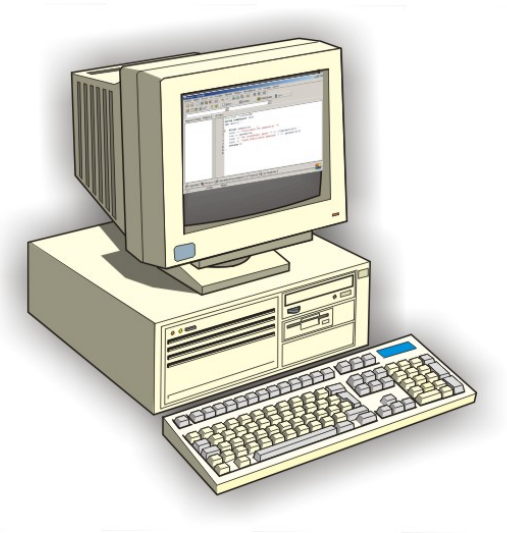


Práctica 2

Introducción al uso de redes bayesianas con R

Febrero de 2017

DECSAI

Modelos Gráficos Probabilísticos

Enero-Febrero de 2017

Contents

1 Introducción al guion

Este guion está basado en el libro *Bayesian Networks With Examples in R*. Marco Scutari y Jean-Baptiste Denis. CRC Press. 2015.

Para crear y manipular redes bayesianas, usaremos principalmente el paquete **bnlearn** (**B**ayesian **n**etwork **learn**ing) de **R**. Si no lo tienes instalado, instálalo:

```
> install.packages("bnlearn")
```

Puedes encontrar la documentación sobre este paquete en **CRAN** en http://cran.r-project.org/web/packages/bnlearn/index.html. La web asociada a este paquete es http://www.bnlearn.com/

Comienza cargando la librería **bnlearn**:

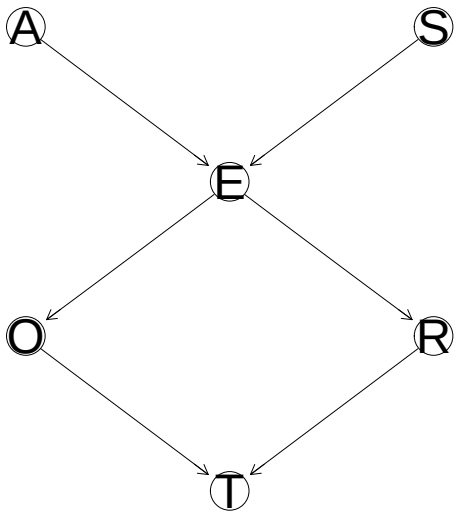
```
> library(bnlearn)
```

2 El problema a modelar: Encuesta sobre uso de medios de transporte

En este guion vamos a crear una red bayesiana para el problema siguiente. Se quiere hacer un estudio sobre los patrones de uso de diferentes medios de transporte por parte de una serie de personas, con el foco puesto en el coche y el tren. Para ello se realizará una encuesta. Para cada individuo, se examinarán seis variables:

- Edad (A): puede tomar los valores $young$ (menos de 30 años), $adult$ (entre 30 y 60 años) y old (mayores de 60 años).
- Sexo (S): puede ser M (hombre) o F (mujer).
- Nivel educativo (E): el nivel de educación más alto alcanzado por el individuo, $high$ (estudios superiores) o uni (universitarios).
- Ocupación (O): indica si el individuo trabaja como empleado (emp) o como autónomo ($self$).
- Residencia (R): el tamaño de la ciudad donde vive el individuo, $small$ (ciudad pequeña) o big (ciudad grande).
- Travel (T): el medio de transporte que suele usar el individuo, car (coche), $train$ (tren) o $other$ (otro).

Supondremos que este problema se puede modelar con el *Grafo Dirigido Acíclico* (DAG) que se muestra a continuación:



Suponemos las siguientes distribuciones de probabilidad:

$P(A)$	$young$	$adult$	old
	0.30	0.50	0.20

(1)

$P(S)$	M	F
	0.60	0.40

(2)

$P(E\|A,S)$	$high$	uni
$young, M$	0.75	0.25
$adult, M$	0.72	0.28
old, M	0.88	0.12
$young, F$	0.64	0.36
$adult, F$	0.70	0.30
old, F	0.90	0.10

(3)

$P(R\|E)$	$small$	big
$high$	0.25	0.75
uni	0.2	0.8

(4)

$P(O\|E)$	emp	$self$
$high$	0.96	0.04
uni	0.92	0.08

(5)

$P(T\|O,R)$	car	$train$	$other$
$small, emp$	0.48	0.42	0.10
$small, self$	0.56	0.36	0.08
big, emp	0.58	0.24	0.18
$big, self$	0.70	0.21	0.09

(6)

3 Creación del DAG de la red bayesiana

Crearemos el DAG de la red bayesiana, con un nodo por cada variable. Usamos la función `empty.graph` que permite generar un grafo dirigido acíclico vacío o aleatorio para un conjunto de nodos dado. Esta función crea el DAG como un objeto de la clase `bn`. En nuestro caso, crearemos un grafo con las variables A, S, E, O, R y T.

```
> dag <- empty.graph(nodes = c("A", "S", "E", "O",
        "R", "T"))
```

