Adversarial search strategies and Decision Trees

Leonor Oliveira Rodrigues, up202304869
Paulo Diogo Lopes Pinto, up202306412
Rui Filipe Rocha Alvarilhão, up202209989

FCUP - Faculdade de Ciências, Universidade do Porto

1 Introdução

Neste trabalho, abordamos duas metodologias essenciais para a tomada de decisão em problemas de jogo e de classificação: pesquisa adversarial e indução de árvores de decisão. Na primeira parte, apresentamos a implementação de um agente para o jogo ***Connect Four***, capaz de enfrentar tanto um jogador humano quanto outro algoritmo. Para isso, utilizamos o Monte Carlo Tree Search (MCTS) com o critério UCT (*Upper Confidence Bound for Trees*), criando diferentes versões do agente com o objetivo de melhorar o desempenho.

Para além disso, construímos uma árvore de decisão com o método ID3. Inicialmente, aplicamos o algoritmo ao *dataset* Iris, explorando a **discretização de atributos** e a avaliação de precisão. Em seguida, geramos um novo *dataset* com base nas partidas simuladas pelo MCTS e utilizámo-lo para treinar a árvore ID3, de modo a sugerir a melhor jogada em distintos estados do jogo.

O relatório inclui ainda um guia de utilização, com instruções para configurar o ambiente, instalar as bibliotecas necessárias e executar os ficheiros do projeto.

2 Guia de utilização

Este projeto foi desenvolvido em Python 3.10.12 e depende das seguintes bibliotecas:

- pygame 2.6.1
- numpy 2.2.3
- joblib 1.4.2
- scikit-learn 1.6.1
- matplotlib 3.10.0
- pandas 2.2.3

O arquivo ZIP contém um notebook organizado em três partes:

1. ***Iris*** – construção da árvore de decisão e análise das métricas de desempenho.
2. ***Tree Connect Four*** – criação da árvore para o jogo Connect Four e avaliação das suas métricas.

3. ***Game*** – interface gráfica que permite jogar contra três variantes do MCTS (MCTS2, MCTS3 e MCTSID3).

Além disso, o ZIP inclui uma pasta Dados, contendo os scripts usados para gerar o dataset e os resultados dos confrontos entre as diferentes versões do MCTS.

3 Monte Carlo Tree Search

O algoritmo MCTS é usado, principalmente, em jogos e outras simulações. É, portanto, necessário tomar decisões onde o espaço de procura contém muitos estados possíveis. O algoritmo pode ser definido e explicado em 4 etapas:

1. **Seleção**
   Nesta etapa o algoritmo percorre a árvore de decisão a partir da sua raiz, escolhendo os nós que segundo a fórmula UCT (*Upper Confidence bound*) parecem ser mais promissores. A fórmula contém uma constante que equilibra os conceitos de ***exploitation*** e ***exploration***. Quanto maior a constante mais peso dá a ***exploration*** e quanto menor mais peso dá a ***exploitation***.

2. **Expansão**
   Nesta fase, quando o algoritmo chega a um nó que ainda não foi completamente explorado, cria um novo estado a partir dele.

3. **Simulação**
   A partir do novo nó, o algoritmo simula o resto do jogo de forma aleatória até que o mesmo termine.

4. **Backpropagation**
   O resultado da simulação realizada é propagado pela árvore até chegar à raiz, aumentando o número de visitas de cada nó e caso tenha ganho aumenta também o número de vitórias.

Para escolher o próximo movimento, o critério utilizado foi o número de visitas do nó. Não se optou pelo critério: percentagem de vitórias, visto que um elevado número de visitas significa que esse ramo foi considerado promissor pela fórmula UCT no decorrer das simulações. Nós com poucas visitas têm elevada variância estatística na percentagem de vitórias. Assim, o número de visitas é um critério mais estável e robusto para a escolha da jogada.

No trabalho foram implementadas 4 versões diferentes do algoritmo Monte Carlo Tree Search (MCTS), cada uma delas concebida para tentar atenuar limitações identificadas nas versões anteriores. Para avaliar o impacto das modificações, realizamos 100 partidas controladas, para cada par de constantes de exploração na lista $[1,\sqrt{2},2,3,4,5]$. Deste modo, foi possível estudar cenários que

vão desde forte ***exploration*** (valores alto de C), até forte ***exploitation*** (valores baixos de C).

Para compensar a desvantagem do jogador em segundo lugar, em 50 das 100 partidas invertemos a ordem do jogador, ou seja, cada versão do MCTS joga 50 vezes como jogador 1 e outras 50 como jogador 2. Além disso, para evitar que a comparação entre as diferentes implementações seja justa, independente de diferenças de *hardware*, fixamos um limite de *rollouts* por jogada (35000 rollouts) em todas as simulações. Desta forma, a comparação torna-se justa, controlada e mais adequada para *benchmarks*. Os resultados, permitiram comparar a eficácia das 4 versões em diferentes *trade-offs* de ***exploitation*** e ***exploration***, mas também fundamentar a escolha da constante.

1º Implementação

Versão canônica do algoritmo MCTS. Nesta versão são sempre expandidos os 7 filhos de um nó antes de conseguir aplicar a fórmula UCT, com o objetivo de selecionar qual deles voltar a visitar. Consequentemente, todas as jogadas, no início, recebem a mesma atenção, mesmo estas sendo fracas. Este comportamento leva à má distribuição das simulações por parte do algoritmo. Assim, jogadas ótimas são exploradas numa fase tardia do algoritmo, principalmente no início, onde o ruído das simulações é mais elevado. Ao atrasar o uso da fórmula UCT e aplicar uma uniformidade inicial na expansão dos filhos, perdemos o poder da ***exploitation*** e exploration, uma vez que a ***exploration*** dispersa-se por ramos irrelevantes e a ***exploitation*** não consegue enfatizar, no início, jogadas favoráveis.

