CineMatch: A Movie Recommendation System

PA026 – Semester Project

Libor Máčka, 568986

June 2026

Abstract

CineMatch is a full-stack movie recommendation system that explores four recommendation strategies: content-based filtering, collaborative filtering via SVD, a cosine-similarity variant of collaborative filtering, and a rule-based hybrid. The system is trained on the MovieLens 32M dataset cross-referenced with TMDB metadata.

Contents

1 Problem Statement

Online film catalogues contain tens of thousands of titles. The core problem CineMatch addresses is:

> *Given a user's history of movie ratings, return a ranked list of unseen movies the user is most likely to enjoy.*

CineMatch is a three-tier web application: a React/TypeScript frontend, a Node.js/TypeScript backend API, and a Flask/Python ML microservice. All three services are containerised with Docker Compose.

2 Dataset

2.1 Sources

MovieLens 32M provides approximately 32 million ratings (scale 0.5–5.0) from ∼200 000 users for ∼80 000 movies, plus a `links.csv` that maps each MovieLens movie to a TMDB identifier.

TMDB Movie Dataset v11 provides rich metadata per film: `title`, `overview`, `genres`, `keywords`, `popularity`, `vote_average`, and `vote_count`.

2.2 Preprocessing

Raw data is processed by `develop/scripts/clean_data.py`:

1. Retain TMDB movies with `vote_count` > 30.
2. Build a unified `content` field: overview $\oplus$ genres $\oplus$ keywords.
3. Compute popularity_log $= \log(1 + \text{popularity})$ as a bootleg filter.
4. Merge MovieLens ratings with TMDB via `links.csv`.
5. Keep only movies with ≥ 5 ratings and users with ≥ 5 ratings.
6. Mean-centre ratings: $r' = r - 2.5$, so that dislikes are negative and likes are positive.

2.3 Evaluation Split

Users are split 93%/7% train/test. Within each test user's liked movies ($r' \geq 1.25$), 20% are held out as ground truth; the model trains on the remaining 80%.

3 Recommendation Approaches

All four strategies share the same inference pattern: build a *user profile vector* from rated movies, score every candidate, multiply by $\log(1 + \text{popularity})$ to hide bootleg movies, and return the top-K.

3.1 Content-Based Filtering

Each movie is represented by a TF-IDF vector of its `content` field (`max_features` = 5000, `min_df` = 5, `max_df` = 0.7). The user profile is a rating-weighted sum of the TF-IDF vectors of rated movies:

$$\mathbf{p}_u = \sum_{i \in R_u} r'_{ui}\, \mathbf{m}_i \tag{1}$$

Recommendations are ranked by cosine similarity between $\mathbf{p}_u$ and each candidate movie vector. Movies rated above 2.5 attract the profile; movies rated below 2.5 repel it. The method works without any other users' data, making it suitable for niche viewers – but it creates a filter bubble, as it cannot introduce styles absent from the user's history.

3.2 Collaborative Filtering (SVD)

The centred rating matrix $R \in R^{|users| \times |movies|}$ is decomposed with Truncated SVD ($k = 35$ components):

$$R \approx UV^\top \tag{2}$$

For a new user, a latent profile is derived by projecting their ratings:

$$\mathbf{p}_u = \sum_{i \in R_u} r'_{ui}\, \mathbf{v}_i^\top \tag{3}$$

Candidates are scored by dot product $\hat{r}_{ui} = \mathbf{p}_u \cdot \mathbf{v}_i$. This method leverages collective user behaviour and can surface movies entirely unlike the user's past, but performs poorly for users with highly unusual tastes.

3.3 Collaborative Similarity (Latent Cosine)

A variant of the SVD approach that replaces the dot-product score with cosine similarity in the latent space:

$$s_i = \cos(\mathbf{p}_u,\, \mathbf{v}_i) = \frac{\mathbf{p}_u \cdot \mathbf{v}_i}{\|\mathbf{p}_u\|\, \|\mathbf{v}_i\|} \tag{4}$$

Normalising removes the effect of vector magnitude, making the scoring more robust to users who use very different rating scales. In practice, results are close to the standard SVD variant for most users.

