Algoritmos e Modelação Computacional (AMC) - 2.º Semestre 2020/2021

Campos de Markov Aleatórios

Relatório do Projeto

Autores

Aline Gouveia, n.º 91594
António Ramiro, n.º 92796
Maria Brazão, n.º 92826

Resumo

O presente relatório descreve sucintamente o funcionamento do algoritmo desenvolvido no âmbito da disciplina de AMC. Este foi desenvolvido com vista a ser treinado em dados médicos e providenciar um provável diagnóstico para um novo vetor de dados, desconhecido. Trata-se, portanto, de um classificador com base em Markov Random Field Trees, que utiliza uma árvore cujos ramos são escolhidos pela implementação do algoritmo de Chow Liu.

Fluxo de Dados

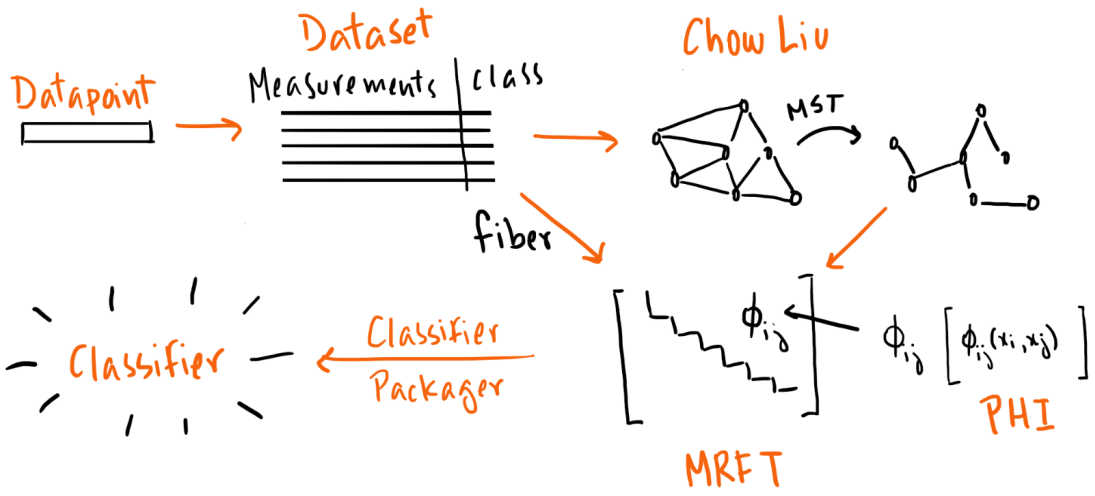


Figura 1. Esquematização do fluxo de dados através dos objetos

A imagem acima resume o algoritmo de forma condensada. Este recebe um conjunto de dados com informação sobre as medições e o *ground truth* do diagnóstico (*Datapoints*), que são guardados num Dataset. A informação mútua de cada par de nós do dataset (nós esses que representam cada uma das medições) é utilizada para construir um grafo pesado (*WGraph*), e é construída uma árvore maximal (*Tree*). Recorrendo agora a cada uma das fibras do Dataset e à árvore proveniente de Chow Liu, é computada a Markov Random Field Tree (*MRFT*), que será empacotada com as frequências de cada classe e permitirá a posterior classificação de novos conjuntos de medições.

Guia de Utilizador da Interface

Génese do Classificador

Para criar um classificador, é primeiro necessário um conjunto de dados armazenado num ficheiro *.csv*, com uma amostra por linha, onde as medições e a classificação (números naturais) estão separados por vírgulas.

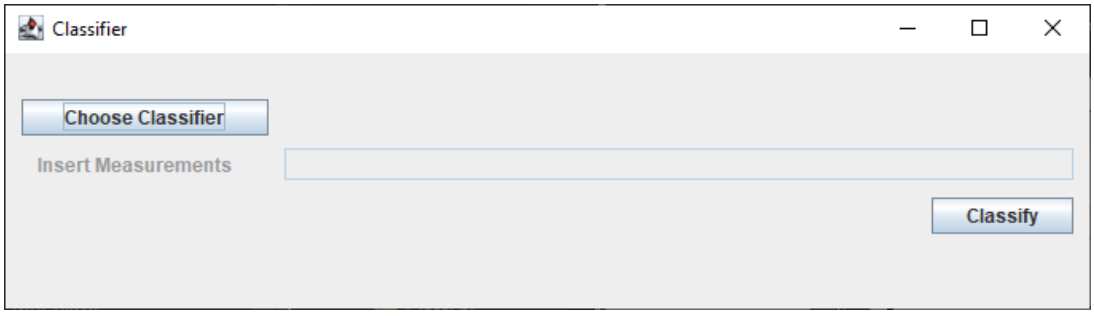
1. Execute, num IDE preparado para java, o ficheiro window1.java;
2. Após ter sido lançada a janela Classifier Creator, clique no botão Choose File... e selecione o dataset que deseja importar

3. Execute o algoritmo de aprendizagem, clicando no botão Import. Se nenhum erro[1] for sinalizado terá sido criado, na pasta Classifiers, um novo ficheiro, com o nome oSeuDataset.classifier.

Diagnóstico para Novos Dados

Para proceder ao diagnóstico de um novo conjunto de medições, necessita de um classificador e das próprias medições.

