OTP7    4   0    0.53    2       0.00
## esp.OTP8    1   0    0.08    1       0.16
##
## Goodness-of-fit for minimum geodesic distance
##
##       obs   min     mean   max MC p-value
## 1    2475  2308  2408.51  2618       0.10
## 2   10672  9717 10526.74 12093       0.62
## 3   31134 32726 36472.28 42060       0.00
## 4   50673 62083 66619.95 71128       0.00
## 5   42563 36051 41272.32 45938       0.44
## 6   18719  6165  9506.31 14061       0.00
```

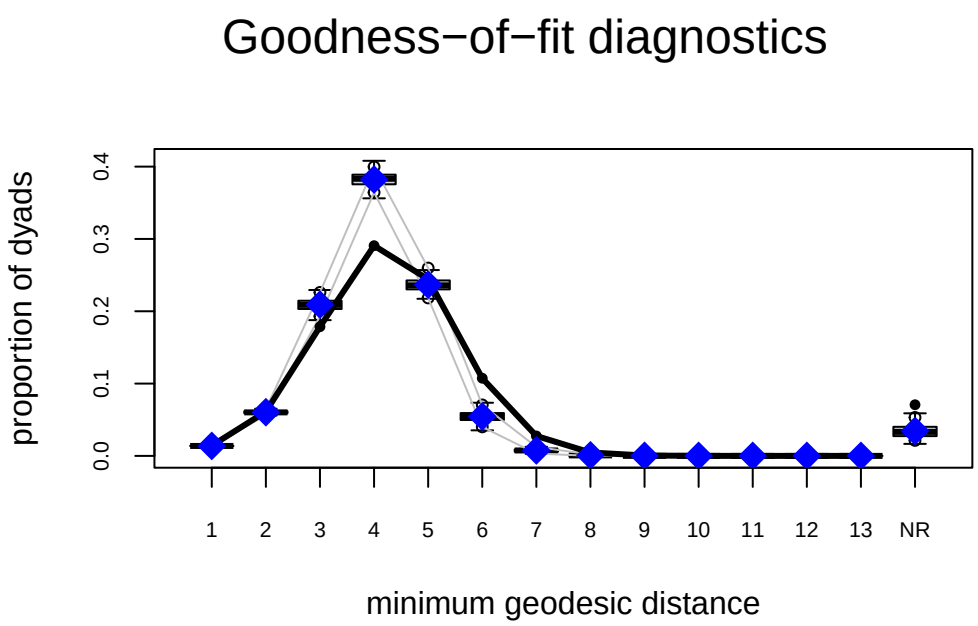
```
## 7    4808   481  1321.87  2588       0.00
## 8     822    11   143.17   505       0.00
## 9     100     0    12.69   269       0.02
## 10      7     0     1.21    84       0.02
## 11      0     0     0.13    12       1.00
## Inf 12333  2904  6020.82 11478       0.00
##
## Goodness-of-fit for model statistics
##
##                              obs       min      mean       max MC p-value
## edges                   2475.000  2308.000  2408.510  2618.000       0.10
## nodematch.hispanic      1832.000  1667.000  1758.880  1860.000       0.08
## nodematch.female1       1879.000  1696.000  1771.860  1957.000       0.04
## nodematch.eversmk1      1755.000  1604.000  1702.570  1880.000       0.24
## mutual                   486.000   428.000   448.360   465.000       0.00
## gwesp.OTP.fixed.0.5     1775.406  1506.382  1601.146  1792.351       0.02
## gwdsp.OTP.fixed.0.5    13187.633 11655.503 12574.732 14495.043       0.12

# Graficando estadísticas GOF
plot(ans_gof)
```




proportion of nodes
0.00
0.05
0.10
0.15
0 1 2 3 4 5 6 7 8 9 11 13 15 17 19 21
odegree


proportion of nodes
0.00
0.05
0.10
0.15
0 2 4 6 8 10 12 14 16 18 20 22 24
idegree





Prueba la siguiente configuración en su lugar

```
ans0_bis <- ergm(
  network_111 ~
    edges +
    nodematch("hispanic") +
    nodematch("female1") +
```

```
    mutual +
    esp(0:3) +
    idegree(0:10)
    ,
  constraints = ~bd(maxout = 19),
  control = control.ergm(
    seed        = 1,
    MCMLE.maxit = 15,
    parallel    = 4,
    CD.maxit    = 15,
    MCMC.samplesize = 2048*4,
    MCMC.burnin = 30000,
    MCMC.interval = 2048*4
    )
  )
```

Aumentar el tamaño de muestra, para que las curvas sean más suaves, intervalos más largos (adelgazamiento), lo que reduce la autocorrelación, y un quemado más grande. Todo esto junto para mejorar la estadística de prueba de Gelman. También agregamos idegree del 0 al 10, y esp del 0 al 3 para coincidir explícitamente con esas estadísticas en nuestro modelo.

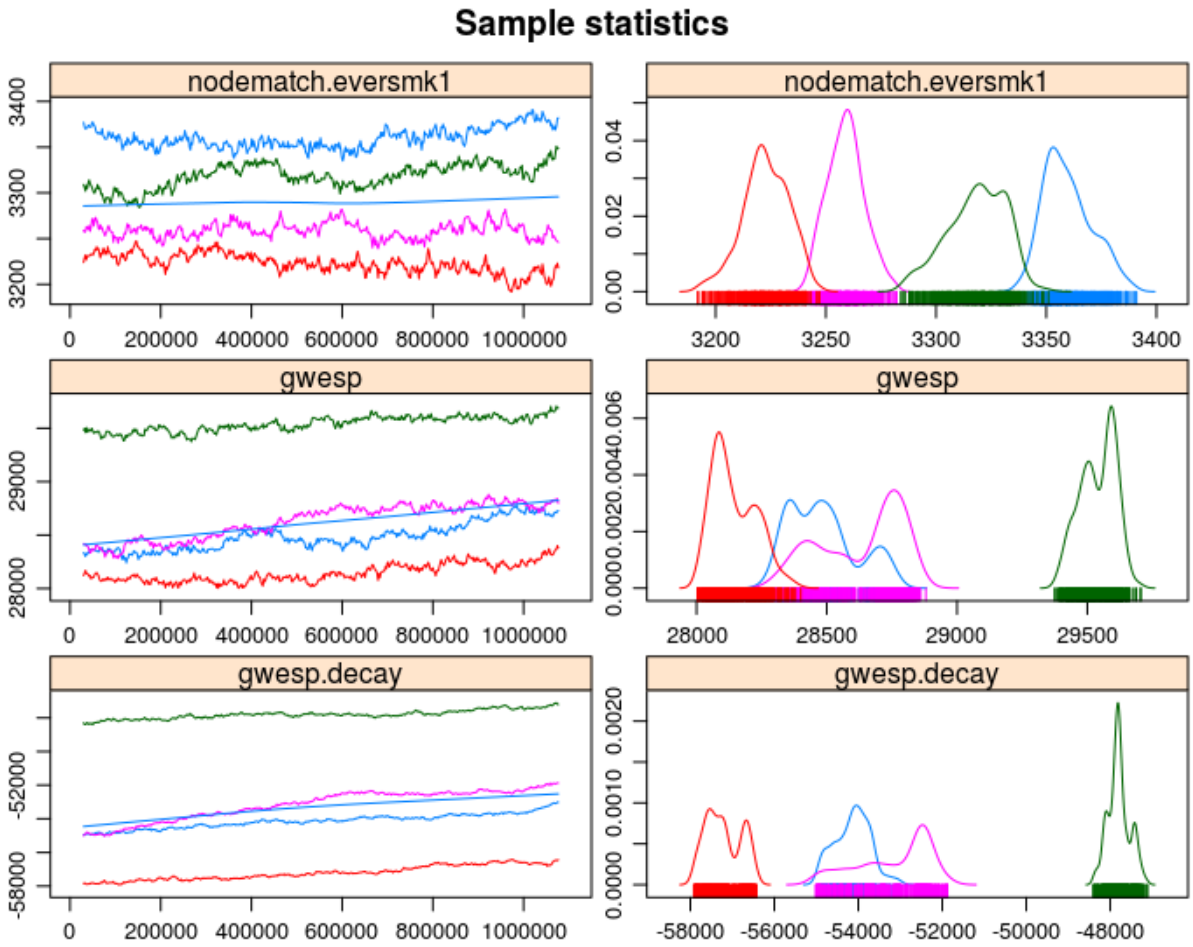


Figure 9.3: Un ejemplo de un ERGM terrible (no hay convergencia en absoluto). También, un buen ejemplo de por qué ejecutar múltiples cadenas puede ser útil

9.6 Más sobre convergencia MCMC

Para más sobre este tema, recomiendo revisar capítulo 1 y capítulo 6 del Manual de MCMC (Brooks et al. 2011). Ambos capítulos están disponibles para descarga gratuita desde el sitio web del libro.

Para GOF echa un vistazo a la sección 6 del tutorial ERGM 2016 Sunbelt, y para una revisión más técnica, puedes echar un vistazo a (David R. Hunter, Goodreau, and Handcock 2008).

9.7 Interpretación Matemática

Una de las partes más críticas del modelado estadístico es interpretar los resultados, si no la más importante. En el caso de ERGMs, un aspecto clave se basa en estadísticas de cambio. Supón que nos gustaría saber qué tan probable es que el vínculo y_{ij} ocurra, dada el resto de la red. Podemos calcular tales probabilidades usando lo que la literatura a veces describe como el muestreador de Gibbs.

En particular, las log-odds del vínculo ij ocurriendo condicional en el resto de la red pueden escribirse como:

$$\text{logit}\left(\Pr\left(y_{ij}=1|y_{-ij}\right)\right)=\theta^{\mathbf{t}}\Delta\delta\left(y_{ij}:0\rightarrow 1\right), \tag{9.2}$$

con $\delta\left(y_{ij}:0\rightarrow 1\right)\equiv s\left(\mathbf{y}\right)_{\text{ij}}^{+}-s\left(\mathbf{y}\right)_{\text{ij}}^{-}$ como el vector de estadísticas de cambio, en otras palabras, la diferencia entre las estadísticas suficientes cuando $y_{ij}=1$ y su valor cuando $y_{ij}=0$. Para mostrar esto, escribimos lo siguiente:

$$\begin{aligned}\Pr\left(y_{ij}=1|y_{-ij}\right)&=\frac{\Pr\left(y_{ij}=1,x_{-ij}\right)}{\mathbb{P}_{+}\left(y_{ij}=1,y_{-ij}\right)\Pr\left(y_{ij}=0,y_{-ij}\right)}\\&=\frac{\exp\left\{\theta^{\mathbf{t}}s\left(\mathbf{y}\right)_{ij}^{+}\right\}}{\exp\left\{\theta^{\mathbf{t}}s\left(\mathbf{y}\right)_{ij}^{+}\right\}+\exp\left\{\theta^{\mathbf{t}}s\left(\mathbf{y}\right)_{ij}^{-}\right\}}\end{aligned}$$

Aplicando la función logit a la ecuación anterior, obtenemos:

$$\begin{aligned}&=\log\left\{\frac{\exp\left\{\theta^{\mathbf{t}}s\left(\mathbf{y}\right)_{ij}^{+}\right\}}{\exp\left\{\theta^{\mathbf{t}}s\left(\mathbf{y}\right)_{ij}^{+}\right\}+\exp\left\{\theta^{\mathbf{t}}s\left(\mathbf{y}\right)_{ij}^{-}\right\}}\right\}-\log\left\{\frac{\exp\left\{\theta^{\mathbf{t}}s\left(\mathbf{y}\right)_{ij}^{-}\right\}}{\exp\left\{\theta^{\mathbf{t}}s\left(\mathbf{y}\right)_{ij}^{+}\right\}+\exp\left\{\theta^{\mathbf{t}}s\left(\mathbf{y}\right)_{ij}^{-}\right\}}\right\}\\&=\log\left\{\exp\left\{\theta^{\mathbf{t}}s\left(\mathbf{y}\right)_{ij}^{+}\right\}\right\}-\log\left\{\exp\left\{\theta^{\mathbf{t}}s\left(\mathbf{y}\right)_{ij}^{-}\right\}\right\}\\&=\theta^{\mathbf{t}}\left(s\left(\mathbf{y}\right)_{ij}^{+}-s\left(\mathbf{y}\right)_{ij}^{-}\right)\\&=\theta^{\mathbf{t}}\Delta\delta\left(y_{ij}:0\rightarrow 1\right)\end{aligned}$$

Por lo tanto, la probabilidad condicional del nodo n ganando función k puede escribirse como:

$$\mathbb{P}_{=}\left(y_{ij}=1|y_{-ij}\right)\frac{1}{1+\exp\left\{-\theta^{\mathbf{t}}\Delta\delta\left(y_{ij}:0\rightarrow1\right)\right\}} \tag{9.3}$$

es decir, una probabilidad logística.

9.8 Independencia de Markov

El desafío de analizar redes es su naturaleza interdependiente. No obstante, en ausencia de tal interdependencia, los ERGMs son equivalentes a regresión logística. Conceptualmente, si todas las estadísticas incluidas en el modelo no involucran dos o más díadas, entonces el modelo es no-Markoviano en el sentido de grafos de Markov.

Matemáticamente, para ver esto, es suficiente mostrar que la probabilidad ERGM puede escribirse como el producto de las probabilidades de cada díada.

$$\mathbb{P}_{=}\left(\mathbf{y}|\theta\right)\frac{\exp\left\{\theta^{\mathbf{t}}s\left(\mathbf{y}\right)\right\}}{\sum_{y}\exp\left\{\theta^{\mathbf{t}}s\left(\mathbf{y}\right)\right\}}=\frac{\prod_{ij}\exp\left\{\theta^{\mathbf{t}}s\left(\mathbf{y}\right)_{ij}\right\}}{\sum_{y}\exp\left\{\theta^{\mathbf{t}}s\left(\mathbf{y}\right)\right\}}$$

Donde $s\left(\right)_{ij}$ es una función tal que $s\left(\mathbf{y}\right)=\sum_{ij}s\left(\mathbf{y}\right)_{ij}$. Ahora necesitamos tratar con la constante normalizadora. Para ver cómo eso puede separarse, comencemos desde el resultado:

$$\begin{aligned}
&=\prod_{ij}\left(1+\exp\left\{\theta^{\mathbf{t}}s\left(\mathbf{y}\right)_{ij}\right\}\right)\\
&=\left(1+\exp\left\{\theta^{\mathbf{t}}s\left(\mathbf{y}\right)_{11}\right\}\right)\left(1+\exp\left\{\theta^{\mathbf{t}}s\left(\mathbf{y}\right)_{12}\right\}\right)...\left(1+\exp\left\{\theta^{\mathbf{t}}s\left(\mathbf{y}\right)_{nn}\right\}\right)\\
&=1+\exp\left\{\theta^{\mathbf{t}}s\left(\mathbf{y}\right)_{11}\right\}+\exp\left\{\theta^{\mathbf{t}}s\left(\mathbf{y}\right)_{11}\right\}\exp\left\{\theta^{\mathbf{t}}s\left(\mathbf{y}\right)_{12}\right\}+\cdots+\prod_{ij}\exp\left\{\theta^{\mathbf{t}}s\left(\mathbf{y}\right)_{ij}\right\}\\
&=1+\exp\left\{\theta^{\mathbf{t}}s\left(\mathbf{y}\right)_{11}\right\}+\exp\left\{\theta^{\mathbf{t}}\left(s\left(\mathbf{y}\right)_{11}+s\left(\mathbf{y}\right)_{12}\right)\right\}+\cdots+\prod_{ij}\exp\left\{\theta^{\mathbf{t}}s\left(\mathbf{y}\right)_{ij}\right\}\\
&=\sum_{\mathbf{y}\in\mathcal{Y}}\exp\left\{\theta^{\mathbf{t}}s\left(\mathbf{y}\right)\right\}
\end{aligned}$$

Donde la última igualdad sigue del hecho de que la suma *es* la suma sobre todas las combinaciones posibles de redes, comenzando desde $exp(0)=1$, hasta $exp(all)$. De esta manera, ahora podemos escribir:

$$\frac{\prod_{ij} \exp\left\{\theta^{\mathbf{t}} s\left(\mathbf{y}\right)_{ij}\right\}}{\sum_{y} \exp\left\{\theta^{\mathbf{t}} s\left(\mathbf{y}\right)\right\}} = \prod_{ij} \frac{\exp\left\{\theta^{\mathbf{t}} s\left(\mathbf{y}\right)_{ij}\right\}}{1 + \exp\left\{\theta^{\mathbf{t}} s\left(\mathbf{y}\right)_{ij}\right\}} \tag{9.4}$$

Relacionado con esto, los ERGMs bloque-diagonales pueden estimarse como modelos independientes, uno por bloque. Para ver más sobre esto, lee (SNIJDERS 2010). De manera similar, dado que la independencia depende–juego de palabras intencionado–de particionar la función objetivo, como señala Snijders, las funciones no lineales hacen que el modelo sea dependiente, ej., $s\left(\mathbf{y}\right) = \sqrt{\sum_{ij} y_{ij}}$, la raíz cuadrada del conteo de enlaces ya no es un grafo de Bernoulli.

10 Usando restricciones en ERGMs

> ⚠ Nota de Traducción
>
> Esta versión del capítulo fue traducida de manera automática utilizando IA. El capítulo aún no ha sido revisado por un humano.

Los Modelos Exponenciales de Grafos Aleatorios [ERGMs] pueden representar una variedad de clases de redes. A menudo observamos redes sociales "regulares" como estudiantes en escuelas, colegas en el lugar de trabajo, o familias. No obstante, algunas redes sociales que estudiamos tienen características que restringen cómo pueden ocurrir las conexiones. Ejemplos típicos son grafos bi-partitos y redes multinivel. Hay dos clases de vértices en redes bi-partitas, y los vínculos solo pueden ocurrir entre clases. Por otro lado, las redes multinivel pueden presentar múltiples clases con vínculos entre clases algo restringidos. En ambos casos, existen restricciones estructurales, lo que significa que algunas configuraciones pueden no ser plausibles.

Matemáticamente, lo que estamos tratando de hacer es, en lugar de asumir que todas las configuraciones de red son posibles:

$$\{\mathbf{y} \in \mathcal{Y} : y_{ij} = 0, \forall i = j\}$$

queremos ir un poco más allá evitando bucles, a saber:

$$\{\mathbf{y} \in \mathcal{Y} : y_{ij} = 0, \forall i = j; \mathbf{y} \in C\}$$

,

donde C es una restricción, por ejemplo, solo redes sin triángulos. El paquete de R `ergm` tiene capacidades incorporadas para lidiar con algunos de estos casos. No obstante, podemos

especificar modelos con restricciones estructurales arbitrarias incorporadas en el modelo. La clave está en usar términos de offset.

10.1 Ejemplo 1: Egos entrelazados y alters desconectados

Imagina que tenemos dos conjuntos de vértices. El primero, grupo `E`, son egos parte de un estudio egocéntrico. El segundo grupo, llamado `A`, está compuesto por personas mencionadas por egos en `E` pero que no fueron encuestadas. Asume que individuos en `A` solo pueden conectarse a individuos en `E`; además, individuos en `E` no tienen restricciones para conectarse. En otras palabras, solo existen dos tipos de vínculos: `E-E` y `A-E`. La pregunta es ahora, ¿cómo podemos aplicar tal restricción en un ERGM?

Usar offsets, y en particular, establecer coeficientes a `-Inf` proporciona una forma fácil de restringir el conjunto de soporte de ERGMs. Por ejemplo, si quisiéramos restringir el soporte para incluir redes sin triángulos, agregaríamos el término `offset(triangle)` y usaríamos la opción `offset.coef = -Inf` para indicar que las realizaciones que incluyen triángulos no son posibles. Usando R:

```
ergm(net ~ edges + offset(triangle), offset.coef = -Inf)
```

En este modelo, un grafo de Bernoulli, reducimos el espacio muestral a redes sin triángulos. En nuestro ejemplo, tal estadística solo debería tomar valores no cero cuando los vínculos dentro de la clase `A` ocurran. Podemos usar el término `nodematch()` para hacer eso. Formalmente

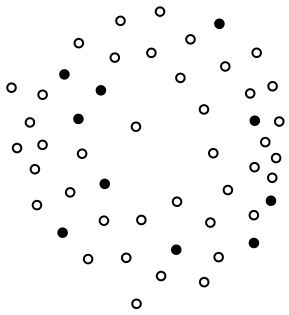
$$\text{NodeMatch}(x) = \sum_{i,j} y_{ij} \mathbf{1}(x_i = x_j)$$

Esta estadística sumará sobre todos los vínculos en los que el atributo X de la fuente (i) y el objetivo (j) son iguales. Una forma de hacer que esto suceda es creando una variable auxiliar que sea igual a, p. ej., 0 para todos los vértices en `A`, y un valor único diferente de cero en caso contrario. Por ejemplo, si tuviéramos 2 `A`s y tres `E`s, los datos se verían algo así: $\{0, 0, 1, 2, 3\}$. El siguiente bloque de código crea un grafo vacío con 50 nodos, 10 de los cuales están en el grupo `E` (ego).

```r
library(ergm, quietly =  TRUE)
library(sna, quietly =  TRUE)

n <- 50
n_egos <- 10
net <- as.network(matrix(0, ncol = n, nrow = n), directed = TRUE)

# Asignemos los grupos
net %v% "is.ego" <- c(rep(TRUE, n_egos), rep(FALSE, n - n_egos))
net %v% "is.ego"
```

```
 [1]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE FALSE
[13] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[25] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[37] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[49] FALSE FALSE
```

```r
gplot(net, vertex.col = net %v% "is.ego")
```



Para crear la variable auxiliar, usaremos la siguiente función:

```r
# Función que crea una variable auxiliar para el modelo ergm
make_aux_var <- function(my_net, is_ego_dummy) {

  n_vertex <- length(my_net %v% is_ego_dummy)
  n_ego_   <- sum(my_net %v% is_ego_dummy)

  # Creando una variable auxiliar para identificar los vínculos no-informante no-informa
  my_net %v% "aux_var" <- ifelse(
    !my_net %v% is_ego_dummy, 0, 1:(n_vertex - n_ego_)
    )

  my_net
}
```

Llamar la función en nuestros datos resulta en lo siguiente:

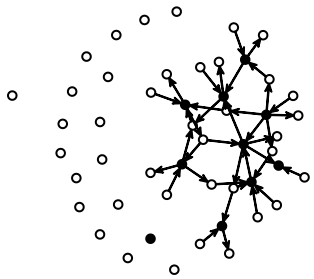
```r
net <- make_aux_var(net, "is.ego")

# Echando un vistazo a las primeras 15 filas de datos
cbind(
  Is_Ego = net %v% "is.ego",
  Aux    = net %v% "aux_var"
) |> head(n = 15)
```

```
      Is_Ego Aux
 [1,]      1   1
 [2,]      1   2
 [3,]      1   3
 [4,]      1   4
 [5,]      1   5
 [6,]      1   6
 [7,]      1   7
 [8,]      1   8
 [9,]      1   9
[10,]      1  10
[11,]      0   0
```

```
[12,]    0   0
[13,]    0   0
[14,]    0   0
[15,]    0   0
```

Ahora podemos usar estos datos para simular una red en la cual los vínculos entre vértices de clase `A` no son posibles:

```
set.seed(2828)
net_sim <- simulate(net ~ edges + nodematch("aux_var"), coef = c(-3.0, -Inf))
gplot(net_sim, vertex.col = net_sim %v% "is.ego")
```



Como puedes ver, esta red solo tiene vínculos del tipo `E-E` y `A-E`. Podemos verificar (i) viendo los conteos y (ii) visualizando cada subgrafo inducido por separado:

```
summary(net_sim ~ edges + nodematch("aux_var"))
```

```
           edges nodematch.aux_var
              39                 0
```

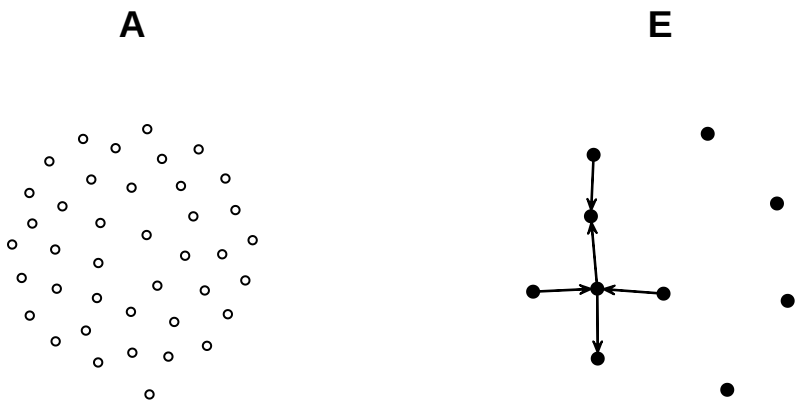
```r
net_of_alters <- get.inducedSubgraph(
  net_sim, which((net_sim %v% "aux_var") == 0)
  )

net_of_egos <- get.inducedSubgraph(
  net_sim, which((net_sim %v% "aux_var") != 0)
  )

# Conteos
summary(net_of_alters ~ edges + nodematch("aux_var"))
```

```
            edges nodematch.aux_var
                0                 0
```

```r
summary(net_of_egos ~ edges + nodematch("aux_var"))
```

```
            edges nodematch.aux_var
                5                 0
```

```r
# Figuras
op <- par(mfcol = c(1, 2))
gplot(net_of_alters, vertex.col = net_of_alters %v% "is.ego", main = "A")
gplot(net_of_egos, vertex.col = net_of_egos %v% "is.ego", main = "E")
```




