Final Project: Building and Deploying a Movie Recommendation System

Course: Machine Learning

1 Overview and Goals

This capstone project integrates the major topics covered in the course—exploratory data analysis (EDA), feature engineering, modeling, evaluation, and lightweight MLOps for deployment—through the design of a movie recommender system using *The Movies Dataset*. **Interpretability vs. Complexity.** Students are encouraged to experiment with both simple, highly interpretable models and more complex, higher-capacity alternatives, explicitly reflecting on the trade-offs (e.g., transparency, latency, maintenance cost) in their report.

By the end of the project you will be able to:

- Clean and join multi-file datasets, quantifying information loss at each step.
- Build strong baselines and implement both content-based (CB) and collaborative-filtering (CF) approaches.
- Fuse CB and CF into a hybrid model and critically evaluate it.
- Publish an interactive demo on Hugging Face Spaces.

2 Dataset

2.1 Context and Files

The Movies Dataset contains metadata for ~45k movies released on or before July 2017 and ~26M explicit ratings from ~270k users [1]. Key files are summarised in Table 1.

File	Description
`movies_metadata.csv`	Movie-level attributes (many JSON-like strings)
`credits.csv`	Cast and crew information
`keywords.csv`	Plot keywords
`links(.csv)`	Bridge `movieId` → `imdbId`, `tmdbId`
`ratings(.csv)`	~26M explicit ratings (1–5)

Table 1: Core dataset files.

2.2 Joining and Quality Considerations

When aligning IDs, cast `id`/`tmdbId` to numeric types *only after* filtering non-numeric rows and quantify rows dropped. Parse JSON-like columns into structured Python objects.

Worked Example of JSON Parsing. The snippet below converts the `genres` column from a JSON-like string to a list of names:

```
import json
raw = '[{"id": 18, "name": "Drama"}, {"id": 28, "name": "Action"}]'
genres = [g["name"] for g in json.loads(raw)]  # ['Drama', 'Action']
```

Store the parsed list either as a separate column or explode it to a long format for easier aggregation.

Treat missing/outlier numeric values (e.g., `runtime`, `budget`) by log transforms or winsorisation, and normalise highly skewed popularity/vote signals.

3 Project Tasks and Deliverables

Task 0: Reproducible Setup

Create a repository with clear directories for `src/`, `data/`, `models/`, `app/`, and `report/`. Provide `requirements.txt` or `environment.yml` (with pinned versions), set random seeds, and persist all inference artefacts (vectorisers, factor matrices, etc.). The `README.md` must describe how to set up, train, evaluate, and run the app locally.

Task 1: EDA and Data Preparation

Summarise the distribution of ratings, the long-tail of users and movies, sparsity, and temporal dynamics (using `timestamp`). Provide coverage across genres, languages, and countries. Illustrate at least two findings with clear plots (e.g., rating histogram, sparsity heat-map). Document row counts before and after each join/filter step.

Task 2: Baselines

Establish transparent benchmarks. Implement a global popularity recommender using IMDb's weighted-rating (WR) formula:

$$\mathrm{WR}(i) = \frac{v_i}{v_i + m} R_i \; + \frac{m}{v_i + m} C, \tag{1}$$

where R_i is the mean rating of movie i, v_i its vote count, C the global mean, and m a minimum-votes threshold (e.g., 80th percentile). Optionally include a per-genre popularity variant for context.

Task 3: Content-Based Recommender

Construct item vectors by concatenating TF–IDF features from *overview* + *tagline* with multi-hot encodings of genres, keywords, and top-k cast/crew. Optionally reduce dimensionality with truncated SVD or enrich with dense text embeddings. User profiles are rating-weighted aggregates of seen items, centred at user mean. Provide concise natural-language explanations (shared genres, cast overlap) in your app.

For cold-start users, solicit a short preference survey (favourite genres or titles) and back off gracefully to popularity.

Task 4: Collaborative Filtering

Implement both a neighbourhood baseline and a model-based matrix factorisation (MF). The neighbourhood method can be user–user or item–item k-NN with cosine or Pearson similarity. For MF, fit a regularised latent-factor model:

$$\min_{P,Q,b} \sum_{(u,i)\in\Omega} \big(r_{ui} - \mu - b_u - b_i - p_u^\top q_i\big)^2 + \lambda\big(\|p_u\|^2 + \|q_i\|^2 + b_u^2 + b_i^2\big). \tag{2}$$

Library Policy. You may *either* implement algorithms from scratch *or* use well-known libraries such as `Surprise`, `LightFM`, or `implicit`. If you rely on a library, clearly cite it and describe hyper-parameters tuned.

