

Estudio de dinámicas sociales y su impacto en el contagio de la
gripe en una pequeña comunidad mediante teléfonos móviles
Master Data Science y Big Data aplicados a la Economía y a la Administración y
Dirección de Empresas

Miguel Pérez Barrero

Diciembre 2020

Índice

1. Resumen	1
2. Descripción de la base de datos	1
2.1. Ficheros de la base de datos con sus columnas	1
3. Diseño técnico	2
4. Procesado de los datos	3
4.1. Carga de datos	3
4.2. Limpieza de los datos	4
5. Modelado de la ubicación usando la wifi.	5
5.1. Caracterización de la nube de puntos	7
5.2. Etiquetado de ubicaciones	11
5.3. Creación del grafo de interacciones wifi	14
6. Grafo de interacción por proximidad (Bluetooth)	17
7. Grafo de llamadas telefónicas y SMS	21
7.1. Organización de los datos en la rejilla de semana	22
7.2. Visualización de matrices de adyacencia	23
7.3. Construcción de la rejilla combinada	25
8. Grafo de amistad	27
8.1. Construcción del grafo por semanas	27
9. Grafo de interacción por actividades comunes	31
9.1. Ajuste de los datos a la rejilla de tiempo	32
9.2. Generación del grafo de actividades compartidas	34
10. Construcción de la matriz de síntomas	36
11. Construcción de la matriz de datos	39
11.1. Análisis de la exposición al síntoma mediante interacciones	39
11.1.1. Carga de datos	39
11.1.2. Verificación de la estructura de los grafos	40
11.1.3. Descripción de los conceptos necesarios para construir la matriz combinada	41
11.1.4. Composición de la matriz combinada	42
11.1.5. Presentación descriptiva de las características de la matriz combinada	43
11.2. Análisis de la influencia de las interacciones en el síntoma “Congestión”	45
11.2.1. Análisis con modelo lineal generalizado	52
11.2.2. Modelado del problema como clasificación	54
11.2.3. Generación de modelos de minería para clasificar	55
11.2.4. Comparación de modelos	67
12. Conclusiones	67
12.1. Ampliaciones y mejoras	68
13. Tecnología y librerías utilizadas	69
14. Bibliografía	69

1. Resumen

Partiendo de una base de datos recogida durante un experimento realizado en una residencia universitaria, se busca relacionar los comportamientos sociales con la propagación de la gripe en dicha comunidad, así como otros factores recogidos. En este contexto, los comportamientos se monitorizan mediante una aplicación de móvil y encuestas periódicas.

Los datos disponibles de los sujetos del estudio son tanto: - de naturaleza estática: planta del edificio donde se alojan, actividades universitarias en las que participan, - como de naturaleza dinámica o temporal: personas cercanas detectadas mediante bluetooth, wifis a las que se conectan, llamadas y sms entre ellos y otros datos recogidos mediante encuestas. Con los datos disponibles se busca tanto llegar a ajustar un modelo que relacione los comportamientos observados desde los móviles y las encuestas con un síntoma de la gripe.

2. Descripción de la base de datos

- La base de datos se compone de 13 archivos CSV o tablas. Cada archivo recoge las entradas sobre un tema específico o las lecturas de sensores en el caso de los datos obtenidos desde el móvil.
- La base de datos original puede obtenerse en:
 - <http://realitycommons.media.mit.edu/socialevolution2.html>.

2.1. Ficheros de la base de datos con sus columnas

1. Llamadas (Calls.csv)

- user_id: identificador seudonimizado del sujeto experimental.
- time_stamp: marca de tiempo del inicio de la llamada.
- duration: duración de la llamada
- dest_user_id_if_known: si la llamada es a otro sujeto experimental, el id seudonimizado del receptor.
- dest_phone_hash: teléfono del destinatario seudonimizado

2. Mensajes de texto antiguos SMS (SMS.csv)

- user.id: identificador seudonimizado del sujeto experimental.
- time: marca de tiempo del momento del envío del mensaje.
- incoming: si se trata de un mensaje enviado (0) o recibido por el terminal (1)
- dest.user.id.if.known: si el destinatario es otro sujeto experimental, su id seudonimizado
- dest.phone.hash: teléfono del destinatario seudonimizado

3. Proximidad de otros dispositivos mediante Bluetooth (Proximity.csv)

- user.id: identificador seudonimizado del sujeto experimental.
- remote.user.id.if.known: identificador seudonimizado de otro sujeto experimental detectado en la zona cercana.
- time: marca de tiempo del escaneo de proximidad.
- prob2: probabilidad de que los dos sujetos estén en el mismo piso del edificio.

4. Conexión Wi-Fi (WLAN2.csv)

- user_id: identificador seudonimizado del sujeto experimental.
- time: marca de tiempo en la que se consulta la conexión a una wifi.
- wireless_mac: identificador único de la red wifi (dirección mac o ssid) seudonimizado.
- strength: fuerza de la señal (permite computar la cercanía al punto wifi)
- unix_time: el mismo dato que "time" pero en formato de segundos desde el 1-1-1970.

5. Encuesta de relaciones personales (RelationshipsFromSurveys.csv)

- id.A: identificador seudonimizado del sujeto que rellena la encuesta

- id.B: identificador seudonimizado del sujeto sobre el que pregunta la encuesta
- relationship: tipo de relación entre estas “Amigo Cercano”, “Hablan de política”, “Se etiquetan en Facebook”, “Hablan por Twitter”, “Socializan dos veces por semana”.
- survey.date: día de realización de la encuesta

6. Actividades escolares (Activities.csv)

- user.id: identificador seudonimizado del sujeto experimental.
- campus.organization: identificador de la actividad
- survey.month: mes de realización de la encuesta

7. Síntomas de gripe (FluSymptoms.csv)

- user_id: identificador seudonimizado del sujeto experimental.
- time: marca de tiempo de la encuesta
- sore.throath.cough: tos de garganta con dolor
- runnynose.congestion.sneezing: moqueo, congestión o estornudos
- fever: fiebre
- nausea.vomiting.diarrhea: náusea, vómitos o diarrea
- sad.depressed: tristeza o depresión
- often.stressed: estrés

8. Hábitos de salud (Health.csv)

- No se va a utilizar

9. Política (Politics.csv)

- No se va a utilizar

10. Preferencias musicales: conocimiento del gusto de otro sujeto (MusicGenreAwareness.csv)

- No se va a utilizar

11. Preferencias musicales: implicación con un género musical (MusicGenreImmersion.csv)

- No se va a utilizar

12. Preferencias musicales: preferencia por un tipo de música (MusicGenrePreference.csv)

- No se va a utilizar

13. Tabla de sujetos (Subjects.csv)

- user_id: identificador de sujeto
- year_school: año de universidad (novato, junior, senior,...)
- floor: planta de la residencia en la que vive.

3. Diseño técnico

El objetivo del trabajo es determinar si existe una manera de modelar la gravedad de alguno de los síntomas contenidos en el dataset “flu” partiendo de los datos de relaciones entre sujetos. Los datos de relaciones entre sujetos necesitamos extraerlos de las tablas que representan algo común entre ellos.

Algunas de ellas representan relaciones de manera directa, como por ejemplo la encuesta de amistades o la proximidad entre móviles. En otros casos se representa algo “común” entre los sujetos que implica que han tenido que estar juntos en un determinado lugar o durante un determinado tiempo, como puede ser estar conectados a las mismas wifis con intensidades similares o participar en la misma actividad del campus.

En todos los casos, se generará una matriz de adyacencia de Grafo que represente las interacciones entre sujetos en cada arco. Además cada arco tendrá un peso asociado que nos indicará la importancia de dicha interacción.

Cada tabla no genera un único grafo, si no uno por cada bloque de tiempo en el que se dividirá el experimento. Al conjunto de estos bloques lo llamaremos *Rejilla de tiempo*

Por otro lado tendremos una tabla de síntomas. En este caso agregaremos los datos de síntomas en un bloque de tiempo que sea coincidente con la Rejilla de tiempo por cada sujeto. La consideraremos una matriz columna.

Una vez construidas las matrices de multi-grafos y la de síntomas., se aplicará una formula para calcular la *exposición* del sujeto al síntoma. Esta formula considera que la exposición de un sujeto a un síntoma es el producto escalar entre la columna de síntomas *de una semana*. y la fila de la matriz de adyacencia correspondiente al usuario y a la semana.

Como tenemos varias matrices, habrá varios tipos de exposiciones: exposición por wifi, exposición por bluetooth, exposición por actividad de campus, exposición por amistad...

El modelo final que se quiere ajustar partirá de una tabla en la que, para cada semana y para cada sujeto se calcularán las exposiciones de las tres semanas anteriores a través de los cuatro medios de relación mencionados (wifi, bluetooth, actividad, amistad).

La columna de “clase” o variable dependiente será el estado del síntoma para cada sujeto en la semana actual.

El número de registros totales de la tabla para el modelo final depende del número de usuarios y de las semanas en las que tenemos datos de síntomas y de las matrices de adyacencia, ya que para cada usuario se generan varias filas, una por semana.

El objetivo final consiste en obtener un modelo que demuestre que existe relación entre alguno de los modos de exposición y la presencia de uno de los síntomas. Eventualmente, dicho modelo se podría usar para predecir los síntomas de semana siguiente usando datos de la semana actual y las dos anteriores.

Esta estrategia se planea en parte haciendo uso de lo mencionado en los artículos que acompañan al dataset, pero evitando modelos dinámicos o de agentes como se hace en los artículos.

Empezaremos generando el grafo de interacción asociado a los datos de la wifi.

4. Procesado de los datos

4.1. Carga de datos

```
base_dir <- '../datasets/SocialEvolution/'

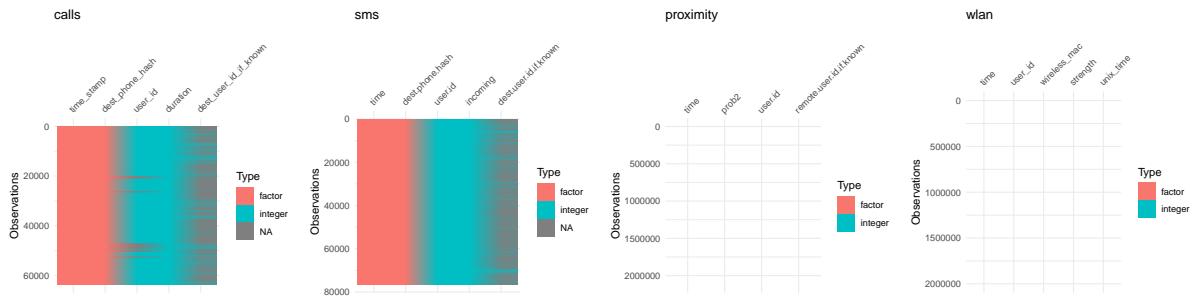
calls <- read.csv2(paste0(base_dir, 'Calls.csv'), sep = ',')
sms <- read.csv2(paste0(base_dir, 'SMS.csv'), sep = ',')
proximity <- read.csv2(paste0(base_dir, 'Proximity.csv'), sep = ',')
wlan <- read.csv2(paste0(base_dir, 'WLAN2.csv'), sep = ',')
relationships <-
  read.csv2(paste0(base_dir, 'RelationshipsFromSurveys.csv'), sep = ',')
activities <-
  read.csv2(paste0(base_dir, 'Activities.csv'), sep = ',')
flu <- read.csv2(paste0(base_dir, 'FluSymptoms.csv'), sep = ',')
health <- read.csv2(paste0(base_dir, 'Health.csv'), sep = ',')
politics <- read.csv2(paste0(base_dir, 'Politics.csv'), sep = ',')
music_aware <-
  read.csv2(paste0(base_dir, 'MusicGenreAwareness.csv'), sep = ',')
music_immersion <-
  read.csv2(paste0(base_dir, 'MusicGenreImmersion.csv'), sep = ',')
music_pref <-
  read.csv2(paste0(base_dir, 'MusicGenrePreference.csv'), sep = ',')
subjects <- read.csv2(paste0(base_dir, 'Subjects.csv'), sep = ',')
```

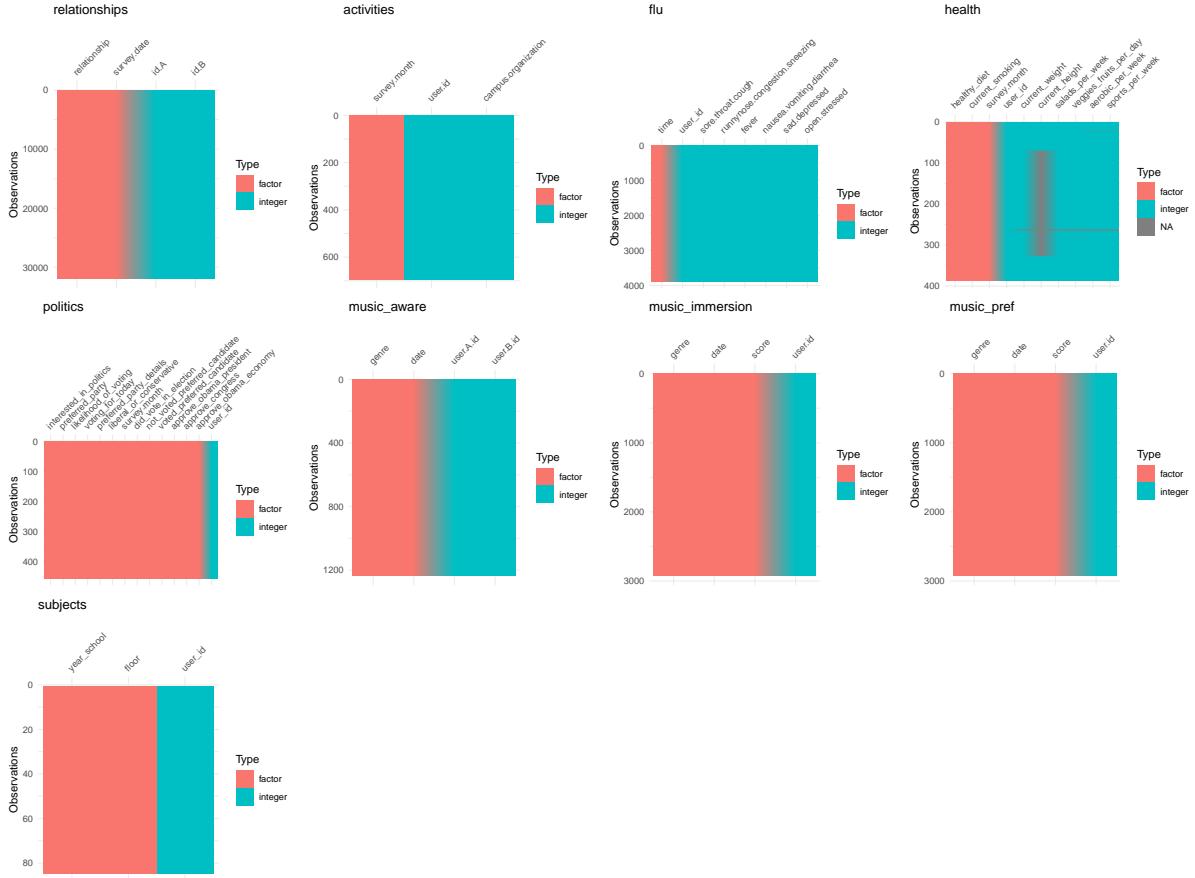
4.2. Limpieza de los datos

```

catalog <-
  c(
    'calls',
    'sms',
    'proximity',
    'wlan',
    'relationships',
    'activities',
    'flu',
    'health',
    'politics',
    'music_aware',
    'music_immersion',
    'music_pref',
    'subjects'
  )
datasets = list(
  calls,
  sms,
  proximity,
  wlan,
  relationships,
  activities,
  flu,
  health,
  politics,
  music_aware,
  music_immersion,
  music_pref,
  subjects
)
for (index in seq_along(datasets)) {
  print(vis_dat(datasets[[index]], warn_large_data = FALSE) + labs(title = catalog[index]))
}

```





Todos los datasets están bastante completos excepto unos pocos que tienen falta de datos:

- **health:** no tiene muchos datos de peso y alguna línea casi sin datos.
- **calls:** faltan muchos destinatarios y algún que otro origen
- **sms:** faltan muchos destinatarios

5. Modelado de la ubicación usando la wifi.

El objetivo es construir ubicaciones a partir de las MACs de las LANs. Por cuestiones de protección de datos, el dataset del experimento no contiene la ubicación física de los puntos de acceso wifi, así que no podremos obtener ubicaciones reales por triangulación.

Sabemos que en un momento dado un usuario está conectado a N LANs con diferente intensidad. Si otro usuario está conectado a las mismas o similares N LANs con intensidades similares, lo razonable es pensar que está cerca.

Se pueden separar las macs mas importantes por ser las que aparecen en el registro con más de un usuario en un rango de tiempo determinado.

En un primer escaneo miramos cuantos usuarios hay en el mismo instante de tiempo

```
hour <- 3600

count_events <- wlan %>%
  mutate(cell = unix_time %/%(hour/6)) %>%
  group_by(wireless_mac, user_id, cell) %>%
  summarise(total = n(), .groups = "drop") %>%
  group_by(total) %>% summarise(count = n(), .groups = "drop")
count_events
```

	total	count
1	789434	
2	602417	
3	1142	
4	88	

Miramos que wifis son más importantes, porque tienen la mayoría de los eventos de los usuarios que traceamos El criterio será: las wifis que son visitadas por al menos 3 de nuestros usuarios objetivo en intervalos de 10 minutos.

```
relevant_wifis <- wlan %>%
  mutate(cell = unix_time %/% (hour/6)) %>%
  group_by(wireless_mac, user_id, cell) %>%
  summarise(total = n(), .groups = "keep") %>%
  filter(total > 3)
macs <- unique(relevant_wifis$wireless_mac)
```

Vamos a verificar que no hemos perdido las trazas enteras de algún sujeto al reducir las WLANs.

```
orig_users <- sort(unique(wlan$user_id))
filtered_wlan_events <- wlan %>% filter(wireless_mac %in% macs)
all_users_present <- orig_users == sort(unique(filtered_wlan_events$user_id))
unique(all_users_present)
```

```
## [1] TRUE
```

Una vez eliminadas las wifis poco relevantes, intentamos clusterizar el resto. Los clusters que obtengamos los consideramos zonas de incidencia. La idea es lograr una especie de teselación. Se puede intentar agrupar las wifis que significan una misma área o agrupar los eventos en áreas cuando tienen la misma distancia a las mismas (o similares) wifis.

```
length(macs)
```

```
## [1] 37
```

Ahora tenemos 37 áreas de interés. La idea es clusterizarlas para manejar un número razonable de ellas.

Después obtener una “traza” del usuario tomando unos intervalos de tiempo definidos. Esta sería la traza de ubicaciones. De otras tablas podemos obtener la traza de contactos.

El propio artículo dice que no se puede obtener la ubicación precisa con los datos que están a disposición en la web. La estrategia por tanto es intentar clusterizar los datos directamente, de tal manera que se vea que los usuarios cuando están “en el mismo sitio” tienen una intensidad de conexión parecida a “las mismas wifis”.

Es mejor usar las 37 wifis como dimensiones, de tal modo que, en un mismo bloque de tiempo, 10 minutos por ejemplo, cada individuo se define por un vector compuesto por las intensidades a cada una de las 37 wifis. De esta forma se puede aplicar bien reducción de dimensión y clustering después para determinar las “zonas”.

Y parece completamente razonable decir que los puntos que están a distancias parecidas de las mismas wifis están “cerca”.

La forma de trabajar con el tiempo en este caso es usar cada posición / registro como una muestra más del “mapa” de wifis. Es decir, como un punto más, no como puntos que se mueven. Posteriormente, cada posición se asociará al usuario para saber donde estaba en ese momento.

Es decir, la estrategia es:

- Clusterizar el conjunto de posiciones de la tabla de intensidades de señal de cada wifi relevante.
- Una vez clusterizado, ver a que región corresponde la posición de un usuario en un tiempo determinado.
- Generar una tabla cruzada entre cada par de sujetos utilizando la ubicación en zona en cada bloque de diez minutos. De esta manera los usuarios que estaban en la misma zona tendrán una relación y el peso de dicha relación corresponderá al número de eventos wifi registrados en esos 10 minutos.

Guardamos los datos que hemos limpiado para procesarlos en la siguiente fase.

```
positions_wifi_space <- data.frame(filtered_wlan_events) %>%
  mutate(position_id = paste0(user_id, ":", unix_time)) %>%
  select(position_id, wireless_mac, strength, time, unix_time, user_id) %>%
  filter(strength != 0) %>% # eliminamos las wifis que pudieran tener cero señal
  pivot_wider(names_from = wireless_mac,
              values_from = strength,
              values_fill = 0)
write.csv(positions_wifi_space, "positions_wifi_space.csv", row.names = FALSE)
```

Cargamos los datos generados en el apartado anterior, y los preparamos eliminando las columnas que no son relevantes para reducir dimensión y clusterizar.

