**Implementação de Árvores de Decisão**

FCUP

Inteligência Artificial (CC2006) 2018/2019

Trabalho de Grupo AK

Eduardo Morgado – 201706894 – MIERSI

Simão Cardoso – 201604595 – MIERSI

Sónia Rocha – 201704679 – MIERSI

1. **Introdução:**
   1. **Uma Aprendizagem Supervisionada:**

Para um qualquer problema/ambiente, existem variáveis possíveis de estudar () e, a partir de uma dada função , podemos concluir/chegar a uma solução para o problema inicial, no entanto, encontrar , a função que mapeia as variáveis para as suas imagens/soluções, é uma tarefa difícil, uma vez que, para um problema do dia-a-dia como por exemplo, a escolha do meio de transporte baseado no tempo e trânsito, essa função não é diretamente observável. O objetivo da aprendizagem máquina é então, encontrar essa função para um qualquer problema.

Existem inúmeros métodos/algoritmos de aprendizagem máquina, tal como a *Figura 1* mostra, os algoritmos que vamos estudar são algoritmos para árvores de decisão.

Uma imagem com texto

Descrição gerada automaticamente

Figura - Mapa geral de algoritmos de aprendizagem máquina (imagem retirada de **[1]**)

Tal como referido anteriormente, um problema é composto por um conjunto de observações/variáveis e pela sua solução/*outcome*, um problema pode ser facilmente resolvido caso a função que mapeie as suas variáveis, , seja conhecida, no entanto, para a maior parte dos problemas essa função é desconhecida, apenas podemos observar as variáveis e o seu resultado, com isso, surgem diferentes técnicas de aprendizagem, baseadas na forma como tratam os dados. As técnicas de aprendizagem máquina podem sem divididas em 3 grandes grupos (*Figura 2*):

* Aprendizagem supervisionada: os dados, as sequências de observações estão rotuladas;
* Aprendizagem semi-supervisionada(*reinforcement*): a sequência de observações possuí dados rotulados e dados não rotulados;
* Aprendizagem não supervisionada: a sequência de observações não tem rótulos

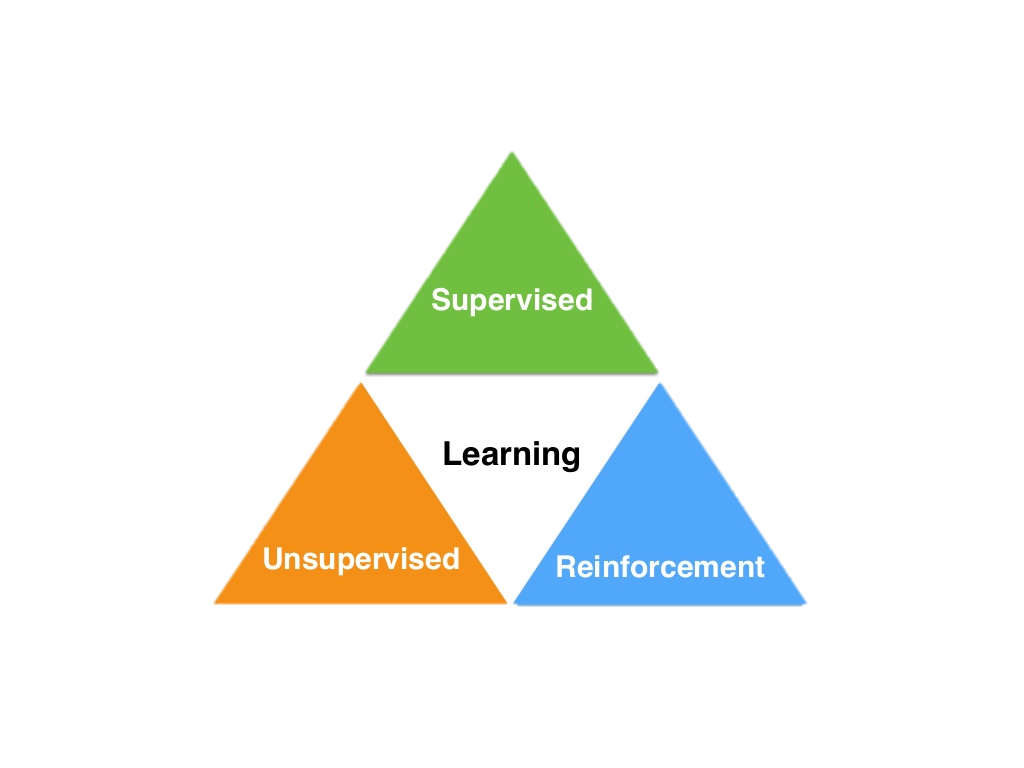


Figura - Distribuição geral de algoritmos de aprendizagem

Sendo assim, na aprendizagem supervisionada, possuímos um conjunto de observações/variáveis (*dataset*), , e um conjunto de classificações corretas (*outcome*), , e os algoritmos vão concluir uma aproximação da função que mapeia os dados, , de forma a, para uma nova observação para o problema possamos concluir o resultado de forma correta, utilizando essa nova função **[1]**.

Os problemas de aprendizagem supervisionada podem ser de dois tipos **[1]**:

* *Classification*: o resultado do problema é uma variável categórica[[1]](#footnote-1);
* *Regression*: o resultado do problema é uma variável contínua[[2]](#footnote-2)
  1. **O que é uma Árvore de Decisão:**

As árvores de decisão enquadram-se na aprendizagem máquina supervisionada. As árvores de decisão podem ser utilizadas para resolver problemas de *classification* e *regression* **[7]**. São muito utilizadas, não só pelo facto de mostrarem como resolver o problema (a deslocação na árvore através de regras inferidas a partir do *dataset* fornecido **[7]** permite ver a lógica da solução) mas também pelo facto de refletirem o pensamento humano tornando-as fáceis de perceber e produzindo interpretações boas.

Numa árvore de decisão, os nós representam atributos[[3]](#footnote-3) cada ramo representa uma regra de decisão[[4]](#footnote-4) e cada folha um rótulo/*label[[5]](#footnote-5).* Um pseudocódigo possível para uma árvore de decisão está representado na *Figura 3*. A *Figura 5* e *4* apresentam uma árvore de decisão para um dado *dataset*, respetivamente.