2º Implementação

Para mitigar o problema do ruído inicial na 1º implementação, colocamos em prática a estratégia *progressive widening*. Esta técnica retarda a expansão de novos filhos, até que o nó pai tenha sido visitado um certo número de vezes. De forma mais específica, o número de filhos expandidos num nó é dado pela fórmula:

$$k = \lceil n^{\alpha} \rceil$$

onde n é o número de visitas daquele nó. Optamos por $\alpha = 0{,}25$. Assim:

- Com 1 visita: $\lceil 1^{0,25} \rceil = 1$ filho
- Com 16 visitas: $\lceil 16^{0,25} \rceil = 2$ filhos
- Com 81 visitas: $\lceil 81^{0,25} \rceil = 3$ filhos

Deste modo o algoritmo consolida estatísticas, nos filhos já expandidos, antes de alargar a árvore, tornando-se assim robusto ao ruído inicial. O que melhora a confiança em jogadas favoráveis, evitando perda de tempo com jogadas pouco interessantes. *Progressive widening* permite então equilibrar ***exploration*** e ***exploitation*** controlada, pois ramos promissores recebem mais simulações antes de novos ramos surgirem. Além disso, na fase ***Backpropagation***, o valor que o algoritmo atribui ao empate foi alterado, passando de 0 para 0.5. Deste modo,

o MCTS aprende a diferenciar os 3 estados finais do jogo, preferindo assim o empate quando já não é possível vencer.

MCTS1 vs MCTS2

MCTS1 \ MCTS2	1	√2	2	3	4	5
1	9-41 (0)	13-37 (0)	21-29 (0)	6-44 (0)	5-45 (0)	5-45 (0)
√2	9-41 (0)	14-36 (0)	13-37 (0)	6-44 (0)	3-47 (0)	2-48 (0)
2	4-46 (0)	10-40 (0)	26-24 (0)	7-43 (0)	5-45 (0)	1-49 (0)
3	9-41 (0)	16-34 (0)	23-27 (0)	5-45 (0)	6-44 (0)	6-44 (0)
4	13-37 (0)	25-25 (0)	20-30 (0)	13-37 (0)	12-38 (0)	15-35 (0)
5	19-31 (0)	23-27 (0)	33-17 (0)	27-23 (0)	20-30 (0)	20-30 (0)

Fig. 1. MCTS1 vs MCTS2

MCTS2 vs MCTS1

MCTS2 \ MCTS1	1	√2	2	3	4	5
1	50-0 (0)	50-0 (0)	49-1 (0)	49-1 (0)	45-5 (0)	32-18 (0)
√2	49-1 (0)	50-0 (0)	50-0 (0)	48-2 (0)	38-12 (0)	34-16 (0)
2	48-2 (0)	50-0 (0)	50-0 (0)	49-1 (0)	47-3 (0)	32-18 (0)
3	47-3 (0)	50-0 (0)	49-1 (0)	48-2 (0)	33-17 (0)	19-31 (0)
4	50-0 (0)	50-0 (0)	50-0 (0)	49-1 (0)	36-14 (0)	26-24 (0)
5	49-1 (0)	50-0 (0)	50-0 (0)	50-0 (0)	46-4 (0)	20-30 (0)

Fig. 2. MCTS2 vs MCTS1

Os resultados obtidos demonstram a superioridade do 2º MCTS sobre o 1º MCTS em, praticamente, todos os cenários testados, o que valida empiricamente as nossas hipóteses teóricas. Mesmo quando joga em segundo, uma posição claramente desvantajosa no jogo *Connect Four*, o 2º MCTS vence a maioria dos jogos. Para constantes entre 1 e 3, o 2º MCTS demonstrou melhores desempenhos tanto como 1º jogador como 2º jogador. Já para as constantes 4 e 5, demonstrou uma ligeira desvantagem ao jogar em segundo, o que indica que demasiada exploração prejudica a consolidação de boas jogadas nos filhos já expandidos. O 2º MCTS

demonstrou a sua capacidade de focar-se em jogadas favoráveis, o que lhe permitiu ganhar robustez estratégica. Dominou o jogo, mesmo sob desvantagem, mostrou resiliência com várias configurações da constante, validando assim a eficácia da estratégia implementada. Resumindo, o 2º MCTS reduziu drasticamente o impacto do ruído das simulações iniciais e concentrou maior esforço computacional nas jogadas mais relevantes.

3º Implementação

Na terceira implementação, procuramos resolver uma limitação observada nas versões anteriores, **a expansão aleatória dos filhos**. Ao testar a 2º versão contra humanos, reparamos que nem sempre a defesa era feita de forma correta e muito raramente jogava em posições que o faziam perder vantagem. Para mitigar este problema, inseriu-se na expansão uma heurística determinística, dividida em 3 fases:

- Expandir uma jogada que dá vitória imediata ao algoritmo;
- Expandir a jogada que dá vitória ao adversário;
- Se não existir vitórias ou ameaça de derrota, escolhemos uma jogada aleatória;

Se a 1ª ou a 2ª condição forem verificadas a jogada crítica é priorizada na expansão. Como o algoritmo já usa a estratégia *progressive widening*, a jogada crítica é uma das poucas a ser explorada, o que aumenta a sua estatística e probabilidade de expansão. Com esta nova forma de abordar o jogo, procuramos reforçar o comportamento tático, evitando derrotas desnecessárias e aproveitando melhor as oportunidades de vitória. O *progressive widening* garante que essas jogadas não se percam entre outros ramos irrelevantes.

