Tuto Package NetworkD3

Claire MAZZUCATO & Adrien JUPITER

21/12/2020

Figure 1: PSB website.

NetworkD3

Qu'est-ce que c'est ?

Le package `NetworkD3` permet de créer un réseau JavaScript D3, un arbre, un dendrogramme et des graphes Sankey à partir de R.

Il prend actuellement en charge les types de graphiques de réseau suivants :

Force directed networks avec `simpleNetwork` et `forceNetwork`

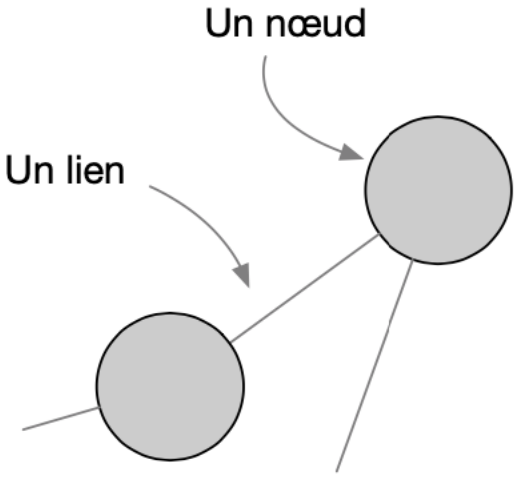
Les diagrammes Sankey avec `sankeyNetwork`

Le réseau Radial avec `radialNetwork`

Pourquoi utiliser les réseaux ?

La notion de réseaux commence avec le célébre casse-tête de Königsberg posé par Euler en 1735. Il a prouvé qu'il était impossible de parcourir les sept ponts de cette ville de manière à ne les traverser qu'une seule fois.

Il inventa le concept de graphe afin de représenter plus simplement la situation géographique de la ville. Chaque parcelle de terre est alors modélisé par un noeud et chaque lien représente un pont. Il le modélisa de la manière suivante:



Pourquoi utiliser R pour l'analyse de réseaux ?

Divers outils sont disponibles pour l'analyse des réseaux. Certains d'entre eux sont des programmes autonomes, comme l'outil Pajek classique ou le plus récent Gephi, tandis que d'autres sont intégrés dans un environnement de programmation. Quelques exemples de ces dernières sont NetworkX en Python et igraph en R.

Plusieurs progiciels d'extension R mettent en œuvre un ou plusieurs algorithmes d'analyse de réseau ou fournissent des outils généraux pour manipuler les données de réseau et mettre en œuvre des algorithmes de réseau. R prend en charge des graphiques de haute qualité et pratiquement tous les formats de fichiers graphiques courants. Étant un langage de programmation complet, R offre une grande flexibilité pour la recherche sur les réseaux. Les nouveaux algorithmes d'analyse de réseau peuvent être rapidement prototypés en s'appuyant sur les package d'extension existants de la science des réseaux, dont le plus couramment utilisé est le package `igraph`. En plus des implémentations de méthodes classiques et récemment publiées d'analyse de réseau, `igraph` fournit des outils pour importer, manipuler et visualiser des graphes, et peut être utilisé comme plateforme pour de nouveaux algorithmes

Installation

Le `networkD3` fonctionne très bien avec la version la plus récente de RStudio. Lorsque vous utilisez cette version de RStudio, des graphiques apparaissent dans la fenêtre de visualisation. Non seulement cela vous permet de voir et de modifier vos graphiques, mais vous pouvez également les exporter vers le presse-papiers ou vers un fichier PNG/JPEG/TIFF/etc.

```
#install.packages('networkD3')
```

Dans ce tutoriel, nous allons voir les 4 types de graphiques proposés via le package NetworkD3:

Simple Network

Pour pouvons modéliser le Simple Network en se basant sur le réseau social d'une groupe de camarades de la promotion Msc Data Management. En effet, chaque membre de la classe désigne 3 personnes avec qui il se considère le plus proche. Nous obtenons le réseau social suivant:

```r
# Chargement du package
library(networkD3)

# Création de données fictives
src <- c("Claire", "Claire", "Claire", "Adrien",
         "Adrien", "Adrien", "Claude", "Claude", "Claude","Siva", "Siva", "Siva","Thuy","Thuy","Thuy")
target <- c("Adrien", "Claude", "Siva", "Arnaud",
             "Siva", "Claude", "Claire", "Arnaud", "Siva","Claude","Adrien","Claire","Claire","Arnaud","
networkData <- data.frame(src, target)

# Plot
simpleNetwork(networkData)
```

Force Network

On utilise `forceNetwork` pour avoir plus de contrôle sur l'apparition du réseau dirigé forcé et pour tracer des réseaux plus compliqués. Voici un exemple :

```r
# Chargement de données
data(MisLinks)
data(MisNodes)

# Plot
forceNetwork(Links = MisLinks, Nodes = MisNodes,
             Source = "source", Target = "target",
             Value = "value", NodeID = "name",
             Group = "group", opacity = 0.8)
```

Sankey diagrammes

L'exemple ci-dessous montre comment les organigrammes du réseau Sankey peuvent être facilement générés dans R à l'aide du package networkD3, et peuvent être utiles pour illustrer les flux de préférences.

