Aplicação de ID3 ou C4.5 ao diagnóstico de doenças do sistema urinário

Relatório Final



Inteligência Artificial

3º ano do Mestrado Integrado em Engenharia Informática e Computação

Fábio Filipe Jesus da Silva ei11107

João Manuel Mesquita Cardoso ei11100

28 de Maio de 2014

Índice

1. Objetivo………………………………………………………………………………….……..3

2. Especificação…………………………………………………………………………….…...4

2.1. *Dataset*…………………………………………………………………………………….....4

3. Desenvolvimento…………………………………………………………………………..6

3.1. Ambiente de Desenvolvimento……………………………………………….....6

3.2. Ferramentas e *API*s utilizadas……………………………………………………..6

3.3. Estrutura do programa…………………………………………………………...…..6

4. Experiências…………………………………………………………………………………...8

5. Conclusões…………………………………………………………………………………….9

6. Melhorias a implementar…………………………………………………………….10

7. Recursos………………………………………………………………………………………11

7.1. Bibliografia………………………………………………………………………………..11

7.2. *Software* utilizado…………………………………………………………………...…11

7.3. Elementos do grupo………………………………………………………………....11

8. Apêndice……………………………………………………………………………………...12

8.1. Manual do Utilizador……………………………………………………………......12

8.2. Exemplos de Execução……………………………………………………………...12

Objetivo

No âmbito da Unidade Curricular de Inteligência Artificial, optámos por desenvolver um projeto que passava pela implementação do algoritmo de aprendizagem *ID3* ou *C4.5*, aplicado ao diagnóstico de duas doenças do sistema urinário: inflamação na bexiga ou nefrite de origem na pelve renal.

Isto é conseguido derivando regras de classificação com base em casos anteriores, determinando para isso a árvore de decisão que traduz tais regras.

Tal escolha recaiu sobre este tema devido ao desafio atrativo de implementar um programa que conseguisse prever corretamente se o paciente padecesse de uma das potenciais patologias, mas também pela compreensão destes algoritmos que são importantes numa área bastante aliciante, *Data Mining*.

2 - Especificação

Optou-se por usar o algoritmo *C4.5* em detrimento do *ID3*, visto que existe um atributo contínuo (a temperatura do paciente) e o *ID3* não consegue lidar com variáveis contínuas, além de que é possível “podar” a árvore, o que diminui o número de testes necessários, bem como permite aumentar a proporção de classificações corretas e combater o problema do *overfitting1*. Optámos por utilizar a biblioteca da *WEKA* (*Waikato Environment for Knowledge Analysis)* para a implementação do algoritmo *C4.5*, representado pela classe *J48*.

O algoritmo *C4.5* faz a “poda” posteriormente à criação da árvore de decisão. Isto significa reduzir algumas subárvores a folhas, ou de outra forma, um ramo da árvore a partir de determinado nó é cortado (transformado em folha). O corte de um ramo da árvore é guiado por um teste estatístico que tem em conta os erros em um nó e a soma dos erros nos nós que descendem do mesmo. Assim, para cada nó, a poda só se concretiza se o desempenho da árvore não diminuir significativamente.

Com o *C4.5* é também possível realizar validação cruzada com dois ou mais grupos (*v-fold* ou validação *Jackknife*), melhorando desta forma a estimativa do erro cometido pelo utilizador. No entanto, tal só deve ser efetuado perante a ausência de um ficheiro de testes separados, o que não se verifica neste projeto.

2.1 – *Dataset*

Os dados usados para treinar e testar a nossa árvore de decisão estão albergados num ficheiro de texto, de modo a modularizar o programa e a separar a informação da parte da resolução do problema. O nosso *dataset* foi obtido do sítio web (http://archive.ics.uci.edu/ml/machine-learning-databases/acute/) fornecido pelos docentes desta Unidade Curricular no enunciado do nosso trabalho. A estrutura do mesmo foi posteriormente editada de modo a facilitar o seu uso com a biblioteca *WEKA*.