MCTS2 vs MCTS3

MCTS2 \ MCTS3	1	√2	2	3	4	5
1	31-19 (0)	34-16 (0)	36-14 (0)	22-27 (1)	24-26 (0)	26-24 (0)
√2	23-27 (0)	27-23 (0)	26-24 (0)	15-35 (0)	14-36 (0)	11-38 (1)
2	19-31 (0)	23-27 (0)	29-21 (0)	19-31 (0)	11-39 (0)	17-33 (0)
3	17-33 (0)	27-23 (0)	25-25 (0)	10-40 (0)	3-47 (0)	5-45 (0)
4	8-42 (0)	19-31 (0)	22-27 (1)	4-46 (0)	5-45 (0)	11-39 (0)
5	15-35 (0)	13-36 (1)	28-22 (0)	7-43 (0)	6-44 (0)	3-47 (0)

Fig. 3. MCTS2 vs MCTS3

MCTS3 vs MCTS2

MCTS3 \ MCTS2	1	√2	2	3	4	5
1	28-22 (0)	36-14 (0)	28-22 (0)	19-31 (0)	14-36 (0)	20-30 (0)
√2	23-27 (0)	28-22 (0)	33-17 (0)	22-28 (0)	8-42 (0)	13-37 (0)
2	16-34 (0)	22-28 (0)	25-24 (1)	15-35 (0)	10-40 (0)	9-41 (0)
3	15-35 (0)	24-26 (0)	28-22 (0)	11-39 (0)	8-42 (0)	9-41 (0)
4	17-33 (0)	22-28 (0)	23-27 (0)	13-37 (0)	8-42 (0)	4-46 (0)
5	11-39 (0)	16-32 (2)	22-28 (0)	15-35 (0)	6-44 (0)	7-43 (0)

Fig. 4. MCTS3 vs MCTS2

Ao contrário do que ocorreu na comparação anterior, desta vez observamos a ocorrência de empates. Isso reforça a ideia de que a nossa decisão de distinguir entre derrota e empate tem, de facto, impacto no comportamento do algoritmo. A 3ª implementação mostrou ser superior quando foi o jogador 2, tornando-se mais nítida a diferença para constantes maiores. Em contrapartida, a sua performance caiu drasticamente quando foi o jogador 1 , principalmente para constantes maiores. Algumas razões que explicam este comportamento são:

- Ao priorizar jogadas críticas, em que ganha ou perde, o algoritmo tornou-se muito eficaz a responder a ameaças. Assim, quando é o segundo jogador, ele pode ver e reagir ao plano do adversário, neutralizando-o. Desta forma, é vantajoso jogar em segundo.
- O critério de expansão torna o algoritmo muito reativo. Quando ele passa a jogar em primeiro ele tem de tomar a iniciativa e a sua estratégia de expansão deixa de ter o mesmo efeito.
- Por último, a constante usada, principalmente quando é elevada, favorece demais a exploração, o que leva o algoritmo a perder tempo em ramos óbvios. Essa abordagem mostra-se eficaz como 2º jogador, mas representa uma limitação significativa quando atua como 1º jogador.

Logo, esta implementação levou o algoritmo a favorecer a defesa e o contra ataque, mas limitou-o na liderança e na criação de estratégias. Estes resultados vieram também confirmar que, embora a estratégia *progressive widening* combinada com a fórmula UCT é o principal motor da força ofensiva do algoritmo, a heurística implementada desloca o equilíbrio para uma abordagem mais defensiva.

4º Implementação

Na quarta implementação, procuramos modificar a política aleatória dos *rollouts*, passando a basear-se em estatística extraídas de um dataset. Para isso foi gerado, tal como pedido no enunciado do trabalho, um dataset com o estado do tabuleiro e a coluna jogada. Contudo, decidimos usar uma abordagem mais sofisticada e inspirada nas políticas de rede usadas no AlphaGo (Silver et al., 2016). Em vez de termos 43 colunas, ampliamos para 100 colunas. Vamos começar então por explicar as que representam o estado do jogo:

- **p0_ricj** (42 colunas): representa o estado do tabuleiro para o jogador 1. Cada célula é binária, 1 se naquela posição estiver a peça do jogador 1 e 0 caso contrário. A árvore de decisão consegue, assim, entender o padrão das peças do jogador 1.
- **p1_ricj** (42 colunas): a mesma lógica que as colunas anteriores, mas para o jogador 2.
- **last_ci** (7 colunas): representam a última jogada, 1 caso tenha sido aquela coluna como última jogada e 0 caso contrário. Permite perceber o fluxo do jogo e capturar a repetição de padrões.
- **legal_ci** (7 colunas): representam as colunas que estão disponíveis para jogar. Através delas, procuramos evitar que a árvore sugira um movimento ilegal.
- **player** (1 coluna): 1 se foi o player 2 e 0 se foi o player 1. Permite à árvore adaptar a decisão a diferentes posições do jogo.
- **move** (1 coluna): é a classe que pretendemos que a árvore aprenda a prever.

Este esquema evita complicações no cálculo do ganho de informação, pois todas as colunas preditoras são binárias, o que facilita a seleção de atributos pela árvore de decisão. Além disso, como representamos apenas o tabuleiro e o contexto do mesmo, estamos a permitir ao algoritmo usar sinais que bons jogadores de *Connect-Four* usam para tomar decisões.

Para gerar o *dataset* e permitir que o algoritmo conseguisse distinguir boas e más jogadas, decidimos usar os diferentes MCTS criados ao longo do trabalho. Deste modo, enriquecemos o dataset com diferentes formas de jogar, realizando 2500 jogos. Cada MCTS foi limitado a 2 segundos de pensamento. O uso do tempo em vez do número máximo de *rollouts* deve-se às seguintes razões:

- **Simulação de condições de jogo em tempo real:** Jogadores humanos e bots muitas vezes têm restrição de tempo. Simular este critério permite que o algoritmo MCTS produza jogadas mais realistas no contexto de um jogo em tempo real. Além disso, como o dataset vai ser usado para treinar o MCTS que joga contra humanos e IA's, valida mais uma vez o uso do tempo.

- **Adaptação dinâmica à complexidade do estado:** Permite que o algoritmo se adapte à dificuldade do estado. Em posições simples são realizados mais *rollouts* e em posições difíceis o algoritmo gasta mais tempo a explorar e a refletir sobre a jogada.

- **Evitar *over* computação em estados triviais:** Estados finais ou de vitórias não requerem imensos *rollouts*. Ao limitar o tempo, o algoritmo consegue avançar mais rapidamente nas jogadas, evitando gerar dados redundantes e desperdiçar simulações.

Deste modo melhoramos a robustez do modelo em cenários reais e introduzimos variabilidade no número de simulações por estado, o que enriquece o dataset.

