

INSTITUTO SUPERIOR TÉCNICO

Algoritmos e Modelação Computacional

Licenciatura de Bolonha Engenharia Biomédica

2º Período 2021-2022

30 de Janeiro de 2022

Relatório de entrega do Projeto

Alexandre Lumier, 92974.

Tiago Bolaños, 90927.

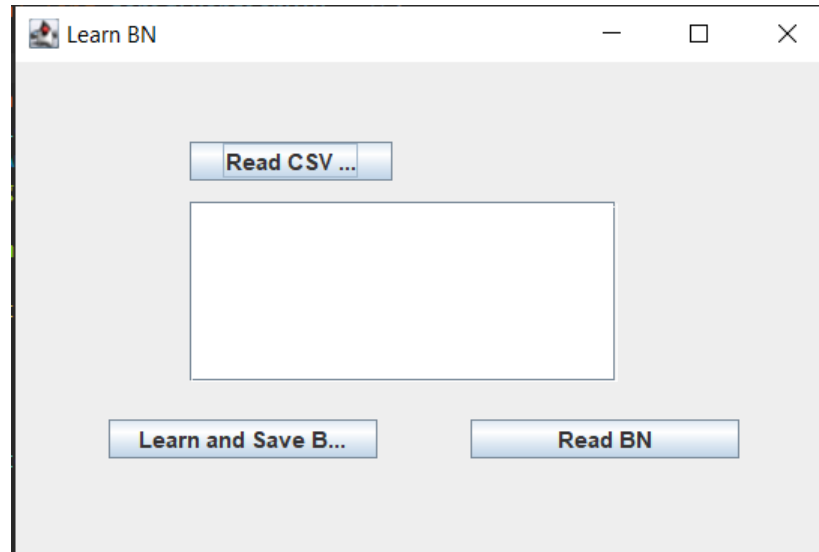
Inês Marques, 99674.

Carolina Machado, 99694.

Aplicações

Foram criadas duas aplicações: uma para aprender a rede de bayes correspondente à amostra fornecida e guardá-la no computador, e outra para, utilizando a rede de bayes guardada na aplicação anterior, classificar um vetor dado.

1ª Aplicação:

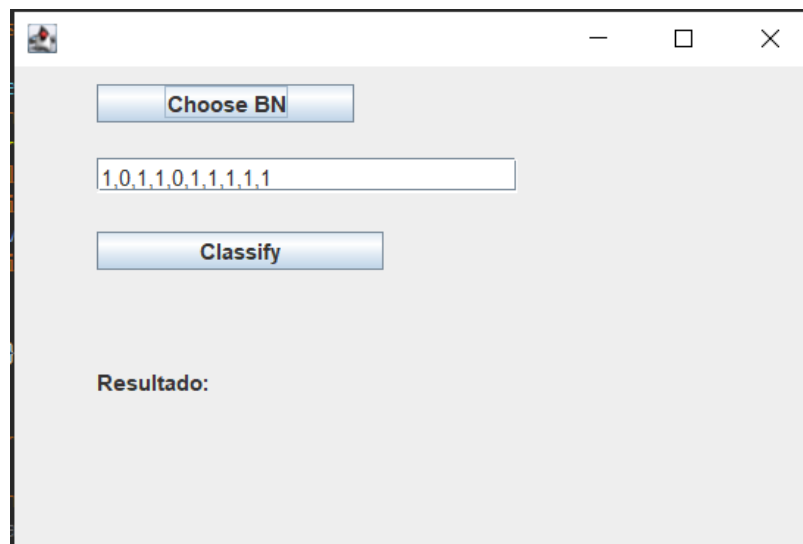


Tem um botão *read CSV* no qual se seleciona um ficheiro CSV que vai ser analisado, este deve conter a classe na última variável.

Um botão *Learn and Save BN* na qual se escolhe onde guardar o ficheiro da BN, este deve ser guardado como **.bin** e no final mostra na caixa de texto *Saved Successfully*.

Um botão *Read BN* que para além de ler o ficheiro BN analisa cada vetor utilizando a BN e retorna na caixa de texto a percentagem de vetores corretamente classificados.

