NEXUS INFO

FINAL PROJECT

MOVIE RECOMMENDATION SYSTEM

PREPARED BY :

AARYA YOGESH PAKHALE
DATA ANALYST INTERN @NEXUS INFO
IIT KHARAGPUR

TABLE OF CONTENTS

INTRODUCTION

In today's digital age, the proliferation of streaming platforms and online movie databases has granted users unprecedented access to a vast array of cinematic content. However, with this abundance of choices comes the challenge of effectively navigating through the multitude of options to discover movies that align with individual preferences and tastes. In response to this challenge, the development of recommendation systems has emerged as a crucial tool for enhancing the user experience by providing personalized movie suggestions tailored to each user's unique interests.

This project report presents the design, implementation, and evaluation of a Movie Recommendation System leveraging collaborative filtering techniques. Collaborative filtering is a widely adopted approach in recommendation systems that relies on analyzing user behavior and preferences to generate personalized recommendations. By harnessing the collective wisdom of users' interactions with movies, our system aims to deliver accurate and relevant recommendations that enhance the overall movie-watching experience.

OBJECTIVES

1. **Personalized Movie Recommendations:** The primary objective of our Movie Recommendation System is to provide users with personalized movie recommendations based on their past viewing behavior and preferences. By analyzing user interactions with movies, including ratings and viewing history, our system aims to identify patterns and similarities to generate tailored recommendations that align with each user's tastes.
2. **User-Based Collaborative Filtering:** One of the key objectives of our recommendation system is to implement and evaluate user-based collaborative filtering techniques. By comparing the preferences of

similar users and leveraging their collective knowledge, our system seeks to generate recommendations that reflect users' interests and preferences.

3. **Movie-Based Collaborative Filtering:** In addition to user-based collaborative filtering, our system also incorporates movie-based collaborative filtering techniques. By analyzing similarities between movies based on user interactions, our system aims to recommend movies that are similar in content, genre, and style to those that users have enjoyed in the past.

DATA DESCRIPTION

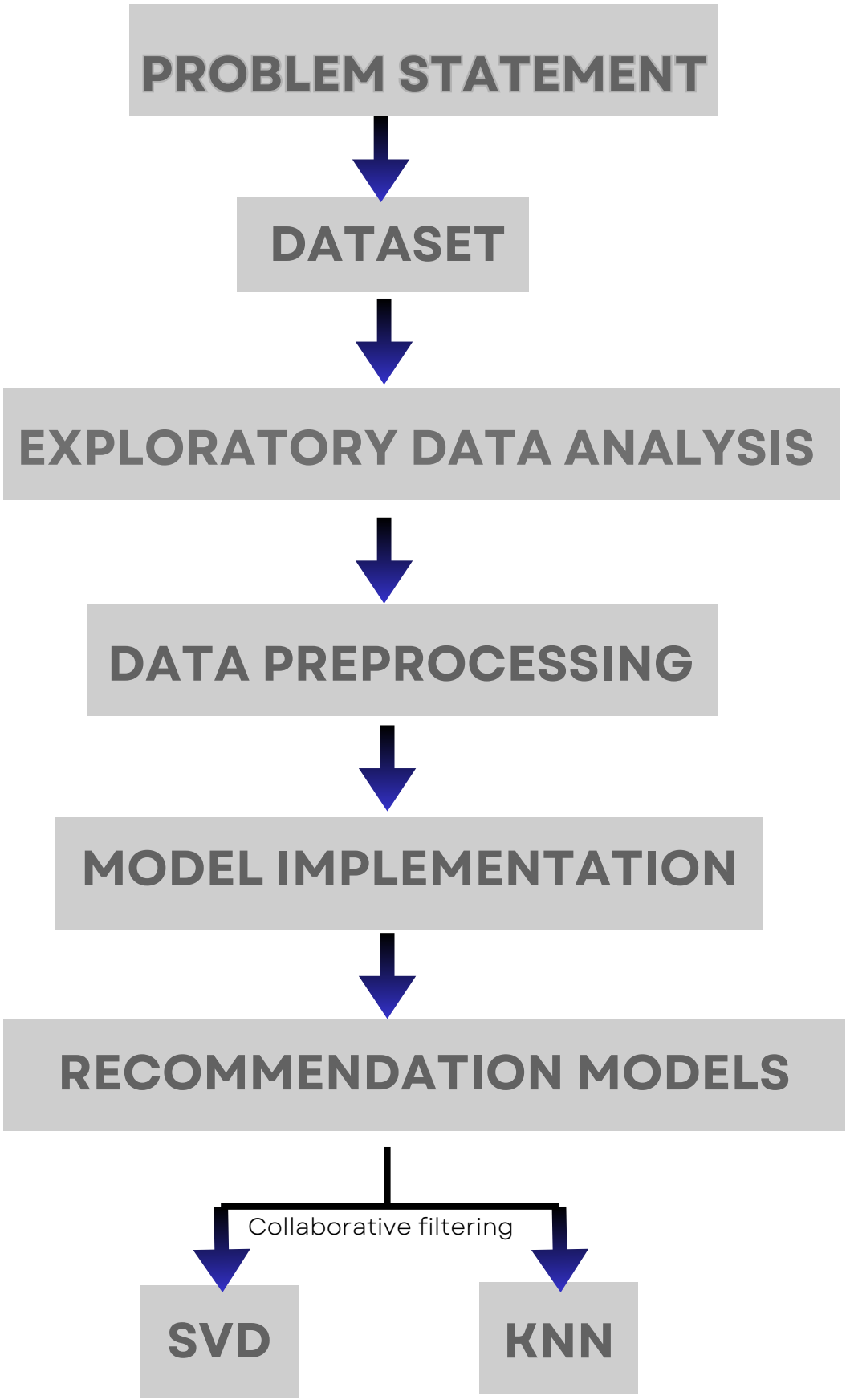
Source:
https://www.kaggle.com/datasets/rounakbanik/the-movies-dataset

