Tic Tac Toe com ML

Augusto Baldino, Felipe Freitas, Isabela Kuser Araujo,
Luiza Heller, Maria Eduarda Maia e Paola Lopes

June 3, 2024

1 Introdução

Este relatório apresenta o desenvolvimento e a implementação de um sistema de Inteligência Artificial (IA) para o clássico jogo da velha em um tabuleiro 3x3. O objetivo principal deste trabalho é construir uma IA capaz de analisar o estado atual do tabuleiro e classificar esse estado em quatro possíveis categorias: 'Tem jogo', 'X Venceu', 'O Venceu' e 'Deu Velha' (empate).

A solução de IA proposta neste trabalho envolve a implementação e teste de três algoritmos classificadores: k-NN, MLP e árvores de decisão. Cada algoritmo foi treinado e validado usando o mesmo conjunto de dados, permitindo uma comparação justa de seu desempenho. A acurácia foi escolhida como a métrica de avaliação principal para medir o desempenho dos algoritmos.

Além disso, foi desenvolvido um front-end mínimo para o jogo da velha, permitindo a interação entre um jogador humano e uma máquina que joga de forma aleatória (ou dois jogadores humanos). A cada turno, a solução de IA é invocada para determinar o estado do jogo e fornecer feedback ao usuário.

Este relatório documenta em detalhes todas as etapas do processo, desde a análise e preparação dos dados até a implementação e avaliação dos algoritmos de IA. Também são apresentados os resultados obtidos e uma discussão sobre a eficácia da solução proposta.

2 Passos

2.1 Dataset

Utilizamos o dataset disponível no link [UCI Machine Learning Repository - Tic-Tac-Toe Endgame Data Set](https://archive.ics.uci.edu/dataset/101/tic+tac+toe+endgame), que contém instâncias do tabuleiro de jogo da velha, para análise e verificação das necessidades do problema proposto. Inicialmente, verificamos que o dataset não atendia aos requisitos de maneira adequada. Em nossa primeira tentativa, testamos um dataset contendo 16 instâncias balanceadas, o que acabou por complicar o nosso trabalho e não trouxe os resultados esperados.

Posteriormente, realizamos uma modificação mais substancial, adicionando 64 novas instâncias representando 'Tem Jogo' (denominada 'neutral') e 16 instâncias de empates (Deu Velha - denominado 'tie'). Esta modificação final resultou em um dataset mais robusto, embora menos balanceado, permitindo-nos testar e utilizar os dados de maneira eficaz. Após testarmos esse dataset percebemos que não tinha dados suficientes para os algoritmos saberem com maior certeza o estado 'Tem Jogo'.

Por isso, adicionamos mais 2000 linhas de testes com diversas casos do 'Tem Jogo'. Com esta abordagem, conseguimos atingir os objetivos do nosso projeto de forma satisfatória. Apesar disso, há uma ressalva de que os algoritmos tiveram grandes dificuldades com o estado 'Deu Velha', o que se manteve até o final pois não conseguimos mais do que 16 exemplos de empate no jogo iniciado por X.

2.2 Classificação

Nesta parte, utilizamos comparações muito simples para transformar o dataset em um formato mais 'computável'. Para isso, iteramos sobre o dataest inicial e comparamos a coluna classe com cada um dos 4 valores esperados, classificando-os em uma das seguintes categorias:

- Tem Jogo: Indica que o jogo ainda está em andamento e nenhum jogador venceu.
- X Venceu: Indica que o jogador X venceu a partida.
- O Venceu: Indica que o jogador O venceu a partida.
- Deu Velha: Indica que o jogo terminou em empate, ou seja, nenhum jogador venceu.

Ainda, transformamos cada instância de 'x', 'o' e 'b' (conforme dataset inicial da UCI) para valores númericos, mais especificamente, optamos por utilizar 1 (positivo) para representar o 'X', -1 (negativo) para representar o 'O' e 0 (neutro) para representar 'b' (espaço não preenchido).

