MOVIE RECOMMENDATION SYSTEM USING MACHINE LEARNING

Eswar Revanth Chigurupati, Parul Ghai, Prateek Gupta, Rohan Nitin Khandke, Yasasvi Keerthi

RUTGERS – THE STATE UNIVERSITY OF NEW JERSEY

Newark, New Jersey, USA

1. **ABSTRACT**

In today's world of lots of information and too many movies to choose from, users often feel overwhelmed. The rise of streaming platforms and tons of available movies has changed how people pick what to watch. With so many choices, users struggle to find movies that match what they like. That's where movie recommendation systems come in. These systems, powered by machine learning like the K-Nearest Neighbors (KNN) algorithm, help users by suggesting movies that fit their tastes. It's a way to make navigating through all the options easier in the digital world.

2. **INTRODUCTION**

This project is about creating a Python Movie Recommendation System to help users deal with too much information. This system uses machine learning, specifically the K-Nearest Neighbors (KNN) algorithm, to give personalized movie suggestions.

The project covers everything from processing data to exploring it, using a couple of important datasets. It uses statistical measures, how often users engage, and a

movie similarity analysis to make the recommendation system better. A special function suggests movies based on what a user likes the most, showing how collaborative filtering works.

The report talks about two types of filtering—collaborative and content-based—and how they each have good and not-so-good points. It explains how combining these approaches makes the recommendation system stronger.

Overall, the Python Movie Recommendation System is a helpful tool to deal with too much information, giving users suggestions that fit them. The report highlights how versatile and user-focused the system is, with room for improvement. In the end, this system makes the user experience better and helps businesses succeed in the digital world.

3. **OBJECTIVE**

The primary objective of implementing a movie recommendation system using the KNN algorithm is to enhance the user experience by offering curated and personalized movie recommendations. This entails the following key goals:

- **Personalization:** Develop a system that understands and adapts to the unique preferences of individual users. By analyzing historical user behavior, the recommendation system aims to provide suggestions aligned with a user's taste and preferences.
- **Relevance:** Ensure that the recommended movies are not only personalized but also relevant. The KNN algorithm, by identifying similarities between movies based on user ratings, strives to recommend films that share characteristics with those the user has enjoyed in the past.
- **User Engagement:** Enhance user engagement with the platform by creating a dynamic and responsive movie recommendation system. The system

should evolve over time, learning from user interactions and continuously improving its suggestions.

- **Algorithmic Transparency:** Provide users with insights into how the recommendation system operates. Transparency builds trust and allows users to better understand the basis for the suggested movies, fostering a positive user-system relationship.
- **Scalability:** Design a system that is scalable and adaptable to accommodate a growing user base and an expanding movie catalog. Scalability ensures that the recommendation system remains effective even as the dataset and user interactions increase.
- **Evaluation:** Implement robust evaluation metrics to assess the performance of the recommendation system. Metrics such as precision, recall, and user satisfaction scores will be employed to measure the effectiveness and accuracy of the KNN algorithm in providing movie suggestions.

4. **PROBLEM STATEMENT**

In the realm of digital entertainment platforms, the sheer abundance of movie choices often overwhelms users, leading to decision fatigue and a less-than-optimal viewing experience. The absence of personalized guidance results in users spending considerable time searching for movies that align with their preferences. This challenge is exacerbated by the ever-expanding libraries of content, making it increasingly difficult for users to discover films tailored to their individual tastes.

The primary problem addressed by this project is the need for an efficient and personalized movie recommendation system. Conventional methods of manual curation and generic suggestions fall short in providing users with a streamlined and tailored content discovery experience. To overcome this, we employ machine learning, specifically the K-Nearest Neighbors (KNN) algorithm, to develop a recommendation system capable of understanding user preferences and delivering relevant movie suggestions.

