Movie Recommender System using Neural Collaborative Filtering and Deep Learning

Anuj Patel (amp10162) - *Electrical Engineering*
Srushti Jagtap (sj4182) - *Computer Engineering*
Harshal Kulkarni (hsk8171) - *Computer Engineering*

1 Introduction

The exponential growth of digital media has led to an overwhelming amount of content on streaming platforms, making recommendation systems essential for guiding users to movies that match their preferences. Traditional methods like collaborative filtering struggle to capture complex, nonlinear relationships as user-item interactions scale.

Deep learning, particularly Neural Collaborative Filtering (NCF), has proven effective in overcoming these limitations. NCF uses neural networks to learn latent factors from user and movie embeddings, capturing intricate patterns in user behavior. This project develops a movie recommendation system using the MovieLens 1M dataset, which contains over a million ratings from 6,040 users on 3,952 movies, to provide more personalized suggestions.

The goal is to improve upon traditional collaborative filtering methods and ensure scalability by leveraging big data frameworks like Apache Spark for distributed computations. While the system focuses on NCF to learn user-movie preferences, future work could explore additional techniques to enhance performance further.

2 Problem Statement & Objectives

Users are often overwhelmed by the large number of movie choices available on streaming platforms, making it difficult to find content that matches their preferences. Traditional collaborative filtering methods that manually compute user similarities are not sufficient for capturing complex patterns in user behavior. Additionally, as user-item interaction matrices become large and sparse, scalability and accuracy issues arise. To solve this, we will:

1. Develop a deep learning-based movie recommendation system using the MovieLens 1M dataset.
2. Learn complex user-movie relationships using Neural Collaborative Filtering (NCF) and Autoencoder models.
3. Generate personalized movie recommendations based on learned latent factors from user and movie embeddings.
4. Evaluate the model's performance using standard recommendation system metrics such as RMSE, Precision@K, and Recall@K.

5. Ensure scalability by incorporating big data frameworks (e.g., Apache Spark) and enabling the system to handle large-scale datasets efficiently.

3 Background

Recommendation systems predict a user's preference for items (movies) based on historical interaction data. The goal is to suggest items that the user is likely to be interested in, without needing explicit feedback for every possible item. Common types of recommendation systems include **Collaborative Filtering** (CF), **Content-Based Filtering**, and **Hybrid Approaches**. Among these, collaborative filtering has become widely used for recommending items based on user-item interactions.

3.1 Collaborative Filtering (CF)

Collaborative Filtering methods use a user-item interaction matrix, where rows represent users, columns represent items (movies), and entries in the matrix represent ratings or preferences. CF can be categorized into two main types:

- **User-based Collaborative Filtering**: Predicts ratings by finding users similar to the target user based on their rating patterns.
- **Item-based Collaborative Filtering**: Predicts ratings by finding items similar to the target item based on user ratings.

However, traditional CF methods often struggle to capture complex, nonlinear patterns, especially when the user-item interaction matrix is sparse. This is where **Neural Collaborative Filtering** (NCF) offers a more efficient solution.

3.2 Neural Collaborative Filtering (NCF)

NCF leverages deep learning techniques to learn complex relationships between users and items. Instead of relying on simple similarity measures, NCF learns latent factors for users and items, embedding them into continuous vector spaces. These embeddings are then used as input to a neural network that predicts ratings.

The NCF architecture can be described mathematically as follows:

- **Embedding Layer**: Users and items are represented by latent vectors $\mathbf{u}_i$ and $\mathbf{m}_j$, respectively. These vectors are learned during training.
- **Interaction Layer**: The embeddings of users and items are combined, typically by concatenating them, and then transformed by a neural network.
- **Output Layer**: The neural network outputs the predicted rating $\hat{r}_{ij}$ for user i and movie j.

The predicted rating $\hat{r}_{ij}$ is given by the function:

$$\hat{r}_{ij} = f(\mathbf{u}_i, \mathbf{m}_j; \theta)$$

where:

- $\mathbf{u}_i \in R^d$ is the embedding of user i,
- $\mathbf{m}_j \in R^d$ is the embedding of movie j,
- $f(\cdot, \cdot)$ is a neural network function parameterized by θ,
- d is the dimensionality of the embeddings.

