Decision Helper – Aplicação para recomendação de filmes usando Machine Learning

Amanda Coca dos Santos Ramos, Carlos Magnus Carlson Filho, Luciene Cavalcanti Rodrigues
e-mail: coca.amanda1@gmail.com
Faculdade de Tecnologia de São José do Rio Preto

Resumo: A presente pesquisa aborda o fenômeno da sobrecarga de escolha (choice overload) que ocorre nas plataformas de streaming devido à grande quantidade de opções disponíveis, o que pode levar à indecisão e insatisfação do usuário. O estudo busca examinar e comparar diferentes métodos de recomendação, incluindo filtragem colaborativa, métodos baseados em conteúdo e técnicas híbridas. O sistema de recomendação foi desenvolvido utilizando a linguagem Python e a biblioteca PyTorch, aproveitando sua flexibilidade e eficiência para a construção de modelos personalizados. Os testes realizados com o sistema indicaram uma redução consistente na métrica de erro médio quadrático (MSE), demonstrando a eficácia do modelo em gerar recomendações precisas. Além disso, o sistema se mostrou eficiente na diminuição do tempo gasto pelos usuários na escolha de conteúdos, promovendo uma experiência de visualização mais satisfatória e menos estressante.

Palavras-chave: Sobrecarga de escolhas; métodos de recomendação; filtragem colaborativa, técnicas híbridas, experiência do usuário.

Abstract: *This research addresses the phenomenon of choice overload occurring on streaming platforms due to the extensive range of available options, which can lead to user indecision and dissatisfaction. The study aims to examine and compare various recommendation methods, including collaborative filtering, content-based methods, and hybrid techniques. The recommendation system was developed using Python and the PyTorch library, leveraging its flexibility and efficiency for building customized models. Tests conducted with the system indicated a consistent reduction in the mean squared error (MSE) metric, demonstrating the model's effectiveness in generating accurate recommendations. Additionally, the system proved efficient in reducing the time users spend choosing content, fostering a more satisfying and less stressful viewing experience.*

Keywords: *Choice overload, Recommendation methods, Collaborative Filtering, Hybrid techniques, user experience.*

1. INTRODUÇÃO

Atualmente, a vasta gama de opções disponíveis para momentos de lazer pode causar desconforto e confusão mental. Frequentemente, ao chegarmos em casa após um dia repleto de atividades, procuramos aliviar a sobrecarga mental por meio de diversas atividades, como cozinhar, praticar exercícios físicos ou assistir a conteúdos em plataformas de streaming. Contudo, é comum sentir-se perdido diante da grande quantidade de opções oferecidas por essas plataformas. As categorias variam entre os mais assistidos, recomendados para você e novos lançamentos, o que pode levar ao fenômeno conhecido como ‘choice overload’ (sobrecarga de escolha).

A pesquisa de Chernev, Bockenholt e Goodman (2015) demonstra que, apesar de um maior número de opções teoricamente oferecer mais liberdade e adequação às preferências individuais, na prática, essa abundância pode resultar em indecisão e insatisfação. A complexidade das opções, a dificuldade na tomada de decisão, a incerteza sobre preferências e objetivos intensificam a sobrecarga decisória. O conceito de "The Burden of Choice", abordado por Cohn J., exemplifica como a ampla gama de opções pode transformar-se em um fardo, gerando ansiedade e dificultando a tomada de decisões satisfatórias. O artigo da The Week corrobora essa perspectiva ao destacar que

a grande quantidade de opções nas plataformas de streaming pode aumentar a ansiedade dos usuários e complicar a escolha, tornando essencial a implementação de um sistema de recomendação eficiente.

A análise dos diferentes métodos de recomendação é, portanto, crucial para o desenvolvimento de um sistema capaz de mitigar os efeitos da '*choice overload'*. É ideal que os sistemas de recomendação adotem os melhores métodos de filtragem de conteúdo com base no interesse do usuário, que é o objetivo principal deste estudo.

Este trabalho utiliza Python como a principal linguagem para desenvolvimento do sistema de recomendação. A linguagem é amplamente reconhecida por suas bibliotecas poderosas, como PyTorch, TensorFlow, entre outras, que oferecem suporte robusto para aprendizado de máquina e modelagem de sistemas de recomendações. Sua flexibilidade e recursos permitem a implementação de técnicas como filtragem colaborativa, métodos baseados em conteúdo e técnicas híbridas, adaptando-se às necessidades específicas do sistema.

2. FUNDAMENTAÇÃO TEÓRICA

A sobrecarga de escolha, conhecida como *choice overload*, tem sido um dos desafios mais relevantes no ambiente digital atual, especialmente em plataformas de streaming, onde os usuários são expostos a uma vasta gama de opções de entretenimento. O fenômeno, descrito por Chernev, Bockenholt e Goodman (2015), ocorre quando a abundância de escolhas, em vez de promover maior liberdade, resulta em indecisão, ansiedade e insatisfação. Essa questão é exacerbada pela complexidade das preferências pessoais e pela incerteza em relação ao que realmente se deseja assistir, conforme também abordado por Cohn (2019). Dessa forma, a implementação de sistemas de recomendação que minimizem essa sobrecarga e orientem os usuários na escolha de conteúdos torna-se essencial.

Os sistemas de recomendação são ferramentas que filtram automaticamente informações relevantes, baseadas no perfil do usuário, ajudando a resolver o problema de sobrecarga de informações (ISINKAYE; FOLAJIMI; OJOKOH, 2015). Entre as abordagens mais utilizadas estão a filtragem baseada em conteúdo (content-based *filtering*), a filtragem colaborativa (collaborative *filtering*), a filtragem demográfica e as técnicas híbridas. Cada uma dessas metodologias apresenta vantagens e desafios específicos, com suas aplicações variando conforme as necessidades e contextos dos sistemas implementados.