Figura -Pseudoçõdigo de árvores de decisão (pseudocódigo retirado de **[7]**)

**DecisionTree(dataset):**

1 Se não existem mais atributos então

2 retorna(valorMaisComum(dataset))

3 root <- bestAtribute(dataset)

4 left <- splitData(root.atribute).left

5 right <- splitData(root.atribute).right

6 node\_left <- DecisionTree(left)

7 node-right <- DecisionTree(right)

8 root <- root U left

9 root <- root U right

10 retorna(root)

Existem vários algoritmos para árvores de decisão, tal como a *Figura 1* mostra, no entanto, iremos focar-nos em 3:

* **ID3**: algoritmo baseado em pesquisa gulosa, utiliza *information gain* e *entropy* para construir a árvore, algoritmo para dados categóricos;[[6]](#footnote-6)
* **C4.5**: algoritmo muito semelhante ao ID3, mas para dados contínuos;
* **CART**: utiliza o *gini index* para calcular a impureza dos dados

Uma imagem com texto

Descrição gerada automaticamente

Uma imagem com texto, mapa

Descrição gerada automaticamente

Figura - Dataset (imagem retirada de **[2]**)

Figura -Árvore de decisão para a Figura 4 (imagem retirada de **[2]**)

1. **Algoritmos de Indução em Árvores de decisão:**

Nesta secção iremos analisar os 3 algoritmos de indução sobre árvores de decisão: ID3, CART e C4.5.

Uma árvore de decisão vai sendo construída através de uma boa divisão dos dados com base num atributo divisor/*split*, no entanto, num conjunto com n atributos torna-se muito difícil decidir qual atributo escolher, uma solução seria ir escolhendo atributos de forma aleatória, no entanto isso poderia produzir classificações erradas e como tal, diminuir a precisão da nossa árvore **[7]**. A forma como o atributo é selecionado define o tipo de algoritmo que está aser utilizado para classificar a árvore.

* 1. **CART (Classification and Regression Trees)**

Este algoritmo consiste numa técnica não paramétrica que permite a indução em problemas de *classification* e *regression* **[5]**. O algoritmo utiliza como uma métrica de avaliação/indução o **Gini index** para calcular a impureza[[7]](#footnote-7) dos dados criando pontos de decisão na árvore **[5]**, a fórmula para cálculo do *Gini index* é a seguinte:

Uma vantagem deste algoritmo encontra-se na grande capacidade de pesquisa das relações entre os dados bem como na produção de resultados sob a forma de árvores de decisão de grande simplicidade e legibilidade **[5]**. As árvores geradas pelo algoritmo são sempre **binárias**, os nós correspondentes a atributos contínuos são agrupados em dois conjuntos **[5]**.

Iremos agora apresentar uma aplicação do algoritmo a um dado problema.

* + 1. **Aplicar CART**

Para esta aplicação iremos considerar o seguinte problema: Pretende-se decidir se um cliente irá entrar num dado restaurante, para isso, realizam-se 12 observações sobre restaurantes nas proximidades para se determinar a ação do cliente bem como as características que o restaurante deverá ter para o cliente entrar nele. A *Figura 6* apresenta o conjunto de observações.

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Alt | Bar | Fri | Hun | Pat | Price | Rain | Res | Type | Est | **WillWait** |
| 1 | Yes | No | No | Yes | Some | $$$ | No | Yes | French | 0-10 | Yes |
| 2 | Yes | No | No | Yes | Full | $ | No | No | Thai | 30-60 | No |
| 3 | No | Yes | No | No | Some | $ | No | No | Burger | 0-10 | Yes |
| 4 | Yes | No | Yes | Yes | Full | $ | Yes | No | Thai | 10-30 | Yes |
| 5 | Yes | No | Yes | No | Full | $$$ | No | Yes | French | >60 | No |
| 6 | No | Yes | No | Yes | Some | $$ | Yes | Yes | Italian | 0-10 | Yes |
| 7 | No | Yes | No | No | None | $ | Yes | No | Burger | 0-10 | No |
| 8 | No | No | No | Yes | Some | $$ | Yes | Yes | Thai | 0-10 | Yes |
| 9 | No | Yes | Yes | No | Full | $ | Yes | No | Burger | >60 | No |
| 10 | Yes | Yes | Yes | Yes | Full | $$$ | No | Yes | Italian | 10-30 | No |
| 11 | No | No | No | No | None | $ | No | No | Thai | 0-10 | No |
| 12 | Yes | Yes | Yes | Yes | Full | $ | No | No | Burger | 30-60 | Yes |

Figura - Tabela deobservações para o problema

Para este problema, temos 10 atributos de estudo:

* Alt: Diz se o restaurante tem outros restaurantes nas proximidades (Yes/No);
* Bar: Diz de o restaurante tem uma zona de bar (Yes/No);
* Fri: Diz se era final de semana quando a observação foi feita (Yes/No);
* Hun: Diz se o cliente tem fome (Yes/No);
* Pat: Diz o estado de ocupação do restaurante (None/Some/Full);
* Price: Diz o preço geral de uma refeição no restaurante ($/$$/$$$);
* Rain: Diz se estava a chover quando aobservação foi feita (Yes/No);
* Res: Diz se o restaurante permite fazer reservas (Yes/No);
* Type: Diz o tipo de comida servida pelo restaurante (Thai/Burger/Italian/French);
* Est: Diz o tempo médio de espera em minutos para ser atendido (0-10/10-30/30-60/>60);

Possuímos ainda uma coluna que classifica uma entrada nas observações, Will Wait que diz se, perante a observação feita, o cliente espera para ser atendido no restaurante ou não (Yes/No).

Iremos agora aplicar o algoritmo CART para este problema. Mais uma vez, é importante referir que, como o método utilizado para induzir a árvore é o *gini index* o atributo que será utilizado para dividir os dados é aquele que produz um menor valor *gini*, ou seja, é aquele atributo que apresenta menos impurezas.