El grafo anterior tiene por ahora un conjunto vacío de arcos. El DAG está almacenado en un objeto de la clase `bn`. Podemos imprimir en la consola el objeto que acabamos de crear:

```
> dag
```

El resultado sería el siguiente:

```
> dag
  Random/Generated Bayesian network

  model:
   [A][S][E][O][R][T]
  nodes:                                6
  arcs:                                 0
    undirected arcs:                    0
    directed arcs:                      0
  average markov blanket size:          0.00
  average neighbourhood size:           0.00
  average branching factor:             0.00

  generation algorithm:                 Empty
```

A continuación, añadiremos los arcos entre los nodos del DAG de la red bayesiana. Por ejemplo, el enlace entre las variables `A` y `E` lo añadimos con:

```
> dag <- set.arc(dag, from = "A", to = "E")
```

Añade tú el resto de arcos al DAG. Una vez hecho, si volvemos a imprimir nuestro DAG por la consola, obtendríamos:

```
> dag

  Random/Generated Bayesian network

  model:
```

```
  [A][S][E|A:S][O|E][R|E][T|O:R]
 nodes:                                  6
 arcs:                                   6
   undirected arcs:                      0
   directed arcs:                        6
 average markov blanket size:            2.67
 average neighbourhood size:             2.00
 average branching factor:               1.00

 generation algorithm:                   Empty
```

Como puede verse en la anterior salida, en el *model string* (modelo) existe un elemento por cada variable. Por ejemplo, `[E|A:S]` expresa que hay una dependencia directa de `E` con `A` y `S`, por lo que en el grafo tenemos los enlaces $A \rightarrow E$ y $S \rightarrow E$. Sin embargo `[A]` expresa que no hay ningún arco apuntando hacia A.

Podemos usar la función `modelstring` para ver o modificar el modelo. Por ejemplo, para verlo usaríamos:

```
> modelstring(dag)
[1] "[A][S][E|A:S][O|E][R|E][T|O:R]"
```

Podríamos haber definido los enlaces entre las variables con la función `modelstring`:

```
> modelstring(dag)= "[A][S][E|A:S][O|E][R|E][T|O:R]"
```

Usando la función `model2network` podríamos haber creado también el DAG (variables y enlaces) especificando directamente el modelo, de la misma forma que con `modelstring`:

```
dag3 <- model2network("[A][S][E|A:S][O|E][R|E][T|O:R]")
```

El DAG resultante del comando anterior (`dag3`), es idéntico al que hemos creado anteriormente (`dag`). Esto puede comprobarse con la función `all.equal`:

```
> all.equal(dag,dag3)
[1] TRUE
```

Podemos mostrar la lista de nodos del DAG de la red bayesiana con la función `nodes`:

```
> nodes(dag)
[1] "A" "S" "E" "O" "R" "T"
```

También podemos mostrar la lista de enlaces con la función `arcs`:

```
> arcs(dag)
     from to
[1,] "A"  "E"
[2,] "S"  "E"
[3,] "E"  "O"
[4,] "E"  "R"
[5,] "O"  "T"
[6,] "R"  "T"
```

Una forma alternativa para añadir los arcos a un DAG es mediante el uso de la función **arcs**. Por ejemplo, podemos crear un nuevo DAG con las mismas variables y arcos de la siguiente forma:

```
> dag2 <- empty.graph(nodes = c("A", "S", "E", "O",
         "R", "T"))
> arc.set <- matrix(c("A", "E",
                      "S", "E",
                      "E", "O",
                      "E", "R",
                      "O", "T",
                      "R", "T"),
            byrow = TRUE, ncol = 2,
            dimnames = list(NULL, c("from", "to")))
> arcs(dag2) <- arc.set
```

Podemos comprobar si los dos DAGs creados hasta ahora son iguales con la función **all.equal**:

```
> all.equal(dag,dag2)
[1] TRUE
```

Las dos formas que hemos visto para añadir los arcos, garantizan que el DAG es acíclico. Por ello, si intentamos introducir un ciclo a propósito, se devolverá un error:

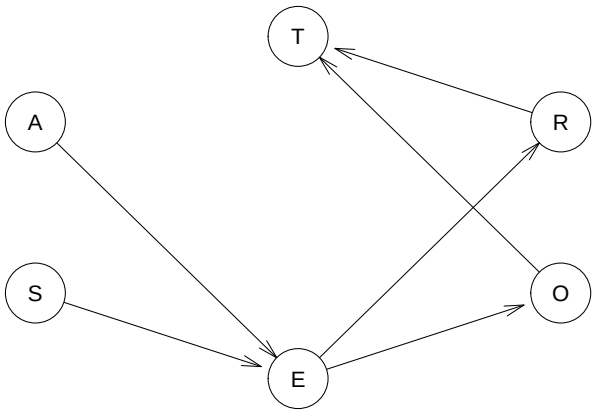
```
> try(set.arc(dag, from="T", to="E"))
Error in arc.operations(x = x, from = from, to = to,
  op = "set", check.cycles = check.cycles,  :
  the resulting graph contains cycles.
```

