Universidade do Porto

Faculdade de Engenharia

FEUP

Aplicação de C4.5 à identificação de Pulsars

Relatório Intercalar

Inteligência Artificial - 3º Ano

3º ano do Mestrado Integrado em Engenharia Informática e Computação

Grupo E2_3:
Afonso Bernardino da Silva Pinto - 201503316 - up201503316@fe.up.pt
Bruno Miguel de Sousa Pinto - 201502960 - up201502960@fe.up.pt
Tomás Sousa Oliveira - 201504746 - up201504746@fe.up.pt

23 de Agosto de 2018

Conteúdo

1 Objetivo

Este projeto foi realizado no âmbito da unidade curricular Inteligência Artificial, do 3º ano do Mestrado Integrado em Engenharia Informática e Computação da Faculdade de Engenharia da Universidade do Porto.

O objetivo deste projeto é implementar uma árvore de decisão que permita a identificação de Pulsars, aplicando o algoritmo C4.5.

O presente relatório servirá para explorar as particularidades deste projeto e fornecer detalhes sobre a sua implementação.

2 Especificação

Os pulsares são um tipo raro de estrela de neutrões com considerável interesse científico que produzem emissões de rádio detetáveis na Terra.

À medida que os pulsares giram, o seu feixe de emissão varre o céu e, quando cruza a nossa linha de visão, produz um padrão detetável de emissão de rádio de banda larga. Como os pulsares giram rapidamente, esse padrão repete-se periodicamente. Assim, a procura por pulsares baseia-se na procura por sinais de rádio periódicos, com grandes telescópios.

Cada pulsar produz um padrão de emissão ligeiramente diferente, variando a cada rotação. Assim, uma deteção de sinal candidata é calculada pela média de muitas rotações do pulsar, conforme determinado pelo comprimento de uma observação.

Na ausência de informações adicionais, cada candidato poderia potencialmente descrever um pulsar real, no entanto, na prática, quase todas as deteções são causadas por interferência de radiofrequência (RFI) e ruído (N), dificultando a localização de sinais legítimos.

Figura 1: Pulsar PSR B1509-58

2.1 Descrição do dataset

O HTRU2 é um conjunto de dados que descreve uma amostra de candidatos, colecionados durante a High Time Resolution Universe Survey (South).

A amostra contém 16.259 exemplos espúrios, causados por RFI/N, e 1.639 exemplos reais de pulsar, todos eles humanamente verificados. Podemos assim afirmar que este *dataset* é claramente desbalanceado, uma vez que possui cerca de dez vezes mais exemplos espúrios que exemplos reais de pulsar.

Cada candidato é descrito por oito variáveis contínuas e uma variável binária de classe. Os quatro primeiros são estatísticas simples, obtidas do perfil de pulso integrado, uma matriz de variáveis contínuas que descrevem uma versão do sinal resolvido em longitude, calculada em tempo e frequência. As quatro variáveis restantes são obtidas da mesma maneira, mas da curva Medida de Dispersão – Relação Sinal-Ruído (DM-SNR).

1. Média do perfil integrado;
2. Desvio padrão do perfil integrado;
3. Excesso de curtose do perfil integrado;
4. Assimetria do perfil integrado;
5. Média da curva DM-SNR;
6. Desvio padrão da curva DM–SNR;
7. Excesso de curtose da curva DM–SNR;
8. Assimetria da curva DM–SNR;
9. Classe.

2.2 Análise do datsaset

——	**Média**	**Desvio Padrão**	**Excesso de curtose**	**Assimetria**
RFI/N	116.5627263	47.33974078	0.210440066	0.380843987
Pulsar	56.69060784	38.71059825	3.130655385	15.55357603

Tabela 1: Média das estatísticas do Perfil Integrado

Como é possível observar, os valores da média para a pulsares reais são aproximadamente inferiores em metade aos dos exemplos espúrios.

——	**Média**	**Desvio Padrão**	**Excesso de curtose**	**Assimetria**
RFI/N	8.863258445	23.28798388	8.862673718	113.6203437
Pulsar	49.82599513	56.46896295	2.757068576	17.93172838

Tabela 2: Média das estatísticas da curva DM–SNR

Neste caso, os valores da média para a pulsares reais são aproximadamente seis vezes superiores aos dos exemplos espúrios.

