



Technical University of Denmark

02807 Computational Tools for Data Science

Hybrid Movie Recommendation with User Clustering and TF–IDF Genres

Student:

Bashar Bdewi (s183356)

Date:

December 4, 2025

Contents

Summary	1
1 Problem and Motivation	1
2 Data and Preprocessing	1
2.1 Data sources	1
2.2 Ratings and user-item matrix	1
2.3 Genre text representation	2
3 Methods	2
3.1 Popularity baseline	2
3.2 K-means user clustering	2
3.3 Cluster-only recommender	3
3.4 Hybrid recommender	3
3.5 Evaluation metric	3
4 Results	4
5 Discussion and Limitations	5
6 Conclusion and Future Work	5
References	6
Use of Generative AI	6
Contribution	6

Abstract

This project builds and evaluates a small movie recommender system using tools from 02807 plus one external method. The main course topic is *clustering* (k-means) on a user–item rating matrix; this is combined with a *content-based* component using TF–IDF on movie genres (external method).

In this report, I first motivate the problem of building a movie recommender system and explain why it is relevant in practice. I then describe the datasets I use, how I clean and merge them, and how I construct the user–item matrix and genre-based features. Afterwards, I present the three recommendation strategies: a simple popularity baseline, a k-means based cluster-only model, and a hybrid model that combines clustering with TF–IDF genre similarity, together with the evaluation setup.

1 Problem and Motivation

Recommender systems are a classic data-science problem with clear real-world relevance in platforms such as Netflix and e-commerce sites. The goal is to suggest items that a user is likely to enjoy, based on past interactions and item content.

In this project, I investigate a *hybrid* movie recommender that combines:

- a collaborative signal based on **k-means clustering of users** (course topic),
- a content-based signal derived from **TF–IDF features over movie genres** (external method).

The main question is whether this simple hybrid design can improve over a popularity baseline when evaluated on held-out ratings.

2 Data and Preprocessing

2.1 Data sources

Two public datasets are used:

- **MovieLens “latest small”**: 100,004 explicit ratings from 671 users on 9,066 movies.
- **Kaggle “The Movies Dataset”**: movie metadata including `id`, `title`, and a JSON-encoded `genres` list.

Both files were downloaded manually and placed in a local `data/` folder. I keep only movies that appear in both datasets and have valid integer IDs. After merging and cleaning, the final movie table contains **2,831 movies** with columns: `movieId` (MovieLens ID), `title`, and `genres` (list of `{id,name}` objects).

2.2 Ratings and user–item matrix

Ratings are randomly split into 80% train (80,003 rows) and 20% test (20,001 rows). The rating distribution shows a strong positive bias (mean 3.54 with most ratings in $\{3, 4, 5\}$). User activity and movie popularity are highly skewed: many users and movies have few ratings, while a few are extremely active or popular.

From the training data I build a user–item matrix:

- rows = 671 users,
- columns = 8,399 movies that appear in the training set,

- entries = rating value if observed, otherwise NaN.

For algorithms that cannot handle missing values (k-means), NaNs are replaced with 0.0. The resulting matrix is extremely sparse (about 1.4% non-zero), which is typical for recommender data.

2.3 Genre text representation

To use content-based similarity, I convert the JSON `genres` field into a space-separated string, e.g. “Drama Crime Thriller”. I then apply `TfidfVectorizer` on `genres_str` for all 2,831 movies, yielding a TF-IDF matrix of shape 2831×22 (22 distinct genre tokens). Pairwise cosine similarity between TF-IDF vectors gives a 2831×2831 similarity matrix where entry (i, j) measures how similar movie i is to movie j based on genres.

As a sanity check, using *Heat* as a query movie returns other crime-action-drama titles such as *Scarface* among the most similar items with cosine similarity 1.0, confirming that the content representation behaves as intended, though it is rather coarse and not very discriminative.

3 Methods

3.1 Popularity baseline

The baseline recommender ranks movies globally by their mean rating in the training set, restricted to movies with at least 20 ratings. For any user, the top- N list is simply the same global top- N , excluding movies that the user has already rated in the training data. This model does not use any personalisation but is a strong and simple baseline when the item popularity distribution is heavy-tailed.

3.2 K-means user clustering

The main course-related method is k-means clustering on the user-item matrix. I apply `KMeans` with $K = 20$ clusters on `user_item_filled` (missing ratings set to 0.0). Each user is assigned to one cluster.

Cluster sizes are highly unbalanced: one cluster contains about 330 users, while several clusters contain fewer than five users. A 2D PCA projection of users coloured by cluster assignment is shown in Figure 1. The clusters overlap strongly in this low-dimensional view, and the Davies-Bouldin index is 2.052, indicating low compactness and separation.