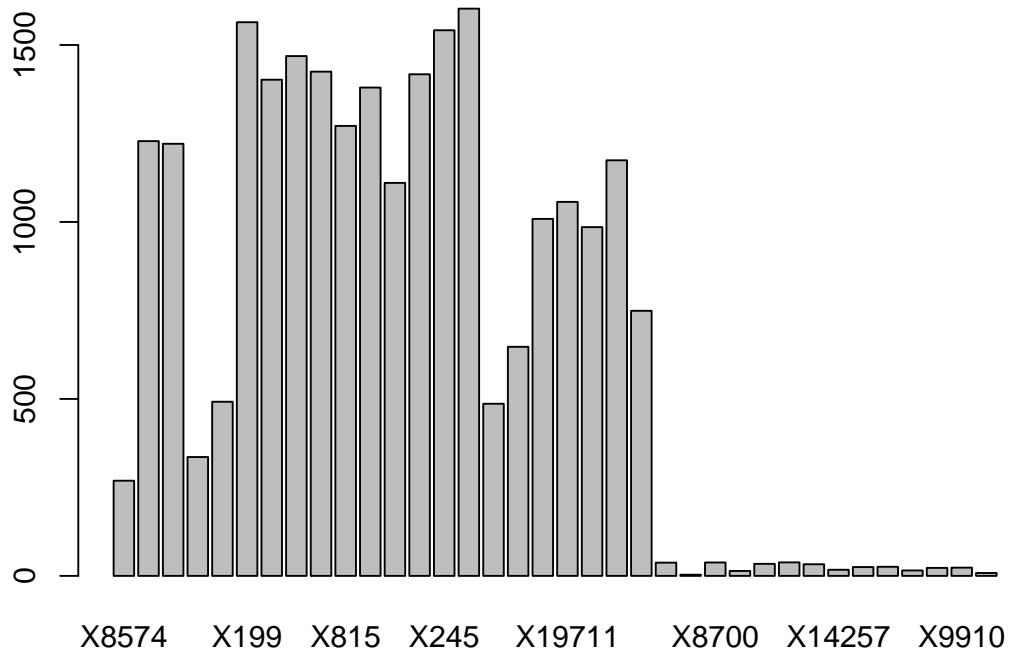
```
base_dir <- '../datasets/SocialEvolution/'
subjects <- read.csv2(paste0(base_dir, 'Subjects.csv'), sep = ',')

positions_wifi_space <- read.csv("positions_wifi_space.csv")
to_cluster <- positions_wifi_space
to_cluster$position_id <- NULL
to_cluster$unix_time <- NULL
to_cluster$time <- NULL
to_cluster$user_id <- NULL
```

5.1. Caracterización de la nube de puntos

Visualizamos la varianza de las dimensiones. Esto nos permitirá tener una idea de que algoritmo puede ser mejor para clusterizar.

```
barplot(colVars(to_cluster))
```



Se puede ver que hay mucha diferencia en las varianzas de las dimensiones. En estos casos K-medias no suele funcionar muy bien porque busca “esferas” alrededor de los puntos y en algunas dimensiones no hay datos como para una esfera.

Además del problema de reducir dimensiones y clusterizar, personalmente me siento más seguro si puedo visualizar de alguna manera este espacio de 36 dimensiones. Las técnicas basadas en scatterplots o uso de símbolos y pseudocolor en gráficas 2D y 3D son demasiado limitadas para trabajar con 36 dimensiones, además de que requieren un gran ordenador.

Por otro lado, los datos de intensidades de señal respecto a puntos fijos son intrínsecamente geométricos, o por lo menos muy similares ya que representan distancias en el mundo real. Se ven afectados por obstáculos y por perturbaciones electromagnéticas, con lo que una aproximación púramente geométrica quedaría destrozada por dichos “ruidos”. Por ello es necesario una solución que pueda representar estas perturbaciones de manera robusta.

Una técnica que proporciona todas estas capacidades a la vez es el SOM, el Mapa Autoorganizativo de Kohonen. Es un tipo de red neuronal que realiza una proyección “cartográfica” de los datos de entrada sobre una rejilla en dos dimensiones. La rejilla como tal es la última capa de neuronas y los valores que se van acumulando en cada neurona tras el entrenamiento son los vectores en 36 dimensiones que mejor recubren la “nube de puntos” que no podemos ver.

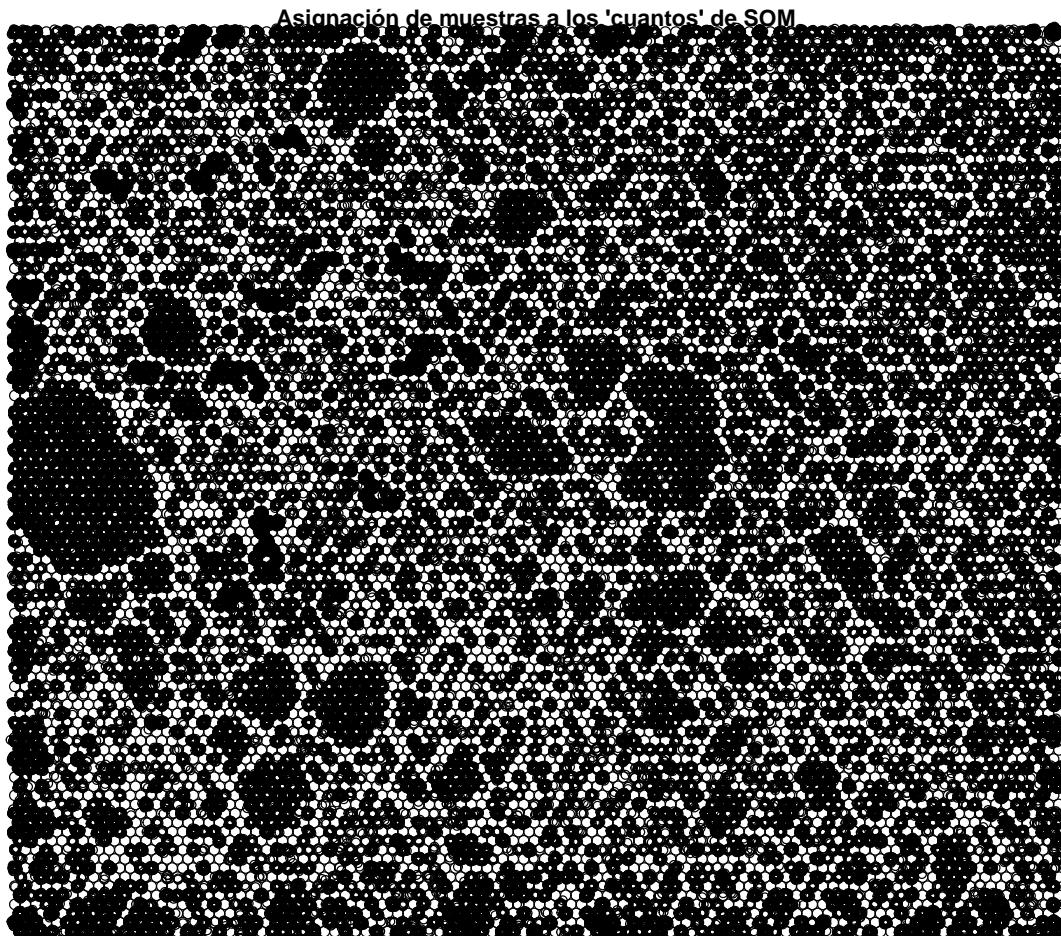
De esta manera se adapta a las zonas muy pobladas de puntos y a las poco densas de una forma que K-means no puede hacer.

Además esta técnica realiza una *cuantización del espacio, que es una primera aproximación al clusterizado. A pesar de eso, es interesante volver a clusterizar las neuronas entre sí para manejar un número más pequeño de ubicaciones de cara a las operaciones posteriores de grafos entre sujetos.

Visualizemos pues el espacio de las wifis con un Mapa de Kohonen,

```
map <- som(as.matrix(to_cluster),  
           somgrid(xdim = 100, ydim = 100, topo = c("hexagonal")))
```

```
plot(map, type = 'mapping',  
      main="Asignación de muestras a los 'cuantos' de SOM")
```



En esta gráfica se muestra la asignación de los “puntos” originales, que habíamos construido limpiando las wifis no relevantes y agrupando por bloques de tiempo.

Afortunadamente, se muestran agrupaciones/clusters en forma de islas más densas y de zonas poco densas de “playa” alrededor. Esto me da más confianza acerca de la existencia de ubicaciones marcadas en las que los sujetos están conectados a las mismas wifis. Esto confirma la intuición de que los datos de señales wifi tienen un reflejo en un espacio físico.

A continuación hacemos clusters sobre el propio mapa SOM, de tal manera que se trabaje con un número razonable de “ubicaciones”. Solo por recordar, no disponemos de las ubicaciones reales del experimento en el dataset, por cuestiones de privacidad, así que tenemos que intuir las a través de los datos.

```

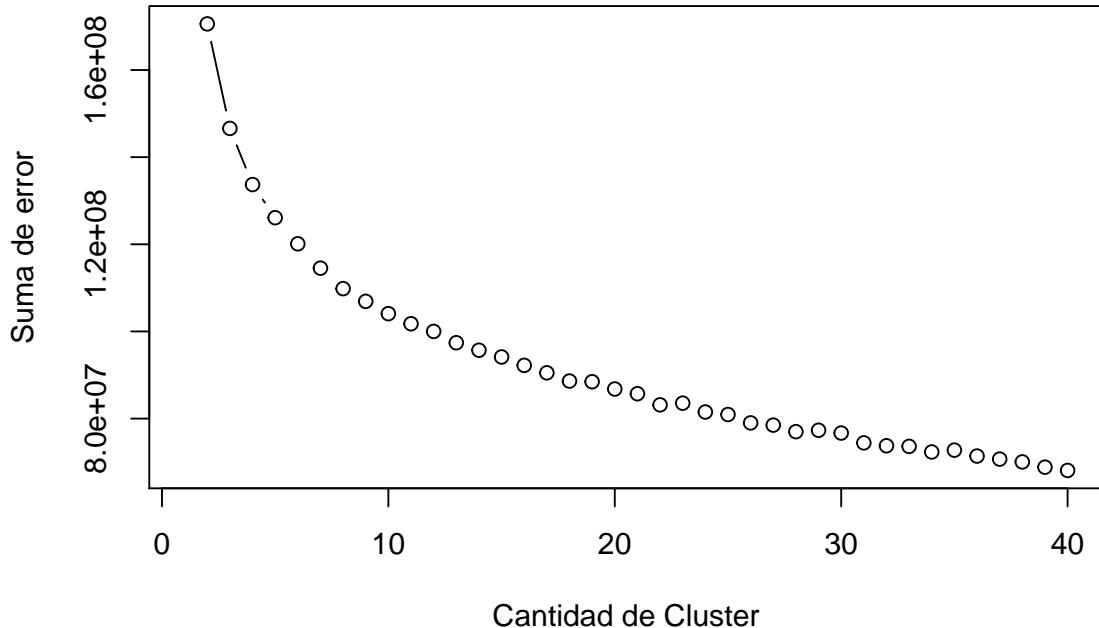
set.seed(7)

error <- NULL
k_max <- 40

for (i in 2:k_max)
{
  error[i] <- sum(
    kmeans(map$codes[[1]], centers = i, iter.max = 100, nstart = 10)$withinss
  )
}

plot(1:k_max, error, type="b", xlab="Cantidad de Cluster", ylab="Suma de error")

```



Parece que la rodilla está en 10 clusters, a partir de ahí añadir más clusters solo nos da una pequeña ganancia en el error.

Vamos a usar ese valor de 10 para construir el modelo final y además mostraremos el mapa con los clusters y los puntos originales.

```

set.seed(8)
k <- 10
som.cluster.k <- kmeans(
  map$codes[[1]],
  centers = k,
  iter.max = 100,
  nstart = 10)$cluster

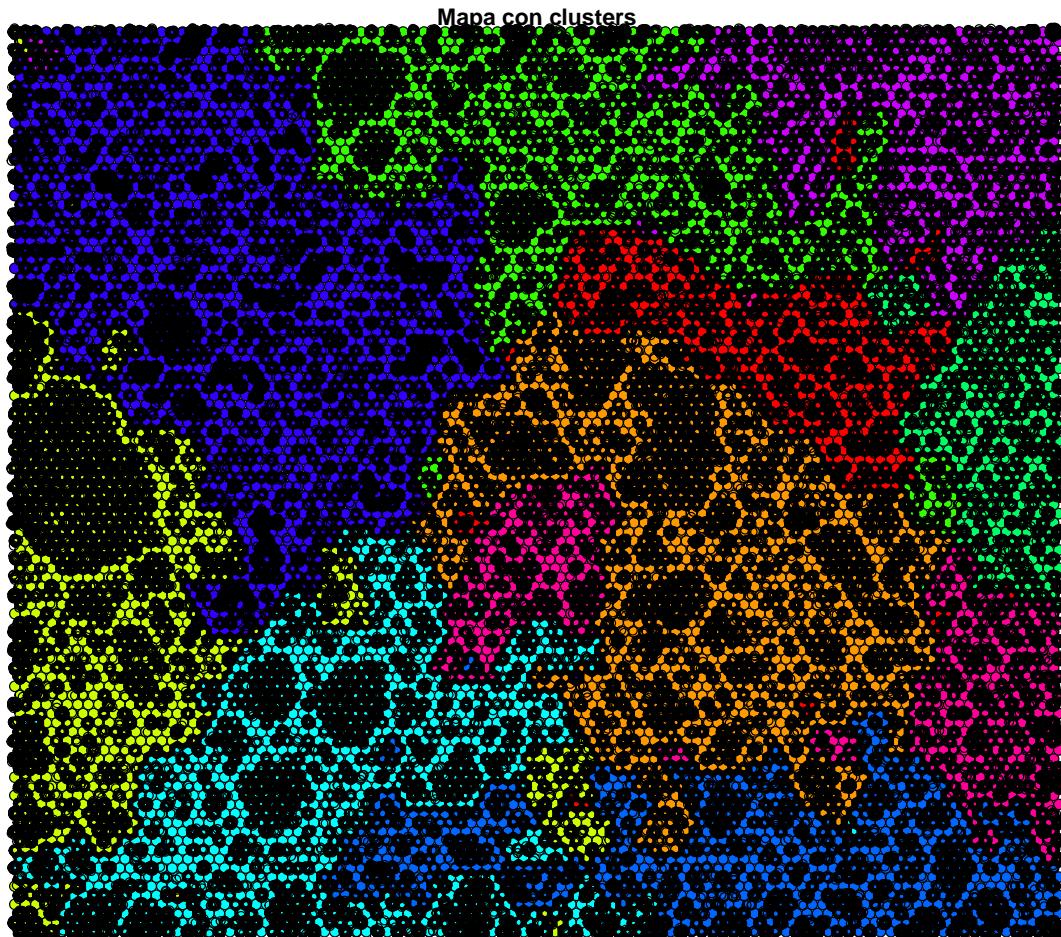
plot(map,

```

```

type = 'mapping',
keepMargins = F,
bgcol = rainbow(10)[som.cluster.k],
bg = 0,
main = 'Mapa con clusters')
add.cluster.boundaries(map, som.cluster.k)

```



Se puede ver que los clusters “densos” marcados en negro se agrupan y atraen parte de la zona de alrededor. Pienso que la distribución que se muestra con 10 zonas es razonable.

Con estos 10 clusters vamos a codificar todas las trazas originales de ubicación wifi. Para esto, se etiqueta la tabla de posiciones_wifi con el resultado del cluster que corresponde.

5.2. Etiquetado de ubicaciones

```

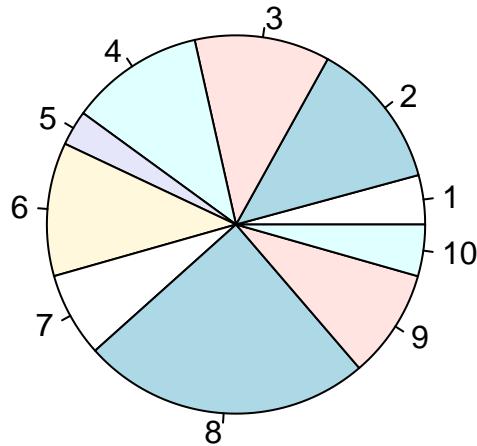
wifi_pos_labeled <- positions_wifi_space
wifi_pos_labeled$location <- som.cluster.k[map$unit.classif]

```

```
saveRDS(wifi_pos_labeled, "wifi_pos_labeled.RData")
write.csv(wifi_pos_labeled, "wifi_with_location.csv", row.names = FALSE)
```

Vemos el porcentaje de puntos en cada ubicación por hacernos una idea y por verificar que son sitios diferentes. Al ser sitios diferentes se espera que tengan diferente número de puntos.

```
por_ubicacion <- wifi_pos_labeled %>%
  group_by(location) %>%
  summarise(sum = n())
pie(por_ubicacion$sum)
```



Vamos a explorar la distribución temporal de los eventos de wifi, para ello anotamos la tabla con columnas temporales: año, mes, día, semana relativa, semana del año y otras que nos ayuden a decidir cual será la rejilla temporal que usaremos para agregar todos los datos de cara al modelo compuesto final.

Como detalle especial, la columna de semanas no se genera como la semana ordinal del año. Se genera como la diferencia en semanas entre la fecha de la fila y una fecha inicial. Esto se hace porque el experimento abarca más de un año y la semana ordinal podría repetirse.

Además usar una misma fecha base nos va a permitir poner todos los datasets en la misma rejilla temporal, independientemente de cuando empiezan a tomarse los datos.

```
# ultimo domingo de 2007 como fecha base para que las semanas sean
# regulares entre años
base <- parse_date_time('2007/12/30 00:00', 'y-m-d H:M')

time_distibution_wlan <- wifi_pos_labeled %>%
  mutate(time = parse_date_time(time, "y-m-d H:M:S")) %>%
```

```

  mutate(
    weekday = wday(time),
    week = as.integer(round(difftime(time, base, units = "weeks"))),
    year_week = week(time),
    hour = hour(time),
    month = month(time),
    year = year(time),
    year_day = yday(time)
  ) %>% filter(year >= 2008) %>% arrange(week, user_id)

```

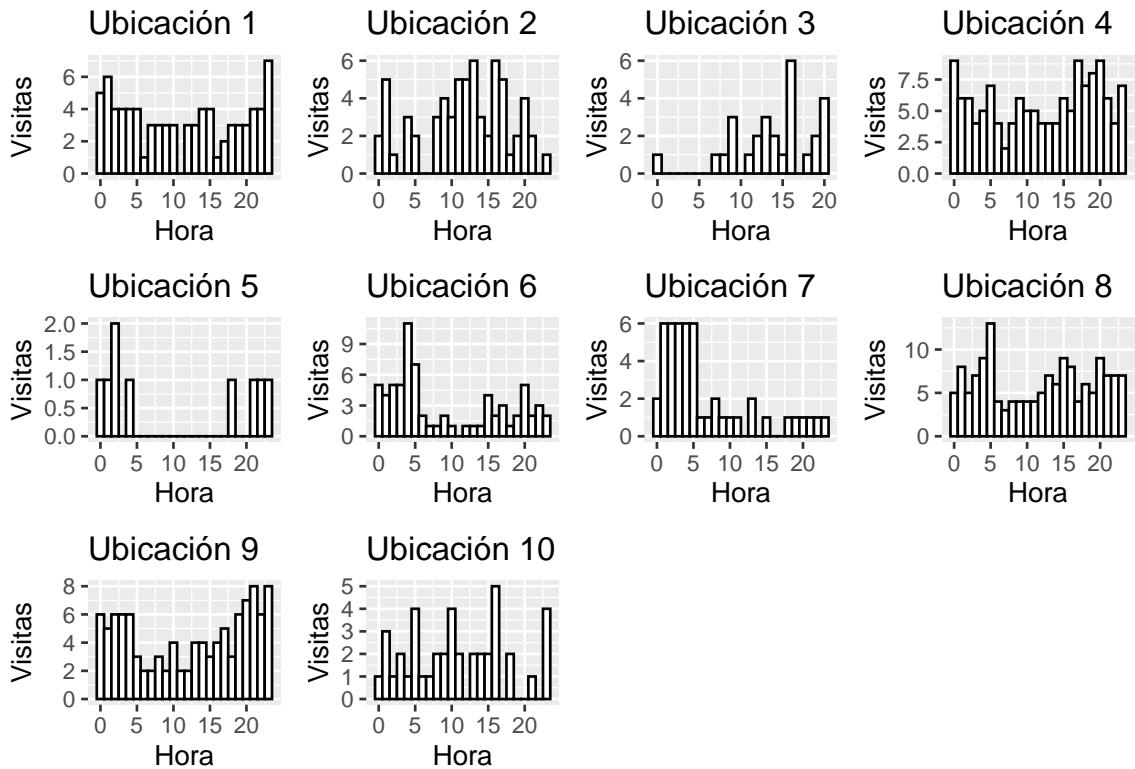
Es interesante hacer un resumen informativo de donde suelen estar los usuarios en los diferentes momentos del día. Para evitar alteraciones debidas a los diferentes momentos del curso, vamos mirando los perfiles limitados por semana. Si no hacemos esta limitación por semana, los perfiles se aplanan y no dan información característica de cada ubicación.

