**Entrega Final Projeto AMC – Grupo 15**

Em grupo o nosso raciocínio por detrás de cada método a ser desenvolvido teve sempre em vista a finalidade do projeto. Sendo o projeto assente em algumas premissas como: a estrutura de dados das amostras é m x (n+1), em que a classe é o último inteiro de cada linha ou que a classe é pai de todos os nós. Assim, a correta execução de todos os métodos só está assegurada de acordo com estas características dos dados e grafo.

**Definimos dois atributos na classe *Amostra*:**

* ***amostra*** (ArrayList < int [] >): Cada elemento da ArrayList é um array de comprimento fixo, uma vez que o tamanho dos elementos da Amostra é igual e constante)
* ***dom*** (int []): Array de comprimento igual ao dos elementos da amostra cujo objetivo é guardar em cada posição o domínio dessa variável. No método *add*, compara-se os valores de cada elemento a adicionar com os valores no array *dom* de modo a atualizá-lo. O método *domain* retorna o produto dos valores de *dom* selecionados.

Além dos métodos pedidos no enunciado:

**Definimos dois atributos na classe *grafoo*:**

* ***dim*** (int): dimensão do grafo, número de nós a contar com a classe.
* ***adj*** (ArrayList<ArrayList<Integer>>): lista de listas em que cada lista tem os pais do nó da posição correspondente.

Além dos métodos pedidos no enunciado desenvolvemos os seguintes:

***- log2*:** que recebe um float e retorna o logaritmo base dois do mesmo. Não foi feita uma condição para o caso do valor ser igual a 0 uma vez que isso nunca irá acontecer no nosso código, devido a uma condição existente dentro do método *I\_T,* uma função auxiliar do método MDL.

**- *I\_T*:** método auxiliar do MDL. Recebe uma amostra e um nó. Foi definida uma sobrecarga que é utilizada no MDLdelta para permitir o cálculo deste sem efetuar alterações no grafoo. Otimizando o MDLdelta.

***- combi*:** recebe uma ArrayList de posições e devolve todas as combinações de todos os valores que as variáveis correspondentes a esses nós podem tomar em estrutura ArrayList de ArrayList. Utilizámos este método como auxiliar do *I\_T*, onde recebe as posições dos pais de um nó e devolve as combinações de *D\_pi\_-* e *D\_C* definidos no enunciado do projeto. Isto permite converter os três somatórios de *I\_T* em apenas dois, o que percorre o domínio do nó e o do retorno de *combi*. Na expressão de *MDL*, no cálculo de *theta* também utilizamos o *domain* dos *parents*, incluindo a classe, para simplificar *|D\_pi\_-|* x *|D\_C|*.

A principal alteração feita na classe *grafoo* foi na função MDLdelta, onde para verificar a alteração no MDL *score* provocada por uma operação deixámos de necessitar de calcular dois valores distintos de MDL, passando a calcular apenas a diferença entre as parcelas internas ao MDL *score* alteradas pela operação em questão. Desta forma, tornámos o processo de aprendizagem significativamente mais eficiente.

Note-se que ao longo de todo o projeto utilizámos sempre os métodos das classes *Amostra* e *grafoo*, visto que estas resultaram de um *extend* das classes *Amostra\_entrega1* e *grafoo\_entrega*1, que correspondem às classes entregues previamente.

**Definimos quatro atributos na classe *BN*:**

* ***am*** (Amostra)
* ***grafo*** *(grafoo)*
* ***DFOs*** (ArrayList<double [][]>): ArrayList com matrizes de *doubles*, com *length* igual ao número de nós do grafo sem contar com a classe. Cada matriz tem tantas linhas quanto o *domain* da variável correspondente à posição na lista e tantas colunas quanto o *domain* dos seus pais. Em cada posição é guardado o respetivo *theta*.
* ***theta\_c*** *(double [])*: vetor de *length* igual ao *domain* da class. Em cada posição temos os *thetas* para o correspondente valor da classe.

Desenvolvemos ainda os métodos:

- ***gettheta\_c*** que permite saber o número de valores que toma a classe no classificador, uma vez que este apenas recebe um BN e um vetor.

- ***getgrafo*** que devolve o grafo da BN e permite o classificador conhecer o seu número de nós (nº de variáveis) , uma vez que este apenas recebe um BN e um vetor.

- ***combi*** e ***combTotal*** que ajudam a obter vetores com as combinações dos pais de um nó e o mesmo com a adição do valor do nó no início desse vetor, correspondentemente.

Após a definição das classes acima referidas, definimos mais duas classes: **Aprendizagem** e **Classificador**.

**Definimos dois atributos na classe Aprendizagem:**

* ***GRAF***(grafoo) *–* armazena o grafo com melhor MDL
* ***RedeBayes***(BN)

Definimos os seguintes métodos:

- O construtor **Aprendizagem** recebe um inteiro *nr\_graf* (número de grafos iniciais), um inteiro *max\_p* (número máximos de pais), um double *S*  (pseudo-contagem) e uma amostra *am.* São aprendidos *nr\_graf* grafos através do método *GHC*, em que o primeiro é sempre um grafo desconexo. Os restantes grafos são criados através do método *randomDAG.* Para cada grafo aprendido, se esse tiver melhor MDL do que o anterior, é armazenado nos atributos GRAF e MDL. No final, RedeBayes é atualizada para a rede de bayes a partir do melhor grafo.

