Trabalho 1

Tic Tac Toe com ML

Inteligência Artificial

Professora Silvia Moraes

Gabriel Guinter Herter

Matheus Susin Timmers

Pedro Henrique De Ros

1. Introdução

Este trabalho tem como objetivo o desenvolvimento de uma solução para o jogo Tic Tac Toe, que contêm tanto a implementação de um front-end interativo em Python quando o desenvolvimento de uma inteligência Artificial (IA) capaz de julgar os resultados dos jogos. A interface permite que dois jogadores interajam com o tabuleiro em tempo real, realizando suas jogadas de forma alternada, enquanto a IA será responsável por determinar o resultado do jogo, verificando condições de vitória, empate ou continuidade.

Diante disso, o trabalho utilizado algoritmos de aprendizados máquina. O processo envolve um dataset contendo exemplos de estados finais do jogo da velha, que será tratado e dividido em conjuntos de treino, validação e teste. Os métodos utilizados para esta implementação neste relatório são k-Nearest Neighbors (k-NN), Multi-Layer Perceptron (MLP) e Árvores de Decisão.

Além disso, serão abordados conceito de pré-processamento de dados, métricas de avaliação e precisão, tendo em vista a possibilidade de análise do melhor desempenho na classificação dos estados dos jogos. O dataset foi fornecido e tratado para garantir o balanceamento e representatividade dos estados possíveis.

Ao final, a solução proposta permitiu a integração da IA com o front-end, o qual avaliou os resultados obtidos no decorrer da partida.

2. Problemas

No decorrer deste trabalho foi verificado um conjunto de desafios, os quais foram divididos em etapas para sua solução. Diante disso, para inicialização dele, é necessário ajustar o dataset fornecido, a fim de deixá-lo balanceado contendo dados normalizados e um conjunto de informações representantes para julgamento.

Outro problema encontrado durante o pré-processamento foi na otimização dos algoritmos de IA, pois, algoritmos como k-NN dependem de parâmetros (k), que determina o número de vizinhos mais próximo a serem considerados. Sendo assim, a utilização de métodos de validação cruzada (método Elbow), oferecendo melhor desempenho nas avaliações.

Contudo, a validação de dados exige atenção no quesito dos conjuntos de treino, pois um treinamento inadequado pode resultar num *overfitting*, o qual o modelo aprende com o treinamento. Assim, as técnicas de validação foram de extrema importância para esta seleção.

Ademais, a integração do front-end com a implementação da IA, gerou dificuldade no quesito de ligação de dados, pois inicialmente foi separado estas áreas de desenvolvimento, o que acarretou danos na vinculação futuras dos arquivos.

3. Soluções

A construção deste trabalho teve várias etapas de desenvolvimento, as quais desempenharam um papel fundamental no desempenho final do modelo. A seguir, será apresentado algumas das etapas seguidas para otimização do processo:

a) **Normalização do dataset:** Durante o processo de verificação, os dataset contidos na fonte original não eram o suficiente para uma aprendizagem mais profunda, em que devido a falta de diversidade ocorreram casos de overfitting. Para mitigar este problema, foi necessário aumentar o número de exemplos, adicionando mais exemplos de "empate" e "em jogo", de modo a garantir a regularidade dos dados e lidar com diferentes cenários. Além disso, utilizamos técnicas de normalização, transformando as variáveis categóricas (X, O, e b) em valores numéricos (-1, 1 e 0), facilitando o processamento pelos algoritmos de aprendizado de máquina.

b) **Treinamento:** Durante o treinamento, experimentamos com diferentes algoritmos de aprendizado de máquina, incluindo o k-Nearest Neighbors (k-NN), Multi-Layer Perceptron (MLP) e Árvores de Decisão. Para o k-NN, o método do cotovelo (elbow method) para localizar o melhor valor de k, que ajudou a identificar o ponto onde a taxa de erro de classificação deixava de melhorar significativamente com o aumento de k. Esse refinamento permitiu ajustes no valor de k de forma eficiente, garantindo que o modelo k-NN não sofresse de underfitting (se k for muito pequeno) ou overfitting (se k for muito grande).

c) **Verificação de desempenho:** Para validar o desempenho dos diferentes algoritmos, foi empregado as métricas de classificação, sendo elas acurácia, precisão, recall e F1-score. A verificação de desempenho foi realizada tanto nos conjuntos de validação quanto nos de teste, para garantir que o modelo fosse capaz de generalizar bem e não apenas memorizar os dados de treino.

d) **Determinação de Métodos:** Ao final da etapa de experimentação e testes, concluiu-se que o Multi-Layer Perceptron (MLP) apresentou os melhores resultados, com uma acurácia de 95%, superando os demais algoritmos testados. O k-Nearest Neighbors (k-NN), apesar de entregar resultados consistentes, alcançou uma acurácia inferior de 84%, mesmo com a otimização do valor de k. A rede MLP, embora tenha demandado ajustes na topologia para melhorar seu desempenho, demonstrou maior capacidade de generalização. Por outro lado, as árvores de decisão, embora eficientes na interpretação dos resultados, mostraram-se mais suscetíveis ao overfitting em um dataset pequeno. A escolha final do MLP como o melhor algoritmo foi fundamentada na análise das métricas de desempenho, que apontaram sua robustez e capacidade de tomar decisões corretas em uma ampla variedade de estados do jogo.