Esto es, supongo, porque el tiempo de uso de las diferentes estancias del Campus se iguala a medida que va pasando más tiempo.

```

plots <- list()
for (loc in 1:10) {
  per_hour <- time_distibution_wlan %>%
    dplyr::select(user_id, location, week, year_week, weekday, hour) %>%
    distinct(user_id, location, week, year_week, weekday, hour) %>%
    filter(location == loc) %>%
    filter(year_week == 2)
  plots[[loc]] <- ggplot(per_hour, aes(x = hour)) +
    geom_histogram(color = "black",
                  fill = "white",
                  binwidth = 1) +
    xlab("Hora") +
    ylab("Visitas") +
    ggtitle(paste0("Ubicación ", loc))
}
wrap_plots(plots)

```



Se pueden observar diferencias entre las ubicaciones, algunas notables.

- La ubicación 1 es más frecuentada por las noches aunque tiene gente por el día, es probable que sea un lugar cercano a los dormitorios o un área común.
- La dos es más frecuentada hacia el centro del día, pueden ser clases u oficinas.
- La tres es frecuentada hacia la tarde - noche pero no de madrugada, podría ser una cafetería o zona de ocio.
- La cuatro tiene visitas regulares durante todo el día, con momentos puntuales de madrugada y de 5 a 6. Podría ser también zona de dormitorios.
- La 5 parece que solo se frecuenta para dormir o para ocio de tarde-nocturno.
- La 6 es más frecuentada temprano en el día y la madrugada. También podría ser un tipo de oficina o clase.
- Un perfil similar tienen la 7 y la 8.
- La 9 es frecuentada durante la tarde y la madrugada, podría ser otro área de dormitorio.
- La 10 tiene un aumento de visitas esporádicas a ciertas horas, podría ser una zona de tránsito.

5.3. Creación del grafo de interacciones wifi

Teniendo en cuenta los datos de los que se dispone en otros datasets (actividades, bluetooth, ect) parece razonable usar una rejilla temporal ajustada a la semana. En una semana se puede ver el cambio en el ciclo de la gripe, ya que suele durar algo menos de una semana. Y ver el comportamiento de los síntomas asociados a la gripe es el objetivo de este trabajo.

Primero se calcula una tabla agregada por los criterios de usuario, semana y ubicación/cluster. El valor de *intensidad de la presencia de un sujeto en una ubicación* o importancia será una simple cuenta de los eventos wifi registrados. Esto es así para tener una aproximación simple.

```
wlan_by_week <- time_distibution_wlan %>%
  group_by(week, user_id, location) %>%
  summarize(count = n(), .groups = 'drop')
```

Ahora se crean los pares de usuarios que están en la misma ubicación en la misma semana, es decir, la matriz de adyacencia a través de wifi.

La matriz de adyacencia se construye como el producto de los eventos que nos ubican a cada usuario en una zona por si misma. Para que tenga sentido, este producto se hace usando la columna “location”, la de zona, como columna para combinar. Como la resolución de nuestros datos es de semana, también se usa esta columna para combinar.

Al combinar por la zona y la semana, el resultado contiene los pares de sujetos que estuvieron en la misma zona en la misma semana, dando ocasión al contagio.

Es importante ordenar la matriz e imputar los vacíos para que luego se pueda hacer producto de matrices con los síntomas y así generar un indicador numérico de la exposición al virus.

```
wlan_edges_by_week <- merge(wlan_by_week,
  wlan_by_week,
  by = c('week', 'location'),
  all.x = TRUE)

# eliminar el cruce de cada usuario con sigo mismo
wlan_edges_by_week <- wlan_edges_by_week %>%
  filter(user_id.x != user_id.y) %>%
  arrange(week, user_id.x, user_id.y)
```

Para la cuenta de la intensidad de la relación se asume el mínimo de las intensidades de los dos extremos. La intensidad de relación se considera que es proporcional a las horas que ha estado un usuario en una ubicación.

Si dos personas A y B han estado en una ubicación y B ha estado menos tiempo, lo razonable es pensar que han coincidido, como mucho, el tiempo que ha estado B en la ubicación. Por ello, agregamos las dos cuentas tomando el mínimo de ellas.

```
wlan_edges_by_week <- wlan_edges_by_week %>% rowwise() %>% transmute(
  week = week,
  user_id.x = user_id.x,
  user_id.y = user_id.y,
  weight = min(count.x, count.y),
) %>%
  group_by(week, user_id.x, user_id.y) %>%
  summarise(weight = sum(weight), .groups = 'drop')
```

Imputamos ahora los datos faltantes para poder construir una matriz cuadrada. Hay dos tipos de datos faltantes: los de sujetos que no han tenido registros esa semana (por errores de software u otras causas), y la de semanas que no tienen datos de algún sujeto o de todos.

Vamos a ver qué sujetos tenemos en la matriz de datos actual y cuantos deberíamos tener.

```
data.frame(
  "sujetos en el mapa de wifis" = length(unique(wlan_edges_by_week$user_id.x)),
  "sujetos totales" = length(unique(subjects$user_id))
)
```

sujetos.en.el.mapa.de.wifis	sujetos.totales
64	84

Se puede ver que no están todos.

Para crear una base para imputar, se genera la rejilla completa, con semanas desde la 40 hasta la

74 y todos los pares de usuarios y con pesos 0. Las semanas elegidas son aquellas en las que existen datos.

La *lista de arcos del grafo* final será la combinación de los datos que tenemos ya generados y la rejilla con los pesos a cero.

Esta combinación se vuelve a agrupar para eliminar duplicados y se agrega sumando el valor de los pesos, de tal manera que si existe un registro de (semana, usuario.x, usuario.y) con peso mayor que cero y otro con cero, prevalezca el que tiene peso mayor que cero.

```
grid.weeks <- 40:74
grid.users <- subjects$user_id
wifi_full_grid <-
  expand.grid(week = grid.weeks,
              user_id.x = grid.users,
              user_id.y = grid.users)
wifi_full_grid$weight = 0

wifi_full_grid <- wifi_full_grid %>%
  full_join(wlan_edges_by_week) %>%
  group_by(week, user_id.x, user_id.y) %>%
  summarise(weight = max(weight), .groups = 'drop')

## Joining, by = c("week", "user_id.x", "user_id.y", "weight")
```

Ahora, desde la *lista de arcos de grafo completa*, construimos la matriz de adyacencia mediante un giro/pivotado de los datos. Al final, la matriz de adyacencia no es más que la lista de arcos puesta en formato ancho.

```
wlan_adjacency_by_week <- wifi_full_grid %>%
  pivot_wider(names_from = user_id.y, values_from = weight, values_fill = 0)
```

Pintamos la matriz de adyacencia como grafo para una semana. La información de las ubicaciones la hemos perdido al agregarla porque no va a ser relevante para calcular la *exposición al síntoma*:

```
suppressPackageStartupMessages(library(network))

sel_week <- 66

edges <- wlan_edges_by_week %>%
  ungroup %>%
  filter(week == sel_week) %>%
  dplyr::select(user_id.x, user_id.y, weight) %>%
  transmute(from = user_id.x,
            to = user_id.y,
            weight = weight) %>% filter(weight > mean(weight))

nodes <- unique(subjects %>% dplyr::select(user_id))

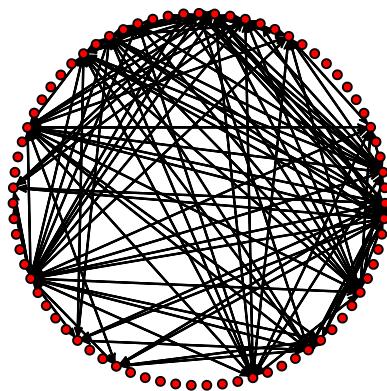
routes_network <-
  network(
    edges,
    vertex.attr = nodes,
    matrix.type = "edgelist",
    ignore.eval = FALSE
  )
```

```

plot(
  routes_network,
  vertex.cex = 1,
  mode = "circle",
  main = paste0("Semana ", sel_week)
)

```

Semana 66



Guardamos la matriz de adyacencia de Wifi para poder usar las relaciones entre personas a través de wifi en la tabla final del modelo como un indicio más de la propagación de la enfermedad.

```

saveRDS(wlan_adjacency_by_week, "wlan_adjacency_by_week.RData")

wlan_adjacency_by_week = readRDS("wlan_adjacency_by_week.RData")
write.csv2(wlan_adjacency_by_week,
           "wlan_adjacency_by_week.csv",
           row.names = FALSE)

```

6. Grafo de interacción por proximidad (Bluetooth)

La siguiente matriz de adyacencia a construir es la de interacción por Bluetooth. Estos datos son muy frecuentes y ya representan directamente pares de usuarios, así que la normalización de la rejilla y la construcción de la lista de arcos es sencilla.

```

base_dir <- '../datasets/SocialEvolution/'

proximity <- read.csv2(paste0(base_dir, 'Proximity.csv'), sep = ',')
subjects <- read.csv2(paste0(base_dir, 'Subjects.csv'), sep = ',')

```

```
str(proximity)
```

```
## 'data.frame': 2124564 obs. of 4 variables:  
## $ user.id : int 58 58 58 58 58 58 58 58 58 58 ...  
## $ remote.user.id.if.known: int 42 49 54 57 74 76 2 14 42 46 ...  
## $ time : Factor w/ 576409 levels "2007-09-05 14:02:11",...: 1 1 1 1 1 1 2 2 2 2 ...  
## $ prob2 : Factor w/ 577 levels "", "0", "0.001", ...: 36 2 1 230 1 1 200 102 36 2 ...
```

Inicialmente eliminamos los pares de proximidad que no tienen alguno de los dos extremos o los que los dos extremos son iguales, pues no podemos imputar el dato faltante.

```
proximity <- proximity %>%  
  filter(!is.na(user.id)) %>%  
  filter(!is.na(remote.user.id.if.known)) %>%  
  filter(user.id != remote.user.id.if.known)
```

Agrupamos por tiempo, con el mismo criterio que en caso de las wifis, generando las semanas desde un día inicial. Primero generamos las columnas de tiempo y luego agrupamos por semana.

```
# ultimo domingo de 2007 como fecha base para que las semanas sean regulares entre años  
base = parse_date_time('2007/12/30 00:00', 'y-m-d H:M')  
  
proximity_time <- proximity %>% mutate(  
  time = parse_date_time(time, "y-m-d H:M:S"))  
  ) %>%  
  mutate(  
    weekday = wday(time),  
    week = as.integer(round(difftime(time, base, units = "weeks"))),  
    year_week = week(time),  
    hour = hour(time),  
    month = month(time),  
    year = year(time),  
    year_day = yday(time)  
  ) %>% filter(year >= 2008)  
  
proximity_by_week <- proximity_time %>%  
  group_by(user.id, remote.user.id.if.known, week) %>%  
  summarize(  
    count = n(),  
    .groups = 'drop'  
  )  
write.csv2(proximity_by_week, "proximity_by_week.csv", row.names = FALSE)
```

Esta es una lista de arcos entre los sujetos del experimento, los arcos son el número de reportes de coincidencia del bluetooth por semana. En realidad son varias matrices de adyacencia, una por semana. Es una matriz incompleta, pues no contiene todos los pares de usuarios ni todas las semanas. Al igual que con la wifi, será necesario generar una rejilla vacía pero completa y combinar los datos.

```
proximity_by_week <- proximity_time %>%  
  filter(year >= 2008) %>%  
  group_by(user.id, remote.user.id.if.known, week) %>%  
  summarize(  
    count = n(),  
    .groups = 'drop'
```

```

) %>% ungroup()

proximity_by_week <- proximity_by_week %>%
  dplyr::transmute(
    week = week,
    user_id.x = user.id,
    user_id.y = remote.user.id.if.known,
    weight = count
  ) %>% filter(user_id.x != user_id.y)

grid.weeks <- 2:81
grid.users <- subjects$user_id
bluetooth_full_grid <- expand.grid(week = grid.weeks,
                                     user_id.x = grid.users,
                                     user_id.y = grid.users)
bluetooth_full_grid$weight = 0

bluetooth_full_grid <- bluetooth_full_grid %>%
  full_join(proximity_by_week) %>%
  group_by(week, user_id.x, user_id.y) %>%
  summarise(weight = max(weight), .groups = 'drop') %>%
  arrange(week, user_id.x, user_id.y)

## Joining, by = c("week", "user_id.x", "user_id.y", "weight")

```

Como ya tenemos la lista de interacciones, podemos presentar el grafo de contactos:

```

suppressPackageStartupMessages(library(network))

sel_week <- 40

edges <- proximity_by_week %>%
  ungroup %>%
  filter(week == sel_week) %>%
  dplyr::select(user_id.x, user_id.y, weight) %>%
  rowwise() %>%
  transmute(from = user_id.x,
            to = user_id.y,
            weight = weight) %>%
  group_by(from, to) %>%
  summarise(weight = mean(weight), .groups = 'drop') %>%
  ungroup %>%
  filter(weight >= mean(weight)) ## pintamos solo las relaciones mas relevantes

nodes <- unique(proximity_by_week %>% dplyr::select(user_id.x))

routes_network <-
  network(
    edges,
    vertex.attr = nodes,
    matrix.type = "edgelist",
    ignore.eval = FALSE
  )
plot(

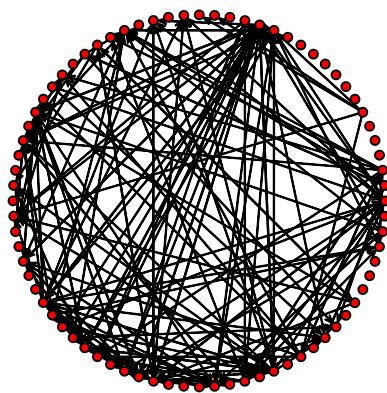
```

```

    routes_network,
    vertex.cex = 1,
    mode = "circle",
    main = paste0("Semana ", sel_week)
)

```

Semana 40



Estas matrices de contacto son muy densas. Solo hemos representado los valores de confluencia más altos para que el gráfico sea informativo. Como valores más altos se ha considerado los mayores que la media.

Generamos la matriz de adyacencia de todas las semanas y guardamos:

```

proximity_adj_matrix <- bluetooth_full_grid %>%
  pivot_wider(names_from = user_id.y, values_from = weight, values_fill = 0)
write.csv2(proximity_adj_matrix, "proximity_adj_matrix_by_week.csv", row.names = FALSE)

```

Visualizamos la matriz de interacciones bluetooth mediante un gráfico raster de la matriz de adyacencia:

```

plots <- list()

for (sel_week in 1:2) {
  prox_week <-
    proximity_by_week %>% filter(week == 39 + sel_week) %>% dplyr::select(-week)
  plots[[sel_week]] <-
    ggplot(prox_week, aes(x = user_id.x, y = user_id.y)) +
    geom_tile(aes(fill = weight)) +
    labs(x = "Subject A",
         y = "Subject B",

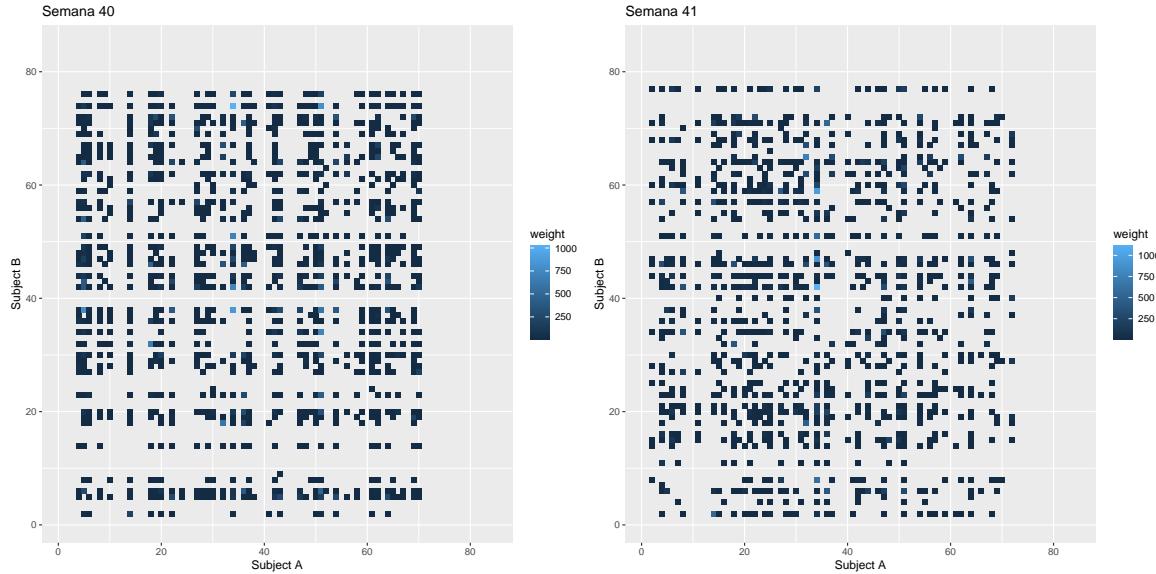
```

```

        title = paste("Semana", 39 + sel_week)) +
      lims(x = c(1, 84), y = c(1, 84))
}

wrap_plots(plots, ncol = 2)

```



Se puede ver que hay usuarios en los bordes (sobre el id 80) que no tienen datos.

7. Grafo de llamadas telefónicas y SMS

Ahora se va a calcular el grafo de interacciones basado en las llamadas por teléfono y SMS enviados entre los sujetos. El objetivo es combinar los dos indicios en una sola matriz de multi-grafo, ya que se considera que el impacto de ambos modos de comunicación es similar a efectos de síntomas de gripe.

Para otras cuestiones como salud u opiniones políticas podría ser interesante separar las dos matrices.

```

base_dir <- '../datasets/SocialEvolution/'

calls <- read.csv2(paste0(base_dir, 'Calls.csv'), sep = ',')
sms <- read.csv2(paste0(base_dir, 'SMS.csv'), sep = ',')
subjects <- read.csv2(paste0(base_dir, 'Subjects.csv'), sep = ',')

str(calls)

## 'data.frame': 63824 obs. of  5 variables:
## $ user_id          : int  19 19 19 19 19 19 19 19 19 19 ...
## $ time_stamp       : Factor w/ 51467 levels "2008-09-05 19:19:04",...: 390 389 418 416 447 451
## $ duration         : int  15 17 421 0 47 0 267 0 26 25 ...
## $ dest_user_id_if_known: int  NA NA NA NA 18 22 NA 22 NA ...
## $ dest_phone_hash   : Factor w/ 5422 levels "", "-", "-00391",...: 1324 1324 2350 1324 3393 3915

str(sms)

## 'data.frame': 76537 obs. of  5 variables:

```

7.1. Organizacion de los datos en la rejilla de semana

Al igual que en los casos anteriores, se agregan los datos por semana, considerando las semanas basadas en el 30 de diciembre de 2007.

```
# ultimo domingo de 2007 como fecha base para que las semanas sean regulares entre años
base = parse_date_time('2007/12/30 00:00', 'y-m-d H:M')

calls_time <- calls %>% mutate(
  time = parse_date_time(time_stamp, "y-m-d H:M:S")
) %>%
  transmute(
    weekday = wday(time),
    week = as.integer(round(difftime(time, base, units = "weeks"))),
    year_week = week(time),
    hour = hour(time),
    month = month(time),
    year = year(time),
    year_day = yday(time),
    user_id.x = user_id,
    user_id.y = dest_user_id_if_known,
    weight = duration
) %>%
  filter(year >= 2008) %>%
  filter(!is.na(user_id.y)) %>% # destinatario desconocido
  filter(user_id.x != user_id.y) # a si mismo
```

Ahora los SMS

```
sms_time <- sms %>% mutate(
  time = parse_date_time(time, "y-m-d H:M:S")
) %>%
transmute(
  weekday = wday(time),
  week = as.integer(round(difftime(time, base, units = "weeks"))),
  year_week = week(time),
  hour = hour(time),
  month = month(time),
  year = year(time),
  year_day = yday(time),
  user_id.x = user.id,
  user_id.y = dest.user.id.if.known,
  weight = 1
) %>%
filter(year >= 2008) %>%
filter(!is.na(user_id.y)) %>% # destinatario desconocido
filter(user_id.x != user_id.y) # a si mismo
```

Una vez generadas las columnas temporales, usamos la semana para agregar.

```

calls_by_week <- calls_time %>%
  group_by(week, user_id.x, user_id.y) %>%
  summarise(
    weight = sum(weight),
    .groups='drop'
  ) %>% ungroup()

```

```

sms_by_week <- sms_time %>%
  group_by(week, user_id.x, user_id.y) %>%
  summarise(
    weight = sum(weight),
    .groups='drop'
  ) %>% ungroup()

```

7.2. Visualización de matrices de adyacencia

Usamos el modo de visualización de matriz para ver quien habla con quién, y representamos la intensidad con puntos de diferente grosor.