Context
These files contain metadata for all 45,000 movies listed in the Full MovieLens Dataset. The dataset consists of movies released on or before July 2017. Data points include cast, crew, plot keywords, budget, revenue, posters, release dates, languages, production companies, countries, TMDB vote counts and vote averages.
This dataset also has files containing 26 million ratings from 270,000 users for all 45,000 movies. Ratings are on a scale of 1-5 and have been obtained from the official GroupLens website.

Files:
This dataset consists of the following files:

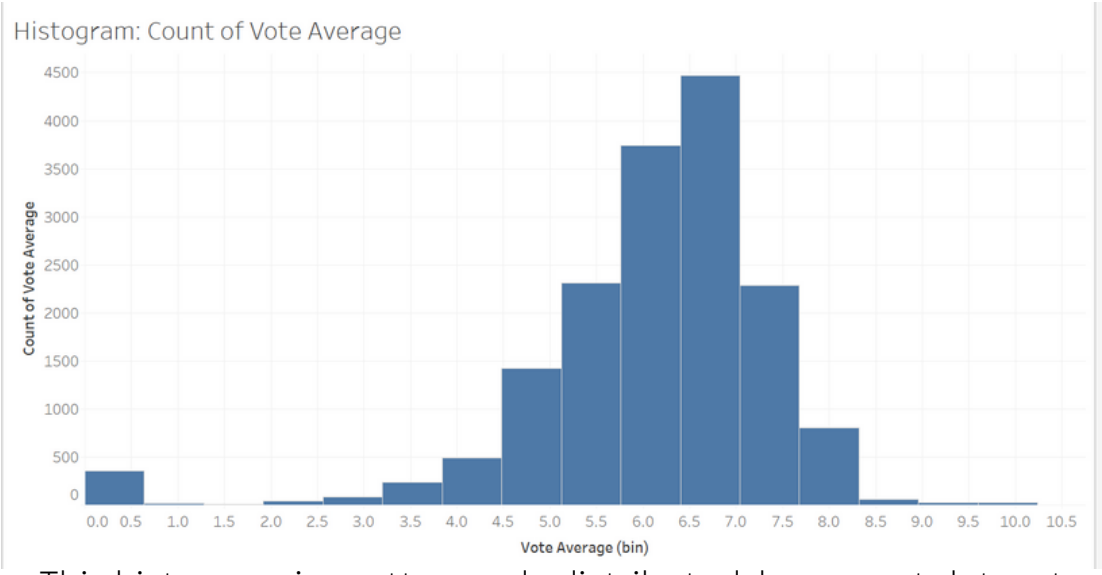
- movies_metadata.csv: The main Movies Metadata file. Contains information on 45,000 movies featured in the Full MovieLens dataset. Features include posters, backdrops, budget, revenue, release dates, languages, production countries and companies.
- ratings.csv: It contains the ratings for the movies by 270,000 users and the timestamp of the ratings.