4 Dibujando el DAG de la red bayesiana

Podemos mostrar un gráfico del DAG de la red bayesiana mediante la función **plot**:

```
> plot(dag)
```

El resultado tendría el siguiente aspecto:



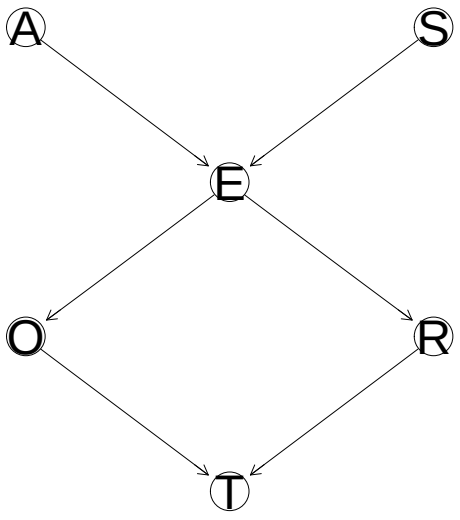
Alternativamente podríamos usar el paquete **Rgraphviz** para mostrar el DAG. Este paquete no está en **CRAN** y debe instalarse de la siguiente forma:

```
> source("http://bioconductor.org/biocLite.R")
> biocLite("Rgraphviz")
```

Una vez instalado, mostramos nuestro DAG de la siguiente forma:

```
> graphviz.plot(dag)
```

El resultado tendría ahora el siguiente aspecto:



La función `graphviz.plot` tiene cuidado de colocar los nodos y arcos de forma que se minimice su solapamiento. Por defecto, los nodos se colocan de forma que los padres se dibujan encima de los hijos y la mayoría de los arcos apuntan hacia abajo. A esta forma de colocación (`layout`) se le llama `dot`. Podemos usar otros layouts como `fdp` y `circo`.

5 Especificación de las probabilidades

En la red bayesiana que estamos creando, todas las variables son discretas y por tanto cada una está definida para un número finito y no ordenado de *estados* (el dominio de esa variable). Vamos a definir los estados de las variables de la red bayesiana con los siguientes objetos:

```
> A.st <- c("young", "adult", "old")
> S.st <- c("M", "F")
> E.st <- c("high", "uni")
> O.st <- c("emp", "self")
> R.st <- c("small", "big")
> T.st <- c("car", "train", "other")
```

El DAG que hemos construido hasta ahora nos indica que la distribución de probabilidad conjunta se factoriza de la siguiente forma:

$$P(A, S, E, O, R, T) = P(A)P(S)P(E|A, S)P(O|E)P(R|E)P(T|O, R)$$

Por tanto, debemos definir las seis distribuciones de probabilidad condicional de la anterior factorización. Las variables A y S se definen con las distribuciones de probabilidad unidimensionales (no tienen padres) $P(A)$ y $P(S)$, las cuales podemos definir con un `array`:

```
> A.prob <- array(c(0.30, 0.50, 0.20), dim=3,
          dimnames = list(A = A.st))
```

Podemos mostrar ahora las probabilidades que acabamos de definir para $P(A)$:

```
> A.prob
A
young adult   old
  0.3   0.5   0.2
```

De igual forma, para $P(S)$ haríamos:

```
> S.prob <- array(c(0.60, 0.40), dim=2,
+                 dimnames = list(S = S.st))
>
```

```
> S.prob
S
  M   F
0.6 0.4
```

Las distribuciones condicionales $P(O|E)$ y $P(R|E)$ se definen con:

```
> O.prob <- array(c(0.96, 0.04, 0.92, 0.08), dim=c(2,2),
+                 dimnames = list(O = O.st, E = E.st))
> O.prob
      E
O      high  uni
  emp  0.96 0.92
  self 0.04 0.08
```

```
> R.prob <- array(c(0.25, 0.75, 0.20, 0.80), dim=c(2,2),
+                 dimnames = list(R = R.st, E = E.st))
> R.prob
       E
R       high uni
  small 0.25 0.2
  big   0.75 0.8
```

Para las distribuciones unidimensionales y bidimensionales, podemos usar también la función **`matrix`**. La sintaxis es casi idéntica a la de **`array`**; la única diferencia es que se debe especificar solo una dimensión (o el número de filas **`nrow`**, o el número de columnas **`ncol`**). Por ejemplo, para $P(R|E)$ podríamos haber ejecutado:

```
> R.prob <- matrix(c(0.25, 0.75, 0.20, 0.80), ncol = 2,
          dimnames = list(R = R.st, E = E.st))
```

Las distribuciones $P(E|A,S)$ y $P(T|O,R)$ las definimos con:

```
> E.prob <- array(c(0.75, 0.25, 0.72, 0.28, 0.88, 0.12, 0.64,
             0.36, 0.70, 0.30, 0.90, 0.10), dim=c(2, 3, 2),
          dimnames = list(E = E.st, A = A.st, S = S.st))
```

```
> T.prob <- array(c(0.48, 0.42, 0.10, 0.56, 0.36, 0.08, 0.58,
             0.24, 0.18, 0.70, 0.21, 0.09), dim=c(3, 2, 2),
          dimnames = list(T = T.st, O = O.st, R = R.st))
```

