Pontifícia Universidade Católica do Rio Grande do Sul

Inteligência Artificial

Engenharia de Software

Carolina Ferreira, Gustavo Willian, Mateus Caçabuena e Vitoria Gonzalez

Tic Tac Toe com ML

Porto Alegre

2025

1. Introdução e Dataset

Este trabalho tem como objetivo desenvolver uma solução de Inteligência Artificial (IA) capaz de classificar corretamente o estado atual de um tabuleiro 3x3 do jogo da velha (tic-tac-toe). Diferente de uma IA que joga, aqui o foco está em identificar automaticamente se o jogo ainda está em andamento, se houve vitória de X, vitória de O ou empate. A aplicação foi construída como parte do primeiro trabalho prático da disciplina de Inteligência Artificial da PUCRS.

Para viabilizar essa classificação, foi necessário construir um dataset adequado, contendo diferentes configurações de tabuleiro já finalizadas ou em andamento. O ponto de partida foi o conjunto de dados “Tic Tac Toe Endgame”, disponível no repositório da UCI. No entanto, ajustes foram necessários para torná-lo mais representativo e balanceado. Inicialmente, todas as classes foram limitadas a 16 amostras para uniformizar com a classe "Empate", que tinha baixa ocorrência. Entretanto, os resultados com esse volume foram insatisfatórios.

Diante disso, o dataset foi ampliado para conter aproximadamente 300 amostras por classe (X venceu, O venceu, Tem jogo), enquanto a classe "Empate" permaneceu com 32 exemplos, totalizando 932 amostras. Cada instância representa o estado completo de um tabuleiro, com nove posições indicadas por x (jogador X), o (jogador O) ou b (casa vazia), além do rótulo correspondente ao estado do jogo.

O conjunto de dados foi dividido de forma estratificada em:

- 80% para treinamento
- 20% para teste

Não foi criado um conjunto de validação separado, devido ao número limitado de amostras. Essa preparação cuidadosa dos dados foi fundamental para garantir a eficácia da IA na tarefa proposta.

2. Soluções

3.1 k-NN

3.1.2 Implementação

As demonstrações foram realizadas utilizando o valor **k = 6**, com variações entre 6 e 10 para avaliar o desempenho do modelo. A escolha da distância de Manhattan como métrica foi baseada em sua adequação a dados discretos, como as marcações do jogo da velha.

Durante os testes com **valores de k entre 1 e 30**, observou-se que o desempenho do K-NN variava consideravelmente. Valores muito baixos de k apresentaram sinais de *overfitting*, enquanto valores muito altos resultaram em perda de precisão, evidenciando uma menor sensibilidade do modelo. Os melhores resultados com acurácia média superior a **0.64** em validação cruzada.

Com base nesses testes, o valor ideal de k para o modelo com a métrica Manhattan foi definido entre 6 e 7, por equilibrar bem precisão e generalização nos dados de treino e teste.