The objective is to minimize the difference between the predicted rating and the actual rating r_{ij} using the Mean Squared Error (MSE) loss function:

$$L = \frac{1}{N} \sum_{i,j} (r_{ij} - \hat{r}_{ij})^2$$

where N is the number of observed ratings in the training set. Regularization terms are added to prevent overfitting:

$$L_{\text{reg}} = \lambda \left(\sum_i \|\mathbf{u}_i\|^2 + \sum_j \|\mathbf{m}_j\|^2 \right)$$

where λ is the regularization parameter.

3.3 Scalability with Big Data Frameworks

To handle large datasets, such as the MovieLens 1M dataset, distributed computing frameworks like **Apache Spark** can be used. Spark allows for efficient processing of large-scale data and distributed model training, making it ideal for large-scale recommendation systems.

4 Methodology

In this section, we describe the methodology used to develop the movie recommendation system using Item-based Collaborative Filtering, based on the MovieLens-1M dataset. The system utilizes a Neural Collaborative Filtering (NCF) approach to recommend movies to users.

4.1 Data Preprocessing

The dataset consists of three main components: movies, users, and ratings. The `movies.dat` file contains movie metadata including the MovieID, Title, and Genres. The `users.dat` file contains user information such as UserID, Gender, Age, Occupation, and Zip-code. The `ratings.dat` file contains user ratings for movies, consisting of UserID, MovieID, Rating, and Timestamp.

The ratings data is loaded into a Pandas DataFrame, and additional data visualizations are performed to explore movie genres and ratings distribution.

This includes:

- A bar chart representing the distribution of movie genres.

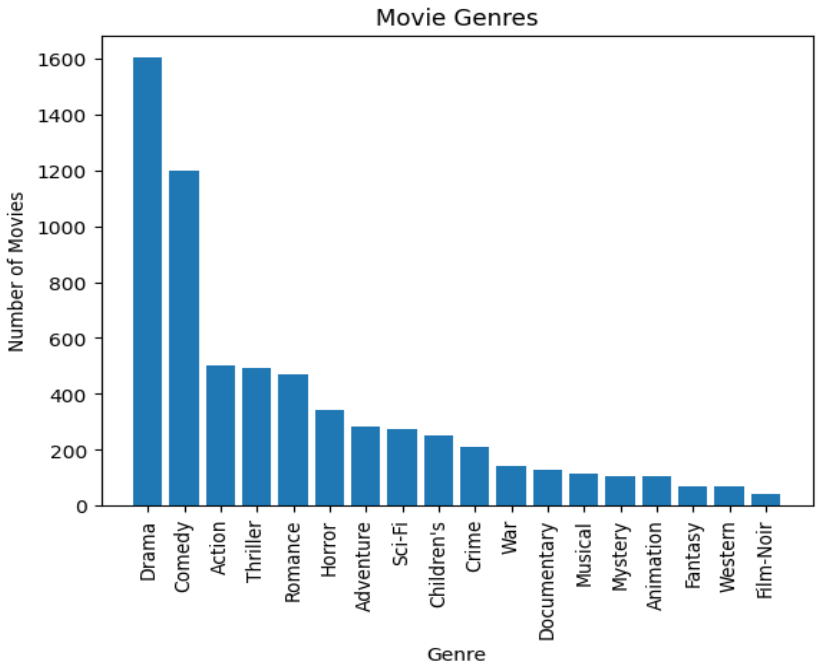

Figure 1: A bar chart representing the distribution of movie genres

- A word cloud based on movie genres.

Figure 2: A word cloud based on movie genres

- A box plot showing the distribution of ratings by genre.