6 Creación de la red bayesiana

Una vez definidas todas las distribuciones de probabilidad necesarias, construimos una lista (que llamaremos `cpt`) con ellas.

```
> cpt <- list(A=A.prob, S=S.prob, E=E.prob, O=O.prob,
        R=R.prob, T=T.prob)
```

Con esto, ya tenemos definido todo lo necesario para crear nuestra red bayesiana mediante un objeto de la clase `bn.fit` que llamaremos `bn`. Para ello, usaremos la función `custom.fit`:

```
> bn <- custom.fit(dag,cpt)
```

El número de parámetros de la red bayesiana puede ser calculado con la función `nparams`, que en este caso nos da el valor 21, como puede comprobarse observando las 6 tablas de distribuciones de probabilidad de la red. Por ejemplo, el número de parámetros en la tabla para $P(A)$ sería de 2, ya que contiene tres probabilidades, pero solo 2 son independientes; la tercera puede obtenerse a partir de las otras ya que deben sumar 1.

```
> nparams(bn)
[1] 21
```

Los objetos de la clase `bn.fit` se usan para describir redes bayesianas en el paquete **bnlearn**. Estos objetos incluyen información sobre el DAG (padres e hijos de cada nodo) y las distribuciones de probabilidad locales (sus parámetros). Los objetos `bn.fit` pueden usarse como si fuesen objetos de la clase `bn` a la hora de investigar sus propiedades gráficas. Por ejemplo, podemos usar `arcs` con un objeto `bn.fit`:

```
> arcs(bn)
     from to
[1,] "A"  "E"
[2,] "S"  "E"
[3,] "E"  "O"
[4,] "E"  "R"
[5,] "O"  "T"
[6,] "R"  "T"
```

Podríamos usar de la misma forma las funciones `nodes`, `parents` y `children` de la clase `bn`.

Podemos también mostrar las tablas de probabilidad condicional del objeto `bn.fit` de la siguiente forma:

```
> bn$R

  Parameters of node R (multinomial distribution)

Conditional probability table:

       E
R       high  uni
  small 0.25 0.20
  big   0.75 0.80
```

Podemos extraer una tabla de distribución de probabilidad condicional para su uso posterior con la función `coef`:

```
> R.cpt <- coef(bn$R)
```

Para mostrar en la consola todas las distribuciones de probabilidad condicional ejecutamos:

```
> bn
```

7 Estimación de los parámetros: distribuciones de probabilidad condicional

En la sección anterior hemos supuesto que conocíamos tanto el DAG como los parámetros (distribuciones de probabilidad locales) de la red bayesiana. Éste sería el caso en que se construyera preguntando a un experto en el tema. Sin embargo en la mayoría de los casos, los parámetros de las distribuciones locales deben estimarse a partir de una muestra de datos observados. Habitualmente los datos estarán almacenados en un fichero de texto que podremos importar con la función `read.table`.

En nuestro caso, vamos a suponer que tenemos los resultados de la encuesta efectuada a un conjunto de 500 personas en el fichero de texto `survey.txt` (que puedes descargar de la plataforma de la asignatura). El fichero contiene una columna por cada variable y una observación por cada línea. Las etiquetas de las variables aparecen en la primera fila.

```
> survey <- read.table("survey.txt", header = TRUE)
```

Podemos mostrar las primeras líneas del fichero que acabamos de leer usando la función `head`:

```
> head(survey)
      A     R    E   O S     T
1 adult   big high emp F   car
2 adult small  uni emp M   car
3 adult   big  uni emp F train
4 adult   big high emp M   car
5 adult   big high emp M   car
6 adult small high emp F train
```

Los parámetros que debemos estimar en una red bayesiana, son las distribuciones de probabilidad condicional. Una forma de obtenerlas es a través de las frecuencias empíricas en el dataset. Por ejemplo:

$$\hat{P}(O = emp|E = high) = \frac{\hat{P}(O = emp, E = high)}{\hat{P}(E = high)} =$$

$$= \frac{\text{número de observaciones para las que } O = emp \text{ y } E = high}{\text{número de observaciones para las que } E = high}$$

Esto produce los estimadores clásicos *frecuentistas* y *de máxima verosimilitud*. En el paquete **bnlearn**, la función `bn.fit` permite calcular estos estimadores especificando un DAG y una tabla con las observaciones a partir de las cuales se estimarán las distribuciones de probabilidad condicional. El argumento `method` permite especificar el modo de calcularlas. Usaremos `mle` para una *estimación por máxima verosimilitud* (frecuentista clásica).

```
> bn.mle <- bn.fit(dag, data = survey, method = "mle")
```

Cada una de las distribuciones de probabilidad podrían haberse calculado una a una manualmente. Por ejemplo, $P(O|E)$ podría estimarse de la siguiente forma:

```
> prop.table(table(survey[, c("O", "E")]), margin = 2)
      E
O        high    uni
  emp  0.9808 0.9259
  self 0.0192 0.0741
```

que como se ve, da el mismo resultado que nos dió `bn.fit`:

```
> bn.mle$O

  Parameters of node O (multinomial distribution)

Conditional probability table:

      E
O        high    uni
  emp  0.9808 0.9259
  self 0.0192 0.0741
```