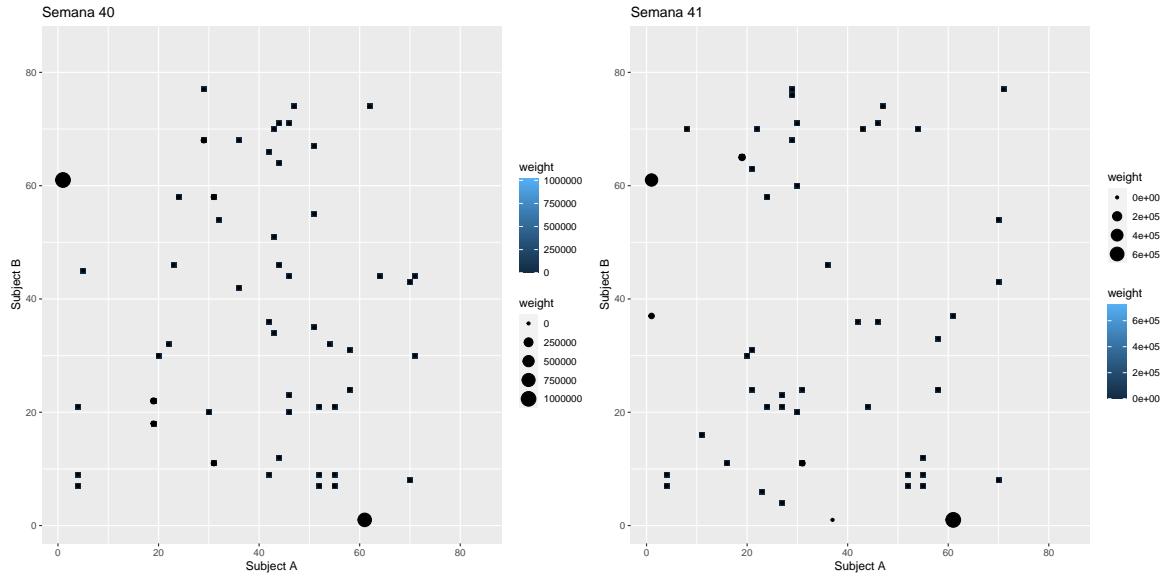
```

plots <- list()

for (sel_week in 1:2) {
  prox_week <-
    calls_by_week %>% filter(week == 39 + sel_week) %>% dplyr::select(-week)
  plots[[sel_week]] <-
    ggplot(prox_week, aes(x = user_id.x, y = user_id.y)) +
    geom_tile(aes(fill = weight)) +
    geom_point(aes(size = weight)) +
    lims(x = c(1, 84), y = c(1, 84)) +
    labs(x = "Subject A",
         y = "Subject B",
         title = paste("Semana", 39 + sel_week))
}

wrap_plots(plots, ncol = 2)

```



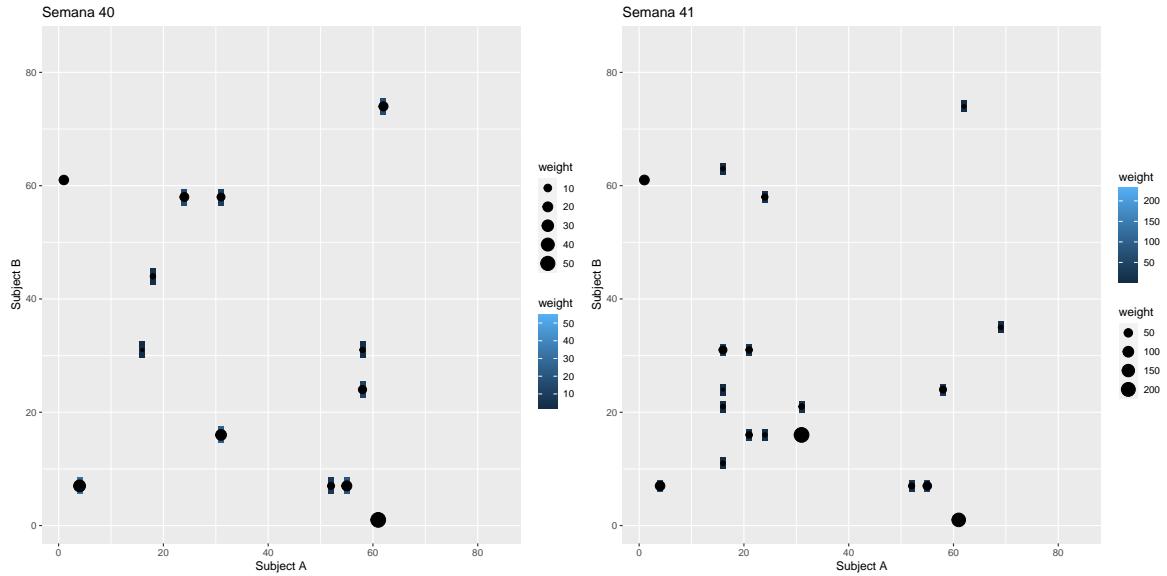
```

plots <- list()

for (sel_week in 1:2) {
  prox_week <-
    sms_by_week %>% filter(week == 39 + sel_week) %>% dplyr::select(-week)
  plots[[sel_week]] <-
    ggplot(prox_week, aes(x = user_id.x, y = user_id.y)) +
    geom_tile(aes(fill = weight)) +
    geom_point(aes(size = weight)) +
    lims(x = c(1, 84), y = c(1, 84)) +
    labs(x = "Subject A",
         y = "Subject B",
         title = paste("Semana", 39 + sel_week))
}

wrap_plots(plots, ncol = 2)

```



7.3. Construcción de la rejilla combinada

Como tenemos los datos de llamadas y de SMS en el mismo formato, podemos proceder a combinarlos. Mostramos los rangos de semanas que tiene cada tipo para tenerlo en cuenta al generar la rejilla vacía.

```
glue("Llamadas: rango {paste(range(calls_by_week$week), collapse=',')}")  
  
## Llamadas: rango 36,78  
  
glue("SMS: rango {paste(range(sms_by_week$week), collapse=',')}")  
  
## SMS: rango 0,78  
  
communication_by_week <- calls_by_week %>%  
  full_join(sms_by_week) %>%  
  group_by(week, user_id.x, user_id.y) %>%  
  summarise(  
    weight = sum(weight),  
    .groups = 'drop'  
  )  
  
## Joining, by = c("week", "user_id.x", "user_id.y", "weight")
```

Mostramos los datos combinados para saber el impacto que tiene la combinación. El efecto es pequeño y las llamadas siguen siendo más relevantes.

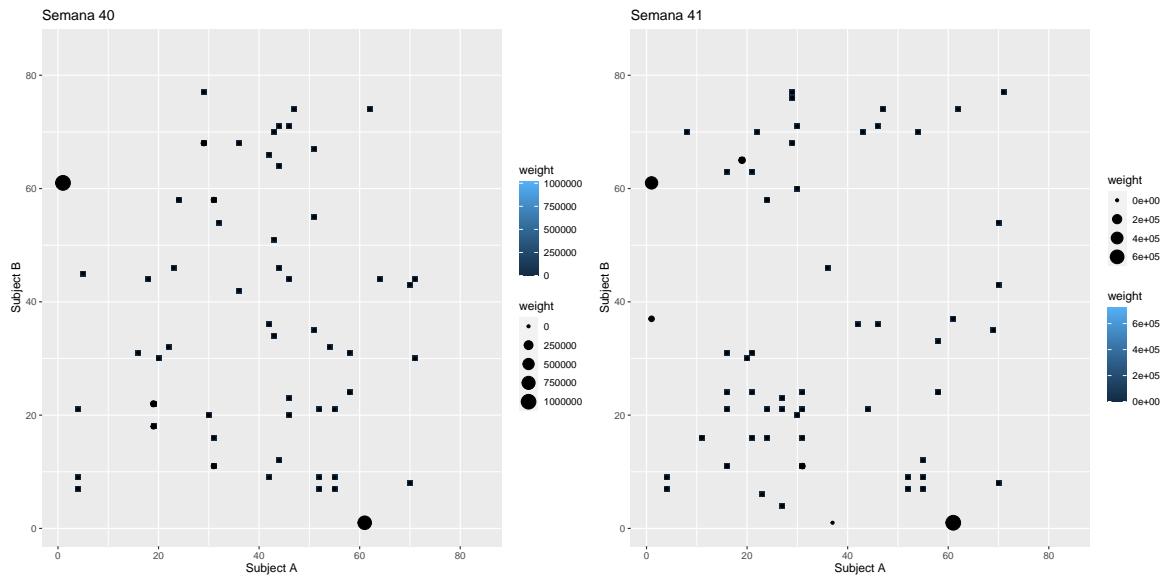
```
plots <- list()  
  
for (sel_week in 1:2) {  
  prox_week <-  
    communication_by_week %>% filter(week == 39 + sel_week) %>% dplyr::select(-week)  
  plots[[sel_week]] <-  
    ggplot(prox_week, aes(x = user_id.x, y = user_id.y)) +
```

```

    geom_tile(aes(fill = weight)) +
    geom_point(aes(size = weight)) +
    lims(x = c(1, 84), y = c(1, 84)) +
    labs(x = "Subject A",
         y = "Subject B",
         title = paste("Semana", 39 + sel_week))
}

wrap_plots(plots, ncol = 2)

```



Ahora construimos la rejilla vacía para imputar los datos faltantes.

```

grid.weeks <- 0:78
grid.users <- subjects$user_id
communication_full_grid <- expand.grid(week = grid.weeks, user_id.x = grid.users, user_id.y = grid.us
communication_full_grid$weight = 0

communication_full_grid <- communication_full_grid %>%
  full_join(communication_by_week) %>%
  group_by(week, user_id.x, user_id.y) %>%
  summarise(weight = max(weight), .groups = 'drop') %>%
  arrange(week, user_id.x, user_id.y)

## Joining, by = c("week", "user_id.x", "user_id.y", "weight")

```

Convertimos la lista de arcos en matriz de adyacencia para usarla en la construcción de la tabla final.

```

communication_adj_matrix <- communication_full_grid %>%
  pivot_wider(names_from = user_id.y,
              values_from = weight,
              values_fill = 0)

write.csv2(
  communication_adj_matrix, "communication_adj_matrix.csv",
  row.names = FALSE)

```

8. Grafo de amistad

La encuesta sobre relaciones incluye varios grados de relación. Para modelar la amistad pensando en el impacto en los síntomas de gripe, las dos relaciones más relevantes serían `CloseFriend` y `SocializeTwicePerWeek`. Sobre las filas de estas dos categorías vamos a construir la matriz de adyacencia del grafo de amistad.

```
base_dir <- '.../datasets/SocialEvolution/'

relationships <- read.csv2(
  paste0(base_dir, 'RelationshipsFromSurveys.csv'),
  , sep = ',')
subjects <- read.csv2(paste0(base_dir, 'Subjects.csv'), sep = ',')

str(relationships)

## 'data.frame': 31918 obs. of 4 variables:
## $ id.A      : int 47 2 57 73 55 47 51 52 40 66 ...
## $ id.B      : int 2 2 2 2 2 2 2 2 2 2 ...
## $ relationship: Factor w/ 5 levels "BlogLivejournalTwitter",...: 2 2 2 5 5 5 5 5 5 ...
## $ survey.date : Factor w/ 6 levels "2008-09-09","2008-10-19",...: 1 1 1 1 1 1 1 1 1 1 ...

levels(relationships$relationship)

## [1] "BlogLivejournalTwitter" "CloseFriend"
## [3] "FacebookAllTaggedPhotos" "PoliticalDiscussant"
## [5] "SocializeTwicePerWeek"

levels(relationships$survey.date)

## [1] "2008-09-09" "2008-10-19" "2008-12-13" "2009-03-05" "2009-04-17"
## [6] "2009-05-18"
```

Aunque la fecha viene con dia, no hay más de una encuesta por mes, así que seguiremos como en la matriz de actividades, ordenando los datos por mes para llevarlos luego a semanas. De nuevo tenemos datos que faltan en los meses, y haremos la misma suposición de que las cosas no cambian en los meses intermedios.

8.1. Construcción del grafo por semanas

De todas las relaciones incluidas en el dataset, solo vamos a usar “`CloseFriends`” y “`SocializeTwicePerWeek`”, que son más relevantes para la transmisión de enfermedades.

```
relationships <- relationships %>%
  mutate(
    time = parse_date_time(survey.date, 'y-m-d'),
    year = year(time)
  ) %>%
  filter(year >= 2008)

friendship_time <- relationships %>%
  filter(relationship %in% c(
    "CloseFriend",
```

```

    "SocializeTwicePerWeek"
)) %>%
transmute(
  year = year,
  month = month(time),
  user_id.x = id.A,
  user_id.y = id.B,
  weight = 1
) %>%
filter(!is.na(user_id.y)) %>% # destinatario desconocido
filter(user_id.x != user_id.y) %>% # a si mismo
group_by(year, month, user_id.x, user_id.y) %>%
summarize(
  weight = sum(weight),
  .groups = 'drop'
)

```

Ahora es necesario imputar los meses que faltan para poder aplicar la multiplicación de matrices de manera sencilla. Para ello creamos una asignación entre los meses originales y los que se copiarán para llenar la rejilla de meses, al igual que se hizo para *Actividades*.

```

# "2008-09-09" "2008-10-19" "2008-12-13" "2009-03-05" "2009-04-17" "2009-05-18"
month_grid <- data.frame(
  year      = c(2008, 2008, 2008, 2008, 2008, 2009, 2009, 2009),
  month     = c( 9,   10,   10,   12,   12,   12,   3,   4,   5),
  year.filled = c(2008, 2008, 2008, 2009, 2009, 2009, 2009, 2009),
  month.filled = c( 9,   10,   11,   12,   1,    2,    3,    4,    5)
)
month_grid

```

year	month	year.filled	month.filled
2008	9	2008	9
2008	10	2008	10
2008	10	2008	11
2008	12	2008	12
2008	12	2009	1
2008	12	2009	2
2009	3	2009	3
2009	4	2009	4
2009	5	2009	5

Hacemos el cruce con los datos originales para que se copien los valores en los huecos vacíos.

```

friendship_full_months <- month_grid %>%
  left_join(
    friendship_time,
    by = c('year', 'month')
  ) %>% arrange(year, month.filled, month)

```

Se puede ver como la cuenta de elementos coincide en los meses que han sido rellenados con el mismo de origen.

Lo siguiente es organizar el dataset para que solo quede una fila por año y mes.

```

friendship_full_months <- friendship_full_months %>% transmute(
  year = year.filled,
  month = month.filled,
  user_id.x = user_id.x,
  user_id.y = user_id.y,
  weight = weight
)

```

Después tenemos que añadir las semanas siguiendo el mismo proceso, generar una rejilla de semana y luego hacer un left join con los datos originales para que se dupliquen.

```

# ultimo domingo de 2007 como fecha base para que las semanas sean regulares entre años
base = parse_date_time('2007/12/30 00:00', 'y-m-d H:M')

norm_week <- function(time) {
  as.integer(round(difftime(time, base, units = "weeks")))
}

month_from <- function(norm_week) {
  month(base + weeks(norm_week))
}

year_from_norm_week <- function(norm_week) {
  year(base + weeks(norm_week))
}

weeks <-
norm_week(
  parse_date_time('2008.9', 'y.m')
):norm_week(
  parse_date_time('2009.5', 'y.m')
)
user_ids <- unique(subjects$user_id)

friendship_full_grid <- data.frame(week = weeks)
friendship_full_grid$year = year_from_norm_week(friendship_full_grid$week)
friendship_full_grid$month = month_from(friendship_full_grid$week)

friendship_full_grid <- friendship_full_grid %>%
  left_join(friendship_full_months,
            by = c('year', 'month')) %>% arrange(year, month, week)

```

En esa cuenta se puede ver que coinciden las cifras de los datos que han sido copiados de cada mes a las semanas.

Dejamos solo las columnas relevantes y quitamos algún NA que se había generado por no acertar con las semanas inicial y/o última.

```

friendship_full_grid <- friendship_full_grid %>%
  transmute(
    week = week,
    user_id.x = user_id.x,
    user_id.y = user_id.y,
    weight = weight
  ) %>% filter(!is.na(user_id.x))
str(friendship_full_grid)

```

```

## 'data.frame':   35649 obs. of  4 variables:
## $ week      : int  36 36 36 36 36 36 36 36 36 36 ...
## $ user_id.x: int  1 2 2 2 2 2 2 2 2 2 ...
## $ user_id.y: int  61 3 13 25 29 34 38 47 49 51 ...
## $ weight    : num  2 1 1 1 1 1 1 1 1 1 ...

Lo que nos falta ahora son los pares de usuarios ausentes, los que tendrían 0 en weight. Para ello es necesario crear una rejilla con todo a ceros y combinarla con los datos reales, como hemos hecho antes.

user_ids <- unique(subjects$user_id)
friendship_empty_grid <- expand.grid(
  week = unique(friendship_full_grid$week),
  user_id.x = user_ids,
  user_id.y = user_ids
) %>% arrange(week, user_id.x, user_id.y) %>%
ungroup()

friendship_empty_grid$weight = 0

friendship_edge_list <- friendship_empty_grid %>%
  full_join(friendship_full_grid) %>%
  group_by(week, user_id.x, user_id.y) %>%
  summarise(weight = max(weight)) %>%
  ungroup() %>% arrange(week, user_id.x, user_id.y)

## Joining, by = c("week", "user_id.x", "user_id.y", "weight")

## `summarise()` regrouping output by 'week', 'user_id.x' (override with `.`groups` argument)

table(friendship_edge_list$weight)

##
##      0      1      2
## 211311  24393  11256

table(friendship_full_grid$weight)

##
##      1      2
## 24393  11256

```

Con las tablas de contingencia se comprueba que solo se han añadido los ceros y no se han modificado los datos originales.

Ahora que tenemos las listas de arcos, construimos la matriz de adyacencia del grafo de amistades.

```

friends_adj_matrix <- friendship_edge_list %>%
  pivot_wider(names_from = user_id.y,
              values_from = weight,
              values_fill = 0)
write.csv2(friends_adj_matrix, "friends_adj_matrix.csv", row.names = FALSE)

```

Y también podemos hacer un plot similar a los que hemos hecho con las otras matrices.

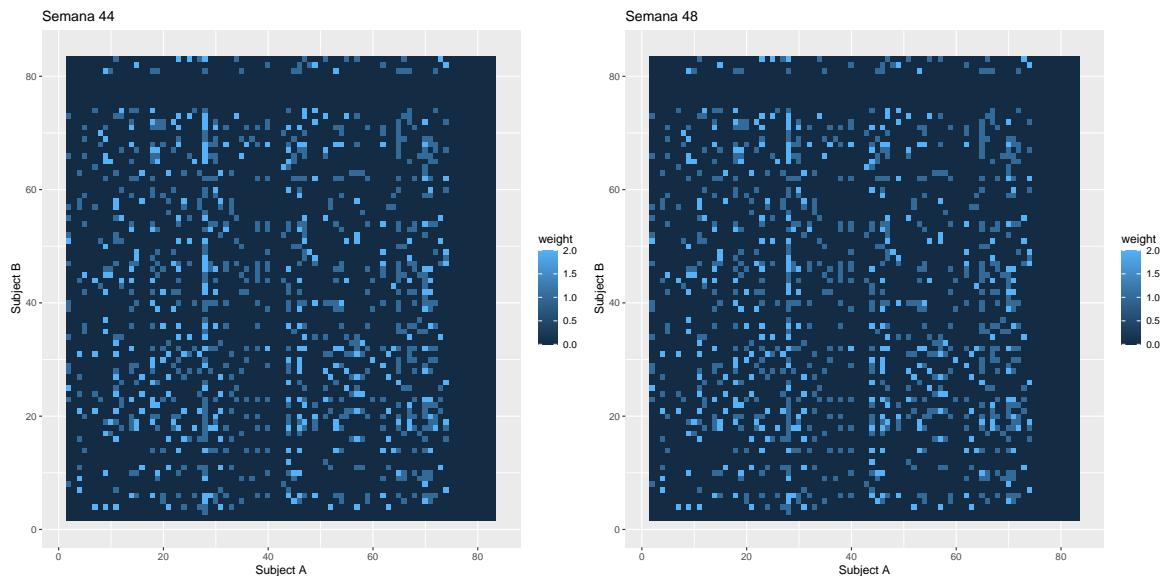
```

plots <- list()

for (sel_week in 1:2) {
  prox_week <-
    friendship_edge_list %>%
    filter(week == 40 + 4 * sel_week) %>%
    dplyr::select(-week)
  plots[[sel_week]] <-
    ggplot(prox_week, aes(x = user_id.x, y = user_id.y)) +
    geom_tile(aes(fill = weight)) +
    lims(x = c(1, 84), y = c(1, 84)) +
    labs(x = "Subject A",
         y = "Subject B",
         title = paste("Semana", 40 + 4 * sel_week))
}

wrap_plots(plots, ncol = 2)

```



Se vislumbra poco movimiento, poco cambio en las amistades, aunque también es porque no tenemos tantas encuestas.

Vamos a calcular otra matriz de relaciones a través de las actividades. Esta es un poco diferente, debido a que los datos los tenemos por mes, así que tendremos que desagregar replicando las columnas de forma que podamos usarlos por semana.

Además será necesario imputar los meses que faltan. Por la naturaleza de los datos, que se refieren a actividades o cursos matriculados, la ausencia de un mes no implica que el usuario se haya desmatriculado. Por eso se asumirá que en los meses que no hay datos, continúan vigentes los datos anteriores.