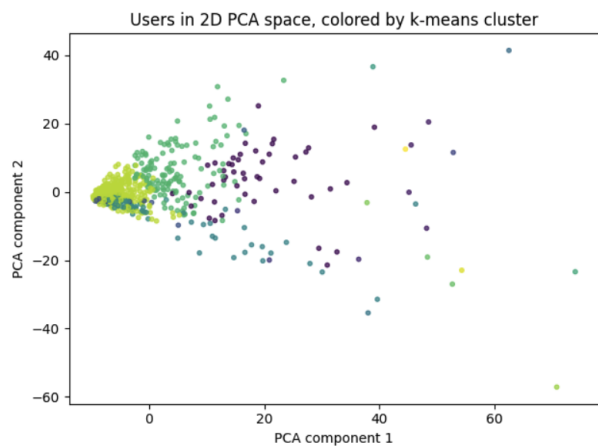


Figure 1: Users projected to 2D PCA space and coloured by k-means cluster.

3.3 Cluster-only recommender

For a user u in cluster c , the cluster-only recommender:

1. selects all users in cluster c ,
2. computes the mean rating per movie within the cluster,
3. filters out movies already rated by u and movies with too few ratings in the cluster,
4. ranks remaining movies by cluster-mean rating and recommends the top- N .

If no eligible movies are found (e.g. very small cluster), the algorithm falls back to the popularity baseline.

3.4 Hybrid recommender

The hybrid recommender combines collaborative and content information:

1. Identify movies that user u liked in the training data (ratings ≥ 4.0).
2. Obtain cluster-based candidates as in the cluster-only model.
3. For each liked movie, retrieve its most similar movies using the TF-IDF cosine similarity matrix.
4. Merge candidate sets, remove movies already rated by u .
5. For each candidate movie m , compute

$$\begin{aligned}\text{cluster_score}(m) &= \text{mean rating of } m \text{ in cluster } c, \\ \text{content_score}(m) &= \max_{l \in \text{liked}(u)} \cos(\text{TFIDF}(m), \text{TFIDF}(l)).\end{aligned}$$

6. Combine them into

$$\text{final_score}(m) = 0.7 \cdot \text{cluster_score}(m) + 0.3 \cdot \text{content_score}(m),$$

and rank by `final_score`.

If u has no liked movies in the training set, the hybrid recommender falls back to the cluster-only model.

3.5 Evaluation metric

To compare models, I use hit-rate@10. For a user u , let $T(u)$ be the set of movies that the user rated in the *test* data and $R(u)$ the top-10 recommendation list from the model. User u counts as a hit if $T(u) \cap R(u) \neq \emptyset$. Hit-rate@10 is the fraction of evaluated users with a hit.

I evaluate the popularity and cluster-only models on 200 test users, and the hybrid model on 160 users (those with at least one liked movie in the training set).

4 Results

For an example user, the cluster-only recommender suggests classic, well-rated films such as *While You Were Sleeping*, *Lassie Come Home*, and *Edward Scissorhands*, which are popular within the user’s cluster. The hybrid recommender returns similar items but slightly reorders them depending on genre similarity; movies that are both highly rated in the cluster and genre-similar to the user’s liked items receive the highest final scores.

Table 1 summarises the hit-rate@10 results.

Table 1: Hit-rate@10 on a sample of test users.

Model	Hit-rate@10
Popularity	0.210
Cluster-only	0.210
Hybrid	0.037

and Figure 2 shows them visually.