* Iteração 1:

Iremos agora calcular os valores de *gini* para todos os atributos.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Will Wait | |  |
| Alt | Yes | No | Número de ocorrências |
| Yes | 3 | 3 | 6 |
| No | 3 | 3 | 6 |
| Total | 6 | 6 |  |

Figura - Tabela de contagens para atributo Alt para tabela original da Figura 6

Gini(Alt=Yes) =

Gini(Alt=No) =

Gini(Alt) = =

|  |  |  |  |
| --- | --- | --- | --- |
|  | Will Wait | |  |
| Bar | Yes | No | Número de ocorrências |
| Yes | 3 | 3 | 6 |
| No | 3 | 3 | 6 |
| Total | 6 | 6 |  |

Figura - Tabela de contagens para atributo Bar para tabela original da Figura 6

Gini(Bar=Yes) =

Gini(Bar=No) =

Gini(Bar) = =

|  |  |  |  |
| --- | --- | --- | --- |
|  | Will Wait | |  |
| Fri | Yes | No | Número de ocorrências |
| Yes | 2 | 3 | 5 |
| No | 4 | 3 | 7 |
| Total | 6 | 6 |  |

Figura - Tabela de contagens para atributo Fri para tabela original da Figura 6

Gini(Fri=Yes) =

Gini(Fri=No) =

Gini(Fri) =

|  |  |  |  |
| --- | --- | --- | --- |
|  | Will Wait | |  |
| Hun | Yes | No | Número de ocorrências |
| Yes | 5 | 2 | 7 |
| No | 1 | 4 | 5 |
| Total | 6 | 6 |  |

Figura - Tabela de contagens para atributo Hun para tabela original da Figura 6

Gini(Hun=Yes) =

Gini(Hun=No) =

Gini(Hun) =

|  |  |  |  |
| --- | --- | --- | --- |
|  | Will Wait | |  |
| Pat | Yes | No | Número de ocorrências |
| None | 0 | 2 | 2 |
| Some | 4 | 0 | 4 |
| Full | 2 | 4 | 6 |
| Total | 6 | 6 |  |

Figura - Tabela de contagens para atributo Pat para tabela original da Figura 6

Gini(Pat=None) =

Gini(Pat=Some) =

Gini(Pat=Full) =

Gini(Pat) =

|  |  |  |  |
| --- | --- | --- | --- |
|  | Will Wait | |  |
| Price | Yes | No | Número de ocorrências |
| $ | 3 | 4 | 7 |
| $$ | 2 | 0 | 2 |
| $$$ | 1 | 2 | 3 |
| Total | 6 | 6 |  |

Figura - Tabela de contagens para atributo Price para tabela original da Figura 6

Gini(Price=$) =

Gini(Price=$$) =

Gini(Price=$$$) =

Gini(Price) =

|  |  |  |  |
| --- | --- | --- | --- |
|  | Will Wait | |  |
| Rain | Yes | No | Número de ocorrências |
| Yes | 2 | 2 | 4 |
| No | 4 | 4 | 8 |
| Total | 6 | 6 |  |

Figura - Tabela de contagens para atributo Rain para tabela original da Figura 6

Gini(Rain=Yes) =

Gini(Rain=No) =

Gini(Rain) =

|  |  |  |  |
| --- | --- | --- | --- |
|  | Will Wait | |  |
| Res | Yes | No | Número de ocorrências |
| Yes | 3 | 2 | 5 |
| No | 3 | 4 | 7 |
| Total | 6 | 6 |  |

Figura - Tabela de contagens para atributo Res para tabela original da Figura 6

Gini(Res=Yes) =

Gini(Res=No) =

Gini(Res) =

|  |  |  |  |
| --- | --- | --- | --- |
|  | Will Wait | |  |
| Type | Yes | No | Número de ocorrências |
| Burger | 2 | 2 | 4 |
| French | 1 | 1 | 2 |
| Italian | 1 | 1 | 2 |
| Thai | 2 | 2 | 4 |
| Total | 6 | 6 |  |

Figura - Tabela de contagens para atributo Type para tabela original da Figura 6

Gini(Type=Burger) =

Gini(Type=French) =

Gini(Type=Italian) =

Gini(Type=Thai)=

Gini(Typer) =

|  |  |  |  |
| --- | --- | --- | --- |
|  | Will Wait | |  |
| Est | Yes | No | Número de ocorrências |
| 0-10 | 4 | 2 | 6 |
| 10-30 | 1 | 1 | 2 |
| 30-60 | 1 | 1 | 2 |
| >60 | 0 | 2 | 2 |
| Total | 6 | 6 |  |

Figura - Tabela de contagens para atributo Pat para tabela original da Figura 6

Gini(Est=0-10) =

Gini(Est=10-30) =

Gini(Est=30-60) =

Gini(Est=>60) =

Gini(Est) =

|  |  |
| --- | --- |
| Atributo | Gini index |
| Alt |  |
| Bar |  |
| Fri |  |
| Hun |  |
| Pat |  |
| Price | 0.396 |
| Rain |  |
| Res |  |
| Type |  |
| Est |  |

Figura -Tabela de gini index para cada atributo para tabela original da Figura 6

O atributo que será escolhido como novo nó na árvore será aquele que minimize o valor de impurezas, o *gini index*, ou seja, o atributo Pat. Dessa forma, podemos agora dividir o conjunto de dados original por cada valor do atributo Pat ficando na árvore um nó de atributo Pat e cada ramo um dos seus valores (None,Some,Full), para cada divisão, uma vez que o atributo Pat já foi avaliado, para evitar ciclos, pode ser retirado da lista de dados de cada ramo.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Alt | Bar | Fri | Hun | Price | Rain | Res | Type | Est | **WillWait** |
| Yes | No | No | Yes | $ | No | No | Thai | 30-60 | No |
| Yes | No | Yes | Yes | $ | Yes | No | Thai | 10-30 | Yes |
| Yes | No | Yes | No | $$$ | No | Yes | French | >60 | No |
| No | Yes | Yes | No | $ | Yes | No | Burger | >60 | No |
| Yes | Yes | Yes | Yes | $$$ | No | Yes | Italian | 10-30 | No |

Full

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Alt | Bar | Fri | Hun | Price | Rain | Res | Type | Est | **WillWait** |
| Yes | No | No | Yes | $$$ | No | Yes | French | 0-10 | Yes |
| No | Yes | No | No | $ | No | No | Burger | 0-10 | Yes |
| No | Yes | No | Yes | $$ | Yes | Yes | Italian | 0-10 | Yes |
| No | No | No | Yes | $$ | Yes | Yes | Thai | 0-10 | Yes |

Some

Pat

None

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Alt | Bar | Fri | Hun | Price | Rain | Res | Type | Est | **WillWait** |
| No | Yes | No | No | $ | Yes | No | Burger | 0-10 | No |
| No | No | No | No | $ | No | No | Thai | 0-10 | No |

Figura – Novo nó gerado na árvore para os dados da Figura 6

Uma observação que pode ser feita à figura 18 é que tanto os dados do ramo None como do ramo Some são dados puros[[8]](#footnote-8) pelo que, podemos então concluir imediatamente a classificação dos dados, caso estejamos a percorrer esses ramos, ficando a *Figura 18* da seguinte forma:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Alt | Bar | Fri | Hun | Price | Rain | Res | Type | Est | **WillWait** |
| Yes | No | No | Yes | $ | No | No | Thai | 30-60 | No |
| Yes | No | Yes | Yes | $ | Yes | No | Thai | 10-30 | Yes |
| Yes | No | Yes | No | $$$ | No | Yes | French | >60 | No |
| No | Yes | Yes | No | $ | Yes | No | Burger | >60 | No |
| Yes | Yes | Yes | Yes | $$$ | No | Yes | Italian | 10-30 | No |

Full

Some: WillWait=**Yes**

Pat

None: WillWait=**No**

Figura – Poda de nó da Figura 18

Este corte não pode ser aplicado para o ramo Full, uma vez que, os dados não são puros, o que significa que o algoritmo irá continuar até que todos os dados sejam agrupados ou se esgotem os atributos para partir os dados, nessa altura, a árvore está concluída.

