Projeto de Algoritmos e Modelação Computacional

Campos de Markov Aleatórios – Árvores

MEBiom – 2º Semestre 2020/2021

Augusto Marques, n.º 89789
Beatriz Filipe, n.º 92799
Miguel Fernandes, n.º 92836

I. INTRODUÇÃO

O principal objetivo do projeto da cadeira de Algoritmos e Modelação Computacional consistiu em implementar um algoritmo de aprendizagem baseado em *Campos de Markov Aleatórios* (*Markov Random Fields*). Este algoritmo faz uso do algoritmo de *Chow-Liu* para aprender MRFs que otimizam o processo de classificação. Todo o projeto foi desenvolvido em JAVA, uma linguagem de programação imperativa universal e fortemente tipada.

Como referido, este projeto insere-se na área de *Machine-Learning*, cujas aplicações têm elevada importância no contexto clínico e hospitalar, nomeadamente, como ferramentas de apoio à decisão médica, sendo, por isso, extremamente relevante para a Engenharia Biomédica.

II. CLASSES E MÉTODOS

Para proceder à implementação do algoritmo de aprendizagem e se conseguir, por fim, classificar um dado conjunto de dados, foram criadas várias classes e, para cada uma, vários métodos, necessários para a sua total definição. De seguida, explicita-se as classes criadas, abordando os métodos e as implementações construídas, ao mesmo tempo que se justificam escolhas tomadas durante a implementação de cada classe.

A. Dataset

A classe `Dataset` é construída a partir de um conjunto de dados, `T`, sendo implementado como `ArrayList`, estrutura dinâmica com métodos próprios, já implementados e otimizados na linguagem JAVA. Esta classe vai ser essencial ao longo do projeto, já que serve de base para a construção da implementação de outras classes e métodos. Tem como atributos: `data` (*protected*) - `ArrayList` que guarda os dados que vão sendo integrados no `Dataset`; `n` (*private*) –número (inteiro) de variáveis aleatórias; `D` (*private*) – vetor com os valores máximos (inteiros) das variáveis aleatórias do conjunto inicial de dados (define o domínio associado a cada variável, D_i); e `Freqlist` (*protected*) – `ArrayList` de inteiros com valores das frequências de cada classe do `Dataset`.

Esta classe tem como principais **métodos**:

- `Dataset(int n)` – método construtor da classe que recebe o número inteiro, `n`, de variáveis aleatórias e inicializa um novo objeto desta classe, definindo o estado inicial do objeto, quanto aos seus atributos.
- `Add(int[] v)` – recebe um vetor e adiciona-o ao dataset se verificar as dimensões apropriadas. Esta adição é feita de modo a que a `ArrayList` do `Dataset` esteja ordenada, implementação que irá melhorar a eficiência e rapidez da contagem de elementos no `Dataset` e, ainda, a acessibilidade destes elementos, nomeadamente na construção das fibras. A ordenação é feita de acordo com a classe atribuída a cada vetor introduzido, recorrendo à `Freqlist`. Esta vai sendo atualizada a cada adição (através do método privado `AtualizaFreqList(int[] v)`), bem como o atributo `D` (atualizado pelo método privado `D_max(int[] v, int[] D)`).
- `Count(int[] vars, int[] val)` – recebe uma lista de variáveis (`vars`) e valores destas (`val`) e retorna o número de vezes que estas variáveis tomam simultaneamente esses valores no `Dataset`. De referir que não é necessário confirmar se os valores pertencem ao domínio de cada variável, D_i, pois este método pressupõe que tal já tenha sido verificado previamente na classe `Classifier` (poupamos algum trabalho computacional).
- `Fiber(int c)` – método que, dado um valor da classe, `c`, retorna a fibra (`Dataset`) associada a esse valor da classe. Este método público tem associado um outro método privado – `fiberAdd(int[] v)` – que permite a adição dos vetores do `Dataset` original, `T`, à `ArrayList data` da fibra e que tem em conta a ordenação de `T`.

B. Weighted Graph

A classe `Weighted Graph` é uma das classes pedidas no enunciado do projeto. Tem como atributos o inteiro `dim` (*private*) – número de variáveis aleatórias de `T` – e a

matriz `ma` (*private*) – matriz de adjacência, cujas entradas ma_{ij} e respetivos valores representam as arestas pesadas que unem as variáveis i e j.