Devido à existência de duas conclusões possíveis - inflamação na bexiga ou nefrite de origem na pelve renal – foi necessário fazer um pré-processamento para fundir as duas, de modo a haver apenas uma dependência de variáveis. Como tal, a classe fundida resultante ficou com 4 valores possíveis: *yy* (o paciente padece das duas doenças), *nn* (o paciente não padece de nenhuma doença), *yn* (o paciente padece apenas de inflamação na bexiga) e por fim *ny* (o paciente padece apenas de nefrite de origem na pelve renal).

1- Termo, em aprendizagem de máquina, estatística e afins, para quando o modelo estatístico se ajusta em demasiado ao conjunto de dados/amostra. É comum que a amostra apresente desvios causados por erros de medição ou fatores aleatórios, ocorrendo o *overfitting* quando o modelo se ajusta a estes. Um modelo *overfitting* apresenta alta precisão quando testado com seu conjunto de dados. Porém, tal modelo não é uma boa representação da realidade e por isso deve ser evitado.

Aqui se apresenta um excerto do ficheiro, podendo ser verificada a forma como está organizado e a sua sintaxe:

**@relation disease**

**@attribute temperature numeric**

**@attribute nausea {yes, no}**

**@attribute lumbarPain {yes, no}**

**@attribute urinePushing {yes, no}**

**@attribute micturitionPains {yes, no}**

**@attribute burningOrItch {yes, no}**

**@attribute inflammationOfBladderOrNephritisOfRenalPelvis {yy, nn, yn, ny}**

**@data**

**35.0,no,yes,no,no,no,nn**

**35.9,no,no,yes,yes,yes,yn**

**35.9,no,yes,no,no,no,nn**

**36.0,no,no,yes,yes,yes,yn**

**36.0,no,yes,no,no,no,nn**

**36.0,no,yes,no,no,no,nn**

**36.2,no,no,yes,yes,yes,yn**

**36.2,no,yes,no,no,no,nn**

**36.3,no,no,yes,yes,yes,yn**

**36.6,no,no,yes,yes,yes,yn**

Como é possível observar neste excerto, o ficheiro contém duas partes fundamentais: *header*: onde se indica o nome da relação – neste caso *disease* – e os tipos de dados que serão lidos (atributos e valores possíveis dos mesmos), e a secção dos dados onde estão os vários dados para treinar e testar a árvore.

O nosso programa divide o *dataset* em duas partes em *runtime*, sendo 2/3 (80 instâncias) usado para treinar a árvore e 1/3 (40 instâncias) para a testar. Esta abordagem (divisão de ficheiros) pareceu-nos a mais correta, uma vez que com esta divisão evita-se a necessidade de processamento extra para testes, quando há apenas um ficheiro indiferenciado.

3 - Desenvolvimento

3.1 – Ambiente de Desenvolvimento

O programa foi desenvolvido utilizando o Sistema Operativo Windows 8 64-bits, utilizando o *IDE* (Integrated Development Environment) Eclipse com recurso à linguagem de programação Java.

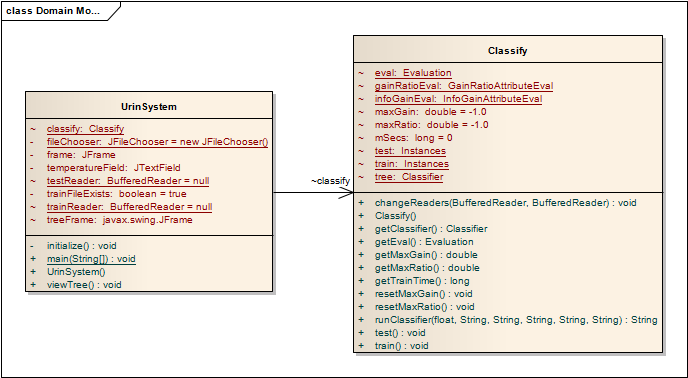
3.2 – Ferramentas e APIs utilizadas

O nosso programa foi desenvolvido com recurso à biblioteca da *WEKA* para a implementação do algoritmo *C4.5*.

3.3 – Estrutura do Programa

O nosso programa dispõe de dois módulos essenciais: o módulo da *GUI* (*Graphical User Interface*) – classe *UrinSystem* - e o módulo que trata da criação, treino e teste da árvore de decisão – classe *Classify*.

