U.PORTO

FACULDADE DE CIÊNCIAS
UNIVERSIDADE DO PORTO

Árvores de Decisão

Relatório 3

Rui Santos, Tiago Teixeira e Carla Henriques

CC2006: Inteligência Artificial

Profª. Inês Dutra

Prof.º Francesco Renna

Prof.º Nuno Guimarães

Maio de 2023

Índice

Introdução

A aprendizagem computacional é um subcampo da área de inteligência artificial, na qual seu foco de estudo é permitir que, a partir de conjuntos de dados, previsões e decisões sejam feitas computacionalmente usando técnicas estatísticas e algoritmos computacionais. Assim é possível criar modelos que identificam automaticamente padrões nos dados, que por sua vez ajudam a fazer generalizações e/ou previsões.

Dentro deste vasto campo, um dos modos usados para classificação e regressão de dados são as árvores de decisão. As árvores de decisão servem como uma enorme ajuda para fazermos escolhas mais bem estruturadas e organizadas no complexo processo de decision-making. Elas são representações que permitem classificação de dados – i.e., generalizações na qual classes são atribuídas a instâncias – consoante a certas propriedades deste (Luger, 2005, p. 408), sendo uma ferramenta fundamental e versátil utilizada como modelo para classificar e organizar dados.

Uma árvore de decisão é um modelo hierárquico que vai dividindo os dados tendo como base algumas de regras ou métodos de decisão (critério de *split*). Consiste num conjunto de nós, folhas e arestas, onde cada nó representa um atributo ou uma escolha, as folhas representam a decisão final e as arestas as possíveis alternativas.

Ela tem como uma das suas principais vantagens a sua fácil e simples interpretação. Seguindo um caminho percorrido na árvore conseguimos rapidamente compreender como foram feitas as decisões e previsões utilizando as variáveis de entrada presentes. Em áreas como finanças, sistemas legais ou na saúde, por exemplo, esta transparência é fundamental.

Algoritmos para árvores de decisão

Nesta secção vamos explorar alguns dos mais populares algoritmos utilizados na indução de árvores de decisão. A escolha do algoritmo correto a implementar na árvore é muito importante, uma vez que é com base nele que será determinada a forma como a estrutura da árvore é construída, como dividimos a informação em cada nó ou a forma de otimizar a performance das próprias árvores, de modo a ter os resultados mais adequados para os conjuntos de dados em questão.

Algoritmos supervisionados, como o ID3, C4.5 e CART, usam atributos dos dados para distinguir padrões, e há um conjunto de rótulos (*labels*) ou classes na qual se deseja classificar o conjunto de dados, que variam consoante a características ou atributos dos dados. É possível então medir o desempenho do modelo de classificação, no quanto de sua taxa de acerto ou erro na classificação, medindo a exatidão, precisão e eficácia do modelo.

ID3

O ID3 (Iterative Dichotomiser 3) é um algoritmo de classificação estatística supervisionado, que constrói árvores de decisão a partir de um conjunto de dados. Este algoritmo serviu como base para outros algoritmos como o C4.5.

A cada nível ele vai selecionando os atributos melhor classificados, com base na diferença de entropia, para assim subdividir os dados, tendo como objetivo o máximo ganho de informação. Deste modo constrói as árvores de forma recursiva separando os dados em subconjuntos usando os atributos escolhidos.