MODEL APPROACH

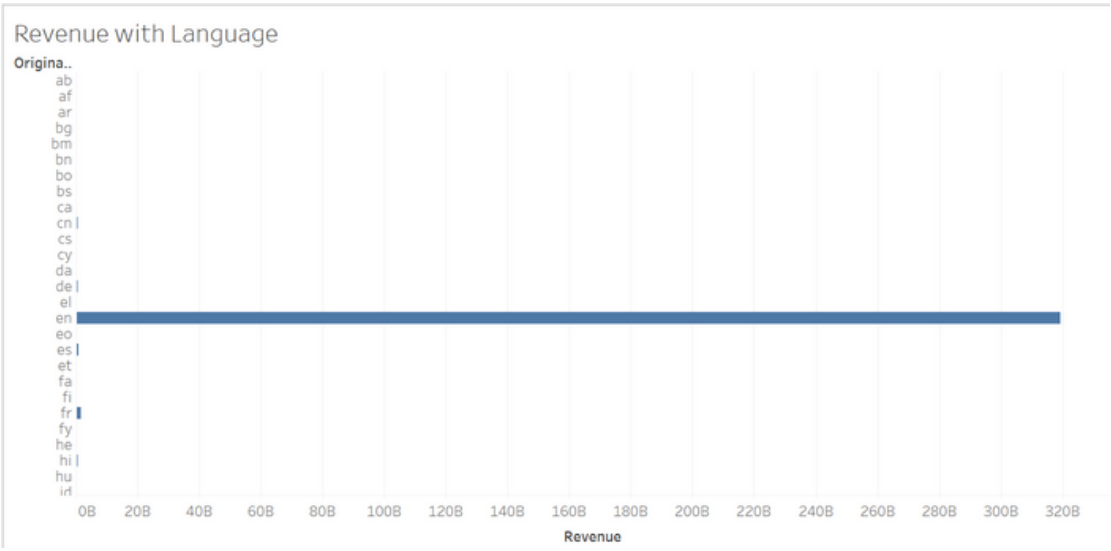


EXPLORATORY DATA ANALYSIS

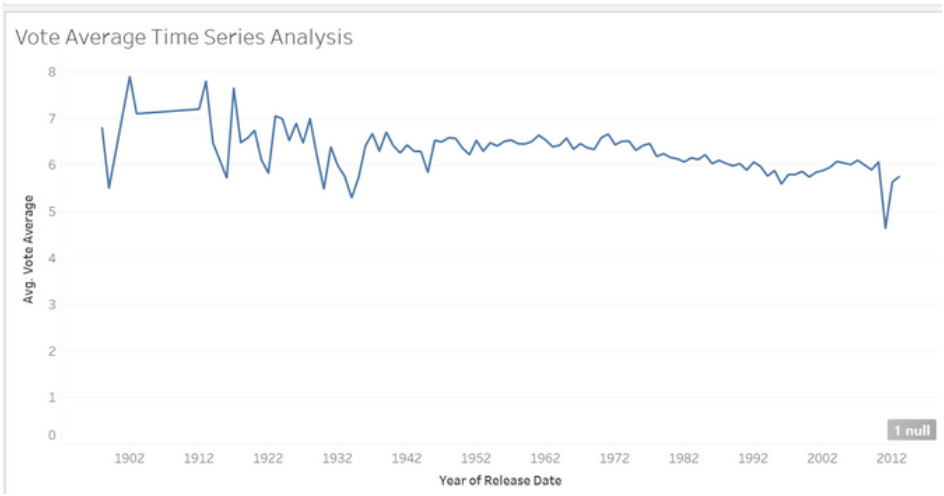
I have done the EDA on Tableau and Uploaded screenshots here.