Figura 1: Matriz de Confusão KNN

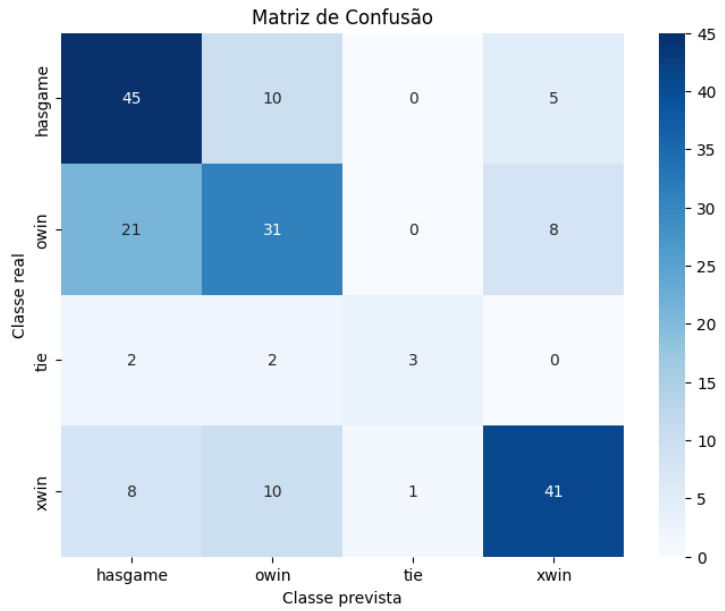


Figura 2: Desempenho do Classificador por Classe KNN

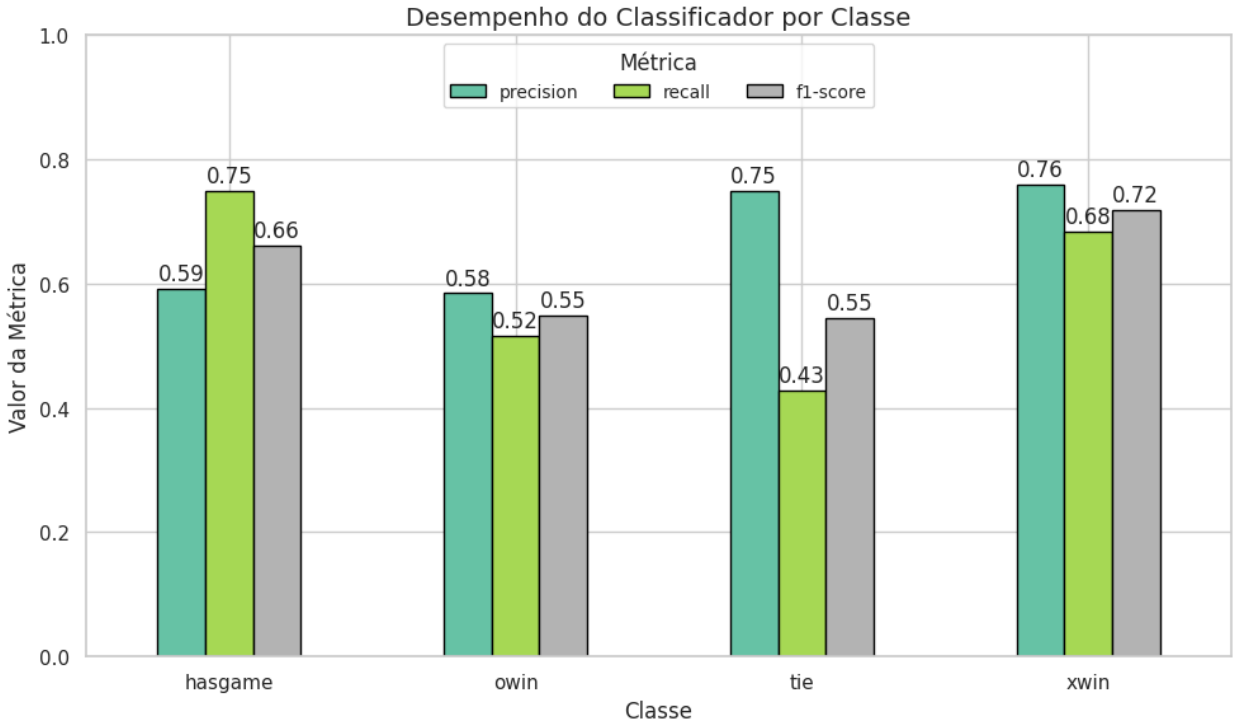


Figura 3: Determinação do melhor K

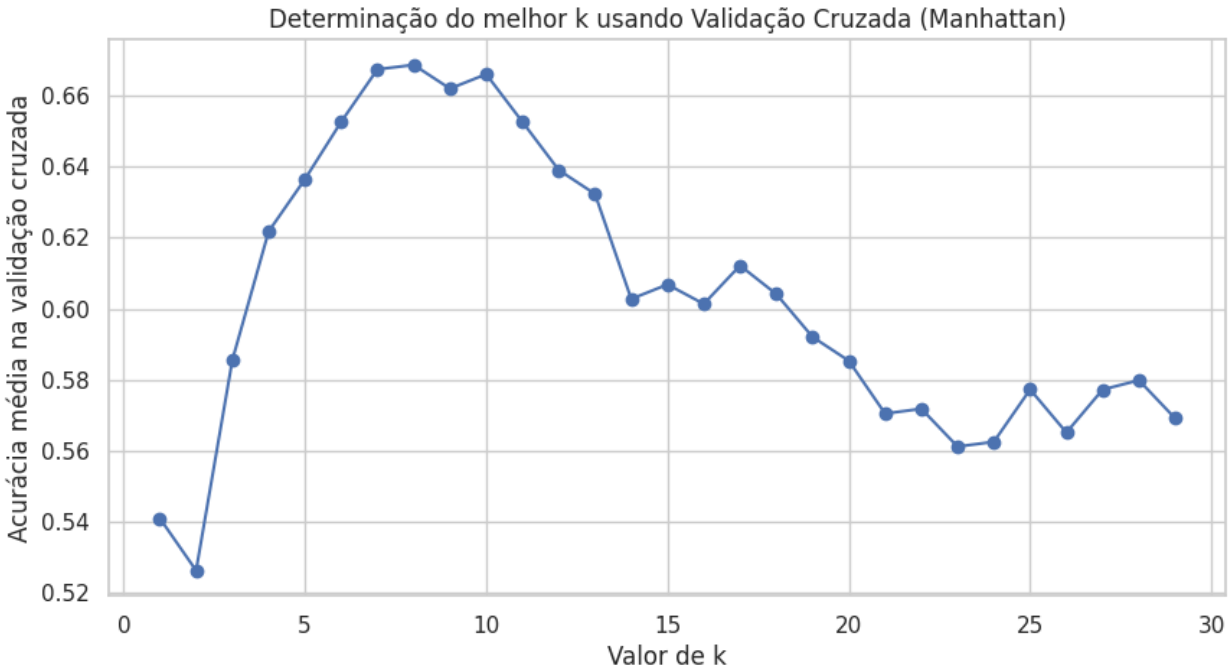
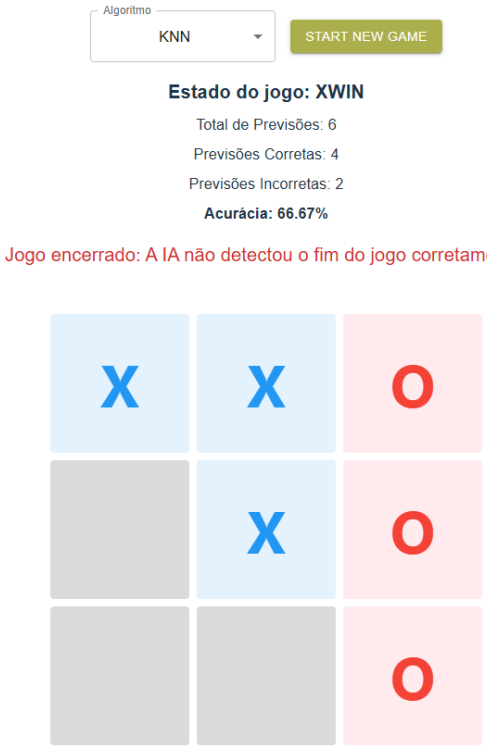


Figura 4: Exemplo de Erro KNN

Figura 5: Exemplo Fim de Jogo KNN



3.2 MLP

3.2.2 Implementação

A implementação do algoritmo MLP foi realizada utilizando a biblioteca scikit-learn, com base em um conjunto de dados que representa o estado de tabuleiros do jogo da velha. A primeira topologia testada foi uma arquitetura com 9 neurônios na entrada (um para cada posição do tabuleiro), uma camada oculta com 10 neurônios e uma camada de saída com 4 neurônios (um para cada classe). Com esta arquitetura, obteve-se uma acurácia de 43,17%. A partir disso, verificou-se oscilações no valor da função de perda (loss), de forma que foram testadas diferentes taxas de aprendizado, ajustou-se diversas vezes o *momentum, max_iter* e *hidden_layer_sizes* até alcançar uma acurácia aceitável e evitando o *overfitting.* Também foi testada uma arquitetura com camadas ocultas de 18 e 10 neurônios, porém a acurácia obtida ficou em torno de 60,9% e o melhor *loss* foi aproximadamente 0.103. Então, o número de neurônios da camada oculta foi incrementado e a arquitetura final contou com camadas ocultas de 36 e 18 neurônios, taxa de aprendizado de 0.01 e máxima iteração de 2000. Após todos os ajustes, obteve-se um modelo com 67% de acurácia.