Tem como **métodos**:

- `WeightedGraph(int dim)` – método construtor do objeto da classe `WeightedGraph`, que recebe um inteiro `dim`, e cria uma matriz de adjacência de dimensões `dim`×`dim`, em que $ma_{ij} = -\infty$, $\forall i, j$. Define também o estado inicial do objeto, quanto aos seus atributos.
- `MST()`– um subgrafo do `Weighted Graph` que é uma árvore (com todos os nós), cuja soma dos pesos das arestas é maximal entre todos os subgrafos que formam o `Weighted Graph`. Este método foi implementado adaptando o algoritmo de Prim (estudado nas aulas para obter a árvore de extensão mínima), trocando o sinal de todas as entradas da matriz de adjacência para obter a árvore de extensão máxima, de acordo com o *Introduction to Algorithms* do Cormen.
- `Add( int i, int j, double w)` – recebe dois nós, `i` e `j`, e um peso, `w`, e adiciona uma aresta entre os nós `i` e `j` com o peso `w`, em que `w` $\in \mathbb{R}$.

C. Tree

A classe `Tree` é uma implementação simples de uma árvore. Nesta implementação, toda a informação relativa à árvore é armazenada num vetor de inteiros `pais`, onde cada entrada de índice i corresponde ao pai do nó i. Esta classe servirá de suporte para a criação da Árvore de Extensão de Peso Maximal (MST), gerada pelo método da classe `Weighted Graph`.

Assim, esta classe tem os métodos:

- `Tree(int dim)` – método construtor que recebe a dimensão da árvore (número de variáveis aleatórias) e inicializa o vetor `pais` como um vetor de dimensão `dim` e com todas as entradas com o valor `-1`.
- `addEdge(int i, int pai)` – cria uma nova aresta na árvore, sendo que o nó `i` passa a ser filho do nó `pai`.
- `EdgeQ(int i, int j)` – verifica se existe uma aresta entre os nós `i` e `j`.
- `pai(int i)` – devolve o pai do nó `i`.

D. Weighted Tree

A classe `WeightedTree` foi criada para ser usada na implementação do atributo `markovtree` da classe `MRFTree`. Para além disso, a `WeightedTree` implementa `Serializable` de forma a podermos guardar um objeto desta classe como um ficheiro de acordo com o que foi explicado nas aulas práticas.

Esta classe possui apenas dois atributos: uma constante inteira, `dim`, que corresponde à dimensão da árvore; e `int[][][][] ma`, que pode ser vista como uma matriz cuja entrada ma_{ij}, representa a aresta entre os nós i e j , que contém uma matriz que representa a função $\phi_{ij}(x_i, x_j)$.

A classe tem como **métodos**:

- `WeightedTree(int dim)` – método construtor da classe que recebe a dimensão da matriz `ma` e define o estado inicial dos atributos.
- `Add(int i, int j, double[][] phi)` – recebe dois nós, `i` e `j`, e a matriz de *doubles* `phi`, e adiciona essa matriz à aresta que liga os nós `i` e `j`. Caso um dos nós recebidos não exista no grafo, lança um erro do tipo `AssertionError`.
- `getWeight(int i, int j)` – recebe dois nós (i e j) e retorna a matriz `phi` da entrada ma_{ij}, caso ambos os nós existam no grafo. Caso um dos nós recebidos não exista no grafo, lança um erro do tipo `AssertionError`.

E. MRFTree (Markov Random Field Tree)

A classe `MRFTree` é equivalente à classe MRFT referida no projeto.

Esta classe possui diversos atributos, como as constantes inteiras (*privadas*) `mc` – dimensão da fibra da classe `c` do `Dataset T`; ((`m` – tamanho do `Dataset T`)); `n` - número de medições (variáveis aleatórias); e a constante `D` - um vetor de inteiros que contém o domínio de cada variável do `Dataset T` (sabemos que as concretizações das variáveis aleatórias são inteiros maiores ou iguais a zero, pelo que, neste vetor de inteiros `D`, guardamos o valor máximo que cada variável toma no `Dataset T` (de origem). Possui ainda `tfiber`, um objeto da classe `Dataset`, que corresponde a uma fibra de `T`. Além destes atributos, a class `MRFTree` possui também atributos que lhe são únicos: `e` (*private*) – um vetor de inteiros de dimensão 2, no qual é armazenada a aresta especial (valores armazenados correspondem aos dois nós que a formam); `delta` (*private*) – um double que indica o valor de pseudo-contagem; e `E` (*private*) – uma `ArrayList` de vetores de inteiros, nos quais são armazenadas as arestas.

