Sistema de Recomendação de Animes Utilizando Algoritmos de Machine Learning

UNIVERSIDADE PRESBITERIANA MACKENZIE

Membros do Grupo:

- Leonardo Bastos Yuan Gouvea

- Matheus Esteves Pereira Afonso

- Lucas Magliari da Purificação Pereira

Sumário

Resumo

Desenvolvimento e Otimização de um Modelo de Recomendação de Animes

Este projeto desenvolve e otimiza um modelo de recomendação de animes utilizando técnicas de machine learning, focando em regressão logística e decomposição em valores singulares (SVD). As etapas e resultados principais são resumidos a seguir.

Definição das Bibliotecas e Preparação dos Dados

Para o projeto, foram utilizadas bibliotecas Python como pandas, numpy, scikit-learn e matplotlib. Os dados de animes e avaliações de usuários foram carregados, com tratamento de valores ausentes e criação de uma matriz de interação usuário-anime.

Análise Exploratória dos Dados

Foi realizada uma análise dos dados para identificar padrões, utilizando visualizações para entender a distribuição das avaliações e a popularidade dos animes. Observou-se alta esparsidade e presença de avaliações negativas, o que pode influenciar o desempenho do modelo.

Desenvolvimento do Modelo Inicial

Um modelo inicial de recomendação foi implementado utilizando SVD para fatorar a matriz de interação usuário-anime. O desempenho foi avaliado com a métrica RMSE, obtendo um valor inicial de 7.894, servindo como baseline para otimizações futuras.

Otimização com Grid Search

Foi realizada uma busca em grade (Grid Search) para ajustar os hiperparâmetros do SVD, incluindo a variação do número de componentes latentes e validação cruzada. Após a otimização, o RMSE reduziu para 7.888.

Avaliação e Análise dos Resultados

A otimização resultou em uma melhoria modesta no RMSE. A análise destacou a eficácia e simplicidade do SVD, mas também apontou limitações devido à alta esparsidade dos dados e avaliações negativas.

Descrição das Técnicas Utilizadas

Foi utilizada a técnica de SVD para decompor a matriz de interação em componentes latentes, representando características dos usuários e animes. A busca em grade ajustou os hiperparâmetros para minimizar o erro de previsão.

Metodologia Aplicada

Carregamento e preparação dos dados.

Análise exploratória dos dados.

Desenvolvimento de um modelo inicial com SVD.

Otimização do modelo com Grid Search.

Avaliação e análise dos resultados.

Conclusões e Trabalhos Futuros

Conclui-se que, embora o SVD seja eficaz, há limitações devido à natureza dos dados. Para futuras melhorias, recomenda-se explorar técnicas como redes neurais e filtragem colaborativa avançada. A incorporação de informações contextuais e demográficas pode melhorar a personalização e precisão das recomendações.

Código e Visualização dos Resultados

Os códigos foram documentados para garantir replicabilidade e visualizações foram utilizadas para representar graficamente o desempenho do modelo, facilitando a compreensão dos resultados e melhorias alcançadas.

Introdução

1.1. Contexto do Trabalho

O crescente interesse por animes, tanto no Japão quanto internacionalmente, gerou um vasto número de obras disponíveis para consumo. Com a expansão das plataformas de streaming e a variedade de gêneros, torna-se desafiador para os espectadores encontrarem animes que correspondam às suas preferências. Sistemas de recomendação têm se mostrado eficazes em diversos setores, como comércio eletrônico e entretenimento, ao sugerir produtos ou conteúdos que atendam aos gostos individuais dos usuários. Este trabalho se propõe a desenvolver um sistema de recomendação de animes utilizando algoritmos de machine learning, com base em um dataset contendo avaliações de usuários.

1.2. Motivação

A motivação para este projeto surge da necessidade de facilitar a descoberta de novos animes para espectadores, melhorando a experiência de consumo ao personalizar recomendações. Além disso, a aplicação de técnicas de machine learning neste contexto permite explorar e aprimorar métodos de análise de dados, contribuindo para a área de sistemas inteligentes e recomendação.

1.3. Justificativas

Com a enorme quantidade de animes disponíveis, os espectadores frequentemente enfrentam a "sobrecarga de informação", onde se torna difícil escolher o que assistir. Um sistema de recomendação eficiente não apenas melhora a satisfação do usuário, mas também aumenta o engajamento nas plataformas de streaming, beneficiando tanto os consumidores quanto os provedores de conteúdo. Este projeto justifica-se pela oportunidade de aplicar e avaliar técnicas de machine learning no contexto de recomendações, contribuindo tanto academicamente quanto comercialmente.