Además, igual que con los datos de wifi, hay que hacer un cruce de la tabla con sigo misma para ver que pares de sujetos están juntos en una actividad.

9. Grafo de interacción por actividades comunes

Se cargan los datos y se muestra la estructura.

```

base_dir <- '../datasets/SocialEvolution/'

activities <- read.csv2(paste0(base_dir, 'Activities.csv'), sep = ',')
subjects <- read.csv2(paste0(base_dir, 'Subjects.csv'), sep = ',')

str(activities)

## 'data.frame': 695 obs. of 3 variables:
## $ user.id : int 9 44 66 6 30 71 46 35 32 28 ...
## $ campus.organization: int 44 44 44 44 44 44 44 44 44 57 ...
## $ survey.month : Factor w/ 4 levels "2008.09","2009.03",...: 1 1 1 1 1 1 1 1 1 1 ...

```

9.1. Ajuste de los datos a la rejilla de tiempo

En este caso, como la encuesta solo se recogió en los meses de septiembre de 2008, marzo, abril y junio de 2009, necesitamos imputar los valores intermedios.

```

unique(activities$survey.month)

## [1] 2008.09 2009.03 2009.04 2009.06
## Levels: 2008.09 2009.03 2009.04 2009.06

```

A priori, al tratarse de cursos universitarios, podemos afirmar que si se declara una actividad en septiembre se continuaría con ella hasta el siguiente informe de marzo. Lo mismo haremos con el mes de mayo que nos falta y asumiremos el valor de abril.

Para ello, completamos los datos duplicando el valor como se ha dicho. La copia se realiza creando un data frame que en una columna tiene el mes original y en otra el mes donde será copiado. Al hacer un cruce con los datos originales, las filas se replicarán en aquellos registros que tienen el mes original.

Luego se descarta la columna del mes original y quedaría una tabla con todos los meses rellenos.

```

mapping.copy <- data.frame(
  survey.month = c(
    "2008.09",
    "2008.09",
    "2008.09",
    "2008.09",
    "2008.09",
    "2008.09",
    "2008.09",
    "2009.03",
    "2009.04",
    "2009.04",
    "2009.06"
  ),
  full.month =   c(
    "2008.09",
    "2008.10",
    "2008.11",
    "2008.12",
    "2009.01",
    "2009.02",
    "2009.03",
    "2009.04",
    "2009.05",
    "2009.06"
  )
)

```

```

    "2009.06"
)
)
head(mapping.copy)

```

	survey.month	full.month
2008.09	2008.09	
2008.09	2008.10	
2008.09	2008.11	
2008.09	2008.12	
2008.09	2009.01	
2008.09	2009.02	

Realizamos la copia con un left join entre la tabla de copias y la inicial.

```

activities_full <- mapping.copy %>%
  left_join(
    activities,
    by = "survey.month"
  ) %>%
  transmute(
    survey.month = full.month,
    user.id = user.id,
    activity = campus.organization
  ) %>%
  arrange(survey.month, user.id)

```

Ahora ponemos columnas año y mes para ayudarnos a insertar los datos en la rejilla Semana-Usuario que estamos usando para todas las demás matrices.

```

base = parse_date_time('2007/12/30 00:00', 'y-m-d H:M')

activities_time <- activities_full %>% mutate(
  time = parse_date_time(survey.month, "y.m")
) %>%
  transmute(
    year = year(time),
    month = month(time),
    user_id = user.id,
    activity = activity
  ) %>%
  filter(year >= 2008) %>% arrange(year, month, user_id)

```

A continuación hay que expandir los datos para que aparezcan agrupados por semana.

Inicialmente, lo que haremos es generar una rejilla completa para imputar los usuarios faltantes, añadir una columna año y mes y hacer un join con los datos originales, de tal manera que se repliquen adecuadamente.

```

grid.months <- unique(activities_full$survey.month)
grid.users <- unique(subjects$user_id)
activities_full_grid <- expand.grid(time = grid.months, user_id = grid.users)

```

```

activities_full_grid <- activities_full_grid %>%
  mutate(
    time = parse_date_time(time, 'y.m')
  ) %>%
  transmute(
    year = year(time),
    month = month(time),
    user_id = user_id
  ) %>% arrange(year, month, user_id)

```

Con el mes el año y el usuario ya podemos hacer left join con la tabla de datos original.

```

activities_full_grid_2 <- activities_full_grid %>%
  left_join(
    activities_time,
    by=c('year', 'month', 'user_id')
  )

```

Aún nos falta poner los datos por semana para que la rejilla sea la misma que el resto de matrices de adyacencia.

Para ello generamos una rejilla con las semanas, mes y año asociado y luego hacemos un left join que cause que se dupliquen los datos de cada mes por la semana.

```

activity_week_grid <- data.frame(week = 35:74) %>%
  transmute(
    year = year(base + weeks(week)),
    month = month(base + weeks(week)),
    week = week
  )

activities_full_grid_3 <- activity_week_grid %>%
  left_join(
    activities_full_grid_2,
    by = c("year", "month")
  ) %>%
  arrange(year, month, week) %>%
  filter(!is.na(user_id))

```

9.2. Generación del grafo de actividades compartidas

Lo siguiente es cruzar la tabla con sigo misma por Actividad y semana para ver que usuarios han compartido actividad, al igual que se hizo con las wifis.

```

activities_shared <- activities_full_grid_3 %>%
  left_join(
    activities_full_grid_3,
    by = c("week", "activity")
  ) %>%
  filter(user_id.x != user_id.y) %>%
  transmute(
    week = week,
    user_id.x = user_id.x,
    user_id.y = user_id.y,
  )

```

```

        activity = activity
) %>%
arrange(
  week, user_id.x, user_id.y
)

```

Finalmente, para generar la lista de ejes, agrupamos por semana, sujeto x y sujeto y y contamos las actividades:

```

activities_edge_list <- activities_shared %>%
  group_by(week, user_id.x, user_id.y) %>%
  summarise(
    weight = n(),
    .groups = 'drop'
) %>% ungroup()
head(activities_edge_list)

```

week	user_id.x	user_id.y	weight
36	1	4	4
36	1	6	4
36	1	7	4
36	1	8	4
36	1	9	12
36	1	10	4

En esta tabla no tenemos todos los pares de usuario en las semanas. Necesitamos completar los datos de nuevo con una rejilla:

```

weeks <- unique(activities_shared$week)
user_ids <- unique(subjects$user_id)
activities_edge_grid <-
  expand.grid(week = weeks,
             user_id.x = user_ids,
             user_id.y = user_ids)
activities_edge_grid$weight = 0

activities_edge_full <- activities_edge_grid %>%
  full_join(activities_edge_list) %>%
  group_by(week, user_id.x, user_id.y) %>%
  summarise(weight = max(weight), .groups = 'drop') %>%
  ungroup() %>% arrange(week, user_id.x, user_id.y)

## Joining, by = c("week", "user_id.x", "user_id.y", "weight")

```

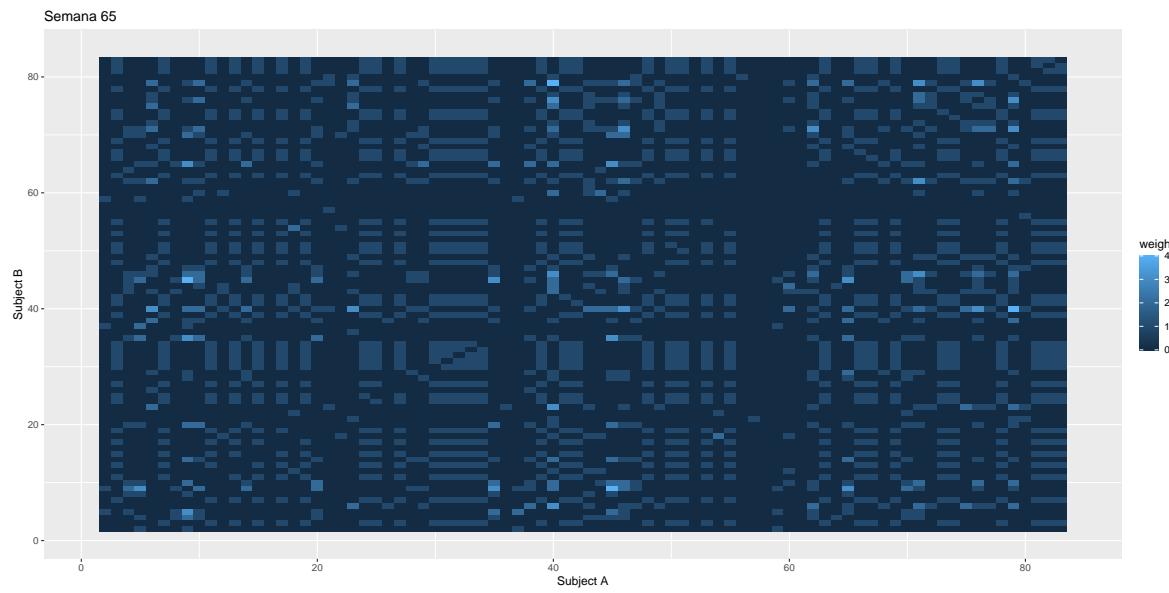
Se muestra un gráfico de la coincidencia en una semana

```

sel_week <- 65
prox_week <-
  activities_edge_full %>% filter(week == sel_week) %>% dplyr::select(-week)
ggplot(prox_week, aes(x = user_id.x, y = user_id.y)) +
  geom_tile(aes(fill = weight)) +
  lims(x = c(1, 84), y = c(1, 84)) +

```

```
labs(x = "Subject A",
     y = "Subject B",
     title = paste("Semana", sel_week))
```



Ahora construimos la matriz de adyacencia a partir del data.frame de lista de arcos:

```
activity_adj_matrix <- activities_edge_full %>%
  pivot_wider(names_from = user_id.y,
              values_from = weight,
              values_fill = 0)
write.csv2(activity_adj_matrix, "activity_adj_matrix.csv", row.names = FALSE)
```

10. Construcción de la matriz de síntomas

Se busca construir una matriz de síntomas por usuario y semana para luego poder calcular la *exposición* de manera simple como multiplicación de matrices.

Para ello vamos a cargar el dataset “flu”. Al igual que con el resto de datasets, añadiremos una serie de columnas de tiempo para facilitar el procesamiento posterior. De nuevo, la columna de semanas se calculará con referencia al 30 de Diciembre de 2007.

Además, se agregan por semana los síntomas.

```
library(tidyr)
library(tibble)
library(dplyr)
library(ggplot2)
library(patchwork)
library(visdat)
library(data.table)
library(graphics)
library(lubridate)
library(lemon)
knit_print.data.frame <- lemon_print
```

```

base_dir <- '../datasets/SocialEvolution/'

flu <- read.csv2(paste0(base_dir, 'FluSymptoms.csv'), sep = ',', )
subjects <- read.csv2(paste0(base_dir, 'Subjects.csv'), sep = ',', )

base <- parse_date_time('2007/12/30 00:00', 'y-m-d H:M')

flu_time <- flu %>% mutate(
  time = parse_date_time(time, "y-m-d H:M:S")
) %>%
  mutate(
    weekday = wday(time),
    week = as.integer(round(difftime(time, base, units = "weeks"))),
    year_week = week(time),
    hour = hour(time),
    month = month(time),
    year = year(time),
    year_day = yday(time)
) %>% filter(year >= 2008) %>% arrange(week, user_id) %>%
ungroup()

```

```

symptoms_by_weeks <- flu_time %>% filter(year == 2009) %>%
  group_by(week, user_id) %>%
  summarize(
    sore.throat.cough = sum(sore.throat.cough),
    runnynose.congestion.sneezing = sum(runnynose.congestion.sneezing),
    fever = sum(fever),
    nausea.vomiting.diarrhea = sum(nausea.vomiting.diarrhea),
    sad.depressed = sum(sad.depressed),
    open.stressed = sum(open.stressed)
) %>% ungroup()

```

```
#> `summarise()` regrouping output by 'week' (override with `.`groups` argument)
```

```
write.csv2(symptoms_by_weeks, "symptoms_by_weeks.csv", row.names = FALSE)
```

```
head(symptoms_by_weeks)
```

week	user_id	sore.throat.cough	runnynose.congestion.sneezing	fever	nausea.vomiting.diarrhea	sad.depressed	open.stressed
54	2	0	0	0	1	3	2
54	4	0	3	0	3	0	0
54	9	0	0	0	0	1	0
54	12	1	2	0	1	1	1
54	20	1	1	1	1	1	1
54	27	0	0	0	0	0	0

```
summary(symptoms_by_weeks)
```

week	user_id	sore.throat	cough	runny.nose	congestion	sneezing	nausea	vomiting	diarrhea	sad.depressed	open.stressed
Min.	Min. :	Min. :	Min. :	0.000	Min.	Min. :0.0000	Min.	Min. :0.0000	Min.	Min. :0.0000	Min. :0.000
:54.00	2.00	0.0000			:0.00000				:0.0000		0.000
1st	1st	1st	Qu.:	1st Qu.:0.000	1st	1st			1st	1st	Qu.:0.00000.000
Qu.:58.00	Qu.:23.00	Qu.:0.0000			Qu.:0.00000	Qu.:0.00000			Qu.:0.00000.000		
Median	Median	Median	Median :	Median :0.000	Median	Median			Median	Median	
:61.00	:44.00	:44.00	0.0000		:0.00000				:0.0000		:0.000
Mean	Mean	Mean	Mean :	Mean :1.273	Mean	Mean	Mean :0.1752	Mean	Mean	Mean	
:60.82	:42.64	:42.64	0.8457		:0.04984				:0.4662		1.031
3rd	3rd	3rd	Qu.:	3rd Qu.:1.000	3rd	3rd			3rd	3rd	
Qu.:64.00	Qu.:62.00	Qu.:62.00	1.0000		Qu.:0.00000	Qu.:0.00000			Qu.:0.00000	Qu.:0.00000	1.000
Max.	Max.	Max.		Max. :16.000	Max.	Max. :7.0000	Max.	Max. :9.0000	Max.	Max.	
:69.00	:80.00	:80.00			:4.00000				:9.0000		:11.000

Construimos la rejilla completa al igual que hemos hecho con el resto de datasets para completar los datos que faltan. En este caso, los datos no declarados se completan con ceros y se asume ausencia del síntoma.

```
grid.weeks <- 40:70
grid.users <- subjects$user_id
symptoms_full_grid <- expand.grid(week = grid.weeks, user_id = grid.users)
symptoms_full_grid$sore.throat.cough = 0
symptoms_full_grid$fever = 0
symptoms_full_grid$runny.nose.congestion.sneezing = 0
symptoms_full_grid$nausea.vomiting.diarrhea = 0
symptoms_full_grid$sad.depressed = 0
symptoms_full_grid$open.stressed = 0

head(symptoms_full_grid)
```

week	user_id	sore.throat	cough	fever	runny.nose	congestion	sneezing	vomiting	diarrhea	sad.depressed	open.stressed
40	1	0	0		0		0		0		0
41	1	0	0		0		0		0		0
42	1	0	0		0		0		0		0
43	1	0	0		0		0		0		0
44	1	0	0		0		0		0		0
45	1	0	0		0		0		0		0

Combinamos los datos de la matriz real. En la práctica esto supone imputar con ceros los datos faltantes.

```
symptoms_full_grid <- symptoms_full_grid %>%
  full_join(symptoms_by_weeks) %>%
  group_by(week, user_id) %>%
  summarise(
    sore.throat.cough = max(sore.throat.cough),
    fever = max(fever),
    runny.nose.congestion.sneezing = max(runny.nose.congestion.sneezing),
    nausea.vomiting.diarrhea = max(nausea.vomiting.diarrhea),
    sad.depressed = max(sad.depressed),
```

```

open.stressed = max(open.stressed),
  .groups = 'keep'
) %>% ungroup()

## Joining, by = c("week", "user_id", "sore.throat.cough", "fever", "runnynose.congestion.sneezing",
head(symptoms_full_grid %>% filter(runnynose.congestion.sneezing > 0))

```

week	user_id	sore.throat.cough	fever	runnynose.congestion.sneezing	vomiting	disorders	depressed	open.stressed
54	4	0	0	3	3	0	0	0
54	12	1	0	2	1	1	1	1
54	20	1	1	1	1	1	1	1
54	46	3	0	1	0	0	0	0
54	57	0	0	2	0	0	0	0
54	66	0	0	2	0	0	0	0

Guardamos la matriz de síntomas.

```
write.csv2(symptoms_full_grid, "symptoms_full_grid.csv", row.names = FALSE)
```

Una vez generados todos los grafos que vamos a usar para modelar la expansión de síntomas de la gripe, es necesario construir la tabla de datos que alimentará al modelo.

Esta tabla de datos se basa en calcular, para cada periodo, es decir para cada semana, un valor de exposición al síntoma en las tres semanas anteriores basado en cada uno de los modos de interacción que hemos visto. Por ello, la tabla tendrá tres columnas para cada grafo de interacción y una columna final, la variable dependiente, con el valor del síntoma en la semana actual.

El cálculo de la exposición se detalla más adelante, pero abreviando se puede decir que se basa en contar todas las interacciones con personas afectadas del síntoma en cada una de las semanas.

Al final, lo importante es la relación entre esas columnas, que reflejan la evolución del síntoma en ventanas de 4 semanas, por eso se omitirá el id del usuario y el número de semana para hacer el modelado.

11. Construcción de la matriz de datos

11.1. Análisis de la exposición al síntoma mediante interacciones

11.1.1. Carga de datos

```

wifi_adj <- read.csv2('wlan_adjacency_by_week.csv')
bluetooth_adj <- read.csv2('proximity_adj_matrix_by_week.csv')
communication_adj <- read.csv2('communication_adj_matrix.csv')
activity_adj <- read.csv2('activity_adj_matrix.csv')
friends_adj <- read.csv2('friends_adj_matrix.csv')

symptoms <- read.csv2('symptoms_full_grid.csv')

base_dir <- '.../datasets/SocialEvolution/'
subjects <- read.csv2(paste0(base_dir, 'Subjects.csv'), sep = ',')

```

11.1.2. Verificación de la estructura de los grafos

Verificamos que las matrices de los grafos son cuadradas y tienen los registros adecuados, deben ser 86 columnas (84 individuos mas la columna del user_id y la de la semana) y 84 registros por cada semana.

```
count.no.84 <- function(df) {  
  dim(df %>%  
    group_by(week) %>%  
    summarise(count = n(), .groups = "drop") %>%  
    filter(count != 84))[[1]]  
}  
  
diags <- function(df) {  
  weeks <- min(df$week):max(df$week)  
  sumdiags <- 0  
  for (w in weeks) {  
    m <- df %>% filter(week == 40) %>% select(-week,-user_id.x)  
    sumdiags <- sumdiags + sum(diag(as.matrix(m)))  
  }  
  sumdiags  
}  
  
data.frame(  
  dataset = c(  
    "wifi",  
    "bluetooth",  
    "calls+sms",  
    "activities",  
    "friends",  
    "symptoms"  
)  
,  
  weeks.without.84.rows = c(  
    count.no.84(wifi_adj),  
    count.no.84(bluetooth_adj),  
    count.no.84(communication_adj),  
    count.no.84(activity_adj),  
    count.no.84(friends_adj),  
    "NO APLICA"  
)  
,  
  min_week = c(  
    min(wifi_adj$week),  
    min(bluetooth_adj$week),  
    min(communication_adj$week),  
    min(activity_adj$week),  
    min(friends_adj$week),  
    min(symptoms$week)  
)  
,  
  max_week = c(  
    max(wifi_adj$week),  
    max(bluetooth_adj$week),  
    max(communication_adj$week),  
    max(activity_adj$week),  
    max(friends_adj$week),  
    max(symptoms$week)
```

```

),
diagonals = c(
  diags(wifi_adj),
  diags(bluetooth_adj),
  diags(communication_adj),
  diags(activity_adj),
  diags(friends_adj),
  "NO APLICA"
)
)

```

dataset	weeks.without.84.rows	min_week	max_week	diagonals
wifi	0	40	74	0
bluetooth	0	2	81	0
calls+sms	0	0	78	0
activities	0	36	74	0
friends	0	36	70	0
symptoms	NO APLICA	40	70	NO APLICA

Se puede ver en la columna *weeks.without.84.rows* que ninguna de las matrices de adyacencia es incompleta, todas tienen todas sus semanas con 84 filas. También se puede ver que *las diagonales son siempre cero en las matrices*. Con esto nos aseguramos que el valor del propio sujeto no cuenta para calcular la exposición, como ahora veremos.

11.1.3. Descripción de los conceptos necesarios para construir la matriz combinada

En cuanto a la construcción de la matriz combinada, la idea es reflejar la evolución en cada periodo de un síntoma en un *sujeto principal*. Para ello se miran los 3 períodos anteriores y para cada uno de ellos se calcula el nivel de exposición. El nivel de exposición tiene en cuenta que otros sujetos tenían el mismo síntoma, con cuantos de ellos y con que intensidad se ha relacionado el *sujeto principal*.

