Trabajo Práctico 2 – Parte 1

Marcelo A. Soria – Mariana Landoni

Objetivo

El objetivo de este segundo trabajo práctico es aplicar varias de las metodologías y teorías que vimos sobre grafos, y además utilizar algunas de las técnicas de análisis geográfico que veremos en las próximas clases.

Datos

Vamos a trabajar con datos de la red social Brighkite, una de las que analizamos en clase, que están disponibles es: http://snap.stanford.edu/data/loc-brightkite.html

En esta página hay dos archivos de datos:

loc-brightkite_edges.txt.gz : datos de interacciones sociales. Es una lista de adjacencia donde linea contienen un par de nodos que indican amistad entre dos usuarios.

loc-brightkite_totalCheckins.txt.gz : datos de geolocalización. Cada registro de este archivo se corresponde a un check-in, y contiene el ID del usuario, la hora y la latitud y longitud del sitio donde se marcó el check-in.

Indicaciones generales para la realización del TP:

- El TP se realizará en grupos de **tres o cuatro** personas.
- **Los informes deberán contener un análisis de redes sociales, un análisis de comunidades y el desarrollo de un mapa**. Cada grupo podrá decidir los tipos específicos de análisis dentro de las tres tareas propuestas.
- La fecha de entrega es el 23 de diciembre a las 23:59.

Análisis de red social

El dataset de Brightkite se analizó en numerosos papers, además del que vimos en clase. Por ejemplo, aquí está publicado un análisis topológico de la red social de Brighkite.

Para el TP tienen que seleccionar algún subgrafo generado a partir de la localización de los check-ins. Esto, es hay que extraer un subgrafo inducido de todos los usuarios que hicieron algún check-in en un determinado país. Esto va a determinar que en el grafo habrá residentes y visitantes temporarios en ese país.

Una vez seleccionado el subgrafo deberán realizar un análisis topológico, calcular números de nodos, vértices, densidad, diámetro, coeficientes de clustering

(transitividad) locales y global, y su distribución, distribuciones de grados, calcular centralidades (varias), etc.

Algunos tips para procesar los datos de la red social

1. El país con más check-ins es Estados Unidos. Si algún grupo decide trabajar con datos de este país les recomendamos trabajar con un estado en particular, que tenga al menos 5,000 check-ins.
2. Para los grupos que seleccionen otro país, éste debe tener al menos 5,000 check-ins.
3. Los IDs de los nodos comienzan en cero, pero *igraph* no puede manejar nodos con estos IDs. Una solución es convertir el dato a tipo carácter, otra, posiblemente más sencilla es, sumar un uno a todos los IDs. Cualquiera sea la opción seleccionada, la operación de conversión de tipo o la de suma debe realizarse sobre los IDs de ambos datasets.
4. Los datos de red social no contienen información de locación. Ésta debe extraerse de los datos de locación de check-ins.
5. Los registros de check-ins incluyen una variable que es un *hash* de los datos de latitud y longitud, esto facilita la búsqueda de sitios únicos
6. Para los que trabajen con linux, pueden necesitar instalar estos dos paquetes: libgeos-dev y libgdal1-dev.
7. Hay check-ins con datos faltantes y otros con localización indeterminada.

Algunos tips de código

Descargar los datos, el argumento *"fill"* en el segundo *read.table()* es para que se puedan leer correctamente los registros con datos faltantes:

```
download.file("http://snap.stanford.edu/data/loc-
brightkite_edges.txt.gz", destfile = "brkred.gz")

red <- read.table(gzfile("brkred.gz"), header=F,
stringsAsFactors = F)

download.file("http://snap.stanford.edu/data/loc-
brightkite_totalCheckins.txt.gz", destfile = "brkchk.gz")

checkins <- read.table(file = gzfile("brkchk.gz"), header=F,
fill=T, stringsAsFactors = F)
```

Se pueden usar otros nombres para las variables, pero con estos les resultará más sencillo seguir los ejemplos que siguen:

```
names(checkins) <- c("id", "time", "lat", "lon", "loc_id")
```

igraph no acepta identificadores numéricos para los vértices que tomen valor 0, le sumamos un 1 a todos los IDs:

```
checkins$id <- checkins$id + 1
red$V1 <- red$V1 + 1
red$V2 <- red$V2 + 1
```