1. Execute, num IDE preparado para java, o ficheiro window2.java;



2. Após ter sido lançada a janela Classifier, clique no botão Choose File... e selecione o classificador que pretende utilizar. Observe que o *path* escolhido aparecerá ao lado do botão.
3. Uma vez selecionado o classificador, digite, no campo Insert Measurements as medições indicadas, separadas por vírgulas.

[1] Se receber a mensagem "Please choose a dataset", certifique-se de que escolheu um ficheiro .csv contendo o dataset

4. Execute o algoritmo clicando em Classify. Se tiver colocado dados inadequados, ser-lhe-á sinalizado[2], e pedida a correção. A classe esperada aparecerá ao lado de Class.

Implementações

Matrizes Esparsas

Dada a esparsidade dos tipos de dados a implementar (árvore, MRFT), optou-se por desenhá-los com base no artigo de Mikel Luján et al.[3], que compara diversas implementações e destaca aquela que foi usada no presente trabalho. Esta recorre ao uso de duas listas de listas, em que uma delas armazenará listas dos índices das colunas ocupadas, e a outra armazenará listas dos objetos que correspondem aos respectivos índices.

De Grafo Pesado a Árvore

Para implementar esta operação, correspondente ao Chow Liu, optou-se por recorrer ao algoritmo de Kruskal[4], ainda que utilizando algumas linhas orientadoras das aulas teóricas, onde foi leccionado o algoritmo de Prim.

Eficiência

Para uma análise da eficiência do algoritmo, sugere-se que se corra a função CrossValidation.java que imprime as matrizes de confusão (ver secção abaixo) e a duração do teste. Num processador Intel i7 de 2017, obteve-se:

```
Elapsed time to create 20 classifiers (trained on a combined number of ~ 4000 datapoints)
and test them on more than 800 sets of measurements (combined): 0.7980568 seconds.
```

Ainda assim, reconhecem-se inúmeras ineficiências, tais como a inexistência de um iterador pelos ramos, na classe MRFT (iterador incompatível com Serializer), e a escolha de um algoritmo reconhecidamente complexo para ordenação das arestas do grafo pesado.

Bibliotecas Externas

Para além das bibliotecas expectáveis para facilitar restantes implementações (ArrayList, Arrays, LinkedList, Queue, Stack), fizeram-se uso de inúmeras outras para implementar a interface, e ainda para proceder ao armazenamento e leitura de objetos e/ou documentos (Seriazable, File, Scanner, ...).

---

[2] Se receber a mensagem "Please choose a classifier", certifique-se de que escolheu um ficheiro contendo o classificador. Se receber a mensagem "Please insert values", certifique-se de que preencheu os valores das medições. Se receber a mensagem "Wrong number of measurements", certifique-se de que inseriu todas as medições necessárias. Se receber a mensagem "Measurement i out of range" verifique o i-ésimo valor inserido.

[3] https://link.springer.com/content/pdf/10.1007%2F11428831_45.pdf - figura 2

[4] https://en.wikipedia.org/wiki/Kruskal%27s_algorithm

Análise estatística

Por fim, fez-se uma análise estatística do desempenho dos classificadores para cada um dos datasets fornecidos, através do método de *cross validation*, usando a classe homónima. Cada Dataset original foi fatiado em 5 partes iguais sendo que, de seguida, 80% dos dados (4 partes) foram utilizados, de forma rotativa, para efeitos de treino e 20% (a restante parte) para efeitos de teste.

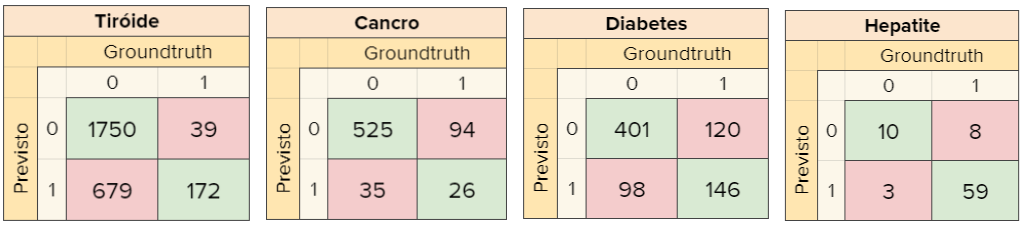


Figura 2. *Confusion Matrices* totais (somatório dos 5 testes) para cada tipo de dados

Tipo de Dados	Previsões Verdadeiras	Recall	Precision	F-score
Tiróide	72%	72%	97%	0.82
Cancro Cereb.	81%	93%	84%	0.89
Diabetes	71%	80%	76%	0.78
Hepatite	86%	76%	55%	0.64

Figura 3. Tabela resumo de métricas estatísticas do desempenho dos classificadores [5]

Conclusões

Por fim, o algoritmo aparenta correr em todos os casos com sucesso[6], com um F-score extremamente satisfatório, em décimas de segundos. Ainda assim, reconhecem-se fatores que poderiam ser alvo de melhorias.

[5] https://www.svds.com/the-basics-of-classifier-evaluation-part-1/

[6] Garantido pelo *CrossValidation*, que testa em todos os pontos do Datasets. Eventuais erros provavelmente poderão ser encontrados nesta classe de teste (*CrossValidation.java*), mas, por não ser um dos objetivos principais do trabalho, não sofreu testes de qualidade tão extensivos. Todas as etapas deste projeto foram executadas em Windows 10, pelo que eventuais erros podem decorrer noutros sistemas operativos (nomeadamente operações que interfiram com *paths* e parsing dos mesmos).