Como las matrices de adyacencia nos dan una “cuantía de la relación” (haber estado en las mismas wifis, hablar por teléfono, compartir actividades...) calcularemos la exposición como el producto de la “cuantía de relación” con el nivel de síntomas que hemos registrado para cada sujeto secundario.

$$E[\text{week}]_i = \sum_{j \neq i} G[\text{week}]_{i,j} * S[\text{week}]_j$$

Donde $G_{i,j}$ es el valor de la matriz de adyacencia de uno de los grafos (wifi, bluetooth, etc) en la fila i y columna j .

Teniendo en cuenta que los registros son temporales, consideramos que:

$$S_i[t] = f(E_i[t-1], E_i[t-2], E_i[t-3])$$

Es decir, que el estado actual de un síntoma está relacionado con las exposiciones al síntoma en tres períodos anteriores. En el mejor caso sería una relación lineal, pero si no lo es podría ser un árbol de decisión o una fórmula no lineal calculada mediante una red neuronal.

Así pues, construimos una matriz de datos con la estructura siguiente:

Para cada Sujeto i , para cada periodo t

- Tres columnas con la Exposición E_i en $t-1, t-2, t-3$ por cada uno de los grafos de interacciones (Wifi, Bluetooth, Llamadas, Actividades, Amistad).
- Una columna con el estado del Síntoma en el periodo t o $S_i[t]$.

Al final, la matriz es como una ventana deslizante de 4 semanas a través de el historial de contactos y síntomas de cada sujeto.

11.1.4. Composición de la matriz combinada

Debemos generar una rejilla que tenga todas las semanas del rango que vamos a estudiar y por cada semana, todos los usuarios. Después, para calcular el valor de cada columna de periodos, usaremos mutate usando la columna de la semana y el usuario como parámetros.

```
weeks <-
  (40 + 4):70 # desplazamos el inicio 4 semanas para que haya datos suficientes para calcular los sintomas
user_ids <- unique(subjects$user_id)

exposition_matrix <-
  expand.grid(week = weeks, user_id = user_ids) %>%
  arrange(week, user_id)
```

Definimos funciones de R para calcular el valor de exposición $E_i[t]$ y el valor de síntoma $S_i[t]$.

```
exposition <-
  function(s_week,
          s_user_id,
          adjacency_matrix,
          symptom_name,
          symptoms) {
    row <- adjacency_matrix %>%
      filter(week == s_week) %>%
      filter(user_id.x == s_user_id) %>%
      select(-week, -user_id.x)
    symptom_col <-
      symptoms %>% filter(week == s_week) %>% select(symptom_name)
    ## producto escalar
    as.matrix(row) %*% as.matrix(symptom_col)
  }

symptom_value <-
  function (s_week, s_user_id, symptom_name, symptoms) {
    as.matrix(
      symptoms %>%
        filter(week == s_week) %>%
        filter(user_id == s_user_id) %>%
        select(symptom_name)
    )
  }
```

Ahora generamos la matriz usando estas dos funciones y moviendo los parámetros. La forma de generar la matriz es similar a lo que se haría con Excel, poniendo todos los períodos multiplicados por todos los usuarios en las primeras dos columnas y luego unas fórmulas que se replican para cada fila.

Para generar la matriz vamos a escoger el síntoma “Congestión”.

```
symptom <- 'runny_nose.congestion.sneezing'
compound_data <- exposition_matrix %>%
  rowwise() %>%
  mutate(
    wifi_1 = exposition(week - 1, user_id, wifi_adj, symptom, symptoms),
    wifi_2 = exposition(week - 2, user_id, wifi_adj, symptom, symptoms),
    wifi_3 = exposition(week - 3, user_id, wifi_adj, symptom, symptoms),
    bluetooth_1 = exposition(week - 1, user_id, bluetooth_adj,
```

```
                symptom, symptoms),
bluetooth_2 = exposition(week - 2, user_id, bluetooth_adj,
                         symptom, symptoms),
bluetooth_3 = exposition(week - 3, user_id, bluetooth_adj,
                         symptom, symptoms),
communication_1 = exposition(week - 1, user_id,
                             communication_adj, symptom, symptoms),
communication_2 = exposition(week - 2, user_id,
                             communication_adj, symptom, symptoms),
communication_3 = exposition(week - 3, user_id,
                             communication_adj, symptom, symptoms),
friends_1 = exposition(week - 1, user_id, friends_adj, symptom, symptoms),
friends_2 = exposition(week - 2, user_id, friends_adj, symptom, symptoms),
friends_3 = exposition(week - 3, user_id, friends_adj, symptom, symptoms),
activity_1 = exposition(week - 1, user_id, activity_adj, symptom, symptoms),
activity_2 = exposition(week - 2, user_id, activity_adj, symptom, symptoms),
activity_3 = exposition(week - 3, user_id, activity_adj, symptom, symptoms),
symptom = symptom_value(week, user_id, symptom, symptoms)
)
```

11.1.5. Presentación descriptiva de las características de la matriz combinada

Vamos a ver la estructura y la distribución del dataset que hemos construido.

```
str(compound_data)
```

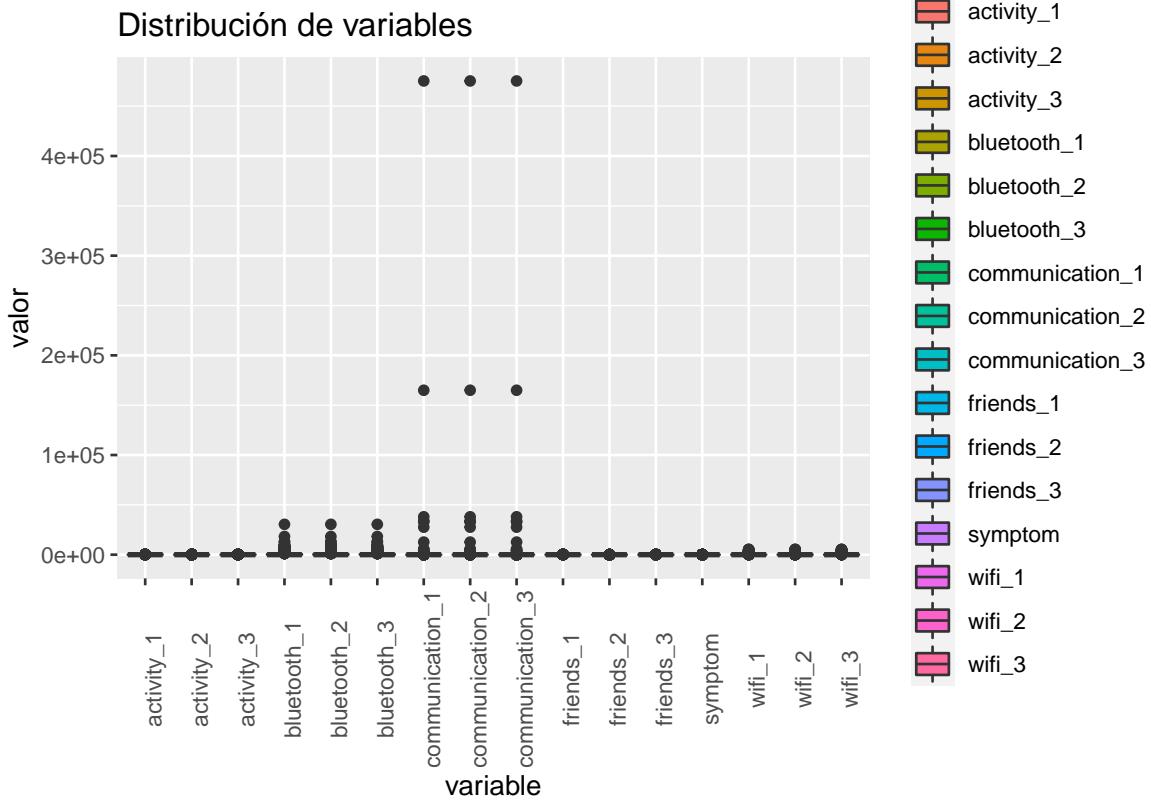
`summary(compound_data)`

Ahora mostramos las distribuciones de cada una de las variables

```
data <- compound_data %>% select(-week, -user_id)
## quitamos las duplicadas
data <- data[!duplicated(data), ]

data2 <-
  data %>%
  pivot_longer(colnames(data), names_to = "variable", values_to = "value")

ggplot(data2, aes(x = variable, y = value)) +
  geom_boxplot(aes(fill = variable)) +
  theme(axis.text.x = element_text(angle = 90)) +
  labs(title = "Distribución de variables") +
  xlab("variable") + ylab("valor")
```



Se puede ver que la mayoría de las variables están muy sesgadas hacia la derecha, esto probablemente complicará la linealidad del modelo y debamos acudir a otras soluciones de minería.

11.2. Análisis de la influencia de las interacciones en el síntoma “Congestión”

Probamos un modelo lineal básico, aunque la poca normalidad de las variables no haga tener muchas esperanzas con esto. También eliminamos las columnas que no necesitamos y las filas que tienen todo ceros.

```
data <- compound_data %>% select(-week, -user_id) %>% ungroup()
## quitamos las columnas que tienen todo a cero
data <-
  data %>%
    rowwise() %>%
    mutate(sum = sum(cur_data())) %>%
    filter(sum > 0) %>%
    select(-sum) %>%
    ungroup()
## quitamos las duplicadas
data <- data[!duplicated(data),]
```

Planteamos el modelo lineal.

```
linear.model.001 <- lm(symptom ~ ., data)
summary(linear.model.001)
```

```
##
```

```

## Call:
## lm(formula = symptom ~ ., data = data)
##
## Residuals:
##    Min      1Q  Median      3Q     Max 
## -5.0809 -0.6128 -0.4737 -0.3931 13.3355 
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)    
## (Intercept) 4.407e-01 7.117e-02   6.192 8.28e-10 ***
## wifi_1       1.621e-04 8.154e-05   1.988  0.04701 *  
## wifi_2       9.044e-05 8.826e-05   1.025  0.30571    
## wifi_3       1.203e-05 8.139e-05   0.148  0.88255    
## bluetooth_1  9.776e-05 3.215e-05   3.041  0.00241 ** 
## bluetooth_2 -2.622e-05 3.379e-05  -0.776  0.43789    
## bluetooth_3 -1.671e-05 3.210e-05  -0.521  0.60274    
## communication_1 5.990e-07 3.401e-06   0.176  0.86022    
## communication_2 2.781e-05 3.402e-06   8.176 7.76e-16 ***
## communication_3 7.394e-06 3.404e-06   2.172  0.03007 *  
## friends_1      6.552e-03 6.036e-03   1.085  0.27794    
## friends_2      -7.505e-03 7.546e-03  -0.994  0.32019    
## friends_3      3.272e-03 6.007e-03   0.545  0.58610    
## activity_1     2.698e-03 1.468e-03   1.839  0.06623 .  
## activity_2     5.907e-04 1.816e-03   0.325  0.74503    
## activity_3     -1.120e-03 1.468e-03  -0.763  0.44557    
## ---    
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.722 on 1142 degrees of freedom
## Multiple R-squared:  0.09007,    Adjusted R-squared:  0.07812 
## F-statistic: 7.537 on 15 and 1142 DF,  p-value: 3.262e-16

```

Salen ciertas columnas como significativas y la F también es significativa (< 0.05), así que existe relación entre la información que hemos recogido de los sensores y las estadísticas y tener síntomas en un momento dado.

Las ubicaciones calculadas por wifi se muestran relevantes, aunque quizás mejorando el clustering pudieran serlo más.

Buscamos más información mediante tests sobre los residuos del modelo.

```
shapiro.test(linear.model.001$residuals)
```

```

## 
## Shapiro-Wilk normality test
## 
## data: linear.model.001$residuals
## W = 0.57156, p-value < 2.2e-16

```

```
bptest(linear.model.001)
```

```

## 
## studentized Breusch-Pagan test
## 
## data: linear.model.001
## BP = 22.696, df = 15, p-value = 0.09082

```

Como salen menores que α ambos, nos indican que los residuos no son normales ni homocésticos, lo que unido al R^2 tan bajo nos indica que las relaciones no se modelan bien con un modelo lineal. Era lo esperado teniendo en cuenta las distribuciones de las variables, pero la idea era solo ver que había cierta influencia de las variables predictoras.

Probamos un modelo lineal con los datos en logaritmos, que es algo que reduce el sesgo lateral.

```
data.log <- log(data + 1)

linear.model.002 <- lm(symptom ~ ., data.log)
summary(linear.model.002)

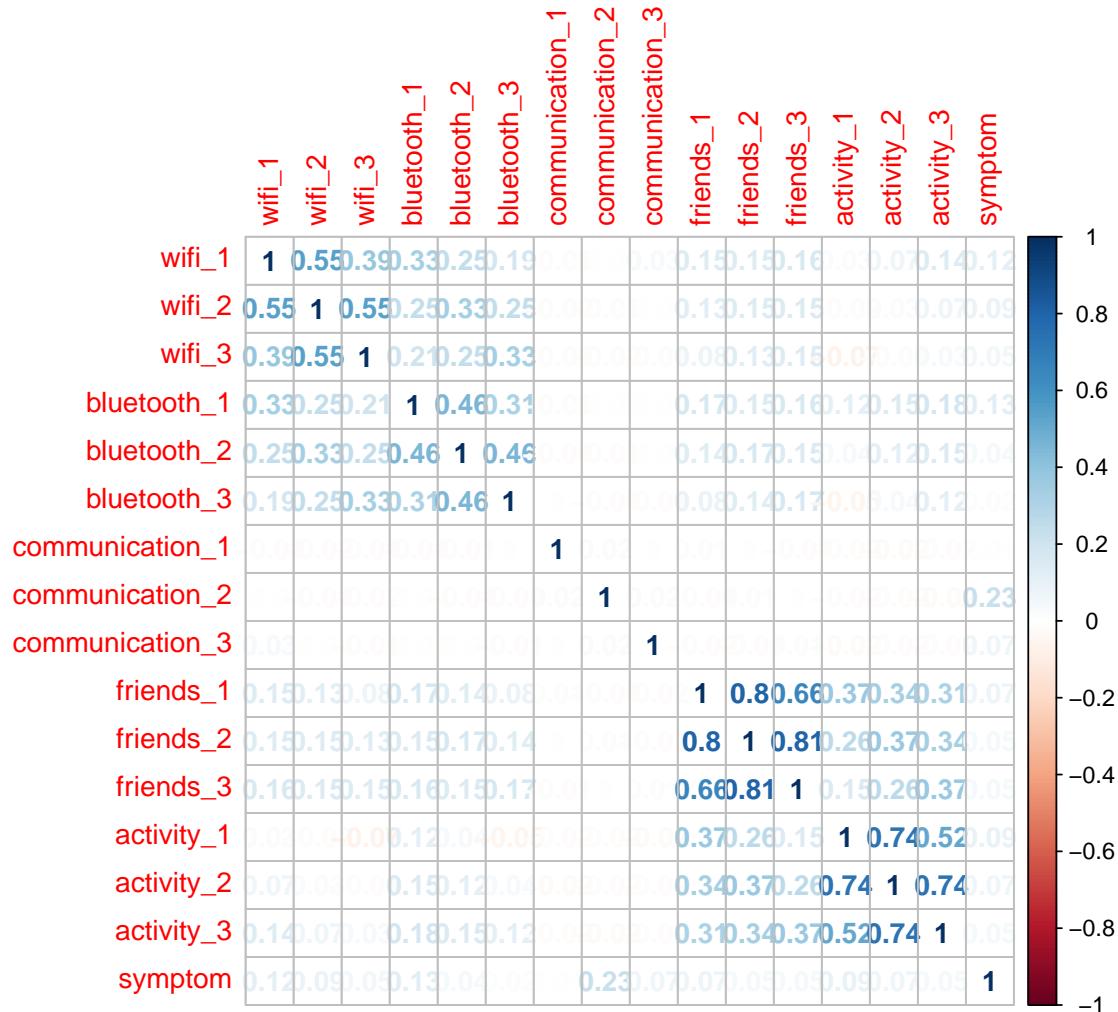
## 
## Call:
## lm(formula = symptom ~ ., data = data.log)
## 
## Residuals:
##      Min       1Q   Median       3Q      Max 
## -0.70914 -0.30993 -0.19107 -0.04778  2.56529 
## 
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)    
## (Intercept) 0.0657772  0.0348474  1.888  0.05934 .
## wifi_1       0.0136986  0.0088732  1.544  0.12291  
## wifi_2       0.0018845  0.0096795  0.195  0.84567  
## wifi_3       0.0005925  0.0093530  0.063  0.94950  
## bluetooth_1  0.0210066  0.0082068  2.560  0.01061 *  
## bluetooth_2 -0.0068128  0.0095792 -0.711  0.47710  
## bluetooth_3  0.0020215  0.0085563  0.236  0.81327  
## communication_1 0.0075267  0.0093118  0.808  0.41909  
## communication_2 0.0039648  0.0095842  0.414  0.67918  
## communication_3 0.0064445  0.0092491  0.697  0.48609  
## friends_1     0.0157236  0.0219471  0.716  0.47387  
## friends_2     0.0118568  0.0268839  0.441  0.65927  
## friends_3     -0.0111962  0.0218068 -0.513  0.60775  
## activity_1    0.0534747  0.0182024  2.938  0.00337 ** 
## activity_2   -0.0020049  0.0234665 -0.085  0.93193  
## activity_3   -0.0178839  0.0181965 -0.983  0.32591  
## --- 
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## 
## Residual standard error: 0.5651 on 1142 degrees of freedom
## Multiple R-squared:  0.0642, Adjusted R-squared:  0.05191 
## F-statistic: 5.223 on 15 and 1142 DF, p-value: 3.425e-10
```

Se ve que no mejora el R^2 y todavía estamos en un nivel de explicación del fenómeno parecido al azar. Algo interesante que se empieza a ver es que las columnas más significativas cambian, centrándose en la proximidad por bluetooth e ir a las mismas actividades.

Y parece algo de sentido común, ya que el bluetooth es un sensor que mucha cercanía y estar en una misma actividad también implica cercanía.

11.2.0.1. Análisis de correlación y componentes principales Mostramos la correlación entre variables como parte de la exploración de los datos.

```
data.cor <- cor(data)
corrplot(data.cor, method = "number")
```



Existen columnas con alta correlación, que son las que provienen de los datos expresados en meses y no en semanas como estamos considerando. En estos casos, como hubo que duplicar datos, que haya correlación es lo esperable. Por el momento no me planteo eliminarlas pues pueden aportar algo de información.

Usamos los test de esfericidad de Bartlett y KMO para ver si podemos aplicar análisis factorial a los datos.

```
cortest.bartlett(data.cor, n = 100)
```

```
## $chisq
## [1] 534.9471
##
## $p.value
## [1] 1.030161e-53
```

```

## 
## $df
## [1] 120

KMO(data.cor)

## Kaiser-Meyer-Olkin factor adequacy
## Call: KMO(r = data.cor)
## Overall MSA =  0.71
## MSA for each item =
##      wifi_1        wifi_2        wifi_3      bluetooth_1      bluetooth_2
##      0.77        0.72        0.76        0.79        0.75
##      bluetooth_3 communication_1 communication_2 communication_3 friends_1
##      0.77        0.45        0.48        0.42        0.75
##      friends_2     friends_3    activity_1    activity_2    activity_3
##      0.67        0.74        0.69        0.63        0.72
##      symptom
##      0.61

```

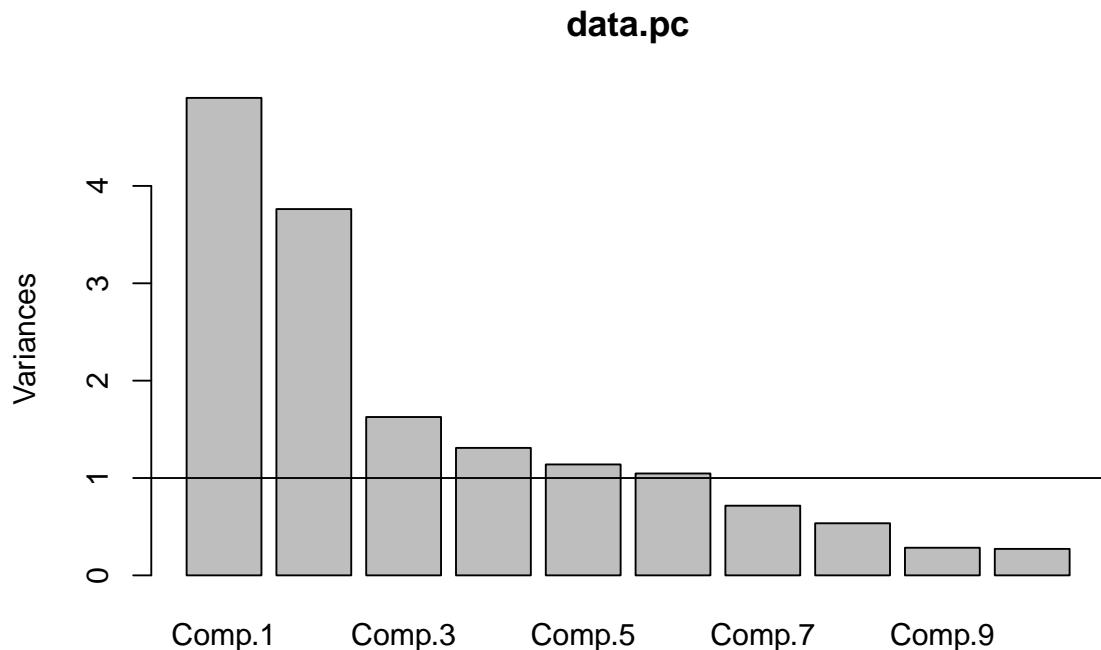
El nivel obtenido en general en los tests es “mediano” pero se puede aplicar el análisis factorial para recabar más información.