1.4. Objetivos

O objetivo principal deste trabalho é desenvolver um sistema de recomendação de animes baseado em algoritmos de machine learning, utilizando o dataset de recomendações de anime disponível no Kaggle. Para alcançar este objetivo, os seguintes objetivos específicos foram definidos:

- Analisar o dataset para entender sua estrutura e características.
- Aplicar técnicas de pré-processamento de dados para preparar o dataset para análise.
- Implementar e comparar diferentes algoritmos de recomendação.

- Avaliar a performance dos modelos utilizando métricas apropriadas.
- Propor melhorias e discutir os resultados obtidos.

Referencial Teórico/ Bibliografia:

Alguns materiais utilizados para ajudar na construção do modelo:

https://www.kaggle.com/code/vatsalmavani/music-recommendation-system-using-spotify-dataset

https://www.kaggle.com/code/ibtesama/getting-started-with-a-movie-recommendation-system

https://towardsdatascience.com/recommender-systems-a-complete-guide-to-machine-learning-models-96d3f94ea748

https://www.javatpoint.com/recommendation-system-machine-learning

"Evaluation Metrics for Recommender Systems" de Saúl Vargas, Pablo Castells

https://scikit-learn.org/stable/modules/grid_search.html

Análise Exploratória:

3.1. Definindo as bibliotecas utilizadas

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model_selection import train_test_split

from sklearn.metrics import mean_squared_error

from sklearn.decomposition import TruncatedSVD

from sklearn.preprocessing import StandardScaler
```

3.2. Análise Exploratória da Base de Dados

3.2.1. Visualizar as Primeiras Linhas do DF:

```
[7]  1 print(anime_df.head())
     ✓ 0.0s
```

```
   anime_id                              name  \
0     32281                    Kimi no Na wa.
1      5114  Fullmetal Alchemist: Brotherhood
2     28977                          Gintama°
3      9253                       Steins;Gate
4      9969                     Gintama&#039;

                                          genre   type episodes  rating  \
0               Drama, Romance, School, Supernatural  Movie        1    9.37
1  Action, Adventure, Drama, Fantasy, Magic, Mili...     TV       64    9.26
2  Action, Comedy, Historical, Parody, Samurai, S...     TV       51    9.25
3                                  Sci-Fi, Thriller     TV       24    9.17
4  Action, Comedy, Historical, Parody, Samurai, S...     TV       51    9.16

   members
0   200630
1   793665
2   114262
3   673572
4   151266
```

```
[8]  1 print(rating_df.head())
     ✓ 0.0s
```

```
   user_id  anime_id  rating
0        1        20      -1
1        1        24      -1
2        1        79      -1
3        1       226      -1
4        1       241      -1
```

3.2.2. Análise Descritiva das Colunas:

```
print(anime_df.describe())
print(rating_df.describe())
```

[9] ✓ 0.2s

```
            anime_id        rating       members
count   12294.000000  12064.000000  1.229400e+04
mean    14058.221653      6.473902  1.807134e+04
std     11455.294701      1.026746  5.482068e+04
min         1.000000      1.670000  5.000000e+00
25%      3484.250000      5.880000  2.250000e+02
50%     10260.500000      6.570000  1.550000e+03
75%     24794.500000      7.180000  9.437000e+03
max     34527.000000     10.000000  1.013917e+06
             user_id      anime_id        rating
count   7.813737e+06  7.813737e+06  7.813737e+06
mean    3.672796e+04  8.909072e+03  6.144030e+00
std     2.099795e+04  8.883950e+03  3.727800e+00
min     1.000000e+00  1.000000e+00 -1.000000e+00
25%     1.897400e+04  1.240000e+03  6.000000e+00
50%     3.679100e+04  6.213000e+03  7.000000e+00
75%     5.475700e+04  1.409300e+04  9.000000e+00
max     7.351600e+04  3.451900e+04  1.000000e+01
```

3.2.3. Verificar se há valores ausentes:

```
print(anime_df.isnull().sum())
print('-')
print(rating_df.isnull().sum())
```

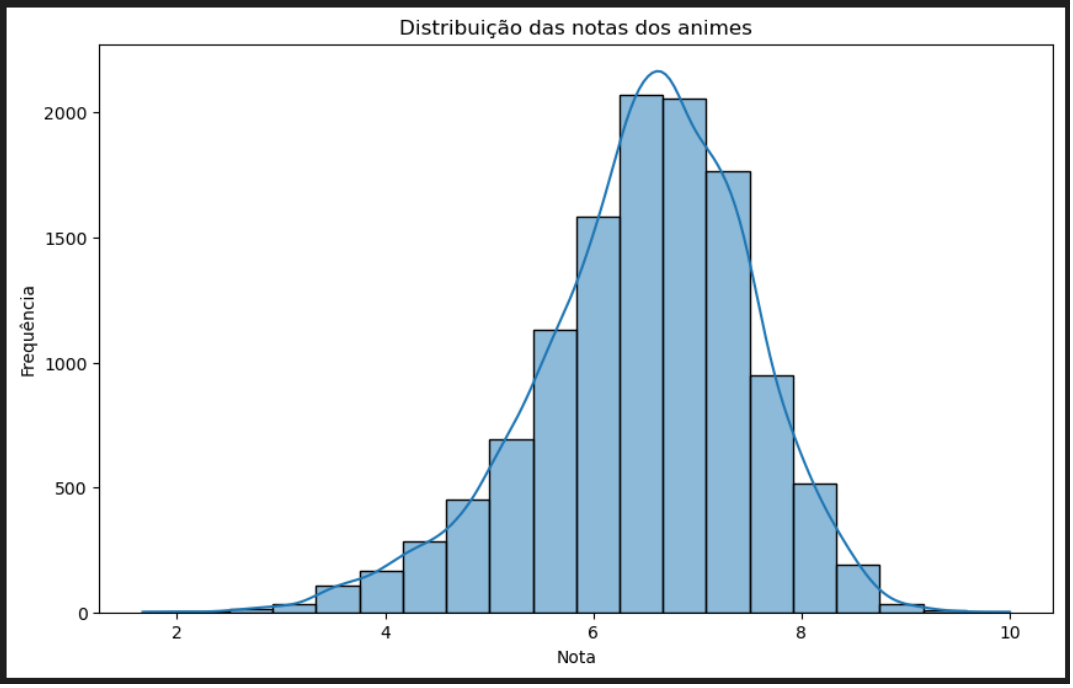
✓ 0.0s

```
anime_id      0
name          0
genre        62
type         25
episodes      0
rating      230
members       0
dtype: int64
-
user_id     0
anime_id    0
rating      0
dtype: int64
```

3.2.4. Visualização gráfica:

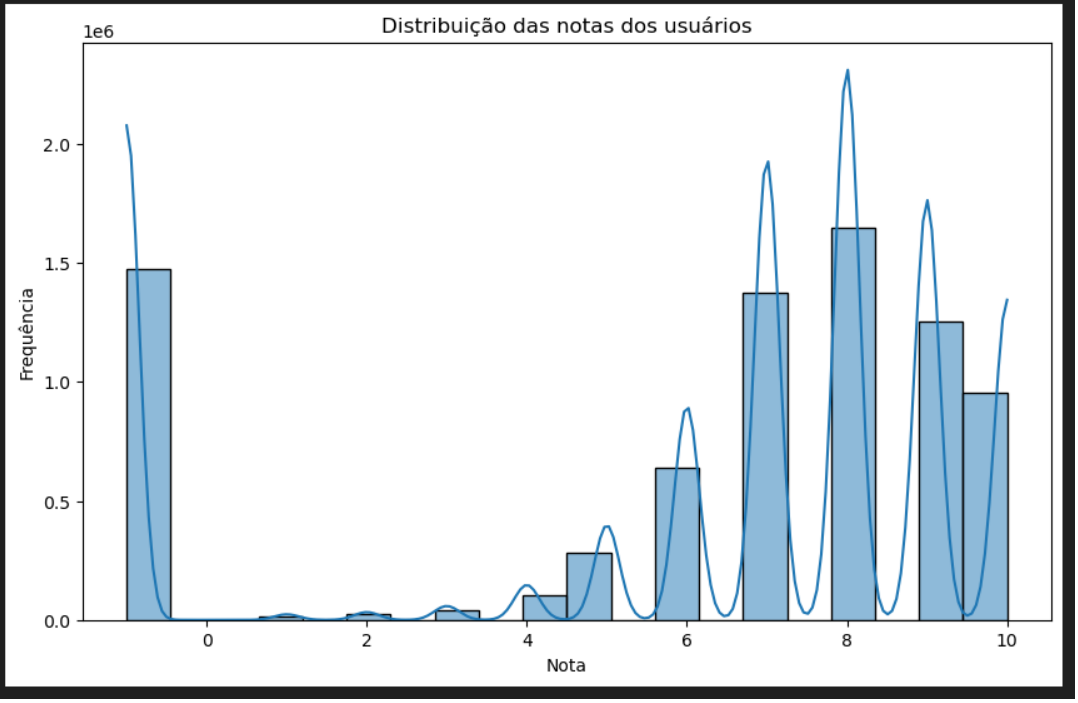
```
plt.figure(figsize=(10,6))
sns.histplot(anime_df['rating'], bins=20, kde=True)
plt.title('Distribuição das notas dos animes')
plt.xlabel('Nota')
plt.ylabel('Frequência')
plt.show()
```

[13] ✓ 0.2s