Aqui apresentamos o nosso diagrama de classes:



4 – Experiências

Foram usadas 40 instâncias de dados para testar a árvore de decisão, resultando num acerto de 100% dos casos. Como tal, podemos concluir que a árvore de decisão foi treinada com sucesso. Esta percentagem só acontece porque, além de serem poucos os casos de teste, todos eles estão de acordo com a árvore de decisão resultante do treino da mesma (o que era de esperar, visto virem da mesma fonte).

A experiência foi conduzida da seguinte forma: para cada linha do ficheiro de teste (cada uma correspondente a um caso), são lidos os vários atributos, sendo que o sucesso deste teste deriva da comparação entre o valor esperado (conclusão resultante da árvore de decisão) com o valor real, que se encontra no ficheiro de teste.

De acordo com os testes medidos, chegou-se à conclusão que a **Razão do Ganho de Informação** é a medida considerada para a escolha do atributo raiz da árvore. Chegou-se a esta conclusão comparando os valores desta medida com o **Ganho de Informação**, e verificou-se que apenas o valor da primeira estava presente no atributo de raiz e, ao mesmo tempo, era o maior de entre todos os atributos. Ou seja, o maior valor do **Ganho de Informação** não correspondia ao atributo de raiz, enquanto o contrário acontecia na **Razão do Ganho**.

5 – Conclusões

Na nossa opinião, pensamos que foram atingidas as metas estabelecidas no enunciado. Implementou-se uma interface gráfica para introduzir novos casos e ver estatísticas, bem como mostrar a árvore de decisão resultante, onde também é possível verificar quantos casos de treino se enquadram em cada uma das folhas.

Pensamos também que foram atingidos como sucessos todos os objetivos aos quais nos propusemos no Relatório Intercalar, e consideramos que o programa contém todas as características necessárias para uma boa predição destas duas doenças do sistema urinário.

Obviamente, realçamos a nossa aprendizagem destes algoritmos (*ID3* e *C4.5*) como o principal objetivo cumprido, e aquele que nos será mais útil no futuro.

6 – Melhorias a implementar

Como melhorias futuras, podia-se permitir uma gama maior de ficheiro de treino e teste, dados estatísticos de todos os atributos, assim como talvez uma maior variedade de dados estatísticos de diferentes tipos.

Outra melhoria possível seria a implementação de raiz do algoritmo C4.5, em detrimento do uso da implementação fornecida pela *WEKA*.

7 – Recursos

7.1 – Bibliografia

1. Czerniak, Jacek, Ficheiro de dados para treino e teste disponível em: http://archive.ics.uci.edu/ml/datasets/Acute+Inflammations
2. Oliveira, Eugénio – Capítulo 6 – Aprendizagem Simbólica Automática Porto [Consult. 09-04-2014 – 29-05-2014]. Disponível em: http://paginas.fe.up.pt/~eol/IA/1314/APONTAMENTOS/6\_ASA.pdf
3. Tutorial da WEKA [Consult. 11-04-2014]. Disponível em: http://weka.wikispaces.com/Use+WEKA+in+your+Java+code
4. Documentação da WEKA [Consult. 11-04-2014 – 29-05-2014]. Disponível em: http://weka.sourceforge.net/doc.dev/overview-summary.html

7.2 – *Software* Utilizado

* Biblioteca WEKA v3.7 disponível em: http://weka.wikispaces.com/
* Eclipse IDE disponível em: http://www.eclipse.org/
* Java 7

7.3 – Elementos do Grupo

Cada um de nós contribuiu de forma igual para o maior sucesso possível do nosso projeto.