Os 2500 jogos foram divididos da seguinte forma:

- 1000 jogos para MCTS3 vs MCTS2, exploramos a fraqueza do MCTS3 como jogador 1 e o forte ataque do MCTS2 como jogador 2.
- 1000 jogos para MCTS2 vs MCTS3, exploramos a vantagem do MCTS3 a criar contra ataques e aproveitar-se dos erros do jogador 1.
- 125 jogos para MCTS3 vs MCTS1 e 125 jogos para MCTS1 vs MCTS3, exploramos os erros do MCTS1 contra um algoritmo extremamente defensivo.
- 125 jogos para MCTS2 vs MCTS1 e 125 jogos para MCTS1 vs MCTS2, exploramos os erros do MCTS1 contra um algoritmo extremamente ofensivo.

Através desta distribuição evitamos que o modelo ID3 aprenda apenas um estilo de jogo de um único MCTS.

Após gerar os dados e criar o modelo ID3, decidimos aplicá-lo à 2ª implementação do MCTS, uma vez que já tem uma boa base ofensiva e defensiva, a árvore de decisão pode refinar exclusivamente a política de *rollout*. Procuramos assim reduzir a variância dos *rollouts* e acelerar a convergência das estimativas da fórmula UCT, pois cada simulação carrega informação relevante. Não será aplicado na 3ª implementação, devido aos conflitos que pode gerar com a heurística do algoritmo. A heurística de vitória/bloqueio, já altera drasticamente a ordem de expansão dos filhos, adicionar a árvore de decisão pode criar conflitos de prioridade e enviar excessivamente contra-ataques.

Para encontrar a combinação dos hiperparâmetros: max_depth, min_samples_split e min_samples_leaf, que maximiza a performance, sem tornar a árvore demasiado profunda e tentando maximizar o número de rollouts na simulação,realizámos um grid search sobre os seguintes valores:

- `max_depth` = 9, 10, 11, 12
- `min_samples_leaf` = 10, 15, 20, 35
- `min_samples_split` = 15, 20, 35, 40

O dataSet tinha aproximadamente 44000 observações e avaliamos cada combinação, a que teve melhor performance foi:

- Profundidade máxima (**max_depth**): 12
- Número mínimo de exemplos por folha (**min_samples_leaf**): 10
- Número mínimo de exemplos para divisão (**min_samples_split**): 15

Estas restrições de profundidade e de observações mínimas por nó, garantem que a árvore se mantenha compacta, essencial para não aumentar o tempo de rollout do MCTS e evite o *overfitting*, criando assim uma árvore robusta que se generaliza bem mesmo com uma estrutura compacta. Em particular, limitamos a profundidade a 12 para evitar que o *rollout* gerado pela árvore reduzisse demasiado o número de simulações do MCTS por movimento, garantindo um *trade-off* entre qualidade e velocidade.

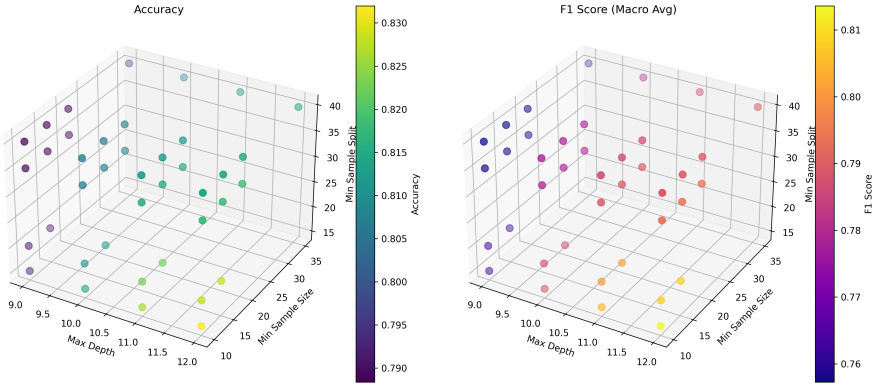


Fig. 5. Grid Search

Depois de construirmos a árvore de decisão com o algoritmo ID3, avaliamos o seu desempenho com as seguintes métricas: *accuracy*, *precision*, *recall* e *f1-score*. Através do módulo *classification_report* da biblioteca sklearn, obtivemos os seguintes dados:

```
accuracy on test set:  83.19747769916319 %
              precision    recall  f1-score   support

    coluna_0      0.792     0.814     0.803       758
    coluna_1      0.763     0.801     0.782      1252
    coluna_2      0.882     0.733     0.800      2539
    coluna_3      0.867     0.949     0.907      3937
    coluna_4      0.779     0.831     0.804      2560
```

coluna_5	0.839	0.701	0.764	1211
coluna_6	0.836	0.835	0.836	747
accuracy			0.832	13004
macro avg	0.823	0.809	0.814	13004
weighted avg	0.834	0.832	0.830	13004

Com uma *accuracy* global de 83.2%, tendo em conta as 7 colunas possíveis, demonstra uma performance bastante elevada para um classificador discreto simples como o ID3. O *f1-score macro* de 81.4% sugere que o desempenho é equilibrado entre as classes, mesmo que algumas tenham mais observações que outras. Além disso o *F1-score weighted* é aproximadamente 83.0%, sendo esta métrica calculada fazendo a ponderação do número de exemplos em cada classe, confirmando, assim, que o bom desempenho não vem só das colunas mais frequentes. A escolha dos parâmetros da complexidade, permitiu evitar o *overfitting* e criar uma árvore robusta que se generaliza bem mesmo com uma estrutura compacta.

