Trabalho 1 - Inteligência Artificial

Nome: Rhaduan Khalek Pacheco, Giovani da Silva Cancherini

Curso: Ciência da Computação

Instituição: PUCRS

Resumo

Este relatório propõe uma solução para o jogo da velha, utilizando o conjunto de dados Tic-Tac-Toe como base. Foram empregados três algoritmos de inteligência artificial (K-NN, MLP e Árvore de Decisão) para uma análise comparativa. O documento inclui uma análise do dataset para sua utilização, explicações sobre o funcionamento de cada algoritmo e a apresentação de uma interface frontend que permite a interação e a realização de simulações.

---

1. Descrição do Problema

Este trabalho envolve a criação de um sistema de IA para o Jogo da Velha em um tabuleiro 3x3. O sistema foi desenvolvido para analisar o status do jogo a cada movimento, indicando se alguém ganhou, se houve empate ou se ainda há jogo. O principal objetivo foi lidar com o dataset disponibilizado para treinamento e predições utilizando diferentes algoritmos de Machine Learning: K-NN, MLP e Árvore de Decisão.

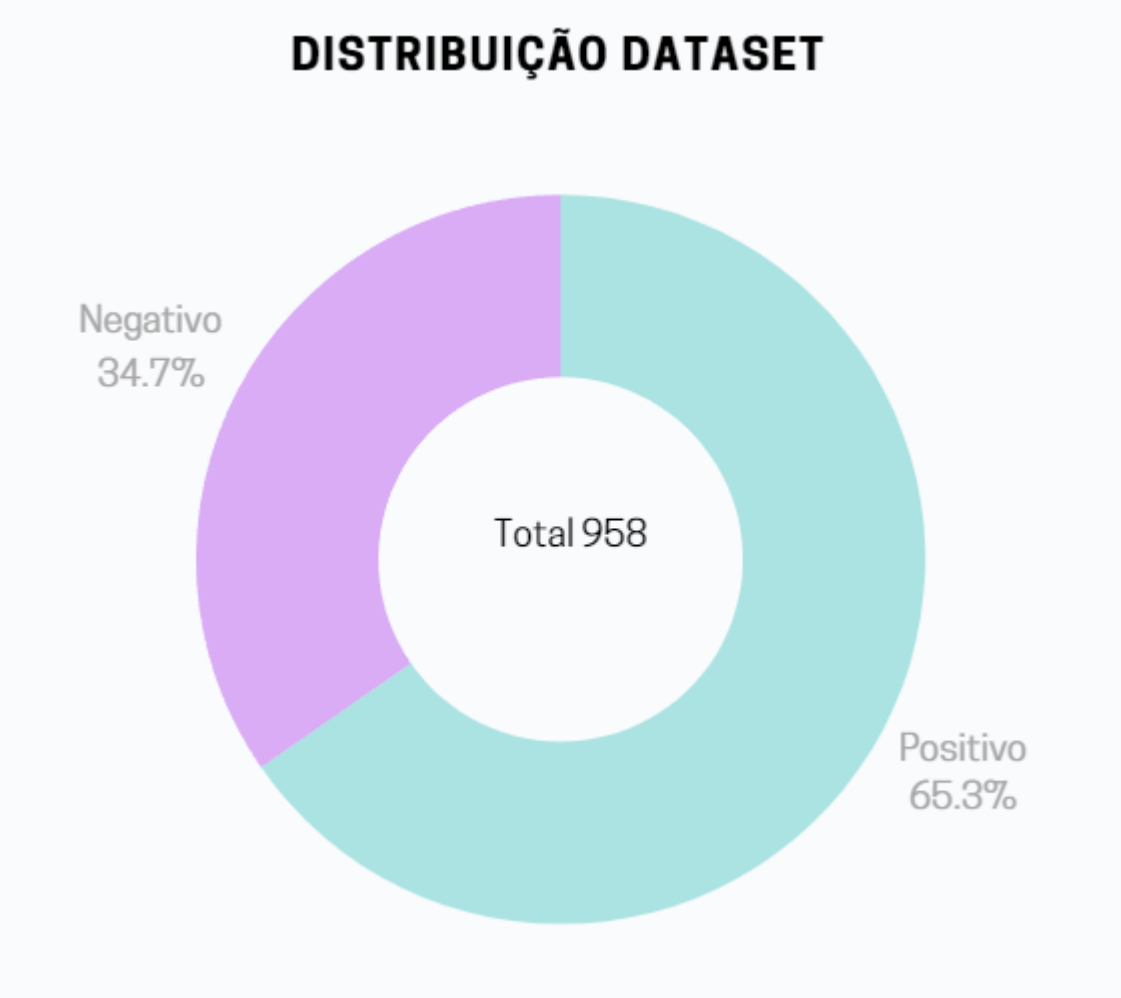
O problema foi dividido em quatro etapas:

1. **Pré-processamento dos dados:** Preparação dos dados para garantir que estejam prontos para uso nos algoritmos de Machine Learning.
2. **Execução dos algoritmos selecionados:** Implementação e execução do dataset nos algoritmos mencionados.
3. **Elaboração de um Front End:** Criação de uma interface básica para permitir ao usuário interagir com o sistema.
4. **Conclusão:** Análise dos resultados obtidos com cada algoritmo, dificuldades encontradas e aprendizado obtido com o trabalho.

---

2. Sobre o Dataset e seu Pré-processamento

O dataset inicial do Jogo da Velha continha em si 10 colunas e 958 linhas, representando uma matriz contendo 626 combinações de casos positivos e 332 combinações de casos negativos.



Ajustes realizados:

1. **Identificação de casos de vitória e empate:** Criamos mais casos de empate, substituindo 'x' por 'o' e vice-versa, para balancear o dataset.
2. **Criação de casos de jogo em andamento:** Substituímos de forma aleatória os 'x' e 'o' por 'b' em tabuleiros válidos, sem ganhadores.
3. **Transformação dos tipos de dados:** Mapeamos as letras 'x', 'o' e 'b' para 1, -1, 0 respectivamente, e os resultados 'empate', 'ganhou' e 'jogando' para 0, 1, -1.

---

3. Implementação dos Algoritmos

3.1 K-Nearest Neighbors (K-NN)

O algoritmo K-Nearest Neighbors (K-NN) é um método simples e eficaz para classificação. Ele funciona encontrando os 'K' exemplos de treinamento mais próximos no espaço de características e tomando uma decisão com base na maioria dos vizinhos.

Implementação:

Para o Jogo da Velha, usamos o K-NN com K=2, que foi selecionado após testar diferentes valores de 'K' e verificar a performance nos dados de validação.

```
python
Copiar código
from sklearn.neighbors import KNeighborsClassifier

# Inicializar o modelo K-NN com K=2
knn_model = KNeighborsClassifier(n_neighbors=2)
knn_model.fit(X_train, y_train)
```

3.2 Multi-Layer Perceptron (MLP)

O MLP é uma rede neural feedforward com uma ou mais camadas ocultas. Ele é poderoso para capturar relações não lineares nos dados.

Implementação:

Usamos um MLP com uma camada oculta de 50 neurônios e treinamento por 1000 iterações.

```
python
Copiar código
from sklearn.neural_network import MLPClassifier

# Inicializar o modelo MLP
mlp_model = MLPClassifier(hidden_layer_sizes=(50,), max_iter=1000, random_state=42)
mlp_model.fit(X_train, y_train)
```

3.3 Árvore de Decisão

A árvore de decisão é um modelo de aprendizado supervisionado que divide o espaço de características em regiões com base nas características de entrada.

Implementação:

Usamos a Árvore de Decisão com os parâmetros padrão do scikit-learn.

```
python
Copiar código
from sklearn.tree import DecisionTreeClassifier
# Inicializar o modelo de Árvore de Decisão
tree_model = DecisionTreeClassifier(random_state=42)
tree_model.fit(X_train, y_train)
```

3.4 Avaliação dos Modelos

Cada modelo foi avaliado usando a acurácia no conjunto de teste.

```
python
Copiar código
from sklearn.metrics import accuracy_score

# Avaliação do modelo K-NN
y_pred_knn = knn_model.predict(X_test)
accuracy_knn = accuracy_score(y_test, y_pred_knn)

# Avaliação do modelo MLP
y_pred_mlp = mlp_model.predict(X_test)
accuracy_mlp = accuracy_score(y_test, y_pred_mlp)

# Avaliação do modelo Árvore de Decisão
y_pred_tree = tree_model.predict(X_test)
accuracy_tree = accuracy_score(y_test, y_pred_tree)
```

