Aprendizaje Automático

Modelos Gráficos Probabilísticos

Vicente Bosch, Enrique Vidal, Francisco Casacuberta
Departamento de Sistemas Informáticos y Computación
Universidad Politécnica de Valencia

Curso 2016-2017

1. Introducción

El objetivo de esta práctica es experimentar con Modelos Gráficos Probabiísticos, también conocidos como Redes Bayesianas. Para ello utilizaremos el *toolkit* "BNT" para MATLAB, desarrollado por Kevin Murphy[1]. Este *toolkit* se puede descargar de:
`https://code.google.com/p/bnt`

Puede instalarse fácilmente siguiendo las instrucciones que se encuentran en esta página web. Una instalación ya preparada para su uso en esta práctica, está disponible en:
`~/asigDSIC/ETSINF/apr/p2/BNT`

La implementación de BNT se basa en el uso de matrices multidimensionales, que no están completamente soportadas por OCTAVE. Por tanto, a diferencia de las anteriores prácticas, en esta usaremos MATLAB, cuya sintaxis y modo de operación son muy similares (o casi idénticos) a OCTAVE. Con la invocación MATLAB se obtiene una interfaz de usuario gráfica completa. No obstante, a los efectos de esta práctica, basta con utilizar la interfaz textual, que es más simple y se obtiene mediante "`matlab -nodesktop`".

Una vez inizializado el entorno, deberemos de añadir al `path` de MATLAB el directorio donde se encuentra la instalación mediante las siguientes instrucciones.

```
addpath('~/asigDSIC/ETSINF/apr/p2/BNT')
addpath(genpathKPM('~/asigDSIC/ETSINF/apr/p2/BNT'))
```

Estas instrucciones producen un *warning* que puede ignorarse. De esta forma se evita tener que copiar la instalación completa de BNT (que es muy grande) a nuestro `home`. *Hay que realizar este paso cada vez que se arranque un nuevo entorno de trabajo.*

[1] Kevin Murphy. "The Bayes net toolbox for MATLAB." Computing science and statistics 33.2 (2001): 1024-1034. `http://people.cs.ubc.ca/~murphyk/Papers/bnt.pdf`

2. Ejemplo simple de uso de BNT: *"Sprinkler"*

Para introducir la funcionalidad básica de BNT en MATLAB nos basaremos en un ejemplo simple adaptado del libro de Russell y Norvig, "Artificial Intelligence: a Modern Approach", Prentice Hall, 1995, p454:

C	$P(C)$	$P(S \mid C), S =$ false	$P(S \mid C), S =$ true	$P(R \mid C), R =$ false	$P(R \mid C), R =$ true
false	0.5	0.5	0.5	0.8	0.2
true	0.5	0.9	0.1	0.2	0.8

S	R	$P(W \mid S,R), W =$ false	$P(W \mid S,R), W =$ true
false	false	1.0	0.0
true	false	0.1	0.9
false	true	0.1	0.9
true	true	0.01	0.99

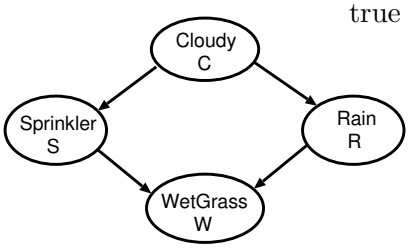

Figura 1: Red bayesiana simple para el ejemplo *"Sprinkler"*

Grafo Acíclico Dirigido

Empezaremos por definir el grafo acíclico dirigido (DAG) del ejemplo mediante una matriz de adyacencia. Para definir dicha matriz hemos numerado topológicamente los nodos; es decir, los antecesores han de estar antes que los descendientes (es imprescindible seguir este orden):

1. Nublado (Cloudy, C)
2. Aspersor (Sprinkler, S)
3. Llueve (Rain, R)
4. Cesped húmedo (WetGrass, W)

La matriz de adyacencia del ejemplo *Sprinkler* se define en BNT como:

```
N = 4;  C = 1; S = 2; R = 3; W = 4;
grafo          = zeros(N, N);
grafo(C,[R S]) = 1;
grafo(R,W)     = 1;
grafo(S,W)     = 1;
```

El grafo definido por esta matriz de adyacencia puede visualizarse mediante `draw_graph(grafo);` aunque en algunas versiones de MATLAB esta invocación suele producir errores.