2.1 Filtragem Baseada em Conteúdo

A filtragem baseada em conteúdo (CB) utiliza descrições de itens e o histórico de interações do usuário com o sistema para prever quais outros itens podem ser de seu interesse. Pazzani e Billsus (2007) explicam que os sistemas CB criam perfis de usuários a partir de dados explícitos, como avaliações, ou implícitos, como o comportamento de navegação. A partir desses dados, algoritmos de classificação, como árvores de decisão e algoritmos de vizinhança, são empregados para prever os itens mais relevantes para o usuário. Esse método foca exclusivamente nas características dos itens e nas preferências já expressas pelo usuário, levando a recomendações personalizadas.

Contudo, a técnica tem algumas limitações. A principal delas é o fenômeno conhecido como over-specialization, que ocorre quando o sistema recomenda apenas itens muito similares aos já consumidos pelo usuário. Isso pode reduzir a diversidade nas sugestões, limitando a descoberta de novos tipos de conteúdo. Além disso, há dificuldades em sugerir itens fora do escopo das interações anteriores do usuário, o que dificulta a capacidade do sistema de expandir o leque de recomendações (MEETEREN; SOMEREN, 2001).

2.2 Filtragem Colaborativa

A filtragem colaborativa (CF) foca nas interações entre usuários e utiliza as avaliações e preferências de outros indivíduos com perfis semelhantes para gerar recomendações. Nessa técnica, a suposição é que se dois usuários compartilham preferências em alguns itens, provavelmente terão gostos semelhantes em outros itens. Essa abordagem pode ser dividida em duas categorias principais: CF baseada em usuário e CF baseada em item. No primeiro caso, são encontradas similaridades entre os perfis de usuários; no segundo, a similaridade é medida entre os itens consumidos (ISINKAYE; FOLAJIMI; OJOKOH, 2015).

A filtragem colaborativa se destaca por promover recomendações mais diversificadas e não depender tanto das características dos itens, mas, sim, dos padrões de consumo da comunidade de usuários. Isso é particularmente eficaz em plataformas como a Netflix, que aproveitam essa técnica para aumentar a diversidade e oferecer sugestões baseadas em tendências coletivas (GOMEZ-URIBE; HUNT, 2015).

Porém, a CF enfrenta desafios relacionados à escalabilidade e à necessidade de grandes quantidades de dados para funcionar de maneira eficaz. Além disso, há o problema do cold start, que ocorre quando não há dados suficientes sobre um novo usuário ou item, dificultando a geração de recomendações (THORAT et al., 2015).

2.3 Filtragem Híbrida

Os sistemas de recomendação híbridos combinam múltiplas abordagens de filtragem para compensar as limitações de cada uma e, ao mesmo tempo, aproveitar suas vantagens. A filtragem híbrida é especialmente eficaz para melhorar a precisão, a diversidade e a personalização das recomendações. Ifada et al. (2019) apontam que esses sistemas são particularmente úteis em situações de cold start, quando há poucos dados sobre as preferências do usuário, além de oferecerem bom desempenho em condições normais.

Os sistemas de recomendação híbridos combinam diversas abordagens para potencializar suas vantagens e minimizar as limitações, entre as mais comuns estão Combinação Ponderada, que nada mais é do que a soma de resultados de forma estática; Switching (ou alternância de métodos), onde uma abordagem é escolhida com base no contexto de usuário; Integração de múltiplas técnicas, que une várias listas de recomendação em uma única.

A vantagem do sistema híbrido é que ele consegue resolver alguns dos maiores desafios dos métodos isolados. Por exemplo, ao integrar a CF com a CB, o sistema pode recomendar itens com base nas preferências de usuários semelhantes, enquanto utiliza as características dos itens para refinar ainda mais essas sugestões. Isso também ajuda a mitigar o problema do cold start na CF, já que o sistema baseado em conteúdo pode

começar a fazer recomendações mesmo sem ter um grande histórico de interações (GOMEZ-URIBE; HUNT, 2015).

Além disso, sistemas híbridos podem oferecer uma experiência mais diversificada e envolvente, aumentando a satisfação do usuário. O estudo de Knijnenburg et al. (2010) ressalta que, ao integrar diferentes tipos de filtragem, os sistemas híbridos não apenas melhoram a precisão das recomendações, mas também oferecem uma interação mais intuitiva e eficiente, o que resulta em maior aceitação e engajamento dos usuários. É possível observar que em plataformas como Netflix, normalmente são utilizadas técnicas híbridas, buscando maximizar a precisão (THORAT et al., 2015).

Por fim, ao analisar o sistema de recomendação da Netflix, Gomez-Uribe e Hunt (2015) demonstram como a combinação de algoritmos avançados e a personalização detalhada são fundamentais para manter o engajamento dos usuários. A Netflix utiliza um modelo híbrido que combina vários métodos de filtragem, resultando em uma interface intuitiva que oferece sugestões relevantes e diversificadas, superando as limitações de cada abordagem individual.

2.4 Pytorch

O PyTorch é uma biblioteca de aprendizado de máquina baseada em Python amplamente utilizada para o desenvolvimento de modelos de aprendizado profundo, o que é particularmente útil em sistemas de recomendação. Lançado em 2016, o PyTorch destaca-se por seu grafo computacional dinâmico, que permite maior flexibilidade e facilita a experimentação durante o desenvolvimento de modelos, especialmente em projetos que exigem alta personalização e ajustes em tempo real. Essa característica é benéfica para algoritmos de filtragem colaborativa, pois permite capturar relações complexas entre usuários e itens, melhorando a precisão das recomendações.

