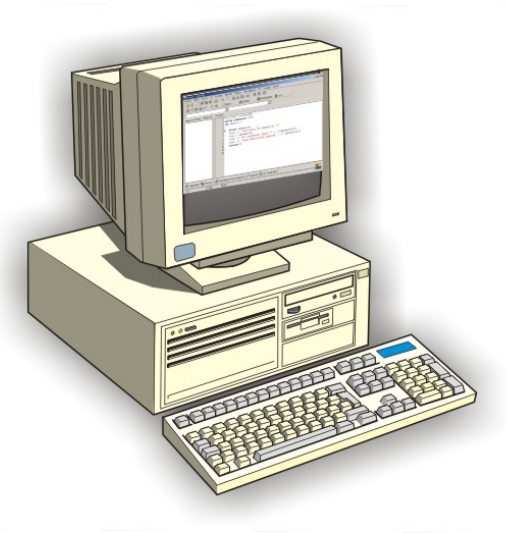

Práctica 2

Aprendizaje de redes bayesianas con R

Febrero de 2022

DECSAI

Modelos Gráficos Probabilísticos

Febrero de 2021

Índice

1. Estimación de los parámetros: distribuciones de probabilidad condicional 3

2. Test de Independencia 6

3. Scores 7

4. Aprendiendo el DAG: Tests y Scores 8

5. Variables Continuas 10

6. Ejercicio Básico 13

7. Ejercicio a Entregar 14

1. Estimación de los parámetros: distribuciones de probabilidad condicional

Para empezar cargamos los paquetes:

```
> library(lattice)
> library(gridExtra)
> library(gRain)
> library(Rgraphviz)
> library(bnlearn)
> dag <- empty.graph(nodes = c("A", "S", "E", "O", "R", "T"))
> dag <- set.arc(dag, from = "A", to = "E")
> dag <- set.arc(dag, from = "S", to = "E")
> dag <- set.arc(dag, from = "E", to = "O")
> dag <- set.arc(dag, from = "E", to = "R")
> dag <- set.arc(dag, from = "O", to = "T")
> dag <- set.arc(dag, from = "R", to = "T")
> dag
```

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, colClasses = "factor")
```

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
```

2. Test de Independencia

La función `ci.test` implementa el test G^2 (basado en la medida de información y el test clásico χ^2 que puede ser considerado una aproximación del anterior). Para el G^2 hay que poner `test="mi"` como argumento. Para el χ^2 hay que poner `test="c2"`,

```
> ci.test("T", "E", c("O", "R"), test = "mi", data = survey)

        Mutual Information (disc.)

data:  T ~ E | O + R
mi = 9.8836, df = 8, p-value = 0.2733
alternative hypothesis: true value is greater than 0

> ci.test("T", "E", c("O", "R"), test = "x2", data = survey)

        Pearson's X^2

data:  T ~ E | O + R
x2 = 8.2375, df = 8, p-value = 0.4106
alternative hypothesis: true value is greater than 0
```

Ambos tests resultan en valores muy altos para el p-valor por lo que el enlace $E \rightarrow T$ no es significativo en la estructura de la red.

También podemos contrastar si un enlace existente en un grafo está soportado por los datos, por ejemplo en este caso, el enlace $O \rightarrow T$:

```
> ci.test("T", "O", "R", test = "x2", data = survey)
```

```
        Pearson's X^2

data:  T ~ O | R
x2 = 3.7988, df = 4, p-value = 0.4339
alternative hypothesis: true value is greater than 0
```

De nuevo si existe un enlace de R a T, el resultado nos dice que no tiene sentido añadir uno de O a T.

Se pueden comprobar la fuerza de todos los enlaces con una sola instrucción, que nos dice para todos los enlaces el p-valor de un test de independencia del hijo respecto al padres, condicionado al resto de los padres de un grafo:

```
> arc.strength(dag, data = survey, criterion = "x2")
  from to     strength
1    A  E 0.0009777168
2    S  E 0.0012537013
3    E  O 0.0026379469
4    E  R 0.0005599201
5    O  T 0.4339127237
6    R  T 0.0013584250
```

Obtenemos que todos los enlaces de `dag` están soportados por los datos, excepto el de O a T. En `criterion` se puede especificar también un score en lugar de un test estadístico.