Key Challenges:

- Information Overload: The vast array of movies available on digital platforms creates a scenario of information overload, making it challenging for users to navigate and find movies that resonate with their interests.
- User Heterogeneity: Users exhibit diverse tastes, preferences, and viewing habits. Creating a recommendation system that adapts to individual user profiles while considering the dynamic nature of preferences poses a significant challenge.
- Scalability: As the user base and content library grow, the recommendation system must scale effectively, ensuring that it can handle large datasets and provide timely and accurate suggestions.
- Cold Start Problem: New users or movies with limited historical data present a challenge known as the cold start problem. The recommendation system must devise strategies to offer relevant suggestions even in the absence of extensive user interaction history.
- Algorithmic Robustness: The selected KNN algorithm should be robust enough to handle noise and outliers in the data, ensuring that recommendations are based on meaningful patterns rather than spurious correlations.

5. LITERATURE REVIEW: MOVIE RECOMMENDATION SYSTEMS

Movie recommendation systems have evolved over the years, driven by advancements in machine learning and data analytics. These systems play a pivotal role in enhancing user experience by providing personalized suggestions tailored to individual preferences. Various approaches and algorithms have been explored in the literature to address the challenges of information overload and user heterogeneity.

Recommendation Approaches:

Content-Based Filtering:

- Description: Content-based filtering recommends items based on the similarity between the content of the items and a user profile. In the context of movies, this could involve considering genres, actors, directors, and other metadata.
- Advantages: Effective for new users, as it doesn't rely on historical user data.
- Limitations: May face challenges in capturing diverse user tastes and addressing the cold start problem for new items.

Collaborative Filtering:

- Description: Collaborative filtering predicts user preferences by leveraging the preferences and behaviors of other users. It can be user-based or item-based, relying on similarity measures.
- Advantages: Effective in capturing complex user preferences, especially when users exhibit similar tastes.
- Limitations: Vulnerable to the cold start problem for new users and items. Scalability can be an issue as the user base grows.

In this project, the focus is on collaborative filtering, particularly user-item collaborative filtering. The user-item matrix serves as the foundation for capturing user preferences and relationships between users and movies. The key steps include:

User-Item Matrix Creation:

- Description: The user-item matrix represents user ratings for movies. Users and movies are mapped to indices, and the matrix is constructed, where each entry represents a user's rating for a specific movie.

- Role in Project: Fundamental for collaborative filtering, as it forms the basis for similarity calculations and recommendation generation.

Similarity Analysis using K-Nearest Neighbors (KNN):

- Description: KNN is employed to find similar movies or users based on similarity measures. It identifies neighbors in the user-item matrix, allowing the system to recommend movies based on user behavior.
- Role in Project: Forms the core of collaborative filtering, enabling the system to suggest movies similar to those liked by a user.
- Explicit and Implicit Feedback
- Recommendation systems handle explicit feedback, such as user ratings, and implicit feedback, which includes user behavior like listening, purchasing, or watching.

Rating Prediction:

Let rx be the vector of user x's rating. Let N be the set of k similar users who also rated item i. Then we can calculate the prediction of user x and item i by using following formula:

$$r_{xi} = \frac{\sum_{y \in N} S_{xy} r_{yi}}{\sum_{y \in N} S_{xy}} \quad S_{xy} = sim(x, y)$$

6. IMPLEMENTATION

6.1. Data Processing and Exploration

The data processing and exploration phase is a critical precursor to any analysis, providing a foundational understanding of the dataset. In this phase, the Python

environment is initialized by importing essential libraries, each serving a specific purpose.

NumPy: Provides support for large, multi-dimensional arrays and matrices, along with mathematical functions to operate on these arrays.

Pandas: Offers data structures like DataFrames, facilitating efficient data manipulation and analysis.

Scikit-learn: A machine learning library that includes tools for data mining and data analysis. It provides various algorithms for clustering, classification, regression, and more.

Matplotlib: A comprehensive plotting library for creating static, animated, and interactive visualizations in Python.