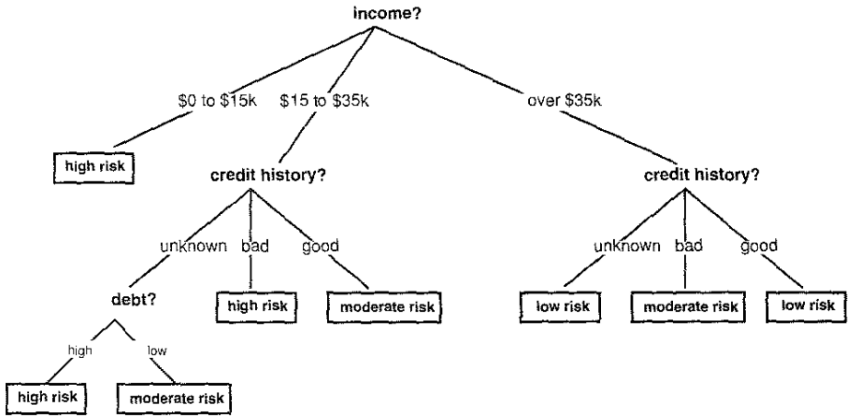


Figura 1 – Exemplo de ID3 de classificação de risco em um conjunto de dados de pedidos de empréstimo (Lugen, 2005, p. 410).

A entropia é a medida de grau de incerteza de um conjunto de dados. Isto se refere a quantificação da distribuição estatística deste conjunto de dados em classes, tendo em conta a falta de homogeneidade. Quanto maior a entropia, maior a incerteza.

A maximização do ganho de informação ocorre pela escolha de atributos que minimizam esta incerteza do conjunto de dados, ou seja, que reduzem a diferença de entropia (entre o nó pai e uma espécie de média ponderada dos nós filhos). Portanto, quanto maior a diferença de entropia, menor o ganho de informação; e quanto menor a diferença entropia, maior o ganho de informação, indicando um atributo mais útil para a classificação.

Deste modo, como o critério de *split* visa maximizar o ganho da informação, são escolhidos atributos que tem informações mais úteis para a classificação, logo os atributos que proporcionam menor diferença de entropia. O critério consequentemente tenta minimizar o valor da diferença entropia a cada nível, para assim homogeneizar o melhor possível a distribuição do conjunto de dados em classes, o que resulta em uma melhor performance na classificação dos dados.

Este processo guia a construção da árvore em um processo de ordem *top-down* (Luger, 2005, p. 411), pois é repetido recursivamente a cada subconjunto de dados ou nó filho, até chegar a um critério de paragem. Esta construção da árvore, a cada nível, o ID3 opera sob uma lógica de busca com *greedy*, já que ele não considera o *outcome* global de toda a extensão da árvore e apenas examina o que naquele momento do *split* proporciona maior ganho de informação (Luger, 2005, p. 418). Como consequência, o algoritmo tem a tendência a criar árvores muito complexas levando a que a árvore se adapte excessivamente aos dados, o chamado *overfitting*.

Contudo, o ID3 apresenta algumas limitações. Além do já mencionado não combate ao *overfitting*, este algoritmo não lida com dados incompletos. Uma outra

caraterística do ID3 original é que ele lida apenas com dados discretos e não com os contínuos.

ID3 - Modificação para o trabalho

Para a realização deste trabalho utilizamos um algoritmo muito próximo do ID3. Uma limitação do ID3 está relacionado com a forma com que o algoritmo lida com os atributos, lidando apenas com atributos discretos e categóricos. Para o trabalho foi pedido eu o algoritmo desse suporte a dados contínuos e numéricos, o que consiste em uma das características do C4.5, sem ser propriamente dito o C4.5 completo.

C4.5

Este algoritmo de árvore de decisão utilizado em machine learning e consiste em uma extensão do ID3. Suas melhorias em relação ao ID3 incluem que ele aceita dados tanto discretos quanto contínuos, e consegue lidar com dados incompletos.

Escolhe os melhores atributos com base na proporção do ganho de informação, diferentemente do ID3, que usa apenas o ganho de informação. Ou seja, no C4.5 o ganho de informação é transformado em uma proporção na qual o ganho de informação é normalizado com a informação intrínseca de um atributo, o que mede o potencial de informação gerada por aquele atributo. Assim, o ganho de informação é ajustado, evitando enviesamento (bias) para alguns atributos.

