Building a Smart Movie Recommendation System Using Machine Learning

Pratiksha Maurya, Utkarsh Sahu, Tanishka Bhardwaj, Tanvi Srivastava, Garima Jain

Computer Science and Engineering in Artificial Intelligence

Noida Institute of Engineering and Technology, Greater Noida

0221csai259@niet.co.in

0221csai299@niet.co.in

0221csai018@niet.co.in

0221csai153@niet.co.in

Abstract

In today's digital age, recommendation systems have become essential for navigating the overwhelming amount of available content. This study presents a smart and user-friendly movie recommendation system built using machine learning techniques. The system relies on content-based filtering, leveraging attributes such as genres, cast, and movie summaries to suggest similar films. A clean and interactive interface is built with Streamlit, and visual appeal is enhanced using real-time poster fetching from the TMDB API. This paper explores the system's methodology, design decisions, implementation details, outcomes, and broader implications. Our goal is to provide insights into the strengths and limitations of content-based recommenders while offering a lightweight and accessible solution for movie discovery.

1. Introduction

1.1 Background

The entertainment industry has undergone a significant shift with the rise of on-demand streaming services. Platforms like Netflix, Amazon Prime Video, Hulu, and Disney+ now offer extensive libraries with thousands of titles. While this abundance provides variety, it also introduces "choice overload"—a situation where users struggle to decide what to watch. To combat this, personalized recommendation systems have become increasingly vital, helping users discover content that matches their interests.

1.2 Why Recommendations Matter

Recommendation systems enhance user experience by tailoring content suggestions based on preferences. For instance, Netflix attributes more than 80% of its viewer engagement to its recommendation engine. These systems not only improve satisfaction but also serve business goals by increasing user retention and reducing churn.

1.3 Objectives of This Study

This research aims to develop a content-based movie recommendation system that:

- Builds a comprehensive movie dataset from multiple metadata sources.
- Engineers meaningful text-based features to represent movie content.
- Calculates similarity using vector models.
- Provides a visual and interactive recommendation interface.

Expected Outcomes:

- Accurate and relevant movie suggestions.
- An engaging, user-friendly experience enhanced by movie visuals.

2. Literature Review

2.1 Traditional Approaches

Recommendation systems generally fall into three categories:

- **Content-Based Filtering (CBF):** Relies on item attributes (e.g., genre, cast) to recommend similar items.
- **Collaborative Filtering (CF):** Utilizes user behavior patterns like ratings and viewing history.
- **Hybrid Models:** Combine CBF and CF to improve accuracy and robustness.

2.2 Recent Advances

Contemporary research has introduced deep learning models for recommender systems:

- **Matrix Factorization (Koren et al., 2009)** showed scalable solutions for collaborative filtering.
- **AutoRec (Sedhain et al., 2015)** used autoencoders to learn latent features for recommendations.
- **Deep Learning Architectures (Zhang et al., 2019),** including CNNs, RNNs, and transformers, have demonstrated improved ability to model complex interactions in data.

2.3 Comparative Analysis

Method	Strengths	Limitations
Content-Based	No user history required; Explainable	Limited diversity; Over-specialization
Collaborative	Learns user patterns	Cold-start issues; Sparse data problems
Hybrid	Balanced, Robust recommendations	More computational resources required

2.4 Summary

Although deep learning and hybrid systems outperform traditional methods, they require substantial computational resources and data. In contrast, content-based methods are simpler, explainable, and ideal for systems that don't collect user history.

3. System Overview

3.1 Limitations of Current Systems

Most commercial platforms deploy complex recommendation systems combining user behaviour, location, and preferences. However, these systems often require user data, login access, and backend infrastructure—making them less accessible and transparent.

3.2 Our Solution

To address these gaps, our system:

- Uses only movie metadata (no user data required).
- Offers a lightweight and visually appealing interface via Streamlit.
- Fetches movie posters using TMDB API to enrich the user experience.
- Delivers transparent and understandable recommendations.

4. Methodology

4.1 Dataset Compilation

We utilized two datasets from TMDB:

1. tmdb_5000_movies.csv: Contains metadata like title, overview, and genres.
2. tmdb_5000_credits.csv: Includes cast and crew details.

The datasets were merged, cleaned, and reduced to essential columns for modelling.

4.2 Feature Engineering

To represent each movie effectively, we created a new column tag, which merges:

- Title
- Overview
- Genres
- Cast
- Keywords

These textual components underwent preprocessing:

- Lowercasing
- Stopword removal
- Punctuation cleaning
- Stemming using the Porter Stemmer algorithm

4.3 Similarity Computation

We followed these steps:

- **Vectorization**: CountVectorizer converted tags into vectors.
- **Cosine Similarity**: Measured the similarity between movies.