Seaborn: Built on top of Matplotlib, Seaborn provides an aesthetically pleasing interface for statistical data visualization.

```
In [42]: # Importing Libraries
         import numpy as np
         import pandas as pd
         import sklearn
         import matplotlib.pyplot as plt
         import seaborn as sns
         import warnings
         warnings.simplefilter(action='ignore', category=FutureWarning)
```

6.2. Loading Datasets

	A	B	C
1	movieId	title	genres
2	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy
3	2	Jumanji (1995)	Adventure|Children|Fantasy
4	3	Grumpier Old Men (1995)	Comedy|Romance
5	4	Waiting to Exhale (1995)	Comedy|Drama|Romance
6	5	Father of the Bride Part II (1995)	Comedy
7	6	Heat (1995)	Action|Crime|Thriller
8	7	Sabrina (1995)	Comedy|Romance
9	8	Tom and Huck (1995)	Adventure|Children
10	9	Sudden Death (1995)	Action
11	10	GoldenEye (1995)	Action|Adventure|Thriller
12	11	American President, The (1995)	Comedy|Drama|Romance
13	12	Dracula: Dead and Loving It (1995)	Comedy|Horror
14	13	Balto (1995)	Adventure|Animation|Children
15	14	Nixon (1995)	Drama
16	15	Cutthroat Island (1995)	Action|Adventure|Romance
17	16	Casino (1995)	Crime|Drama
18	17	Sense and Sensibility (1995)	Drama|Romance
19	18	Four Rooms (1995)	Comedy
20	19	Ace Ventura: When Nature Calls (1995)	Comedy
21	20	Money Train (1995)	Action|Comedy|Crime|Drama|Thriller
22	21	Get Shorty (1995)	Comedy|Crime|Thriller
23	22	Copycat (1995)	Crime|Drama|Horror|Mystery|Thriller

	A	B	C
1	userId	movieId	rating
2	1	1	4
3	1	3	4
4	1	6	4
5	1	47	5
6	1	50	5
7	1	70	3
8	1	101	5
9	1	110	4
10	1	151	5
11	1	157	5
12	1	163	5
13	1	216	5
14	1	223	3
15	1	231	5
16	1	235	4
17	1	260	5
18	1	296	3
19	1	316	3
20	1	333	5
21	1	349	4
22	1	356	4
23	1	362	5

Two pivotal datasets, namely "movies.csv" and "ratings.csv," are loaded into Pandas DataFrames. This step is fundamental for several reasons:

- Foundation for Analysis: Loading datasets into DataFrames establishes the foundation for comprehensive movie recommendation analysis. It allows for seamless manipulation and exploration of the data, providing a clear overview of the dataset's structure and contents.
- Data Understanding: This process is crucial for understanding the characteristics of the datasets, including the types of information available, data types, and potential relationships between variables.
- Preparation for Modeling: Loading datasets is a prerequisite for building recommendation models. These models rely on accurate and up-to-date information to generate meaningful recommendations.

6.3. Statistical Analysis of Ratings

In this step, key statistics are computed to glean insights into the dataset's characteristics. These statistics include:

- Number of Ratings: The total count of ratings in the dataset provides an overview of the user engagement with the movies. It indicates the overall activity level within the system.
- Unique Movie IDs: Counting the distinct movie IDs reveals the diversity of movies available in the dataset. This metric is essential for understanding the range of content covered.
- Unique Users: The count of unique user IDs indicates the number of individual users contributing ratings. It is a crucial metric for evaluating user participation.
- Average Ratings per User and per Movie: Calculating the average ratings per user and per movie offers insights into user engagement and movie popularity. It helps identify trends and patterns in user behavior.