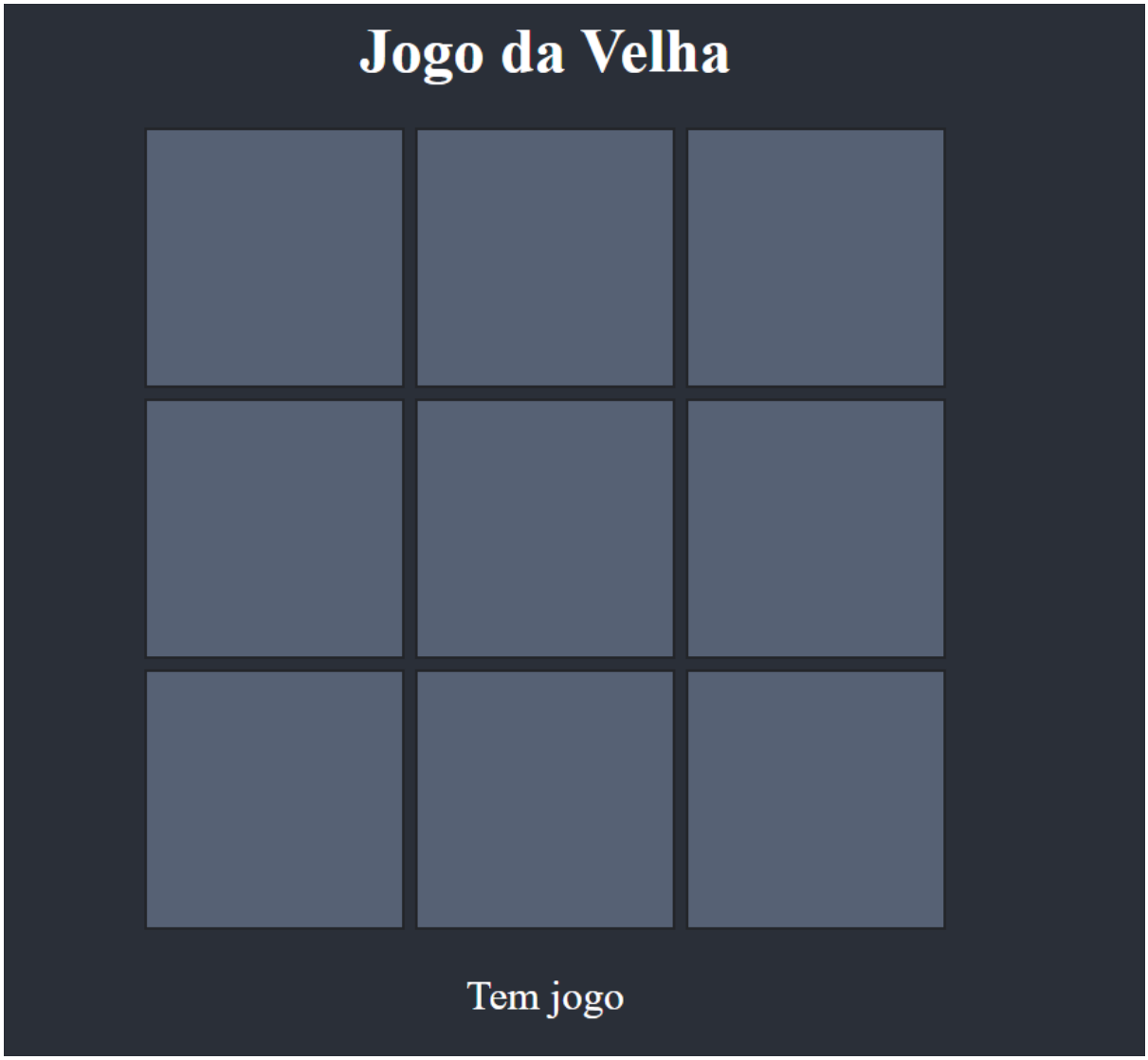
Os resultados de acurácia para cada modelo foram:

- **K-NN**: 0.99
- **MLP**: 0.98
- **Árvore de Decisão**: 0.94

---

4. Interface Frontend

Para permitir a interação do usuário com o sistema, foi desenvolvida uma interface frontend usando Flask para criar um servidor web simples. Essa interface permite que o usuário insira um estado do tabuleiro e obtenha a previsão do estado do jogo.