```
par(op)
```

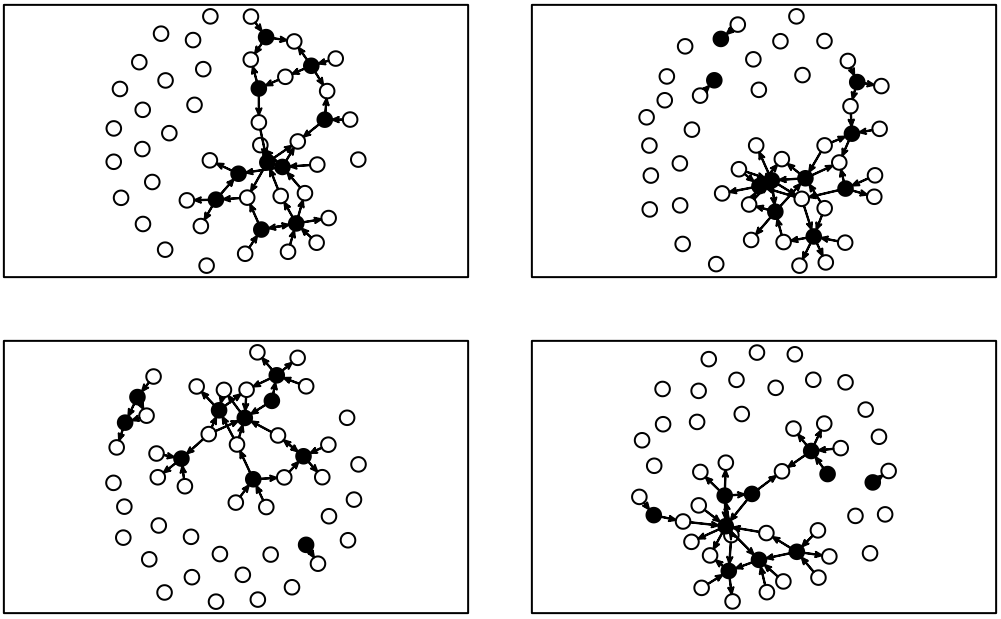
Ahora, para ajustar un ERGM con esta restricción, simplemente necesitamos hacer uso de los términos offset. Aquí hay un ejemplo:

```
ans <- ergm(
  net_sim ~ edges + offset(nodematch("aux_var")), # El modelo (nota el offset)
  offset.coef = -Inf                              # El coeficiente offset
  )
## Starting maximum pseudolikelihood estimation (MPLE):
## Obtaining the responsible dyads.
## Evaluating the predictor and response matrix.
## Maximizing the pseudolikelihood.
## Finished MPLE.
## Evaluating log-likelihood at the estimate.
summary(ans)
## Call:
## ergm(formula = net_sim ~ edges + offset(nodematch("aux_var")),
##     offset.coef = -Inf)
##
## Maximum Likelihood Results:
```

```
##
##                          Estimate Std. Error MCMC % z value Pr(>|z|)
## edges                     -3.0829     0.1638      0  -18.83   <1e-04 ***
## offset(nodematch.aux_var)     -Inf     0.0000      0    -Inf   <1e-04 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
##      Null Deviance: 3396.4  on 2450  degrees of freedom
##  Residual Deviance:  320.2  on 2448  degrees of freedom
##
## AIC: 322.2  BIC: 327  (Smaller is better. MC Std. Err. = 0)
##
##  The following terms are fixed by offset and are not estimated:
##   offset(nodematch.aux_var)
```

Este modelo ERGM–que por cierto solo presentaba términos diádicos independientes, y por lo tanto puede reducirse a una regresión logística–restringe el soporte excluyendo todas las redes en las que existen vínculos dentro de la clase `A`. Para finalizar, veamos algunas simulaciones basadas en este modelo:

```
set.seed(1323)
op <- par(mfcol = c(2,2), mar = rep(1, 4))
for (i in 1:4) {
  gplot(simulate(ans), vertex.col = net %v% "is.ego", vertex.cex = 2)
  box()
}
```



```
par(op)
```

Todas las redes sin vínculos entre nodos `A`.

10.2 Ejemplo 2: Redes bi-partitas

En el caso de redes bipartitas (a veces llamadas redes de afiliación,) podemos usar los términos de `ergm` para grafos bipartitos para corroborar lo que discutimos aquí. Por ejemplo, el término de dos estrellas. Comencemos simulando una red bipartita usando los parámetros `edges` y `two-star`. Dado que el término `k-star` usualmente es complejo de ajustar (tiende a generar modelos degenerados,) aprovecharemos la función de transformación `Log()` en el paquete `ergm` para suavizar el término.[1]

La red bipartita que estaremos simulando tendrá 100 actores y 50 entidades. Los actores, que mapearemos al primer nivel de los términos `ergm`, esto es, `b1star b1nodematch`, etc.

[1]Después de escribir este ejemplo, se hizo aparente que el uso de la función de transformación `Log()` puede no ser ideal. Dado que muchos términos usados en ERGMs pueden ser cero, p. ej., triángulos, el término `Log(~ ostar(2))` está indefinido cuando `ostar(2) = 0`. En la práctica, el paquete ERGM establece un límite inferior para el log de 0, así que, en lugar de tener `Log(0) ~ -Inf`, lo establecen como un número negativo realmente grande. Esto causa todo tipo de problemas a las estimaciones; en nuestro ejemplo, una sobreestimación del parámetro poblacional y una log-verosimilitud positiva. Por lo tanto, no recomendaría usar esta transformación muy a menudo.

enviarán vínculos a las entidades, el segundo nivel del ERGM bipartito. Para crear una red bipartita, crearemos una matriz vacía de tamaño `nactors x nentitites`; por lo tanto, los actores están representados por filas y las entidades por columnas.

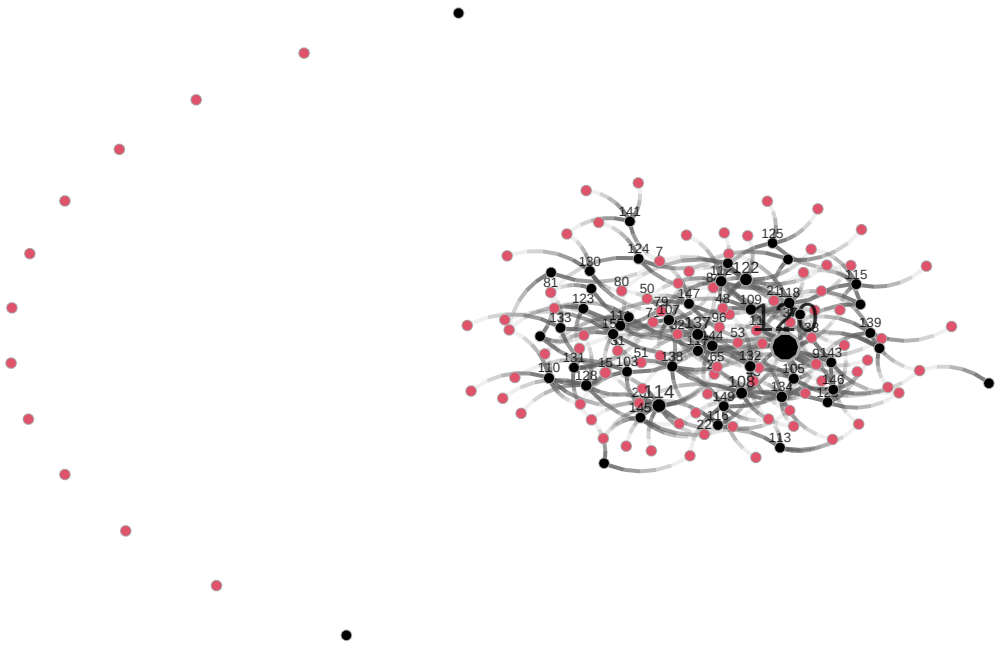
```
# Parámetros para la simulación
nactors   <- 100
nentities <- floor(nactors/2)
n         <- nactors + nentities

# Creando una red bipartita vacía (línea base)
net_b <- network(
  matrix(0, nrow = nactors, ncol = nentities), bipartite = TRUE
)

# Simulando el ERGM bipartito,
net_b <- simulate(net_b ~ edges + Log(~b1star(2)), coef = c(-3, 1.5), seed = 55)
```

Veamos qué obtuvimos aquí:

```
summary(net_b ~ edges + Log(~b1star(2)))
```

```
      edges Log~b1star2
 245.000000    5.746203
```

```
netplot::nplot(net_b, vertex.col = (1:n <= nactors) + 1)
```



Nota que los primeros `nactors` vértices en la red son los actores, y los restantes son las entidades. Ahora, aunque el paquete `ergm` presenta términos de red bipartita, aún podemos ajustar un ERGM bipartito sin declarar explícitamente el grafo como tal. En tal caso, el término `b1star(2)` de una red bipartita es equivalente a un `ostar(2)` en un grafo dirigido. De manera similar, `b2star(2)` en un grafo bipartito coincide con el término `istar(2)` en un grafo dirigido. Esta información será relevante al ajustar el ERGM. Transformemos la red bipartita en un grafo dirigido. El siguiente bloque de código hace eso:

```
# Identificando los enlaces
net_not_b <- which(as.matrix(net_b) != 0, arr.ind = TRUE)

# Necesitamos compensar el punto final de los vínculos por nactors
# para que los ids vayan de 1 hasta (nactors + nentitites)
net_not_b[,2] <- net_not_b[,2] + nactors

# El grafo resultante es una red dirigida
net_not_b <- network(net_not_b, directed = TRUE)
```

Ahora casi hemos terminado. Como antes, necesitamos usar covariables a nivel de nodo para

poner las restricciones en nuestro modelo. Para que este ERGM refleje un ERGM en una red bipartita, necesitamos dos restricciones:

1. Solo se permiten vínculos de actores a entidades, y
2. las entidades solo pueden recibir vínculos.

Los términos offset correspondientes para este modelo son: `nodematch("is.actor") ~ -Inf`, y `nodeocov("isnot.actor") ~ -Inf`. Matemáticamente:

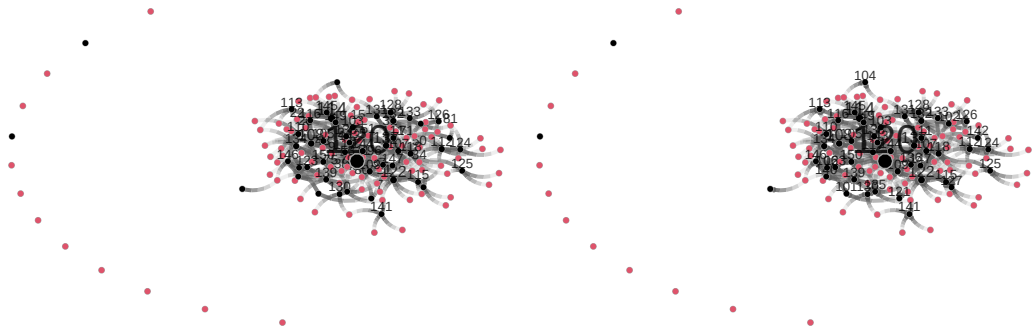
$$\text{NodeMatch}(\text{x} = \text{"is.actor"}) = \sum_{i<j} y_{ij} \mathbb{1}\left(x_i = x_j\right)$$
$$\text{NodeOCov}(\text{x} = \text{"isnot.actor"}) = \sum_{i} x_i \times \sum_{j<i} y_{ij}$$

En otras palabras, estamos estableciendo que los vínculos entre nodos de la misma clase están prohibidos, y los vínculos salientes están prohibidos para las entidades. Creemos los atributos de vértice necesarios para usar los términos mencionados:

```
net_not_b %v% "is.actor" <- as.integer(1:n <= nactors)
net_not_b %v% "isnot.actor" <- as.integer(1:n > nactors)
```

Finalmente, para asegurarnos de que hemos hecho todo bien, veamos cómo ambas redes–bipartita y unimodal–se ven lado a lado:

```
# Primero, obtengamos el diseño
fig <- netplot::nplot(net_b, vertex.col = (1:n <= nactors) + 1)
gridExtra::grid.arrange(
  fig,
  netplot::nplot(
    net_not_b, vertex.col = (1:n <= nactors) + 1,
    layout = fig$.layout
     ),
  ncol = 2, nrow = 1
)
```



```
# Viendo los conteos
summary(net_b ~ edges + b1star(2) + b2star(2))
```

```
  edges b1star2 b2star2
    245     313     645
```

```
summary(net_not_b ~ edges + ostar(2) + istar(2))
```

```
 edges ostar2 istar2
   245    313    645
```

Con las dos redes coincidiendo, ahora podemos ajustar los ERGMs con y sin términos offset y comparar los resultados entre los dos modelos:

```
# ERGM con un grafo bipartito
res_b     <- ergm(
  # Fórmula principal
  net_b ~ edges + Log(~b1star(2)),

  # Parámetros de control
```

```
  control = control.ergm(seed = 1)
  )
```

```
Warning: 'glpk' selected as the solver, but package 'Rglpk' is not available;
falling back to 'lpSolveAPI'. This should be fine unless the sample size and/or
the number of parameters is very big.
```

```
# ERGM con un digrafo con restricciones
res_not_b <- ergm(
  # Fórmula principal
  net_not_b ~ edges + Log(~ostar(2)) +

  # Términos offset
  offset(nodematch("is.actor")) + offset(nodeocov("isnot.actor")),
  offset.coef = c(-Inf, -Inf),

  # Parámetros de control
  control = control.ergm(seed = 1)
  )
```

Aquí están las estimaciones (usando el paquete de R `texreg` para una salida más bonita):