Además hay que especificar la *variable aleatoria* aociada a cada nodo; es decir, su tipo y tamaño. Si es discreta, el tamaño del nodo es el número total de posibles valores que puede tomar la variable. Por otra parte si es de tipo real/continuo, puede ser un vector y el tamaño es la dimensión de dicho vector. En el ejemplo *Sprinkler* todos los nodos son binarios y por tanto discretos:

```
nodosDiscretos = 1:N;
tallaNodos     = 2*ones(1,N);
```

Si los nodos fueran de tamaños diferentes, se especificaría un tamaño distinto para cada nodo, por ejemplo, de la siguiente forma: `tallaNodos = [4 2 3 5];` Finalmente, se crea la red bayesiana uniendo el DAG y la especificación de los nodos:

```
redB = mk_bnet(grafo, tallaNodos,  'discrete', nodosDiscretos);
```

Conviene notar que, para usar cualquiera de los parámetros opcionales que permite una función, solo hay que añadir a la llamada el par formado por el *valor* y el *nombre* , como hemos visto para `mk_bnet`.

2.1. Probabilidades Condicionales

Para completar la especificación de una red bayesiana, aparte de definir el DAG y los nodos correspondientes, hay que indicar las Distribuciones de Probabilidad Condicional (*Conditional Probability Distribution*, CPD) asociadas a cada nodo. Para un nodo cuya variable aleatoria asociada es Z, su CPD especifica la probabilidad condicional $P(Z \mid \text{pred})$, donde "pred" es el conjunto de variables asociadas a los nodos antecesores. La CPD más simple es una Tabla, que abreviaremos como TPC. Esto es adecuado para nodos de naturaleza discreta, como es el caso en este ejemplo.

Es importante notar que los valores de una variable aleatoria discreta no están ordenados de ninguna forma en particular; es decir, son valores categóricos y *no* ordinales.

Las dimensiones de cada TPC deben de estar ordenadas según el orden definido en los nodos del DAG. Si un nodo no tiene antecesores, su TPC es un vector columna con la probabilidad para cada valor de ese nodo. Por ejemplo, la TPC para el nodo 4 (WetGrass, cesped húmedo) de la figura**??**, con los nodos en el orden definido es:

S	R	W	$P(W \mid S, R)$
false	false	false	1.0
true	false	false	0.1
false	true	false	0.1
true	true	false	0.01
false	false	true	0.0
true	false	true	0.9
false	true	true	0.9
true	true	true	0.99

Para especificar la TPC de un nodo, en MATLAB podemos definir cada uno de los valores de la matriz uno a uno; por ejemplo:

```
TPC_W        = zeros(2, 2, 2);
TPC_W(1,1,1) = 1.0;
TPC_W(2,1,1) = 0.1;
TPC_W(1,2,1) = 0.1;
TPC_W(2,2,1) = 0.01;
...
```

Pero resulta más sencillo y eficaz utilizar el constructor de TPC proporcionado por BNT:

```
redB.CPD{W} = tabular_CPD(redB, W, [1.0 0.1 0.1 0.01 0.0 0.9 0.9 0.99]);
```

Y de forma similar para los demás nodos:

```
redB.CPD{C} = tabular_CPD(redB, C, [0.5 0.5]);
redB.CPD{S} = tabular_CPD(redB, S, [0.5 0.9 0.5 0.1]);
redB.CPD{R} = tabular_CPD(redB, R, [0.8 0.2 0.2 0.8]);
```

2.2. Inferencia

Una vez creada la red bayesiana, podemos usarla para inferir cualquier distribución de probabilidad sobre las variables aleatorias definidas por la red. Hay diversos algoritmos de inferencia para redes bayesianas, con distintos compromisos entre velocidad, complejidad, generalidad y precisión. `BNT` ofrece varios de estos algoritmos, pero en esta práctica solo usaremos el motor de inferencia denominado *junction tree engine*:

```
motor = jtree_inf_engine(redB);
```

Otros motores de inferencias tienen constructores muy similiar a este, pero pueden requerir parametros adicionales. Una vez definida una instancia de un motor de inferencia, sea cual sea, se usan todos de la misma forma.