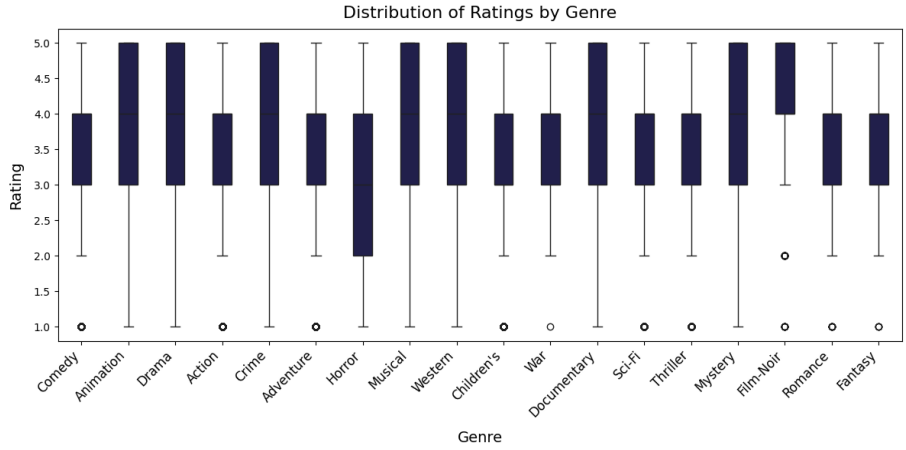

Figure 3: A box plot showing the distribution of ratings by genre

- A word cloud based on movie titles.

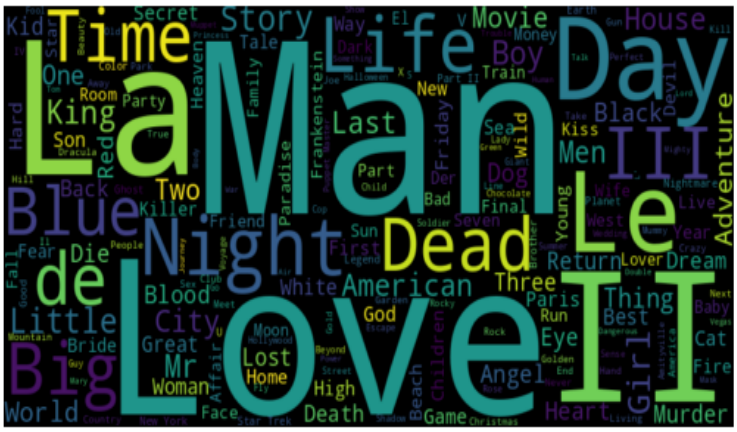

Figure 4: A word cloud based on movie titles

- Top 10 Movies with Maximum Number of User Ratings as 5.