In [45]:
```
n_ratings = len(ratings)
n_movies = len(ratings['movieId'].unique())
n_users = len(ratings['userId'].unique())

print(f"Number of ratings: {n_ratings}")
print(f"Number of unique movieId's: {n_movies}")
print(f"Number of unique users: {n_users}")
print(f"Average ratings per user: {round(n_ratings/n_users, 2)}")
print(f"Average ratings per movie: {round(n_ratings/n_movies, 2)}")
```

```
Number of ratings: 100836
Number of unique movieId's: 9724
Number of unique users: 610
Average ratings per user: 165.3
Average ratings per movie: 10.37
```

6.4. User Rating Frequency

User rating frequency analysis involves computing user-specific statistics, specifically the total number of ratings submitted by each user. This step contributes to:

- Understanding User Engagement: Analyzing user rating frequency provides insights into how actively users are participating in the rating system. It helps identify power users and those who may be less engaged.
- Detecting Patterns: Patterns in user rating frequency can reveal trends, such as users who consistently rate movies or those who sporadically contribute. This information is valuable for understanding user behavior.

In [46]:
```
user_freq = ratings[['userId', 'movieId']].groupby(
    'userId').count().reset_index()
user_freq.columns = ['userId', 'n_ratings']
print(user_freq.head())
```

```
   userId  n_ratings
0       1        232
1       2         29
2       3         39
3       4        216
4       5         44
```

In [47]:

```
# Find Lowest and Highest rated movies:
mean_rating = ratings.groupby('movieId')[['rating']].mean()
# Lowest rated movies
lowest_rated = mean_rating['rating'].idxmin()
movies.loc[movies['movieId'] == lowest_rated]
# Highest rated movies
highest_rated = mean_rating['rating'].idxmax()
movies.loc[movies['movieId'] == highest_rated]
# show number of people who rated movies rated movie highest
ratings[ratings['movieId']==highest_rated]
# show number of people who rated movies rated movie lowest
ratings[ratings['movieId']==lowest_rated]

## the above movies has very low dataset. We will use bayesian average
movie_stats = ratings.groupby('movieId')[['rating']].agg(['count', 'mean'])
movie_stats.columns = movie_stats.columns.droplevel()
```

6.5. Movie Similarity Analysis

The implementation of the K-Nearest Neighbors (KNN) algorithm in this step underscores the system's ability to identify similar movies based on user behavior. The user-item matrix, representing user preferences for movies, serves as the foundation for this analysis. The KNN algorithm identifies comparable movies by measuring similarity using metrics like cosine similarity or Euclidean distance.

- User-Item Matrix: The user-item matrix encapsulates user ratings for different movies. Each row corresponds to a user, each column corresponds to a movie, and the matrix values represent user ratings. This structured representation allows the system to quantify user preferences.
- K-Nearest Neighbors (KNN): The KNN algorithm is employed to find movies with similar user appeal. By considering the user-item matrix, the algorithm identifies movies that share characteristics with a target movie, effectively recommending movies based on their similarity to the user's preferences.

In [48]:

```
# Now, we create user-item matrix using scipy csr matrix
from scipy.sparse import csr_matrix

def create_matrix(df):
    N = len(df['userId'].unique())
    M = len(df['movieId'].unique())

    # Map Ids to indices
    user_mapper = dict(zip(np.unique(df["userId"]), list(range(N))))
    movie_mapper = dict(zip(np.unique(df["movieId"]), list(range(M))))

    # Map indices to IDs
    user_inv_mapper = dict(zip(list(range(N)), np.unique(df["userId"])))
    movie_inv_mapper = dict(zip(list(range(M)), np.unique(df["movieId"])))

    user_index = [user_mapper[i] for i in df['userId']]
    movie_index = [movie_mapper[i] for i in df['movieId']]

    X = csr_matrix((df["rating"], (movie_index, user_index)), shape=(M, N))

    return X, user_mapper, movie_mapper, user_inv_mapper, movie_inv_mapper

X, user_mapper, movie_mapper, user_inv_mapper, movie_inv_mapper = create_matrix(ratings)
```