Distribuciones Marginales

Supongamos que queremos calcular la probabilidad de que el aspersor esté en marcha ($S = \text{true} \equiv 2$) si la hierba está mojada ($W = \text{true} \equiv 2$). La evidencia es pues $W = 2$, lo que se especifica de la siguiente forma:

```
evidencia    = cell(1, N);
evidencia{W} = 2;
```

Usamos un *cell array* para poder acomodar diferentes tipos de evidencia o tamaño para cada nodo. `BNT` entiende el valor `[]` para denotar que un nodo no tiene evidencia. Una vez creada, se inserta la evidencia en un motor de inferencia de la siguiente forma:

```
[motor, logVerosim] = enter_evidence(motor, evidencia);
```

Esta función devuelve el motor modificado con la evidencia incorporada y la log-verosimilitud de dicha evidencia. Finalmente podemos calcular $P(S = 2 \mid W = 2)$ como sigue:

```
m = marginal_nodes(motor, S);
m.T
ans =
      0.57024
      0.42976
```

Es decir, $P(S = 2 \mid W = 2) = 0{,}42976$. Si ahora se añade la evidencia de que está lloviendo ($R = \text{true} \equiv 2$) podemos ver que esto cambia el resultado:

```
evidencia{R}        = 2;
[motor, logVerosim] = enter_evidence(motor, evidencia);
m                   = marginal_nodes(motor, S);
m.T
ans =
      0.8055
      0.1945
```

En este caso hemos calculado $P(S = 2 \mid W = 2, R = 2) = 0{,}1945$. Este valor es menor que el anterior ya que el la lluvia explica que el cesped esté húmedo y ya no es "tan necesario" que aspersor esté encendido para explicar la humedad del cesped.

Nodos Observados

¿Que ocurre si se calcula la probabilidad marginal de un nodo observado; por ejemplo, $P(W \mid W = 2)$? Con probabilidad 1.0, un nodo discreto que al que le hemos asignado una evidencia solo tiene un posible valor: justamente el que hemos asignado. Por eficiencia, `BNT` trata los nodos discretos observados como si solo tuvieran un valor y su pronbibilidad es 1.0. Esto puede verse en el siguiente ejemplo:

```
evidencia    = cell(1, N);
evidencia{W} = 2;
motor        = enter_evidence(motor, evidencia);
m            = marginal_nodes(motor, W);
m.T
ans =
    1.0000
```

Puede resultar un tanto confuso que solo se muestre una probabilidad, en vez de una para cada valor de la variable aleatoria asociada al nodo considerado. Si, por mayor claridad, se desea la respuesta completa hay que añadir un argumento adicional:

```
m = marginal_nodes(motor, W, 1);
m.T
ans =
         0
    1.0000
```

Así se obtienen de forma explícita las probabilidades de todos los valores; dos en este ejemplo: $P(W = 1 \mid W = 2) = 0$ y $P(W = 2 \mid W = 2) = 1$.

Distribuciones Conjuntas

En el siguiente ejemplo calcularemos la probabildad conjunta de un grupo de nodos: *Sprinkler*, *Rain* y *GrassWet* para el caso en el que no hay ningún nodo observado (es decir la evidencia está vacía):

```
evidencia     = cell(1,N);
[motor, ll] = enter_evidence(motor, evidencia);
m             = marginal_nodes(motor, [S R W]);
```

donde el resultado, `m`, es una estructura cuyo campo `T` es un vector multi-dimensional (en este caso de tres dimensiones) que contiene la distribución de probabilidad conjunta para los nodos especificados; es decir, $T(i, j, k) \equiv P(S = i, R = j, W = k \mid \text{evidencia})$.

```
m.T
ans(:,:,1) =
    0.2900    0.0410
    0.0210    0.0009
ans(:,:,2) =
         0    0.3690
    0.1890    0.0891
```

Podemos ver que $P(S = 1, R = 1, W = 2) = 0$. Esto tiene sentido puesto que en nuestro ejemplo es imposible que el césped esté húmedo ($W = 2$) cuando no llueve ($R = 1$) y el aspersor está parado ($S = 1$). Si ahora se asigna a la variable lluvia el valor `true` ($R = 2$):

```
evidencia{R} = 2;
[motor, ll] = enter_evidence(motor, evidencia);
m = marginal_nodes(motor, [S R W])
m =
    domain: [2 3 4]
         T: [2x1x2 double]
    ...
m.T
ans(:,:,1) =
    0.0820
    0.0018
ans(:,:,2) =
    0.7380
    0.1782
```