Essas etapas permitiram a construção de uma IA eficaz para julgar o jogo da velha, pronta para integração com o front-end interativo, oferecendo uma solução completa para o problema proposto.

4. Resultados

Neste capítulo vamos abordas os resultados encontrados ou sua implementação.

a) **Front-end:** O tabuleiro foi criado de forma simples e prática, em que existe dois jogadores X e O, os quais preenchem campos de linha e coluno, as quais são atualizadas no terminal.

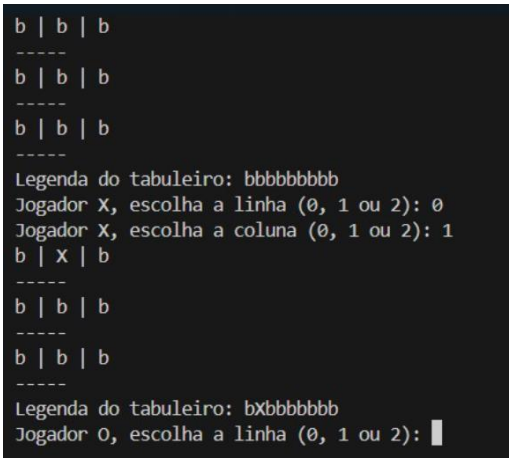

Imagem 1 – Tabuleiro

b) **Expansão do dataset:** Para evitar overfitting (como mencionado anteriormente), foi necessário a criação de mais dados.

Imagem 2 – Dados criados (exemplo)

c) **Verificação dos métodos utilizados:** Para analisarmos os resultados obtidos, bem como sua relevância para os meios de avaliação, foi utilizados campos de validação de dados, com base na escolha do 'k', os quais apresentam valores conforme imagem 3.

```
Acuracia:  0.8875
0.8875
              precision    recall  f1-score   support

          -2       0.87      1.00      0.93        20
           2       0.90      0.95      0.93        20
           3       0.92      0.60      0.73        20
           4       0.87      1.00      0.93        20

    accuracy                           0.89        80
   macro avg       0.89      0.89      0.88        80
weighted avg       0.89      0.89      0.88        80
```

Imagem 3 – validação de dados

```
Acuracia:  0.9
0.9
              precision    recall  f1-score   support

          -2       0.90      1.00      0.94        60
           2       0.89      0.98      0.94        60
           3       0.86      0.30      0.44        20
           4       0.95      0.95      0.95        20

    accuracy                           0.90       160
   macro avg       0.90      0.81      0.82       160
weighted avg       0.90      0.90      0.88       160
```

Imagem 4 – validação de dados com o passar do tempo

d) Utilização do método "Elbow" para determinação do 'k'

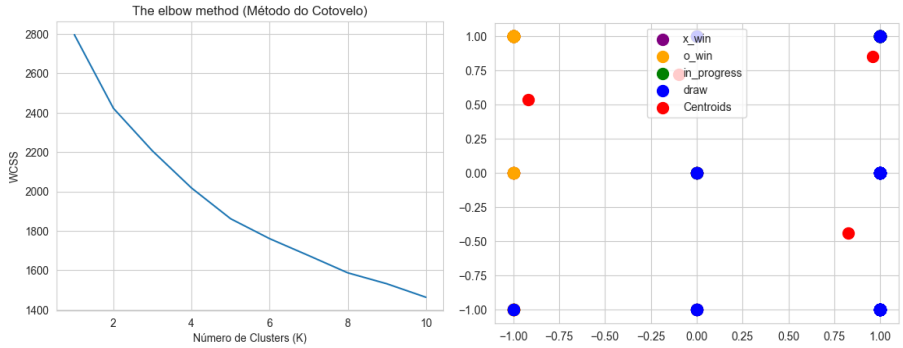

Imagem 5 – Método Elbow

e) **Implementação da arvore de decisão:** Visivelmente analisando a arvore é possível verificar que a quantidade de caos de teste dificulta a praticidade de julgamento.