A `MRFTree` tem como principais **métodos**:

- `MRFTree(Tree arvore, Dataset tfiber)` – método construtor, que recebe um objeto da classe `Tree` (`arvore`) e um `Dataset` do tipo fibra (`tfiber`), referente a uma classe, c, e coloca os $\phi_{ij}(x_i, x_j)$ em cada aresta da `MRFTree`. Além disso, define o estado inicial do objeto, quanto aos seus

atributos. Para construir a `MRFTree`, fixa-se o primeiro nó como raiz, sendo um dos nós da aresta especial, o que induz a direção das arestas.

- `phi(int i, int j)` – método que recebe dois inteiros correspondentes aos nós `i` e `j`, e retorna a matriz de *doubles* $\phi_{ij}(x_i, x_j)$ da aresta que liga o nó `i` ao nó `j` da árvore.

- `prob (int[] v)` – recebe um vetor de inteiros $(x_1, ..., x_n)$, e retorna a probabilidade destes dados no dataset – ou seja, P_{Mc}. Caso um dos valores x_i presentes no vetor não pertença ao domínio da variável aleatória correspondente, D_i, lança-se um `AssertionError`.

F. Classifier

Esta classe tem como objetivo classificar uma amostra, $(x_1, ..., x_n)$, de **n** variáveis aleatórias, $(X_1, ..., X_n)$, para uma dada doença, retornando o valor da classe, c, mais provável associada a essa amostra (diagnóstico), com base no modelo aprendido com esse tipo de dados, através do dataset `T`. Tem como atributos: uma `ArrayList` de objetos da classe `MRFTree`, a `MRFTList` e uma `ArrayList` com as frequências de cada classe, c, no dataset `T`, a `FreqList`. Além disso, implementa `Serializable` de forma a podermos guardar um objeto desta classe como um ficheiro de acordo com o que foi explicado nas aulas práticas.

Tem como **métodos**:

- `Classifier`(ArrayList<MRFTree>MRFTList, ArrayList<Integer> FreqList) – método construtor que recebe um array de MRFT's, um para cada valor da classe, c, e uma `ArrayList` com as frequências das classes no dataset `T`.

- `classify(int[] amostra)` – dados valores $(x_1, ..., x_n)$ das variáveis retorna o valor da classe c mais provável.

III. RESULTADOS E ANÁLISE

Vários datasets já tratados no formato .csv foram cedidos pelo docente na página da disciplina. Estes dados públicos foram retirados do *UCI Machine Learning Repository*, que contém dados referentes a diversas patologias:

- bcancer.csv – Cancro da mama
- diabetes.csv – Diabetes
- hepatitis.csv – Hepatite
- thyroid.csv – Tiróide

Para testar o modelo, retirou-se uma linha do dataset escolhido e treinou-se o modelo com o dataset resultante dessa alteração. Posteriormente, utilizou-se como amostra os dados da linha que retiramos, excluindo a classificação, e submeteu-se a amostra ao classificador. Por fim, procurou-se verificar se a classificação atribuída após a aprendizagem corresponde à classificação original, presente na linha que removemos do dataset. Repetiu-se este procedimento para todas as linhas de todos os datasets, tendo obtido os seguintes resultados:

Dataset	certos/total	exatidão
bcancer.csv	556/683	81.41%
diabetes.csv	590/768	76.82%
hepatitis.csv	73/80	91.25%
thyroid.csv	2558/2643	96.78%

Consideramos que os resultados são bastante satisfatórios. Em baixo, é mostrado um exemplo ilustrativo de um grafo completo para $n = 10$ variáveis e respetiva MST.

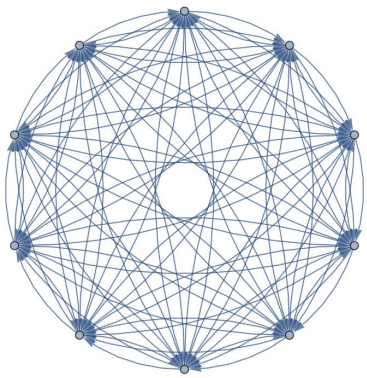


Figura 1 - Exemplo ilustrativo de um grafo completo para $n = 10$ variáveis aleatórias. Neste caso, para a fibra de classe 0 do dataset bcancer.csv

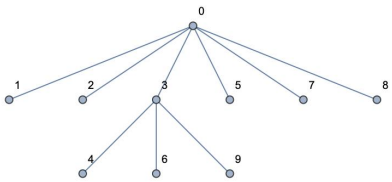


Figura 2 - Exemplo ilustrativo de uma árvore de extensão maximal, obtida a partir do grafo da figura 1.