Las probabilidades $T(i, 2, k) \equiv P(S = i, R = 2, W = k \mid \text{evidencia})\ \forall i, k,$ son todas 0.0 (y no se muestran) ya que, si llueve ($R = 2$), la probabilidad conjunta para cualquier situación donde $R = 1$ es 0.0 (¡pues se sabe que llueve!). Ante esta situación `BNT` opta por no devolver (ni mostrar) todos estos ceros. Al igual que se ha visto anteriormente, pueden obtenerse todos los resultados de la siguiente forma:

```
m = marginal_nodes(motor, [S R W], 1)
m =
    domain: [2 3 4]
         T: [2x2x2 double]
    ...
m.T
ans(:,:,1) =
            0       0.0820
            0       0.0018
ans(:,:,2) =
            0       0.7380
            0       0.1782
```

Explicación más probable

Dada una evidencia, la función `calc_mpe` de `BNT` permite obtener la "explicación más probable" (es decir, el valor más probable de la variable aleatoria de cada nodo) y su log-probabilidad. Por ejemplo, si no se especifica ninguna evidencia:

```
evidencia = cell(1,N);
[explMaxProb, logVer] = calc_mpe(motor, evidencia)
explMaxProb =
    [2]    [1]    [2]    [2]
logVer =
   -1.127
```

Es decir, $C = 2$, $S = 1$, $R = 2$, $W = 2$. Al no haber ningún nodo observado (ninguna evidencia), `calc_mpe` obtiene una combinación de valores de todas las variables aleatorias de la red bayesiana, para la que la probabilidad conjunta es máxima. En este caso, según las tablas de probabilidad especificadas al principio de esta sección, la situación conjunta más probable es: que esté nublado, que el aspersor esté parado, que llueva y que el cesped esté húmedo. Es interesante observar que, si se especifica cualquier evidencia parcial que no contradiga la explicación más probable, se seguirá obteniendo el mismo resultado:

```
evidencia{W} = 2;
[explMaxProb, logVer] = calc_mpe(motor, evidencia)
explMaxProb =
    [2]    [1]    [2]    [2]
logVer =
   -1.127
```

2.3. Ejercicio Propuesto

En la transparencia 6.10 del tema 6 de la teoría sobre modelos gráficos probabilísticos podemos ver una sencilla red bayesiana para dianóstico de cáncer de pulmon. Se pide:

1. desarrollar un script MATLAB que implemente dicha red usando BNT
2. ¿Cuál es la probabilidad de que un paciente no fumador no tenga cáncer de pulmón si la radiografía ha dado un resultado negativo pero sufre disnea?
3. ¿Cuál es la explicación más probable de que un paciente sufra cáncer de pulmón?

3. Tareas de experimentación

Se describen a continuación dos tareas de aprendizaje, generación de muestras y clasificación, relacionadas con mixturas de gaussianas representadas mediante redes bayesianas.

3.1. Aprendizaje no-supervisado de una mixtura de gausianas

A continuación realizaremos un pequeño experimento de aprendizaje no-supervisado utilizando datos bidimensionales generados aleatoriamente mediante una mixtura de dos gaussianas. A partir de estos datos, trataremos de estimar los parámetros de una mixtura de dos gaussianas y comprobaremos visualmente hasta qué punto la mixtura aprendida aproxima a la distribución con la que se habían generado los datos de entrenamiento.

Una mixtura de distribuciones se puede modelar mediante una red bayesiana con dos nodos: uno discreto, $\mathcal{K}$, que representa cual de las K componentes de la mixtura es responsable la generación del dato observado y otro nodo, X, dependiente de $\mathcal{K}$, que representa la distribución de los datos observados.

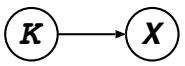

Figura 2: Red bayesiana que representa una mixtura de distribuciones de probabilidad.