É importante referir que o algoritmo **CART não faz este corte imediatamente**, para o CART não é realizada pré-poda, ou seja, a árvore é **expandida exaustivamente**, iremos ver mais para a frente algoritmos que realizam podam a árvore da mesma forma que está demonstrada na *Figura 19*.

Para este exemplo, consideramos um conjunto de dados onde os atributos são categóricos, no entanto, este algoritmo suporta também dados contínuos, para esses casos, o algoritmo irá primeiro agrupar os dados em conjuntos de valores **[5]**.

* 1. **ID3 (Iterative Dichotomiser)**

Este algoritmo é um algoritmo recursivo baseado em busca *greedy[[9]](#footnote-9)*. A métrica utilizada por este algoritmo para induzir a árvore é a **entropia[[10]](#footnote-10)**ou ***information gain[[11]](#footnote-11)***. As fórmulas utilizadas para calcular a entropia e o *information gain* são as seguintes **[8]**:

onde S é o conjunto de dados a ser estudado e .

onde

Este algoritmo irá em cada iteração, para o conjunto de dados a analisar encontrar o atributo que melhor divide os dados e para cada valor possível desse atributo, dividir os dados. A escolha do atributo pode ser feita através do método de entropia, onde iremos escolher o atributo que minimiza a entropia dos dados ou através do método de *information gain* onde, o atributo que melhor divide os dados é aquele que maximiza o *information gain*.

Este algoritmo apresenta algumas limitações. Uma vez que, este algoritmo é ***greedy*** produz uma **solução não ótima** uma vez que procura sempre o atributo que melhor divide os dados para um dado nível e não aquele que divide melhor para todos os níveis **[2-8]**. Este método pode sofrer de ***overfitting****[[12]](#footnote-12)*, como tal, de forma a evitar esse possível erro, este algoritmo tem uma tendência de criar árvores relativamente pequenas (mas não as mais pequenas), quando comparado a outros algoritmos, uma vez que, quanto mais pequena a árvore, menor a probabilidade de existir *overfitting* **[8]**.

A versão original deste algoritmo proposta por Ross Quinlan **[8]** apenas trabalha com valores **categóricos** e **não é capaz de lidar com tabelas de dados com valores em falta**, no entanto, ao longo dos anos, houve mais adaptações produzindo algoritmos que não sofrem das mesmas limitações, desses novos algoritmos surge o C4.5 capaz de lidar com dados categóricos e contínuos bem como dados com valores em falta **[3-8]**.

A *Figura 20* apresenta o pseudocódigo deste algoritmo.

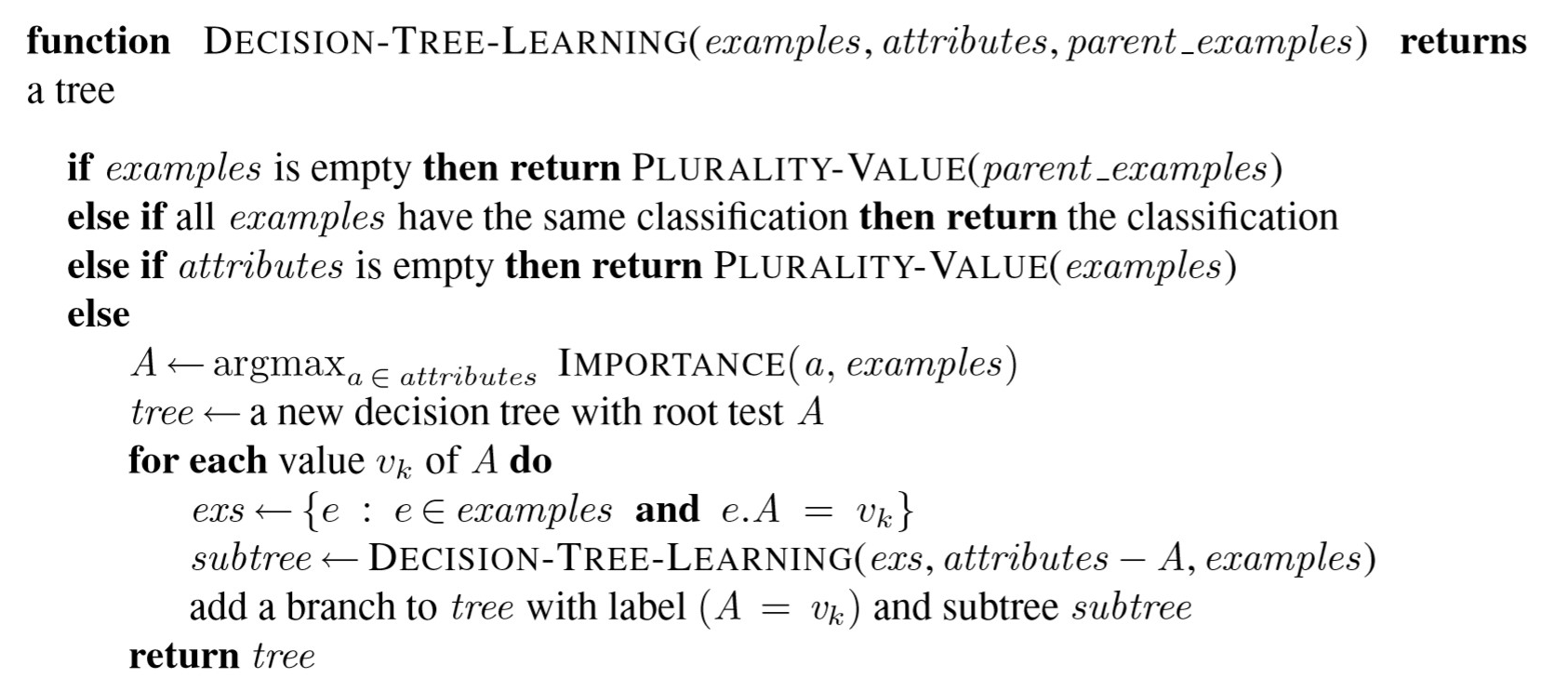


Figura -Pseudocódigo de ID3 (imagem retirada de **[2]**)