Figura 6: Matriz de Confusão MLP

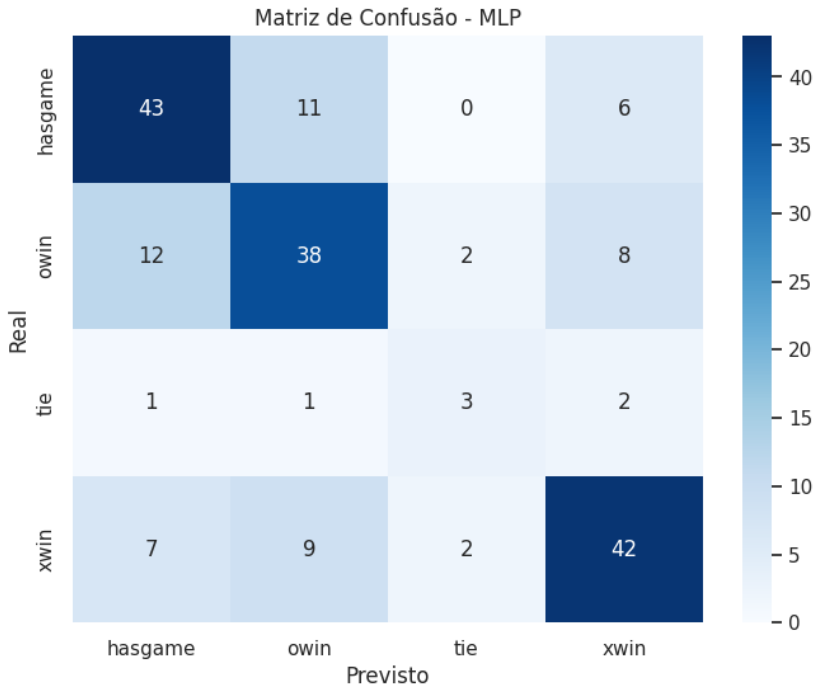


Figura 7: Exemplo de Erro MLP

Figura 8: Exemplo de Fim de Jogo MLP

3.3 Árvores de decisão

3.3.2 Implementação

O algoritmo foi implementado utilizando a biblioteca scikit-learn. O conjunto de dados utilizado foi previamente dividido na fase de preparação dos dados onde manualmente dividimos em treinamento.csv e teste.csv, mantendo a proporção de classes. A árvore foi treinada com max_depth=6, após testes para evitar *overfitting*. Foram utilizadas 9 features representando as posições do tabuleiro (com valores: 1 = X, 0 = O, -1 = vazio).