6.6.1 Movie Recommendation Based on Movie Preferences

Movie recommendation based on movie preferences involves leveraging collaborative filtering techniques to provide personalized movie suggestions to users. Collaborative filtering relies on the preferences and behaviors of users to identify patterns and recommend items that similar users have liked. In the context of movie recommendations, this approach aims to suggest films based on the preferences of users who share similar tastes.

```
def find_similar_movies(movie_id, X, k, metric='cosine', show_distance=False):

    neighbour_ids = []

    movie_ind = movie_mapper[movie_id]
    movie_vec = X[movie_ind]
    k+=1
    kNN = sklearn.neighbors.NearestNeighbors(n_neighbors=k, algorithm="brute", metric=metric)
    kNN.fit(X)
    movie_vec = movie_vec.reshape(1,-1)
    neighbour = kNN.kneighbors(movie_vec, return_distance=show_distance)
    for i in range(0,k):
        n = neighbour.item(i)
        neighbour_ids.append(movie_inv_mapper[n])
    neighbour_ids.pop(0)
    return neighbour_ids


movie_titles = dict(zip(movies['movieId'], movies['title']))
```

6.6.2 Movie Recommendation Based on User Preferences

The creation of a recommendation function showcases the practical application of collaborative filtering. This function recommends movies based on a user's highest-rated movie, integrating the KNN algorithm. The process involves:

- User's Highest-Rated Movie: The function identifies the movie that has received the highest rating from the user. This movie serves as a reference point for recommending similar content.

- KNN Integration: Leveraging the KNN algorithm, the function identifies movies similar to the user's highest-rated movie. By recommending movies that align with the user's preferences, the system provides a personalized and relevant movie list. This functionality underscores the collaborative nature of the recommendation system, tailoring suggestions to individual user behavior.

In [50]:
```
def recommend_movies_for_user(user_id, X, user_mapper, movie_mapper, movie_inv_mapper, k=10):
    df1 = ratings[ratings['userId'] == user_id]

    if df1.empty:
        print(f"User with ID {user_id} does not exist.")
        return

    movie_id = df1[df1['rating'] == max(df1['rating'])]['movieId'].iloc[0]

    movie_titles = dict(zip(movies['movieId'], movies['title']))

    similar_ids = find_similar_movies(movie_id, X, k)
    movie_title = movie_titles.get(movie_id, "Movie not found")

    if movie_title == "Movie not found":
        print(f"Movie with ID {movie_id} not found.")
        return

    print(f"Since you watched {movie_title}, you might also like:")
    for i in similar_ids:
        print(movie_titles.get(i, "Movie not found"))
```

6.7. Recommendation Implementation

The practical implementation of the recommendation system involves showcasing examples of movie recommendations for specific user IDs. This step brings together the various components developed, including data loading, statistical analysis, user engagement metrics, and collaborative filtering.

By presenting real-world examples, the system's effectiveness and ability to deliver meaningful and personalized movie recommendations are demonstrated.

6.8. Result and Analysis

The results section evaluates the performance and effectiveness of the movie recommendation system. It involves a thorough analysis of processed data, the accuracy of the KNN-based movie similarity, and the relevance of recommendations based on user preferences. Key aspects include:

Data Accuracy:

Assessing the accuracy of processed data to ensure the reliability of subsequent analyses and recommendations.

KNN Accuracy:

Evaluating the precision of the KNN algorithm in identifying similar movies. This includes assessing the chosen similarity metric and the overall effectiveness of the collaborative filtering approach.

Recommendation Relevance:

Analyzing the relevance of recommendations based on user preferences. This involves user feedback and subjective assessment to determine if the system is providing valuable and personalized suggestions.

```
movie_id = 9

similar_ids = find_similar_movies(movie_id, X, k=10)
movie_title = movie_titles[movie_id]

print(f"Since you watched {movie_title}")
for i in similar_ids:
    print(movie_titles[i])
```

```
Since you watched Sudden Death (1995)
Theodore Rex (1995)
Eraser (1996)
Bed of Roses (1996)
Bloodsport 2 (a.k.a. Bloodsport II: The Next Kumite) (1996)
Relic, The (1997)
Juror, The (1996)
Spy Hard (1996)
Barb Wire (1996)
Eddie (1996)
Striptease (1996)
```

```
In [58]: user_id = 6 # Replace with the desired user ID
         recommend_movies_for_user(user_id, X, user_mapper, movie_mapper, movie_inv_mapper, k=10)

         Since you watched Grumpier Old Men (1995), you might also like:
         Grumpy Old Men (1993)
         Striptease (1996)
         Nutty Professor, The (1996)
         Twister (1996)
         Father of the Bride Part II (1995)
         Broken Arrow (1996)
         Bio-Dome (1996)
         Truth About Cats & Dogs, The (1996)
         Sabrina (1995)
         Birdcage, The (1996)
```