3.4 Hybrid Approach

The hybrid routes each user to whichever base method best suits their profile, using `vote_count` as a proxy for how mainstream their tastes are. The median (not mean) across rated movies is used so that a single blockbuster does not dominate:

$$\text{method} = \begin{cases} \text{content-based} & \text{if median(vote\_count)} < 1000 \\ \text{collaborative (SVD)} & \text{otherwise} \end{cases} \tag{5}$$

Users who prefer niche films (median vote count < 1000) go to content-based filtering; mainstream viewers go to collaborative. The threshold was chosen empirically.

3.5 Why Only Two Approaches Were Deployed

The final ML-API exposes only content-based and collaborative endpoints for two reasons:

- **Hybrid** is a stateless router over the two base models, not a new model. The same effect is achieved by letting the user choose which endpoint to call in the frontend, which is also more transparent.
- **Collaborative Similarity** produced near-identical evaluation results to content-based filtering (Hit Rate@20 of 0.0011 for both), not to the dot-product SVD variant (0.1490). Normalising by vector magnitude discards the absolute popularity signal preserved by the SVD dot product. Because its performance profile duplicates the already-deployed content-based endpoint, a separate deployment would increase operational complexity with no benefit.

4 Offline Evaluation

4.1 Protocol and Metrics

Offline evaluation is implemented in `develop/notebooks/model_training.ipynb` with the following parameters: $K = 20$, `holdout_ratio`=0.2, `threshold_rating`=1.25 (original ≈3.75/5.0), `threshold_count`=10 (minimum profile size).

Metric	Definition
Hit Rate@20	Fraction of test users with ≥ 1 hidden movie in the top-20 list.
Precision@20	Mean fraction of the top-20 list that overlaps with the hidden set.
Recall@20	Mean fraction of hidden movies that appear in the top-20 list.
Semantic Recall (TF-IDF)	For each hidden movie, max cosine similarity to any recommended movie using TF-IDF vectors. Rewards near-misses in content space.
Semantic Recall (SVD)	Same as above but using SVD latent vectors.

Table 1: Offline evaluation metrics.

4.2 Results

Table 2 summarises the evaluation over **14 033** test users (93/7 train/test split, ≥ 10 rated movies per profile, $r' \geq 1.25$ for a movie to count as liked).

Method	HR@20	P@20	R@20	Sem. TF-IDF	Sem. SVD
Content-based	0.0011	0.0001	0.0003	0.0436	0.2992
Collaborative (SVD)	**0.1490**	**0.0098**	**0.0147**	**0.1078**	0.2450
Collab Sim	0.0011	0.0001	0.0003	0.0768	0.1845
Hybrid	0.1430	0.0093	0.0144	0.1051	**0.2503**

Table 2: Offline evaluation results ($K = 20$, 14 033 test users). Bold = best per column.

Collaborative filtering is the clear leader on all exact-match metrics. Content-based and Collab Sim both exhibit a near-zero Hit Rate ($\approx$0.1 %), confirming that recommending exact held-out films from a text or latent-cosine profile is very difficult. Both still achieve meaningful semantic recall, indicating their recommendations are thematically related to the hidden films even when they do not match exactly. The hybrid scores slightly below pure collaborative on exact-match metrics because a fraction of test users are routed to the weaker content-based path; the marginal gain in SVD semantic recall (0.2503 vs. 0.2450) reflects this mixing.

5 User Testing

5.1 Study Design

In addition to offline evaluation, the live system was tested with real users. Interaction logs were captured automatically: each time a user requested recommendations or rated a recommended movie, a structured JSON entry was written to a log file. In total, 14 log files were collected, yielding **74 unique recommendation sessions**.

Due to the limited number of available testers, the dataset is small and results should be interpreted with caution. Nevertheless, the log-based evaluation provides concrete real-world feedback that complements the offline metrics.

For each session, the following metrics were computed from the user's ratings on recommended movies:

- **Median user rating** – median star rating (0.5–5.0) given to recommended movies.
- **Success@4** – fraction of recommended movies rated ≥ 4.0.
- **Success@4.5** – fraction of recommended movies rated ≥ 4.5.