Alternativamente, podemos estimar las probabilidades por un *procedimiento bayesiano*, usando sus distribuciones a posteriori, usaremos `bayes` en el argumento `method` de la función `bn.fit`. Por ejemplo:

```
> bn.bayes <- bn.fit(dag, data = survey, method = "bayes",
+                    iss = 10)
```

Las probabilidades a posteriori estimadas se calculan a partir de una distribución uniforme sobre cada tabla de distribución de probabilidad condicional. El argumento opcional `iss` en el anterior comando, indica el *imaginary sample size* o *equivalent sample size*(tamaño muestral equivalente), que determina cuánto peso se asigna a las distribuciones a priori respecto a los datos al calcular las a posteriori. El peso se especifica como el tamaño de una muestra imaginaria que define la distribución a priori. Su valor se divide por el número de celdas en la tabla de probabilidad condicional (ya que la a priori es plana) y se usa para calcular el estimador de la a posteriori como una media ponderada con las frecuencias empíricas.

Por ejemplo, supongamos que tenemos una muestra de tamaño n, valor que podemos obtener con `nrow(survey)`. Sean

$$\hat{p}_{\text{emp,high}} = \frac{\text{número de observaciones para las que } O = emp \text{ y } E = high}{n}$$

$$\hat{p}_{\text{high}} = \frac{\text{número de observaciones para las que } E = high}{n}$$

y denotemos las correspondientes probabilidades a priori como:

$$\pi_{\text{emp,high}} = \frac{1}{\text{nO} \times \text{nE}}$$

y

$$\pi_{\text{high}} = \frac{1}{\text{nO} \times \text{nE}}$$

donde `nO=nlevels(bn.bayes$O)` y `nE=nlevels(bn.bayes$E)`. De esta forma tenemos que:

$$\hat{P}(\text{O=emp}, \text{E=high}) = \frac{iss}{n + iss}\pi_{\text{emp,high}} + \frac{n}{n + iss}\hat{p}_{\text{emp,high}}$$

$$\hat{P}(\text{E=high}) = \frac{iss}{n + iss}\pi_{\text{high}} + \frac{n}{n + iss}\hat{p}_{\text{high}}$$

Finalmente, el estimador bayesiano para la distribución condicional sería:

$$\hat{P}(\text{O=emp}|\text{E=high}) = \frac{\hat{P}(\text{O=emp}, \text{E=high})}{\hat{P}(\text{E=high})}$$

El valor de `iss` se elige normalmente pequeño entre 1 y 15 para permitir que la distribución a priori sea fácilmente dominada por los datos.

Si consultamos ahora la distribución estimada para $P(O|E)$ obtenemos:

```
> bn.bayes$O

  Parameters of node O (multinomial distribution)

Conditional probability table:

      E
O        high    uni
  emp  0.9743 0.9107
  self 0.0257 0.0893
```

Las nuevas probabilidades están ahora más lejos de los valores 0 y 1 que cuando se usó el método de máxima verosimilitud debido a la influencia de la distribución a priori. Esto suele ser deseable por varias razones. Primero, esto asegura que no obtendremos tablas de probabilidad condicional con muchos ceros, incluso aunque tengamos un dataset pequeño.

Además, estos estimadores son más robustos que los de máxima verosimilitud ya que producen redes bayesianas con una mejor potencia predictiva.

Cuanto mayor sea el valor de `iss`, más planas serán las distribuciones a posteriori estimadas, empujándolas hacia la distribución uniforme usada como la a priori. Puedes comprobarlo tú mismo, si comparas las distribuciones condicionales que obtuviste anteriormente (usando $iss = 10$) con las que se obtienen usando $iss = 20$.

```
> bn.bayes <- bn.fit(dag, data = survey, method = "bayes",
+                iss = 20)
> bn.bayes$O
  Parameters of node O (multinomial distribution)

Conditional probability table:

      E
O       high   uni
  emp  0.968 0.897
  self 0.032 0.103
```

8 Aprendiendo el DAG: Tests y Scores

En las secciones previas hemos asumido que el DAG de la red bayesiana era conocido. Esto no siempre es posible o deseable. El DAG puede ser también aprendido a partir de una muestra de las variables del problema. Existen dos clases de criterios estadísticos usados en los algoritmos para evaluar posibles DAGs: *tests de independencia condicional* y *scores de redes*.

