Trabajo IA

Inferencia exacta en Redes Bayesianas

Grupo 2 - IIS

Víctor José Marín Aguilar

Índice

Contenido

1. Descripción del ejemplo elegido

Solucionador de problemas para impresoras de Windows 95

La red bayesiana elegida como ejemplo fue modelada por los ingenieros de Microsoft para desarrollar un solucionador de problemas para impresoras. Se compone de 76 nodos y 112 aristas. Puede encontrarse en http://www.bnlearn.com/bnrepository/.

Se contemplan 6 problemas distintos a los que encontrar solución:

1. ***No Output*:** la impresora no produce ningún resultado al solicitar una impresión.
2. ***Too Slow:*** la impresora tarda demasiado en imprimir.
3. ***Incomplete Page:*** las páginas se imprimen parcialmente.
4. ***Graphics Distorted or Incomplete*:** las imágenes se imprimen distorsionada o parcialmente.
5. ***Fonts Missing or Distorted:*** la impresión no presenta la fuente esperada o está distorsionada.
6. ***Garbled Output:*** la impresión es ilegible.

El sistema operativo trataría de encontrar las máximas evidencias que pudiera por sí mismo para el problema detectado y, a partir de ellas, inferir las posibles causas para informar al usuario de ello o darle solución.

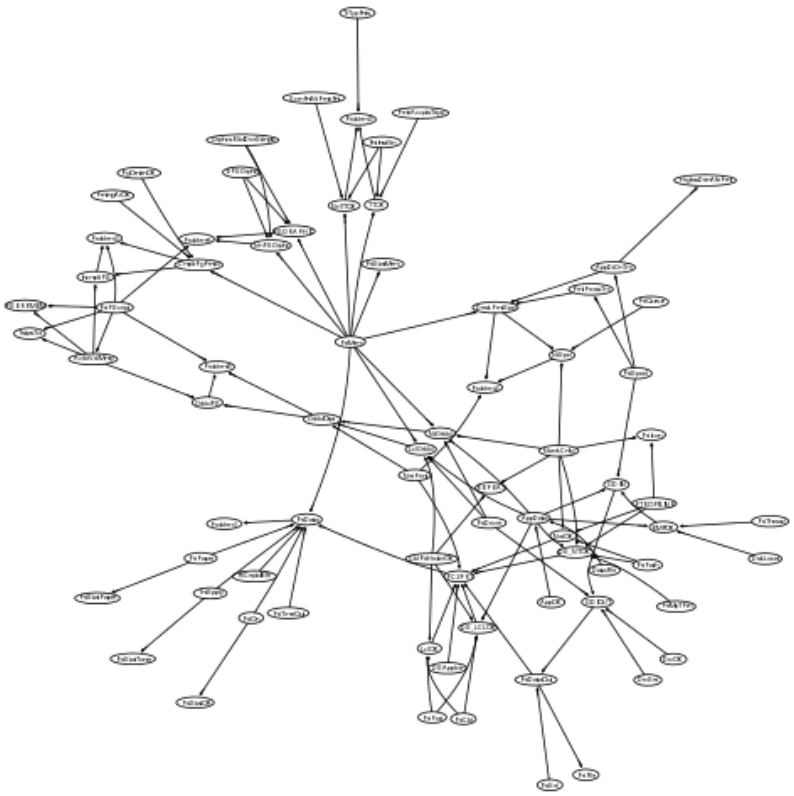

Figura. Topografía de la red.

2. Descripción de la representación escogida

Red

La representación escogida para la red ha sido un Grafo Acíclico Dirigido (*DAG* por sus siglas en inglés) debido a que esta es la estructura subyacente bajo las redes bayesianas.

Para la implementación de la clase *DAG*, se ha utilizado como base la clase di*graph* de la librería *pygraph* en conjunción con el método *find_cycle*. De esta manera, basta comprobar que las aristas que se añadan al grafo no formen ciclos con las ya existentes, manteniendo así la integridad de la red bayesiana.