2.3 Solução de IA

2.3.1 Desenvolvimento do código

Após ser tratado o dataset, foi desenvolvido um código em Python que implementa um sistema capaz de avaliar o estado de um jogo da velha (Tic Tac Toe) utilizando técnicas de aprendizado de máquina. Na configuração inicial são estabelecidos os possíveis resultados do jogo da velha, como "X Ganhou", "O Ganhou", "Deu Velha" e "Tem Jogo", e são definidos tipos literais para representar os tipos de células do jogo, como "x" ou 1, "o" ou -1 e "b" ou 0 para vazio. Além disso, é criada uma classe chamada Cell, que representa uma célula do tabuleiro do jogo, com métodos para conversão entre representação textual e numérica. Estas precauções foram implementadas para garantir consistencia e comparações mais seguras ao longo do código, além de permitir mais fácil manutenção.

O código conta também com algumas funções auxiliares, como to_cell, que converte uma entrada textual ou numérica em uma célula válida, e print_board, responsável por imprimir o tabuleiro do jogo de forma legível. Em seguida, o dataset do jogo da velha é importado e pré-processado, sendo transformado para representar o tabuleiro do jogo de forma numérica, conforme explicado no parágrafo anterior.

Após o pré-processamento, o dataset é dividido em conjuntos de treinamento, validação e teste, com proporções pré-definidas e constantes (através do parâmetro random_state), e são exibidas informações sobre a distribuição dos resultados em cada conjunto. Finalmente, é realizada uma verificação visual do dataset de validação para garantir que os dados estejam corretos. Em suma, o código fornece uma estrutura sólida para o desenvolvimento de modelos de aprendizado de máquina capazes de prever o resultado de um jogo da velha com base no estado atual do tabuleiro.

2.3.2 KNN

Após o que foi citado acima, temos a parte que conta com o algoritmo kNN (k-Nearest Neighbors), que foi implementado para classificar o jogo da velha. O processo começa apresentando um gráfico de distorção e outro de inércia para dar uma ideia visual de qual deveria ser o número de k (seguindo o método do cotovelo). Posteriormente, é determinando o melhor valor de k, que representa o número de vizinhos mais próximos considerados para fazer uma previsão. Um loop é iniciado, percorrendo os valores de k de 1 até o número do dobro de instâncias com o resultado "Deu Velha" no conjunto de teste - um valor arbitrário pequeno. Para cada valor de k, um classificador kNN é criado e treinado com o conjunto de treinamento. Em seguida, são feitas previsões para o conjunto de teste e a acurácia é calculada comparando as previsões com os rótulos verdadeiros. Se a acurácia atual for maior ou igual à melhor acurácia registrada até o momento, o valor de k atual é definido como o melhor k e a acurácia correspondente é atualizada. Após o loop, o melhor valor de k encontrado e sua respectiva acurácia são impressos para identificação.

Em seguida, o kNN é configurado com o melhor valor de k encontrado e é treinado novamente com os conjuntos de treinamento. A acurácia do modelo é avaliada nos conjuntos de teste e validação usando o método '.score()' e as acurácias resultantes são impressas para avaliação do desempenho do modelo. Esse processo é crucial para selecionar um modelo com boa capacidade de generalização para novos dados, uma vez que diferentes valores de k podem afetar significativamente o desempenho do algoritmo kNN.

Na execução dos testes do algoritmo kNN, foram avaliadas diversas configurações de k, variando de 1 a 32. Para cada valor de k, a acurácia do modelo foi calculada utilizando o conjunto de teste. Após a análise dos resultados, foi identificado que o melhor valor de k para esse problema específico

foi 7, com uma acurácia de aproximadamente 0.9303. Esse valor de k foi selecionado devido à sua alta acurácia na classificação do jogo da velha. A validação desse modelo também foi realizada, resultando em uma acurácia de aproximadamente 0.9342. Esses resultados indicam que o modelo kNN com k=7 apresentou um desempenho consistente e preciso na classificação do jogo da velha.

2.3.3 MLP

Após o kNN é implementado o algoritmo MLP (Multi-Layer Perceptron) para classificar o jogo da velha. O MLP é uma rede neural artificial composta por múltiplas camadas de neurônios, incluindo uma camada de entrada, uma ou mais camadas ocultas e uma camada de saída. A configuração do MLP inclui a definição de parâmetros importantes, como a semente para a inicialização aleatória dos pesos, o otimizador utilizado para ajustar os pesos durante o treinamento e a arquitetura da rede neural, que é determinada pelo número de neurônios em cada camada oculta (diversos valores aleatórios foram testados). Após a configuração, o MLP é treinado com os conjuntos de treinamento, onde os pesos da rede são ajustados iterativamente para minimizar a função de perda.