No contexto de recomendação, o PyTorch é utilizado para treinar redes neurais profundas que podem explorar dados de interação entre usuários e itens, como avaliações e preferências. No estudo de Omar, Frikha e Jumaa (2024), foram desenvolvidos dois modelos de recomendação utilizando PyTorch: um que usa apenas dados estruturados (usuário, item e avaliação) e outro que incorpora texto de avaliações dos usuários. O uso de texto mostrou uma melhora significativa na precisão das recomendações, destacando como o PyTorch permite integrar o processamento de linguagem natural (NLP) em sistemas de recomendação para extrair informações contextuais e enriquecer o entendimento das preferências dos usuários.

A capacidade do PyTorch de lidar com grandes volumes de dados e de utilizar otimizações com GPUs torna-o uma escolha apropriada para sistemas de recomendação escaláveis, como aqueles presentes em plataformas de streaming. Além disso, sua integração com técnicas de filtragem colaborativa e NLP permite que o sistema supere desafios comuns, como o problema de cold start e a quantidade avulsa de dados, melhorando a qualidade e a diversidade das recomendações.

Assim, observa-se que, para o desenvolvimento de um sistema de recomendação eficaz, é essencial considerar uma série de critérios, incluindo as preferências do usuário, a experiência interativa e as ferramentas utilizadas no processamento desses dados. Tecnologias como o PyTorch, são fundamentais para personalizar as recomendações e aprimorar a qualidade do sistema. Além disso, o uso de interfaces amigáveis e explicações claras sobre o processo de recomendação contribui para reduzir a ansiedade e aprimorar a experiência do usuário, conforme

sugerido por Isinkaye, Folajimi e Ojokoh (2015). A capacidade de personalização, combinada com o aprendizado contínuo a partir das interações dos usuários, é fortalecida por ferramentas robustas como o PyTorch, aumentando, assim, a precisão e a relevância das recomendações.

3. DESENVOLVIMENTO

No tópico de desenvolvimento será descrito todas as etapas que foram necessárias para o desenvolvimento deste trabalho, desde o pré-processamento de dados até a apresentação do resultado de buscas por gênero, simulando a interação usuário-sistema. O dataset escolhido foi retirado da comunidade Kagle, que é uma comunidade onde estão disponíveis diversos datasets para estudo ou desenvolvimento de trabalhos.