Figura 9: Matriz de Confusão Árvore de Decisão

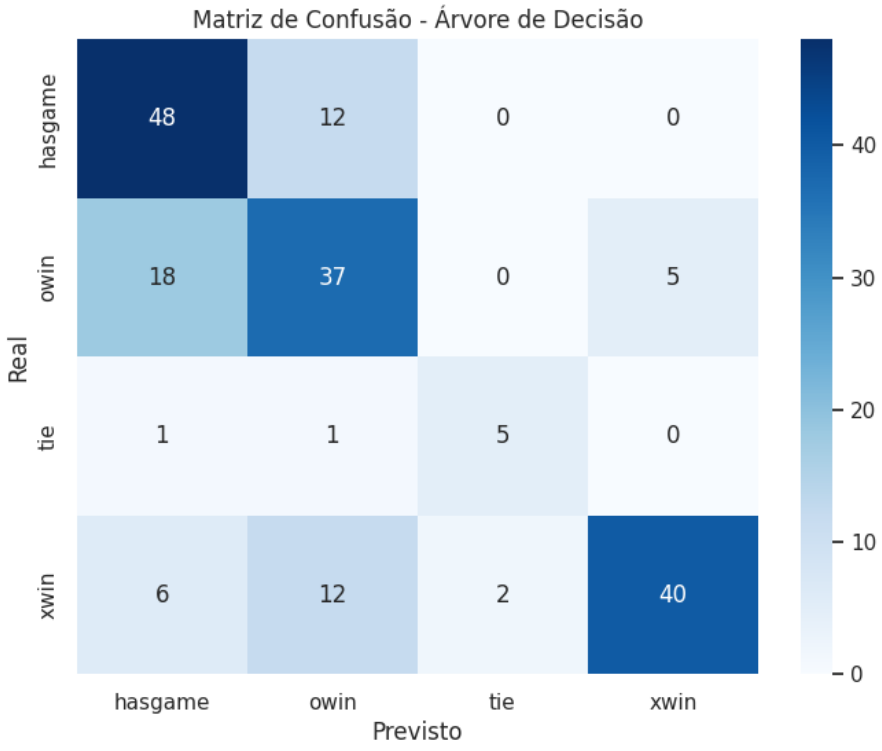


Figura 10: Exemplo de Erro Árvore de Decisão

Figura 11: Exemplo Fim de Jogo Árvore de Decisão

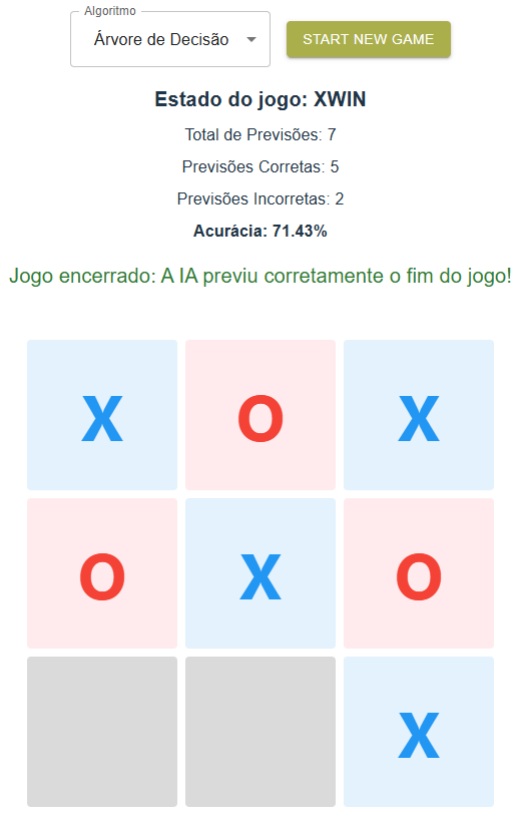
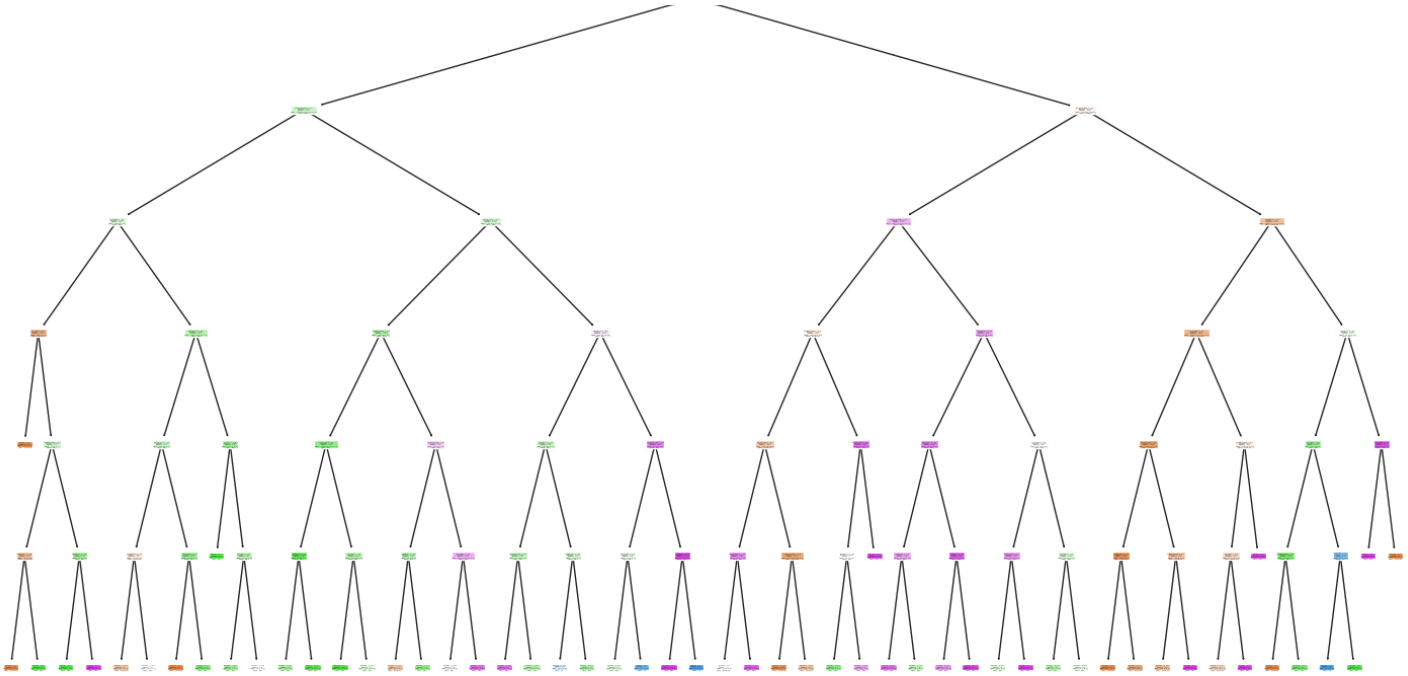


