Redes Bayesianas
Sesión2

5 de octubre de 2023

1. Introducción

El código de OpenMarkov está estructurado de una forma modular en subproyectos[1]. Cada uno de estos subproyectos está alojado en repositorios distintos en Bitbucket y se pueden gestionar usando Maven. En la Figura 1 se muestra parte de esta estructura.

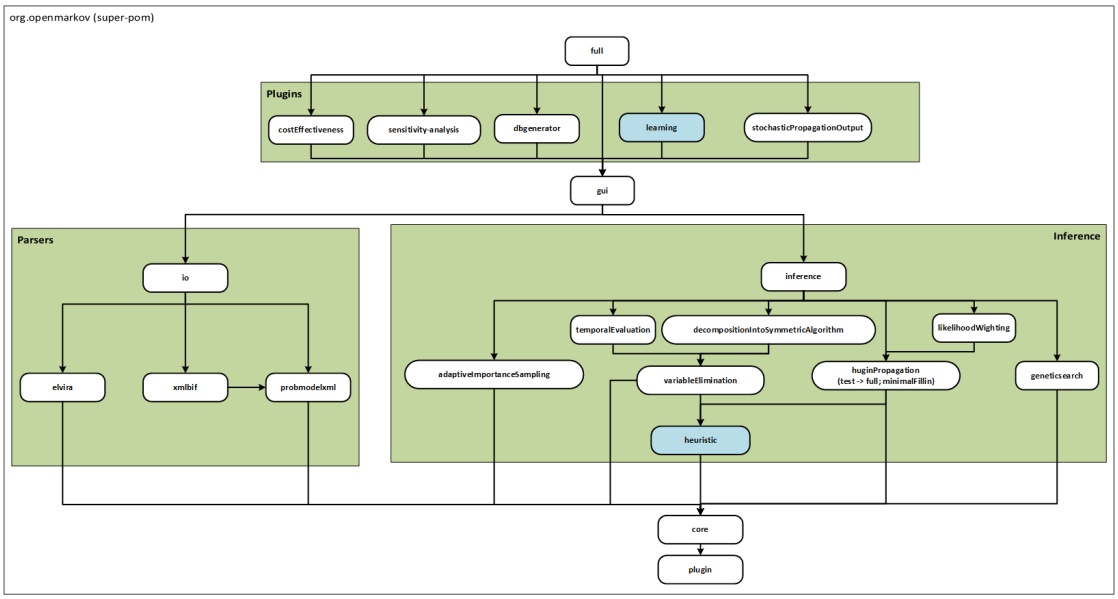

Figura 1: Estructura de OpenMarkov.

En esta sesión vamos a trabajar programando con OpenMarkov, pero en lugar de usar el código del repositorio oficial importaremos en Eclipse el proyecto contenido en el archivo openmarkov-0.3.2.zip.

2. Primer contacto con el código

En esta sesión vamos a realizar la inferencia sobre una red bayesiana de forma programática. Antes de lanzarnos a ello, echemos un vistazo a como se lanza el proceso de inferencia desde la interfaz gráfica de OpenMarkov.

[1] https://bitbucket.org/cisiad/org.openmarkov/wiki/OpenMarkov's_organization

Esto se encuentra en la clase `EditorPanel`, que como se indica en la documentación Javadoc, se encarga de implementar el comportamiento cuando hay cambios sobre la red cargada. Más en concreto, nos interesa el método `doPropagation()`.

Método `doPropagation`

```
1  public boolean doPropagation(EvidenceCase evidenceCase, int caseNumber) {
2      Map<Variable, TablePotential>
3      individualProbabilities = null;
4      boolean propagationSucceded = false;
5      try {
6          long start = System.currentTimeMillis();
7          try {
8              calculateMinAndMaxUtilityRanges();
9
10             Propagation vePosteriorValues = new VEPropagation(probNet);
11             vePosteriorValues.setVariablesOfInterest(probNet.getVariables());
12             vePosteriorValues.setPreResolutionEvidence(preResolutionEvidence);
13             vePosteriorValues.setPostResolutionEvidence(evidenceCase);
14             individualProbabilities = vePosteriorValues.getPosteriorValues();
15         } catch (OutOfMemoryError e) {
16             ...
17         }
18         long elapsedTimeMillis = System.currentTimeMillis() - start;
19         System.out.println("Inference took " + elapsedTimeMillis + " milliseconds.");
20         updateNodesFindingState(evidenceCase);
21         paintInferenceResults(caseNumber, individualProbabilities, evidenceCase);
22         propagationSucceded = true;
23
24     } catch ...
25     }
26     ...
27 }
```

El primer parámetro de este método es `evindeceCase`, de la clase `EvidenceCase` que codifica la configuración de los valores que son conocidos, es decir, se obtendrá la probabilidad condicionada a esos valores. Además de la evidencia, para poder hacer la inferencia necesitamos la red. Esta clase tiene un atributo llamado `probNet` de la clase `ProbNet` que tiene el objeto que representa red con la que se está trabajando.