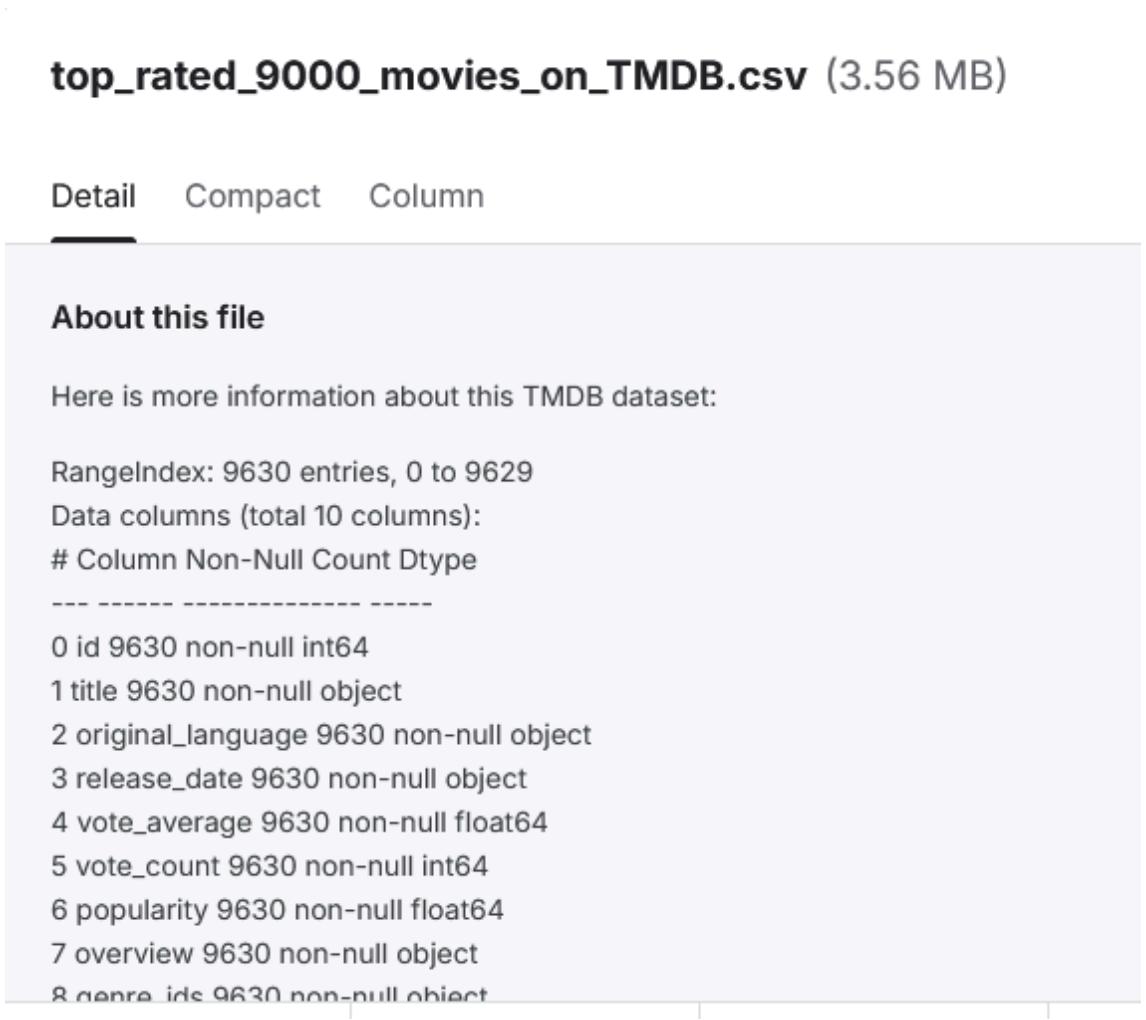

Figura 1: Informações do dataset escolhido

3.1 Carregamento e pré-processamento de dados

Para construir o sistema de recomendação, foi necessário a importação de diversas bibliotecas, como numpy e pandas para manipulação dos dados, torch para

construção de modelos de modelos de aprendizado, e scikit-learn para dividir o dataset entre conjunto de treino e teste e para pré-processamento de rótulos.

```
import pandas as pd
import numpy as np
import torch
import torch.nn as nn
import torch.optim as optim
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
```

[1] ✓ 6.7s

Figura 2: Importação das bibliotecas

A leitura dos dados foi realizada utilizando a função ‘read_csv’ da biblioteca pandas.

```
df = pd.read_csv('C:/Users/amand/Downloads/movieDataSet.csv')
print(df.head())
print(df.columns)
```

```
]
      id                    title original_language release_date  vote_average  \
0  278  The Shawshank Redemption                en   1994-09-23         8.706
1  238             The Godfather                en   1972-03-14         8.690
2  240     The Godfather Part II                en   1974-12-20         8.575
3  424           Schindler's List               en   1993-12-15         8.565
4  389              12 Angry Men                en   1957-04-10         8.546

   vote_count  popularity                                           overview  \
0       26840     150.307  Imprisoned in the 1940s for the double murder ...
1       20373     122.973  Spanning the years 1945 to 1955, a chronicle o...
2       12291      94.204  In the continuing saga of the Corleone crime f...
3       15695      74.615  The true story of how businessman Oskar Schind...
4        8522      54.678  The defense and the prosecution have rested an...

          genre_ids                        Genres
0         [18, 80]           ['Drama', 'Crime']
1         [18, 80]           ['Drama', 'Crime']
2         [18, 80]           ['Drama', 'Crime']
3  [18, 36, 10752]  ['Drama', 'History', 'War']
4             [18]                    ['Drama']
Index(['id', 'title', 'original_language', 'release_date', 'vote_average',
       'vote_count', 'popularity', 'overview', 'genre_ids', 'Genres'],
      dtype='object')
```

Figura 3: Leitura dos dados presentes no dataset

Após a leitura ter sido feita, foi possível visualizar que o dataset contém informações relações relevantes como títulos, gêneros, data de lançamento, média de votos, e sinopse, entre outros atributos que servirão como base para o modelo.

3.2 Preparação e criação de colunas adicionais

Para preparar os dados para o modelo de recomendação, foram criadas colunas adicionais que identificam o usuário (userId) e o filme (movieId). A coluna rating foi definida com base na média de votos dos filmes, representando a avaliação. Nessa etapa, identificamos o valor máximo para userId e movieId para definir o tamanho das matrizes de *embeddings*.

```
df['userId'] = np.random.randint(0, 1000, df.shape[0])
df['movieId'] = df['id']

df['rating'] = df['vote_average']

max_user_id = df['userId'].max()
max_movie_id = df['movieId'].max()

print(f"Max userId: {max_user_id}, Max movieId: {max_movie_id}")
```

```
Max userId: 999, Max movieId: 1226578
```

Figura 4: Criação de colunas essenciais e volume máximo

3.3 Implementação da classe CollaborativeFiltering

O modelo foi construído utilizando uma classe customizada chamada *CollaborativeFiltering*, que herda da classe nn.Module do PyTorch. O modelo possui duas camadas de *embeddings* (user_embedding e item_embedding) que aprendem representações para usuários e filmes. O modelo recebe o número de usuários e itens e o tamanho do *embedding* como parâmetros. O tamanho dos *embeddings* foi definido como 50, mas pode ser ajustado dependendo dos recursos e da complexidade desejada. A função *forward* define o fluxo de dados, retornando o produto entre os *embeddings* de usuário e item.

```
class CollaborativeFiltering(nn.Module):
    def __init__(self, num_users, num_items, embedding_size=50):
        super(CollaborativeFiltering, self).__init__()
        self.user_embedding = nn.Embedding(num_users, embedding_size)
        self.item_embedding = nn.Embedding(num_items, embedding_size)

    def forward(self, user, item):
        user_embedded = self.user_embedding(user)
        item_embedded = self.item_embedding(item)
        return (user_embedded * item_embedded).sum(1)

num_users = max_user_id + 1
num_items = max_movie_id + 1

model = CollaborativeFiltering(num_users, num_items)
```

Figura 5: Criação do modelo de Filtragem Colaborativa

3.4 Definição da função de perda e otimizador

Para treinar o modelo de recomendação, foi utilizada a função de perda MSELoss (*Mean Squared Error Loss*), adequada para modelos de recomendação. Essa função é comumente usada em sistemas de recomendação, pois calcula a diferença entre as avaliações do modelo e as avaliações observadas. Para o otimizador, foi utilizado o Adam, que possui uma taxa de aprendizado de 0.01, sendo ele amplamente utilizado em deep learning por sua eficiência e rápida convergência.

```
criterion = nn.MSELoss()
optimizer = optim.Adam(model.parameters(), lr=0.01)
```