3. Scores

Los scores permiten medir cómo de bueno es una red respecto a un conjunto de datos. En este caso podemos ver cómo de buena es la red `dag` respecto a los datos `survey` usando el score `BIC` o `BDe` con `iss=10`:

```
> score(dag, data = survey, type = "bic")
[1] -2012.687
> score(dag, data = survey, type = "bde", iss = 10)
[1] -1998.284
```

En valores pequeños de `iss`. `BDe` y `BIC` suelen dar resultados similares:

```
> score(dag, data = survey, type = "bde", iss = 1)
[1] -2015.647
```

Usando un score podemos comparar dos redes para comprobar cual es mejor. Podemos comprobart si añadir $E \rightarrow T$ aumenta el score:

```
> dag4 <- set.arc(dag, from = "E", to = "T")
> nparams(dag4, survey)
[1] 29
> score(dag4, data = survey, type = "bic")
[1] -2032.603
```

El score empeora por lo que no es una buena idea añadir este enlace. Como vemos, estos scores penalizan la complejidad de los modelos, aunque puedan ajustar mejor los datos.

Con los scores podemos comparar dos grafos cualesquiera. Por ejemplo, generar un grafo aleatorio y ver si es mejor que el actual (no debería):

```
> rnd <- random.graph(nodes = c("A", "S", "E", "O", "R", "T"))
> modelstring(rnd)
[1] "[A][O][T][S|A][E|A][R|A:S:E:O]"
> score(rnd, data = survey, type = "bic")
[1] -2067.097
```

4. Aprendiendo el DAG: Tests y Scores

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")
```

Podemos comprobar que el `BIC` no recomienda quitar ningún arco de la red aprendida con la función `arcstrength` que nos dice cuanto aumenta el score si quitamos cada uno de los arcos de una red:

```
> arc.strength(learned, data = survey, criterion = "bic")
  from to   strength
1    R  E -3.3896261
2    E  S -2.7260640
3    R  T -1.8484171
4    E  A -1.7195441
5    E  O -0.8266937
```

Como se ve, quitar cualquiera de los arcos, empeora la calidad del score, como cabía esperar.

Esto no es cierto para `dag` sugiriendo que los datos no son sufientes para aprender bien la red correcta.

```
> arc.strength(dag, data = survey, criterion = "bic")
  from to   strength
1    A  E  2.4889383
2    S  E  1.4824183
3    E  O -0.8266937
4    E  R -3.3896261
5    O  T 10.0457874
6    R  T  2.9734338
```

Los algoritmos basados en restricciones están implementados con las funciones `gs`, `iamb`, `fast.iamb`, `inter.iamb`, `mmpc`, `si.hiton.pc` y `hpc`.

Se pueden probar a aprender redes con estos algoritmos:

```
> learned3 <- iamb(survey)
> cpdag(learned3)

  Bayesian network learned via Constraint-based methods

  model:
    [undirected graph]
  nodes:                                6
  arcs:                                 4
    undirected arcs:                    4
    directed arcs:                      0
  average markov blanket size:          1.33
  average neighbourhood size:           1.33
  average branching factor:             0.00

  learning algorithm:                   IAMB
  conditional independence test:        Mutual Information (disc.)
  alpha threshold:                      0.05
  tests used in the learning procedure: 85

> vstructs(learned3)
     X Z Y
> learned4= cextend(learned3)
> modelstring(learned4)
[1] "[A][E][R|E][O|E][S|E][T|R]"
```

En general, estos algoritmos orientan los grafos de forma parcial y se pueden extender de forma consistente con la función `cextend`.

Si llamamos a los algoritmos de aprendizaje con la opción `debug=TRUE` dan más información sobre la evolución de los algoritmos. La función `compare` sirve para comparar dos redes bayesianas proporcionando los verdaderos positivos, los falsos positivos y los falsos negativos. La función `shd` da un valor numérico de los errores, mientras que la función `hamming` da un valor numérico de la diferencia entre las clases de equivalencia (no tiene en cuenta los arcos que permiten otra orientación sin cambiar las independencias del problema).

```
> compare(dag,learned)
$tp
[1] 2

$fp
[1] 3

$fn
[1] 4

> shd(learned,dag)
[1] 6
> hamming(learned,dag)
[1] 1
```

Se puede aprender empezando con cualquier red, incluso con una red aleatoria.