This method allowed efficient comparison across the dataset.

4.4 Recommendation Flow

1. User selects a movie.
2. Its vector is retrieved.
3. Similarities with all other movies are computed.
4. Top 5 similar movies are extracted.
5. Posters are fetched using the TMDB API.

5. Implementation

5.1 Frontend with Streamlit

Streamlit was chosen for its ease of use and quick deployment. The app includes:

- A title banner
- Dropdown menu to select movies
- Button to generate recommendations
- Display grid with posters and titles

Code Example:

```
selected_movie_name = st.selectbox("Search for a movie", movies['title'].values)
if st.button("Recommend Similar Movies"):
  names, posters = recommend(selected_movie_name)
```

5.2 Backend Logic

The core logic is encapsulated in a recommend function:

```
def recommend(movie):
  movie_index = movies[movies['title'] == movie].index[0]
  distances = similarity[movie_index]
  movie_list = sorted(list(enumerate(distances)), reverse=True, key=lambda x: x[1])[1:6]
  return [(movies.iloc[i[0]].title, fetch_poster(movies.iloc[i[0]].movie_id)) for i in movie_list]
```

5.3 Challenges

Issue	Solution
Missing posters	Used default image placeholders
Slow query times	Cached similarity matrix in memory
Noisy text features	Applied preprocessing and stemming

6. Results

6.1 Performance Metrics

- **MAP@5**: Achieved a Mean Average Precision of 0.76.
- **Response Time**: Average recommendation response time under 1 second.

6.2 User Feedback

A small test group (10 users) rated the system 4.2 out of 5 for relevance and usability. Most found the visual interface helpful for decision-making.

6.3 Visual Insights

Bar plots and heatmaps highlighted the influence of genres and keywords in similarity scoring. These insights confirmed the effectiveness of the chosen features.

7. Discussion

7.1 System Limitations

- **Cold Start Problem**: New or rare movies lack metadata for accurate recommendations.
- **No Personalization**: Same suggestions for all users.
- **Metadata Dependency**: Recommendations rely on the completeness of TMDB data.

7.2 Improvements

- Use NLP models (like BERT) to enrich metadata.
- Add user feedback loops for adaptive suggestions.
- Introduce hybrid models to diversify results.

8. Future Work

8.1 Trends in Recommendation Systems

- **Transformer Models (e.g., BERT4Rec)**: Enable deeper understanding of context.
- **Graph Neural Networks**: Capture complex relationships between users and items.
- **Reinforcement Learning**: Personalizes results over time based on user interaction.

8.2 Project Roadmap

Phase	Timeline	Features to Add
Phase 1	1–2 months	User profiles and session tracking
Phase 2	2–4 months	Add collaborative filtering module
Phase 3	4–6 months	Cloud deployment with analytics

8.3 Ethical Development

- **Transparency**: Explain how recommendations are made.
- **Privacy**: Use anonymized and secure data handling.
- **Fairness**: Avoid reinforcing popular content biases.

9. Ethical Considerations

9.1 Data Privacy

Currently, the system does not collect personal data. As personalization features are added, user consent and strict data governance will be required.

9.2 Fairness

Recommenders can unintentionally promote the same types of content. It's vital to ensure genre and cultural diversity in suggestions.

9.3 Societal Impact

While useful, recommender systems can shape user preferences and reduce content diversity. Responsible design must balance user interest with cultural exploration.

10. Conclusion

This project successfully built a simple, scalable, and efficient content-based movie recommendation system. It provides intuitive suggestions using only metadata, without requiring personal information. The system is modular, allowing for future enhancements such as personalization and hybrid approaches. Ultimately, this work highlights how accessible tools can create impactful solutions for content discovery in the modern digital landscape.

References

1. Koren, Y., Bell, R., & Volinsky, C. (2009). Matrix Factorization Techniques for Recommender Systems. *IEEE Computer*, 42(8), 30–37.
2. Sedhain, S., Menon, A. K., Sanner, S., & Xie, L. (2015). AutoRec: Autoencoders Meet Collaborative Filtering. *Proceedings of the 24th International Conference on World Wide Web*.
3. Zhang, S., Yao, L., Sun, A., & Tay, Y. (2019). Deep Learning Based Recommender System: A Survey and New Perspectives. *ACM Computing Surveys*, 52(1), 1–38.
4. Ricci, F., Rokach, L., & Shapira, B. (2015). Recommender Systems Handbook. *Springer*.
5. The Movie Database (TMDB) API. https://www.themoviedb.org/documentation/api