2.3 Pré-processamento dos dados

Antes de serem utilizados, os dados fornecidos são dispostos numa ordem aleatória, impedindo assim que a ordem destes tenha influência sobre as regras da árvore. Para além disso, os dados foram normalizados, ficando assim com valores compreendidos no intervalo [0,1].

O grupo optou por utilizar as técnicas de validação cruzada em detrimento de um também típico *random split* de 75\25 % no dataset, que corresponderiam a um conjunto de treino e conjunto de testes, respetivamente. (ver secção 5).

No que toca aos problemas de balanceamento várias técnicas foram estudadas, das quais salientamos os filtros Smote (*Synthetic Minority Oversampling Technique*) e o ClassBalancer, da biblioteca Weka.

Enquanto o Smote deteta as instâncias minoritárias e cria novas instâncias próximas das anteriores, o ClassBalancer redistribui o peso das instâncias de dados de modo que cada classe (pulsar ou não) tenha o mesmo na sua totalidade.

O Smote foi o processo utilizado pelo grupo para tornar o nosso processamento mais equilibrado e foi o que teve melhor resultados como demonstrado nas experiências (ver secção 4.1.11).

2.4 Modelo de Aprendizagem

As árvores de decisão são um método de aprendizagem supervisionada muito efetivo, utilizadas para construir modelos de classificação ou regressão com a estrutura de uma árvore. O objetivo é dividir um conjunto de dados em subconjuntos cada vez menores, enquanto outra árvore de decisão é desenvolvida de forma incremental. O resultado final esperado é uma árvore com nós de decisão e folhas a representar uma classificação.

A nossa implementação contempla uma árvore de classificação binária onde cada nó de decisão tem exatamente 2 ramos subsquentes e onde o *root node* corresponde à decisão com maior *information gain*, i.e a decisão que mais reduz a entropia do conjunto.

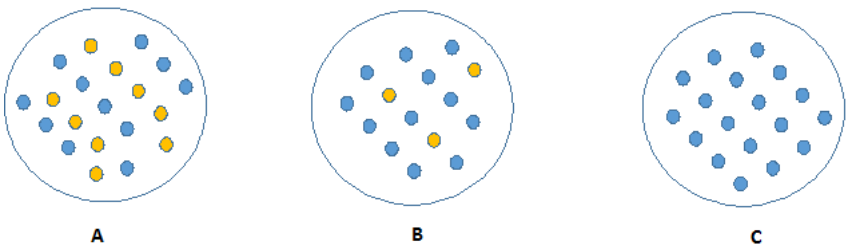

Figura 2: Analogia que representa que menor entropia requer menos informação para distinguir o conjunto

2.5 Algoritmo Utilizado

São vários os algoritmos que permitem construir árvores de decisão, no entanto, aqueles que são mais frequentemente utilizados são: CART, ID3 e C4.5. O algoritmo escolhido pelo nosso grupo foi o C4.5. Este algoritmo é visto como o sucessor do ID3, mas sem a restrição de apenas poder ser utilizado com dados discretos.

A construção da árvore segue os seguintes passos:

1. Se todos os dados de um conjunto de dados S pertencer à mesma classe, a árvore é classificada com a classe em questão;

2. Se não, avalia-se um dos atributos, criando um ramo para cada uma das possíveis classificações, parte-se o conjunto S nos correspondentes subconjuntos S1, S2 e aplica-se o mesmo procedimento de forma recursiva, em cada um dos subconjuntos.

A forma de avaliar o atributo neste último segue uma das seguintes heurísticas:

- *Information Gain*: minimiza o total de entropia nos subconjuntos (não é muito útil em casos com várias classificações possíveis);

- *Ratio Gain* : razão entre o tópico acima e as classificações possíveis.

A árvore resultante pode depois ser podada para evitar *overfiting* (sobreajuste ao conjunto de dados). Este processo tem como base uma estimativa pessimista da razão entre o número de dados que não pertencem à classe classificada no ramo E e o número de dados N.

Este processo é realizado apenas numa iteração das folhas para a raiz. O rácio supracitado é adicionado ao longo dos ramos, comparando-se o resultado com o rácio que existiria se o nó passasse a folha. Se este último não for maior que o primeiro, a árvore é podada.

2.6 Variantes Consideradas

Além do C4.5 foi ponderada também a utilização do algoritmo ID3. Contudo este último é visto como uma versão inferior do primeiro, apresentando as seguintes desvantagens:

- Apenas aceita dados discretos.