Figura 12: Árvore de Decisão do Jogo da Velha



3.4 SVM

3.4.2 Implementação

A implementação do algoritmo SVM foi realizada por meio da classe SVC, disponível na biblioteca scikit-learn. Foram realizados testes utilizando dois tipos distintos de kernel: rbf (função de base radial) e poly (polinomial), ambos com os valores padrão para os parâmetros de regularização (C), largura do kernel (gamma) e grau do polinômio, quando aplicável. Após o treinamento de ambos os modelos, foi feita a predição sobre os dados de teste utilizando o método .predict(), e os resultados foram comparados com base na acurácia obtida. O kernel rbf apresentou desempenho superior em relação ao poly, motivo pelo qual foi selecionado para análise final. Com base no modelo treinado com kernel rbf, foi gerada a matriz de confusão, permitindo uma avaliação mais detalhada da distribuição dos acertos e erros por classe.

Figura 13: Matriz de Confusão SVM

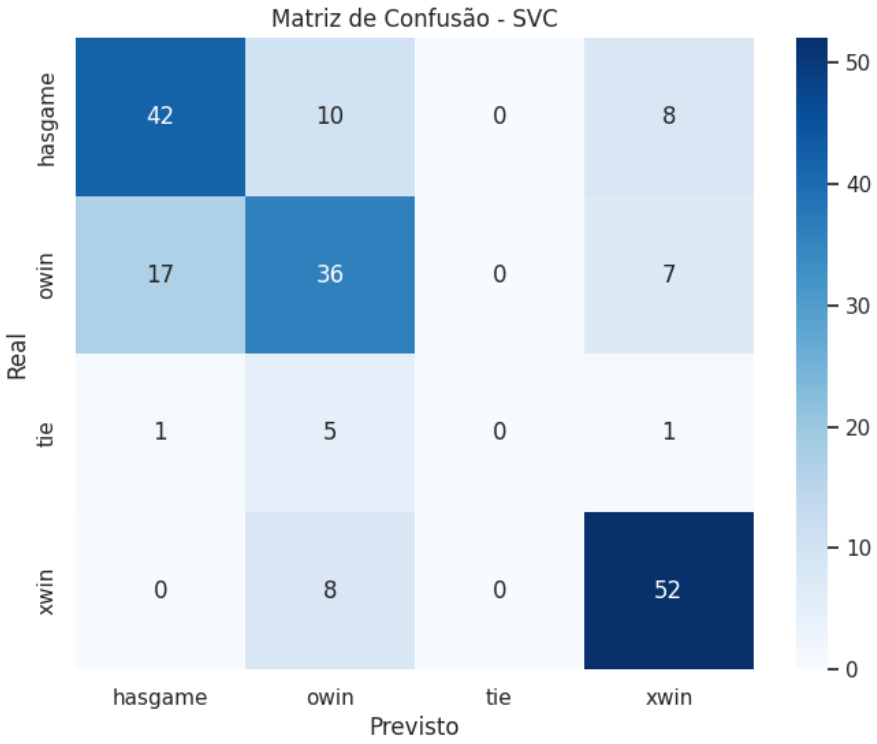
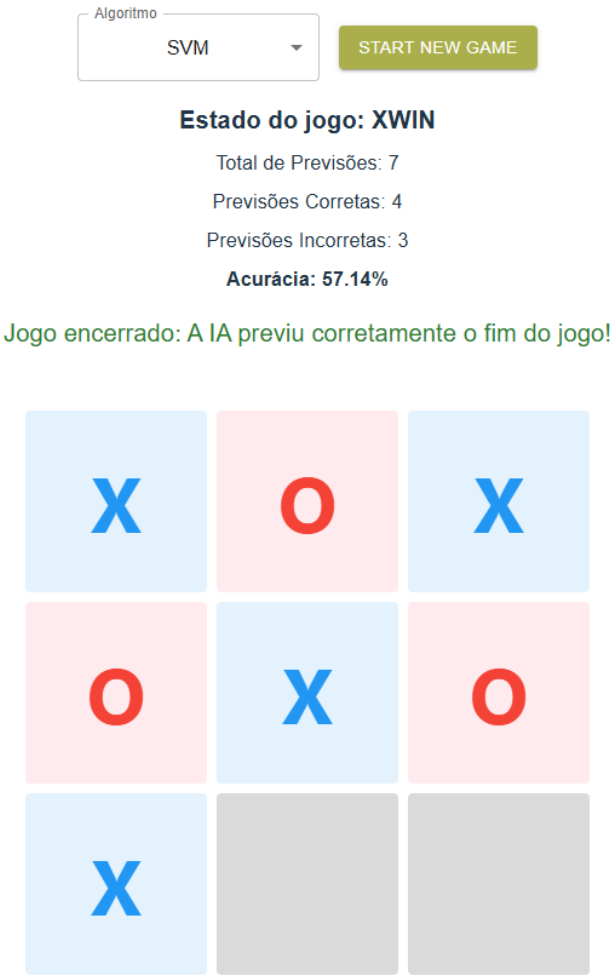