5.2 Overall Results

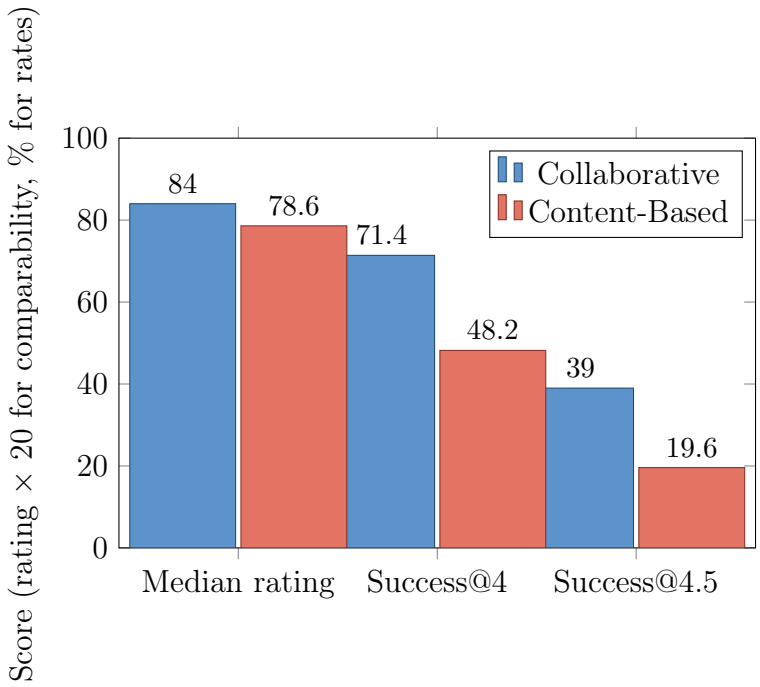

Figure 1: Overall user study results across 74 sessions. Median rating is scaled to a 0–100 range (original $\times 20$) for visual comparability with the percentage metrics. Raw values: Collaborative 4.20 / Content-Based 3.93.

Collaborative filtering outperformed content-based on every aggregate metric. Users rated collaborative recommendations a median of **4.20** stars and found 71% of them to be "good" (≥ 4 stars), compared to **3.93** and 48% for content-based.

5.3 Results by Profile Size

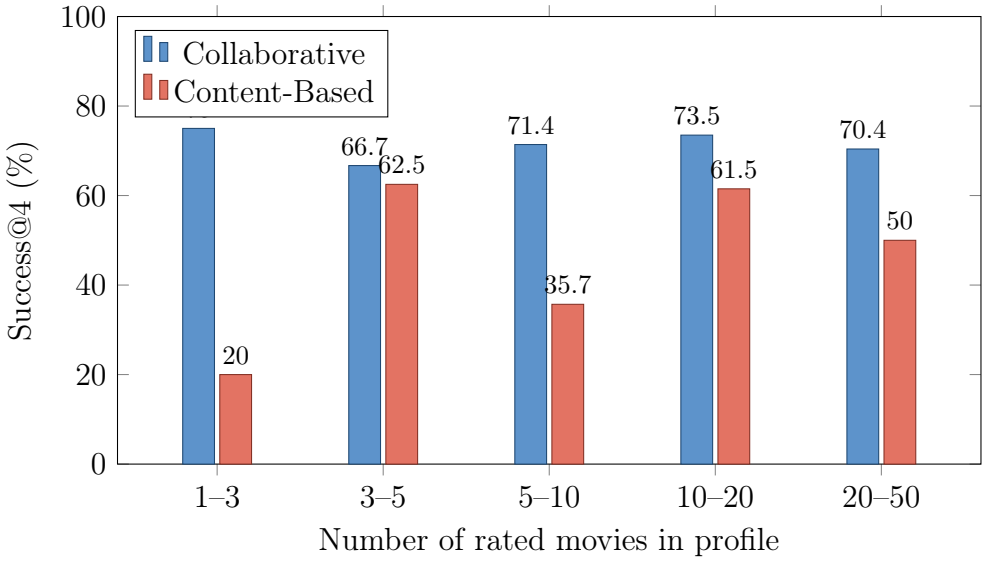

Figure 2: Success@4 (%) broken down by user profile size (number of rated movies).