Além disso, ele tem uma forma a evitar que árvore se torna demasiado complexa, ou seja, trata-se de uma estratégia que combate o *overfitting*. Isto é implementado ao adicionar ao algoritmo o *prunning* de nós cuja informação é considerada com baixa utilidade para a classificação dos dados.

Outros algoritmos de árvores de decisão

O CART (Classification and Regression Trees) é um algoritmo versátil usado para criar árvores de decisão em machine learning. Usa várias métricas como: erro médio quadrático, ganho de informação e impureza de Gini, todas estas utilizadas de forma a dividir a informação em cada nó. Ele aplica estas métricas recursivamente onde é capaz de criar uma árvore binária que consiga fazer as classificações necessárias e também as tarefas de regressão. Este modelo é muito popular, pois fornece um resultado claro e fácil de interpretar nas tomadas de decisão.

Existe também o Random Forest que junta várias árvores de decisão para fazer previsões e usa machine learning. Este algoritmo tem como melhores características a sua capacidade de generalização e a sua excelente precisão. Funciona de forma diferente dos algoritmos anteriores, neste caso cada árvore é treinada com uma amostra aleatória usando os dados e escolha aleatória de atributos onde depois as previsões da cada árvore são reunidas por média com o objetivo de obter a previsão final.

O algoritmo Gradiente Boosting combina em sequência diferentes árvores de decisão. As árvores posteriores são treinadas para corrigir os problemas do modelo anterior dando prioridade às que foram previamente classificadas incorretamente. Utiliza machine learning e procura ir melhorando sempre a precisão do modelo aprendendo com

os erros anteriormente cometidos. Este algoritmo consegue cria modelos de elevada precisão em várias tarefas como a classificação e regressão.

Com base neste último algoritmo existem outros dois: XGBoost e AdaBoost. De forma semelhante ao funcionamento do Gradiente Boosting, nestes algoritmos cada árvore está treinada para corrigir os erros do modelo anterior. No caso do XGBoost, é muito parecido ao referido, mas junta ainda técnicas adicionais com o objetivo de melhorar mais a precisão do modelo, usando a otimização de gradiente, por exemplo. Já no AdaBoost difere na parte em que este dá maior importância às instâncias classificadas incorretamente. Ambos algoritmo tem a capacidade de lidar com dados complexos.

Implementação

Linguagem Utilizada

A escolha da linguagem Java para implementar as árvores de decisão deve-se a várias razões. Em primeiro lugar, Java é uma linguagem de programação amplamente utilizada e com uma grande comunidade de desenvolvedores, o que torna mais fácil encontrar recursos e soluções para problemas específicos. Além disso, Java é uma linguagem orientada a objetos, o que significa que a implementação do problema pode ser modelada usando objetos e classes, tornando a implementação mais fácil de entender e manter.

Estrutura de dados usadas

Detalhamento das principais estruturas utilizadas no código que implementamos neste projeto:

Classe Coluna

Utilizamos a classe Coluna para representar uma coluna da tabela.Para realizar este objetivo temos os seguintes atributos e estruturas de dados pertencentes a classe coluna:

- String nome:
  Representa o nome da coluna.
- Map<String,Map<String,Set<Integer>>>m:
  Representa um mapa, que dado uma variável e dado uma classe “retorna” a lista de índices dos exemplos que pertence a essa variável e classe.

Classe Main

Na classe Main utilizamos principalmente as estruturas de dados para receber a informação do csv e criar estruturas auxiliares para uso posterior.De seguida temos uma breve descricao das principais estruturas de dados utilizadas:

- static ArrayList<String> Attributes:
  Guarda os atributos iniciais.
- static ArrayList<String[]> Examples:
  Guarda a lista de exemplos iniciais
- static int Example_size:
  Tamanho de cada exemplo.
- static Set<String> Classes:
  Guarda as classes diferentes que existem no DataSet
- static Map<String,Set<String>> atributo_variavel:
  Guarda os pares atributos variáveis iniciais (ajuda a obter impressão correta).