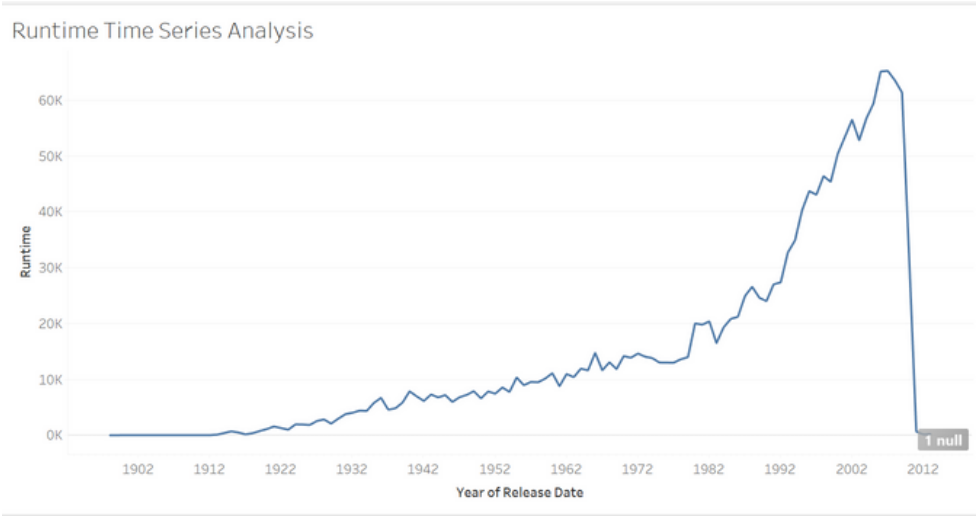
This histogram is pretty evenly distributed, hence out dataset need not be normalized.



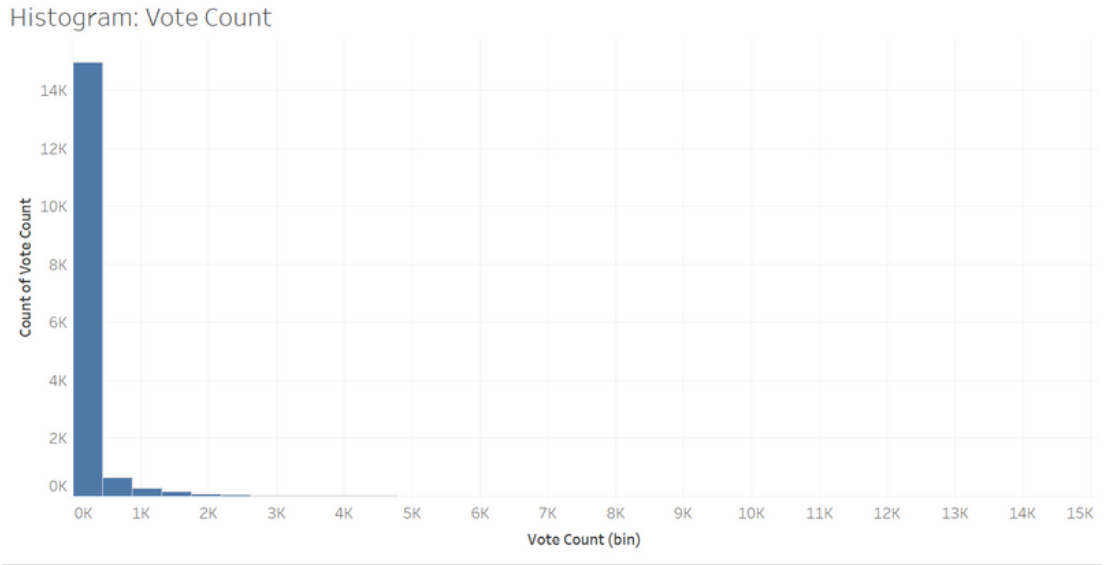
English Movies have a great dominance in the film industry