Figura 14: Exemplo Erro SVM

Figura 15: Exemplo Fim de Jogo SVM



4. Comparações

Abaixo, comparamos o desempenho final dos modelos treinados no conjunto de teste:

Algoritmo	Acurácia	Precisão	Recall	F1-Score
k-NN (k=6)	64,17%	67,15%	59,46%	61,87%
MLP	67,37%	61,98%	61,96%	61,95%
Árvore Decisão	**69,51%**	**71,43%**	**69,94%**	**70,11%**
SVM (RBF)	70,58%	53,91%	55%	54,35%

A Árvore de Decisão apresentou o melhor equilíbrio geral entre acurácia e f1-score, sendo escolhida como a melhor solução para o problema.

5. Resultado

Todos os modelos testados apresentaram bons índices de acurácia com os dados do conjunto de teste, sendo a Árvore de Decisão o mais eficiente no desempenho geral. No entanto, ao aplicar a IA em situações fora do dataset original, especialmente em jogos ainda em andamento (*hasgame*), surgiram erros recorrentes de classificação — esses casos foram muitas vezes confundidos com vitórias de X ou O.

Esse padrão de erro indica que a classe *hasgame* está sub-representada nos cenários mais complexos do jogo, o que compromete a generalização da IA. Nos exemplos que estavam bem representados no dataset, os modelos obtiveram sucesso, mas os dados limitados para jogos em progresso reforçam a necessidade de incluir maior variedade de exemplos intermediários para melhorar a cobertura da IA em contextos reais.

6. Conclusão

Este trabalho permitiu aplicar conceitos de Inteligência Artificial em um problema prático: a classificação de estados do jogo da velha. Entre os principais desafios enfrentados, destacam-se a limitação e desbalanceamento do dataset, especialmente na classe *Empate* e nos casos de *hasgame*, que geraram erros recorrentes na classificação de jogos em andamento.

Apesar disso, o projeto proporcionou aprendizados significativos sobre o funcionamento e a comparação entre diferentes algoritmos supervisionados (k-NN, MLP, Árvores de Decisão e SVM). Como enfatizado anteriormente, O modelo da Árvore de Decisão se destacou como o mais eficaz na maioria dos testes.

A construção do dataset, o ajuste de hiperparâmetros e a integração da IA com um *frontend* simples contribuíram para consolidar o conhecimento técnico do grupo, além de reforçar a importância de dados bem estruturados para o sucesso de soluções baseadas em IA.

7. Referências

Durante o desenvolvimento, foram utilizadas ferramentas de IA para otimizar a implementação do projeto. A IA da IDE Cursor auxiliou na lógica e na estilização do front end, enquanto o ChatGPT foi utilizado para integrar a IA com o front end e organizar a comunicação com os modelos de classificação.

Além disso, diversas fontes foram consultadas para aprofundar o entendimento teórico e prático sobre os algoritmos implementados (SVM, MLP, k-NN e Árvores de Decisão), como artigos técnicos, documentações oficiais e tutoriais. Entre os principais estão Artigos técnicos e tutoriais da plataforma Geeks for Geeks, Documentação oficial do Scikit-learn e publicações em blogs e plataformas como Medium e Baeldung