Por ejemplo, la función `hc` usa un método que utiliza scores para hacer una búsqueda por los posibles DAGs haciendo uso de un mecanismo greedy (*hill-climbing*). El algoritmo comienza con un DAG vacío y va añadiendo, borrando o invirtiendo arcos cada vez al DAG, eligiendo el cambio que incremente más el score actual. En nuestro caso, lo usaríamos de la siguiente forma:

```
> learned <- hc(survey)
> modelstring(learned)
[1] "[R][E|R][T|R][A|E][O|E][S|E]"
> score(learned, data = survey, type = "bic")
[1] -1998
```

En el caso anterior, el algoritmo de aprendizaje usó el score BIC, que es el de por defecto en la función `hc`. Podemos usar el argumento `score` para indicar otro tipo de score. Por ejemplo, para usar el score BDE:

```
> learned2 <- hc(survey, score = "bde")
```

9 Inferencia en redes bayesianas

9.1 Obtención de las independencias expresadas por el DAG

En una red bayesiana, el DAG define el siguiente tipo de independencia entre variables. Dos conjuntos de variables $\mathbf{X}$ y $\mathbf{Y}$ son independientes dado un tercer conjunto $\mathbf{Z}$ si todos los caminos que conectan variables de $\mathbf{X}$ con $\mathbf{Y}$ están *bloqueados* o *separados* por las variables de $\mathbf{Z}$. Para ello se aplica el criterio de *D-separación*. Las independencias expresadas gráficamente en el DAG con este criterio, se dan también como independencias probabilísticas entre las variables de la red bayesiana.

La función `dsep` permite saber si dos variables `x` e `y` del DAG están d-separados dada otra variable `z` o un vector de variables `z`. Por ejemplo, para ver si S y R son independientes (S y R están d-separados sin ninguna evidencia) ejecutamos:

```
> dsep(dag, x = "S", y = "R")
[1] FALSE
```

De igual forma, con O y R:

```
> dsep(dag, x = "O", y = "R")
[1] FALSE
```

La función `path` permite ver si existe un camino dirigido de un nodo a otro. Por ejemplo:

```
> path(dag, from = "S", to = "R")
[1] TRUE
```

El camino entre S y R queda bloqueado si condicionamos sobre E (S y R se vuelven independientes conocido el valor de E) como puede verse con:

```
> dsep(dag, x = "S", y = "R", z = "E")
[1] TRUE
```

Esta independencia indica que:

$$P(S, R|E) = P(S|E) \cdot P(R|E)$$

Otro ejemplo, S y T son independientes conocido el valor de O y de R:

```
> dsep(dag, x="S", y="T",z=c("O", "R"))
[1] TRUE
```

Las variables O y R también están d-separadas dada la variable E:

```
> dsep(dag, x = "O", y = "R", z = "E")
[1] TRUE
```

Por otro lado, el condicionar sobre otras variables, puede hacer que variables que eran independientes marginalmente (sin evidencia) se vuelvan dependientes. Por ejemplo las variables A y S son independientes marginalmente, pero se vuelven dependientes al condicionar sobre E.

```
> dsep(dag, x = "A", y = "S")
[1] TRUE
> dsep(dag, x = "A", y = "S", z = "E")
[1] FALSE
```

9.2 Inferencia exacta

La inferencia exacta está implementada en el paquete **gRain** (**gRa**phical model **in**ference). Este paquete usa los paquetes **graph**, **RBGL** y **Rgraphviz** que no están en **CRAN**, sino en **bionconductor**. Si no los tenemos instalados, podemos instalarlos con:

```
> source("http://bioconductor.org/biocLite.R")
> biocLite(c("graph","RBGL","Rgraphviz"))
```

Una vez instalados los anteriores paquetes, podremos instalar el paquete `gRain`:

```
> install.packages("gRain")
```

Finalmente, cargamos el paquete `gRain`:

```
> library(gRain)
```

El paquete **gRain** fue desarrollado por Søren Højsgaard. Podemos encontrar información sobre éste y otros de sus paquetes en http://people.math.aau.dk/~sorenh/software/gR/. También conviene echar un vistazo a la web http://cran.r-project.org/web/views/gR.html que da un vistazo general a los paquetes **CRAN** para modelos gráficos.

El algoritmo exacto que contiene el paquete **gRain** es el algoritmo de Lauritzen-Spiegelhalter que es un algoritmo basado en la construcción de un árbol de grupos (*junction tree*) y es muy similar a los algoritmos de Hugin y el de Shenoy-Shafer. Puedes ver una comparativa de estos algoritmos en el artículo *A Comparison of Lauritzen-Spiegelhalter, Hugin, and Shenoy-Shafer Architectures for Computing Marginals of Probability Distributions* disponible en http://arxiv.org/pdf/1301.7394.pdf.

Para construir el árbol de grupos con el paquete **gRain**, debemos convertir nuestra red bayesiana, que es de la clase `bn.fit`, en un objeto de la clase `grain`. Para ello usaremos la función `as.grain` del paquete **bnlearn**. Con el objeto `grain`, usaremos la función `compile` para construir el árbol del grupos y hacer la propagación:

```
> junction <- compile(as.grain(bn))
```

La función `querygrain` permite obtener la distribución de probabilidad a posteriori para una variable dada la evidencia. Por ejemplo, para obtener $P(T|evidencia)$ usaremos:

```
> querygrain(junction, nodes="T")$T
T
      car     train     other
0.5618340 0.2808573 0.1573088
```

En la operación anterior, todavía no hemos introducido evidencia en la red bayesiana, por lo que nos ha mostrado $P(T)$ para cada estado de T.