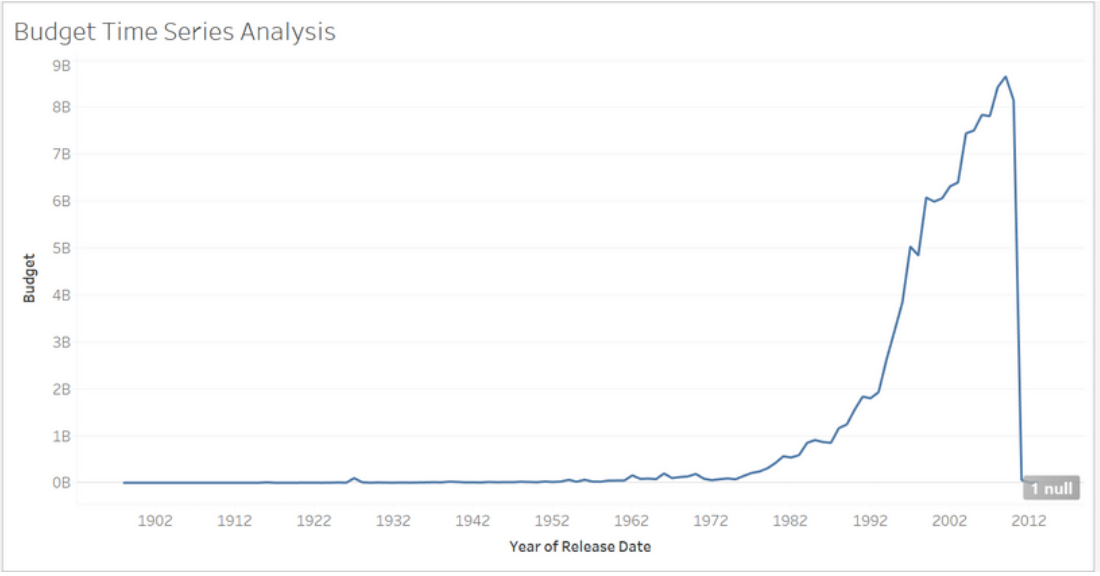
The average movie ratings in the 1910's was greater than the other decades



Runtimes peaked in the 2000's



Vote Counts are right skewed



It is seen that the budget has significantly peaked in the later years

DATA PREPROCESSING

1. **Data Cleaning:** Handling missing values, duplicates, and inconsistencies in the dataset. We eliminated the null values by replacing them by 0 in the user_rating_matrix.

1. **Feature Selection:** For this task, feature selection was direct, we picked up the user Ids and the movie Ids with the corresponding user ratings to generate the matrix which we used in this model.

1. **User-Item Matrix Creation:** Constructed a matrix where rows represent users Ids, columns represent movie Ids, and each cell contains user ratings, using the built-in pivot function. The system then uses various algorithms to analyze this matrix, find patterns, and generate recommendations.

1. **Similarity Calculation:** This approach was specifically used for the k- nearest neighbors model, User-based collaborative filtering recommendation model. It computes similarity between users or items using distance metric of cosine similarity. Cosine similarity is widely used in NLP to find similar context words.

1. **Sparse Matrix Creation:** Sparse matrices, by definition, store only the non-zero elements, significantly reducing memory usage. This is crucial when working with big data where storing dense matrices would be impractical or even impossible due to memory limitations. Sparse matrices are designed to optimize computations by focusing only on non-zero values. Our user_rating_matrix has dimensions of 1261, 9607 where most of the values are sparse. We are using only a small dataset but for the original large dataset of movie lens which has more than 100000 features, our system may run out of computational resources when that is feed to the model. To reduce the sparsity we use the csr_matrix function from the SciPy library.

1. **Filtered Movies:** We updated the user_rating_matrix by including only those movies which have more than 50 votes to have more reliable results and recommendations.

MODEL TRAINING

KNN Approach:

User-based collaborative filtering with K-Nearest Neighbors (KNN) is a technique for making movie recommendations based on the similarities between users in a recommendation system. Here's how it works:

1. User-Item Rating Matrix:
   - This matrix represents user-movie interactions, with rows representing users and columns representing movies.
   - Each cell contains the rating given by a specific user to a specific movie, or it might be left blank if the user hasn't interacted with the movie.
2. Finding Similar Users (KNN):
   - For a target user (the user for whom you want to make recommendations), you identify the k most similar users based on their rating history.
   - This similarity can be measured using various metrics, such as Pearson Correlation Coefficient, Cosine Similarity, or Euclidean Distance. These metrics quantify how closely two users' ratings align. We used cosine similarity in our case.
3. Predicting Ratings and Recommendations:
   - Once you have the k most similar users, you predict the target user's rating for unrated movies by:
     - Averaging the ratings of the k similar users for those movies.
     - Using weighted averages where weights represent the degree of similarity between the target user and each neighbor.
4. Top N Recommendations:
   - Based on the predicted ratings for unrated movies, you select the top N movies with the highest predicted ratings as recommendations for the target user.

Benefits:

- Simple to implement: User-based KNN is relatively easy to understand and implement compared to other collaborative filtering methods.
- Effective for sparse data: It can work well even when data is sparse, meaning users haven't rated many movies, as it focuses on finding similar users rather than a minimum number of ratings for each item.