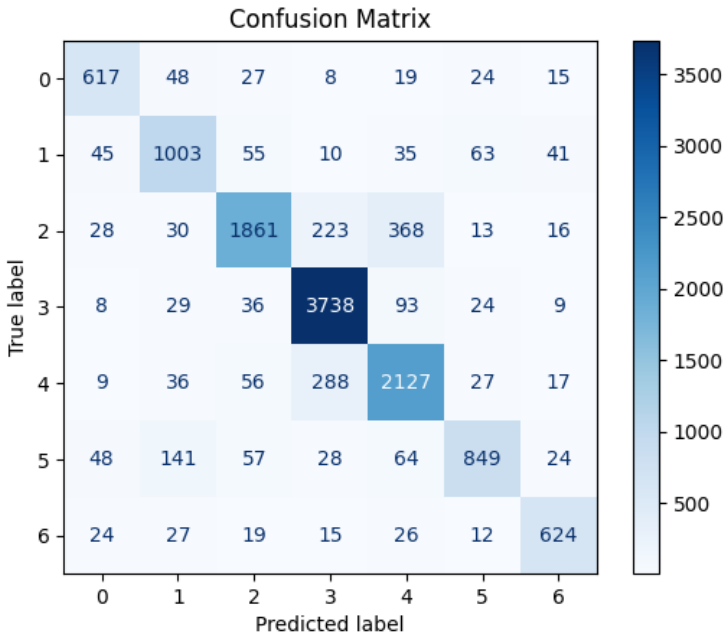


Fig. 6. Confusion Matrix

A árvore de decisão ID3 parece favorecer as colunas centrais, especialmente a coluna 3, o que a torna consistente com as estratégias humanas e a estrutura do

jogo, o que mostra que a árvore aprendeu padrões táticos importantes. As colunas laterais, especialmente a 0 e a 6, têm menos suporte mas não são ignoradas pelo classificador, um ponto bastante positivo, pois o modelo consegue reconhecer situações onde estas jogadas são válidas. O equilíbrio entre *recall* e *precision*, em todas as classes, demonstra que o modelo não é enviesado para prever apenas um tipo de jogadas, pois consegue captar nuances táticas com base nas *features* fornecidas.

A árvore de decisão ID3, demonstrou ser um modelo eficaz para prever a próxima jogada, com uma precisão de aproximadamente 83% e com excelente desempenho, especialmente nas colunas centrais. A decisão de manter todas as *features* binárias e uma estrutura com 100 colunas revelou-se benéfica, o que facilitou o cálculo do ganho de informação e a segmentação por padrões. Com estes resultados esperamos que a árvore tenha um excelente desempenho ao ser integrada na política de *rollout* orientada do MCTS.

MCTS2 \ MCTSID3	1	√2	2	3	4	5
1	30-20 (0)	36-14 (0)	37-13 (0)	23-27 (0)	21-29 (0)	35-15 (0)
√2	19-31 (0)	35-15 (0)	36-14 (0)	17-32 (1)	11-39 (0)	12-38 (0)
2	22-28 (0)	30-20 (0)	28-22 (0)	16-34 (0)	10-40 (0)	7-43 (0)
3	15-35 (0)	18-32 (0)	28-22 (0)	7-43 (0)	7-43 (0)	4-46 (0)
4	13-37 (0)	18-32 (0)	27-23 (0)	10-40 (0)	2-48 (0)	8-42 (0)
5	13-36 (1)	18-32 (0)	20-30 (0)	7-43 (0)	5-45 (0)	5-45 (0)

Fig. 7. MCTS2 vs MCTSID3

MCTSID3 \ MCTS2	1	√2	2	3	4	5
1	24-26 (0)	30-20 (0)	34-16 (0)	25-25 (0)	19-31 (0)	20-30 (0)
√2	19-30 (1)	26-24 (0)	29-21 (0)	15-35 (0)	18-32 (0)	11-39 (0)
2	15-35 (0)	23-27 (0)	30-20 (0)	17-33 (0)	10-40 (0)	10-40 (0)
3	16-33 (1)	24-26 (0)	27-23 (0)	10-40 (0)	10-40 (0)	2-48 (0)
4	7-43 (0)	17-33 (0)	23-27 (0)	11-39 (0)	8-42 (0)	4-46 (0)
5	11-38 (1)	23-27 (0)	29-21 (0)	6-44 (0)	3-47 (0)	3-47 (0)

Fig. 8. MCTSID3 vs MCTS2

Como primeiro jogador, a 4ª implementação do algoritmo obteve resultados interessantes e competitivos para constantes menores ou iguais a $\sqrt{2}$, principalmente quando o MCTS2 utiliza constantes menores. O mesmo não se pode dizer quando o MCTSID3 utiliza constantes maiores ou iguais a 3, onde o seu desempenho caiu, principalmente quando o MCTS2 utiliza constantes maiores. Para constantes menores, o MCTSID3 beneficia mais da política de rollout da ID3, para escolher jogadas táticas fortes no início, o que compensa jogar em 1º. Para constantes mais altas, como a fórmula UCT prioriza visitar nós pouco explorados, reduzimos o peso das jogadas aprendidas pelo classificador ID3, o que leva a uma má performance.

Como segundo jogador, a 4ª implementação do algoritmo obteve resultados melhores. Quando o MCTS2 tem constante 1, este manteve uma pequena vantagem. Para qualquer outra constante usada no MCTS2, o MCTSID3 passa a vencer a maior parte das partidas. Esta vantagem torna-se ainda mais clara quando ambas as constantes usadas nos algoritmos aumentam. A política de rollout de ID3, derivada de jogos onde o MCTS3 neutraliza ameaças, confere ao MCTSID3 forte capacidade defensiva e de contra-ataque. Deste modo o MCTSID3 aproveita para antecipar e bloquear as jogadas do inimigo no rollout.

MCTSID3 \ MCTS3	1	√2	2	3	4	5
1	26-24 (0)	26-24 (0)	33-17 (0)	22-28 (0)	25-25 (0)	25-25 (0)
√2	19-30 (1)	22-27 (1)	34-16 (0)	19-31 (0)	16-34 (0)	17-33 (0)
2	16-34 (0)	19-31 (0)	21-29 (0)	10-39 (1)	9-41 (0)	8-42 (0)
3	17-33 (0)	18-32 (0)	19-30 (1)	13-37 (0)	5-45 (0)	7-43 (0)
4	16-34 (0)	22-27 (1)	24-26 (0)	10-40 (0)	3-47 (0)	5-45 (0)
5	10-39 (1)	14-35 (1)	21-29 (0)	11-38 (1)	8-42 (0)	2-48 (0)