- Não aplicável aquando de conjuntos de dados incompletos.

- Maior tendência para *overfitting*.

3 Desenvolvimento

3.1 Linguagens e Ferramentas Utilizadas

Elaborado e testado tanto em Linux (*Manjaro*) como em Windows (*10*) este projeto, desenvolvido em Java 8 num ambiente de desenvolvimento integrado (*IntelliJ*), tem como bibliotecas externas o *Weka 3.9.2* - um software de prospecção de dados que fornece algoritmos de *machine learning*.

3.2 Estrutura da Aplicação

A aplicação encontra-se estruturada em 2 módulos:

- GUI - Responsável por gerar a interface gráfica do utilizador.
- Decision Tree - Responsável por construir efetivamente a árvore de decisão, incluindo processamento de dados, funções de avaliação e classificação.

As 3 etapas realizadas pela classe *Decision Tree* são expostas de seguida:

1. Processamento de dados: O processamento de dados é realizado no construtor da árvore de decisão, aqui são criados os filtros de balanceamento, baralhados os dados e gerada a árvore com o algoritmo c4.5.

```
DecisionTree(String filePath) {
    try {
        dataset = new ConverterUtils.DataSource(filePath).getDataSet();
        dataset.setClassIndex(dataset.numAttributes() - 1);

        smote = new SMOTE();
        smote.setInputFormat(dataset);
        Random random = new Random(Double.doubleToLongBits(Math.random()));
        dataset.randomize(random);
        filterNorm.setInputFormat(dataset);
        dataset = Filter.useFilter(dataset, filterNorm);
    } catch (Exception e) {
        System.out.println("Couldn't load data set");
        e.printStackTrace();
    }
    loadTree();
}

private void loadTree() {
    try {
        tree.setOptions(treeOptions);
        filteredClassifier.setFilter(smote);
        filteredClassifier.setClassifier(tree);
        filteredClassifier.buildClassifier(dataset);
    } catch (Exception e) {
        System.out.println("Couldn't load tree");
        e.printStackTrace();
    }
}
```

Figura 3: Construtor da Árvore de Decisão

2. Função de avaliação: Esta função é a responsável por aplicar a validação cruzada tendo em consideração os filtros utilizados na construção da árvore. Retorna uma *String* com as estatísticas da avaliação.

```
public String score() {
    Evaluation eval = null;
    try {

        eval = new Evaluation(dataset);
        eval.crossValidateModel(filteredClassifier, dataset, numFolds: 10, new Random( l: 1));
    } catch (Exception e) {
        e.printStackTrace();
    }

    assert eval != null;
    return eval.toSummaryString();

}
```

Figura 4: Função de Avaliação

3. Função de classificação: Estas funções são responsáveis por classificar novas instâncias de dados segundo a árvore gerada. Enquanto uma aceita como input uma *String* que indique o caminho de um ficheiro .arff a outra aceita os dados necessários expressados num *array* de *doubles*. Retorna um *array* de *doubles* com 2 entradas que representam respetivamente as probabilidades de não ser e ser pulsar.

```
public double[] classify(ClassifyData classifyData) {
    double[] values = classifyData.getValues();

    int valuesSize = dataset.numAttributes() - 1;
    if (values.length != valuesSize)
        return values;

    Instances unlabeled = new Instances(dataset, capacity: 0);
    unlabeled.setClassIndex(unlabeled.numAttributes() - 1);

    Instance newInstance = new DenseInstance(valuesSize);
    for (int i = 0; i < valuesSize; i++) {
        newInstance.setValue(i, values[i]);

    }
    unlabeled.add(newInstance);
    try {
        return tree.distributionForInstance(unlabeled.firstInstance());
    } catch (Exception e) {
        e.printStackTrace();
    }

    return new double[]{0.0,0.0};
}

public ArrayList<Pair<Double, Double>> classify(String filePath) {
    ArrayList<Pair<Double, Double>> ret = new ArrayList<>();

    try {
        Instances unlabeled = new Instances(
                new BufferedReader(
                        new FileReader(filePath)));
        unlabeled.setClassIndex(unlabeled.numAttributes()-1);
        for (int i = 0; i < unlabeled.numInstances(); i++) {
            double[] clsLabel = tree.distributionForInstance(unlabeled.instance(i));
            Pair<Double,Double> pair;
            pair = new Pair<>(clsLabel[0], clsLabel[1]);
            ret.add(pair);
        }
    } catch (Exception e) {
        e.printStackTrace();
    }


    return ret;
}
```

