Tutorial no. 1: Introducción a las redes Bayesianas en R

Borja Calvo, Aritz Pérez

Resumen

Este es el primer tutorial de una serie cuyo objetivo es familiarizarse con la creación y manejo de redes Bayesianas en `R`. Los siguientes tutoriales se basan fundamentalmente en dos paquetes, `bnlearn` y `gRain` . En cada uno de ellos veremos como crear redes usando estos paquetes. El objetivo de este documento es introducir algunos de los paquetes básicos que usaremos, repasando para ellos algunos de los conceptos aprendidos en la parte teórica.

1 Preparación

Como paso previo a la realización de los tutoriales necesitamos instalar los paquetes a utilizar, para lo que ejecutaremos:

```
> source("http://bioconductor.org/biocLite.R")
> biocLite("graph")
> biocLite("RBGL")
> biocLite("Rgraphviz")
> install.packages("bnlearn")
> install.packages("gRain")
> install.packages("ggplot2")
> install.packages("gridExtra")
```

Una vez instalados los paquetes cargaremos en la sesión los que necesitaremos ejecutando:

```
> library(bnlearn)
> library(gRbase)
> library(RBGL)
```

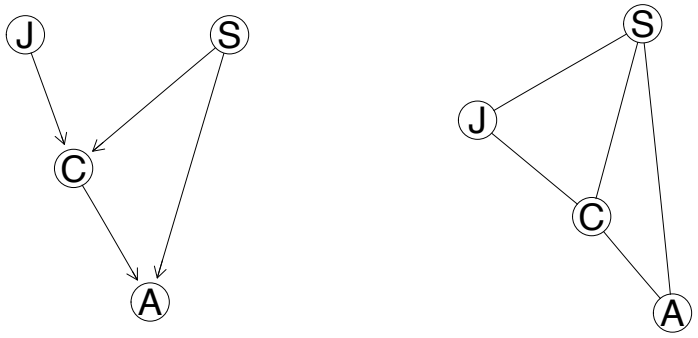
2 Creación y análisis de grafos acíclicos dirigidos

El paquete `gRbase` nos permite crear tanto grafos no dirigidos como grafos dirigidos. En particular, podemos crear un DAG usando cualquiera de las siguientes opciones:

```
> dag.study<-dag(~S,~J,~C*S*J,~A*S*C)
> dag.study<-dag(~S + J + C*S*J + A*S*C)
> dag.study<-dag(~S + J + C|S*J + A|S*C)
> dag.study<-dag("S","J",c("C","S","J"),c("A","S","C"))
> dag.study
```

Año: Master ICSI/KISA
Profesor(a): Borja Calvo, Aritz Pérez
Teléfono: 943 01 50 13
E-mail: [borja.calvo,aritz.perez]@ehu.es

GIPUZKOAKO CAMPUSA
CAMPUS DE GUIPUZCOA
Paseo Manuel de Lardizabal, 1
20018 Donostia (Gipuzkoa)



t

(a) DAG (b) Grafo moral

Figura 1: Ejemplo de DAG

```
## A graphNEL graph with directed edges
## Number of Nodes = 4
## Number of Edges = 4
```

Como se puede apreciar, la función `dag` crea un objeto de tipo `graphNEL`, el cual es uno de los tipos básicos que manejaremos en estos tutoriales. Este objeto tiene definido el método `plot`, por lo que podemos visualizar el gráfo creado simplemente ejecutando el siguiente comando (el resultado se muestra en la Figura 1-a).

```
> plot(dag.study)
```

Este tipo de objeto se basa en una representación de tipo lista de nodos + lista de vértices. No obstante, en ocasiones es interesante manejar otro tipo de representación, la matriz de adjacencia. Esto lo podemos conseguir con la misma función, añadiendo la opción `result="matrix"`.

```
> adjmat.study<-dag(~S + J + C|S*J + A|S*C,result="matrix")
> adjmat.study

##   S J C A
## S 0 0 1 1
## J 0 0 1 0
## C 0 0 0 1
## A 0 0 0 0
```

Dada una matriz de adjacencia podemos construir un objeto de tipo `graphNEL`.

```
> dag.study<-as(adjmat.study,"graphNEL")
> dag.study

## A graphNEL graph with directed edges
## Number of Nodes = 4
## Number of Edges = 4
```

La lista de nodos y de arcos de un grafo las podemos obtener con las funciones `nodes` y `edges`, implementadas en el paquete `graph`. Con la primera función podemos tener problemas, ya que el paquete `bnlearn` también implementa una función con el mismo nombre, por lo que si cargamos este paquete la función quedará enmascarada. En estos casos lo que podemos hacer es usar la llamada completa, indicando el nombre de la función más el del paquete.

```
> nodes(dag.study)

## [1] "S" "J" "C" "A"

> graph::nodes(dag.study)

## [1] "S" "J" "C" "A"

> edges(dag.study)

## $S
## [1] "C" "A"
##
## $J
## [1] "C"
##
## $C
## [1] "A"
##
## $A
## character(0)
```