Classe ROOTNode

Na classe ROOTNode temos estruturas de dados que nos permitem representar um nó na arvore de decisão e também ter estruturas de dados para fazer cálculos como a entropia e o ganho. Segue-se uma descrição breve das estruturas de dados e atributos mais relevantes:

- String name_col:
  Indica o nome do atributo ou nome do nó folha (caso não seja um atributo ou um nó folha fica igual a null)
- String name_var:
  Indica o nome da variavel associada ao nó atual (caso nao seja uma variavel fica igual a null)
- Double Entropia_R:
  Contem a entropia a do no atual
- Map <String ,Set<Integer>> class_indice:
  Representa o mapa que associa uma classe ao indice de exemplos que pertencem a mesma.
- Map<String,Map<String,Set<Integer>>> var_class_ind:
  Representa um mapa, que dado uma variável e dado uma classe “retorna” a lista de índices dos exemplos presentes no nó atual que pertence a essa variável e classe.
- ArrayList <ROOTNode> filhos
  Representa os filhos de um nó na árvore de decisão, caso existam.
- ArrayList<String> Original_attributes:
  Representa a lista de atributos associadas a um nó.

ArrayList<String[]> Example_Population: Representa a lista de exemplo associada a um nó.

- Coluna[] colunas:
  Lista de todas as colunas associadas a uma dada tabela, que auxiliam a inicialização de outras estruturas de dados presentes como Map<String,Map<String,Set<Integer>>> var_class_ind e Map <String ,Set<Integer>> class_indice.

Classe Tabela

Representa a estrutura de dados auxiliar para calcular a entropia e auxiliar na criação da arvore de decisão. Os seus atributos e estruturas de dados mais relevantes são:

- ArrayList<String[]> examples:
  Contem os exemplos associadas a esta tabela.
- ArrayList<String> attributes;
  Contem os atributos associadas a esta tabela
- Coluna[] colunas;
  Contem todas as colunas associadas a tabela e cria as respetivas estruturas de dados auxiliares na classe Coluna.
- String best_splitting_attribute:
  Guarda o melhor *splitting atribute* possível nesta tabela.

Organização do Código

Para explicar a organização do código vamos mencionar a "contribuição" de cada classe para o problema e vamos também referir alguns dos seus principais métodos.

Classe Tabela

Decidimos criar uma classe especifica para a tabela para tornar mais fácil o acesso a estruturas de dados cruciais para a criação da arvore de decisão. Os métodos mais relevantes da classe Tabela são:

- Void Initialize():
  Inicializa o array de colunas (Coluna[] colunas).
- int getPos_col(String attribute):
  Retorna o índice em atributes correspondente a um atributo.
- String[] Remove_Col_From_Examples(String[] e,int col):
  Permite remover a coluna col de um exemplo e.
- Void Filter_Examples(boolean[] indices_remove,ArrayList<String[]> new_examples,ArrayList<String[]> old_examples):
  Método que permite filtrar exemplos, ou seja , remove todos os índice de exemplos presentes em índices_remove de old_examples, e coloca esses novos exemplos em new_examples.

- void Find_Best_Splitting_Attribute():
  É o método que faz o processo de escolher *best splitting attribute* e guarda no atributo best_splitting_attribute o nome da coluna correspondente.

Classe Coluna

Esta classe serve para representar uma coluna da tabela. Decidimos separar numa classe só, para facilitar a interpretação do código e para facilitar a própria implementação. Como métodos auxiliares consideramos importante referir os seguintes:

- void Iniatialize(int nr_col,ArrayList<String> col,ArrayList<String[]> examples):
  Inicializa a estrutura de dados auxiliar mapa_coluna, que dado uma variável e classe "retorna" os índices dos exemplos associados sobre o formato de uma string(num métodoe representação dos exemplo passa a ser feita com um conjunto de inteiros em vez de string).
- void Iniatialize():
  Inicializa a estrutura de dados auxiliar, que é semelhante a estrutura de dados mapa_coluna, mas em vez de guardar os índices dos exemplos sobre o formato de string , guarda num conjunto de inteiros.