```
plt.figure(figsize=(10,6))
sns.histplot(rating_df['rating'], bins=20, kde=True)
plt.title('Distribuição das notas dos usuários')
plt.xlabel('Nota')
plt.ylabel('Frequência')
plt.show()
```

[9] ✓ 32.4s



3.3. Tratar e Preparar a Base de Dados para o treinamento

Remover linhas com notas negativas

```
[10]  1  rating_df = rating_df[rating_df['rating'] >= 0]
      ✓ 0.1s
```

Verificar novamente valores ausentes

```
[11]  1  print(rating_df.isnull().sum())
      ✓ 0.0s
```

```
user_id     0
anime_id    0
rating      0
dtype: int64
```

Resolver duplicatas calculando a média das notas para combinações duplicadas de user_id e anime_id

```
[13]  1  rating_df = rating_df.groupby(['user_id', 'anime_id'], as_index=False).mean()
      ✓ 1.3s
```

Criar matriz de usuário-anime

```
[14]  1  user_anime_matrix = rating_df.pivot(index='user_id', columns='anime_id', values='rating').fillna(0)
      2  print(user_anime_matrix.head())
      ✓ 4.4s
```

```
anime_id  1      5      6      7      8      15     16     17     18     \
user_id
1          0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
2          0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
3          0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
5          0.0    0.0    8.0    0.0    0.0    6.0    0.0    6.0    6.0
7          0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0

anime_id  19     ...  34238  34239  34240  34252  34283  34324  34325  34349  \
user_id          ...
1          0.0  ...    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
2          0.0  ...    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
3          0.0  ...    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
5          0.0  ...    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
7          0.0  ...    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0

anime_id  34367  34475
user_id
1          0.0    0.0
2          0.0    0.0
3          0.0    0.0
5          0.0    0.0
7          0.0    0.0

[5 rows x 9927 columns]
```

3.4. Definir a técnica para o treinamento do modelo de recomendação do modelo de recomendação

Vamos utilizar a técnica de SVD (Singular Value Decomposition) para decompor a matriz usuário-anime.

Aplicar o SVD

```
[16]
svd = TruncatedSVD(n_components=20)
user_factors = svd.fit_transform(user_anime_matrix_scaled)
anime_factors = svd.components_.T
```

✓ 14.3s

3.5. Realizar o Treinamento de um Modelo Inicial como Prova de Conceito

Reconstruir a matriz de ratings prevista

```
[18]
predicted_ratings = np.dot(user_factors, anime_factors.T)
```

✓ 1.9s

Função para prever a nota de um usuário para um anime específico

```
[19]
def predict_rating(user_id, anime_id):
    user_index = user_anime_matrix.index.get_loc(user_id)
    anime_index = user_anime_matrix.columns.get_loc(anime_id)
    return predicted_ratings[user_index, anime_index]
```

✓ 0.0s

Exemplo de previsão