Figura 6: Definição da função de perda e do otimizador

3.5 Treinamento do modelo

Dividimos os dados em conjuntos de treino e teste utilizando a função train_test_split do scikit-learn. Em seguida, o modelo foi treinado por meio de um loop de épocas, onde em cada época ele faz previsões para os dados de treino, calcula a perda, realiza a retropropagação, isso é, ajusta os parâmetros para reduzir a diferença entre as previsões e os valores reais. O valor da perda foi exibido ao final de cada época para acompanhar a evolução do treinamento.

```
train_data, test_data = train_test_split(df[['userId', 'movieId', 'rating']], test_size=0.2, random_state=42)

user_train = torch.tensor(train_data['userId'].values, dtype=torch.long)
item_train = torch.tensor(train_data['movieId'].values, dtype=torch.long)
rating_train = torch.tensor(train_data['rating'].values, dtype=torch.float32)

epochs = 20
for epoch in range(epochs):
    model.train()
    optimizer.zero_grad()

    predictions = model(user_train, item_train)

    loss = criterion(predictions, rating_train)

    loss.backward()
    optimizer.step()

    print(f"Epoch {epoch+1}/{epochs}, Loss: {loss.item():.4f}")
```

[19] ✓ 4.9s

```
Epoch 1/20, Loss: 31.1119
Epoch 2/20, Loss: 27.3216
Epoch 3/20, Loss: 23.9139
Epoch 4/20, Loss: 20.8633
Epoch 5/20, Loss: 18.1441
Epoch 6/20, Loss: 15.7310
Epoch 7/20, Loss: 13.5993
Epoch 8/20, Loss: 11.7248
Epoch 9/20, Loss: 10.0842
Epoch 10/20, Loss: 8.6551
Epoch 11/20, Loss: 7.4161
Epoch 12/20, Loss: 6.3470
Epoch 13/20, Loss: 5.4289
Epoch 14/20, Loss: 4.6442
Epoch 15/20, Loss: 3.9767
Epoch 16/20, Loss: 3.4114
Epoch 17/20, Loss: 2.9346
Epoch 18/20, Loss: 2.5341
Epoch 19/20, Loss: 2.1987
Epoch 20/20, Loss: 1.9185
```

Figura 7: Código de treinamento do modelo e resultado de perdas

3.6 Definição da função de acurácia

Após o treinamento do modelo, se mostrou necessário a implementação de uma função que pudesse garantir a acurácia do treinamento, portanto foi utilizado novamente a biblioteca sklearn para que pudéssemos fazer este cálculo antes de fazer a avaliação do mesmo.

```
from sklearn.metrics import mean_squared_error, accuracy_score

def calculate_rmse(predictions, actuals):
    return np.sqrt(mean_squared_error(actuals, predictions))

def calculate_accuracy(predictions, actuals, threshold=3.5):

    predicted_relevance = (predictions >= threshold).numpy()
    actual_relevance = (actuals >= threshold).numpy()
    return accuracy_score(actual_relevance, predicted_relevance)
```

[15] ✓ 0.0s

Figura 8: Código da função para cálculo de acurácia

3.7 Avaliação do modelo

Após o treinamento, o modelo foi avaliado utilizando o conjunto de teste, com o objetivo de medir sua capacidade de generalização. Durante a avaliação, o modelo foi colocado em modo de teste (*evaluation mode*), e as previsões para os dados de teste foram geradas. Em seguida, três métricas principais foram calculadas para avaliar o desempenho:

3.7.1 A perda no conjunto de teste foi calculada utilizando a função de perda criterion, que no caso é o erro quadrático médio *(Mean Squared Error* - MSE). Isso permite quantificar a discrepância entre as previsões do modelo e os valores reais dos ratings.

3.7.2 O RMSE (*Root Mean Squared Error*) foi calculado a partir das previsões e dos ratings reais, oferecendo uma interpretação mais intuitiva da perda, já que o erro é expresso na mesma escala dos ratings.

3.7.3 A precisão (*Accuracy*) foi calculada com base na relevância dos ratings, considerando notas iguais ou superiores a 3.5 como relevantes. O valor obtido indica que o modelo foi capaz de identificar corretamente a relevância de um filme para um usuário em apenas 31.36% dos casos. Esse desempenho é próximo ao que seria obtido por acaso em um modelo aleatório (assumindo uma distribuição equilibrada de ratings), o que sugere que o modelo ainda não está capturando bem os padrões do dataset."