Limitations:

- Scalability: As the dataset grows larger, finding the k nearest neighbors for each user can become computationally expensive.
- Cold Start Problem: New users or items with no ratings can be challenging to recommend for, as the system lacks sufficient data to identify similar users.

SVD Approach:

This technique aims to find latent factors that explain user preferences and item characteristics, ultimately making personalized recommendations.

Steps:

1. User-Item Rating Matrix: We construct a matrix where rows represent users, columns represent items, and each cell contains the rating given by a user for an item (or is left blank if unrated).
2. Matrix Factorization:
    - We decompose the user-item rating matrix (M) into three matrices using Singular Value Decomposition (SVD):
        - U (user matrix): Represents latent user factors, capturing user preferences.
        - Σ (diagonal matrix): Contains the singular values in decreasing order, indicating the importance of each latent factor.
        - V^T (transposed item matrix): Represents latent item factors, capturing item characteristics.

3. Recommendation:
    - For a target user and an unrated item, we:
        - Look up their corresponding latent factor vectors from U and V^T.
        - Predict the missing rating by calculating the dot product of these vectors.
        - Recommend items with the highest predicted ratings.

Benefits:

- Effective for sparse data: Works well even when users haven't rated many items, as it captures underlying relationships through latent factors.
- Handles cold start problem: Can potentially recommend for new users and items by analyzing their latent factors.

Limitations:

- Interpretability: Understanding the meaning of individual latent factors can be challenging.
- Scalability: SVD computation can become computationally expensive for very large datasets.

RESULTS

MODEL:KNN
RMSE: 0.9656
MAE: 0.7431
COMBINED: 0.85435

MODEL:SVD
RMSE: 0.9022
MAE: 0.6942
COMBINED: 0.7982

WHICH IS BETTER ?

Best Algorithm: SVD
RMSE: 0.9021792932050215
MAE: 0.6941951930408117
Combined Score: 0.7981872431229167

REASONS:

1. Handling Sparse Data:
   - SVD: Works well with sparse data, which is common in recommendation systems where users haven't rated many movies.
     - It captures underlying relationships between users and items through latent factors, even when explicit data is missing.
   - KNN: Relies on finding similar users based on their ratings.
     - In sparse data, finding truly similar users can be challenging, leading to less accurate recommendations.

2. Addressing Cold Start Problem:
   - SVD: Can potentially recommend for new users or items by analyzing their latent factors learned from the existing data.
   - KNN: Struggles with new users or items with no ratings, as it has no neighbors to compare them to.

3. Scalability:
   - SVD: While computationally expensive for very large datasets, it generally scales better than KNN, especially for denser datasets.

- KNN: Finding nearest neighbors becomes computationally expensive as the number of users and items increases in the dataset.

4. Capturing Complexities:
   - SVD: Can capture complex relationships between users and items by decomposing the data into multiple latent factors.
   - KNN: Primarily relies on direct similarities in user ratings, potentially missing subtle nuances in user preferences.

5. Overfitting:
   - KNN: More prone to overfitting in smaller datasets, as it relies solely on the training data for recommendations.
   - SVD: Can be less prone to overfitting due to its dimensionality reduction and focus on latent factors.

Techniques like hybrid models that combine the strengths of SVD and KNN or other approaches to potentially achieve even better results.

HYBRID MODELS (RESEARCH)

According to sources, the hybrid model should work the best, it is the one which combines best of both the worlds and gives accurate recommendations. Due to technical and resource constraints I was unable to implement this approach. Hybrid recommendation systems aim to overcome the limitations of individual recommendation techniques by combining two or more approaches to provide more accurate and diverse recommendations.

- Content-based filtering (CBF) + Collaborative filtering (CF):
  - Exploit both user preferences and item features.
  - Example: Recommend movies similar to those the user liked (CF) based on genres, actors, or directors (CBF).

- Benefits:

1. Improved accuracy and diversity: Can outperform individual methods by leveraging various recommendation strategies.
2. Addresses limitations: Mitigate the cold start problem (new users/items) or data sparsity issues by utilizing different data sources and recommendation techniques.
3. Flexibility: Can be customized based on the specific application and available data.

- Challenges:

1. Increased complexity: Designing and implementing hybrid systems can be more complex than single methods.
2. Data integration: Efficiently merging data and features from different sources can be challenging.
3. Parameter tuning: Finding the optimal combination of techniques and their parameters requires careful experimentation.