```
texreg::screenreg(list(Bipartite = res_b, Directed = res_not_b))
```

```
=========================================================
                              Bipartite    Directed
---------------------------------------------------------
edges                          -3.14 ***    -3.14 ***
                               (0.15)       (0.15)
Log~b1star2                    21.89
                              (17.13)
Log~ostar2                                   22.40
                                            (16.39)
offset(nodematch.is.actor)                    -Inf
```

```
offset(nodeocov.isnot.actor)                  -Inf

---------------------------------------------------
AIC                            1958.00      1957.57
BIC                            1971.03      1973.60
Log Likelihood                 -977.00      -976.78
===================================================
*** p < 0.001; ** p < 0.01; * p < 0.05
```

Como se esperaba, ambos modelos producen la "misma" estimación. Las diferencias menores observadas entre los modelos son cómo el paquete `ergm` realiza el muestreo. En particular, en el caso bipartito, `ergm` tiene rutinas especiales para hacer el muestreo más eficiente, teniendo una tasa de aceptación más alta que la del modelo en el que el grafo bipartito no fue explícitamente declarado. Podemos decir esto inspeccionando las tasas de rechazo:

```
data.frame(
  Bipartite = coda::rejectionRate(res_b$sample[[1]]) * 100,
  Directed  = coda::rejectionRate(res_not_b$sample[[1]][, -c(3,4)]) * 100
) |> knitr::kable(digits = 2, caption = "Tasa de rechazo (porcentaje)")
```

Table 10.1: Tasa de rechazo (porcentaje)

	Bipartite	Directed
edges	2.48	4.47
Log~b1star2	1.24	1.68

El ERGM ajustado con los términos offset tiene una tasa de rechazo mucho más alta que la del ERGM ajustado con el ERGM bipartito.

Finalmente, el hecho de que podamos ajustar ERGMs usando offset no significa que necesitemos usarlo TODO el tiempo. A menos que haya una muy buena razón para evitar las capacidades de `ergm`, no recomendaría ajustar ERGMs bipartitos como acabamos de hacer, ya que los autores del paquete han incluido (MUCHAS) características para hacer nuestro trabajo más fácil.

11 Modelos de Grafos Aleatorios de Familia Exponencial Temporal

> ⚠ Nota de Traducción
>
> Esta versión del capítulo fue traducida de manera automática utilizando IA. El capítulo aún no ha sido revisado por un humano.

¡Este tutorial es genial! https://statnet.org/trac/raw-attachment/wiki/Sunbelt2016/tergm_tutorial.pdf

12 Pruebas de hipótesis en redes

> ⚠ Nota de Traducción
>
> Esta versión del capítulo fue traducida de manera automática utilizando IA. El capítulo aún no ha sido revisado por un humano.

En general, hay muchas formas en las que podemos ver las pruebas de hipótesis dentro del contexto de redes:

1. **Comparar dos o más redes**, p. ej., queremos ver si la densidad de dos redes son *iguales*.
2. **Prevalencia de un motivo/patrón**, p. ej., verificar si el número observado de tríadas transitivas es diferente del esperado por casualidad.
3. **Multivariado usando ERGMs**, p. ej., probar conjuntamente si la homofilia y las estrellas de dos puntos son los motivos que impulsan la estructura de red.

Esta última ya la revisamos en el capítulo de ERGM. En esta parte, veremos los tipos uno y dos; ambos usando métodos no paramétricos.

12.1 Comparando redes

Imagina que tenemos dos grafos, $(G_1, G_2) \in \mathcal{G}$, y nos gustaría evaluar si una estadística dada $s(\cdot)$, p. ej., densidad, es igual en ambos. Formalmente, nos gustaría evaluar si $H_0 : s(G_1) - s(G_2) = k$ vs $H_a : s(G_1) - s(G_2) \neq k$.

Como es usual, la distribución verdadera de $s(\cdot)$ es desconocida, por lo tanto, un enfoque que podríamos usar es una prueba bootstrap no paramétrica.

12.1.1 Bootstrap de redes

Los métodos bootstrap no paramétrico y jackknife para redes sociales fueron introducidos por (Tom A. B. Snijders and Borgatti 1999). El método mismo se usa para generar errores estándar para estadísticas a nivel de red. Ambos métodos están implementados en el paquete de R `netdiffuseR`.

12.1.2 Cuando la estadística es normal

Cuando tratamos con cosas que están distribuidas normalmente, p. ej., medias muestrales como la densidad[1], podemos hacer uso de la distribución de Student para hacer inferencia. En particular, podemos usar Bootstrap/Jackknife para aproximar los errores estándar de la estadística para cada red:

1. Dado que $s(G_i) \sim \mathrm{N}(\mu_i, \sigma_i^2/m_i)$ para $i \in \{1, 2\}$, en el caso de la densidad, $m_i = n_i * (n_i - 1)$. La estadística es entonces:

   $$s(G_1) - s(G_0) \sim \mathrm{N}(\mu_1 - \mu_0, \sigma_1^2/m_1 + \sigma_1^2/m_2)$$

   Por lo tanto

   $$\frac{s(G_1) - s(G_0) - \mu_1 + \mu_2}{\sqrt{\sigma_1^2/m_1 + \sigma_1^2/m_2}} \sim t_{m_1+m_2-2}$$

   Pero, si estamos probando $H_0 : \mu_1 - \mu_2 = k$, entonces, bajo la nula

   $$\frac{s(G_1) - s(G_0) - k}{\sqrt{\sigma_1^2/m_1 + \sigma_1^2/m_2}} \sim t_{m_1+m_2-2}$$

   Donde ahora procedemos a aproximar las varianzas.

2. Usando el *principio plugin* (Efron and Tibshirani 1994), podemos aproximar las varianzas usando Bootstrap/Jackknife, es decir, calcular $\hat{\sigma}_1^2 \approx \sigma_1^2/m_1$ y $\hat{\sigma}_2^2 \approx \sigma_2^2/m_2$. Usando netdiffuseR

[1] La densidad es en efecto una media muestral ya que estamos, en principio calculando el promedio de una secuencia de variables de Bernoulli. Formalmente: $\text{densidad}(G) = \frac{1}{n(n-1)} \sum_{ij} A_{ij}$.

```r
library(netdiffuseR)

# Obtener 100 réplicas
sg1 <- bootnet(g1, function(i, ...) sum(i)/(nnodes(i) * (nnodes(i) - 1)), R = 100)
sg2 <- bootnet(g2, function(i, ...) sum(i)/(nnodes(i) * (nnodes(i) - 1)), R = 100)

# Recuperando las varianzas
hat_sigma1 <- sg1$var_t
hat_sigma2 <- sg2$var_t

# Y los valores reales
sg1 <- sg1$t0
sg2 <- sg2$t0
```

3. Con las aproximaciones en mano, podemos entonces usar la "tabla de prueba t" para recuperar el valor correspondiente, en R:

```r
# Construyendo la estadística
k <- 0 # Para varianzas iguales
tstat <- (sg1 - sg2 - k)/(sqrt(hat_sigma1 + hat_sigma2))

# Calculando el valor p
m1 <- nnodes(g1)*(nnodes(g1) - 1)
m2 <- nnodes(g2)*(nnodes(g2) - 1)
pt(tstat, df = m1 + m2 - 2)
```

12.1.3 Cuando la estadística NO es normal

En el caso de que la estadística no esté distribuida normalmente, ya no podemos usar la estadística t. Sin embargo, el Bootstrap puede venir a ayudar. Aunque en general es mejor usar distribuciones de estadísticas pivote (ver (Efron and Tibshirani 1994)), aún podemos aprovechar el poder de este método para hacer inferencias. Para este ejemplo, $s(\cdot)$ será el rango del umbral en un grafo de difusión.

Como antes, imagina que estamos tratando con una estadística $s(\cdot)$ para dos redes diferentes, y nos gustaría evaluar si podemos rechazar H_0 o fallar en rechazarla. El procedimiento es

muy similar:

1. Un enfoque que podemos probar es si $k \in \text{ConfInt}(s(G_1) - s(G_2))$. Construir intervalos de confianza con bootstrap podría ser más intuitivo.

2. Como antes, usamos bootstrap para generar una distribución de $s(G_1)$ y $s(G_2)$, en R:

```
# Obtener 1000 réplicas
sg1 <- bootnet(g1, function(i, ...) range(threshold(i)), R = 1000)
sg2 <- bootnet(g2, function(i, ...) range(threshold(i)), R = 1000)

# Recuperando las distribuciones
sg1 <- sg1$boot$t
sg2 <- sg2$boot$t

# Definir la estadística
sdiff <- sg1 - sg2
```

3. Una vez que tenemos `sdiff`, podemos proceder y calcular el, por ejemplo, 95% intervalo de confianza, y evaluar si k cae dentro. En R:

```
diff_ci <- quantile(sdiff, probs = c(0.025, .975))
```

Esto corresponde a lo que Efron y Tibshirani llaman "intervalo de percentil." Esto es fácil de calcular, pero un mejor enfoque es usar el método "BCa", "Corregido por Sesgo y Acelerado." (TBD)

12.2 Ejemplos

12.2.1 Promedio de estadísticas a nivel de nodo

Supón que nos gustaría comparar algo como el grado de entrada promedio. En particular, para ambas redes, G_1 y G_2, calculamos el grado de entrada promedio por nodo:

$$s(G_1) = \text{GradoEntProm}(G_1) = \frac{1}{n} \sum_i \sum_{j \neq i} A^1_{ji}$$

donde A^1_{ji} es igual a uno si el vértice j envía un vínculo a i. En este caso, dado que estamos viendo un promedio, tenemos que $\text{GradoEntProm}(G_1) \sim N(\mu_1, \sigma_1^2/n)$. Por lo tanto, aprovechando la normalidad de la estadística, podemos construir una estadística de prueba como sigue:

$$\frac{s(G_1) - s(G_2) - k}{\sqrt{\hat{\sigma}_1^2 + \hat{\sigma}_2^2}} \sim t_{n_1+n_2-2}$$

Donde $\hat{\sigma}_i$ es el error estándar bootstrap, y $k = 0$ cuando estamos probando igualdad. Esto se distribuye t con $n_1 + n_2 - 2$ grados de libertad. Como diferencia del ejemplo anterior usando densidad, los grados de libertad para esta prueba son menores ya que, en lugar de tener un promedio a través de todas las entradas de la matriz de adyacencia, tenemos un promedio a través de todos los vértices.

13 Modelos Estocásticos Orientados al Actor

> **Nota de Traducción**
>
> Esta versión del capítulo fue traducida de manera automática utilizando IA. El capítulo aún no ha sido revisado por un humano.

Los Modelos Estocásticos Orientados al Actor (SOAM), también conocidos como modelos Siena fueron introducidos por CITATION NEEDED.

Como diferencia de los ERGMs, los modelos Siena observan el proceso de generación de datos desde el punto de vista de los individuos. Basado en las ideas de McFadden sobre elección probabilística, el modelo se fundamenta en la siguiente ecuación

$$U_i(x) - U_i(x') \sim \text{Distribución de Valor Extremo}$$

En otras palabras, los individuos eligen entre estados x y x' de manera probabilística (con algo de ruido),

$$\frac{\exp\left\{f_i^Z(\beta^z, x, z)\right\}}{\sum_{Z' \in \mathcal{C}} \exp\left\{f_i^Z(\beta, x, z')\right\}}$$

snijders_(sociological methodology 2001)

Ripley et al. (2011)

14 Cálculo de poder en estudios de redes

> ⚠ Nota de Traducción
>
> Esta versión del capítulo fue traducida de manera automática utilizando IA. El capítulo aún no ha sido revisado por un humano.

En el diseño de encuestas y estudios, calcular el tamaño de muestra requerido es crítico. Hoy en día, abundan las herramientas y métodos para calcular el tamaño de muestra requerido; no obstante, el cálculo de tamaño de muestra para estudios que involucran redes sociales aún está subdesarrollado. Este capítulo ilustrará cómo podemos usar simulaciones por computadora para estimar el tamaño de muestra requerido. El Capítulo Chapter 17 proporciona una vista general del análisis de poder.

14.1 Ejemplo 1: Efectos de derrame en estudios egocéntricos[1]

Supón que queremos ejecutar una intervención sobre una población particular, y estamos interesados en los efectos de tal intervención en los alters de los egos. En economía, este problema, que llaman el "efecto de derrame," se estudia activamente.

Asumimos que los alters solo se exponen si los egos adquieren el comportamiento para el cálculo de poder. Además, para esta primera ejecución, asumiremos que no hay refuerzo social o influencia entre alters. Posteriormente relajaremos esta suposición. Para calcular el poder, haremos lo siguiente:

1. Simular el comportamiento de los egos siguiendo una distribución logit.

[1] El problema original fue planteado por Dr. Shinduk Lee de la Escuela de Enfermería de la Universidad de Utah.

2. Eliminar aleatoriamente algunos egos como resultado de la deserción.
3. Simular el comportamiento de los alters usando sus egos como el tratamiento.
4. Ajustar una regresión logística basada en el modelo anterior.
5. Aceptar/rechazar la nula y almacenar el resultado.

Los pasos anteriores se repetirán 500 veces para cada valor de n que analicemos. Finalizaremos graficando el poder contra los tamaños de muestra. Comencemos primero escribiendo los parámetros de simulación:

```
# Diseño
n_sims    <- 500 # Número de simulaciones
n_a       <- 4   # Número de alters
sizes     <-     # Tamaños a probar
  seq(from = 100, to = 200, by = 25)

# Suposiciones
odds_h_1  <- 2.0 # Odds de Aumento/
attrition <- .3
baseline  <- .2  # Baja prevalencia en 1s

# Parámetros
alpha    <- .05
beta_pow <- 0.2
```

Como discutimos en Chapter 17, siempre es una buena idea encapsular la simulación en una función:

```
# Los odds convertidos a una prob
theta_h_1 <- plogis(log(odds_h_1))

# Función de simulación
sim_data <- function(n) {

  # Asignación de tratamiento
  tr  <- c(rep(1, n/2), rep(0, n/2))
```

```r
  # Paso 1: Muestreando población de egos
  y_ego <- runif(n) < c(
    rep(theta_h_1, n/2),
    rep(0.5, n/2)
  )

  # Paso 2: Simulando deserción
  todrop <- order(runif(n))[1:(n * attrition)]
  y_ego  <- y_ego[-todrop]
  tr     <- tr[-todrop]
  n      <- n - length(todrop)

  # Paso 3: Simulando efecto del alter. Asumimos lo mismo que en
  # ego
  tr_alter <- rep(y_ego * tr, n_a)
  y_alter  <- runif(n * n_a) < ifelse(tr_alter, theta_h_1, 0.5)

  # Paso 4: Calculando estadística de prueba
  res_ego   <- tryCatch(glm(y_ego ~ tr, family = binomial("logit")), error = function(e)
  res_alter <- tryCatch(glm(y_alter ~ tr_alter, family = binomial("logit")), error = fun

  if (inherits(res_ego, "error") | inherits(res_alter, "error"))
    return(c(ego =  NA, alter = NA))

  # Paso 5: ¿Rechazar?
  c(
    ego   = summary(res_ego)$coefficients["tr", "Pr(>|z|)"] < alpha,
    alter = summary(res_alter)$coefficients["tr_alter", "Pr(>|z|)"] < alpha
  )


}
```

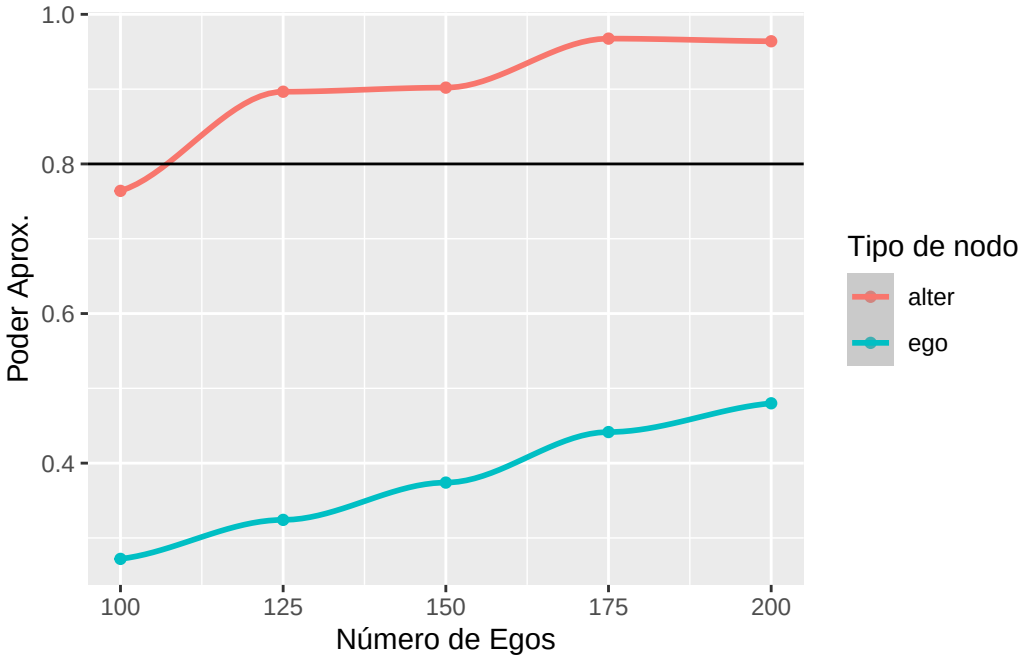
Ahora que tenemos la función de generación de datos, podemos ejecutar las simulaciones para aproximar el poder estadístico dado el tamaño de muestra. Los resultados se almacenarán

en la matriz `spower`. Dado que estamos simulando datos, es crucial establecer la semilla para que podamos reproducir los resultados.

```
# Siempre establecemos la semilla
set.seed(88)

# Haciendo espacio, ¡y ejecutando!
spower <- NULL
for (s in sizes) {

  # Ejecutar la simulación para el tamaño s
  simres <- rowMeans(replicate(n_sims, sim_data(s)), na.rm = TRUE)

  # Y almacenar los resultados
  spower <- rbind(spower, simres)

}
```

La siguiente figura muestra el poder aproximado para encontrar efectos en ambos niveles, ego y alter:

```
library(ggplot2)

spower <- rbind(
  data.frame(size = sizes, power = spower[,"ego"], type =  "ego"),
  data.frame(size = sizes, power = spower[,"alter"], type =  "alter")
)

spower |>
  ggplot(aes(x = size, y = power, colour = type)) +
  geom_point() +
  geom_smooth(method = "loess", formula = y ~ x) +
  labs(x = "Número de Egos", y = "Poder Aprox.", colour = "Tipo de nodo") +
  geom_hline(yintercept = 1 - beta_pow)
```



Como se muestra en el Capítulo Chapter 17, podemos usar un modelo de regresión lineal para predecir el tamaño de muestra como una función del poder estadístico:

```
# Ajustando el modelo
power_model <- glm(
  size ~ power + I(power^2),
  data = spower, family = gaussian(), subset = type == "alter"
)

summary(power_model)
```

```
Call:
glm(formula = size ~ power + I(power^2), family = gaussian(),
    data = spower, subset = type == "alter")

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)     1460       1342   1.088    0.390
power          -3532       3124  -1.131    0.376
I(power^2)      2293       1805   1.270    0.332
```

```
(Dispersion parameter for gaussian family taken to be 317.0856)

    Null deviance: 6250.00  on 4  degrees of freedom
Residual deviance:  634.17  on 2  degrees of freedom
AIC: 46.404

Number of Fisher Scoring iterations: 2
```

```r
# Predecir
predict(power_model, newdata = data.frame(power = .8), type = "response") |>
  ceiling()
```

```
  1
102
```

De la figura, se hace evidente que, aunque no hay suficiente poder para identificar efectos a nivel ego, porque cada ego trae cinco alters, el tamaño de muestra de alter es lo suficientemente alto como para que podamos alcanzar por encima de 0.8 de poder estadístico con un tamaño de muestra relativamente pequeño.

14.2 Ejemplo 2: Efectos de derrame efecto pre-post

Ahora las dinámicas son diferentes. En lugar de tener un grupo tratado y de control, tenemos un solo grupo sobre el cual mediremos el cambio de comportamiento. Simularemos individuos en su estado inicial, aún 0/1, y luego simularemos que la intervención los hará más propensos a tener $y = 1$. También asumiremos que los sujetos generalmente no cambian su comportamiento y que la prevalencia basal de ceros es más alta. Los pasos de simulación son los siguientes:

1. Para cada individuo en la población, extraer la probabilidad subyacente de que $y = 1$. Con esa probabilidad, asignar el valor de y. Esto aplica tanto para ego como para alter.

2. Eliminar aleatoriamente algunos egos, y sus alters correspondientes debido a la deserción.

3. Simular el comportamiento de los alters usando sus egos como el tratamiento. Tanto la probabilidad subyacente de ego como de alter se incrementan por los odds elegidos.

4. Para controlar la probabilidad subyacente de que un individuo tenga $y = 1$, usamos regresión logística condicional (también conocida como logit de caso-control pareado,) para estimar los efectos del tratamiento.

5. Aceptar/rechazar la nula y almacenar el resultado.

```
beta_pars <- c(4, 6)
odds_h_1  <- 2.0
```

```
# Función de simulación
library(survival)
sim_data_prepost <- function(n) {



  # Paso 1: Muestreando población de egos
  y_ego_star <- rbeta(n, beta_pars[1], beta_pars[2])
  y_ego_0    <- runif(n) < y_ego_star

  # Paso 2: Simulando deserción
  todrop     <- order(runif(n))[1:(n * attrition)]
  y_ego_0    <- y_ego_0[-todrop]
  n          <- n - length(todrop)
  y_ego_star <- y_ego_star[-todrop]

  # Paso 3: Simulando efecto del alter. Asumimos lo mismo que en
  # ego
  y_alter_star <- rbeta(n * n_a, beta_pars[1], beta_pars[2])
  y_alter_0    <- runif(n * n_a) < y_alter_star

  # Simulando post
  y_ego_1   <- runif(n) < plogis(qlogis(y_ego_star) + log(odds_h_1))
  tr_alter  <- as.integer(rep(y_ego_1, n_a))
  y_alter_1 <- runif(n * n_a) < plogis(qlogis(y_alter_star) + log(odds_h_1) * tr_alter)
```

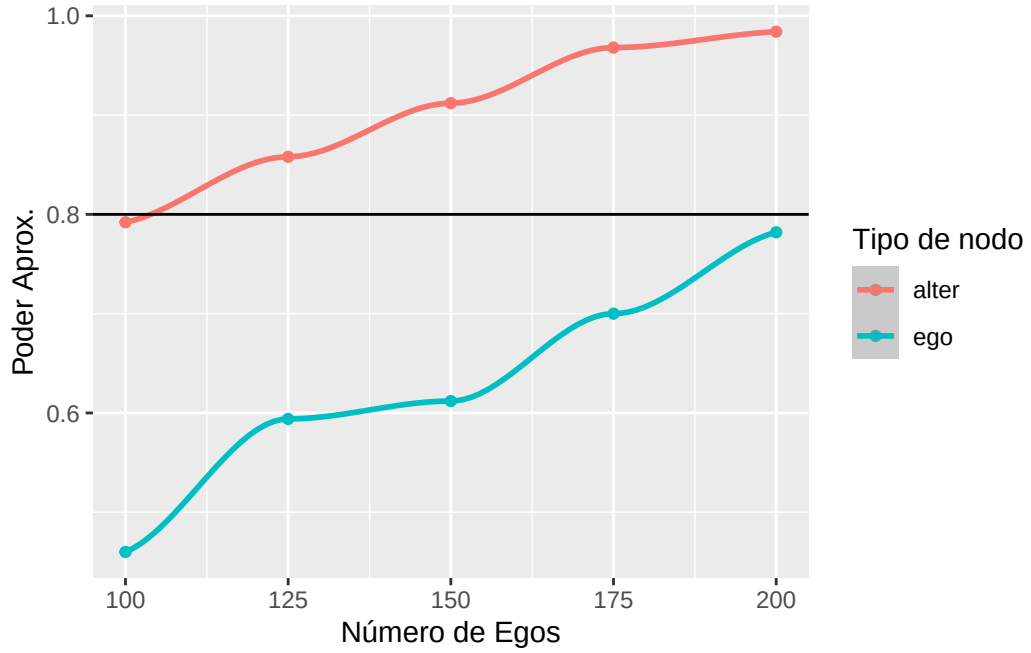
```r
# Paso 4: Calculando estadística de prueba
y_ego_0 <- as.integer(y_ego_0)
y_ego_1 <- as.integer(y_ego_1)
y_alter_0 <- as.integer(y_alter_0)
y_alter_1 <- as.integer(y_alter_1)

d <- data.frame(
  y  = c(y_ego_0, y_ego_1),
  tr = c(rep(0, n), rep(1, n)),
  g  = c(1:n, 1:n)
)

res_ego   <- tryCatch(
  clogit(y ~ tr + strata(g), data = d, method = "exact"),
  error = function(e) e
  )

d <- data.frame(
  y  = c(y_alter_0, y_alter_1),
  tr = c(rep(0, n * n_a), tr_alter),
  g  = c(1:(n * n_a), 1:(n * n_a))
)

res_alter <- tryCatch(
  clogit(y ~ tr + strata(g), data = d, method = "exact"),
  error = function(e) e
  )

if (inherits(res_ego, "error") | inherits(res_alter, "error"))
  return(c(ego =  NA, alter = NA))

# Paso 5: ¿Rechazar?
c(
  # ego        = res_ego$p.value < alpha,
  ego        = summary(res_ego)$coefficients["tr", "Pr(>|z|)"] < alpha,
  alter      = summary(res_alter)$coefficients["tr", "Pr(>|z|)"] < alpha,
```

```
    ego_test   = coef(res_ego),
    alter_glm  = coef(res_alter)
  )


}
```

```
# Siempre establecemos la semilla
set.seed(88)