Después, en la línea 10 se crea el objeto que implementa el algoritmo de inferencia. En OpenMarkov hay varios algoritmos implementados, pero en la interfaz gráfica se usa siempre el algoritmo de eliminación de variables mediante la clase `VEPropagation`. Al objeto de esta clase se le pasa la red en el constructor, mediante los métodos apropiados se le indica las variables de interés, de las que queremos conocer su probabilidad, y la evidencia.

Por último, se obtiene el resultado llamando al método `getPosteriorValues()`. Al final, se muestran los valores obtenidos y también se muestra, en la línea 19, el tiempo que se tardó en este cálculo. Este mensaje aparecerá en la ventana "Ventana de mensajes".

Para tener una visión un poco más amplia, en la Figura 2 se muestran algunas de las clases más importantes para manejar una red bayesiana, o en general un modelo gráfico probabilístico, en OpenMarkov. Como ya hemos visto, la clase `ProbNet` define la red. Esta clase extiende de la clase `Graph` que, como se puede intuir, implementa las operaciones de esta estructura de datos. Junto con el grafo tenemos la clase `Link` que codifica las uniones entre los nodos del grafo en la red bayesiana. Estas dos clases están parametrizadas en función del tipo de nodo cuya clase base es `Node`. A su vez un nodo de la red tiene acceso a un objeto de la clase `Variable` y a otro de la clase `Potential`. La primera encapsula información sobre la variable aleatoria que está asociada con cada nodo y la segunda define la función de probabilidad de esta variable/nodo. En estas sesiones vamos a trabajar solo con variables de tipo categórico y por tanto su función de probabilidad será una tabla de probabilidad condicional.

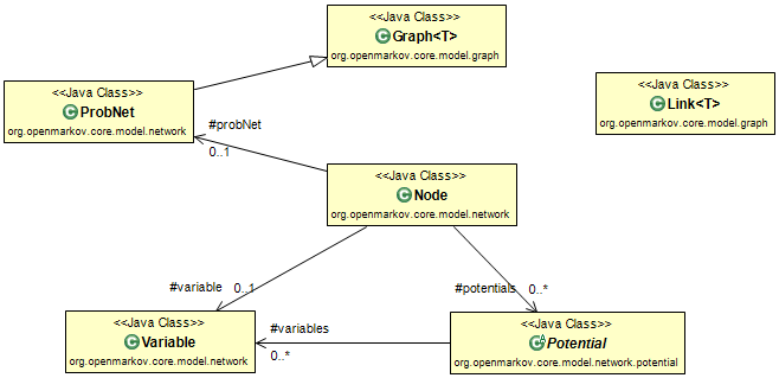

Figura 2: Diagrama UML con las clases más importantes para representar una red bayesiana en OpenMarkov.

Pasando ahora a la parte de inferencia, en la Figura 3 se muestran de nuevo las clases más importantes de OpenMarkov en este aspecto. En primer lugar tenemos la interfaz `Propagation` que define estos tres métodos que podemos ver en la figura. Con ellos se indica la evidencia a propagar, las variables de interés, aquellas por las que vamos a preguntar, y el método para obtener la probabilidad de las variables de interés condicionadas a la evidencia. En la versión actual de OpenMarkov existen varios algoritmos de inferencia implementados. Dos de ellos para inferencia exacta, eliminación de variables[2] (`VEPropagation`) y árboles de uniones[3] (`HuginPropagation`), y otros dos para inferencia aproximada, muestreo estocástico (`LogicSampling`) y ponderación de la verosimilitud (`LikelihoodWeighting`). Además de estas clases que implementan los algoritmos de inferencia también tenemos la clase `EvidenceCase` que representa la evidencia en base a objetos de la clase `Finding`. Esta última clase simplemente empareja cada variable de la evidencia con el valor concreto que se fija.