Los nodos de la red se han representado mediante una clase *RandomVariable* encargada de almacenar el nombre, dominio y tabla de probabilidad (*CPT* por sus siglas en inglés) de cada uno de ellos.

Las aristas de la red son tuplas del tipo (*Node1*, *Node2*), representando una arista dirigida con sentido *Node1* hacia *Node2*.

Tablas de probabilidad

Las tablas de probabilidad se han modelado como un diccionario en el que las claves son valores del dominio de los nodos incidentes (si los tuviera) en orden, seguidos de valores del dominio del propio nodo y, los valores, son las probabilidades asociadas a cada clave.

3. Descripción del algoritmo y las heurísticas

Eliminación de Variables (EV)

La implementación del algoritmo de Eliminación de Variables realizada sigue los siguientes pasos:

1. Obtener los factores iniciales a partir de la red.
2. Descartar las variables irrelevantes para la consulta.
3. Proyectar observaciones si las hubiera.
4. Eliminar las variables ocultas.
5. Multiplicar los factores finales si los hubiera.
6. Normalizar.

La clase *VariableElimination* se encarga de controlar el flujo del algoritmo.

1. Obtener los factores iniciales a partir de la red

Si es la primera consulta realizada sobre la red bayesiana se calculan todos los factores iniciales. En caso contrario, se recuperan los factores iniciales calculados previamente para no tener que ser computados nuevamente con cada consulta.

2. Descartar las variables irrelevantes para la consulta

Se eliminan las variables irrelevantes según el criterio especificado en el **Tema 7: Representación y razonamiento con conocimiento incierto** de la asignatura.

"Toda variable que no sea antecesor (en la red) de alguna de las variables de consulta o de evidencia, es irrelevante para la consulta y por tanto puede ser eliminada."

Si una variables es antecesora de otra en la red se determina según la siguiente proposición (*Dietz's numbering scheme*):

"For two given nodes x and y of a tree T, x is an ancestor of y if and only if x occurs before y in the preorder traversal of T and after y in the post-order traversal."

3. Proyectar observaciones si las hubiera

Se proyectan las observaciones y se calculan los nuevos factores resultantes.

4. Eliminar variables ocultas

Se eliminan las variables ocultas en el orden establecido por la heurística escogida.

5. Multiplicar los factores finales si los hubiera

En el caso de que al final haya más de un factor relevante, se multiplican los factores finales.

6. Normalizar

Se normaliza la CPT del factor resultante, siendo esta la solución a la consulta.

Heurísticas

Las heurísticas seleccionan la siguiente variable a eliminar en cada paso de eliminación de variables ocultas.

Se encuentran implementadas en la clase *Heuristics*, de forma que dada una lista de variables a eliminar y una lista de factores, devuelven la variable a eliminar según su criterio.

Se dispone de las heurística exigidas:

- **Min. Degree:** devuelve la variable que está "conectada" con menos variables en el conjunto de factores.
- **Min. Fill**: devuelve la variable que al eliminarse introdujera menos conexiones nuevas.
- **Min. Factor:** devuelve la variable que al eliminarse produjera el factor más pequeño.

Así como de un orden de eliminación aleatorio, para ver la eficiencia de las heurísticas:

- **Random:** orden de eliminación de variables aleatorio.

4. Consultas y experimentación

Uso del algoritmo

A continuación se detalla cómo utilizar el algoritmo tanto con la interfaz de usuario como sin ella.

Con interfaz de usuario

Para inicializar la interfaz de usuario hay que ejecutar el archivo *Main.py* localizado en el directorio raíz.

La interfaz le guiará a través del proceso.

Sin interfaz de usuario

Cada red dispone de un archivo en el directorio raíz. A continuación se muestra la asociación:

- **Fire Alarm:** *FireAlarmBeliefNetwork.py*
- **Simple Diagnosis:** *SimpleDiagnosticExample.py*
- **Electrical Diagnosis Problem:** *ElectricalDiagnosisProblem.py*
- **Windows 95 Printer Troubleshooter:** *Win95PrinterTroubleshooter.py*

Abrir el archivo sobre cuya red se desee realizar la consulta y modificar la consulta predefinida para ajustarla a la deseada, así como el modo de ejecución y la heurística.