La probabilidad conjunta es: $P(k, x) = p(k)\ p(x \mid k)$ y la probabilidad de observar un dato x se obtiene marginalizando en k; es decir:

$$P(x) = \sum_{k=1}^{K} P(k)P(x \mid k)$$

Si $P(x \mid k)$ se define como $\mathcal{N}(\vec{\mu}_k, \Sigma_k)$, esta es justamente la expresión de una mixtura de gussianas. Usando `BNT` vamos a definir una red bayesiana para $K = 2$:

```
grafo          = [ 0 1 ; 0 0];
nodosDiscretos = [1];
tallaNodos     = [2 2];
redB           = mk_bnet(grafo, tallaNodos, 'discrete', nodosDiscretos);
redB.CPD{1}    = tabular_CPD(redB, 1);
redB.CPD{2}    = gaussian_CPD(redB,2);
```

El nodo 1 es discreto y representa la distribución de los pesos de las $K = 2$ gaussianas que vamos a entrenar a partir de los datos. Conviene notar que al indicar que este nodo es de tamaño 2 estamos fijando el número de gaussianas que se van a aprender a 2. El segundo nodo también es de tamaño 2, pero en este caso es porque los datos que representa la población de entrada que son vectores de 2 dimensiones.

Para entrenar esta red bayesiana en primer lugar cargamos los datos de entrenamiento:

```
datosApr = load('./data/gaus2D/gauss2Dtr.data',   '-ascii');
```

Una vez definida la red bayesiana y cargados los datos (no etiquetados), procedemos a entrenarla mediante el algoritmo EM:

```
training                = cell(2, length(datosApr));
training(2,:)           = num2cell(datosApr', 1);
motor                   = jtree_inf_engine(redB);
maxIter                 = 15;
epsilon                 = 1e-100;
[redB2, logVer, motor2] = learn_params_em(motor, training, maxIter, epsilon);
EM iteration 1, ll = -972.7500
EM iteration 2, ll = -458.8179
...
EM iteration 15, ll = -381.1985
```

Es importante notar que *no* se ha indicado al algoritmo de aprendizaje (EM) a que clase corresponde cada uno de los datos de entrenamiento y por lo tanto estamos realizando un aprendizaje *no supervisado*. El EM en este caso estimará los parámetros de las dos distribuciones gausianas que con mayor verosimilitud expliquen los datos observados.

La red bayesiana entrenada, puede usarse para generar datos sintéticos. Se espera que estos datos se distribuyan de forma similar a los datos usados para entrenamiento. Para generar nuevos datos hay que fijar la evidencia el nodo $\mathcal{K}$ a uno de sus dos valores y pedir una muestra aleatoria:

```
K1 = cell(3,1); K2 = K1;
K1{1} = 1;      K2{1} = 2;
% Generar 75 muestras de cada clase
NM = 75
for i=1:NM
  muestra1         = sample_bnet(redB2, 'evidence', K1);
  muestra2         = sample_bnet(redB2, 'evidence', K2);
  muestras(i,:)    = muestra1{2}';
  muestras(i+NM,:) = muestra2{2}';
end
```

Se pueden comparar visualmente las muestras de entrenamiento con las generadas por nuestro modelo para ver si se ha aprendido de forma correcta la distribución original.

```
figure
subplot(2,1,1);
etiqsApr = load('./data/gaus2D/gauss2Dtr.labels', '-ascii');
plot(datosApr(etiqsApr==1,1), datosApr(etiqsApr==1,2), 'x', ...
     datosApr(etiqsApr==2,1), datosApr(etiqsApr==2,2), 'o');
axis([-4 5 -4 4])
subplot(2,1,2);
plot(muestras(1:NM,1),    muestras(1:NM,2),    'x', ...
     muestras(NM:2*NM,1), muestras(NM:2*NM,2), 'o');
axis([-4 5 -4 4])
```

En la Fig **??** podemos ver dicha comparación visual. Las muestras generadas de forma aleatoria son muy similares a las muestras originales de entrenamiento. En nuestro caso el generador ha asignado por casualidad las etiquetas de la misma forma que nuestras muestras de entrenamiento, pero es importante que esto no tiene por que ser asi puesto que no le hemos dado dicha información.