# ¡Haciendo espacio y ejecutando!
spower <- NULL
for (s in sizes) {

  # Ejecutar la simulación para el tamaño s
  simres <- rowMeans(
    replicate(n_sims, sim_data_prepost(s)),
    na.rm = TRUE
    )

  # Y almacenar los resultados
  spower <- rbind(spower, simres)

}
```

```
library(ggplot2)

spowerd <- rbind(
  data.frame(size = sizes, power = spower[,"ego"], type =  "ego"),
  data.frame(size = sizes, power = spower[,"alter"], type =  "alter")
)

spowerd |>
  ggplot(aes(x = size, y = power, colour = type)) +
  geom_point() +
  geom_smooth(method = "loess", formula = y ~ x) +
```

```
  labs(x = "Número de Egos", y = "Poder Aprox.", colour = "Tipo de nodo") +
  geom_hline(yintercept = 1 - beta_pow)
```



Como se muestra en el Capítulo Chapter 17, podemos usar un modelo de regresión lineal para predecir el tamaño de muestra como una función del poder estadístico:

```
# Ajustando el modelo
power_model <- glm(
  size ~ power + I(power^2),
  data = spowerd, family = gaussian(), subset = type == "alter"
)

summary(power_model)
```

```
Call:
glm(formula = size ~ power + I(power^2), family = gaussian(),
    data = spowerd, subset = type == "alter")

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
```

```
(Intercept)    611.4      666.3   0.918    0.456
power        -1553.8     1504.7  -1.033    0.410
I(power^2)    1147.9      844.8   1.359    0.307

(Dispersion parameter for gaussian family taken to be 52.1536)

    Null deviance: 6250.00  on 4  degrees of freedom
Residual deviance:  104.31  on 2  degrees of freedom
AIC: 37.379

Number of Fisher Scoring iterations: 2
```

```r
# Predecir
predict(power_model, newdata = data.frame(power = .8), type = "response") |>
  ceiling()
```

```
  1
104
```

14.3 Ejemplo 3: Primera diferencia

Ahora, en lugar de mirar un resultado dicotómico, evaluemos qué pasa si la variable es continua. Los efectos que estamos interesados en identificar son el efecto ego y alter, γ_{ego} y γ_{alter}, respectivamente. Además, el proceso de generación de datos es

$$
\begin{aligned}
y_{itg} &= \alpha_i + \kappa_g + X_i\beta + \varepsilon_{itg} \\
y_{itg} &= \alpha_i + \kappa_g + X_i\beta + D_i^{ego}\gamma_{ego} + D_i^{alter}\gamma_{alter} + \varepsilon_{itg}
\end{aligned}
$$

Donde $D_i^{ego/alter}$ es una variable indicadora. Aquí, el comportamiento de ego y alter están correlacionados a través de un efecto fijo. En otras palabras, dentro de cada grupo, estamos asumiendo que hay una prevalencia basal compartida del resultado. La diferencia principal es que ego y alter pueden tener diferentes resultados con respecto al tamaño del efecto del tratamiento. Otra forma de abordar la correlación a nivel de grupo podría ser a través de

un proceso de autocorrelación, como en un modelo autocorrelacionado espacial; no obstante, estimar tales modelos es computacionalmente costoso, así que optamos por usar el anterior.

Para simplicidad, asumimos que no hay efecto de tiempo. Dos componentes esenciales aquí, α_i y κ_g son efectos fijos no observados a nivel individual y de grupo. El enfoque más directo aquí es usar un estimador de primera diferencia:

$$(y_{it+1g} - y_{itg}) = D_i^{ego}\gamma_{ego} + D_i^{alter}\gamma_{alter} + \varepsilon_i', \quad \varepsilon_i' = \varepsilon_{it+1g} - \varepsilon_{itg}$$

Al tomar la primera diferencia, los efectos fijos se eliminan de la ecuación, y podemos proceder con un modelo lineal regular.

```
effect_size_ego   <- 0.5
effect_size_alter <- 0.25
sizes <- seq(10, 100, by = 10)
```

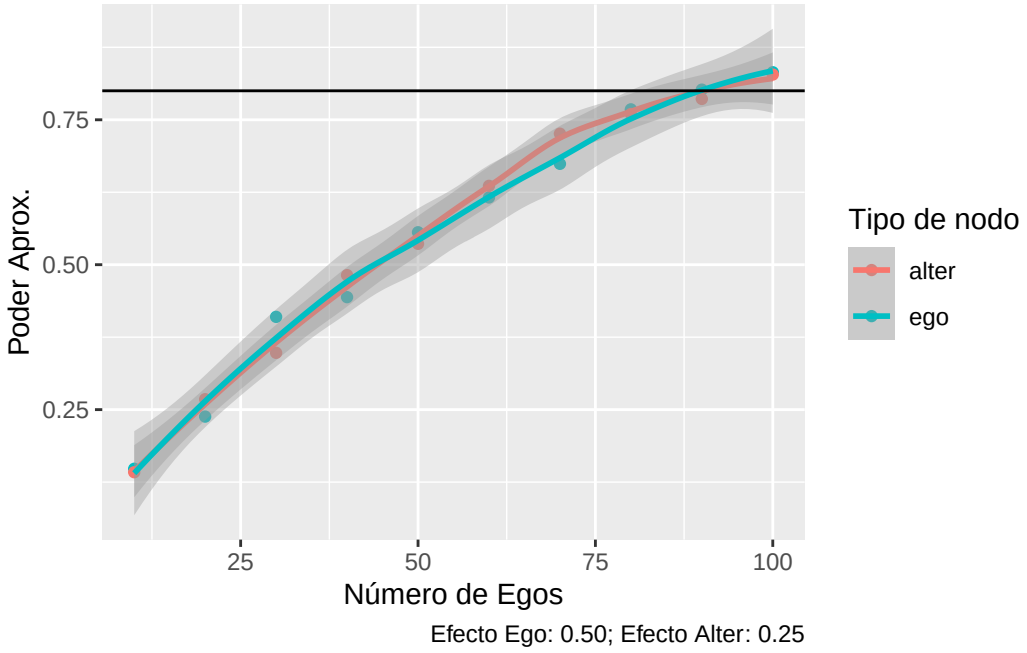
```
# Función de simulación
sim_data_prepost <- function(n) {

  # Aplicando deserción
  n <- floor(n * (1 - attrition))

  # Paso 1: Muestreando efectos fijos
  alpha_i <- rnorm(n * (n_a + 1))
  kappa_g <- rep(rnorm(n_a + 1), n)

  # Paso 2: Generando el resultado en t = 1
  is_ego   <- rep(c(1, rep(0, n_a)), n)
  is_alter <- 1 - is_ego
  y_0      <- alpha_i + kappa_g + rnorm(n * (n_a + 1))
  y_1      <- alpha_i + kappa_g +
    is_ego * effect_size_ego +
    is_alter * effect_size_alter +
    rnorm(n * (n_a + 1))

  # Paso 4: Calculando estadística de prueba
```

```
  res <- tryCatch(
    glm(I(y_1 - y_0) ~ -1 + is_ego + is_alter, family = gaussian("identity")),
    error = function(e) e
  )

  if (inherits(res, "error"))
    return(c(ego =  NA, alter = NA))

  # Paso 5: ¿Rechazar?
  c(
    # ego      = res_ego$p.value < alpha,
    ego        = summary(res)$coefficients["is_ego", "Pr(>|t|)"] < alpha,
    alter      = summary(res)$coefficients["is_alter", "Pr(>|t|)"] < alpha,
    coef(res)[1],
    coef(res)[2]
  )


}
```

```
# Siempre establecemos la semilla
set.seed(88)

# ¡Haciendo espacio y ejecutando!
spower <- NULL
for (s in sizes) {

  # Ejecutar la simulación para el tamaño s
  simres <- rowMeans(
    replicate(n_sims, sim_data_prepost(s)),
    na.rm = TRUE
    )

  # Y almacenar los resultados
  spower <- rbind(spower, simres)
```

```
}
```

```
library(ggplot2)

spowerd <- rbind(
  data.frame(size = sizes, power = spower[,"ego"], type =  "ego"),
  data.frame(size = sizes, power = spower[,"alter"], type =  "alter")
)