Para a implementação do ID3 para o trabalho prático proposto iremos considerar uma versão modificada do ID3 capaz de lidar com valores contínuos, para isso, o conjunto de dados deve, para atributos contínuos ser ordenado por ordem crescente e depois ser aplicado uma média entre valores consecutivos, seguidamente deverá ser feito, para cada um desses novos valores, um split binário, onde iremos calcular a *information gain* para cada um desses valores contínuos, aquele que produzir um melhor valor (máximo) será o i*nformation gain* do atributo em si.

* + 1. **Aplicar ID3**

Para a aplicação deste algoritmo iremos considerar o conjunto de dados da *Figura 6* iremos também, para a interação 1 considerar as *Figuras 7-16* como as tabelas de contagem para cada atributo.

Entropia(Dados)

InfoGain(Alt)

InfoGain(Bar) =

InfoGain(Fri)

A *Figura 21* apresenta todos os *information* *gain* para os atributos.

|  |  |
| --- | --- |
| Atributo | InfoGain |
| Alt |  |
| Bar |  |
| Fri |  |
| Hun |  |
| Pat |  |
| Price |  |
| Rain |  |
| Res |  |
| Type |  |
| Est |  |

Figura -Tabela de information gain para cada atributo para tabela original da Figura 6

Uma vez que, utilizando o método do *information gain* o atributo que melhor divide o conjunto de dados é aquele que maximiza o InfoGain, sendo assim, iremos escolher como split o atributo Pat, um novo nó será criado e os dados divididos em cada ramo (onde cada ramo representa um certo valor possível do atributo) a árvore fica então com um formato parecido ao da *Figura 18* e numa próxima iteração (ou seja, aplicar um ID3 a cada um dos ramos) irá ficar com um formato igual ao da *Figura 19* pelo menos para o nível dos valores do atributo Pat, não estamos a considerar para a representação outras subárvores.

* 1. **C4.5**

Este algoritmo foi criado de forma a corrigir algumas limitações do ID3, nomeadamente a falta de aplicabilidade do ID3 a atributos contínuos (criando *thresholds[[13]](#footnote-13)*) **[6]**. Como tal, o algoritmo acaba por funcionar da mesma forma que o anterior, utilizando como métodos de seleção de atributos divisores, os métodos da entropia e *information gain*, através de melhorias em relação ao ID3 este algoritmo acabou por se tornar uma dos mais populares **[6]**.

Este algoritmo possuí uma capacidade de podar a árvore, ou seja, após criar a árvore, faz uma busca das folhas para a raiz e, para todos os ramos que não apresentem um ganho significativo, estes são transformados em folhas **[4-6]**.

Ainda é capaz também de trabalhar com um conjunto de dados onde algumas entradas não possuem valor, para isso, ignora os casos com valores em falta para cálculo de entropias e InfoGain**[4-6]**.

Este algoritmo é um melhoramento em relação ao ID3, após a sua criação outros algoritmos como o C5.0 e See5 foram criados como melhoramentos sobre o C4.5 **[6]**.

1. **Implementação de Árvores de Decisão**

Para esta implementação de resolução do problema utilizamos como linguagem o Java, um dos fatores principais que levou à escolha desta linguagem foi a experiência anterior com a linguagem bem como o fornecimento de uma biblioteca considerável de estruturas de dados que facilitaram a resolução. É importante referir que, numa dada altura do problema, devido a alguns erros que tínhamos encontrado, decidimos fazer uma implementação desta resolução em Python, sendo uma tentativa bem sucedida e mostrando ser consideravelmente mais fácil que em Java, uma das principais razões deve-se ao facto do Python trabalhar muito bem com listas e de não exigir que uma variável mantenha sempre o mesmo tipo de dados facilitando a representação da tabela de dados e possibilitando a existência de dados contínuos. No entanto, decidimos continuar a resolver o problema em Java, o que nos forçou a implementar estruturas de dados para retorno de vários valores e mesmo de manipulação da tabela de dados.

Apresentamos a seguir algumas das estruturas de dados utilizadas na resolução do problema.

* 1. **Estruturas de Dados Escolhidas**

Para a implementação dos programas utilizamos estruturas de dados como *ArrayLists* (para representar o conjunto de atributos e o conjunto de dados), *HashMaps* (para mapear as entradas na tabela para os seus valores em cada atributo e guardar a árvore de decisão) e *LinkedHashSets* (para guardar o conjunto ordenado de atributos).

A estrutura *ArrayList* é uma estrutura que foi muito utilizada neste programa uma vez que, apresenta complexidades temporais de para remoções/inserções e pesquisa de elementos, tornando a deslocação pelos atributos e pela tabela de observações muito mais eficiente.

A estrutura *HashMap* foi utilizada para guardar uma observação, onde consideramos como chaves os atributos e valores, os valores da observação para cada atributo, isto foi utilizado para não ser necessário guardar a matriz de valores o que seria muito “caro” quando tivéssemos que dividir os dados e remover as colunas de um dado atributo, sendo assim, quando for necessário remover uma coluna, para cada observação apenas temos que remover a chave de uma observação com o atributo que desejamos remover .

A estrutura *LinkedHashSet* acaba por ser uma *LinkedList* sem repetição, operações de remoção e verificar a existência de um valor têm um custo de , uma melhoria seria, em vez de utilizar uma *LinkedHashSet* utilizar um *ArrayList* com verificação de elementos repetidos.

* 1. **Estrutura do Código**

Para a implementação deste trabalho, iremos receber o conjunto de observações em formato CSV, o programa implementado apresenta 5 ficheiros *.java*, *Atribute.java* (representa um atributo), *DataEntry.java* (representa uma observação, uma linha na tabela de dados), *DecisionTree.java* (representa a árvore de decisão, contém também a implementação do ID3), *Main.java* (programa que arranca a resolução) e *TreeBuilder.java* (apresenta os métodos para organização dos dados antes de construir a árvore, desde ler o ficheiro CSV até criar as estruturas para atributos e observações), ainda apresentamos outros ficheiro *TreeFrame.java* numa 2º versão do programa, o seu objetivo é apenas converter a árvore num formato GUI tornando a observação da árvore mais fácil, os resultados que serão expostos, serão apresentados neste formato.