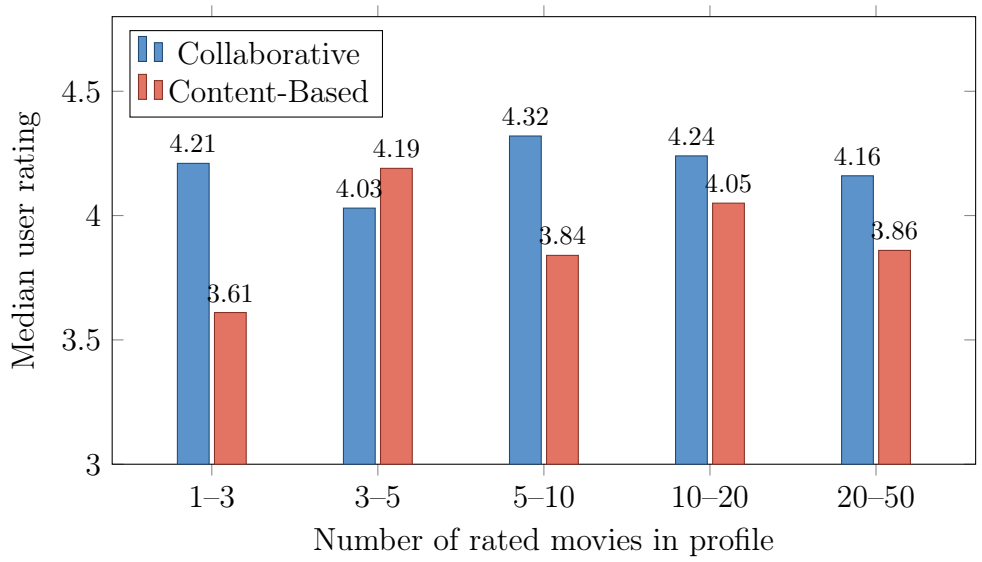

Figure 3: Median user rating broken down by profile size.

The notable exception is the **3–5 movie** group, where content-based is competitive (4.19 vs. 4.03 median). With very few ratings, SVD has little signal to project into latent space, while TF-IDF can still capture genre and keyword similarity effectively. For all other profile sizes collaborative filtering is clearly ahead.

5.4 Results by Profile Popularity

Users were grouped by the median TMDB `vote_count` of their rated movies into four tiers: *niche* (<1k votes), *semi-niche* (1k–10k), *mainstream* (10k–50k), and *very mainstream* (>50k).

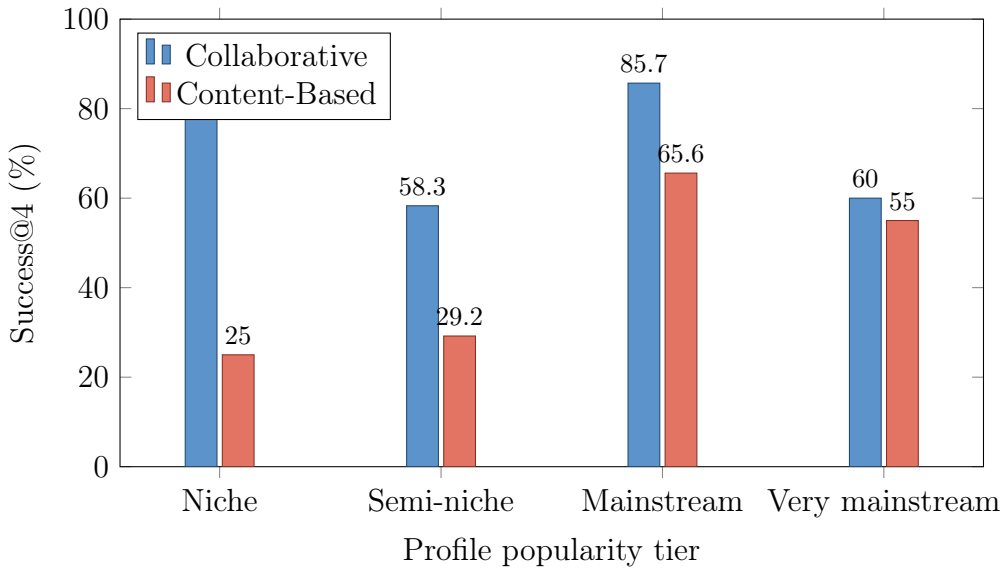

Figure 4: Success@4 (%) by profile popularity tier.