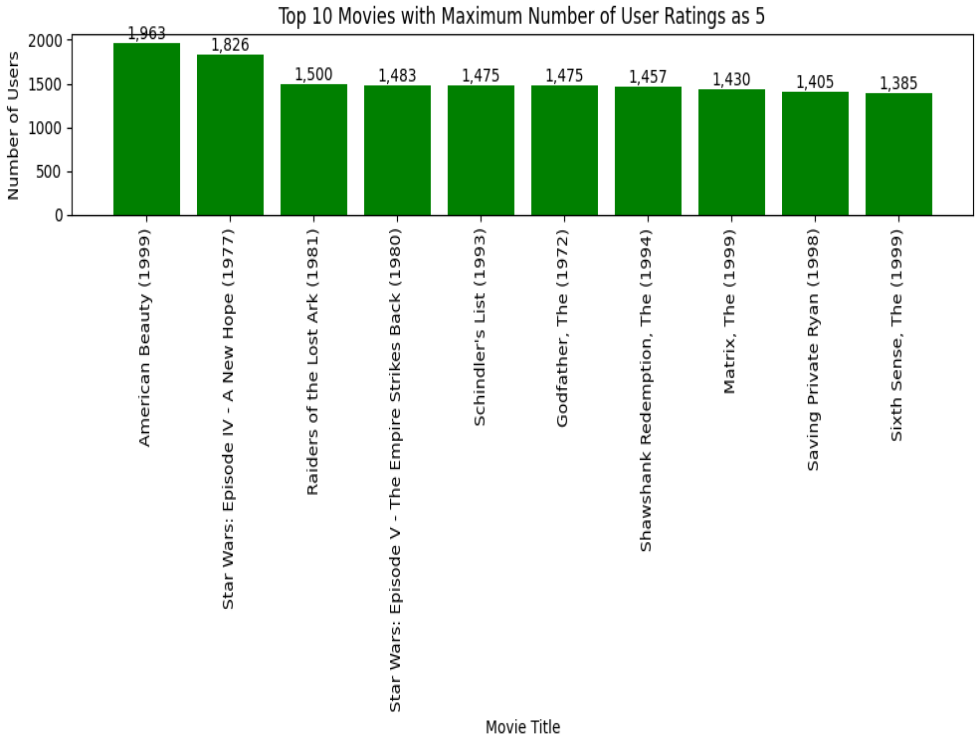

Figure 5: Top 10 Movies with Maximum Number of User Ratings as 5

- A bar graph showing count of ratings by genre.

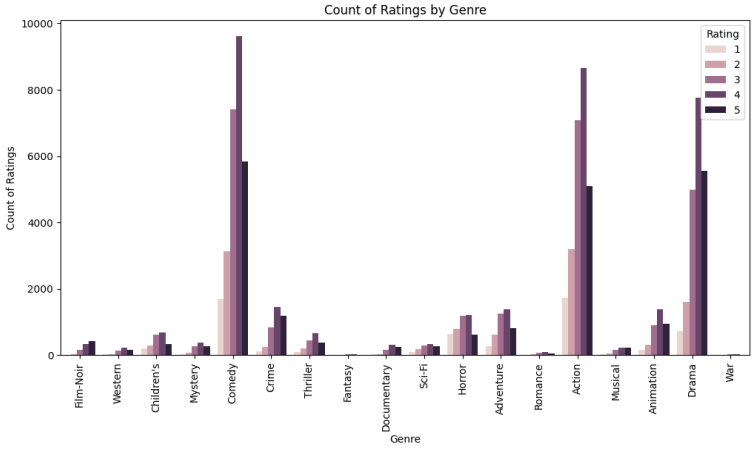

Figure 6: A bar graph showing count of ratings by genre

4.2 Data Splitting and Generation of Negative Samples

The dataset is split into training and testing sets. The most recent rating for each user is reserved for the test set, while all other ratings are used for training. This ensures that the model is tested on data that the user has not interacted with during training.

For the training data, negative samples are generated by introducing a 4:1 ratio of negative to positive samples. A negative sample refers to a movie that the user has not rated, while a positive sample corresponds to a movie that the user has rated. To generate these negative samples, we randomly select movies that the user has not interacted with and assign them a label of 0, indicating a negative interaction. This process helps mitigate the sparsity of the ratings data, where most users interact with only a small fraction of items.

Negative sampling is essential due to the inherent sparsity of the ratings data, as most users rate only a limited number of movies. For each positive user-item interaction, four negative samples are created by randomly selecting unrated movies for that user. These negative samples, assigned a label of 0, represent non-interactions and are used to train the model to differentiate between movies the user has rated and those they have not interacted with. The 4:1 ratio of negative to positive samples helps balance the training data and enables the model to effectively learn from both positive and negative interactions.

4.3 Collaborative Filtering

Collaborative filtering is a technique that makes recommendations based on the past interactions between users and items (in this case, movies). The system attempts to predict user preferences based on the behavior of similar users.