Classe ROOTNode

Classe que permite representar um nó na arvore de decisão, decidimos só ter uma classe para representar um nó para simplificar a criação da arvore de decisão. Os métodos mais relevantes associados a esta classe são:

- void Iniatialize():
  Inicializa as estruturas de dados auxiliares Map <String ,Set<Integer>> class_indice e Map<String,Map<String,Set<Integer>>> var_class_ind
- void Calculate_Entropia():
  Permite calcular e guardar a entropia associada a este nó.

Classe Main

Classe "principal" onde temos a criação da arvore de decisão e o método main. Os métodos mais relevantes são os seguintes:

- String[] remove(String[] s,int indice):
  Permite remover a coluna indice de um exemplo s.
- void Filter_Examples(boolean[] indices_remove,int indice_col_remove,ArrayList<String[]> new_examples,ArrayList<String[]> old_examples):
  Método que permite filtrar exemplos, ou seja, remove todos os índice de exemplos presentes em índices_remove de old_examples, e coloca esses novos exemplos em new_examples.
- String Most_Common(ArrayList<String[]> examples:
  Retorna a classe mais comum na lista de exemplos exemples.

- ROOTNode ID3(ArrayList<String[]> examples,String Target_Attribute,ArrayList<String> attributes,String Ident):
  Método que Executa o algoritmo ID3 com algumas modificações.
- public static void discretize():
  Método que discretiza valores numéricos, caso existam.
- public static void main(String[] args) throws Exception:
  Corresponde ao método em que inicializamos a execução do nosso progama.Também é possível categorizar exemplos no main colocando para cada dataset o respetivo exemplo no formato que seja aceita para a arvore respetiva.

Funcionamento do programa

Neste tópico vamos explicar o funcionamento com o nosso programa, dividindo a explicação por as partes fundamentais.

Compilação

Para compilar devemos executar o seguinte comando:

```
rui@rui-VirtualBox:~/Desktop/IA/Trabalho_pratico_3$ javac *.java
```

Execução

Para executar o programa, que gera a arvore de decisão e testa um exemplo definido no método main da class Main, temos de colocar o seguinte comando:

```
rui@rui-VirtualBox:~/Desktop/IA/Trabalho_pratico_3$ javac *.java && java Main nome_do_csv.csv
```

É importante informar que na classe Main é necessário definir a variável String[] teste_ex (que se encontra no método main) com um exemplo apropriado para o data set dado(se não for um exemplo apropriado para o data set o programa devolve um erro ou não termina). Também é importante referir que esse exemplo deve respeitar, possíveis discretizações que tenham sido feitas (as discretizações feitas aos dados podem ser observadas na pasta com o nome Tabelas_Com_Discretizacao). De seguida temos alguns exemplos de teste possíveis e respetivo resultados:

- Restaurante.csv:

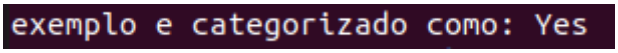
String[] teste_ex = {"X14","Yes","No","Yes","Yes","Full","$$$","No","Yes","Thai","0-10"};

```
exemplo e categorizado como: Yes
```

String[] teste_ex={"X14","Yes","No","Yes","Yes","None","$$$","No","Yes","Thai","0-10"};

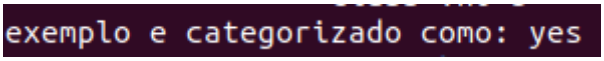
```
exemplo e categorizado como: No
```

String[]teste_ex={"X14","Yes","No","Yes","Yes","Full","$$$","No","Yes","French","0-10"};