Los posibles modos de ejecución son:

- **Modo traza:** *'verbose'*
- **Modo rápido:** *'brief'*

Las posibles heurísticas son:

- **Min. Degree:** *'min-degree'*
- **Min. Fill:** *'min-fill'*
- **Min. Factor:** *'min-factor'*
- **Random:** *'random'*

Ejecutar el archivo en cuestión.

Formato de una consulta

Las consultas se representan mediante objetos del tipo *Query*. Éstos reciben como parámetros un nodo de la red bayesiana y, opcionalmente, una lista de observaciones. Cada observación es del tipo *EvidenceVariable* y requiere un nodo de la red y un valor observado del dominio del mismo.

Ejemplos de consulta

La siguiente consulta representaría *P(w0 | w2 = live, p1 = T)* sobre la red *network.*

evidence_variable_1 = EvidenceVariable(network.get_node_by_name('w2'), 'live')
evidence_variable_2 = EvidenceVariable(network.get_node_by_name('p1'), 'T')

query = Query(network.get_node_by_name('w0'), [evidence_variable_1, evidence_variable_2])

La siguiente consulta representaría *P(w4)* sobre la red *network.*

query = Query(network.get_node_by_name('w4'))

Consulta aleatoria

Es posible crear una consulta al azar mediante el método *random* de la clase *Query*. Para ello es necesario darle como parámetros: la red bayesiana en primer lugar y, opcionalmente, un número de observaciones.

Si el número de observaciones no se ha especificado, éste será aleatorio también y se encontrará entre 0 y el número máximo posible.

Ejemplos de consulta aleatoria

Consulta aleatoria sobre la red *network* con 2 observaciones.

random_query = Query.random(network, 2)

Consulta aleatoria sobre la red *network* con un número de observaciones aleatorio.

random_query = Query.random(network)

Rendimiento de las heurísticas

El orden de eliminación de las variables ocultas tiene gran impacto en el tiempo de ejecución del algoritmo.

Las tres heurísticas implementadas ofrecen rendimientos muy similares, destacando *Min. Fill* como ligeramente más rápida para redes de cierta complejidad.

Esto se debe a que dicha heurística produce un orden de eliminación de variables mejor, pero el coste de computarla es considerablemente superior al de las otras dos. Por esto, en redes simples, el tiempo empleado en calcular la heurística contrarresta la mejora obtenida durante la eliminación de variables.

A continuación se presentan los resultados de resolver 100 consultas con 3 observaciones cada una en redes de distinta complejidad. Ejecutar el archivo *HeuristicsTest.py* para reproducir el test.

Red *Windows 95 Printer Troubleshooter*

HEURÍSTICA	MIN. DEGREE	MIN. FILL	MIN. FACTOR	RANDOM
TIEMPO	2.17 s.	2.12 s.	2.21 s.	> 10 min.

Es destacable que el algoritmo no termina con un orden de eliminación aleatorio al tratarse de una red de cierta complejidad.

Red *Electrical Diagnosis Problem*

HEURÍSTICA	MIN. DEGREE	MIN. FILL	MIN. FACTOR	RANDOM
TIEMPO	0.51 s.	0.53 s.	0.52 s.	2.01 s.

En este caso, un orden de eliminación aleatorio las resuelve en un tiempo admisible. No obstante, tarda 4 veces más que con el resto de heurísticas.

5. Referencias utilizadas

- Material de la asignatura:
  - http://www.cs.us.es/cursos/iais/temas/tema-07.pdf

- Librería Python para grafos pygraph:
  - https://code.google.com/p/python-graph/

- Manipulación del DOM:
  - http://www.w3schools.com/dom/dom_element.asp
  - https://docs.python.org/2/library/xml.dom.html

- Ancestros de un nodo:
  - http://www.cs.arizona.edu/xiss/numbering.htm

- Conditional Probability Table:
  - http://en.wikipedia.org/wiki/Conditional_probability_table