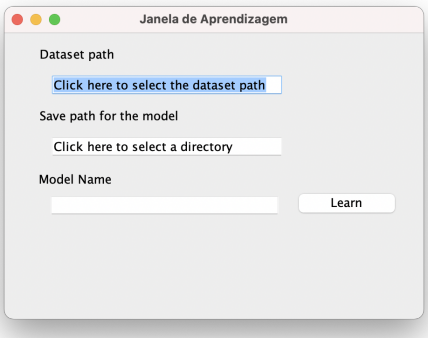
IV. MANUAL DO UTILIZADOR

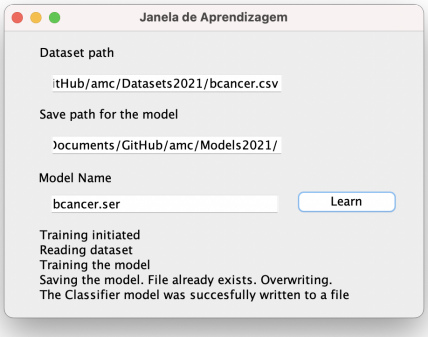
Nesta secção explica-se sucintamente como usar o software desenvolvido neste projeto.

A. Aprendizagem

Numa primeira etapa, é necessário criar um modelo da patologia em estudo a partir de um dataset. Este deve incluir dados já tratados e estar no formato .csv.

1. Clicar na caixa de texto do *Dataset path* e escolher o diretório do ficheiro .csv a ler.
2. Clicar na caixa de texto do *Save path for the model* e escolher o directório onde guardar o ficheiro do modelo treinado.
3. Clicar no botão *Learn*.

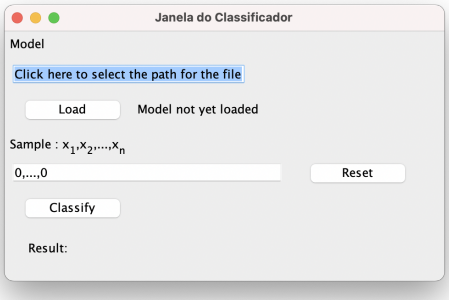


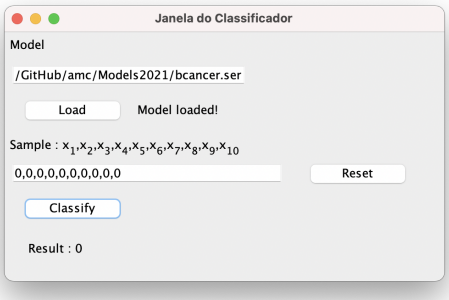


B. Classificação

Com o modelo da patologia criado, é possível classificar um dado paciente (*sample*) com base nos dados guardados neste modelo. O resultado (*Result*) retorna o valor da classe (inteiro) referente a um diagnóstico.

1. Clicar na caixa de texto do *Model* e escolher o ficheiro com o modelo.
2. Clicar no botão *Load* para carregar o modelo.
3. Clicar na caixa de texto da *Sample* e introduzir a amostra.
4. Clicar no botão *Classify*.





V. CONCLUSÕES

Neste projeto, foi desenvolvido um algoritmo que classifica pacientes, para uma dada patologia, com base em dados obtidos, *à priori*, sobre a mesma. Este foi implementado através do algoritmo de *Chow liu* – um método gráfico probabilístico, de segunda ordem, que evita o sobreajuste de dados (*overfitting*), calculando probabilidades entre variáveis com o auxílio de certos tipos de objetos, designadamente grafos e árvores. Este revelou-se importante e eficiente, sendo uma possível contribuição para o futuro da tecnologia em medicina.

Por fim, esta temática, estando intimamente relacionada com a nossa área (engenharia biomédica), foi também um fator de motivação para a boa implementação e funcionamento do algoritmo, permitindo, também, desenvolver as nossas competências e habilidades na linguagem de programação JAVA, nomeadamente na criação eficiente de algoritmos.

VI. REFERÊNCIAS

1. Santos Ribeiro, Rodrigo. (2018). Algoritmos para Estimação de Modelos Gráficos. Universidade de São Paulo, Instituto de Matemática e Estatística. Acedido dem `https://linux.ime.usp.br/~rodrigorsdc/mac0499/monografia.pdf`.
2. Introduction to Algorithms, T. Cormen, C. Leiserson, R. Rivest e C. Stein, 2001, McGraw Hill e MIT Press, 3ª edição.