Esse processo de avaliação não só verifica a capacidade do modelo de fazer previsões precisas em dados não vistos, mas também fornece uma visão mais abrangente de seu desempenho, por meio de diferentes métricas que avaliam tanto o erro absoluto quanto sua capacidade de classificar corretamente itens relevantes.

```
user_test = torch.tensor(test_data['userId'].values, dtype=torch.long)
item_test = torch.tensor(test_data['movieId'].values, dtype=torch.long)
rating_test = torch.tensor(test_data['rating'].values, dtype=torch.float32)

model.eval()
with torch.no_grad():
    test_predictions = model(user_test, item_test)
    test_loss = criterion(test_predictions, rating_test)

    rmse = calculate_rmse(test_predictions.numpy(), rating_test.numpy())
    accuracy = calculate_accuracy(test_predictions, rating_test)

print(f"Test Loss: {test_loss.item():.4f}")
print(f"Test RMSE: {rmse:.4f}")
print(f"Test Accuracy: {accuracy:.4f}")
```

```
[17] ✓ 0.0s
...  Test Loss: 92.1273
     Test RMSE: 9.5983
     Test Accuracy: 0.3136
```

Figura 9: Código da avaliação do modelo e conjunto de testes

3.8 Função de recomendação por gênero

Para gerar recomendações específicas de filmes baseados em gêneros, foi criada uma função recommend_movies_by_genre, que filtra os filmes do gênero desejado e realiza previsões de avaliação para o usuário. Em seguida, os filmes são

ordenados com base nas avaliações previstas, e os principais títulos são exibidos como recomendação.

```python
def recommend_movies_by_genre(user_id, genre, top_n=5):
    genre_filtered_movies = df[df['Genres'].apply(lambda genres: genre in genres)].copy()

    movie_ids = genre_filtered_movies['movieId'].values

    user_tensor = torch.tensor([user_id] * len(movie_ids), dtype=torch.long)
    movie_tensor = torch.tensor(movie_ids, dtype=torch.long)

    model.eval()

    with torch.no_grad():
        predictions = model(user_tensor, movie_tensor)

    genre_filtered_movies.loc[:, 'predicted_rating'] = predictions.numpy()

    top_recommendations = genre_filtered_movies.sort_values('predicted_rating', ascending=False).head(top_n)

    return top_recommendations[['title', 'Genres', 'predicted_rating']]
```

Figura 10: Código da função de recomendação por gênero

3.9 Exemplo de uso da função de recomendação criada

Após definir a função de recomendação, foi realizado um teste para demonstrar seu funcionamento. Nesse exemplo, o usuário com id 123 requisita uma recomendação de filmes do gênero "Drama".

```python
user_id = 123
genre = 'Drama'

recommendations = recommend_movies_by_genre(user_id, genre, top_n=5)
print(recommendations)
```

[32] ✓ 0.0s

```
                title                                Genres  predicted_rating
900       Detachment                          ['Drama']         25.511501
1520     Peeping Tom  ['Thriller', 'Horror', 'Drama']         20.749077
7526      Beach Rats            ['Drama', 'Romance']         20.390316
1789            Rudy            ['Drama', 'History']         20.165062
3987  About Schmidt             ['Drama', 'Comedy']         19.697943
```

Figura 11: Filtragem por filmes que contenham o gênero `Drama` em sua categorização

É possível observar que, ao realizar uma nova requisição com o usuário de id 213, a recomendação gerada entrega outro resultado de recomendação.

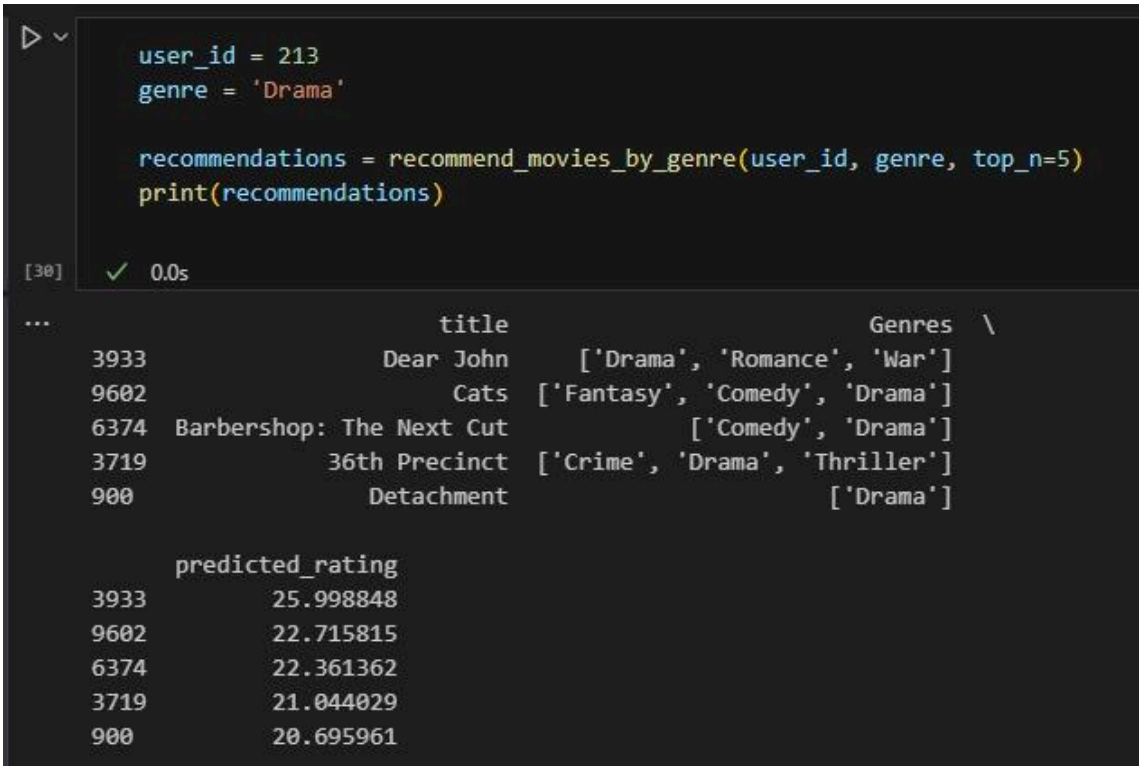

Figura 12: Filtragem por filmes que contenham o gênero `Drama` para outro usuário

3.10 Salvamento do modelo

Ao final do processo, o modelo treinado foi salvo em um arquivo utilizando o método *torch.save*, o que permite reutilizar o modelo no futuro sem a necessidade de novo treinamento. Esse arquivo pode ser carregado posteriormente para gerar recomendações sem a necessidade de refazer o processo completo de treinamento.