Além disso, o relatório de classificação fornece métricas detalhadas de precisão, recall, f1-score e suporte para cada classe. Destaca-se que a classe "Deu Velha" apresenta uma precisão e recall de 0%, indicando que o modelo não conseguiu prever corretamente essa classe. Por outro lado, as classes "O Ganhou", "Tem Jogo" e "X Ganhou" apresentam métricas mais favoráveis, com precisões de 98%, 99% e 99%, respectivamente, e recalls de 100%, 100% e 97%, respectivamente - ou seja, o modelo deveriar funcionar muito bem para prever todas as saídas que possuem diversos exemplos.

A acurácia geral do modelo no conjunto de teste foi de 99%, o que indica a proporção de previsões corretas em relação ao total de previsões. Esses resultados fornecem uma avaliação abrangente do desempenho do MLP na classificação do jogo da velha, destacando suas áreas de sucesso e áreas que podem precisar de melhorias. Mais especificamente, em testes manuais, não observou-se essa precisão para determinar os casos onde ainda havia jogo, com o modelo prevendo quase em 100% dos casos que algum dos jogadores já havia vencido a partida.

2.3.4 Árvore de Decisão

Nesta parte do código, é implementado o algoritmo de árvore de decisão para classificar o jogo da velha. O modelo de árvore de decisão é configurado com diversos parâmetros importantes, como a semente para a inicialização aleatória, o critério utilizado para medir a qualidade da divisão de um nó (no caso, o índice de Gini) e o fato de que a distribuição é balanceada (class_weight). Observou-se que os parâmetros padrão do método deram resultados satisfatórios.

Após a configuração, o modelo é treinado com os conjuntos de treinamento, onde ele ajusta os parâmetros da árvore de decisão para minimizar a função de custo. Uma vez treinado, o modelo é utilizado para fazer previsões nos conjuntos de teste. As previsões são comparadas com os rótulos verdadeiros para calcular a matriz de confusão, que mostra o número de previsões corretas e incorretas para cada classe, e métricas de desempenho, como a precisão, o recall e o f1-score para cada classe.

Os resultados do treinamento do modelo de árvore de decisão revelam que a precisão geral do modelo no conjunto de teste foi de 89%. Além disso, o relatório de classificação fornece detalhes sobre a precisão, recall e f1-score para cada classe específica do jogo da velha. Por exemplo, a classe "Deu Velha" teve uma precisão de 25% e recall de 50%, indicando que o modelo não conseguiu prever mais corretamente do que por sorte esta classe, mas que ao menos conseguiu prever algum empate, ao contrários dos dois últimos modelos. Por outro lado, as classes "O Ganhou", "Tem Jogo" e "X Ganhou" apresentaram métricas mais favoráveis, com precisões de 82%, 91% e 89%, respectivamente. Esses resultados fornecem insights importantes sobre o desempenho do modelo de árvore de decisão na classificação do jogo da velha e sua capacidade de generalização para novos dados. Apesar de não parecer tão preciso quanto os outros, foi o único algoritmo - nos testes de mesa - a mais consistentemente conseguir analisar a situação de empate, embora o kNN tenha também tido seus acertos.

2.3.5 Conclusão

Durante o desenvolvimento desta etapa, enfrentamos um grande desafio devido às diferenças entre os algoritmos KNN, MLP e Árvore de Decisão. Além de fornecerem resultados distintos, todos os algoritmos apresentaram dificuldades em reconhecer quando o jogo ainda estava em andamento. Baseando-se

nos exemplos fornecidos, o aprendizado dos algoritmos, por vezes foi 'precipitado', resultando frequentemente em saídas que indicavam a vitória prematura de algum dos jogadores.

A implementação de algoritmos de aprendizado de máquina para classificar o jogo da velha oferece uma visão abrangente de como diferentes abordagens podem ser aplicadas para resolver o mesmo problema. Começando com o kNN, vemos que é uma abordagem simples e intuitiva, onde a escolha adequada do parâmetro k é crucial para o desempenho do modelo. Neste caso, o kNN apresentou uma acurácia média em relação aos outros modelos, de 93% com k=7, demonstrando algumas falhas na classificação do jogo da velha mas, no geral, um bom resultado.