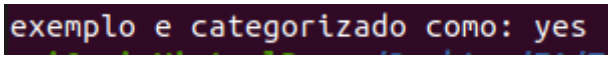


- Weather.csv:

String[] teste_ex = {"1","rainy",">74","<=80","FALSE"};



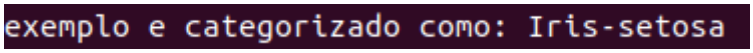
String[] teste_ex = {"1","overcast","<=74","<=80","FALSE"};



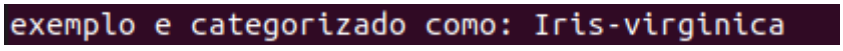
String[] teste_ex = {"1","sunny",">74",">80","FALSE"};

- Iris.csv:

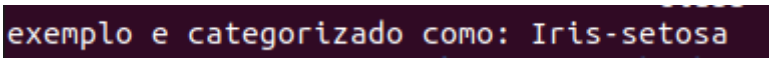
String[] teste_ex = {"32","<=6.1","<=3.2","<=3.95","<=1.3"};



String[] teste_ex = {"32","<=6.1","<=3.2",">3.95",">1.3"};

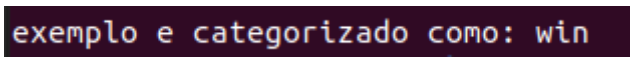


String[] teste_ex = {"32","<=6.1",">3.2","<=3.95",">1.3"};



- Connect4.csv:

String[] teste_ex
={"a1","o","o","b","b","b","b","b","b","b","b","b","b","b","x","b","b","b","b","b","b","b","b","b","b","b","b","x","x","b","b","b","b","b","b","b","b","b","b","b","x","o","o","b","b","b"};



Possíveis erros

Consideramos importante referir possíveis erros de execução, que levam o programa a não funcionar como pretendido.

- Erro proveniente da incorreta definição de o exemplo teste String[] teste_ex. Por exemplo definir um teste_ex para o data set restaurante e correr o programa sobre o data set weather.

String[]teste_ex={"X14","Yes","No","Yes","Yes","Full","$$$","No","Yes","French","0-10"};

```
javac *.java && java Main weather.csv
```

- Erro proveniente de ter um exemplo sem as discretizações feitas:
  Por exemplo definir um teste_ex para o data set weather que não inclui a discretização.

  String[] teste_ex = {"1","sunny","80","80","FALSE"};

```
javac *.java && java Main weather.csv
```

Resultados

De seguida temos as figuras que representam as arvores de decisão geradas para cada um dos data sets.

```
Atributo Pat
      Some:
         Class :Yes 4
      None:
         Class :No 2
      Full:
         Atributo Hun
               No:
                  Class :No 2
               Yes:
                  Atributo Type
                        French:
                           Class :Yes 0
                        Burger:
                           Class :Yes 1
                        Italian:
                           Class :No 1
                        Thai:
                           Atributo Fri
                                 No:
                                    Class :No 1
                                 Yes:
                                    Class :Yes 1
```