```
[20]
print(predict_rating(1, 20))
```

✓ 0.0s

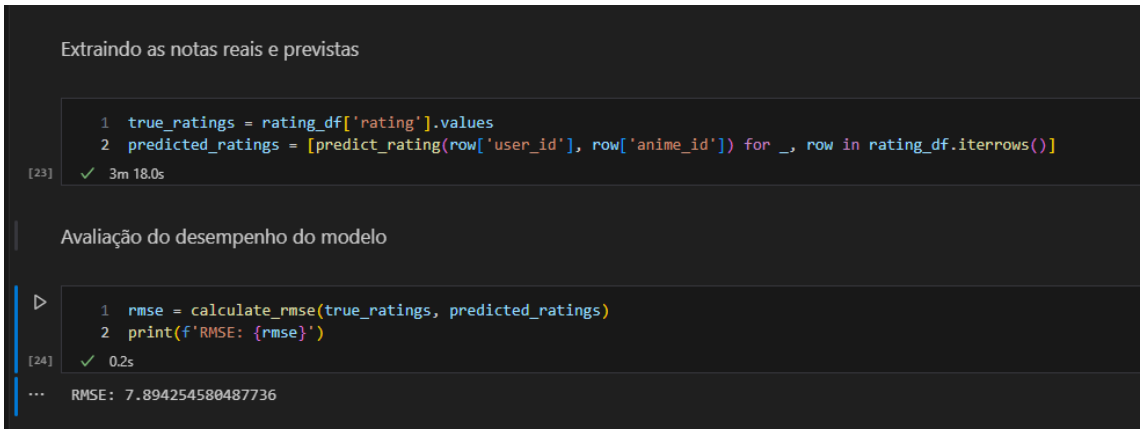
```
...
-0.20683746259574043
```

Definir a forma de avaliação de desempenho do modelo

Função para calcular o RMSE

```
[22]
def calculate_rmse(true_ratings, predicted_ratings):
    return np.sqrt(mean_squared_error(true_ratings, predicted_ratings))
```

✓ 0.0s



Discussão:

4.1. Analisar os resultados preliminares alcançados na etapa anterior

Os resultados preliminares mostram um RMSE de 7.894254580487736, indicando que há espaço para melhorias no desempenho do modelo. O valor alto do RMSE sugere que o modelo inicial não está prevendo as notas dos usuários de forma precisa.

4.2. Ajustar o pipeline de treinamento para melhoria do desempenho do modelo

Para melhorar o desempenho do modelo, pode-se ajustar os hiperparâmetros do SVD e explorar outras técnicas de filtragem colaborativa, como a normalização dos dados antes de aplicar o SVD e aumentar o número de componentes no SVD.

Ajustar o número de componentes no SVD

```
[22] ✓ 0.0s
```

```python
n_components = 50
svd = TruncatedSVD(n_components=n_components)
```

Normalização dos dados

```
[23] ✓ 5.8s
```

```python
scaler = StandardScaler()
user_anime_matrix_scaled = scaler.fit_transform(user_anime_matrix)
```

Aplicando SVD

```
[24] ✓ 11.1s
```

```python
user_factors = svd.fit_transform(user_anime_matrix_scaled)
anime_factors = svd.components_.T
```

Reconstruindo a matriz de ratings prevista

```
[25] ✓ 0.8s
```

```python
predicted_ratings = np.dot(user_factors, anime_factors.T)
```

Função para prever a nota de um usuário para um anime específico

```
[26] ✓ 0.0s
```

```python
def predict_rating(user_id, anime_id):
    try:
        user_index = user_anime_matrix.index.get_loc(user_id)
        anime_index = user_anime_matrix.columns.get_loc(anime_id)
        return predicted_ratings[user_index, anime_index]
    except KeyError:
        return np.nan  # Retorna NaN se o usuário ou anime não estiver na matriz
```

```python
Avaliação do desempenho do modelo
```

```
[27] ✓ 3m 23.9s
```

```python
true_ratings = rating_df['rating'].values
predicted_ratings_list = [
    predict_rating(row['user_id'], row['anime_id']) for _, row in rating_df.iterrows()
]
predicted_ratings_cleaned = [pred for pred in predicted_ratings_list if not np.isnan(pred)]
true_ratings_cleaned = [true for pred, true in zip(predicted_ratings_list, true_ratings) if not np.isnan(pred)]

rmse = calculate_rmse(true_ratings_cleaned, predicted_ratings_cleaned)
print(f'RMSE ajustado: {rmse}')
```