Alternativamente podemos visualizar los arcos no como la lista de hijos de cada nodo sino como una lista.

```
> edgeList(dag.study)

## [[1]]
## [1] "S" "C"
##
## [[2]]
## [1] "S" "A"
##
## [[3]]
## [1] "J" "C"
##
## [[4]]
## [1] "C" "A"
```

El paquete `gRbase` nos ofrece también funciones para consultar la lista de padres e hijos de un nodo (de nuevo, estas funciones puede que esten enmascaradas por las correspondientes del paquete `bnlearn`).

```
> gRbase::parents(set="C",object=dag.study)

## [1] "S" "J"

> gRbase::children(set="C",object=dag.study)

## [1] "A"
```

Extendiendo esta idea, podemos obtener, dado un conjunto de nodos, su conjunto/grafo ancestral.

```
> ancestors(set="A",object=dag.study)

## [1] "S" "C" "J"
```

```
> ancestralSet(set="C",object=dag.study)

## [1] "S" "J" "C"

> plot(ancestralGraph(set="C",object=dag.study))
```

Una operación típica en el manejo de redes Bayesianas es la moralización. Este paso lo podemos llevar a cabo con la función `moralize` (ver Figura 1-b). Dado el grafo moralizado podemos ver la lista de cliques maximales del grafo.

```
> moral.graph<-moralize(dag.study)
> plot(moral.graph)
> maxClique(moral.graph)

## $maxCliques
## $maxCliques[[1]]
## [1] "S" "C" "J"
##
## $maxCliques[[2]]
## [1] "S" "C" "A"
```

Para finalizar, podemos utilizar la función `separates` definida en el paquete `RBGL` para determinar independencias condicionales. Esta función considera solo grafos no dirigidos, por lo que debemos chequear las independencias en el grafo moral.

```
> separates("A","J","C",moral.graph)

## [1] FALSE

> separates("A","J",c("C","S"),moral.graph)

## [1] TRUE
```

3 Ejercicios

Problema 3.1 *Empleando las funciones que se han presentado en el tutorial, implementar una funcion que implemente el metodo de la d-separación para leer independencias condicionadas a partir del grafo*

Problema 3.2 *Dado el DAG de la Figura 2:*

- *Comprobar si se verifican las independencias* $(1;4|5)$, $(1;5|3,4)$, $(6;7|3)$, $(1;7|5)$, $(1;4|6)$ *y* $(1;4|2,3)$
- *Qué conjunto(s) de variables* C *verifican* $(1;7|C)$ *y* $(1;5|C)$

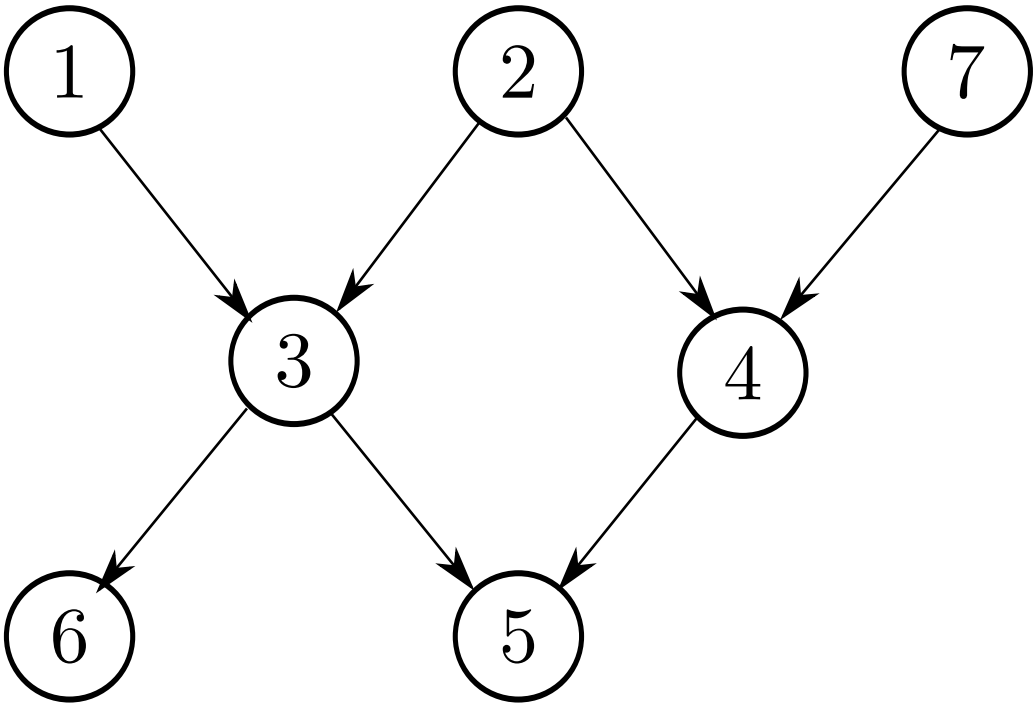


Figura 2: DAG