8 – Apêndice

8.1 – Manual do Utilizador

A lista de passos a efetuar para testar o nosso programa é a seguinte:

* Descompactar o ficheiro .zip enviado
* Correr o ficheiro UrinSystem.java que se encontra dentro da pasta gui, e que esta se encontra dentro da pasta src (UrinSyst/src/gui/UrinSystem.java)
* Após correr o ficheiro UrinSystem.java irão aparecer 2 janelas ao utilizador, uma que contém a árvore de decisão resultante, contendo uma GUI onde são apresentado várias estatísticas sobre o programa, bem como introduzir novos casos para teste e mudar o ficheiro de treino ou de teste
* Para introduzir um novo caso, é necessário introduzir os sintomas do novo caso nos sítios designados (*text* *box* para a temperatura, e *check* *boxes* para os restantes sintomas) e posteriormente carregar no botão TEST
* Para mudar o ficheiro de teste é apenas preciso carregar no botão Change Test e escolher o ficheiro pretendido
* Para mudar o ficheiro de treino é apenas preciso carregar no botão Change Train e escolher o ficheiro pretendido
* Depois de fazer a ação pretendida, pode repetir todo o processo se desejar
* Caso deseje terminar o programa basta fechar a janela da onde é possível introduzir novos casos

8.2 – Exemplos de Execução

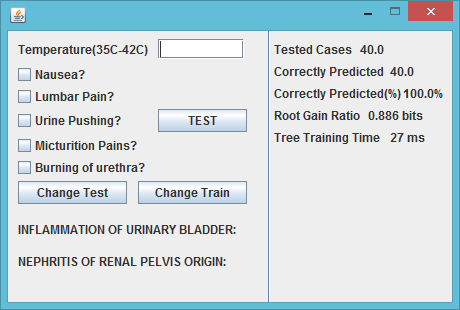
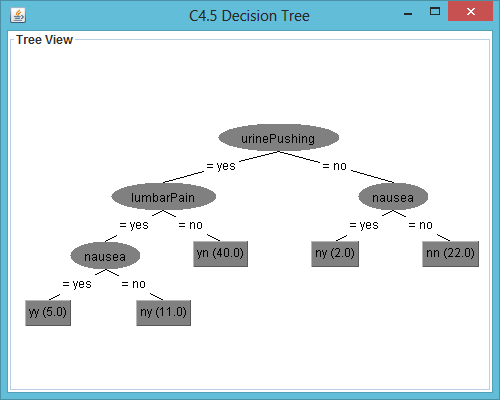


Fig. 1 – Janela Principal do programa



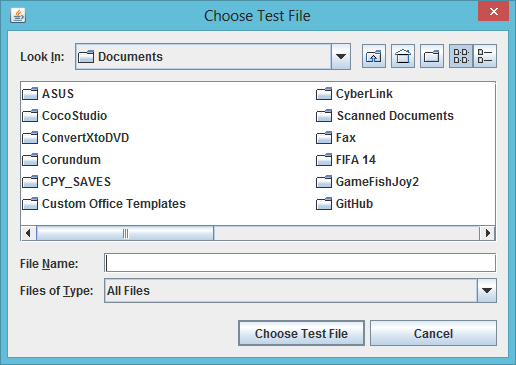


Fig. 2 – Visualizador da árvore de decisão

Fig. 3 – Escolha do novo ficheiro de teste

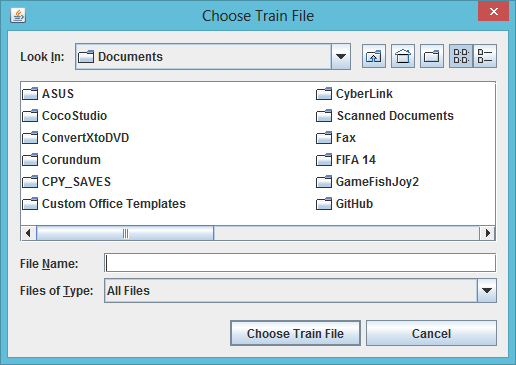


Fig. 4 – Esolha do novo ficheiro de treino

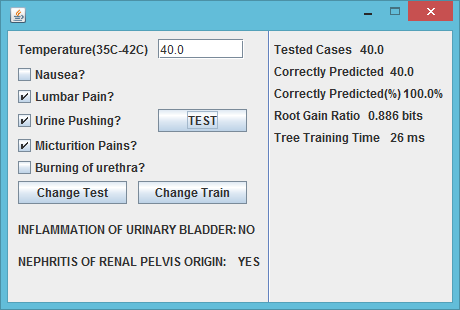


Fig. 5 – Resultado da inserção de um novo caso