Los check-ins con datos faltantes se eliminan; los que tienen geo-localización indeterminada, no hace falta, porque no corresponden a ningún país.

```
checkins_c <- checkins[complete.cases(checkins),]
head(sort(table(checkins_c$loc_id), decreasing=T), n=14)
head(checkins_c[,3:5][checkins_c$loc_id ==
"00000000000000000000000000000000",])
```

El dataframe de check-ins tiene muchos registros, muchos de ellos repetidos, para reducir los tiempos de búsqueda, es conveniente construir un dataframe con geolocalizaciones únicas.

```
uniq_checks <- unique(checkins_c[,3:5])
```

Para determinar el país donde se hizo un check-in hacen falta tres paquetes diferentes de R y la función que sigue:

```
install.packages("sp")
install.packages("rworldmap")
install.packages("rworldxtra")

library(sp)
library(rworldmap)
library(rworldxtra)

coords_pais = function(points){
  # Una función para recuperar el código ISO 3
  # de pais de una locación
  countriesSP <- getMap(resolution='high')
  # convertir las coordenadas en puntos espaciales
  pointsSP = SpatialPoints(points,
proj4string=CRS(proj4string(countriesSP)))
  # recuperar el poligono-pais de cada punto espacial
  indices = over(pointsSP, countriesSP)
```

```
  # recuperar el nombre de cada pais por código ISO
  # de tres letras
  indices$ISO3
}

pais_iso <- coords_pais(uniq_checks[, 2:1])
head(sort(table(pais_iso), decreasing=T), 12)
table(pais_iso)["ARG"]
```

Agregar la información de país a los datos de check-ins únicos:

```
uniq_checks <- data.frame(uniq_checks, ISO3 =
as.character(pais_iso))

head(uniq_checks)
```

Ejemplo: construir un subgrafo inducido de los usuarios que hicieron check-ins en la Federación Rusa. Primero hay que extraer los usuarios con al menos un check-in en Rusia, luego hay que preparar una lista única de usuarios:

```
# install.packages("dplyr")

library(dplyr)

uniq_checks_rusia <- filter(uniq_checks, ISO3 == "RUS")

checkins_rusia <- inner_join(checkins_c,
uniq_checks_rusia[,3:4], by = "loc_id")

uniq_usuarios_rusia <- unique(checkins_rusia$id)

install.packages("igraph")

library(igraph)

red.amistad <- graph.edgelist(as.matrix(red), directed = F)

summary(red.amistad)

red_rusia <- induced_subgraph(red.amistad, vids =
uniq_usuarios_rusia)

summary(red_rusia)
```

Análisis de comunidades

Para los análisis de comunidades seleccionar uno o dos métodos de búsqueda de comunidades y validar los resultados. Para este fin se pueden usar los métodos de detección de comunidades que incluye *igraph*. Algunos métodos son

computacionalmente demandantes y puede ser que no corran bien si tienen un dataset de usuarios únicos grande. En general, *walktrap.community()* funciona bien aún para grafos muy grandes. Les recomendamos empezar por con este método.

Es suficiente mostrar el resultado de dos métodos e interpretar porqué dan resultados parecidos, o diferentes.

También se puede analizar dentro de un país como se relacionan los usuarios teniendo en cuenta si son residentes, visitantes frecuentes o esporádicos, si viven en zonas rurales o ciudades, etc.

Algunos tips de código

```
walktrap_rusia <- walktrap.community(red_rusia, steps = 6)
walktrap_rusia
names(walktrap_rusia)
walktrap_rusia$modularity
walktrap_rusia$membership
```

La mayoría de los clusters tienen un solo miembro porque el grafo no es conectado y hay muchos nodos de grado cero:

```
sort(table(walktrap_rusia$membership), decreasing=T)
is.connected(red_rusia)
```

Lo podemos ver también con un gráfico:

```
color_grado <- ifelse(degree(red_rusia) > 0, "red", "blue")
plot(red_rusia, vertex.label = NA,
     vertex.size = 8, vertex.color = color_grado)
```

Las comunidades más grandes se encuentran dentro del componente mayor:

```
color_grado[walktrap_rusia$membership == 4] <- "green"
color_grado[walktrap_rusia$membership == 2] <- "orange"
plot(red_rusia, vertex.label = NA,
     vertex.size = 8, vertex.color = color_grado)
```

En este caso convendría buscar primero el mayor componente, extraer el subgrafo correspondiente y buscar comunidades dentro de ese componente.

Construcción de mapas

Esta sección se detalla en la parte 2 de esta guía.