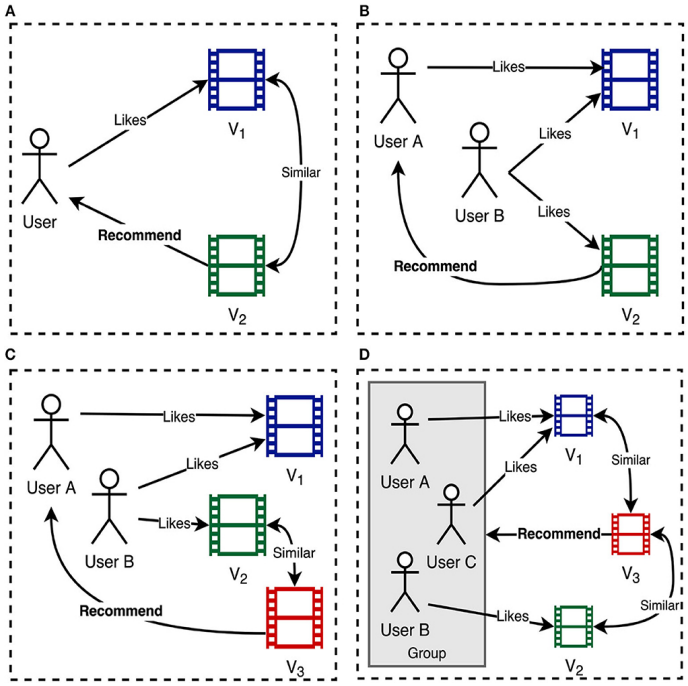

Figure 7: Image showing how collaborative filtering works.

4.4 Neural Collaborative Filtering (NCF) Model

NCF extends traditional collaborative filtering by integrating neural networks to capture complex interactions between users and items.

4.4.1 Mathematical Formulation

Let u denote a user and i denote an item (movie).

User and Item Embeddings: Each user u is represented by an embedding vector $\mathbf{e}_u \in R^d$, where d is the dimensionality of the embedding. Similarly, each item i (movie) is represented by an embedding vector $\mathbf{e}_i \in R^d$.

$$\mathbf{e}_u = \text{Embedding}(u) \quad \text{for each user} \quad u$$
$$\mathbf{e}_i = \text{Embedding}(i) \quad \text{for each item} \quad i$$

These embeddings are learned during the training process.

Concatenation of Embeddings: The user and item embeddings are concatenated to form a single vector that represents the interaction between the user and the item:

$$\mathbf{v}_{ui} = [\mathbf{e}_u, \mathbf{e}_i] \quad \text{where} \quad \mathbf{v}_{ui} \in R^{2d}$$

Neural Network Layers: The concatenated vector $\mathbf{v}_{ui}$ is passed through several fully connected layers, which capture higher-order interactions between the user and item embeddings. These layers are as follows:

- First hidden layer:

$$\mathbf{h}_1 = \text{ReLU}(\mathbf{W}_1\mathbf{v}_{ui} + \mathbf{b}_1)$$

- Second hidden layer:

$$\mathbf{h}_2 = \text{ReLU}(\mathbf{W}_2\mathbf{h}_1 + \mathbf{b}_2)$$

Here, $\mathbf{W}_1, \mathbf{W}_2$ are the weight matrices and $\mathbf{b}_1, \mathbf{b}_2$ are the bias vectors.

Output Layer: The final output of the neural network is passed through a sigmoid function to predict the probability $\hat{r}_{ui}$ that user u will rate item i highly (close to 1 indicates a higher rating):

$$\hat{r}_{ui} = \sigma(\mathbf{W}_3\mathbf{h}_2 + \mathbf{b}_3)$$

where σ is the sigmoid activation function:

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

4.4.2 Loss Function

The objective is to minimize the binary cross-entropy loss between the predicted ratings $\hat{r}_{ui}$ and the actual ratings r_{ui}, where:

$$\mathcal{L}(r_{ui}, \hat{r}_{ui}) = -[r_{ui} \log(\hat{r}_{ui}) + (1 - r_{ui}) \log(1 - \hat{r}_{ui})]$$

The total loss is the average of the individual losses over all user-item interactions in the training set:

$$\mathcal{L} = \frac{1}{N} \sum_{(u,i)} \mathcal{L}(r_{ui}, \hat{r}_{ui})$$

where N is the number of training samples.

4.4.3 Training and Optimization

The model is trained using the Adam optimizer, which updates the model parameters by minimizing the binary cross-entropy loss. Stochastic gradient descent (SGD) or other optimization algorithms can also be used for this purpose.

4.5 Evaluation Metrics

The model's performance is evaluated using the Hit Ratio at 10, which measures whether the true item (movie) is among the top 10 recommended items. If the true item appears in the top 10, it is counted as a hit.

$$\text{Hit Ratio @ 10} = \frac{\text{Number of hits}}{\text{Total number of test cases}} \times 100$$

A higher hit ratio indicates better model performance.