* + 1. **Uma Classe Atribute**

Esta classe tem como objetivo representar um atributo dos atributos estudados no problema, sendo assim, apenas guarda o nome do atributo, e um *ArrayList* que contém os valores possíveis de um dado atributo, possuí ainda métodos *getters*  e um método para verificar se o atributo é contínuo (.

class Atribute{ //classe para representar atributo

String atribute; //nome do atributo

ArrayList<String> values; //valores possíveis (n repetidos) qu o atributo pode tomar

Atribute(String atribute,ArrayList<Sring>values); //construtor de atributo

boolean isContinuous(); //retorna true se os valores do atributo são contínuos

String getAtribute(); //retorna o nome do atributo

ArrayList<String> getAtributeVals(); //retorna o conjunto de valores possíveis do atributo

}

Figura -Classe Atribute

* + 1. **Uma Classe DataEntry**

Esta classe tem como objetivo guardar uma linha da tabela de observações, guarda a classificação final da linha em questão e o mapa de valores da linha para cada um do atributos, ou seja se a linha a ser convertida/armazenada for a da *Figura 23* os valores ficam guardados num *HashMap* como o da *Figura 24*  e a sua classificação (WillWait) fica guardada numa *string=yes*.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Alt | Bar | Fri | Hun | Pat | Price | Rain | Res | Type | Est | **WillWait** |
| Yes | No | No | Yes | Some | $$$ | No | Yes | French | 0-10 | Yes |

Figura - Linha a guardar

|  |  |
| --- | --- |
| **Classification=Yes** | |
| **Mapa de valores** | |
| **Keys** | **Values** |
| Alt | Yes |
| Bar | No |
| Fri | Yes |
| Hun | No |
| Pat | Some |
| Price | $$$ |
| Rain | No |
| Res | Yes |
| Type | French |
| Est | 0-10 |

Figura -Representação da DataEntry da Figura 23

class DataEntry{ //classe para armazenar uma linha/observação da tabela de observações

String final\_classification; //string com a classificação da linha

HashMap<String,String> entry\_values; //mapa dos valores da observação para cada atributo

DataEntry(String classification,String[]atributes,String[]vals); //construtor da DataEntry, recebe a classificação final da observação, o array de chaves (atributos) e o array de valores

DataEntry(String[]atributes,String[]vals); //construtor da DataEntry, recebe o array de chaves (atributos) e o array de valores, este construtor é utilizado na fase de teste de uma árvore, quando é fornecida uma linha de uma possível observação mas onde não sabemos a classificação (queremos descobrir)

String getAtributeVal(String atribute); //dado um atributo como chave, retorna o seu valor no mapa de valores

String getClassification(); //retorna a classificação da entrada

}

Figura - Classe DataEntry

* + 1. **Uma Classe DecisionTree**

Esta classe é responsável por criar a árvore de decisão (através de uma aplicação de ID3 modificado para valores contínuos) e por classificar uma dada observação para a qual não sabemos a sua classificação (só feito depois da árvore ser criada, método de teste de árvore).

class DataDivision{ //classe para armazenar a divisão de dados, dado um split

ArrayList<DataEntry> data\_above\_split;

ArrayList<DataEntry> getBelow; DataDivision(ArrayList<DataEntry>data\_above\_split,ArrayList<DataEntry>data\_below\_split); //construtor que guarda os dois novos subconjuntos

ArrayList<DataEntry> getAbove(); //retorna os dados acima o valor do split

ArrayList<DataEntry> getBelow(); //retorna os dados abaixo o valor do split

}

class BestSplit{ //classe para armazenar a divisão de dados, dado um split

String split\_value; //string com a valor do split (caso contínuo)

Atribute split; //guarda o atributo divisor

BestSplit(Atribute split,String split\_value); //construtor

Atribute getBestSplit(); //retorna o atributo divisor

String getBestValue(); //retorna o melhor valor do atributo divisor (se este for contínuo)

}

class **DecisionTree**{ //classe representar/construir a árvore de decisão

ArrayList<DataEntry> parent\_data\_set; //dados do nó pai (se este existir)

ArrayList<DataEntry> data\_set; //dados do nó a ser construído

LinkedHashSet<Atribute> atributes; //atributos para este nó por analisar

HashMap<String,DecisionTree> descendents; //guarda as árvores descendentes da atual (filhos)

DecisionTrees(ArrayList<DataEntry>data,ArrayList<DataEntry>parent\_data,LinkedHashSet<Atribute>atributes); //construtor, recebe o conjunto de atributos por analisar, guarda o conjunto de dados a tual (data) e o conjunto de dados do nó pai (caso o atual não possa ser analisado)

BestSplit getSplit(); //retorna o split deste nó

bopolean isPure(ArrayList<DataEntry>data); //retorna true se todos os dados em data tiverem a mesma classificação

String MostCommonTarget(ArrayList<DataEntry>data); //retorna a classe mais comum em data

double dataEntropy(ArrayList<DataEntry>data); //calcula a entropia de data

double overallEntropy(ArrayList<DataEntry>above, ArrayList<DataEntry>below); //calcula a entropia de um split para depois aplicar o InfoGain

DataDivision splitData(Atribute atribute,String value); //divide os dados para o atributo atribute de valor value

double atributeInfoGainCategorical(Atribute atribute,ArrayList<DataEntry>data); //calcula o InfoGain para atributos categóricos

BestSplit bestAtributeToSplit(); //retorna o melhor atributo divisor

String classify(DataEntry entry); //classifica uma dada entrada teste

void buildTree(); //algoritmo ID3

void printTree(); //imprime a árvore

}

Figura - Classe DecisionTree e classes auxiliares BestSplit e DataDivision

1. **Resultados**

Iremos agora apresentar as árvores de decisão para cada problema (*restaurant.csv*, *iris.csv* e *weather.csv*), representados a <..> estão os atributos, os valores e são as atribuições para cada atributo, a [..] estão as classificações do caminho e a (..) estão os *couters* para cada valor.