4.1 Configuração do Servidor Flask

Utilizamos Flask para criar um servidor web que recebe requisições HTTP POST com o estado atual do tabuleiro do jogo da velha. Flask é um micro framework para Python que facilita a criação de aplicações web.

4.2 Carregamento do Modelo Treinado

O modelo de machine learning treinado foi carregado no servidor Flask. Utilizamos o modelo que apresentou a melhor acurácia, que neste caso foi o K-NN. No entanto, também é possível utilizar outros modelos treinados, como MLP e Árvore de Decisão. Para alterar o modelo utilizado pelo servidor, basta carregar o arquivo correspondente ao modelo desejado.

Exemplo:

```
# Caminho do modelo K-NN
model_path = "best_model_k-NN.pkl"

# Alterar para o caminho do modelo MLP
model_path = "best_model_MLP.pkl"

# Alterar para o caminho do modelo Árvore de Decisão
model_path = "best_model_DecisionTree.pkl"
```

4.3 Avaliação do Tabuleiro

Quando o servidor recebe uma requisição com o estado do tabuleiro, ele transforma o tabuleiro em um formato que pode ser utilizado pelo modelo (convertendo 'x', 'o' e espaços vazios para valores numéricos). O modelo então faz a predição do estado do jogo.

4.4 Determinação do Status do Jogo

O resultado da predição é analisado para determinar se o jogo foi ganho por 'x', por 'o', se houve empate, ou se o jogo ainda está em andamento. Essa lógica inclui verificar se há uma linha, coluna ou diagonal com todos os mesmos símbolos, ou se todas as posições do tabuleiro estão preenchidas sem um vencedor, caracterizando um empate.

4.5 Rota de Avaliação

O servidor Flask possui uma rota /evaluate que aceita requisições POST. A requisição deve conter o estado do tabuleiro, e o servidor responde com o status do jogo.

4.6 Dificuldades Encontradas

Durante o desenvolvimento deste projeto, diversas dificuldades foram encontradas e superadas, o que proporcionou um aprendizado significativo. As principais dificuldades foram:

1. Pré-processamento de Dados

O pré-processamento de dados foi um dos maiores desafios iniciais. O dataset original continha valores categóricos que precisavam ser transformados em valores numéricos para serem utilizados nos algoritmos de Machine Learning. Foi necessário substituir 'x', 'o', e 'b' por 1, -1, e 0, respectivamente, e 'positive', 'negative', e 'empate' por 1, -1, e 0, respectivamente. Além disso, foi importante garantir que todas as transformações fossem aplicadas corretamente, evitando a perda de informação.

2. Balanceamento do Dataset

O dataset original era desbalanceado, com uma maior quantidade de exemplos positivos (vitórias) em comparação com exemplos negativos (derrotas) e empates. Para resolver isso, criamos manualmente mais casos de empate e ajustamos casos de jogo em andamento. Isso foi essencial para evitar que os modelos ficassem enviesados e previssem sempre o resultado majoritário.

3. Integração com o Frontend

Integrar o backend de Machine Learning com o frontend utilizando Flask foi desafiador. Garantir que a comunicação entre a interface do usuário e o modelo de Machine Learning fosse eficiente e sem erros envolveu lidar com a serialização de dados, manipulação de JSON e gestão de estados de erro.

4. Validação e Testes

Validar o funcionamento correto do sistema foi fundamental. Criar testes abrangentes para todos os possíveis estados do tabuleiro do jogo da velha (vitória, derrota, empate e jogo em andamento) ajudou a garantir que o modelo de IA estava tomando decisões corretas. No entanto, isso também revelou alguns bugs e inconsistências que precisaram ser corrigidos. Ajuste dos hiper parâmetros dos modelos para melhorar a acurácia também foi um fator definitivo para aumentar a precisão.

5. Conclusão

Neste trabalho, desenvolvemos um sistema de IA para o Jogo da Velha utilizando três algoritmos de machine learning: K-NN, MLP e Árvore de Decisão. Após o pré-processamento dos dados e a criação de casos adicionais para balanceamento, treinamos e avaliamos os modelos. Com os hiper parâmetros escolhidos, foi notado que o k-NN acabou por desempenhar melhor. Implementamos também uma interface frontend utilizando Flask para permitir a interação com o sistema.

(O projeto pode ser iniciado com npm run dev)