Figura 12: Salvamento do modelo

4. RESULTADO E DISCUSSÃO

O sistema de recomendação de filmes baseado em filtragem colaborativa, desenvolvido com o PyTorch, apresentou resultados promissores, apesar de algumas limitações. Durante o treinamento, foi possível observar uma redução consistente na métrica de perda (*Mean Squared Error* - MSE) nas épocas iniciais, o que indica que o

modelo conseguiu aprender padrões de interação entre usuários e filmes. Essa capacidade de aprendizado sugere que a abordagem tem potencial para capturar, em certa medida, as preferências dos usuários a partir dos dados fornecidos.

No conjunto de teste, os valores obtidos para as métricas *Test Loss*, RMSE e acurácia refletem tanto os pontos positivos quanto os desafios encontrados. O RMSE, por exemplo, apresentou um valor de 9.5983, o que indica que, em média, as previsões do modelo estavam razoavelmente distantes das avaliações reais dos usuários. Embora este valor esteja acima do ideal para sistemas de recomendação, ele ainda demonstra que o modelo foi capaz de fornecer predições com base nas interações observadas. A acurácia, calculada em 31.36%, mostra que o sistema conseguiu identificar corretamente filmes relevantes em cerca de um terço dos casos, o que é significativo para um modelo inicial baseado em dados simulados. Mesmo que os resultados ainda não atinjam os níveis esperados para aplicações reais, eles oferecem uma base importante para melhorias futuras.

A funcionalidade de recomendação por gênero destacou-se como um ponto positivo do sistema, permitindo a personalização das sugestões de filmes de acordo com os interesses específicos dos usuários. Essa abordagem contribui diretamente para aliviar a sobrecarga de escolha, um dos maiores desafios em plataformas de streaming. Filtrar e ordenar os filmes por gênero e avaliação prevista mostrou-se eficiente para entregar ao usuário uma lista personalizada, reduzindo o tempo necessário para encontrar algo interessante para assistir.

Por outro lado, desafios inerentes à filtragem colaborativa também foram observados. O problema de "*cold start*" — dificuldade em fazer boas recomendações para novos usuários ou itens com poucas avaliações — destacou-se como uma limitação do modelo. Esse problema, comum em sistemas baseados exclusivamente em filtragem colaborativa, sugere a necessidade de explorar abordagens híbridas no futuro, como a combinação de filtragem colaborativa com modelos baseados em conteúdo. Isso poderia aumentar a diversidade e a precisão das recomendações.

A escolha do PyTorch como framework foi fundamental para o desenvolvimento do sistema. Sua flexibilidade permitiu implementar *embeddings* personalizados e ajustar o modelo conforme necessário, demonstrando que a plataforma é adequada para lidar com problemas de recomendação em larga escala. Além disso, o PyTorch mostrou-se eficiente e escalável, o que é uma vantagem significativa para a adoção em ambientes reais, onde o volume de dados cresce rapidamente.

Em resumo, os resultados alcançados reforçam que, mesmo com limitações, o sistema foi capaz de capturar aspectos relevantes das preferências dos usuários e oferecer uma experiência inicial personalizada. Com ajustes e refinamentos, como a inclusão de dados reais, arquiteturas mais robustas e abordagens híbridas, o modelo pode evoluir para uma solução mais completa e eficiente.

5. CONCLUSÃO

O estudo realizado buscou explorar a implementação de um sistema de recomendação baseado em filtragem colaborativa, utilizando o framework PyTorch. O modelo foi avaliado com base em métricas como perda (*Test Loss*), RMSE e acurácia, e os resultados obtidos indicaram algumas limitações no desempenho. A métrica de RMSE, com valor de 9.5983, sugere que as previsões do modelo estiveram distantes das avaliações reais, refletindo um desempenho abaixo do esperado para sistemas de recomendação. Além disso, a acurácia de 31.36%, que indica a taxa de acerto do modelo ao classificar filmes como relevantes ou não, foi relativamente baixa, sugerindo

que o modelo não conseguiu aprender os padrões dos dados de maneira eficaz. Em comparação, um desempenho aleatório, onde o modelo não faria nenhuma previsão baseada em dados, poderia gerar uma taxa de acerto similar em um cenário com distribuição equilibrada de ratings.

Esses resultados apontam para a necessidade de ajustes no modelo, tanto na escolha do conjunto de dados quanto na arquitetura utilizada. O baixo desempenho pode ser atribuído a fatores como a limitação do dataset simulado, que não reflete interações reais de usuários com filmes, e a arquitetura simples do modelo, que pode não ser suficiente para capturar a complexidade das preferências dos usuários. O desafio do "*cold start*", que dificulta a criação de boas recomendações para novos usuários ou filmes, também contribui para essas limitações.

No entanto, apesar dessas dificuldades, o sistema desenvolvido demonstrou o potencial de oferecer recomendações personalizadas. Ele ajudou a reduzir a sobrecarga de escolha ao sugerir filmes baseados em preferências passadas, um fator crucial para melhorar a experiência do usuário. Com base nisso, podemos concluir que o modelo tem uma base promissora, mas requer melhorias, especialmente na coleta de dados mais representativos e na complexidade do modelo.