Figura 5: Função de Classificação

É apresentado de seguida o diagrama de classes do projeto:

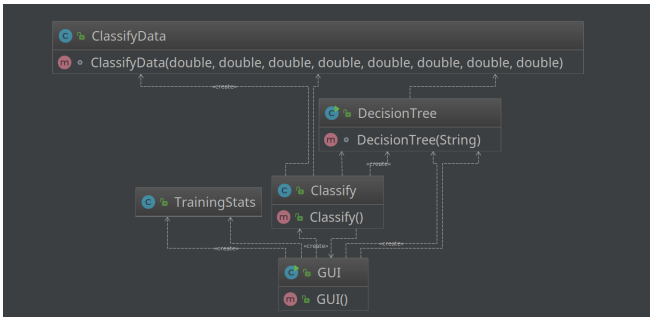

Figura 6: Diagrama de Classes

3.3 Detalhes Relevantes

Foi criado uma interface gráfica de utilização para simplificar a geração da árvore a partir de ficheiros - graças à intercomunicação implementada com o sistema de ficheiros do computador - bem como para facilitar a interação com a árvore gerada.

Figura 7: GUI

4 Experiências

4.1 Opções da Árvore

4.1.1 *Pruning*

Numa primeira análise, foi considerada a hipótese de usar uma árvore não-podada (*unpruned*) no lugar de uma árvore podada (*pruned*), usada por *default*. As árvores *unpruned* são mais largas, uma vez que não passam pelo passo adicional de verficicação dos nós e ramos que podem ser removidos, sem que haja um grande decréscimo no desempenho, pelo qual passam as árvores *pruned*. Para além disso, a poda (*pruning*) pode ser usada como ferramenta que impeça um potencial *overfitting*, ou seja, deixar a árvore de certa forma "viciada"nos dados de treino, tal levando a uma redução na eficácia de avaliação de outros dados.

Os resultados obtidos após três experiências tiveram em média os resultados seguintes:

——	**Tempo Gasto a Construír o Modelo (s)**	**Instâncias Classificadas de Forma Correta (%)**	**Tamanho da Árvore**	**Número de Folhas**
Pruned	0.34	97.8433	49	25
Unpruned	0.26	97.7819	65	33

Concluímos que a decisão entre uma árvore *unpruned* ou *pruned* terá sempre que ver com a escolha entre *performance* ou eficácia de classificação, respetivamente. A partir desta subsubsecção usaremos uma árvore podada para levar a cabo as nossas experiências.

4.1.2 *Tree Collapse*

De seguida, procedeu-se à verificação da influência de usar uma *non-collapsed tree* no lugar da *collapsed tree* que é usada por predefinição. Resumidamente, o colapso da árvore é uma poda feita a partir da minimização de erros de classificação nos dados de treino.

Os resultados obtidos para os dois tipos de árvore foram idênticos.

4.1.3 *Pruning Confidence*

O teste ao nível de confiança nos intervalos de confiança usados na poda da árvore foi feito usando três valores: 0.125, 0.25 (*default*) e 0.5. Quanto mais baixo for o valor mais amplo será o intervalo de confiança e mais pessimistas serão as estimativas, produzindo uma pode mais pesada.

Os resultados obtidos foram os seguintes:

——	**Tempo Gasto a Construír o Modelo (s)**	**Instâncias Classificadas de Forma Correta (%)**	**Tamanho da Árvore**	**Número de Folhas**
0.125	0.29	97.8322	35	18
0.25	0.24	97.8433	49	25
0.5	0.26	97.8154	61	31

Verificamos assim que o *pruning confidence* ideal é o de 0.25, uma vez que não só é aquele que gasta menos tempo a construír o modelo como ainda é o que tem maior percentagem de instâncias classificadas de forma correta. Daqui em diante é esse o *pruning confidence* usado.

4.1.4 Número Mínimo de Instâncias por Folha

Este parâmetro força a que pelo menos dois dos nós sucessores de um *split* tenham, pelo menos, este número mínimo de instâncias por folha para que um *split* seja permitido. Foram testados os valores 1, 2 (*default*) e 4.