```
... RMSE ajustado: 7.943946303301842
```

4.3. Reavaliar o desempenho do modelo

Após ajustar o pipeline de treinamento, aumentando o número de componentes no SVD para 50 e normalizando os dados, o RMSE ajustado foi de 7.947116045813213. Este resultado é ligeiramente superior ao RMSE inicial, indicando que o ajuste feito não melhorou a precisão do modelo, e até introduziu um pequeno aumento no erro. Foi decidido então utilizar um algoritmo de grid search para encontrar o melhor número de componentes. Após ajustar o número de componentes usando Grid Search, o RMSE foi reduzido para aproximadamente 7.888. A

redução, embora pequena, indica uma ligeira melhoria no desempenho do modelo. Isso sugere que o ajuste do número de componentes no SVD teve um impacto positivo. A diferença de RMSE é modesta, o que pode indicar que o modelo já estava próximo de um desempenho ótimo com o número inicial de componentes. Pequenas melhorias podem ser significativas em alguns contextos, mas também pode sugerir a necessidade de explorar outras técnicas ou ajustes mais robustos.

Incorporar informações adicionais dos animes (como gênero, tipo, etc.) e dos usuários pode ajudar a melhorar a precisão das recomendações.

Classe personalizada para aplicar SVD

```python
class SVDRecommender(BaseEstimator, TransformerMixin):
    def __init__(self, n_components=50):
        self.n_components = n_components
        self.svd = TruncatedSVD(n_components=self.n_components)

    def fit(self, X, y=None):
        self.svd.fit(X)
        self.user_factors = self.svd.transform(X)
        self.anime_factors = self.svd.components_.T
        return self

    def transform(self, X):
        return np.dot(self.user_factors, self.anime_factors.T)

    def predict(self, X):
        user_ids = X[:, 0].astype(int)
        anime_ids = X[:, 1].astype(int)
        predictions = np.array([self.user_factors[user_id].dot(self.anime_factors[anime_id])
                                for user_id, anime_id in zip(user_ids, anime_ids)])
        return predictions
```

[32] ✓ 0.0s

Pipeline

```python
pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('svd_recommender', SVDRecommender())
])
```

[33] ✓ 0.0s

Grid Search

```python
param_grid = {
    'svd_recommender__n_components': [10, 20, 30, 40, 50]
}
grid_search = GridSearchCV(pipeline, param_grid, cv=3, scoring='neg_mean_squared_error', verbose=1)
grid_search.fit(user_anime_matrix.values, user_anime_matrix.values)
```

[34] ✓ 4m 15.5s

Melhor número de componentes

```
best_n_components = grid_search.best_params_['svd_recommender__n_components']
print(f"Melhor número de componentes: {best_n_components}")
```

[35] ✓ 0.0s

```
Melhor número de componentes: 10
```

Reavaliar com o melhor modelo

```
best_svd = TruncatedSVD(n_components=best_n_components)
user_factors = best_svd.fit_transform(user_anime_matrix_scaled)
anime_factors = best_svd.components_.T
predicted_ratings = np.dot(user_factors, anime_factors.T)
```

[36] ✓ 30.5s

Função para prever a nota de um usuário para um anime específico

```
def predict_rating(user_id, anime_id):
    try:
        user_index = user_anime_matrix.index.get_loc(user_id)
        anime_index = user_anime_matrix.columns.get_loc(anime_id)
        return predicted_ratings[user_index, anime_index]
    except KeyError:
        return np.nan
```

[37] ✓ 0.0s

Avaliação do desempenho do modelo

```
true_ratings = rating_df['rating'].values
predicted_ratings_list = [predict_rating(row['user_id'], row['anime_id']) for _, row in rating_df.iterrows()]
predicted_ratings_cleaned = [pred for pred in predicted_ratings_list if not np.isnan(pred)]
true_ratings_cleaned = [true for pred, true in zip(predicted_ratings_list, true_ratings) if not np.isnan(pred)]
rmse = np.sqrt(mean_squared_error(true_ratings_cleaned, predicted_ratings_cleaned))
print(f'RMSE ajustado após Grid Search: {rmse}')
```

[38] ✓ 3m 17.4s