2ª Aplicação:

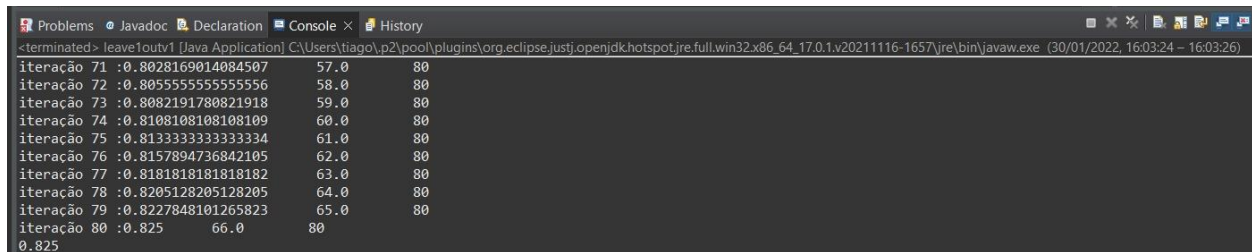


Esta aplicação inclui um botão onde se pode escolher o ficheiro **.bin** onde foi guardada a BN aprendida.

Uma caixa de texto onde se pode introduzir o vetor a analisar separado por vírgula, a aplicação automaticamente ajusta o vetor para ter o tamanho adequado, caso seja pequeno adicionando zeros nas outras posições, caso seja demasiado grande, contando apenas os primeiros índices. Um botão *classify* para classificar o vetor introduzido, o resultado aparecendo em baixo a classe mais provável.

Para testar as redes de bayes geradas foi ainda criado um método **Leave one out** que gera uma rede de bayes com a amostra excluindo um vetor, de seguida classifica esse vetor e verifica se a classe obtida era efetivamente a classe da amostra. Isto é iterado ao longo de toda a amostra de modo a obter uma percentagem de vetores bem classificados, uma vez que algumas amostras levavam mais de 1 minuto a gerara rede de Bayes o método Leave One Out inclui um inteiro que indica como discriminar a amostra, isto é, de quantos em quantos vetores a analisar pois algumas amostras levaria cerca de 1h ou em alguns casos, várias horas a analisar tudo.

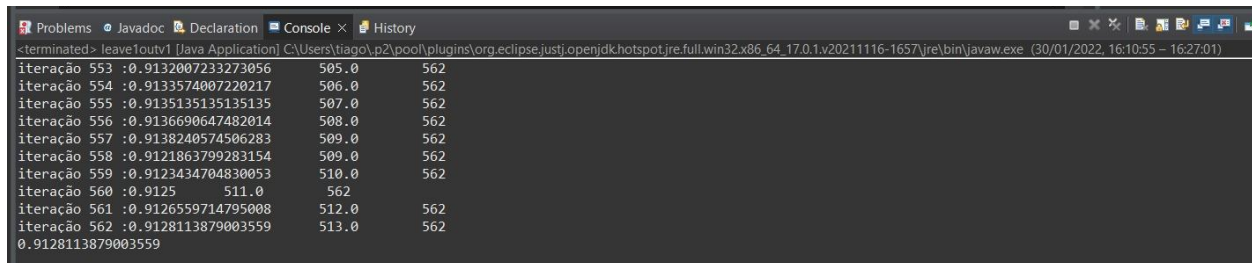
Resultados



```
<terminated> leave1outv1 [Java Application] C:\Users\tiago\p2\pool\plugins\org.eclipse.justi.openjdk.hotspot.jre.full.win32.x86_64_17.0.1.v20211116-1657\jre\bin\javaw.exe (30/01/2022, 16:03:24 - 16:03:26)
iteração 71 :0.8028169014084507 57.0 80
iteração 72 :0.8055555555555556 58.0 80
iteração 73 :0.8082191780821918 59.0 80
iteração 74 :0.8108108108108109 60.0 80
iteração 75 :0.8133333333333334 61.0 80
iteração 76 :0.8157894736842105 62.0 80
iteração 77 :0.8181818181818182 63.0 80
iteração 78 :0.8205128205128205 64.0 80
iteração 79 :0.8227848101265823 65.0 80
iteração 80 :0.825 66.0 80
0.825
```