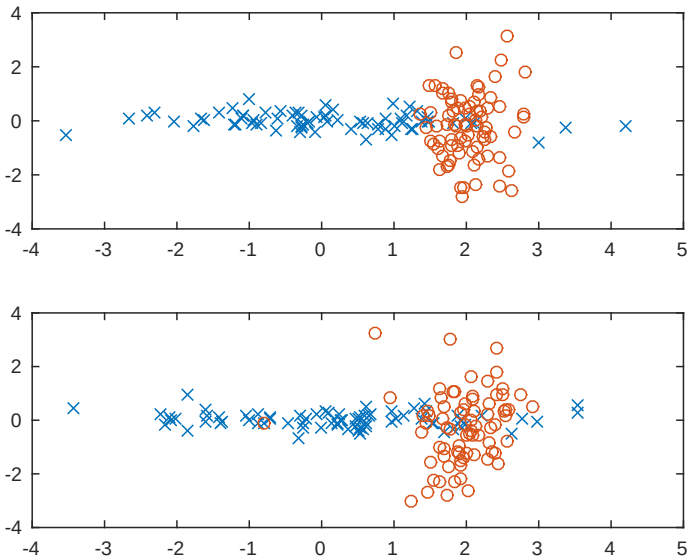

Figura 3: Figura resultante de los plots de las poblaciones de entrenamiento y de las muestras generadas.

3.2. Classificación con mixtura de gausianas

En el ejemplo de la sección anterior creamos una simple red bayesiana para aprende de forma no-supervisada distribuciones gaussianas a partir de de datos no etiquetados. Ese ejemplo muestra la potencia y versatilidad de las redes bayesianas. Aprovechando esta versatilidad, a continuación crearemos un clasificador basado en mixturas de gausianas, según el modelo gráfico de la Figura **??**.

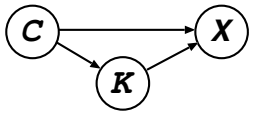

Figura 4: Red bayesiana para clasificación basda en mixturas de distribuciones.

La distribución conjunta que define a esta red es $P(c, k, x) = P(c)\, P(k \mid c)\, P(x \mid k, c)$. La probabilidad a posteriori de clase se obtiene marginalizando en k; es decir:

$$P(c \mid x) \;=\; \frac{P(c)P(x \mid c)}{P(x)} \;=\; \alpha\, P(c) \sum_{k=1}^{K} P(x, k \mid c) \;=\; \alpha\, P(c) \sum_{k=1}^{K} P(k \mid c)\, P(x \mid k, c) \quad (1)$$

donde $\alpha = 1/P(x)$ es una constante que no depende de c y puede ignorarse para clasificación. Así pues, la red de la Figura **??** define de forma efectiva un clasificador basado en mixtura de gausianas. Para crear esta red hay que definir sus tres nodos:

- Un nodo discreto asociado a la variable $\mathcal{C}$ que representa las probabilidades a priori de las clases
- Un nodo discreto asociado a la variable latente $\mathcal{K}$ que representa la distribución de probabilidad ("pesos") de las K componentes de la mixtura de cada clase
- Un nodo gausiano X que representa la distribución de los datos observables.

Seguiremos este esquema para implementar un clasificador para la tarea "SPAM". Los datos de esta tarea son vectores 58-dimensionales que representan mensajes de correo electrónico representados por la frecuencia de uso de 58 palabres preseleccionadas. Estas frecuencias permiten discriminar entre "mensajes basura" (SPAM) y mensajes normales (NO-SPAM).

Usaremos datos de este tipo, supervisados con las correspondientes etiquetas de clase, para entrenar la red mediante aprendizaje supervisado. Para ello empezamos cargando los datos supervisados de entrenamiento del corpus SPAM y determinamos el número de datos leidos, su dimensionalidad y el número real de clases:

```
datApr       = load('./data/spam/tr.dat',       '-ascii');
etqApr       = load('./data/spam/trlabels.dat', '-ascii');
etiqApr      = etqApr + 1;
[numVec dim] = size(datApr)
numClas      = max(etiqApr)
```

En el corpus SPAM la pertenencia a la clase SPAM o NO-SPAM se representa con un 1 o un 0, respectivamente, pero `BNT` asume que las etiquetas de clase son números naturales. La tercera instrucción del código anterior sirve para reetiquetar las clases.