4.5.1 Reasons for Choosing Hit Rate Over Precision@k and Recall@k

1. Focus on User Perspective

Hit Rate (HR) emphasizes whether at least one relevant item appears in the top-k, aligning closely with user satisfaction. For many real-world applications, users are primarily interested in whether the system can recommend a few highly relevant items, rather than ensuring all relevant items are included. On the other hand, Precision@k and Recall@k, while valuable, might overly focus on ranking all relevant items, which is less critical for scenarios where users only interact with a subset of recommendations.

2. Practical Evaluation of Top-k Recommendations

Hit Rate reflects practical scenarios where users are likely to explore only the top-k recommendations. Precision@k demands a higher proportion of relevant items in the top-k, which might not align with the goal of ensuring that at least one relevant item is included.

3. Simplification for Interpretability

Hit Rate provides a binary outcome (success or failure) for each user-item interaction in the top-k, making it straightforward to compute and interpret. In contrast, Precision@k and Recall@k require additional consideration of how many items are relevant overall or within the top-k, which could complicate evaluation without significantly improving insights for this project's goal.

4. Alignment with Dataset Characteristics

The MovieLens 1M dataset might include sparse user-item interactions, where evaluating the presence of at least one relevant item (Hit Rate) is more meaningful than demanding precision or recall over multiple items.

5. Cold-Start and Long-Tail Recommendations

Hit Rate works well in cold-start scenarios or for long-tail items, where the focus is on ensuring a hit rather than achieving a high precision score, which can penalize models for including diverse or less-predictable recommendations.

By using Hit Rate, the evaluation emphasizes user-centric success, avoids the complexity of balancing multiple metrics, and stays aligned with the project's primary objective: delivering personalized, scalable recommendations with relevance at the top-k level.