Task 5: Hybrid Model

Blend CB and CF scores via

$$s_{\text{hyb}}(u,i) = \alpha\, s_{\text{CF}}(u,i) + (1-\alpha)\, s_{\text{CB}}(u,i), \tag{3}$$

with α tuned on validation. Alternate designs—switching, two-stage retrieve-then-rerank, or learning-to-rank—are welcome if evaluated fairly.

Task 6: Evaluation and Experiment Design

Use a time-aware protocol (e.g., leave-one-out per user). Report at least three ranking metrics at $K \in \{10, 20\}$—*Precision@K*, *Recall@K*, *Hit Rate@K*, *MAP@K*, or *NDCG@K*—and supplement with catalog coverage, novelty, and intra-list diversity.

Visual Summaries Required. Include a table or plot of metric comparisons across models and two ablation studies (e.g., removing cast features, changing TF-IDF to embeddings). Provide 95% confidence intervals via bootstrapping.

Task 7: Deployment on Hugging Face Spaces

Deploy a public Space using Gradio or Streamlit. Load pre-computed artefacts at launch, allow a title picker or user ID entry, and return a top-k list with short explanations and posters. Keep memory modest by persisting large matrices to disk.

Reference UI. For inspiration, see the excellent Space "CLIP Playground". By emulating its clean layout and responsive hints, you can set user expectations visually.

Task 8: Reporting, Ethics, and Attribution

Submit a concise report (~8–10 pages) covering problem framing, data decisions, model details, evaluation, and error analysis. Reflect on ethical aspects such as popularity bias, representation skew, and user privacy, and propose at least one actionable mitigation (e.g., diversity-aware reranking). Clearly attribute TMDb and GroupLens/MovieLens data.

4 Implementation Guidance

4.1 Feature Engineering and Profiles

Parse JSON-like fields into Python objects, derive interpretable features, and optionally reduce dimensionality with truncated SVD. Build user vectors as rating-weighted averages and ℓ_2-normalise vectors before cosine similarity. Prototype with `ratings_small.csv`, switch to sparse matrices, and leverage approximate nearest neighbours when needed.

4.2 Explanations and UX

Expose rationales (shared genres, similar users) and, for hybrid scores, a simple bar/colour indicator for CB vs. CF contribution.

5 Bonus Challenges (Optional)

Earn up to ten extra points with extensions such as implicit-feedback MF, diversity/fairness reranking (e.g., MMR, xQuAD), neural recommenders (NCF, two-tower), graph-based signals (Personalised PageRank), feature attributions, or lightweight MLOps (MLflow, CI).

6 Deliverables and Submission

Submit a clean repository with executable training/evaluation scripts, saved artefacts, environment file, *and* a link to the live Hugging Face Space. Include instructions to reproduce offline metrics and launch the app locally.

7 Practical Notes and Pitfalls

Ensure correct ID alignment (`movieId` $\leftrightarrow$ `tmdbId`) and avoid data leakage (no future interactions during training). Provide a cold-start path, tune for sparse users, manage memory via pre-computed artefacts, and respect licensing.

8 Minimal API Sketches (Illustrative)

```
# Content-Based
class ContentIndexer:
def fit(self, movies_df): ...
def encode_items(self, movies_df): ...
def build_user(self, user_history): ...
def recommend(self, user_vector, k=10, exclude_seen=True): ...

# Collaborative Filtering (MF)
class MatrixFactorization:
def fit(self, interactions): ...
def user_scores(self, user_id, item_ids=None): ...
def recommend(self, user_id, k=10, exclude_seen=True): ...
```

```
# Hybrid
class HybridRecommender:
def __init__(self, cb, cf, alpha=0.5): ...
def recommend(self, user_id=None, user_profile=None, k=10): ...
```

Review Checklist

- Data joined correctly and transparently.
- Baselines implemented and clearly described.
- CB **and** CF models built and explained.
- Evaluation metrics well-presented with discussion.
- App deployed with meaningful recommendations and rationales.
- Ethical reflection and error analysis included.

References

[1] F. Maxwell Harper and Joseph A. Konstan. "The MovieLens Datasets: History and Context." *ACM Transactions on Interactive Intelligent Systems*, 5(4):19, 2015.