

**Figura 1:** Representação da rede bayesiana utilizada no projeto

Uma rede bayesiana consiste num modelo que representa a relação probabilística entre um conjunto de varíaveis. As redes bayesianas tem por base o Teorema de Bayes, em que dado dois eventos A e B, P(A|B) = . Esta probabilidade, em teoria, representa o grau de confiança de um agente que um dado evento vai ocorrer, baseado na observação prévia.

As redes bayesianas deste projeto são acíclicas e constituídas por nodes, em que para cada node, é possível calcular a probabilidade desse node ocorrer (e não ocorrer) a partir de uma lista de probabilidades e parents – a probabilidade de cada node é calculada através da função computeProb, com base numa lista de probabilidades e numa lista de parents.

Dentro das redes bayesianas, a partir de uma evidência, é possível calcular a probabilidade conjunta (joint probability). A partir da probabilidade conjunta, é possível chegarmos à probabilidade a-posteriori de uma varíavel (post probability), que corresponde à seguinte fórmula para calcular essa probabilidade: dadas duas varíaveis aleatórias discretas, X e Y, então: P(X = x e Y = y) = . Esta probabilidade é calculada, dada uma evidência, através das combinações possíveis para a variável desconhecida da evidência.

Após executar o ficheiro mainBN.py, observou-se que o código que foi produzido obteve resultados iguais aos pretendidos no ficheiro. Isto aplica-se tanto na função de computeProb, computeJointProb e computePostProb.

Face às desvantagens do uso de redes bayesianas, elas restringem a previsão apenas aos próximos n+1 acontecimentos dados os n acontecimentos previamente observado. Além disso, a complexidade das redes bayesianas – são computacionalmente caras.

Em termos de vantagens, permite a atualização desses valores de probabilidade sempre que uma nova observação é realizada e obtém-se novos dados. Além disso, face aos métodos normais, permite uma compactação dos dados em relações de independência condicional – sabendo os parentes de um node, permite-nos saber de que eventos ele depende para ocorrer.

A complexidade do código depende do número de combinações máximas que existem entre [0,1] e o número de valores desconhecidos na evidência que a função computePostProb recebe como argumento. Quanto mais valores desconhecidos existem, mais combinações possíveis existem, e mais lento será o programa. Ou seja, sendo o array [0,1]( (valores possíveis dentro da evidência) identificado como A e o número de valores desconhecidos [] representado por n, a complexidade do programa é O(C(A,n)), sendo C o símbolo de combinações.

É possível melhorar o algoritmo da seguinte forma: como na função computePostProb usamos todas as combinações possíveis para calcular o resultado – o que torna o programa pouco eficiente – podemos reduzir o número de combinações possíveis, analisando se existissem variáveis independentes.

---------------------------------------------------------------//------------------------------------------------------------