Se hace un análisis de componentes principales para ver como se distribuye la varianza.

```

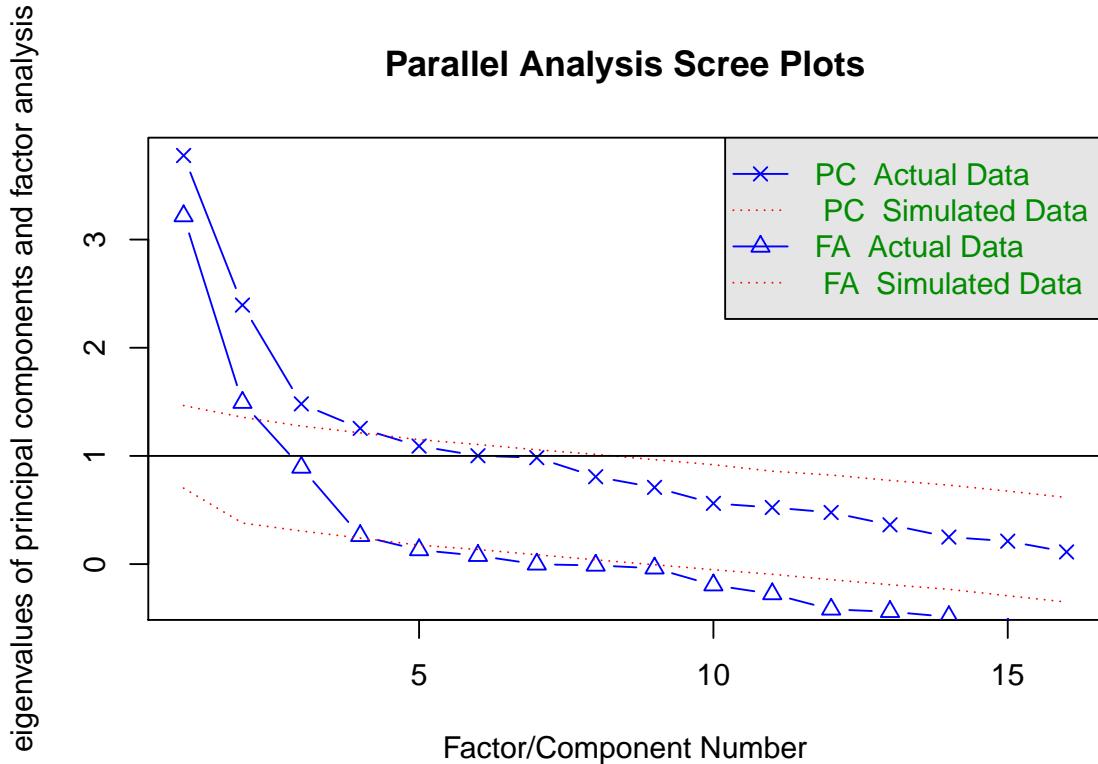
data.pc <- princomp(data.cor, cor = TRUE)
plot(data.pc)
abline(h = 1)

```



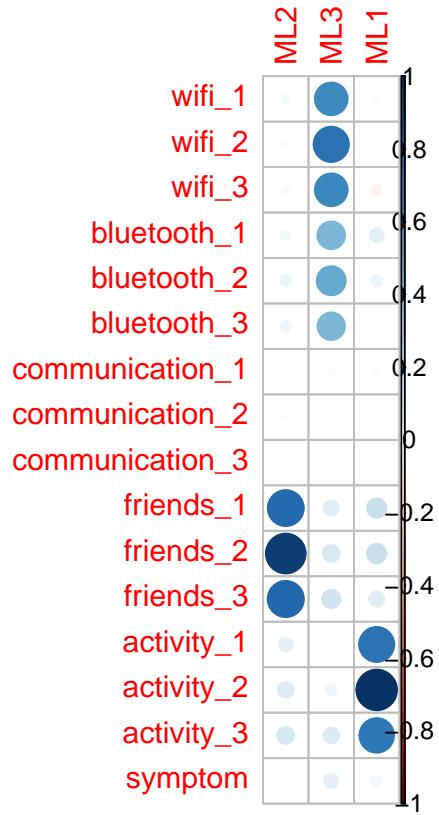
Parece que la mayor cantidad de información está en dos componentes.

```
fa.model.002 <-
  fa.parallel(data.cor, n.obs = length(data.cor), fm = "ml")
```



```
## Parallel analysis suggests that the number of factors = 3 and the number of components = 3
```

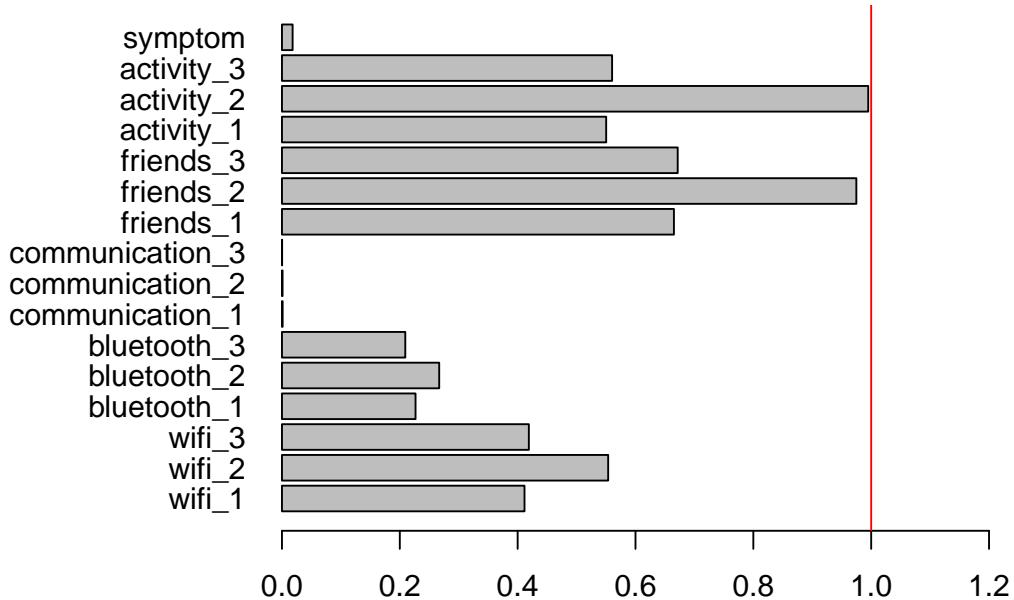
```
fa.model.003 <-
  fa(
    data.cor,
    n.obs = length(data.cor),
    fm = "ml",
    nfactors = 3,
    rotate = "varimax",
    scores = T
  )
corrplot(fa.model.003$loadings)
```



Según esta distribución espectral de la información, hay tres grupos principales de varianza: los sensores (wifi y bluetooth), la encuesta de amistad y la encuesta de actividades comunes. Según parece, los datos de llamadas y SMS no van a tener mucha influencia en la transmisión del síntoma.

```
par(mar = c(4, 9, 4, 4))
barplot(
  fa.model.003$communalities,
  main = "Comunalidades de las variables",
  xlim = c(0, 1.2),
  horiz = TRUE,
  las = 1
)
abline(v = 1, col = 'red')
```

Comunalidades de las variables



Las variables de amistad, wifis y bluetooth están bastante relacionadas, pero sin embargo la variable de comunicación parece no estarlo en absoluto. Parece ser que las matrices de actividad, amistad, wifi y bluetooth realmente llevan mucha información común. Podríamos reducirlas de cara a modelizar mejor, pero por el momento me interesa ver como se comportan separadas.

11.2.1. Análisis con modelo lineal generalizado

Como en el modelo lineal se podía ver que los residuos no eran de distribución normal y tampoco homocedásticos, parece interesante usar un modelo generalizado, ya que es capaz de trabajar con estas condiciones.

Usamos varias familias y funciones link para determinar si es un problema de regresión u otra cosa.

```

formula <- symptom ~ .
g1 <-
  glm(formula, data = data, family = poisson(link = "log"))
g2 <- glm(formula, data = data, family = gaussian())
g3 <-
  glm(log(symptom + 1) ~ ., data = data, family = gaussian())

formula <-
  symptom + 1 ~ wifi_1 + wifi_2 + wifi_3 + bluetooth_1 + bluetooth_2 +
  bluetooth_3 + communication_1 + communication_2 + communication_3 +
  friends_1 + friends_2 + friends_3 + activity_1 + activity_2 + activity_3

g4 <-
  mgcv::gam(formula, data = data, family = gaussian(link = "identity"))
g5 <-
  mgcv::gam(formula, family = gaussian(link = "inverse"), data = data)
g6 <-

```

```

mgcv:::gam(formula, family = gaussian(link = "log"), data = data)

g7 <-
  mgcv:::gam(formula, family = Gamma(link = "identity"), data = data)
g8 <-
  mgcv:::gam(formula, family = Gamma(link = "inverse"), data = data)
g9 <-
  mgcv:::gam(formula, family = Gamma(link = "log"), data = data)

formula2 <-
  log(symptom + 1) + 1 ~ wifi_1 + wifi_2 + wifi_3 + bluetooth_1 + bluetooth_2 +
  bluetooth_3 + communication_1 + communication_2 + communication_3 +
  friends_1 + friends_2 + friends_3 + activity_1 + activity_2 + activity_3

g10 <-
  mgcv:::gam(formula2,
              data = data,
              family = gaussian(link = "identity"))

g11 <-
  mgcv:::gam(formula2, family = gaussian(link = "inverse"), data = data)
g12 <-
  mgcv:::gam(formula2, family = gaussian(link = "log"), data = data)

g13 <-
  mgcv:::gam(formula2,
              family = Gamma(link = "identity"),
              data = data)

g14 <-
  mgcv:::gam(formula2, family = Gamma(link = "inverse"), data = data)
g15 <-
  mgcv:::gam(formula2, family = Gamma(link = "log"), data = data)

formula3 <-
  log(symptom + 1) + 1 ~ bluetooth_1 + bluetooth_2 +
  bluetooth_3 + communication_1 + communication_2 + communication_3 +
  activity_1 + activity_2 + activity_3

g16 <-
  mgcv:::gam(formula3,
              data = data,
              family = gaussian(link = "identity"))

g17 <-
  mgcv:::gam(formula3, family = gaussian(link = "inverse"), data = data)
g18 <-
  mgcv:::gam(formula3, family = gaussian(link = "log"), data = data)

g19 <-
  mgcv:::gam(formula3,
              family = Gamma(link = "identity"),
              data = data)

g20 <-
  mgcv:::gam(formula3, family = Gamma(link = "inverse"), data = data)
g21 <

```

```

mgcv:::gam(formula3, family = Gamma(link = "log"), data = data)

AIC(
  g1,g2,g3,g4,g5,g6,g7,g8,g9,g10,g11,g12,g13,g14,g15,
  g16,g17,g18,g19,g20,g21
)

```

	df	AIC
g1	16	3400.187
g2	17	4562.457
g3	17	1981.180
g4	17	4562.457
g5	17	4518.536
g6	17	4550.291
g7	17	3105.759
g8	17	3115.750
g9	17	3109.454
g10	17	1981.180
g11	17	1977.287
g12	17	1979.066
g13	17	1365.338
g14	17	1368.003
g15	17	1366.603
g16	11	1986.625
g17	11	1987.327
g18	11	1986.130
g19	11	1378.989
g20	11	1383.718
g21	11	1380.843

Los resultados del modelo general con diferentes familias y funciones link son atroces. Esta situación ya ha ocurrido alguna vez durante las prácticas y probablemente indica que el problema que estamos tratando no es una regresión, sino una clasificación.

11.2.2. Modelado del problema como clasificación

Los resultados que aquí aparecen en el código son la fase final del calibrado de parámetros y de la prueba con varios modos de preprocessamiento con caret (escalado, centrado, boxcox). Por brevedad solo se han incluido los que dan el resultado definitivo.

Lo primero que vamos a hacer es discretizar la variable de salida. Vamos a considerar dos clases: Cuando no hay sintomas y cuando los hay.

```

dataC <- data

discretize <- function(val) {
  if (val == 0)
    "NO"
  else
    "SI"
}

dataC$symptom = sapply(dataC$symptom, discretize)

```

```

dataC$symptom = as.factor(dataC$symptom)

dataC <- dataC[!duplicated(dataC), ]

```

Se puede ver que la variable dependiente es un factor, así que los algoritmos interpretarán el problema como una clasificación.

Miramos la distribución de las clases:

```
table(dataC$symptom)
```

```

## 
## NO SI
## 922 233

```

Las clases están muy desbalanceadas, lo que puede dar lugar a que los algoritmos no generen buenos modelos.

Vamos a balancear las clases de una forma simple, intentando igualar el número de elementos de la clase NO con la suma de todas las demás clases.

```

dataC <- downSample(dataC, dataC$symptom)
dataC$Class = NULL
table(dataC$symptom)

```

```

## 
## NO SI
## 233 233

```

Para modelar, vamos a probar algoritmos de minería que han dado buenos resultados en trabajos anteriores, para empezar, un K-vecinos que es el menos preciso pero muy sencillo y nos da una idea del rendimiento que se puede obtener.

11.2.3. Generación de modelos de minería para clasificar

```

set.seed(7)

cl <- makePSOCKcluster(10)
registerDoParallel(cl)

particion <-
  createDataPartition(dataC$symptom, p = 0.7, list = FALSE)

entrenamiento <- dataC[particion, ]
validacion <- dataC[-particion, ]
entrenamiento$Class <- NULL
validacion$Class <- NULL

fiveStats = function(...)
  c(twoClassSummary(...), defaultSummary(...))
control <- trainControl(
  method = "repeatedcv",
  number = 10,
  repeats = 5,
  classProbs = TRUE,

```

```

summaryFunction = fiveStats,
returnResamp = "final",
allowParallel = TRUE
)
metrica <- "ROC"

set.seed(7)

grid_knn <- expand.grid(.k = c(3, 5, 10, 20, 30, 50, 100))

fit.knn <-
train(
  symptom ~ .,
  data = entrenamiento,
  method = "knn",
  metric = metrica,
  tuneGrid = grid_knn,
  trControl = control
)
saveRDS(fit.knn, file = "models/knn_1.rda")

fit.knn <- readRDS("models/knn_1.rda")
predic_knn <- predict(fit.knn , newdata = validacion)
confusionMatrix(predic_knn, validacion$symptom)

## Confusion Matrix and Statistics
##
##          Reference
## Prediction NO SI
##          NO 42 33
##          SI 27 36
##
##          Accuracy : 0.5652
##          95% CI : (0.4782, 0.6493)
##          No Information Rate : 0.5
##          P-Value [Acc > NIR] : 0.0738
##
##          Kappa : 0.1304
##
##  Mcnemar's Test P-Value : 0.5186
##
##          Sensitivity : 0.6087
##          Specificity : 0.5217
##          Pos Pred Value : 0.5600
##          Neg Pred Value : 0.5714
##          Prevalence : 0.5000
##          Detection Rate : 0.3043
##          Detection Prevalence : 0.5435
##          Balanced Accuracy : 0.5652
##
##          'Positive' Class : NO
##

```

```

fit.knn

## k-Nearest Neighbors
##
## 328 samples
## 15 predictor
## 2 classes: 'NO', 'SI'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 5 times)
## Summary of sample sizes: 294, 294, 296, 295, 296, 295, ...
## Resampling results across tuning parameters:
##
##   k    ROC      Sens     Spec     Accuracy   Kappa
##   3   0.6113392 0.5968382 0.5655882 0.5812266 0.1623370
##   5   0.6163396 0.6177206 0.5455147 0.5813380 0.1630285
##  10   0.6311050 0.6031618 0.5892647 0.5958679 0.1923459
##  20   0.6426615 0.6295588 0.5818382 0.6056027 0.2112218
##  30   0.6439150 0.6550735 0.5072059 0.5812556 0.1621655
##  50   0.6403568 0.7368382 0.4501471 0.5935907 0.1868833
## 100   0.6201153 0.8247794 0.3129412 0.5690798 0.1373529
##
## ROC was used to select the optimal model using the largest value.
## The final value used for the model was k = 30.

```

Se ha obtenido un 0.64 de ROC. No parece que este dataset vaya a dar grandes rendimientos.
Ahora probamos un XGBoost

```

set.seed(7)

tune_grid <- expand.grid(
  nrounds = c(700, 800, 900),
  eta = c(0.1, 0.15),
  max_depth = c(6),
  gamma = 0,
  colsample_bytree = 1,
  min_child_weight = 1,
  subsample = 1
)

fit.xgbTree <-
  train(
    x = (entrenamiento %>% dplyr::select(-symptom)),
    y = entrenamiento$symptom,
    trControl = control,
    tuneGrid = tune_grid,
    method = "xgbTree",
    metric = metrica
  )

saveRDS(fit.xgbTree, file = "models/xgbTree_1.rda")

```

```

fit.xgbTree <- readRDS("models/xgbTree_1.rda")
predic_xgbTree <-
  predict(fit.xgbTree , newdata = (validacion %>% dplyr::select(-symptom)))
confusionMatrix(predic_xgbTree, validacion$symptom)

## Confusion Matrix and Statistics
##
##             Reference
## Prediction NO SI
##           NO 48 10
##           SI 21 59
##
##           Accuracy : 0.7754
##             95% CI : (0.6966, 0.842)
##   No Information Rate : 0.5
## P-Value [Acc > NIR] : 2.744e-11
##
##           Kappa : 0.5507
##
## Mcnemar's Test P-Value : 0.07249
##
##           Sensitivity : 0.6957
##           Specificity : 0.8551
##           Pos Pred Value : 0.8276
##           Neg Pred Value : 0.7375
##           Prevalence : 0.5000
##           Detection Rate : 0.3478
##           Detection Prevalence : 0.4203
##           Balanced Accuracy : 0.7754
##
##           'Positive' Class : NO
##

xgbTreeImp <- varImp(fit.xgbTree)
print(xgbTreeImp)

## xgbTree variable importance
##
##             Overall
## activity_1      100.000
## bluetooth_1     52.104
## friends_1       50.557
## friends_2       47.380
## activity_3       42.161
## friends_3       37.481
## wifi_1          32.682
## bluetooth_3     30.198
## bluetooth_2     27.063
## activity_2       22.162
## communication_2    9.441
## communication_1    3.570
## wifi_3          3.288
## wifi_2          3.260
## communication_3    0.000

```

```

fit.xgbTree

## eXtreme Gradient Boosting
##
## 328 samples
## 15 predictor
## 2 classes: 'NO', 'SI'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 5 times)
## Summary of sample sizes: 296, 296, 295, 295, 294, 295, ...
## Resampling results across tuning parameters:
##
##   eta    nrounds   ROC      Sens      Spec      Accuracy     Kappa
##   0.10    700     0.7635654  0.6881618  0.6819118  0.6848975  0.3698151
##   0.10    800     0.7627449  0.6857353  0.6794118  0.6824365  0.3649149
##   0.10    900     0.7627871  0.6919118  0.6818382  0.6867012  0.3734661
##   0.15    700     0.7615620  0.6797794  0.6944853  0.6865876  0.3737697
##   0.15    800     0.7610429  0.6786029  0.6956618  0.6866076  0.3738048
##   0.15    900     0.7619529  0.6761029  0.6967647  0.6860016  0.3724785
##
## Tuning parameter 'max_depth' was held constant at a value of 6
## Tuning
##
## Tuning parameter 'min_child_weight' was held constant at a value of 1
##
## Tuning parameter 'subsample' was held constant at a value of 1
## ROC was used to select the optimal model using the largest value.
## The final values used for the model were nrounds = 700, max_depth = 6, eta
## = 0.1, gamma = 0, colsample_bytree = 1, min_child_weight = 1 and subsample = 1.

```

Se ha obtenido un resultado muy interesante calibrando XGBoost. Es probable que sea el máximo que se pueda obtener de estos datos.