Para implementar un clasificador basado en mixturas de gaussianas para tarea SPAM, la red bayesiana se define como sigue:

```
numGaus        = 2
grafo          = [ 0 1 1 ; 0 0 1 ; 0 0 0 ];
numNodos       = length(grafo)
tallaNodos     = [numClas numGaus dim];
nodosDiscretos = [1 2];
redB           = mk_bnet(grafo, tallaNodos, 'discrete', nodosDiscretos);
redB.CPD{1}    = tabular_CPD(redB, 1);
redB.CPD{2}    = tabular_CPD(redB, 2);
redB.CPD{3}    = gaussian_CPD(redB, 3, 'cov_type', 'diag');
```

El tamaño del nodo gausiano se ha definido para acomodar adecuadamente la dimensionalidad (`dim`) de los datos leidos. Por otra parte, el tamaño del nodo $\mathcal{K}$, representa el número de gausianas por mixtura que debe tener el modelo (`numGaus`). Para empezar se ha definido `numGaus=2`, por lo que acabaremos con dos mixturas de gausianas de dos gausianas cada una. Las gausianas se definen com matriz de covarianzas simplificada, de tipo diagonal. Si se desea usar matrices de covariaza completas, basta no incluir los parámetros `'cov_type','diag'`.

Para realizar aprendizaje supervisado con el BNT lo único que falta es poblar en cada vector de entrenamiento el campo donde van las etiquetas de clase (en este caso en la posición 1) e invocar el motor de aprendizaje:

```
datosApr             = cell(numNodos, numVec);
datosApr(numNodos,:) = num2cell(datApr',  1);
datosApr(1,:)        = num2cell(etiqApr', 1);
motor                = jtree_inf_engine(redB);
maxIter              = 16;
epsilon              = 0.1;
[redB2, ll, motor2]  = learn_params_em(motor, datosApr, maxIter, epsilon);
```

Una vez entrenada la red, podemos cargar los datos de test:

```
datTest  = load('./data/spam/tr.dat',       '-ascii');
etqTest  = load('./data/spam/trlabels.dat', '-ascii');
etiqTest = etqTest + 1;
```

y realizar un experimento de clasificación. Para ello primero obtenemos, para cada vector de test, las probabilidades a posteriori de cada clase y las almacenado en el array `p()`:

```
p = zeros(length(datTest), numClas); %% Limpiamos p por si se ha usado antes
evidencia = cell(numNodos,1); %% Un cell array vacio para las observaciones
for i=1:length(datTest)
  evidencia{numNodos} = datTest(i,:)';
  [motor3, ll]  = enter_evidence(motor2, evidencia);
  m = marginal_nodes(motor3, 1);
  p(i,:) = m.T';
end
```

Para realizar la clasificación propiamente dicha, se asume como clase resultante la de mayor probabilidad a posteriori y esta clase se compara con la clase correcta. Si no coinciden se incrementa el contador de errores:

```
err = 0;
for i=1:length(p)
  [val,ind] = max(p(i,:));
  if(ind ~= etiqTest(i)) err = err+1; end
end
errorFinal = 100*err/length(p)
```

El aprendizaje mediante el algoritmo EM requiere una inicialización de los parámetros de los modelos (medias y matrices de covarianza de las gausianas). BNT inicializa estos parámetros aleatoriamente, usando el generador de nÂ´ umeros aleatorios de MATLAB. Por esta razón, si se realizan varios procesos de aprendizaje y test con los los mismos datos, probablemente no se obtendrán idénticos resultados. Como generalmente es conveniente garanzizar la repetitividad de los experimentos, se puede usar la función `rng(seed)` de MATLAB para inicializar el generador de números aleatorios con un valor de las semialla (seed) dado. Como se ha comentado, variando este valor se obtendrán variaciones (usualmente pequeñas) en los resultados, para los mismos datos de entrenamiento y test.

3.2.1. Ejercicios Propuestos

1. *A entregar:* Implementar una función de MATLAB que determine la tasa de error de clasificación, tomando como entrada los siguientes parámetros:
   - Fichero de datos de entrenamiento,
   - Fichero de etiquetas de los datos de entrenamiento,
   - Fichero de datos de prueba o *test*,
   - Fichero de etiquetas de los datos de prueba,
   - Número de gausianas por mixtura,
   - Número de iteraciones de aprendizaje.
2. Usando la función implementada, obtener la tasa de error para los datos de test del corpus SPAM. Completar una tabla donde se muestre la tasa de error al variar el número de gausianas por mixtura y el número de iteraciones de aprendizaje.
3. *Opcional:* Realizar una experimentación similar con el corpus USPS (reconocimiento de dígitos manuscritos).