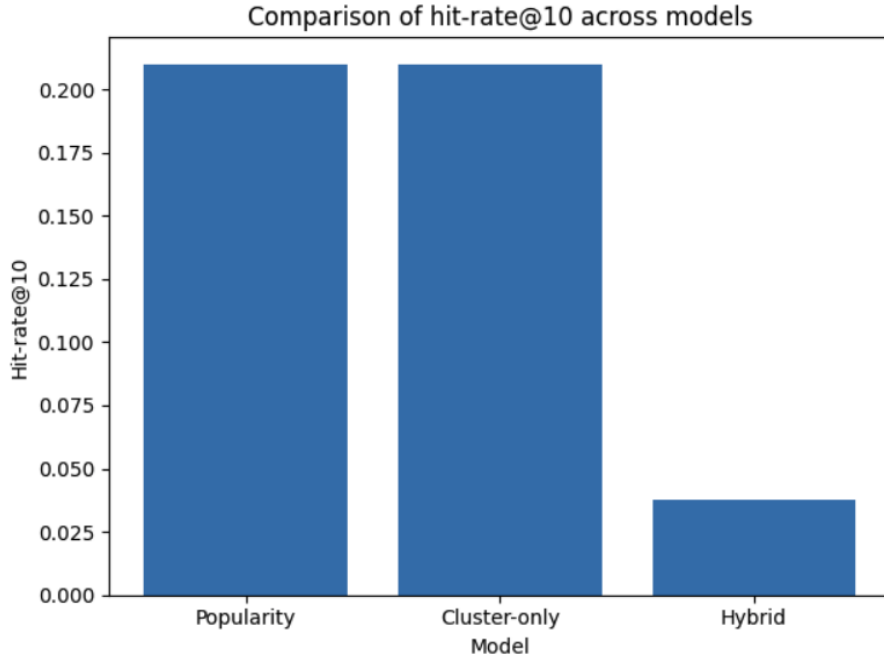


Figure 2: Comparison of hit-rate@10 across models.

The popularity baseline and cluster-only model both achieve a hit-rate@10 of 0.210, i.e. for about 21% of evaluated users at least one held-out movie appears in the top-10 list. The hybrid model performs substantially worse, with hit-rate@10 of only 0.037.

During evaluation, the cluster-only model often fails to find enough unseen movies within a cluster satisfying the minimum rating threshold and therefore falls back to the global popularity list, which explains why both models obtain the same hit-rate. The hybrid model also frequently falls back to cluster-only or popularity when users lack liked movies in the training set or when candidate filtering becomes too strict.

5 Discussion and Limitations

The experiments show that, in this setting, user clustering with k-means does not provide a clear improvement over a global popularity model, and the proposed hybrid approach even degrades performance.

Several factors likely contribute:

- The user–item matrix is extremely sparse, and representing missing ratings as 0.0 is a crude approximation that distorts Euclidean distances used by k-means.
- The resulting clusters are unbalanced and overlapping (large Davies–Bouldin index and the PCA visualisation in Figure 1), meaning that users within a cluster are not necessarily much more similar to each other than to users in other clusters.
- The content features are very coarse: TF–IDF over a small set of genre labels produces many nearly identical vectors, so the content signal is weak and not very discriminative.
- The hybrid design and hyperparameters (rating threshold, minimum rating counts, 0.7/0.3 weighting) were chosen heuristically and not tuned. The additional filtering introduced by the hybrid logic appears to remove many relevant test items from the candidate set.

6 Conclusion and Future Work

This project implemented and analysed a hybrid movie recommender that combines k-means user clustering with TF–IDF-based genre similarity. While the popularity and cluster-only models achieved a modest hit-rate@10 of 0.21, the current hybrid approach performed significantly worse at 0.037.

The main lessons are that k-means on sparse rating vectors is a weak collaborative model compared to more specialised techniques such as matrix factorisation, and that simple genre-only TF–IDF features provide limited additional information. Future work could explore matrix factorisation or neural collaborative filtering as the collaborative component, richer content features (plots, tags, or learned text embeddings), and systematic hyperparameter tuning of the hybrid weighting scheme and candidate generation strategy.

The full code, notebook, and report sources are available at: <https://github.com/basharbd/02807-ctds-hybrid-movie-recommender>

Bibliography

References

- [1] F. Maxwell Harper and Joseph A. Konstan. The MovieLens datasets: History and context. *ACM Transactions on Interactive Intelligent Systems*, 5(4), 2015. Dataset available at: <https://grouplens.org/datasets/movielens/>
- [2] The Movies Dataset. Kaggle. Available at: <https://www.kaggle.com/datasets/rounakbanik/the-movies-dataset>
- [3] F. Pedregosa et al. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12, pp. 2825–2830, 2011. Project documentation: <https://scikit-learn.org/>
- [4] J. Leskovec, A. Rajaraman, and J. Ullman. *Mining of Massive Datasets*. Cambridge University Press, 2nd edition, 2014. Book homepage: <http://www.mmds.org/>
- [5] Saadat Rizvi. Movie Recommendation System (GitHub repository). Available at: <https://github.com/SaadatRizvi/Movie-Recommendation-System>

Use of Generative AI

The generative AI tools I have used: ChatGPT (OpenAI, GPT-5.1), Which were used to help brainstorm the project design, structure the notebook(plotting, k-means, TF-IDF, loops), and to rephrase parts of the report text.

I reviewed, adapted, and tested all suggested code before using it, and edited to match my own understanding and the actual results.

All final design decisions, experiments, and interpretations are my own responsibility.

Contribution

The project was completed as an individual project, with this arrangement approved by Karl and David.