Nous sommes intéressés de voir comment les résultats du vote des 12 régions britanniques ont contribué au résultat global du référendum, où le Royaume-Uni a voté pour quitter l'Union européenne par 17 410 742 voix contre 16 141 241.

Nous commençons par charger les bibliothèques et lire les données brutes qui peuvent être obtenues sur le site de la Commission électorale.

```r
## chargement des librairies

library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(networkD3)
library(tidyr)

# lire le dataset des résultats du referundum EU
refresults <- read.csv("https://data.london.gov.uk/download/eu-referendum-results/52dccf67-a2ab-4f43-a6
```

```
head(refresults)
```

```
##    id Region_Code Region Area_Code                Area Electorate
## 1 108   E12000006   East E06000031         Peterborough     120892
## 2 109   E12000006   East E06000032               Luton     127612
## 3 112   E12000006   East E06000033      Southend-on-Sea     128856
## 4 113   E12000006   East E06000034             Thurrock     109897
## 5 110   E12000006   East E06000055              Bedford     119530
## 6 111   E12000006   East E06000056 Central Bedfordshire     204004
##   ExpectedBallots VerifiedBallotPapers Pct_Turnout Votes_Cast Valid_Votes
## 1           87474                87469       72.35      87469       87392
## 2           84633                84636       66.31      84616       84481
## 3           93948                93939       72.90      93939       93870
## 4           79969                79954       72.75      79950       79916
## 5           86136                86136       72.06      86135       86066
## 6          158904               158896       77.89     158894      158804
##   Remain Leave Rejected_Ballots No_official_mark Voting_for_both_answers
## 1  34176 53216               77                0                      32
## 2  36708 47773              135                0                      85
## 3  39348 54522               69                0                      21
## 4  22151 57765               34                0                       8
## 5  41497 44569               69                0                      26
## 6  69670 89134               90                0                      34
##   Writing_or_mark Unmarked_or_void Pct_Remain Pct_Leave Pct_Rejected
## 1               7               38      39.11     60.89         0.09
## 2               0               50      43.45     56.55         0.16
## 3               0               48      41.92     58.08         0.07
## 4               3               23      27.72     72.28         0.04
## 5               1               42      48.22     51.78         0.08
## 6               1               55      43.87     56.13         0.06
```

Nous regroupons les données par région, supprimer les données inutiles et les formater pour permettre la construction plus évidente d'un réseau Sankey.

```
# aggregate by region

results <- refresults %>%
  dplyr::group_by(Region) %>%
  dplyr::summarise(Remain = sum(Remain), Leave = sum(Leave))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```r
# format in prep for sankey diagram

results <- tidyr::gather(results, result, vote, -Region)

head(results)
```

```
## # A tibble: 6 x 3
##   Region           result    vote
##   <chr>            <chr>    <int>
## 1 East             Remain 1448616
## 2 East Midlands    Remain 1033036
## 3 London           Remain 2263519
## 4 North East       Remain  562595
## 5 North West       Remain 1699020
## 6 Northern Ireland Remain  440707
```

Enfin, nous générons l'ensemble des nœuds et l'ensemble des liens pour le réseau Sankey.

```r
# création des noeuds

regions <- unique(as.character(results$Region))
nodes <- data.frame(node = c(0:13),
                    name = c(regions, "Leave", "Remain"))

#création des liens

results <- merge(results, nodes, by.x = "Region", by.y = "name")
results <- merge(results, nodes, by.x = "result", by.y = "name")
links <- results[ , c("node.x", "node.y", "vote")]
colnames(links) <- c("source", "target", "value")
```

Nous avons maintenant les nœuds et les liens dans le format dont nous avons besoin.

```r
head(links)
```

```
##   source target   value
## 1      0     12 1880367
## 2      1     12 1475479
## 3      2     12 1513232
## 4      3     12  778103
## 5      4     12 1966925
## 6      5     12  349442
```

```r
head(links)
```

```
##   source target   value
## 1      0     12 1880367
## 2      1     12 1475479
## 3      2     12 1513232
## 4      3     12  778103
## 5      4     12 1966925
## 6      5     12  349442
```

```
#draw sankey network

networkD3::sankeyNetwork(Links = links, Nodes = nodes, Source = 'source',
                         Target = 'target', Value = 'value', NodeID = 'name',
                         units = 'votes')
```

Radial Network

La fonction `radialNetwork` permet de modéliser un dataset sous forme d'arbre Reingold-Tilford comme avec l'exemple ci-dessous:

```
Flare <- jsonlite::fromJSON(
  "https://gist.githubusercontent.com/mbostock/4063550/raw/a05a94858375bd0ae023f6950a2b13fac5127637/fla
  simplifyDataFrame = FALSE
)

hc <- hclust(dist(USArrests), "ave")

radialNetwork(List = Flare, fontSize = 10, opacity = 0.9, margin=0)
```

```
radialNetwork(as.radialNetwork(hc))
```

```
# and with a different font
radialNetwork(List = Flare, fontSize = 10, opacity = 0.9, margin=0, fontFamily = "sans-serif")
```

```
diagonalNetwork(List = Flare, fontSize = 10, opacity = 0.9, margin=0)
```

```
diagonalNetwork(as.radialNetwork(hc), height = 700, margin = 50)
```

Sources :

Source CRAN

Source Towards Data Science

[Source Statistical Analysis of Network Data with R]

R Pubs

Rdrr