Os resultados obtidos foram os seguintes:

——	**Tempo Gasto a Construír o Modelo (s)**	**Instâncias Classificadas de Forma Correta (%)**	**Tamanho da Árvore**	**Número de Folhas**
0.125	0.26	97.8545	51	26
2	0.23	97.8433	49	25
4	0.27	97.8657	41	21

Uma vez que o valor 4 é aquele que oferece uma maior percentagem de instâncias classificadas de forma correta, este será usado para as experiências levadas a cabo daqui em diante.

4.1.5 *Reduced-error Pruning*

Há a possibilidade de usar poda de redução de erros *reduced-error pruning* no lugar da poda baseada em erro *error-based pruning*, usado por predefinição.

Tendo em conta esta hipótese levamos a cabo a experiência da qual obtivemos estes resultados:

——	**Tempo Gasto a Construír o Modelo (s)**	**Instâncias Classificadas de Forma Correta (%)**	**Tamanho da Árvore**	**Número de Folhas**
Reduced-error Pruning	0.15	97.8042	21	11
Error-based Pruning	0.24	97.8657	41	21

Os resultados mostram-nos que o *error-based pruning* nos oferece uma melhor exatidão de resultados, sendo por isso a que será usada a partir daqui.

4.1.6 *Number of Folds*

O tamanho do conjunto de poda também pode ser ajustado, sendo que o seu valor *default* é 3. Experimentamos então, junto com este, os valores 2 e 6, tendo obtido resultados idênticos para as três situações.

Ficamos então com o valor por predefinição.

4.1.7 *Divisões Binárias*

É possível forçar a que as divisões (*splits*) sejam feitas de forma binária.

Os valores obtidos foram, de novo, semelhantes, pelo que decidimos não usar esta opção nas experiências seguintes.

4.1.8 Não Executar *Subtree Raising*

Há duas operações de *possíveis*: *subtree replacement*, ativada quando desativamos *subtree raising*, usada por *default*.

Da experiência das duas operações resultaram os seguintes valores:

——	**Tempo Gasto a Construír o Modelo (s)**	**Instâncias Classificadas de Forma Correta (%)**	**Tamanho da Árvore**	**Número de Folhas**
Subtree Replacement	0.34	97.8266	41	21
Subtree Raising	0.24	97.8657	41	21

Uma vez que o uso de *subtree replacement* no lugar de *subtree raising* se revelou menos exata, decidimos usar a operação ativa por predefinição.

4.1.9 Não Usar a Correção MDL para Ganho de Informação em Atributos Numéricos

O uso da correção MDL para ganho de informação em atributos numéricos pode levar a que a divisão não seja permissível, sendo assim possível desligar esta opção.

Dividimos os resultados em "Com Correção"e "Sem Correção", obtendo os seguintes resultados:

——	**Tempo Gasto a Construír o Modelo (s)**	**Instâncias Classificadas de Forma Correta (%)**	**Tamanho da Árvore**	**Número de Folhas**
Sem Correção	0.5	97.8266	103	52
Com Correção	0.28	97.8657	41	21

Para além de gerar uma árvore mais de duas vezes maior, o facto de não usarmos a correção MDL leva a que haja uma menor percentagem de instâncias classificadas de forma correta, pelo que decidimos continuar a usar esta correção.

4.1.10 Semente

Por último, levamos a cabo uma experiência que visa procurar qual a melhor semente para o gerador de números aleatórios para baralhar os dados, antes de o conjunto de poda ser separado.

Foram usados os valores 0.5, 1 (*default*) e 2, tendo obtido resultados semelhantes.

4.1.11 Balanceamento de Dados

Os dois filtros mais adequados para o balanceamento de dados são o SMOTE e o ClassBalancer. Obtivemos os seguintes resultados, usando esses filtros:

——	**Tempo Gasto a Construír o Modelo (s)**	**Instâncias Classificadas de Forma Correta (%)**	**Tamanho da Árvore**	**Número de Folhas**
Sem filtros	0.28	96.465	41	21
ClassBalancer	0.28	95.8375	41	21
SMOTE	0.35	97.8042	41	21

5 Resultados Esperados e Forma de Avaliação

Esperamos que o nosso programa seja capaz de distinguir exemplos reais de pulsares de exemplos espúrios.