spowerd |>
  ggplot(aes(x = size, y = power, colour = type)) +
  geom_point() +
  geom_smooth(method = "loess", formula = y ~ x) +
  labs(x = "Número de Egos", y = "Poder Aprox.", colour = "Tipo de nodo") +
  geom_hline(yintercept = 1 - beta_pow) +
  labs(
    caption = sprintf(
      "Efecto Ego: %.2f; Efecto Alter: %.2f", effect_size_ego, effect_size_alter)
      )
```



Desde el punto de vista inferencial, aún podríamos usar un operador demean para estimar

efectos a nivel individual. En particular, necesitaríamos usar el operador demean a nivel de grupo y luego ajustar un modelo de efecto fijo para estimar parámetros a nivel individual.

Part III

Fundamentos

15 Regla de Bayes

> **Nota de Traducción**
>
> Esta versión del capítulo fue traducida de manera automática utilizando IA. El capítulo aún no ha sido revisado por un humano.

La Regla de Bayes es una ecuación fundamental en estadística bayesiana. Con ella, podemos reformular problemas inferenciales escribiendo probabilidades en términos de cantidades conocidas. La regla de Bayes puede enunciarse como sigue:

$$\mathbb{P}(X = x|Y = y) = \frac{\mathbb{P}(Y = y|X = y)\mathbb{P}(X = x)}{\mathbb{P}(Y = y)} \tag{15.1}$$

Aquí, decimos que la probabilidad condicional de X dado Y puede expresarse en términos de la probabilidad condicional de Y dado X. Por ejemplo, sea X un vector desconocido de parámetros $\theta \in \Theta$ y Y un conjunto de datos $D \sim f(\theta)$ cuyo proceso de generación de datos depende del θ no observado. Como la distribución posterior de los parámetros del modelo es en general, elusiva, en su lugar, usamos la regla de Bayes para reformular el problema:

$$\mathbb{P}(\theta|D) = \frac{\mathbb{P}(D|\theta)\mathbb{P}(\theta)}{\mathbb{P}(D)}$$

Dado que el denominador de la ecuación no depende de θ, podemos, en su lugar, escribir

$$\mathbb{P}(\theta|D) \propto \mathbb{P}(D|\theta)\mathbb{P}(\theta)$$

En el mundo bayesiano, se asume que la distribución incondicional de los parámetros del modelo proviene de una distribución particular, mientras que en el mundo frecuentista, no se hacen suposiciones distribucionales sobre los parámetros del modelo. Lo último es entonces

equivalente a decir que $\theta \sim$ Uniforme$(-\infty, +\infty)$; por lo tanto, ¡incluso los frecuentistas asumen algo sobre los parámetros del modelo![1]

La regla de Bayes puede derivarse usando probabilidades condicionales. En particular, $\mathbb{P}(x = x|Y = y)$ se define como $\mathbb{P}(x = x, Y = y)/Pr(Y = y)$. De manera similar, $\mathbb{P}(y = y|X = x)$ se define como $\mathbb{P}(y = y, X = x)/Pr(X = x)$, que puede reescribirse como $\mathbb{P}(x = x, Y = y) = \mathbb{P}(y = y|X = x)Pr(X = x)$. Reemplazando la última igualdad en la primera ecuación, obtenemos

$$\mathbb{P}(x = x|Y = y) = \frac{\mathbb{P}(x = x, Y = y)}{Pr(Y = y)}$$
$$\frac{\mathbb{P}(y = y|X = x)Pr(X = x)}{Pr(Y = y)}$$

[1]La discusión sobre diferencias y similitudes entre frecuentistas y bayesianos tiene una larga tradición. En conclusión, nadie puede decir 100% que son una cosa u otra. En rigor, los frecuentistas dicen que los parámetros del modelo no son aleatorios sino determinísticos.

16 Cadena de Markov

Una Cadena de Markov es una secuencia de variables aleatorias en la cual la distribución condicional del n-ésimo elemento solo depende de $n-1$.

16.1 Algoritmo de Metropolis

El Algoritmo de Metropolis, o MCMC de Metropolis, construye una Cadena de Markov que, bajo ciertas condiciones, converge a la distribución objetivo. La clave está en aceptar un movimiento propuesto de θ a θ' con probabilidad igual a:

$$r = \min\left(1, \frac{\mathbb{P}(\theta'|D)}{\mathbb{P}(\theta|D)}\right) \tag{16.1}$$

La secuencia resultante converge a la distribución objetivo. Podemos probar convergencia mostrando que (a) la secuencia es ergódica y (b) la distribución posterior coincide con la distribución objetivo. La ergodicidad describe tres propiedades de una cadena:

- Irreductibilidad: No hay probabilidad cero de transición entre cualquier par de estados.
- Aperiodicidad: Como el término sugiere, la cadena no tiene períodos/secuencias repetitivos.
- No transitoria: Transitoria se refiere a una cadena que tiene probabilidad no-cero de nunca regresar a un estado inicial.

Las tres propiedades son alcanzadas por cualquier caminata aleatoria basada en una distribución de probabilidad bien definida, así que nos enfocaremos en mostrar que la posterior coincide con la distribución objetivo.

16.2 Metropolis-Hastings

$$\min\left(1, \frac{\mathbb{P}(d|\theta')\mathbb{P}(\theta')\mathbb{P}(\theta'|\theta)}{\mathbb{P}(d|\theta)\mathbb{P}(\theta)\mathbb{P}(\theta|\theta')}\right)$$

Si la probabilidad de transición es simétrica, entonces la ecuación anterior se reduce al probabilidad de Metropolis.

16.3 MCMC libre de verosimilitud

1. Inicializar el algoritmo con θ_0, $\theta^* = \theta_0$–el estado aceptado actual,–y estadística de resumen observada $s_0 = S(D_{observados})$:
2. Para $t = 1$ hasta T hacer:
    a. Extraer θ_t de la distribución de propuesta $J(\theta_t|\theta^*)$
    b. Extraer datos simulados D_t del modelo $M(\theta_t)$
    c. Calcular las estadísticas de resumen $s_t = S(D_t)$
    d. Aceptar el estado propuesto con probabilidad

    Si se acepta, establecer $\theta^* = \theta_t$.

    e. Siguiente t

17 Poder y tamaño de muestra

> ⚠ Nota de Traducción
>
> Esta versión del capítulo fue traducida de manera automática utilizando IA. El capítulo aún no ha sido revisado por un humano.

Calcular el poder y el tamaño de muestra son tareas comunes en el diseño de estudios. Este capítulo te guiará a través del análisis de poder para estudios de redes. Primero, comenzaremos con algunos preliminares sobre tipos de error y poder estadístico.

17.1 Tipos de error

Una de las tablas más importantes que veremos es la tabla de contingencia de aceptar/rechazar la hipótesis nula condicional en el estado verdadero:

	Aceptar H0	Rechazar H0
H0 es verdadera	Verdadero positivo	Falso negativo
H1 es verdadera	Falso positivo	Verdadero negativo

Una mejor manera, versión más estadísticamente precisa de esta tabla sería

	Aceptar H0	Rechazar H0
H0 es verdadera	Inferencia correcta	Error Tipo I
H1 es verdadera	Error Tipo II	Inferencia Correcta

Con $\mathbb{P}(\text{Error Tipo I}) = \alpha$ y $\mathbb{P}(\text{Error Tipo II}) = \beta$. De esta manera, el poder puede definirse como la probabilidad de rechazar la nula dado que la alternativa es verdadera, $\mathbb{P}(\text{Rechazar H0}|\text{H1 es verdadera}) = 1 - \beta$.

17.2 Ejemplo 1: Tamaño de muestra para una proporción

Imaginemos que estamos preparando un estudio en el cual nos gustaría estimar la proporción de individuos con un estado dado. Formalmente, entonces decimos que la variable $Y \sim \text{Bernoulli}(p)$. Para hacerlo, necesitaremos encuestar n individuos y estimar tal número tomando el promedio muestral. Además, hipotetizamos que bajo la nula la proporción es $H_0 : p = p_0$.

La clave aquí es pensar en una regla de rechazo simple. De nuevo, el poder es la probabilidad de **rechazar la nula** dado que **la alternativa es verdadera**. Así que, para escribir la ecuación, necesitamos pensar en regiones de aceptación y rechazo. Sea $\hat{p}$ nuestro estimado para el parámetro poblacional, además, $\hat{p} = n^{-1} \sum_i y_i$. Nuestra estadística de prueba puede ser–y será, la mayoría de los casos–estandarizada para aprovechar la ley de los grandes números; bajo la nula, escribimos lo siguiente:

$$\begin{aligned} \mathbb{E}(\hat{p}) &= p_0 \\ \mathbf{Var}(\hat{p}) &= \sqrt{p_0(1-p_0)/n} \end{aligned}$$

Por lo tanto, la estadística:

$$\frac{\hat{p} - p_0}{\sqrt{p_0(1-p_0)/n}} = \frac{\sqrt{n}(\hat{p} - p_0)}{\sqrt{p_0(1-p_0)}} \sim \text{N}(0, 1)$$

Dado que la estadística está distribuida normalmente, podemos entonces decir cuándo rechazaremos la nula. Para este caso, eso depende del valor crítico, que la mayoría de las veces se define en términos de la tasa de error tipo I. Formalmente, rechazamos la nula si

$$\frac{\sqrt{n}(\hat{p} - p_0)}{\sqrt{p_0(1-p_0)}} > Z_{1-\alpha/2}$$

Esto es equivalente a decir que la **estadística de prueba cayó en la región de rechazo**. Con esto en mano, ahora podemos escribir la ecuación que usaremos para calcular el tamaño de muestra. Volviendo a la definición de poder:

$$\mathbb{P}(\text{Rechazar H0}|\text{H1 es verdadera}) = 1-\beta$$
$$\mathbb{P}\left(\frac{\sqrt{n}(\hat{p}-p_0)}{\sqrt{p_0(1-p_0)}} > Z_{1-\alpha/2}\,\middle|\, p=p_1\right) = 1-\beta$$

Observa que no podemos calcular el poder para todo $p \neq p_0$; en su lugar, vemos un valor de parámetro dado. Una buena idea es empezar desde uno previamente conocido o identificado en otros estudios. La idea clave aquí es poder manipular el argumento de la probabilidad para convertirlo en una distribución conocida, por ejemplo, la distribución normal:

Para un Tipo I dado de 0.05 y poder de 0.8, el tamaño de muestra requerido puede calcularse como sigue:

$$\begin{aligned}
1-\beta &= \mathbb{P}\left(\frac{\sqrt{n}(\hat{p}-p_0)}{\sqrt{p_0(1-p_0)}} > Z_{1-\alpha/2}\,\middle|\, p=p_1\right)\\
&= \mathbb{P}\left(\frac{\sqrt{n}(\hat{p}-p_0)}{\sqrt{p_0(1-p_0)}} < Z_{\alpha/2}\,\middle|\, p=p_1\right)\\
&= \mathbb{P}\left(\frac{\sqrt{n}(\hat{p}-p_0)}{\sqrt{p_1(1-p_1)}} < \frac{Z_{\alpha/2}\sqrt{p_0(1-p_0)}}{\sqrt{p_1(1-p_1)}}\,\middle|\, p=p_1\right)\\
&= \mathbb{P}\left(\frac{\sqrt{n}(\hat{p}-p_0+p_0-p_1)}{\sqrt{p_1(1-p_1)}} < \frac{Z_{\alpha/2}\sqrt{p_0(1-p_0)}+\sqrt{n}(p_0-p_1)}{\sqrt{p_1(1-p_1)}}\,\middle|\, p=p_1\right)\\
&= \mathbb{P}\left(\frac{\sqrt{n}(\hat{p}-p_1)}{\sqrt{p_1(1-p_1)}} < \frac{Z_{\alpha/2}\sqrt{p_0(1-p_0)}+\sqrt{n}(p_0-p_1)}{\sqrt{p_1(1-p_1)}}\,\middle|\, p=p_1\right)\\
&= \Phi\left(\frac{Z_{\alpha/2}\sqrt{p_0(1-p_0)}+\sqrt{n}(p_0-p_1)}{\sqrt{p_1(1-p_1)}}\,\middle|\, p=p_1\right)
\end{aligned}$$

La última igualdad sigue de la cantidad $\frac{\sqrt{n}(\hat{p}-p_1)}{\sqrt{p_1(1-p_1)}}$ distribuyendo normal estándar. Ahora podemos tomar la inversa de la función de distribución acumulativa (cdf) para aislar el tamaño de muestra n:

$$\Phi^{-1}(1-\beta) = \frac{Z_{\alpha/2}\sqrt{p_0(1-p_0)} + \sqrt{n}(p_0 - p_1)}{\sqrt{p_1(1-p_1)}}$$
$$Z_{1-\beta}\sqrt{p_1(1-p_1)} = Z_{\alpha/2}\sqrt{p_0(1-p_0)} + \sqrt{n}(p_0 - p_1)$$
$$\frac{\left(Z_{1-\beta}\sqrt{p_1(1-p_1)} - Z_{\alpha/2}\sqrt{p_0(1-p_0)}\right)^2}{(p_0 - p_1)^2} = n$$

Por lo tanto, para los parámetros $(1-\beta, \alpha, p_0, p_1) = (0.8, 0.05, 0.5, 0.6)$, el tamaño de muestra requerido es $193.8473 \sim 194$.

17.3 Ejemplo 2: Tamaño de muestra para una proporción (vis)

Ahora, ¿qué pasa si el modelo que estamos planeando estimar no tiene una forma cerrada? Si las soluciones analíticas no están disponibles, las simulaciones pueden ser una excelente alternativa para salvar el día. Rehagamos el cálculo de tamaño de muestra usando simulaciones.

El procedimiento para calcular el tamaño de muestra basado en simulaciones es computacionalmente intensivo. El concepto es directo, escoger un conjunto de mejores conjeturas para el tamaño de muestra, y para cada una de ellas, simular el sistema para estimar el poder. Ahora, para un valor dado de n, nosotros:

1. Simulamos una muestra de tamaño n bajo la alternativa.
2. Calculamos la estadística de prueba correspondiente a la nula.
3. Aceptamos o rechazamos de acuerdo al α seleccionado, y almacenamos el resultado.
4. Repetimos los pasos 1-3 muchas veces. El promedio obtenido es el poder correspondiente.

Cuando ejecutamos simulaciones, es conveniente escribir una función para el proceso de generación de datos. En nuestro caso, la función se llamará `sim_fun`. Las siguientes líneas de código logran nuestro objetivo: aproximar el poder simulando 10,000 experimentos para cada candidato de tamaño de muestra:

```
# Parámetros del modelo
p0        <- .5
p1        <- .6
betapower <- 1 - 0.8
alpha     <- 0.05
nsims     <- 10000

# Paso 1: Simular los datos bajo H1
z_one_minus_alpha_half <- qnorm(1 - alpha / 2)
sim_fun <- function(n) {

    # Generando los datos
    y <- as.integer(runif(n) < p1)
    phat <- mean(y)

    # ¿Aceptar o rechazar?
    sqrt(n) * (phat - p0) / sqrt(p0 * (1 - p0)) >
        z_one_minus_alpha_half

}

# Paso 2: Para un arreglo de n, simular múltiples experimentos
n_seq <- seq(from = 150, to = 250, by = 10)

simulations <- NULL
set.seed(12312)
for (n in n_seq) {

    # Ejecutar los nsims experimentos
    res <- replicate(nsims, sim_fun(n))
```

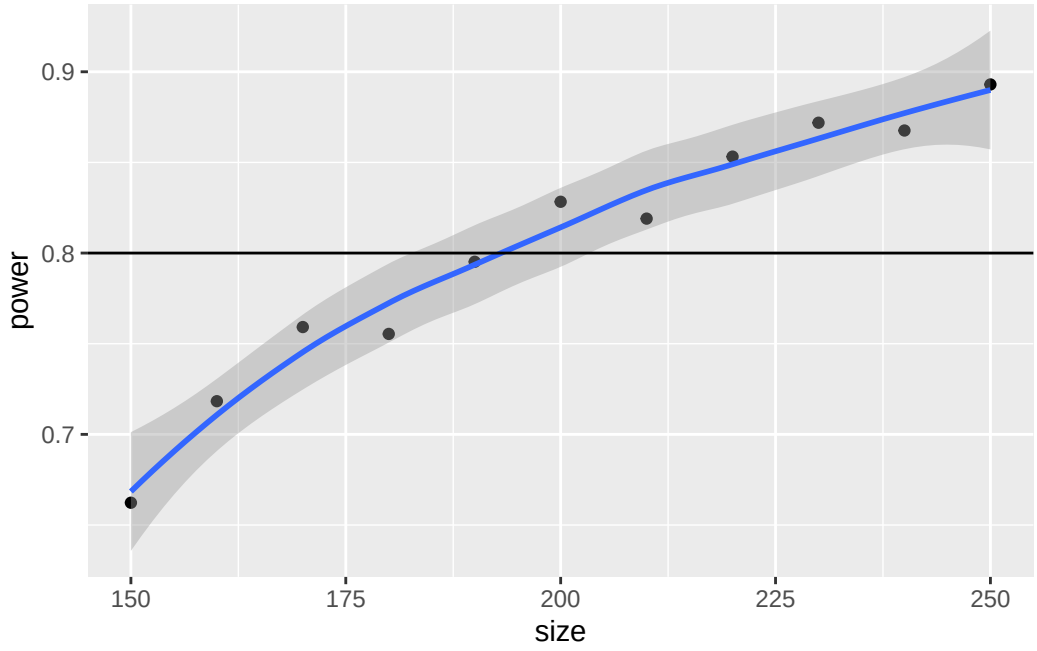
```
    # Calcular poder y almacenar el valor
    simulations <- rbind(
        simulations,
        data.frame(size = n, power = mean(res))
    )
}

# Descubriendo cuál es el valor más cercano
best <- which.min(
    abs((1 - betapower) - simulations$power)
    )

simulations[best,,drop=FALSE]
```

```
  size  power
5  190 0.7952
```

Visualicemos la curva de poder que generamos de esta simulación:

```
library(ggplot2)
ggplot(simulations, aes(x = size, y = power)) +
    geom_point() +
    geom_smooth() +
    geom_hline(yintercept = 1 - betapower)
```

```
`geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```



Alternativamente, podemos ajustar un modelo de regresión lineal donde predecimos el poder como una función del tamaño de muestra usando efectos lineales y cuadráticos:

$$n = \theta_0 + \theta_1(1-\beta) + \theta_2(1-\beta)^2$$

```
# Ajustando el modelo
power_model <- glm(
  size ~ power + I(power^2),
  data = simulations, family = gaussian()
)

# Imprimiendo los resultados
summary(power_model)
```

```
Call:
glm(formula = size ~ power + I(power^2), family = gaussian(),
    data = simulations)

Coefficients:
```

```
            Estimate Std. Error t value Pr(>|t|)
(Intercept)    632.5      232.9   2.715  0.02644 *
power        -1590.3      598.1  -2.659  0.02885 *
I(power^2)    1301.0      381.6   3.410  0.00923 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for gaussian family taken to be 34.83159)

    Null deviance: 11000.00  on 10  degrees of freedom
Residual deviance:   278.65  on  8  degrees of freedom
AIC: 74.769

Number of Fisher Scoring iterations: 2
```

```r
# Predecir
predict(power_model, newdata = data.frame(power = .8), type = "response") |>
  ceiling()
```

```
  1
193
```

Según nuestro estudio de simulación, el más cercano a nuestro 80% de poder es usar un tamaño de muestra igual a 193, que está muy cerca de la solución analítica de 194.