- Overall, hybrid recommendation systems can offer powerful solutions for building robust and effective recommendation systems, combining the strengths of different approaches while potentially overcoming their individual limitations.

ERROR MEASUREMENT METRICS USED

1. Mean Absolute Error (MAE):

- Calculates the average absolute difference between predicted and actual ratings.
- Interpretation: Measures the average magnitude of errors, regardless of their direction (positive or negative).
- Advantages:
  - Easy to interpret in the original rating scale (e.g., 1-star difference in movie rating).
  - Less sensitive to outliers compared to RMSE.
- Disadvantages:
  - Ignores the direction of errors (over or underestimation).

2. Root Mean Squared Error (RMSE):

- Calculates the square root of the average squared difference between predicted and actual ratings.
- Interpretation: Measures the average magnitude of errors, but penalizes larger errors more heavily due to squaring.
- Advantages:
  - Takes into account the magnitude of errors.
  - Commonly used metric for regression problems, allowing comparison with other models.
- Disadvantages:
  - Sensitive to outliers, as squaring large errors amplifies their impact.
  - Results are not interpretable in the original rating scale.

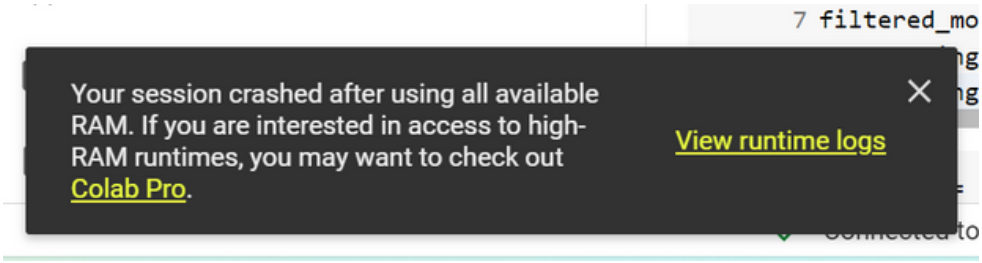
We have averaged out or combined the above two metrics to get a better estimation of the efficiency the two movie recommendation systems

DRAWBACKS/CONSTRAINTS

- I downloaded a dataset from kaggle for my recommendation system, which had been compiled 6 years ago. So, my model won't work as efficiently on the latest movies. It has to be continuously updated with the latest editions of the dataset for it to be of real time great use.

- Due to limited RAM and disk space, my code kept on crashing when I tried to run it on the movies_metadata.csv file. Due to I had to run it on just a part of the dataset and not the whole.

- In the SVD model, the parameter k could not be fine-tuned due to lack of actual data which can has to collected from user reviews.



Prompt given

CODE SNIPPETS

IMPORTS AND FILE READING

```
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy.sparse import csr_matrix
from sklearn.neighbors import NearestNeighbors
from surprise import Dataset, Reader
from surprise.model_selection import train_test_split, GridSearchCV
from surprise import SVD, KNNBasic
from surprise import accuracy
```

```
user_ratings_df = pd.read_csv("ratings.csv",encoding='ISO-8859-1')
user_ratings_df.isna().sum()
user_ratings_df.head()
```

```
movie_metadata = pd.read_csv("movies_metadata.csv",encoding='ISO-8859-1', skiprows=lambda x: x == 5075)
movie_metadata =movie_metadata.rename(columns={'id': 'movieId'})
movie_names=movie_metadata[['title', 'genres']]
movie_metadata.head()

```

```
<ipython-input-3-99c6d429882c>:1: DtypeWarning: Columns (10) have mixed types. Specify dtype option on import or set
  movie_metadata = pd.read_csv("movies_metadata.csv",encoding='ISO-8859-1', skiprows=lambda x: x == 5075)
```

DATA PREPROCESSING

```
movie_metadata['movieId'] = movie_metadata['movieId'].astype(str)
user_ratings_df['movieId'] = user_ratings_df['movieId'].astype(str)
movie_data = user_ratings_df.merge(movie_metadata, on='movieId')

```

```
user_rating_matrix=user_ratings_df.pivot(index=['userId'], columns=['movieId'], values='rating').fillna(0)
user_rating_matrix.head()