Além disso, a adoção de abordagens híbridas, que combinam filtragem colaborativa com técnicas baseadas em conteúdo, poderia ajudar a mitigar os problemas do "*cold start*" e melhorar a qualidade das recomendações desde o início. Outra melhoria importante seria a incorporação de técnicas de Processamento de Linguagem Natural (NLP), que permitiriam ao sistema captar nuances nas resenhas dos usuários, oferecendo uma camada adicional de personalização.

Essas melhorias podem aumentar a precisão e a diversidade das recomendações, proporcionando uma experiência mais dinâmica e personalizada para os usuários. Em resumo, o sistema de recomendação baseado em filtragem colaborativa e desenvolvido com PyTorch mostrou um bom ponto de partida, com várias possibilidades de aprimoramento, especialmente em termos de dados e arquitetura. Com ajustes adequados, o modelo pode evoluir para uma ferramenta eficaz e escalável para plataformas de streaming, oferecendo uma experiência mais satisfatória aos usuários.

REFERÊNCIAS

BOLLEN, D.; KNIJNENBURG, B. P.; WILLEMSEN, M. C.; GRAUS, M. Understanding choice overload in recommender systems. In: *Proceedings of the Fourth ACM Conference on Recommender Systems (RecSys '10)*. New York, NY: Association for Computing Machinery, 2010. p. 63–70. DOI: 10.1145/1864708.1864724.

CHERNLEV, A.; BOCKENHOLT, U.; GOODMAN, J. Choice Overload: A Conceptual Review and Meta-Analysis. *Journal of Consumer Psychology*, v. 25, p. 333–358, 2015. DOI: 10.1016/j.jcps.2014.08.002.

COHN, J. *The Burden of Choice: Recommendations, Subversion, and Algorithmic Culture*. New Brunswick, NJ: Rutgers University Press, 2019.

GÓMEZ-URIBE, C. A.; HUNT, N. The Netflix recommender system: algorithms, business value, and innovation. *ACM Transactions on Management Information Systems*, v. 6, n. 4, p. 13, dez. 2015. DOI: 10.1145/2843948.

GOOGLE. Fundamentos da recomendação colaborativa. *Google Developers*, 2024. Disponível em: https://developers.google.com/machine-learning/recommendation/collaborative/basics?hl=pt-br. Acesso em: 12 set. 2024.

IFADA, N.; NARIDHO, S.; SOPHAN, M. Multi-criteria based Item Recommendation Methods. *Rekayasa*, v. 12, n. 2, p. 78-84, 2019. DOI: 10.21107/rekayasa.v12i2.5913.

ISINKAYE, F. O.; FOLAJIMI, Y. O.; OJOKOH, B. A. Recommendation systems: Principles, methods and evaluation. *Egyptian Informatics Journal*, v. 16, n. 3, p. 261-273, 2015. ISSN 1110-8665. DOI: 10.1016/j.eij.2015.06.005.

KNIJNENBURG, B. P.; WILLEMSEN, M. C.; HIRTBACH, S. Receiving recommendations and providing feedback: The user-experience of a recommender system. In: BUCCAFURRI, F.; SEMERARO, G. (eds.). *E-Commerce and Web Technologies. EC-Web 2010*. Lecture Notes in Business Information Processing, v. 61. Berlin, Heidelberg: Springer, 2010. p. 295-306. DOI: 10.1007/978-3-642-15208-5_19.

KOREN, Y.; RENDLE, S.; BELL, R. Advances in Collaborative Filtering. In: RICCI, F.; ROKACH, L.; SHAPIRA, B. (eds.). *Recommender Systems Handbook*. New York, NY: Springer, 2022. p. 101–136. DOI: 10.1007/978-1-0716-2197-4_3.

MEETEREN, R. van; SOMEREN, M. van. Using Content-Based Filtering for Recommendation. Pesquisa apoiada pela NetlinQ. NetlinQ Group, Gerard Brandtstraat 26-28, 1054 JK, Amsterdam, The Netherlands; University of Amsterdam, Roeterstraat 18, The Netherlands. E-mail: robin@netlinq.nl; marten@swi.psy.uva.nl.

MEQUE, R. E. *Desenvolvimento de um Sistema de Recomendação utilizando Aprendizado de Máquina Não Supervisionado*. 2019. 18 f. Trabalho de conclusão de curso (Graduação) – Curso de Informática para Negócios, Faculdade de Tecnologia de São José do Rio Preto, São José do Rio Preto, 2019.

OMAR, H. K.; FRIKHA, M.; JUMAA, A. K. *PyTorch and TensorFlow Performance Evaluation in Big Data Recommendation System.* Ingénierie des Systèmes d'Information, v. 29, n. 4, p. 1357-1364, 2024. DOI: 10.18280/isi.290411.

PASKIN, J. Anxious to press play. *The Week*, 2 nov. 2019. Disponível em: https://theweek.com/articles/883299/anxious-press-play. Acesso em: 05 set. 2024.

PAZZANI, M. J.; BILLSUS, D. Content-Based Recommendation Systems. In: BRUSILOVSKY, P.; KOBSA, A.; NEJDL, W. (eds.). *The Adaptive Web*. Lecture Notes in Computer Science, v. 4321. Berlin, Heidelberg: Springer, 2007. p. 325–341. DOI: 10.1007/978-3-540-72079-9_10.

THORAT, P. B.; GOUDAR, R. M.; BARVE, S. Survey on collaborative filtering, content-based filtering and hybrid recommendation system. *International Journal of Computer Applications*, v. 110, n. 4, January 2015, p. 31-36