No que toca à forma de avaliar a qualidade da árvore de decisão gerada optámos por utilizar a técnica de validação cruzada. Este processo que consiste em dividir o conjunto de dados em k (no nosso caso 10) subconjuntos mutuamente exclusivos e com o mesmo tamanho, após esta etapa um dos subconjuntos é utilizado para testes e os k-1 (9) restantes são utilizados para treino. Este processo é repetido até que todos os k conjuntos tenham sido o subconjunto de teste. A cada iteração é calculada a precisão do modelo para aquela seleção. A precisão final do modelo é estimada através da média dos resultados destas k iterações. Embora computacionalmente mais exigente, esta abordagem permite um uso mais eficiente do *dataset* utilizado.

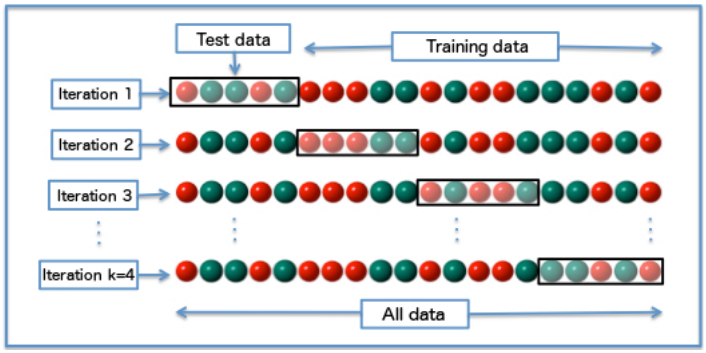

Figura 8: Diagrama de validação cruzada *k-fold* com k = 4

6 Conclusões

Com a realização deste projeto pudemos familiarizar-mos-nos com a técnicas de *machine learning* mais concretamente no que diz respeito à geração de árvores de decisão de uma forma geral e em particular usando uma das bibliotecas mais conceituadas e utilizadas deste campo de estudo - Weka. Tendo em conta que os dados foram cuidadosamente analisados e pré-processados, foi aplicado o algoritmo C4.5. na geração da árvore, as regras de classificação se encontram dispostas na visualização da árvore resultante, é estimado de forma detalhada a precisão do nosso classificador e é possível classificar novos casos, consideramos que todos os objetivos propostos foram cumpridos.

7 Melhoramentos

Em última análise, o grupo delineou alguns possíveis melhoramentos tais como:

- A elaboração de gráficos no que toca à disposição das estatísticas em detrimento do texto simples atual.
- A disposição gráfica dos dados a classificar para quando lida a partir de ficheiros.
- Possibilitar a exportação da árvore de decisão em diferentes formatos (imagem, dot, etc).
- Possibilitar a leitura de ficherios .csv para além dos ficheiros .arff.
- Possibilitar ao utilizador uma maior personalização da árvore a gerar.

8 Recursos

8.1 Referências

Udacity. (23 fevereiro 2015). Cross Validation. Disponível em: https://www.youtube.com/watch?v=sFO2ff-gTh0. Acesso em: 8 abril 2018

Udacity. (23 fevereiro 2015). K-Fold Cross Validation - Intro to Machine Learning. Disponível em: https://www.youtube.com/watch?v=TIgfjmp-4BA. Acesso em: 8 abril 2018

Singh, Aishwarya, Faizan Shaikh, Pranav Dar, and Analytics Vidhya Content Team. "A Complete Tutorial on Tree Based Modeling from Scratch."Analytics Vidhya. 2 de Maio de 2017. Acesso em: 8 abril 2018 https://www.analyticsvidhya.com/blog/2016/04/complete-tutorial-tree-based-modeling-scratch-in-python/.

"Cross-validation (statistics)."Wikipedia. 5 de Abril de 2018. Acesso em: 8 abril 2018 https://en.wikipedia.org/wiki/Crossvalidation_(statistics).

"Information Gain in Decision Trees."Wikipedia. 28 de Fevereiro de 2018. Acesso em: 8 abril 2018 https://en.wikipedia.org/wiki/Information_gain_in_decision_trees.

Decision Tree. Acesso em: 8 abril 2018 http://www.saedsayad.com/decision_tree.htm.

"Use Weka in Your Java Code."Weka. Acesso em: 8 abril 2018 https://weka.wikispaces.com/Use Weka in your Java code.