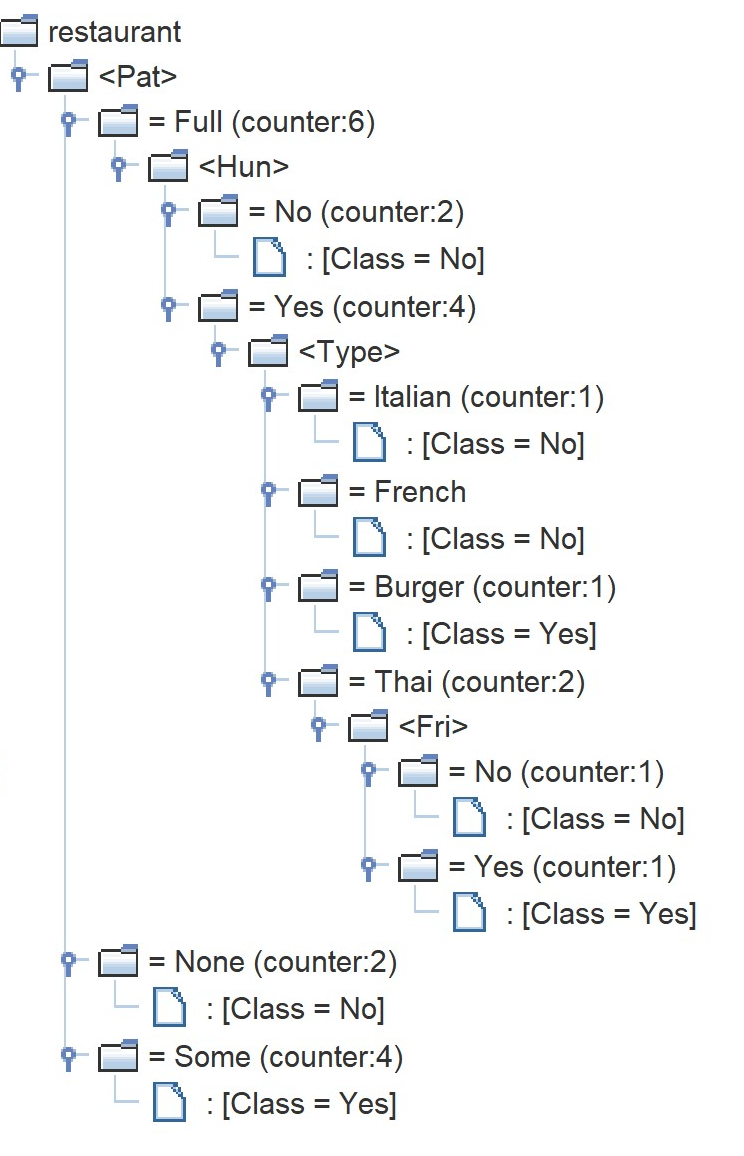


Figura - Árvore de decisão para restaurant.csv

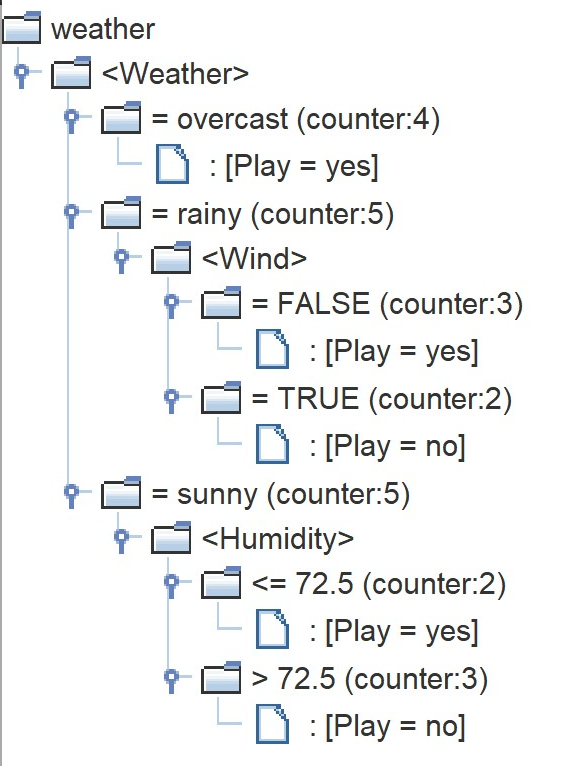


Figura - Árvore de decisão para weather.csv

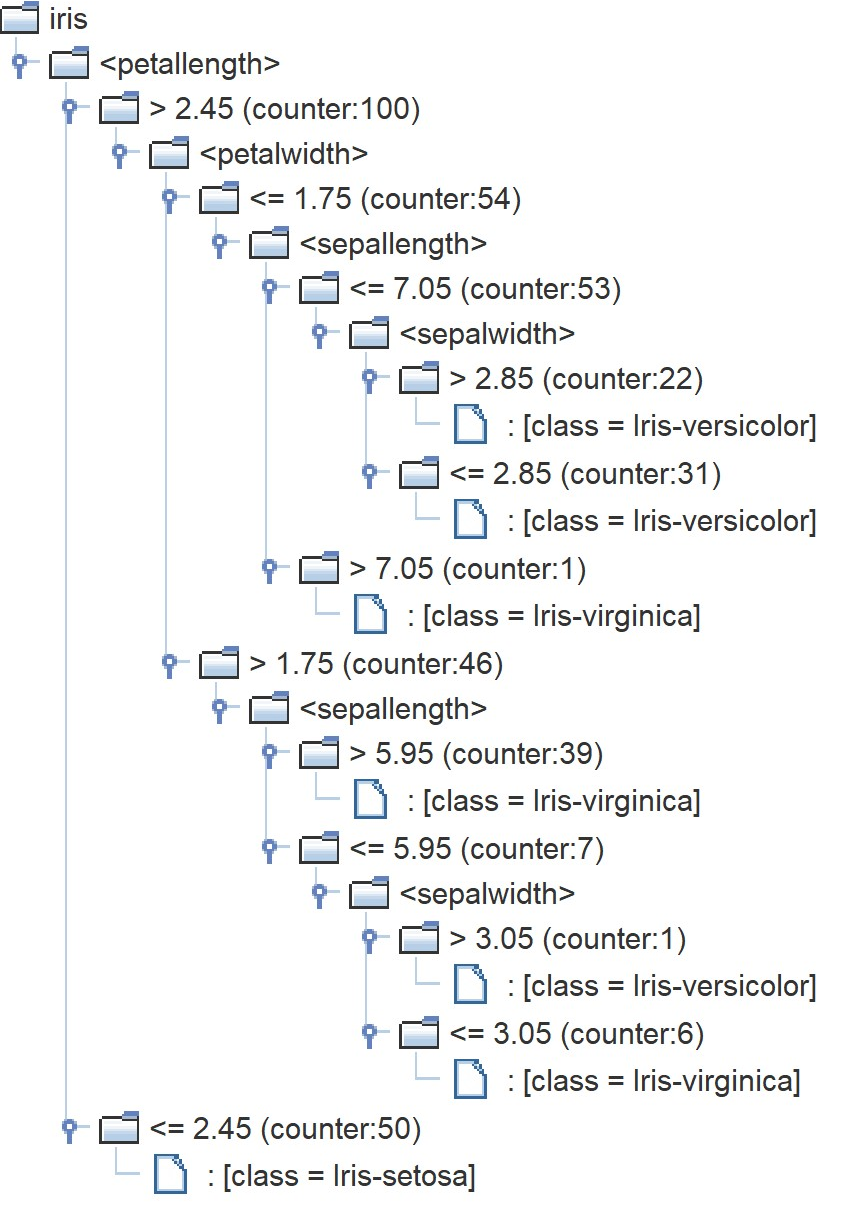


Figura - Árvore de decisão para iris.csv

**Nota Adicional:** Este relatório, bem como o programa utilizado para obtenção dos dados, pode ser acedido no repositório da equipa, <https://github.com/thejoblessducks/the-15-that-was-the-puzzle.git>, ou em <https://github.com/eamorgado/ProjetoIA-15Puzzle.git>

**7. Referências:**

**[1]** Machine Learning Mastery n.d., *Supervised and Unsupervised Machine Learning Algorithms*, Jason Brownlee, consultado pela última vez a 27 de abril de 2019, [<https://machinelearningmastery.com/supervised-and-unsupervised-machine-learning-algorithms/>](https://machinelearningmastery.com/supervised-and-unsupervised-machine-learning-algorithms/)

**[2]** S.Russell, P. Norving, “Artificial Intelligence A Modern Approach”, 3rd Ed. , Chapter 18. Learning from Examples 698-700, 2009

**[3]** QUINLAN, J. R. (1986). Induction of decision trees. Machine Learning, 1(1):81-106.

**[4]** QUINLAN, J. R. (1993). C4.5: programs for machine learning. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

**[5]** BREIMAN, L., FRIEDMAN, J. H., OLSHEN, R. A., & STONE, C. J. (1984). Classification and Regression Trees. Wadsworth.

**[6]** Wikipédia 16 de fevereiro de 2019, *C4.5 algorithm*, visitado pela última vez a 20 de Abril de 2019, [<https://en.wikipedia.org/wiki/C4.5\_algorithm>](https://en.wikipedia.org/wiki/C4.5_algorithm)

**[7]** Dataaspirant 30 de janeiro de 2017, *How Decision Tree Algorithm Works*, Rahul Saxena, visitado pela última vez a 27 de abril de 2019, [<http://dataaspirant.com/2017/01/30/how-decision-tree-algorithm-works/>](http://dataaspirant.com/2017/01/30/how-decision-tree-algorithm-works/)

**[8]** Wikipedia 25 de abril de 2019, *ID3 algorithm*, visitado pela última vez a 27 de abril de 2019, [<https://en.wikipedia.org/wiki/ID3\_algorithm>](https://en.wikipedia.org/wiki/ID3_algorithm)

1. Variável não numérica, por exemplo “yes”/”no”. [↑](#footnote-ref-1)
2. Variável numérica, por exemplo idade, pode tomar um qualquer valor real. [↑](#footnote-ref-2)
3. Um atributo é o tipo de observação feita, por exemplo, para a escolha de um transporte público podemos fazer observações baseados em critérios como a existência de trânsito, as condições meteorológicas, o tempo de viagem, esses critérios são atributos. [↑](#footnote-ref-3)