```
 > learned5 = hc(survey, score = "bic", start = random.graph(names(survey)))
> modelstring(learned5)
[1] "[R][E|R][T|R][A|E][O|E][S|E]"
```

Los algoritmos híbridos de búsqueda tienen una parte de restricción y otra de búsqueda. La función básica es `rsmax2` Cuando el algoritmo de búsqueda es `hc` y el de restricciones es `mmpc` el algoritmo se puede llamar como `mmhc`:

```
 > l6 = mmhc(survey)
> ## ----hybrid-learning-general----------------------------------------------
> l7= rsmax2(survey, restrict = "mmpc", maximize = "hc")
> compare(l6,l7)
$tp
[1] 4

$fp
[1] 0

$fn
[1] 0

> ## ----hybrid-learning-with-arguments---------------------------------------
> l8= rsmax2(survey, restrict = "si.hiton.pc", maximize = "tabu",
+   restrict.args = list(test = "x2"),
+   maximize.args = list(score = "bde", iss = 1))
> plot(l8)
```

MMHC está implementado en `bnlearn` con la función `mmhc`, que es equivalente a `rsmax2` con `mmpc` como restricción y `hc` como búsqueda. Realiza una única iteración del método híbrido.

5. Variables Continuas

Primero establecemos el grafo continuo `crop`

```
 > dag.bnlearn <- model2network("[G][E][V|G:E][N|V][W|V][C|N:W]")
> dag.bnlearn

  Random/Generated Bayesian network

  model:
   [E][G][V|E:G][N|V][W|V][C|N:W]
  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

> crop.nodes <- nodes(dag.bnlearn)
> E.dist <- list(coef = c("(Intercept)" = 50), sd = 10)
> G.dist <- list(coef = c("(Intercept)" = 50), sd = 10)
> V.dist <- list(coef = c("(Intercept)" = -10.35534,
+                E = 0.70711, G = 0.5), sd = 5)
> N.dist <- list(coef = c("(Intercept)" = 45, V = 0.1), sd = 9.949874)
> W.dist <- list(coef = c("(Intercept)" = 15, V = 0.7), sd = 7.141428)
> C.dist <- list(coef = c("(Intercept)" = 0, N = 0.3, W = 0.7), sd = 6.25)
> dist.list = list(E = E.dist, G = G.dist, V = V.dist,
+                  N = N.dist, W = W.dist, C = C.dist)
> gbn.bnlearn <- custom.fit(dag.bnlearn, dist = dist.list)
```

Ahora generamos unos datos aleatorios a partir de la red para aprender:

```
> set.seed(4567)
> cropdata200 <- rbn(gbn.bnlearn, n = 200)
> set.seed(1234)
> cropdata20k <- rbn(gbn.bnlearn, n = 20000)
```

Ahora se pueden aprender los parámetros:

```
> dim(cropdata200)
[1] 200   6
> round(head(cropdata200), 2)
      C     E     G     N     V     W
1 48.83 51.48 42.64 54.10 42.96 41.96
2 48.85 73.43 40.97 60.07 65.29 48.96
3 67.01 71.10 52.52 51.64 63.22 62.03
4 37.83 49.33 56.15 49.01 47.75 38.77
5 55.30 49.27 63.55 54.62 60.57 56.66
6 56.12 48.72 66.02 43.95 55.54 52.39

> crop.fitted <- bn.fit(dag.bnlearn, data = cropdata200)
> crop.fitted$E

  Parameters of node E (Gaussian distribution)

Conditional density: E
```

```
Coefficients:
(Intercept)
   50.80161
Standard deviation of the residuals: 10.7432

> crop.fitted$C

  Parameters of node C (Gaussian distribution)

Conditional density: C | N + W
Coefficients:
(Intercept)             N             W
  2.4026177     0.2734476     0.6858202
Standard deviation of the residuals: 6.31327
```

Se hace por máxima verosimilitud, pero se pueden usar otros métodos de forma indirecta.

Se puede aprender la estructura con métodos de restricciones o score. Para aprender con el método `iamb`