**- *RedeBayes***devolve o atributo RedeBayes.

**- *randomDAG***recebe um inteiro k (número máximo de pais) e um inteiro n (tamanho do grafo a criar contando com a classe). Tem como objetivo criar grafos aleatórios sem ciclos e garante o número máximo de pais e que a classe é pai de todos

**- GHC** recebe um grafo g e uma amostra am. Para o grafo g vamos avaliar todas as operações em todos os nós, caso elas possam ser executadas e não comprometam o número máximo de pais para nenhum nó, e armazenar a informação sobre o pai, filho e operação (op) que produzem menor MDLdelta, sem a realizar. Depois de percorrermos tudo e obtermos essa informação, vamos então realizar a operação armazenada. Repete-se o processo até não ser possível melhorar o MDL.

**NOTA**: A nossa função MDLdelta calcula o MDLdelta da operação sem garantir que possa ser executada. Isto porque na função GHC queremos verificar se a operação pode ser realizada antes de tentar calcular o seu MDLdelta.

**- getRandomInRange** gera um valor aleatório entre um mínimo e um máximo.

**Definimos dois atributos na classe *Classificador*:**

* ***c*** (int)
* ***p***(double)

- O método construtor**Classificador** recebe uma BN e um vetor de inteiros (valores das variáveis s/ classe) a ser classificado. Adiciona ao vetor os valores possíveis da classe e calcula a prob desse vetor para cada valor possível da classe. Atualiza o atributo c com o valor da classe mais provável e o atributo p com a probabilidade de ser esse o valor correto da classe.

- O método ***class\_value*** retorna os atributos num tuplo.

Dentro desta classe, no *main*, está definido o ***Leave One Out***. Para correr o programa basta seguir algumas instruções que estão no comentário do código. Para desenvolver este programa fizemos um *extend* da classe Amostra com o nome **AmostraLOO** que tem mais dois atributos, o vetor que se retira à amostra sem a classe que será classificado e a própria classe. O método construtor define os valores destes atributos e são definidos métodos para disponibilizar estas variáveis.

Utilizámos o **LOO** para testar a precisão de aprendizagem do nosso programa, experimentando diferentes combinações de nº máximo de pais / nº de grafos para diferentes amostras:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **Freq. Relativa** | **3 / 4** | **4 / 4** | **4 / 10** | **6 / 10** | **4 / 30** |
| **Breast Cancer** | 82,43% | 83,02% | 83,16% | - | - | - |
| **Hepatitis** | 83,75% | 90% | 90% | 90% | 90% | 90% |
| **Diabetes** | 65,10% | 78,52% | 77,99% | 78,52% | - | - |

Apesar de o volume de testes não ter sido tão elevado como gostaríamos, concluímos que, de forma geral, o aumento do nº de grafos está associado a um aumento da precisão, mas também do tempo de processamento e que a diminuição do número máximo de pais leva a uma diminuição no tempo de processamento, não afetando significativamente a precisão dos resultados.

Não realizámos testes completos de LOO para a amostra de cancro da tiroide, uma vez que a amostra é de grande dimensão e comporta bastantes variáveis, no entanto testámos o LOO para partes diferentes da amostra isoladas e os resultados obtidos foram superiores à frequência relativa da classe mais frequente.

De forma geral, a precisão no nosso algoritmo foi sempre acima da frequência relativa da classe mais frequente, o que evidencia a sua credibilidade a nível de diagnósticos.

**Guia do Utilizador da Aplicação**

- Em primeiro lugar, o utilizador deve abrir a aplicação correspondente à Aprendizagem da Rede de Bayes e selecionar o ficheiro que contém a amostra desejada. Se o ficheiro for possível de ler aparecerá no output a mensagem “ficheiro aceite”. De seguida, deve inserir nas caixas de texto correspondentes o número máximo de pais (int >0), número de grafos inicial (int >0) e pseudo-contagem (double >0). Deve ainda indicar qual o nome que pretende atribuir ao ficheiro que contém a rede de Bayes aprendida e clicar no botão “Aprender”. Se os parâmetros forem válidos aparecerá uma mensagem a dizer que a rede de Bayes foi aprendida e guardada com o nome atribuído pelo utilizador.

- A seguir, deve abrir-se a aplicação correspondente ao classificador e selecionar o ficheiro correspondente à rede de Bayes que se quer utilizar. A mensagem “ficheiro aceite” aparecerá no output se o ficheiro for possível de ler. De seguida devem ser inseridos os parâmetros do paciente no campo correspondente, separando os números por vírgulas, e carregar ENTER. Aparece assim o botão “Classificar”. Por fim, clica-se nesse mesmo botão e, caso o ficheiro seja válido, o nº de parâmetros inseridos seja igual ao nº de variáveis variáveis e os valores estejam dentro do domínio das variáveis, é devolvido o valor que a classe assume, assim como a respetiva probabilidade.