Resultado leave one out “Hepatitis”: 83%

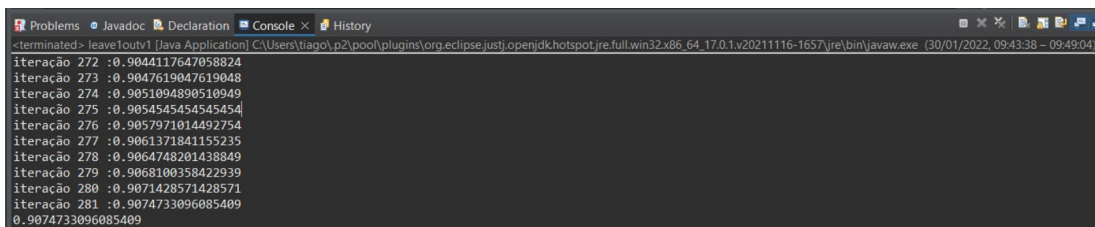
Tempo: 2s



```
<terminated> leave1outv1 [Java Application] C:\Users\tiago\p2\pool\plugins\org.eclipse.justi.openjdk.hotspot.jre.full.win32.x86_64_17.0.1.v20211116-1657\jre\bin\javaw.exe (30/01/2022, 16:10:55 - 16:27:01)
iteração 553 :0.9132007233273056 505.0 562
iteração 554 :0.9133574007220217 506.0 562
iteração 555 :0.9135135135135135 507.0 562
iteração 556 :0.9136690647482014 508.0 562
iteração 557 :0.9138240574506283 509.0 562
iteração 558 :0.9121863799283154 509.0 562
iteração 559 :0.9123434704830053 510.0 562
iteração 560 :0.9125 511.0 562
iteração 561 :0.9126559714795008 512.0 562
iteração 562 :0.9128113879003559 513.0 562
0.9128113879003559
```

Resultado leave one out “Tiroide”, incremento 30: 91%

Tempo:16min 6s



```
<terminated> leave1outv1 [Java Application] C:\Users\tiago\p2\pool\plugins\org.eclipse.justi.openjdk.hotspot.jre.full.win32.x86_64_17.0.1.v20211116-1657\jre\bin\javaw.exe (30/01/2022, 09:43:38 - 09:49:04)
iteração 272 :0.9044117647058824
iteração 273 :0.9047619047619048
iteração 274 :0.9051094890510949
iteração 275 :0.9054545454545454
iteração 276 :0.9057971014492754
iteração 277 :0.9061371841155235
iteração 278 :0.9064748201438849
iteração 279 :0.9068100358422939
iteração 280 :0.9071428571428571
iteração 281 :0.9074733096085409
0.9074733096085409
```

Resultado leave one out “Soybean”, incremento 1:91%

Tempo:5 min 26s

```
Problems Javadoc Declaration Console X History
<terminated> leave1outv1 [Java Application] C:\Users\tiago\p2\pool\plugins\org.eclipse.justi.openjdk.hotspot.jre.full.win32.x86_64_17.0.1.v20211116-1657\jre\bin\javaw.exe (30/01/2022, 16:28:05 - 16:53:16)
iteração 2085 :0.9592320135008723 2000.0 2643
iteração 2086 :0.9592521572387345 2001.0 2643
iteração 2087 :0.9592716818399617 2002.0 2643
iteração 2088 :0.9592911877394636 2003.0 2643
iteração 2089 :0.9593106749640976 2004.0 2643
iteração 2090 :0.9593301435406698 2005.0 2643
iteração 2091 :0.959349593495935 2006.0 2643
```

Resultado leave one out "Tyroide", incremento 1:95%
Tempo: 24 min 9s

```
Problems Javadoc Declaration Console X History
<terminated> leave1outv1 [Java Application] C:\Users\tiago\p2\pool\plugins\org.eclipse.justi.openjdk.hotspot.jre.full.win32.x86_64_17.0.1.v20211116-1657\jre\bin\javaw.exe (30/01/2022, 15:41:46 - 15:42:01)
iteração 674 :0.7314540059347181 683 683
iteração 675 :0.7318518518518519 683 683
iteração 676 :0.7307692307692307 683 683
iteração 677 :0.7311669128508124 683 683
iteração 678 :0.7315634218289085 683 683
iteração 679 :0.7319587628865979 683 683
iteração 680 :0.7323529411764705 683 683
iteração 681 :0.7327459618208517 683 683
iteração 682 :0.7331378299120235 683 683
iteração 683 :0.7335285505124451 683 683
0.7335285505124451
```