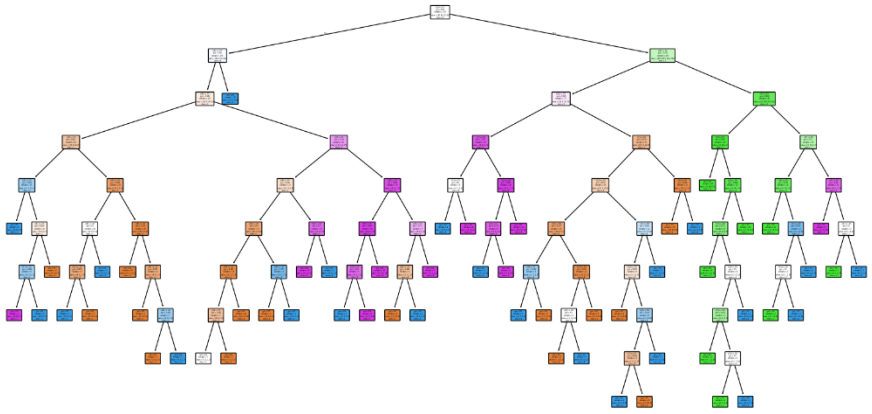

Imagem 6 – Evidência gerada pela arvore de decisão

4.1 Soluções

Legenda para entendimento:

-2 0 -> o_win

2 -> x_win

3 -> in_progress

4 -> draw

a) K-NN

Foi utilizado o algoritmo k-Nearest Neighbors (k-NN) com o parâmetro k=1, após a realização de testes. Com esse valor, obteve-se uma acurácia de 84,81%. Os resultados mostraram boa precisão e recall nas classes com maior representatividade, embora algumas classes com suporte menor tenham apresentado desafios, sugerindo a necessidade de futuros ajustes no balanceamento dos dados.

<table>
<tr><td colspan="2">Acurácia geral: 84,81%</td></tr>
<tr><td colspan="2">Resultados por classe</td></tr>
<tr><td>

Classe -2:

Precisão: 0.83
Recall: 1.00
F1-Score: 0.91
Suporte: 63

Classe 2:

Precisão: 0.84
Recall: 0.99
F1-Score: 0.91
Suporte: 125

</td><td>

Classe 3:

Precisão: 0.91
Recall: 0.53
F1-Score: 0.67
Suporte: 79

Classe 4:

Precisão: 0.00
Recall: 0.00
F1-Score: 0.00
Suporte: 3

</td></tr>
</table>

b) Árvore de decisão:

A árvore de decisão foi utilizada como um dos algoritmos para a classificação dos estados do jogo da velha, com a aplicação de pesos para as diferentes classes, a fim de lidar com o desbalanceamento do dataset. Essa abordagem permitiu que o modelo ajustasse suas decisões de acordo com a importância de cada classe, influenciando diretamente na precisão final. Embora o modelo tenha apresentado desempenho razoável, os resultados indicaram uma tendência ao overfitting, especialmente em classes com menor representatividade no conjunto de dados.

- **Acurácia: 0.812807881773399**
- **Precisão: 0.8098534258281886**
- **Recall: 0.812807881773399**
- **F-measure: 0.810798980481085**

C) MLP

Para identificar o melhor modelo de MLP, realizamos uma série de testes utilizando diferentes solvers: sgd, adam e lbfgs. Os resultados mais satisfatórios foram obtidos com o lbfgs, que atingiu uma acurácia de 0,95. Além disso, este solver apresentou uma convergência mais estável e rápida, tornando-se a escolha ideal para o nosso modelo, considerando tanto o desempenho quanto a eficiência computacional.

- Solver: lbfgs
  - Acurácia: 0.95
  - Precisão: 0.96
  - Recall: 0.95
  - F-measure: 0.9496527777777779
- Solver: adam
  - Acurácia: 0.925
  - Precisão: 0.9520833333333332
  - Recall: 0.95
  - F-measure: 0.9497101449275362
- Solver: sgd
  - Acurácia: 0.85
  - Precisão: 0.9045673076923076
  - Recall: 0.925
  - F-measure: 0.9123774509803921

5. Conclusão

Neste trabalho, foi desenvolvido um jogo da velha integrado com uma inteligência artificial capaz de julgar o estado do jogo. A interface foi implementada pelo terminal de comandos do python, permitindo que dois jogadores humanos joguem alternadamente, enquanto a IA avalio o estado do jogo, alegando se a vencedor ou empate.

O desenvolvimento da IA decorreu de vários desafios como limpeza do dataset, validação de dados, algoritmo de aprendizado, a fim de tornar os dados representativo e balanceado. Sendo essa, uma etapa crucial para garantir que o modelo fosse capaz de aprender sem overfitting e underfitting. O uso de validação cruzada foi de extrema importância neste cenário.

Após a compilação e aprendizado da IA, o algoritmo de validação utilizando métricas de desempenho como acurácia, precisão, recall e F1-score administrou um papel de extrema importância no levantamento de dados para verificação de eficiência dos resultados obtidos, proporcionando uma solução funcional correta e eficaz.

Concluindo-se, assim, que este trabalhou proporcionando um ambiente de teste e aplicação dos conhecimentos vistos em aula, os quais associam a vinculação (conteúdo x prática x aprendizado). Em que, ao final foi visualizado a eficácia de um dataset normalizado, as aplicações corretas dos métodos e a integração com o front-end adequado.