A continuación un ADABoost, ya que los árboles funcionan bien.

```

set.seed(7)

ada_grid <- expand.grid(
  mfinal = c(11, 12, 13, 14, 15),
  maxdepth = c(6, 7),
  coeflearn = c("Breiman")

)

fit.ada <- train(
  symptom ~.,
  data = entrenamiento,
  method = "AdaBoost.M1",
  metric = metrica,
  trControl = control,
  tuneGrid = ada_grid,
  verbose = FALSE
)
saveRDS(fit.ada, file = "models/ada_1.rda")

```

```

set.seed(7)

fit.ada <- readRDS("models/ada_1.rda")
predic_ada <-
  predict(fit.ada , newdata = (validacion %>% dplyr::select(-symptom)))
confusionMatrix(predic_ada, validacion$symptom)

## Confusion Matrix and Statistics
##
##           Reference
## Prediction NO SI
##       NO 50  4
##       SI 19  65
##
##           Accuracy : 0.8333
##             95% CI : (0.7605, 0.8913)
##   No Information Rate : 0.5
## P-Value [Acc > NIR] : 3.265e-16
##
##           Kappa : 0.6667
##
## Mcnemar's Test P-Value : 0.003509
##
##           Sensitivity : 0.7246
##           Specificity  : 0.9420
##           Pos Pred Value : 0.9259
##           Neg Pred Value : 0.7738
##           Prevalence    : 0.5000
##           Detection Rate : 0.3623
## Detection Prevalence : 0.3913
##           Balanced Accuracy : 0.8333
##
##           'Positive' Class : NO
##          

fit.ada

## AdaBoost.M1
##
## 374 samples
## 15 predictor
## 2 classes: 'NO', 'SI'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 5 times)
## Summary of sample sizes: 336, 337, 336, 337, 336, 338, ...
## Resampling results across tuning parameters:
##
##   maxdepth  mfinal   ROC      Sens      Spec      Accuracy     Kappa
##   6          11       0.7055472 0.6423977 0.6314035 0.6373107 0.2739161
##   6          12       0.7053256 0.6438596 0.6411696 0.6428481 0.2850893
##   6          13       0.7101038 0.6414620 0.6333333 0.6377944 0.2749542
##   6          14       0.7122296 0.6461404 0.6417544 0.6442674 0.2880663
##   6          15       0.7114192 0.6427485 0.6439766 0.6437135 0.2869173

```

```

##    7      11  0.7005509  0.6266082  0.6326901  0.6299139  0.2593987
##    7      12  0.7032664  0.6201754  0.6379532  0.6292982  0.2582009
##    7      13  0.7022439  0.6297076  0.6305263  0.6304512  0.2603784
##    7      14  0.7059521  0.6274854  0.6467836  0.6374530  0.2743920
##    7      15  0.7062134  0.6285380  0.6501170  0.6395440  0.2786989
##
## Tuning parameter 'coeflearn' was held constant at a value of Breiman
## ROC was used to select the optimal model using the largest value.
## The final values used for the model were mfinal = 14, maxdepth = 6
## and coeflearn = Breiman.

```

Adaboost tiene un rendimiento un poco peor que XGBoost como máxima, aunque quizás tenga mejor media o rango. Sin embargo es preocupante que aparezcan tantos falsos positivos y falsos negativos.

La siguiente prueba será una SVM lineal.

```

set.seed(7)

svm_grid <- expand.grid(C = seq(0.001, 0.02, length = 10))

fit.svm <- train(
  symptom ~ .,
  data = entrenamiento,
  method = "svmLinear",
  trControl = control,
  preProc = c("center", "scale"),
  metric = metric,
  tuneGrid = svm_grid
)
saveRDS(fit.svm, file = "models/svm_1.rda")

fit.svm <- readRDS("models/svm_1.rda")
predic_svm <-
  predict(fit.svm, newdata = (validacion %>% dplyr::select(-symptom)))
confusionMatrix(predic_svm, validacion$symptom)

## Confusion Matrix and Statistics
##
##             Reference
## Prediction NO SI
##       NO 48 31
##       SI 21 38
##
##           Accuracy : 0.6232
##             95% CI : (0.5368, 0.7042)
##   No Information Rate : 0.5
##   P-Value [Acc > NIR] : 0.002393
##
##           Kappa : 0.2464
##
##   Mcnemar's Test P-Value : 0.212003
##
##           Sensitivity : 0.6957
##           Specificity : 0.5507

```

```

##          Pos Pred Value : 0.6076
##          Neg Pred Value : 0.6441
##          Prevalence : 0.5000
##          Detection Rate : 0.3478
## Detection Prevalence : 0.5725
##          Balanced Accuracy : 0.6232
##
##          'Positive' Class : NO
##         

fit.svm

## Support Vector Machines with Linear Kernel
##
## 374 samples
## 15 predictor
## 2 classes: 'NO', 'SI'
##
## Pre-processing: centered (15), scaled (15)
## Resampling: Cross-Validated (10 fold, repeated 5 times)
## Summary of sample sizes: 337, 336, 336, 336, 336, 337, ...
## Resampling results across tuning parameters:
##
##          C      ROC      Sens      Spec      Accuracy     Kappa
## 0.001000000 0.6417316 0.8250292 0.3837427 0.6048538 0.2091330
## 0.003111111 0.6451903 0.6865497 0.5371345 0.6118698 0.2237427
## 0.005222222 0.6502054 0.6961404 0.5199415 0.6081429 0.2161623
## 0.007333333 0.6501264 0.7026316 0.5091813 0.6060661 0.2118886
## 0.009444444 0.6523600 0.7016959 0.5082456 0.6050727 0.2099639
## 0.011555556 0.6539056 0.7049123 0.4998246 0.6025004 0.2048025
## 0.013666667 0.6529773 0.7092398 0.5050877 0.6072357 0.2143368
## 0.015777778 0.6545633 0.7114035 0.5081871 0.6098965 0.2196378
## 0.017888889 0.6561080 0.7091813 0.5082456 0.6088446 0.2175109
## 0.020000000 0.6557628 0.7103509 0.5135088 0.6120452 0.2239693
##
## ROC was used to select the optimal model using the largest value.
## The final value used for the model was C = 0.01788889.

```

El rendimiento de este tipo de algoritmo parece menos apropiado para nuestros datos que los basados en árboles.

Probamos también una regresión logística.

```

set.seed(7)

fit.rl <-
  train(
    symptom ~.,
    data = entrenamiento,
    method = "LMT",
    metric = metric,
    preProc = c("center", "scale"),
    trControl = control
  )
saveRDS(fit.rl, file = "models/r1_1.rda")

```

```

predic_rl <- predict(fit.rl , newdata = validacion)
confusionMatrix(predic_rl, validacion$symptom)

## Confusion Matrix and Statistics
##
##          Reference
## Prediction NO SI
##           NO 50 36
##           SI 19 33
##
##           Accuracy : 0.6014
##                 95% CI : (0.5147, 0.6838)
## No Information Rate : 0.5
## P-Value [Acc > NIR] : 0.01060
##
##           Kappa : 0.2029
##
## Mcnemar's Test P-Value : 0.03097
##
##           Sensitivity : 0.7246
##           Specificity : 0.4783
## Pos Pred Value : 0.5814
## Neg Pred Value : 0.6346
## Prevalence : 0.5000
## Detection Rate : 0.3623
## Detection Prevalence : 0.6232
## Balanced Accuracy : 0.6014
##
## 'Positive' Class : NO
##

fit.rl

## Logistic Model Trees
##
## 328 samples
## 15 predictor
## 2 classes: 'NO', 'SI'
##
## Pre-processing: centered (15), scaled (15)
## Resampling: Cross-Validated (10 fold, repeated 5 times)
## Summary of sample sizes: 294, 295, 294, 296, 294, 296, ...
## Resampling results across tuning parameters:
##
##   iter  ROC      Sens      Spec      Accuracy     Kappa
##   1    0.6373932  0.5861765  0.6228676  0.6048117  0.2091175
##   21   0.6353014  0.6238235  0.5613235  0.5929022  0.1853187
##   41   0.6313431  0.6478676  0.5363971  0.5919118  0.1842493
##
## ROC was used to select the optimal model using the largest value.
## The final value used for the model was iter = 1.

```

El rendimiento es bajo, aunque la ejecución ha sido muy rápida comparada con el resto de algoritmos excepto k-vecinos.

Ahora se prueba una red neuronal.

```

set.seed(7)

mlp_grid <- expand.grid(size = 5:9)

fit.mlp <-
  train(
    symptom ~ .,
    data = entrenamiento,
    method = "mlp",
    metric = metric,
    preProc = c("center", "scale"),
    trControl = control,
    tuneGrid = mlp_grid
  )
  saveRDS(fit.mlp, file = "models/mlp_1.rda")

fit.mlp <- readRDS("models/mlp_1.rda")
predic_mlp <- predict(fit.mlp , newdata = validacion)
fit.mlp

## Multi-Layer Perceptron
##
## 374 samples
## 15 predictor
## 2 classes: 'NO', 'SI'
##
## Pre-processing: centered (15), scaled (15)
## Resampling: Cross-Validated (10 fold, repeated 5 times)
## Summary of sample sizes: 336, 337, 336, 336, 337, 338, ...
## Resampling results across tuning parameters:
##
##     size   ROC      Sens      Spec      Accuracy      Kappa
##     5       0.6503319  0.6185965  0.5921637  0.6052829  0.2106720
##     6       0.6575815  0.6352047  0.5709357  0.6026948  0.2060106
##     7       0.6406250  0.6072515  0.6009357  0.6039893  0.2082043
##     8       0.6501811  0.6240936  0.5996491  0.6118168  0.2237528
##     9       0.6448054  0.6256140  0.5783626  0.6015260  0.2038240
##
## ROC was used to select the optimal model using the largest value.
## The final value used for the model was size = 6.

confusionMatrix(predic_mlp, validacion$symptom)

## Confusion Matrix and Statistics
##
##             Reference
## Prediction NO SI
##           NO 44 22
##           SI 25 47
##
##           Accuracy : 0.6594
##             95% CI : (0.5739, 0.7379)
##   No Information Rate : 0.5

```

```

##      P-Value [Acc > NIR] : 0.0001127
##
##          Kappa : 0.3188
##
##  Mcnemar's Test P-Value : 0.7704931
##
##          Sensitivity : 0.6377
##          Specificity : 0.6812
##          Pos Pred Value : 0.6667
##          Neg Pred Value : 0.6528
##          Prevalence : 0.5000
##          Detection Rate : 0.3188
##          Detection Prevalence : 0.4783
##          Balanced Accuracy : 0.6594
##
##          'Positive' Class : NO
##

```

De nuevo, parece no adaptarse tan bien como los árboles de decisión.

Y finalmente un Random Forest. Este algoritmo es fácil de calibrar y suele dar buen resultado.

```

set.seed(123)

grid_rf <- expand.grid(mtry = c(2,3))

fit.rf <-
train(
  symptom ~ .,
  data = entrenamiento,
  method = "rf",
  metric = metric,
  tuneGrid = grid_rf,
  trControl = control
)
saveRDS(fit.rf, file = "models/rf_1.rda")

fit.rf <- readRDS("models/rf_1.rda")
predic_rf <- predict(fit.rf , newdata = validacion)
confusionMatrix(predic_rf, validacion$symptom)

## Confusion Matrix and Statistics
##
##          Reference
## Prediction NO SI
##          NO 48  1
##          SI 21  68
##
##          Accuracy : 0.8406
##          95% CI : (0.7686, 0.8973)
##          No Information Rate : 0.5
##          P-Value [Acc > NIR] : < 2.2e-16
##
##          Kappa : 0.6812
##
```

```

## McNemar's Test P-Value : 5.104e-05
##
##          Sensitivity : 0.6957
##          Specificity : 0.9855
##          Pos Pred Value : 0.9796
##          Neg Pred Value : 0.7640
##          Prevalence : 0.5000
##          Detection Rate : 0.3478
##          Detection Prevalence : 0.3551
##          Balanced Accuracy : 0.8406
##
##          'Positive' Class : NO
##         

print(varImp(fit.rf))

## rf variable importance
##
##          Overall
## activity_1      100.000
## friends_1       82.871
## activity_2       77.222
## bluetooth_1     76.142
## activity_3       75.550
## friends_2       63.502
## friends_3       59.561
## bluetooth_2     51.620
## bluetooth_3     51.587
## wifi_1           42.805
## wifi_3           23.603
## wifi_2           22.738
## communication_1   8.864
## communication_2   6.900
## communication_3   0.000

fit.rf

## Random Forest
##
## 374 samples
## 15 predictor
## 2 classes: 'NO', 'SI'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 5 times)
## Summary of sample sizes: 337, 336, 336, 338, 337, 337, ...
## Resampling results across tuning parameters:
##
##    mtry   ROC      Sens      Spec      Accuracy   Kappa
##    2      0.7531187 0.6531579 0.6812281 0.6669322 0.3341147
##    3      0.7519568 0.6477778 0.6885380 0.6678442 0.3359791
##
## ROC was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 2.

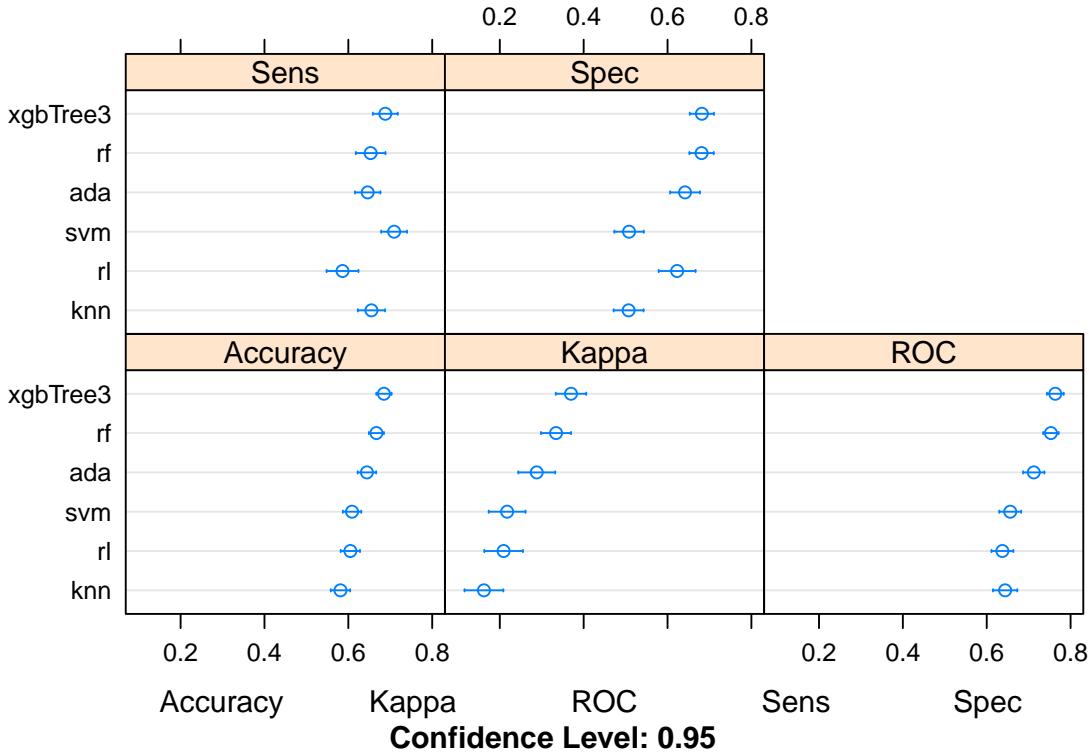
```

El random forest también nos da una predicción bastante buena sin casi complicaciones. Definitivamente los árboles de decisión dan buenos resultados en este problema.

11.2.4. Comparación de modelos

A continuación se muestra una comparativa de los modelos generados:

```
modelos <- list(
  xgbTree3 = readRDS("models/xgbTree_1.rda"),
  ada = readRDS("models/ada_1.rda"),
  rl = readRDS("models/rl_1.rda"),
  rf = readRDS("models/rf_1.rda"),
  svm = readRDS("models/svm_1.rda"),
  knn = readRDS("models/knn_1.rda")
)
resultados <- resamples(modelos)
dotplot(resultados)
```



12. Conclusiones

Como se puede ver en la comparativa la balanza se inclina en favor de los algoritmos basados en árboles de decisión, probablemente porque el espacio de los datos es bastante complicado con distribuciones muy sesgadas.

Este problema se debe seguramente a que los datos provienen de contadores y los contadores suelen tener una distribución de tipo poisson.

Los resultados de ROC, aunque están dentro de lo que se suele considerar “acceptable/fair”, es decir, entre 0.7 y 0.8 no son nada satisfactorios de cara a una predicción. En las matrices de confusión se

puede ver una abundancia de falsos positivos y falsos negativos que harían el modelo poco practicable tal cual está planteado.

Esto también puede deberse a que, tras eliminar duplicados y filas con ceros, el conjunto de datos ha quedado “pequeño” y eso no favorece la consecución de un modelo eficaz, sobre todo siendo datos con mala distribución.

Analizando un poco los resultados, y con idea de seguir planteando mejores modelos, la importancia de las variables en XGBoost nos habla de que el factor más determinante serían las actividades comunes en el campus y la proximidad directa medida por el teléfono. Además nos dice que las mejores variables son las de la semana anterior, lo que apunta a que el síntoma debe tener una incubación de 1 semana.

En el lado de las menos relevantes están las llamadas de teléfono y sms y parece razonable que una enfermedad no se pueda transmitir llamando por teléfono. Por otro lado parece que los sujetos no se comunican por teléfono con la misma gente con la que comparten actividades y proximidad.

La relevancia de las columnas de wifi es baja. Esto se puede deber a que el algoritmo de clusterizado que hemos usado para calcular las ubicaciones no describe bien las ubicaciones reales. Es algo que hay que mejorar.

xgbTreeImp

```
## xgbTree variable importance
##
##                               Overall
## activity_1           100.000
## bluetooth_1          52.104
## friends_1             50.557
## friends_2             47.380
## activity_3            42.161
## friends_3             37.481
## wifi_1                32.682
## bluetooth_3           30.198
## bluetooth_2           27.063
## activity_2            22.162
## communication_2       9.441
## communication_1       3.570
## wifi_3                3.288
## wifi_2                3.260
## communication_3       0.000
```

Desde el punto de vista de los objetivos del trabajo, creo que se han satisfecho pues se ha mostrado como adaptar los datos para construir modelos de regresión y de clasificación, como trabajar con grafos de relaciones e interacciones aplicados al problema y maneras razonables de visualizar estos grafos.

12.1. Ampliaciones y mejoras

Varios puntos se han quedado fuera del trabajo que podrían requerir atención en un futuro.

Por un lado, la técnica descrita para construir matrices de exposición desde datos de móviles podría aplicarse de manera más amplia con mejores datos lo que seguramente beneficiaría la calidad del modelo.

Por otro lado, sería necesario refinar la manera de contar las interacciones, ya que en el trabajo se ha dado por supuesto que todas cuentan lo mismo, pero resulta obvio que no es igual de representativo que haya 4 interacciones en un minuto que 4 a lo largo de 4 horas. Probablemente esta mala cualificación de las interacciones (medida de proximidad bluetooth, traza de wifi...) sea la causante de la extraña distribución de los datos y de que algunos sujetos tengan gran cantidad de lecturas, similares a outliers, que perjudican el cálculo de la exposición.

Otro factor a mejorar es la calidad de las encuestas. En algunos de los papers asociados a estos datos se han generado modelos dinámicos capaces de imputar los datos de manera precisa. La aproximación

a la imputación de datos en este trabajo es básica y se basa en asunciones demasiado elementales para dar buenos resultados.

De cualquier manera, salvando estas áreas de mejora, creo que la estrategia es utilizable incluso con otros datos del mismo dataset, como otros síntomas o las preferencias musicales, políticas y de salud. Desde el punto de vista metodológico no hubieran aportando nada pues simplemente consisten en cambiar la variable que se usa para construir la matriz de exposición y replicar código, pero quizás hubieran arrojado modelos de mejor calidad.

13. Tecnología y librerías utilizadas

Se ha utilizado R en el entorno R Studio por ser más adecuado a la generación de informes con RMarkdown. Las librerías de R utilizadas son las siguientes:

Package	Version
caret	6.0-86
CCA	1.2
corrplot	0.84
data.table	1.13.2
doParallel	1.0.15
dplyr	1.0.2
gdata	2.18.0
ggplot2	3.3.2
glue	1.4.2
kohonen	3.0.10
lemon	0.4.5
lmtest	0.9-38
lubridate	1.7.9
patchwork	1.0.1
psych	2.0.8
resample	0.4
scales	1.1.1
tibble	3.0.3
tidyverse	1.1.2
visdat	0.5.3
graphics	3.6.2
mgcv	1.8-31
nnet	7.3-12
rpart	4.1-15

14. Bibliografía

1. Manuales del Master en Data Science y Big Data aplicados a la Economía y a la Administración y Dirección de Empresas
2. Modeling the Co-evolution of Behaviors and Social Relationships Using Mobile Phone Data
 - Wen Dong¹, Bruno Lepri^{1,2} and Alex (Sandy) Pentland¹
 - MIT Media Laboratory
3. Graph-Coupled HMMs for Modeling the Spread of Infection
 - Wen Dong, Alex Pentland
 - MIT. Media Laboratory

4. Network analysis with R
 - <https://www.jessesadler.com/post/network-analysis-with-r/>
5. Static and dynamic network visualization with R
 - <https://kateto.net/network-visualization>
6. RSSI-based Localization Zoning using K-Mean Clustering
 - <https://iopscience.iop.org/article/10.1088/1757-899X/705/1/012038/pdf>
7. Introduction Self-Organizing Maps (SOM)
 - <https://rpubs.com/inayatus/som>