Como comentario final para este ejemplo, recuerda que mientras más simulaciones mejor.

A Conjuntos de datos

> ⚠ Nota de Traducción
>
> Esta versión del capítulo fue traducida de manera automática utilizando IA. El capítulo aún no ha sido revisado por un humano.

A.1 Datos SNS

A.1.1 About the data

- This data is part of the NIH Challenge grant # RC 1RC1AA019239 "Social Networks and Networking That Puts Adolescents at High Risk".
- In general terms, the SNS's goal was(is) "Understand the network effects on risk behaviors such as smoking initiation and substance use".

A.1.2 Variables

The data has a *wide* structure, which means that there is one row per individual, and that dynamic attributes are represented as one column per time.

- `photoid` Photo id at the school level (can be repeated across schools).
- `school` School id.
- `hispanic` Indicator variable that equals 1 if the indivual ever reported himself as hispanic.

- `female1`, ..., `female4` Indicator variable that equals 1 if the individual reported to be female at the particular wave.
- `grades1`,..., `grades4` Academic grades by wave. Values from 1 to 5, with 5 been the best.
- `eversmk1`, ..., `eversmk4` Indicator variable of ever smoking by wave. A one indicated that the individual had smoked at the time of the survey.
- `everdrk1`, ..., `everdrk4` Indicator variable of ever drinking by wave. A one indicated that the individual had drink at the time of the survey.
- `home1`, ..., `home4` Factor variable for home status by wave. A one indicates home ownership, a 2 rent, and a 3 a "I don't know".

During the survey, participants were asked to name up to 19 of their school friends:

- `sch_friend11`, ..., `sch_friend119` School friends nominations (19 in total) for wave 1. The codes are mapped to the variable `photoid`.
- `sch_friend21`, ..., `sch_friend219` School friends nominations (19 in total) for wave 2. The codes are mapped to the variable `photoid`.
- `sch_friend31`, ..., `sch_friend319` School friends nominations (19 in total) for wave 3. The codes are mapped to the variable `photoid`.
- `sch_friend41`, ..., `sch_friend419` School friends nominations (19 in total) for wave 4. The codes are mapped to the variable `photoid`.

Referencias

> ⚠ Nota de Traducción
>
> Esta versión del capítulo fue traducida de manera automática utilizando IA. El capítulo aún no ha sido revisado por un humano.

Admiraal, Ryan, and Mark S Handcock. 2006. "Sequential Importance Sampling for Bipartite Graphs with Applications to Likelihood-Based Inference." Department of Statistics, University of Washington.

Bojanowski, Michał. 2023. *intergraph: Coercion Routines for Network Data Objects*. https://mbojan.github.io/intergraph/.

Brooks, Steve, Andrew Gelman, Galin Jones, and Xiao-Li Meng. 2011. *Handbook of Markov Chain Monte Carlo*. CRC press.

Csárdi, Gábor, Tamás Nepusz, Vincent Traag, Szabolcs Horvát, Fabio Zanini, Daniel Noom, and Kirill Müller. 2024. *igraph: Network Analysis and Visualization in r*. https://doi.org/10.5281/zenodo.7682609.

Efron, Bradley, and Robert J Tibshirani. 1994. *An Introduction to the Bootstrap*. CRC press.

Fellows, Ian E. 2012. "Exponential Family Random Network Models." *ProQuest Dissertations and Theses*. PhD thesis. https://login.ezproxy.lib.utah.edu/login?url=https://www.proquest.com/dissertations-theses/exponential-family-random-network-models/docview/1221548720/se-2.

Geyer, Charles J., and Elizabeth A. Thompson. 1992. "Constrained Monte Carlo Maximum Likelihood for Dependent Data." *Journal of the Royal Statistical Society. Series B (Methodological)* 54 (3): 657–99. http://www.jstor.org/stable/2345852.

Handcock, Mark S., David R. Hunter, Carter T. Butts, Steven M. Goodreau, Pavel N. Krivitsky, and Martina Morris. 2023. *Ergm: Fit, Simulate and Diagnose Exponential-Family Models for Networks*. The Statnet Project (https://statnet.org). https://CRAN.R-project.org/package=ergm.

Haye, Kayla de la, Heesung Shin, George G. Vega Yon, and Thomas W. Valente. 2019. "Smoking Diffusion Through Networks of Diverse, Urban American Adolescents over the High School Period." *Journal of Health and Social Behavior*. https://doi.org/10.1177/0022146519870521.

Hunter, David R. 2007. "Curved Exponential Family Models for Social Networks." *Social Networks* 29 (2): 216–30. https://doi.org/10.1016/j.socnet.2006.08.005.

Hunter, David R, Steven M Goodreau, and Mark S Handcock. 2008. "Goodness of Fit of Social Network Models." *Journal of the American Statistical Association* 103 (481): 248–58. https://doi.org/10.1198/016214507000000446.

Hunter, David R., Mark S. Handcock, Carter T. Butts, Steven M. Goodreau, and Martina Morris. 2008. "ergm : A Package to Fit, Simulate and Diagnose Exponential-Family Models for Networks." *Journal of Statistical Software* 24 (3). https://doi.org/10.18637/jss.v024.i03.

Lazega, Emmanuel, and Tom AB Snijders. 2015. *Multilevel Network Analysis for the Social Sciences: Theory, Methods and Applications*. Vol. 12. Springer.

Leifeld, Philip. 2013. "texreg: Conversion of Statistical Model Output in R to LaTeX and HTML Tables." *Journal of Statistical Software* 55 (8): 1–24. https://doi.org/10.18637/jss.v055.i08.

LeSage, James P. 2008. "An Introduction to Spatial Econometrics." *Revue d'économie Industrielle* 123 (123): 19–44. https://doi.org/10.4000/rei.3887.

LeSage, James P., and R. Kelley Pace. 2014. "The Biggest Myth in Spatial Econometrics." *Econometrics* 2 (4): 217–49. https://doi.org/10.2139/ssrn.1725503.

Lusher, Dean, Johan Koskinen, and Garry Robins. 2013. *Exponential Random Graph Models for Social Networks: Theory, Methods, and Applications*. Cambridge University Press.

Matloff, Norman. 2011. *The Art of r Programming: A Tour of Statistical Software Design*. No Starch Press.

Morris, Martina, Mark Handcock, and David Hunter. 2008. "Specification of Exponential-Family Random Graph Models: Terms and Computational Aspects." *Journal of Statistical Software, Articles* 24 (4): 1–24. https://doi.org/10.18637/jss.v024.i04.

Plummer, Martyn, Nicky Best, Kate Cowles, and Karen Vines. 2006. "CODA: Convergence Diagnosis and Output Analysis for MCMC." *R News* 6 (1): 7–11. https://journal.r-project.org/archive/.

R Core Team. 2023. *Foreign: Read Data Stored by 'Minitab', 's', 'SAS', 'SPSS', 'Stata', 'Systat', 'Weka', 'dBase', ...* https://svn.r-project.org/R-packages/trunk/foreign/.

———. 2024. *R: A Language and Environment for Statistical Computing*. Vienna, Austria: R Foundation for Statistical Computing. https://www.R-project.org/.

Ripley, Ruth M., Tom AB Snijders, Paulina Preciado, and Others. 2011. "Manual for RSIENA." *University of Oxford: Department of Statistics, Nuffield College*, no. 2007. https://www.uni-due.de/hummell/sna/R/RSiena%7B/_%7DManual.pdf.

Robins, Garry, Philippa Pattison, and Peter Elliott. 2001. "Network Models for Social Influence Processes." *Psychometrika* 66 (2): 161–89. https://doi.org/10.1007/BF02294834.

Roger Bivand. 2022. "R Packages for Analyzing Spatial Data: A Comparative Case Study with Areal Data." *Geographical Analysis* 54 (3): 488–518. https://doi.org/10.1111/gean.12319.

Sarkar, Deepayan, and Felix Andrews. 2022. *latticeExtra: Extra Graphical Utilities Based on Lattice*. http://latticeextra.r-forge.r-project.org/.

Snijders, Tom a B. 1996. "Stochastic Actor-Oriented Models for Network Change." *The Journal of Mathematical Sociology* 21 (1-2): 149–72. https://doi.org/10.1080/0022250X.1996.9990178.

Snijders, Tom A B, and Stephen P Borgatti. 1999. "Non-Parametric Standard Errors and Tests for Network Statistics." *Connections* 22 (2): 1–10. https://insna.org/PDF/Connections/v22/1999_I-2_61-70.pdf.

Snijders, Tom A B, Gerhard G. van de Bunt, and Christian E G Steglich. 2010. "Introduction to stochastic actor-based models for network dynamics." *Social Networks* 32 (1): 44–60. https://doi.org/10.1016/j.socnet.2009.02.004.

SNIJDERS, TOM A. B. 2010. "Conditional Marginalization for Exponential Random Graph Models." *The Journal of Mathematical Sociology* 34 (4): 239–52. https://doi.org/10.1080/0022250X.2010.485707.

Snijders, Tom A. B. 2017. "Stochastic Actor-Oriented Models for Network Dynamics." *Annual Review of Statistics and Its Application* 4 (1): 343–63. https://doi.org/10.1146/annurev-statistics-060116-054035.

Snijders, Tom AB. 2002. "Markov Chain Monte Carlo Estimation of Exponential Random Graph Models." *Journal of Social Structure* 3.

Ushey, Kevin, Jim Hester, and Robert Krzyzanowski. 2021. *Rex: Friendly Regular Expressions*. https://github.com/kevinushey/rex.

Valente, Thomas W., and George G. Vega Yon. 2020. "Diffusion/Contagion Processes on Social Networks." *Health Education & Behavior* 47 (2): 235–48. https://doi.org/10.1177/1090198120901497.

Valente, Thomas W., Heather Wipfli, and George G. Vega Yon. 2019. "Network Influences on Policy Implementation: Evidence from a Global Health Treaty." *Social Science and Medicine*. https://doi.org/10.1016/j.socscimed.2019.01.008.

Wang, Peng, Ken Sharpe, Garry L. Robins, and Philippa E. Pattison. 2009. "Exponential

Random Graph (p*) Models for Affiliation Networks." *Social Networks* 31 (1): 12–25. https://doi.org/https://doi.org/10.1016/j.socnet.2008.08.002.

Wang, Zeyi, Ian E. Fellows, and Mark S. Handcock. 2023. "Understanding Networks with Exponential-Family Random Network Models." *Social Networks*, August, S0378873323000497. https://doi.org/10.1016/j.socnet.2023.07.003.

Wickham, Hadley. 2023. *Stringr: Simple, Consistent Wrappers for Common String Operations*. https://stringr.tidyverse.org.

Wickham, Hadley, and Jennifer Bryan. 2023. *Readxl: Read Excel Files*. https://readxl.tidyverse.org.

Wickham, Hadley, Romain François, Lionel Henry, Kirill Müller, and Davis Vaughan. 2023. *Dplyr: A Grammar of Data Manipulation*. https://dplyr.tidyverse.org.

Wickham, Hadley, Jim Hester, and Jennifer Bryan. 2024. *Readr: Read Rectangular Text Data*. https://readr.tidyverse.org.

Wickham, Hadley, Davis Vaughan, and Maximilian Girlich. 2024. *Tidyr: Tidy Messy Data*. https://tidyr.tidyverse.org.

B Novedades

B.1 Versión 2024.09.07

- Se corrigieron errores tipográficos en las fórmulas del capítulo de estadística bayesiana y en el capítulo de comportamiento.

B.2 Versión 2025.09.03

- Desde ahora, el libro cuenta con una sección de novedades. Intentaré mantener esta al día incluyendo los cambios realizados entre versiones. Versiones anteriores se encontrarán disponibles en GitHub como “releases”.
- Ediciones menores al capítulo de ERGMs.