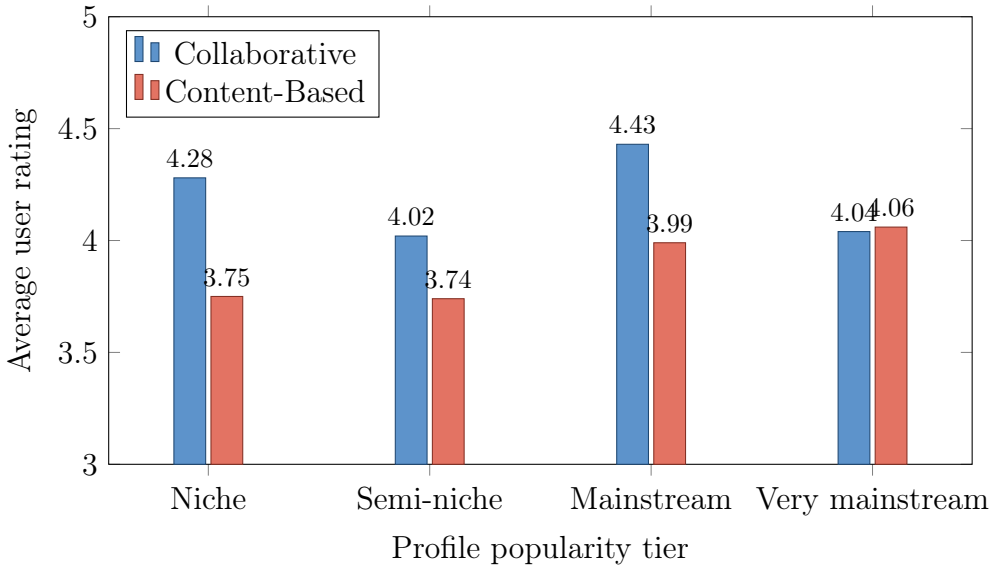

Figure 5: Average user rating by profile popularity tier. Content-based draws nearly even only at the *very mainstream* tier (4.06 vs. 4.04).

Collaborative filtering dominates across all tiers. The near-tie in the *very mainstream* tier (4.06 vs. 4.04) is most likely a statistical artefact of the small evaluation set rather than a meaningful difference.

6 How to Run the System

6.1 Full Stack (Docker Compose)

```
cd cinematch
docker-compose up --build
```

This starts the React frontend (port 5173), Node.js backend (port 7777), and Python/Flask ML-API (port 5000). Pre-trained model artefacts must be present in `ml-api/model/` before starting (see Section 6.3) and all enviroment files must configured (see section 6.2).

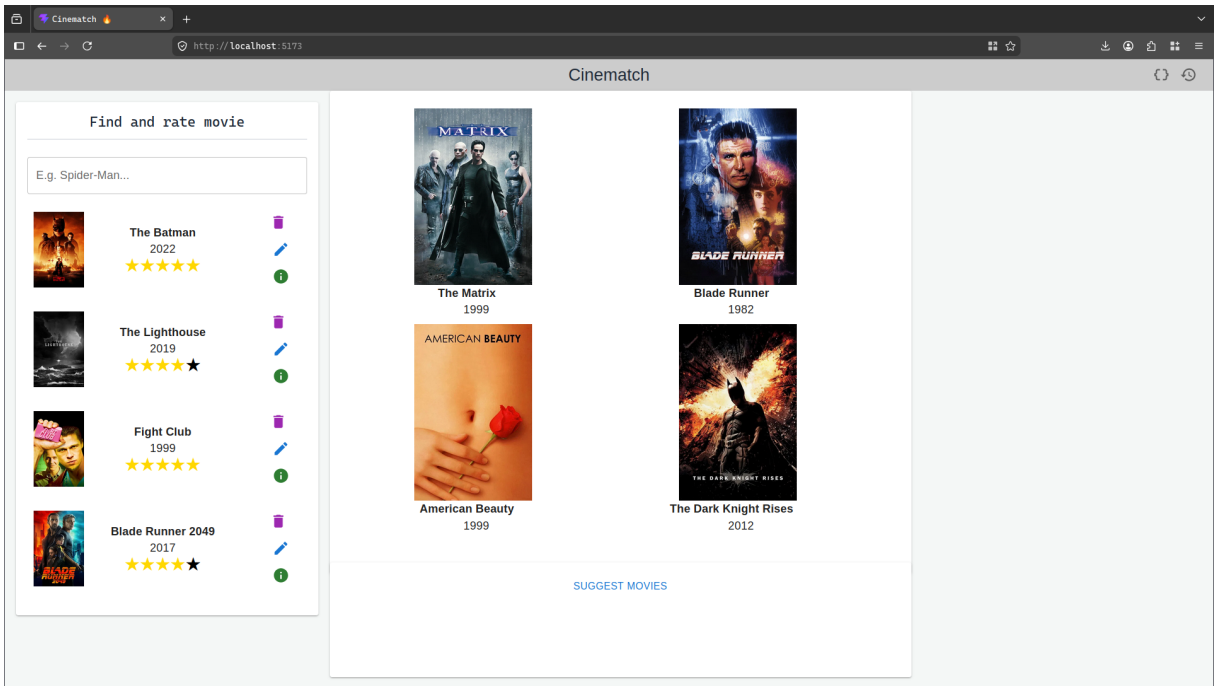

Figure 6: CineMatch running on `localhost:5173`: recommended movies (The Matrix, Blade Runner, American Beauty, The Dark Knight Rises) generated from the four-movie profile visible in the left sidebar.