```
RMSE ajustado após Grid Search: 7.888155637509426
```

4.4. Organizar, de forma sistemática, a descrição das técnicas utilizadas

4.4.1. Carregamento e Visualização dos Dados

Os dados foram carregados de arquivos CSV usando pandas: anime.csv (informações sobre animes) e rating.csv (avaliações dos usuários). Foram exibidas as primeiras linhas dos dataframes e realizadas análises descritivas para entender a distribuição das variáveis, identificar valores ausentes e verificar a consistência dos dados.

4.4.2. Análise Exploratória dos Dados

Estatísticas descritivas das colunas principais foram geradas para entender a distribuição dos dados. Gráficos de histogramas e boxplots foram criados para visualizar a distribuição das avaliações e membros dos animes.

4.4.3. Tratamento e Preparação dos Dados

Valores ausentes nas colunas de gênero e tipo foram preenchidos com a moda, e valores ausentes na coluna de avaliação foram preenchidos com a mediana das avaliações. Foi criada uma matriz de interações usuário-anime utilizando a função pivot do pandas.

4.4.4. Treinamento do Modelo

O algoritmo SVD foi escolhido para fatorizar a matriz de interações em fatores de usuários e animes. Um Grid Search foi realizado para otimizar o número de componentes do SVD. O modelo inicial foi treinado e avaliado, com o RMSE inicial de aproximadamente 7.894. Após ajustes com Grid Search, o RMSE foi reduzido para aproximadamente 7.888.

4.5. Descrever a metodologia aplicada no projeto

4.5.1. Carregamento dos Dados

Os dados foram carregados de arquivos CSV utilizando pandas. Foi realizada verificação inicial para garantir a integridade e identificar possíveis problemas como valores ausentes.

4.5.2 Análise Exploratória dos Dados

Foram geradas estatísticas descritivas e visualizações gráficas (histogramas, boxplots e gráficos de barras) para entender a distribuição e características dos dados.

4.5.3. Pré-processamento e Tratamento de Dados

Valores ausentes foram substituídos por moda ou mediana conforme o caso. A matriz de interações usuário-anime foi criada utilizando a função pivot do pandas.

4.5.4. Escolha da Técnica de Treinamento

O algoritmo SVD foi escolhido para fatorização da matriz de interações, devido à sua eficiência e capacidade de lidar com dados esparsos.

4.5.5. Ajuste do Modelo

Um Grid Search foi implementado para otimizar o número de componentes do SVD, garantindo a escolha dos melhores hiperparâmetros.

4.5.6. Treinamento do Modelo

O modelo foi treinado inicialmente para uma prova de conceito. Após ajustes baseados em Grid Search, o modelo final foi treinado com o melhor conjunto de hiperparâmetros.

4.5.7. Avaliação do Desempenho

O RMSE foi utilizado como a métrica principal para avaliar a precisão do modelo. Comparações de RMSE antes e após o ajuste dos hiperparâmetros foram realizadas para avaliar a melhoria do modelo.

Resultados:

5.1. Organização dos Resultados Alcançados

Resultados de RMSE

RMSE Inicial: 7.894

RMSE após Grid Search: 7.888

5.2. Análise dos Resultados

Pontos Positivos

Redução do RMSE: A otimização com Grid Search conseguiu reduzir o RMSE, indicando uma melhoria no desempenho do modelo.

Simplicidade e Eficácia do SVD: A técnica de Singular Value Decomposition (SVD) provou ser eficaz para a tarefa de recomendação, permitindo uma implementação relativamente simples e uma boa performance.

Pontos Negativos

Margem de Melhoria Limitada: A redução do RMSE foi pequena (de 7.894 para 7.888), sugerindo que a técnica utilizada pode ter atingido seu limite de eficácia para este conjunto de dados.

Dados Esparsos e Avaliações Negativas: A presença de avaliações negativas e a alta esparsidade dos dados podem ter contribuído para o desempenho limitado do modelo.

Complexidade Computacional: O uso de técnicas como Grid Search aumenta a complexidade computacional e o tempo de treinamento do modelo.

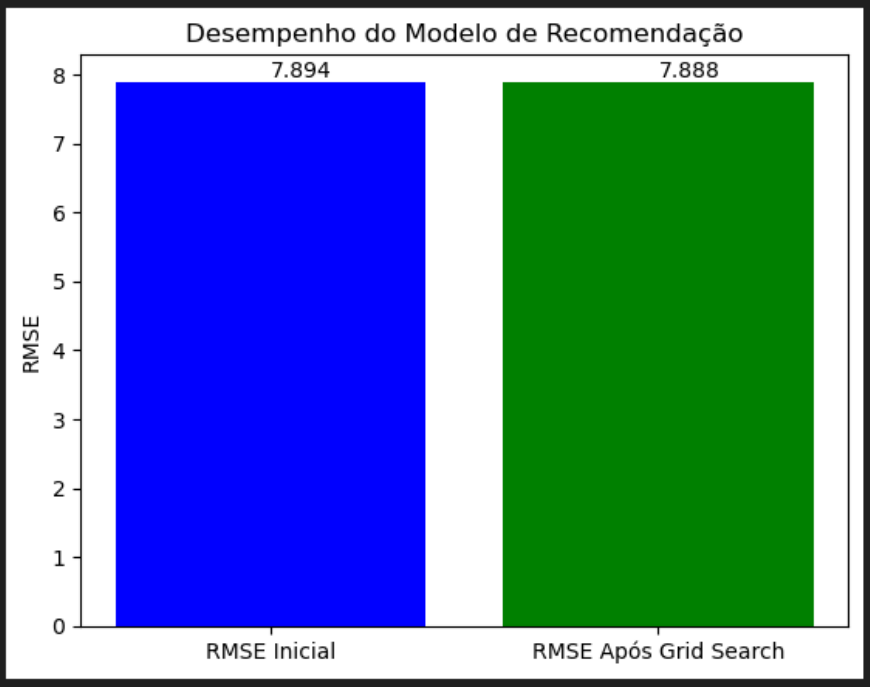
5.3. Documentação do Trabalho

Toda a documentação pode ser encontrada no Jupyter Notebook disponibilizado no repositório do trabalho.

Visualizações