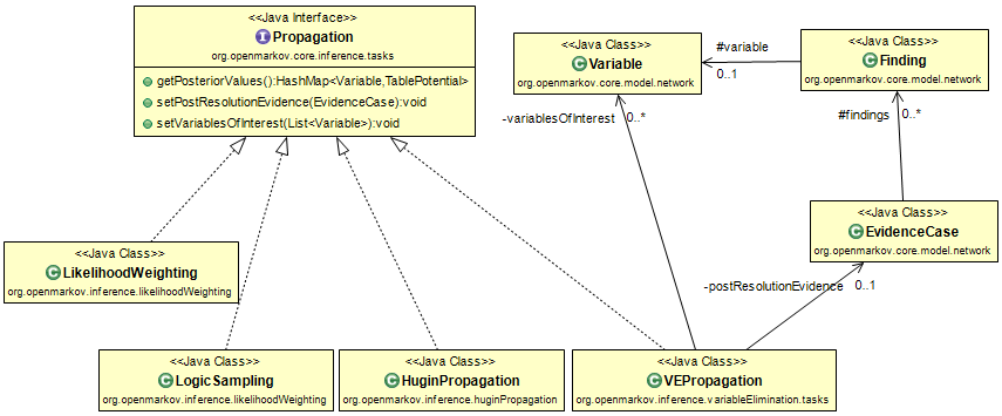

Figura 3: Diagrama UML con las clases más importantes para realizar inferencia en OpenMarkov.

3. Análisis de los algoritmos de inferencia

Aunque la interfaz gráfica de OpenMarkov no permite elegir entre los algoritmos de inferencia que están implementados, podemos usarlos de forma programática.

[2] https://en.wikipedia.org/wiki/Variable_elimination
[3] https://en.wikipedia.org/wiki/Junction_tree_algorithm

En el proyecto que hemos importado en Eclipse, además del código de OpenMarkov tenemos disponible la clase `es.uniovi.ssii.rb.InferenceTester`. Esta clase tiene definidos una serie de método para cargar redes y realizar inferencia con ellas. Con el código que se entrega, si ejecutamos el método `main` se cargará la red *asia.pgmx*, se establece una semilla para la generación de números aleatorios, se muestran algunos datos de la red cargada, se crean una evidencia (e) de forma aleatoria, seleccionando un número de variables y un valor concreto para cada una de ellas; después se eligen también de forma aleatoria unas variables de interés ($\mathbf{x}$) que no estén en la evidencia y por último se realiza la inferencia para obtener $P(x_1|e), \cdots, P(x_n|e)$.

Ejercicio 1. Utilizando el código de la clase `es.uniovi.ssii.rb.InferenceTester` como base, añade los métodos para hacer inferencia con los otros tres algoritmos disponibles para compararlos entre sí. En esta comparación vamos a analizar el tiempo empleado por cada algoritmo. Por cada una de las redes que se muestran en la Tabla 1, realiza 5 ejecuciones distintas para cada uno de los 4 algoritmos, una variable en la evidencia y 6 variables de interés. Para los algoritmos aproximados también hay que especificar el número de muestras, usaremos 10.000 y además les pasaremos la misma semilla de números aleatorios. Con esto, calcula en tiempo medio para cada red y completa la Tabla 2 con los valores obtenidos.

Nombre	Nodos	Enlaces	Media padres	Media MB	Media estados	Parámetros
alarm	37	46	1.24	3.89	2.84	752
Barley	48	84	1.75	6.25	8.77	130180
Child	20	25	1.25	3.10	3.00	344
Diabetes	413	602	1.46	4.20	11.34	461069
insurance	27	52	1.93	6.22	3.30	1419
Link	724	1125	1.55	5.38	2.53	20502
Pigs	441	592	1.34	4.03	3.00	8427
win95pts	76	112	1.47	6.50	2.00	1148

Tabla 1: Redes bayesianas usadas en la comparación. Fuente: https://www.bnlearn.com/bnrepository/.

Red	`VEPropagation`	`HuginPropagation`	`LogicSampling`	`LikelyhoodWeighting`
alarm				
Barley				
Child				
Diabetes				
insurance				
Link				
Pigs				
win95pts				

Tabla 2: Resultados del tiempo empleado por cada algoritmos de inferencia.

Ejercicio 2. Según los datos de la Tabla 2, ¿qué algoritmo es el más rápido en general? ¿Hay algún caso en el que su velocidad no sea la mayor? Intenta explicar este hecho según las propiedades de las redes en la Tabla 1.

Ejercicio 3. En lugar de mirar al tiempo de ejecución miremos también al valor de probabilidad calculada por cada algoritmo. Los algoritmos de eliminación de variables y de árboles de uniones son exactos y por tanto ambos algoritmos deben mostrar los mismos resultados. ¿Qué ocurre con los algoritmos aproximados? ¿Cómo se puede obtener una mejor aproximación de la probabilidad? De entre el muestreo estocástico y la ponderación por la verosimilitud, ¿cuál es mejor? ¿Concuerda este resultado con lo que sabemos de cada algoritmo?