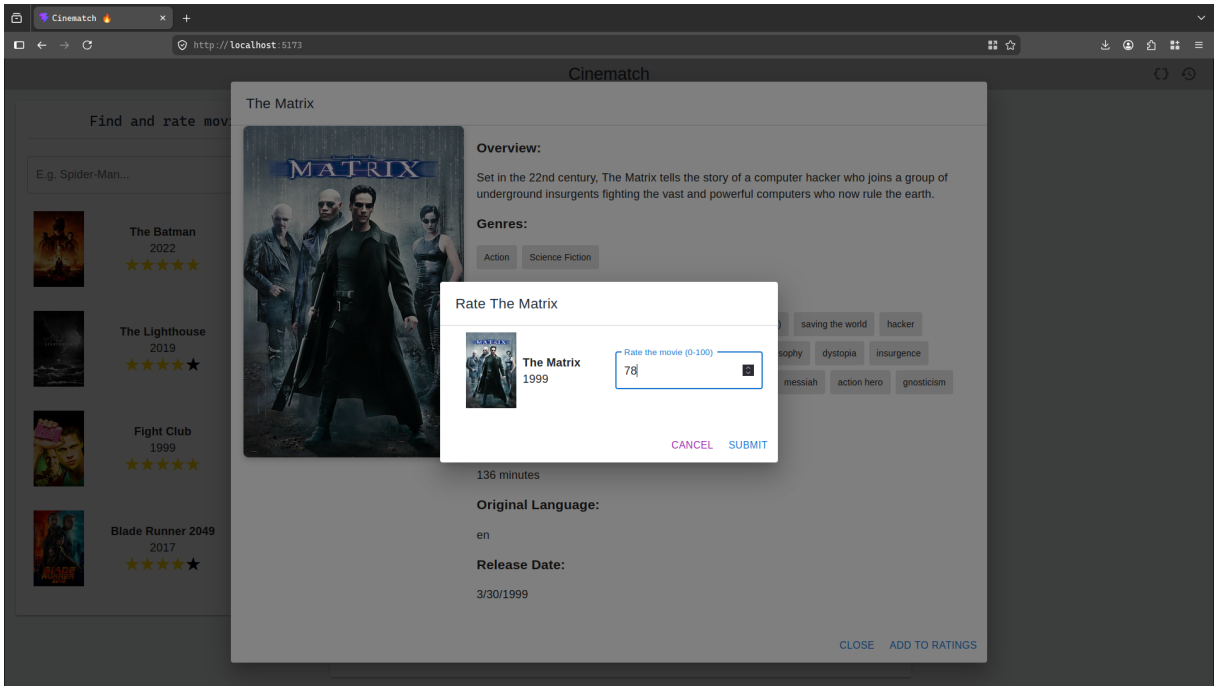

Figure 7: Movie detail and rating dialog: the user opens a recommended film, reads its overview and metadata, and submits a rating to add it to their profile.

6.2 Environment Configuration (.env Files)

Each component reads its runtime configuration from a `.env` file that must be created before building. Copy the snippets below or the bundled `.env.example` files as a starting point.

backend/.env

```
PORT=7777
ML_API_IP=http://ml-api:5000
TMDB_CSV_PATH=/app/dataset/tmdb-movies/TMDB_movie_dataset_v11.csv
```

`PORT` sets the listening port (must match the Docker Compose binding). `ML_API_IP` uses the Docker Compose service name so the backend can reach the ML-API container over the internal network. `TMDB_CSV_PATH` points to the TMDB dataset inside the mounted volume (`./develop/datasets/clean` → `/app/dataset`).

ml-api/.env

```
TMDB_CSV_PATH=/app/dataset/tmdb-movies/TMDB_movie_dataset_v11.csv
RATINGS_CSV_PATH=/app/dataset/ml-32m/ratings.csv
```

Both paths resolve through the same volume mount as the backend. `RATINGS_CSV_PATH` is required by the collaborative recommender; the content-based recommender uses only `TMDB_CSV_PATH`.

frontend/.env

```
VITE_BE_URL=http://localhost:7777
```

The frontend is a Vite/React app run on the host (not inside Docker Compose). `VITE_BE_URL` must point to the host-side port exposed by the backend container.