Una vez construido el árbol de grupos y calculadas las tablas de probabilidad a posteriori, podemos definir la evidencia disponible mediante la función `setEvidence`. Esto hará que automáticamente se propague dicha evidencia en el árbol de grupos. Por ejemplo, podemos definir la evidencia $S = F$:

```
> jsex <- setEvidence(junction, nodes="S", states="F")
```

Para obtener de nuevo $P(T|evidencia)$ ($P(T|S = F)$ en este caso) usaremos:

```
> querygrain(jsex,nodes="T")$T
T
      car     train     other
0.5620577 0.2806144 0.1573280
```

Puede verse que las probabilidades obtenidas para T con la evidencia $S = F$, no han cambiado sustancialmente respecto a cuando no había evidencia. Esto nos indica que las mujeres muestran más o menos las mismas preferencias hacia el uso de coche o tren que la población entrevistada como un todo.

Hagamos ahora una consulta para conocer cómo afecta el vivir en una ciudad pequeña al uso de coche o tren, o sea, queremos calcular $P(T|R = small)$. La gente que trabaja en grandes ciudades a menudo vive en pueblos vecinos y viajan diariamente a sus lugares de trabajo, ya que los precios de las casas son más bajos a medida que te alejas hacia el campo. Esto fuerza al uso de tren o coche para viajar a la ciudad, ya que otros medios de transporte (bicicleta, metro, líneas de autobús, etc) no están disponibles o no son prácticas.

```
> jres <- setEvidence(junction, nodes = "R", states = "small")
> querygrain(jres, nodes = "T")$T
T
       car      train      other
0.48388675 0.41708494 0.09902831
```

Después de las anteriores consultas, tenemos:

	$P(T)$	$P(T\|S=F)$	$P(T\|R=small)$
car	0.5618340	0.5620577	0.48388675
train	0.2808573	0.2806144	0.41708494
other	0.1573088	0.1573280	0.09902831

Como podemos ver en la tabla anterior, $P(T = other)$ pasa de 0.1573 (población general) a 0.099 (gente que vive en pequeñas ciudades), mientras que $P(T = train)$ pasa de 0.2808 (población general) a 0.4170 (gente que vive en pequeñas ciudades). La probabilidad combinada de usar coche o tren ($P(T = car \cup T = train)$ pasa de 0.8426 (población general) a 0.9009 (gente que vive en pequeñas ciudades).

La tabla anterior, también nos permite saber cual es la *explicación más probable* para la variable T en cada caso. Por ejemplo, para la gente que vive en pequeñas ciudades, el coche es el medio de transporte preferido (probabilidad igual a 0.48388675).

Es posible también obtener la probabilidad conjunta condicional dada la evidencia (probabilidad conjunta a posteriori) para más de una variable a la vez. Por ejemplo, para obtener $P(S, T|E = high)$ haríamos:

```
> jedu <- setEvidence(junction, nodes = "E", states = "high")
> SxT.cpt <- querygrain(jedu, nodes = c("S", "T"),
+                       type = "joint")
> SxT.cpt
   T
S         car     train      other
  M 0.3426644 0.1736599 0.09623271
  F 0.2167356 0.1098401 0.06086729
```

Esta consulta, nos permite también ver que la *explicación más probable* para las dos variables S (sexo) y T (medio de transporte usado) conjuntamente cuando tienen un nivel de estudios $high$ (alto) es que son hombres que usan el coche.

El argumento `type` en la función `querygrain` especifica el tipo de consulta que queremos. El valor por defecto es `"marginal"`, que nos da la distribución marginal a posteriori para cada variable. Usando `"marginal"` en el anterior caso obtendríamos la marginal para S y la marginal para T

```
> querygrain(jedu, nodes = c("S", "T"), type = "marginal")
$S
S
       M        F
0.612557 0.387443

$T
T
   car  train  other
0.5594 0.2835 0.1571
```

Otra posibilidad para el argumento `type` de la función `querygrain` es usar el valor `conditional`. Esto devuelve la distribución de probabilidad de la primera variable en `nodes` condicionado al resto de variables en `nodes` y a la evidencia especificada con `setEvidence`. Por ejemplo, para calcular $P(S|T, E = high)$ ejecutamos:

```
> querygrain(jedu, nodes = c("S", "T"), type = "conditional")
   T
S         car    train    other
  M 0.612557 0.612557 0.612557
  F 0.387443 0.387443 0.387443
```

Los resultados obtenidos en esta consulta muestran que todas las probabilidades

$$P(S = M|T = t, E = high), \quad t \in \{car, train, other\}$$

son idénticas, independientemente del valor de T. Lo mismo ocurre cuando S es igual a F. En otras palabras, esto nos indica que:

$$P(S = M|T = t, E = high) = P(S = M|E = high)$$

y

$$P(S = F|T = t, E = high) = P(S = F|E = high)$$

Esto sugiere que S es independiente de T condicionado al valor de E; conocer el sexo de una persona no es informativo sobre sus preferencias en el transporte si conocemos su nivel educativo. Esta conclusión podemos también extraerla con el criterio de la d-separación, puesto que S y T están d-separados por E.

```
> dsep(bn, x = "S", y = "T", z = "E")
[1] TRUE
```