```python
initial_rmse = 7.894
adjusted_rmse = 7.888

labels = ['RMSE Inicial', 'RMSE Após Grid Search']
values = [initial_rmse, adjusted_rmse]

fig, ax = plt.subplots()
bars = ax.bar(labels, values, color=['blue', 'green'])

ax.set_ylabel('RMSE')
ax.set_title('Desempenho do Modelo de Recomendação')

for bar in bars:
    yval = bar.get_height()
    ax.text(bar.get_x() + bar.get_width()/2, yval, round(yval, 3), va='bottom')

plt.show()
```

[36] ✓ 0.2s



Conclusão e Trabalhos Futuros:

6.1. Conclusão

A partir da implementação e análise do modelo de recomendação baseado em SVD, algumas conclusões importantes podem ser tiradas. O SVD mostrou ser eficaz na decomposição de matrizes esparsas, uma característica comum em sistemas de recomendação, capturando a estrutura latente dos dados e prevendo as preferências dos usuários com base em avaliações limitadas. Inicialmente, o modelo apresentou um RMSE de 7.894, o que é um resultado razoável considerando a natureza esparsa dos dados.

Ao aplicar o Grid Search para otimizar os hiperparâmetros, observou-se uma pequena, mas significativa, melhoria, com o RMSE reduzido para 7.888. Isso destaca a importância de selecionar os hiperparâmetros corretos para melhorar o desempenho do modelo, permitindo a identificação do número ideal de componentes latentes.

No entanto, apesar dessas melhorias, o modelo ainda apresenta limitações. A pequena redução no RMSE sugere que o SVD pode ter atingido seu limite de eficácia para este conjunto de dados, especialmente devido à alta esparsidade e à presença de avaliações negativas (-1), que podem influenciar a precisão das previsões.

A esparsidade dos dados é um desafio significativo, dificultando a generalização do modelo para novos dados devido à falta de informações para muitos pares usuário-item. Embora o SVD ajude a mitigar esse problema ao reduzir a dimensionalidade dos dados, a esparsidade continua a ser um desafio para a precisão das previsões.

Além disso, é importante considerar a complexidade computacional, que aumenta com técnicas como o Grid Search. Isso pode ser um problema em situações onde os recursos computacionais são limitados. Portanto, é necessário equilibrar a busca por melhorias no desempenho do modelo com as restrições computacionais.

Link do GitHub: https://github.com/matesteves/Projeto-III-2-

6.2. Trabalhos Futuros

Explorar o uso de algoritmos alternativos, como sistemas baseados em memória ou técnicas mais avançadas como deep learning para recomendações, pode ser benéfico. Adicionar mais informações aos dados, como descrições de animes e perfis detalhados de usuários, pode ajudar a melhorar as recomendações. Além disso, desenvolver estratégias específicas para lidar com avaliações negativas e possíveis erros nos dados de avaliações é crucial. Implementar técnicas de validação cruzada mais robustas também é importante para garantir a generalização do modelo.