Em seguida, o MLP oferece uma abordagem mais avançada, utilizando uma rede neural artificial com múltiplas camadas para realizar a classificação. Tendo alcançado a maior acurácia em comparação aos outros algoritmos (99%), o MLP apresentou mais flexibilidade e capacidade de lidar com problemas mais complexos como a avaliação dos tabuleiros. Isto, porém, não se provou no 'mundo real', sendo o algoritmo com mais falhas nos testes de mesa por meio do frontend.

Por fim, a árvore de decisão apresentou resultados promissores, com uma precisão geral de 89% no conjunto de teste. A interpretabilidade das decisões tomadas pela árvore de decisão pode ser uma vantagem significativa em muitos casos, permitindo uma compreensão mais clara das relações entre as características e os resultados. Apesar da baixa acurácia, parece ser o algoritmo, dentre os três, mais adequado para resolver o problema proposto.

Em conclusão, cada algoritmo apresenta vantagens e desvantagens, e a escolha do melhor modelo depende das características específicas do problema e das necessidades do projeto. Experimentar diferentes algoritmos e técnicas de aprendizado de máquina é essencial para encontrar a solução mais adequada para o problema em questão. Além disso, a avaliação cuidadosa do desempenho do modelo é fundamental para garantir que ele seja capaz de generalizar bem para novos dados e produzir resultados confiáveis na prática.

Inicialmente, optamos por desenvolver o front end utilizando tecnologias web, como CSS, HTML e JavaScript. No entanto, conforme avançamos na implementação, percebemos que o código resultante tornou-se excessivamente complexo e difícil de manter, com uma quantidade significativa de arquivos dispersos. Diante disso, decidimos buscar uma abordagem alternativa para resolver esse problema de forma mais eficiente.

Após análise e discussão, optamos por implementar o front end em Python, aproveitando as capacidades das bibliotecas PyGame e Tkinter. Essa escolha nos permitiria uma abordagem mais integrada e simplificada, facilitando a implementação e manutenção do sistema como um todo. Devido a transição para essas bibliotecas ter sido inicialmente desafiadora devido à falta de experiência prévia da equipe e problemas para configurar o ambiente de desenvolvimento e compreender a sintaxe e funcionalidades específicas de cada biblioteca, a equipe optou por focar mais na solução de IA e reaproveitar um front-end antigo de um dos membros do grupo, fazendo as atualizações necessárias.

2.4 Frontend

O front-end foi implementado em Angular e foram necessárias algumas modificações, como a adição de campos para exibir a saída de cada algoritmo e a chamada da IA para avaliar o tabuleiro. Para isso, utilizou-se a API fetch do JavaScript para fazer requisições a uma API criada em FastAPI, permitindo uma integração simples entre os códigos. Além disso, implementamos a opção de jogar contra o computador, que não é um modelo de IA, mas sim um algoritmo simples que faz um "chute" para marcar uma casa vazia. A interface também foi modificada para ficar visualmente mais agradável.

3 Conclusão

Durante a execução deste trabalho, enfrentamos diversos desafios, alguns mais simples para resolver, como a implementação da lógica para identificar vitórias nas diagonais para "X" ou "O", alguns mais trabalhosos como a obtenção de mais exemplos de tabuleiros. Nos deparamos também com nossa maior dificuldade: garantir que os resultados fossem precisos devido ao treinamento dos algoritmos, que inicialmente tomavam decisões precipitadas. Particularmente, o algoritmo MLP apresentou uma tendência a superestimar vitórias prematuras para "X" ou "O" em estados do jogo ainda não concluídos, um problema comum na aprendizagem de máquinas conhecido como superajuste. Esse desafio

foi exacerbado pela limitada diversidade em certos segmentos do conjunto de dados que inicialmente não representavam adequadamente todos os estados possíveis do jogo.

Apesar desses obstáculos, os avanços foram significativos. A solução de IA desenvolvida mostrou-se capaz de classificar o estado do jogo da velha e interagir eficazmente com os jogadores humanos. A integração do frontend proporcionou uma experiência interativa e visualmente agradável, destacando a capacidade da IA de responder em tempo real às ações dos jogadores.

Este projeto ressaltou a importância crucial da seleção apropriada de algoritmos, do ajuste meticuloso de parâmetros e da paciência durante o treinamento da IA para alcançar resultados satisfatórios em problemas complexos. A experiência reforçou a necessidade de testes e validações extensivas sob uma variedade de condições para garantir a robustez e a precisão dos modelos de IA, especialmente em aplicações interativas como jogos, onde a experiência do usuário é diretamente impactada pela performance do algoritmo.