Fig. 9. MCTSID3 vs MCTS3

MCTS3 \ MCTSID3	1	√2	2	3	4	5
1	33-17 (0)	26-24 (0)	38-12 (0)	17-33 (0)	19-31 (0)	23-27 (0)
√2	19-31 (0)	24-26 (0)	28-22 (0)	14-36 (0)	12-38 (0)	11-39 (0)
2	20-30 (0)	25-25 (0)	28-21 (1)	13-37 (0)	10-40 (0)	10-40 (0)
3	9-41 (0)	16-34 (0)	20-30 (0)	10-40 (0)	4-46 (0)	8-42 (0)
4	17-32 (1)	19-31 (0)	22-28 (0)	5-45 (0)	2-48 (0)	3-47 (0)
5	14-36 (0)	18-32 (0)	19-31 (0)	12-38 (0)	5-45 (0)	5-45 (0)

Fig. 10. MCTS3 vs MCTSID3

Concluímos que, como primeiro jogador, a 4ª implementação do algoritmo obteve resultados satisfatórios principalmente para constantes menores ou iguais a 2, principalmente quando enfrenta o MCTS3 com constantes igualmente baixas. Em contrapartida, o seu desempenho caiu significativamente com constantes maiores ou iguais a 3, tanto do lado do MCTSID3 como do MCTS3. Isso deve-se ao facto de que, como já referido, valores mais elevados na constante da fórmula UCT, favorecem a exploração de nós pouco visitados, sendo reduzido o peso das jogadas aprendidas pelo classificador ID3, comprometendo a sua eficácia. Além disso, o MCTS3 é muito eficaz a responder a ameaça, por isso, quando é o segundo jogador, pode neutralizar facilmente o adversário.

Como segundo jogador, a 4ª implementação com constantes maiores (como 3, 4 ou 5) obteve resultados significativamente superiores. Nesses casos, a maior liberdade para explorar novas possibilidades parece ser vantajosa, dado que o MCTSID3 parte de um estado parcialmente definido, podendo reagir com estratégias mais adaptativas. Com constantes mais baixas, os resultados tendem a ser mais equilibrados, embora o MCTSID3 se mostre superior neste caso.

Em suma, os resultados indicam que o MCTSID3 beneficia de constantes baixas quando joga primeiro, para aproveitar completamente a política de rollout ofensivo. No entanto, quando joga em segundo, constantes mais elevadas revelam-se eficazes, sugerindo que a política de rollout do MCTSID3 é sensível ao equilíbrio entre ***exploitation*** e ***exploration*** e que esse equilíbrio ideal pode depender da ordem de jogo.

4 Iris - DataSet

Ao analisar o *DataSet* podemos verificar que existem 4 atributos, todos eles do tipo numérico contínuo. Atendendo que o algoritmo ID3 não lida diretamente com este tipo de dados foi necessário aplicar uma discretização nos mesmos. Para simplificar esse processo e torná-lo mais eficiente, desenvolvemos a função *getPredictorsDomain*, que devolve uma lista com os possíveis valores que cada atributo pode ter. Ao separar os dados, para cada valor do domínio de cada atributo, dois subconjuntos são formados, um contendo amostras cujo valor do atributo é maior ou igual a esse valor, e outro com amostras cujo valor é menor. A entropia condicional é determinada para cada divisão e, entre todas as opções, escolhemos o par atributo/valor que tiver maior ganho de informação. As amostras com valores menores vão para o nó esquerdo enquanto as demais vão para o nó direito.

Para avaliar o desempenho do modelo, utilizamos *Stratified K-Fold Cross-Validation*, como k igual a 5. Como o dataset é composto por 150 observações, distribuídas uniformemente por 3 classes (50 por classe) esta técnica é apropriada, uma vez que garante que cada *fold* mantém a mesma proporção de classes, essencial para evitar viés na avaliação. Assim, obtivemos uma avaliação mais robusta e representativa da sua performance, em diferentes subconjuntos dos dados. Em conjunto como o método anteriormente descrito foram utilizadas 3 métricas diferentes para avaliar o desempenho do modelo:

1. **Accuracy**
   Como o dataset é balanceado, esta estatística é adequada ao problema. Através desta métrica obtemos uma indicação geral do nosso modelo e conseguimos perceber a quantidade de acertos.

2. **Macro F1-score**
   Calculamos a média do F1-score para cada uma das classes individualmente. Uma vez que o dataset é balanceado, podemos dar o mesmo peso para todas as targets. Esta métrica combina precision e recall,

medindo o equilíbrio entre falsos positivos e falsos negativos.

3. **Macro ROC AUC**
   Calcula a área sob a curva ROC (Receiver Operating Characteristic), utilizando a estratégia um-vs-resto, ou seja, considera uma classe em relação às outras. O resultado final é a média das AUCs de cada classe. A partir desta métrica é possível avaliar a capacidade do modelo de distinguir as classes.

Para mitigar o risco de overfitting, foi criada uma restrição. Cada atributo, após ser usado para fazer uma divisão, só pode ser utilizado mais uma vez em cada subárvore geradas pela divisão. Assim evitamos que o mesmo atributo seja usado várias vezes para realizar a divisão. Além disso, para otimizar os hiperparâmetros da ID3, realizámos um grid search sobre os seguintes valores:

- depth = 2, 3, 4, 5
- min_samples_leaf = 2, 3, 4, 5

Este atributos foram escolhidos tendo em conta a dimensão reduzida do dataset. Avaliamos cada combinação através *Stratified K-Fold Cross-Validation* e a que teve melhor performance foi:

- depth = 3
- min_samples_leaf = 2

Ao limitar a profundidade e o número mínimo de amostras por folha, conseguimos criar um modelo robusto, que evita decisões baseadas em subdivisões muito específicas (ruído), mantendo a interpretação fácil e a capacidade de generalização