```
> pdag1 <- iamb(cropdata200, test = "cor")
> modelstring(pdag1)
[1] "[E][G][N][V|E:G][W|V][C|N:W]"
> plot(pdag1)
> compare(dag.bnlearn,pdag1)
$tp
[1] 5

$fp
[1] 0

$fn
[1] 1
```

Aunque hay muy pocos datos, se aprende de forma casi correcta.

Al aprender redes (discretas o continuas) se puede especificar que algunos enlaces tienen que estar presentes (`whitelist`), mientras que otros están prohibidos (`blacklist`);

```
> wl <- matrix(c("V", "N"), ncol = 2)
> wl
     [,1] [,2]
[1,] "V"  "N"
> pdag2 <- iamb(cropdata200, test = "cor", whitelist = wl)
> all.equal(dag.bnlearn, pdag2)
[1] TRUE
> compare(dag.bnlearn, pdag2)
$tp
[1] 6

$fp
[1] 0

$fn
[1] 0
```

Y ya las dos redes son iguales. También podemos aprender con más datos:

```
> dim(cropdata20k)
[1] 20000     6
> pdag3 <- iamb(cropdata20k, test = "cor")
> all.equal(dag.bnlearn, pdag3)
[1] TRUE
> compare(dag.bnlearn, pdag3)
$tp
[1] 6

$fp
[1] 0

$fn
[1] 0
```

Y ya se aprende la red sin necesidad de forzar ningún arco.

También se pueden utilizar scores BIC (`bic-g`) o bayesiano (`bge`). Este último se calcula suponiendo unas distribuciones 'a priori' para las medias y varianzas.

```
> score(dag.bnlearn, data = cropdata20k, type = "bic-g")
[1] -416421.2
> score(dag.bnlearn, data = cropdata20k, type = "bge")
[1] -416494.5
>
```

El bge admite dos parámetros `iss.mu` y `iss.w`, que juegan el mismo papel que el `iss` para la Dirhlet, pero ahora para las medias y varianzas.

Si aprendemos una red con hill climbling los resultados son similares a usar IAMB.

```
> pdag4 <- hc(cropdata200, score = "bge")
> compare(dag.bnlearn, pdag4)
$tp
[1] 5

$fp
[1] 0

$fn
[1] 1
```

6. Ejercicio Básico

- Crear una red con tres nodos A, B, C, añadir enlaces de A y B a C y establecer distribuciones condicionadas manualmente:

```
 library(bnlearn)
> cptA = matrix(c(0.4, 0.6), ncol = 2, dimnames = list(NULL, c("LOW", "HIGH")))

> cptB = matrix(c(0.8, 0.2), ncol = 2, dimnames = list(NULL, c("GOOD", "BAD")))

> cptC = c(0.5, 0.5, 0.4, 0.6, 0.3, 0.7, 0.2, 0.8)
```

```
> dim(cptC) = c(2, 2, 2)
> dimnames(cptC) = list("C" = c("TRUE", "FALSE"), "A" =  c("LOW", "HIGH"),
+                       "B" = c("GOOD", "BAD"))
> net = model2network("[A][B][C|A:B]")
> dfit = custom.fit(net, dist = list(A = cptA, B = cptB, C = cptC))
```

- Simular una muestra de tamaño 5000 a partir de la red.
- Obtener las redes aprendidas con distintos algoritmos

El objetivo de este ejercicio es comprobar empíricamente que la causalidad se puede aprender a partir de datos.

7. Ejercicio a Entregar

Como trabajo obligatorio de esta parte de la asignatura, se debe entregar un script en R y un documento que haga lo siguiente:

- Seleccione una red bayesiana con al menos 7 variables. No tiene que ser una red de nueva creación. Puede ser una usada en otras partes de la asignatura o del repositorio https://www.bnlearn.com/bnrepository/
- Simular dos conjuntos de datos de distintos tamaños a partir de la red (por ejemplo uno con 200 casos y otro con 5000 casos. Esto se hace con la instrucción

  `rbn(x, n = 1, data, fit, ..., debug = FALSE)`
- Aprender la estructura con dos métodos distintos, uno basado en test de independencia y otro en scores, y con los dos conjuntos de datos,
- Comparar la estructura de las redes obtenidas con las originales. Comentar las diferencias
- Aprender los parámetros de las redes

Debe de entregarse el fichero R junto con un informe en pdf del trabajo realizado.