6.3 Training Pipeline

```
cd cinematch/develop/scripts
pip install -r requirements.txt
python pipeline.py          # download -> clean -> train
```

`pipeline.py` runs three steps sequentially:

1. **download_data.py** – fetches TMDB and MovieLens 32M via the Kaggle API (requires access token `~/.kaggle/access_token`).
2. **clean_data.py** – filters, merges, and centres the ratings.
3. **train_models.py** – fits TF-IDF and SVD and saves seven `.pkl` artefacts to `ml-api/model/`.

6.4 Running the ML-API Directly

For ML-API to work, python enviroment file must configured (see section 6.2)

```
cd cinematch/ml-api
pip install -r requirements.txt
# Set paths in .env (copy from .env.example)
python run.py                # starts on http://0.0.0.0:5000
```

Endpoints:

```
POST /api/v1/recommend_content_based
POST /api/v1/recommend_collaborative

# Request body:
{ "ratings": {"<tmdbId>": <0-5>, ...}, "k": 20 }
```

Example – collaborative recommendations for a four-movie profile:

```
curl -X POST http://localhost:5000/api/v1/recommend_collaborative \
  -H "Content-Type: application/json" \
  -d '{
    "ratings": {
      "414906": 5,
      "503919": 3.8,
      "550":    4.75,
      "335984": 4
    },
    "k": 5
  }'
```

Response:

```
[
  {"rank":1,"title":"The Matrix",          "id":603,"vote_average"
     :8.206,"predicted_rating":0.311},
  {"rank":2,"title":"American Beauty",     "id":14, "vote_average"
     :8.0,  "predicted_rating":0.205},
  {"rank":3,"title":"The Dark Knight",     "id":155,"vote_average"
     :8.512,"predicted_rating":0.201},
  {"rank":4,"title":"Memento",             "id":77, "vote_average"
     :8.188,"predicted_rating":0.178},
  {"rank":5,"title":"American History X",  "id":73, "vote_average"
     :8.352,"predicted_rating":0.171}
]
```

Example – content-based recommendations for the same profile:

```
curl -X POST http://localhost:5000/api/v1/recommend_content_based \
  -H "Content-Type: application/json" \
  -d '{
    "ratings": {
      "414906": 5,
      "503919": 3.8,
      "550":    4.75,
      "335984": 4
```

```
    },
    "k": 5
  }'
```

Response:

```
[
  {"rank":1,"title":"Blade Runner",          "id":78,    "
     vote_average":7.931,"match_score":1.170},
  {"rank":2,"title":"The Dark Knight Rises", "id":49026, "
     vote_average":7.777,"match_score":1.090},
  {"rank":3,"title":"The Dark Knight",       "id":155,   "
     vote_average":8.512,"match_score":1.071},
  {"rank":4,"title":"Batman Begins",         "id":272,   "
     vote_average":7.701,"match_score":1.046},
  {"rank":5,"title":"Scream VI",             "id":934433,"
     vote_average":7.2,  "match_score":1.038}
]
```

6.5 Jupyter Notebooks

```
cd cinematch/develop
jupyter lab
# Open develop/notebooks/model_training.ipynb
```

The main notebook `model_training.ipynb` implements and evaluates all four approaches. `evaluations.ipynb` reproduces the user-study analysis from Section 5.

7 Conclusion

CineMatch demonstrates a practical end-to-end recommendation system and a controlled comparison of four strategies on the same dataset and user population.

Collaborative filtering (SVD) was the strongest performer in both offline and live evaluation, achieving a 71% success rate in user testing versus 48% for content-based filtering. Content-based filtering is more competitive for users with very small profiles (3–5 rated movies).

Only content-based and collaborative endpoints were deployed in production. The hybrid adds no new model; the cosine-similarity collaborative variant offered no consistent improvement over the dot-product SVD.

Source Code

The full source code is available on GitHub: https://github.com/liBORECm/cinematch

Figure 8: *
github.com/liBORECm/cinematch