Figura 2 – Árvore gerada para restaurante.csv

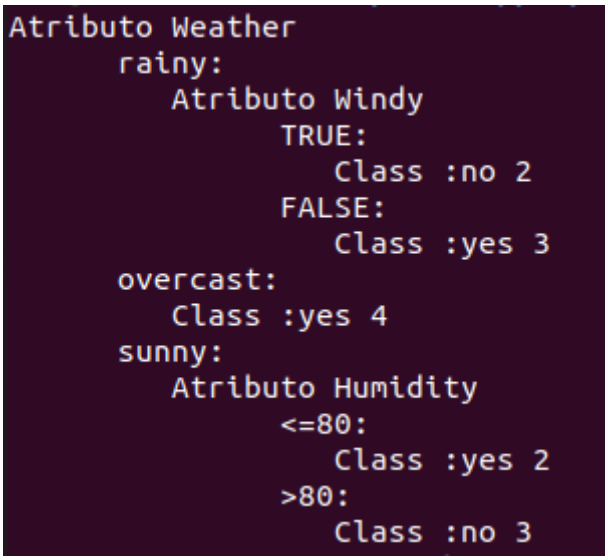


Figura 1 - Árvore gerada para weather.csv

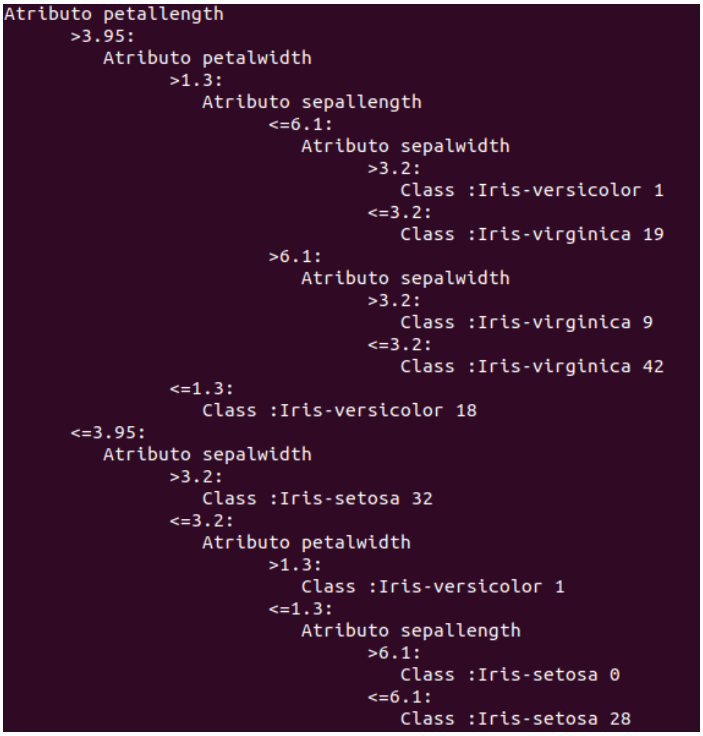


Figura 2 - Árvore gerada para iris.csv

Não incluímos a arvore gerada para o data set do connect4, mas está presente no zip associado ao projeto.

Conclusões

Com este trabalho apercebemo-nos de que as arvores de decisão podem ser usadas em vários contextos, contudo achamos que existem casos em que o uso das mesmas se torna mais complexo em vez ao uso de outros algoritmos (como da escolha da próxima jogada do connect4).

Bibliografia

Abedinia, A. (2019). *Survey of the Decision Trees Algorithms (CART, C4.5, ID3).* Disponível em https://medium.com/@abedinia.aydin/survey-of-the-decision-trees-algorithms-cart-c4-5-id3-97df842831cd

Breiman, L., Friedman, J., Olshen, R., & Stone, C. (1984). *Classification and Regression Trees*. CRC Press.

Luger, G. (2005). *Artificial Inteligence: Structures and Strategies or Complex Problem Solving*. (5a ed). Addison-Wesley.

Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction.* Springer.

Nilsson, N. (1998). *Artificial Inteligence: a new synthesis.* Morgan Kaufmann Publishers.

Ribeiro, P. (2022). *Estruturas de Dados.* Disponível em https://www.dcc.fc.up.pt/~pribeiro/aulas/edados2223/

Quinlan, J. R. (1993*). C4.5: Programs for Machine Learning.* Morgan Kaufmann Publishers.

Rokach, L., & Maimon, O. (2008). *Data Mining with Decision Trees: Theory and Applications*. World Scientific Publishing Co.

Russell, S., Norvig, P. (2022). *Artificial Intelligence: A Modern Approach*. (4ª edição). Pearson Education Limited. Global Edition.