Resultado leave one out "bcancer", incremento 1: 73%
Tempo: 15 s

```
Problems Javadoc Declaration Console X History
<terminated> leave1outv1 [Java Application] C:\Users\tiago\p2\pool\plugins\org.eclipse.justi.openjdk.hotspot.jre.full.win32.x86_64_17.0.1.v20211116-1657\jre\bin\javaw.exe (30/01/2022, 15:44:56 - 15:45:17)
iteração 759 :0.761528326745718 768 768
iteração 760 :0.7605263157894737 768 768
iteração 761 :0.7608409986859396 768 768
iteração 762 :0.7611548556430446 768 768
iteração 763 :0.7614678899082569 768 768
iteração 764 :0.7604712041884817 768 768
iteração 765 :0.7607843137254902 768 768
iteração 766 :0.7597911227154047 768 768
iteração 767 :0.758800521512386 768 768
iteração 768 :0.7591145833333334 768 768
0.7591145833333334
```

Resultado leave one out "diabetes", incremento 1:76%
Tempo: 21 s

```
Problems Javadoc Declaration Console × History
<terminated> leave1outv1 [Java Application] C:\Users\tiago\p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.1.v20211116-1657\jre\bin\javaw.exe (30/01/2022, 15:51:33 - 15:51:35)
iteração 141 :0.9716312056737588      150      150
iteração 142 :0.971830985915493      150      150
iteração 143 :0.965034965034965      150      150
iteração 144 :0.9652777777777778      150      150
iteração 145 :0.9655172413793104      150      150
iteração 146 :0.958904109589041      150      150
iteração 147 :0.9591836734693877      150      150
iteração 148 :0.9594594594594594      150      150
iteração 149 :0.959731543624161      150      150
iteração 150 :0.96      150      150
0.96
```

Resultado leave one out "Íris", incremento 1: 96%
Tempo: 2s

Classes

Amostra

A amostra foi implementada como uma ArrayList cujos elementos (cada linha da amostra) são arrays de inteiros, sendo que a classe apenas possui um atributo, **list** que corresponde a ArrayList. Foi escolhida a ArrayList em vez de um vetor de inteiros pois o tamanho do vetor teria de ser dinâmico.

A função **add** consiste apenas em adicionar um vetor de inteiros, à ArrayList, enquanto que a função **element** devolve o array que se encontra na posição pedida na ArrayList. O método **domain** recorre a um ciclo que percorrendo toda a amostra e retorna o domínio da posição que recebe. Na função **count** foi implementada recorrendo a dois ciclos, percorrendo a amostra toda e as posições desejadas para cada elemento desta. Foi também definido um método **domainV**, não pedido no enunciado, que retorna um vetor com os domínios de cada variável dos elementos da amostra, de forma a melhorar o tempo de execução de outros métodos. Foram implementadas a função auxiliar **lixo** e a função **clean**, que tem como objetivo retirar da amostra todos os elementos das variáveis que têm domínio 1 e retorna a nova amostra.

Foi adicionado o método remove, que retira o elemento da amostra na posição dada, que foi usado para o teste Leave One Out.

Grafo

A classe **grafoo** foi implementada contendo 2 atributos, um inteiro **dim** que guarda a dimensão da matriz e um vetor de vetores double **ma** que guarda a matriz de adjacência do grafo e os seus pesos.