J48. 22 de Dezembro de 2017. Acesso em: 8 abril 2018 http://weka.sourceforge.net/doc.dev/weka/classifiers/trees/J48.html.

"Decision Tree Learning."Wikipedia. 7 de Abril de 2018. Acesso em: 8 abril 2018 https://en.wikipedia.org/wiki/Decision_tree_learning.

"1.10. Decision Trees."1.10. Decision Trees - Scikit-learn 0.19.1 Documentation. Acesso em: 8 abril 2018 http://scikit-learn.org/stable/modules/tree.html#tree-algorithms-id3-c4-5-c5-0-and-cart.

Kumar, Vipin, and Xindong Wu. The Top Ten Algorithms in Data Mining:. Boca Raton, FL: Chapman & Hall/CRC Press, 2009.

"What Is Pruned and Unpruned Tree in Weka?"Stack Overflow. Acesso em: 18 maio 2018 https://stackoverflow.com/questions/11585715/what-is-pruned-and-unpruned-tree-in-weka.

"Details of J48 Pruning Parameters."WEKA - Details of J48 Pruning Parameters. Acesso em: 18 maio 2018 http://weka.8497.n7.nabble.com/Details-of-J48-pruning-parameters-td42456.html.

"Classification Problem Using Imbalanced Dataset."Cross Validated. Acesso em: 18 maio 2018 https://stats.stackexchange.com/questions/81111/classification-problem-using-imbalanced-dataset.

"ImbalanceProcessingAve Java Source Code."Java Code Examples. Acesso em: 18 maio 2018 https://www.programcreek.com/java-

api-examples/?code=Gu-Youngfeng/CraTer/CraTer-
.master/src/sklse/yongfeng/experiments/ImbalanceProcessingAve.java.
"ClassBalancer."RandomForest. December
22, 2017. Acesso em: 18 maio 2018
http://weka.sourceforge.net/doc.dev/weka/filters/supervised/instance/ClassBalancer.html.
"SMOTE."RandomForest. December 22, 2017. Acesso em: 18 maio 2018
http://weka.sourceforge.net/doc.packages/SMOTE/weka/filters/supervised/instance/SMOTE.html.
"Programmatic Use."Weka. Acesso em: 18 maio 2018
https://weka.wikispaces.com/Programmatic Use.

8.2 Software

- Linguagem de programação Java 1.8
- Biblioteca Weka 3.1.9.

8.3 Percentagem de trabalho efectivo

- Afonso Pinto - 33%
- Bruno Pinto - 33%
- Tomás Oliveira - 33%

9 Apêndice

9.1 Manual do Utilizador

1. Ao iniciar o programa é mostrada a seguinte janela:

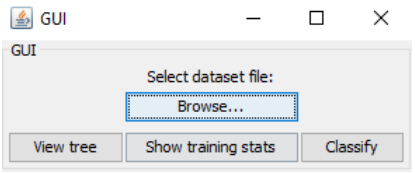

Figura 9: GUI do programa

2. Antes de usarmos qualquer uma das funções disponíveis: *View tree*, *Show training stats* e *Classify*, temos que selecionar o ficheiro que contém o *dataset*.

   (a) *View tree* - Mostra a árvore de decisão, construída a partir dos dados do ficheiro selecionado previamente.

   (b) *Show training stats* - Mostra as estatíticas dos dados de treino do ficheiro selecionado.

   (c) *Classify* - Permite selecionar entre obter classificações para um conjunto de dados contidos num ficheiro ou para um conjunto de dados fornecido, como é possível ver na janela seguinte:

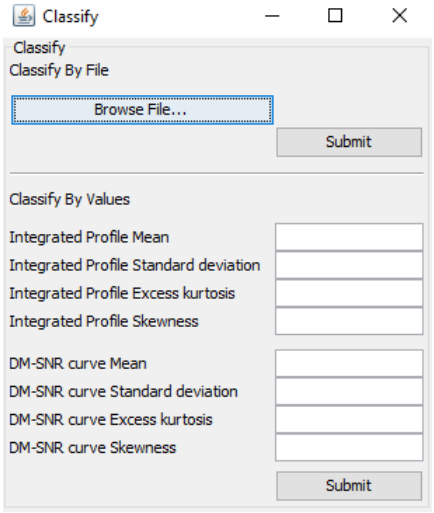

Figura 10: GUI da função *Classify*