9.3 Inferencia aproximada

El paquete **gRain** tiene implementados algoritmos aproximados de inferencia por Monte Carlo para redes bayesianas. Los algoritmos que tiene implementados son el de *muestreo lógico* (rejection sampling, logic sampling o muestreo por rechazo) y el de *ponderación por verosimilitud* (likelihood weighting). Para usar el algoritmo de *muestreo lógico* usaremos la función `cpquery`. Por ejemplo, para calcular la distribución conjunta a posteriori para $S = M$ y $T = car$ ejecutamos:

```
> cpquery(bn, event = (S == "M") & (T == "car"),
+         evidence = (E == "high"))
[1] 0.351592
```

Podemos ver que esta probabilidad es algo distinta a la exacta que obtuvimos anteriormente con `querygrain` (0.3426644). La calidad de la aproximación puede mejorarse usando el argumento `n` en `cpquery` para incrementar el número de muestras aleatorias que utilizamos. El valor por defecto es $5000 \cdot nparams(bn)$. Lo pondremos a un millón:

```
> cpquery(bn, event = (S == "M") & (T == "car"),
+         evidence = (E == "high"), n = 10^6)
[1] 0.343198
```

Podemos observar que el tiempo empleado es mayor que en el caso anterior. Como se vió en la clase de teoría, el muestreo lógico da malas estimaciones cuando la probabilidad de la evidencia tiene baja probabilidad, ya que este método rechaza las muestras que no son consistentes con la evidencia.

Una aproximación que suele funcionar mejor es el método de *ponderación por verosimilitud*, que funciona de forma que ninguna muestra es rechazada. Puede usarse con la función `cpquery` usando `method="lw"`. El método por defecto es el muestreo lógico (`method="ls"`). Por ejemplo, para la consulta anterior de $P(S = M, T = car|E = high)$ ejecutamos:

```
> cpquery(bn, event = (S == "M") & (T == "car"),
+         evidence = list(E = "high"), method = "lw")
[1] 0.3422035
```

Podemos ver que el valor obtenido 0.3422035 es más cercano al exacto 0.3426644 que el obtenido que el muestreo lógico (0.343198) sin tener que generar 10^6 muestras aleatorias. En el anterior comando, podemos observar que la evidencia `evidence` se proporciona al algoritmo de ponderación por similitud como una lista de valores, una para cada variable de condicionamiento.

El algoritmo para muestreo lógico implementado en **gRain** permite hacer consultas más complejas. Por ejemplo la probabilidad de que un hombre viaje en coche dado que su edad es `young` (joven) y su nivel educativo es `uni` (universitario) o bien él es un `adult` (adulto) independientemente de su nivel educativo:

$$P(S = M, T = car|\{A = young, E = uni\} \cup \{A = adult\})$$

Esta consulta la haríamos con:

```
> cpquery(bn, event = (S == "M") & (T == "car"),
+         evidence = ((A == "young") & (E == "uni")) | (A == "adult"))
[1] 0.3336759
```

El algoritmo de ponderación por verosimilitud no está implementado para permitir hacer el anterior tipo de consulta.

La función `cpdist` del paquete **gRain** devuelve un *data frame* que contiene un conjunto de muestras consistentes con la evidencia (`evidence`) para las variables indicadas en `nodes`. Por ejemplo:

```
> SxT <- cpdist(bn, nodes = c("S", "T"),
+               evidence = (E == "high"))
> head(SxT)
  S     T
1 M   car
```

```
2 M train
3 M   car
4 M   car
5 M train
6 F other
```

El data frame obtenido puede usarse ahora, por ejemplo, para obtener la probabilidad conjunta a posteriori de S y T ($P(S,T|E = high)$) usando la función `prop.table` de **R**:

```
> prop.table(table(SxT))
   T
S          car      train      other
  M 0.35464961 0.16653344 0.09805489
  F 0.21209699 0.10285105 0.06581402
```

La anterior consulta nos permite también obtener la *configuración más probable* para S y para T dada la evidencia $E = high$. De nuevo, la respuesta es que entre la gente con nivel de educación alto ($high$) la combinación más frecuente para S y T es *hombre* (M) y *coche* (car).

Ejercicio 1 *Como trabajo obligatorio de esta parte de la asignatura, se debe entregar un script en R para construir la red bayesiana (DAG y tablas de probabilidad) usada en esta práctica. El script debe también hacer tres consultas de la probabilidad a posteriori de alguna variable dada cierta evidencia, usando el método exacto y los dos aproximados.*

Como trabajo opcional de esta parte de la asignatura, se debe entregar un script en R para construir una red bayesiana de tamaño pequeño (mínimo 7 nodos). Podría entregarse la misma red que se ha hecho para el trabajo a entregar a Serafín Moral. El script debe hacer lo siguiente:

- *Construir el DAG de la red bayesiana.*
- *Crear las tablas de probabilidad.*
- *Hacer al menos tres consultas de la probabilidad a posteriori dada cierta evidencia, de alguna variable, usando el método exacto y los dos métodos aproximados.*
- *Comprobar la d-separación en unos tres casos.*

Debe entregarse también un pequeño informe con la descripción del problema, y la descripción de la red (parte cualitativa y cuantitativa).