```

```
#filtering the movies based on ratings count
movie_counts = user_rating_matrix.sum(axis=0)
filtered_movies = movie_counts[movie_counts > 50].index
user_rating_matrix = user_rating_matrix.loc[:, filtered_movies]
user_rating_matrix.head()
```

```
csr_data = csr_matrix(user_rating_matrix.values)
user_rating_matrix.reset_index(inplace=True)
```

PS: Data cleaning Code not included

MODEL IMPLEMENTATION

KNN MODEL

```
knn = NearestNeighbors(metric='cosine', algorithm='brute', n_neighbors=20, n_jobs=-1)
knn.fit(csr_data)
```

```
def movie_recommender_engine(movie_name, matrix, cf_model, n_recs):
    # Fit model on matrix
    knn.fit(matrix)

    # Extract input movie ID
    movie_id = process.extractOne(movie_name, movie_names['title'])[2]

    # Calculate neighbour distances
    distances, indices = cf_model.kneighbors(matrix[movie_id], n_neighbors=n_recs)
    movie_rec_ids = sorted(list(zip(indices.squeeze().tolist(),distances.squeeze().tolist())),key=lambda x: x[1])[:0:-1]

    # List to store recommendations
    cf_recs = []
    for i in movie_rec_ids:
        cf_recs.append({'Title':movie_names['title'][i[0]],'Distance':i[1]})

    # Select top number of recommendations needed
    df = pd.DataFrame(cf_recs, index = range(1,n_recs))

    return df
```

```
n_recs = 14
movie_recommender_engine('Iron man', csr_data, knn, n_recs+1)

```

	Title	Distance
1	Escape from New York	0.725209
2	When a Man Loves a Woman	0.724886
3	Notorious	0.723352
4	Chinatown	0.719292
5	Raise the Red Lantern	0.716614
6	Koyaanisqatsi	0.716560
7	Gumby: The Movie	0.711370
8	Color of Night	0.710739
9	The Getaway	0.706452
10	Showgirls	0.706425
11	Bitter Moon	0.690645
12	Happy Gilmore	0.688997
13	The Seventh Seal	0.683068
14	Highlander	0.655813

SVD MODEL

```python
#svd code for user-vise collaborative recommendations
from scipy.sparse.linalg import svds

def recommend_movies(user_rating_matrix, k, user_id, N=5):

  # Perform SVD with chosen k
  U, S, Vt = svds(user_rating_matrix, k)

  # Predict missing ratings
  predicted_ratings = np.dot(U, np.diag(S) @ Vt)

 # Recommend top N items
  user_ratings = predicted_ratings[user_id]
  item_indices = np.argsort(user_ratings)[::-1][:N]  # Sort and get top N indices
  return item_indices

```

```python
k = 3
user_rating_matrix = user_rating_matrix.to_numpy()
recommended_movies = recommend_movies(user_rating_matrix, k, user_id=1, N=3)
movie_list=[]
for i in recommended_movies:
  movie_list.append(movie_metadata.iloc[i]['title'])
print(f"Recommended movies for user 1: {movie_list}")

```

```
Recommended movies for user 1: ['Toy Story', 'Broken English', 'La Collectionneuse']
```

ERROR MEASUREMENT METRICS AND COMPARISON OF THE EFFICIENCY OF THE MODELS

```python
#Code to find which model works better
data = pd.read_csv("ratings_small.csv",encoding='ISO-8859-1')
df = pd.DataFrame(data)
df = df.drop(columns = 'timestamp')
```

```python
reader = Reader(rating_scale=(0,5))
data = Dataset.load_from_df(df , reader)
```

```python
train_set , test_set = train_test_split(data ,test_size = 0.2 )
algorithms = [SVD(), KNNBasic()]
```

```python
rmse_vals = []
mae_vals = []
for algo in algorithms:
  algo.fit(train_set)
  preds = algo.test(test_set)
  rmse = accuracy.rmse(preds)
  mae = accuracy.mae(preds)
  rmse_vals.append(rmse)
  mae_vals.append(mae)
```

```python
combined_acc = [(rmse + mae) / 2 for rmse, mae in zip(rmse_vals, mae_vals)]
```

```python
best_algo_index = combined_acc.index(min(combined_acc))
best_algo_name = algorithms[best_algo_index].__class__.__name__
print(f"Best Algorithm: {best_algo_name}")
print(f"RMSE: {rmse_vals[best_algo_index]}")
print(f"MAE: {mae_vals[best_algo_index]}")
print(f"Combined Score: {combined_acc[best_algo_index]}")
```

THE END