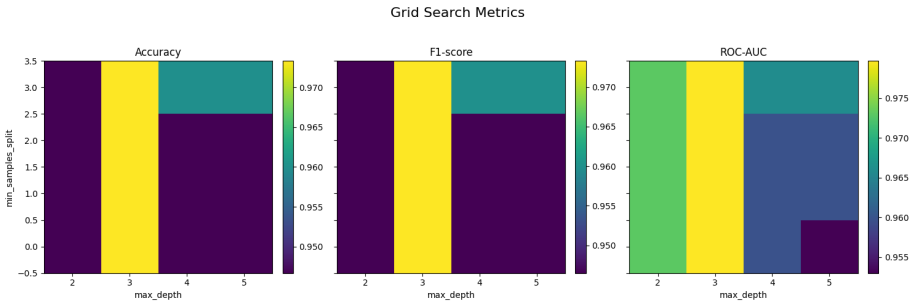


Fig. 11. Grid Search

No nosso modelo obtivemos uma *accuracy* de 97%. Dado que o dataset Íris é perfeitamente balanceado, este valor verifica que a árvore está a generalizar

muito bem os dados. Os valores obtidos para o *F1-score para cada classe* são:

- Setosa: 1,00
- Versicolor: 0,96
- Virginica: 0,96

O *F1-score* na classe Setosa revela que não houve qualquer falso positivo, nem falso negativo para esta classe. Todas as observações foram perfeitamente identificadas. Para as classes Versicolor e Virginica ainda existe um pequeno desequilíbrio entre a precision e o *recall*, logo houve casos que o modelo não consegue diferenciar as classes. Apesar disso, como o *F1-score* é superior a 9%, o modelo continua a ser excelente, pois raramente existem falsos positivos e falsos negativos. Os valores obtidos para ROC-AUC para cada classe são:

- Setosa: 1,00
- Versicolor: 0,98
- Virginica: 0,96

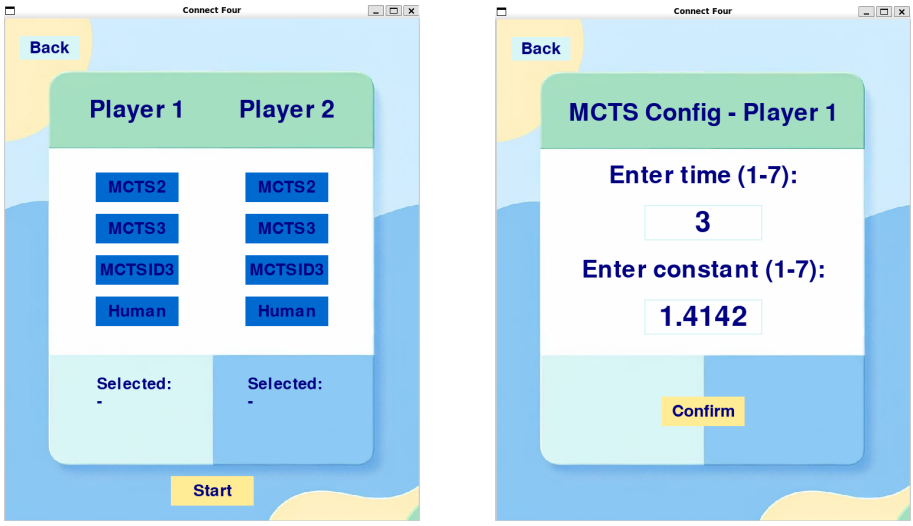
O AUC de 1,00 para a classe Setosa confirma mais uma vez a separação perfeita desta classe em relação às outras. Embora a seja ligeiramente mais difícil distinguir as classes Versicolor e Virginica, o que já é esperado, dada a semelhança biológica entre as espécies, os valores são muito próximos de 1, indicando uma excelente descriminação.

O valor *macro F1-score* de 0.97 mostra que no geral, o modelo atingiu um equilíbrio excelente entre *precision* e *recall*. O valor macro ROC-AUC de 0.98, reforça que a ID3 consegue distinguir quase perfeitamente cada espécie das outras.

Os resultados obtidos confirmam que o modelo gerado, em conjunto da discretização realizada e a validação cruzada estratificada, atinge uma performance excelente no *dataset* Iris. A perfeita classificação da classe Setosa e o desempenho quase perfeito nas classes Versicolor e Virginica, demonstram que a árvore de decisão capturou de forma eficaz as fronteiras de decisão nos atributos.

5 Interface

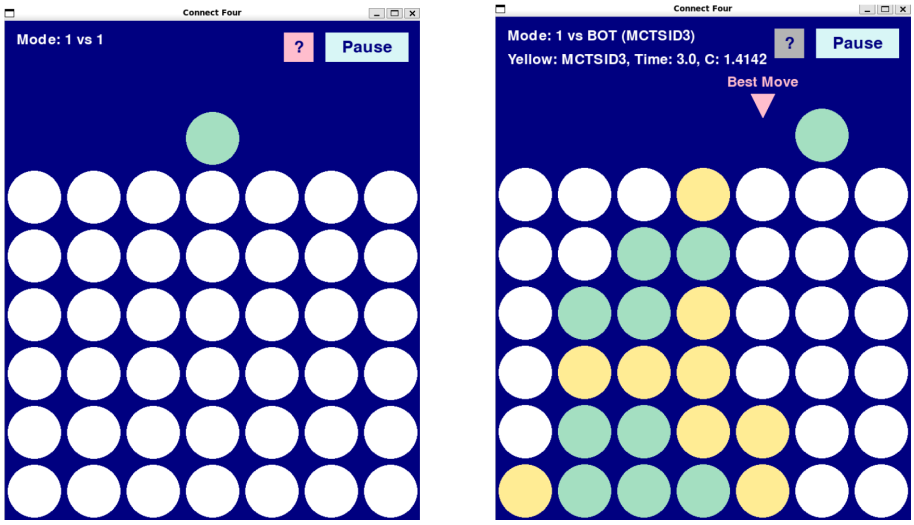
Na interface desenvolvida (Fig.12), o utilizador pode selecionar quem será o primeiro e o segundo jogador. As opções disponíveis incluem não só os algoritmos **MCTS2**, **MCTS3**, **MCTSID3**, como também a opção **Human**. Quando um dos MCTS é selecionado, surge a possibilidade de definir os parâmetros ***time*** e ***constant***, como mostra a Fig.12(b). Estes campos já aparecem preenchidos com os valores **3** e $\sqrt{\mathbf{2}}$ **(aproximadamente 1.4142)**, respetivamente, uma vez que esta combinação demonstrou ser a mais eficaz durante os testes realizados, conforme discutido nos pontos anteriores.