6.9. Evaluation: Precision and Recall

The evaluation section focuses on assessing the performance of the recommendation system using precision and recall metrics, providing a detailed analysis of its effectiveness.

Precision:

Precision measures the accuracy of the recommendation system in providing relevant suggestions to users. It is calculated as the ratio of relevant items recommended to the total number of items recommended. High precision indicates that the system is adept at suggesting items that align with user preferences.

Recall:

Recall measures the system's ability to capture all relevant items within the dataset. It is calculated as the ratio of relevant items recommended to the total number of relevant items in the dataset. A high recall value suggests that the system can identify a substantial portion of items that users would find relevant.

```
    for threshold in thresholds:
        y_true = []
        y_pred = []

        for user_id in ratings['userId'].unique():
            if user_id in user_mapper:
                true_movie_ids = ratings[(ratings['userId'] == user_id) & (ratings['rating'] >= threshold)]['movieId'].tolist
                true_movie_indices = [movie_mapper[movie_id] for movie_id in true_movie_ids]
                true_labels = [1 if i in true_movie_indices else 0 for i in range(X.shape[0])]

                user_ratings = ratings[ratings['userId'] == user_id]
                if not user_ratings.empty:
                    max_rated_movie_id = user_ratings[user_ratings['rating'] == max(user_ratings['rating'])]['movieId'].iloc[
                    similar_movie_ids = find_similar_movies(max_rated_movie_id, X, k=10)
                    similar_movie_indices = [movie_mapper[movie_id] for movie_id in similar_movie_ids]

                    pred_labels = [1 if i in similar_movie_indices else 0 for i in range(X.shape[0])]

                    y_true.extend(true_labels)
                    y_pred.extend(pred_labels)

        precision = precision_score(y_true, y_pred)
        recall = recall_score(y_true, y_pred)

        print(f"Threshold: {threshold}, Precision: {precision:.2f}, Recall: {recall:.2f}")

# Example usage:
thresholds_to_try = [3.0, 3.5, 4.0, 4.5, 5.0]
```

Detailed Evaluation Results:

The report presents a granular analysis of precision and recall, offering insights into the system's strengths and areas for improvement. It may include performance metrics for different user segments, item categories, or time periods, providing a comprehensive understanding of the recommendation system's efficacy.

```
Threshold: 3.0, Precision: 0.41, Recall: 0.03
Threshold: 3.5, Precision: 0.35, Recall: 0.03
Threshold: 4.0, Precision: 0.32, Recall: 0.04
Threshold: 4.5, Precision: 0.19, Recall: 0.05
Threshold: 5.0, Precision: 0.13, Recall: 0.06
```

7. **CONCLUSION**

The Python Movie Recommendation System shows how recommendation systems can help with too much information and make user experiences better. It uses machine learning to give personalized suggestions, making users happier and helping businesses grow.

Key Emphases in Conclusion:

- Versatility: The implemented functions for data analysis, user-specific insights, personalized recommendations, and collaborative filtering showcase the versatility of the recommendation system.
- User-Centric Approach: The system prioritizes user preferences, delivering recommendations that align with individual tastes and behaviors.
- Ongoing Potential for Improvement: The report emphasizes that the recommendation system is not static. It highlights the potential for continuous improvement, suggesting that user feedback and evolving data can further enhance the system's accuracy and relevance.

In short, the report says that recommendation systems are useful for dealing with too much information. They give users content that matters to them and help businesses engage more with users, leading to success.