5 Output

Output screenshot from the Colab file

Figure 8: Output screenshot from the Colab file

Output on the Streamlit Dashboard

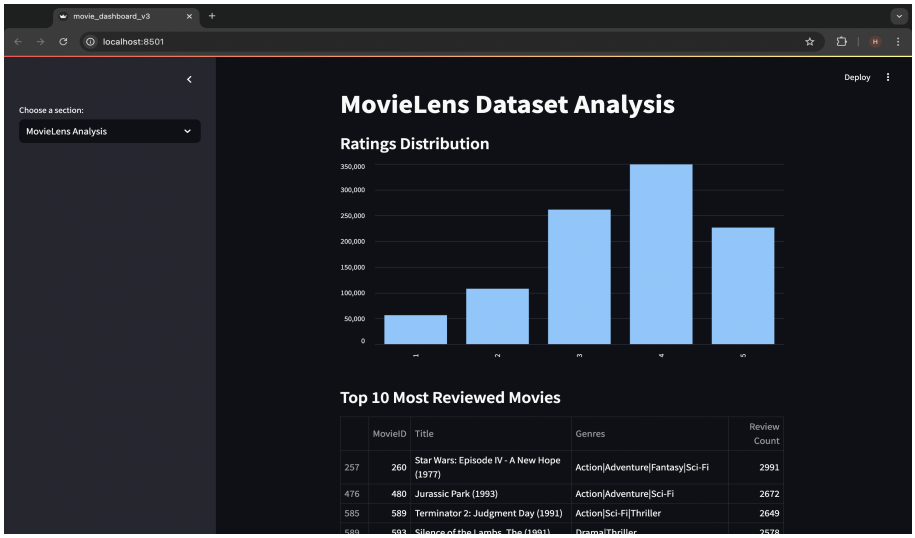

Figure 9: MovieLens Dataset Analysis Dashboard

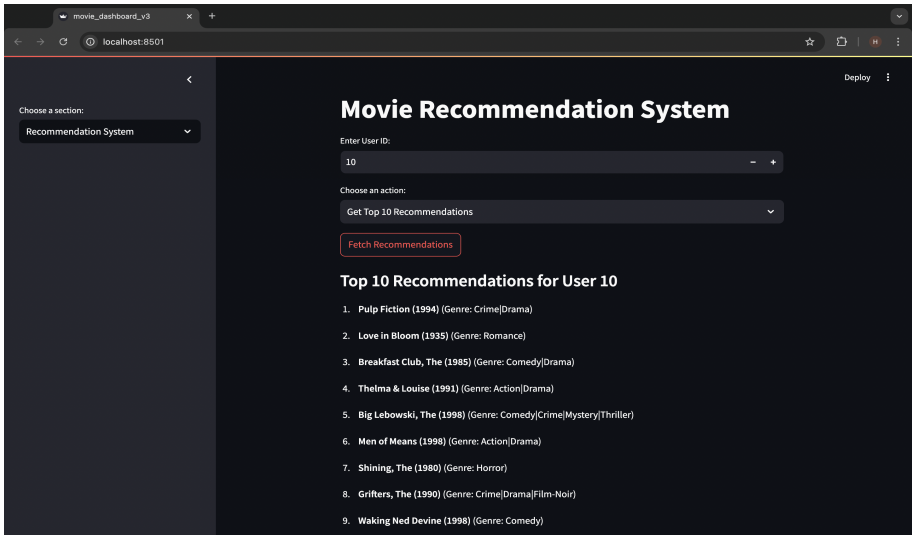

Figure 10: Movie Recommendation System Dashboard

6 Results

6.1 Model Performance Evaluation

The movie recommendation system effectively personalized user experiences by leveraging advanced machine learning and deep learning models. Key performance metrics included:

- **Hit Ratio (HR):** Achieved a value of **0.5**, indicating that 50% of the top recommendations matched user preferences.

This result demonstrates the model's ability to generate relevant suggestions and meet user expectations.

6.2 Effectiveness of Neural Collaborative Filtering (NCF)

The implementation of the NCF model captured intricate user-item interaction patterns, which were instrumental in achieving accurate recommendations. The model's architec-

ture allowed for learning both linear and nonlinear relationships, significantly improving recommendation quality compared to traditional methods.

6.3 Scalability and Big Data Integration

The integration of **Apache Spark** enabled the system to handle the **MovieLens 1M dataset** efficiently, ensuring the solution was robust and scalable for larger datasets. This underscores the system's readiness for real-world applications with substantial data requirements.

6.4 Insights from Metrics and Observations

The Hit Ratio metric provided valuable insights into model performance, highlighting areas of strength and limitations:

- **Strengths:** The model successfully captured user preferences for frequently rated movies.
- **Limitations:** Cold-start issues persisted for new users or items, suggesting the need for hybrid approaches incorporating content-based features.

6.5 User Experience and Real-World Applicability

The system's ability to generate relevant recommendations and process large-scale datasets within a reasonable timeframe demonstrates its applicability for deployment in real-world environments.

The results validate the effectiveness of the NCF model combined with big data frameworks in building a robust and scalable recommendation system. These insights lay a foundation for further optimization and extension of the system to incorporate additional features and models.

7 Conclusion

The recommendation system predicts user preferences based on learned embeddings for users and items. By capturing higher-order interactions between these embeddings, the NCF model is able to make accurate predictions. The model is trained using a loss function that minimizes the error between the predicted and actual ratings, and its performance is evaluated using the Hit Ratio at 10.

8 Future Work

Future work could explore multimodal data (e.g., user reviews or movie trailers), hybrid models, and reinforcement learning approaches to further improve recommendation accuracy and user satisfaction.
One can also capture the watch duration of the previous movie for a user and build upon that to generate recommendations.

References

[1] https://medium.com/data-science-in-your-pocket/recommendation-systems-using-neural-collaborative-filtering-ncf-explained-with-codes-21a97e48a2f7

[2] https://librecommender.readthedocs.io/en/latest/tutorial.html

[3] https://www.freecodecamp.org/news/how-to-build-a-movie-recommendation-system-based-on-collaborative-filtering/

[4] https://www.analyticsvidhya.com/blog/2020/11/create-your-own-movie-movie-recommendation-system/

9 Github Link

https://github.com/HARSHALK2598/BigDataProject