Possui também os seguintes métodos:

- Método construtor **grafoo** que recebe um inteiro **n** e cria um grafo com uma dimensão **n** e uma matriz **ma** com um vetor de vetores **n** por **n** vazia.
- Método **add_edge** que recebe dois inteiros **m** e **n** e um double **p** e adiciona à matriz **ma** na posição **mn** e na posição **nm**, pois o grafo não têm direção, o peso **p** guardando assim a aresta entre **m** e **n** com o valor **p**.
- Os métodos **findMaxVertex**, **printMaximumSpanningTree** e **maximumSpanningTree** que foram adaptados do seguinte site, <https://www.geeksforgeeks.org/prims-minimum-spanning-tree-mst-greedy-algo-5/> e que implementa o algoritmo de prim para obter a max spanning tree.
- O método **amostrado** que recebe como argumento uma Amostra **amostra** e acrescenta as arestas com os pesos calculados a partir da amostra usando a fórmula:

$$I_T(X; Y) = \sum_{x,y} Pr_T(x, y) \log \left(\frac{Pr_T(x, y)}{Pr_T(y)Pr_T(x)} \right)$$

Para isso utiliza a função auxiliar **pesagem**, que calcula o peso individual de cada aresta e adiciona-o à matriz **ma** usando a função **add_edge**.

Floresta

A classe **Floresta** foi implementada contendo 3 atributos, **nodes**, um inteiro que guarda o número de nós que a floresta tem; **LP**, um vetor de inteiros que guarda o valor do pai do nó correspondente ao índice no vetor; **LW**, um vetor de doubles que indica o peso da ligação do nó correspondente ao índice com o seu pai.

Possui também os seguintes métodos:

- Método construtor **Floresta**, que recebe um inteiro **n** e cria uma floresta com **n** nós, uma lista vazia de **LW**, e uma lista **LP** preenchida com -1.
- Método **set_parent**, recebe dois inteiros **n** e **m** e torna **m** o pai de **n**, trocando o seu valor em **LP**.
- Método **set_weight**, recebe um inteiro **n** e um double **p** e torna o peso de **n** **p**.
- Método **weight** que recebe um inteiro **n** e devolve o valor do peso do nó **n** da lista **LW**.
- Método **parent** que recebe um inteiro **n** e devolve pai do nó com índice **n** da lista **LP**.
- Método **parentQ** que recebe um inteiro **o** e devolve se o nó **o** têm pai, verificando, com a função **parent()**, se o pai desse nó é -1, que foi o valor definido para nós sem pai.
- Método **treeQ** que verifica se a floresta é uma árvore verificando se há mais que uma raiz, isto é, um nó sem pai (pai=-1).

Redes Bayesianas

As redes de Bayes, **BN**, foram definidas com 4 atributos, uma **Floresta**, uma ArrayList de doubles **theta** e os inteiros **vectorSize** e **classSize**.

A função **BN** é um método construtor que gera uma rede de Bayes utilizando a estrutura de uma árvore previamente construída e com as distribuições DFO's amortizadas com pseudo-contagens **S**. Assim a função recebe como argumentos uma árvore, uma amostra e um double **S** que será o valor do coeficiente de pseudo contagem.

O método **BN** funciona em dois passos, onde no primeiro passo criou-se um ciclo **for** no qual será criado uma arraylist de matrizes, na qual cada matriz terá como dimensão o domínio dos nós (pais x filhos). Uma vez criadas essas matrizes, cada posição nas matrizes é preenchida com o valor do theta calculado para esse conjunto pai-filho usando a segunda fórmula do ponto 2.4 do enunciado.

$$\Theta_{i|w_i}(d_i) = \frac{|T_{d_i, w_i}| + S}{|T_{w_i}| + S \times |D_i|}.$$

Depois, foi usado no segundo passo um ciclo *for* para criar uma matriz cuja a dimensão é o domínio dos (filhos x 1), e vai preencher cada posição da matriz calculando os thetas da classe com base na primeira fórmula descrita no ponto 2.4 do enunciado.

$$\Theta_{i|w_i}(d_i) = \frac{|T_{d_i, w_i}|}{|T_{w_i}|}$$

Esta segunda lista será adicionada a arraylist criada no primeiro passo, e esse conjunto de matrizes será então a rede de bayes criada pela função **BN**.

Nessa classe também foi implementada a função **prob**, que recebe um como argumento um vetor e retorna a probabilidade desse vetor na rede de Bayes construída pela **BN**, com base na fórmula do ponto 2.4.1 do enunciado.

$$\Pr(\vec{V} = (x_1, \dots, x_n, x_{n+1})) = \prod_{i=1}^{n+1} \Pr(X_i = d_i | \Pi_i = (d_{i,1} \dots d_{i,k_i}))$$