4. Uma regra de decisão é um valor de um critério, por exemplo, para o problema do transporte público, se considerar como um critério a existência de trânsito e se, perante todas as observações, os seus dados possíveis são “sim” e “não” então para o nó/atributo <existência de trânsito> iremos ter dois ramos/regras na nossa árvore: =sim e =não. [↑](#footnote-ref-4)
5. Um rótulo é o *outcome* do caminho que está a ser explorado na árvore, para o problema do transporte público, podemos considerar como rótulos/*outcomes* possíveis: metro,autocarro,elétrico,comboio e avião, por exemplo. [↑](#footnote-ref-5)
6. Na implementação que iremos fazer, o ID3 irá poder classificar dados categóricos e dados contínuos. [↑](#footnote-ref-6)
7. Mede quantas vezes um elemento escolhido aleatoriamente seria classificado incorretamente, logo, **atributos com menor impureza/gini index seriam escolhidos [7]** [↑](#footnote-ref-7)
8. Um conjunto de dados é puro se e só se, todos os elementos nesse conjunto tiverem a mesma classificação. [↑](#footnote-ref-8)
9. Procura num conjunto de atributos, aquele que melhor divide os dados **[3]**. [↑](#footnote-ref-9)
10. A entropia mede a incerteza dos dados, mede a quantidade de informação sobre o problema pode ser obtida através da medição de um dado atributo **[8]**. O valor de entropia encontra-se em [0,1], uma entropia de valor 1 é muito instável uma vez que é incapaz de classificar completamente os dados enquanto que, uma entropia de 0 classifica perfeitamente os dados, sendo assim, atributos que **minimizem** a **entropia** são os atributos que **melhor dividem os dados [8]**. [↑](#footnote-ref-10)
11. O *information gain* mede a **diferença de entropia**dos dados serem divididos por um atributo, ou seja, o *information gain* mede a  **redução de incerteza após os dados serem divididos por um dado atributo [8]** Sendo assim, para um conjunto de dados, iremos escolher como atributo divisor o atributo que **maximize o *information gain***, uma vez que, a partir dessa divisão ficaremos a saber mais sobre os dados e sobre o problema. [↑](#footnote-ref-11)
12. A classificação dos dados corresponde muito proximamente à sua classificação real pelo que pode existir um erro quando novos dados são classificados ou mesmo ser impossível classificar novos valores. [↑](#footnote-ref-12)
13. Encontra um valor com bom InfoGain nos valores do atributo e divide todos os dados em dois conjuntos, aqueles que são que to *threshold* e aqueles que são > que o *threshold* **[6]**, ou seja, a divisão é uma divisão binária. [↑](#footnote-ref-13)