(a) Seleção dos modos de jogo (b) Configurações do MCTS

Fig. 12. Interface de seleção e configuração

Já no jogo, no canto superior direito, estão presentes dois botões: o de ajuda ("?") e o de pausa ("Pause"). O botão de ajuda permite que o jogador receba uma sugestão sobre a melhor jogada a realizar, caso necessite de apoio estratégico. Para esse efeito, é utilizado o MCTSID3 com *time* de **3** segundos e *constant* **1.4142 ($\sqrt{2}$)**. Como já foi referido, o MCTSID3 revela-se particularmente eficaz na identificação e resposta a ameaças. Assim, observa a jogada realizada pelo adversário e reage de forma a neutralizá-lo. Por esse motivo, o MCTSID3 é uma excelente escolha para fornecer sugestões durante o jogo, já que compara rapidamente diferentes possibilidades e, com base no contexto atual, propõe a jogada mais promissora.



(a) Jogo

(b) Melhor movimento indicado pelo MCTS3

Fig. 13. Interface de jogo e sugestão de jogada

O botão de pausa, por sua vez, permite ao utilizador regressar ao jogo, recomeçar a partida ou deslocar-se para o menu principal.

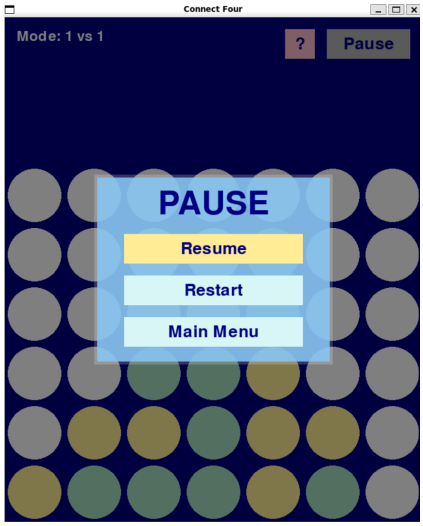


Fig. 14. Menu de Pausa

Para além da interface desenvolvida para o jogo, foram criadas duas interfaces adicionais para visualização dos classificadores ID3,uma para o conjunto de dados Iris e outra para o *Connect* 4, bem como exibição das métricas de avaliação desses modelos. Na interface da árvore de decisão aplicada ao *dataset* Iris, o utilizador pode fornecer os quatro atributos de entrada e, como saída, obtém a classe prevista pelo ID3 e a probabilidade associada à escolha desse ramo no nó correspondente.

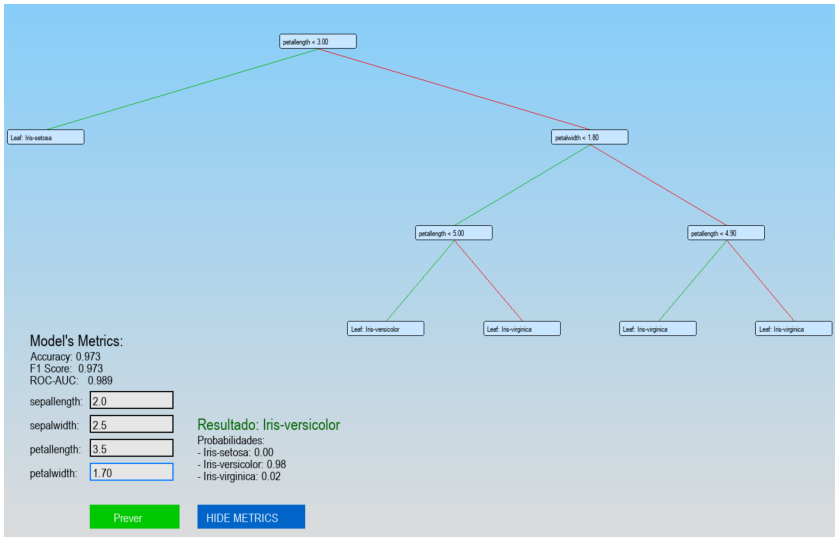


Fig. 15. Iris ID3

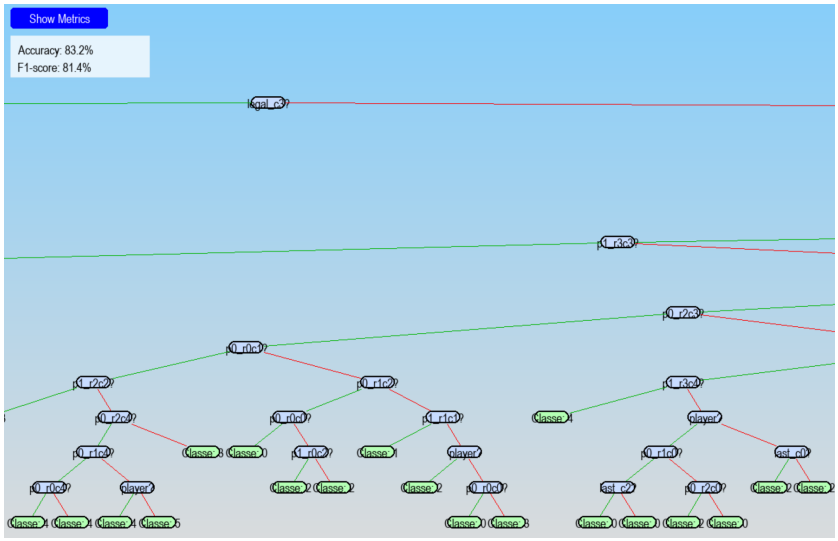


Fig. 16. Connect 4 ID3

References

[1] Docentes da UC: Moodle da unidade curricular
[2] Hui, J.: Alphago – how it works technically. https://jonathan-hui.medium.com/alphago-how-it-works-technically-26ddcc085319 (2018)
[3] Tabletop Games AI Wiki: Mcts - tabletop games ai wiki. https://tabletopgames.ai/wiki/agents/MCTS (2024)